

# Similarity Searching in Medical Image DataBases <sup>1</sup>

*Euripides G.M. Petrakis*<sup>2</sup>

*Christos Faloutsos*<sup>3</sup>

MULTimedia Systems Institute of Crete  
(MUSIC),  
Technical University of Crete,  
PO BOX 134 Chania, GR 73100 Greece.  
e-mail: petrakis@ced.tuc.gr

Department of Computer Science and  
Institute for Systems Research  
(ISR),  
University of Maryland.  
e-mail: christos@cs.umd.edu

## Abstract

We propose a method to handle approximate searching by image content in medical image databases. Image content is represented by attributed relational graphs holding features of objects and relationships between objects. The method relies on the assumption that a fixed number of “labeled” or “expected” objects (e.g., “heart”, “lungs” etc.) are common in all images of a given application domain in addition to a variable number of “unexpected” or “unlabeled” objects (e.g., “tumor”, “hematoma” etc.). The method can answer queries by example such as “*find all X-rays that are similar to Smith’s X-ray*”. The stored images are mapped to points in a multidimensional space and are indexed using state-of-the-art database methods (R-trees). The proposed method has several desirable properties: (a) Database search is approximate so that all images up to a prespecified degree of similarity (tolerance) are retrieved, (b) it has no “false dismissals” (i.e., all images qualifying query selection criteria are retrieved) and (c) it is much faster than sequential scanning for searching in the main memory and on the disk (i.e., by up to an order of magnitude) thus scaling-up well for large databases.

**Index Terms:** image database, image retrieval by content, query by example, image content representation, attributed relational graph, image indexing, R-tree, similarity searching.

## 1 Introduction

In many applications, images comprise the vast majority of acquired and processed data. In remote sensing and astronomy, large amounts of image data are received 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). In medicine, in particular, a large number of images of various imaging modalities (e.g., Computer Tomography, Magnetic Resonance etc.) are produced daily and used to support clinical decision making. The capabilities of

---

<sup>1</sup>IEEE Transactions on Knowledge and Data Engineering. Vol. 9, No. 3, May/June 1997, pp. 435-447.

<sup>2</sup>This research was partially funded by project DAIDALOS ECUS008:9828, EC-US Exploratory Activity, partially by project MILORD A2024, under programme AIM, and partially by project HERMES no 9141, under programme ESPRIT, of the European Union.

<sup>3</sup>On leave at AT&T Bell Laboratories, Murray Hill, NJ. This research was partially funded by the National Science Foundation under Grants IRI-9205273 and IRI-8958546 (PYI), with matching funds from EMPRESS Software Inc. and Thinking Machines Inc.

the above application fields can be extended to provide valuable teaching, training and enhanced image interpretation support, by developing techniques supporting the automated archiving and the retrieval of images by content. For example, in medicine, before making a diagnosis, a clinician could retrieve similar cases from the medical archive. Content-based retrievals would not only yield cases of patients with similar image examinations and similar diagnosis but also, cases of patients with similar image examinations and different diagnoses [1].

To support queries by image content in an Image DataBase (IDB), all images must be analyzed prior to storage so that, descriptions of their content can be extracted and stored in the database together with the original images. These descriptions are then used to search the IDB and to determine which images satisfy the query selection criteria. The effectiveness of an IDB system ultimately depends on the types and correctness of image content representations used, the types of image queries allowed and the efficiency of search techniques implemented.

Query formulation must be flexible and convenient (as opposed to queries expressed by a command-oriented query language like SQL). Ideally, queries must be specified through a graphical user interface, such as by example (i.e., by providing an example image or by drawing a sketch on the screen). Query by example permits even complicated queries: The user may specify several objects with complex shapes and inter-relationships and may ask for all images containing similar objects with similar relationships. The retrieved images need not be exactly similar to the query. Instead, database search must be approximate so that, all images up to a prespecified degree of similarity (tolerance) are retrieved.

Fast responses are essential to an IDB. An IDB system must employ searching methods that are faster than sequential scanning methods, and which must “scale-up” well (i.e., their performance remains consistently better than the performance of sequential scanning methods as the database grows).

In this work we deal with the following problem: Given a set of images, retrieve those which are similar to an example query (e.g., “find all X-rays that are similar to Smith’s X-ray”). We propose a general methodology which (a) uses an efficient representation of image content based on “Attributed Relational Graphs” (ARGs), (b) indexes the stored ARGs with state-of-the-art database methods (R-trees), (c) supports queries by example, and (d) supports approximate retrieval of images by content (i.e., based on both object properties and relationships between objects).

The proposed method can be used to accelerate the search for a given, established similarity metric and for a given set of attributes. The search method allows no “false dismissals”: It will find (or miss) exactly these images that sequential scanning would do (only faster). The performance of the proposed methodology has been evaluated based on an IDB of synthetic, but realistic, medical images. We have experimented with an index stored in main memory as well as with one stored on the disk. The results of this evaluation demonstrate significant performance improvements over traditional sequential scan with graph matching. Therefore, the method scales-up well for large databases.

The rest of this paper is organized as follows: The definition of the problem, the assumptions made and a short presentation of the underlying theory are presented in Section 2. A review

of related work done in the areas of Computer Vision and DataBases is presented in Section 3. The proposed methodology is presented in Section 4. In Section 5, experimental results are given and discussed. In section 6, we make several interesting observations on the proposed approach. Finally, the conclusions and the issues for future research are given in Section 7.

## 2 Problem Definition and Background

Given a collection of images we must derive appropriate representations of their content and organize the images together with their representations in the database so that, we can search efficiently for images similar to an example image. All images are segmented into closed contours corresponding to dominant image objects or regions. We assume that all images contain a number of “expected” or “labeled” objects. These are objects common in all images of a given application domain. For example, in medical images, the expected objects may correspond to the usual anatomical structures (e.g., “heart”, “lungs”) and the outline contour of the body. Similarly, the labeled objects may be the cell outline in microscope images; the sun and the horizon in outdoor images etc. All expected objects are identified prior to storage and a class or name is assigned to each one. The labeled objects need not be similar in all images.

Not all objects need to be identified: Images may also contain “unexpected” or “unlabeled” objects. These may be either objects not present in all images or objects whose characterization is difficult or ambiguous. For example, in medical images, the unexpected objects may correspond to abnormal (pathological) structures (e.g., “hematoma”, “tumor” etc.).

In this work, we deal with images containing a fixed number ( $k$ ) of labeled objects and a variable number ( $u \geq 0$ ) of unlabeled objects. We also assume that the labeled objects have different labels. Objects not common in all images are treated as unlabeled (unexpected). Similarly, queries by example may specify a fixed number of labeled objects and a variable number of unlabeled objects. Notice that this is a general setting. One special case is the case where all objects are unlabeled ( $k = 0$ ). Another special case is when all objects are labeled ( $u = 0$ ) in all images as was the case in [2]. There, the problem was to search a database of face images; from each image, a fixed number of labeled objects are identified (eyes, nose, etc.) and their attributes and relative positions are computed.

### 2.1 Background

Image descriptions are given in terms of object properties and in terms of relationships between objects. The textbook approach to capture this information is the Attributed Relational Graphs (ARGs) [3, 4]. In an ARG, the objects are represented by graph nodes and the relationships between objects are represented by arcs between such nodes. Both nodes and arcs are labeled by attributes corresponding to properties (features) of objects and relationships respectively.

Figure 1 shows an example image (a line drawing showing a face) containing four objects (numbered 0 through 3) and its corresponding ARG. Each object has an attribute ( $c$ ) denoting its name or class and, in this toy example, only one, attribute: the length ( $l$ ) of its boundary.

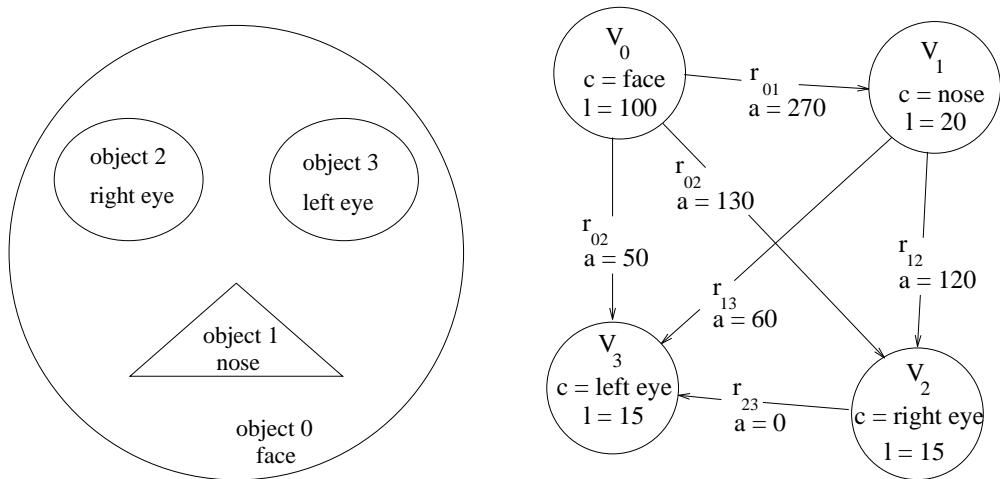


Figure 1: Example image showing a sketch of a face (left) and its corresponding ARG (right).

The relationship between any two objects has (in this toy example again) also one attribute, namely, the angle ( $a$ ) with the horizontal direction of the line connecting the centers of mass of these objects.

The specific features which are used in ARGs are derived from the raw image data and, depending on the application, can be geometric (i.e., independent of pixel values), statistical or textural, or features specified in some transform domain (e.g., Fourier coefficients of object shapes). In the case of medical CT and MRI images used in this work, the set of features is given in Section 4.1. However, the proposed methodology is independent of any specific kind of features.

The problem of retrieving images which are similar to a given example image is transformed into a problem of searching a database of stored ARGs: Given a query, its ARG has to be computed and compared with all stored ARGs. Matching between ARGs is a well known problem and has been studied extensively in the Computer Vision literature [5, 6, 3]. Specifically, matching a query and a stored graph is treated as a subgraph isomorphism problem.

Figure 2 shows an example query and its corresponding ARG. In this example, query object 0 can only be associated with object 0 of the image of Figure 1 since, this is the only object having the same label with it. Similarly, query object 1 is matched with object 2. Their corresponding relationships are matched too. Equivalently, query node  $v'_0$  is associated to node  $v_0$ ,  $v'_1$  to  $v_2$  and arc  $r'_{01}$  is associated to arc  $r_{02}$  of the graph of the original image. However, if the label of a query object is unknown, all possible associations between this query object and the objects in the original image have to be examined. The problem becomes harder if the query or the original image contains many unlabeled objects or objects with the same label. Then matching becomes a hard combinatorial problem.

In comparisons between ARGs, we need a measure of the “goodness” of matching. A measure of goodness is defined in [3]: Let  $Q$  be a query image consisting of  $q$  objects and  $S$  be a stored image consisting of  $s$  objects. Let  $F()$  be a mapping from objects in  $Q$  to objects in

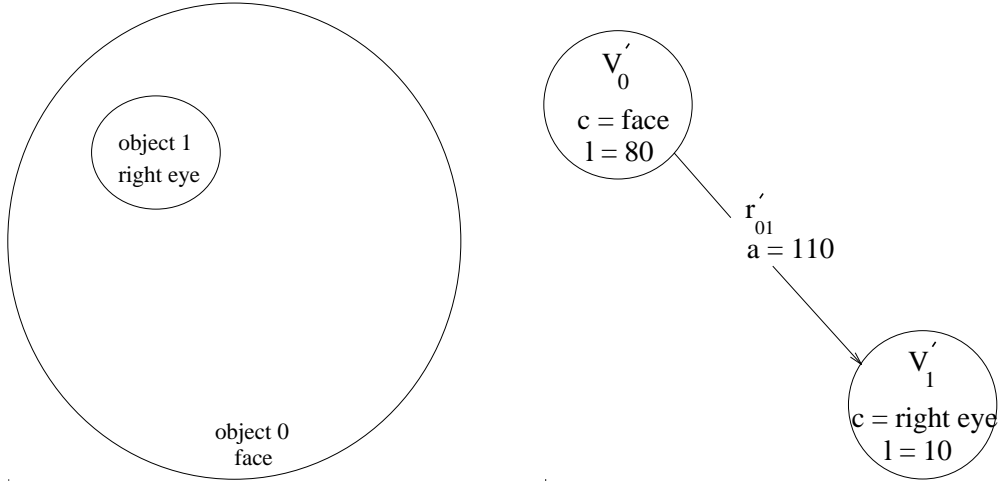


Figure 2: Example query image (left) and its corresponding ARG (right).

$S$  (e.g., such a mapping associates objects with the same labels). The cost of this mapping is defined as:

$$Dist_F(Q, S) = \sum_{i \in [1, q]} COST(i, F(i)) + \sum_{i, j \in [1, q]} COST(i, j, F(i), F(j)) \quad (1)$$

The first term in Equation 1 is the cost of matching associated nodes while, the second term is the cost of matching the relationships between such nodes. In our setting, only a subset of the objects in the stored image  $S$  need to be matched. There is no cost if the data image contains extra objects; however, we assume that the cost is infinite if the data image is missing one of the objects of the query. In this work, we don't examine mappings that assign a finite cost for missing objects.  $COST$  is the cost of matching features of objects or features of relationships between associated objects. The distance between images  $Q$  and  $S$  is defined as the minimum distance computed over all possible mappings  $F()$ :

$$Dist(Q, S) = \min_F \{Dist_F(Q, S)\}. \quad (2)$$

The typical way to compute  $Dist_F(Q, S)$  is using an  $L_p$  metric. This is done as follows: Let  $(q_1, q_2 \dots q_K)$  be a vector of feature values derived from  $Q$  by taking the features of all its objects and of their relationships in some prespecified order (e.g., object 1 and its relationships with the remaining objects are taken first, followed by the features of object 2, etc.). Let  $(s_1, s_2, \dots s_K)$  be the vector derived from  $S$  by taking the features of the objects associated to objects in  $Q$  in the same order. Then, Equation 1 can be written as follows:

$$Dist_F(Q, S) = Dist_{p, F}(Q, S) = \left[ \sum_{i=1}^K |q_i - s_i|^p \right]^{1/p} \quad (3)$$

$p$  is the order of the metric. For  $p = 1$  and  $p = 2$  we obtain the Manhattan (city-block) and the Euclidean distance respectively. For example, the Manhattan distance between the query image

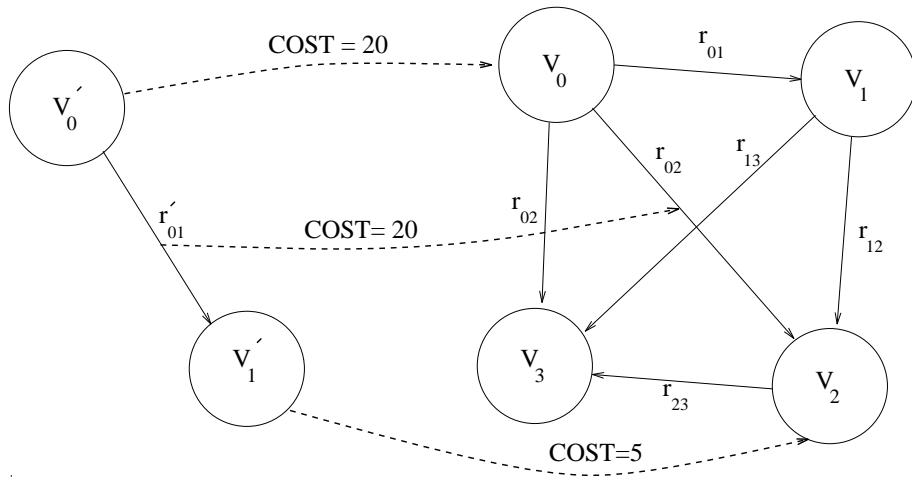


Figure 3: Matching between the query and the original example image.

of Figure 2 and the example image of Figure 1 is  $Dist(Q, S) = |100 - 80| + |15 - 10| + |130 - 110| = 45$ . We have omitted the subscript  $F$  because there is only one mapping.

Similarity searching in an IDB of stored ARGs requires that all images within distance  $t$  must be retrieved. Specifically, we have to retrieve all the images  $S$  that satisfy the following condition:

$$Dist(Q, S) \leq t. \quad (4)$$

Without loss of generality, we use the Euclidean distance ( $p = 2$ ). However, the proposed method can handle *any*  $L_p$  metric.

### 3 Survey - Related Work

Important considerations in the design and implementation of IDB systems supporting queries by image content are: Image feature extraction, image content representation and organization of stored information, search and retrieval strategies, and user interface design. Addressing such issues has become object of intensive research activities in many areas of Computer Science over the past few years [7, 4, 8]. Advances mainly in the areas of Databases and Computer Vision research resulted in methods which can be used for image archiving, retrieval and IDB design work. However, as observed in [9], there is a need for increased communication between the vision and the database communities to deal with the above issues. Combining results from both areas is an important next step.

#### 3.1 Image Retrieval by Content

Image content can be described indirectly through attributes (e.g., subject, speaker, etc.) or text (e.g., captions) [10]. Queries by image content require that, prior to storage, images are processed, and appropriate descriptions of their content are extracted and stored in the database.

Retrievals by image content is not an exact process (two images are rarely identical). Instead,

all images with up to a pre-specified degree of similarity have to be retrieved. The design of appropriate image similarity/distance functions is a key issue and is application-dependent. An almost orthogonal issue is speed of search. Next, we review techniques for exact and approximate database search, along with methods to accelerate the search.

**Exact Match Searching in Image Databases:** 2-D strings [11] constitute an efficient image content representation and provide low complexity (i.e., polynomial) matching in image databases. A unique name or class is assigned to each object. The relative positions between all objects are then represented by two one-dimensional strings. The problem of image retrieval is transformed into one of string matching: All 2-D strings containing the 2-D string of the query as a substring are retrieved. To speedup retrievals, methods for indexing 2-D strings in a database has been proposed in [12, 13]. 2-D C strings [14] deal with situations of overlapping objects with complex shapes.

As shown in [12], 2-D strings may yield “false alarms” (non-qualifying images) and “false dismissals” (qualifying but not retrieved images). Techniques for inexact match retrievals based on 2-D strings have also been proposed in [15, 16], but they require exponential time for matching.

**Approximate Searching in Image Databases - No Indexing:** Systems described in the Machine Vision literature typically focus on the quality of the features and the matching function, with little or no emphasis on the speed of retrieval. Thus, each image is described by a set of features; to respond to a query, the system searches the features of all the images sequentially. A typical, recent system along these lines is described in [2]. The system supports the segmentation and interactive retrieval of facial images from an IDB. A-priori knowledge regarding the kind and the positioning of expected image objects (e.g., face outline, nose, eyes etc.) is employed and used to guide the segmentation of face images into disjoint regions corresponding to the above objects. The database search is exhaustive, using sequential scanning.

**Approximate Searching in Image Databases - With Indexing:** An attempt to combine indexing and approximate database search is proposed in [17]. The main idea is to extract  $f$  features from each image, thus mapping images into points in a  $f$ -dimensional space. Any spatial access method can then be used to handle range and nearest-neighbor queries efficiently. The original paper did not address the issue of false dismissals, nor the problem of retrieving images by specifying properties of objects and relationships between objects. However, this paper had a significant impact. Most of the papers below, were either directly influenced, or discovered this approach independently.

The work in [18] continues along these lines, on medical images. Given a picture, the four “most important” objects are taken and their centers of mass are used to represent their spatial relationships. However, this approach seems to allow for false dismissals: If an X-ray has 5 objects, one of them will be ignored. Thus, queries on this fifth object will never retrieve that X-ray.

In the QBIC project of IBM [19], we proposed an indexing method for queries on color, shape and texture, achieving faster-than-sequential scanning retrievals. The idea was again to use an R-tree for indexing; the main contribution was to transform the features, lower-bounding the actual distance, so that we can guarantee no false dismissals.

The CAFIIR system [20] proposes the “iconic index tree” to accelerate the search on facial images. One novelty of the system is that it can process “fuzzy” (i.e., subjective or incomplete) queries, through the so-called “fuzzification” technique, which translates the feature space to a fuzzy space.

Along the same lines, Grosky and Mehrotra [21] proposed the “feature index tree” to handle the recognition of two-dimensional objects; Rabitti and Savino [22] used a multilevel signature technique to search line-drawings.

However, in all of the above methods, one or more of the following statements holds: (a) they do not index on the relationships between objects (b) they do not scale up, for large, disk-based databases (c) they have false dismissals. Our proposed method (in Section 4) solves all these three issues.

## 3.2 Spatial Access Methods

As mentioned before, we can achieve faster-than-sequential searching by using the so-called “spatial access methods”. These are file structures to manage a large collection of  $f$ -dimensional points (or rectangles or other geometric objects) stored on the disk so that, “range queries” can be efficiently answered. A range query specifies a region (e.g., hyper-rectangle or hyper-sphere) in the address space, requesting all the data objects that intersect it. If the data objects are points (as eventually happens in our application), the range query requires all the points that are inside the region of interest. An example of a range query on point data is “retrieve all the cities that are 200 km away from Brussels”. Spatial access methods can also handle “nearest neighbor” [23] and “all-pairs (or “spatial-join”)” [24] queries. For clarity, in this paper we focus on range queries only.

Several spatial access methods have been proposed, forming the following classes: (a) Methods that transform rectangles into points in a higher dimensionality space [25]; (b) methods that use linear quad-trees or, equivalently, the “ $z$ -ordering” [26] or other “space filling curves” [27, 28]; and finally, (c) methods based on trees (k-d-trees [29] etc.). One of the most characteristic approaches in the last class is the R-tree [30].

The R-tree can be envisioned as an extension of the B-tree for multidimensional objects. A geometric object is represented by its Minimum Bounding Rectangle (MBR). Non-leaf nodes contain entries of the form  $(ptr, R)$  where  $ptr$  is a pointer to a child node in the R-tree;  $R$  is the MBR that covers all rectangles in the child node. Leaf nodes contain entries of the form  $(object - id, R)$  where  $object - id$  is a pointer to the object description, and  $R$  is the MBR of the object. The main idea in the R-tree is that father nodes are allowed to overlap. This way, the R-tree can guarantee good space utilization and remains balanced. Extensions, variations and improvements to the original R-tree structure include the packed R-trees [31], the  $R^+$ -tree [32],



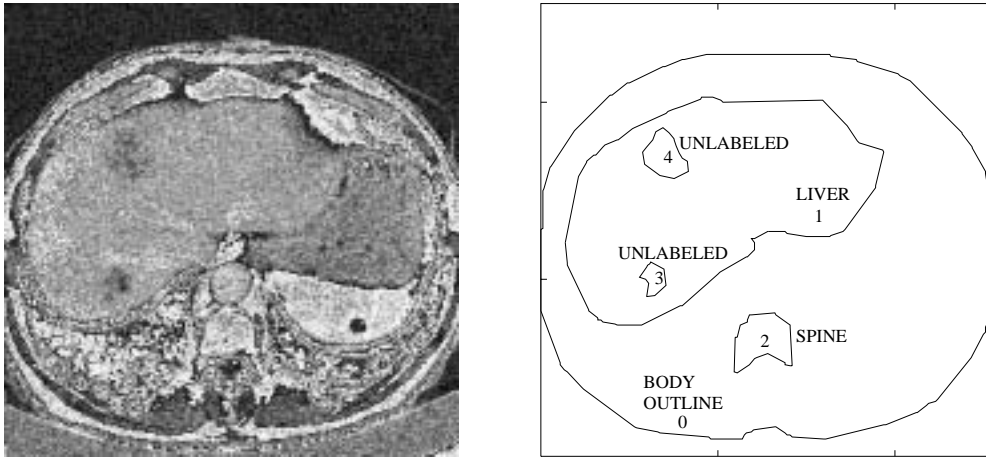


Figure 4: *Example of an original grey-level image (left) and its segmented form (right) showing 3 labeled objects (body outline with index 0, liver with index 1 and spine with index 2) and 2 unlabeled with indices 3 and 4.*

the R\*-tree [33], and the Hilbert R-tree [34].

## 4 Proposed Solution

In this work, we use the R-tree as the underlying method for indexing images by content. The reason for our choice is that the R-tree is more robust in high-dimensionality address spaces. As we shall see, in IDB applications the number of attributes/dimensions can be high (20-30), which can still be handled by an R-tree [35]. On the contrary, the main competitors of the R-tree (see Section 3.2) require time or space that is exponential on the dimensionality.

Henceforth, we assume that images are segmented into closed contours, corresponding to labeled and unlabeled objects of interest. Figure 4 shows an example of an original MRI image and of its segmented form. We can easily identify three labeled objects namely, spine (object 2), liver (object 1) and body outline (object 0). Henceforth, the number of labeled objects in all images will be  $k = 3$ .

Techniques for the automatic segmentation and recognition of tomographic images are currently becoming available [36, 37, 38]. However, image automatic segmentation and labeling of the components are outside the scope of this paper. For our purposes, we assume that each image has been segmented automatically or manually (e.g., by tracing the contours of the objects of interest) and that its components have been labeled (typically, by a domain expert). The contribution of our work is on the fast searching after the images and the queries have been segmented and labeled.

## 4.1 Image Content Description

The descriptions used in this work are given in terms of properties of objects contained in images and in terms of relationships between such objects. Individual objects are described by properties corresponding to characteristics of their position, size and shape. A set of features that has been used successfully [39, 12] is the following:

- *Size* ( $s$ ), computed as the size of the area it occupies.
- *Roundness* ( $r$ ), computed as the ratio of the smallest to the largest second moment.
- *Orientation* ( $o$ ), defined to be the angle between the horizontal direction and the axis of elongation. This is the axis of least second moment.

The following properties are used to describe the spatial relationships between two objects:

- *Distance* ( $d$ ), computed as the minimum distance between all pairs of line segments, taking one from each object.
- *Relative Position* ( $p$ ), defined as the angle with the horizontal direction of the line connecting the centers of mass of the two objects.

The above set of features is by no means unique. Additional features that could be used include the average grey-level and texture values, moments or Fourier coefficients etc. as object descriptors; relative size, amount of overlapping or adjacency etc. can be also used to characterize the relationships between objects. Notice that the proposed method can handle *any* set of features that the domain expert may deem appropriate. The contribution of this work is not on choosing a good set of features, but on accelerating the sequential search on a given, expertly chosen, set of features.

Figure 5 shows the ARG representing the content of the example image of Figure 4. Nodes correspond to objects and arcs correspond to relationships between objects. Both nodes and arcs are labeled by the attribute values of the object properties and the relationship properties, respectively. Angles are in degrees.

The derived representation is both scale and translation invariant (i.e., images translated or scaled with respect to each other result in the same representation). To achieve scale invariance, we normalize lengths and areas by dividing them respectively by the diameter and the area of the largest object. The features we have chosen are also independent of image translation since, only relationships between objects are used to characterize object positions. To achieve rotation invariance, all images are registered to a standard orientation (e.g., the axis of elongation of the outline object is made horizontal).

## 4.2 Image Indexing - File Structure

Our goal is to achieve fast searching in a database of stored ARGs. The approach we follow is to map ARGs into points in a multidimensional space. Then, a multidimensional access method

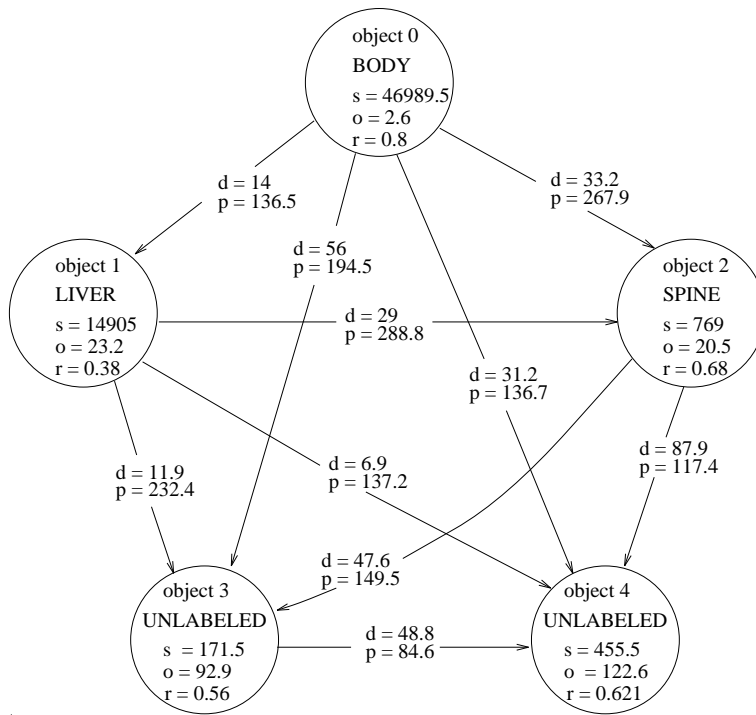


Figure 5: Attributed graph representing the content of the example image of Figure 5.

can be used. However, such a method requires that the number of dimensions (equivalently, the number of keys or axes) are known in advance and are fixed. In our case, the number of objects is not the same in all images, and therefore, the number of dimensions cannot be fixed. We solve this problem by decomposing each image into images containing an equal number of objects, called “sub-images”:

**Definition 1** A  $(k, u)$  sub-image of a given image contains the  $k$  labeled objects and  $u$  unlabeled objects.

An attribute vector is then computed for each derived sub-image as follows: The vector consists of the property values of the contained objects, as well as of the property values of the relationships between these objects, taking them in a prespecified order (i.e., object 1 and its relationships with the remaining objects are taken first, followed by the attribute values of object 2 etc.). For example, using the features described before, a  $(k, 1)$  sub-image ( $k = 3$ ) is represented by a 24-dimensional vector: 4 objects with 3 attributes each, plus  $\binom{4}{2} = 6$  relationships, with 2 attributes each. The vectors of all  $(k,1)$  sub-images are then stored in an R-tree data structure. For each vector, an image identifier corresponding to the image from which it has been derived is also stored.

Figure 6 demonstrates the proposed file structure of the data on the disk. Specifically, the IDB consists of the following parts:

- The “image file store” holding the original image files. For faster display, we have also kept the segmented polygonal forms of all images.

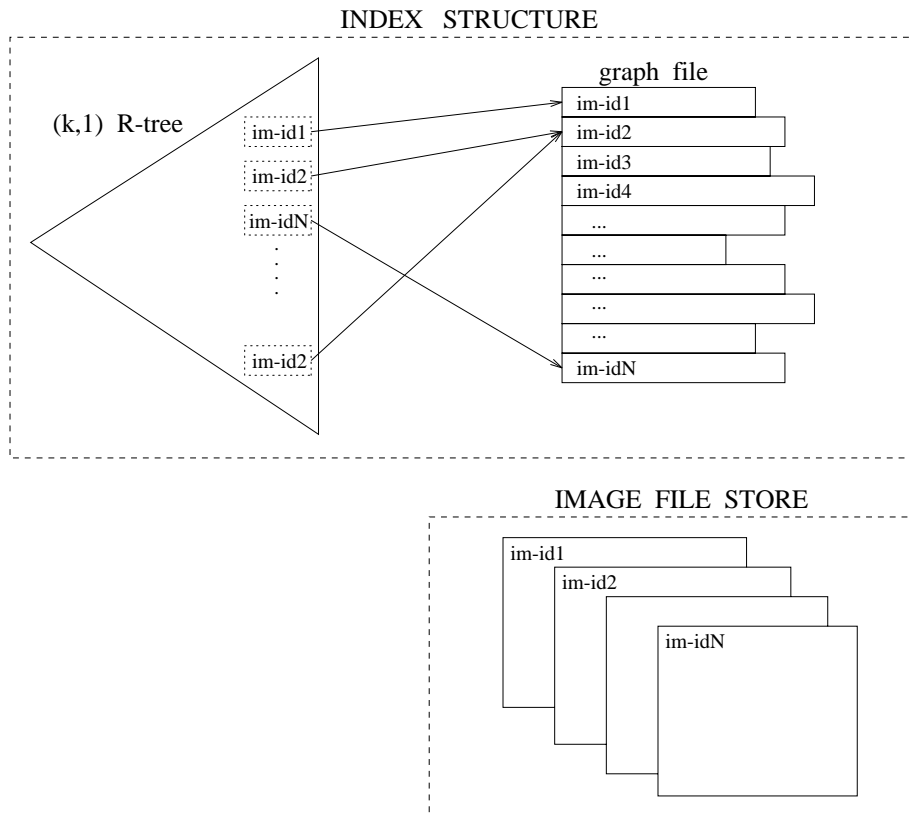


Figure 6: *Proposed file structure.*

- The “graph file”. This is a file holding the ARGs. Each record in this file consists of (a) an identifier (e.g., the image file name) corresponding to the image from which the ARG has been derived and (b) the features of each object together with its relationships with the remaining objects.
- The R-tree data structure holding the vectors computed to all the  $(k, 1)$  sub-images. We call this R-tree “ $(k, 1)$  R-tree”.

The graph file together with the  $(k, 1)$  R-tree form the “index structure”. In the IDB literature [4], the image file store and the index structure are called “physical” and “logical” database respectively. There is a plethora of alternative designs (e.g., R-trees holding vectors for  $(k, 2)$  sub-images). We considered several of them and experimented with them all. The  $(k,1)$  R-tree results in the best search times for a small space overhead and is the one that we mainly focus on next. In [40], we present alternative designs along with implementation details and experiments.

### 4.3 Query Processing

Given a  $(k, u)$  query image and a tolerance  $t$ , we want to retrieve all images that contain a  $(k, u)$  sub-image which matches the query within tolerance  $t$ . As we show soon, the  $(k,1)$  R-tree index does not have false dismissals (i.e., all qualifying images are retrieved). However, it may return false alarms (i.e., not qualifying images) in which case, it returns a superset of the

required images. A post-processing step is required to clean-up the false alarms. The generic search algorithm is as follows:

**R-tree search:** Issue (one or more) range queries on the  $(k,1)$  R-tree, to obtain a list of promising images (image identifiers).

**Clean-up:** For each of the above obtained images, retrieve its corresponding ARG from the graph file and compute the actual distance between this ARG and the ARG of the original  $(k, u)$  query. If the distance is less than the threshold  $t$ , the image is included in the response set.

As mentioned earlier, we decided to use the Euclidean distance ( $p=2$  in Equation 3). However, the proposed indexing method can handle any  $L_p$  metric. For the Euclidean distance, the query region is a (hyper-)sphere; for the city-block distance ( $L_1$ ) it is a diamond; for the  $L_\infty$  it is a square etc. All of the above can be handled by the R-tree (i.e., it replaces the query region by its minimum bounding rectangle and it fetches the points that fall within tolerance  $t$ ).

Next, we distinguish between queries specifying one unlabeled object and queries specifying two or more unlabeled objects.

#### 4.3.1 One Unlabeled Object

A  $(k,1)$  query specifies all labeled objects and one unlabeled. Such a query is mapped to a point in a multidimensional space of 24 dimensions ( $f = 24$ ) and treated as a range query: using the Euclidean distance, we want the points in  $f$ -dimensional space that fall within a (hyper-)sphere of radius  $t$ , where  $t$  is the tolerance. The R-tree is searched and all vectors within radius  $t$  (i.e., those satisfying Equation 4) are retrieved. Feature vectors falling outside the query sphere are excluded from the answer set. Range queries on an R-tree yield neither false alarms nor false dismissals. Therefore, in this specific case of  $(k,1)$  queries there is no need for “clean-up” and the graph file need not be examined.

#### 4.3.2 Two or More Unlabeled Objects

Here consider the case of a  $(k,2)$  query specifying 2 unlabeled objects. We break the query into two  $(k,1)$  query sub-images called “sub-queries”. Then, we can apply either of the following strategies:

**With-Intersection:** Apply both  $(k,1)$  sub-queries to the  $(k,1)$  R-tree with tolerance  $t$ . Intersect their resulting response sets to obtain the set of common image identifiers. The ARGs corresponding to these image identifiers are retrieved from the graph file and matched with the original  $(k, u)$  query to discard the false alarms.

**No-Intersection:** Search the  $(k,1)$  R-tree for the first sub-query with tolerance  $t$ . The second sub-query is ignored. Retrieve the resulting ARGs from the graph file and match them with the original query to discard all possible false alarms.

Both strategies introduce false alarms. The first strategy, attempts to minimize the false alarms, but involves excessive R-tree search and set intersections. The second strategy avoids the R-tree search as much as possible, but it employees an expensive clean-up stage. We can prove that the above strategies will have no false dismissals:

**Lemma 1** *Replacing a  $(k,2)$  query of tolerance  $t$  with two  $(k,1)$  queries of tolerance  $t$  each and intersecting their results will give no false dismissals.*

**Proof:** Let  $Q$  be a  $(k,2)$  query and  $S$  be a qualifying image. We want to prove that the two  $(k,1)$  sub-queries will *each* retrieve image  $S$ . It is equivalent to prove that the vectors computed for the above two  $(k,1)$  sub-queries each qualifies Inequality 4. However, Inequality 4 holds for each vector, since it is derived from that of the original query  $Q$  by omitting some positive terms. This completes the proof of the lemma, for both strategies.  $\square$

The search algorithms for queries with more than 2 unlabeled objects are straightforward extensions of the above ideas.

### 4.3.3 Other Queries

The definition of similarity given in Section 2.1 serves as the basis for more complicated situations. Our proposed scheme can handle cases where the query specifies only a few of the labeled objects (e.g., we don't care for some of them). In these cases, the summation of Equation 3 excludes the features of the unspecified objects (partial match queries). The R-tree index can still handle these queries: The range of values along the unspecified axes stretches from  $-\infty$  to  $+\infty$ .

The proposed method can also handle the case where the user considers some of the properties more important than others. We can give higher weights to these properties. If weights are used with Equation 3, the query specifies an ellipse in the feature space, instead of a sphere. The weights could even be adjusted on-the-fly by the user. Since a weighted Euclidean distance presents no additional indexing problems, we do not consider weights for the rest of this paper.

In certain cases, it is important to find the the most similar image to a given query. Then, instead of searching in a neighborhood of radius  $t$ , we must search for the closest point in the neighborhood of the query. Algorithms for nearest neighbor searching in R-tree do exist [23]. It would be conceptually straightforward to adapt them to our system for  $(k,1)$  queries. For the  $(k,2)$  nearest neighbor queries an obvious solution would be to ask a sequence of range queries with successively increasing radius.

## 5 Experiments

We implemented the system in C, under UNIX. To test the efficiency of our methodology a large data set of segmented tomographic images has to be used. However, large data sets of tomographic images are difficult to be processed manually to derive the desired segmented forms

(i.e., closed contours). In addition, robust automatic segmentation techniques are currently not available in our laboratory.

As a testbed, we used a database consisting of 13500 synthetic segmented images (synthetic workload) <sup>4</sup>. Originally we used 20 MRI images. We segmented these images by manual tracing the contours of labeled objects. To produce a synthetic image from a given original, we allowed the objects in the original to rotate, scale and translate by a certain amount computed by a random number generator (e.g., each object is allowed to rotate between 0 and 20 degrees). To produce various shapes, the contour points of each object were allowed to extend along the line connecting the center of mass of the object with each point. A number of unlabeled objects (at most 5 per image) having random sizes, shapes and positions was then added to the above derived images. Objects were not allowed to intersect with each other. Among the 13500 images we produced, there are 4500 images with 8 objects, 3600 with 7 objects, 2700 with 6, 1800 with 5 and 900 with 4 objects. All images contain 3 labeled objects.

We carried out several groups of experiments based on a  $(k,1)$  R-tree (i.e., holding vectors consisting of all  $k$  labeled objects and one unlabeled). Queries specifying all labeled objects and one unlabeled ( $(k,1)$  queries) are the basic queries and are used to process more complex queries specifying more unlabeled objects. The experiments were designed to:

- Study the performance of the processing of  $(k,1)$  and  $(k,2)$  queries.
- Study the search time when the index is in the main memory, as well as on the disk.
- Illustrate the superiority of the proposed method over sequential scan searching.

Each point in our plots is the average over 45 characteristic queries. The times reported correspond to the elapsed time, computed using the `time` system call of UNIX. These are times for database search; times for image display are not reported, since they are common to both methods. We experimented with both main memory and secondary memory search. In the first case, the system loads as much as it can from the index structure in the main memory. In the second case, each query accesses the index structure on the disk.

We run the experiments on a SUN/470 with 32 Mbytes of main memory. The database has been implemented on a MICROPOLIS disk with average seek time less than 12msecs. The disk was connected to the host computer. Therefore, there are no time delays due to data transfers through a communications network. We run the experiments on a dedicated system (no other user was logged on). In all the experiments, we used 4 Kbytes as the page size for the R-tree. The space required for the implementation of the  $(k,1)$  R-tree is 22.5 Mbytes while, the graph file requires 5.1 Mbytes for storage. In [40] we have also experimented with a  $(k,2)$  R-tree which requires 52.2 Mbytes.

---

<sup>4</sup>The test data we used in the experiments are available from <ftp://ftp.ced.tuc.gr/pub/petrakis>.

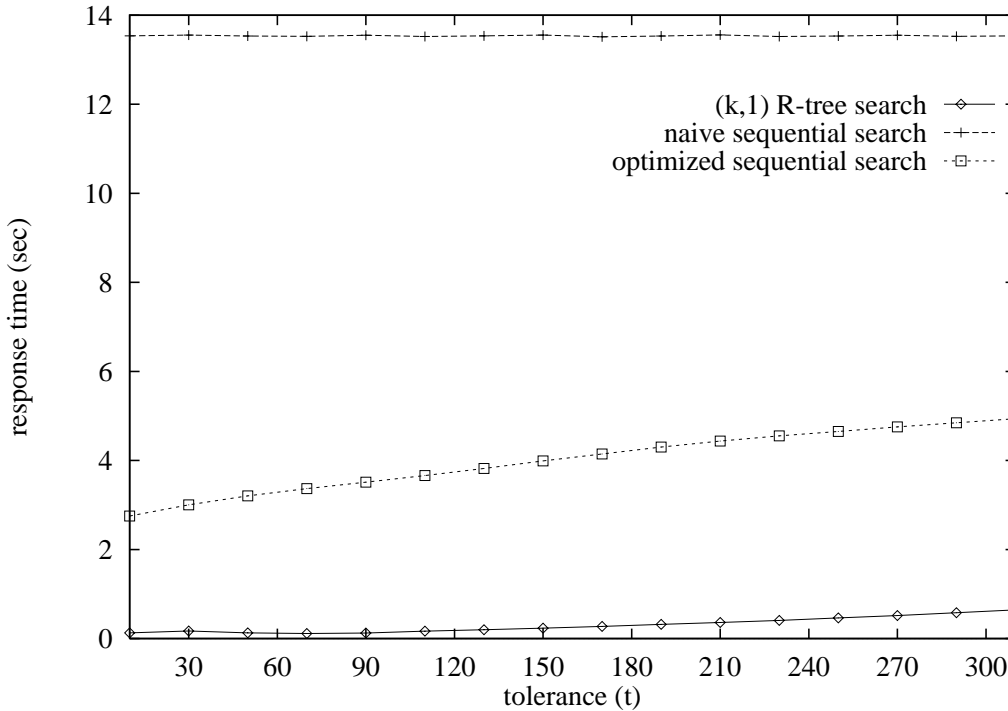


Figure 7: Average retrieval response time as a function of the tolerance ( $t$ ) for  $(k,1)$  queries corresponding to (a) search on a  $(k,1)$  R-tree (b) naive and (c) optimized sequential scan of the database. The index structure is loaded in the main memory.

## 5.1 Queries With One Unlabeled Object

The goal of this set of experiments is to illustrate the performance gains of our method with respect to sequential scanning for the basic  $(k,1)$  queries (i.e., queries specifying all 3 labeled objects and 1 unlabeled).

Figure 7 plots the response time (averaged over 45 queries) for  $(k,1)$  queries. The index structure is loaded in the main memory. Sequential searching is performed in the main memory too. Our method achieves large savings, even for high values of the tolerance (for  $t=300$  the retrieved set is almost 150 images; presumably a user would like to retrieve 20-50 images, in which case,  $t$  must be less than 200). Even for such large queries, the response time of the proposed method is below 2 seconds. Notice that sequential scanning is always above 2 seconds.

Sequential scanning is performed by matching the vector computed to a given query with the vectors produced from each stored ARG having the same size with it. For  $(k,1)$  queries, these are at most 5 (i.e., all images contain between 1 and 5 unlabeled objects). For sequential scanning we performed the following optimization: For every image, the partial sum of the distance function of Equation 3 is continuously compared with the threshold, to achieve early rejections. As the tolerance increases, more and more images delay to be rejected, thus increasing the response time. We call this method “optimized sequential” as opposed to the “naive sequential”



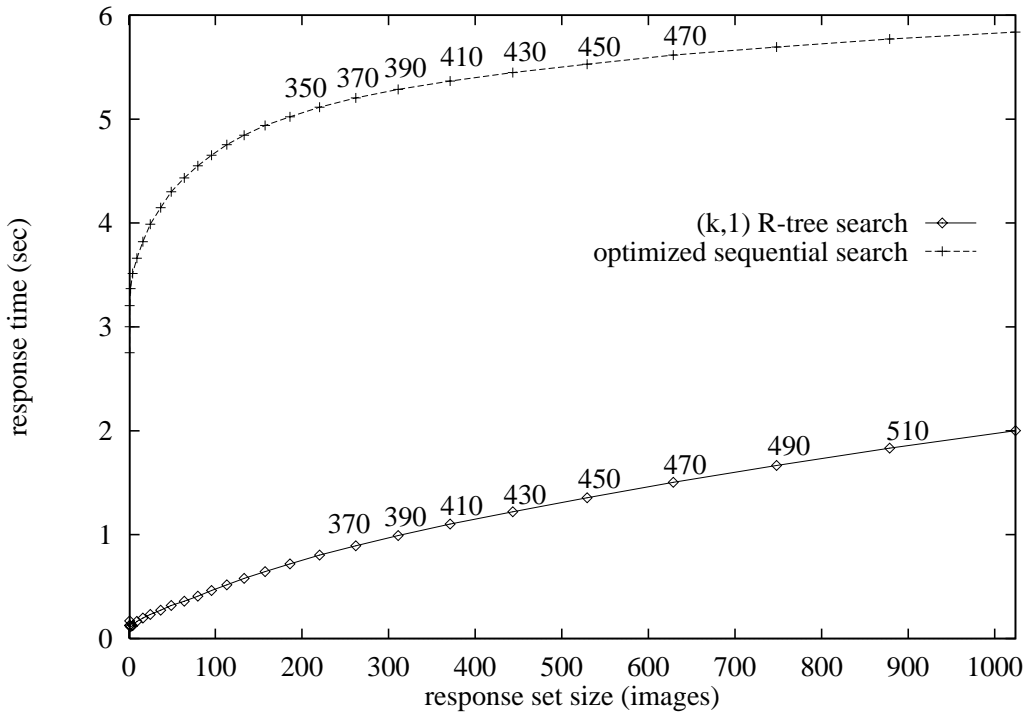


Figure 8: Average retrieval response time as a function of the average retrieval response set size for  $(k,1)$  queries. The labels denote the corresponding values of the tolerance ( $t$ ).

method (i.e., the sum of Equation 3 is computed first and then compared with the tolerance). For the naive sequential scan the response time was almost constant at 13.5 secs. Henceforth, the optimized sequential scan is used in the experiments.

The shape of the curves can be justified as follows: For the optimized sequential scanning, the search time increases with the tolerance because of the optimization we described earlier. For the proposed method, the shape of the curve resembles an exponential curve. This is expected, because the number of qualifying images increases exponentially with the tolerance.

Figure 8 plots the response time as a function of the number of qualifying images (response-set size). The labels indicate the corresponding value of the tolerance. This figure contains almost the same information with Figure 7, showing in addition the response-set size.

Figure 9 plots the response time for  $(k,1)$  queries, when the index structure is on the disk. Sequential searching is performed on the disk, too. As expected, the response times are slower compared to the response times for searches in the main memory (see Figure 7). The system time accounts for the time for fetching pages of the index structure from the disk into the main memory for processing. The number of retrieved pages increases with the tolerance, thus slowing down retrievals. However, the proposed method achieves again large savings over sequential searching, especially for low values of the tolerance  $t$ .

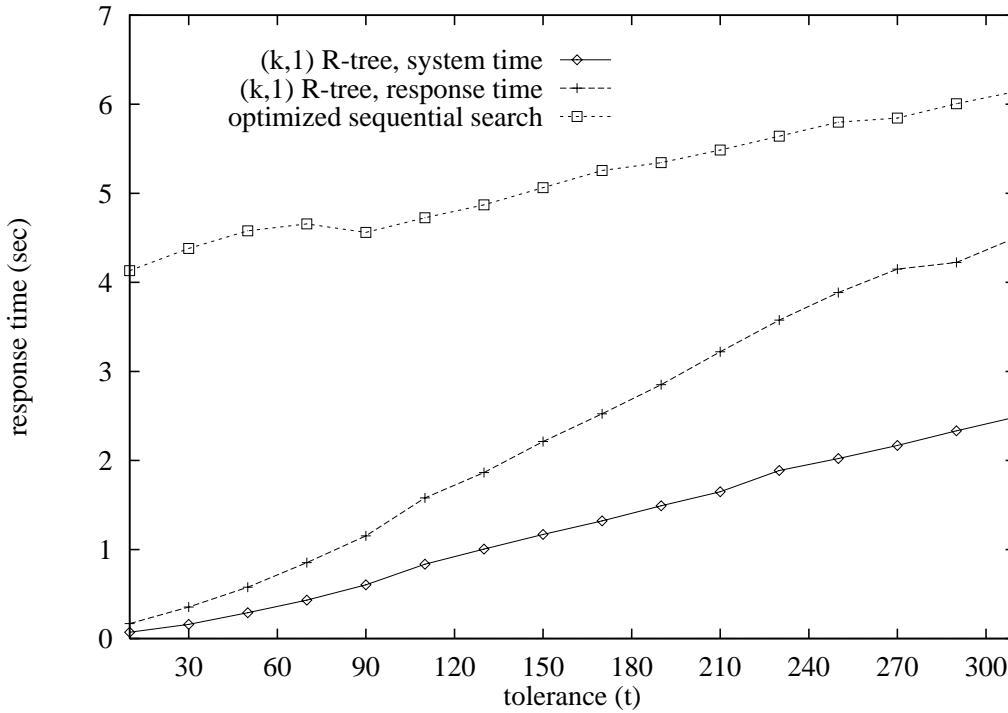


Figure 9: Average retrieval response and system time for  $(k,1)$  queries when the  $(k,1)$  R-tree is stored on the disk. The time for the sequential scan of the database is also shown.

## 5.2 Queries With Two Unlabeled Objects

The next set of experiments examines  $(k,2)$  queries (i.e., queries specifying all 3 labeled objects and 2 unlabeled). Figure 10 plots the average retrieval response time of  $(k,2)$  queries as a function of the tolerance  $t$ . The index structure is loaded in the main memory.

A  $(k,2)$  query breaks into two  $(k,1)$  sub-queries. As explained in Section 4.3, we have two alternative methods to search the  $(k,1)$  R-tree:

**With-Intersection:** Both  $(k,1)$  sub-queries are processed by the R-tree.

**No-Intersection:** Only the first  $(k,1)$  sub-query is processed by the R-tree.

The experimental results show that above two methods are roughly comparable with respect to response time. The recommended method is the second one, since it is faster. The shape of the curves is also expected, resembling curves that are exponential as a function of the tolerance. The sequential scanning was the slowest, even for large values of the tolerance, ranging from 6 seconds to 13 seconds. Its response time is not constant, because of the optimization we described in the case of  $(k,1)$  queries. Sequential searching is performed by matching all the  $(k,2)$  sub-images derived from each stored image with the query. The number of matches which are attempted is  $\binom{u}{2}$ . It increases fast with  $u$  thus slowing down retrievals. The proposed method uses a  $(k,1)$  R-tree and a  $(k,1)$  sub-query to reduce the number of images that are

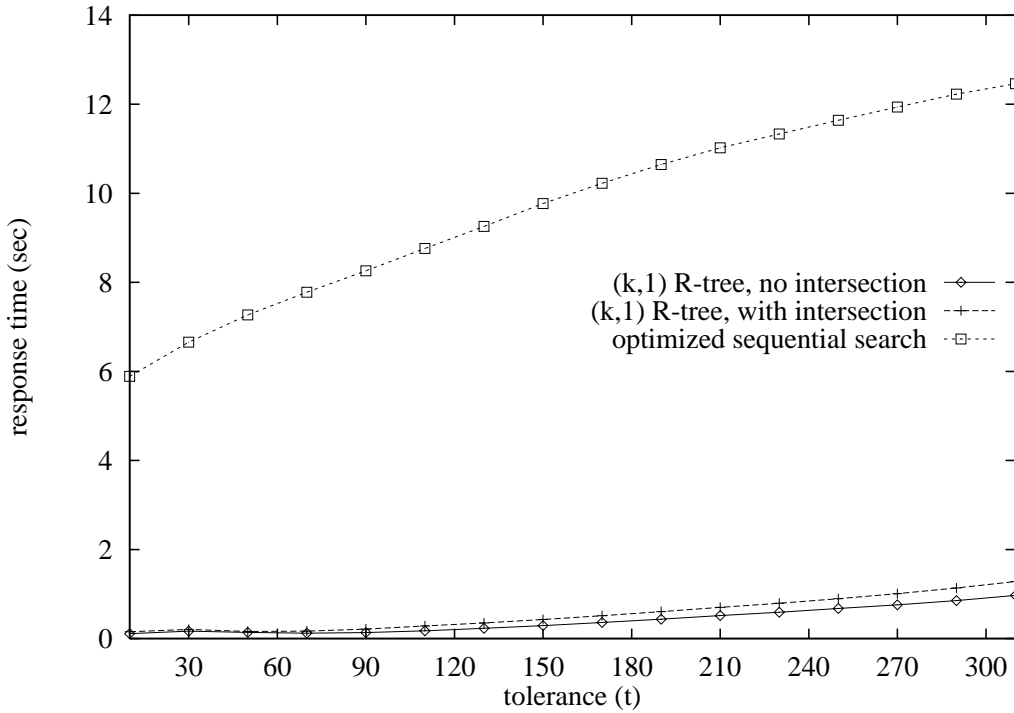


Figure 10: Average retrieval response time as a function of the tolerance ( $t$ ) for  $(k,2)$  queries corresponding to (a) search on a  $(k,1)$  R-tree utilizing one  $(k,1)$  sub-query, (b) utilizing both  $(k,1)$  sub-queries and (c) optimized sequential scan of the database. The index structure is loaded in the main memory.

matched with the original query. Therefore, retrievals will be faster than sequential scanning for any  $u$  ( $u > 1$ ).

Figure 11 plots the response time for  $(k,2)$  queries accessing the index structure on the disk. The conclusions are the same with the above for searching in the main memory. The rest of the experiments concentrate on  $(k,1)$  queries and search in the main memory.

### 5.3 Examples of Retrievals

Figure 12 demonstrates a characteristic example of a  $(k,1)$  query (left) and of a retrieved image (right). The cost of matching their corresponding 24-dimensional vectors is 219.4. Observe that, both labeled and unlabeled objects in the query and the retrieved image respectively, have approximately similar characteristics and similar spatial relationships. The retrieved image contains also one additional unlabeled object (object 4) not specified by the query.

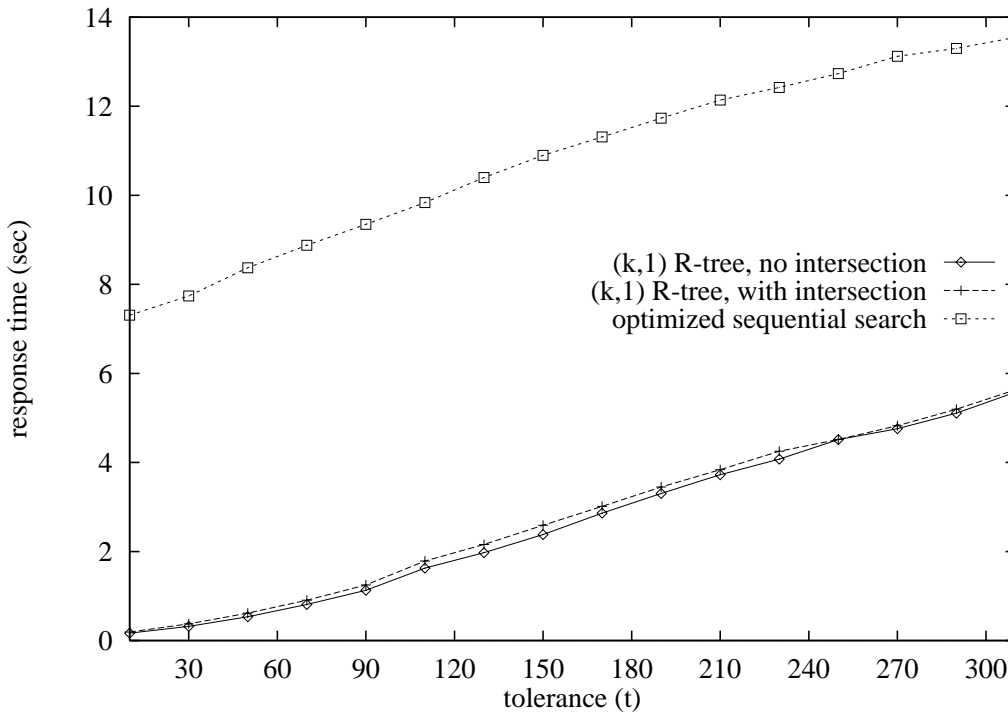


Figure 11: Average retrieval response for  $(k,2)$  queries when the index structure is accessed on the disk. The time for the optimized sequential scan of the database on the disk is also shown.

## 6 Discussion - Observations

There are several interesting observations, as well as a plethora of optimization techniques that can be used. Due to space constraints, we refer to the related technical report [40] (<ftp://ftp.ced.tuc.gr/pub/petrakis/TR-01-94.ps.gz>).

There, we discuss three issues:

- Alternative index designs, comparing  $(k,1)$  R-tree with a  $(k,2)$  one. The conclusion is that the former is typically faster even for  $(k,2)$  queries, requiring smaller space overhead.
- Dimensionality reductions techniques. The experiments showed that, by simply dropping some of the attributes, we may obtain better response time (more false alarms, but a faster R-tree search).
- The “effective” or “fractal” dimension of the resulting set of points. Our measurements showed that it is approximately  $\approx 2.97$ ; this implies that a small number of attributes (around 3-5) might capture most of the necessary information. Thus, a dimensionality reduction technique, such as the Karhunen-Loeve transform, is a very promising research direction.

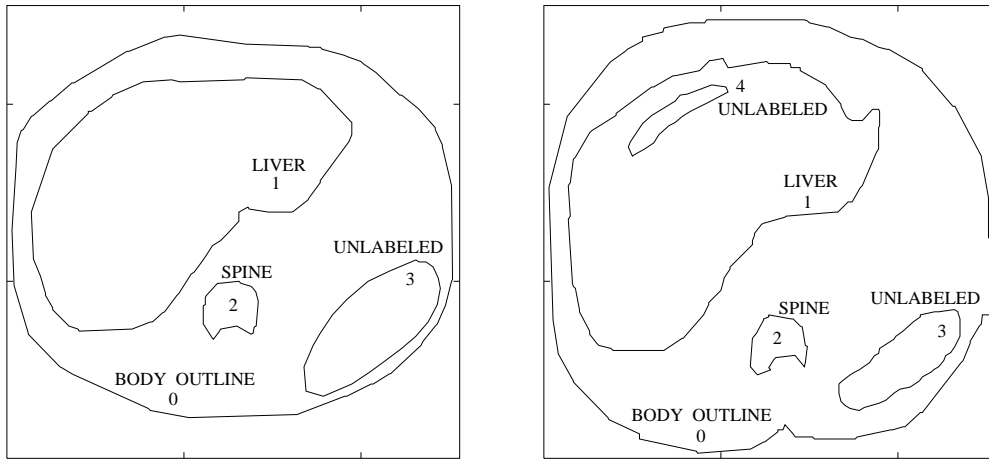


Figure 12: *Example of a query image specifying all 3 labeled and 1 unlabeled object (left) and example of a retrieved image (right).*

## 7 Conclusions

In this paper, we proposed a method to handle approximate searching by image content in image databases. Our approach allows for continuous, quantitative estimates of similarity. Older methods, such as 2-D strings [11], give binary (i.e., “yes/no”) answers, while others have no index support and therefore are slow and not scalable [16]. In addition, image content representation methods based on strings have been proven to be ineffective in capturing image content and may yield inaccurate retrievals.

Attributed relational graphs (ARGs) provide an effective means for image content representation. However, retrievals based on attributed relational graphs are inefficient. This is mostly due to the complexity of search [6]. In addition, search is exhaustive. In this work, we proposed a method for the indexing of stored attributed relational graphs. We make the assumption that certain labeled objects can be identified in all images. This situation is common to images found in many application domains including medicine, remote sensing, microscopy, robotics etc. In this work, we focused our attention on medical images (i.e., tomographic scans of the body).

Our method allows similarity search to be performed on both labeled and unlabeled (i.e., not identified) objects. Indexing is performed by decomposing each input image into sets of objects, called “sub-images”, containing all labeled objects and a fixed number of unlabeled. All sub-images are mapped to points in a multidimensional feature space and are stored in an R-tree. Image database search is then transformed into spatial search. We provide experimental results on a synthetic, but realistic database. The experimental results are a good support to the claims of efficiency. We show that the proposed method outperforms sequential scanning significantly (i.e., up to an order of magnitude) for searching in the main memory or on the disk, never missing a hit (i.e., no false dismissals). The method can accommodate any set of attributes that the domain expert will provide.

Future work includes (a) the examination of dimensionality-reduction techniques such as the k-L transform and their implications on the performance, (b) the examination of alternative indexing structures, such as the the R\*-tree [33], which seem to provide faster search times for smaller space overhead and (c) the extension of this method to work on a parallel machine supporting parallel disk access.

## References

- [1] S. C. Orphanoudakis, C. Chronaki, and S. Kostomanolakis. I2C: A System for the Indexing, Storage and Retrieval of Medical Images by Content. *Journal of Medical Informatics*, 19(2):109–122, 1994.
- [2] Jeffrey R. Bach, Santanu Paul, and Ramesh Jain. A Visual Information Management System for the Interactive Retrieval of Faces. *IEEE Trans. on Knowledge and Data Engineering*, 5(4):619–627, August 1993.
- [3] Dana H. Ballard and Christopher M. Brown. *Computer Vision*. Prentice Hall, 1982.
- [4] Shi-Kuo Chang. *Principles of Pictorial Information Systems Design*. Prentice Hall International Editions, 1989.
- [5] Martin A. Fischler and Robert A. Elschlager. The Representation and Matching of Pictorial Structures. *IEEE Transactions on Computers*, c-22(1):67–92, 1973.
- [6] 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.
- [7] Hideyuki Tamura and Naokazu Yokoya. Image Database Systems: A Survey. *Pattern Recognition*, 17(1):29–49, 1984.
- [8] Shi-Kuo Chang and Arding Hsu. Image Information Systems: Where Do We Go From Where? *IEEE Transactions on Knowledge and Data Engineering*, 4(5):431–442, 1992.
- [9] Ramesh Jain and Wayne Niblack. NSF Workshop on Visual Information Management, February 1992.
- [10] 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.
- [11] 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.
- [12] Euripides G.M. Petrakis and Stelios C. Orphanoudakis. Methodology for the Representation, Indexing and Retrieval of Images by Content. *Image and Vision Computing*, 11(8):504–521, October 1993.

- [13] E. G.M. Petrakis and S. C. Orphanoudakis. A Generalized Approach for Image Indexing and Retrieval Based Upon 2-D Strings. In S.-K. Chang, E. Jungert, and G. Tortora, editors, *Intelligent Image Database Systems - Spatial Reasoning, Image Indexing and Retrieval using Symbolic Projections*. World Scientific Pub. Co., 1995. To be published. Also available as FORTH-ICS/TR-103.
- [14] Suh-Yin Lee and Fang-Jung Hsu. 2D C-String: A New Spatial Knowledge Representation for Image Database Systems. *Pattern Recognition*, 23(10):1077–1087, 1990.
- [15] Suh-Yin Lee, Man-Kwan Shan, and Wei-Pang Yang. Similarity Retrieval of Iconic Image Databases. *Pattern Recognition*, 22(6):675–682, 1989.
- [16] Suh-Yin Lee and Fang-Jung Hsu. Spatial Reasoning and Similarity Retrieval of Images using 2D C-String Knowledge Representation. *Pattern Recognition*, 25(3):305–318, 1992.
- [17] H.V. Jagadish. A retrieval technique for similar shapes. *Proc. ACM SIGMOD Conf.*, pages 208–217, May 1991.
- [18] Hou, Hsu, Liu, and Chiu. A content-based indexing technique using relative geometry features. *SPIE 92*, 1662:59–68, 1992.
- [19] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intell. Inf. Systems*, 3(3/4):231–262, July 1994.
- [20] Jian Kang Wu and Arxot Desai Narasimhalu. Identifying Faces Using Multiple Retrievals. *IEEE Multimedia*, 1(2):20–38, 1994.
- [21] William I. Grosky and Rajiv Mehrotra. Indexed-Based Object Recognition in Pictorial Data Management. *Computer Vision, Graphics and Image Processing*, 52:416–436, 1990.
- [22] F. Rabitti and P. Savino. An Information Retrieval Approach for Image Databases. In *Proc. of the 18th Intern. Conf. on VLDB*, pages 574–584, Vancouver, British Columbia, Canada, August 1992.
- [23] Nick Roussopoulos, Steve Kelley, and F. Vincent. Nearest Neighbor Queries. *Proc. of ACM-SIGMOD*, pages 71–79, May 1995.
- [24] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. Multi-step processing of spatial joins. *ACM SIGMOD*, pages 197–208, May 1994.
- [25] K. Hinrichs and J. Nievergelt. The Grid-File: A Data Structure to Support Proximity Queries on Spatial Objects. Technical Report 54, Institut fur Informatik, ETH, Zurich, July 1983.
- [26] Jack A. Orenstein. Spatial Query Processing in an Object Oriented Database System. In *ACM Proceedings SIGMOD 86*, pages 326–336, Washington, May 1986.

- [27] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. *Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 247–252, March 1989. also available as UMIACS-TR-89-47 and CS-TR-2242.
- [28] H. V. Jagadish. Linear Clustering of Objects with Multiple Attributes. In *Proceedings of ACM SIGMOD*, pages 332–342, Atlantic City, May 1990.
- [29] J.L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *CACM*, 18(9):509–517, September 1975.
- [30] Antonin Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD*, pages 47–57, June 1984.
- [31] N. Roussopoulos and D. Leifker. Direct spatial search on pictorial databases using packed R-trees. *Proc. ACM SIGMOD*, May 1985.
- [32] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. The  $R^+$ -tree: A Dynamic Index for Multidimensional Objects. In *Proceedings of 13th International Conference on VLDB*, pages 507–518, England, September 1987.
- [33] Nobert Beckmann, Hans-Peter Kriegel, Ralf Schneider, 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.
- [34] Ibrahim Kamel and Christos Faloutsos. Hilbert R-tree: An Improved R-tree Using Fractals. In *Proceedings of VLDB Conference*, pages 500–509, Santiago, Chile, September 1994.
- [35] Michael Otterman. Approximate matching with high dimensionality r-trees. M.Sc. scholarly paper, Dept. of Computer Science, Univ. of Maryland, College Park, MD, 1992. supervised by C. Faloutsos.
- [36] Ioannis Kapouleas. Segmentation and Feature Extraction for Magnetic Resonance Brain Image Analysis. In *Proc. of 10th Intern. Conf. on Patt. Recognition.*, pages 583–590, Atlantic City, New Jersey, June 1990.
- [37] Silvana Dellepiane, Giovanni Venturi, and Gianni Vernazza. Model Generation and Model Matching of Real Images by a Fuzzy Approach. *Pattern Recognition*, 25(2):115–137, 1992.
- [38] A.V. Raman, S. Sarkar, and K. L. Boyer. Hypothesizing Structures in Edge-Focused Cerebral Magnetic Images Using Graph-Theoretic Cycle Enumeration. *CVGIP: Image Understanding*, 57(1):81–98, January 1993.
- [39] S. C. Orphanoudakis, E. G. Petrakis, and P. Kofakis. A Medical Image DataBase System for Tomographic Images. In *Proceedings of Computer Assisted Radiology, CAR89*, pages 618–622, Berlin, June 1989.
- [40] Euripides G.M. Petrakis and Christos Faloutsos. Similarity Searching in Medical Image Databases. Technical Report 01, MULTimedia Systems Institute of Crete, Chania, Crete, July 1994.