University of Alberta

A Study of Image Indexing Techniques for Multimedia Database Systems

by

Youping Niu, M. Tamer Özsu, Xiaobo Li
Laboratory for Database Systems Research
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1
{niu,ozsu,li}@cs.ualberta.ca

Technical Report TR 95–19
July 1995

DEPARTMENT OF COMPUTING SCIENCE
The University of Alberta
Edmonton, Alberta, Canada

Abstract

The role of indexing in query optimization is well-understood in the database community. Indexes enable efficient access to a subset of a database. There have been many studies on indexing within the context of relational database management systems (RDBMS). The most popular indexing techniques are based on B-trees [1]. However, these index structures do not apply for images in Multimedia Database Systems (MMDBS). Some novel index structures to retrieve images have been proposed. In this report[1], we discuss some issues concerning these advanced indexing techniques, and report the results.

[1]This research has been supported by a grant from the Canadian Institute for Telecommunication Research (CITR), a Federal Network of Centre of Excellence funded by the Government of Canada.

Contents

List of Figures

1 Introduction

Today's hardware technology enables us to acquire, store, manipulate and transmit various types of storage media such as *image, audio* and *video*. This has stimulated a great interest in *Multimedia Information Systems* (MMIS) in general and *Multimedia Database Systems* (MMDB) in particular.

An important feature to be considered in the design of a MMDBS is content-based retrieval of the multimedia types such as image, audio and video. The scope of the report is limited to retrieval of (still) images from MMDBS.

The effectiveness of still image retrieval in a MMDBS ultimately depends on image content representations, the types of image queries allowed and the search and retrieval strategies. One important consideration in selecting appropriate types of image content representations is their computational efficiency in the search procedures in order to meet the performance requirements of MMDB applications. Other than the traditional types of queries, such as query by commands, query by identifier and range query, most MMDB applications allow queries by example. In this case, the query image or its sketch is provided and the system must analyze it and extract the appropriate features which can be used to compare with the image content representations stored in the database.

There are two main classes of existing approaches for image search and retrieval. One is called *model-driven* approach, where, to determine which image must be retrieved, content representations corresponding to all stored images in database are compared individually with a similar representation extracted from the query image. Thus, retrievals can be inefficient since comparisons often involve time intensive operations. For a small size database, sequential scanning will be acceptable. However, as the database grows in size, sequential scanning due to its linear scale-up becomes prohibitively slow. One of the solutions is to use indexing methods to facilitate the search and retrieval. This set of techniques is known as *data-driven*, in that an index structure on the data is constructed such that matching images can be retrieved by using indexing search procedures.

The mainstream work in the data-driven direction is to convert the image content representation into a set of coordinates for a point in a multi-dimensional space and then to use any standard multi-dimensional index structure, such as R^*-trees [3] and the rest of the R-tree family [6, 9, 21], *linear quadtrees* [16] and *grid-files* [7] etc.. The other approach is to use various *iconic indexes* [17, 14, 15]. The term "index" here is more in the sense of "indexes i, j for a variable a_{ij}" than "index" used in the database. For example, in [17], an image is represented by a symbolic image, which is a 2-dimensional string. An image query can also be specified as a 2-dimensional string. The problem of image information retrieval then becomes a problem of 2-dimensional string matching. The problem with the methods is however, they are not dynamic; they have not yet been integrated with the query processing, insertion, and deletion strategies [51].

In this report, we focus on the multi-dimensional indexing approaches. We discuss some issues concerning these advanced indexing techniques, and report the results. We will talk about the iconic indexing when we point to our research problems.

2 Image representations

The search and retrieval of images by content, using indexing methods, mainly relies on image content representations (the representations of the features chosen) and similarity measures used. Among all the existing image features, *texture*, *color*, *shape* and *spatial* features are often selected, because they have broad, intuitive applicability [40, 41, 42, 50]. One or more features of the above types can be used in a specific application. For example, in a satellite information system, texture and color features may be more important, whereas shape and spatial features are often used in medical image systems. The problem here is that most image feature representations have been addressed with little regard to efficient image search; sequential search is assumed. Not only are their similarity measure functions expensive to compute, but they also preclude efficient implementation using existing index structures.

2.1 Texture features

Texture has been long and deeply studied with the development of image processing and pattern recognition, and is regarded as one of the most important features to identify different images. It also allows discrimination between surfaces and other information such as shape and motion can be estimated based on it. Although no formal definition of texture exists, intuitively some abstract concepts, such as uniformity, density, coarseness, roughness, regularity, direction, frequency and so on, can be reflected by texture features.

Several approaches used in image processing to describe the texture of an image region have been proposed. A set of designed masks to extract texture features was used in [33, 34]. The method in [36] devised the texture spectrum to characterize texture image. A resolution dependent frequency mean diagram representation scheme was developed in [37]. Features from Fourier power spectrum, which are used primarily to detect global periodicity in an image by identifying high-energy, narrow peaks in the spectrum, were developed in [29]. The approaches in [30, 31, 32] used spatial grey tone dependence co-occurrence matrices to represent texture. The random field and Markov models to perform texture classification and segmentation were proposed in [35]. A new texture representation scheme in the form of feature frequency matrices (FFM) was proposed in [38]. A good survey covering the general techniques of texture analysis can be found in [30].

Although there is a wide variety of texture features described in the literature so far, many are not appropriate for MMDBS applications due to their computational complexity or unrealistic assumptions. And there are still not a single model that can be used in general visual data recognition and classification. The methods proposed in [39] were chosen in QBIC project [40], which used the coarseness, contrast, and directionality features to reflect texture features. The multi-resolution simultaneous auto-regressive (MR-SAR) model of texture has also been reported to have better performance [35]. In multi-resolution feature analysis, gross features are extracted first to provide a high level prospective of the global characteristics.

Finer features are retained to allow sufficient fine scales feature characterization.

2.2 Color features

Color is a powerful feature that often simplifies object identification in an image. There are several color models, such as *RGB, YIQ* and *HSI.* Images in the RGB color consist of three independent image planes, one for each primary color, *red, green* and *blue*. The YIQ color model is the standard for color TV broadcast. Here the Y corresponds to luminance, and I and Q are two chromatic components called *inphase*, and *quadrature*, respectively. The HSI model is commonly used to manipulate color images among the models. H (hue) is a color attribute that describes a pure color, whereas S (saturation) gives a measure of the degree to which a pure color is diluted by white light. The H and S components are intimately related to the way in which human beings perceive color. The I (intensity) component is decoupled from the color information in the image.

Normalized colors such as r (red), g (green) and b (blue) values in the HSI model are defined with respect to RGB model by $r = \frac{R}{(R+G+B)}$, $g = \frac{G}{(R+G+B)}$ and $b = \frac{B}{(R+G+B)}$.

The intensity component in the HSI model is defined as

$$I = \frac{1}{3}(R + G + B) \tag{1}$$

and the H and S also can be derived from R, G and B as

$$H = \cos^{-1}(\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-G)(G-B)]^{\frac{1}{2}}} \tag{2}$$

$$S = 1 - \frac{3}{(R+G+B)}[\min(R, G, B)] \tag{3}$$

2.3 Shape and Spatial features

Object shape representation is an important issue in pattern recognition. Shape features are very important in most MMDBS, especially in systems such as some medical image database systems, in which the colors and textures of the objects are similar. However, retrieval by shape similarity is a difficult problem since the meaning of similarity depends on a particular application . Shape features can be represented by Chain coding [43], Fourier descriptors [44], invariant moments [45], auto-regressive models [46], polar signatures [47] and syntactic approaches [48, 4]. Chain codes are used to represent a boundary by a connected sequence of straight-line segments of specified length and direction. When the coefficients of the high frequency components are ignored, the gross essence of a 2-D shape boundary can be captured by a set of a few Fourier descriptors. Moments are by far one of the most popular method used to reduce the description of the shape features, for example, a 2-D shape boundary can be represented as a 1-D function by using moments. In [4], the semantic components of an image are analyzed and recognized. For example, an object in an image is represented by its *type, name* and *pointer* to its physical image. The shape features also can be approximated with arbitrary accuracy by a polygon. A summary on this issue can be found in [49].

Spatial features represent the spatial relationships among objects in an image, such as directional relationships, adjacency, overlap, and containment involving two or more objects. Usually, spatial features are represented in the symbolic string representation, such as 2-D (2-dimensional) strings [50]. Some of the above relationships can be reflected in the string representation. For example, let o_1, o_2 and o_3 denote three domain objects in an image, the 2-D string: $\{o_1 < o_2 = o_3,\ o_1 < o_3 = o_2\}$ can be used to represent an image in which o_1 is in the left to o_2 and o_3, and o_3 and o_2 are above o_1. The problem of content-based retrieval of images then becomes one of 2-D string subsequence matching.

2.4 Image representations and similarity measures

A quantitative or structural description of an object or some other entity of interest in an image is often called a *pattern*. In general, a pattern is formed by one or more features, such as texture, color and shape features etc..

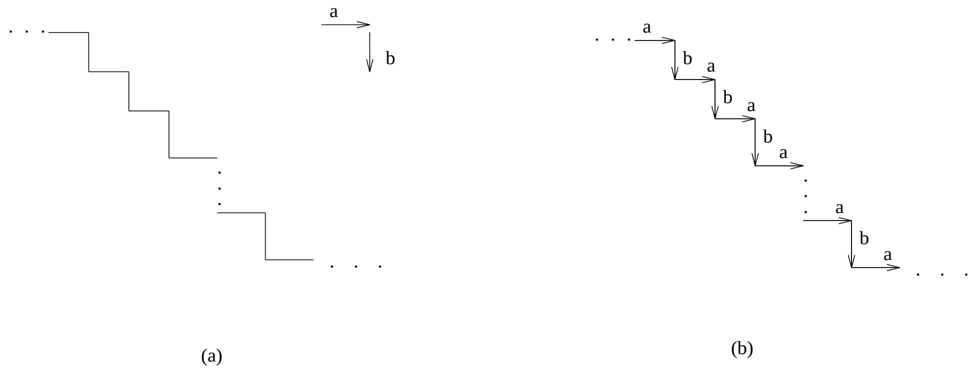


Figure 1: (a) A simple staircase structure; (b) string structure.

In practice, patterns can be represented by *vectors* (for quantitative descriptions) and *strings* and *trees* (for structural descriptions).

Pattern vectors take the form:

$$\vec{X} = (X_0, ..., X_{n-1}) \tag{4}$$

where each component, X_i, represents the $i+1$th feature and n is the total number of such features. The nature of the components of a pattern vector $\vec{X}$ depends on the measurement technique used to describe the physical pattern itself. For example, in the RGB color model, R, G and B are typically in the range 0-255 (or [0, 1]). Thus a vector $\vec{X} = (R_{avg}, G_{avg}, B_{avg})$ can be used to represent the color of an image. Here R_{avg}, G_{avg} and B_{avg} are the average values of n pixels' R, G and B values, respectively.

String representations adequately generate patterns of objects and other entities whose structures is based on relatively simple connectivity of primitives, usually associated with boundary shape. For example, Figure 1(a), after defining the two primitive elements a and b shown, can be coded in the form shown in Figure 1(b). Then the pattern in Figure 1(a) can be represented as the 1-D string symbols $w = ...ababab...$. As stated earlier, 2-D strings also have been used to represent objects and their spatial relationships in an image.

A more powerful approach for many applications is to use tree structures, such as quadtrees and $k-d$ trees, since they are capable of dealing with disjoint structures. Usually, the nodes in a tree structure hold

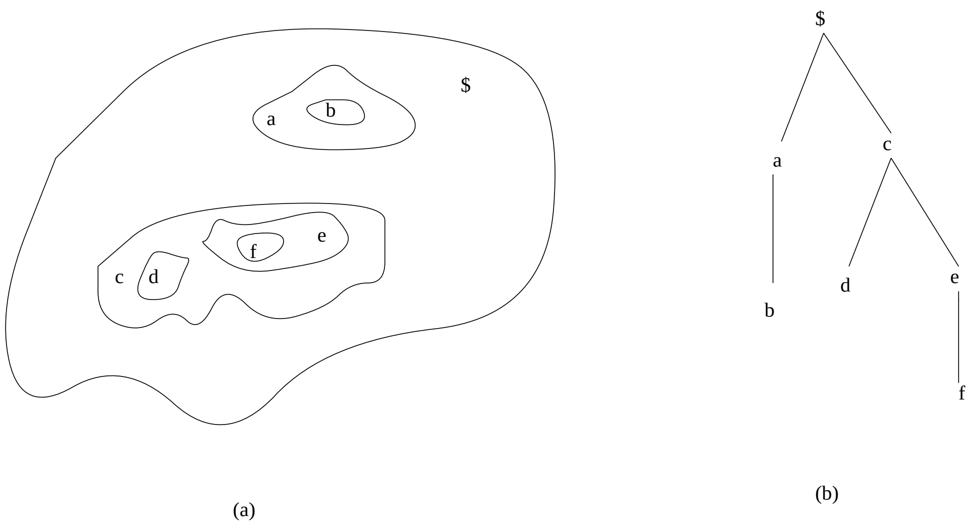


Figure 2: (a) A simple composite region; (b) its tree representation.

two types of information. The first type of information identifies a pattern primitive, whereas the second type defines the physical relationship of the primitive to other substructures. For example, Figure 2(a) can be represented by a simple tree structure in Figure 2(b) by using the relationship "inside of".

Similarity between images is used for storage and retrieval in MMDBS. The *distance* between a pair of image feature representations is commonly used as similarity measure, especially when the images are represented by feature vectors. From indexing points of view, similarity measures are used in two phases. In the first phase, they are used to eliminate unnecessary matchings between images (or objects) and query image (or object). In the second phase, when only a small set of images (or objects) is left, similarity measures are used to provide sufficient discrimination to select resulting images (or objects). Many similarity measures have been proposed, but most of them are not convenient for developing an efficient indexing mechanism due to their quadratic nature and/or time complexity. The commonly used similarity measures are Euclidean and weighted Euclidean distance functions, L_1 and L_2 metrics [52, 40], and 1-D and 2-D string matching procedures [17, 53, 54, 55, 56].

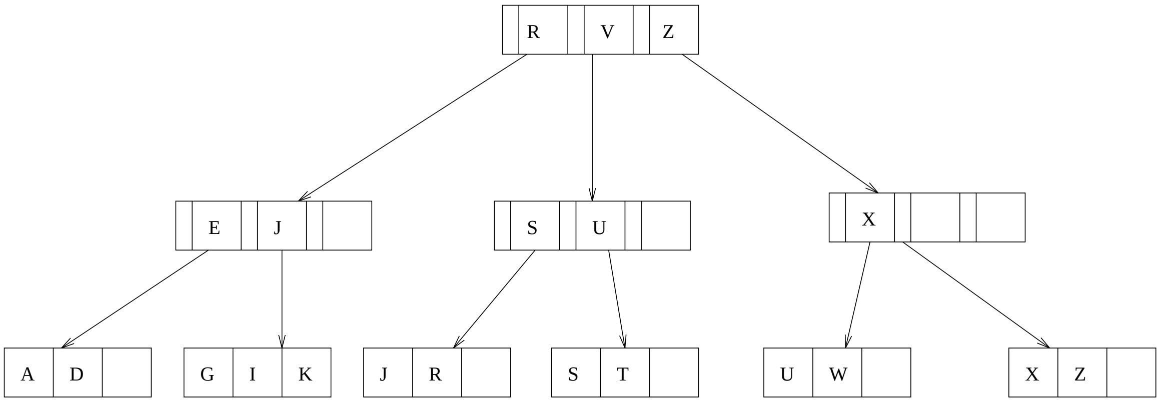


Figure 3: An example of the B^+ tree.

3 Standard database indexing Techniques

The role of indexing in query optimization is well-understood in the database community. Indexes enable efficient access to a subset of a database. There have been many studies on indexing within the context of relational database management systems (RDBMS). The most popular indexing techniques are based on B-trees [1] (particularly B^+ trees [2]). The primary difference between the B^+ tree and the B-tree is that all the key and record information is stored in the leaf-nodes which are linked as a list in an ascendant or descendant order. The B^+ tree is a dynamic indexing structure in terms of insertion and deletion of records and always balanced, that is, the leaf-nodes are at the same level and achieve an average of about 50% node utilization.

Figure 3 shows an example of the B^+ tree. A non-leaf node contains entries of the form (S, p) where S is the separator which is derived from the keys of the records and is used to tell which of subtrees holds the searching key, and p is the pointer to its child node. A leaf node contains entries of the form (key, p) where p is the pointer to the record corresponding to the key. The *fanout* of the B^+ tree is the maximum number of entries in the non-leaf nodes. Thus, the fanout of the B^+ tree in Figure 3 is 3.

Searching a record in a B^+ tree is straightforward. For example, suppose we wish to retrieve the record which has the key T. From the root, we find that $R \leq T \leq V$, which leads to the node which contains S

and U as separators. Next, since $S \leq T \leq V$, the target record could be in the leaf-node which contains the keys of S and T. Searching this leaf-node results in locating T. Usually, only one path from the root to the leaf-node is required for a search.

Deletion of a record from a B^+ tree may cause node underflow (become less than 50% full). Redistribution and concatenation of the records in the underflow nodes help ensure that the 50% full property is maintained. On the other hand, insertion of a record into a B^+ tree may cause node overflow (number of records is more than the fanout). Thus, the overflow nodes have to be split. Splitting of a node results in the need to promote a key to a higher-level node to provide an index separating the two new node.

The key and separator in B^+-trees are usually numbers or 1-D (1-dimensional) strings. They are not likely to provide enough information to deal with complex image data which may contain overlapping due to the limitations of features provided. And a B-tree node does not usually correspond to a n-dimensional rectangle, and hence the representation is not particularly conductive to a range query. Therefore, the traditional indexing structures such as B^+-trees do not apply for images in MMDBS. To overcome this deficiency, several novel multi-dimensional indexing structures to retrieve images have been proposed. For example, *k-d-B-trees* [8] use an adaptive *k-d-tree* partition to dictate the contents of each B-tree node (corresponding to disjoint region). The problem with this technique is that B-tree performance guarantees are no longer valid, it is highly dependent on the sequence of insertions and deletions of records.

4 Multi-dimensional indexing structures

Among all the multi-dimensional indexing structures, linear quadtrees [16], grid-files [7] and the R-tree family [3, 6, 9, 21] are frequently used in MMDBS.

4.1 Quadtrees

Earlier approaches for multi-dimensional indexing are linear quadtrees and grid-files. A 2-dimensional quadtree is a tree whose non-leaf nodes have four (2^2) children and leaf nodes point to regions (or points) The 2-dimensional space is represented as a $2^k \times 2^k$ array of 1×1 squares. A *block* is a $2^m \times 2^m$ $(0 \leq m \leq k)$ square obtained as the result of recursive decomposition of the space into quadrants and sub-quadrants. Each block represents a node in a 2-dimensional quadtree, which contains 2-dimensional points.

The 2-dimensional quadtree is easily extended to represent 3-dimensional data and the resulting data structure is called an *octree*, in which each node is either a leaf or has eight (2^3) children. In general, a n-dimensional quadtree nodes are either leaves or have (2^n) children. Thus, for linear quadtrees, the effort is proportional to the hyper surface (total number of nodes in the trees) of the query region [20]; the hyper surface grows exponentially with the dimensionality.

4.2 Grid files

In a grid file, the data space is partitioned into *grid blocks*, in which the records lie. The *bucket* is the storage unit that contains records. All records in one grid block are stored in the same bucket, however, a bucket can be shared by several grid blocks as long as the union of them forms a n-dimensional rectangle in the space records. The *grid directory* consists of two n-dimensional arrays to maintain a dynamic correspondence between the grid blocks in the records space and the data buckets. The first array contains one entry for each grid block. The values of the entries are pointers to the relevant data buckets. The second array defines a partition of the domain of each attribute. The appropriate grid blocks can be accessed by computing their address based on the value of the relevant attributes. The grid file is attractive because of its graceful growth as more and more records are inserted. However, it faces similar problems as the quadtree does, since the grid directory grows exponentially with dimensionality.

Both linear quadtrees and grid files scale exponentially for high dimensionalities, eventually reducing to sequential scanning. The indexing structures such as quadtrees, grid files, k-d trees and their variants have been studied for many years. An excellent survey on those structures is [13]. Newly developed variants of those schemes are [25, 10, 11].

4.3 R-tree family

The most promising multi-dimensional indexing approaches are R-trees [6] and its variants such as R^*-trees [3], R^+-trees [9] and Packed R-trees [21]. The R-trees are a class of multi-dimensional data structures. They have many graceful properties which make them more attractive than other multi-dimensional data structures. First, the R-tree is a dynamic B-tree like structure which has the self-balancing property, that is; all the leaf nodes appear at the same level. Second, it achieves an average of about 50% node utilization [3]. Third, compared with grid file methods, R-trees store the multi-dimensional rectangles in the original data space (which has lower dimensionality) without transforming them to higher dimensional points. Compared with quadtree methods, R-trees store multi-dimensional rectangles as complete objects and do not need to divide them into several pieces (such as quadtree blocks in quadtrees).

The R-trees are a hierarchical collection of n-dimensional rectangles. Similar to B-trees, the leaf-nodes contain the actual data (or the pointers to the actual data). Each data object is stored together with its minimum enclosing rectangle. Each leaf-node is associated with a bounding rectangle that encloses the smaller rectangles of its children. The grouping into bigger rectangles continues up the tree until at the top a single all encompassing bounding rectangle, the root, is left. A non-leaf node contains entries of the form (p, R) where p is the pointer to its child node and R is the minimum bounding rectangle of all rectangles which are entries in that child node. A leaf node contains entries of the form (oid, R) where oid refers to a record in a database, describing a spatial object, and R is the enclosing rectangle of that spatial object. Figure 4 illustrates data rectangles (in solid line). The dotted rectangles indicate the bounding rectangles in the non-leaf nodes. Figure 5 shows the R-tree with fanout 3 for Figure 4.

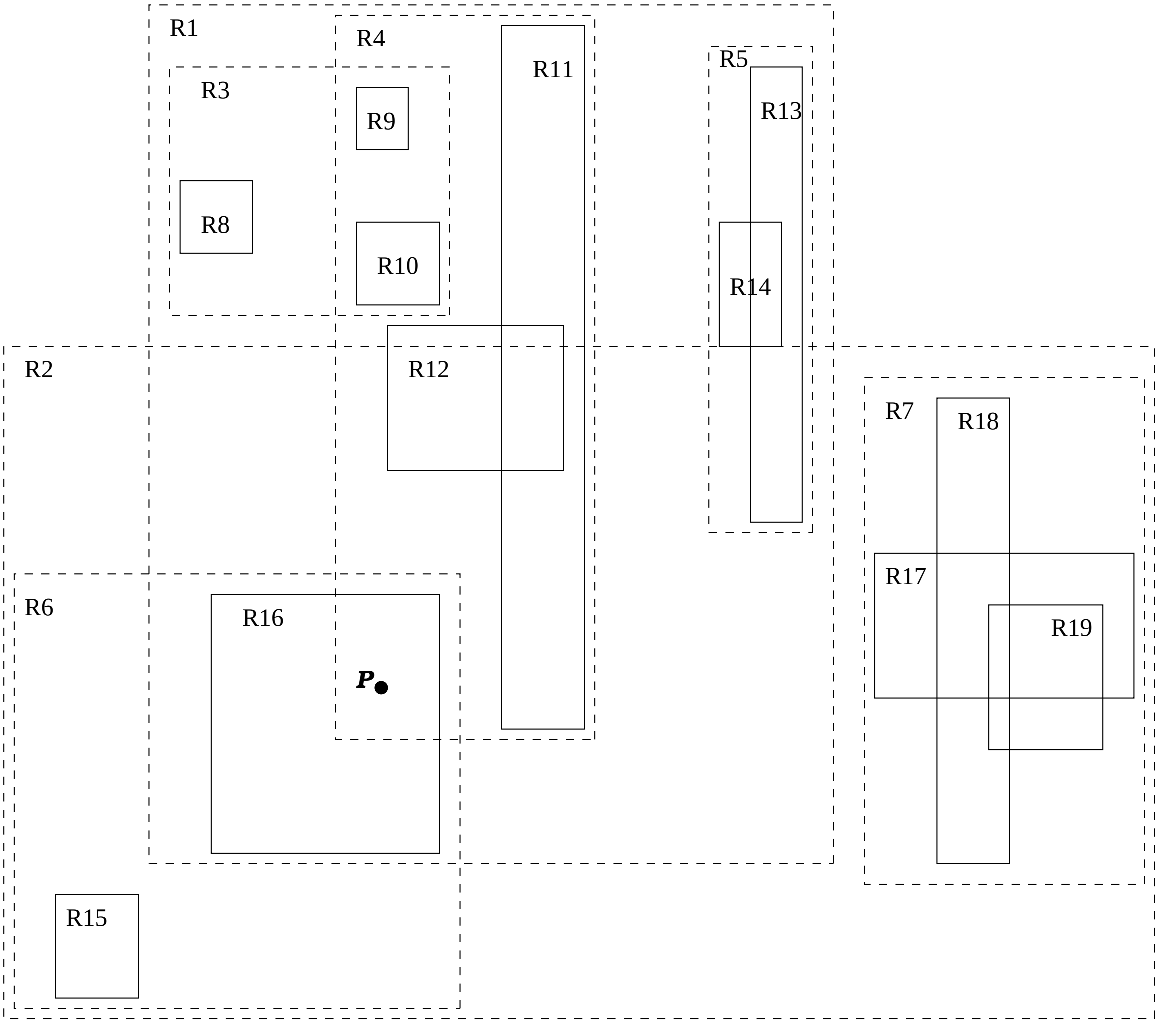


Figure 4: Data rectangles.

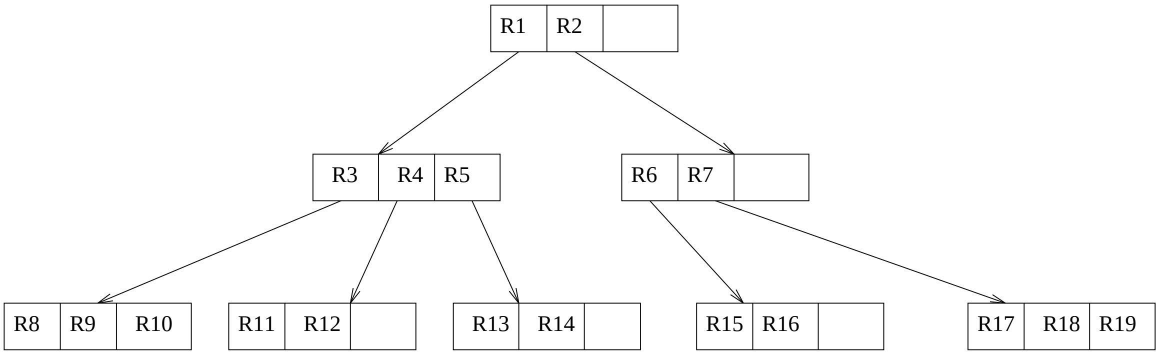


Figure 5: A R-tree for Figure 4.

Two types of queries can be posed against R-trees and other multi-dimensional indexing structures:

1. *range query*: Given a rectangle S, find all rectangles R such that $R \cap S \neq \phi$.

2. *point query*: Given a point P, find all rectangles R such as $P \in R$ (here points can be considered as degenerated rectangles).

Searching for points or rectangles in a R-tree is similar to the procedures for B-tree. Since there are overlaps among rectangles, it is quite possible that the target points or rectangles are contained in more than one bounding rectangle. Therefore, a large number of nodes (bounding rectangles) may have to be examined to retrieve the results. However, the target record is only stored in one leaf-node. This means some searching paths may not lead to the target points or rectangles. For example, we want to search the rectangle in Figure 4 which contains the point P. Since either of R_1 or R_2 can contain P, they may have to be searched. Searching R_1 leads to the rectangle R_4 since P is in it. But none of its leaf-nodes R_{11} and R_{12} contains P. Thus R_2 have to be examined. Searching R_2 lead to R_6 and searching R_6 results in locating R_{16}, the target rectangle.

Insertion of a rectangle into a R-tree may cause splitting of a node. The goal of split is to distribute the records among the nodes to reduce the likelihood that the both nodes will be visited in the subsequent searchs. Three splitting algorithms, the *linear split*, the *quadratic split* and *the exponential split* are proposed in [6]. Their names indicate their complexity. Among the three, the quadratic split algorithm is the one that achieves the best trade-off between splitting time and search performance. All of them are designed to minimize the overlapping area by the two bounding rectangles from a split.

Deletion of a rectangle from a R-tree is similar to the deletion of a record from a B-tree. The only difference is that for a R-tree, in the case of underflow, nodes are reinserted instead of being merged with adjacent nodes since there is no concept of adjacency in a R-tree (in a B-tree, the keys in the leaf-nodes are kept in an ascendant or descendant order). If a rectangle maintains the same parent throughout its lifetime, the R-tree performance could gradually degrade due to the insertions of records. Reinsertion enables the tree to reflect dynamically the changing spatial structure of the data.

The R-tree index allows overlapping in the non-leaf nodes rectangles. Therefore, it can not guarantee that only one search path is required for a search. Due to the insertions and deletions of records, its search performance can gradually degrade. To overcome the deficiencies, a set of new members of R-tree family such as Packed R-trees [21], R^+-trees [9] and R^*-trees [3], are introduced.

If the data (objects) in a database are known beforehand and the database can be expected to be stable, the packed R-tree can be used to build a R-tree. The packed R-tree is built by successively applying a nearest neighbor relation to group records in a node after the set of records has been stored according to a spatial criterion. Empirical tests [21] show that in a database of 2-dimensional points, the packed R-tree leads to significant improvements over a R-tree built using the R-tree insertion routine. But the packed R-tree is a static method in the sense that we must know all of the data points before we can build the tree.

The motivation for the R^+-tree is to avoid overlap among the bounding rectangles, that is all non-leaf bounding rectangles are nonoverlapping. Therefore, each rectangle is associated with all the bounding rectangles that it intersects. This may introduce several paths from the root leading to the same rectangle. Thus the height of the tree will increase. Empirical tests [9] show that R^+-tree outperforms R-tree when there are few large rectangles and several small ones. Thus, R^+-trees may not be robust in terms of data distributions. The main drawback of R^+-trees is that B-tree performance guarantees are no longer valid: nodes are not guaranteed to be half full, and the same record may be stored in different leaf-nodes. Furthermore, it is pointed out in [3, 21] that nonoverlapping may not always be convenient or even possible and the overlapping technique does not imply bad average retrieval performance.

In the R-trees family, the R^*-tree [3] is a relatively new member and seems to perform the best. To enhance the performance of R-trees, the following optimization criteria are proposed:

1. The area covered by a non-leaf R should be minimized.
2. The overlap between non-leaf Rs should be minimized.
3. The margin of a non-leaf R should be minimized.
4. Storage utilization should be optimized.

The R^*-tree splitting algorithms not only test the area parameter (as R-tree does), but also test the area, margin and overlap in different combinations. The main contributions is the idea of "forced reinsert" by deleting some rectangles (30% of the most remote rectangles) from the overflowing node, and reinserting them. The belief behind the idea of "forced reinsert" is that in a R-tree, different sequences of insertions will build up different trees, therefore R-tree suffers from its old entries. Data rectangles inserted during the early growth of the structure may have introduced some bounding rectangles, which are not suitable to guarantee a good retrieval performance in the current situation [3]. The performance comparison shows that R^*-trees clearly outperform R-trees and the storage utilization achieves an average of about 70%, higher than that for R-trees. Furthermore, the average insertion cost of R^*-trees is lower than that of R-trees.

5 Multi-dimensional indexing Methods

To use the multi-dimensional indexing structures mentioned in the previous section, every item (object or image)'s properties of color, texture and shape etc. are extracted and mapped to a *feature vector*, say $\vec{X} = (X_0, ..., X_{n-1})$. For efficiency, these feature vectors are precomputed and stored. Then each feature vector is treated as a *point* in a *n-d* (n-dimensional) space, and a multi-dimensional indexing method is employed. For example, when R^*-trees are used, the $\vec{X}$'s can be considered as degenerated rectangles and stored in the leaf-nodes with their images (or the pointers to their images) together. From the multi-dimensional indexing points of view, two types of queries are classified as follows:

1. *range query*: retrieve all the feature vectors $\vec{X}$ within tolerance ε from the query vector $\vec{Q}$

2. *point query*: retrieve all the feature vectors $\vec{X}$ equal to the query vector $\vec{Q}$

The main distinction between pattern matching and searching in the image database is that while exact pattern matching is the main criterion in pattern recognition, in database retrieval the objective is to help the user navigate through a large number of images. In fact, the users may not even be interested in the best match at all! They might use the retrieved patterns to further modify the search specifications. Thus, the constraints on pattern recognition is somewhat relaxed in database environment, and efficiency in terms of reducing the retrieval time and robustness in terms of not missing targets (allowing some *false hits*, but no *false dismissals*) is a larger concern. Notice that "false hits" are acceptable, since they can be discarded easily through a post-processing step. Furthermore, as we mentioned earlier that most multi-dimensional index structures' retrieval costs will grow exponentially with the dimensionality. Experiments [22, 23] indicate that R-trees and R^*-trees work well when the dimensionality is less than 20. With higher dimensionality, the performance of R-trees and R^*-trees degrade very fast.

Thus, the goal here is to find a mapping of the feature vector $\vec{X}$ into a vector $\vec{X}' = f(\vec{X})$, where $\vec{X}'$ is a vector in a more suitable space, with a *distance function* or *domain expert* D' which will underestimate the actual distance:

$$D'(\vec{X}', \vec{Q}') \leq D(\vec{X}, \vec{Q}) \tag{5}$$

The standard distance preserving transformation, such as the *Karhunen Loeve* (KL), *Discrete Fourier* (DFT), or *Discrete Cosine* (DCT) Transforms can be used as the function f, which map n-dimensional feature vectors into another form of n-dimensional vectors. The advantage of the transformations is that the transformed vectors will have most of the *energy* in the first few coefficients for indexing. This results in a lower dimensional index and reduces the retrieval costs (exponentially). Of course, we have to pay the price of a small number of false hits.

Lemma 1 *If equation 5 holds, it will guarantee that a range query will not miss any actual hit (no false dismissals) [19].*

Consider the typical example of a range query: *retrieve all the feature vectors $\vec{X}$ within tolerance ε from the query vector $\vec{Q}$.* It will first operate on the lower-dimensional space, retrieving all $\vec{X}'$ such that $D'(\vec{X}', \vec{Q}') \leq \varepsilon$. Therefore, if a vector $\vec{X}$ satisfies $D(\vec{X}, \vec{Q}) \leq \varepsilon$, then by equation 5, it is guaranteed to qualify for filtering step $D'(\vec{X}', \vec{Q}') \leq D(\vec{X}, \vec{Q}) \leq \varepsilon$.

Suppose that we use DFT as the distance preserving transformation here.

The n-point Discrete Fourier Transform of a vector $\vec{X} = (X_0, ..., X_{n-1})$ is defined to be a vector $\vec{X}' = (X_0', ..., X_{n-1}')$, given by

$$X_u' = 1/\sqrt{n} \sum_{k=0}^{n-1} X_k exp(-j2\pi uk/n) \tag{6}$$

for $u = 0, 1, ..., n-1$. The complex coefficients X_u' are called the *Fourier descriptors.* The inverse Discrete Fourier Transform of the X_u''s restore $\vec{X}$. That is,

$$X_k = 1/\sqrt{n} \sum_{u=0}^{n-1} X_u' exp(j2\pi uk/n) \tag{7}$$

for $k = 0, 1, ..., n-1$. Suppose, however, that instead of all the X_u''s, only the first m coefficients are used. This is equivalent to setting $X_u' = 0$ for $u > m-1$ in equation 7. The result is the following approximation to $\vec{X}$:

$$\hat{X}_k = 1/\sqrt{n} \sum_{u=0}^{m-1} X_u' exp(j2\pi uk/n) \tag{8}$$

for $k = 0, 1, ..., n-1$. The energy of a vector $E(\vec{X})$ is defined to be

$$E(\vec{X}) = \sum_{k=0}^{n-1} |X_k|^2 \tag{9}$$

where $|X_k|$ is called the *Fourier spectrum* and defined as

$$|X_k| = \sqrt{a^2 + b^2} \tag{10}$$

Notice that X_k is a complex number, given by $X_k = a + jb$, where j is the imaginary unit $j = \sqrt{-1}$ and a and b are the real and imaginary components of X_k, respectively.

The correctness of the approach is guaranteed by *Parseval's* theorem [18]:

Theorem 1 *Let $\vec{X}'$ be the Discrete Fourier Transform of the vector $\vec{X}$. Then we have*

$$E(\vec{X}) = E(\vec{X}') \tag{11}$$

The *Euclidean distance* D between two vectors $\vec{X}$ and $\vec{Y}$ is defined as:

$$D(\vec{X}, \vec{Y}) = \sqrt{\sum_{k=0}^{n-1} (X_k - Y_k)^2} \tag{12}$$

Parseval's theorem implies that the DFT also preserves the Euclidean distance between two vectors [19].

Lemma 2 *Let $\vec{X}'$ and $\vec{Y}'$ be the Discrete Fourier Transform of $\vec{X}$ and $\vec{Y}$ respectively. Then we have*

$$D(\vec{X}, \vec{Y}) = D(\vec{X}', \vec{Y}') \tag{13}$$

Then if we can keep the first few coefficients of the DFT as the features, it will underestimate the actual distance of two vectors (because the positive terms from equation 12 are ignored) and thus it will introduce no false dismissals, according to **Lemma 1**.

It is important to choose an appropriate number m of coefficients to be retained. A larger m reduces the false hits but at the same time increases the dimensionality of the multi-dimensional index, and hence the search time. On the other hand, a smaller m reduces the search time but at the same may increases the false hits, and hence the post-processing time.

Let's use an example to illustrate how the average color vector in the RGB color model, a 3-dimensional vector $\vec{X} = (R_{avg}, G_{avg}, B_{avg})$, can be projected on the 2-dimensional space (x-y plane). We use the Euclidean distance as the distance function D between two average RGB vectors, that is, the sum of squared differences. Suppose that $\vec{X} = (100, 60, 30)$, according to equation 6, we have

$$X_0' = 1/\sqrt{3} \sum_{k=0}^{2} X_k e^0$$

$$
\begin{aligned}
&= 1/\sqrt{3}(100+60+30) \\
&\simeq 109.7
\end{aligned}
$$

$$
\begin{aligned}
X_1' &= 1/\sqrt{3}\sum_{k=0}^{2} X_k e^{-j2\pi k/3} \\
&= 1/\sqrt{3}(100*e^0 + 60*e^{-j2\pi/3} + 30*e^{-j4\pi/3}) \\
&= 1/\sqrt{3}(100 + 60*(\cos(-2\pi/3) + j\sin(-2\pi/3)) + 30*(\cos(-4\pi/3) + j\sin(-4\pi/3))) \\
&\simeq 66.4 - 45j
\end{aligned}
$$

Continuing with this procedure gives $X_2' \simeq 49.1 - 15j$.

If we keep the first two features (the Fourier spectrum of the first and second coefficients), then $\vec{X} = (100, 60, 30)$ can be mapped into $\vec{X}' = (109.7, \sqrt{66.4^2 + 45^2}) = (109.7, 80.2)$.

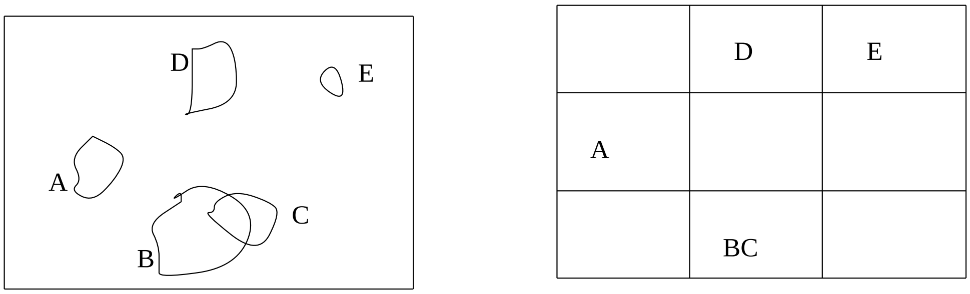
6 Conclusions

Among all the multi-dimensional indexing approaches, R-tree family (R^*-trees in particular), linear quadtrees, and grid-files are most promising. However, they all suffer the so called "dimensionality curse". Both linear quadtrees and grid-files scale exponentially for high dimensionalities, eventually reducing to sequential scanning. R^*-trees, on the other hand, can be efficient if the fanout of their nodes remains greater than 2 and the dimensionality stays under 20 [22, 23]. There are basicly three limitations of using these indexing methods. First, Euclidean distance of the points in the feature space is assumed. However, this is not necessarily the case. In reality, most likely, there may be *cross-talk* among the features, resulting in a full quadratic form of distance function involving all cross terms [26]. Not only is such a function much more expensive to compute than a Euclidean distance, but it also precludes efficient implementation of these indexing methods. Second, the queries to a database may use combination of different set of features, say color, texture and shape features, which usually have very high dimensionalities. It may introduce too many false hits by simply applying the method (reducing dimensionality) in section 5. Finally, these indexing schemes in general do not maintain the neighborhood information between points within a partition, especially the partition's boundary points, thus, are not suitable for nearest neighbor search.

The basic requirement on indexing in the MMDBS is that it provide sufficient discrimination to prevent the retrieval of a large fraction of the database, and not that it produce only the exact match. Therefore, the constraints on pattern recognition (where exact pattern matching is the main criterion) can be somewhat relaxed in the MMDBS environment. For example, in [57], only a few of image features are used first to conduct similarity matching and the additional features are used only when the additional discriminatory power is absolutely necessary. We can expect to achieve better performance in MMDBS by combining image processing with database search in an intelligent way [12, 4, 5, 19, 57].

As we discussed above, putting the combination of different set of features into one vector can lead to very high dimensionalities and make it difficult to use the commonly used multi-dimensional indexing methods. One way to solve the problem is to keep a separate feature vector for each set of features and use separate R^*-trees to index the features. However, this may cause great storage overhead and search inefficiency since all the images have to be stored several times (in different R^*-trees) and all the R^*-trees may have to be visited for the retrieval of a query which uses combination of different set of features. And of course, the update costs will go high. Thus, it will be very useful to develop an integrated indexing structure which can keep all the features separate, but support an efficient search.

Finally, it is possible to employ parallel computing techniques to enhance the performance of the multi-dimensional indexing schemes. As pointed out in [27], the problem here is how to achieve the performance in that the small queries will only activate small number of the processors (disks), and meanwhile the large queries should engage all processors (disks).



(a) (b)

Figure 6: (a) An image ; (b) the symbolic image of (a).

6.1 Our research problems

The capability of conducting queries on the objects in images and their spatial relationships becomes an important issue of MMDBS. As proposed in [17], after preprocessing by image processing and pattern recognition techniques, the objects in an original image are segmented and labelled as A, B, C .... The spatial relationship between two objects is denoted by a set of relations, $\{=, <, :\}$. A symbolic image of the original image then can be represented by a 2-D string according to its projection along the x- and y-axis, respectively. For example, the five objects in the original image in Figure 6(a) can be represented by a symbolic image in Figure 6(b). The 2-D string representation of the symbolic image is:

$$\{A < B : C = D < E, B : C < A < D = E\} \tag{14}$$

The 2-D string, containing symbols from the symbolic image and representing spatial relations in the original image, is called *iconic indexing*. The problem of content-based retrieval of images then becomes one of 2-D string subsequence matching. The 2-D string approach supports spatial reasoning, flexible information retrieval, and visualization. A typical query by using 2-D string can be "Find all images containing a dog next to a man".

Newly proposed string representations are 2-D G-string [56], 2-D C-string [54], 2-D B-string [53] and

$RCOS$-string [55]. The problem with the approaches, however, is that the 2-D string subsequence matching is done sequentially. That is, the query image, represented by a 2-D string , is matched against all 2-D strings (representations) of images in the database. This is not desirable when the database is reasonably large. To avoid exhaustive searching through the entire database, the 2-D strings should be organized into classes. An object-oriented approach can be used. Different indexing methods should be implemented to eliminate unnecessary subsequences as much as possible. Our research will focus on the 2-D string organization and the corresponding indexing schemes.

References

[1] R. Bayer and E. McCreight, Organization and maintenance of lager ordered indexes, *Acta Informatica*, 1(3): 173-189, 1972.

[2] D. Comer, The Ubiquitous B-tree, *ACM Computing Surveys*, 11(2), June 1979.

[3] N. Beckmann, H.P. Keiegel, R. Schneider, and B. Seeger, The R^*-tree: An Efficient and Robust Access Method for Points and Rectangles, *Proc. ACM SIGMOD int'l Conf. on the Management of Data*, 322-331, 1990.

[4] S-K. Chang and S-H. Liu, Picture Indexing and Abstraction Techniques for Pictorial Database, *IEEE Trans, on Pattern Analysis and Machine intelligence*, 6(4), 475-483, 1984.

[5] W.I. Grosky, P. Neo, and R. Mehrotra, A Pictorial Index Mechanism for Model Based Matching, *Proc. 5th IEEE int'l Conf. on Data Engineering*, 180-187, 1989.

[6] A. Guttman, R trees: A Dynamic Index Structure for Spatial Searching, *Proc. ACM SIGMOD int'l Conf. on the Management of Data*, 47-57, 1984.

[7] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, The Grid file: An Adaptable Symmetric Multikey File Structure, *ACM Trans. on Database Systems*, 9(1), 1984.

[8] J. T. Robinson, *K-d-B*-trees: A Search Structure for Large Multidimensional Dynamic Indices, *Proc. ACM SIGMOD int'l Conf. on the Management of Data*, 1981.

[9] T. Sellis, N. Roussopoulos, and C. Faloutsos, The R^+ Tree: A Dynamic Index for Multidimensional Objects, *Proc. 13th int'l Conf. on Very Large Database*, 507-518, 1987.

[10] B.Seeger and H. P. Kriegel, The Buddy Tree: An Efficient and Robust Access Method for Spatial Database Systems, *Proc. ACM SIGMOD int'l Conf. on the Management of Data*, 590-601, 1990.

[11] D. B. Lomet and B. Salzberg, A Robust Multi-Attribute Search Structure, *Proc. Fifth IEEE int'l Conf. on Data Engineering*, 296-304, 1989.

[12] H. V. Jagadish, A retrieval technique for similar shapes, *Proc. ACM SIGMOD int'l Conf. on the Management of Data*, 208-217, 1991.

[13] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addision-Wesley, 1989.

[14] W. I. Grosky, P. Neo, and R. Mehrotra, An Iconic Index for Model-Based Matching, *Proc. Fifth IEEE int'l Conf. on Data Engineering*, 180-187, 1989.

[15] R. Mehrotra and W. I. Grosky, Shape Matching Utilizing Indexed hypotheses Generation and Testing, *IEEE Trans. Robotics and Automation*, 5(1), 70-77, 1989.

[16] I. Gargantini, An effective way to represent quadtrees, *Communications of the ACM*, 25(12), 905-910, 1982.

[17] S. K. Chang, Q. Y. Shi, and C. W. Yan, Iconic Indexing by $2D$ Strings, *IEEE Trans. Pattern Analysis and Machine Intelligence*, 9(3), 413-428, 1987.

[18] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, 1975.

[19] C. Faloutsos, M. Ranganathan and Y. Manolopoulos, Fast Subsequence Matching in Time-Series Databases, CS-TR-3190, Dept. of Computer Science, Univ. of Maryland, December 1993.

[20] G. M. Hunter and K. Steiglitz, Operations on Images using quad trees. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 1(2), 145-153, 1979.

[21] N. Roussopoulos and D. Leifker, Direct spatial search on pictorial database using packed R-trees, *Proc. ACM SIGMOD int'l Conf. on the Management of Data*, 17-31, 1985.

[22] M. Otterman, *Approximate matching with high dimensionality R-trees*, M.Sc. scholarly paper, Dept. of Computer Science, Univ. of Maryland, College Park, MD, 1992.

[23] A. D. Alexandrov, W. Y. Ma, A. EL Abbadi and B. S. Manjunath, Adaptive Filtering And Indexing For Image Database, In *Storage and Retrieval for Image and Video Database III*, v(2420), 12-23, SPIE, 1995.

[24] H. Zhang and D. Zhong, A scheme for visual feature based image indexing, In *Storage and Retrieval for Image and Video Database III*, v(2420), 36-46, SPIE, 1995.

[25] M. Freeston, The BANG file: a new kind of grid file, *Proc. ACM SIGMOD int'l Conf. on the Management of Data*, 260-269, 1987.

[26] M. Miyahara and Y. Yoshida, Mathematical transform of (r, g, b) color data to munsell (h, v, c) color data, In *Visual Communication and Image Processing*, v(1001), 650-657, SPIE, 1988.

[27] I. Kamel and C. Faloutsos, Parallel R-trees, CS-TR-2820, Dept. of Computer Science, Univ. of Maryland, 1992.

[28] T. Y. Young and K. S. Fu, Eds, *Handbook of Pattern recognition and Image processing*, Academic Press, New York, 1986.

[29] J. O. Eklundh, On the use of Fourier phase features for texture discrimination, *Comput. Graphics Image Process.*, 9, 199-201, 1979.

[30] R. M. Haralick, K. Shanmugam and I. Dinstein, Textural features for Image classification, *IEEE Trans. Syst. Man Cybern.* SMC-3(6), 610-621, 1973.

[31] R. W. Conners and C. A. Harlow, A theoretical comparison of texture algorithms, *IEEE Trans. Pattern Analysis Mach. Intell.*, PAMI-2(3), 204-222, 1980.

[32] M. Unser, Sum and difference histograms for texture classification, *IEEE Trans. Pattern Analysis Mach. Intell.*, PAMI-8(1), 118-125, 1986.

[33] K. I. Laws, Texture energy measures, *Proc. Image Understanding Workshop*, 47-51, 1979.

[34] M. Pietikainen, A. Rosenfeld and L. S. Davis, local pattern matches, *IEEE Trans. Syst. Man Cybern.* SMC-13(3), 421-426, 1983.

[35] J. Mao and A. K. Jain, Texture classification and segmentation using multi-resolution simultaneous auto-regressive models, *Pattern Recognition* 25, 173-188, 1992.

[36] D. C. He and L. Wang, Unsupervised texture classification of images using the texture spectrum, *Pattern Recognition* 25, 247-255, 1992.

[37] H. C. Shen and A. K. C. Wong, Generalized texture representation and metric, *Comput. Graphics Image Process.*, 23, 187-206, 1983.

[38] H. C. Shen and C. Y. C. Bie, Feature frequency matrices as texture representation, *Pattern Recognition Lett.*, 13, 195-205, 1992

[39] H. Tamura, S. Mori and T. Yamawaki, Texture features corresponding to visual perception, *IEEE Trans. Syst. Man Cybern.* SMC-8(6), 460-473, 1978.

[40] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic and P. Yanker, The QBIC project: Ouerying images by content using color, texture, and shape, In *Storage and Retrieval for Image and Video Database*, v(1908), 173-187, SPIE, 1993.

[41] E. Binaghi, I. Gagliardi and R. Schettini, Indexing and fuzzy logic-based retrieval of color images, *IFIP Visual Database Systems II*, Elsevier Science Publishers B. V., 79-92, 1992.

[42] T. Hermes, C. Klauck and J. Zhang, Image retrieval for Information systems, In *Storage and Retrieval for Image and Video Database III*, v(2420), 394-405, SPIE, 1995.

[43] H. Freeman and L. S. Davis, A corner-finding algorithm for chain-coded curves, *IEEE Trans. Comput.*, 26, 297-303, 1977.

[44] E. Persoon and K. S. Fu, Shape discrimination using Fourier descriptors, *IEEE Trans. Syst. Man Cybern.* SMC-8, 170-179, 1977.

[45] M. F. Zakaria, L. J. Vroomen, P. J. A. Zsombor-Murray and J. M. H. M. Kessel, Fast algorithm for the computation of moment invariants, *Pattern Recognition* 20, 639-643, 1987.

[46] S. R. Dubois and F. H. Glanz, An autoregressive model approach to two-dimensional shape classification. *IEEE Trans. Pattern Anal, Mach. Intell.*, 8, 55-66, 1986.

[47] W. N. Lie and Y. C. Chen, Shape representation and matching using polar signature, *Proc. Intl. Comput. Symp. 1986*, 710-718, 1986.

[48] H.H. Chen and J. S. Su, A syntactic approach to shape recognition, *Proc. Intl. Comput. Symp. 1986*, 103-122, 1986.

[49] J. S. Wu and J. J. Leou, New polygonal approximation schemes for object shape representations, *Pattern Recognition* 26, 471-484, 1993.

[50] S. K. Chang, C. W. Yan, D. C. Dimitroff and T. Arndt, An intelligent image database system, *IEEE Trans. Software Engineering*, 14(5), 681-688, 1988.

[51] W.I. Grosky and R. Mehrotra, Image database management, *IEEE Computer*, 22(12), 7-8, 1989.

[52] M. J. Swain and D. H. Ballard, Color indexing, *Intern. Journal of Computer Vision*, 7(1): 11-32, 1991.

[53] S. Y. Lee, M. C. Yang and J. W. Chen, 2D B-string: a spatial knowledge representation for image database systems, *Proc. ICSC'92 Second Int. Computer Sci. Conf.*, 609-615, 1992.

[54] S. Y. Lee and F. J. Hsu, 2D C-string: a new spatial knowledge representation for image database systems, *Pattern Recognition* 23(10), 1027-1034, 1990.

[55] C. Chang and C. Lee, Relative coordinates oriented symbolic string for spatial relationship retrieval, *Pattern Recognition* 28(4), 563-570, 1995.

[56] S. K. Chang, E. Jungert and Y. Li, Representation and retrieval of symbolic pictures using generalized 2D strings, *Visual Commun. and Image Process.*, 1360-1372, SPIE, 1989.

[57] K. I. Lin, H. V. Jagadish and C. Faloutsos, The TV-Tree: An index structure for High-Dimensional Data, *VLDB Journal*, 3, 517-542, 1994.