# Image Representation, Indexing and Retrieval Based on Spatial Relationships and Properties of Objects

Euripides G.M. Petrakis

March 1993

# Image Representation, Indexing and Retrieval Based on Spatial Relationships and Properties of Objects

A dissertation

presented to the faculty of the

Department of Computer Science of the

University of Crete

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

by

Euripides G.M. Petrakis

# Acknowledgements

THIS DISSERTATION could not have been completed without the help of many people. First and foremost, the completion of this work owes to the support and encouragement I received from my parents. I feel also grateful to my thesis advisor Professor Stelios Orphanoudakis for his guidance, his tireless support, the patience and the confidence he showed to me and to my work.

I wish to thank the members of my dissertation committee, Professors Panos Constantopoulos, Costas Courcoubetis, Siegfried Stiehl and Stavros Christodoulakis for valuable contributions to this work. They got into deep questions and with their comments helped me to improve this dissertation in both content and presentation.

I feel grateful to my colleagues for their cooperation and friendship during all these years. I want to thank Adam Damianakis, Antonis Argyros, Maria Mamalaki, Tasos Sorilos, Costas Vezeridis, Thodoros Topaloglou, Mihalis Mertikas, Giorgos Bebis and all members of the Image Processing Laboratory. Especially, I want to thank Rena Kalaitzaki, our tireless secretary, for her friendship and support. I want to thank Manolis Lourakis for his cooperation and for the excellent work he did in implementing the user interface. He and Dimitris Lagouvardos did a lot of tiresome work for me. I wish them both best of luck with their graduate studies.

I am thankful to Professors Nicos Alvertos and Vasilis Moustakis and my colleague Thodoros Topaloglou for their comments and for the careful reading of the manuscript. Finally, I feel grateful to Doctor Manolis Vernadakis, Professor Manolis Kontopirakis and many more good friends of mine who always were there whenever I needed them.

This research was supported in part by the Department of Computer Science, University of Crete and the Institute of Computer Science, Foundation for Research and Technology, Hellas (FORTH). The AZ-VUB Hospital, Brussels and the Department of Diagnostic Imaging, Yale University School of Medicine provided most of the medical images used in this work.

# Abstract

IN THIS thesis, a new methodology is presented which supports the efficient representation, indexing and retrieval of images by content. Images may be indexed and accessed based on spatial relationships between objects, properties of individual objects, and properties of object classes. In particular, images are first decomposed into groups of objects, called “image subsets”, and are indexed by computing addresses to all such groups. All groups up to a predefined maximum size are considered. This methodology supports the efficient processing of queries by image example and avoids exhaustive searching through the entire image database.

The image database consists of a “physical database” which stores the original image files and a “logical database” which stores all image subsets together with their representations. The logical database consists of a set of data files each storing subsets of equal size.

Queries are resolved through the logical database. Searching is performed in two steps, namely “hypothesis generation” and “hypothesis verification”. Queries are distinguished into “direct access”, corresponding to queries specifying a number of objects which is less than or equal to the maximum size of image subsets stored, and into “indirect access”, corresponding to queries specifying more objects than the maximum size of image subsets stored.

The performance of the proposed methodology has been evaluated based on simulated images, as well as images obtained with computed tomography and magnetic resonance imaging. Measurements of the size of the answer sets and of the retrieval response times have been obtained. The results of this evaluation are presented and discussed.

This work completes and extends the work of others. In particular, the image representations used in this work may be considered as extensions of the representation of “2-D strings”. The classical framework of representation of 2-D strings is specialized to the cases of scaled and unscaled images. Based on 2-D strings, an indexing scheme and a retrieval strategy are proposed, which in contrast to 2-D strings, avoid the exhaustive search through the entire image database. The performance of the proposed methodology has been compared with the performance of existing techniques of 2-D string indexing and retrieval. The results demonstrate significant retrieval performance improvements.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

IN MANY applications, images comprise the vast majority of acquired and processed data. For example, in remote sensing and astronomy, large amounts of image data are received daily by land stations for processing, analysis and archiving. A similar need of processing, analysis and archiving of images has been identified in applications such as cartography (images are analog or digitized maps) and meteorology (images are meteorological maps). The medical imaging field, in particular, has grown substantially in recent years and has generated additional interest in methods and tools for the management, analysis, and communication of medical images. Picture Archiving and Communication Systems (PACS) are currently used in many medical centers to manage the image data produced by computed tomography (CT), magnetic resonance (MRI), digital subtraction angiography (DSA), digital radiography, ultrasound, and other diagnostic imaging modalities which are available and routinely used to support clinical decision making. It is important to extend the capabilities of techniques used in such application fields by developing database systems supporting the automated archiving and retrieval of images by content.

An “Image DataBase” (IDB) is a “system in which a large amount of image data are integratedly stored” [1]. Image data may include: the raw images themselves, attributes (e.g., dates, names), text (e.g., diagnosis related text), information extracted from images by automated or computer assisted image analysis, modality and image file header information (e.g., ACR/NEMA [2]) etc. Such data can be viewed as forming either “multimedia documents” [3], or “objects” [4].

Important considerations in the design and implementation of IDB systems are: image feature extraction, image content representation and organization of stored information, search and retrieval strategies, and user interface design. Traditional database concepts such as data independence, data integrity and control of shared information are also of great significance. Addressing such issues has become the object of intensive research activities in many areas of Computer Science over the past few years. In particular, advances mainly in the areas of Databases and Computer Vision research resulted in the development of techniques which can be used for purposes of image archiving, retrieval, and general IDB work. Furthermore, the development of optical disk technology has made the production and usage of such systems feasible, at a relatively low cost.

## 1.1 Problem Description

The effectiveness of an IDB system, which supports the archiving and retrieval of images by content, ultimately depends on the types and correctness of image representations used, the types of image queries allowed, and the efficiency of search techniques implemented. In selecting an appropriate type of image representation, an attempt must be made to reduce the dependence on the application domain as much as possible and to ensure some tolerance to uncertainty with regard to image content. Furthermore, image representations must be compact to minimize storage space, while image processing, analysis and search procedures must be computationally efficient in order to meet the efficiency requirements of many IDB applications. Query response times and the size of the answer set depend highly on query type, specificity, complexity, amount of on-line image analysis required and the size of the search. In addition, query formulation ought to be iterative and flexible, thus enabling a gradual resolution of user uncertainty. All images (and/or information related to images) satisfying the query selection criteria are retrieved and displayed for viewing.

The retrieval capabilities of an IDB must be embedded in its query language. Command oriented query languages allow the user to issue queries by conditional statements involving various image attributes (values of attributes and/or ranges of such values). Other types of image queries include: queries by identifier (a unique key is specified), region queries [5] (an image region is specified and all regions that intersect it are returned), text queries [6] etc. The highest complexity of image queries is encountered in queries by example. In this case, a sample image or sketch is provided and the system must analyze it, extract an appropriate description and representation of its content and, finally, match this representation against representations of images stored in the database. This type of query is easy to be expressed and formulated, since the user need not be familiar with the syntax of any special purpose image query language.

So far, in order to determine which images must be retrieved, content representations corresponding to all stored images are compared (one by one) with a similar representation extracted from the query image. Thus, retrievals can be inefficient due to the fact that comparisons often involve time intensive operations such as graph matching [7, 8]. Various other techniques, with lower time complexity, can be used to resolve such queries. An example of such a technique is matching based on “2-D strings” [9]. A successful match associates the query image with only part of a stored image (a subset of the objects it contains), but, it is usually the whole image which is retrieved.

The effectiveness of an IDB system supporting the archiving and retrieval of images by content can be significantly enhanced by incorporating into the IDB storage and search mechanisms efficient techniques supporting the indexing of images by content. However, the problem of indexing images by content is difficult due to reasons related to complexity and uncertainty inherent to image analysis and interpretation tasks, the large amounts of data involved in derived image representations, as well as the dependence of such techniques on the content of images relating to a specific application.

## 1.2 Overview of the Present Work

In this thesis, a new methodology is presented which supports the representation, indexing and retrieval of images by content. This methodology supports the efficient processing of queries by image example and avoids exhaustive searching through the entire IDB. Images may be indexed and accessed based on spatial relationships between objects, properties of individual objects, and properties of object classes. The proposed methodology works by decomposing a given image into groups of objects, called “image subsets”. All image subsets up to a maximum size are considered. The maximum size of image subsets is constant for a particular implementation and must be specified in advance.

The content description of each image subset is given in terms of certain types of spatial relationships between objects, such as the “left/right”, “below/above” and “inside/outside”, in terms of properties specific to individual objects, such as size (area), roundness (elongation), orientation with respect to a reference direction, and in terms of properties of object classes. Such image properties have the advantage of being generally useful for many kinds of images and imaging applications.

The objects contained in each subset are first ordered. Two ordering criteria are presented and discussed. The first ordering criterion can be used when images are scaled with respect to each other, while the second ordering criterion can be used only when images are at a fixed scale. Each of these ordered image subsets is then represented by a set of attribute strings corresponding to the set of properties involved in a particular image description. An image is indexed based on addresses computed to all the derived image subsets. In particular, indexing is performed by computing an address, first to each individual attribute string and then to each ordered subset as a whole.

The IDB consists of two parts: the “physical database” which stores the original image files and the “logical database” [10] which stores the image subsets together with their representations. The logical database consists of a set of data files each storing subsets of equal size. Each data file is partitioned in segments (data pages) of fixed capacity, each corresponding to a unique address. Each subset is stored together with image subsets having the same address. Overflows are handled by creating linked lists of data pages. The address space (i.e., number of different addresses) corresponding to a data file increases with the size of image subsets stored in that file and with the number of properties which are used for the computation of addresses, called “primary properties”. Pointers are implemented from the logical to the physical database representing the association between the stored image subsets and the images from which they have been derived.

All queries address the logical database rather than the raw image data stored in the physical database. Searching is performed in two steps, namely “hypothesis generation” and “hypothesis verification”. During the first step, a set of candidate images is retrieved. In particular, all images which contain at least one image subset having the same address with the query image are retrieved. The number of candidate images is always less than the total number of images stored. During the second step, all candidate images are compared (one by one) to the query image and the final answer set is produced. All images containing at least one image subset matching the query (i.e., having both equal size and same property strings) are retrieved and displayed for viewing. Representations of image subsets of equal size having the same property strings with those of the query are also retrieved and displayed.

Query processing depends on the number of objects contained in the query image. In particular, queries are distinguished into “direct access queries”, corresponding to queries specifying a number of objects which is less than or equal to the maximum size of image subsets stored, and into “indirect access queries”, corresponding to queries specifying more objects than the maximum size of image subsets stored. Direct access queries address files which store image subsets of the same size. Indirect access queries address the file which stores subsets of maximum size. In this case, the objects of a given query image are taken in groups of maximum size. Each group acts as a separate direct access query and retrieved a set of candidate images. The intersection of these sets is then obtained and used to hypothesize the existence of images matching the original query.

Evaluations of the performance of retrievals have been carried out based on a number of test queries addressing an IDB consisting of 1,000 simulated images. Measurements of both the size of answer sets (i.e., percentage of images returned with respect to the total number of images stored) and of the retrieval response times have been taken. To obtain average performance measures, the average performance to queries specifying an equal number of objects has been computed. Both, the size of an answer set and the retrieval response time decrease with the number of query objects and with the number of image properties used in image comparisons.

Direct access queries perform much faster than indirect access queries. Compared to direct access queries, indirect access queries incur an overhead due to the retrieval and processing of intermediate query results. The performance of indirect access queries improves when the maximum size of image subsets stored increases, since intermediate queries contain more objects and return smaller answer sets. Queries respond faster, since the intersection of these sets contains fewer candidate images. In general, the performance depends on the size of the address space corresponding to the addressed data file and on the maximum size of image subsets stored. The dependences of the performance on both the size of the address space and on the maximum size of image subsets stored are explored.

In developing a prototype IDB which supports the indexing and retrieval of images by content, we have chosen to work with two-dimensional tomographic images of the human body (i.e., CT and MRI) and an IDB of 226 computed tomographic (CT) and magnetic resonance (MR) images has been constructed. The choice of this particular type of images is based mostly on their relatively well-structured content and their widely acknowledged clinical significance. Prior to any processing, images are segmented into regions. Segmentations are performed interactively and under supervision. An appropriate user interface has been designed and implemented to facilitate the editing of primal segmentations, support the flexible formulation and drawing of example queries, and the browsing of query results. Examples of typical retrievals on this database are also presented and discussed.

## 1.3 Contributions of the Present Work

Representations in the form of strings, and in particular in the form of two-dimensional strings (2-D strings), have been previously proposed in [9]. 2-D strings represent the “left/right” and “below/above” relationships between objects contained in an image. Each individual object is represented by a single value corresponding to a class or a name. The representations used in this work may be considered as extensions of the original representations of 2-D strings to take into account the inclusion relationships between objects, as well as more than one object properties. Furthermore, two ordering criteria are introduced and the representation of 2-D strings is specialized to the cases of scaled and unscaled images respectively.

The methodology presented in this thesis and 2-D strings can both be used to resolve queries based on image content. However, the search based on 2-D strings is exhaustive: in order to determine which images must be retrieved, 2-D string representations corresponding to all stored images are compared (one by one) with a similar representation extracted from the query image. Three algorithms for 2-D string matching are considered. The proposed methodology outperforms 2-D strings in all cases of direct access queries. 2-D strings perform better in certain cases of indirect access queries (e.g., when the maximum size of image subsets stored is less than 5). The IDB of simulated images has been used as a testbed for all comparisons.

An independent research activity which took place in parallel to ours and was published at the time of writing [11], resulted in the proposal of a technique for image indexing which is also based on 2-D strings. This technique focuses entirely on the problem of indexing. In particular, an image is indexed based on representations corresponding to all pairs of image objects. Each pair of objects is assigned an index and entered into a hash table. Similarly, the objects contained in a given query image are taken in pairs. Each pair of query objects acts as a separate query and used to retrieve a set of images. The intersection of the retrieved sets contains images which are then compared to the original query.

There are certain overlaps between the technique in [11] and our methodology. However, our methodology takes into account, in addition to indexing, the problems of the representation and retrieval of images, as well as the problem of the evaluation of the performance of retrievals. Regarding indexing, the technique in [11] is similar to our approach in the case where the maximum size of image subsets which are stored is 2. Queries which specify more than 2 objects are indirect access queries and, therefore, incur a significant overhead due to excessive data transfers and processing of intermediate query results.

The addressing scheme proposed in [11] requires that the images to be stored in the IDB are known in advance. A preprocessing step is followed to derive a hash function which is optimum for this particular set of images (i.e., guarantees that no two pairs of objects are mapped to the same address unless they have the same properties). However, if new images are entered into the IDB, the hash function ceases to be perfect. The addressing scheme proposed in this thesis, requires no preprocessing, the number of images need not be known in advance and remains perfect regardless of the number of images which are entered in the IDB.

Previous work, referred to in the literature as “image indexing” or “spatial indexing” addresses the following problem: given a region, find all regions in the IDB which intersect it (“region queries”) or, given a point in an image, find all regions which contain it (“point queries”). Such queries (spatial queries) are common in applications where it is important to retrieve images based on the position and the size of individual regions or objects (e.g., CAD/CAM and geographic database applications). Techniques such as the R-trees [12] and techniques based on fractal curves [13] can be used to resolve such queries. However, such techniques cannot index or access images by content (i.e., in terms of relationships and properties of objects).

## 1.4 Thesis Outline

A review of related work done in the areas of Computer Vision and DataBases is presented in Chapter 2. Image preprocessing, the ordering criteria, the representation of image properties by attribute strings, and the mapping of attribute strings to addresses, are the issues discussed in Chapter 3. The indexing scheme and the storage file structure used are presented in Chapter 4. The retrieval of images by content, the search strategy, as well as the evaluation of the performance of retrievals are discussed in Chapter 5 followed by conclusions and directions for further work in Chapter 6. The representation of 2-D strings, extensions implemented as part of this work, and three algorithms for 2-D string matching are presented in Appendix A. Finally, a description of the user interface is given in Appendix B.

# Chapter 2

# Related Work

## 2.1 Introduction

ISSUES related to IDB system design and implementation have been raised previously by many investigators [14, 1, 15, 16]. However, a general methodology for the design of such systems has not been developed so far. Only when the requirements of a particular application domain have been determined, techniques of image analysis, description, and image content representation, with known database design methods, are adopted to develop an IDB system which satisfies these requirements. The analysis and representation of images in an IDB system, is closely related to the indexing, storage and retrieval of images by content. In the following sections, we discuss related issues separately for ease of presentation.

## 2.2 Image Analysis and Interpretation

Techniques of image analysis and interpretation have been used in other domains (e.g., character recognition, industrial automation, remote sensing, robot vision, medical imaging etc.) to describe and model images and image classes [17, 18, 19, 20]. The content of images can be determined, based on the correspondence between the derived description of a particular image and some appropriate model(s) of an image class [21, 22, 23]. So far, most of the techniques which have been developed are knowledge-based, making use of application and domain-specific knowledge and are not particularly well suited for image retrieval by content and general IDB work; this is mostly due to problems related to computational efficiency, uncertainty, and general knowledge representation issues [24, 25]. Nevertheless, for certain applications within well specified domains, it is possible to develop and implement techniques of image analysis, content representation and modeling which can be used to manipulate images in IDB systems efficiently.

Image descriptions are generally given in terms of properties of objects or regions, contained within the image and relationships among such objects or regions. An image description may also consist of property values which are defined for the image as a whole. In the case when image descriptions are based on object/region recognition and labeling, the descriptions can be represented by relational structures such as graphs. The specific properties which are used in image descriptions are derived from the raw image data and can be geometric (i.e., independent of pixel values), statistical or textural, or properties specified in some transform domain (e.g., Fourier, Hough). They can also be characterized as local or global depending on their short or long range impact.

The description of a given image is the task of low to intermediate-level vision. The task of high-level vision is to find correspondences between descriptions of specific images and models of image classes for the purpose of understanding and interpreting image content [22, 26]. The effectiveness of both low and high level vision is strongly constrained by the effectiveness of segmentation. The purpose of segmentation is to group image pixels into meaningful units on the basis of a number of predetermined criteria (i.e., proximity, similarity, continuity) [27, 28]. Segmentation becomes very difficult especially when images are noisy or taken under poor lighting conditions.

At each level of image description and modeling mentioned above, there are two general types of models which can be used to describe classes of images: declarative and procedural [29]. Declarative models consist of constraints on the properties of pixels, objects or regions and on the relationships among such image components. Procedural models are implicitly defined in terms of processes which generate or recognize images. An important class of procedural models is that of syntactic models, based on the definition of various types of grammars for the generation and modeling of arrays, region shapes and relational structures. Such grammars can also be attributed, allowing numerical parameter values to be included in the model which may be used to increase its specificity. Both declarative and procedural models can be fuzzy or probabilistic, involving probabilistic constraints and probabilistic control of syntactic rules respectively.

In order to interpret the content of a given image, so that objects and regions can be identified correctly, descriptions of the image at different levels must be matched with one of possibly many models of different classes of images. This task can become very difficult and computationally intensive if the models are complex and a large number of models must be considered. Matching with the complex models is attempted only when matching with simpler models has been achieved. Furthermore, in a top-down approach to model matching, the model may be used to guide the generation of appropriate image descriptions, rather than first generating the description and then attempting to match it with a model. Another alternative is to combine top-down and bottom-up processes in order to avoid certain problems with uncertainty involved in hypothesizing the existence of a particular object in the image on the basis of a model. Such problems are inherent in the top-down approach [22].

The above control strategies for image analysis and understanding are simplified when one is dealing with two-dimensional images taken under certain conditions of good lighting and low noise. Such images can be found in applications such as microscopy, character recognition etc. Many of the problems alluded to above are not encountered in this case and image descriptions and class models are easier to construct. The set of properties which can be used to build image descriptions is then smaller and complex model matching can be avoided. However, even in two dimensions, in applications such as medical imaging and remote sensing, images are often very noisy, segmentations are inexact, while image models are difficult to formulate and use.

## 2.3 Image Representation

Prior to storage, derived image descriptions need to be appropriately represented (mapped) in database storage structures and models (e.g., relational tables). The amount of stored data, data processing and communication overhead can be reduced when compact (economical in space) image representations are used. The choice of a particular type of representation is closely related to the types of image queries allowed and to the specific access mechanisms developed to process such queries. In general, we distinguish between two kinds of image representations: (a) representations of the image semantic or structural content given in terms of properties of objects or regions and relationships among them, and (b) representations specific to individual image objects or regions.

Representations of the image semantic or structural content are of particular importance. At one extreme, once an image description has been derived, it can be easily mapped to database relations, stored in database relational tables and indexed using the indexing mechanisms provided by a particular DBMS [30, 31, 32]. Such representations can be used to answer image queries (see Section 2.5) specifying constraints on property values. Queries by image example will have to be translated into conditional statements involving constraints on object properties and relationships. However, such queries are difficult to be processed due to reasons related to the complexity of search, especially when queries specify more than two objects and many image properties.

“2-D strings” [9] is one of a few representation structures originally designed for use in an IDB environment. The technique provides a simple and compact representation of spatial image properties in the form of two one-dimensional strings. Before the 2-D string representation of a given image can be derived, the image must first be segmented into disjoint regions or objects and transformed into a “symbolic image”. The positions of objects in a symbolic image are represented by their center of mass, while the objects themselves are represented by values corresponding to names or classes. By taking the objects in a symbolic image from left to right and from below to above, two one-dimensional strings are produced forming the 2-D string representation of the original image.

The technique of 2-D strings is applicable whenever information concerning object names or classes and the “left/right”, “below/above” relationships between objects is an adequate representation of the semantic or structural image content. This is the case in applications where images contain objects which are mutually disjoint and have rather simple shapes. However, there are applications in which images may contain overlapping objects having complex shapes. The spatial relationships between overlapping objects cannot be clearly identified based on their center of mass and, therefore, symbolic images and 2-D strings are not sufficient to provide a complete representation of the semantic or structural image content. To deal with such situations, various extensions of the original representations of 2-D strings has been proposed.

2-D C strings [33] is a representation structure based on 2-D strings. 2-D C strings provides a more precise and complete representation of object spatial relationships than 2-D strings in cases of images consisting of complex and overlapping objects. It is complete, in the sense that every spatial relationship can be derived from it. Overlapping objects are segmented into smaller subparts for which their spatial relationships can be classified uniquely either as “disjoint”, “same position” or “edge to edge”. Based on these three new types of relationships between object subparts, up to 169 different spatial relationships between any two objects can be identified and coded in the 2-D C string representation of a given image. However, the 2-D C string representation of an image is not as simple and compact as the original 2-D string representation, nor can it be produced very efficiently.

A symbolic image can be reconstructed from its 2-D or 2-D C string representation.

Symbolic images reconstructed from their stored 2-D or 2-D C representations can be used for browsing the IDB. 2-D or 2-D C strings can also be used for purposes of image retrieval by content (see Section 2.7.2). However, retrievals based on 2-D C strings are less efficient compared to retrievals based on 2-D strings.

Fractal codes are examples of representations of the second type. They are used to represent image regions of arbitrary shape and facilitate database search and retrievals to queries specifying points or regions (spatial queries). Such image representations are discussed in Section 2.8 in the context of spatial indexing techniques.

## 2.4 Image Data Modeling

There are various types of image data that need to be stored and manipulated within an IDB including original image files, image content descriptions, text descriptions, voice, attributes (e.g., names, dates) etc. In developing an IDB system which supports the efficient management of various kinds of image data, while being extensible and easy to use, an appropriate data model and a database management system (DBMS) must be adopted. Such a DBMS must provide persistent storage of both the model (database schema) and the data, as well as mechanisms for defining, creating, modifying and accessing both the model and the data. Furthermore, it must provide a query language, transaction management, concurrency control and authorization for a multiuser environment, as well as performance features such as secondary indexing and clustering. Such features and mechanisms are inherent within classical DBMSs such as the relational ones [34].

Earlier approaches to IDB systems design distinguish image data into: original image files, called "physical images", and image related data (i.e., descriptions, attributes etc.), called "logical images" [35, 10]. Physical and logical images are stored into a physical and a logical database respectively. There may be a need to treat physical and logical data independently since, in most cases, they serve different purposes. Access methods are usually applied to logical data rather than the vast amount of physical data. Furthermore, logical images are subject to updates, while physical images are permanently stored (e.g., on an optical disc), and retrieved only for viewing.

In earlier IDB systems (REDI, GRAIN, IMAID) [1] the logical database is implemented as a conventional database using the relational data model and a relational DBMS. This was a very natural way to extend conventional database technology to handle image data. The physical database is implemented as a separate image store holding the original image files. Pointers are implemented from the logical database to the images stored in the physical database. Prior to storing physical images, image compression techniques [36] can be applied to reduce storage requirements.

Recent proposals regarding the design of IDB systems and the management of image data are influenced by the "object oriented" approach [37, 4, 38, 39]. This approach offers a framework within which different types of entities (e.g., different kinds of image data) and operations (e.g., image processing functions, image access mechanisms etc.) may be uniformly represented as "objects". An object, is defined as either a primitive or composite entity integrating within the same representation both data and operations. Objects are grouped into "classes" which can also be objects. Object classes are organized into hierarchies thus, taking advantage of the property of "inheritance". PROBE [40] is an instance of an IDB system which has been designed based on the object oriented paradigm.

Other approaches consider image data to be part of "multimedia documents" [3]. A multimedia document can be regarded as a structured collection of image, voice, text and attribute data. MUSE [41] and MINOS [42] are two characteristic instances of multimedia document systems.

## 2.5 Image Query Languages

The retrieval capabilities of an IDB must be embedded in its query language. Query response times and the size of the answer set depend highly on query type specificity, complexity, amount of on-line image analysis required and the size of the search. Query formulation ought to be iterative and flexible, thus enabling a gradual resolution of user uncertainty. All images (and/or information related to images) satisfying the query selection criteria are retrieved and displayed for viewing. Furthermore, a query response can be refined by “browsing”: before a final selection is made, characteristic representations (e.g., icons, image miniatures) corresponding to all images contained in the answer set are displayed. Browsing can be especially helpful when the specification of pictorial content is ambiguous and it may be the only method for making a difficult final selection [29]. In addition, displaying such image forms, instead of the original images, avoids extensive data transfers through a communication network during retrievals.

Command oriented query languages allow the user to issue queries by conditional statements involving various image attributes (i.e., exact values of attributes or ranges of such values). Other types of image queries include: queries by identifier (i.e., a unique key is specified), region queries [5] (i.e., an image region is specified and all regions that intersect it are returned), text queries [6] etc. The highest complexity of image queries is encountered in queries by example. In this case, a sample image or sketch is provided, the system must analyze it, extract an appropriate description and representation of its content and, finally, match this representation against representations of images stored in the database.

Most of the known IDB systems make use of a command-oriented query language extended to manipulate all kinds of image data. For example, GRAIN [10] makes use of a relational query language (RAIN) which has been extended with additional commands for image display and sketch drawing. In addition, query formulation may be assisted through the use of appropriately designed user interfaces [43, 44], as well as by special purpose tools (e.g., graphic tools) and techniques such as the “zooming technique” [45] used in GRAIN. REDI [30] makes use of the QPE (Query by Pictorial Example) query language [46] which is an extended version of the relational QBE (Query By Example) language. Various pictorial query languages related to a specific underlying image representation have also been proposed: PSQL [47] is an SQL-based query language operating on representations based on $R^{+}$-trees [48]. PIQUERY [49] is an image query language based on the paradigm of QBE operating on “grid” image representations.

In most cases, the definition of an image query language is based on a conventional query model, such as a relational one. Relational query models, have the advantage of being powerful and simple. However, object oriented database models do not provide simple and powerful query languages. The main reason is that, the set and join operations cannot be easily defined on hierarchies of classes. Most of the recent proposals, such as the query model developed for ORION [4, 38], restrict the target of a query to a single class or a hierarchy rooted at that class. This, may be found to be a serious constraint in designing an image query language which satisfies the requirements of a specific imaging application.

## 2.6 Image Similarity

The definition of similarity criteria influences significantly the performance of retrievals. Similarity criteria can be either global or local depending on whether they are defined for images as a whole or for parts of images (e.g., objects). Similarity (conversely dissimilarity) is usually computed in terms of appropriate “distance measures” [50]. In general, distance measures are defined with regard to specific matching techniques and specific kinds of image representations [51, 52, 53, 54]. For example, a distance measure between two “attributed relational graphs”, representing the content of two images which are compared, is defined as the cost of the minimum cost transformations required in order to transform the first graph to the second [55, 31].

The similarity between images often depends on the definition of appropriate threshold values: when the value of a distance measure is less than a predefined threshold value, the match is successful; otherwise it is rejected. In particular, when the value of a global distance measure is less than the value of the corresponding threshold, the candidate image is considered similar to the query; otherwise it is considered dissimilar and the match is rejected. The similarity criteria are usually defined for pairs of entities (e.g., images, objects etc.). However, it can also be defined between an image and a group of images (e.g., an image and a class of images) or between two groups of images (e.g., two image classes).

## 2.7 Image Retrieval by Content

The highest complexity of image queries is encountered in the case of queries by example image. A query by example may specify: (a) an object (or a part of an object), in which case, all images containing objects which are similar to it (e.g., have similar shape) must be retrieved, and (b) the semantic or structural content of an image, in which case, all images containing similar objects having the same relationships with those between corresponding query objects must be retrieved.

### 2.7.1 Image Retrieval Based on the Shape of Objects

The problem of retrieving images containing objects which are similar to the objects specified by a query is transformed into a problem of "object recognition" or "object classification" which are well known problems in Computer Vision research. So far, a large number of techniques for object recognition and classification are known to exist. A review of this kind of techniques can be found in [56]. However, such techniques, are not particularly well suited for IDB work. This is mostly due to the following reasons: most of them are "model based", since they assume that the number and the kinds of objects (i.e., classes) to be recognized are known in advance. Others, such as those proposed in [57, 58, 59, 60, 61, 53, 62, 63], perform an exhaustive search: all database objects or object class models are compared (one by one) against all query objects. To our knowledge, none of the above techniques has ever been tested on large databases storing hundreds or thousands of objects or models.

An object recognition technique is well suited for the retrieval of images based on the shape of objects they contain, if: (a) it is robust against noise, (b) it can recognize similar objects even if they are partially visible and (c) it is translation, scale and rotation invariant (i.e., it can recognize similar objects at various positions, scales and orientations). Furthermore, it must avoid an exhaustive database search by allowing the indexing of objects based on characteristics of their shape. Some of the most important known techniques, which fulfill the above requirements, are reviewed in the following paragraphs.

The "geometric hashing" technique is one of the first to be proposed [64]. This technique uses translation and rotation invariant representations of contour segments called "footprints", obtained by taking sequences of constant-length segments and computing the "angle-arclength" ($\theta - s$) representation of their shape. Footprints are Fourier-analyzed and represented by feature vectors consisting of a number of their first order Fourier coefficients. Finally, footprints are indexed in a multi-dimensional feature space. The technique proposed in [65] uses a K-D-tree to index feature vectors derived in a similar manner: a number of local contour segments near high curvature points are obtained, represented in the angle-arclength space and transformed into multi-dimensional vectors using the Karhunen-Loeve expansion. A simpler technique is proposed in [66]: the contour of objects is first approximated by polygons. Groups of successive line segments, called "super segments", are obtained, represented by a set of attribute values, and entered into a hash table. This technique performs at a fixed scale. When a query object is given, the same process is followed, a representation similar to those of stored objects is derived and all objects having the same representation or the same class with it are retrieved.

A technique capable of indexing and recognizing partially visible objects at various scales and orientations has been proposed in [67]. Objects are represented by sequences of their most significant boundary segments obtained from the resampled curvature scale space. The database consists of two components: an Artificial Neural Network (ANN), which has been taught to classify contour segments and a model database consisting of multiple ANNs structured in a hierarchical manner. Each of these ANNs has been trained to recognize an object model based on a specific number of segment classifications. When a query object is given, the object contour is segmented starting from a coarse resolution and moving to a finer one. Segments obtained at a specific resolution are forwarded to the ANN representing the first component of the database in order to be classified. The proper ANN in the hierarchy is then chosen to perform the recognition of the unknown object, based on the segment classifications of the first component.

### 2.7.2 Image Retrieval Based on Structural Image Content

Queries specifying the semantic or structural content of images are difficult to be resolved: so far, in order to determine which images must be retrieved, content representations corresponding to all stored images are compared (one by one) with a similar representation extracted from the query image. Thus, retrievals can be inefficient due to the fact that, comparisons often involve time intensive operations such as graph matching [7, 8, 58, 31]. The time complexity of matching increases exponentially with the number of objects in the images which are compared. Various other techniques with lower time complexity, can be used to resolve such queries. Such a technique is matching based on 2-D strings [9]. Techniques such as [68, 69, 70, 71, 72, 73] provide alternative solutions to reduced complexity matching.

The similarity between two images (e.g., a query and a stored image) whose content is represented by 2-D strings, can be determined based either on exact or approximate matching techniques. In the first case, in order for two images to be similar, every object in the first (query) image has to be associated to at least one similar object (i.e., an object having the same name or class with it) in the second (stored) image and the matched objects in these two images must have exactly the same relationships. The problem of determining the similarity between two images is then transformed into one of 2-D string subsequence matching. Algorithms of polynomial time complexity for determining whether a given 2-D string is a two-dimensional subsequence of a second 2-D string can be found in [9, 74]. To our knowledge, algorithms for subsequence matching performing on 2-D C string representations of images has not been proposed so far.

The definition of similarity is less strict in the case of approximate matching techniques, since the 2-D string of the query need not be an exact subsequence of the second 2-D string. The approximate similarity between two 2-D strings can be determined based either on a maximum likelihood or a minimum distance criterion. In the first case, the similarity is determined based on the longest 2-D string subsequence that the two 2-D strings under consideration have in common. Regarding image retrieval, the problem is then to retrieve the most similar image from a set of stored images. This is the one having the longest common subsequence with the query among the images stored in the IDB. The problem of finding the longest common subsequence between two 2-D strings is transformed into a problem of finding the maximal complete subgraphs (cliques) of a given graph. Therefore, the 2-D string longest common subsequence problem has nonpolynomial time complexity. The problem of finding the longest common subsequence between 2-D strings is treated in [75]. The problem of finding the longest common subsequence between 2-D C strings is treated in [76].

The similarity between two two-dimensional strings can also be determined based upon a minimum distance criterion. For example, the minimum distance between two 2-D or 2-D C strings can be defined as the cost of the minimum cost transformations required in order to transform the first string to the second. Algorithms for determining the minimum distance between 1-D strings do exist [77, 78] and can be generalized to the case of two dimensional strings.

## 2.8 Image Indexing

The image indexing techniques proposed to date, referred in the literature as “region” or “spatial indexing techniques”, are related to certain types of image representations and to two specific types of image queries, namely “region” and “point queries” [5]. In a typical region query, a region (or object) is specified and all regions (or objects) that intersect it (i.e., share a common area with it) are returned. In point queries, a point is given and all regions (or objects) that contain it are returned. Such queries are common in applications where it is important to retrieve images based on the position and the size of individual regions or objects (e.g., CAD/CAM and geographic database applications). Techniques such as the “R-trees” [12] and techniques based on “fractal curves” [13] can be used to resolve such queries.

There are certain application domains (e.g., medical imaging, robotics, geographic database applications etc.) in which images need to be accessed by content. To our knowledge, techniques which support the indexing of images by content have not been proposed so far, with the exception of the technique proposed in [11]. The technique is based on 2-D strings. Each pair of image objects is assigned an index and entered into a hash table. Similarly, the objects contained in a given query image are taken in pairs. Each pair of query objects acts as a separate query and used to retrieve a set of images. These are images containing at least one pair of objects having the same index with it. The intersection of all retrieved sets is then obtained and used to hypothesize the existence of images matching the original query.

### 2.8.1 Database Indexing Techniques

Indexing techniques are used widely in a variety of database applications to increase the efficiency of retrievals. In particular, indexing ensures fast data retrieval by providing efficient address calculation of data in secondary storage based on the values of one or more attributes which uniquely specify data entities (records), called “keys”. Retrievals become much faster compared to those corresponding to an exhaustive file search.

Indexing techniques fall into two broad categories: those making use of tree structured indices, which achieve logarithmic access times and those based on “hashing” which achieve constant access times. Data records are stored in “data pages” or “data buckets” of fixed capacity. An attempt to store one more record in an already full page causes an “overflow”. Overflows are handled by “open addressing” or “separate chaining”, as well as with splitting overflowed pages. Furthermore, unless the keys are known in advance, it is impossible to guarantee that no “collisions” will occur, in which case, two or more records are mapped to the same address.

The “B-tree” is one of the first and most important tree-indexing techniques proposed so far. B-trees have become a standard and a large number of database systems along with a very wide variety of access methods have been proposed and implemented based on B-trees. The major variants of the B-tree are discussed in [79]. B-trees ensures fast access times by allowing high “branching ratios” (i.e., up to a large number of entries can be stored in each node of a B-tree). Therefore, tree height is usually very small. A B-tree adapts its storage space to the amount of stored data and always remains balanced, that is the length of each path from the root to each leaf-node (and therefore the access time to each leaf-node) is the same.

Hashing schemes are distinguished into “static” and “dynamic”. In static hashing schemes, storage space is allocated statically and cannot be changed during processing without excessive overhead. As the number of stored records increases, collisions and overflows become frequent thus resulting in deterioration of performance. Dynamic hashing schemes adapt their storage space dynamically to the amount of stored data. When a page overflows it is split and its contents are distributed between itself and a newly allocated page. Conversely, two pages may be merged, in which case a page is freed.

Dynamic hashing schemes can be further distinguished into those making use of a “directory” (i.e., an array of pointers to the actual data pages) and those without a directory [80, 81]. “Dynamic hashing” [82] and “extendible hashing” [83] are characteristic instances of techniques of the first category. In directory schemes (such as the above mentioned), retrievals can be completed in two disk accesses: one to access the directory (if stored on disk) and one to access the data page itself. Directoryless schemes assume that a large contiguous storage space is available. Instead of addressing a table, addresses are assigned to the data pages themselves. “Linear hashing” [84], “linear hashing with partial expansions” [85] and “spiral storage” [86] are characteristic instances of techniques of this category. However, directoryless hashing schemes cannot ensure retrieval in one disk access. Hashing schemes ensuring retrieval in one disk access do exist [87].

In comparison to tree indexing schemes, hashing schemes suffer from a serious drawback. Specifically, keys are stored unordered, thus making the treatment of “range queries” inefficient. In this case, all records having keys within a certain range of values must be retrieved. Techniques capable of treating such queries are discussed in Section 2.8.2. The techniques discussed so far perform on single keys. However, techniques which perform on multiple keys also exist. Earlier approaches, such as “inverted lists” and “multilists” [88] perform well for queries involving a single key chosen among a set of keys. Queries involving more than one keys, take much longer to answer. Others, such as the “K-D-B-tree” [89], can be regarded as extensions of one-dimensional techniques in multiple dimensions. Multi-dimensional indexing can be reduced to one-dimensional indexing by mapping every point of the multi-dimensional space into one dimension [13, 90]. This way, every multiple-key record is assigned a single-key and any of the known single-key indexing techniques can be used.

The most important among the known techniques for multi-dimensional indexing is the “grid file” [91], which was originally designed for multi-dimensional indexing and retrieval. This technique makes no distinction between primary and secondary keys. The grid file achieves retrieval in two disk accesses, one to access the directory (if stored on disk) and one to access the data page. Furthermore, the grid file supports efficient treatment of range queries.

### 2.8.2 Spatial Indexing Techniques

Spatial (region) indexing techniques fall in two broad categories: those based on tree-index structures and those based on “fractal curves” or “space filling curves” [13]. With the sole exception of the grid file technique, hashing techniques are not particularly well suited for region indexing and range queries since they store keys unordered. Tree indices avoid this problem, but result in slower retrieval times. For further reading on the subject, the reader is referred to [92].

Among the tree-indexing techniques, “R-trees” [12] seem to be most efficient. R-trees behave much like B-trees. R-trees are especially designed to index data represented by intervals in multiple dimensions (e.g., two or three-dimensional image regions). They are dynamic, since the storage space they require may grow or shrink gracefully according to the number of stored entries (e.g., image regions). Like B-trees they always remain balanced. R-tree nodes represent regions in the two-dimensional (in general multi-dimensional) space. Each such region completely covers the regions represented by all its sibling nodes. Regions corresponding to adjacent nodes may be overlapping. Searching always starts at the root of the tree and descends to the leafs. At each level, the decision of which path to follow is taken based on whether the query region is completely covered by the region represented by the nodes. Since overlapping among node regions is allowed, more than one nodes may need to be searched, thus resulting in a degradation of performance.

“$R^+$-trees” [48] avoid overlapping, and thus searching on multiple paths, at the expense of space while increasing the height of the tree. $R^+$-trees outperform R-trees (i.e., result in less disc accesses for searching the tree of indices) in most cases and especially when point queries are processed. $R^*$-trees [93] is yet another variant of the original R-tree which achieves better retrieval performance.

Similar to R-trees are a number of other techniques [94, 95], such as “k-d-trees” and “quad-trees” [96]. Compared to R-trees, quad-trees and k-d-trees have the disadvantage of not taking paging of secondary storage into account. They are mostly useful as image representation data structures or for indexing image regions in main memory. Indexing techniques performing in multiple dimensions such as the grid file can also be used for spatial indexing and range searching [97]. Image regions must first be represented by points in a higher dimensionality space. For example, a two-dimensional rectangle can be mapped to a point in a four dimensional space if each of its four coordinates (lower x, lower y, upper x and upper y) is considered to be a separate attribute.

A second important category of spatial indexing techniques is based on fractal curves [13]. Fractal curves provide a linear ordering of multi-dimensional points or a mapping of a multi-dimensional space onto one dimension in a way that preserves proximity: neighboring points in the multi-dimensional space are likely to be also neighbors in the one dimension. Such a mapping is called “distance preserving mapping”. Regarding image indexing, fractal curves work as follows: consider a disc being a one dimensional space on which two-dimensional image regions are to be stored (mapped). Region queries require mostly the retrieval of neighboring image points. After a distance preserving mapping has been applied, neighboring points are coded by successive code values. A technique for single-key indexing and capable of range searching, such as the B-tree technique, can be used to index the coded image regions. The number of disk accesses (and thus response time) required by range queries is then minimum and depends on how well the mapping which has been used preserves the distances in the one dimension.

The use of fractal curves to treat spatial indexing and searching was first introduced in [98]. “Z-curves” (known also as “Peano curves”) were proposed for mapping two-dimensional image regions onto one dimension. Image regions are represented by sequences of successive “Z-codes” (values), grouped on the basis of their common prefix and stored in database relations. To determine whether any two image regions are either overlapping or disjoint, a procedure similar to the classical “natural join” database operation is followed. Such a procedure is the basis of PROBE’s [40] “geometry filter” which acts as an optimizer for spatial queries. The purpose of PROBE’s geometry filter is to produce a list of all candidate image regions which are either overlapping or disjoint with respect to a given query region. Results are then fine-tuned by further processing. Any relational database management system can be easily extented to support spatial indexing and searching. Two more curves providing better distance preserving mappings than the Z-curve, namely the “Gray curve” [99] and the “Hilbert curve” [13] have been proposed as well. The Hilbert curve has been proven [13, 90] to outperform the other two (i.e., results in less disk accesses for searching the indices).

## 2.9 Image Database Systems

In early IDB systems [1, 14], the archiving and retrieval of images is mostly based on information already available in some form (e.g., dates, serial number etc.) or provided by a user (e.g., names, dates, quantities etc.). Such information is then stored in IDB tables together with pointers to the original image files which are stored separately. Accordingly, image retrievals are performed by providing values to specific key-attributes and by forming simple types of image queries (e.g., queries by identifier, by conditional statement etc.). Corresponding images are then retrieved and returned to the user for viewing or processing.

More recently, IDB systems have been developed which support the use of simple techniques of image analysis and representation of image content. Derived image representations are stored in the database together with other kinds of image data already provided (e.g., names, dates etc.) and which may be used by retrievals. REDI [30, 1] and GRAIN [10, 1] are two of the most important instances of this kind of IDB systems. From organizational and functional point of view the above two systems are rather similar. Both make use of a relational database in which logical images (i.e., image descriptions) are stored while, physical images (i.e., original images) are stored separately. Queries always address the logical database. Both use query languages (namely QPE and GRAIN respectively) which are extensions of relational query languages to handle image data (e.g., by providing new commands for image display, query drawing etc.). A third IDB system based on the relational approach and the $R^+$-trees for the indexing of image regions has been proposed in [47]. The system supports the efficient processing of region queries and makes use of the PSQL query language.

Modern IDB design proposals are mostly influenced by the object-oriented paradigm. PROBE [40] is one of the most important systems falling in this category. PROBE supports the processing, representation and retrieval of images based on Z-codes. There are two types of objects defined in PROBE, namely the “entities” which represent concepts, quantities, image objects etc., and the “functions” which represent properties of entities, relations among entities and operations on entities. Both types of objects are organized into classes which consist of objects sharing common characteristics. Classes are further organized into a “generalization hierarchy” [100], thus taking advantage of the property of inheritance. Database search is facilitated through the use of the geometry filter (see Section 2.8.2). ISR [101] is a newly proposed system based on “frame” representations of image data. ISR is based on low to intermediate level representations of image content such as points and polygonal lines. Using appropriate models and techniques (e.g., decision trees) higher level representations can be derived. ISR supports image retrievals based on proximity and simple image properties.

A prototype IDB system based on 2-D string representations is proposed in [9]. Database search is performed by comparing the 2-D string representation of a query with similar representations corresponding to all stored images. The system introduces the use of “iconic indices”, which are simple iconic representations of the original images reconstructed from their corresponding 2-D strings. Iconic indices are then used for image display and for browsing the IDB in place of the original images themselves. In [29, 102] 2-D strings are considered to be an integral component of a multimedia document representing the content of images in terms of properties of both objects and relationships. Document retrieval by image content can be done by specifying properties of objects, relationships among objects or both.

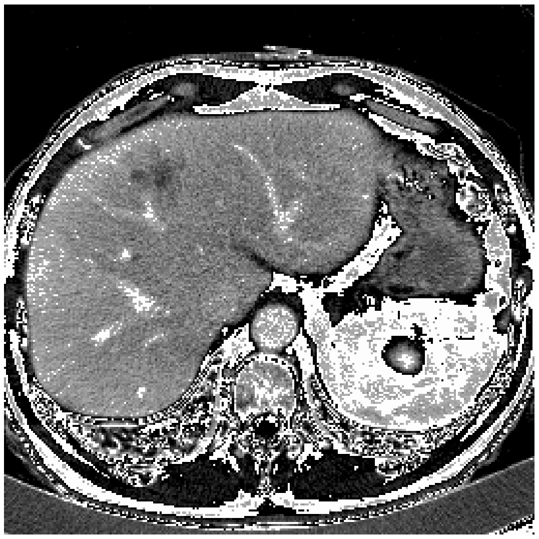
Other IDB designs have been proposed in the context of "Geographic Information Systems" [1, 103] in order to enhance their capabilities with efficient user interfaces [43] and flexible query languages [49]. In such systems emphasis is also given to issues related to the processing of maps such as the digitization and visualization of analog maps, the automated reading of printed names, the characterization of geographic forms (e.g., mountains, rivers, borders) etc.

# Chapter 3

# Image Representation

## 3.1 Introduction

THE PROPOSED methodology works by decomposing the set of objects contained in a given image into groups of objects, called “image subsets”. The content description of each image subset is given in terms of certain types of spatial relationships between the objects it contains such as “left/right”, “below/above” and “inside/outside”, and in terms of properties specific to individual objects such as size (area), roundness (elongation), orientation with respect to a reference direction, and properties of object classes. Such image properties have the advantage of being generally useful for many kinds of images and imaging applications. The objects contained in each group are first ordered. Each of these ordered subsets is then represented by a set of attribute strings corresponding to the set of properties involved in a particular image description. An address to each attribute string is then computed.



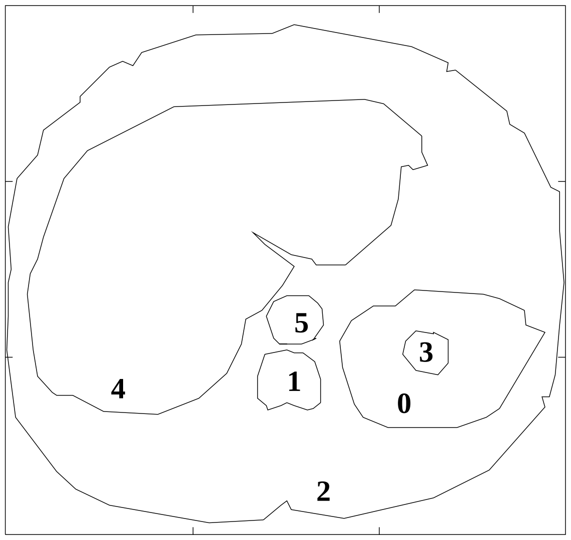


Figure 3.1: Example of an original grey-level image (left) and its segmented form (right).

## 3.2 Image Preprocessing

The search and retrieval of images by content, using the method to be described in this thesis, relies on image descriptions based on the segmentation of images into dominant disjoint regions or objects. Although accurate and robust image segmentation techniques are currently becoming available [104, 105, 106], this is an independent area of active research and is beyond the scope of this work. However, it should be pointed out that the requirement for accurate and robust image segmentation is more relaxed for indexing and retrieving images by content than it is for image analysis and image understanding. Thus, we have opted for a conventional image segmentation technique resulting in polygonal approximations of object contours. The desired segmentation results are obtained by editing (i.e., the user may delete insignificant segments or correct the shape of others). The edited segmented forms are then used to compute a variety of image features constituting a particular image representation, and for efficient browsing of the query response sets. Figure 3.1 shows an example of an original grey-level MRI image and its corresponding final segmented polygonal form. The segmented image contains 6 objects numbered 0 through 5.

## 3.3 Image Decomposition into Subsets

An image containing $n$ objects is decomposed into groups of image subsets. The subsets in each group contain an equal number of objects $k$, with $k$ ranging from 2 up to a prespecified number $K_{max}$. In particular, $k \in [2, \min(n, K_{max})]$. An image subset of size $k$ can be viewed as an answer to a possible image query specifying $k$ objects. Therefore, $K_{max}$ can be set equal to the maximum number of objects allowed in queries, if such a value can be specified in advance. Typically, the number of objects specified in image queries is not greater than 6. Therefore, we can arbitrarily set $K_{max} = 6$. In general, the value of $K_{max}$ depends on the application (see Section 5.5).

Producing all subsets of a given image containing $n$ objects is equivalent to generating all "combinations" of $n$ elements taking them $k$ at a time. Algorithms for producing combinations can be found in [107, 108]. Such algorithms have linear time complexity with respect to the number of combinations produced. For each value of $k$, $\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}$ image subsets are produced. Taking all subsets for all values of $k$, results in a total of $\sum_{k=2}^{\min(n, K_{max})} \binom{n}{k}$ subsets. For example, the image of Figure 3.1 which contains 6 objects, gives rise to 15 image subsets of size 2, 20 subsets of size 3, 15 subsets of size 4, 6 subsets of size 5 and 1 subset of size 6 (which is the original image itself).

Subsets of size 1 correspond to individual objects and are not taken into account. Emphasis is given to indexing images in terms of properties of relationships rather than properties of individual objects. However, objects can still be indexed and recognized on the basis of attributes corresponding to global object characteristics such as size, roundness etc., and attributes representing object classes.

Subsets are inherently unordered: no distinction can be made among subsets containing the same elements in different order. For example, subset (0, 1, 2) is equivalent to subsets (1, 0, 2) and (2, 1, 0). Prior to indexing, image subsets are ordered. Each object can then be associated, through its position, with attribute values. Ordering must be based on criteria that clearly differentiate the objects among them. Position is such a criterion, since objects are usually scattered in the image. However, size or shape do not necessarily provide good ordering criteria, since in certain cases images may contain similar objects. Therefore, we have chosen to base ordering mainly on criteria relating to position.

Ordering can be avoided if, instead of combinations, the "permutations" of $n$ objects taken $k$ at a time are used. Permutations can easily be produced from combinations by repeating the elements of each subset in all possible orders. However, the use of permutations results in a large space overhead. For example, 1910 subsets are produced from the example image of Figure 3.1.

## 3.4 Representation of the "Left/Right" and "Below/Above" Relationships

Two ordering criteria are presented and discussed. The first criterion can be used in cases where images are scaled with respect to each other, while the second ordering criterion can be used only in cases where images are at a fixed scale. Ordering is the first step towards producing the string representations of the "left/right" and of the "below/above" relationships between the objects contained in an image subset.

### 3.4.1 First Ordering Criterion

Let $(a_0, a_1, \ldots, a_{k-1})$, be an image subset of size $k$, where $k \in [2, \min(n, K_{max})]$. For any two objects $a_i$, $a_j$, where $i, j \in [0, k-1]$, with centers of mass $(x_i, y_i)$ and $(x_j, y_j)$ respectively, either $a_i$ is a "predecessor" of $a_j$ which is written as $a_i \prec a_j$, or $a_i$ is a "successor" of $a_j$ which is written as $a_i \succ a_j$. Specifically, the first ordering criterion is

| object | center of mass: $(x, y)$ |
|---|---|
| $a_0 = 0$ | (193.19, 88.58) |
| $a_1 = 1$ | (126.93, 77.69) |
| $a_2 = 2$ | (124.12, 122.60) |
| $a_3 = 3$ | (188.30, 90.94) |
| $a_4 = 4$ | (96.09, 141.16) |
| $a_5 = 5$ | (129.65, 104.12) |

Table 3.1: Centers of mass corresponding to the objects contained in the example image of Figure 3.1.

written as follows:

$$\forall\, i, j \;\in\; [0, k-1] \begin{cases} a_i \prec a_j, & \text{if } x_i < x_j \text{ OR, } y_i < y_j \text{ if } x_i = x_j; \\ a_i \succ a_j, & \text{otherwise.} \end{cases} \tag{3.1}$$

The application of this ordering criterion to the objects contained in an image subset gives rise to a permutation string $p$, which is actually the ordered sequence of indices corresponding to the above objects. In particular, string $p$ corresponds to the sequence of objects produced by projecting their positions along the $x$ axis and by taking them from left to right. Henceforth, string $p$ will be used to represent the original image subset itself. For example, the permutation string $p$ corresponding to the image subset (2 3 4 5) is (4 2 5 3). Table 3.1 lists the centers of mass of the objects contained in the example image of Figure 3.1.

A second permutation string $p^{'}$ is produced by projecting the positions of objects along the $y$ axis and by taking them from below to above. In particular, string $p^{'}$ is produced by ordering objects according to the following rule:

$$\forall\, i, j \;\in\; [0, k-1] \begin{cases} a_i \prec a_j, & \text{if } y_i < y_j \text{ OR, } x_i < x_j \text{ if } y_i = y_j; \\ a_i \succ a_j, & \text{otherwise.} \end{cases} \tag{3.2}$$

We now define a new entity called “rank”. The rank $r_i$ of object $p_i$ is defined as the number of objects preceding it in string $p^{'}$. The rank $r_i$ of object $p_i$ can be viewed as the number of objects which are below it. Given strings $p_i$ and $p^{'}$, $r_i$ is computed as

$$r_i = j \iff p_i = p'_j, \quad 0 \leq i, j < k. \tag{3.3}$$

Equation 3.3, a rank string $r$ is produced corresponding to the ordered sequence $p$. For example, the permutation string $p'$ corresponding to the image subset (2 3 4 5) is (3 5 2 4) and its rank string $r$ is (3 2 1 0).

The first ordering criterion guarantees that objects are ordered with respect to the $x$-axis. The $i$-th object from the left is object $p_i$. Moreover, the rank string $r$ is such that $r_i$ denotes the number of objects which are below the $i$-th object from the left. Therefore, the rank string $r$ completely characterizes the relative positions of the objects contained in image subset $p$. String $r$ defines a permutation over the set $\{0, 1, \ldots k-1\}$, since no two objects have equal ranks. Thus, string $r$ may take $k!$ different values. An ordered image subset $p$ is mapped to a unique address in an address space of size $k!$ by computing the rank (order) of $r$ with respect to a listing of permutations. For example, the rank corresponding to the ordered image subset (4 2 5 3) is 12. The ranking algorithm we used is that by Johnson and Trotter [108].

It has been assumed that no two objects have the same center of mass. The ordering of image subsets containing concentric objects (this may happen when one object contains another) is ambiguous and may result in different representations of subsets with similar spatial relationships (see also Section 3.7). In such cases, the ordering of objects can be based on more image properties in addition to the center of mass. For example, given two objects with the same center of mass, the first ordering criterion can be defined so that the object having the smaller size proceeds the second in the ordered sequence. If the objects have the same size, then a third property (e.g., the roundness), a fourth and so on, can be used to resolve ambiguities related to the order of objects. However, such situations are very rare.

### 3.4.2 Second Ordering Criterion

Let $(a_0, a_1, \ldots, a_{k-1})$, be an image subset of size $k$, where $k \in [2, \min(n, K_{max})]$. The image area is partitioned by an $M \times N$ rectangular grid, whose cells are indexed from 0 to $M \cdot N - 1$. Each object is assigned a rank or position $r_i$ equal to the index of the grid cell containing its center of mass $(x_i, y_i)$. In particular, the rank $r_i$ of object $a_i$ is computed as follows:

$$r_i = s_{x_i} \cdot N + s_{y_i}, \quad 0 \leq i < k, \tag{3.4}$$

where

$$s_{x_i} = \left\lfloor \frac{x_i}{X} \right\rfloor \cdot N; \quad s_{y_i} = \left\lfloor \frac{y_i}{Y} \right\rfloor \cdot M; \qquad 0 \leq i < k. \tag{3.5}$$

$X$ and $Y$ are the actual sizes (in pixels) of the rectangular grid along the horizontal and vertical dimension respectively, while $s_{x_i}$ and $s_{y_i}$ are the $x$ and $y$ coordinates of the position $r_i$ of object $a_i$ with respect to the $N \times M$ rectangular grid ($s_{x_i} \in [0, N-1]$ and $s_{y_i} \in [0, M-1]$). Figure 3.2 illustrates a 3 $\times$ 3 grid of size 256 $\times$ 256 (left). The same grid is then placed over the example image of Figure 3.1 (right). Table 3.2 lists the positions of the objects contained in the example image, computed with respect to the above grid.

Ordering is based on object positions. However, objects sharing the same grid position cannot be ordered unless a secondary criterion is used. We have chosen to base such a secondary criterion on object properties. In the definition given below, the size of objects is used as a secondary criterion: among two or more objects sharing the same position, ordering is based on size. In particular, for any two objects $a_i$, $a_j$, where $i, j \in [0, k-1]$, with positions $r_i$, $r_j$ respectively, the second ordering criterion is written as

$$\forall\, i, j \;\in\; [0, k-1] \begin{cases} a_i \prec a_j, & \text{if } r_i < r_j \text{ OR, } size(a_i) < size(a_j) \text{ if } r_i = r_j; \\ a_i \succ a_j, & \text{otherwise.} \end{cases} \tag{3.6}$$

If among objects sharing the same grid position there are objects of equal size, then the ordering is ambiguous and a third criterion has to be used (e.g., roundness). In general, the

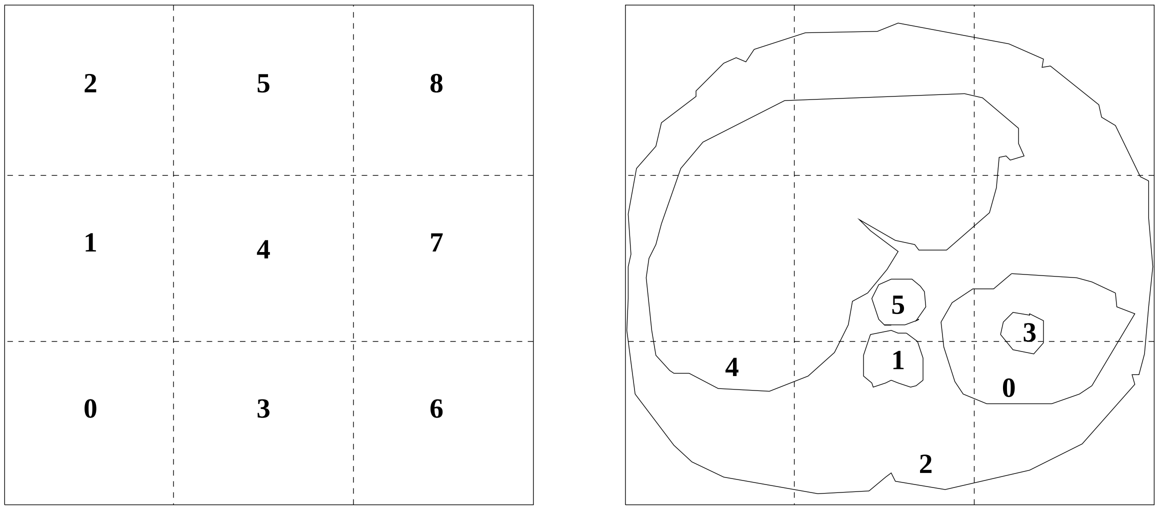


Figure 3.2: A $3 \times 3$ rectangular grid (left) placed over the example image of Figure 3.1 (right).

definition of the second ordering criterion can be extended to included any number of criteria (e.g., object roundness, orientation etc.) in addition to the position and the size of objects. However, ambiguities are reduced when grids are dense (e.g., $N, M \geq 3$) and the number of objects contained in image subsets is low (e.g., 2 - 6).

The application of the second ordering criterion to the objects contained in a given image subset gives rise to a permutation string $p$, which is the ordered sequence of object indices, and a string $r$ of positions (ranks) corresponding to the above ordered sequence of objects. For example, the permutation string $p$ corresponding to the image subset (2 3 4 5) is (5 4 2 3) and the rank string $r$ is (4 4 4 7). String $r$ represents the relative positions and the relative distance between objects in $p$. In particular, if the $x$, $y$ coordinates $(s_{x_i}, s_{y_i})$ of each object $p_i$ are used instead of $r_i$, the relative distance between any two objects in $p$ can be computed as follows:

$$d_{ij} = \sqrt{(s_{x_i} - s_{x_j})^2 + (s_{y_i} - s_{y_j})^2}, \quad 0 \leq i, j < k. \tag{3.7}$$

The number of different ways for placing $k$ objects on an $M \times N$ rectangular grid, is equal to the number of "l-part compositions of k", where $l = N \cdot M$. Therefore, the number

| $N = 3$, $M = 3$, $X = 256$, $Y = 256$ | | | |
|---|---|---|---|
| object | $s_{x_i}$ | $s_{y_i}$ | position: $r_i$ |
| $a_0 = 0$ | 2 | 1 | 7 |
| $a_1 = 1$ | 1 | 0 | 3 |
| $a_2 = 2$ | 1 | 1 | 4 |
| $a_3 = 3$ | 2 | 1 | 7 |
| $a_4 = 4$ | 1 | 1 | 4 |
| $a_5 = 5$ | 1 | 1 | 4 |

Table 3.2: Positions corresponding to the objects contained in the example image of Figure 3.1 computed with respect to a 3 $\times$ 3 rectangular grid of size $(X, Y) = (256, 256)$.

of different values a string $r$ may take is equal to $\binom{k+l-1}{l-1} = \binom{k+l-1}{k} = \frac{(k+l-1)!}{k! \cdot (l-1)!}$. An ordered image subset $p$ can be indexed in terms of the "left/right", "below/above" relationships, as well as in terms of the relative distances between the objects it contains, by computing the rank of string $r$ with respect to a listing of compositions [109]. For example, the index computed to the ordered image subset (5 4 2 3), which has rank string (4 4 4 7), is 244.

### 3.4.3 Comparison of the Two Ordering Criteria

Representations derived by the application of the first ordering criterion are both translation and scale invariant (i.e., images translated or scaled with respect to each other result in the same representation). Translation invariance is assured, since only relative positions are taken into account in determining the order of objects. Similarly, scale invariance is assured, since no distance criterion is used. Therefore, when the first ordering criterion is used, the property of distance cannot be used in queries. However, in certain situations, and in particular in cases where all images are at a fixed scale, distance is important in determining the positions of objects. This may be the case when objects which are close enough to each other or the one contains the other must be assigned the same position. In such cases, the second ordering criterion must be used.

Representations derived by the application of the second ordering criterion are neither scale nor translation invariant. Translation invariance can however, be easily achieved by a simple coordinate transformation: first, the minimum enclosing rectangle specified by the centers of mass of the objects contained in $p$ is computed. Let $(\alpha_x, \alpha_y)$ be the coordinates of its lower left corner. The position $r_i$ of each object $p_i$ is then computed according to Equation 3.4 by using, in place of the pair of coordinates $(s_{x_i}, s_{y_i})$, the pair $(s_{x_i} - \alpha_x, s_{y_i} - \alpha_y)$. Translated image subsets make use of a smaller part of the rectangular grid (i.e., the upper left and right positions of a grid are not used in most cases). Therefore, the actual size of the address space is less than that computed by compositions. In general, the second ordering criterion results in larger address spaces than the first ordering criterion.

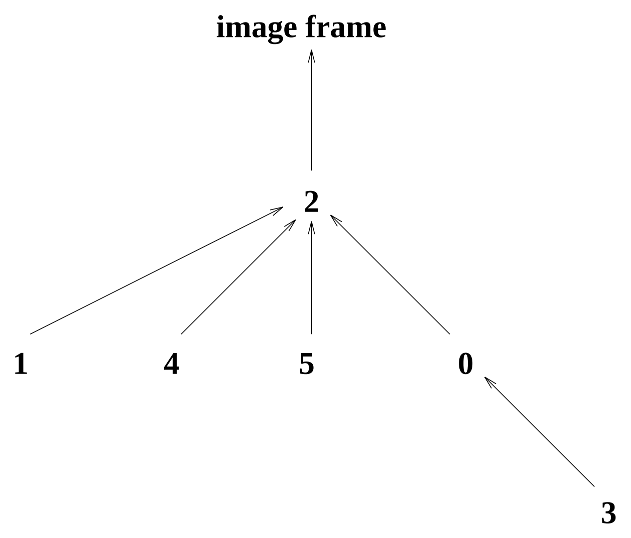
Both kinds of representations are rather sensitive to image rotations. Rotation invariance can be achieved only in cases where a reference direction can be identified (e.g., specified interactively by the user) so that, prior to any processing, images are rotated to a standard orientation.

## 3.5 Representation of the Inclusion Relationships

Given an ordered image subset $p = (p_0, p_1, \dots p_{k-1})$, an inclusion string $w = (w_0, w_1, \dots w_{k-1})$ representing the "inside/outside" relationships between objects is constructed as follows:

$$\forall\, i \in [0, k-1],\; w_i = \begin{cases} j & \text{if } p_i \text{ is both closer and inside } p_j, \quad 0 \le j < k; \\ i & \text{if } p_i \text{ is contained by the \textit{image frame} only.} \end{cases} \tag{3.8}$$

An object is contained by another, if all its contour points are contained by the polygonal contour corresponding to the second object. Algorithms for determining the relative position



**p[0] = 4, p[1] = 2, p[2] = 1, p[3] = 5, p[4] = 3, p[5] = 0**

**w[0] = 1, w[1] = 1, w[2] = 1, w[3] = 1, w[4] = 5, w[5] = 1**

Figure 3.3: Tree representation of string $w$ corresponding to the example image of Figure 3.1. String $p$ has been derived by applying the first ordering criterion.

between polygons can be found in [110]. Besides, the minimum distance between all pairs of objects contained in the image under consideration must be computed. In particular, for each pair of objects, all pairs of line segments are taken (one from each contour) and their minimum distance is compared against the minimum distance value computed so far. The computation of both the inclusion relationships and of the minimum distances between objects are time intensive operations. Such relationships are computed once for all pairs of objects in an image and not separately for each individual image subset.

A string $w$ depends on the ordered sequence $p$, which is turn depends on the ordering criterion applied. A string $w$ may be considered as one of the “k-base representations of k” elements. These are $k^k$. Therefore, an ordered subset $p$ can be indexed in terms of the inclusion relationships between objects it contains by computing the rank of $w$ with respect to a listing of the $k$-base representations of $k$ elements [111]. For example, the inclusion string $w$ corresponding to the ordered (according to the first criterion) subset (4 2 5 3) is (1 1

```
function valid(w, k)
    for $i = 0$ to $k - 1$ do
        $path = 0$;
        $j = w[i]$;
        while $j \neq w[j]$ do
            $path = path + 1$;
            if $path > k - 1$ then return(0);
            $j = w[j]$;
        end; while
    end; for i
    return(1);
end. valid
```

Figure 3.4: Function $valid$ determines whether $w$ can be an inclusion string corresponding to an image containing $k$ objects.

1 1) and has rank 85.

The inclusion relationships between objects can be represented by a tree data structure: each object corresponds to a node and the parent of each node is the node corresponding to the object which is both closer to it and contains it. For the objects whose inclusion relationships are represented by $w$, such a tree representation can be constructed as follows:

$$\forall\ i\ \in\ [0, k-1],\ \text{parent of } p_i = \begin{cases} p_{w_i} & \text{if } i \neq w_i; \\ \textit{image frame} & \text{otherwise.} \end{cases} \tag{3.9}$$

Figure 3.3 illustrates the tree representation corresponding to the example image of Figure 3.1, whose objects are ordered according to the first ordering criterion.

The actual size of the address space corresponding to inclusion strings $w$ of size $k$ is less than $k^k$. Spare addresses correspond to graph structures which are not trees (i.e., contain cycles). Two such strings and their corresponding graphs are shown in Figure 3.5. The actual

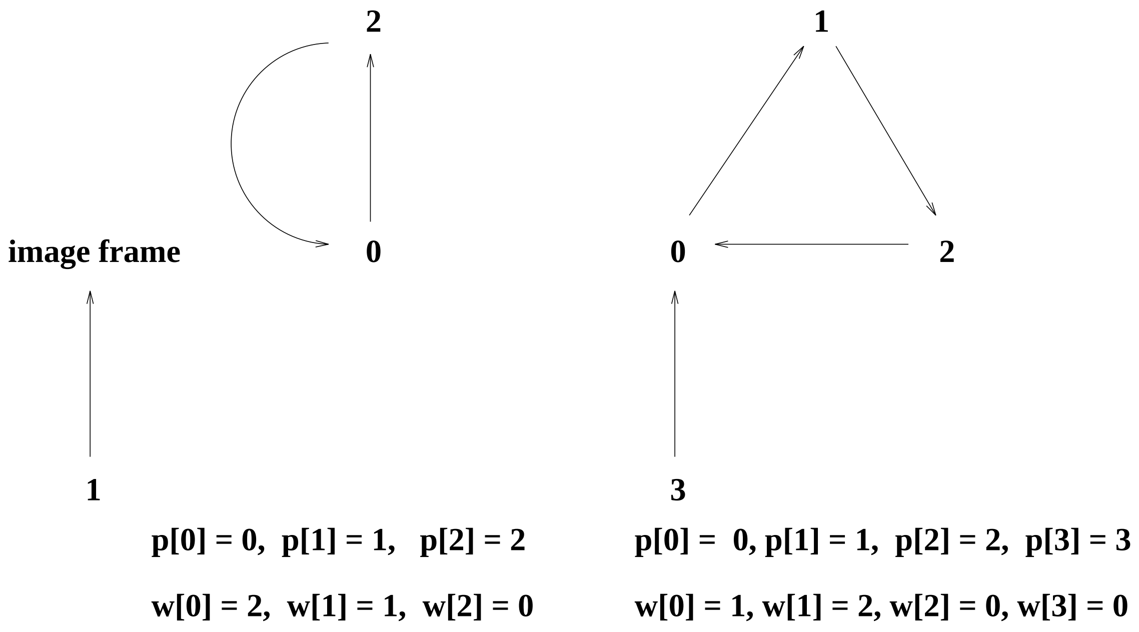

2
image frame
0
1
1
0
2
3
p[0] = 0, p[1] = 1, p[2] = 2
w[0] = 2, w[1] = 1, w[2] = 0
p[0] = 0, p[1] = 1, p[2] = 2, p[3] = 3
w[0] = 1, w[1] = 2, w[2] = 0, w[3] = 0

Figure 3.5: Examples of graph structures and $w$ strings not corresponding to inclusion relationships.

size of the address space corresponding to inclusion strings $w$ of size 2 is 3 ($3^2 = 4$), 16 for strings of size 3 ($3^3 = 27$), 125 for strings of size 4 ($4^4 = 256$), 1,296 for strings of size 5 ($5^5 = 3{,}125$) and 16,807 for strings of size 6 ($6^6 = 46{,}656$).

The function of Figure 3.4 examines whether the argument string $w$ can be an inclusion string corresponding to an image containing $k$ objects, in which case it returns 1; otherwise it returns 0. If $w$ is an inclusion string, then a path from any object to all its ancestors has length less than $k-1$, since an object may not have more than $k-1$ ancestors; otherwise a cycle has been found and the path never ends. In the worst case, the algorithm has square time complexity with respect to $k$: for each $w_i$, $i \in [0, k-1]$, a "while" loop which counts the length of the path from $p_{w_i}$ to its ancestors is executed at most $k-1$ times.

## 3.6 Representation of Object Properties

The description of individual objects is given in terms of properties corresponding to global object characteristics such as area, perimeter, roundness, orientation with respect to a reference direction, properties of classes etc. In general, the description of individual objects must consist of features that are shown to be effective in quickly narrowing down the search space for the purpose of image retrieval by content in a particular application domain. Given an ordered image subset $p$, each of the above properties gives rise to a separate attribute string $(u_0, u_1, \dots u_{k-1})$, where

$$\forall\, i \ \in \ [0, k-1],\ u_i = \left\lfloor \frac{\text{property value of } p_i}{\text{maximum property value}} \right\rfloor \cdot q. \tag{3.10}$$

If the property value of $p_i$ equals to the maximum property value, then $u_i = q - 1$. The "maximum property value" is usually different for different properties. For example, roundness has maximum value 1 (corresponding to a circle), orientation has maximum value $\pi = 3.141\ldots$, etc. In cases where the resulting representation has to be independent of scale (e.g., when the first ordering criterion is used), maximum size values (e.g., perimeter, area) are set equal to the size of the biggest object in the image. Otherwise, maximum size values can be set equal to a reference size (e.g., the size of the image). $q$ is an integer, called "quantization value", corresponding to the number of different values a property may take. In particular, an object is assigned an integer property value in the range $[0, q-1]$. For example, if $q = 3$ the property of orientation takes values 0, 1, 2 corresponding to objects having orientation (with respect to the horizontal direction), between 0 and $\frac{\pi}{3}$, $\frac{\pi}{3}$ and $\frac{2\pi}{3}$, $\frac{2\pi}{3}$ and $\pi$ respectively. The quantization of continuous property values may result in an information loss and properties with similar continuous values may be assigned different discrete values (see Section 3.7).

Properties such as roundness, orientation etc., are computed based on the polygonally approximated contours of objects. In particular, the orientation of an object is defined to be equal to the angle, with respect to the horizontal direction, of the axis of elongation.

This is the axis of least second mement. The roundness of an object is defined as the ratio of the smallest to the largest second moment [112]. Moreover, the area of an object may easily be computed from its polygonal contour $((x_0, y_0), (x_1, y_1), \ldots (x_{m-1}, y_{m-1}))$ as $\frac{1}{2}\sum_{i=0}^{m-1}(x_{i+1} \cdot y_i - x_i \cdot y_{i+1})$, where subscript calculations are modulo $m$ (number of contour points). Such computations have time complexity proportional to the number of contour points. The computation of all properties for all objects contained in an image is done once for all objects and not separately for each individual image subset.

Each attribute string of size $k$ is mapped to a unique address in an address space of size $q^k$ by computing its rank with respect to a listing of the "$q$-base representations of $k$" elements. Higher (lower) quantization values increase (decrease) the size of the address space exponentially. Higher (lower) quantization values increase (decrease) the accuracy of representations. The ordered image subset (4 2 5 3) has size (area) string (1 2 0 0) which has rank 7, roundness string (1 2 2 2) which has rank 79 and orientation string (0 0 0 2) which has rank 54. In all cases $q = 3$.

## 3.7 Stability of Image Representations

Image representations are unique; only subsets having similar properties result in the same attribute strings and are mapped to the same indices. However, in certain cases, representations are not tolerant to small variations of property values and become unstable; subsets with slightly different property values may be mapped to different attribute strings and, therefore, different indices.

Instabilities may occur when (a) objects are close enough to each other and the first ordering criterion is used (e.g., objects 0 and 3 of the image of Figure 3.1) or (b) they have centers of mass near the borders of a grid cell (e.g., object 3) and the second ordering criterion is used. A slight translation of such an object along the $x$ or the $y$ direction would change the ordered sequence of object indices. To deal with such situations, one must consider all possible orders of sequences containing objects which are very close to each other. For example, for the image subset consisting of objects 0, 3 and 4, two ordered subsets, (4 3 0) and (4 0 3), must be represented and stored separately. In addition, object 3 of the right image of Figure 3.2 must be considered as having both positions 6 and 7. An image subset containing object 3, e.g., (5 4 3), must then be represented and stored twice, once with rank string (4 4 6) and once with rank string (4 4 7).

Instabilities may also occur when objects have continuous property values approximately equal to the threshold values. For example, for the property of roundness and a quantization value $q = 3$, the threshold values are $0.33\ldots$ and $0.66\ldots$. Object 4 in Figure 3.2 has roundness $0.33$ and may take a discrete value of 0 or 1. The representation of image subsets containing object 4 is unstable. Such objects must be assigned two discrete property values and an image subset containing such an object must be represented and stored twice. For example, the image subset (4 1 5) must be represented and stored twice, once with roundness string (0 2 2) and once with roundness string (1 2 2).

Finally, instabilities may occur when objects are very rounded (e.g., objects 1, 2, 3 and 5 of the image of Figure 3.1). The orientation of a very rounded object is ambiguous and may take any discrete value in the range $[0, q-1]$. Each image subset containing a very rounded object must be taken and stored $q$ times. For example, for quantization value $q = 3$, the image subset (4 3 0) must be taken and stored three times, once with orientation string (0, 0, 0), once with orientation string (0, 1, 0) and once with orientation string (0, 2, 0).

## 3.8 Completeness of Image Representations

The proposed representations succeed in capturing image content in cases of images consisting of disjoint objects having simple shapes. However, such representations cannot be used when images contain occluded objects. Global object characteristics (e.g., size, roundness) change drastically and become unreliable if substantial parts of objects are hidden;

Similarly, such representations cannot capture information related to details of object shapes. However, in each of the above two cases, such representations can be used to index the images of an IDB. In particular, even if objects are partially visible, if object positions are identified correctly (e.q., specified by the user), indexing can still be based on relationships. In addition, even if objects have complex shapes, indexing can still be based on global object properties. Retrievals respond with a set of candidate images which can then be compared against the query on the basis of additional properties, either relational or specific to the shape of objects they contain, using techniques such as those proposed in [53, 64, 66].

Object positions play an important role in deriving reliable image representations and must be specified precisely. So far, it has been assumed that object positions are defined by their center of mass. In cases where the true center of mass cannot be obtained or falls outside the object contour, a point which is representative of the object position can be specified by an expert user.

# Chapter 4

# Image Indexing and Storage

## 4.1 Introduction

IMAGE DATA can be distinguished into “physical” and “logical” data [10]. Original (grey-level) images and segmented images, are physical images. Image related data (i.e., information extracted from images, attributes, text etc.) are logical images. Physical and logical images are stored separately in a physical and a logical database respectively. Pointers are implemented from the logical to the physical database.

The physical database has been implemented as a separate disk space storing the original image files together with their polygonal contours. To reduce storage requirements, physical images are compressed prior to storage and decompressed upon retrievals. No image file structure has been introduced. However, images will eventually be stored in clusters based on the likelihood of being retrieved together in response to a particular query (e.g., the set of all images corresponding to a patient’s exam may be stored close together on the disc).

The logical database stores computed representations of subsets derived from images stored in the physical database. The logical database consists of a set of data files, each storing subsets of equal size. A data file is divided into segments (data pages), each corresponding to an address. In turn, a data page stores image subsets mapping to the same address. Overflows are handled by creating linked lists of data pages.

## 4.2 Indexing Image Subsets

The logical database consists of a set $(H_2 \ldots, H_{K_{max}})$ of files, where $K_{max}$ is the maximum size of image subsets under consideration. The image subsets of size $k,\ 2 \leq k \leq K_{max}$, together with their representations are all stored in file $H_k$. Each image subset $p$ (which is the ordered sequence of object indices) is represented by a tuple of strings of the form $(r, w, \ldots)$ where, $r$ is its corresponding rank string representing the "left/right" and "below/above" relationships between objects, $w$ is the inclusion string, and the remaining strings correspond to properties of individual objects. Three such strings are used: $s$ to encode the size property, $c$ to encode the roundness property and $o$ to encode the orientation property of those objects whose indices are in $p$. Therefore, the representation of an image subset $p$ is given by a tuple of the form $(r, w, s, c, o)$.

The relational model and a relational database management system (DBMS) offer the most direct and easy way of implementing the logical database: each $H_k$ file is implemented as a relational table storing tuples of the form $(image, p, r, w, s, c, o)$, where "image" is a name or an index corresponding to the original image file from which $p$ has been derived. Indexing can be performed by creating a secondary index for each attribute string or for combinations of strings. Image subsets are then stored, indexed and accessed using mechanisms specific to the particular DBMS. However, such mechanisms may not be well suited for indexing and accessing image subsets, while usually, they are not even known to the end-user.

The problem of indexing image subsets can be viewed as a problem of multi-dimensional (multi-key) indexing, if each attribute string or the address computed to each attribute string is treated as a separate key-value. It can be reduced to a problem of single-key indexing by mapping the set of keys representing each image subset into a single key. Techniques for both single-key and multi-key indexing are known to exist (see Chapter 2). However, we have to be careful in using such techniques for the purpose of indexing image subsets. In particular, most of these techniques (e.g., B-trees, extendible hashing, linear hashing, etc.) are intended for organizing a file based on the primary key which must be unique; otherwise they may fail. This will happen as soon as subsequent insertions cause a data page storing records with same keys to split. Furthermore, they are dynamic, since they assume that the keys and the size of the address space are not known in advance.

Here, we are faced with a different situation. Both the keys and the size of the address space corresponding to attribute strings are known in advance. Furthermore, there may exist a very large number of image subsets with the same properties and, therefore, with the same keys. All image subsets with same properties must be stored together on disk, so that they can be retrieved together in response to a particular query. However, we may consider indexing as being performed on the set of keys used (which are unique), rather than on the image subsets themselves. Each key corresponds to a separate storage space where image subsets are stored. A dynamic indexing technique will keep the size of space for the storage of keys minimum. However, for a large database storing thousands of images it is more likely that all keys will be in use. In such a case, there is no need for a dynamic behavior. In addition, with the exception of hashing techniques, tree-indexing techniques will not result in optimum performance. For example, a B-tree technique requires $\mathcal{O}(\log N)$ page fetches to access a page of image subsets, where $N$ is the number of disc pages storing the keys.

There are techniques which may perform well in the case of image subsets. In particular, “multilists” and “inverted lists” are two commonly used file organizations supporting the indexing of records based on secondary keys. A separate index list, one for each secondary index, consisting of all stored records (e.g., image subsets) is created. Such an organization, will perform well for a request for all image subsets with a given attribute value, but a request based on more than one attributes will take much more time to answer. However, given a key, the set of keys will have to be searched in order to locate the corresponding list. Furthermore, such techniques will result in a large space overhead for the storage of index lists.

In order to satisfy the needs for efficient storage and access of image subsets, an addressing scheme is proposed and a file structure specific to the above addressing scheme is introduced. It is similar to a hash addressing file organization. An array of key-address pairs, which represents the association between keys and corresponding data spaces, is initially maintained and stored on disc.

## 4.3 Proposed Addressing Scheme

An image subset is considered to be the basic entity in the proposed indexing scheme. In particular, images are indexed based on representations of the set of all derived subsets. An image containing $n$ objects is decomposed into the set of ordered image subsets

$$\left\{ p_k^l \;\middle|\; 2 \leq k \leq \min(n, K_{max}), \quad 0 \leq l < \binom{n}{k} \right\}. \tag{4.1}$$

The representation of an image subset $p_k^l$ takes the form $(d_k^{l,0}, d_k^{l,1}, \ldots, d_k^{l,\nu-1})$, where $\nu$ is the number of attribute strings and $d_k^{l,i}, 0 \leq i < \nu$, is the address computed to the $i$-th attribute string. In particular, the $(r, w, s, c, o)$ representation of the $l$-th subset of size $k$ can be written as $(d_k^{l,0}, d_k^{l,1}, d_k^{l,2}, d_k^{l,3}, d_k^{l,4})$ or $(d_k^{l,r}, d_k^{l,w}, d_k^{l,s}, d_k^{l,c}, d_k^{l,o})$ (for clarity, the index $i$ has been substituted by the symbol of its corresponding attribute string). $p_k^l$ can then be mapped to a single address $I_k^l$: $(d_k^{l,0}, d_k^{l,1}, \ldots, d_k^{l,\nu-1})$ is considered to be the representation of $I_k^l$ in the "mixed radix system" $(D_k^0, D_k^1, \ldots, D_k^{\nu-1})$, where $D_k^i,\ 0 \leq i < \nu$, is the size of the address space corresponding to the $i$-th attribute string. The index $I_k^l$ is computed as follows:

$$I_k^l = d_k^{l,0} + d_k^{l,1} \cdot D_k^0 + d_k^{l,2} \cdot D_k^0 \cdot D_k^1 + \cdots + d_k^{l,\nu-1} \cdot \prod_{i=0}^{\nu-2} D_k^i, \tag{4.2}$$

$$2 \leq k \leq \min(n, K_{max}), \quad 0 \leq l < \binom{n}{k}.$$

Each $d_k^{l,i}$ satisfies $0 \leq d_k^{l,i} < D_k^i$ and $d_k^{l,\nu-1} \neq 0$, except for $I_k^l = 0$. The size of the address space specified by Equation 4.2 is equal to

$$D_k = \prod_{i=0}^{i=\nu-1} D_k^i, \quad 2 \leq k \leq K_{max}. \tag{4.3}$$

The size of the address space is equivalent to disk space (see section 4.5). $D_k$ becomes extremely large for relatively large values of $k$ and $\nu$ (e.g., for $k > 4$ and $\nu > 2$, $D_k$ takes values in the order of $10^9$). One way to deal with large address spaces is to allow collisions, in which cases, two or more image subsets with different representations are mapped to the same address. If $DIRSIZE$ is the maximum size of the address space allowed (e.g., $DIRSIZE = 10^6$), a mapping that allows collisions is

$$A_k^l = I_k^l \bmod DIRSIZE, \quad B_k^l = \frac{I_k^l}{DIRSIZE}, \tag{4.4}$$
$$2 \leq k \leq \min(n, K_{max}), \quad 0 \leq l < \binom{n}{k}.$$

An image subset $p_k^l$ is now characterized by the pair $(A_k^l, B_k^l)$. $A_k^l$ is a new address, called "primary index" and takes values in the range $[0, DIRSIZE - 1]$. $B_k^l$ is a second address, called "secondary index", which takes values in the range $\left[0, \left\lfloor \frac{D_k}{DIRSIZE} \right\rfloor\right]$ and is used to distinguish among image subsets with the same primary indices.

The size of the address space decreases when $\kappa < \nu$ attribute indices are used in Equation 4.2 to compute $I_k^l$. Such indices correspond to attribute strings and image properties which are called "primary attributes" and "primary properties" respectively. The $\nu - \kappa$ remaining attribute strings, which are not primary, and their corresponding image properties are called "secondary" and are used to compute a secondary index $C_k^l$ according to Equation 4.2. Therefore, an image subset $p_k^l$ is represented by a triple $(A_k^l, B_k^l, C_k^l)$ consisting of its corresponding primary and secondary indices.

## 4.4 Selection of Primary Attributes

The primary attributes must correspond to the most discriminant image properties (see Section 5.4). In the optimum case, primary indices have a uniform distribution and each index holds the same number of image subsets. The selection of primary attributes is based on measurements obtained from a prototype set of images which are considered to be characteristic of the application under consideration. In particular, for the $i$-th attribute string of a given image representation consisting of $\nu$ attributes and for the indices corresponding to all attribute strings of size $k \in [2, K_{max}]$, we compute the variance $\sigma_k^i$ as

$$\sigma_k^i = \frac{\sum_{j=0}^{D_k^i - 1} \left( N_{k,j}^i - \overline{N_k^i} \right)^2}{D_k^i - 1}, \quad 0 \leq i < \nu, \quad 2 \leq k \leq K_{max}, \tag{4.5}$$

where $D_k^i$ is the address space corresponding to image subsets of size $k$ and the $i$-th attribute string. $N_{k,j}^i$ is the number of attribute strings having index $j$. The sum $\sum_{j=0}^{D_k^i - 1} N_{k,j}^i$ equals to the total number of image subsets of size $k$. $\overline{N_k^i}$ is their mean value computed over $D_k^i$ as $\overline{N_k^i} = \frac{\sum_{j=0}^{D_k^i - 1} N_{k,j}^i}{D_k^i}$. The variance measures the actual amount of variation of a set of data and depends on the scale of measurement: the variance $\sigma_k^i$ is computed with respect to the address space $D_k^i$ which varies for different attribute strings of the same size $k$. Therefore, the variance is computed with respect to mean values which vary depending on the kind of the attribute under consideration. To compare the variation of several sets of data we use the "coefficient of variation" $CV_k^i$, which gives the variation as a percentage of the mean values $\overline{N_k^i}$ which in turn are normalized with respect to $D_k^i$. In particular, $CV_k^i$ is computed as

$$CV_k^i = \frac{\sqrt{\sigma_k^i}}{\overline{N_k^i}} \cdot 100, \quad 0 \leq i < \nu, \quad 2 \leq k \leq K_{max}. \tag{4.6}$$

The coefficient of variation $CV_k^i$ is computed for all attributes and for all string sizes $k \in [2, K_{max}]$. For a specific size $k$, the most discriminant attributes are those having the smallest coefficient of variation. Therefore, different primary properties may correspond to subsets of different size $k$. Based on the computed values of coefficient of variation,

| $k$ | $CV_k^r$ | $CV_k^w$ | $CV_k^c$ | $CV_k^s$ | $CV_k^o$ |
|---|---|---|---|---|---|
| 2 | 9.01 | 109.37 | 15.20 | 161.73 | 39.30 |
| 3 | 10.90 | 251.72 | 35.62 | 274.55 | 49.32 |
| 4 | 22.42 | 765.55 | 59.42 | 452.92 | 69.38 |
| 5 | 48.02 | 2,601.81 | 98.89 | 747.55 | 108.32 |
| 6 | 121.60 | 10,402.70 | 280.21 | 1,254.8 | 195.49 |

Table 4.1: Coefficient of variation computed for all attribute indices corresponding to image subsets of size $k \in [2, 6]$, derived from a set of medical images.

attribute strings and corresponding image properties can be ordered in terms of decreasing discriminative power. A number of attributes (e.g., 2 or 3), of the first in this ordered sequence, are then chosen to be the primary attributes. The specification of the number of primary attributes is another important problem which is discussed separately in Section 5.5.

In developing a prototype IDB which supports the indexing of images by content we used 226 computed tomographic (CT) and magnetic resonance (MR) images. We considered this set of images to be characteristic of the application domain under consideration and we used them for the selection of primary attributes. Table 4.1 shows the values of the coefficient of variation computed for five attribute strings, namely the $r$, $w$, $s$, $c$ and $o$, having size $k \in [2, 6]$ (for clarity, the index $i$ in $CV_k^i$ has been substituted by the symbol of its corresponding attribute string). We used the first ordering criterion and a quantization value of $q = 3$. In the computation of $CV_k^w$, the actual size of the address space $D_k^w$ has been taken into account (see Section 3.5). According to Table 4.1, $r$ is the most discriminating attribute followed by $c$, $o$ and $w$. In particular, the ordered sequence of attributes in order of decreasing discriminating power is ($r$, $c$, $o$, $s$, $w$).

Henceforth, the roundness property of objects represented by string $c$, together with the "left/right" and the "below/above" relationships of objects represented by string $r$, are

| $k$ | $D_k^r = k!$ | $D_k^c = q^k = 3^k$ | $D_k = D_k^r \cdot D_k^c$ |
|---|---:|---:|---:|
| 2 | 2 | 9 | 18 |
| 3 | 6 | 27 | 162 |
| 4 | 24 | 81 | 1,944 |
| 5 | 120 | 243 | 29,160 |
| 6 | 720 | 729 | 524,880 |

Table 4.2: Size of address space for values of $k$ in the range [2,6] corresponding to the attributes of position $r$ and roundness $r$. $D_k^r$ corresponds to the first ordering criterion. The quantization value $q$ is 3.

considered to be primary image properties and primary attributes respectively and are used to compute the primary indices. The remaining three attribute strings, namely the $o$, $s$ and $w$ corresponding to the properties of orientation, (relative) size and the inclusion relationships respectively are considered to be secondary and are used to compute the secondary indices. For example, the ordered image subset (4 2 5 3) derived from the image of Figure 3.1 is represented by the triple $(1{,}908,\ 0,\ 562{,}066)$ of primary and secondary indices.

The sizes of the address spaces corresponding to primary attribute strings $r$ and $c$ and image subsets of size $k \in [2, 6]$ are shown in Table 4.2. The address space $D_k^r = k!$ corresponds to the first ordering criterion. The address spaces corresponding to low values of $k$ are very small. In particular, subsets of size $k = 2$ are distributed over 18 addresses. However, indexing for low values of $k$ (e.g., $k \leq 4$) can be based on more primary attributes (e.g., the $c$, $r$ and $o$) than indexing for higher values of $k$ (e.g., $k > 4$). Address spaces become much larger for greater values of the quantization value. For example, for $q = 5$ and $k = 5$, $D_k = 5! \cdot 5^5 = 375{,}000$.

## 4.5 Proposed File Structure

All image subsets of size $k$, $k \in [2, K_{max}]$, are stored in the $H_k$ file. A file $H_k$ consists of a set of "data pages" of fixed capacity. In particular, for each subset $p_k^l$ having index $A_k^l$, a tuple $(image, p_k^l, B_k^l, C_k^l)$ is stored, where "image" is the name of the image from which $p_k^l$ has been derived and acts as a pointer to the original image file stored in the physical database. Each page stores image subsets with the same primary index. A pointer is kept representing the association between page addresses on disk and primary indices. In fact, an array of such pointers is maintained, called "directory". The size of a directory is computed according to Equation 4.3. A directory entry may also contain a second address field pointing to the next free position within a page.

An attempt to insert an image subset in an already full page causes an "overflow". Overflows are handled by chaining: a new page is allocated (in which the image subset is stored) and linked with the overflowed page. Each page contains an address field pointing to its successor page in the list. New pages are always inserted at the beginning of the list so that nonfull pages are always directly available for writing: there is no need to search the whole list of pages in order to locate a nonempty page. Before any data are stored, an "empty" directory (i.e., a directory having all its pointers set to null) is maintained for each $H_k$ file. Figure 4.1 illustrates a file structure with the above characteristics.

The file structure described so far consists of two separate files: one storing the directory and one storing the actual data pages. During processing, the directory is loaded in main memory unless it is too large. In such a case, it is divided in smaller pieces (segments). Only one segment at a time (the one containing the most recently acquired page address) is kept in main memory. Directory segments are returned back to the disk when processing is ended or when a required address falls outside the current directory segment. The size of segments can be specified by the user.

The advantage of this organization is that it provides very fast direct access on the basis of

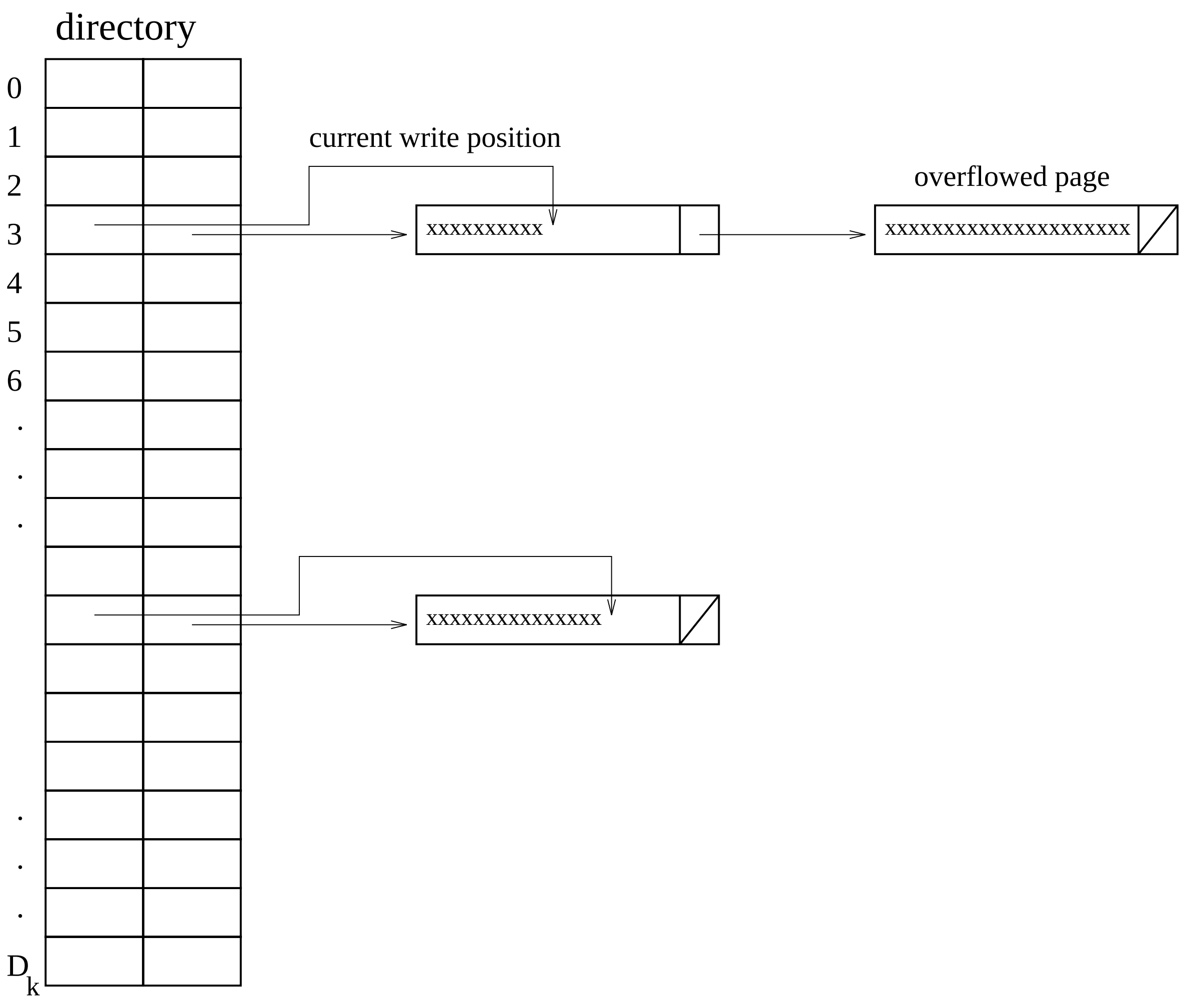


Figure 4.1: File structure $H_k$ with directory of size $D_k$. Unused directory pointers are null.

a given address. It has the disadvantage that updates (i.e., modifications of stored data) and deletions are difficult to be performed. However, insertions and retrievals are the two more common operations that take place in an IDB; updates and deletions are rather rare. So far, we have only studied insertions and retrievals.

Insertions are performed in two disc access: one to access a directory (if stored on disk) and read the address corresponding to the first page in the list of data pages, and one to locate the next free position within this page. Similarly, retrievals require at least two disk accesses: one to access a directory and one to fetch the first data page. Successive disc accesses may return the whole list of data pages. However, as page chains grow, the number of disk accesses increase and retrievals are slowed down.

When a data entry has to be updated or deleted, corresponding pages have to be located and fetched in main memory. The data entry can then be updated or removed. The page has to be reconstructed by closing the gaps created by deletions before it is returned back on disk. Successive deletions may cause pages to become empty. Eventually, data files may contain holes corresponding to empty pages. If pages are not reused by future insertions, the whole file will have to be rebuilt.

## 4.6 Storage Requirements

The storage space required by each $H_k$ file structure is

$$S_{N,k} = b_1 \cdot \sum_{i=1}^{N} \binom{n_i}{k} + b_2 \cdot \min(D_k, DIRISIZE), \quad 2 \leq k \leq \min(n_i, K_{max}). \tag{4.7}$$

Therefore, the storage space for the IDB as a whole is

$$S_{N,K_{max}} = \sum_{k=2}^{K_{max}} S_{N,k}. \tag{4.8}$$

The first term in Equation 4.7 represents the space occupied by the actual image data (image subsets). It is dynamic, since it increases with the number of images stored in the IDB. $N$ is the number of images stored, $n_i$ is the number of objects contained in the $i$-th image and $b_1$ is the size of information stored per image subset ($b_1 = 15$ bytes on the average). The second term of Equation 4.7 represents the space required to store the directories. It is static, since it never changes no matter how large or small is the number of images stored. $b_2$ is the amount of space required to implement the two directory pointers (usually $b_2 = 8$ bytes).

The number of image subsets produced, and thus the amount of data stored, can be very large especially when the number of objects contained in images is large (e.g., more than

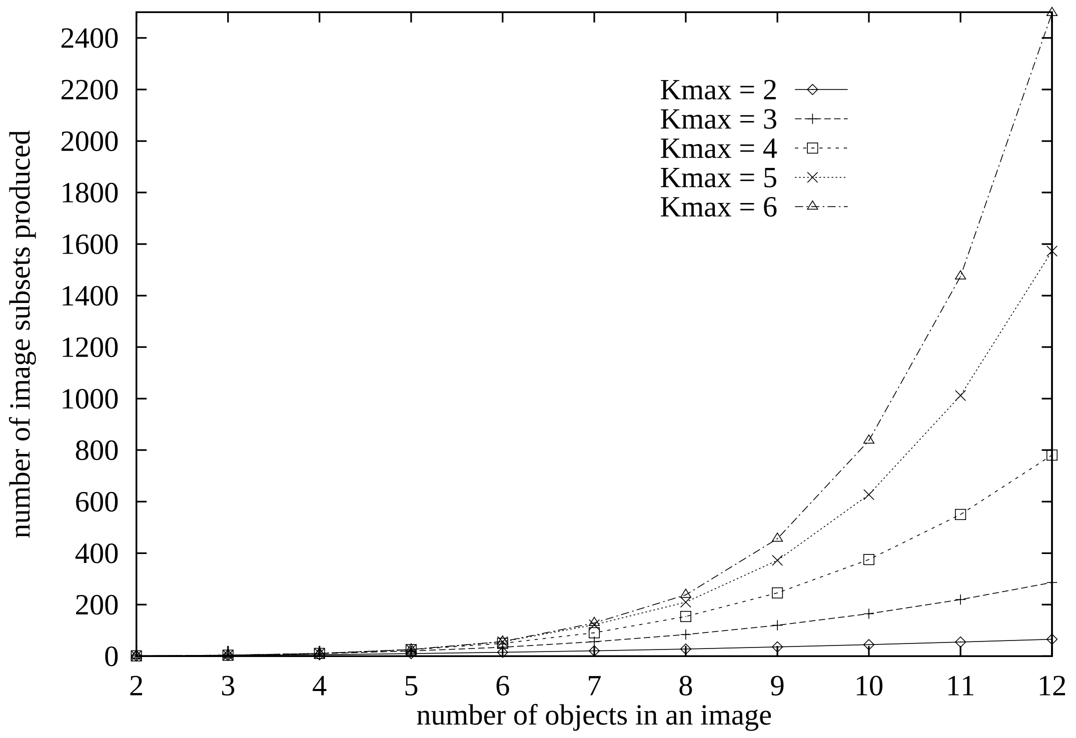


Figure 4.2: Number of image subsets produced from an image, as a function of the number of objects it contains, for (a) $K_{max} = 2$, (b) $K_{max} = 3$, (c) $K_{max} = 4$, (d) $K_{max} = 5$ and (e) $K_{max} = 6$.

10). It can become even larger when images are noisy and segmentations are inexact. To avoid such difficulties and to keep the amount of data manageable, segmentations need to be carried out interactively so that insignificant segments and segments resulting from noise are deleted. In this case and for the kinds of images found in most applications, the average number of objects stored per image can be kept small (typically less than 10).

The number of image subsets stored per image, and thus the amount of space required, increases also with $K_{max}$. Figure 4.2 shows the number of image subsets produced from an image as a function of the number of objects it contains, for (a) $K_{max} = 2$, (b) $K_{max} = 3$, (c) $K_{max} = 4$, (d) $K_{max} = 5$ and (e) $K_{max} = 6$. If we consider $b_1 = 15$ bytes and $K_{max} = 6$, then images containing up to 10 objects require less than $800 \cdot 15 = 12{,}000$ bytes for storage. Images containing more than 10 objects require much more space for storage.

For the prototype IDB consisting of 226 medical images and for $K_{max} = 6$, there are $16{,}404$ stored image subsets. Among them, there are $2{,}557$ subsets of size 2, $3{,}917$ subsets of size 3, $4{,}286$ subsets of size 4, $3{,}480$ subsets of size 5 and $2{,}163$ subsets of size 6. Assuming $b_1 = 15$ bytes on the average, the average required amount of storage space per stored image subset is $1{,}088$ bytes (the amount of storage space corresponding to the directories has not been taken into account).

# Chapter 5

# Image Retrieval

## 5.1 Introduction

ALL QUERIES address the logical database rather than the raw image data stored in the physical database. We concentrate our attention on the case of queries by image example: a query image or a sketch of image segments is given, it is analyzed and a representation similar to those of the images stored in the IDB is created. Sketches corresponding to all images matching the query are retrieved and displayed. The user can then specify which (original) images are going to be retrieved from the physical database using an "image browser" (see Appendix B). Representations of image subsets of equal size having the same property strings with those of the query are also retrieved and displayed.

## 5.2 Image Similarity Criteria

Let $p_m$ be a image consisting of $m$ objects and let $(d_m^0, d_m^1, \ldots, d_m^{\nu-1})$ be its representation corresponding to a set of $\nu$ properties. Furthermore, let $p_k$ be an image subset obtained from an image consisting of $n$ objects and let $(d_k^0, d_k^1, \ldots, d_k^{\nu-1})$ be its representation. The image

$p_m$ is "similar" to the image subset $p_k$ or $p_m$ "matches" $p_k$, in which case we write $p_m \sim p_k$, if the following condition holds:

$$p_m \;\sim\; p_k \iff k = m \;\wedge\; d_m^i = d_k^i, \quad \forall\, i \;\in\; [0, \nu - 1]. \tag{5.1}$$

Equivalently, an image is similar to an image subset if they have both equal size and same property strings, or they have the same representation of primary and secondary indices (see Section 4.3). If $p_m$ matches $p_k$, then the $i$-th ($i \in [0, m-1]$) object of $p_m$ matches the $i$-th object of $p_k$. Furthermore, an image $p_m$ matches an other image consisting of $n$ objects if, $m \leq n$ and $p_m$ matches at least one of the image subsets of size $m$ generated from the image under consideration. If $p_m$ is a query image, its corresponding answer set contains all image subsets which are similar to it.

## 5.3 Search Strategy

Query processing depends on $m$, the number of objects contained in the query image. In particular, we distinguish between "direct access" queries corresponding to $2 \leq m \leq K_{max}$ and "indirect access" queries corresponding to $m > K_{max}$, where $K_{max}$ is the maximum size of image subsets stored. The search is performed in two steps, namely "hypothesis generation" and "hypothesis verification". During the first step, a number of candidate image subsets (or images) matching the query are retrieved. During the second step, the final answer set containing image subsets (or images) matching the query is constructed.

### 5.3.1 Direct Access Queries: $2 \leq m \leq K_{max}$

- **Hypothesis generation:** the primary and secondary indices of the query image are computed first (see Section 4.3). The query addresses the $H_m$ file and all image subsets with the same primary index are retrieved.

- **Hypothesis verification:** all hypothesized image subsets are matched (one by one) against the query with respect to the secondary indices.

### 5.3.2 Indirect Access Queries: $m > K_{max}$

- **Hypothesis generation:** given a query image specifying $m$ objects, all subsets consisting of $K_{max}$ objects are generated, thus creating $\rho = \binom{m}{K_{max}}$ new queries. Each of these queries performs as a direct access query on the $H_{K_{max}}$ file and produces an answer set consisting of names or indices corresponding to images which are similar to it. Therefore, $\rho$ answer sets namely $S_0, S_1, \ldots S_{\rho-1}$, are produced. Their intersection $S = S_0 \cap S_1 \ldots \cap S_{\rho-1}$ is then obtained and used to hypothesize the existence of images matching the original query.

- **Hypothesis verification:** the images contained in the set $S$ are retrieved[1], all image subsets corresponding to retrieved images are generated, their representations are computed, and they are matched against a similar computed representation of the original query image.

Retrieved images may also be compared on the basis of additional properties, either relational or specific to the shape of objects they contain, using techniques such as those reviewed in Section 2.7.

## 5.4 Performance Evaluation

Response time is one possible measure of the performance of retrievals. However, response time depends on characteristics of the particular implementation and of the hardware used (e.g., how the logical and the physical databases are implemented, how quick is the

[1]Instead of original images, their segmented forms are retrieved.

access to the secondary storage, the size of the main memory etc.). Response time depends on the size of the hypothesized answer set and increases with it. The size of the hypothesized answer set is independent of characteristics of the implementation and can be used as a second measure of the performance of retrievals in addition to response time. It is computed as the percentage of images (or image subsets) retrieved with respect to the total number of images (or image subsets of the same size) stored.

The size of the hypothesized answer set returned in response to queries specifying $k$ objects, is always inverse proportional to the size of the address space $D_k$. In particular, as $D_k$ increases, the stored image subsets are distributed over a larger space and the number of image subsets stored per index decreases. In general, the evaluation of the performance of retrievals must be based on a mathematical formal model of the distribution of the stored image subsets, which in addition to $D_K$, takes into account the content of image subsets stored (which in turn depends on the content of images of the application domain under consideration), query content (which effects the value of the primary index) and the interdependences between image subsets derived from the same image (two or more image subsets may differ by one object). However, such a mathematical formal model cannot be easily derived.

In the special case where all indices appear with the same probability, the size of the hypothesized answer set is equal to $\frac{1}{D_k}$. We attempt to achieve a uniform distribution by selecting as primary, those attributes which distribute the stored image subsets uniformly over their corresponding address space (see Section 4.4). If the distribution is uniform, we can ignore the dependences of performance on the application domain, query content (the performance is fairly the same regardless of query content) and the interdependences between image subsets derived from the same image. In studying the performance of retrievals, independence of query content is achieved by taking the average performance of a large number of queries (i.e., more than 10). Henceforth, in order to keep the analysis of the performance tractable, we consider performance as being only a function of the size of the address space $D_k$. Specifically:

1. $D_k$ depends on the size $k$ of image subsets stored and increases with it. Therefore, the performance of retrievals improves with the number of query objects, since queries address files storing image subsets of equal size. In particular, the performance of indirect access queries depends on the performance of direct access queries specifying $K_{max}$ objects. Therefore, the performance of indirect access queries improves with $K_{max}$.

2. $D_k$ depends on the number of primary attributes used and increases with it. Therefore, the performance of retrievals improves with the number of primary attributes. If the number of primary attributes is equal to the total number of attributes of an image representation, then the size of the hypothesized answer set becomes equal to the size of the final answer set. However, the size of the address space corresponds to disc space and the number of primary attributes is limited (see Section 5.5).

3. $D_k$ depends on the quantization value and increases with it. Greater quantization values not only increase the accuracy of image representations, but also improve the performance of retrievals.

4. $D_k$ depends on the ordering criterion which has been applied. In particular, the size of the address space corresponding to the first ordering criterion depends only on the size of image subsets. The size of the address space corresponding to the second ordering criterion also depends on the size of the rectangular grid which has been used and increases with it. In general, the second ordering criterion results in larger address spaces and achieves better performance.

### 5.4.1 Experimental Results

Evaluations have been carried out using a prototype IDB consisting of $1{,}000$ simulated images each containing between 2 and 10 objects. For each $n \in [2, 10]$, there are over than 100 images containing $n$ objects. The number of stored image subsets depends on $K_{max}$.

For $K_{max} = 6$, there are 18,459 stored subsets of size 2, 36,856 stored subsets of size 3, 51,439 subsets of size 4, 51,274 subsets of size 5 and 36,528 subsets of size 6. All images are of size $256 \times 256$ and they are not scaled with respect to each other. The second ordering criterion has been applied using a rectangular grid of size $3 \times 3$. The quantization value $q$ is set to 3 for all object properties.

Queries are distinguished based on the number $m$ of objects they specify. Measurements of both the size of the answer sets and of the retrieval response times have been obtained. To obtain average performance measures, for each value of $m$ ranging from 2 to 6, 20 image queries have been applied and the average performance to queries specifying an equal number of objects has been computed. By applying the methodology of Section 4.4 we found that the ordered sequence of attributes in order of decreasing discriminating power is $(r, c, o, s, w)$.

Query response times account for the time spent in computing the query representation plus the time spent in searching the database and retrieving image data. The later, characterizes the performance of the search mechanism which has been applied and, henceforth, will be used in performance evaluations. The IDB has been implemented on a magnetic disc connected to a host computer (SUN 4/280). Therefore, the time delays due to data transfers through a communications network are zero.

**Performance of Direct Access Queries**

First we study direct access queries and we set $K_{max} = 6$. Figure 5.1 shows the average size of the answer sets obtained, as a percentage of image subsets retrieved, plotted against the number of query objects. The relative positions $r$ and the roundness $c$ of objects have been used as primary attributes. For queries specifying 2 objects, the size of the hypothesized answer set (i.e., percentage of image subsets matching the query with respect to $r$ and $c$) is 5% on the average. Queries become more specific and the size of an answer set decreases as the number of query objects increases. In particular, the size of an answer set drops to 0% for

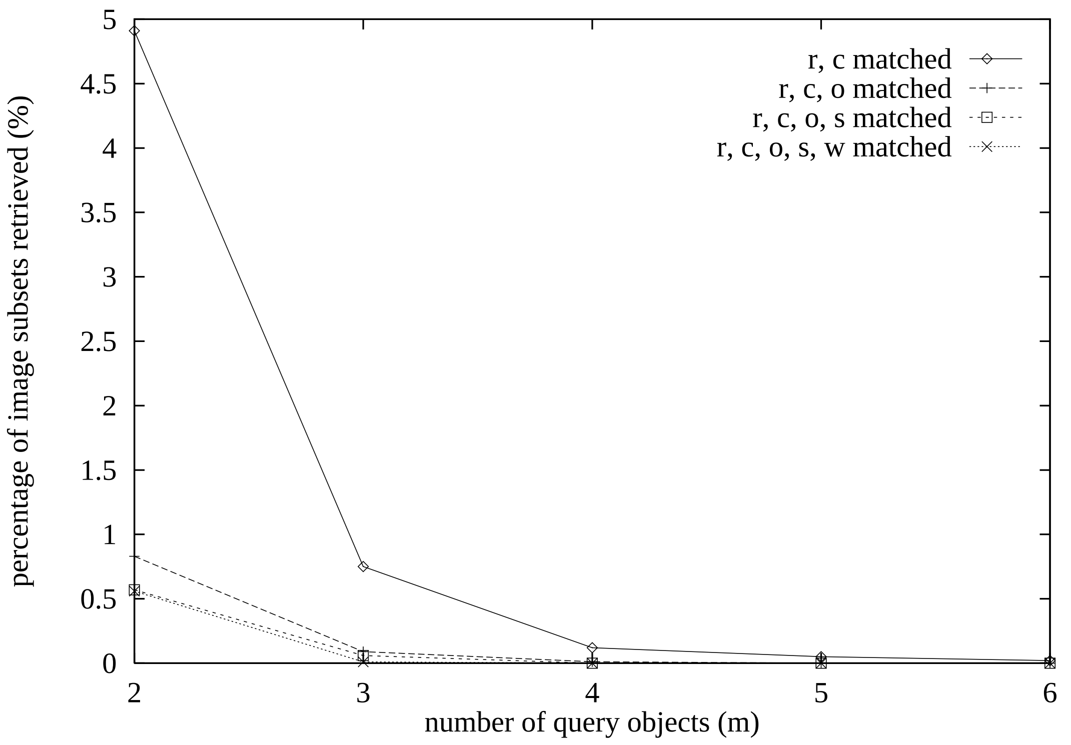


Figure 5.1: Average size of answer set corresponding to direct access queries, as a percentage of image subsets retrieved, plotted against the number of query objects.

queries specifying more than 5 objects. The same hold in the case of Figure 5.2 which shows the average size of the answer sets obtained, as a percentage of images retrieved, plotted against the number of query objects.

Similarly, queries become more specific and the size of an answer set decreases when comparisons between hypothesized image subsets (or images) and queries are based on more attributes. Attributes are used in comparisons in order of decreasing discriminative power. As shown in Figure 5.1 and Figure 5.2, comparisons have been performed based, first on the position $r$ and the roundness $c$ properties of objects (in which case the hypothesized answer sets are obtained), then on the orientation $o$, then on the size $s$ and, finally, on the inclusion $w$ properties of objects (in which case the final answer sets are obtained). Comparisons with

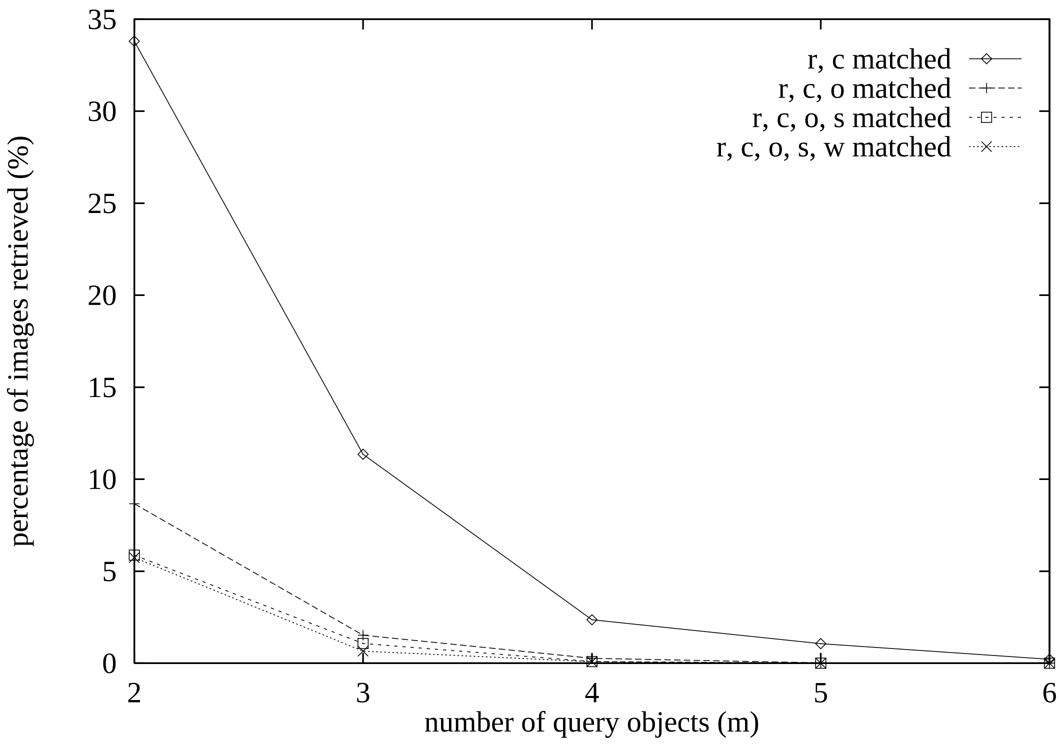


Figure 5.2: Average size of answer set corresponding to direct access queries, as a percentage of images retrieved, plotted against the number of query objects.

$w$ have little effect on the sizes of answer sets: either images do not differ with respect to the inclusion properties of objects or the attributes used before $w$ are enough to minimize the search space, or both. Therefore, comparisons with $w$ are not necessary for the set of images under consideration.

The response time accounts for the time spent in retrieving the hypothesized answer set plus the time spend in processing the retrieved image data. All retrieved image subsets are compared against the query on the basis of their corresponding secondary indices. The processing takes place in the main memory and is very fast. Therefore, response time accounts mainly to the time spend for retrievals. Figure 5.3 shows the retrieval response times as a function of the number of query objects. Response times decrease with the number

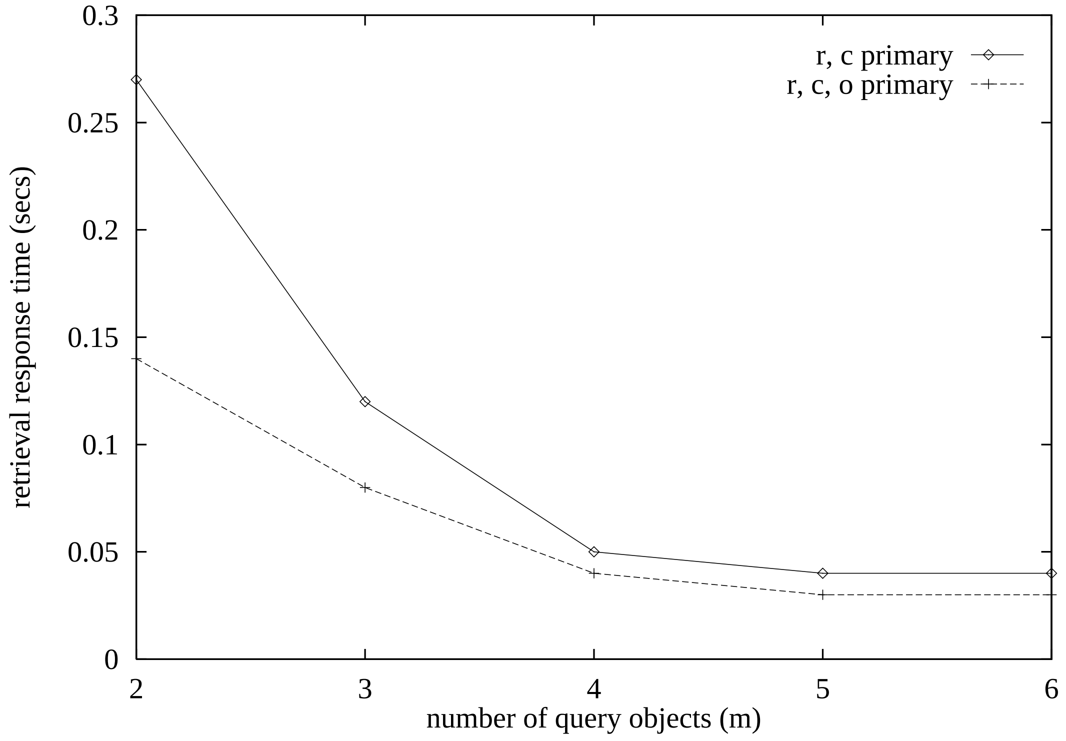


Figure 5.3: Average retrieval response times corresponding to direct access queries, plotted against the number of query objects.

of query objects, since the size of the answer sets obtained and thus the amounts of data to be processed, decrease too. Similarly, query responses are faster when more attributes are used as primary. The response times corresponding to (a) primary attributes $r$ and $c$, and (b) primary attributes $r$, $c$ and $o$ are demonstrated. If instead of two, three primary attributes are used, the response time is not significantly reduced and becomes minimum for queries specifying more than 3 objects.

As the number of primary attributes increases, directory sizes also increase and stored data (i.e., image subsets) are distributed over larger address spaces. Therefore, the amounts of data which are retrieved and processed decrease. However, the amounts of data which are retrieved are rather small and cannot influence response times significantly, especially in the case of large directories corresponding to files storing subsets consisting of more than 4 objects. For example, for primary attributes $r$ and $c$ and for queries specifying 4 objects, the size of the hypothesized answer set is approximately $0.1\%$ on the average. Only $0.1 \cdot 51{,}439/100 \approx 52$ image subsets are retrieved. If the amount of space per stored subset is 15 bytes, then approximately 770 bytes are retrieved. Such amounts of data can be read in one disc access. Increasing the size of the directory, will result decreasing the amount of retrieved data. However, one disc access is still necessary and the speedup is minimum.

**Performance of Indirect Access Queries**

An image query specifying $m > K_{max}$ objects, is decomposed into $\rho = \binom{m}{K_{max}}$ query subsets. Each query subset performs as a direct access query on the $H_{K_{max}}$ file and produces an answer set $S_i$, $0 \leq i < \rho$, consisting of names or indices corresponding to images which are similar to it. The size of each $S_i$ is, on the average, equal to the average size of the final answer set obtained to direct access queries specifying $K_{max}$ objects. The hypothesized answer set $S$ is computed as $S = S_0 \cap S_1 \ldots \cap S_{\rho-1}$. The size of $S$ is, on the average, less than the size of each $S_i$ but, greater than the average size of the final answer set corresponding to direct access queries specifying $m$ objects (i.e., $S$ may contain images which do not match the original query). Let $F_{images}(m)$, $m \in [2, K_{max}]$, be the size of the answer set of direct access queries specifying $m$ objects. $S$ is such that $F_{images}(m) \leq sizeof(S) \leq F_{images}(K_{max})$.

The size of $S$ for $K_{max} = 2$ and $K_{max} = 3$ is shown in Figure 5.4. The size of $F_{images}(m)$ for $m \in [2, 6]$ ($K_{max} = 6$), is also shown and corresponds to the curve of Figure 5.2 with all attributes matched. The size of $S$ decreases with $K_{max}$: the evidence that an image in $S$ matches the original query is stronger for higher values of $K_{max}$. Furthermore, the size of $S$ approaches to $F_{images}(m)$ as the number of query objects and the number of properties used in image comparisons increases. For example, the size of $S$ becomes approximately equal to $F_{images}(m)$ for $K_{max} \geq 4$ and $m > 4$. So far, we have been using 5 properties, namely the $r$, $c$, $o$, $s$ and $w$, which have been proven effective in quickly narrowing down the search

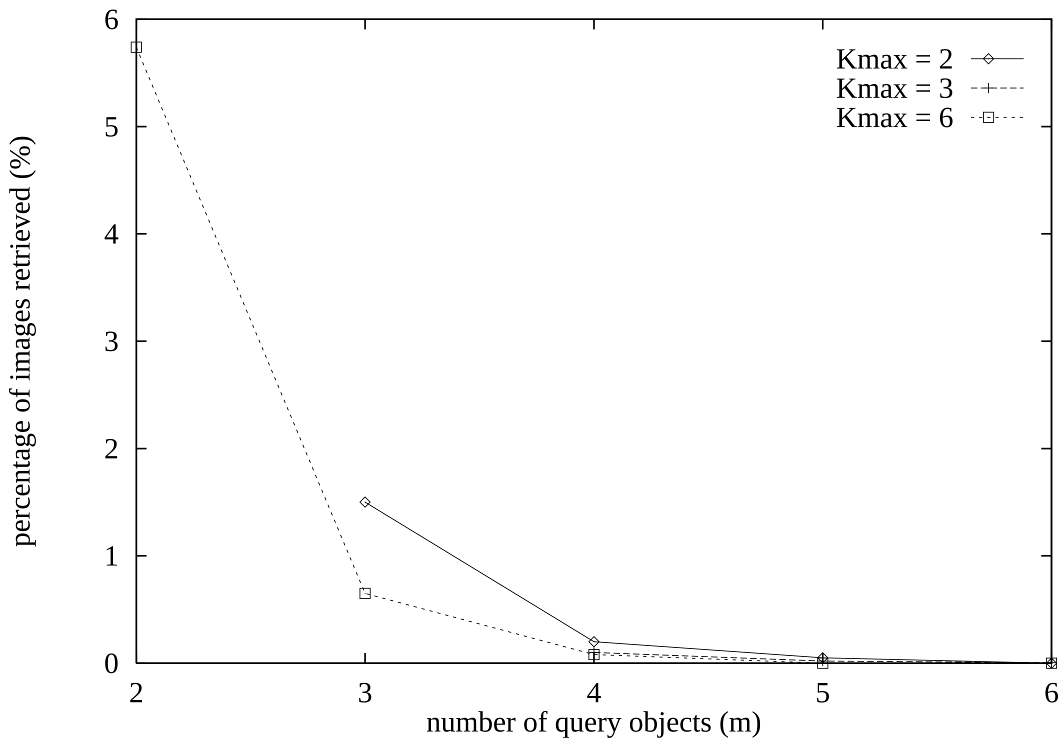


Figure 5.4: Average size of hypothesized answer set corresponding to indirect access queries, as a percentage of images retrieved, plotted against the number of query objects.

space in the case of indirect access queries.

All images contained in $S$ are then retrieved and matched against the original query (i.e., all image subsets of size $m$ obtained from images in $S$ are produced, their representations are computed and matched against a similar computed representation of the original query). By comparison with direct access queries, indirect access queries incur an overhead which is due to: (a) the retrieval and processing of intermediate query results and (b) the retrieval of the images contained in $S$, the processing and the matching of these images against the original query. As $m$ increases, the number $\rho$ of intermediate queries specifying $K_{max}$ objects increases too, thus resulting in an additional overhead. Therefore, indirect access queries respond slower than direct access queries specifying the same number of objects.

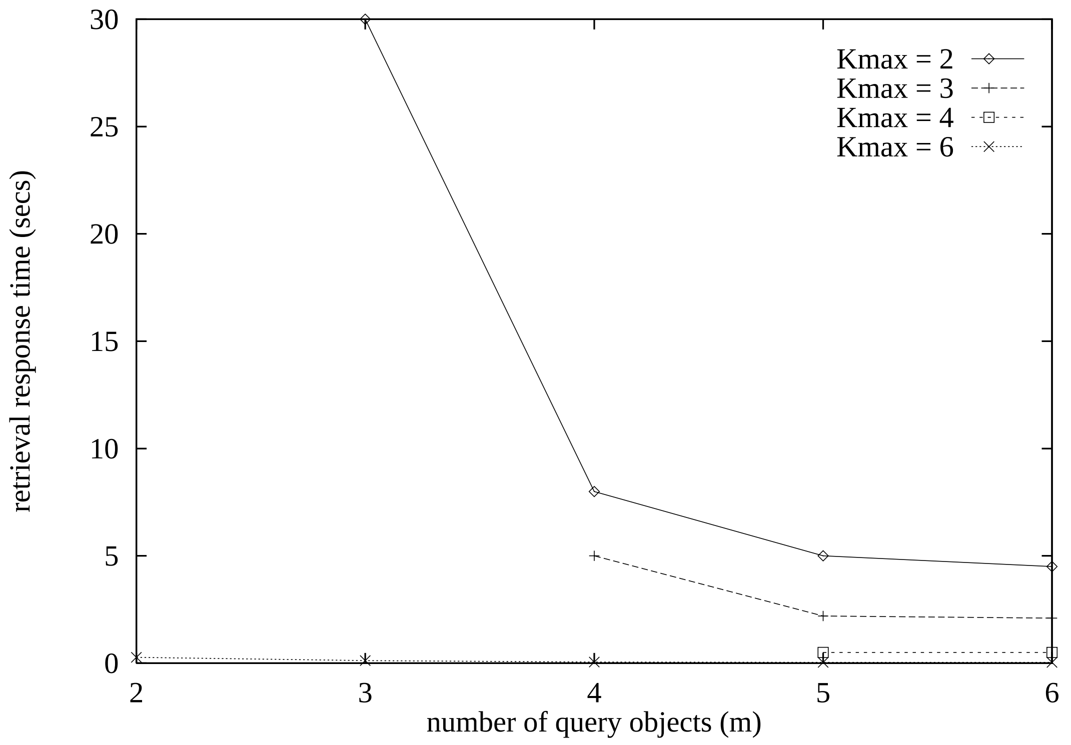


Figure 5.5: Average retrieval response times corresponding to indirect access queries, plotted against the number of query objects.

The retrieval response times corresponding to indirect access queries for (a) $K_{max} = 2$, (b) $K_{max} = 3$ and (c) $K_{max} = 4$ are shown in Figure 5.5. The response times corresponding to direct access queries ($K_{max} = 6$) are also shown and correspond to the curve of Figure 5.3 with $r$ and $c$ as primary attributes. The response times for $K_{max} \geq 4$ ($m \geq 5$) approach the response times of direct access queries. For such queries, $S$ contains only a few images (or none) and the overhead is minimum. When $K_{max} = 2$, as proposed in [11], the overhead is maximum.

## 5.5 Tailoring Parameters to an Application

Two parameters must be specified before an IDB is put to work: the number of primary attributes and the maximum size of stored image subsets $K_{max}$. It has been shown that the performance of retrievals improves both with the number of primary attributes and with $K_{max}$. However, the amount of storage space increases with both of these parameters. The specification of these parameters depends on the application domain, the content of images involved, and the number of images stored. The IDB of simulated images has been used for the specification of both the number of primary attributes and $K_{max}$.

### 5.5.1 Specification of the Number of Primary Attributes

According to Figure 5.3, increasing the size of directories by using more primary attributes, speeds-up retrievals in cases of queries specifying 2 or 3 objects. In this case, we may choose 3 primary attributes. In cases of queries specifying more than 3 objects, the speedup is minimum. Such queries address directories which are large enough even when 2 primary attributes are used. In this case, we may choose 2 primary attributes. For larger IDBs, the selection of the number of primary attributes can be performed by scaling the retrieval response times accordingly. However, the speedup may not be proportional to the increase of the size of the directories. According to Figure 5.3, the time responses to queries specifying 2 objects is approximately $0.25$ seconds on the average when 2 primary attributes (i.e., the $r$ and $s$) are used and $0.15$ seconds when 3 primary attributes (i.e., the $r$, $c$ and $s$) are used. The directories have sizes 18 and 162 respectively. When 3 primary attributes are used, the directory is 9 times larger, while time responses are only 1.6 times faster.

### 5.5.2 Specification of $K_{max}$

A threshold must be defined representing the maximum allowable size of $S$. This threshold value must always be greater than the average size of the final answer set corresponding to direct access queries specifying the same number of objects. Such a threshold may also be defined for the retrieval response times. In the following, we require that $K_{max}$ is such that the performance of indirect access queries becomes approximately equal to the performance of corresponding direct access queries.

Based on Figure 5.5 we must choose $K_{max} = 4$ or $K_{max} = 5$. The performance of indirect access queries in terms of the retrieval response times, and therefore the specification of $K_{max}$ based on it, depends mainly on the implementation (i.e., how fast an image is retrieved, how fast its representation is computed, how the intermediate query results are processed etc.). So far, the processing of indirect access queries, the retrieval and the computation processes has not been optimized. Based on Figure 5.4 we may choose $K_{max} = 3$. For $K_{max} = 3$ the size of the hypothesized answer set $S$ becomes approximately equal to the size of the final answer set corresponding to direct access queries specifying the same number of objects. Therefore, there is space to reduce $K_{max}$ from 5 down to 3 by optimizing the processing of indirect access queries.

The performance of indirect access queries depends also on the number of objects contained in images and on the number of properties involved in an image representation. In particular, for images consisting of more than 10 objects on the average, the number of image subsets which are derived and stored for a given value of $K_{max}$ will be larger (see Section 4.6). In this case, the size of the hypothesized answer set $S$ will be larger too. The same will happen when images are compared on the basis of a smaller number of properties (e.g., 1 to 3). In such cases, $K_{max}$ may take greater values.

## 5.6 Indexed Search Versus Search by 2-D Strings

The performance of the proposed methodology is compared to the performance of 2-D strings (see Appendix A). The comparison has been based on retrieval response times obtained to queries addressing the IDB of simulated images. A database of 2-D strings obtained from all images has been created as well. However, using 2-D strings search is exhaustive: all stored 2-D strings are compared (one by one) to the 2-D string representation

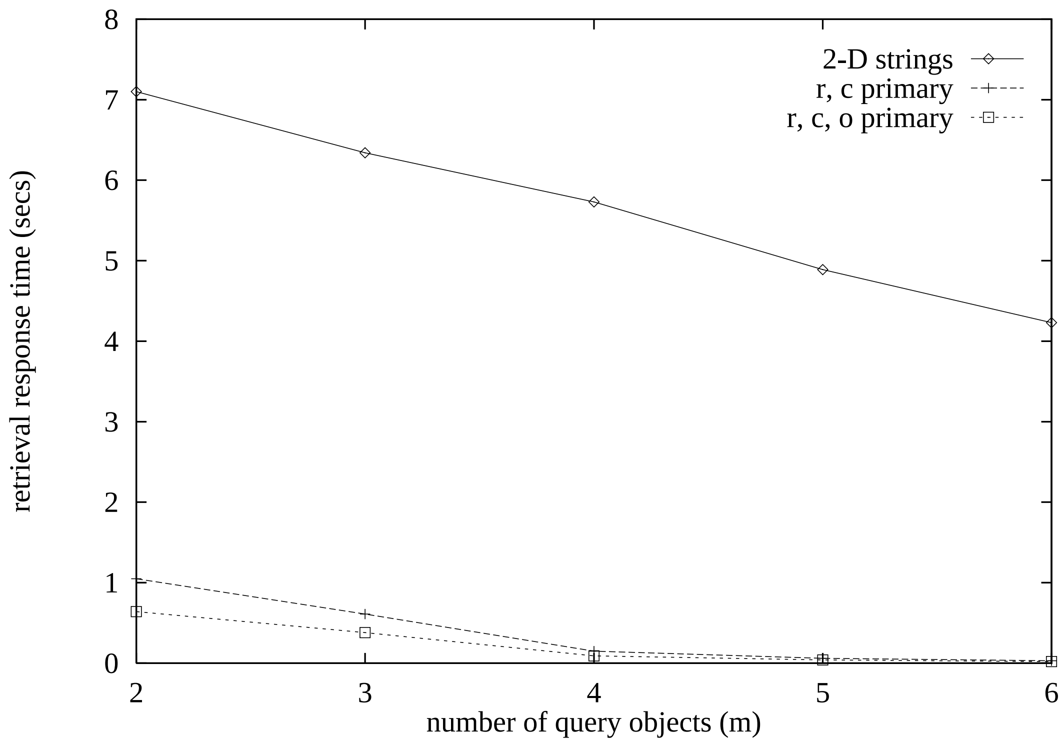


Figure 5.6: Average retrieval response times as a function of the number of query objects corresponding to the first ordering criterion and (a) indexed search, (b) exhaustive search using 2-D strings and $match2D$ algorithm.

of a query. The 2-D string representation has been extended to take into account the inclusion relationships between objects (i.e., the 2-D string representation includes string $w$), and more than one object properties (i.e., the 2-D string representation includes strings $c$, $o$ and $s$).

The algorithms for 2-D string matching make use of two strings $r^{2-D}$ and $s^{2-D}$ which represent the ranks of objects along the $x$ and $y$ axes respectively. When the first ordering criterion is used, $r^{2-D} = (0, 1, 2, \ldots n-1)$ and $s^{2-D} = r$, where $n$ is the number of objects in an image. In this case, two or more objects may not share the same position, while images are not compared on the basis of the relative distance between objects. The relative positions between all pairs of objects, given by the difference of their corresponding rank values along

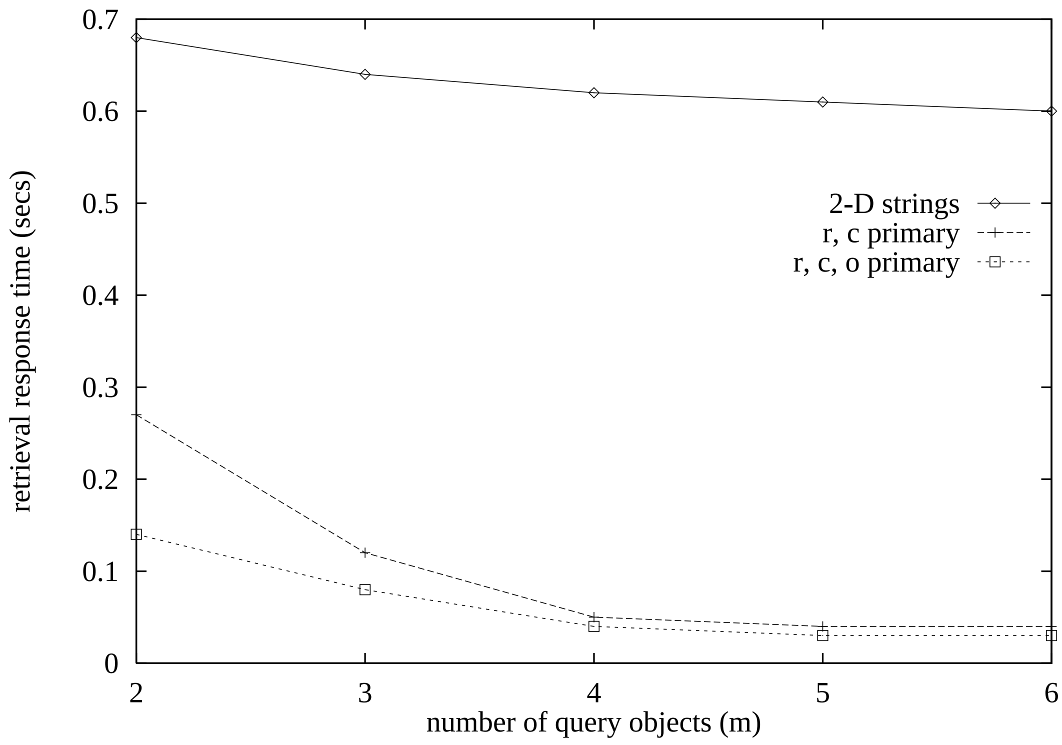


Figure 5.7: Average retrieval response times as a function of the number of query objects corresponding to the second ordering criterion and (a) indexed search, (b) exhaustive search using 2-D strings and $match2D2$ algorithm.

both the $x$ and the $y$ axes are compared. Searching is performed using type-0 matching and the algorithm $match2D$ of Figure A.3 of Appendix A. An algorithm for type-1 2-D string matching can be used as well. Figure 5.6 shows the retrieval response times corresponding to both indexed search and 2-D strings.

When the second ordering criterion is used, two or more objects may share the same position, while objects are also compared on the basis of their relative distance. The representation of the relative distance between any two objects is inherent within string $r$ (see Section 3.4.2). The relative positions between all pairs of objects and their relative distances, given by the difference of their corresponding rank values along both the $x$ and the $y$ axes are compared. Searching is performed using type-2 matching and the algorithm $match2D2$ of Figure A.5 of Appendix A. The rank strings $r^{2-D}$ and $s^{2-D}$ corresponding to the positions of objects along the $x$ and the $y$ axes respectively are constructed from $r$ as follows:

$$r_i^{2-D} = \left\lfloor \frac{r_i}{N} \right\rfloor, \quad s_i^{2-D} = r_i \text{ mod } M, \quad 0 \leq i < n. \tag{5.2}$$

The time complexity of both $match2D$ and $match2D2$ algorithms depends on the number of objects contained in images. However, as the number of objects in queries increases, queries become more specific and the initial matching lists (i.e., $MI$ and $match$ respectively) contain fewer objects. Therefore, time responses become faster with the number of query objects. In addition, $match2D$ makes use of termination conditions which are satisfied faster as queries become more specific. The algorithm $match2D2$ results in faster time responses than $match2D$, since $match2D2$ has lower time complexity than $match2D$. The time responses of direct access queries become faster if, instead of two (i.e., the $r$ and $c$), three primary attributes (e.g., the $r$, $c$ and $o$) are used.

The proposed methodology results in faster retrieval response times than 2-D strings in the case of direct access queries. In the case of indirect access queries, 2-D strings result in faster retrieval response times when $K_{max} \leq 5$ (see Figure 5.5). 2-D strings outperform the proposed methodology when $K_{max} \leq 4$. Both techniques work by finding all sequences of objects matching a given query image and produce identical answer sets. However, compared to the proposed methodology, 2-D strings have the advantage of requiring much less space for data storage.

## 5.7 Indexing of 2-D Strings

We can combine both indexed search and search by 2-D strings. Instead of representations of image subsets, 2-D strings corresponding to whole images are stored in data pages. In particular, for each image subset, a tuple $(image,\ 2-D\ string)$ is stored, where “image” is a unique identifier corresponding to the image from which the image subset has been derived, and "2-D string" is its 2-D string representation. Alternatively, to avoid storing the 2-D string representation of the same image many times, a tuple consisting only of the image identifier is stored, while its 2-D string is stored separately and indexed using its identifier as a unique key.

The search strategy must then be modified as follows: the index of a given query image is computed and all corresponding 2-D strings are retrieved. The retrieved 2-D strings are then compared (one by one) to the 2-D string representation of the query and all image subsets (i.e., matching subsequences) which are similar to it are produced. The size of the search, as a percentage of images retrieved, is shown in Figure 5.2 and corresponds to the curve with $r$ and $c$ matched. For example, for queries specifying 2 objects and primary attributes $r$ and $c$, approximately 35% of the total number of images are retrieved and compared against the query.

Direct access queries will become slower, since instead of simply comparing values of secondary indices corresponding to query and hypothesized image subsets respectively, their 2-D string representations are compared using either $match2D$ or $match2D2$ algorithm. Direct access queries will be faster than those of Figure 5.7, but slower than those of Figure 5.3. Indirect access queries will be faster than those of Figure 5.5, since the representations of the hypothesized images are found in the retrieved data pages and need not be computed from raw data (i.e., from the segment forms corresponding to hypothesized images).

A stored tuple may also include information which is better suited for the representation of the content of images of a particular application domain. For example, in applications where images contain overlapping objects with complex shapes, a stored tuple may include the 2-D C string representation [33] corresponding to the image from which the image subset at hand has been derived. However, retrievals will be slowed-down, since image comparisons based on 2-D C strings involve graph mathing techniques [76].

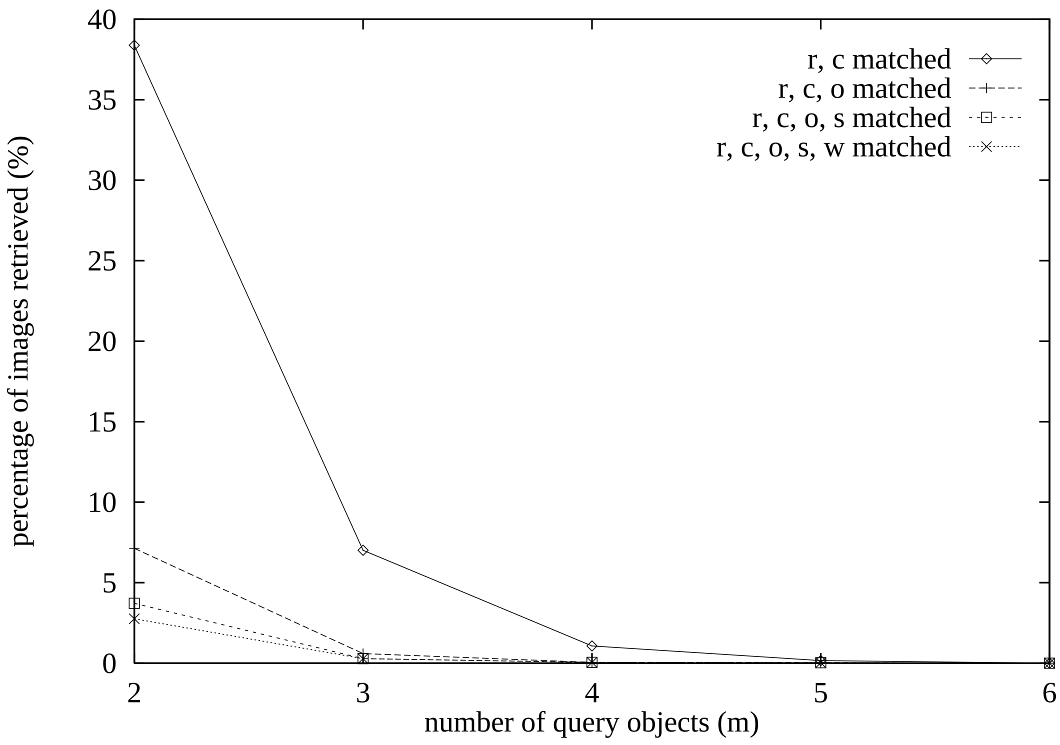


Figure 5.8: Medical IDB: average size of answer set, as a percentage of images retrieved, plotted against the number of query objects.

## 5.8 Retrievals on a Medical IDB

The proposed methodology has also been used for the retrieval of images in an IDB consisting of 226 computed tomographic (CT) and magnetic resonance (MR) images. These are images of various parts of the body (e.g., head, abdomen etc.), each consisting of one or more objects surrounded by an outer contour (e.g., skull). All images contain 2 to 8 objects. The images have been obtained at various scales. To obtain scale independent representations the first ordering criterion has been applied. The size values of objects are normalized with respect to the size of the biggest object in the image. The quantization parameter $q$ was set to 3 for all object properties.

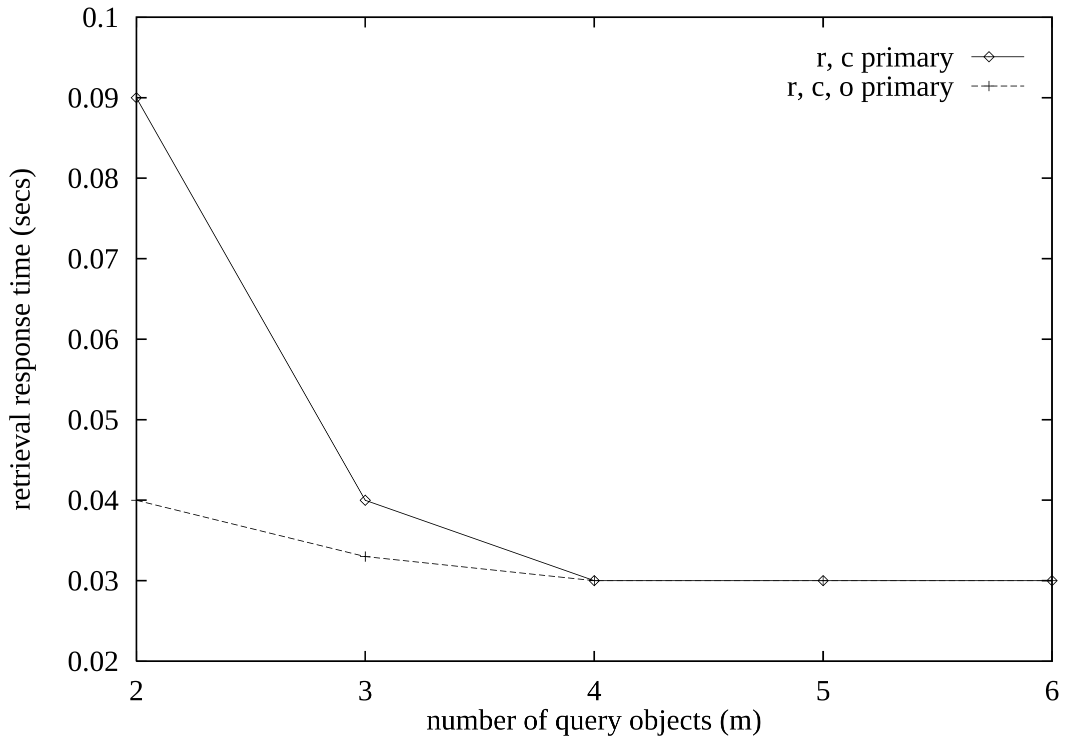


Figure 5.9: Medical IDB: average retrieval response times plotted against the number of query objects.

Measurements of both the size of the answer sets and of the retrieval response times have been obtained and shown in Figures 5.8 and 5.9 respectively. To achieve average performance measures, for each value of $m$ ranging from 2 to 6, 16 of the most common types of image queries (similar to the ones described in Section 5.8.1) have been applied and the average performance to queries specifying an equal number of objects has been computed. As expected, both the size the answer sets and of the retrieval response times decrease with the number of query objects. The size of the final answer set corresponding to queries specifying more than 3 objects is approximately 0%. Due to the small size of the IDB, retrievals are very fast. The overhead encountered in cases of indirect access queries results in retrievals which are, on the average, much slower than those shown in Figure 5.9.

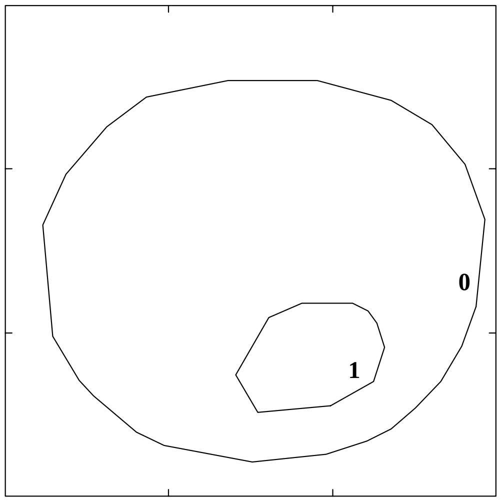
However, based on Figure 5.8, we may choose $K_{max} = 4$, since the size of the answer set corresponding to 4 objects is zero.

For $K_{max} = 4$, there are $10{,}760$ image subsets stored. Assuming $b_1 = 15$ bytes, the storage space per image is 714 bytes on the average. The primary attributes correspond to the relative positions and the roundness properties of objects (see Section 4.4). The time response to queries specifying less than 4 objects can become faster, when more than 2 primary attributes are used. However, the amounts of data which are retrieved and processed in response to queries specifying more than 4 objects are very small and retrievals cannot be made more efficient.

### 5.8.1 Examples of Retrievals

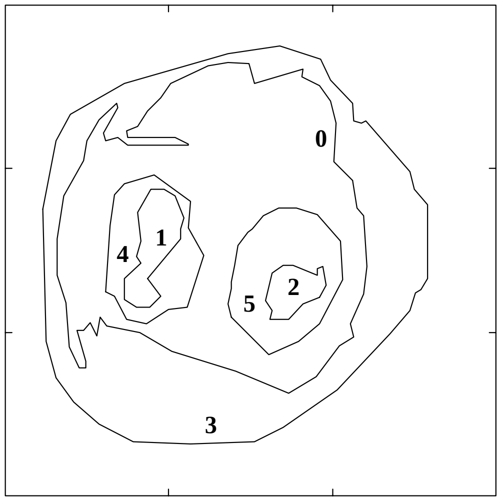
In the following, four characteristic examples of retrievals are presented and discused. In Figure 5.10, query object 0 matches object 3 of the retrieved image and query object 1 matches object 2. Query and retrieved objects match each other with respect to their global characteristics (i.e., relative size, roundness and orientation) and have similar relative positions. However, they have rather different shapes, since objects have not been compared on the basis of local properties of their shape. Object 5 of the retrieved image matches query object 1 with respect to the properties of size and roundness but not with respect to orientation (object 5 has orientation 1, while query object 1 has orientation 0). Therefore, the answer does not include object 5.

In Figure 5.11 the image subsets matching the query are (3 2 6) and (3 5 6). Query object 0 matches both objects 2 and 5. Query object 1 is bigger than query object 2. However, query object 1 matches object 3 of the retrieved image which is smaller than object 6 which matches query object 2. All these objects have size 0 with respect to the size to their outer object. To make similarity of sizes more accurate, quantization parameter $q$ must take a value greater than 3.



Query Image:

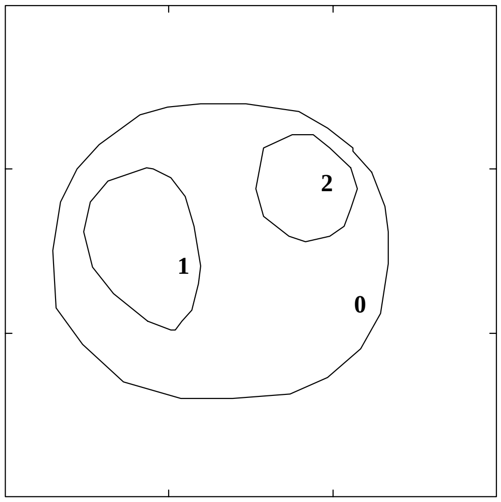
| | | | |
|---|---|---|---|
| p | = | 0 | 1 |
| r | = | 1 | 0 |
| c | = | 2 | 1 |
| s | = | 2 | 0 |
| o | = | 0 | 0 |
| w | = | 0 | 0 |



Retrieved Image:
Matching Subset: 3 2

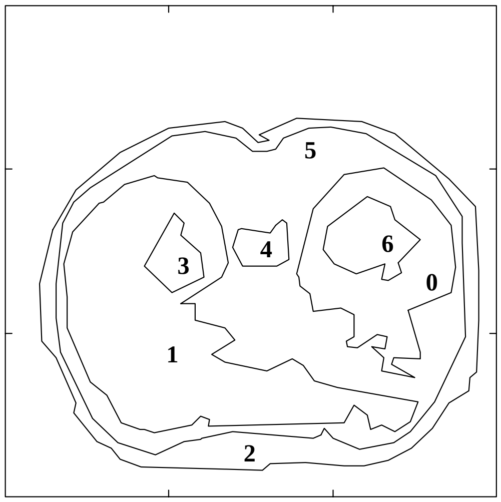
Figure 5.10: Retrieval example 1.

The query of Figure 5.12 looks similar to the query of the previous example. However, object 1 is below object 0 (i.e., the center of mass of object 1 is below the center of mass of object 0), while in the previous example, object 1 was above object 0. The answer set is now completely different. In this case, we are faced with a problem of stability of representation (see Section 3.7). We can avoid this problem by applying the same query twice, once with rank string (0 1 2) and once with rank string (1 0 2). When the second rank string is used, the query has exactly the same representation with the query of Figure 5.11. Then, the image



Query Image:

|   |   |   |   |   |
|---|---|---|---|---|
| p | = | 1 | 0 | 2 |
| r | = | 1 | 0 | 2 |
| c | = | 1 | 2 | 2 |
| s | = | 0 | 2 | 0 |
| o | = | 1 | 0 | 2 |
| w | = | 1 | 1 | 1 |



Retrieved Image:

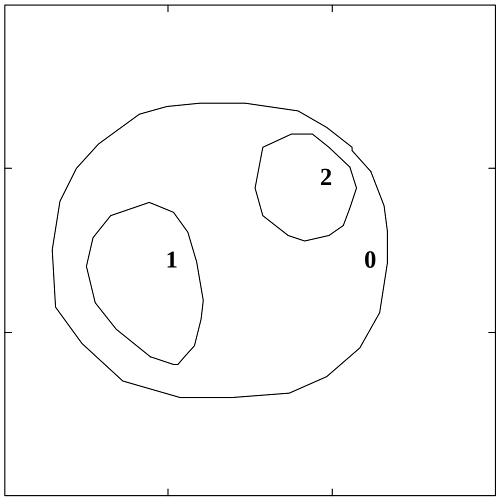
Matching Subset: 3 2 6

Matching Subset: 3 5 6

Figure 5.11: Retrieval example 2.

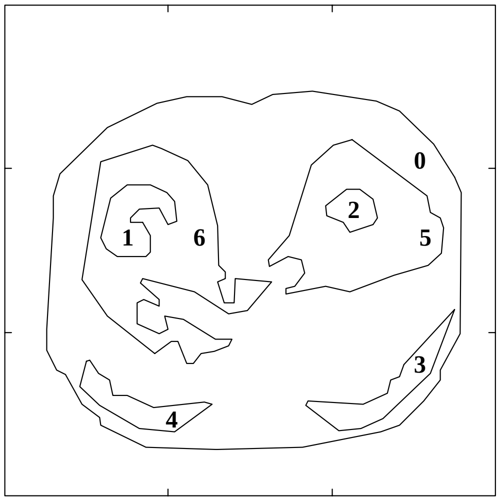
subset (6 0 2) will be included in the answer set. If the second ordering criterion were used, both query objects 0 and 1 would have assigned equal ranks. However, the second ordering criterion can be used only in cases where images are not scaled with respect to each other.

The query of Figure 5.13 returns an empty answer set. The two images shown below the query image do not satisfy query criteria. However, there are cases in which we would like the left image to be included in the answer set, since both image subsets (7 2 5 0) and (7 2 5 1) match the query, except for object 7 which does not have the same orientation with



Query Image:

| | | | | |
|---|---|---|---|---|
| p | = | 1 | 0 | 2 |
| r | = | 0 | 1 | 2 |
| c | = | 1 | 2 | 2 |
| s | = | 0 | 2 | 0 |
| o | = | 1 | 0 | 2 |
| w | = | 1 | 1 | 1 |



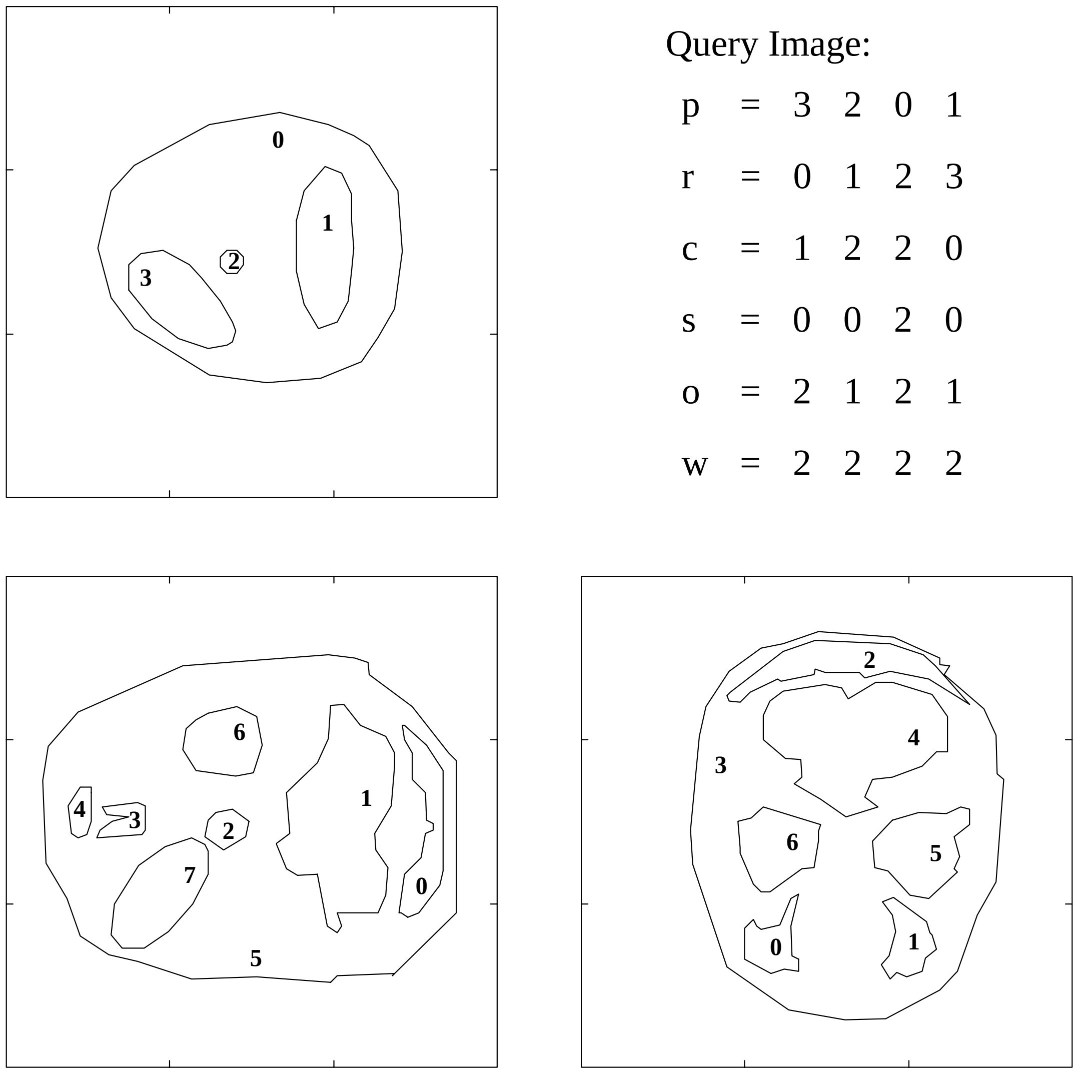
Retrieved Image:

Matching Subset: 6 0 2

Figure 5.12: Retrieval example 3.

query object 3. The left image is included in the answer set if the orientation is not used in comparisons. However, if the property of orientation is not used, then the right image and its subset (0 6 3 2) is included in the answer set, although it looks completely different than the query. To prevent images having different orientations (e.g., the right image) to be included in the answer set, a preprocessing step is required (before images are entered in the IDB) which puts all images in a standard orientation.

To deal with situations in which images match the query with respect to less than the total



Query Image:

| | | | | | |
|---|---|---|---|---|---|
| p | = | 3 | 2 | 0 | 1 |
| r | = | 0 | 1 | 2 | 3 |
| c | = | 1 | 2 | 2 | 0 |
| s | = | 0 | 0 | 2 | 0 |
| o | = | 2 | 1 | 2 | 1 |
| w | = | 2 | 2 | 2 | 2 |

Figure 5.13: Retrieval example 4.

number of attributes, retrievals have to be performed in stages. Instead of always using the same set of attributes in image comparisons, at each stage of query processing, the user is allowed to choose the number and the kind of properties to be used in retrievals. There may be more than one answer sets returned in response to a given query (one at each stage of retrieval), while a query answer set may be enlarged due to inexact matching. To deal with such situations, an appropriate user interface, which supports this kind of query processing assisted by an efficient mechanism for browsing must be developed.

## 5.9 Accuracy of Retrievals

The evaluation of the accuracy of retrievals is subject to human interpretation and has to do with how well the methodology succeeds in retrieving images which a user expects to be returned in response to a given query. In contrast to response times, where 2-D strings offer a good basis of comparison, a common basis for comparing the accuracy of retrievals is difficult to be adopted. Retrievals are accurate, in the sense that all images having exactly the same representation with the query are retrieved. However, in certain cases, such as those discussed in the previous section, the methodology may fail to retrieve images which look similar to the query.

Accuracy, is mainly a mater of image content representation and depends on how well a representation captures image content. Regarding accuracy, both the proposed methodology and (extended) 2-D strings are equivalent (i.e., they produce identical answer sets), since they make use of the same kind of representations. These are discrete (symbolic) representations of image content. However, in certain cases, these representations may be proven to be neither stable (see Section 3.7) nor complete (see Section 3.8). Increasing the accuracy of image content representations in IDB systems has become subject of independent research activities [113, 33]. Future proposals regarding image content representation, indexing and retrieval must take such efforts seriously into account.

# Chapter 6

# Epilog

## 6.1 Conclusions

IN THIS thesis, a new methodology has been presented which supports the efficient representation, indexing and retrieval of images by content. The contributions of this work are of both a theoretical and a practical nature.

At the theoretical level, this work can be viewed as an extension and generalization of the work of others. In particular, based on the known representations of 2-D strings [9], an indexing scheme has been proposed, which in contrast to 2-D strings, avoids an exhaustive search through the entire IDB. The representation of 2-D strings has been extended to take into account the inclusion relationships between objects and more than one object properties. In addition, two ordering criteria have been proposed and the representation of 2-D strings has been specialized to the cases of scaled and unscaled images respectively.

Indexing and retrievals are based on image subsets derived from all stored images. All subsets up to a prespecified maximum size $K_{max}$ are taken. Image subsets keep much redundant information, since image content can be represented by the subsets of size 2. However, the methodology owes its efficiency to the way redundant information is manipulated: an image subset of a specific size can be regarded as an outcome to a possible image query specifying the same number of objects. Subsets sharing common properties are indexed and stored together in groups of subsets of the same size, so that they can be retrieved together in response to a particular query. Furthermore, the proposed methodology generalizes and extends the work of [11], which focuses entirely on the problem of indexing and is based on image subsets of size 2. In [11], problems relating to image content representation, retrieval methodologies, and system performance are not taken in to account.

At the practical level, the efficiency of the methodology has been assessed. Performance evaluations have been carried out based on a database of simulated images, as well as images obtained with computed tomography and magnetic resonance imaging. The average size of the hypothesized answer set and the average retrieval response times has been used as performance measures. The results of the evaluations has been presented and discussed. The IDB of simulated images has been used as a testbed for comparing the performance of the proposed methodology with the performance of 2-D strings and that of [11]. The results demonstrate very significant retrieval performance improvements.

The performance improves with the size of the address space, which in turn increases with the number of query objects and the number of primary attributes. In addition, the performance depends on the ordering criterion which has been applied and the quantization value (performance improves with quantization value). In cases of indirect access queries, the performance improves with the maximum size of subsets stored $K_{max}$. In general, the performance depends also on the content of images of a particular application domain. Thus the need for tailoring indexing parameters to each application domain: given an example set of images which is large enough and characteristic of the application domain under consideration, the kind and the number of primary attributes, as well as the maximum size of image subsets $K_{max}$ need to be selected. Methods for the selection of the above parameters has been proposed.

The proposed methodology ensures fast retrieval response times in cases of direct access queries. Queries are fast even in the case of indirect access queries, especially when $K_{max}$ takes relatively large values (e.g., $K_{max} > 4$). The methodology results in faster retrieval response times than 2-D strings in all cases of direct access queries. The indirect access queries are faster for $K_{max} > 4$. However, indirect access queries can be further speeded-up.

This methodology is applicable in cases of images consisting of non-overlapping objects. However, even in cases where objects are partially visible, object positions may be specified by an expert user and indexing can still be based on the proposed representations. To ensure that all retrieved images satisfy the query criteria, one more step is needed in the search and retrieval process: all retrieved images must be compared to the query image on the basis of additional image properties and a more complete description of image content. 2-D C strings [33] provide a representation which has been shown to be better suited for the representation of images consisting of overlapping objects with complex shapes.

In certain cases, image representations may be proven to be unstable. Due to unstable representations, retrievals may be proven to be inaccurate. Problems relating to instabilities of image representations can be avoided at the expense of storage space. However, any attempt at increasing the accuracy and specificity of retrieval by storing more complete descriptions of image content would result in lower response times and more demanding average requirements. This is a tradeoff which must be carefully considered in the context of specific applications. Query mechanisms supporting the treatment of variable similarity criteria, assisted by appropriately designed user interfaces may also be used to increase the retrieval capabilities of the proposed methodology.

## 6.2 Directions for Further Research

The proposed methodology can be extended in many ways. For instance, we may consider the possibility of integrating, within the same indexing mechanism, additional kinds of image properties, either relational such as “connected to”, “adjacent”, “in between”, “overlap” or properties specific to the shape of objects. An image representation may also be augmented with properties specific to images found in a particular application domain. Furthermore, one may consider the possibility of using higher level properties (e.g., properties defined in terms of others) or classes of images sharing common characteristics, thus creating a higher level of indexing and retrieval. The proposed methodology can also be extended to include the indexing of image sequences in three or four dimensions by ordering objects in a space of three or four dimensions respectively, where the fourth dimension is time.

The retrieval capabilities of an IDB system can be significantly enhanced by extending the user interface with additional tools and mechanisms for image processing and with a powerful query language supporting the treatment of variable similarity criteria as well as the processing of various types of image queries. In particular, the methodology may be extended to support the processing of fuzzy queries e.g., “partial match” and “range” queries [99, 13]. In partial match queries, one or more properties of objects contained in a query image are left unspecified, in which case these properties are allowed to take any value in a specific domain. In range queries, instead of exact values, ranges of values of one or more properties corresponding to one or more query objects are specified. The proposed indexing mechanism must be extended to satisfy the requirements of the above types of images queries.

The time performance of image archiving and retrieval can be significantly improved when processing and retrievals are performed in parallel. In this case, the IDB may be partitioned to more than one processors. The kind of parallelism inherent to the image processing and retrieval tasks needs to be made explicit and an appropriate type of parallel architecture must be selected. Parallelism is best suited in the case of partial match queries, since such kinds of queries require more than one accesses to secondary storage and large amounts of data are expected to be retrieved and processed.

A number of techniques, such as cross-correlation (or point-pattern) matching techniques [68, 69, 71], techniques based on graph matching [8, 58, 70, 31] etc., are known to exist and can be used to retrieve images by content (see Section 2.7). The performance of the proposed methodology must also be compared against such techniques. Furthermore, the retrieval of images based on characteristics of the shape of objects is of particular importance in many application domains (i.e., robot vision, medical imaging etc.). A large number of techniques for object recognition are known to exist. The efficiency of existing object recognition techniques in retrieving images based on object similarity must be assessed in cases of large databases consisting of many thousands of objects.

## 6.3 IDB System Characteristics

The functional characteristics of a prototype medical IDB system in which the proposed methodology will be integrated are discussed in this section. The system is based on the integration of tools and methodologies which support the interactive processing, classification and browsing of medical images, as well as methodologies for the efficient automated indexing, storage and retrieval of such images by content. Earlier implementations of this system have been previously described in [114, 115, 16, 116].

The IDB system is capable of handling various types of medical image data, including the raw images themselves, attributes (e.g., dates, names), text (e.g., diagnosis related text), information extracted from images by automated or computer assisted image analysis, modality and image file header information etc. Image data are further distinguished into physical and logical. Logical image data are organized into classes forming hierarchies. Different kinds of classes can be identified and used to assist application modeling and to reduce the search space in specific queries. Specifically, logical image data may be organized into classes based on anatomy, diagnosis, imaging modality etc. Knowledge in the form of procedures (e.g., image processing and retrieval procedures corresponding to a specific class), rules, and parameters may be assigned to each class and may be inherited by the lower level classes. In designing and developing an IDB system which satisfies the needs for a hierarchical database organization, takes advantage of the property of inheritance, and is extensible, the object-oriented approach has been adopted. Physical data (images), are stored on a separate image store (e.g., on an optical disc) and organized in clusters based on the likelihood of being retrieved together in response to a particular query (e.g., the set of all images of the same patient may be stored in adjacent locations on the disc). Pointers are implemented from the logical to the physical database.

Commercial object oriented DBMSs continue to emerge, each providing different functionalities and obeying its own design principles. As a consequence, given the requirements of an application, the decision of which data model to use is not an easy one and, therefore, adopting one of the currently available models may not turn out to be proven to be the best solution in the long term. We have chosen to base the development of our prototype medical IDB system on a low-level storage subsystem (storage manager) [117, 118]. A storage subsystem provides mechanisms for the persistent storage and retrieval of objects. The subsystem to be selected must be extensible with DBMS features such as mechanisms for defining, creating, modifying and accessing both the data model (database schema) and the data. In addition, it must be extensible so that it can provide transaction management, concurrency control, authorization for a multiuser environment etc. A higher level subsystem is built on top of the storage manager and designed to support the desired functionality. The front-end module of this subsystem consists of an interactive user interface environment which supports the communication between the user and the various system components. In particular, this environment supports the following:

**Image Processing** It includes tools and mechanisms for image filtering, image registration, and the interactive or automated segmentation of medical images (see Section 3.2).

**Image Classification** Images may be classified into one or more predefined classes of an anatomical or a diagnostic hierarchy. An image may be simultaneously an instance of more than one classes found at any level in a particular hierarchy. The instances of each class may be stored (logically or physically) separately, so that the IDB is partitioned into database segments. This reduces the search space leading to faster retrievals, since a query addresses only a specific database segment.

**Image Queries** Queries may be specified by: (a) identifier, in which case the value of a single key (e.g., an image file name) is specified, (b) conditional statement involving values (or ranges of values) of various attributes, (c) an example image (grey level image or sketch) and (d) a combination of the above. A query language integrating within its syntax all kinds of image queries is under development. A user may interactively specify the specific database segment to be searched; otherwise, the whole hierarchy will be searched. All images satisfying query selection criteria need to be retrieved, while a user is allowed to make a final selection by browsing. Images and image related data (e.g., text, attributes) stored in the database may both have to be retrieved in response to a particular query.

**Image Browsing** The user interface is equipped with interactive graphical tools and mechanisms for both displaying and selecting image classes or class properties, and for exploring their interrelationships. Such display tools are referred to as “graph browsers” in [115].Furthermore, the user interface is equipped with an “image browser” whose purpose is to display reduced resolution images (miniatures) or sketches corresponding to images stored in the database.

Once the content of images has been extracted reliably, the system resumes responsibility for the efficient automated archiving and retrieval of images. The effectiveness of the IDB system will be significantly enhanced by incorporating into its storage and search mechanisms the methodology which has been presented in detail in this dissertation.

# Appendix A

# Image Indexing by 2-D Strings

## A.1 Introduction

2-D STRINGS [9] is one of a few representation structures originally designed for use in an IDB environment. The technique provides a simple and compact representation of spatial image properties in the form of two one-dimensional strings. Before the 2-D string representation of a given image is derived, the image is segmented and the locations of all objects (e.g., their center of mass) are projected along the $x$ and $y$ directions. By taking the objects from left to right and from below to above, two one-dimensional strings are produced forming the 2-D string representation of the image. A 2-D string represents the “left/right” and the “below/above” relationships between image objects. The objects themselves are represented by values corresponding to classes or names.

2-D strings can be used to resolve queries based on image content: the problem of image retrieval is transformed into one of two-dimensional string matching. However, search is exhaustive: 2-D string representations corresponding to all stored images are compared (one by one) with the 2-D string representation of a query. Algorithms of polynomial time complexity for comparing 2-D strings have been proposed in [9, 74]. In the following, the 2-

D string representation technique and three algorithms for 2-D string matching are presented. Both, the original 2-D string representation and the matching algorithms have been extented to take into account the inclusion relationships between objects as well as any number of object properties.

## A.2 Image Representation by 2-D Strings

Let $V$ be a finite set of symbols corresponding to names or classes of objects (e.g., $V = \{a, b, c, \ldots\}$) and let $A = \{<,\ =,\ :\}$ be a set of symbols denoting spatial relationships between objects ($V \cap A = \emptyset$). A 1-D string over $V$ is any string of the form $x_0, x_1, \ldots x_n$ where $n \geq 0$ and $x_i \in V,\ 0 \leq i < n.$ A 2-D string $(u, v)$ over $V$ is defined as

$$(u, v) = \Big(x_0\, y_0\, x_1\, y_1, \ldots y_{n-2}\, x_{n-1},\ x_{p[0]}\, z_0\, x_{p[1]}\, z_1, \ldots z_{n-2}\, x_{p[n-1]}\Big) \qquad \text{(A.1)}$$

where

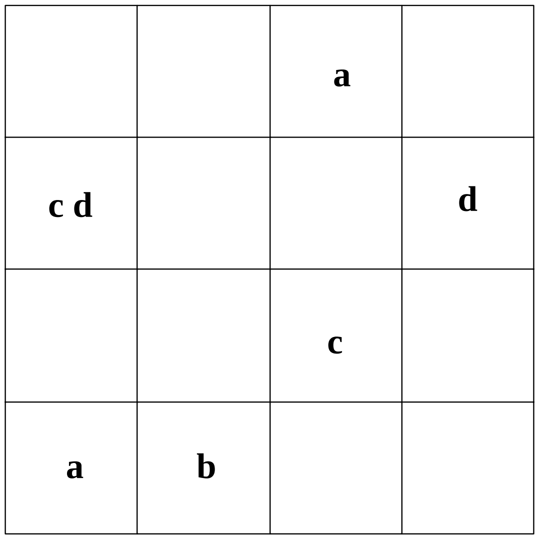
| | |
|---|---|
| $x_0\, x_1 \ldots x_{n-1}$ | is a 1-D string over $V$ |
| $y_0\, y_1 \cdots y_{n-1}$ | is a 1-D string over $A$ |
| $z_0\, z_1 \ldots z_{n-1}$ | is a 1-D string over $A$ |
| $p : \{0, 1, \ldots n-1\} \rightarrow \{0, 1, \ldots n-1\}$ | is a permutation over $\{0, 1, \ldots n-1\}$. |

For example, the 2-D string representation of the image of Figure A.1 is

$$(u, v) = (a = c : d < b < c = a < d,\ a = b < c < c : d = d < a) \qquad \text{(normal 2-D string)},$$

where

$$\begin{aligned}
x_0\, x_1\, x_2\, x_3\, x_4\, x_5\, x_6 &= a\ c\ d\ b\ c\ a\ d \\
y_0\, y_1\, y_2\, y_3\, y_4\, y_5\, y_6 &= \ = : < < = < \\
z_0\, z_1\, z_2\, z_3\, z_4\, z_5\, z_6 &= \ = < < : = < \\
p_0\, p_1\, p_2\, p_3\, p_4\, p_5\, p_6 &= 0\ 3\ 4\ 1\ 2\ 6\ 5.
\end{aligned}$$



**Augmented 2-D string:**

$$(u, v) = (acd < b < ca < d,\ 03 < 4 < 126 < 5)$$

Figure A.1: A symbolic image (left) and its corresponding augmented 2-D string (right).

The symbol ":" between objects denotes objects which have the same position in the image. For example, we write $c : d$, since the objects $c$, $d$ share the same grid position in the image of Figure A.1. The symbol "$=$" between objects in string $u$ or in string $v$ denotes objects which have the same projection along the $x$ or the $y$-axis respectively. Finally, the symbol "$<$" denotes the "left/right" relationships in string $u$ and the "below/above" relationships in string $v$. For example, the object $b$ in the image of Figure A.1 is on the right of the objects $a$, $c$, $d$, and on the left of the objects $c$, $a$, $d$. The objects $a$, $b$ are below all other objects.

The 2-D string representation of a given image may take various forms, namely "normal", "absolute", "augmented" etc. Details about these forms and the transformations between them can be found in [9]. Here, we concentrate our attention on the augmented form which can be derived directly from the above discussed normal form by eliminating all ":" and "$=$" symbols and by substituting the name of each object in string $v$ by an integer (index) denoting its position in string $u$. For simplicity, we consider the objects to be ordered according to the $x$-coordinate of their center of gravity. In general, any of the two ordering criteria proposed in Section 3.4 can be used for ordering image objects. The augmented form is unique in the sense that only images consisting of the same number of objects having the same names and the same spatial relationships result in the same augmented 2-D string representation. The augmented 2-D string representation of the image of Figure A.1 is

$$(u,v) = (a\ c\ d\ < b < c\ a < d,\ 0\ 3 < 4 < 1\ 2\ 6 < 5) \qquad \text{(augmented 2-D string).}$$

## A.3 Image Matching Using 2-D Strings

Here we present three algorithms for 2-D string matching following [9, 74], along with some corrections and some modifications to the meaning of symbols. First, we give some necessary definitions:

1. The "rank" $r_u$ of an object in a one-dimensional string $u$ equals to the number of "$<$" symbols preceding it in $u$. In particular, the rank $r_u$ of an object in a string $u$ representing the projections of objects along the $x$-axis, equals the number of objects on its left. Similarly, the rank $r_v$ of an object in a string $v$ representing the projections of objects along the $y$-axis, equals the number of objects which are below it.

2. A one-dimensional string $u$ is "contained" in a string $v$, if $u$ is a subsequence of a permutation of $v$. Furthermore, a string $u$ is a "type-$i$" ($i = 0,1,2$) 1-D subsequence of $v$ if (a) $u$ is contained in $v$ and (b) if $a_1 w_1 b_1$ is a substring of $u$, $a_1$ matches $a_2$ in $v$ and $b_1$ matches $b_2$ in $v$, then

   (type-0): $r_v[b_2] - r_v[a_2] \geq r_u[b_1] - r_u[a_1]$ or
   $r_u[b_1] - r_u[a_1] = 0$

   (type-1): $r_v[b_2] - r_v[a_2] \geq r_u[b_1] - r_u[a_1] > 0$ or
   $r_v[b_2] - r_v[a_2] = r_u[b_1] - r_u[a_1] = 0$

   (type-2): $r_v[b_2] - r_v[a_2] = r_u[b_1] - r_u[a_1]$

3. Let $(u_1, v_1)$, $(u_2, v_2)$ be the 2-D string representations of the images $f_1$, $f_2$ respectively. $(u_1, v_1)$ is a type-$i$ ($i = 0,1,2$) 2-D subsequence of $(u_2, v_2)$ if (a) $u_1$ is a type-$i$ 1-D

subsequence of $u_2$ and (b) $v_1$ is a type-$i$ 1-D subsequence of $v_2$. If $(u_1, v_1)$ is a type-$i$ 2-D subsequence of $(u_2, v_2)$, then $f_1$ is a type-$i$ match of $f_2$.

A matching algorithm must determine whether the 2-D string representation of an image (query) is a type-$i$ ($i = 0, 1, 2$) 2-D subsequence of the 2-D string representation of a second image. In the case of a successful match there is at least one 2-D subsequence within the second 2-D string matching the query. An algorithm must determine the existence of a match and, in the case of a successful match, must find all the matching subsequences. Three such algorithms have been proposed in [9]. The first is more general than the other two and may be used either for type-0, type-1 or for type-2 matching, while the remaining two can be used only for type-1 and type-2 matching respectively.

The augmented 2-D string representation $(u, v)$ of a given image is first expanded into a form $(x, r, s)$ of three one-dimensional strings, where $x$ contains the names or the classes of all objects in the same order as they appear in $u$, while the strings $r$ and $s$ contain the ranks of all objects along the $x$ and $y$ directions respectively. For example, the expanded representation of the image of Figure A.1 is

$$(x, r, s) = (a\ c\ d\ b\ c\ a\ d,\ 0\ 0\ 0\ 1\ 2\ 2\ 3,\ 0\ 2\ 2\ 0\ 1\ 3\ 2) \qquad \text{(expanded 2-D string).}$$

Once the expanded representations $(x_1, r_1, s_1)$ and $(x_2, r_2, s_2)$ of the two images under consideration, have been derived, an initial matching table $MI$ is constructed so that each object in the query image is associated with the set of objects in the second image which have the same name or class with it. If $m$ and $n$ is the number objects in the above two images, the initial matching table $MI$ is constructed as follows:

$$\forall\ i\ \in\ [0, m-1],\ \ MI[i] = \left\{ j \in [0, n-1] \,\middle|\, x_1[i] = x_2[j] \right\}. \qquad \text{(A.2)}$$

In the following, the objects from the two images are taken in pairs and their relative positions are compared. Let $(\mu, \nu)$ be a pair of objects from the first image with indices $\mu$

```
**function** $agree(\mu, \nu, j, k)$
    $ds_1 = s_1[\mu] - s_1[\nu]$;    $ds_2 = s_2[j] - s_2[k]$;
    $dr_1 = r_1[\mu] - r_1[\nu]$;    $dr_2 = r_2[j] - r_2[k]$;

  **if** $j = k$ **or** $ds_1 \cdot ds_2 < 0$ **then return(**0**);**
  **case** $type - 0$**:**
      **if** $dr_1 \cdot dr_2 >= 0$ **and** $|dr_2| \geq |dr_1|$ **and** $|ds_2| \geq |ds_1|$ **then return(**1**);**
        **else return(**0**);**
  **case** $type - 1$**:**
      **if** $(dr_2 \geq dr_1 > 0$ **or** $dr_2 = dr_1 = 0)$ **and**
        $(|ds_2| \geq |ds_1| > 0$ **or** $ds_2 = ds_1 = 0)$ **then return(**1**);**
            **else return(**0**);**
  **case** $type - 2$**:**
      **if** $dr_2 = dr_1$ **and** $ds_1 = ds_2$ **then return(**1**);**
        **else return(**0**);**
  **end;** *case*
**end.** *agree*
```

Figure A.2: Function $agree$ determines whether the pair of objects $(\mu, \nu)$ is a type-$i$ ($i = 0, 1, 2$) match of the pair $(j, k)$.

and $\nu$ respectively, where $\mu, \nu \in [0, m-1]$, and let $(j, k)$ be a pair of objects from the second image with indices $j$ and $k$ respectively, where $j, k \in [0, n-1]$. The function $agree$ of Figure A.2 determines whether the pair $(\mu, \nu)$ is a type-$i$ ($i = 0, 1, 2$) match of the pair $(j, k)$, in which case it returns 1; otherwise it returns 0.

The function $match2D$ of Figure A.3 determines whether the 2-D string representation of an image is a type-$i$ ($i = 0, 1, 2$) match of a second image, in which case it returns 1; otherwise it returns 0. The algorithm makes use of a triple indexed array $a$ taking values such that

$$k \in a[j,\nu,\mu] \Longleftrightarrow (k,\dots j) \sim (\nu-\mu-1,\dots \nu), \quad \nu,\mu \in [0,m-1],\ k,j \in [0,n-1]. \tag{A.3}$$

By writing $(k,\dots j) \sim (\mu,\dots \nu)$ we mean that the sequence $(k,\dots j)$ is a type-$i$ ($i = 0,1,2$) 2-D subsequence of $(\mu,\dots \nu)$ and both sequences have the same length. By setting $\mu = m-2$ and $\nu = m-1$ we get

$$k \in a(j, m-1, m-2) \Longleftrightarrow (k,\dots j) \sim (0,\dots m-1), \tag{A.4}$$

which means that the sequence $(k,\dots j)$ matches the sequence of all the $m$ query objects, and therefore, we have a successful match. The algorithm has time complexity $\mathcal{O}(n) + \mathcal{O}\left(m^2 \cdot lp^3\right)$, where $lp$ is the maximum length of all $MI[i]$ lists, where $i \in [0, m-1]$. If, in case of a successful match and prior to exiting $match2D$, the function $sequences$ of Figure A.4 is called, all type-$i$ ($i = 0,1,2$) subsequences matching the query are obtained. The algorithm makes use of a one-dimensional array $b$ which holds one matched subsequence at a time. The elements in all lists used by $match2D$ must be unique; otherwise, the same matched subsequence may appear in the output more than once. The time complexity of this algorithm is approximately $\mathcal{O}(m \cdot m_s)$, where $m_s$ is the number of matched subsequences. Therefore, the time complexity of $match2D$ becomes $\mathcal{O}(n) + \mathcal{O}\left(m^2 \cdot lp^3\right) + \mathcal{O}(m \cdot m_s)$.

When one is interested in finding all type-2 subsequences the algorithm $match2D2$ of Figure A.5 is preferred. This algorithm makes use of a table $match$ instead of $MI$, which is constructed as follows:

$$\forall\, a \in\ V,\ \ match[a] = \left\{ i \in [0, n-1] \,\middle|\, x_2[i] = a \right\}. \tag{A.5}$$

This table is also used by $match2D$ for the construction of $MI$. However, now the elements in all $match$ lists must be not only unique but also sorted. Furthermore, the algorithm uses an array $min$ of pointers to the above $match$ lists. The time complexity if this algorithm is approximately $\mathcal{O}(n \cdot m_f)$ where $m_f$ is the length of $match[x_1(0)]$ list.

An algorithm for producing all type-1 matched subsequences has also been proposed. Henceforth, we call this algorithm $match2D1$. It is the same as $match2D2$ except that before the $match$ array is constructed, the function $curb$ of Figure A.7 is invoked. The function $curb$ computes for every object in the query image the indices of its "front" and "back" objects. The front object of a query object is defined as the one on its left which is both above and closer to it. Similarly, the back object of a query object is defined as the one on its left which is both below and closer to it. Moreover, $match2D1$ makes use of a function $query1$ which is the same as $query2$ except that, instead of $agree$, the function $check$ of Figure A.6 is used. This function determines whether the pairs $(\mu, \nu)$, $(front[\mu], \mu)$ and $(\mu, back[\mu])$ are type-1 matches of the pairs $(j, k)$, $(b[front[\mu]], j)$ and $(j, b[back[\mu]])$ respectively, where $b[front[\mu]]$ and $b[back[\mu]]$ are the indices of the objects matching the front and back objects of the query object with index $\mu$.

## A.4 Extensions to 2-D Strings

The representation of 2-D strings and the algorithms for 2-D string matching has been extended to take into account more than one object properties as well as, the inclusion relationships between them. In particular, instead of a single string of names, a set of $k$ strings $(a_0, a_1, \dots a_{k-1})$ is used to characterize image objects. Each string corresponds to a distinct property such as size, roundness, orientation etc. For an image containing $n$ objects, a single value $x[i]$ is then assigned to each object $i$ as follows:

$$x[i] = a_0[i] + a_1[i] \cdot A_0 + a_2[i] \cdot A_0 \cdot A_1 + \cdots + a_{k-1}[i] \cdot \prod_{j=0}^{k-2} A_j, \quad 0 \leq i < n, \qquad \text{(A.6)}$$

where $0 \leq a_j[i] < A_j$ and $a_{k-1}[i] \neq 0$ except for $x[i] = 0$. It is assumed that each property $a_j$, $j \in [0, k-1]$, takes values within a range of values $[0, A_j - 1]$. For example, we may set $A_j = q, \forall j \in [0, k-1]$, where $q$ is the quantization parameter (see Section 3.6). The number of different values an object may take is

$$A = \prod_{j=0}^{j=k-1} A_j. \tag{A.7}$$

The tables $MI$ and $match$ can then be constructed as before. Furthermore, the 2-D string representation of an image can be extended with the inclusion string $w$ (see Section 3.5) which represents the "inside/outside" relationships between objects. In this case, a preprocessing step is followed, which uses $w$, to fill with values a double index array $io$. For an image containing $n$ objects, $io$ is defined as follows:

$$\forall\, i,j \in [0, n-1],\ i \neq j,\ io[i,j] = \begin{cases} inside & \text{if object } i \text{ is inside object } j; \\ outside & \text{if object } i \text{ is outside object } j; \\ invinside & \text{if object } j \text{ is inside object } i. \end{cases} \tag{A.8}$$

The array $io$ is constructed by applying the function $iorelations$ of Figure A.8. First, for each object $i$, a list $list[i]$ is created which holds the indices of all objects which contain it. The algorithm $match2D$ can then be directly extended to take into account the inclusion relationships between image objects. In particular, in function $agree$ it is required that $io1[\mu,\nu] = io2[j,k]$, where $io1$ and $io2$ are the $io$ arrays corresponding to the query and the candidate image respectively. The preprocessing step has, at most, square time complexity with respect to the number of objects in an image, which is added to the total time complexity of $match2D$. Therefore, the time complexity of $match2D$ becomes $\mathcal{O}(n^2) + \mathcal{O}(m^2) + \mathcal{O}\left(m^2 \cdot lp^3\right) + \mathcal{O}(m \cdot m_s)$. In practice, the extended 2-D string matching algorithm is likely to perform faster than the original, since the initial matching lists $match$ and $MI$ contain fewer objects, while matching based on $agree$ is more strict.

The algorithm $mathc2D$ guarantees that all pairs of query objects have been compared against the objects of a matched subsequence on the basis of their inclusion relationships in addition to the "left/right" and the "below/above" relationships. However, this is not the case with algorithms $match2D1$ and $match2D2$. To extend these algorithms to take into account the inclusion relationships between objects, all derived matching subsequences must be compared against the query on the basis of the inclusion relationships between the objects they contain. Specifically, all query objects and their corresponding objects in a matching subsequence are taken in pairs. If they have the same inclusion relationships the match is successful and the matched subsequence is included in the answer set. In order for an image as a whole to be included in the answer set, there must be at least one subsequence matching the query. This process has time complexity $\mathcal{O}(m^2 \cdot m_s)$, where $m_s$ is the number of subsequences returned. Therefore, the time complexity of $match2D1$ and $match2D2$ becomes $\mathcal{O}(n^2) + \mathcal{O}(m^2) + \mathcal{O}(n \cdot m_f) + \mathcal{O}(m^2 \cdot m_s)$.

## A.5 Correctness of 2-D String Matching

It has been shown in [74] that the algorithm $match2D$ allows “false drops” in certain cases (i.e., there may exist instances in the answer set which do not actually match the query). The conditions of occurrence of false drops are discussed in [74] and an algorithm which avoids false drops has also been proposed. In particular, false drops may occur with type-0 and type-1 matching, while false drops cannot happen in type-2 matching. Type-2 of matching is transitive: if the pair $(\mu, \nu)$ is type-2 match of the pair $(j, k)$ and the pair $(\lambda, \mu)$ is type-2 match of the pair $(l, j)$, then the pair $(\lambda, \nu)$ is also type-2 match of the pair $(l, k)$. The algorithm $mathc2D$ guarantees that all pairs $(k+1, k)$, $k \in [0, m-2]$, of query objects are examined. The same holds for $match2D2$ algorithm. Therefore, $match2D2$ does not allow false drops. Type-0 and type-1 of matching are not transitive. So far, we have not examined the possibility of occurrence of false drops of $match2D1$ algorithm.

```
function $match2D(m, p_1, x_1, r_1, s_1, n, p_2, x_2, r_2, s_2)$
    for $i = 0$ to $n - 1$ do insert $i$ in $match[x_2[i]]$;
    for $i = 0$ to $m - 1$ do $MI[i] = match[x_1[i]]$;
    for $i = 0$ to $m - 2$ do
        $MC = \emptyset$;
        for all $k$ in $MI[i]$, for all $j \in MI[i+1]$ do
            if $agree(i+1, i, j, k) \neq 1$ continue;
            if $k \notin a[j, i+1, 0]$ then insert $k$ in $a[j, i+1, 0]$;
            if $i > 0$ then
                $AP = a[k, i, 0]$;
                for $l = 1$ to $i$ do
                    $a[j, i+1, l] = \emptyset$;
                    for all $la$ in $AP$ do
                        if $agree(i+1, i-l, j, la) = 1$ then
                            if $la \notin a[j, i+1, l]$ then insert $la$ in $a[j, i+1, l]$;
                    end; for la
                    $AP = \bigcup_{t \in a[j,i+1,l]} a[t, i-l, 0] \cap a[k, i, l]$;
                end; for l
            end; if i
            if $a[j, i+1, l] \neq \emptyset,\ \forall\ l\ \in [0, i]$, then insert $j$ in $MC$;
        end; for k,j
        if $MC = \emptyset$ then return(0);
        $MI[i+1] = MC$;
    end for i
    return(1);
end. match2D
```

Figure A.3: Function $match2D$ determines whether the 2-D string $(x_1, r_1, s_1)$ is a type-$i$ ($i = 0, 1, 2$) subsequence of the 2-D string $(x_2, r_2, s_2)$.

```
function $sequences()$
    for all $i$ in $MI[m-1]$ do
        $b[m-1] = i$;
        $list(i, m-2)$;
    end; for i
end. sequences

function $list(i, k)$
    if $k < 0$ then
        for $l = 0$ to $m-1$ output $p_2[b[l]]$;
        return;
    end; if k
    for all $l$ in $a[i, k+1, 0] \cap a[b[m-1], m-1, m-k-2]$ do
        $b[k] = l$;
        $list(l, k-1)$;
    end; for l
end. list
```

Figure A.4: Function $sequences$ produces all type-$i$ ($i = 0, 1, 2$) matched subsequences when called before $match2D$ exits.

```
function $match2D2(m, p_1, x_1, r_1, s_1, n, p_2, x_2, r_2, s_2)$
    for $i = 0$ to $n - 1$ do insert $i$ in $match[x_2[i]]$;
    for all $v$ in $V$ do $min[v] = match[v]$;
    $k = 0$;    $a = x_1[0]$;
    for all $i$ in $min[a]$ do
        $b[0] = i$;
        $min[a]$ points to the element next to $i$;
        $query2(k, i)$;
    end; for i
end. match2D2

function $query2(k, i)$
    $a = x_1[k + 1]$;
    if $match[a] = \emptyset$ then return;
    for all $j$ in $min[a]$ do
        if $j \geq i$ and $agree(k + 1, k, j, i) = 1$ then
            $b[k + 1] = j$;
            if $k = m - 2$ then for $l = 0$ to $m - 1$ output $p_2[b[l]]$;
                else
                    $t = min[a]$;
                    $min[a]$ points to the element next to $j$;
                    $query2(k + 1, j)$;
                    $min[a] = t$;
                end; else if k
        end; if j
    end; for j
end. query2
```

Figure A.5: Function $match2D2$ determines whether the 2-D string $(x_1, r_1, s_1)$ is a type-2 subsequence of the 2-D string $(x_2, r_2, s_2)$ and produces all matched subsequences.

```
function $check(\mu, \nu, j, k)$
    $ds_1 = s_1[\mu] - s_1[\nu]$;                  $ds_2 = s_2[j] - s_2[k]$;
    $dr_1 = r_1[\mu] - r_1[\nu]$;                  $dr_2 = r_2[j] - r_2[k]$;
    $dz_1 = s_1[front[\mu]] - s_1[\mu]$;   $dz_2 = s_2[b[front[\mu]]] - s_2[j]$;
    $dt_1 = s_1[m] - s_1[back[\mu]]$;     $dt_2 = s_2[j] - s_2[b[back[\mu]]]$;

  if $j = k$ or $\mu = \nu$ then return(0);
  case $front[\mu] \neq null$ and $back[\mu] \neq null$:
    if $ds_1 \cdot ds_2 < 0$ or $dz_1 \cdot dz_2 < 0$ or $dt_1 \cdot dt_2 < 0$ then return(0);
    if ($dr_2 \geq dr_1 > 0$ or $dr_2 = dr_1 = 0$) and ($ds_2 \geq ds_1 > 0$ or $ds_2 = ds_1 = 0$) and
      ($dz_2 \geq dz_1 > 0$ or $dz_2 = dz_1 = 0$) and ($dt_2 \geq dt_1 > 0$ or $dt_2 = dt_1 = 0$)
        then return(1);
          else return(0);
  case $front[\mu] = null$ and $back[\mu] \neq null$:
    if $ds_1 \cdot ds_2 < 0$ or $dt_1 \cdot dt_2 < 0$ then return(0);
    if ($dr_2 \geq dr_1 > 0$ or $dr_2 = dr_1 = 0$) and ($ds_2 \geq ds_1 > 0$ or $ds_2 = ds_1 = 0$) and
      ($dt_2 \geq dt_1 > 0$ or $dt_2 = dt_1 = 0$) then return(1);
        else return(0);
  case $front[\mu] \neq null$ and $back[\mu] = null$:
    if $ds_1 \cdot ds_2 < 0$ or $dz_1 \cdot dz_2 < 0$ then return(0);
    if ($dr_2 \geq dr_1 > 0$ or $dr_2 = dr_1 = 0$) and ($ds_2 \geq ds_1 > 0$ or $ds_2 = ds_1 = 0$) and
      ($dz_2 \geq dz_1 > 0$ or $dz_2 = dz_1 = 0$) then return(1);
        else return(0);
  end; case
end. check
```

Figure A.6: Function $check$ determines whether the pairs of objects $(\mu, \nu)$, $(front[\mu], \mu)$ and $(\mu, back[\mu])$ are type-1 matches of the pairs $(j, k)$, $(b[front[\mu]], j)$ and $(j, b[back[\mu]])$ respectively.

```
**function** $curb()$
    **for** $i = 0$ **to** $m - 1$ **do**
        $front[i] = back[i] = null$;
    **for** $i = 0$ **to** $m - 1$ **do**
        $ds = \infty$;
        **for** $j = i - 1$ **to** $0$ **do**
            $ds_j = s_1[j] - s_1[i]$;
            **if** $ds_j \geq 0$ **and** $ds_j < ds$ **then**
                $front[i] = j$;
                $ds = ds_j$;
            **end;** *if* $ds_j$
        **end;** *for j*
        $ds = \infty$;
        **for** $j = i - 1$ **to** $0$ **do**
            $ds_j = s_1[i] - s_1[j]$;
            **if** $ds_j > 0$ **and** $ds_j < ds$ **then**
                $back[i] = j$;
                $ds = ds_j$;
            **end;** *if* $ds_j$
        **end;** *for j*
    **end;** *for i*
**end.** *curb*
```

Figure A.7: Function $curb$ computes, the indices of the “front” and “back” objects of the objects in a query image.

```
function $iorelations(w, n, io)$
    for $i = 0$ to $n - 1$ do
        for $j = 0$ to $n - 1$ do
            if $i \neq j$ then $io[i, j] = outside$;
        end; for j
    end; for i
    for $i = 0$ to $n - 1$ do
        if $w[i] \neq i$ then insert $w[i]$ in $list[i]$;
    for $i = 0$ to $n - 1$ do
        $j = element\ of\ list[i]$;
        $list[i]$ points next to $list[j]$;
    end for i
    for $i = 0$ to $n - 1$ do
        for all $j$ in $list[i]$ do
            $io[i, j] = inside$;
            $io[j, i] = invinside$;
        end; for j
    end; for i
end. iorelations
```

Figure A.8: Function $iorelations$ constructs, using the inclusion string $w$, the array $io$ which holds the inclusion relationships between objects in an image.

# Appendix B

# User Interface

THE USER interface[1] has been designed to facilitate the editing of primal segmentations, support the flexible formulation and drawing of example queries and the browsing of query results. In particular, the user interface consists of two separate window tools. The first, whose front-end is shown in Figure B.1, is a general purpose image editing and drawing tool which allows the user to specify (by clicking the mouse on the appropriate menu window) a function among the following:

- Draw a (query) image (option **"Draw"**).
- Edit an image by deleting, points, line segments or objects (option **"Delete"**).
- Close an open polygonal contour (option **"Close"**).
- Connect a pair of contours thus creating a new one (option **"Connect"**).
- Restore an edited image to a previous form (option **"Undo"**).
- Load or store an image (option **"Load/Store"**).

[1]The user interface has been implemented using X-windows, Athena widgets [119] and the C programming language.

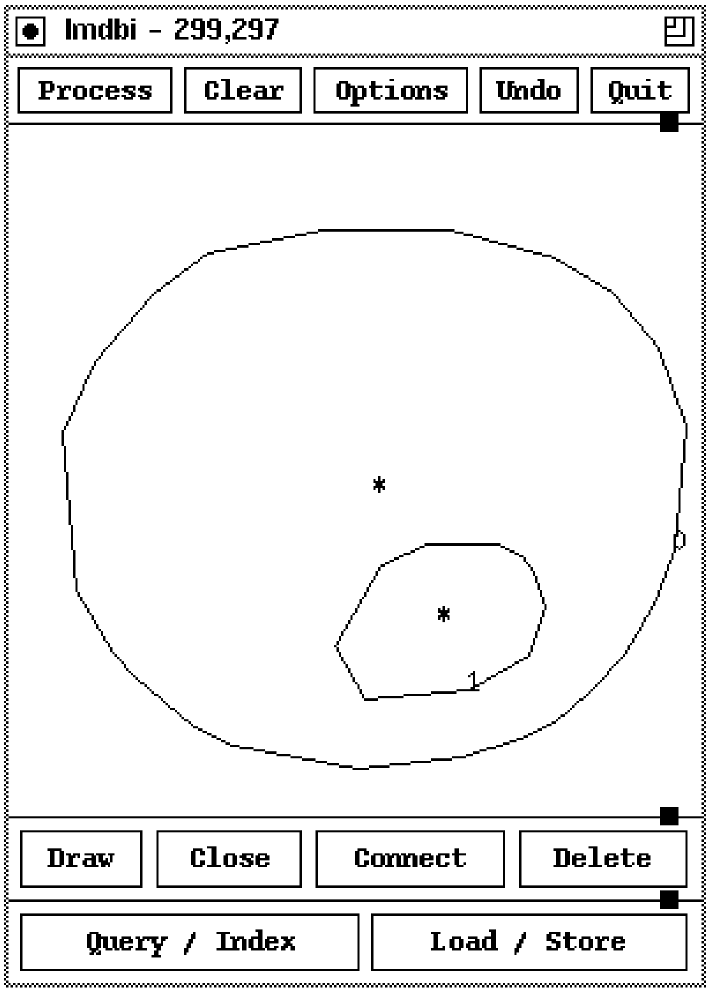


Figure B.1: Image editing and query drawing tool.

- Compute and display the representation corresponding to the loaded image (option **“Process”**).

- Emit a query or store operation (option **“Query/Index”**).

- Clear the window (option **“Clear”**).

- Exit the tool (option **“Quit”**).

In addition, there is a window menu (**“Options”**) which allows the user to display or hide the indices corresponding to all objects (integers near the objects), to display or hide the

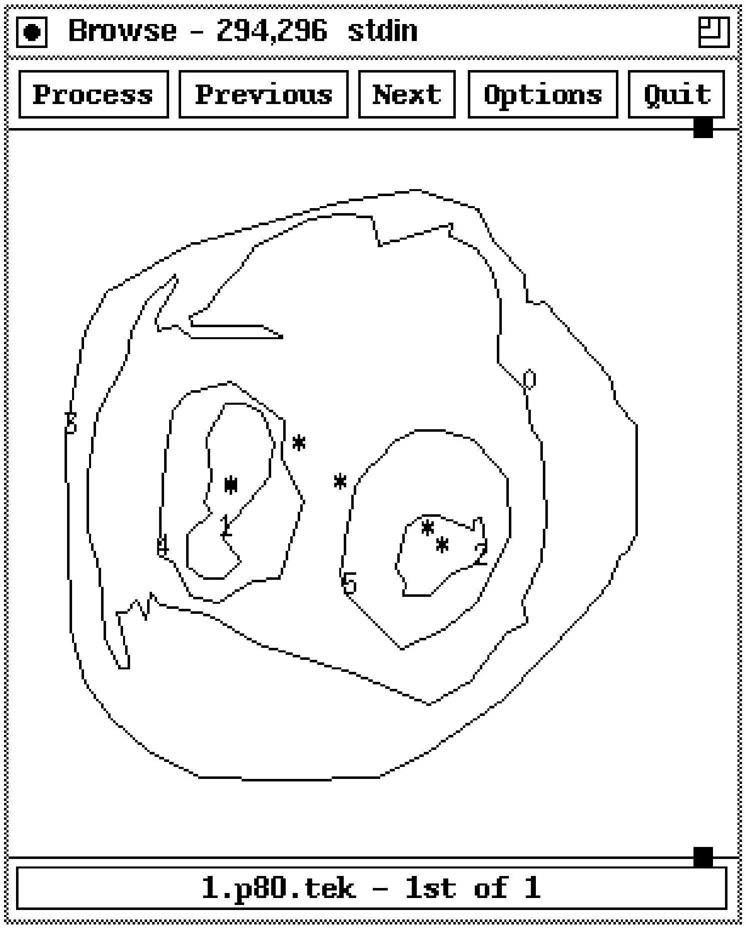


Figure B.2: Image browsing tool.

centers of mass of all objects (appear as asterisks), and finally, display or hide a rectangular grid of arbitrary size. Moreover, by pressing the middle mouse bottom, the user can move the object whose contour is close to the mouse pointer. In Figure B.1 a query image specifying two objects is shown.

In Figure B.2 the front-end of the browsing tool is illustrated. Except of options common to the first tool, there are options to load either the next or the previous image among the images retrieved (options **“Next”** or **“Previous”** respectively). The name of the retrieved image is shown at the bottom (left).

# Bibliography

[1] Hideyuki Tamura and Naokazu Yokoya. Image Database Systems: A Survey. *Pattern Recognition*, 17(1):29–49, 1984.

[2] Jayaram K. Udupa, Hsiu-Mei Hung, Dewey Odhner, and Roberto Goncalves. Multidimensional Data Format Specification: A Generalization of the American College of Radiology - National Electric Manufacturers Association Standards. *Journal of Digital Imaging*, 5(1):26–45, February 1992.

[3] Dennis Tsichritzis et.al. A Multimedia Office Filing System. In *Proceedings of 9th International Conference on Very Large Data Bases*, 1983.

[4] Won Kim. Object Oriented Databases: Definitions and Research Directions. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):327–341, September 1990.

[5] Christos Faloutsos and Yi Rong. Spatial Access Methods Using Fractals: Algorithms and Performance Evaluation. Technical Report UMIACS-TR-89-31, CS-TR-2214, University of Maryland, Colledge Park, Maryland, February 1989.

[6] Christos Faloutsos. Access Methods for Text. *ACM Computing Surveys*, 17(1):49–74, March 1985.

[7] Martin A. Fischler and Robert A. Elschlager. The Representation and Matching of Pictorial Structures. *IEEE Transactions on Computers*, c-22(1):67–92, 1973.

[8] Linda G. Shapiro and Robert M. Haralick. Structural Discriptions and Inexact Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(5):504–519, 1981.

[9] Shi-Kuo Chang, Qing-Yun Shi, and Cheng-Wen Yan. Iconic Indexing by 2-D Strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(3):413–428, May 1987.

[10] Shi-Kuo Chang. *Principles of Pictorial Information Systems Design*, chapter 8, pages 172–211. Prentice Hall International Editions, 1989.

[11] Chin-Chen Chang and Suh-Yin Lee. Retrieval of Similar Pictures on Pictorial Databases. *Pattern Recognition*, 24(7):675–680, 1991.

[12] Antonin Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD*, pages 47–57, June 1984.

[13] Christos Faloutsos and Shari Roseman. Fractals for Secondary Key Retrieval. Technical Report UMIACS-TR-89-47, CS-TR-2242, University of Maryland, Colledge Park, Maryland, May 1989.

[14] Shi-Kuo Chang and King-Sun Fu, editors. *Pictorial Information Systems*. Springer-Verlag, 1980.

[15] Shi-Kuo Chang. *Principles of Pictorial Information Systems Design*. Prentice Hall International Editions, 1989.

[16] Petros Kofakis and Stelios C. Orphanoudakis. Image Indexing by Content. In M. Osteaux et. al., editor, *A Second Generation PACS Concept*, chapter 7, pages 250–293. Springer-Verlag, 1992.

[17] Harry G. Barrow and Jay M. Tenenbaum. Computational Vision. *IEEE Proceedings*, 69(5):572–595, 1981.

[18] Thomas O. Binford. Survey of Model Based Image Analysis Systems. *Pattern Recognition*, 17(1):18–64, 1982.

[19] Roland T. Chin and Charles R. Dyer. Model-Based Recognition in Robot Vision. *ACM Computing Surveys*, 18(1):67–108, 1986.

[20] Stelios C. Orphanoudakis. Supercomputing in Medical Imaging. *IEEE Engineering in Medicine and Biology*, 7(4):16–20, 1988.

[21] Narendra Ahuja and B. J. Schachter. Image Models. *ACM Computing Surveys*, 13(4):372–397, 1981.

[22] Makoto Nagao. Control Strategies in Pattern Analysis. *Pattern Recognition*, 17(1):45–56, 1984.

[23] King-Sun Fu and Azriel Rosenfeld. Pattern Recognition and Computer Vision. *IEEE Computer*, 17(10):274–282, 1984.

[24] Azriel Rosenfeld. Image Analysis: Problems, Progress and Prospects. *Pattern Recognition*, 17(1):3–12, 1984.

[25] John K. Tsotsos. Knowledge and the Visual Process: Content, Form and Use. *Pattern Recognition*, 17(1):13–27, 1984.

[26] A. Ravishankar Rao and Ramesh Jain. Knowledge Representation and Control in Computer Vision Systems. *IEEE Expert*, 3(1):64–79, 1988.

[27] Dana H. Ballard and Christopher M. Brown. *Computer Vision*. Prentice Hall, 1982.

[28] Theo Pavlidis. *Algorithms for Graphics and Image Processing*. Computer Science Press, 1981.

[29] Panos Constantopoulos, Stelios C. Orphanoudakis, and Euripides G. Petrakis. An Approach to Multimedia Document Retrieval on the Basis of Pictorial Content. Technical

Report 011, Institute of Computer Science, Foundation of Research and Technology - Hellas, Heraklion, Greece, February 1988.

[30] Shi-Kuo Chang and King-Sun Fu. A Relational Database System for Images. In Shi-Kuo Chang and King-Sun Fu, editors, *Pictorial Information Systems*, pages 288–321. Springer-Verlag, 1980.

[31] M. A. Eshera and King-Sun Fu. An Image Understanding System Using Attributed Symbolic Representation and Inexact Graph-Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(5):604–618, 1986.

[32] Stelios C. Orphanoudakis, Euripides G. Petrakis, and Petros Kofakis. A Medical Image DataBase System for Tomographic Images. In *Proceedings of Computer Assisted Radiology, CAR89*, pages 618–622, Berlin, June 1989.

[33] Suh-Yin Lee and Fang-Jung Hsu. 2D C-Ssring: A New Spatial Knowledge Representation for Image Database Systems. *Pattern Recognition*, 23(10):1077–1087, 1990.

[34] C. J. Date. *An Introduction to Database Systems*, volume 1 of *The systems programming series*. Addison-Wesley, fifth edition, 1990.

[35] Shi-Kuo Chang and Shao-Hung Liu. Picture Indexing and Abstraction Techniques for Pictorial Databases. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(4):475–484, July 1984.

[36] Shi-Kuo Chang. *Principles of Pictorial Information Systems Design*, chapter 5, pages 82–121. Prentice Hall International Editions, 1989.

[37] Francois Bancilhon. Object Oriented Database Systems. In *ACM SIGART/SIGMOD/SIGACT, Proceedings of the 7th Symposium on Principles of Database Systems*, pages 152–162, Austin, Texas, March 1988.

[38] Won Kim. *Introduction to Object Oriented Database Systems*. MIT Press, 1990.

[39] John V. Joseph, Satish M. Thatte, Craig W. Thompson, and David L. Wells. Object Oriented Databases: Design and Implementation. *Proceedings of the IEEE*, 79(1):42–64, January 1990.

[40] Jack A. Orenstein and Frank A. Manola. PROBE Spatial Data Modeling and Query Processing in an Image Database Application. *IEEE Transactions on Software Engineering*, 14(5):611–629, 1988.

[41] Panos Constantopoulos. MUSE: A Multimedia Document System. *IEEE Office Knowledge Engineering*, 1(1):28–38, February 1987.

[42] S. Christodoulakis, M. Theodoridou, F. Ho, M. Papa, and A. Pathria. Multimedia Document Presentation, Information Extraction and Document Formation in MINOS: A Model and a System. *ACM Transactions on Office Information Systems*, 4(4):345–383, October 1986.

[43] M. Tanaka and T. Ichikawa. A Visual User Interface for Map Information Retrieval Based on Semantic Significance. *IEEE Transactions on Software Engineering*, 14(5):666–670, 1988.

[44] Koji Wakimoto, Mitsuhide Shima, Satoshi Tanaka, and Akira Maeda. An Intelligent User Interface to an Image Database Using a Figure Interpretation Method. In *Proccedings of 10th International Conference On Pattern Recognition*, pages 516–520, Atlantic City, New Jersey, June 1990.

[45] Shi-Kuo Chang, B. S. Lin, and R. Walser. A Generalized Zooming Technique for Pictorial Database Systems. In Walser Chang, Lin, editor, *Pictorial Information Systems*, pages 257–287. Springer-Verlag, 1980.

[46] Ning-San Chang and King-Sun Fu. Query by Pictorial Example. *IEEE Transactions on Software Engineering*, SE-6(6):519–524, November 1980.

[47] Nick Roussopoulos, Christos Faloutsos, and Timos Sellis. An Efficient Pictorial Database System for PSQL. *IEEE Transactions on Software Engineering*, 14(5):639–650, 1988.

[48] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. The R$^+$-tree: A Dynamic Index for Multidimensional Objects. In *Proceedings of 13th International Confernece on VLDB*, pages 507–518, England, September 1987.

[49] Thomas Joseph and Alfonso F. Gardenas. PIQUERY A High Level Query Language for Pictorial Database Management. *IEEE Transactions on Software Engineering*, 14(5):630–638, 1988.

[50] H. T. Clifford and W. Stephenson. *An Introduction to Numerical Classification*. Academic Press, 1975.

[51] Alberto Sanfeliou and King-Sun Fu. A Distance Measure Between Attributed Relational Graphs for Pattern Recognition. *IEEE Transactions on Systems Man and Cybernetics*, SMC-13(3):353–362, 1983.

[52] Andrew K. C. Wong and Manlai You. Entropy and Distance of Random Graphs with Application to Structural Pattern Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(5):599–609, September 1985.

[53] Wen-Hsiang Tsai and Shiaw-Shian Yu. Attributed String Matching with Merging for Shape Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(4):453–462, 1985.

[54] Michael Brady and Jonathan H. Connell. Generating and Generalizing Models of Visual Objects. *Artificial Intelligence*, 31:159–183, 1987.

[55] M. A. Eshera and King-Sun Fu. A Graph Distance Measure for Image Analysis. *IEEE Transactions on Systems Man and Cybernetics*, SMC-14(3):353–363, 1984.

[56] Paul Suetens, Pascal Fua, and Andrew J. Hanson. Computational Strategies for Object Recognition. *ACM Computing Surveys*, 24(1):5–61, March 1992.

[57] Larry S. Davis. Shape Matching Using Relaxation Techniques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(1):60–72, 1979.

[58] Robert C. Bolles and Ronald A. Cain. Recognizing and Locating Partially Visible Objects: The Local-Feature-Focus Method. *The International Journal of Robotics Research*, 1(3):57–82, 1982.

[59] Bir Bhanu and Olivier D. Faugeras. Shape Matching of Two-Dimensional Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(2):137–156, 1984.

[60] Zhisheng You and Anil K. Jain. Performance Evaluation of Shape Matching via Chord Length Distribution. *Computer Vision, Graphics and Image Processing*, 28:185–198, 1984.

[61] Farzin Mokhtarian and Alan Mackworth. Scale-Based Description of Plannar Curves and Two-Dimensional Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):34–43, 1986.

[62] Mark W. Koch and Rangasami L. Kashyap. Using Polygons to Recognize and Locate Partially Occluded Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):483–494, 1987.

[63] W. Eric L. Grimson and Tomas Lozano-Perez. Locating Overlapping Parts by Searching the Interpretation Tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):469–482, 1987.

[64] Alan Kalvin, Edith Schonberg, Jacob T. Schwartz, and Micha Sharir. Two-Dimensional, Model-Based, Boundary Matching Using Footprints. *The International Journal of Robotics Research*, 5(4):38–55, 1986.

[65] Paul G. Gottschalk, Jerry L. Turney, and Trevor N. Mudge. Efficient Recognition of Partially Visible Objects Using a Logarithmic Complexity Matching Technique. *The International Journal of Robotics Research*, 8(6):110–131, December 1989.

[66] Fridtjof Stein and Gerard Medioni. Efficient Two Dimensional Object Recognition. In *In Proccedings of 10th International Conference On Pattern Recognition*, pages 13–17, Atlantic City, New Jersey, June 1990.

[67] George Bebis, George Papadourakis, and Stelios C. Orphanoudakis. Model Based Object Recognition Using Multiresolution Segmentation and Artificial Neural Networks. Submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence, 1991.

[68] Sanjay Ranade and Azriel Rosenfeld. Point Pattern Matching by Relaxation. *Pattern Recognition*, 12:269–275, 1980.

[69] Daryl J. Kahl, Azriel Rosenfeld, and Alan Danker. Some Experiments in Point Pattern Matching. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-12(2):105–116, February 1980.

[70] Ramakant Nevatia and Keith E. Price. Locating Structures in Aerial Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(5):476–484, September 1982.

[71] H. S. Baird. *Model Based Image Matching Using Location*. MIT Press, 1985.

[72] Keith E. Price. Relaxation Matching Tecniques - A Comparison. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(5):617–622, September 1985.

[73] Andrew K.C. Wong and Richard Salay. An Algorithm for Constellation Matching. In *Proceedings of 8th International Conference on Pattern Recognition*, pages 546–549, Paris, France, October 1986.

[74] John Drakopoulos and Panos Constantopoulos. An Exact Algorithm for 2-D String Matching. Technical Report 021, Institute of Computer Science, Foundation of Research and Technology - Hellas, Heraklion, Greece, November 1989.

[75] Suh-Yin Lee, Man-Kwan Shan, and Wei-Pang Yang. Similarity Retrieval of Iconic Image Databases. *Pattern Recognition*, 22(6):675–682, 1989.

[76] Suh-Yin Lee and Fang-Jung Hsu. Spatial Reasoning and Similarity Retrieval of Images using 2D C-Sstring Knowledge Representation. *Pattern Recognition*, 25(3):305–318, 1992.

[77] Robert A. Wagner and Michael J. Fischer. The String-to-String Correction Problem. *Journal of the Association for Computing Machinery*, 21(1):168–173, January 1974.

[78] Patrick A. V. Hall and Geoff R. Dowling. Approximate String Matching. *ACM Computing Surveys*, 12(4):381–402, December 1980.

[79] Douglas Comer. The Ubiquitous B-Ttree. *ACM Computing Surveys*, 11(2):121–137, June 1979.

[80] R. J. Enbody and H. C. Du. Dynamic Hashing Schemes. *ACM Ttransactions on Database Systems*, 20(2):84–113, June 1988.

[81] Per-Ake Larson. Dynamic Hash Tables. *Communications of the ACM*, 31(4):446–457, April 1988.

[82] Per-Ake Larson. Dynamic Hashing. *BIT*, 18(2):184–201, 1978.

[83] R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong. Extendible Hashing - A Fast Access Method for Dynamic Files. *ACM Transactions on Database Systems*, 4(3):315–344, 1979.

[84] Witold Litwin. Linear Hashing: A New Tool for File and Table Addressing. In *Proceedings of 6th International Conference on VLDB*, pages 212–223, Montreal, October 1980.

[85] Per-Ake Larson. Linear Hashing With Partial Expansions. In *Proceedings of 6th International Conference on VLDB*, pages 224–232, Montreal, October 1980.

[86] G. N. N. Martin. Spiral Storage: Incrementally Augmentable Hash Addressed Storage. Technical Report 27, Department of Computer Science, University of Warwick, U.K, 1979.

[87] Per-Ake Larson. Linear Hashing with Seperators - A Dynamic Hashing Scheme Achieving One-Acess Retrieval. *ACM Ttransactions on Database Systems*, 13(3):366–388, September 1988.

[88] David Hsiao and Frank Harary. A Formal System for Information Retrieval from Files. *Communications of the ACM*, 13(2):67–73, February 1970.

[89] John T. Robinson. The k-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. In *Proceedings of ACM SIGMOD*, pages 10–18, 1981.

[90] H. V. Jagadish. Linear Clustering of Objects with Multiple Attributes. In *Proceedings of ACM SIGMOD*, pages 332–342, Atlantic City, May 1990.

[91] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The Grid File: An Adaptive, Symmetric, Multikey File Structure. *ACM Ttransactions on Database Systems*, 9(1):38–71, March 1984.

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

[93] Nobert Beckmann, Hans-Peter Kriegel, Ralf Scneider, and Bernhard Seeger. The R$^*$-tree: An Efficient and Robust Access Method for Points and Rectangles. In

*Proceedings of the 1990 ACM SIGMOD*, pages 322–331, Atlantic City, NJ, May 1990.

[94] Jon Louis Bentley and Jerome H. Friedman. Data Structures for Range Searching. *ACM Computing Surveys*, 11(4):397–409, December 1979.

[95] Hanan Samet. Hierarchical Representations of Small Rectangles. *ACM Computing Surveys*, 20(4):271–309, December 1988.

[96] Hanan Samet. The Quadtree and Related Data Structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

[97] Christos Faloutsos and Winston Rego. A Grid File for Spatial Objects. Technical Report UMIACS-TR-87-15, CS-TR-1829, University of Maryland, Colledge Park, Maryland, April 1987.

[98] Jack A. Orestein. Spatial Query Procesing in an Object Oriented Database System. In *ACM Proceedings SIGMOD 86*, pages 326–336, Washington, May 1986.

[99] Christos Faloutsos. Gray Codes for Partial Match and Range Queries. *IEEE Transactions on Software Engineering*, 14(10):1381–1393, October 1988.

[100] John Mylopoulos and Hector J. Levesque. An Overview of Knowledge Representation. In Michael L. Brodie, John Mylopoulos, and Joachim W. Schmidt, editors, *On Conceptual Modeling*, pages 3–17. Springer-Verlag, 1984.

[101] John Brolio, Bruce A. Draper, J. Ross Beveridge, and Allan A. Hanson. ISR: A Database for Symbolic Processing in Computer Vision. *IEEE Computer*, 22(12):22–30, 1989.

[102] Panos Constantopoulos, John Drakopoulos, and Yannis Yeorgaroudakis. Retrieval of Multimedia Documents by Pictorial Content: A Prototype System. In *Proceedings of International Conference on Multimedia Information Systems '91, ACM SIGIR*, Singapore, January 1991.

[103] Ranganchar Kasturi, Rodney Fernandez, Mukesh L. Amlani, and Wu-chun Feng. Map Data Processing in Geographic Information Systems. *IEEE Computer*, 22(12):10–20, 1989.

[104] Gianni L. Vernazza, Sebastiano B. Serpico, and Silvana G. Dellepiane. A Knowledge-Based System for Biomedical Image Processing and Recognition. *IEEE Transactions on Circuits and Systems*, CAS-34(11):1399–1416, 1987.

[105] S. Back, H. Neumann, and H. S. Stiehl. On Segmenting Computed Tomograms. In *Computer Assisted Radiology, CAR89,* pages 691–696, Berlin, June 1989.

[106] Ioannis Kapouleas. Segmentation and Feature Extraction for Magnetic Resonance Brain Image Analysis. In *Proceedings of 10th International Conference on Pattern Recognition*, pages 583–590, Atlantic City, New Jersey, June 1990.

[107] Edward M. Reingold, Jurg Nievergelt, and Narsingh Deo. *Combinatorial Algorithms*, chapter 5, pages 172–189. Prentice Hall, 1977.

[108] Dennis Stanton and Dennis White. *Constructive Combinatorics*, chapter 1, pages 1–25. Springer-Verlag, 1986.

[109] Edward M. Reingold, Jurg Nievergelt, and Narsingh Deo. *Combinatorial Algorithms*, chapter 5, pages 90–91. Prentice Hall, 1977.

[110] Theo Pavlidis. *Algorithms for Graphics and Image Processing*, chapter 15, pages 316–357. Computer Science Press, 1981.

[111] Edward M. Reingold, Jurg Nievergelt, and Narsingh Deo. *Combinatorial Algorithms*, chapter 2, pages 32–35. Prentice Hall, 1977.

[112] Berthold Klaus Paul Horn. *Robot Vision*, chapter 3, pages 46–64. MIT Press, 1986.

[113] Erland Jungert and Shi-Kuo Chang. An Algebra for Symbolic Image Manipulation and Transformation. In T. L. Kunii, editor, *Visual Database Systems*. Elsevier Science Publishers B.V. (North Holland), 1989.

[114] Stelios C. Orphanoudakis, Petros Kofakis, Yannis Kavaklis, and Euripides G.M. Petrakis. Medical Image Databases: Image Indexing by Content. In *Proceedings of North Sea Conference on Biomedical Engineering*, Antwerp, November 1990.

[115] Petros Kofakis and Stelios C. Orphanoudakis. Graphical Tools and Retrieval Strategies for Image Indexing by Content. In *Proceedings of Computer Assisted Radiology, CAR91*, pages 519–524, Berlin, July 1991.

[116] Euripides G.M. Petrakis and Stelios C. Orphanoudakis. Tools and Methodologies for the Indexing, Storage and Retrieval of Medical Images. In A. Todd-Pokropek and H. Lemke, editors, *Computer Assisted Management of Medical Images*. Springer-Verlag, 1992. to be published.

[117] M. Carey, D. DeWitt, and E. Shekita. Storage Management for Objects in Exodus. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases and Applications*. Addison-Wesley, 1989.

[118] Alexandros Billiris. The performance of Three Database Storage Structures for Managing Large Objects. In *Proceedings of the 1992 ACM SIGMOD*, pages 276–285, San Diego, California, June 1992.

[119] Joel McCormack, Paul Asente, and Ralph R. Swick. *X Toolkit Intrinsics - C Language Interface, X Window System*. Digital Equipment Corporation, External Research Group, MIT Project Athena, x version 11, release 4 edition, 1988.