

Indexing by Shape of Image Databases Based on Extended Grid Files

Carlo Combi, Gian Luca Foresti, Massimo Franceschet, Angelo Montanari
Department of Mathematics and Computer Science, University of Udine
Via delle Scienze 206, 33100 Udine, Italy
{combi,foresti,francesc,montana}@dimi.uniud.it

Abstract

In this paper, we propose an original indexing by shape of image databases based on extended grid files. We first introduce a recently developed shape description method and tailor it to obtain suitable representation structures for image databases. Then, in order to efficiently support image retrieval, we define an indexing structure based on grid files. Since grid files were originally developed to speed up point (exact match) and range (nearest neighbors within a threshold) queries on multidimensional data with a fixed number of attributes, we extend them to cope with data provided with a varying number of attributes and to deal with a new class of queries relevant to image databases, namely, nearest neighbor queries. We give a detailed description of the proposed search algorithms and a systematic analysis of their complexity, and discuss the outcomes of some experimental tests on sample image databases.

1. Introduction

Digital image databases are convenient media for representing and storing information in a variety of domains, including industrial, biomedical, and public administration domains, provided that an efficient automatic procedure for indexing and retrieving images from them is given. Most image databases take a text-based approach to indexing and retrieval, according to which keywords, captions, or free text are associated with each image in the database. However, such an approach suffers from several limitations. First, since automatic extraction of semantic information from images is beyond the capabilities of current machine vision techniques, a human interaction is required to describe the contents of images in terms of keywords and/or captions. This process is quite time-consuming and extremely subjective. Second, certain visual properties of images, such as some textures and shapes, are very difficult or nearly impossible to describe with text. As an alternative, one can work with descriptions based on properties which

are inherent to the visual contents of an image: colors, textures, shape, and location of image objects, spatial relationships between objects, etc. Most work in image database retrieval has concentrated on a single feature. Color and texture are reliable features only for retrieving generic images (landscapes, outdoor views, etc.); on the contrary, shape is very useful to represent objects, but it requires a large number of attributes. One of the most promising shape-based approaches exploits morphological skeletons to represent images [2]. It encodes a picture as a set of object skeletons; a sketch picture can then be partially reconstructed and progressively refined back to the original image by dilating the skeleton function. This approach allows one to reduce ambiguity in similarity retrieval and to reduce memory requirements, but it has two major drawbacks: the morphological skeleton is not robust to noise [3] and it requires a huge amount of attributes to represent object shapes. A solution to the first problem has been obtained by introducing the notion of Statistical Morphological Skeleton (SMS) [1]. The second problem is addressed in this paper. Whenever we want to efficiently store data provided with several attributes, each of them being possibly treated as a primary key, multidimensional data structures are needed. Most techniques for storing and searching these structures hierarchically decompose the multidimensional space into regions. Such a space can be either the data to be stored or the embedding space from which the data is drawn. Grid files adopt the latter solution. They are especially well-suited when the domains over which the attributes take their values are large and linearly ordered, and the attributes are independent. Under such assumptions, grid files guarantee a high data storage utilization, a smooth adaptation to the contents to be stored, fast access to individual records, and efficient processing of range queries [4].

The goal of our work is to provide a compact representation and efficient indexing structures for image databases. In Section 2, we present a method for SMS extraction and approximation. In Section 3, we first propose an efficient indexing technique, based on a suitable extension of grid files; then, we describe the algorithms we developed for

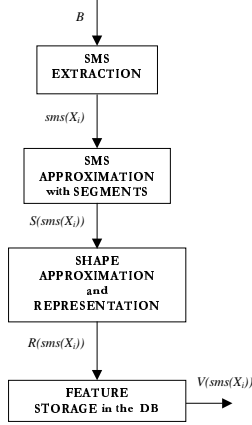


Figure 1. General scheme of the method for SMS extraction and approximation

range and nearest neighbor queries, analyze their complexity, and discuss some experimental results. In Section 4, we report on a concrete application of extended grid files to indexing by shape an image database. In the conclusions, we point out some advantages and limitations of the proposed solution.

2. Shape Extraction and Approximation

In this section, we show how the SMS can be tailored to obtain suitable representation structures for image databases. In Figure 1, the general scheme of the method for SMS extraction and approximation is given.

The method consists of three main steps: (a) extraction of the Statistical Morphological Skeleton; (b) geometrical approximation of skeleton points by means of a set of segments; (c) approximation of the shape function associated with each segment by means of parametric splines. Figure 2 shows a running example image containing two different shapes: a rectangle and a sort of oval.

A binary image B , where each blob X_i represents a possible object (Figure 2a), is provided as input to a module which extracts its SMS $sms(X_i)$ [1]. The SMS can be described as:

$$sms(X_i) = \{(x, y, n(x, y)) : (x, y) \in B\}$$

where the function $n(x, y)$ associates with each skeleton point the iteration at which the point itself has been detected. Figure 2b shows the SMS extracted from the two binary objects in Figure 2a. The SMS of an object X_i is provided as input to the approximation module, whose goal is to efficiently represent information contained in the SMS itself. More precisely, each $sms(X_i)$ is approximated through the set $S(sms(X_i)) = \{s_j : j = 1, \dots, N_i\}$ of segments which best fits with the (x, y) points of $sms(X_i)$. Figure 2c shows the segment-based approximation of the

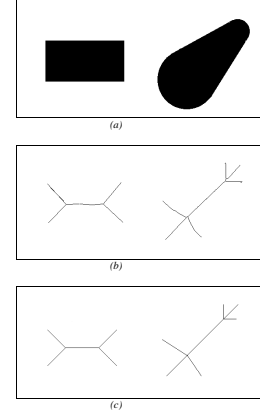


Figure 2. Example image with two shapes (a), the corresponding SMSs (b), and the approximated SMSs (c)

SMS in Figure 2b. The shape approximation and representation module pairs shape information on the skeleton with points of the segments in $S(sms(X_i))$. In such a way, each segment s_j is associated with a function $g_j(\cdot)$ representing the behaviour of the shape on the segment. Then, each function $g_j(\cdot)$ is interpolated by means of Hermite splines. A vector α_j of L coefficients describes the spline of the j -th segment. The coefficients of the interpolating functions are associated with each segment to generate a shape-representation $R(sms(X_i))$ of the object X_i :

$$R(sms(X_i)) = \{ \langle s_j, \alpha_j \rangle : s_j \in S(sms(X_i)) \},$$

$$\alpha_j = [\alpha_{j,u} : u = 1, \dots, L], j = 1, \dots, N_i \}$$

Finally, each object X_i is represented by means of a multidimensional vector $V(sms(X_i))$ composed by a variable number of elements, i.e.,

$$V(sms(X_i)) = [N_i, s_1, \alpha_1, \dots, s_{N_i}, \alpha_{N_i}],$$

where each segment s_j is represented by means of the vector $[x_j^i, y_j^i, x_j^f, y_j^f]$ of its extremes on the image plane. Such a vector is stored in the database as a record, whose attributes correspond to the above detailed elements of the vector. To efficiently support accesses to these records, we developed suitable multidimensional indexing techniques, which are described in the following section.

3. Using Grid Files for Indexing by Shape

In this section, we first introduce the basic concepts of grid files; then, we extend them to cope with multidimensional data provided with a varying number of attributes and to deal with nearest neighbor queries.

3.1. Basics on Grid Files

The basic idea of grid files is that of dividing the search space, where each record is a point, in a grid [4]. The

record space S is partitioned into blocks, called *grid regions*. When records are inserted or deleted, the corresponding grid partition can possibly be modified, by splitting an interval on a space axis or by merging two adjacent intervals [4]. The fixed-size physical storage unit is called *cluster*: there is a cluster in correspondence with each grid region, while it is possible that several grid regions are associated with the same cluster. The *grid directory* is the data structure providing the correspondence between grid regions and clusters: for a k -dimensional search space, a grid directory consists of a dynamic k -dimensional array, called *grid-array*, and k one-dimensional arrays, called *linear scales*. The elements of the grid array are pointers to clusters, and are in correspondence with the grid regions of the grid partition. Linear scales, one for each dimension of the space, contain the bounds of the partition on the corresponding axis. The efficiency of grid arrays is based on the assumption that the linear scales are relatively small and can be maintained in the main memory; the grid array is usually quite large and it is stored on the disk. In this situation, point queries are performed with only two disk-accesses [4].

3.2. Extended Search Algorithms

In this section we propose an extension of grid files with algorithms for range and nearest neighbor queries. We consider a k -dimensional *search space* $S = \times_{i=1}^k D_i$, where the domains D_i are linearly ordered. A record $r \in \{1, 2, \dots, k\} \times S$ on S is characterized by k *attributes* and by an integer which denotes the *valid dimension* of the record, that is, the actual number of attributes that identify the record. If $r = (v, x_1, x_2, \dots, x_k)$, then $x_{v+i} = \text{null}$, for every $i \in \{1, 2, \dots, k - v\}$. Moreover, a distance function on S is chosen. Records with different valid dimensions generally identify different objects and are not comparable.

3.2.1 Nearest Neighbors within a Threshold Problem

The *nearest neighbors within a threshold (NNWT)* problem can be described as follows: given a record r with valid dimension v and threshold d , retrieve the records comparable with r (i.e. with valid dimension v) that belong to the sphere centered at r with radius d .

The idea of the algorithm is quite simple: it accesses, starting from the cluster containing r and moving off in spiral, all the clusters intersecting the sphere with center r and radius d . In order to terminate the search, the algorithm computes the number of steps to be performed before starting the computation. In Figure 3 we present the pseudo-code implementing a procedure `Neighbors` for the NNWT problem.

Before starting the search, the procedure `Neighbors` invokes the subprocedure `Steps` in order to compute the

```
Neighbors(Value, ValidDim, Limit)

A = dynamic array of pointers
Clusters, L = linked list

s <-- Steps(Value, Limit)
for i <-- 1 to s do
  A[i] <-- Adjacents(A, i, Value, ValidDim, Limit)
  p <-- A[i]
  while p \= NIL do
    c <-- RetrieveCluster(key[p])
    if ListSearch(Clusters, c) = NIL then
      ListInsert(Clusters, c)
      ScanCluster(c, Value, ValidDim, Limit, L)
    endif
    p <-- next[p]
  endwhile
endfor
OrderRecords(L, Value)
return L
```

Figure 3. The NNWT algorithm

number s of steps to be performed, taking advantage of the grid file structure, the center and the radius of the searching sphere. After that, the procedure `Neighbors` enters a *for* loop to iterate s steps. At each iteration of the loop, the procedure computes the adjacent regions, with respect to the regions computed at the previous step, that intersect the search sphere (this is performed by the subprocedure `Adjacents`), and it scans the relative clusters searching for records with valid dimension `ValidDim` and distance from `Value` less than or equal to `Limit` (this is done by the subprocedure `ScanCluster`). Before accessing any cluster, the procedure checks if the cluster to visit has been already processed, taking advantage of the list `Clusters` of visited clusters. At the end of the *for* loop, the procedure orders the retrieved records in increasing order with respect to the distance from `Value` (subprocedure `OrderRecords`).

3.2.2 Nearest Neighbor Problem

The *nearest neighbor (NN)* problem can be described as follows: given a record r with valid dimension v , retrieve the record nearest to r with the same valid dimension.

The sketch of the algorithm is as follows: given a record r with valid dimension v , the algorithm finds one neighbor r' of r with valid dimension v ; then, it computes the actual distance d from r to r' and issues a new query centered at r with radius d .

In Figure 4 we present the pseudo-code implementing a procedure `NearestNeighbor` for the NN problem. The procedure `NearestNeighbor` relies on the main subprocedure `FON`, a variant of the previously proposed algorithm for the search of the nearest neighbors within a threshold. This procedure returns *one* neighbor of `Value` with valid dimension `ValidDim`, searching in the current sphere. It uses the new functions `Intersects` (which tests intersection of a region with a sphere) and `ScanN` (which retrieves

```

NearestNeighbor(Value, ValidDim)

A = dynamic array of pointers
Clusters, L = linked list

Limit <-- + inf ; r <-- 1 ; p <-- NIL ; x <-- NIL
s <-- Steps(Value,Limit)
while r <= s do
  (x,r,p) <--
  FON(Value,ValidDim,Limit,Clusters,A,r,p,s)
  if x \= NIL then
    Limit <-- Distance(Value,x)
    s <-- Steps(Value,Limit)
  endif
endwhile
return (x, Limit)

FON(Value,ValidDim,Limit,Clusters,A,r,p,s)

Found <-- False
while (not Found) and (r <= s) do
  if (p = NIL) then
    A[r] <-- Adjacents(A,r,Value,ValidDim,Limit)
    p <-- A[r]
  endif
  while (not Found) and (p \= NIL) do
    if Intersects(key[p],Value,Limit) then
      c <-- RetrieveCluster(key[p])
      if ListSearch(Clusters,c) = NIL then
        ListInsert(Clusters,c)
        t <-- ScanN(c,Value,ValidDim,Limit)
        if t \= NIL then
          Found <-- true
          x <-- t
        endif
      endif
    endif
    p <-- next[p]
    if (p = NIL) then r <-- r + 1
  endwhile
endwhile
return (x,r,p)

```

Figure 4. The NN algorithm

the nearest record within a given cluster).

3.2.3 Complexity Analysis

In this section, we determine the worst-case and average-case complexity of the procedures `Neighbors` and `NearestNeighbors`, in case of uniform distribution of the data, by taking the number of disk accesses as the relevant complexity parameter. Let r be a record (with valid dimension v) on a k -dimensional space S and Z be a sphere centered at r with radius d . We denote by K_Z (resp. L_Z) the number of grid regions (resp. clusters) that intersect the sphere Z . The number of disk accesses performed by the procedure `Neighbors` is $K_Z + L_Z$: the subprocedure `RetrieveClusters` makes one disk access (to the grid directory) for every region intersecting the sphere Z , while the subprocedure `ScanCluster` makes one disk access (to the data) for every cluster intersecting the sphere Z . Note that L_Z can range from 1 (best case, the sphere Z is entirely contained in one cluster) to K_Z (worst case, every region has its own cluster). Hence, in the worst case, the number of disk accesses is $2K_Z$. It is possible to show that $K_Z = b_k \cdot (2s - 1)^k$, where s is the number of steps com-

puted by the subprocedure `Steps` and $0 < b_k \leq 1$. Hence, the worst-case complexity of the procedure `Neighbors` is $2K_Z = 2b_k D_s = 2b_k (2s - 1)^k$. Since the dimension k of the search space is fixed and $b_k \leq 1$, the complexity is $\mathcal{O}(s^k)$, and hence depends on the radius and the density of records of the search sphere, and on the capacity of the clusters.

As for the average-case complexity, $L_Z \approx K_Z/t$, where t is the average number of regions per cluster. As shown in [4], for uniform distributions of the data, independent attributes and capacity $c \geq 10$, the number t is quite stable and it varies from 2 to 4. Let u be the average number of records per cluster (cluster occupancy) and $R_{W(Z)}$ be the average number of records in the smallest rectangle $W(Z)$, built on using the grid structure, containing the sphere Z . It is immediate to see that $L_Z \approx R_{W(Z)}/u$. As pointed out in [4], for uniform distributions, the average occupancy u approximates to $c \cdot \ln 2$. Moreover, $R_{W(Z)} \approx (V_{W(Z)}/V_S) \cdot n$, where $V_{W(Z)}$ (resp. V_S) is the volume of $W(Z)$ (resp. S), and n is the number of records in the database. Hence, the average complexity of the procedure `Neighbors`, in case of uniform distribution of the data, is

$$K_Z + L_Z \approx (1 + t) \cdot L_Z \approx (1 + t) \cdot \frac{V_{W(Z)} \cdot n}{V_S \cdot c \cdot \ln 2}$$

Let us now bound the complexity of the `NearestNeighbor` procedure. We denote with $C(r, d)$ the complexity (in the worst- or average-case) of the procedure `Neighbors` with query point r and search sphere centered at r with radius d . Moreover, we denote with $\hat{C}(r)$ the complexity (in the worst- or average-case) of the procedure `NearestNeighbor` with query point r . We have that $C(r, d') \leq \hat{C}(r) \leq C(r, d'')$, where d' is the distance between r and its *nearest* neighbor and d'' is the distance between r and its *first* neighbor, i.e. the first neighbor that the procedure `FON` finds.

3.3. Experimental Results

We evaluated the algorithms on records having a different number of attributes and having random uncorrelated attribute values. We executed these tests on a database of more than 6000 records, each record having, in different tests, an increasing number of attributes. Each search test has been repeated 100 times with randomly generated values, to obtain sound mean values as result.

As for the time needed to perform the search, we can distinguish two different situations by fixing, respectively, the number of space dimensions and the threshold value: (i) given the number of space dimensions, NNWT performances decrease with increasing threshold values; (ii) given a threshold value, NNWT performances decrease when the

number of space dimensions increases. The reason of such a behaviour is that, for high threshold values or for spaces with a high number of dimensions, NNWT must perform heavy computations to find the path for accessing the suitable grid regions. Similar tests on the NN algorithm have shown that NN performances are always better than the sequential search (the algorithm we used for comparison) ones: differences in search time decrease with the increase of the number of space dimensions.

Image databases are usually characterized by a huge number of clusters and, when shape is used to represent image objects, by a high-dimensional search space. It is worth noting that in high-dimensional spaces, image features allow one to precisely discriminate among different objects and thus limited threshold values are sufficient. This allows us to conclude that extended grid files are suitable structures for indexing image databases.

4. An application to image databases

The application image database was created by scanning a large number of 2D geometric figures. For each class of figures, e.g., squares, rectangles, circles, etc., several images have been considered by traslating, scaling and rotatimg a reference image.

During pre-processing, the feature vector, $V(sm.s(X_i))$, was computed for each image and stored in the database instead of the digitized images themselves. The dimensions of the feature vector vary according to the object class, i.e., the number of segments needed to approximate the SMS. For a given object class whose skeleton approximation requires N_i segments, the feature vector will be composed by $N_i \cdot (L + 4)$ components. In particular, the number N_i of segments depends on the complexity of the object shape (it is proportional to the number of sides of the polygon which approximates the shape) and the parameter L is strictly dependent on the accuracy required to represent the object itself. In our application, the parameter N_i ranges from 2 to 16 and the parameter L has been fixed to 4. We have studied the behaviour of our image retrieval system, on the basis of the typical performance parameters: (i) precision ($P = m_T / (m_F + m_T)$), (ii) recall ($R = m_T / M_k$), and (iii) goodness ($G = (m_T - m_F) / M_k$), where m_T (resp. m_F) is the number of correct (resp. false) objects returned by the query and M_k the number of correct objects really existing in the database. As an example, let us consider the retrieval of the images which are similar to the query image in Figure 5a. The query image is contained into the database. Figure 5b shows the corresponding skeleton, and Figure 5c shows the retrieved images. The left image is exactly the query image, while the right image is the most similar. The query has obtained the optimum result: $m_T = 2$, $m_F = 0$, and $M_k = 2$, hence $P = R = G = 1$.

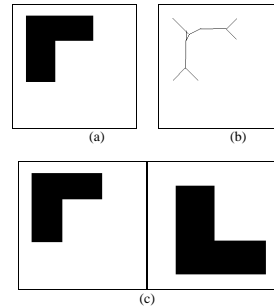


Figure 5. NN search for an image in the database

5. Conclusions

In this paper, we have proposed an extension of grid files for indexing image databases. Grid files are a good choice for indexing multidimensional databases with a fixed number of attributes, whenever point and range queries are needed. Since many image databases need to support point, range and nearest neighbors queries on data with a varying number of attributes, we proposed a suitable extension of grid files and checked it by means of some experimental tests. These tests have shown that the proposed NN and NNWT algorithms are suitable for image databases with a varying number of attributes, provided that the number of clusters is large and the threshold values (NNWT algorithm) are low.

References

- [1] G.L. Foresti, C.S. Regazzoni, and A.N. Venet-sanopoulos. Coding of Noisy Binary Images by Using Statistical Morphological Skeleton. *IEEE Workshop on Non Linear Signal Processing*, Cyprus, 354–359, 1995.
- [2] P. W. Huang and Y.R. Jean. Reasoning about Pictures and Similarity Retrieval for Image Information Systems Based on sk-Set Knowledge Representation. *Pattern Recognition*, 28(12): 1915–1925, 1995.
- [3] P. Maragos and R.W. Schafer. Morphological Skeleton Representation and Coding of Binary Images. *IEEE Transactions on Acoustic, Speech and Signal Processing*, 34: 1228–1244, 1986.
- [4] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The Grid File: An Adaptable, Symmetric, Multikey File Structure. *ACM Transactions on Database Systems*, 9: 39–71, 1984.