Retrieval by content in symbolic-image databases

Aya Soffer *

Computer Science and Electrical Engineering Department
University of Maryland Baltimore County, Baltimore, MD 21228-5398 and
Center of Excellence in Space Data and Information Sciences
NASA Goddard Space Flight Center
E-mail: soffer@cs.umbc.edu

Hanan Samet †

Computer Science Department and Center for Automation Research and
Institute for Advanced Computer Science
University of Maryland at College Park, College Park, Maryland 20742
E-mail: hjs@umiacs.umd.edu

ABSTRACT

Two approaches for integrating images into the framework of a database management system are presented. The classification approach preprocesses all images and attaches a semantic classification and an associated certainty factor to each object found in the image. The abstraction approach describes each object in the image by using a vector consisting of the values of some of its features (e.g., shape, genus, etc.). The approaches differ in the way in which responses to queries that are based on image content are computed. In the classification approach, images are retrieved on the basis of whether or not they contain objects that have the same classification as query objects. In the abstraction approach, retrieval is on the basis of similarity of feature vector values of these objects. Both the pattern recognition and indexing aspects of the method are addressed for each approach. The emphasis is on extracting both contextual and spatial information from the raw images. Methods for storing and indexing symbolic images as tuples in a relation are presented for each approach. Indices are constructed for both the contextual and the spatial data. The user interface for a pictorial information system based on these two approaches is also presented.

Keywords: image databases, retrieval by content, spatial databases, image indexing, query optimization

1 INTRODUCTION

There are two approaches for integrating images into a database so that images can be retrieved based on content (e.g., finding all campgrounds within 3 miles of a beach in a database of map images that contain tourist symbols). These two approaches differ in the time that the content of the image is classified. One approach, termed the *classification approach*, preprocesses all images and attaches a semantic classification and an associated certainty factor to each object that it finds in the images. The certainty factor enables more than one possible classification for each object in the database images. For example, all symbols that are likely to be restaurants,

*The support of USRA/CESDIS and NASA Goddard Space Flight Center is gratefully acknowledged.
†the support of the National Science Foundation under Grant IRI-9017393 is gratefully acknowledged.

campgrounds, etc, are labeled with their semantic meaning (i.e., that they are restaurants, campgrounds, etc. in contrast to being something else or even being invalid symbols). Images are retrieved from the database on the basis of containing objects that have the same classification as the objects specified by the query. An alternative approach, termed the *abstraction approach*, attempts to find some description of the objects in terms of properties of their visual representation (e.g., shape, length, connectivity, genus, etc.), termed a *feature vector*, and then retrieves images from the database that contain objects whose feature vectors match or are close to that of the feature vector of the objects specified by the query.

In essence, in the abstraction approach we are delaying the classification of the objects in the images until execution time. This permits greater flexibility in responding to queries in that the tolerance with which the objects in the database images match those that are requested in the query can be varied for each query. In contrast, in the classification approach this tolerance must be decided at the time the database is populated with the images. However, we can still vary the tolerance with which we retrieve matching images when using the classification approach by permitting the retrieval of images that contain objects whose matching classifications have smaller (larger) certainty values. The ability to vary the tolerance is important in image database applications as it is quite rare that an exact match is found. Rather, the goal of image retrieval in an image database is to find a set of candidate images that most likely fit a given specification.

In this paper we study the use of the classification and abstraction approach in an actual image database system and show how they can be integrated into a relational database management system. The paper is organized as follows. Section 2 discusses the background for this problem and discusses related work. Section 3 outlines the image input methodologies that we use. Section 4 describes how images are stored in a database management system including schema definitions and example relations. Section 5 gives sample queries along with execution plans for each approach. Section 6 describes our implementation of the system along with some snapshots of the system performing an example query. Section 7 contains concluding remarks.

2 BACKGROUND AND RELATED WORK

In order to support retrieval by content in image databases, images should be interpreted to some degree when they are inserted into the database. This process is referred to as converting an image from a *physical* representation to a *logical* representation. The logical representation may be a textual description of the image, a list of objects found in the image, a collection of features describing the objects in the image, etc. It is desirable that the logical representation also preserve the spatial information inherent in the image (i.e., the spatial relation between the objects found in the image). We refer to the information regarding the objects found in an image as *contextual information*, and to the information regarding the location of the objects and the spatial relation between these objects as *locational-spatial information* and *relational-spatial information*, respectively. An index mechanism based on the logical representation can then be used to retrieve images based on both contextual and spatial (both locational and relational) information in an efficient way.

Most commercial systems do not have any of these capabilities. These systems are actually image catalogs where the user may associate keywords (ASCII text) with images and retrieve images based on these keywords. These systems usually provide some form of browsing via *thumbnails* (small icons of the images). Some recent commercial systems do have a notion of retrieval by content. The Illustra object-relational DBMS[18] provides media-specific class libraries (DataBlades) for storing and managing image data. IBM's new UltiMedia Manager offers content-based image query (based on QBIC[10] technology) in conjunction with standard search.

Numerous prototype research IDMS's have been reported in recent years. These systems can be divided into two categories: those that do not deal with image processing and recognition issues, and those that do deal with these issues. The systems that do not address recognition issues[2,4,5] assume that as a result of applying some unspecified object recognition technique, they are dealing with tagged images. The main issue that these systems address is how to index these tagged images in order to support retrieval by image similarity. They are mainly

concerned with the spatial relationship between the objects in the images (i.e., relational-spatial information). They do not deal with locational-spatial information. The most common data structure that is used for this purpose is the *2-D string* and its variants.[2]

The systems that do deal with image processing and object recognition issues may also be roughly divided into two categories: (i) those that treat the image as a whole, and (ii) those that treat the image as a collection of objects that need to be recognized. The systems that treat the image as a whole index the images based mainly on color and texture. QBIC,[10] Photobook,[12] and FINDIT[17] are examples of systems that use such methods. Very few systems[7,8] try to recognize individual objects in an image. These systems do not, however, address the issues of spatial relationship between the objects, and efficient indexing. Some issues that deal with storing spatial information in a relational database have been discussed as part of the SEQUOIA 2000 project.[16] However, this work did not address the image interpretation, and contextual indexing aspects involved in integrating images (rather than just spatial data) into a relational database management system (DBMS).

In contrast, our approach combines indexing on spatial information as well as permitting retrieval on the basis of contextual (i.e., semantic) information. In our work, we have chosen to focus on images where the set of objects that may appear in them is known a priori, the geometric shapes of these objects are relatively primitive, and they convey symbolic information. For example, in the map domain many graphical symbols are used to indicate the location of various sites such as hospitals, post offices, recreation areas, scenic areas etc. We call this class of images *symbolic images*. We distinguish between *fully-symbolic* images and *partially-symbolic* images. In a fully-symbolic image we can fully classify each symbol found in the image and report the certainty of this classification. In a partially-symbolic, we assume that the symbols can be abstracted in such a way that given two symbols we can compute the certainty that they belong to the same class. All of the examples and experiments in this paper are from the map domain. However, images from many other interesting applications fall into the category of symbolic images. These include CAD/CAM, engineering drawings, floor plans, and more. Hence, the methods that we describe here are applicable to them as well.

3 IMAGE INPUT

Images can be represented in one of two ways. In the *physical image* representation, an image is represented by a two-dimensional array of pixel values. The physical representation of an image is denoted by I_{phys}. In the *logical image* representation, an image I is represented by a list of tuples, one for each symbol $s \in I$. In the classification approach, the tuples are of the form: $(C, certainty, (x, y))$ where $C \neq undefined$, (x, y) is the location of s in I, and $0 < certainty \leq 1$ indicates the certainty that $s \in C$. In the abstraction approach, the tuples are of the form: $\{\vec{s}, (x, y)\}$ where $\vec{s}$ is the feature vector representing symbol s, and (x, y) is the location of s in I. Image input is the process of converting an image from its physical to its logical representation. This process varies according to the image integration approach used.

In the classification approach, an input image I is converted to a logical image by classifying each symbol s found in I using a training set library. Symbols are classified using a modification a *weighted bounded several-nearest neighbor classifier*. An initial training set library is constructed by giving the system one example symbol for each class that may be present in the application. In the map domain, the legend of the map may be used for this purpose. A more representative training set is built by having the user verify the results of the classifications for the few first images that are inserted into the database. Feature vectors of symbols that could not be classified correctly using the current training set library are added to the library. Once the current recognition rate is deemed adequate, the remaining images are processed automatically. More than one candidate classification may be output for each symbol. Symbols that are classified as undefined are not inserted into the database. See[14] for more details.

In the abstraction approach, an image is converted into a logical image by creating an abstraction (the feature vector) for each symbol in the input image. There is no attempt to classify symbols or to weed out undefined

symbols during image input using the abstraction approach. A feature vector is inserted into the database for each connected component in the input image. One sample symbol from each class is also required in this case. The feature vector of this sample is used to search for symbols that belong to the same class as the sample when querying the database by content.

4 IMAGE STORAGE

Images and other information pertaining to the application are stored in relational tables. The spatial database that is used for both approaches is SAND[1] (spatial and non-spatial database) developed at the University of Maryland. It is a home-grown extension to a relational database, in which the tuples may correspond to geometric entities such as points, lines, polygons, etc. having attributes which may be both of a locational (i.e., spatial) and a non-locational nature. Both types of attributes may be designated as indices of the relation.

4.1 Classification approach

```
(create table classes
  name STRING,
  semant STRING,
  bitmap IMAGE);

(create table physical_images
  img_id INTEGER,
  descriptor STRING,
  upper_left POINT,
  raw IMAGE);

(create table logical_images
  img_id INTEGER,
  class STRING,
  certainty FLOAT,
  loc POINT);
```

Figure 1: Schemas of relations `classes`, `physical_images`, `logical_images` using the classification approach.

name	semant	bitmap
S	harbor	
square	hotel	
scenic	scenic view	
fish	fishing site	
R	restaurant	
P	post office	
air	airfield	
K	cafe	
waves	beach	
triangle	camping site	
pi	picnic site	
	⋮	

Figure 2: Example instance for the `classes` relation in the map domain using the classification approach.

img_id	descriptor	upper_left	raw
image_1	tile 003.012 of Finish road map	(6144,1536)	Figure 4
image_2	tile 003.013 of Finish road map	(6656,1536)	Figure 5

Figure 3: Example instance for the `physical_images` relation in the map domain using the classification approach.

The schema definitions given in Figure 1 define the relations in the DBMS following the classification approach. We use an SQL-like syntax. The `classes` relation has one tuple for each possible class in the application. The `name` field stores the name of the class (e.g., star), the `semant` field stores the semantic meaning of the class in this application (e.g., site of interest). The `bitmap` field stores a bitmap of an instance of a symbol representing this class. It is an attribute of type `IMAGE`. The `classes` relation is populated using the same data that is used to create the initial training set for the image input system (i.e., one example symbol for each class that may be present in the application along with its name and semantic meaning). See Figure 2 for an example instance of the `classes` relation in the map domain.

The `physical_images` relation has one tuple per image I in the database. The `img_id` field is an integer identifier given to I when it is inserted into the database. The `descriptor` field stores an alphanumeric description of I given by the user. The `raw` field stores the actual image in its physical representation. The `upper_left` field stores an offset value that locates the upper left corner of I with respect to the upper left corner of some larger image J. This is useful when a large image J is tiled, as in our example map domain. Subtracting this offset value from the absolute location of s in the the non-tiled image J yields the location of s in the tile I that contains it. Additional data about the images such as origin, camera angles, scale, etc. can be added as fields of this relation. See Figure 3 for an example instance of the `physical_images` relation in the map domain.

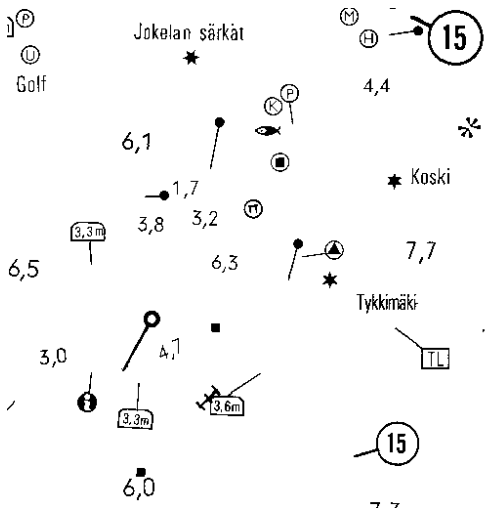

Figure 4: Example image 1 (`image_1`).

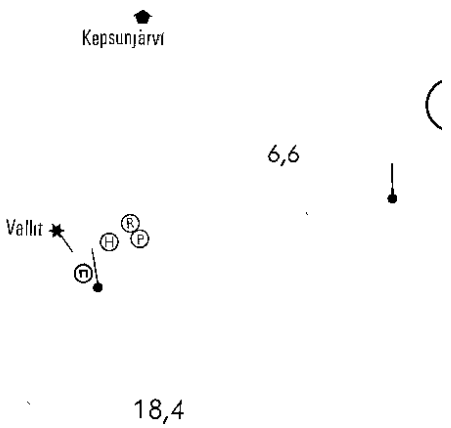

Figure 5: Example image 2 (`image_2`).

img_id	class	certainty	loc
image_1	M	1	(6493,1544)
image_1	P	0.99	(6161,1546)
⋮			
image_1	box	1	(6280,2011)
image_2	arrow	0.99	(6861,1544)
image_2	scenic	0.72	(6803,1565)
⋮			
image_2	R	0.99	(6800,1807)

Figure 6: Example instance for the `logical_images` relation in the map domain using the classification approach. The tuples correspond to the symbols in the images of Figures 4 and 5.

The `logical_images` relation stores the logical representation of the images. It has one tuple for each candidate class output by the image input system for each valid symbol s in each image I. The `img_id` field is the integer identifier given to I when it was inserted into the database. The `class` and `certainty` fields store the name of the class C to which s was classified and the certainty of the classification. The `loc` field stores the (x, y) coordinate values of the center of gravity of s relative to the non-tiled image. See Figure 6 for an example instance of the `logical_images` relation in the map domain for the images given in Figures 4 and 5.

Alphanumeric indices `cl_sem` and `cl_name` are constructed to search the `classes` relation by `semant` and `name`, respectively. An alphanumeric index `pi_id` is used to search the `physical_images` relation by `img_id`. A spatial index on points `pi_ul` is used to search the `physical_images` relation by the coordinates of the upper left corner of the images. An alphanumeric index `li_cl` is used to search the `logical_images` relation by `class`. It has a secondary index on attribute `certainty`. A spatial index on points `li_loc` is used to search the `logical_images` relation by location (i.e., to deal with spatial queries regarding the locations of the symbols in the images such

as distance and range queries). The spatial indices are implemented using a PMR quadtree for points.[9]

4.2 Abstraction approach

```
(create table classes  (create table physical_images  (create table logical_images
 name STRING,            img_id INTEGER,                img_id INTEGER,
 semant STRING,          descriptor STRING,             fv POINT,
 bitmap IMAGE,           upper_left POINT,              loc POINT);
 fv POINT);              raw   IMAGE);
```

Figure 7: Schemas of the relations `classes`, `physical_images`, and `logical_images` when using the abstraction approach.

img_id	fv	loc
image_1	(0.909,0.032,24.854,0.053,17.829,12.047,14.412,55.906)	(6162,1546)
image_1	(1.144,0.024,21.053,0.158,15.597,14.062,9.766,18.891)	(6494,1546)
⋮		
image_1	(0.687,0.021,35.948,0.471,11.205,0.000,0.000,0.000)	(6158,1614)
image_2	(0.166,0.063,61.654,0.263,2.244,0.000,0.055,0.000)	(6862,1545)
image_2	(0.456,0.023,46.094,0.500,9.517,0.000,0.000 0.000)	(6804,1565)
⋮		
image_2	(0.513,0.054,19.448,0.138,9.834,0.000,0.000,0.000)	(6776,1766)

Figure 8: Example instance for `logical_images` relation in the map domain using the abstraction approach. The tuples correspond to part of the symbols in the images of Figures 4 and 5 (the full table would have 78 tuples).

The schema definitions given in Figure 7 define the tables for the abstraction approach. These tables are very similar to those used in the classification approach. They differ in the presence of the `fv` field which corresponds to the feature vector for a sample of the class in the case of the `classes` relation, and to a feature vector for each of the actual symbols in the image in the case of relation `logical_images`. Spatial indices `cl_fv` and `li_fv` are constructed for both the `classes` and the `logical_images` relations on the basis of the `fv` field. They make use of These indices are realized with a disk-based version of an adaptive k-d tree,[3] a data structure that is suitable for indexing points in high dimensions. By using this disk-based adaptive k-d tree for indexing the feature vectors, we get very efficient searches at query time since by definition the adaptive k-d tree is balanced and separates the vectors by the best features. The index has to be recomputed when images are added to the database. However, since images are usually entered in batches, and since preprocessing an image is quite time consuming, this is not a concern.

5 RETRIEVING IMAGES BY CONTENT

In order to describe the methods that we use for retrieving images by content, we present some example queries and demonstrate the strategies used to process these queries.

5.1 Example queries

The example queries in this section are first specified using natural language. For the first example, we also give two equivalent SQL-like queries. The first assumes the classification approach, and the second assumes the abstraction approach. For the other example queries, we only give an SQL-like query for the classification approach. An SQL-like query for the abstraction approach for these queries can be created in a similar manner.

Query Q1: display all images that contain a scenic view.

```
display PI.raw
    from logical_images LI, classes C, physical_images PI
    where C.semant = "scenic view" and C.name = LI.class and LI.img_id = PI.img_id;

display PI.raw
    from logical_images LI, classes C, physical_images PI
    where C.semant = "scenic view" and wdist(C.fv,LI.fv) <= MD and LI.img_id = PI.img_id;
```

The function `wdist` takes two feature vectors and returns the weighted Euclidean distance between them. `MD` is a parameter, which may be set by the user, that determines the maximum distance (in weighted feature space) between two feature vectors such that the two symbols that are abstracted by these two feature vectors are considered to be in the same classification. That is, $(wdist(\vec{s_1}, \vec{s_2}) \leq MD) \wedge s_1 \in C \Rightarrow s_2 \in C$.

Query Q2: display all images that contain a site of interest within 5 miles of a hotel.

```
display PI.raw
    from logical_images LI1, logical_images LI2, classes C1, classes C2,  physical_images PI
    where C1.semant = "hotel" and C2.semant = "site of interest"
        and C1.name = LI1.class and C2.name = LI2.class and dist(LI1.loc,LI2.loc) <= 5
        and LI1.img_id = LI2.img_id and LI1.img_id = PI.img_id;
```

The function `dist` takes two geometric objects (such as two points in the example above) and returns a floating point number representing the Euclidean distance between them.

Query Q3: display all images with a site of interest and output the semantics of anything within 2 miles of these sites of interest.

```
display PI.raw C2.semant
    from logical_images LI1, logical_images LI2, classes C1, classes C2,  physical_images PI
    where C1.semant = "site of interest" and C1.name = LI1.class and dist(LI1.loc,LI2.loc) <= 2
        and LI1.img_id = LI2.img_id and LI1.img_id = PI.img_id and C2.name = LI2.class;
```

5.2 Query processing

The following plans outline how responses to queries Q1 and Q2 are computed using the two approaches. These plans utilize the indexing structures available for each organization. Indices on alphanumeric attributes are capable of locating the closest value greater than or equal to a given string or number. Indices on spatial attributes are capable of returning all neighbors within D of a query point in increasing distance. The X^{th} plan, labeled Px_C and Px_A, use the classification approach and the abstraction approach, respectively. See[15] for detailed plans for these queries, as well as an analysis of the expected costs of these plans.

Query Q1: display all images that contain a scenic view.

Plan P1$_C$: Search using an alphanumeric index on `class`.

```
Get all tuples of logical_images which correspond to "scenic view" (use index li_cl)
For each such tuple t
  display the physical image corresponding to t
```

Plan P1$_A$: Search using spatial index on `fv`.

```
Get all tuples of logical_images whose fv field is within MD
     of the feature vector of "scenic view" (use index li_fv)
For each such tuple t
  display the physical image corresponding to t
```

Query Q2: display all images that contain a hotel within 5 miles of a site of interest.

Finding a plan for Q2 gives rise to many query optimization issues within the domain of image databases. Most of these issues are also applicable to spatial databases.[1] To illustrate just how complex this issue may be, we present several plans for computing an answer to Q2. They differ in the selection of indices that are used to process the queries, and in whether or not they build intermediate structures while processing the query.

Plan P2A$_C$ Search for "site of interest" and "hotel" tuples using the alphanumeric index on `class`. For each "site of interest" tuple, check all "hotel" tuples, and see which ones are within 5 miles.

Plan P2B$_C$ Search for "site of interest" tuples using the alphanumeric index on `class` and search for "hotel" tuples within 5 miles using the spatial index on `loc`.

Plan P2C$_C$ Search for "hotel" tuples using alphanumeric index on `class`. Build a temporary spatial index on the `loc` attribute of these tuples. Search for "site of interest" tuples using alphanumeric index on `class`. Search for "hotel" tuples within 5 mile using the temporary index.

Plan P2A$_A$ Search for "site of interest" and "hotel" tuples using the spatial index on `fv`. For each "site of interest" tuple, check all "hotel" tuples, and see which ones are within 5 miles.

Plan P2B$_A$ Search for "site of interest" tuples using the spatial index on `fv` and search for "hotel" tuples within 5 miles using the spatial index on `loc`.

Plan P2C$_A$ Search for "hotel" tuples using the spatial index on `fv`. Build a temporary spatial index on the `loc` attribute of these tuples. Search for "site of interest" tuples using the spatial index on `fv`, and search for "scenic view" tuples within 5 miles using the temporary index.

5.3 User interface

Queries may be posed using either an SQL-like language or a graphical user interface (GUI). The SQL-like language used is part of SAND and it includes primitives for spatial queries such as distance, intersect, nearest neighbor, etc. By using this language, users can pose a wide range of queries to the system. However, this extended SQL-like language is not trivial and requires that the user know the schema definitions. In contrast, the graphical user interface provides access to a number of query categories that are common in such a map image database application. The variety of queries that can be posed using the GUI is limited; however, it is very easy to use. Currently, we have defined five query categories as follows:

Contain Query: find all map tiles that contain symbol(s) from some given *class(es)* (e.g., query Q1).

One Within Query: find all map tiles in which a symbol from *class2* is within a given *distance* from a symbol from *class1* (e.g., query Q2).

All Within Query: find all map tiles in which a symbol from any class is within a given *distance* from a symbol from *class1* (e.g., query Q3).

Nearest Query: find all map tiles with the *nearest* symbol from *class2* to a symbol from *class1*.

Directional Location Query: find all map tiles in which a symbol from *class1* is located in a given *direction* relative to a symbol from *class2*.

An additional difference between queries specified using SQL and queries specified using the GUI is in the cost (in terms of time) of responding to the queries. For queries that are specified using SQL, the query plan is generated automatically. Thus the cost of computing the result is determined by the quality of SAND's query optimizer. As mentioned above, the problem of writing a query optimizer for a spatial database is very complex, and thus these plans will most likely not be optimal. On the other hand, using the GUI, the user has access to only a limited number of query categories. The plans for these query categories are hard-wired into the system, and thus they are very efficient.

6 IMPLEMENTATION

The image database system was tested on the red sign layer of the GT_3 map of Finland, which is one of a series of 19 GT maps that cover the whole area of Finland. The red sign layer contains geographic symbols that mostly denote tourist sites. The map was scanned at 240dpi. The layer was split into 425 tiles of size 512×512. Each one of these tiles that contained at least one symbol was considered to be an image. Figure 4 is an example image. Of these 425 tiles, 280 contained at least one symbol. These 280 images contained 4111 symbols (both valid and invalid).

The images were input using the image input methodology described in Section 3 for the classification and the abstraction approach. The initial training set was created by giving one example symbol of each class as taken from the legend of the map. There were 22 classes in the map. The first 50 images were processed in user verification mode. At that point, the training set contained 100 instances of symbols and the current recognition rate was determined sufficient. The remaining images were processed automatically. The results of this conversion (i.e., the logical images) were input to SAND and inserted into relations as defined in Section 4. There were a total of 1093 tuples in the `logical_images` relation corresponding to 280 logical images when using the classification approach. In the abstraction approach the feature vectors of both valid and invalid symbols are inserted into the image database. Since these 280 images had 4111 such symbols, there were a total of 4111 tuples in the `logical_images` relation when using the abstraction approach. Query execution plans were created for each one of the queries given in Section 5 following the strategies outlined there. These plans were written in Tcl (short for Tool Command Language), an interpreted scripting language developed by Ousterhout.[11]

6.1 An example query execution

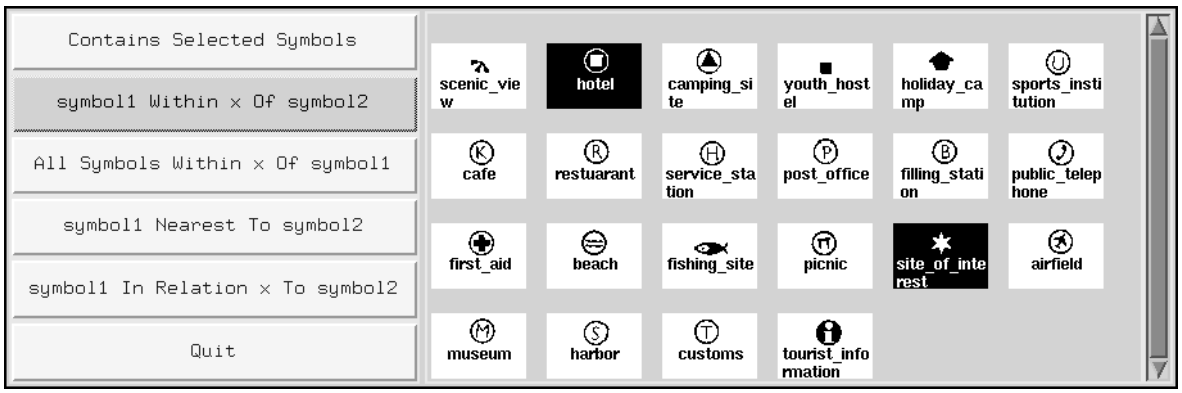

Figure 9: Graphical User Interface for query initiation. User has selected a "One Within Query" between a "hotel" and a "site of interest".

The following scenario describes how example query Q2 is specified using the GUI and how the results are presented. Recall, that Q2 was the query "display all images that contain a site of interest within 5 miles of a hotel". Figure 9 shows the GUI for initiating a query. It consists of a button for each query category and an icon for each of the symbol classes. The icon is composed of the `bitmap` and `semant` fields of the tuples in the `classes` relation. To perform a "one within" query, the user first selects the icons of the two required classes followed by clicking the "symbol1 Within x of symbol2" button. The user is then prompted for the required distance. Once the user enters the required distance, the result of the query is computed using one of the plans outlined in Section 5.2. The result of this query is displayed in a window as seen in Figure 10. A thumbnail (i.e., a reduced bitmap of the whole tile) is displayed for each layer tile that was found that meets the query specification. Recall, that in the classification approach each tuple in the `logical_images` relation has a certainty value that estimates the certainty that the symbol in the location corresponding to the tuple belongs

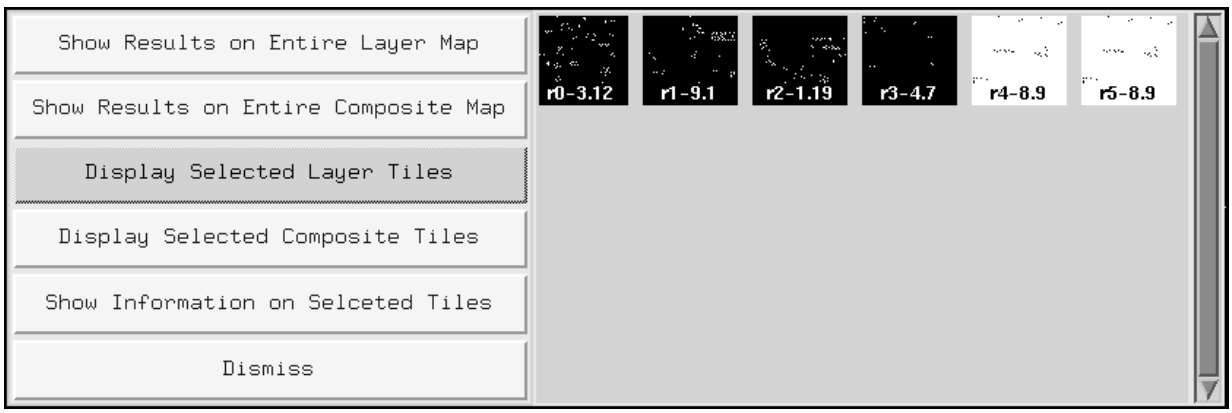

Figure 10: Results of query computation. The user has selected to display the layer tiles of the first four results.

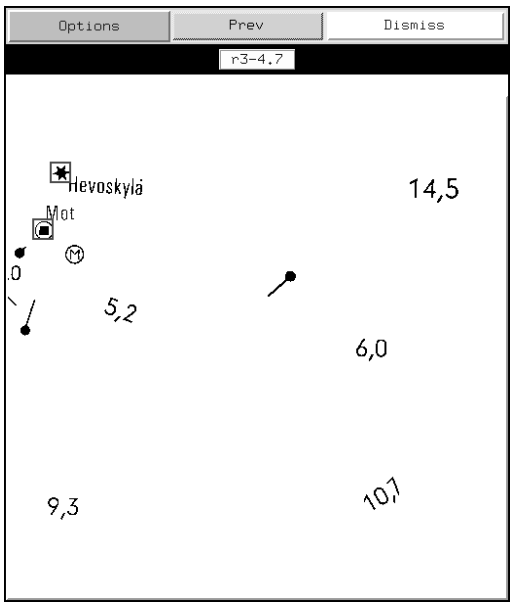

Figure 11: Displaying the selected layer tiles.

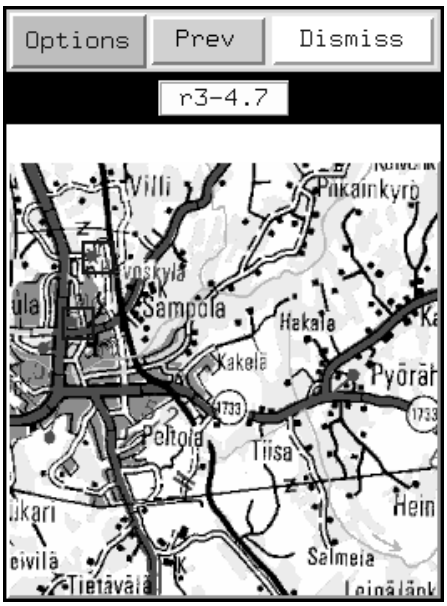

Figure 12: Displaying the selected composite tiles.

to the class corresponding to the tuple. The result tiles are displayed in decreasing order of the certainty value. Therefore, the first result tiles are more likely to be correct (i.e., meet the query specification) and the last tiles are more likely to be incorrect. In the case of the abstraction approach, these certainties are computed on the fly according to the distance from the sample feature vector that is used in the query. The user may now display any of the result tiles by selecting the corresponding thumbnails followed by clicking either the "Display Layer" or "Display Composite" buttons. Figures 11 and 12 show the results of clicking these two buttons. The "prev" button is used to step through the selected tiles. A square is drawn around the two symbols that were given to the query. By clicking the "Information" button, the user can see the information stored regarding each of these tiles in the `physical_images` relation and the locations of the symbols in these tiles. In addition, the user may choose to display the non-tiled map with the query result tiles highlighted.

6.2 Experimental study

	Classification Approach	*Abstraction Approach*
Image preprocessing workload	heavy	moderate
Image insertion time	faster	slower
User interaction required	yes	no
Spatial indexing space	location	location and feature
Retrieval by content workload	low	moderate
Hybrid query workload	low	moderate
Number of applications	smaller	larger
Adaptable at run time	slightly	highly
Accuracy: type I	good	moderate
Accuracy: type II	good	moderate

Table 1: Comparison of the classification approach and the abstraction approach.

We conducted an experimental study that measured various parameters regarding storage and retrieval performance using these two image storage approaches. These include image insertion time, storage space, retrieval accuracy, and query execution time. In[15] we present the details of these tests and the results. Table 1 summarizes the conclusions from this experimental study.

7 CONCLUDING REMARKS

Throughout this paper we discussed how to input, store, index, and retrieve symbolic images in an image database following the classification and the abstraction approaches. The choice of which approach to use depends on the application at hand. In addition to the (GUI) that we described here, we are currently incorporating pictorial query specifications. In this case, a sample symbol for each class will not be required when using the abstraction approach, as it can be taken directly from the pictorial query.

The examples and experiments in this paper were from the map domain. However, images from many other interesting applications also fall into the category of symbolic images. One complication that could arise in other applications is that the spatial extent of symbols may be of importance. In our example application, symbols were represented by a point. In other applications, we may need to use bounding boxes or other geometric entities to represent the symbols in the logical image. However, by using the methods suggested in this paper we can handle objects with spatial extent just as easily. The only difference would be in the selection of the spatial data structure that is used to index the locational information. For example, if symbols are represented by bounding boxes, then any data structure that is suitable for indexing rectangles such as an R-tree[6] can be used. The well-known algorithms that exist for range queries on these data structures can then be utilized for efficient processing of queries that have a spatial component. In contrast, it is considerably harder to deal with spatial extent in methods based on 2-D strings.[2] Note that we have used similar methods in a system for the interpretation of floor plans.[13]

8 ACKNOWLEDGEMENTS

We are grateful to Karttakeskus, Map Center, Helsinki, Finland for providing us the map data.

9 REFERENCES

[1] W. G. Aref and H. Samet. Optimization strategies for spatial query processing. In G. Lohman, editor, *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, pages 81–90, Barcelona, September 1991.

[2] S. K. Chang, Q. Y. Shi, and C. Y. Yan. Iconic indexing by 2-D strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(3):413–428, May 1987.

[3] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[4] V. Gudivada and V. Raghavan. Design and evaluation of algorithms for image retrieval by spatial similarity. *ACM Transactions on Information Systems*, 13(2):115–144, April 1995.

[5] A. Gupta, T. Weymouth, and R. Jain. Semantic queries with pictures: the VIMSYS model. In G. Lohman, editor, *Proceedings of the Seventeenth International Conference on Very Large Databases*, pages 69–79, Barcelona, Spain, September 1991.

[6] A. Guttman. R–trees: a dynamic index structure for spatial searching. In *Proceedings of the SIGMOD Conference*, pages 47–57, Boston, June 1984.

[7] H. V. Jagadish. A retrieval technique for similar shapes. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, pages 208–217, Denver, CO, May 1991.

[8] R. Mehrotra and J. Gary. Feature-index-based similar shape retrieval. In *Third Working Conference on Visual Database Systems*, pages 39–55, Lausanne, Switzerland, March 1995.

[9] R. C. Nelson and H. Samet. A consistent hierarchical representation for vector data. *Computer Graphics*, 20(4):197–206, August 1986. (also *Proceedings of the SIGGRAPH'86 Conference*, Dallas, August 1986).

[10] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker. The QBIC project: Querying images by content using color, texture, and shape. In *Proceeding of the SPIE, Storage and Retrieval of Image and Video Databases*, volume 1908, pages 173–187, San Jose, CA, February 1993.

[11] J. K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, Reading, MA, April 1994.

[12] A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Content-based manipulation of image databases. In *Proceeding of the SPIE, Storage and Retrieval of Image and Video Databases II*, volume 2185, pages 34–47, San Jose, CA, February 1994.

[13] H. Samet and A. Soffer. Automatic interpretation of floor plans using spatial indexing. In S. Impedovo, editor, *Progress in Image Analysis and Processing III*, pages 233–240. World Scientific, Singapore, 1994.

[14] H. Samet and A. Soffer. A legend-driven geographic symbol recognition system. In *Proceedings of the 12th International Conference on Pattern Recognition*, volume II, pages 350–355, Jerusalem, Israel, October 1994.

[15] A. Soffer. *Retrieval by Content in Symbolic-Image Databases.* PhD thesis, University of Maryland, 1995.

[16] M. Stonebraker, J. Frew, and J. Dozier. The SEQUOIA 2000 project. In D. Abel and B. C. Ooi, editors, *Advances in Spatial Databases - Third International Symposium, SSD'93*, number 692 in Lecture Notes in Computer Science, pages 397–412, Singapore, June 1993.

[17] M. Swain. Interactive indexing into image databases. In *Storage and Retrieval for Image and Video Databases*, pages 95–103. SPIE, Vol. 1908, 1993.

[18] M. Ubell. The montage extensible dataBlade architecture. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, page 482, Minneapolis, MN, June 1994.