# MULTILEVEL AUTOMATON-BASED IMAGE RETRIEVAL AND SEARCHING

by

**ADLI ANWAR ABUSHABAN, B.C.S.**

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

August 2, 2001

The undersigned hereby recommends to the

Faculty of Graduate Studies and Research

acceptance of the thesis,

# MULTILEVEL AUTOMATON-BASED IMAGE RETRIEVAL AND SEARCHING

Submitted by

**Adli Anwar AbuShaban, B.C. S.**

Dr. Frank Dehne

(Director, School of Computer Science)

Dr. B. John Oommen

(Thesis Supervisor)

Carleton University

August 2, 2001

## ABSTRACT

The use of digital image libraries is becoming essential in many areas of software applications. With the growing number of images that are included in digital image libraries, there is a need for efficient and easy-to-use image search and retrieval applied techniques. The methods that currently exist for solving the image search and retrieval problem are based on either a textual annotation of each image or on a visual analysis of each image. Both methods lack the capability of classifying a group of images based on the users' feedback. In this thesis, we present an application that uses a Learning Automaton solution to the Object-Partitioning Problem (OPP). The OPP is the problem of partitioning a group of objects into a set of classes or cluster. The solution presented in this thesis offers a learning mechanism, and operates by recording user's actions while searching for a target image. The recorded actions are used to better partition the images for subsequent searches.

## ACKNOWLEDGMENTS

I wish to thank God for giving me the strength to finish this thesis and for destining me to work with an excellent supervisor.

I record my sincere regards and gratitude to my Supervisor, Professor B. John Oommen, who enlightened me with his knowledge, spent time with me, and was patient with my schedule.

I also thank my friend Asham El-Rayess for giving me valuable pieces of advise while implementing and writing this thesis.

I thank my parents for encouraging and supporting me during my studies. I also thank my brothers and my sister for praying for me to finish this thesis.

Finally, I thank my wife and my children for their encouragement and patience.

# TABLE OF CONTENTS

## LIST OF FIGURES

# LIST OF TABLES

# Chapter 1: INTRODUCTION

With the rapid increase of the number of software applications and areas, and the growing use of the Internet, there is an ever-increasing need to have digital image libraries in many applications, or as a stand-alone search and retrieval system. Digital image databases are usually developed using two approaches. The first, and currently the most widely used, enables searching for images based on pre-created text annotations that describe the contents of the image.

This approach requires that the image annotations be entered by hand, and that, with great amount of effort and cost. To include a detailed description of each image, and to make the description unique so that it involves many objects in the image is tedious. Spatial relations are important for understanding image content relationships in terms of distance and orientation to each other. Therefore, to adequately describe an image, many attributes have to be entered. This makes the task extremely time consuming and susceptible to human errors. If the system is to include relations among entire images, the number of entries becomes even larger.

The user of a text-based system is expected to provide the system with a description of the image in the query language specific to the application. This, generally

requires a minimum number of attributes to describe the sought for image. The returned results might not be accurate because of the possible different perceptions of images between the operator and the user. Furthermore, this approach does not permit important forms of searching such as "Give me images of all racy cars."

The second approach to solving the image search and retrieval problem is based on mathematical analysis of the colours and objects of the image. This approach is referred to as the content-based approach. An extensive amount of research has been done in this area. The main drawback of this approach is the fact that the user is expected to provide an example image by drawing or *searching* through the database of images. The returned results are usually either identical images or images that share the same visual characteristics, such as color distributions, or brightness levels, although, in fact, the image sought for does not share these visual characteristics with the provided example. Again, this approach may not cover searches such as "Give images of all racy cars."

Generally, an image retrieval application can essentially be broken down into two parts. The first part deals with the indexing of images in the database. The image index is used to distinguish each image from the rest of images when a search is conducted. The second part deals with the user interaction with the system including presenting the selected or the desired images to the user. Therefore, a good image retrieval application should have a good indexing mechanism coupled with clear and easy to use user interface tools.

Most of the methods found in the literature concentrate on the indexing part, and offer a less effective user interface, since the user is expected to describe the image he/she is searching for. Furthermore, these systems offer no learning that is based on the result of user interactions.

The CANDID [KEL95] approach, for example, takes an example image from the user, and extracts the features to be compared with the database of features of all images. Queries could thus take the form of: "Show me all images with similar texture to the example image."

The CHABOT [OLG95] system allows the user to specify a textual-based query to come up with example images that can be selected to perform another search based on visual similarities. The user may specify a compound query such as: "Show the pictures of Lake Ontario at sunset."

In this thesis, we propose a new approach using a learning automaton philosophy that was initially used to solve the object-partitioning problem. This solution is then modified to solve the image retrieval problem.

## 1.1 Learning Automata

The Russian mathematician Tsetlin first proposed the "learning automaton" in 1962. The aim was to create a learning machine that learns from previous responses

offered from a random environment to a given action. The response can be either deterministic or stochastic.

The Tsetlin automaton [TSE62] defines rules for a learning automaton in which one would determine the optimal action out of a set of allowable actions. The Tsetlin learning automaton works by utilizing a sequence of repetitive feedback cycles in which the automaton interacts with the environment. The environment, in turn, offers the automaton a finite set of actions. The environment responds to the automaton's actions with either a reward or a penalty. The automaton uses these responses and the knowledge acquired from the past actions and responses to determine the next action.

Tsetlin, Krinsky, and Krylov modified the original Tsetlin deterministic learning automaton. The modified automata offer more power to the paradigm, and define specific rules for automaton and environment behaviors.

## 1.2 Object Partitioning Problem (OPP)

Suppose we are given W objects to be partitioned into R classes, where each class may have a different number of objects. The aim of the partition is to make the objects that are accessed more frequently together lie within the same class. Therefore, a solution to this problem would seek to maximize the number of times that any two accessed objects will exist in the same class.

This thesis is concerned with a special case of the OPP known as the Equi-Partitioning Problem (EPP) where all classes have equal number of objects.

The Basic Adaptive Method (BAM) [YU81] attempted to solve the EPP by migrating the two accessed objects closer to each other. The final result was that all the jointly-frequently accessed objects came together and subsequently migrating randomly selected objects away from each other within the same class.

The Tsetlin, Krinsky, and Krylov automata can be utilized to solve the EPP by assuming that each action of the automaton is a class of objects. In these cases, all possible combinations of object arrangements must be represented by different classes. If the two accessed objects are found in the same action, the automaton is rewarded by moving towards the most internal state of its current action. If the two accessed objects are not in the same action, the automation will be penalized by moving the current state one state away from the internal state of its class or into the most external state of another class.

Oommen and Ma in 1988 [OOM88] first proposed the Object Migration Automation (OMA) to solve the EPP problem. In the OMA, the number of actions is always equal to the number of the desired partitions and not the number of combination that a given number of objects can be arranged into. This results in a significantly smaller number of classes, which makes the memory representation of the automaton feasible and practical. The OMA automaton also allows the member objects to move around within

itself unlike the traditional automata where the entire automaton moves from one state to another. In the OMA, the states of the automaton are assigned to classes equally. Therefore, rules are defined to outline how *objects* are moved from one partition to the another.

## 1.3 Contribution of Thesis

This research involves the design and implementation of a realistic adaptive image retrieval system called MAIRS. It is built on the previous theoretical contribution by Oommen and Ma [OOM88]. A simplistic concept of how to utilize the results of [OOM88] was proposed by Oommen and Fothergill [OOM93], which involved only two levels of images and a small database. This research work includes vertical object movement in addition to the horizontal migrating and an arbitrary number of levels of images. This can be seen in the implementation section where the structure of the image database is a tree of classes of objects rather than a group of classes. When an image is requested, it and all its "similar" images are migrated. These "objects" can be moved anywhere in the object tree. This thesis introduces procedures by which all such pairs of objects are rewarded whether they are in the same class, have an ancestor relationships or whether they are distant relatives in the tree.

The implementation part of this thesis is very significant when compared with the previous implementation [OOM93]. While the previous implementation was a simple prototype to demonstrate to concept of the OMA with very few features, MAIRS is a complete system that supports different image formats, multilevel browsing, and the

comprehensive migrating of objects. It also supports a fully operational User Interface. The commercial aspects of MAIRS are currently being investigated.

## 1.4 Thesis Outline

This thesis introduces a new approach to solving the image retrieval problem by using the OMA solution to the EPP problem. The main advantage of this approach is that it relieves the user from providing a description of the image, unlike either the textual or the visual methods. The user of the proposed system is required to have a mental view of the required image.

Oommen and Fothergill [OOM93] utilized the OMA solution to the EPP problem to implement a prototype solution to the image retrieval problem. The prototype system is known as the Image Database Learning Automation (IDLA) system. The IDLA supported a main level view of images, and second level subclasses of the main images. The IDLA demonstrated the capability of the OMA solution to the image retrieval problem and provided a comparative study of the BAM and the OMA solutions.

This thesis presents a significantly enhanced version of the original OMA solution to allow a multilevel image database system. The modified modules define new rules for handling the multilevel rewards and penalties. The system that is proposed in this thesis is the Multilevel Automaton-based Image Retrieval and Searching (MAIRS) system which is a complete system that supports multiple level of subclasses and multi image formats.

## 1.4 Thesis format

In chapter 2 of this thesis, we are going to discuss the methods currently used to solve image search and retrieval problem. Each method is studied in terms of its user interface and heuristic techniques used.

Chapter 3 presents an overview of the filed of learning automata with the emphasis on the deterministic automata which is the foundation of our work.

The Object-Partitioning Problem (OPP) and the Equi-Partitioning Problem (EPP) are discussed in chapter 4. The chapter also discusses the existing solutions to the Equi-Partitioning Problem (EPP) which is the main approach to the work of this thesis.

Chapter 5 will present the work done in this thesis starting with theoretical aspect and ending with a detailed description of the application developed.

Chapter 6 will be the conclusions of this thesis report with the suggested future work.

# Chapter 2: THE IMAGE RETRIEVAL PROBLEM

## 2.1 Introduction

Digital image libraries are becoming an important component of computer software and Internet applications. The use of image database applications is growing in a faster rate than the technology that is available for processing the large collection of images. Currently the most prominent approach to searching for an image is to create text annotations that describes the content of the image, and then to enter these textual descriptions into a textual database for retrieval and manipulation.

The problem with this approach is that one description can include multiple images. In most images, therefore, there are hundreds of objects that could be referenced, and typically, each image has a long list of attributes to be included in its image description. Spatial relations are important for understanding image contents. To include the spatial relation in the image description of $n$ objects each with $m$ attributes, $O(n^2m^2)$ [BEL98] entries are needed. If the system is to include relations among entire images, the number of entries becomes even larger.

Generally speaking, image annotations are entered by hand, and that, with great amount of efforts and costs. For access, users are expected to enter a minimum number of attributes to specify an image query. If the result is not rich enough, the attributes are normally re-entered for different applications.

Any image retrieval project or application can be broken down into two parts. First, there is the issue of indexing the images in the database so as to caption each image with a specific title, and to search for a specific title when the user invokes a search. Second, there is the issue of the interaction with the system, and that of presenting the selected (or hopefully, the desired) images to the user.

In a text database, a fixed format is assigned for each document and a search mask is based upon the fields of the format. Typically, the system imposes constraints on the valid field values. Thus, one field may be restricted to integers ranging from one to ten, and another field may be restricted to names of 10 letters or less. The user can create or specify search criteria with specific constraints. The system takes on the task of composing the search criteria into a form that can be compared to each document form in the database.

To perform a similar task in image databases, the user would have to draw a picture or provide a similar example picture. This approach however does not cover searches such as "Find all pictures of political leaders." Such search tasks require the

use of search masks that are assigned for each image. Since each image can be described in many different ways, such technique would not present images that are very similar to a search mask simply because the search format could potentially describe a picture in a different way.

This chapter reviews the user interface and the theoretical literature on the existing image retrieval techniques. The survey is fairly comprehensive in that it covers the main two parts of any image retrieval application, the user interface and heuristic techniques used. The emphasis is on recent work, which also includes the currently used applications.

Present day techniques for solving the image retrieval problem are based on one of two techniques. The traditional techniques utilize textual description or visual descriptions for indexing and searching of images in the database. In some cases a combination of the two methods is used. As an overview of the field, in Sections 2.2 to 2.7, we describe other systems that are currently used to search for target images in databases. These systems also include the user interaction sub-systems, and are outlined in detail. Section 2.9 present techniques, found in the literature, that are used in the image search and retrieval problem. In these cases, we do not discuss the user interaction sub-systems, as they are not described in the relevant source. Therefore, it is assumed that the user interaction is based on the query-by-image approach.

The following sections describe each technique in detail. First, a brief introduction is presented about the application overall. Second, the available reported interface of the particular project is discussed and the existing screen shots of the main user interface windows are displayed. Third, the heuristic techniques used in the project are discussed with the emphasis being on user tools and a brief overview of the mathematical calculations used.

## 2.2 The Expectation-Maximization Strategy

The Expectation-Maximization(EM) system presented in [BEL98] is based on the principle of extracting objects from a given image. The system includes an algorithm which is used to segment the image into objects. The extracted objects are determined based on a homogeneous part of the image analyzed in terms of color and texture. The objects are referred to as blobs that compose the blobworld, which is the image. The user can present an example image to the system and get it analyzed into blobs. The user can also use these blobs to query the system for similar blobs in the database of images.

### 2.2.2 User Interface

In the EM system, the user submits an image to the system. The image is then decomposed into blobs. The user can select some of the blobs from the presented image to query the database for similar blobs. Blobs from several images can also be selected to collectively construct one query, which is referred to as a compound query. Each image in the database is scored based on how closely it satisfies the compound query. The images

in the database is scored based on how closely it satisfies the compound query. The images are ranked and presented according to overall score. Each image has a set of blobs that the user can use to invoke further queries. The user can also change the weighting of each blob so to focus the search on few features. The following figures illustrate a query in the EM system. In figure 2.1, the user may initiate a search by clicking on a presented blobworld or by entering a keyword for the search. In figure 2.2, the user adjusts the weight of the given blob attributes for a refined search result. In figure 2.3, the user is presented with the most matching images.

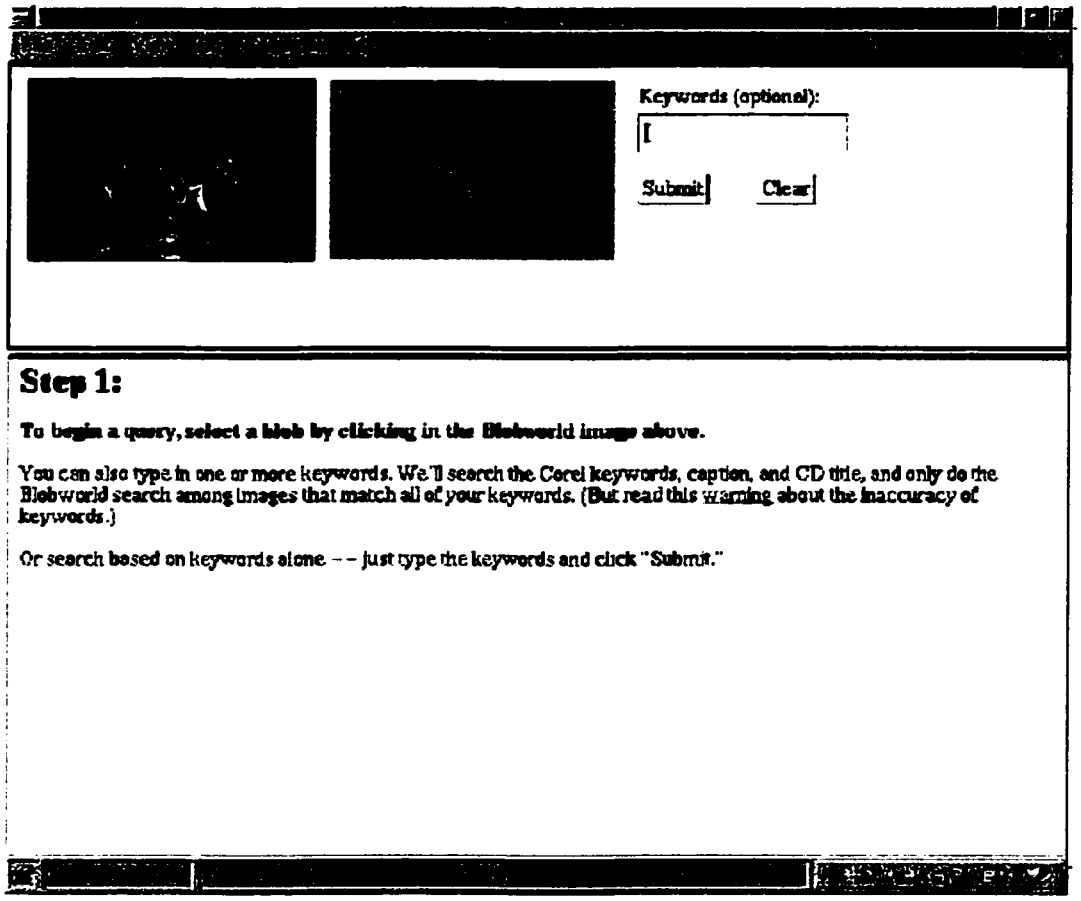



**Figure 2. 1**: Step one of the user interface of the EM method. The user selects a blob or specifies a key word to be used to construct a query.

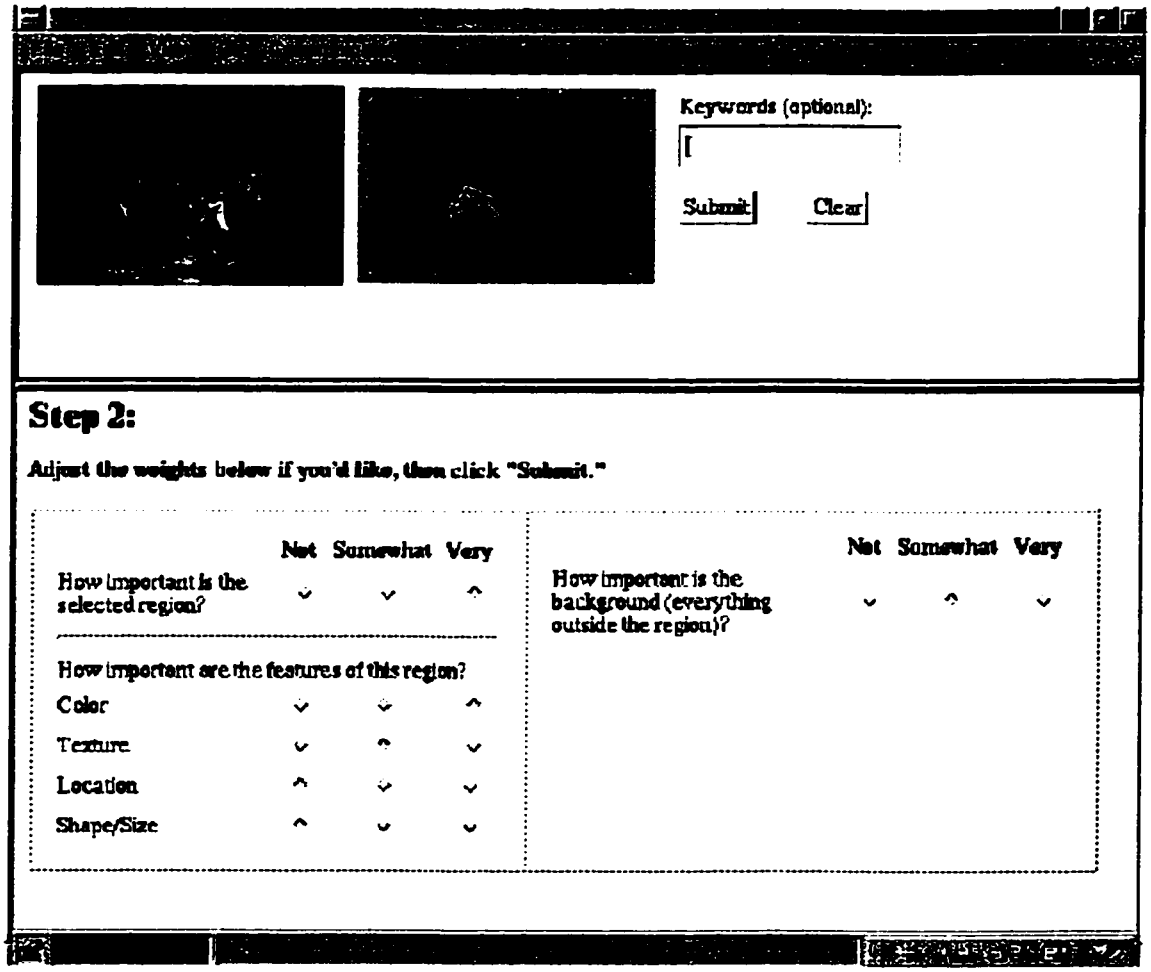



**Figure 2. 2**: Step 2 of the user interface of the EM method. The user adjusts the weight of the desired attribute of the given blob.

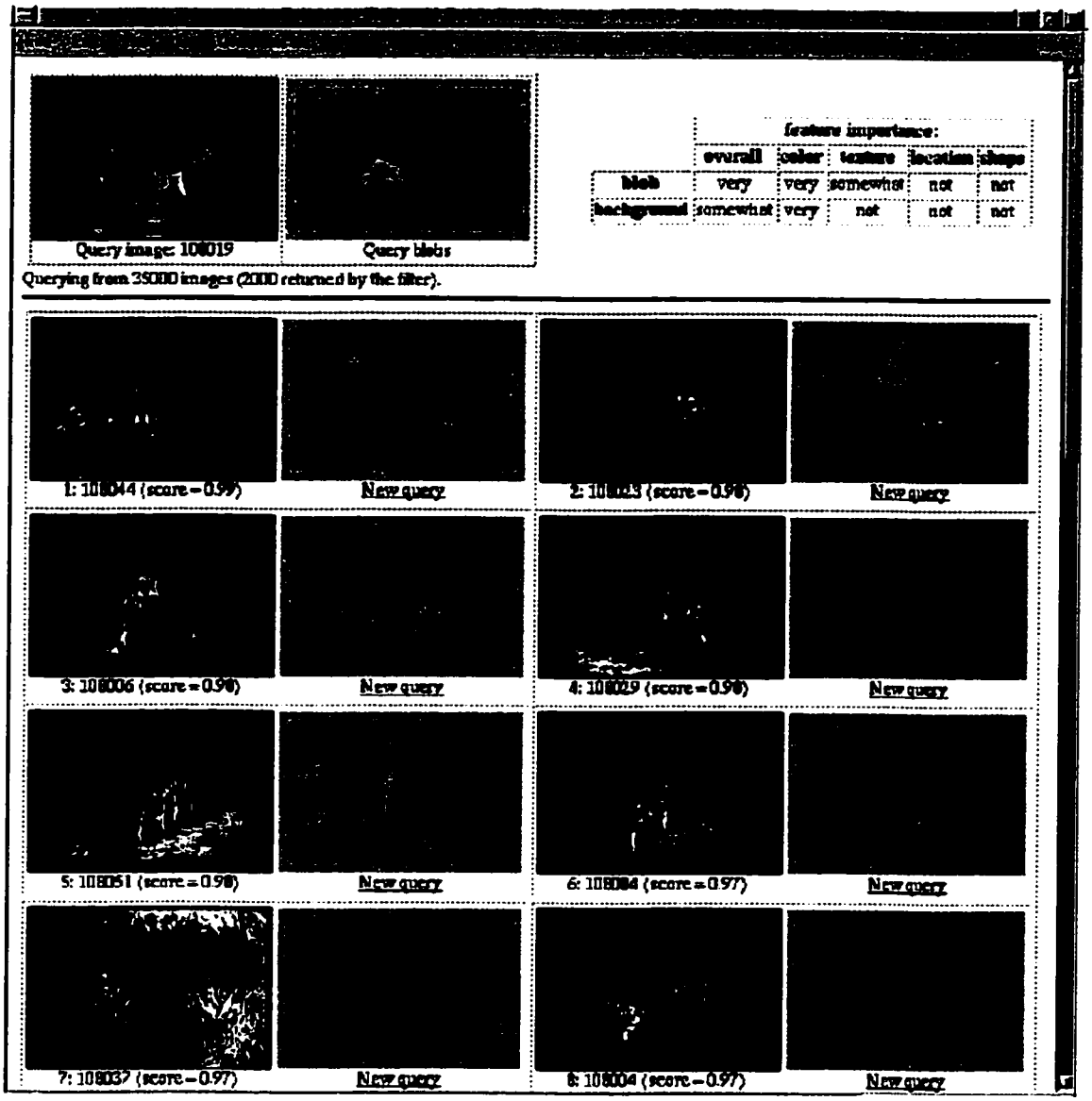



**Figure 2. 3**: Step 3 of the user interface of the EM method. The user sees the most matching images and can start further queries on the presented images.

## 2.2.3 Heuristic Techniques

In the EM system, the image is modeled as a set of groups of pixels. Each group represents a set of coherent pixels in terms of color and texture. The hue saturation value

is used to judge which color a pixel belongs to. The values of *height*, *radius*, and *coordinates* of the point on a disk are treated as a cone. A point with a small *radius* is considered black even if it has large height. The cone representation maps all similar points together.

To make texture computation more descriptive of the actual image, the EM system uses a scale selection strategy. The particular scale selection method used is based on what is referred to as an edge/bar polarity stabilization scheme. The integration scale factor $\sigma$ is used to control the size of the integration window around a pixel. To determine the quantity $\sigma(x,y)$ for each pixel, the system uses the so-called polarity property. The polarity of a pixel is the extent to which the collection of gradient vectors in a certain neighborhood of the pixel point in the same direction. Thus, the polarity at a given pixel is computed with respect to the dominant orientation in the neighborhood of a pixel. The polarity $p\sigma$ varies as the scale $\sigma$ changes, and its behavior in typical image regions can be characterized as follows:

**(i) Edge Characteristics**: If all values of $\sigma$ are close to 1 for all $p\sigma$, an edge is flagged.

**(ii) Texture Characteristics:** Multiple orientations of 2D texture or 1-D flow reduce the polarity value.

**(iii) Uniform Characteristics:** In any constant-intensity neighborhood, the polarity takes an arbitrary value because the gradient vectors have negligible magnitude.

The number of groups that are used to represent any given image, $K$, is selected using the so-called Minimum Descriptor Length principle. This value of $K$ is first estimated to be between 2 and 5 and then successively refined, based on the suitability of the representation. A model, with a given value of $K$, is selected to fit the image data.

The polarity is computed for each pixel in the image that satisfies $\sigma_k = k/2$, $k = 0, 1, \ldots, 7$. The polarity image is computed at the scale of $\sigma_k$, and smoothed using a Gaussian kernel function. For each pixel, a scale is selected as the first value of $\sigma_k$ for which the difference between successive values of polarity is less than 2%. The authors of [BEL98] claim that this method will detect at most 10 pixels.

After selecting a scale for the pixel, it is assigned three texture descriptors: polarity, anisotropy, and the normalized texture contrast. In turn, color is also assigned three descriptors for the color cone coordinates. The EM uses these six descriptors to find the maximum likelihood parameter estimate when there is missing or incomplete data which is the region to which the points belong. The $K$ groups are represented in a matrix and initialized to have a random mean vector. The Gaussian color mixture parameters are inspected to determine what color and texture descriptors are present for each component. As a result of this various groups result, some of which are as below:

*Bright, bluish, and textureless regions could be sky.*

*Anisotropic and non-polar regions could be zebra.*

*Green weak-isotropic texture could be grass.*

Once a $K$ model has been selected, the spatial grouping of image pixels is done so as to organize similar color and texture features together in the same groups. The K-level image is produced to contain pixel memberships by representing each pixel with the label of the cluster for which it attains the highest likelihood. After filtering the raw cluster membership for better spatial smoothing, the data is then converted to a set of labeled image regions.

## 2.3 The Virage System

### 2.3.1 Introduction

The Virage system [GUP97] gives the user several tools to refine search criteria. The user can provide an example image, clip parts of it, and highlight other part to specify a search for an image. The details of the system are given below.

### 2.3.2 User Interface

The Virage system expects the user to provide an example image for querying the images in the database. The query can either be in terms of an existing image or by

drawing a sketch of what the intended image looks like. After an example image has been provided, the user can alter the example image to make it closer to the intended image. This can be done by erasing part of the image, changing brightness values, changing colors, or increasing the relative weight of a shape-like structure. The database is queried for matching images and the matching images are returned to the user after which it is possible to use *these* images for further queries.

The AltaVista Internet search engine uses the Virage system for image search. The Alta Vista utilizes some of the available features to enable users to perform a search. The user is first asked to provide a textual description of the image. The database is searched based on the textual information provided by the user. The user may select one of the result images to submit a query for the visually similar image of the selected image. The following are screen shots of a typical image search. Figure 2.4 is the main user interface window where the user specifies a text to search for images' attributes. Figure 2.5 is the result displaying images obtained when searching for "golf". Figure 2.6 shows the visually similar images to a selected image from the result images. Figure 2.7 gives the textual attributes for a given image.

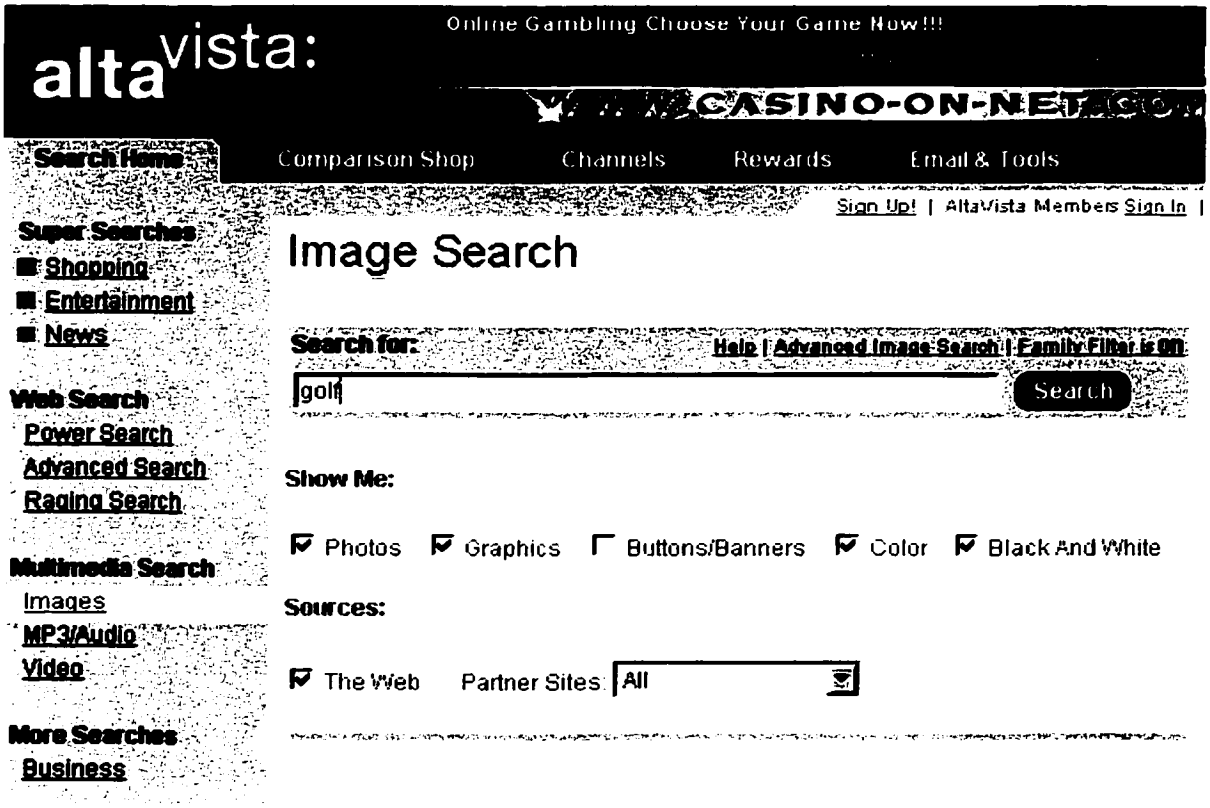



**Figure 2. 4:** The Altavista image search engine interface. The user specifies a text to search for in the images' attributes.

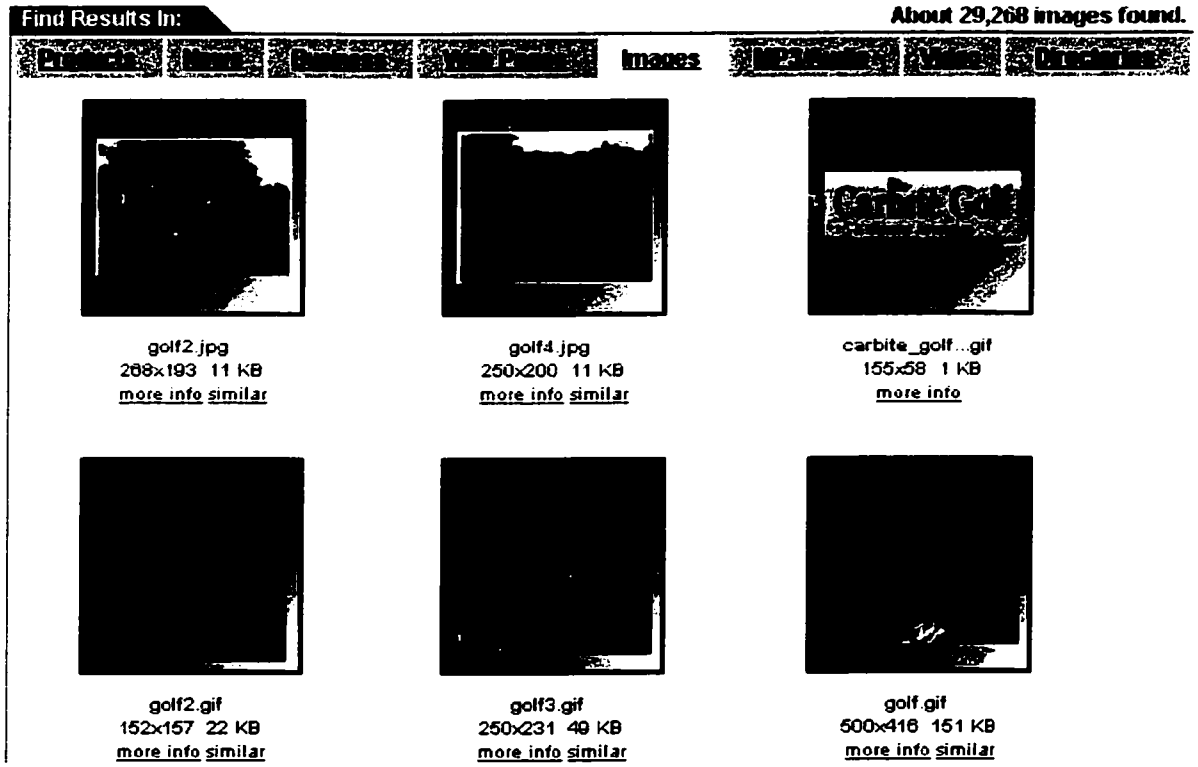



**Figure 2. 5:** The result images of searching for "golf" on Altavista image search engine.

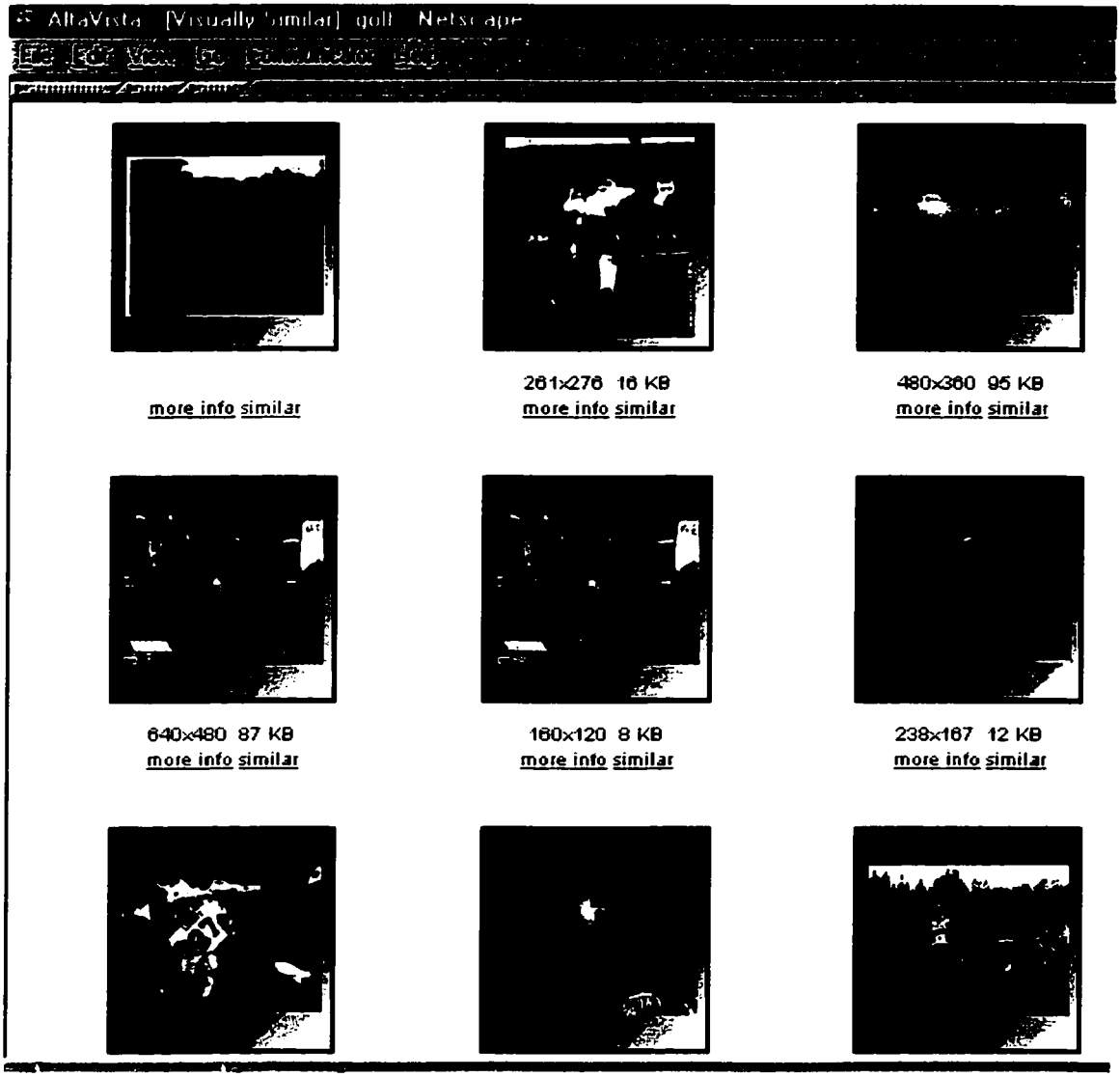



**Figure 2. 6:** The visually similar images to a selected image shown the search result.

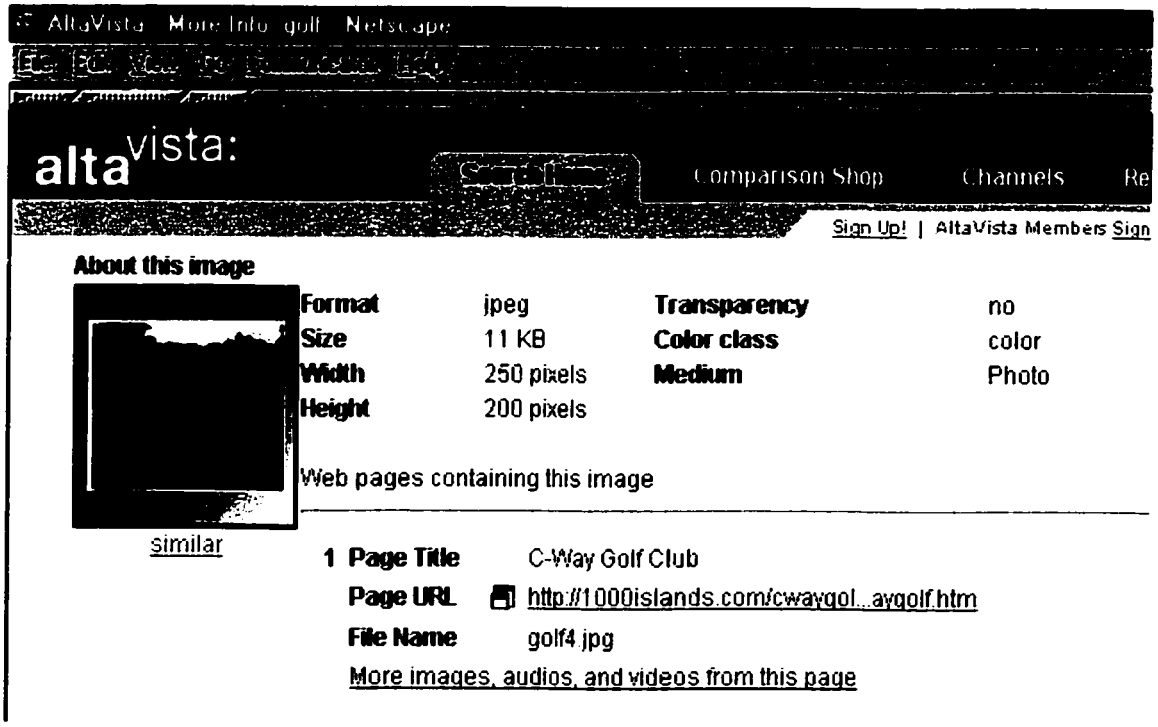



**Figure 2. 7**: The textual attributes for a given image.

## 2.3.3 Heuristic Techniques

The Virage system provides several tools for the user to enhance the user description of the intended image. A brief description of these tools is given below, and the reader is encouraged to refer to [GUP97] for a more complete description.

*Image processing tool:* This tool divides the image into segments or modifies the shape of certain regions of the image. The tool changes the color of a region or background, alters the texture, or highlights certain areas, as schemes by which the system can compare the example image against the database images.

*Feature space manipulation tool*: The user can use this tool to enhance specifying the search condition on the feature instead of using it on an image. The tool uses histogram-based query methods and can also be used by the system directly without a direct user interaction.

*Object specification tool*: This tool allows the user to specify a search condition on an object within the image rather than on the image itself. This can be useful when the user is interested in an action being taken on an object in an image, for example, for enlarging a red car in the background of an image.

*Measurement specification tool:* When the size of the objects on an image are important, the user can also perform online measurements and permit tolerance specification from these measurements, which can, in turn, be used for query specification and processing.

*Classification tool*: This tool would allow the user to specify grouping conditions based on a feature of interest. For example, the user would request all images that have the same texture or elongation.

*Spatial arrangement tool*: This tool would allow the user to specify an object and a location of interest. The system will use this information to find the object in the database of images that satisfy the location that the user had specified.

*Data definition tool:* The user can use this tool to modify an example image that is being presented to the search engine. One possible query could be, for example: "Find all mug shot with the similar facial features, but with bigger eyes."

The simplest query specification strategy is a pixel-based query. A query that uses this approach would look like "Find all images for which the $100^{th}$ to $200^{th}$ pixels are orange".

Another technique is the color-based specification. The spectrum of colors and saturation of gray levels for each color is calculated. The system computes a 2D-histogram of spectrum and saturation for each image. The system can then answer queries like: "Find all the images in which more than 30% of the pixels are sky blue and more than 25% of the pixels are grass green."

The last specification scheme is shape-based. Most images are usually composed of constant spectral colors with little variation of color density. Each image can be segmented into a number of color regions. Each region contains a connected set of points having the same pure color. For each segment, four values are computed: color, area, the ratio of the perimeter square to area, and the distance of the centroid of the region from the center of the image. The system can then answer queries like: "Find all the images having a dominant white square in the center.

## 2.4 The CANDID System

### 2.4.1 Introduction

The CANDID [KEL95] (Comparison Algorithm for Navigating Digital Image Database) project uses a technique similar to the N-gram approach for document comparison. The system computes and maintains a global signature for each image in the database. The signature is composed of features that an image has such as color, shape, and texture. The user provides an example image to the system. The signature for the example image is computed and compared against all existing signatures to determine if there are similar images or not.

### 2.4.2 User Interface

The user provides the system with an example image. The features of the example image are calculated and compared to the information stored for each image in the database. Queries could take the form of: "Show me all images with areas of landcover similar to those in this example." The similar images are returned to the user in the order of similarity. The user can use the returned images for other queries. The user can also visualize why images are judged to be similar. A probability density function signature is used to display results in such a way as to indicate exactly where the matching occurred.

The CANDID approach is currently applied to the problem of retrieving multispectral satellite data (i.e. Landsat TM data). This is mainly to demonstrate queries to large databases such as, ``Show me all images of areas with landcover that is similar to this example."

A demo of this application is demonstrated below. Figure 2.8 shows the areas that a user can select to search for similar images. Figure 2.9 shows what the user will get as a result of selecting "Cairo" scene from the first images.

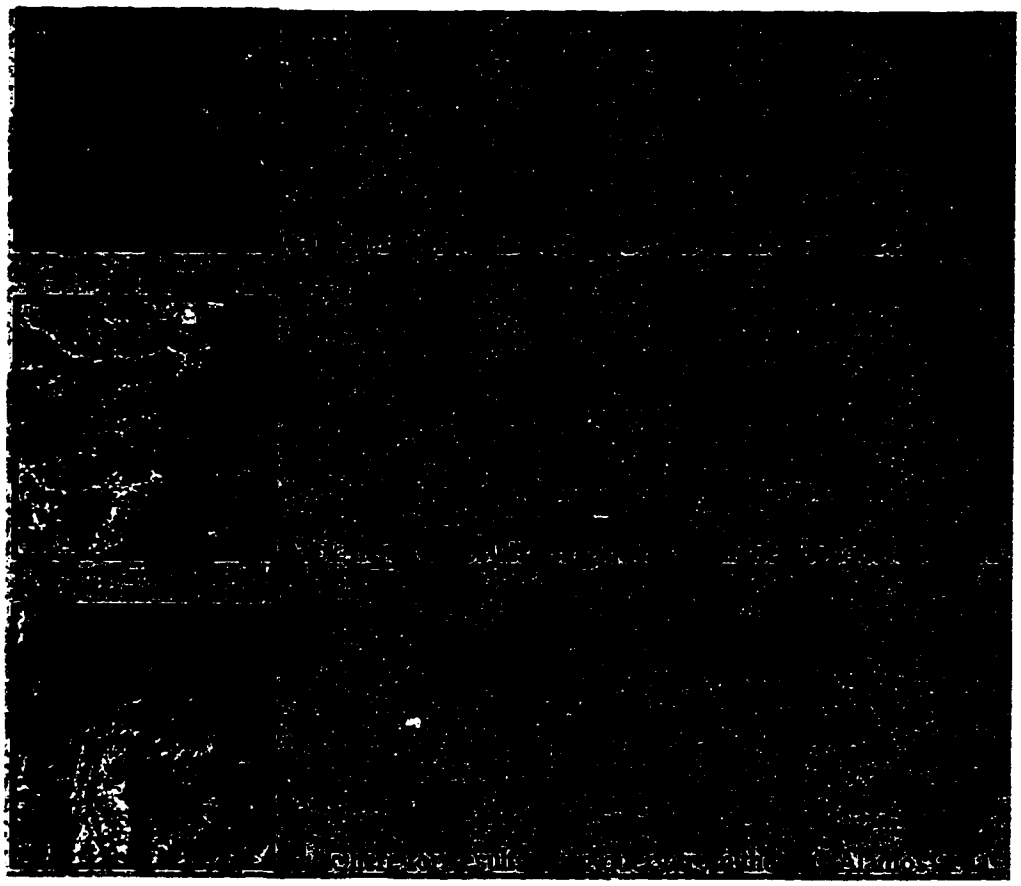


**Figure 2. 8**: The CANDID application shows the areas that a user can select to search for similar images.

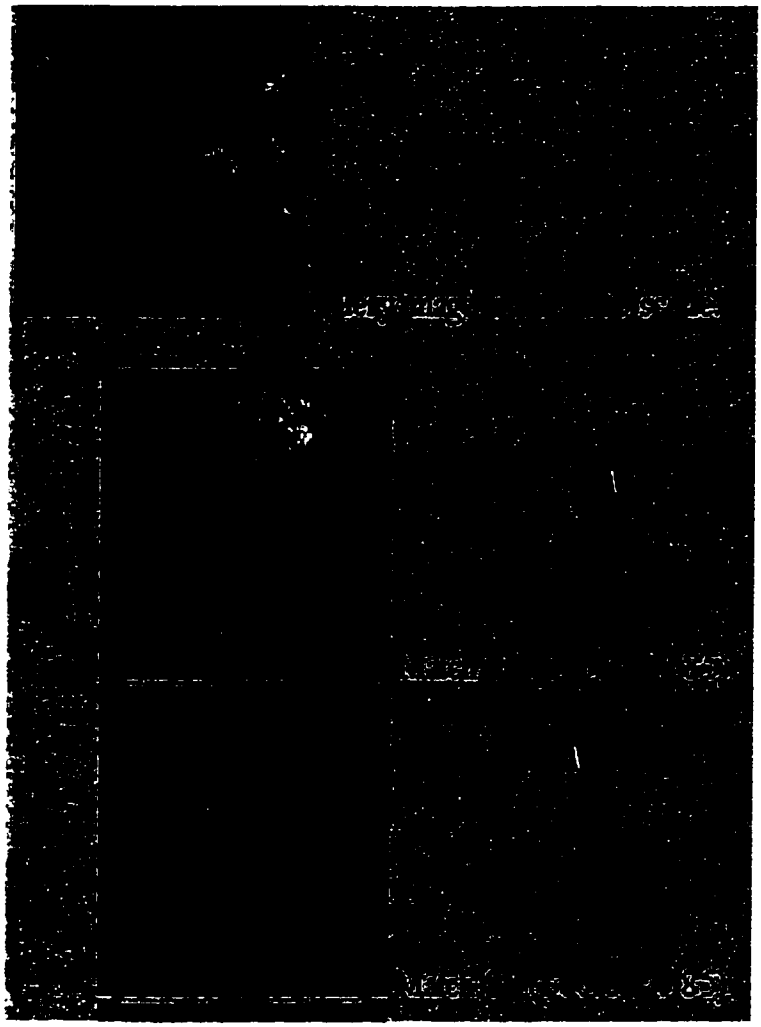


**Figure 2. 9**: The CANDID application shows the closest images similar to the "Cairo" scene from the first images.

## 2.4.3 Heuristic Techniques

The N-gram method for document comparison computes a global signature for each document in the database. This signature represents the information presented in the document. The signature is the histogram of the number of times that each substring of

length N occurs in the document. For example, a document that is characterized by only three-letter words can have $26^3$ or 17,576 possibilities. Therefore, the signature for a document with three-letter words is a vector of 17,576.

CANDID used the N-gram strategy to describe the image with a global signature. The signature is calculated for each image in the database from features that the image has such as it's color, shape, and texture. When the user provides an example image, the signature is also calculated for the example image and compared with the rest of the database.

Color, texture, and shape are first calculated at each pixel in the image. The probability density function that describes the distribution of these features and the image signature, is then calculated for these features. To achieve this the system assumes a Gaussian mixture to estimate the probability density function. Each Gaussian distribution function is defined by a mean vector $\mu_i$ and a covariance matrix $\sum_i$ . which are evaluated using a clustering methodology. The resulting Gaussian distribution function is weighted by the number of elements in its cluster. Any cluster that contains a single covariance matrix is deleted and not included in the subsequent calculations. After the Gaussian mixtures are analyzed, a signature over the specific *N*-dimensional feature spaces for an image *I* are represented as:

$$P_I(x) \approx \sum_{i=1}^{K} w_i G_i(x) \quad ; \qquad \text{where}$$

$$G_i(x) = (2\pi)^{-\frac{N}{2}} |\Sigma_i|^{-\frac{1}{2}} \exp[-\tfrac{1}{2}(x-\mu_i)^t \Sigma_i^{-1}(x-\mu_i)]$$

Two signatures can be compared using several comparison techniques. The $L_2$ distance measure is used to compare the continuous probability density functions represented by the Gaussian mixture.

$$dist_{L_2}(I_1, I_2) = [\int_{\Re}(P_{I_1}(x) - P_{I_2}(x))^2 dx]^{\frac{1}{2}}$$

The inner product technique is also used to compare continuous probability density functions. Such a quantity would be $sim(I_1, I_2)$ described below:

$$sim(I_1, I_2) = \int_{\Re} P_{I_1}(x) P_{I_2}(x) dx$$

where a result of 0 from this equation implies that the two images are not similar. This measure is often normalized and interpreted as a cosine of the angle between the distance and the inner product methods as:

$$nsim(I_1, I_2) = \frac{\int_{\Re} P_{I_1}(x) P_{I_2}(x) dx}{[\int_{\Re} P_{I_1}^2(x) dx \int_{\Re} P_{I_2}^2(x) dx]^{\frac{1}{2}}}$$

## 2.5 The QBIC System

### 2.5.1 Introduction

QBIC [NIB95] uses image analysis to process queries in an image database. This project uses color, shape, and textures to match images in the database to a user query of the form "Find more pictures like this one."

### 2.5.2 User Interface

The user can sketch a shape, select colors and color distribution from a color wheel, select a sample image, or select texture from a predetermined range. The system returns a ranked list of best matches to the user query. The following is found in the demonstration site of QBIC at: http://www.hermitagemuseum.org/. Figure 2.10 is the color search method. Figure 2.11 is the layout search method.

1.Use your mouse to select a color from the palette.
2. Click the arrow button to add the color to the bucket.
3. Slide the triangular handles on the bucket to adjust the percentage of this color.
4. You may repeat this process until the bucket is full.
When you are ready, click Search. You may also use the Color Mixer to adjust RGB (red, green, blue) values to use in your search.
Click Delete to remove a color from the bucket.
Click Clear All to empty the bucket.

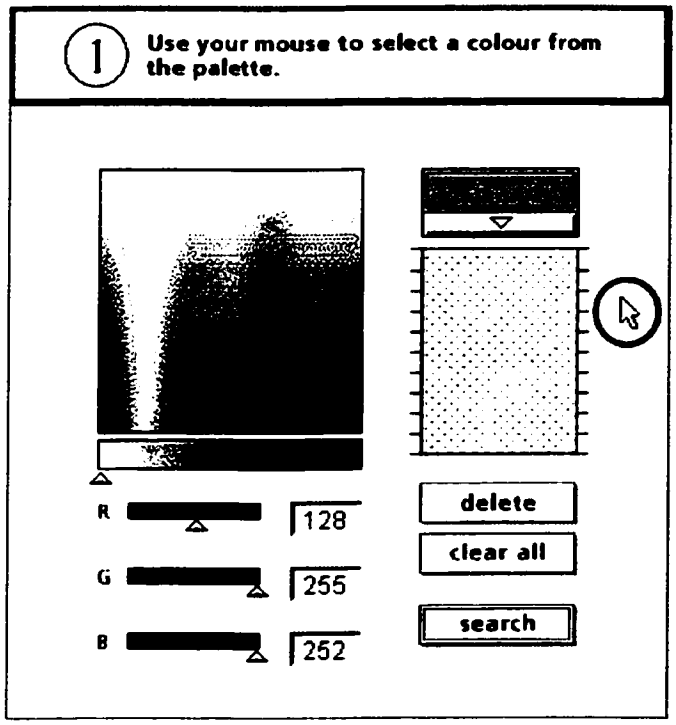



**Figure 2. 10**: Step one of QBIC User Interface. The user may select a color from the palette.

1. Use your mouse to choose a color from the palette.
2. Select either the round tool or the square tool.
3. Hold down your mouse button and drag the cross on the canvas to create a colored shape.
4. Repeat this process until you complete your custom layout. When you're ready, click Search.

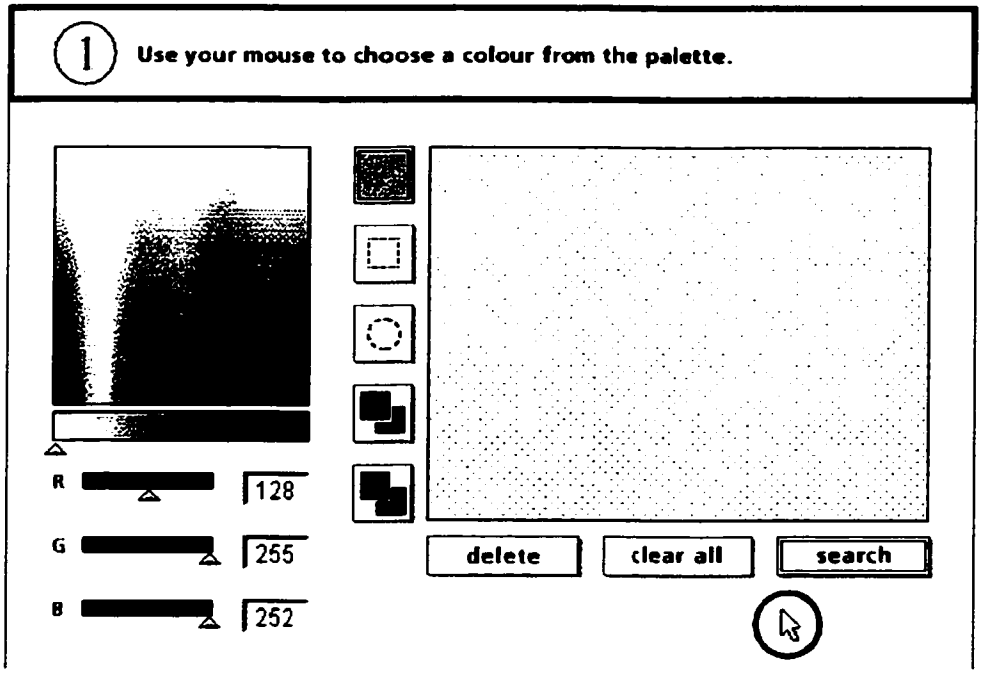



**Figure 2. 11**: Step two of QBIC User Interface. The user may select the color and specify a desired layout.

## 2.5.3 Heuristic Techniques

The main two components of QBIC are the database population module and database query processing module. During database population, the feature extraction from the images takes place. As mentioned earlier, the image features include colors, texture, and shape. These features are stored in the database along with the image record. On querying the database, the user specifies the query and QBIC searches the database for similar images.

QBIC uses foreground/background modeling, object segmentation, filtering and indexing techniques to speed up query processing while comparing a query image to the database of images.

Foreground/background and object segmentations are done automatically. Objects can also be identified semiautomatically. An enhanced flood-fill technique is used which starts at a single object pixel and repeatedly adds adjacent pixels whose values are within some given threshold of the original pixel. The threshold is determined from the user interaction by clicking on the background as well as object points. This technique works best for objects that are very distinct from the background color of the image. The QBIC system also uses another technique that is based on the snake concept developed in computer vision research. The user can track object edges by drawing a curve or a line on the image. The curve or line is then aligned to the object edges.

Filtering is applied to all image data and only the items that pass the filter are used in the next step of the true similarity matrix computation. For example, in the described prototype with a database of 10,000 images, the filter first filters the closest 1000 histograms to the query image histogram. The filtered histograms are passed to the next level, which is also a more specific filter, to further reduce the number of selected images.

## 2.6 The CHABOT System

### 2.6.1 Introduction

Chabot [OLG95] is an image search and retrieval project that uses textual description of the image as well as color analysis techniques to construct a query result. The project was originally developed for a specific client who required image editing or database loading in addition to searching for a required image.

### 2.6.2 User Interface

The user selects the features of interest to be included in the image search. Features of the query could be subject, location of where the picture was taken, description, colored or not, color histogram, entry date, and shoot date. Once features are selected, the user can look up the images that match the search. The user can also build, define, or use predefined concept queries, which include descriptive terms like "sunset" and "snow." Concept queries can also be used in conjunction with other criteria. For example, the user can specify a query such as "Find pictures of Lake Ontario at sunset," and the matching images are displayed to the user. The following is the first user interface screen for CHABOT.

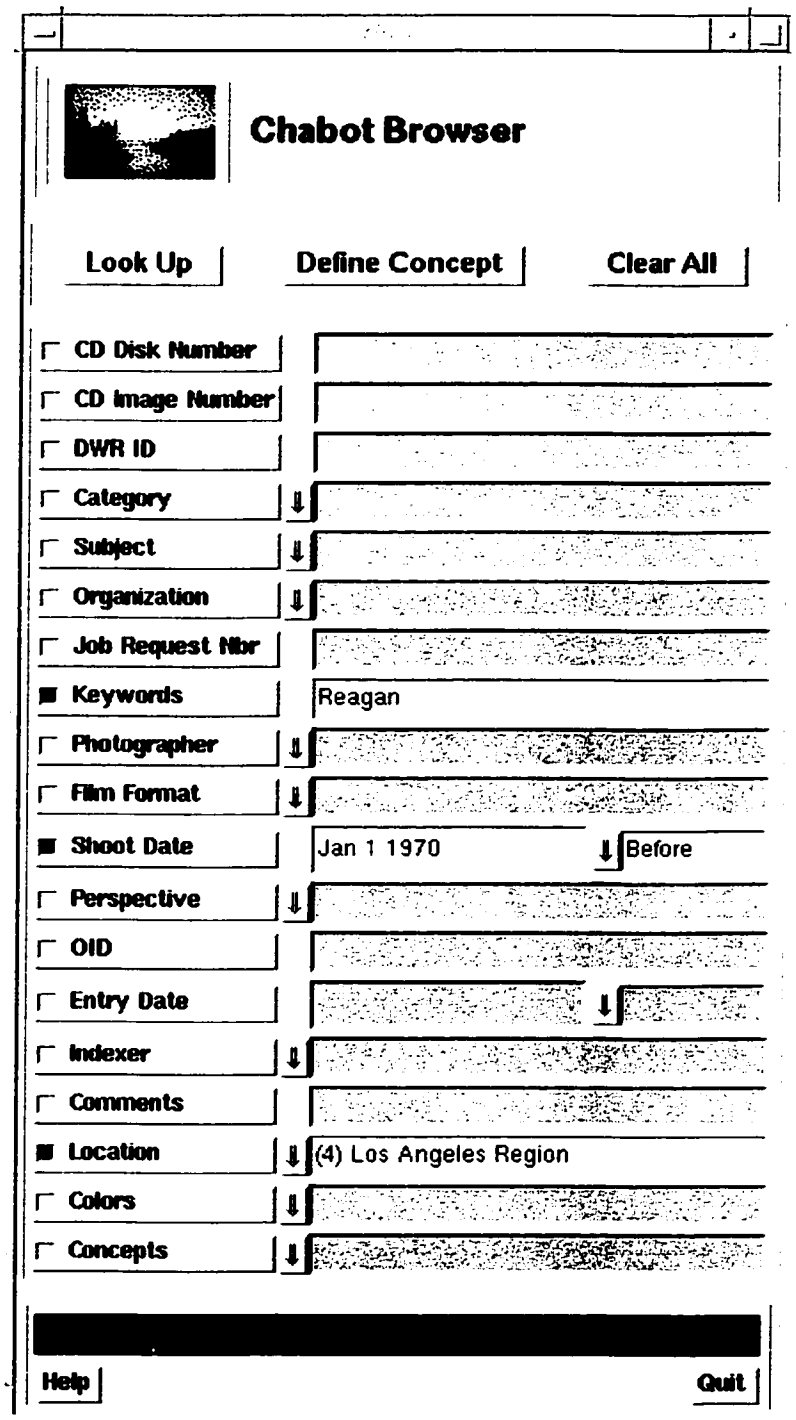



**Figure 2. 12**: CHABOT User Interface. The user may specify the desired attributes of an image.

### 2.6.3 Heuristic Techniques

Each image is stored in the database with its textual description given by the operator. The content of the image, mainly a color histogram, is computed at entry time. The function MeetsCriteria is used at run time to perform a concept query. MeetsCriteria takes two arguments: color criterion, such as "some blue," and the color histogram. The user selects the color criterion from a pull-down menu of suggested color criteria. The histogram data is obtained from each image at entry time. The 20 colors with the most pixels are selected to build the histogram. This histogram can be used in MeetsCriteria in conjunction with the selected color criterion to decide whether this image meets the criteria or not. Consequently, each image in the database must be compared to the desired color criterion. MeetsCriteria uses two techniques to improve the outcome of the color criterion screening: compliance and count.

Compliance predefines values for each color. For example, in the RGB model, 255, 255, and 255 for red, green, and blue represent the color white. Any color can be classified as white if it has all RGB values above 241.

If a given color meets the compliance test, it is checked against color criteria. The color criteria test is used for the "Some" color test. For example, "Some Yellow" has the value "True" if only one or two colors in the image histogram qualify as yellow. The total number of pixels in the matching color is also checked against the total number of pixels in the image. This, in turn, is used for the "Most" matches type of query. For example, an

image is qualified as "Mostly red" if more than 50% of the image pixels are red compliance.

Concept queries use MeetsCriteria for color analysis in conjunction with some other textual criteria. For example, if the user selects "sunset," the following query is constructed:

*retrieve(q.all) from q in photocd_bib where*

*q.description~"sunset" or*

*MeetsCriteria("MostlyRed", q.histogram) or*

*MeetsCriteria("MosltyYellow", q.histogram) or*

*MeetsCriteria("MosltyPurple", q.histogram)*

## 2.6 Pass and Zabih's Method

### 2.6.1 Introduction

Pass and Zabih [PAS97] describe a method that adds additional constraints to the histogram-based matching schemes. In the histogram refinement methods, the pixels within a given bucket are split into classes based upon various local properties. Histogram buckets are partitioned based on spatial coherence. The coherence measure classifies pixels as either coherent or incoherent. A coherent pixel is part of a sizeable contiguous region, while an incoherent pixel is not. A color coherence vector represents

this classification for each color in the image.

### 2.6.2 User Interface

The user interaction strategy of this technique is not discussed in [PAS97] but we assume that the user will have to provide the system with a model of the target image by either sketching an image or selecting a similar image from the database.

### 2.6.3 Heuristic Techniques

First, all images have to be scaled to have the same number of pixels $M$. An image color histogram $H$ is a vector $(h_1, h_2, \ldots, h_n)$ in which each bucket $h_j$ contains the number of pixels of color $j$ in the image. The histogram refinement technique used involves subdividing the pixels into classes based on local features. The features in this case include texture, orientation, distance from the nearest edge, and relative brightness. Histogram refinement prevents two pixels in the same bucket from matching each other if they do not fall in the same class. Pixels within a class are compared using any standard method for comparing histograms such as the $L_1$ distance. A positional refinement, for example, is when each pixel in the color bucket is classified as being either in the center or not. This produces a split histogram in which the pixels of color buckets are constrained by their location in the image, where the $L_1$ distance can be used to compare the split histogram.

Color coherence vectors create a finer distinction than plain histogram refinement. In color coherence vectors, histogram buckets are partitioned based on spatial coherence. Each pixel is classified as either coherent or incoherent. A pixel is considered coherent if it is in a sizable contiguous region. A color coherent vector is essentially this classification for each color in the image. To compute the color coherence vectors, the image is first manipulated by replacing pixel values with the average values in the small local neighborhood of 8 pixels.

The colorspace is then tightened so as to consider only $n$ distinct colors in the image. Pixels are then classified within a given bucket according to color coherence where a color coherent pixel is part of a large group of pixels of the same color. The pixel groups are determined by computing connected components. A connected component $C$ is a maximal set of pixels such that for any two pixels $p$, $p'$ in $C$, there is a path between $p$ and $p'$. The connected component is computed using 4 connected neighbors within a given tightened color bucket. A pixel is classified as being either coherent or incoherent based on the number of connected components. A pixel is coherent if the size of its connected component exceeds a fixed value. The authors of [PAS97] report that color coherence vectors give better refinement results which makes them more accurate when comparing two images with the same number of objects when only one of them has coherent pixels and the other contains incoherent pixels.

## 2.7 The Photobook System

### 2.7.1 Introduction

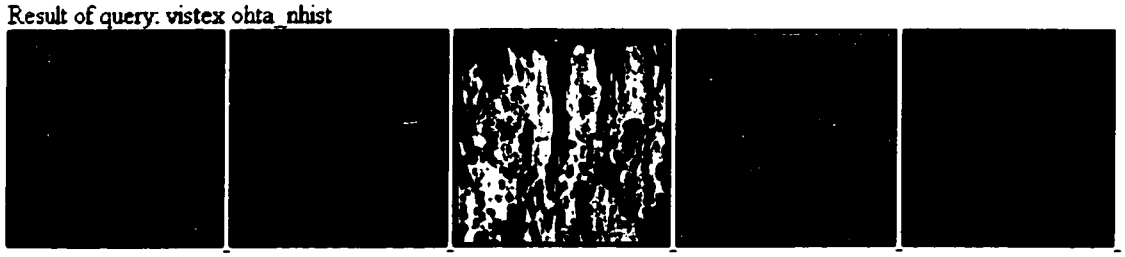
The Photobook [PEN94] project at the MIT Media Lab attempts to circumvent the issue of using predetermined search criteria by storing enough information about each image to make run time computations more efficient and feasible. Images are classified at load time as having face, shape texture or properties. Once classified, the image is compressed by encoding salient semantic information according to the specified categories. These smaller encoded images are used at query time to reconstruct the image and compute any additional search criteria, such as the color histogram.

### 2.7.2 User Interface

The user first selects the class of images of interest. This can be done by a textual query; for example by specifying the keywords "sport cars". The text is then searched against the database of textual description for all the images. The located images are displayed in the first screen. The user can scroll back and forth between screens of images. The user can ask the system to sort the images with respect to a selected image or group of images. The "select" and "sort" procedures can be performed several times for better tuning. The Photobook system is demonstrated on the Internet. Figure 2.13 is what the user would see as an example images. The user clicks on any image to start a search. Figure 2.14 gives the result for clicking on one of the presented images. Figure 2.15 gives a different result for the same image with different similarity metric.

# Photobook Web Demo

Result of query: vistex ohta_nhist



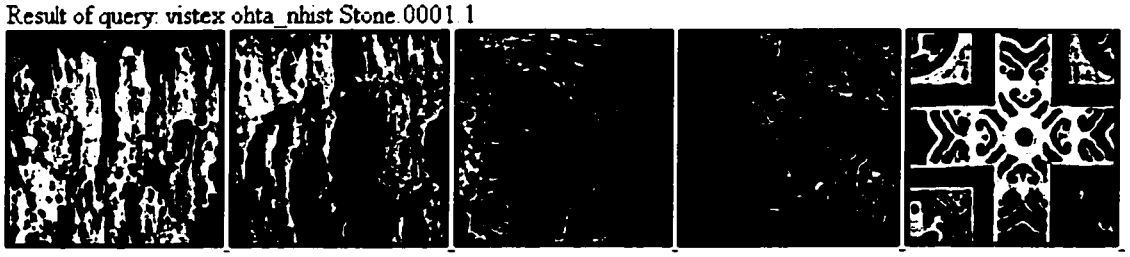
Click on an image to make a new query, or

Randomize
Change the number of images to display
5 10 15 20
Change the similarity metric (which image features to use)
ohta_nhist sar
Change the database
vistex feret faces

**Figure 2. 13**: Example images taken from the Photobook Web Demo example images.

# Photobook Web Demo

Result of query: vistex ohta_nhist Stone.0001.1



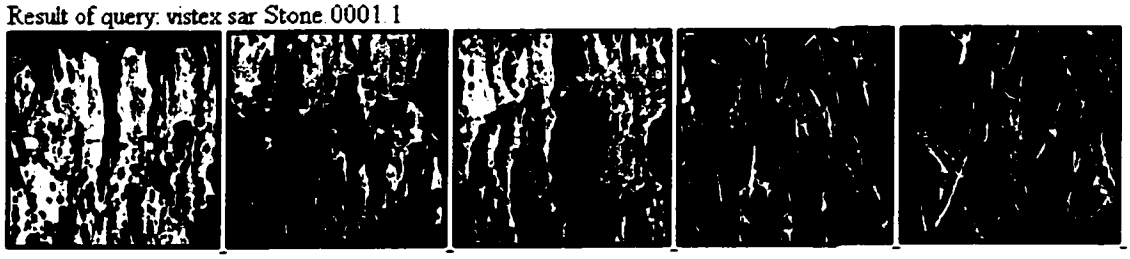
Click on an image to make a new query, or

Randomize
Change the number of images to display
5 10 15 20
Change the similarity metric (which image features to use)
ohta_nhist sar
Change the database
vistex feret faces

**Figure 2. 14:** Results from the Photobook Web Demo display similar images obtained after selecting one image.

# Photobook Web Demo

Result of query: vistex sar Stone.0001.1



Click on an image to make a new query, or

Randomize
Change the number of images to display
5 10 15 20
Change the similarity metric (which image features to use)
ohta nhist sar
Change the database
vistex feret faces

**Figure 2. 15**: Results from the Photobook Web Demo displaying similar images obtained after selecting the same image but with different similarity metric.

## 2.7.3 Heuristic Techniques

Photobook stores enough information about each image to make run time computations fast. Images are classified at load time as having face, shape, texture or other properties. Once an image is classified, it can be compressed by encoding salient semantics information according to the respective category. These encoded images can be used at query time to reconstruct the image and compute any relevant additional search criteria.

The Photobook system extracts the foreground objects from the image by using clustering, optical flow, and color difference information. The extracted foreground objects can be analyzed to come up with a description that can be inserted into the

database of images. Besides foreground extraction, Photobook also uses object detection. The system defines prototypes for objects of general interest, for example, a human face. The search for a given object in an image looks for an object that will match the prototyped object with some expected variations. The user can tune these variations by including or excluding a certain range of objects.

Three main techniques are used to describe an image. The Appearance Photobook is applied to face and keyframe databases. The Texture Photobook is applied to texture-swatch and keyframe databases. The Shape Photobook is applied to hand-tool databases, and databases of different kinds of fish.

The Appearance Photobook tries to measure similarities in appearance by extracting certain features to describe an image. Images are first normalized for position, scale, orientation, and similar effects. Eigenvectors of normalized image covariances are calculated for a set of training images and subregions of the training images, and this is achieved by calculating the eigenvectors for the whole image and it's components. Appearance Photobook represents the input data in terms of its varaitions from the prototyped image for a given class. If the class of the image is not known from the textual description, appearance can be used to compare each image with the appearance models for each class.

The Shape Photobook takes into consideration the variations of the same objects

in two images. The variations include different viewing perspectives of the same object, different degrees of closeness to the object, and different degrees of stretching or denting for an image.

The approach of the Appearance Photobook to is to build models for model shapes. The shape model is built of virtual material that gathers features, edges and corners, into a model. This approach uses the finite element method, that computes a positive-definite symmetric matrix, called the stiffness matrix. The stiffness matrix describes how each point on the object is connected to every other point. The eigenvectors of the stiffness matrix are calculated and used to encode deformations relative to some base or average shape. The eigenvector-based shape description is used to compare shapes by looking at the amplitudes of the eigenvectors.

The Texture Photobook fills the gap that the Appearance Photobook and the Shape Photobook have left regarding texture similarities between images. The Photobook requires a texture model whose parameters reflect the similarities between two images. The model is successful if the distance between its parameters corresponds to ordering images by their similarity. The Photobook uses the Wold decomposition for regular stationary stochastic processes in 2-D images to develop a texture model. If an image is assumed to be a homogeneous 2-D discrete random field, then the 2-D Wold-like decomposition is the sum of three mutually orthogonal components: a harmonic field, a generalized-evanescent field, and a purely indeterministic field.

# 2.8 VisualSEEk

## 2.8.1 Introduction

The VisualSEEk system [SIM97] attempts to integrate feature-based image indexing with spatial query methods. The integration relies on the representation of color regions by color sets.VisualSEEk uses indexing on raw image data. Prior to queries, images in the database are processed once to extract regional color information or salient color regions. Along with color information, region sizes and spatial locations of the regions are also extracted and are used as features to index images. To select the images matching with the query image, the color regions and location information of the query image are compared against all the images in the database.

## 2.8.2 User Interface

The user is expected to provide a sketch of regions, position them on the query grid and assign them properties of color, size, and absolute location. The user may choose to assign boundaries for location and size. A typical query interface in VisualSEEk is illustrated in the following figure.

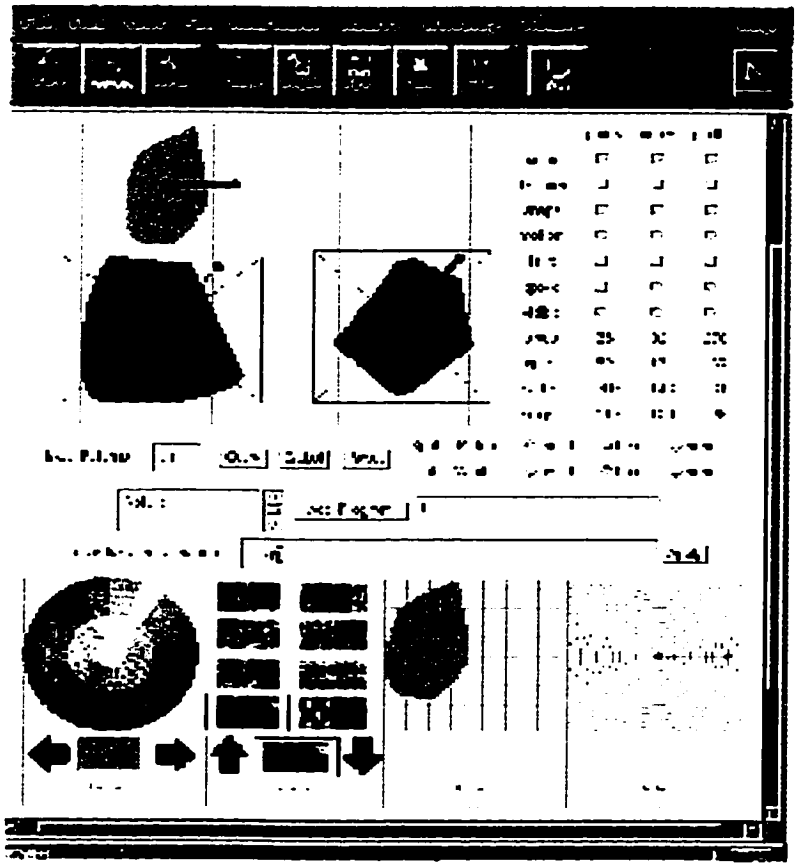


**Figure 2. 16**: The main user interface of VisualSEEk.

The user can specify the query as in the upper images and have the result displayed in bottom images. The first image is an example of the absolute location of a single region. The second is specifies two absolute regions. The third specifies the spatial relationship of three regions. The fourth specifies both absolute and relative locations of the regions.

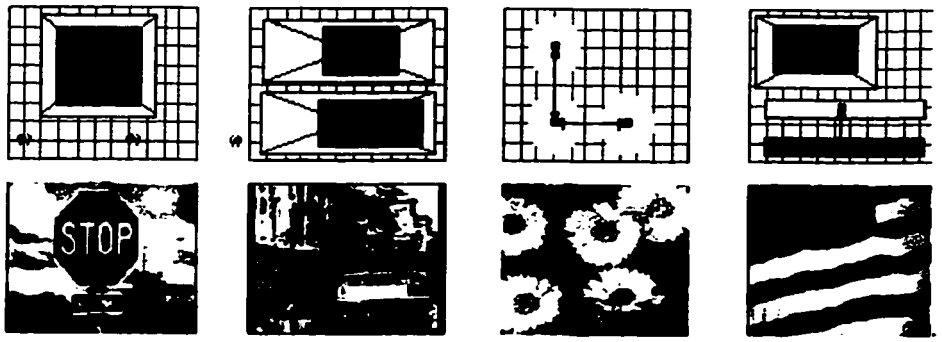



**Figure 2. 17**: The user Interface VisualSEEk. The user may specify spatial properties of regions.

### 2.8.3 Heuristic techniques

Image similarity is constructed from original and derived parameters. Original parameters are the image's colors, region sizes, and spatial locations. The derived parameters are the relationships that can be constructed from the original parameters such as relative spatial locations and overall assessment of image matches consisting of multiple regions. Similarity among images' colors can be matched using color sets, which is a compact version of the color histograms. A color set is a binary vector that corresponds to a selection of colors. The RGB of each color is transformed into HSV to represent the indices for its hue, saturation, and value. A unique number is assigned to each quantified HSV color. The color set back projection is used to extract color regions. Information about regions such as the color set used for back-projection, the spatial location, and size, are added to the REGION relation. Other attributes are also added to the region relation such as the x-y coordinates, width, height, and minimum-bounding rectangle. The information is later used during query generation. A query for a color set can be in the form of "find the best match to color set A". A maximum tolerance is also

defined for the color set distance. The query is executed by performing several range queries on the query color set's colors, taking the intersection of these lists and minimizing the sum of attributes in the intersection list. The best match minimizes the color set distance.

Spatial properties of the individual regions in the query are used for indexing of region centroids and the minimum bounding rectangles. The user is given the flexibility in designating the spatial bounds for each region in the query within which a target region is assigned a spatial distance of zero. The overall region query strategy consist of computing individual queries on color set, region location and area, and spatial extent, as specified by user. The user may assign a relative weighting to each of the attributes of the region. For example, the user may weight the size parameter more heavily than the color and location in the query. The absolute region and color can also be joined in one query.

The overall image query strategy consists of joining the queries on the individual regions in the query image. The join, which consists of the intersection of the results of region matches, identifies the candidate target images which contain matches to all the query regions. For these images, the image match score is computed by adding the weighted scores from the best region matches. In the final stage, the relative spatial locations are evaluated to determine the best match image that satisfies the constraints of relative region placement.

## 2.9 Wavelet coefficients based techniques

### 2.9.1 Introduction

The analysis of wavelet coefficients [ALB98] is a technique used in an image database system to decompose each image into a set of features that can be related to other images features. The authors [ALB98] argue that, for a large database of images, the sequential comparison of image features for all images is prohibitively time consuming and that there is a need for a transform-domain based indexing. The use of the transformation technique is more effective in characterizing image properties and improves performance. The authors have chosen the wavelet transform because it is known to provide a better frequency localization. A review of wavelets is inappropriate here; for more information about wavelets, the reader is advised to read [KAI94]. The literature reports a few image retrieval schemes using wavelet coefficients. This section presents a brief overview of these schemes.

### 2.9.2 User Interface

Unfortunately, we cannot present the user interaction sub-systems of these systems as they are not described in the relevant sources. Therefore, it is assumed that the user interaction is based on a query-by-image approach.

### 2.9.3 Heuristic Techniques

Mallat's [MAL89] method decomposes each image into another image that resembles the original image with half the resolution. It also calculates three coefficient matrices that contain detailed information about different frequency components referred to as subbands. The decomposition is recursively applied to the resembling image to produce progressively smaller images at progressively lower resolutions. Each image is then transformed into the wavelet domain, and the coefficients are available as a 2-D matrix that is partitioned into regions corresponding to the different subbands. Typically, the number of regions is proportional to the number of decompositions applied to the image.

Albuz *et. al.* [ALB98] use features derived from the sum of squares of wavelet coefficients in each subband while allowing significant coefficients to contribute more to the total energy. However, the feature vector is constructed using information from all the images in the database. Within each subband, the images that have similar energy levels are clustered together, and the properties of each cluster are used to construct the feature vector. The feature vector, in turn, consists of a key, where different portions of the key represent different subbands in the wavelet domain.

Jacobs *et.al.* [JAC95] suggest a computationally faster wavelet transform and use a fixed number of significant coefficients, say m, from multiple resolutions, as feature vector. These coefficients are organized into six linear-array data structures. Each pair of

arrays corresponds to a single color plane. Within each pair, if the magnitude of the coefficient is above a certain threshold, the corresponding image is listed in the "plus array" or "minus array" depending on the sign of the coefficient. Subsequently, for each query image, the same wavelet decomposition is applied, and the important coefficients are chosen. Each important coefficient is used to index the array data structures, and using them the image lists are retrieved.

Ma *et al.* [MAM95] have proposed a wavelet based method for retrieving texture patterns and using the energies of these patterns as feature vectors. A small subset from a large dictionary of Gabor filters is chosen to extract image features. Each image in the database is processed using all the filters in the set, and is characterized by the mean of the energy generated by each filter. Each image has a number of features like the mean and standard deviation. The number of features is equal to the number of filters. Given a query image, all filters are applied to it. The feature vector of the query image is compared with those of the images in the database. The selection of filters is done adaptively. After a candidate filter is applied to all the images in the database and to the query image, the energy of the query image is compared with the mean energy of all the images in the database. The filters providing the best results are selected.

Kuo *et al.* [LIA97] have proposed the use of four features from the wavelet domain: the frequency of the important coefficient in each subband, the luminance histogram extracted from successive quantization of the wavelet coefficients, the binary

quantization map of the coefficients, and color histogram of the coefficients. Each feature can be used independently or in conjunction with other features for the purpose of retrieval.

## 2.10 Conclusions

In this chapter, we have described the Image Retrieval Problem and discussed the various reported methods for image search and retrieval. Present day techniques for solving the image retrieval problem are based on one of two techniques. The traditional techniques utilize either textual description or visual descriptions for indexing and searching of images in the database. In some cases a combination of the two methods is used. The textual description method is time consuming, and the search result is dependent on the strategy by which the operator, (who adds the images to the database), describes the images. The visual description method does not cover textual similarity queries, such as "show me pictures of all political leaders." Several existing methods have been presented by explaining the user interface method and heuristic techniques used.

In chapter 5, a different and hopefully better solution to the Image Retrieval problem is presented. This solution uses learning automata and adaptively groups the images according to the user's subjective judgment.

# Chapter 3: LEARNING AUTOMATA-AN OVERVIEW

## 3.1 Introduction

Learning in general can be defined as the process by which a behavior is adjusted as a result of previous experience. The automaton approach to learning is concerned with predicting the best action offered by a random environment. The environment responds to the actions by a set of permitted outputs that are related to the input in a probabilistic manner. The field of learning studies how stochastic finite state machines can learn from such environments. Its applications include parameter optimization, statistical decision-making, telephone routing, pattern recognition, game playing, natural language processing, and the modeling of biological learning systems. This thesis presents a learning automaton approach to solve the object-partitioning problem, which is later utilized for image retrieval.

In 1961, the Russian mathematician Tsetlin invented a new model of computer learning that is now known as the *learning automaton*. Tsetlin's proposal of a learning automaton was a scheme by which one would determine the optimal action out of a set of allowable actions. The learning automaton operates based on a sequence of repetitive feedback cycles in which the automaton interacts with the environment. The environment

offers the automaton a finite set of actions. The automaton chooses an action that triggers a response from the environment. The environment responds with either reward or a penalty. The automaton uses this response and the knowledge acquired from the past actions and responses to determine which is the next action.

The term *environment* can be defined as the sets of all the external conditions and factors that have impacts on the life and development of an organism or system. However, the environment can not easily be defined in the terms of learning automata. For the learning automata to define the environment, a large class of general unknown media in which an automaton or a group of automata can operate is defined. To simplify issues, an environment is mathematically, defined by the triple $\{\underline{\alpha}, \underline{c}, \underline{\beta}\}$, where:

- $\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the set of inputs.
- $\underline{c} = \{c_1, c_2, \ldots, c_r\}$ is the set of penalty probabilities.
- $\underline{\beta} = \{\beta_1, \ldots, \beta_k\}$ is the set of output. If k = 2, the output is represented by the set {0,1}. The response '1' is said to be a penalty, and '0' a reward.

## 3.2 Definition of Automaton

The automaton can, in a systematic way, select the input actions from the outputs of the environment to increase the occurrence of favorable response. Mathematically, an automaton is defined as a set of inputs $\underline{\beta}$, a set of states Q, a set of output actions $\underline{\alpha}$. It also defines the functions F and G that are needed to compute the next state of the automaton and its output respectively. The input given to the automaton and the current

state of the automaton determine how the automaton will move from the current state to the next state, as shown in Figure 3.1.

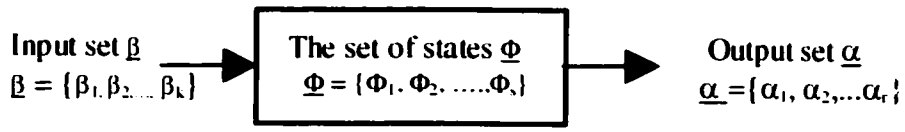



**Figure 3.1**: The automaton.

The following is a formal definition of an automaton:

**Definition 3.1:** An *automaton* is a quintuple $<\underline{\alpha}, \underline{\beta}, \underline{\Phi}, F, G>$, where:

- $\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the set of output actions of the automaton.
- $\underline{\beta} = (\beta_1, \beta_2, \ldots, \beta_k)$ is the set of input actions that can be finite or infinite.
- $\underline{\Phi} = (\Phi_1, \Phi_2, \ldots, \Phi_s)$ is the vector state of the automaton with $\Phi(n)$ denoting the state at instant $n$.
- $F: \Phi \times \beta \rightarrow \Phi$ is the *transition function* that determines the state at the instant $n+1$ in terms of the state and input at the instant n:

  $$\Phi(n+1) = F(\Phi(n), \beta(n)).$$

This mapping can be either deterministic or stochastic.

- The output function G determines the output of the automaton at any instant 'n' based on the state at the current instant:

  $$\alpha(n) = G(\Phi(n)).$$

The mapping $G: \underline{\Phi} \rightarrow \underline{\alpha}$ can, with no loss of generality, be considered deterministic [NAR89].

The automaton is considered to be *finite* if the sets $\underline{\alpha}$, $\underline{\beta}$ and $\underline{\Phi}$ are all finite.

An automaton can be either classified as deterministic or stochastic as explained in the next section.

### 3.2.1 Deterministic Automaton

The automaton is a *deterministic automaton* if both F and G functions have deterministic mappings. A deterministic automaton that starts with an initial state and input will have a uniquely determined next state and action.

### 3.2.2 Stochastic Automaton

The automaton is called *stochastic* if at least one of the F and G function is stochastic. If the state transition mapping F is stochastic, given a current state and input, the next state is not uniquely determined. It can be determined randomly by calculating the probabilities of reaching various states. The function F is determined in terms of the conditional probability matrices $F(\beta_1)$, $F(\beta_2)$, ..., $F(\beta_m)$, where each $F(\beta)$, for $\beta \in B$ is a $s \times s$ matrix whose entries are given by:

$$f_{ij}^{\beta} = \Pr\{\Phi(n+1) = \Phi_j \mid \Phi(n) = \Phi_i, \beta(n) = \beta\}, \qquad \begin{array}{l} i = 1,\ldots,s \\ j = 1,\ldots,s \\ \beta = \beta_1, \beta_2, \ldots, \beta_m. \end{array} \tag{3.1}$$

Therefore, the quantity $f^{\beta}_{ij}$ specifies the probability that the automaton moves from state $\Phi_i$ to $\Phi_j$ on receiving the input $\beta$. Since $f^{\beta}_{ij}$ specifies a probability which dictates that at the next step the automaton has to reach one of its states, we have:

$$\sum_{j=1}^{s} f_{ij}^{\beta} = 1, \text{ for each } \beta \in B, \tag{3. 2}$$

and therefore $F^{\beta}$ is a Markov matrix.

The mapping of the stochastic G can be represented by a conditional probability matrix of dimension $s \times r$ having the following elements:

$$g_{ij} = \Pr\{\alpha(n) = \alpha_j \mid \Phi(n) = \Phi_i\}, \qquad i = 1,\ldots,s, \;\; j = 1,\ldots,r. \tag{3.3}$$

This basically specifies the probability that the automaton chooses $\alpha_j$ if it is in state $\Phi_i$. Since $g_{ij}$ are probabilities, it implies:

$$\sum_{j=1}^{r} g_{ij} = 1, \text{ for each } i = 1,\ldots,s. \tag{3. 4}$$

The family of *Fixed Structure Stochastic Automaton* (FSSA) are those in which the conditional probabilities $f^{\beta}_{ij}$ and $g_{ij}$ are independent of both n and the input sequence. The automaton is called a *Variable Structure Stochastic Automaton* (VSSA), if the transition probabilities, $f^{\beta}_{ij}$ vary based on the time instant n,

It is not possible to accurately determine the state of the automaton if the transition mapping function is stochastic. Only a calculation of the probability with which the automaton is in particular state at a given instant can be done. These state probabilities are known as *state probabilities*. The *state probability vector* can be defined

as $\pi(n) = [\pi_1(n), \pi_2(n), \ldots, \pi_r(n)]^T$, where $[]^T$ denotes the transposed matrix and

$$\pi_j(n) = \Pr\{\Phi(n) = \Phi_j\}. \tag{3.5}$$

## 3.3 The Automaton-Environment Feedback

The goal of an automaton as outlined, is to determine the optimal action out of a set of allowable actions. The automaton performs its actions in a random environment that, in turn, generates a response for that action. The environment responds to the input action by producing an output from a set of allowable outputs that is probabilistically determined based on the input action. The characteristics of the environment will be described in the following subsection.

### 3.3.1 The Environment

In the context of learning automata, the environment can be defined by a triple $\{\underline{\alpha}, \underline{C}, \underline{\beta}\}$ where $\underline{\alpha}=\{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ represents a finite input set, $\underline{\beta}=\{\beta_1, \beta_2, \ldots, \beta_l\}$ $(2 \le l < \infty)$ is the output set of the environment, and $\underline{C} = \{c_1, c_2, \ldots, c_r\}$ is a set of penalty probabilities, where each element $c_i$ of C corresponds to an input action $\alpha_i$.

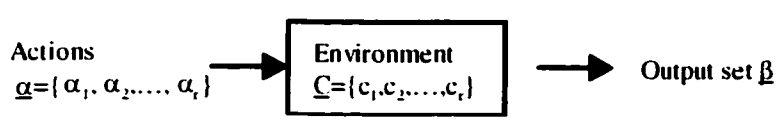



**Figure 3.2:** The environment

If the environment receives an input $\alpha(n)$ at time n it will respond with an output $\beta(n)$. The output set that the environment responds with is a set of two elements $\beta_1$ and $\beta_2$, which is usually either 0 or 1. An output of $\beta(n)=1$ is said to be a *failure*, an *unfavorable response* or a *penalty* and an output $\beta(n)=0$ is said to be a *success*, a *favorable response* or a *reward*.

A *P-model* system is a system that interacts with an environment that generates only two output values: either 0 or 1. A *Q-model* system is a system that interacts with an environment that generates an output of a finite set. The broadest kind of system is a system that interacts with an environment that has a continuous random variable, which assumes values in the interval [0,1]. This model is referred to as the *S-model*.

Another classification of the type of environment can be made based on its evolutionary properties. A *stationary environment* is an environment that has the penalty probabilities $c_i$ (i=1,2,....,r) constant with respect to time. If any of the penalty probabilities $c_i$ (i=1,2,...,r) vary with time, the environment is classified as *nonstationary*.

### 3.3.2 Definition of the Automaton-Environment Interaction

When an automaton interacts with a random environment with the goal of improving its future behaviors based on environment's feedback, this automaton is referred to as a learning automaton. Figure 3. shows the environment and the automaton

connected in a feedback loop in such a way that the input of the automaton is the output of the environment, and the output of the automaton is the input of the environment.

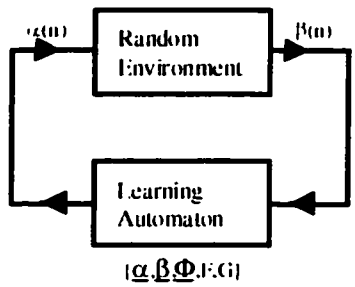



**Figure 3.3:** Feedback connection of the automaton and environment.

The automaton begins with an initial state $\Phi(0)$ and it generates an action $\alpha(0)$, which corresponds to the initial state. The environment responds to this action by $\beta(0)$. This response forces the automaton to switch its state based on its transition matrix F, the state would change to $\Phi(1)$. This cyclic process keeps going on until the probability of choosing the action that has the smallest penalty probability become as close to unity as desired.

### 3.3.3 Norms of behavior

If previous automaton-environment interaction information is not available, an action is chosen with equal probability, by pure chance. The pure chance should be the baseline for comparison with any other automaton. By doing such a comparison, we can qualify the behavior of the automaton operating in a stationary random environment with the penalty probabilities $\{c_1, c_2, \ldots, c_r\}$. Comparing two automata operating in such an environment, the automaton that receives a bigger number of favorable responses from

the environment is considered more efficient. Therefore, a superior automaton is the one that learns to choose the "best" action and avoids the actions that cause more penalty responses from the environment.

There is one simple approach by which an automaton could learn to choose the best action based on the pure chance approach. If there is no available information about each previous action, the automaton can not distinguish between the different actions. Therefore, the automaton will choose each action with equal probability $p_i(n) = 1/r$, $i=1,2,\ldots,r$. This kind of automaton, that uses this simplistic approach, is called a "pure-chance automaton", and it is considered a baseline for comparison of the behavior of all learning automata systems.

To achieve a comparison of various learning automata, the *average penalty* for a given action probability vector **P**(n) at time 'n' is defined as:

$$\begin{aligned} M(n) &= E[\beta(n) \mid \mathbf{P}(n)] = \Pr[\beta(n) = 1 \mid \mathbf{P}(n)] \\ &= \sum_{i=1}^{r} \Pr[\beta(n) = 1 \mid \alpha(n) = \alpha_i] \cdot \Pr[\alpha(n) = \alpha_i] \\ &= \sum_{i=1}^{r} c_i p_i(n) \end{aligned} \tag{3.6}$$

In a "pure-chance" automaton, the average penalty is calculated to be [NAR89]:

$$M_0 = \frac{1}{r}\sum_{i=1}^{r} c_i \tag{3.7}$$

Any automaton with an average penalty M(n) that is smaller than $M_0$ at least asymptotically is considered better than the pure-chance automaton.

As M(n) and $\lim_{n \to \infty} M(n)$ are random variables, one can compare E[M(n)] with $M_0$, where

$$E[M(n)] = E\{E[\beta(n) \mid \mathbf{P}(n)]\} = E[\beta(n)] \tag{3.8}$$

Therefore, based on the comparison with the pure-chance automaton, any automaton that performs better than the pure-chance automaton is considered *expedient*. A formal mathematical definition can be expressed as follows:

**Definition 3.1:** A learning automaton is considered to be *expedient* if

$$\lim_{n \to \infty} E[M(n)] < M_0. \tag{3.9}$$

The behavior of an automaton can be characterized for all time instances by the following definition:

**Definition 3.2:** A learning automaton is said to be *absolutely expedient* if

$$E[M(n+1)P(n)] < M(n), \tag{3.10}$$

where **P** is the probability vector and M(n) is the probability of penalty at instant 'n'.

The absolute expediency condition imposes an inequality on the expected penalty probability M(n) at instant n. Taking expectations again in equation (3.10) we obtain [NAR89]:

$$E[M(n+1)] < E[M(n)]. \tag{3.11}$$

which shows that E[M(n)] is actually decreasing with 'n' in all stationary random environments.

The goal of any automaton, as established earlier, is to learn to asymptotically choose the best action. If an automaton succeeds in achieving this goal, it is considered *optimal* automaton. Mathematically, optimality can be defined, in the context of learning automata, as follows:

**Definition 3.3:** A learning automata is considered *optimal* if

$$\lim_{t \to \infty} p_b(n) \to 1 \text{ with probability } 1, \tag{3.12}$$

where $p_b(n)$ is the action probability associated with the action having the minimum penalty probability $c_{min}$. However, it is currently not possible to come up with optimal learning automata. In this case, one might try to achieve a sub-optimal performance, termed as *ε-optimality*.

**Definition 3.4:** Let $\lambda$ be a learning parameter. A learning automaton is said to be *ε-optimal* if for every $\varepsilon > 0$ and $\delta > 0$, there exists $n_0 > \infty$ and $\lambda_0 > 0$ such that

$$\Pr\left[ | p_b(n) - 1 | < \varepsilon \right] > 1 - \delta \ , \tag{3.13}$$

for all $n \geq n_0$ and $\lambda < \lambda_0$.

The above definition of *ε-optimal* automaton states that given an internal parameter $\lambda$, usually depending on the number of internal states, if enough time is

provided, the probability of choosing the best action almost all the time can be made as close to unity as desired.

It has been proven that if an automaton is *absolute expedient*, then it is also *ε-optimal* in all stationary random environments [NAR89].

## 3.4 Fixed Structure and Variable Structure Learning Automata

Various kinds of learning automata have been presented in the literature, beginning with Tsetlin's pioneering paper in 1962 [TSE62]. Learning automata were first designed with time invariant transitions, and therefore their output functions being deterministic. Tsetlin, Krylov, and Krinsky [TSE62] presented distinguished examples of these automata. Later, the *Variable Structure Stochastic Automata* (VSSA) were designed. VSSA had time dependent state transition and output functions [NAR89].

### 3.4.2 Fixed Structure Automata

This section discusses some fixed structure learning automata. We start with a simple two-state automaton and enhance it to yield the superior machines.

### 3.4.2.1 The Two-State Automaton $L_{2,2}$

An automaton proposed by Tsetlin [TSE62] that has two states, $\{\phi_1, \phi_2\}$ and two outputs $\{\alpha_1, \alpha_2\}$, is shown in Figure 3.4.

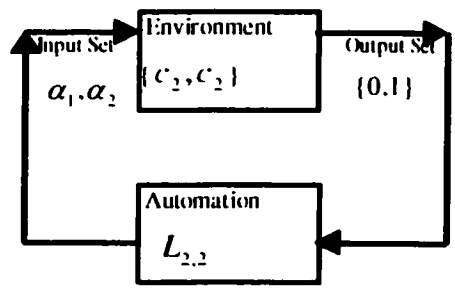



**Figure 3.4:** Two-State Automaton.

The automaton takes inputs from a set {0,1} and changes the states if it receives an input of 1, which is the unfavorable response. Otherwise, the automaton remains in the same state if it receives 0, which is the favorable response. Therefore, the transition matrices of such an automaton looks like the following:

$$F(0) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad F(1) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

This automaton is classified as $L_{2,2}$ to reflect that it uses 2 states when it has a choice of 2 actions. The environment is characterized by the set of penalty probabilities $\{c_1, c_2\}$ where $c_i$ $(i = 1,2)$ corresponds to the probability of getting a response $\beta = 1$ from the environment when the input is $\alpha_i$.

A pure-chance automaton operating in the same random environment would choose the input actions $\alpha_1$ and $\alpha_2$ with equal probability so that

$$M_0 = \tfrac{1}{2}(c_1 + c_2). \tag{3.14}$$

Assuming that at stage $n$, the automaton is in state $\phi_i$, the probability $f_{ij}$ that the automaton will make the transition from state $\phi_i$ to $\phi_j$ is determined from the formula

$$\bar{f}_{ij} = P_{ij} = c_i f_{ij}^1 + d_i f_{ij}^0 \tag{3.15}$$

where $d_i = 1 - c_i$ is the probability of obtaining a favorable response of the environment to an action $\alpha_i$.

### 3.4.2.2 Tsetlin $L_{2N,2}$ Automaton

Tsetlin has also suggested the modification of $L_{2,2}$ [TSE62]. The modified automaton is a deterministic fixed structure automaton, denoted as $L_{2N,2}$, with 2N states and 2 actions, *i.e.* N states for each action. Moreover, the new structure can easily be extended to deal with r ($2 < r < \infty$) actions. The $L_{2N,2}$ automaton is a typical learning machine that in attempts to choose the best next action based on knowing the past behavior of the system. The automaton "keeps track" of the number of successes and failures received as a result of each action. It then switches to the successive action after a certain number of failures are encountered, depending on its current state.

The Tsetlin automaton output function is as follows: if it is in a state $\Phi_i$ ($1 \le i \le N$), it chooses action $\alpha_1$, and if it is in a state $\Phi_i$ ($N+1 \le i \le 2N$) it chooses action $\alpha_2$. Since each action has N states associated with it, N is called the memory associate with each action, and the whole automaton has a total memory of 2N.

The illustrations of the state transitions are shown in Figure 3.55. The first figure shows the case of a favorable response, and the second one shows the behavior for an unfavorable response.

The deepest states in memory are referred to as the most internal states, or the end states and the boundary states are referred to as the most external states. If the environment replies with a reward or a favorable response, the automaton moves deeper into the memory of the corresponding action or towards the most internal state. In the case of a penalty or an unfavorable response from the environment, the automaton moves towards the outside boundary state of the memory of the corresponding action.

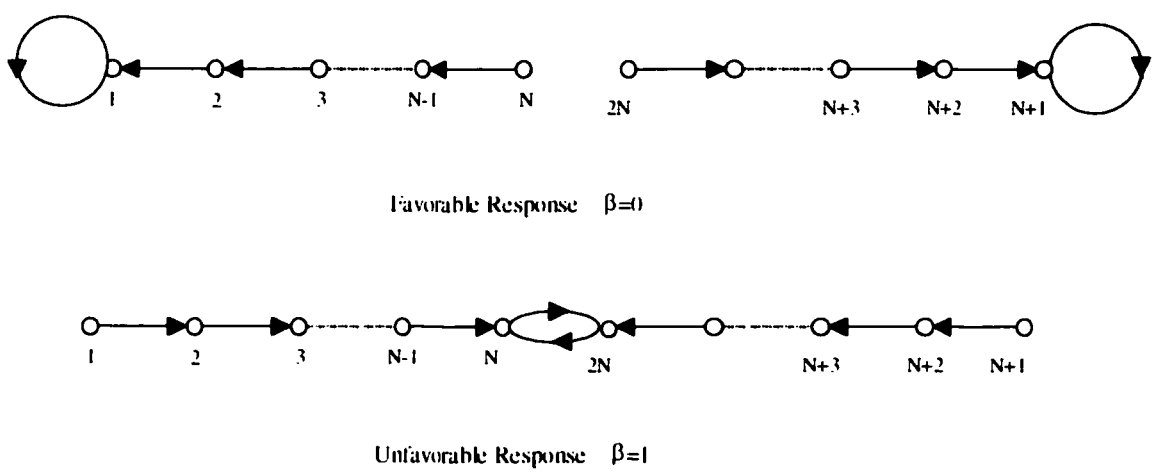



**Figure 3.5**: State transition graphs for the Tsetlin automaton $L_{2N,2}$. The Tsetlin automaton is *ε-optimal* whenever the minimum penalty probability is less than the OS.

### 3.4.2.3 Krinsky Automaton

This Krinsky automaton, denoted $K^{1}_{2N,2}$, is a deterministic automaton. It is very much like the Tsetlin automaton in that it is an automaton with 2N states and 2 actions.

The output function is identical to that of Tsetlins automaton, *i.e.* if the automaton is in any state $\Phi_i$ (i=1,2...N) it chooses action $\alpha_1$, and if it is in any state $\Phi_i$ (i=N+1,N+2...2N), it chooses action $\alpha_2$.

The state transition function of the Krinsky automaton is similar but not identical to that of the Tsetlin automaton. When the response of the environment is a penalty, the automaton shows the same behavior in that it moves towards the outside of their current action's domain. However, when the environment replies with a reward, it is drastically different. While the Tsetlin automata moves exactly one state closer or further from its internal states for each reward or penalty, the intent of the Krinsky's automaton is to give a *maximum* effect for each reward. Therefore, when the environment rewards an action, the automaton moves to the deepest state in the memory. N *consecutive* penalties are required to offset this action of the automaton. In the case of a favorable response, if the automaton is in any state $\Phi_i$ (i=1,2...N), it passes to state $\Phi_1$ and if it is in any state $\Phi_i$ (i=N+1,N+2...2N), it passes to the state $\Phi_{N+1}$, as shown in Figure 3.6.

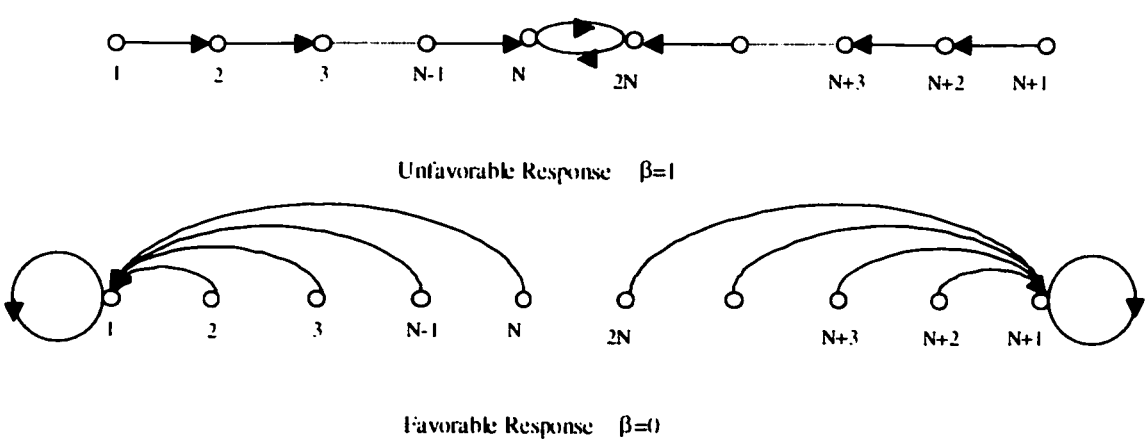



**Figure 3.6:** State transition graphs for the Krinsky automaton $K^1_{2N,2}$. The Krinsky automaton is $\epsilon$-*optimal* in all random environment.

Both the Tsetlin and Krinsky automata are classified as deterministic automata, because both have an output and state transition functions that are deterministic. For the purpose of presenting the complete domain of fixed structure automata, the stochastic Krylov learning automaton that was introduced in 1964 is presented in the next subsection.

#### 3.4.2.4 Krylov Automaton

Similar to the Tsetlin and Krinsky, the Krylov automaton ($K^2_{2N,2}$) is an automaton that has 2N states and 2 actions. It also has the same output transition function as the $L_{2N,2}$ automaton. Moreover, the Krylov's automaton has the identical state transition function as the $L_{2N,2}$ automaton when the response of the environment is favorable. However, the behavior changes when the automaton is penalized, in which case the behavior of the Krylov automaton is stochastic rather than deterministic. As shown in the Figure 3.7, if the automaton is penalized, it moves towards or away from it's internal states with a probability 0.5 each.

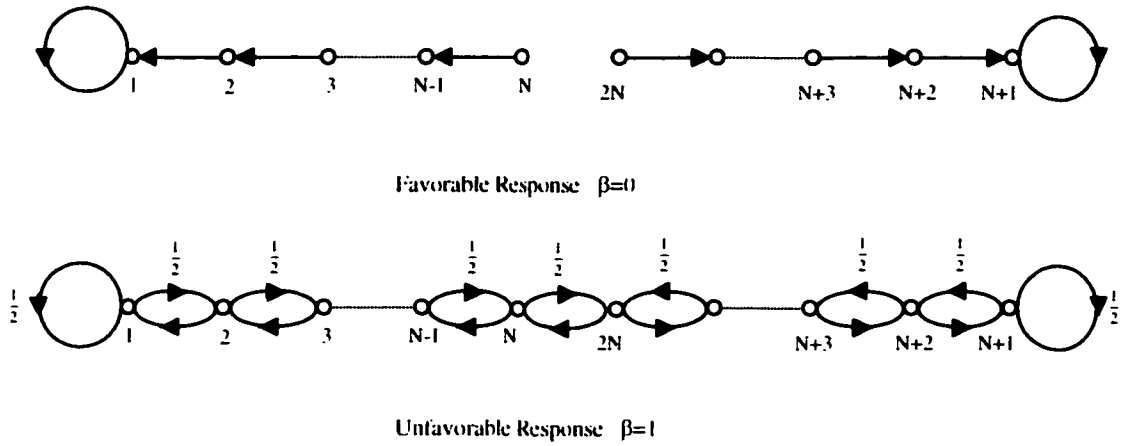



**Figure 3.7:** State transition graphs for the Krylov automaton $K^2_{2N,2}$

Krylov's automaton is a FSSA and is $\epsilon$-optimal in all random environments. All the three, the Tsetlin, Krinsky, and Krylov automata can be extended to make the automata perform in $r$ ($2 \leq r < \infty$) actions $\{\alpha_1, \alpha_2 \ldots \alpha_r\}$. The difference between the automata with many actions and the automata with only two actions is primarily in the states where the automaton switches from one action to the next.

### 3.4.3 Variable Structure Stochastic Automata

In the quest for more flexibility in automata, Varshavskii and Vorontsova [VAR63] in 1963 were the first to propose a class of automata that update of transition probabilities, and which are now known as the *Variable Structure Stochastic Automata* (VSSA). The main characteristic of the VSSA is that the state transition probabilities or the action selecting probabilities are constantly updated with time.

In VSSA, for mathematical simplicity, it is assumed that each state corresponds to a distinct action. This requires that the VSSA have the same number of states and actions. Therefore, the action mapping G becomes the identity mapping. Varshavskii and Vorontsova proved that every VSSA is completely defined by a set of action probability updating rules. The state transition mapping F, then, becomes equivalent to the probability-updating rule for the definition of a VSSA. The learning automata operates on a probability vector $\mathbf{P}(n)=[p_1(n),\ldots,p_r(n)]^T$, where $p_i(n)$ $(i=1,\ldots,r)$ is the probability that the automaton will select the action $\alpha_i$ at the time n, and thus $p_i(n)=\Pr[\alpha(n)=\alpha_i]$. The formal mathematical definition of the VSSA is given below.

**Definition 3.5:** A *variable structure stochastic automaton* is a 4-tuple <A,B,T,**P**>, where A is the set of actions, B is the set of inputs of the automaton (the set of outputs of the environment), and $T:[0,1]^r\times B\rightarrow[0,1]^r$ is an updating scheme such that

$$\mathbf{P}(n+1)=T(\mathbf{P}(n),\beta(n)), \tag{3.16}$$

where **P** is the action probability vector, $\mathbf{P}(n)=[p_1(n),p_2(n),\ldots,p_r(n)]^T$, with $p_i(n)=\Pr[\alpha(n)=\alpha_i]$, $i=1,\ldots,r$, and $\sum_{i=1}^{r} p_i(n)=1$ for all 'n'.

The VSSA are described in terms of the function used for updating of the probability vector. If **P**(n+1) is a linear function of **P**(n), the automaton is classified as *linear*, otherwise it is considered *nonlinear*. Two or more automata could be combined to form a *hybrid* automaton. VSSA usually follow some basic learning principles without being impacted by the form of updating scheme. If the automaton finds an action $\alpha_i$

being rewarded, it increases the probability for this action, and decreases the probability for all other actions. In the case of an action $\alpha_i$ being penalized, the automaton decreases the probability for this action and increases the probability for all other actions. There can be many combinations of updating schemes that can be enumerated based on the learning principle of VSSA. These schemes are of the following types:

- **RP (Reward-Penalty)** – the probabilities of the automaton get updated when the automaton is rewarded and penalized.
- **RI (Reward-Inaction)** – the probabilities are updated only when the automaton is rewarded. No change happens when the automaton is penalized.
- **IP (Inaction-Penalty)** – the probabilities are updated when the automaton is penalized and are left unchanged when the automaton is rewarded.

The automaton is classified is a *continuous automaton* only if the mapping T is continuous. The original VSSA that were presented in the literature were continuous algorithms. Thathachar and Oommen introduced discretized versions of learning VSSA in 1979 [THA79]. These versions have been later modified to produce varieties of absorbing, ergodic and estimator families of learning automata.

A simple updating scheme for a continuous VSSA operating in a stationary environment with $\beta=\{0,1\}$ can be represented as follows:

If action $\alpha_i$ is chosen *i.e.*, $\alpha(n)=\alpha_i$, the updated probabilities are:

$$\begin{aligned} p_j(n+1) &= p_j(n) - g_i(\mathrm{P}(n)), \quad when\ \beta(n)=0 \\ p_j(n+1) &= p_j(n) + h_j(\mathrm{P}(n)), \quad when\ \beta(n)=1 \end{aligned} \quad \text{, for all j, } j \neq i. \tag{3.17}$$

Because **P**(n) is a probability vector, it has to satisfy $\sum_{j=1}^{r} p_j(n) = 1$. This means that

$$
\begin{aligned}
p_i(n+1) &= p_i(n) + \sum_{\substack{j=1 \\ j\neq i}}^{r} g_j(\mathbf{P}(n)), \qquad when\ \beta(n) = 0 \\
p_i(n+1) &= p_i(n) - \sum_{\substack{j=1 \\ j\neq i}}^{r} h_j(\mathbf{P}(n)), \qquad when\ \beta(n) = 1
\end{aligned}
\tag{3.18}
$$

Functions $h_j$ and $g_j$ have the following properties:

- For mathematical simplicity, $h_j$ and $g_j$ are assumed to be continuous functions [NAR89])
- $h_j$ and $g_j$ are nonnegative functions,

  $0 < g_j(\mathbf{P}) < p_j$
- $0 < \sum_{\substack{j=1 \\ j\neq i}}^{r} [p_j + h_j(\mathbf{P})] < 1$,

for all i=1,2,...,r and all **P** whose elements are in the open interval (0,1).

Usually, VSSA are implemented using a random number generator. The automaton will decide on the next action to be chosen based on the action probability distribution **P**(n).

## 3.5 Conclusions

In this chapter, we introduced the concepts of learning automata and discussed different kinds of known learning automata. We also showed how they could be

compared and how the learning evolved. The primary concepts of learning automata were introduced in the first section. Then, a detailed description of various Fixed Structure Stochastic Automata was presented. This group of learning automata was exemplified by presenting the Tsetlin's, Krinsky's and Krylov's automata. These machines will serve as the foundation for the algorithm that will be presented in the next chapter, and used in the remaining chapters. The concepts of the family of Variable Structure Stochastic Automata were also briefly described.

# Chapter 4: THE OBJECT PARTITIONING SOLUTION

## 4.1 Introduction

Chapters 2 of this thesis discussed the existing methods for solving the image retrieval problem. All these methods can be classified as being either one of image recognition or image classification. The main contribution of this thesis is a solution to the image retrieval problem by viewing the problem as one that uses the new approach of a learning automaton solution for the object partitioning problem. This chapter presents the object partitioning problem and discusses the existing solution including the latest approach that is the basis for the current work.

## 4.2 The Object Partitioning Problem

The Object Partitioning Problem (OPP) deals with the partitioning of a set of objects into a set of classes. Mathematically, let A = { $A_1$,....., $A_W$ } be a set of W objects to be partitioned into R classes { $P_1$,...., $P_R$ }. The aim is to have objects partitioned in such a way that objects that have been accessed more frequently together are placed in the same class.

Objects are generally accessed in pairs represented as $<A_i,A_j>$. Normally, the objects denoted in the pair are considered, with a given probability, to belong to the same group. This pair of objects is referred to as a query. An object partitioning algorithm will execute a series of queries so as to increase the likelihood that a pair of objects accessed will reside in the same class the next time the same query is performed. The image retrieval problem can be solved using a solution to the OPP by considering each object to be a representation of an image in the database. Therefore, whenever an abstract object is a member of a certain class, it implies that the corresponding physical image is a member of that class.

This thesis utilizes a special case of the OPP where all the classes have an equal number of objects. This problem is referred to as the Equi-Partitioning Problem (EPP). All objects are initially placed randomly into the available classes. It is assumed that the number of classes R is equivalent to the number of grouping of objects. In specific cases where it is not possible to equally partition objects into grouping, some grouping will have objects that do not correspond to any image.

A few solutions to the OPP have been reported. Some of these solutions will be presented in the following sub-sections.

### 4.2.1 Basic Adaptive Method

The Basic Adaptive Method (BAM) [YU81] is a simple solution for the EPP since no statistical calculation or object movement is performed during query processing. If there are W objects, the BAM method associates a set of abstract objects $\underline{O} = \{O_1 \ldots\ldots, O_W\}$ with the physical set of objects $\underline{A} = \{A_1 \ldots\ldots, A_W\}$. The abstract set $\underline{O}$, unlike the physical set $\underline{A}$, contains only the name and the location of the object on a partitioned line. This makes the process of manipulation less expensive.

After performing the queries on the physical objects, migration operations are done only on the abstract set $\underline{O}$. The order of the actual set of objects $\underline{A}$ is periodically checked with the abstract set. If they are different, the physical set is repartitioned when the system is idle to reflect the changes made earlier to the abstract set.

The BAM method associates a real number $X_i$ with each member object $O_i$. This number represents the location of the object on the partitioned line. When processing a query of two objects $<A_i, A_j>$, the corresponding real numbers $<X_i, X_j>$ are moved towards their centroid by some amount $\Delta_1$, where the centroid of $X_i$ and $X_j$ is calculated by $(X_i+X_j)/2$. This will bring the queried objects to cluster together. However, if this is done too many times, then the clusters of different classes will all coalesce. Consequently, another step is carried out to offset the accumulating impact of the first step. This is done by randomly selecting another pair of objects $<A_p, A_q>$, $(1 \leq p, q \leq W)$, for processing, and the corresponding real numbers $<X_p, X_q>$ are moved away from *their*

centroid by some amount $\Delta_2$. Therefore, the BAM is a two-step method. The first step is to move the two objects that are accessed together, and the second is to randomly pick two other objects and move them apart. The outcome of this process is that objects that are accessed frequently together will cluster together. Figure 4.1 shows the first step of BAM when applied on four objects and two classes. It can be shown that the objects that are accessed together are now placed closer together. However, the impact of doing queries several times leads to both classes clustering together to form a one big class. Executing the second step, by randomly selecting two other objects and moving them apart, will tend to make the two classes separate. The final result for executing both steps together is shown in Figure 4.2.

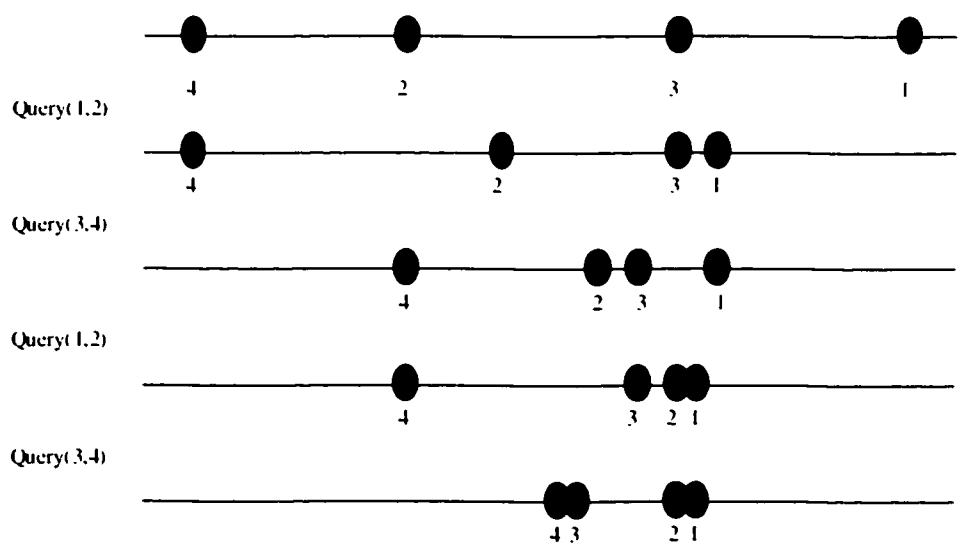



**Figure 4. 1**: First step of BAM method.

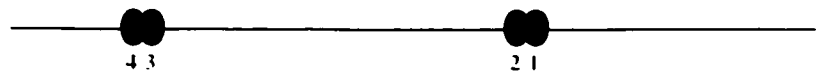



**Figure 4. 2**: The two classes are clustered away from each other as a result of applying both steps of the BAM method.

### 4.2.2 Tsetlin, Krinsky, and Krylov Automata

The Tsetlin, Krinsky, and Krylov automata presented in the previous chapter can be adapted to solve the EPP problem. These automata are all deterministic and have K actions $\{\alpha_1 \ldots \alpha_K\}$ and KN states $\{\Phi_1 \ldots \Phi_{KN}\}$. The actions of the automata can be thought of as a partition in the EPP. If there are W objects and R classes of equal size, then each class will have W/R objects. Since all three automata are similar in structure, we use the Tsetlin automaton to illustrate the solution to the EPP.

Assume that we are given a set of four objects $\{A_1, A_2, A_3, A_4\}$ to be partitioned into two classes so that every class has two objects. These four objects can be partitioned in three different ways. Mapping this in terms of the Tsetlin automaton, each partition is represented by an action as follows:

1. $\{(A_1, A_2), (A_3, A_4)\} \rightarrow$(action $\alpha_1$)
2. $\{(A_1, A_3), (A_2, A_4)\} \rightarrow$(action $\alpha_2$)
3. $\{(A_1, A_4), (A_2, A_3)\} \rightarrow$(action $\alpha_3$)

If the automaton has two states per action, the automaton can be referred to as $L_{6,3}$ to reflect that the automaton has six states and three actions.

At start up, the automation is allocated to the boundary state of a random action out of the three available actions. Suppose that the following four queries are to be processed by the automaton:

1. $<A_1,A_2>$
2. $<A_1,A_3>$
3. $<A_1,A_2>$
4. $<A_1,A_4>$

Figure 4.3 shows the condition when the automaton is started, by assigning the automaton to a boundary state $\Phi_4$ of a random action $\alpha_2$.

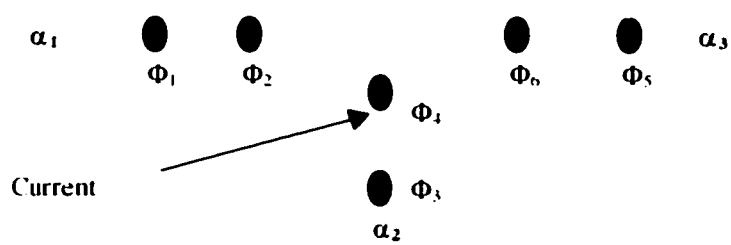



**Figure 4. 3**: At system start up of Tsetlin automaton, it resides in a random state.

**Query 1**: The query processing starts by the first query of $<A_1,A_2>$. This query actually supports action $\alpha_1$ and not the current one $\alpha_2$, which is $\{(A_1, A_3), (A_2, A_4)\}$. Therefore, the automaton is penalized by moving it to the next action $\alpha_3$ state $\Phi_6$. Figure 4.4 shows the transition on achieving this query processing.

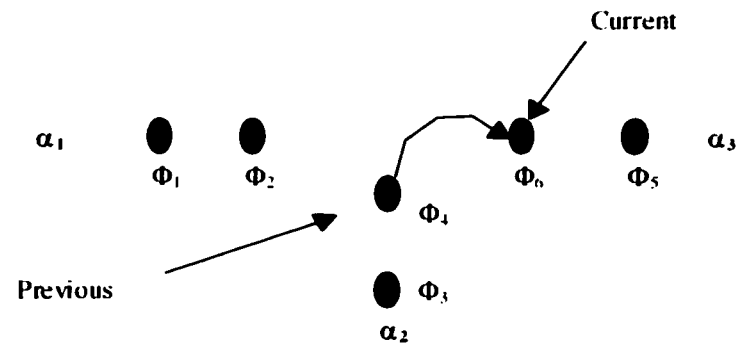



**Figure 4. 4**: Automaton is penalized as a result of query <$A_1$,$A_2$>.

**Query 2**: The next query is <$A_1$,$A_3$>. This query does not support the partition represented by the current action $\alpha_3$, which is {($A_1$, $A_4$), ($A_2$, $A_3$)}. The automaton is penalized again by moving it to the next action $\alpha_1$ state whose boundary state is $\Phi_2$. Figure 4.5 shows this effect of processing this query.

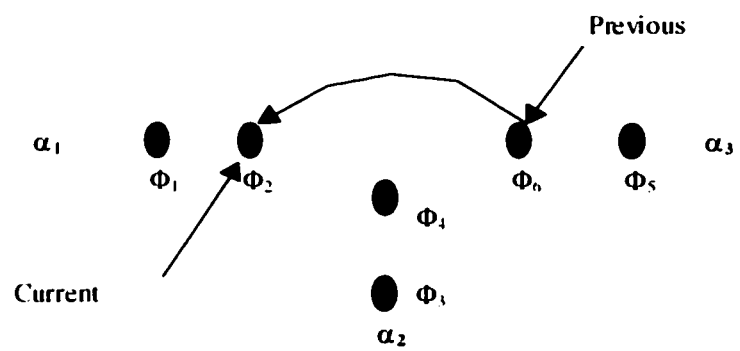



**Figure 4. 5**: Automaton is penalized as a result of query <$A_1$,$A_3$>.

**Query 3**: The next query is <$A_1$,$A_2$>. This query supports the partition represented by the current action $\alpha_1$. Here, the automaton will be rewarded by moving it to the most internal state $\Phi_2$. Figure 4.6 shows how this query is

processed.

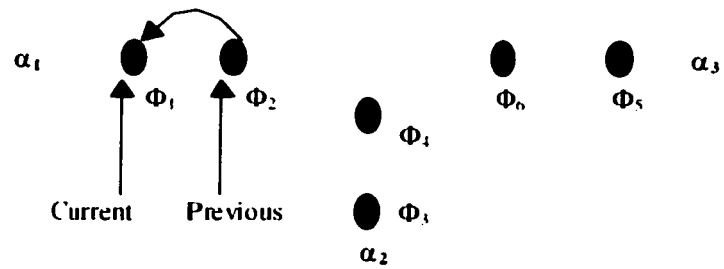



**Figure 4. 6**: Automaton is rewarded as a result of query <$A_1$,$A_2$>.

**Query 4**: The last query is <$A_3$,$A_4$>. This query again supports the partition represented by the current action $\alpha_1$. Normally, the automaton will be rewarded by moving to the next internal state. Since the automaton, however, is already in the most internal state, the automaton is considered to have converged and the partition {($A_1$, $A_4$), ($A_2$, $A_3$)} is considered to be the solution to the EPP for the given set of objects. Figure 4.7 shows this query processing.

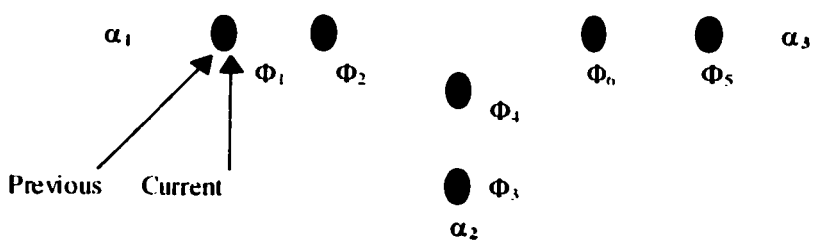



**Figure 4. 7**: The automaton converges after processing the fourth query of <$A_3$,$A_4$>.

The Tsetlin, Krinsky, and Krylov automata have been proven [OOM88] to have good learning properties and to be superior to the BAM. However, they are also proven [OOM88] to be practically feasible only when there is a small number of objects and partitions. Given W objects and R classes, the number of actions needed is $W!/[(M!)]^{R} R!$, where M=W/R. Therefore, for W =12 and R =3, the automaton must have 5775 actions. As the number of classes and objects grow, the EPP solution becomes infeasible due to the large amount of computer memory and resources needed to maintain the representing structure.

### 4.2.3 Object Migration Automation

#### 4.2.3.1 OMA advantages

Oommen and Ma in 1988 [OOM88] first proposed the Object Migration Automation (OMA). While the Tsetlin, Krinsky, and Krylov automata have an increasing number of actions with more objects and partitions, the number of OMA actions is always equal to number of partitions. Furthermore, the OMA automaton allows the member objects to move around within itself while the previous automata move entirely from one state to another. When the OMA is applied to the EPP, a solution is not defined by the current state of the OMA but by the entire structure of automation. The mapping function defining the transition of the automaton from one state to another essentially defines the motion of the two or three objects within the structure of the automation.

Since the OMA is capable of moving objects within its structure, it is applied in a manner different from the previous algorithms. The states of the automaton are partitioned into groups where each group represents an action. Instead of defining rules for moving the automaton from one state to another, rules are defined to outline how *objects* are moved from one partition to the another.

#### 4.2.3.2 OMA description

If we have W objects to be partitioned into R classes, the aim is to place exactly W/R objects in each class. The OMA associates a set of abstract objects, $\{O_1, \ldots O_W\}$ with the physical objects. The actual processing happens to the abstract object but, for simplicity, we will be referring to the abstract objects by objects.

The OMA is denoted as $Q_{KN,K}$ and defined as:

$Q_{KN,K} = (\{\alpha_1, \ldots, \alpha_R\}, \{\Phi_1, \ldots, \Phi_{RN}\}, \{0,1\}, Q(.,.), G(.)$, where

1. Actions: $\{\alpha_1, \ldots, \alpha_R\}$ is the set of actions or classes.
2. States: $\{\Phi_1, \ldots, \Phi_{RN}\}$ is the set of states. Each action has N states. For each action, the states ordered as: $\Phi_{(i-1)N+1}, \ldots \Phi_{iN}$. It is also assumed that $\Phi_{(i-1)N+1}$ is the boundary state and $\Phi_{iN}$ is the most internal state.
3. $\{0,1\}$ is the set of inputs from the environment, which represents the reward and penalty responses respectively.
4. Q(.,.) is the transition matrix, which defines the actual transition of objects as a result of the query processing.

5. G(.) is the output map. If an object is in state $\Phi_i$, then it chooses the action number $\lceil \Phi_i/N \rceil$.

Let $\{A_1, A_2\}$ be two queried objects where $A_1 \neq A_2$. Let the corresponding states of $A_1$ and $A_2$ be denoted as $\Phi_{i1}$ and $\Phi_{i2}$. Let $\alpha_{j1}$ and $\alpha_{j2}$ represent the respective actions of $\Phi_{i1}$ and $\Phi_{i2}$ as defined by function G above. Let $A_3$ be a third object chosen from the same class as either $A_1$ or $A_2$. The state of $A_3$ is denoted by $\Phi_{i3}$. For all $1 \leq k \leq K$, the function Q behaves as follows.

(a) if $\alpha_{j1} = \alpha_{j2}$, then for $i = i1, j = j1$ and $i = i2, j = j2$

$$Q(\Phi_i, 0) = \Phi_{i-1} \quad \text{if } (j-1)N + 1 < i \leq jN$$

$$= \Phi_i \quad \text{if } i = (j-1)N+1$$

This case occurs when the two queried objects are found in the same class. The reward for this occurring is that they are both moved one step towards the most internal state of their class. Figure 4.8 shows this process.

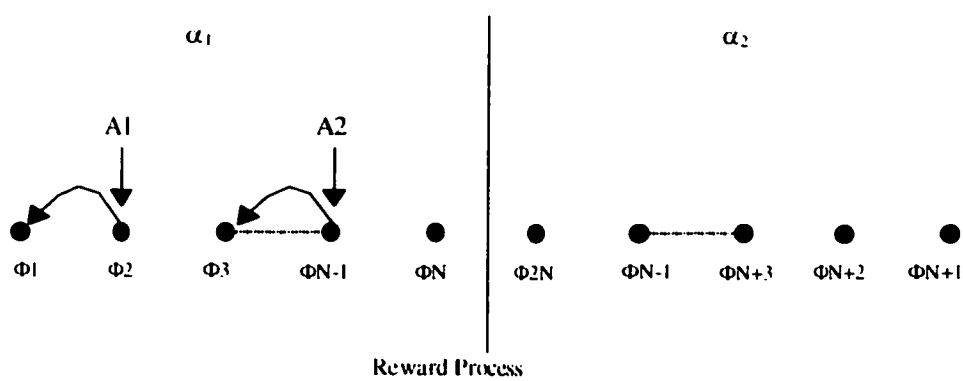



**Figure 4. 8**: Reward process when the two accessed objects are found in the same class.

(b) if $\alpha_{j1} \neq \alpha_{j2}$ , and $\Phi_{i1} \neq kN$ and $\Phi_{i2} \neq kN$, then for i = i1, j = j1 and i = i2.

$Q(\Phi_i,0) \quad = \Phi_{i+1}$

This case occurs when the two queried objects are found in two different classes and both of the objects are not in the boundary state of their class. The penalty for this occurring is they are both moved one step toward the boundary state of their respective class. Figure 4.9 shows this transition.

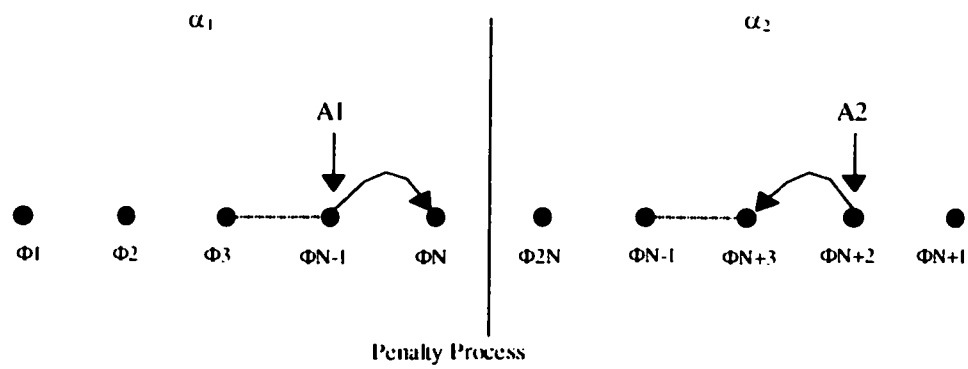



**Figure 4. 9**: Penalty process when the two objects are found in the two different classes and both objects are not in the boundary state of their class.

(c) if $\alpha_{j1} \neq \alpha_{j2}$, and $\Phi_{i1} \neq kN$ and $\Phi_{i2} = kN$, then

$Q(\Phi_{i1}, 1) \quad = \Phi_{i1+1}$

$Q(\Phi_{i2}, 1) \quad = \Phi_{(j1)N}$

Select $A_3$ from $\alpha_{j1}$ such that $\Phi_{i3}$ is closest to (j1)N

$Q(\Phi_{i3}, 1) \quad = \Phi_{(j2)N}$

This is the case when the two queried objects are found in two different classes and only one of the objects is in the boundary state of its class. The penalty is invoked by

moving the object that is not in the boundary state of its class one step toward the boundary state of its class. An object that is the closest to the boundary state of this class is selected and swapped with the other queried object from the other class. Both are placed in the boundary states of their new classes. Figure 4.10 shows how this takes place.

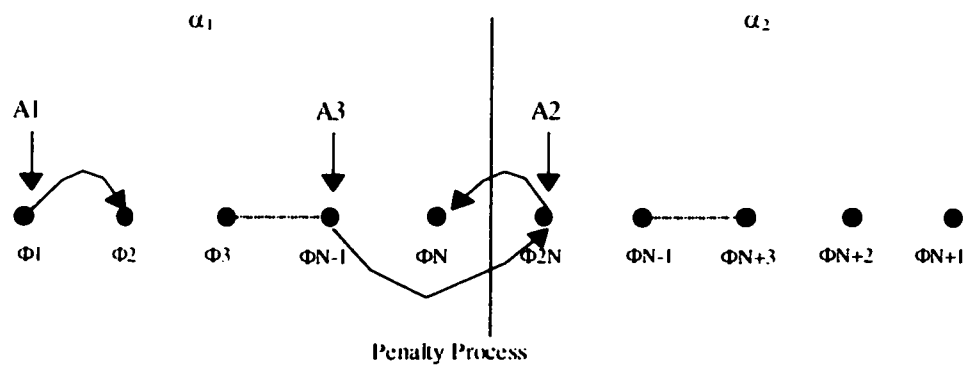



**Figure 4. 10**: Penalty process when the two objects are found in the two different classes and one object only is in the boundary state of its class.

(d) if $\alpha_{j1} \neq \alpha_{j2}$, and $\Phi_{i1} = kN$ and $\Phi_{i2} = kN$, then

$Q(\Phi_{i1}, 1) \quad = \Phi_{i1}$

$Q(\Phi_{i2}, 1) \quad = \Phi_{(j1)N}$

Select $A_3$ from $\alpha_{j1}$ such that $\Phi_{i3}$ is closest to (j1)N

$Q(\Phi_{i3}, 1) \quad = \Phi_{(j2)N}$

This is the case when the two queried objects are found in two different classes and both objects are in the boundary state of their class. The objects are panelized by moving an arbitrary one of the two objects to other object's class. To make both classes

of equal sizes, an object that is in the closest state to the boundary state of this class is selected and moved to the other class. Both newly moved objects are placed in the boundary states of their new classes. Figure 4.11 shows how this is achieved.

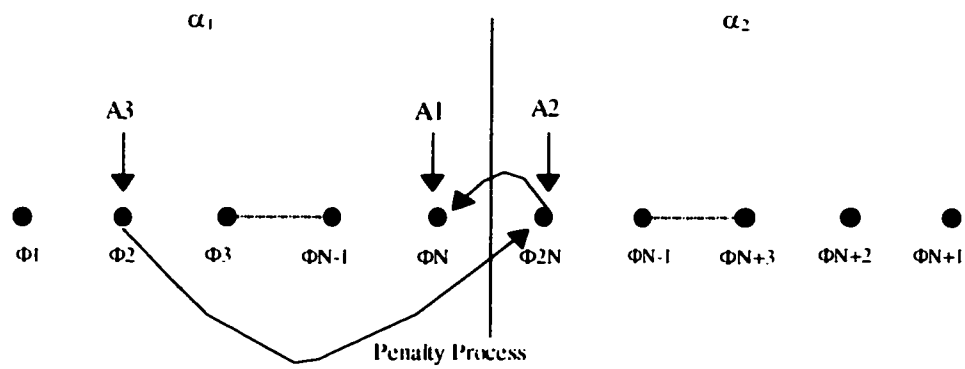



**Figure 4. 11**: Penalty process when the two objects are found in the two different classes and both objects are in the boundary state of their classes.

#### 4.2.3.3 OMA example

To show how the OMA works, we go through the same example that was visited in the BAM method and the Tsetlin automaton, *i. e.* four objects to be partitioned into two classes. W for the OMA is 4, R is only 2, and the number of objects per class is two, *i. e.* 4/2. Therefore, we need a two action OMA each with two states. Here are the queries to be executed:

1. $<A_1,A_2>$
2. $<A_1,A_3>$
3. $<A_1,A_2>$
4. $<A_1,A_4>$
5. $<A_1,A_2>$

At start up, the objects are randomly assigned to the two actions and their states, say action $\alpha_1$ has the two objects $A_1$ and $A_4$ and $\alpha_2$ has the two objects $A_1$ and $A_2$. Figure 4.12 shows how the objects are assumed to be.


**Figure 4. 12:** Objects are randomly assigned to the available actions.

**Query 1**: The first query is <$A_1$,$A_2$>. Since $A_1$ was initially allocated to action $\alpha_1$ and $A_2$ was initially allocated to action $\alpha_2$, both objects will be penalized for not being in the same action. $A_1$ is moved toward the boundary state of its action. Since $A_2$ is already in boundary state of its action, it is swapped to $\alpha_1$ with unaccessed object which is $A_4$. Figure 4.13 shows how this is done.

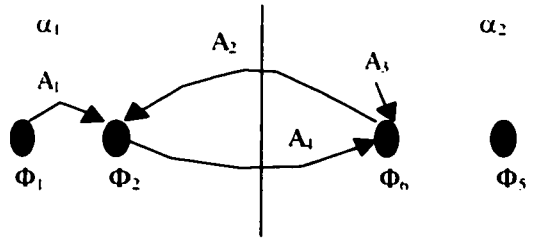



**Figure 4. 13**: Objects $A_1$ and $A_2$ are panelized for not being in the same action when accessed together.

**Query 2**: The second query is <$A_1$,$A_3$>. Since both objects are not in the same class, both are penalized. However, both objects are already in the boundary state of their actions. In this case, only one of them will be moved to the other action. Let's pick $A_3$

and move it to $\alpha_1$, where $A_1$ is. To do this, we swap $A_3$ with an object that is not accessed from $\alpha_1$. Here, we only have 2 objects per class and therefore the only choice is $A_2$. This swapping is shown in Figure 4.14.

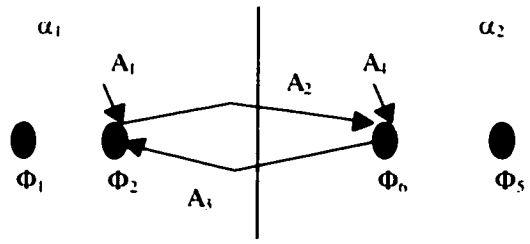



**Figure 4. 14**: The two objects being penalized are in the boundary state. One of them is swapped with another object to the other action.

**Query 3**: The two objects <$A_1$,$A_2$> are accessed together. Both objects are in different actions and already in their boundary states. The same process is taken as the previous query. $A_2$ is picked up to be moved to the other action and swapped with $A_2$ as shown in Figure 4.15.

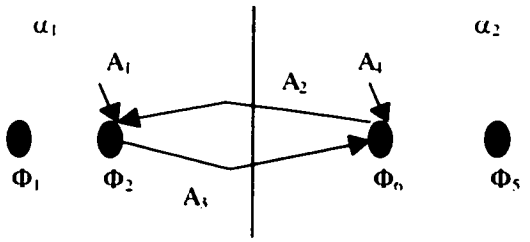



**Figure 4. 15**: The two objects being penalized are in the boundary state. One of them is swapped with another object to the other action.

**Query 4**: The two objects <$A_3$,$A_4$> are accessed together. Now, both of them are in the same action and therefore will be rewarded by moving them toward the most internal state of their action $\alpha_2$ as shown in Figure 4.16.

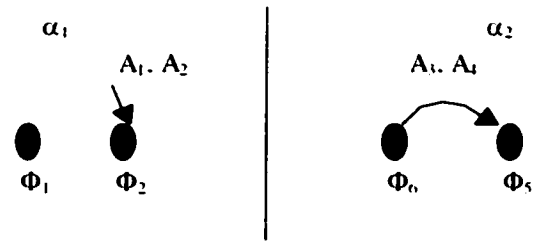



**Figure 4. 16**: Both objects are found in the same action and therefore rewarded by moving them toward their action's most internal state.

**Query 5**: The two objects $<A_1,A_2>$ are accessed together. Both of them are in them same action, again, and therefor will be rewarded by moving them toward the most internal state of their action $\alpha_1$ as shown in Figure 4.17.

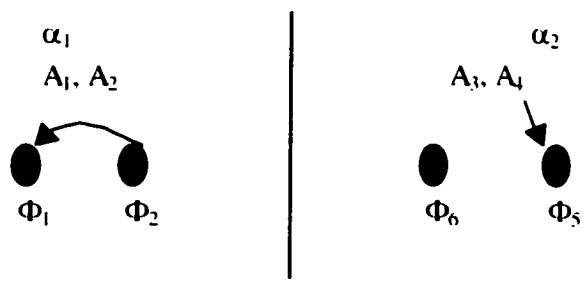



**Figure 4. 17**: Both objects are found in the same action and therefore rewarded by moving them toward their action's most internal state.

#### 4.2.3.4 Experimental results

The authors of [OOM88] have simulated the BAM, Testlin, Krinsky, and OMA to measure the number of iterations needed to achieve grouping and the number of iterations needed to converge into a total grouping. The simulations were done with varying number of objects and classes. Table 4.1 presents the number of iterations performed based on varying probabilities that the two queried objects are located in the same class.

| #Objects | #Classes | Prob. | BAM | Tselin | Krinsky | OMA |
|---|---|---|---|---|---|---|
| 4 | 2 | 0.6 | (80, 135) | (15, 30) | (23, 25) | (48, 93) |
| | | 0.7 | (75, 120) | (6, 20) | (20, 21) | (14, 40) |
| | | 0.8 | (60, 85) | (6, 17) | (7, 8) | (3, 34) |
| | | 0.9 | (50, 80) | (5, 14) | (6, 7) | (1, 26) |
| 6 | 2 | 0.6 | (350, 375) | (75, 100) | (21, 22) | (106, 202) |
| | | 0.7 | (280, 295) | (20, 35) | (20, 21) | (32, 95) |
| | | 0.8 | (225, 240) | (10, 25) | (16, 17) | (12, 64) |
| | | 0.9 | (200, 240) | (10, 20) | (11, 12) | (5, 46) |

**Table 4. 1**: Comparision of the BAM, Tsetlin, Krinsky, and OMA solutions of the EPP.

Notation: Prob. is the probability of two objects from same group being selected. Solutions are compared using the numbers (x, y) where x is the number of iteration performed to achieve grouping and y is the number of iteration needed to reach convergence.

Table 4.2 shows the result of comparing the BAM and the OMA solution of the EPP. The OMA was configured with 10 states per action. The probability of selecting two objects from the same group is 0.9.

| #Objects | #Objects/Class | #Classes | BAM | OMA |
|---|---|---|---|---|
| 4 | 2 | 2 | (21.40) | (18. 45) |
| 6 | 2 | 3 | (255.269) | (33. 73) |
| | 3 | 2 | (243.251) | (115. 160) |
| 9 | 3 | 3 | (745.762) | (209. 294) |
| 12 | 2 | 6 | (1520.1527) | (70. 163) |
| | 3 | 4 | (1885.1894) | (243. 350) |
| | 4 | 3 | (2914. 2925) | (569. 577) |
| | 6 | 2 | (3593. 3609) | (1993. 2092) |
| 15 | 3 | 5 | (2501. 2508) | (317. 454) |
| | 5 | 3 | (6442. 6445) | (2000. 2192) |
| 18 | 2 | 9 | (3348. 3352) | (31. 169) |
| | 3 | 6 | (2910. 2917) | (66. 201) |
| | 6 | 3 | (12754. 12756) | (81. 211) |

**Table 4. 2**: Comparison of the BAM, and OMA for the EPP.

## 4.3 Conclusion

In this chapter, we proposed a kernel solution for the image retrieval problem based on a solution for the equal partitioning problem. The equal partitioning problem is a special case of the object portioning problem which deals with equally partitioning a set of objects among a set of groups. The technique should increase the likelihood that similar objects would be placed in the same group. Several solutions for the problem were discussed starting with the Basic Adaptive Method [YU81]. Learning automata

solutions were also discussed using Tsetlin's, Krinsky's and Krylov's automata. Finally, the OMA solution was presented, and its advantages were clearly emphasized. The OMA will be used in the next chapter as the kernel for the image retrieval algorithm proposed in this thesis.

# Chapter 5: THE MULTILEVEL AUTOMATON-BASED IMAGE RETRIEVAL AND SEARCHING (MAIRS) SYSTEM

## 5.1 Introduction

In chapter 2 of this thesis report, we discussed several existing methods for solving the image search and retrieval problem. The methods presented had various approaches to solving the problem but all shared one peculiarity and disadvantage, namely that of having the user provide search criteria for the intended image. The user is expected to describe the intended image in words, provide a sketch, or look through existing images to come up with an example image. Furthermore, all existing methods lack the capability to classify a group of images in one group possessing a certain property, for example "sport champions".

This thesis modifies the OMA to come up with a solution for the image retrieval problem by means of a system called the Multilevel Automaton-based Image Retrieval and Searching (MAIRS) system. The MAIRS system gives the user the choice of classifying images based on his/her subjective feeling of what a certain image is similar

to. The user is not expected to provide any explicit search criteria. A mental view of the intended image in the user's mind is enough to start searching

## 5.2 OMA approach to the image search and retrieval

The philosophy of the OMA is applied to the image search and retrieval problem by assuming that once the images are properly sorted and indexed, there is an underlying unknown grouping. For example, a given set of images based on transportation may be divided into four image groups. These groups may represent cars, airplanes, boats and trains. At system startup these images are scattered amongst the four available groups. As the system is used, and the user selects images by informing the system that he/she considers various images to be similar to the intended image. The MAIRS system will utilize this similarity information between these images and intelligently partition them into groups. Hence, the images will be moved around and placed with other images that they are similar to. Observe that all of this is done in the background without the user being aware of the underlying data transition.

## 5.3 Previous OMA implementation

Oommen and Fothergill [OOM93] earlier implemented an elementary OMA solution to the image search and retrieval problem in their Image Database Learning Automation (IDLA) system. The IDLA is a prototype image search and retrieval application, which we have significantly enhanced.

### 5.3.1 User Interface

At system startup, the user selects the size of the catalogue that he/she prefers. The choices are either 4 groups of 4 images each, or 16 groups of 16 images each, as shown in Figure 5.1. Therefore, the catalogue is composed of either 16 images or 256 images. When the user selects a catalogue size, a catalogue is created with the specified size. Images can be added to the catalogue according to user's choice, and up to the maximum number of images which is either 16 or 256. Figure 5.2 shows the choices that are available to the user once a catalogue is initiated. Figure 5.2 shows how the search can be simulated on numbers artificially used to represent the images.

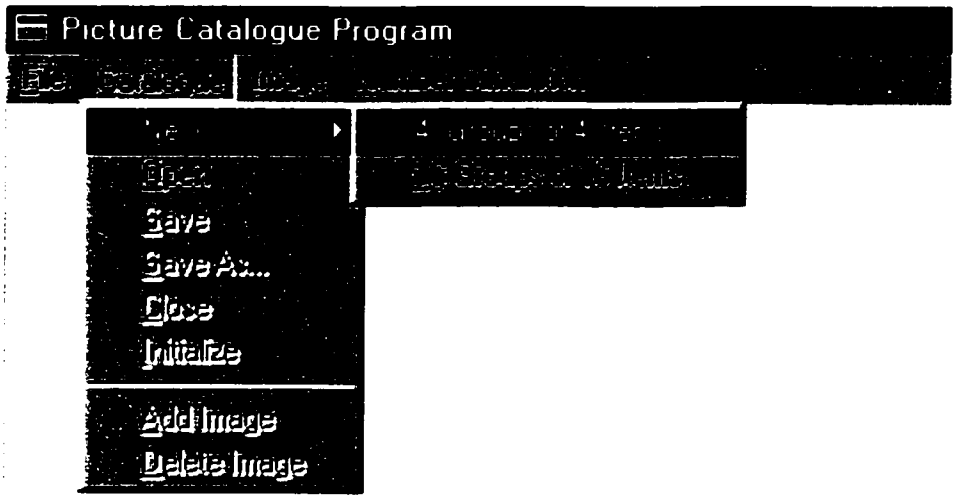



**Figure 5. 1:** User's choice of size of catalogue

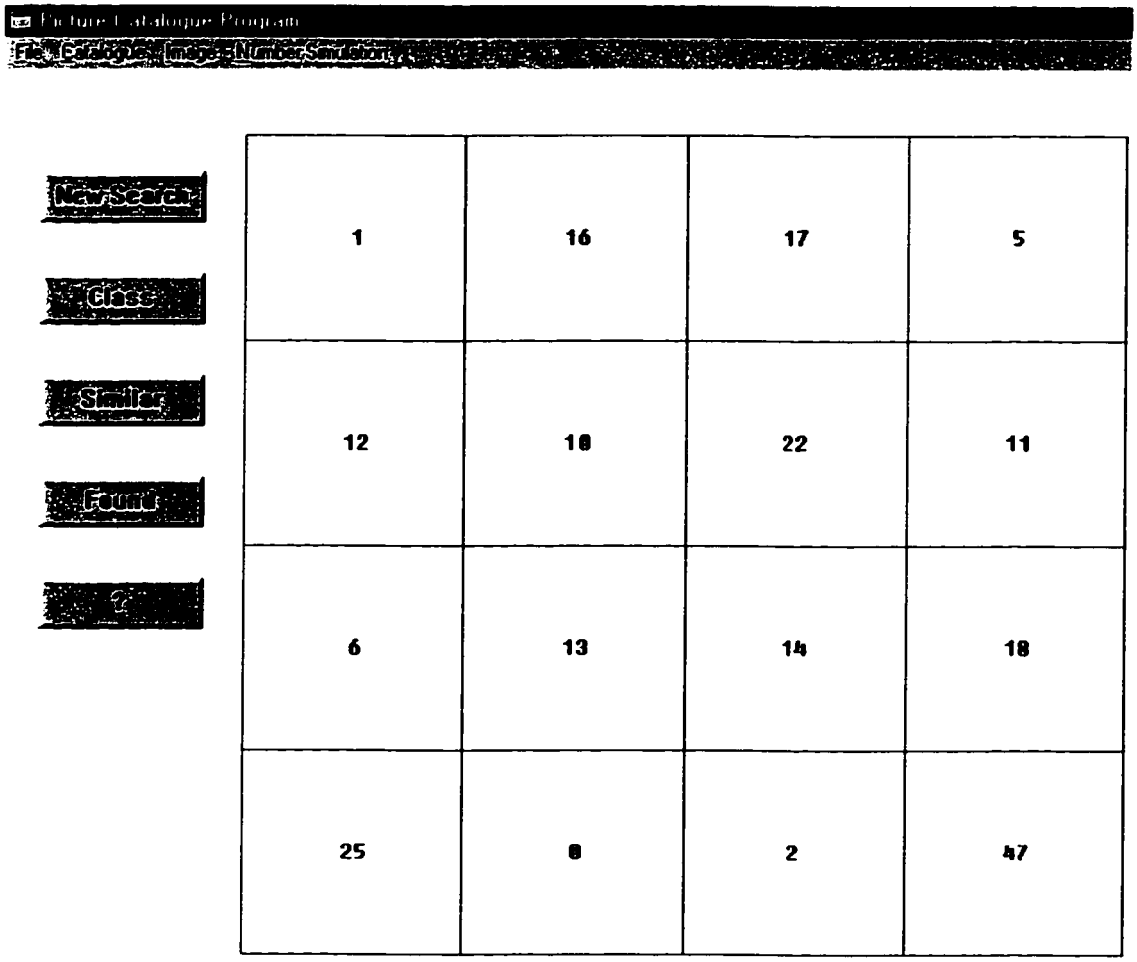



**Figure 5. 2**: Choices available to user in the IDLA system.

The user of the system is presented with a set of images. This set may be a representative overview of all the classes or may be all the images of a particular class. The current image set that is displayed depends on the user's search action. Upon initiation of the image search, the user is presented with an overview set of images. This overview consists of one image from each class, where each image presented is the current best representative image of its class. If the user selects a particular class for examination, then the overview set will be replaced by the set of images in the chosen class.

Figure 5.3 shows the partitioning and viewing of images. The figure shows the view of four classes, where each class contains 4 images.

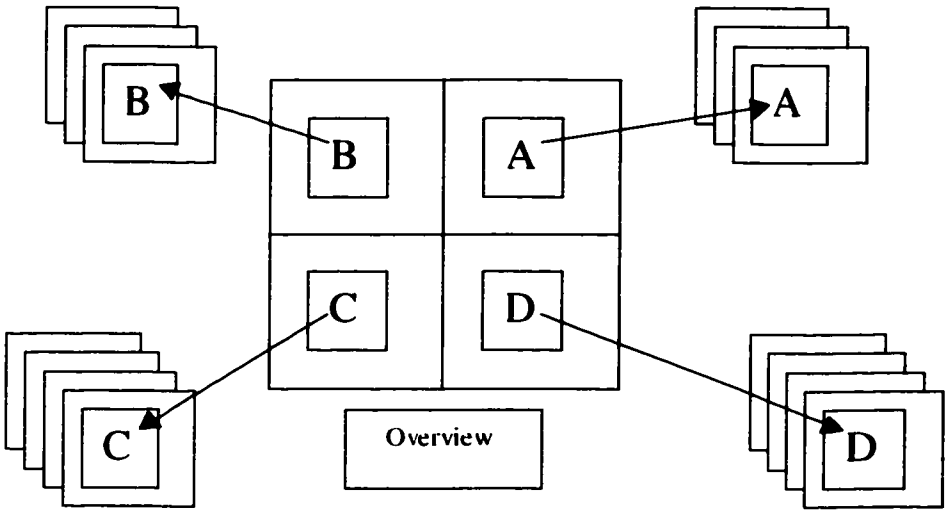



**Figure 5. 3**: Overview of the IDLA structure of images for a 4-class catalogue.

A representative image is not a permanent member of its class. Hence, if image 'A' is swapped as a result of the user interacting with the system, image 'A' will lose its membership to its class and will be classified under another class.

The user who has a mental view of the image he/she is searching for in the image database initiates a search. The first group of images presented is the overview set. The user may see the desired image and will therefore tag it as being 'found'. At this point, the search is terminated and the user can initiate a new search. If the desired image is not found from the representative images, the user tags some of the presented images as 'similar' to enable the system to proceed with the search.

The user may also examine all images of a given class by selecting the desired class. If the user finds the desired image then the search is complete. Otherwise the user selects which images, if any, are similar to the desired image. The process of viewing the overview level of images, and selecting classes for further examination is progresses until the desired image is found.

Once the desired image is found, the system partitions the database based on the OMA philosophy. Further searches of the catalogue will reflect the updated (new) partition.

### 5.3.2 General system design

The IDLA system is broken down into components, and each one performs certain functions as shown in Figure 5.4.

The 'Menu Input' and 'Display Output' modules are collections of user input and output routines that are implemented in Microsoft Windows GUI calls. The 'Menu Input' module also handles switching between the various kinds of simulations that the system performs.

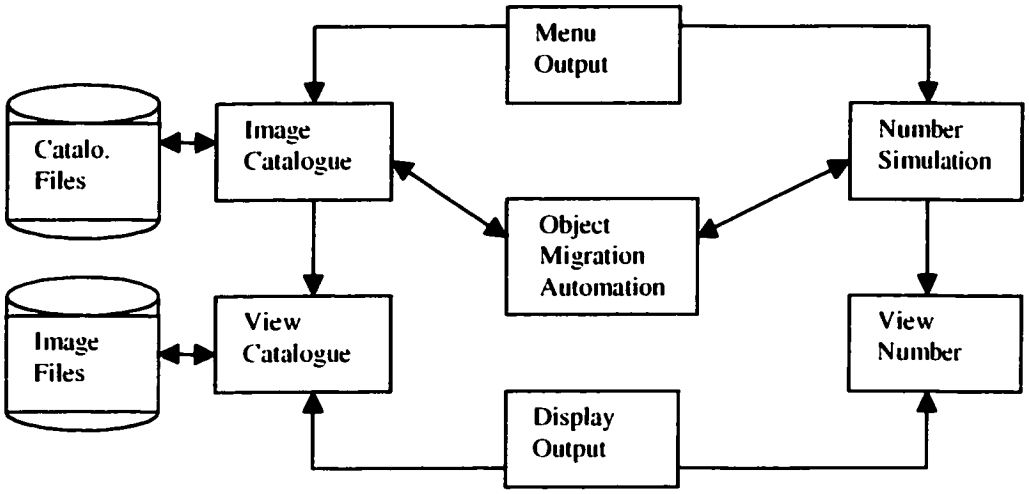



**Figure 5. 4**: IDLA System Design.

The main kernel of the system is the 'Object Migration Automation'. It manipulates the automaton data structure, and processes all queries made by either the 'Number Simulation' module or the 'Image Catalogue' Module.

The 'Number Simulation' module is a tool that allows the user to view the state of the automation. After each query, it obtains the status of the automation from the 'Object Migration Automation' module, and through the 'View Number' module it displays this status. This module is also capable of running simulations that repeatedly query the automaton, and end when the 'Object Migration Automaton' module sets the convergence flag to 'true'.

The 'Image Catalogue' module also uses automaton module to process the user's queries received from the 'Menu Input' module. It interacts with the Catalogue Files for configuration purposes. Any time a query triggers a class update from the 'Object Migration Automation' module, the new partitioning information is passed onto the

'View Catalogue' module. This module uses Image files on disk to attain the images, and displays them.

### 5.3.3 IDLA Object Migration system engine description

The IDLA applies the OMA method of dividing the objects equally among the available actions. For the purpose of the image search and retrieval problem the objects being considered are abstract representation of images. Therefore, if an image X is represented by abstract object $A_X$, and that abstract object is in action $\alpha_X$, then image X is displayed when all members of $\alpha_X$ are displayed. For the purposes of image search and retrieval, the actions are referred to as classes. Images are initially scattered among all the available classes. As the user performs queries on the database of images, similar images get clustered in one class. Since the task of deciding which images are similar is subjective, it is left to the user to determine similar images.

There are different states within each class to quantify how certain we are that a given image belongs to that class. At system startup, all images are placed in the boundary state of their initial class to indicate that the system is uncertain of where an image should be placed. As the system is used, certain images will be rewarded for their being present in a given class and will move towards the most internal state of that class, to indicate that the system is more certain now that the image belongs to this class. Correspondingly, other images will be penalized for being in the wrong class by moving

them towards their boundary state or to another class to indicate that the system is uncertain about which class should have this image.

### 5.3.4 Evaluation

The authors of [OOM93] have evaluated the system based on correctness, memory usage, and real-time speed.

Correctness was proved even when the most similar images are not found in the same class. The system gives the user the choice of determining which images are similar, which makes the system practical. This also relieves the user from specifying search criteria for the desired image.

The system was also proved to be memory efficient because a small version of the image in the catalogue is displayed to the user. Observe that the user has the option of obtaining the full size image at any time. This makes the program very efficient in terms of memory use.

The IDLA system was only designed to be a prototype to demonstrate the OMA solution to the image search and retrieval problem. It restricts catalogue sizes to be either 16 or 256 images, which is very small size compared to the sizes of the existing image databases. The predetermined size also confines the user's needs to expand or shrink the size of the catalogue. The IDLA has only two levels of images: the main level displaying the representative of the various classes, and the member images of classes in the second

level. In practical application involving image search and retrieval, there is a need for more than two levels to reflect sub-classing of any given class. For example, the first level may have images of transportation methods including an image of a car, the second level could have cars including the image of a "racy car", and the third level could have the class of "racy car images" including a Ferrari!

The image formats used in [OOM93] were limited to specific types that are not widely used. The prototype did not support the well-known formats, such as the bitmap, GIF, or JPEG. This was a serious limitation of the prototype.

## 5.4 The Multilevel Automaton-based Image Retrieval and Searching (MAIRS) System

This thesis presents a more practical solution to the image retrieval problem by modifying the IDLA approach. The proposed solution is called the Multilevel Automaton-based Image Retrieval and Searching (MAIRS) system. The IDLA offered only two levels of images, the first for the representative images and the second is for the actual images of a given class. As opposed to this, the catalogues in the MAIRS system can be expanded to many levels according to database size. This offers multi sub-classing of classes to include smaller more specific groups, such as those involving a specific type of racy cars. The number of images that can be placed into the catalogue is always $16^n$, where n is the number of levels needed to fit all images.

The system uses an add on class library for Visual C++ called ImageObject [RIC98], which allows the users to easily load, display, and manipulate images.

The ImageObject also supports six file formats: BMP, GIF, JPG, PCX, TGA, and TIF. This allows the user of the MAIRS system to include several image file formats in the database. The user only specifies where the images exists, and the system starts finding the image files names, allocating storage records for their information, and expanding the database of classes and levels so as to include the newly read images.

### 5.4.1 User Interface

The user interface of the MAIRS system is based on the Microsoft Windows look and feel, which makes it easy to use. Considering that writing the primitive modules could have been prohibitively time consuming; utilizing the existing libraries saved a lot of time and effort.

At system startup, the user provides the directory name of where the images reside and the system reads and populates the image database with the images themselves. Figure 5.5 shows a screenshot of the system view after the user provides the directory of the image files. We can see from the screen that all images are shown in a smaller compressed version suitable for class viewing. The user may choose to view the images at the full image size, if desired.

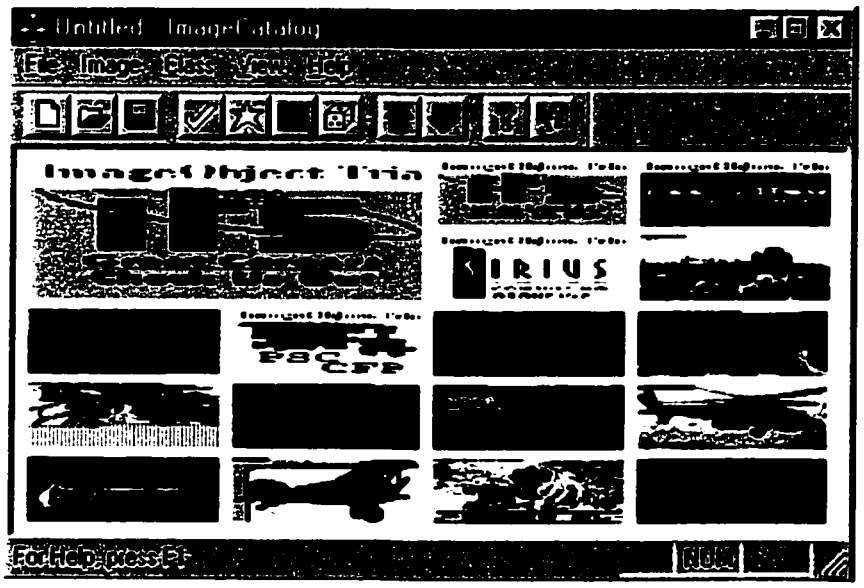



**Figure 5. 5:** The MAIRS system window.

As show in Figure 5.5, images of a class are shown with the main image enlarged. The main image is referred to as the *Parent* image, which is the image that represents its class in the upper level class of images. In this context, the term "Upper Class" will be used to refer to the class that has the *Parent* image as a member image. The concepts of *Parent* image and image class will be described in detail in the section highlighting the system description.

The MAIRS system has a feature that makes it flexible when the system window is resized so as to fit the available screen space. If the application window is resized, the presented images will be automatically rescaled so as to fit in the newly resized window. Accordingly, the user can see bigger or smaller images merely by resizing the system's window.

The first screen of images represents the representative images of other classes, and yet they all belong to the first class of the database. Generally speaking, one screen of shown images constitutes the representative overviews of more classes, and all these images are also members of the currently-shown class of images. The current image class being displayed depends on the user's search action. Upon initiation of the image search, the user is presented with the first class of the database, which is also an overview of images from the next level of classes originating the first class. Each image in this overview is a *Parent* image of a *Child* class of images.

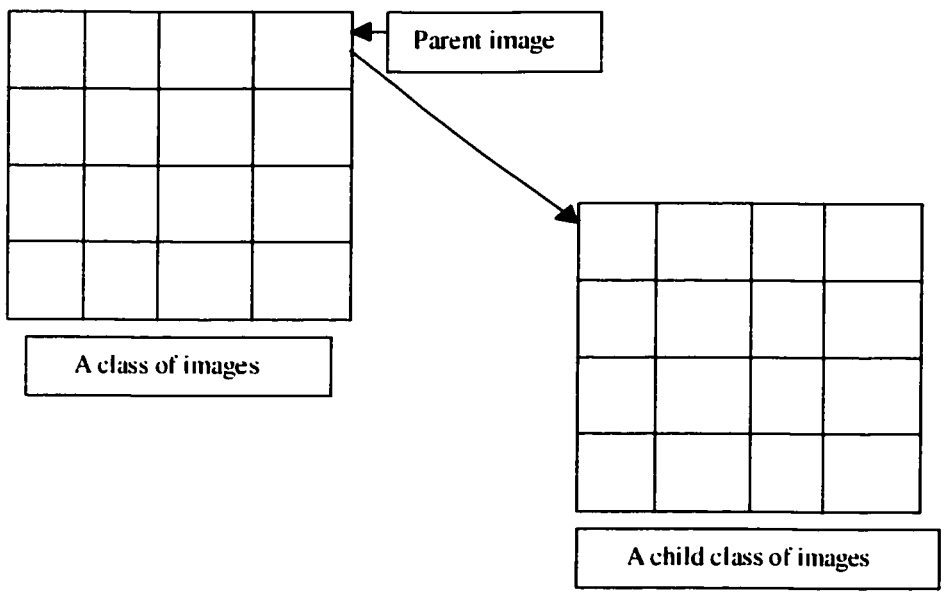



**Figure 5. 6:** The *Parent* image and its *Child* class of images.

If the user selects a particular class for examination, then the currently viewed class images will be replaced with the image class *Child* of the selected image.

Each image shown in a given view is the best representative of its class. If the user selects a particular class for examination, then the overview set will be replaced by the set of images in the chosen class.

It should be emphasized that although a representative image is physically placed in the first position of its class, it is not a permanent member of its class. Therefore, any representative image can be swapped to another class or "downgraded" to be a normal member of its class.

The user starts by having a mental view of the image that he/she is searching for in the image database. The first group of images presented is the first class, and an overview of the next level of classes. If the user sees the desired image, he/she will tag it as being 'found'. At this point, the search is terminates and the user is free to start a new search. However, if the desired image is not found from the representative images, the user may tag some of the presented images as 'similar'. The images that are tagged as 'similar' will have a yellow sticker posted on them indicating 'similar' as shown in Figure 5.7.

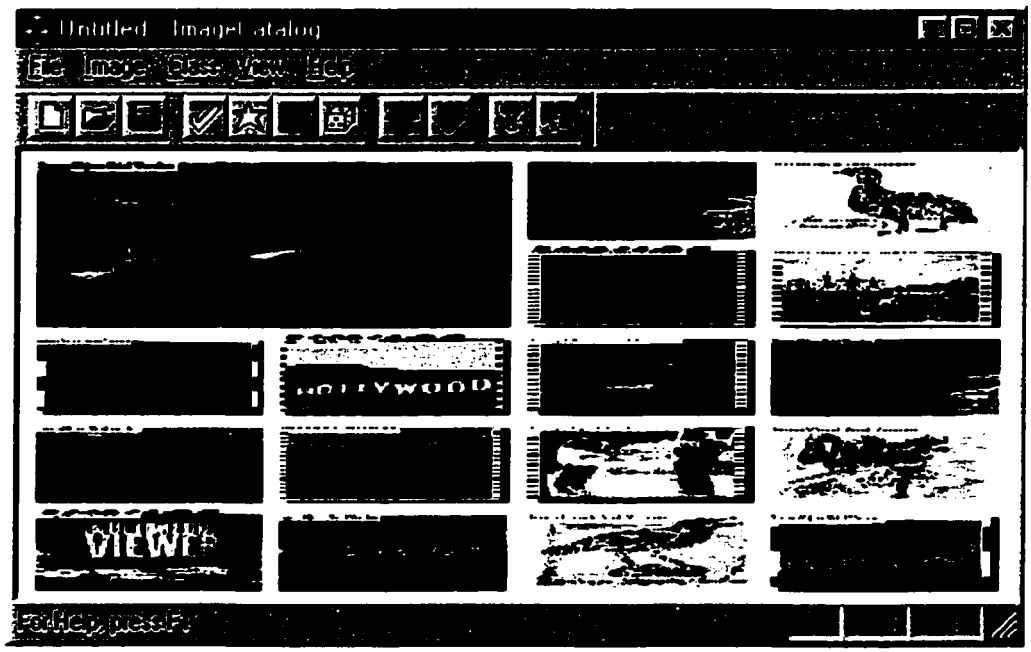



**Figure 5. 7**: Images being marked as 'Similar' by user.

The user may also check the children images of any given image by selecting the desired image. Again, the user continues the process of either finding the desired image or selecting the images, if any, that are "similar" to the desired image until the desired image is found. The user may go down to as many levels as needed in the image tree to search for the desired image and mark the similar images. Figure 5.8 shows an example of an image tree structure. For simplicity, the example shows only 3 images per class, while the actual size of a class in the MAIRS is 16 images.

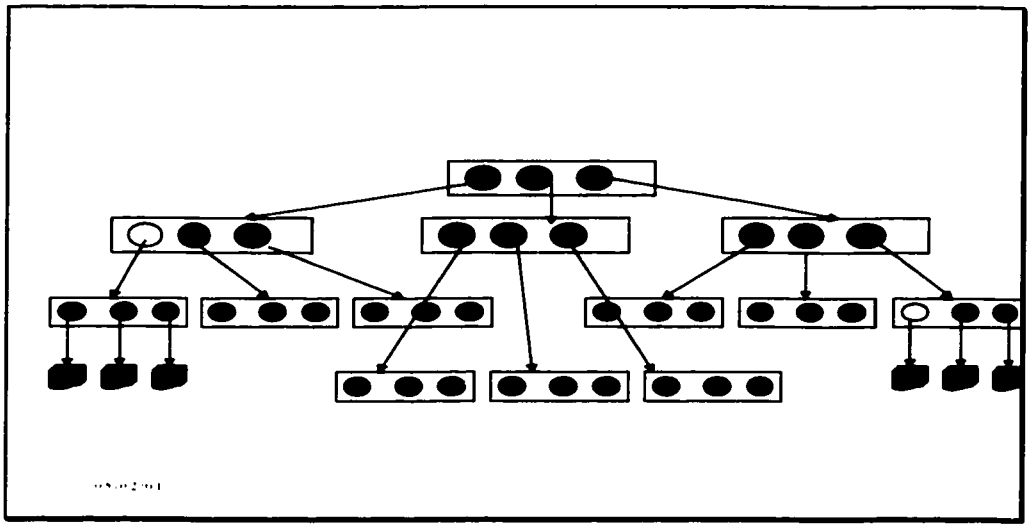


**Figure 5. 8**: Image tree structure example.

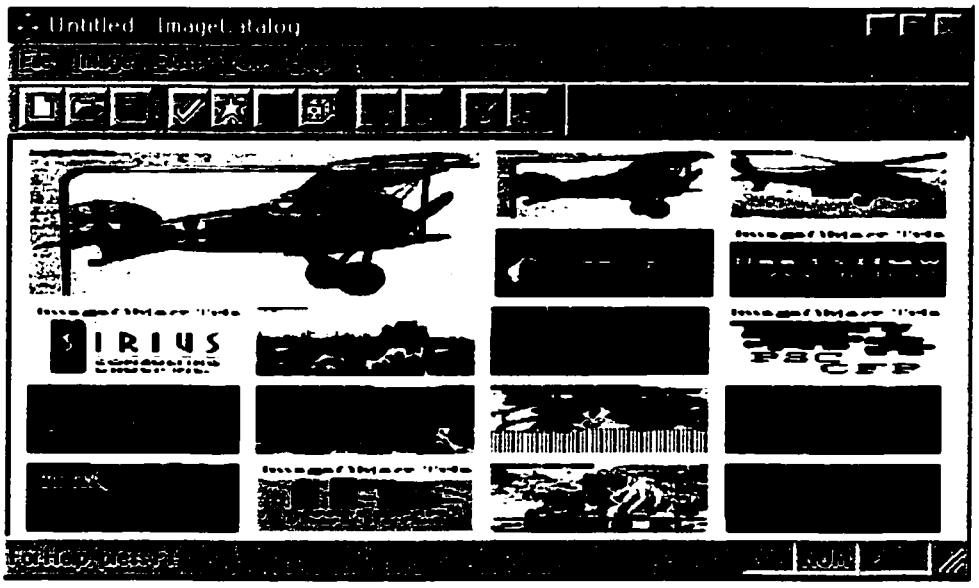



**Figure 5. 9**: The result of a search; similar images are now together.

Once the desired image is found, the user marks it as 'found'. The system will partition the database and further searches of the database will reflect the new partition. Figure 5.9 shows a search for transportation methods that has just been done. It shows

that airplanes and a car are placed in the same class as a result of the user's responses of marking the 'similar' images to the intended image, and marking the intended image as 'found'.

### 5.4.2 System description

#### 5.4.2.1 MFC and ImageObject

The system was developed using the C++ language and the Microsoft Foundation Classes (MFC). The MFC, written in C++, is an application framework for programming in Microsoft Windows. MFC is used to provide the code necessary for managing the GUI part of windows, menus, and dialog boxes. In addition, it also provides the code needed to manipulate known types of data structures such as lists, strings, stacks, *etc*. Programmers can access this code, by adding the application-specific code segments into this framework.

The system also uses an additional library for Visual C++ called ImageObject [RIC98]. This library enables users to load, display, and manipulate images of six different file formats: BMP, GIF, JPG, PCX, TGA, and TIF. Only the display part is currently used, but the others can be used to give the user more choices on manipulating the images.

### 5.4.2.2 System Modules

This subsection will describe the details of how the system works. Figure 5.10 shows the major modules of the MAIRS system

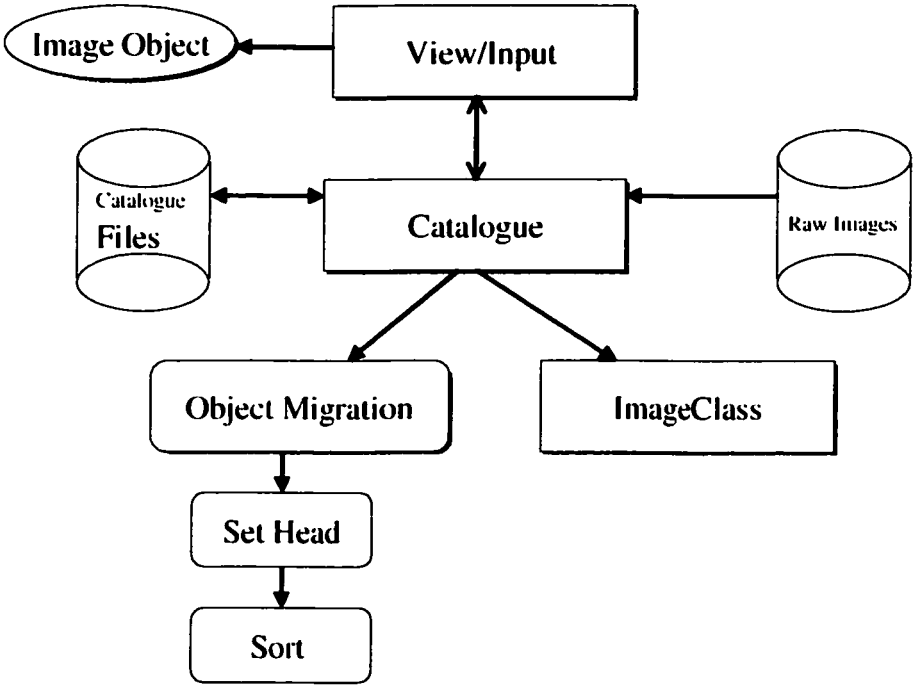



**Figure 5. 10:** The modules of the MAIRS System

The smallest entity of the system is an image. The term image is used to refer to the physical image characteristics, such as the format, width, length, display location and size, orientation, *etc*. This part of the image properties is used by the add-on library of ImageObject to display the image according to the specified format and location. The term image is also used in the system to refer to the image as an abstract entity. This entity is a struct that is defined for any task, and has the following fields:

```
ImageClass    *child;
ImageStruct   *parent;
Int           state;
```

*Int* *headState;*
*String* *sFileName;*

The term image will be mentioned in this chapter to refer to the abstract entity, since the physical image is not of a major concern to the system. Only the abstract part of the image is used to manipulate where a certain image should reside in the *ImageClasses*.

As outlined in the user interface section, each image has a *Child* of image class where an image class contains 16 images. This makes each image know its children images which is needed when going up and down in the image tree. Also, each image is required to know its *Parent* for the same reason in order to keep track of image *Parent*/*Child* relationship.

***State***

The *state* field is defined to keep track of the status of the image within its class. When images are being read into the database and allocated to the available classes, they are all assigned the boundary state of their arbitrary allocated class. This indicates that the system is not sure if the image belongs to this class or not, and therefore the image is placed on the boundary state of this class. As the system gets used, certain images will be rewarded for being in "right" class by decreasing the status index by unity. When the image state reaches the most internal state, it will not be decreased any more. This means that the system is as certain as it can be that the image belongs to this class. Likewise, certain images will be penalized for being in the "wrong" class by incrementing the state number by unity. The images that are in the boundary state are all swap candidates, which

can be swapped to other classes since their memberships to the class are weak. Image swapping mechanism, will be described in details in the next section.

***headState***

Each image is a member of an *ImageClass* of 16 images. However there is one extra image which is shown in each class view. This image is referred to as the *Parent* image. The *Parent* is a not technically a member of its *Child* class. It is actually a member of the class at the next upper level. It is shown in the *Child* class in enhanced size so as to distinguish it from the other images. A *Parent* image, however, becomes a *Parent* because it achieved the closest state to the most internal state among its class members. When a member image becomes the *Parent*, its *State* number gets assigned to its *headState* field, and its state will be inherited from the previous *Parent* image of the class. Therefore, the *headState* field is used to keep track of the status of the *Parent* image within its *Child* class. After each query, the system checks the states of the images of each class along with the *headState* of the *Parent* image. If it finds any image with a *State* that is closer to the most internal state than the *headState* of the *Parent* image, it swaps the image with the *Parent* image. The *Parent* image becomes a normal member of the class, and the swapped image becomes the new *Parent* image and a member of the upper class. Figure 5.11 shows how the *headState* field is used for a given image.

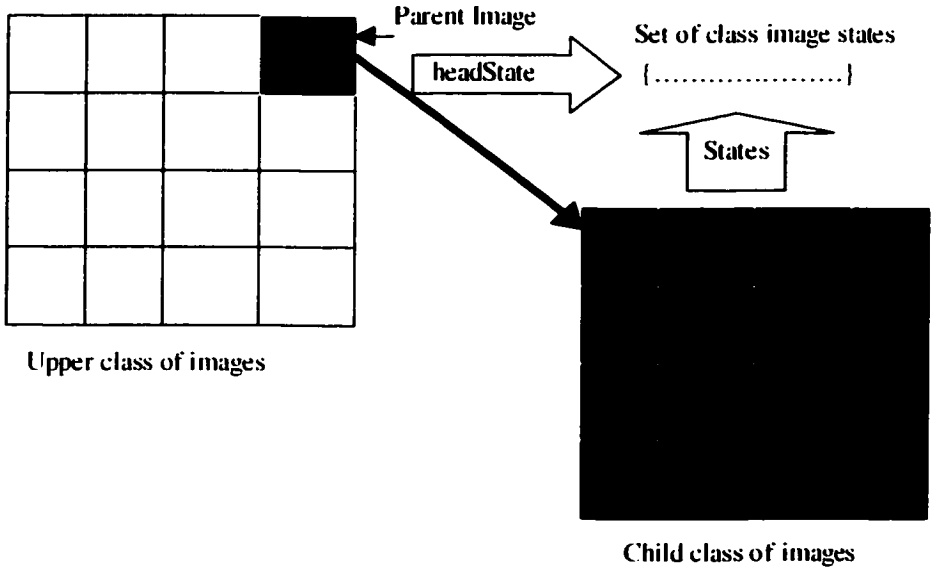



**Figure 5.11**: *headState* field of the *Parent* image is added to the states of its children.

**View/Input module**

The View/Input module defines resources and implements functions needed to take the user input and display the system window with the images according to the last partition of images. Different kinds of icons are defined to change as the user's request changes. For example, if a user selects the option to go down the image tree, the mouse pointer icon will change until all images of the selected class are displayed. The ImageObject library is used to display images with the specified size and location. The catalogue gives the View/Input module the structure of the images and their names to be displayed. The View/Input module passes the image name, desired size, and location to the ImageObject library to enable it to display the image.

***ImageClass* module**

The *ImageClass* module is a class that defines the list of images that are members of the class. It also defines methods needed to access and move images. Except for the first *ImageClass*, each ImageCalss is *Child* of an image. Also, except for the leaves of tree, every image is *Parent* of an ImageCalss. The *ImageClass* defines a method to add images to it as their names are read from the directory supplied by the user, and records are created for them. Additionally, at that time each *ImageClass* knows its *Parent* image. The *ImageClass* methods are used to find an image with the closest *state* to the boundary state for swapping to another class, when needed. The swapping operation is also done by an *ImageClass* method.

**Catalogue module**

The main part of the system is the Catalogue module. It contains the heart of the system, which is the actual process after a search concludes. It defines data structures necessary to keep track of the necessary details as follows:

**Image list**

Image list contains all the images that currently exist in the catalogue.

***ImageClass*es list**

This list keeps track of all the *ImageClass*es that are allocated in the system.

**Similarity list**

During a search, all the similar images to the intended image are kept in the Similarity list for processing after the search concludes.

### 5.4.3 System in action

At system start up, images are allocated as they are read into the *ImageClasses* that are built for the images which are to reside on the catalogue. All images are assigned the most external state and head state by default. With the exception of the images of first class of images, every image is assigned a *Parent* image. The *Parent* image is the best representative image of its class in the upper level. Since there is no level higher than the first class, images of the first class have no *Parent*. Images that are not leaves have a *Child* class of images and each one of these images has this image as a *Parent*. The user may browse through the images by going from one class to another up and down the image tree. If the user sees a similar image, he/she will highlight the image as being "similar". The image will instantly be shown highlighted with a similar tag on top of it. During a search, the user tags a few, say t, images to be similar to the intended image. All the similar images are kept in a list as $S = \{S_1, S_2, \ldots\ldots S_t\}$. When the user finds the intended image *X*, the search concludes and the system invokes the kernel of the MAIRS.

The system starts by considering each image in the list of the similar images S with the found image X as a query of the form $< S_i, X>$. Likewise, the system also considers every two images in the similarity list to be a query $<S_i, S_j>$. If both images in the query are in the same class, they are both rewarded, and if both are in different classes, they are both penalized. The MAIRS reward and penalty process is similar to the one of IDLA but it also has the capabilities to handle special situations that are specific to multiple levels of images. Special situations may come up as a result of one image being

the parent of the other image in the query. Now, we will describe the actual penalty and reward processes.

*Reward process (Both images are in the same class)*

Given the two images in the query $<A_i, A_j>$, if both are found in the same class, both images will be rewarded. The reward is accomplished by moving them towards the most internal state of their class. However if the rewarded image is already in the most internal state of its class, no action will be taken.

*Penalty process (Both images are not in the same class and not in boundary state)*

If the two images in a query $< A_i, A_j>$ are not in the same class and at the same time are not in the boundary state of their classes, both will be penalized. The penalty for this situation is accomplished by moving both the images one step toward their boundary states.

*Penalty process (Both images are not in the same class and one of them is in boundary state)*

If the two images in the query $< A_i, Aj>$ are not in the same class and one of them is not in the boundary state of their class, both will be panalized. The image that is not in the boundary state will be moved one step toward the boundary state. The image that is in the boundary state is swapped to the other class with an image that has the closest state to the boundary state. Both of the swapped images will be assigned the boundary state of their new classes. By

"swapping", we mean here that there is a complete swap of the image and its children from one location to another as shown in Figure 5.12.

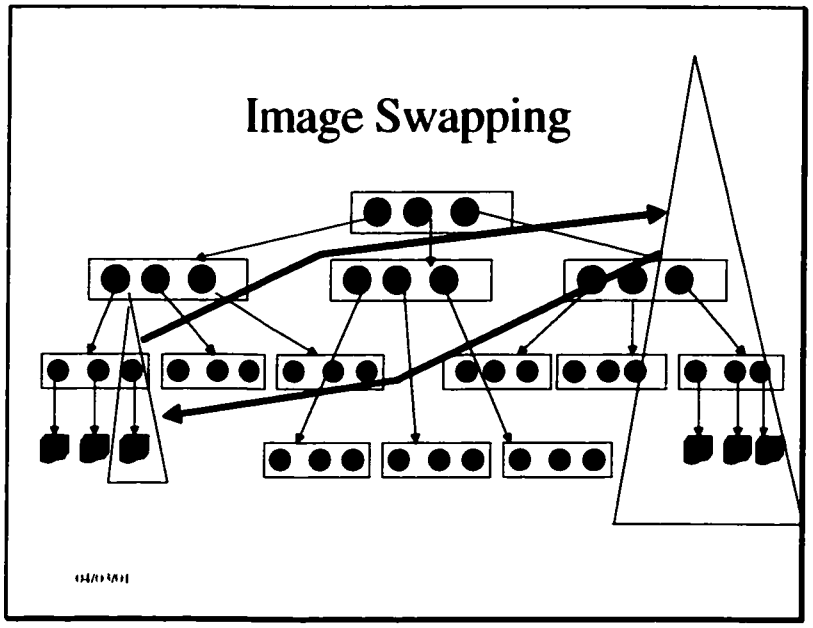



**Figure 5 12**: Swapping of two images means swapping the whole tree below.

The swapping of the whole tree ensures the previous image relationship with its children stays the same as before. For example, consider a nearly-converged catalogue that has a *Parent* image of a car and this image has a whole tree of different kinds of cars below it in the image tree. Swapping this image with its tree ensures the car image continues to be related to the images in the tree below it.

*Penalty process (Both images are not in the same class and both of them are in boundary state)*

If the two images in the query < $A_i$, Aj> are not in the same class and both of them are in the boundary state of their class, both will be panelized. One of those two queried images will be arbitrarily picked to be swapped to the other image class. It is swapped to the other class with an image that has the closest state to the boundary state. Both of the swapped images will be assigned the boundary state of their new class.

***Special case***

If the two images in the query < $A_i$, $A_j$> are not in the same class and one of them is the *Parent* of the other one, both will be rewarded as if they both belong to the same class. This case is classified as special case because it is always the case that if the two images are not in the same class, both will be penalized. In this case, however, both will be rewarded for having the right relationship with each other. They are both seen in the same view with one of them being the *Parent* of the other. To stress out this relationship, the *Parent* image is rewarded by moving its headState one step towards the most internal state. The *Child* image state is also moved one step toward the most internal state. This ensures that the *Parent* image stays as the *Parent* after this step and also affirms the membership of the *Child* image to this class. If both images are already in their most internal state, no operation are performed. This operation, was not permitted in the IDLA system [OOM93] because only two levels of images were permitted.

Since there are changes in the states of images of any class, there may be an image now with a state closer to the most internal state than its *Parent headState*. Therefore, the state $C_S$ of the image C with the closest state to the most internal state is compared with the *headState* $P_{HS}$ of the *Parent* image P of the class. If $C_S > P_{HS}$, no action will be taken and the *Parent* image of the class remains the representative image of the class in the upper class. However, if $P_{HS} > C_S$, then both images will be swapped according to the following rules:

1. The names of the images are swapped.
2. $C_S$ becomes the new $P_{HS}$.
3. $P_{HS}$ becomes the new $C_S$.
4. *Parent* image state stays as is. Since the image names are swapped, this means that the new *Parent* image will inherent this state from its previous *Parent*.
5. *Child* of the new C is inherited from the old C.

No pointer swapping that involves moving the whole tree is done at this step, only name changes and states swap are done here. Figure 5.13 shows how the *Parent* image loses its parent status to its children, and is swapped with one of its children that has the closest state to the most internal state. This *Child* becomes the *Parent* of the class, and a member of the upper level class.

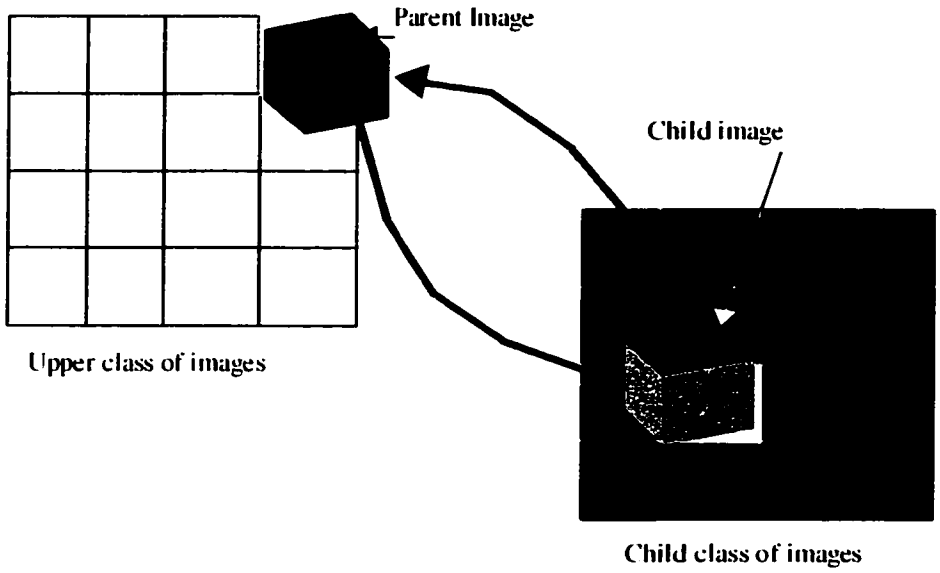



**Figure 5.13**: *Parent* swapping for a class based on the *headState* of the *Parent* image and its children states.

After the reward/penalty process completes and the *Parent* image for the class is set, images of altered classes will be sorted according to their states in an ascending order. This will result in the images of a class being shown to the user going from the closest image to the most internal state toward the most external state.

### 5.4.4 Example

To explain the hehaviour of MAIRS, we demonstrate the system progression based on a typical user interaction with the system. It is assumed that all images have not been partitioned before, and therefore they are in the most external state and *headStates*. Figure 5.14 illustrates the user action of browsing through three *ImageClasses* in which one of them has two images that are parents of the other two classes. The images that are shown in the white color are not considered in this example and are assumed to not be

relevant to this particular example. The user starts at *ImageClass* 1 and finds two images that are similar to his/her intended image, shown in Figure 5.14 as $S_1$ and $S_2$. The user, then, goes to examine all the images underneath one of the similar images, which is $S_2$. He/she will mark image $S_3$ as similar too, and go back up to the upper level. The user finally examines all the images below the similar image $S_1$, and finds the image marked as found, F, in Figure 5.14. All similar images and found images are added to a list referred to as the similarity list.

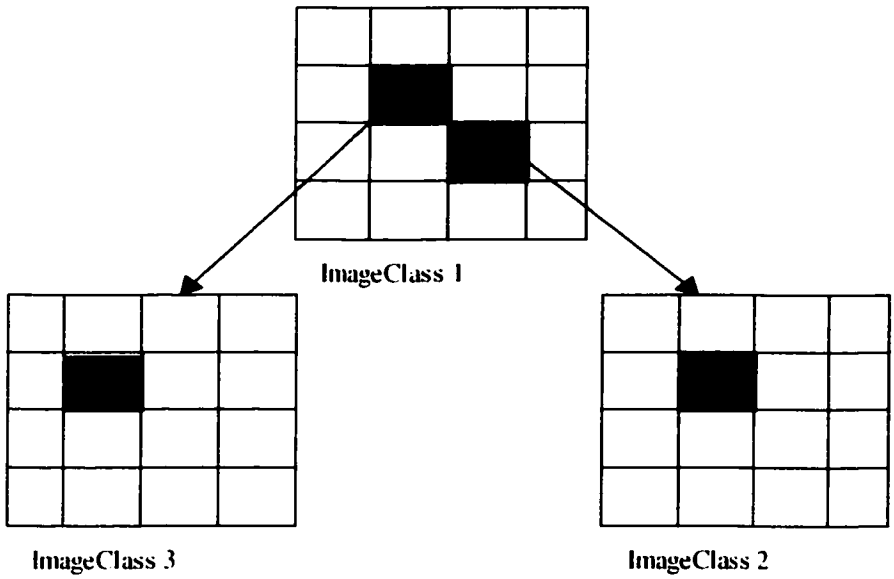



**Figure 5.14**: The user selects few images as similar before finding the sought for image.

Once the user marks an image as found, the image will be shown to the user in its normal size, and the system will start its process of migration by considering each combination of two images in the similarity list. This process will be referred to as query processing. During this process, the system will decide to reward or penalize images for being in the 'right' class or 'wrong' class. As a result of a penalty, object migration might

take place too. As a result of the above user selection, the following queries will be executed:

- <F, $S_1$>
- <F, $S_2$>
- <F, $S_3$>
- < $S_1$, $S_2$>
- < $S_1$, $S_3$>
- < $S_2$, $S_3$>

Figure 5.15 shows the outcome of the first query, which is for the found image marked with F, and the first similar image marked with $S_1$. Although both images are not in same class, both will be rewarded for having the correct *Parent/Child* relationship. The image $S_1$ is rewarded by moving its *headState* one step towards the most internal state and the image F is rewarded by moving its state one step towards the most internal state. This will ensure that the correct *Parent/Child* relationship stays the same when the images in the class are sorted, and the *Parent* image is selected.

Figure 5.15 shows the movement of objects since the class sorting and *Parent* image selection for this case takes place after executing each query. The movement to sort the objects will happen within the class, according to their states. Image F is moved to be the first of its class as a result of having its state decreased by one, which happens to be the closest state to the most internal state of its class.

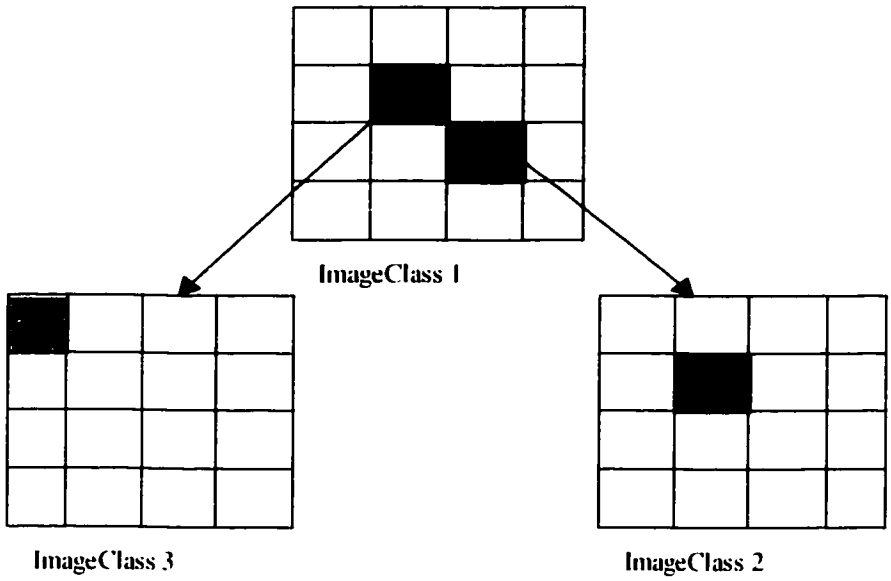



**Figure 5.15**: A query of images F and $S_1$. F is rewarded by moving its state one step towards the most internal state. $S_1$ is rewarded by moving its *headState* one step towards the most internal state.

The next query is for the images F and $S_2$. Both images here do not belong to the same class and therefore a penalty will be executed. Image F is penalized by moving its state one step towards the most external state. Note here that the image F has moved one step towards the most internal state in the previous query. This query will reverse the previous query and therefore image F is placed back into the most external state. Since image $S_2$ is already in the most external state of its class and it is about to be penalized again, the penalty is accomplished by swapping $S_2$ to the class that contains the F image. To keep equal number of images per class, an image that has not been accessed and is closest to the most external state is chosen to be swapped with $S_2$. Figure 5.16 shows this swap, where the image $S_2$ is swapped from ImageClass 1 to ImageClass 3.

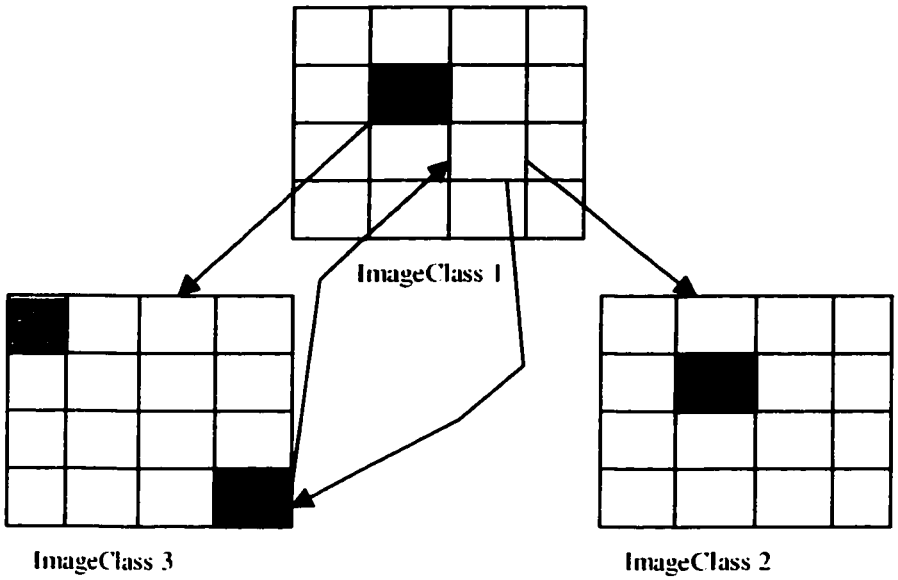



**Figure 5.16**: Image $S_2$ is moved from ImageClass 1 to ImageClass 2 as a result of being penalized after executing query <F, $S_2$>.

The next query is <F, $S_3$>. Both F and $S_3$ are in not in the same class and in the most external state of their class. In this case, $S_3$ will be swapped to ImageClass 3 with another unaccessed image of ImageClass 3. Image F stays in the same location and state. Figure 5.17 shows the swap of image $S_3$ from ImageClass 2 to ImageClass 3.

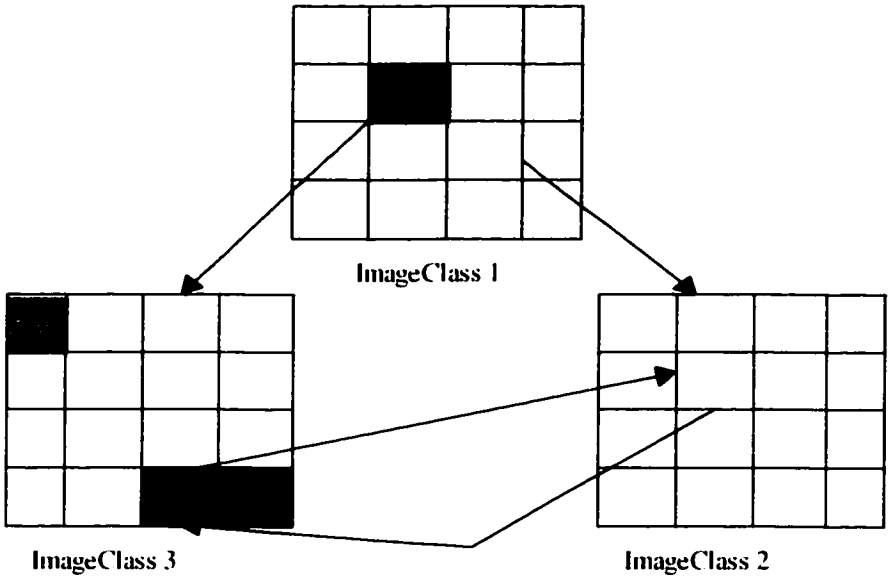



**Figure 5.17**: Image $S_3$ is swapped to ImageClass 3 as a result of executing query <F, $S_3$>.

Before considering the remaining queries < $S_1$, $S_2$>, < $S_1$, $S_3$>, and < $S_2$, $S_3$>, we shall make a note of the current states of all images. Image $S_1$ is one step away from the most internal state. The images F, $S_1$ and $S_3$ are in the most external state of their class, which is ImageClass 3.

Query < $S_1$, $S_2$> will result in moving $S_1$ *headState* one more step towards the most internal state, and $S_2$ one step towards the most internal state.

Query < $S_1$, $S_3$> will result in moving $S_1$ *headState* one more step towards the most internal state, and $S_3$ one step towards the most internal state.

Query < $S_2$, $S_3$> will result in moving $S_1$ and $S_2$ one more step towards the most internal state. Table 5.1 shows the updated values of states and *headStates* of all images considered.

| | State | headState |
|---|---|---|
| F | Most external | Most external |
| $S_1$ | Most external | Most external - 3 |
| $S_2$ | Most external - 2 | Most external |
| $S_3$ | Most external - 2 | Most external |

**Table 5.1**: The final states/headStates of all images considered before sorting ImageClass 3 member images.

After each query, the final two steps of the algorithm are to select a *Parent* image and sort images according to their states. For simplicity, we outline this process for all the three queries together since the final result will be the same as if we discussed the process after each query. The *Parent* image is selected by going through all images in the class and finding the image with an internal state that is the closest to the most internal state. From Table 5.1, this image can either be $S_3$ or $S_2$. Lets assume that image $S_3$ is picked to be the one with the closest state to the most internal state. The state of image $S_3$ is compared with the headState of image $S_1$. Since headState of $S_1$ is less than the state of $S_3$, $S_1$ remains the as *Parent* image of this class.

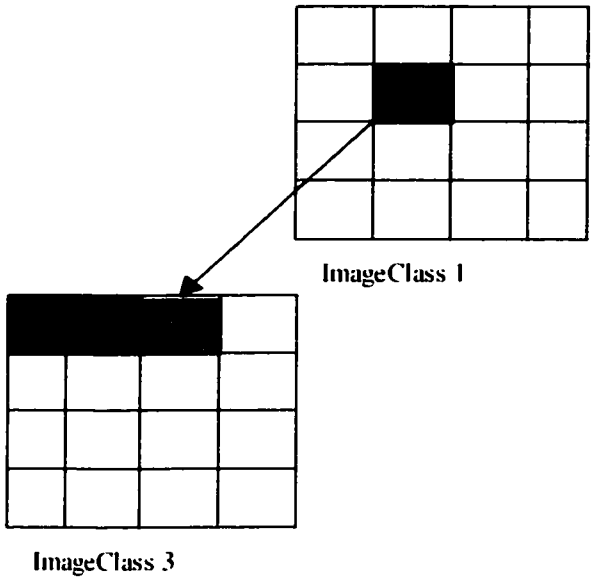



**Figure 5.18**: The final result after executing all queries. Images with the closest states to the most internal state are placed in the first places of the class.

Figure 5.19 shows what the user will see in the upper level ImageClass 1 and what he/she will see if the child images at the level lower to $S_1$ are to be examined.

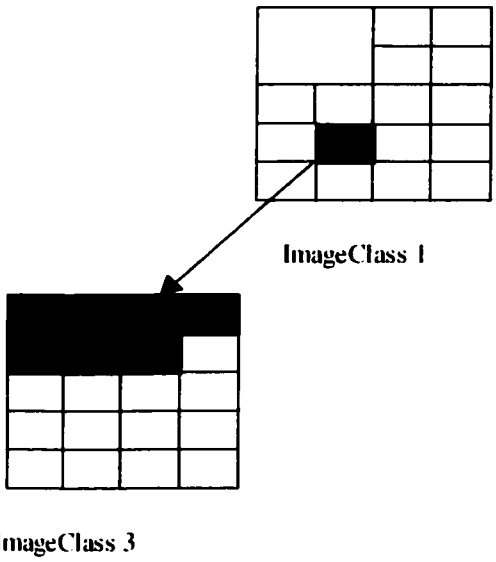



**Figure 5.19**: The user would see images sorted according to their states after all queries are performed.

### 5.4.5 MAIRS system Pseudo-code

We now present a formal Pseudo-code for the algorithm. This is doen by starting with the system that includes the main loop of waiting for the user, action and the calls to the other procedures that achieve the actual querying of images.

```
PROCEDURE MAIRS_System
  S← { }                                        /*Initialize set of similar images */
  PopulateCatalogue(UserDirectoryOfFiles):
  Repeat
        Get(UserAction)
        Case(UserAction) Of
                ImageFound:
                  DisplayImage(M):
                  S← S ∪ M                      /*Add found image to similar list */
                  For i ← 1 to Size(S) –1 by 1 Do   /*Found image with all similar list */
                        PROCESS_MAIRS(Image(M), Image(i))
                  EndFor
                  For i ← 1 to Size(S) –2 by 1 Do   /*Similar images with with each other */
                        For j ← i +1 to Size(S) –1 by 1 Do
                              PROCESS_MAIRS(Image(i), Image(j))
                        EndFor
                  EndFor
                ImageSimilar:
                  If(M) in similar list Then
                        S← S ∩ M                    /*Remove image from similar list */
                        RemoveSimillarSign
                  Else
                        S← S ∪ M                    /*Add image to similar list */
                        HighlightImageWithSimilarSign
                  EndIf
                ImageView:
                  DisplayImage(M):
                DisplayImageClass:
                        If (M->child)                       /*Image has a child */
                              CurrentImageClass = M->child:
                              DisplayImagesOf(M->child)
                        EndIf
        EndCase
END PROCEDURE MAIRS_System
```

```
PROCEDURE Process_MAIRS(iImage, jImage)
{
        /*
         First: the reward process where both files
         are in the same class and are not in internal state
        */
        If (iParent == jParent) Then
                If (iImage.state>MOST_INTERNAL) Then
                        iImage.state← iImage.state -1;
                        SortClassByStates(iParent);
                EndIf
                If (jImage.state>MOST_INTERNAL) Then
                        jImage.state← jImage.state -1;
                        SortClassByStates(jParent);
                EndIf
        /* Special case when one image is the parent of the other image */
        ElseIf (iImage == jParent) Then
                If (iImage.headState > MOST_INTERNAL) Then
                        jImage.headState ← jImage.headState -1;
                EndIf
                If (jImage.state > MOST_INTERNAL) Then
                        jImage.state← jImage.state  -1;
                        SortClassByStates(jParent);
                EndIf
        EndIf
        ElseIf (jImage == iParent) Then
                If (jImage.headState > MOST_INTERNAL) Then
                        iImage.headState ← iImage.headState -1;
                EndIf
                If (iImage.state > MOST_INTERNAL) Then
                        iImage.state← iImage.state  -1;
                        SortClassByStates(iParent);
                EndIf
        EndIf
        /*
         Third: Penalty process where the two files are not
         in the same class and both are internal states
        */
        ElseIf
                        (iImage.state<MOST_EXTERNAL) &&
                        (jImage.state<MOST_EXTERNAL)
                        ) Then
                        iImage.state← iImage.state +1;
                        jImage.state← jImage.state +1;
                        SortClassByStates(iParent);
                        SortClassByStates(jParent);
        /*
         Forth: Penalty process where the two files are not
         in the same class and one of them is in the
         external state of its class
        */
```

```
ElseIf (iImage.state==MOST_EXTERNAL) Then
        If (iImage.state<MOST_EXTERNAL) Then
                iImage.state← iImage.state +1;
        EndIf
        SwapImages(iImage, iParent);
        SortClassByStates(jParent);

Else
        jImage.state← jImage.state +1;
        SwapImages(iImage, jParent);
        SortClassByStates(iParent);
        SortClassByStates(jParent);
EndIf

END PROCEDURE Process_MAIRS
```

### 5.4.6 Evaluation

The MAIRS system is an intelligent and easy system which can be used for efficient image search and retrieval. The best feature of the system is its ability to relieve the user from the burden of providing search criteria information, a sketch, or an example image.

The MAIRS system uses the skeleton form of its predecessor, the IDLA [OOM93], and thus has all the advantages of the IDLA system in terms of implementation and efficient use of resources. In addition, the MAIRS system is easy to use since it has the same style as a typical MS Windows application. The system overcomes the limitation of the IDLA system by supporting a multilevel image-tree structure. This provides the capability of achieving sub-grouping of images and the ability to move abstract images from one class to another whether the new class is at the same level of the tree, in a higher or lower class. In our implementation, this task was

accomplished by utilizing the access of the *headState* variable of the *Parent* image within its *Child* class members. The MAIRS system also supports different image file formats.

The performance of the MAIRS system has not been fully studied due to the fact that the use of the system is subjective and is different from one user to another. One psychologically-based way to do a true evaluation of the system is to request hundreds of users to employ the system and to measure the average time it takes them to interact with the system for a variable/fixed sets of queries. This approach requires a great deal of resources, is time consuming, and is indeed a project for the cognitive sciences. Another approach is by simulating different users' behaviors via a computer program. This approach, however, is still an open problem, as the query generation process is still to be formalized. This work is currently being investigated by Ooomen. In addition to the difficulties encountered with performance testing of the system, the issue of testing against well-defined benchmarks is still open.

### 5.4.7 Future feature

The MAIRS system relies on the swapping of images to implement a correct object migration algorithm. When the image tree consists of 2 levels, this can be done easily with no impact on the balance of the tree. However, for multilevel trees, the philosophy is more complicated. Moving or swapping of an image means swapping the whole sub-tree below the image. If this is done several times, this could create an unbalanced tree, which could be less user friendly. This problem can be overcome by attempting to balance the tree after each search concludes, which, in turn, can be done by

reallocating the unaccessed images from one side of the tree to the other side to make the tree balanced.

Certain users may desire to see a certain image within a given class all the time, or to see a certain image excluded from a given class all the time. This could be an added feature that will allow the user to add or delete an image from a class or the database. Of course, adding or deleting an image from a class will have to be done in an organized way so as to keep the classes to be of the same size.

Users may also want to be able to manipulate certain images by, for example, enlargements, or by emphasizing or de-emphasizing certain colors or features. These can be accomplished by adding the necessary routines as add-on libraries and should be a straightforward exercise.

The MAIRS system can be converted into corresponding Intranet/Internet versions so as to serve the needs of employees within a company or to be an image search engine globally.

## 5.5 Conclusions

This chapter presents a system that utilizes the OMA solution for the EPP problem to solve the image retrieval problem. The existing methods for solving the image retrieval problem expect the user to provide search criteria, which is often not practical in

some cases. The OMA solution to the problem was originally prototyped in a skeleton form by Oommen and Fothergill with the IDLA system. In this chapter, we presented complete description of the IDLA system so that the design of MAIRS system could be explained.

The MAIRS system is the main contribution of this thesis, which is a multilevel image search and retrieval system. The MAIRS system builds on the advantages of the IDLA system in terms of correctness and efficiency. Furthermore, the MAIRS system provides an easy-to-use and more flexible solution by allowing sub-grouping of images in arbitrary levels, and by having the capability to consider most of the known image formats. In conclusion, we also discussed features that can be added to further enhance the already powerful capabilities of the currently implemented system.

We plan to publish the work of this project in a scientific journal and we will study the possibility of making the project a commercial product.

# Chapter 6: CONCLUSIONS

## 6.1 Summary of achievements

This thesis presents a new approach to the image search and retrieval problem. The previous methods for image search and retrieval imposed search criteria on users that were often not practical. The method presented here allows the user to search based on a mental view of the intended image.

In this thesis, we have presented a complete image search and retrieval system. The philosophy of the system approach is based on an automaton solution earlier presented by Oommen and Ma [OOM88], known as the OMA, to solve the Object Partitioning Problem (OPP). This solution utilizes learning automata. The OMA studies a special case of the OPP where the objects are equally divided among the given classes, and is known as the Equi-Partitioning Problem (EPP). We have examined other existing solutions to the EPP. The previous solutions included the Basic Adaptive Method (BAM) [YU81] and the Tsetlin and Krinsky automata based solutions. Since the previous solutions were proven to be infeasible for large numbers of classes and objects, the OMA solution was used as the foundation for the system proposed in this thesis.

The solution extends the OMA solution to handle a multilevel solution for the image search and retrieval problem. The system we have designed and implemented is called the Multilevel Automaton-based Image Retrieval and Searching (MAIRS) system.

As the name implies, the MAIRS is a multilevel system. It provides an easy-to-use and flexible solution that allowing sub-classing of images in arbitrary levels. It also has the capability to consider most of the known image formats. A detailed description of the system and its user interface is found in Chapter 5 of this thesis.

## 6.2 Future Research

We suggest a few features that can be added to the system to make it even more user-friendly.

Image tree balancing can be done after each search by reallocating the unaccessed images from one side of the tree to the other side to make the tree balanced.

A future implementation of MAIRS could provide the user with the choice to add or delete an image from and to a class, or from the database itself.

Future implementation of MAIRS could also support image manipulations such as enlargements, emphasizing, or de-emphasizing of certain colors or features.

The MAIRS system can be utilized to be an Intranet/Internet system so as to serve the needs of employees within a company or to be a general-purpose image search engine.

We believe that MAIRS has great potential, and with marginal modifications it could become a commercial product. Indeed, we also plan to publish the fundamental contributions of this thesis in a scientific journal.

# REFERENCES


[ALB98] E. Albuz, E Kocalar, and A. A. Khokhar, "Scalable Image indexing and Retrieval using wavelet," Department of Electrical Engineering and Computer Engineering, University of Delaware, November 1998.

[ASH95] J. Ashley, R. Barber, and M. Flickner. "Automatic and semi-automatic methods for image annotation and retrieval in QBIC," In *Proceedings in Storage and Retrieval for Image and Video Database-III*, Feb 1995, Volume 2420, San Jose, CA, pp 24-35.

[BEL98] S. Belongie, C. Carson, H. Greenspan, and J. Malik, "Color-and-Texture-Based Image Segmentation Using EM and Itxs Application to Content_Based Image Retreival," *ICCV* 98.

[GUP97] A. Gupta, R. Jain, "Visual Information Retrieval," *Communications of the ACM*, May 1997, Volume 40, Number 5, pp 71-79.

[HIR92] K. Hirata and T. Kato. "Query by visual example, In Advances in Database Technology EDBT '92," *Third International Conference on Extending Database Technology*, Vienna, Austria, march 1992. Spring-Verlag.

[HSU95] W. Hsu, C. T. S, and P. H. K. "An Integrated Color-Spatial Approach to content-based Image Retrieval," Department of IS and CS, National University of Singapore.

[JAC95] C. E. Jacobs, A. Finkelstein, and D H. Salesin, "Fast Multiresolution image querying," *In Proceedings of SIGGRAPH 95*, 1995.

[KAI94] G. Kaiser. *A Friendly Guide to Wavelets*, Birkhauser, 1994.

[KEL95] P. M. Kelly, Donald, R. Hush, and M. Cannon, "Query by image example: the CANDID approach," *In SPIE Proceedings Storage and Retrieval for Image and Database*, 1995, pp 238-248.

[LIA97] K.C. Liang and C.C.J.Kuo. "Wavelet-compressed image retrieval using successive approximation qunatization features," *In SPIE Voice, Video, and Data Communications*, Dallas, Texas, November 1997.

[MAL89] S.G Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Number 11, pp 674-693, 1989.

[MAM95] W.Y. Ma and B.S.Manjunath, "Pictorial queries: Combining feature extraction with database search," *Technical Report 18*, University of California at Santa Barbara, Department of Electrical Engineering. 1995

[NIB95] W. Niblack, M. Flickner, H. Shawney, J. Ashley, Q Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steel, and P Yonker, "Query by Images and Video Content: The QBIC System," *IEEE computer*, September 1995, Volume 28, Issue 9, pp 23-31.

[NAR89] K. Narendra and M. A.L. Thathachar, *Learning Automata, An Intorduction*, Prentice-Hall, Inc., 1989.

[OLG95] V. E. Ogle, and M. Stonebarker, "Chabot: Retrieval from a Relational Database of Images," *IEEE Copmuter*, Vol. 28, Issue 9, September 1995, pp 40-48.

[OOM88] B. J. Oommen and D. C. Y. Ma, "Deterministic learning Automata Solutions to the Equipartitioning Problem," *IEEE Transactions on Computers*, January 1988, Volume 37, number. 1, pp 2-13.

[OOM93] B. J. Oommen and C. Fothergill, "Fast Learning Automaton-Based Image Examination and Retreival," *The Computer Journal*, 1993, Vol 36, No. 6, pp 542-553.

[PAS97] G. Pass, and R. Zabih, "Histogram Refinement for Content-Based Image Retrieval," Computer Science Department, Cornell University. 1997

[PEN94] A. Pentlan, R. Picard. and S. Sclaroff, "Photobook: Content-Based Manipulation of Image Database," *SPIE Storage and Retrieval Image and Video Database II*, No. 2185, Feb 6-10 1994, Vol.2 185, pp 34-47.

[RIC98] Richard Leinecker and Tom Archer, "Visual C++ ® 6 Bible," IDG Books Worldwide, 1998.

[ROH95] Rohini K. Srihari. "Automatic Indexing and Content-Based Retrieval of Captioned Images," *IEEE computer*, September 1995, Volume 28, Issue 9, pp 49-56.

[SMI96] John R. Smith and Shih-Fu Chang. "Visualseek: a fully automated content-based image query system," *ACM Multimedia*, 1996.

[STR95] M. Stricker and M. Orengo, "Similarity of Color Images, " *Communications Technology Laboratory*, Swiss Federal Institute of Technology, ETH, CH-8092 Zurich, Switzerland. SPIE 95.

[THA79] M.A.L. Thathachaar and B.J. Oommen, "Discretized Reward-Inaction Learning Automata," *J. Cybern Inform. Sci.*, 1979, pp 24-29.

[TSE62] M. L. Tsetlin, "On the Behavior of Finite Automata in Random Media," *Automation and Remote Control*, 1962, Vol. 22, pp 1210-19.

[VAR63] V. I. Varshavaskii and I.P. Vorontsova, "On the Behaviour of Stochastic Automata with a Variable Structure," *Automation and Remote Control*, 1963, Vol. 24, pp 327-333.

[WAN97] J. Z. Wang, G. Wiederhold, O. Firschein, and S. X. Wei. "Wavelet-based image indexing techniques with partial sketch retrieval capability," *Journal of Digital Libraries*, 1997.

[YU81] C. T., Siu, M. D., Lam, D., and Tai F. "Adaptive Clustering Schemes General Framework," *Proceedings of the IEEE COMPSAC conference*, 1981, pp 81-89.