Integrating Symbolic Images into a Multimedia Database System using Classification and Abstraction Approaches

Aya Soffer [*]
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: aya@umiacs.umd.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

December 23, 1999

Abstract

Symbolic images are composed of a finite set of symbols that have a semantic meaning. Examples of symbolic images include maps (where the semantic meaning of the symbols is given in the legend), engineering drawings, and floor plans. Two approaches for supporting queries on symbolic-image databases that are based on image content are studied. The classification approach preprocesses all symbolic images and attaches a semantic classification and an associated certainty factor to each object that it finds in the image. The abstraction approach describes each object in the symbolic 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 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 the objects in the query. On the other hand, in the abstraction approach retrieval is on the basis of similarity of feature vector values of these objects.

Methods of integrating these two approaches into a relational multimedia database management system so that symbolic images can be stored and retrieved based on their content are described. Schema definitions and indices that support query specifications involving spatial as well as contextual constraints are presented. Spatial constraints may be based on both locational information (e.g., distance) and relational information (e.g., north of). Different strategies for image retrieval for a number of typical queries using these approaches are described. Estimated costs are derived for these strategies. Results are reported of a comparative study of the two approaches in terms of image insertion time, storage space, retrieval accuracy, and retrieval time.

Categories: symbolic-image databases, multimedia databases, retrieval by content, spatial databases, image indexing, query optimization

[*]The support of the National Science Foundation under Grant CDA-950-3994 is gratefully acknowledged.
[†]The support of the National Science Foundation under Grant IRI-97-12715 is gratefully acknowledged.

1 Introduction

Symbolic images are composed of a finite set of symbols that have a semantic meaning. Examples of symbolic images include maps (where the semantic meaning of the symbols is given in the legend), engineering drawings, and floor plans. There are two approaches for integrating symbolic images into a multimedia database so that they 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 at which the content of the image[1] 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 images in the database. For example, all symbols that are likely to be restaurants, 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 multimedia 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. Therefore, 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 when the objects of each image are classified (i.e., at the time the database is populated with the images). Some additional tolerance can still be achieved at retrieval time by permitting the retrieval of images that contain objects whose matching classifications have smaller (larger) certainty values.

In this paper we demonstrate the use of the classification and abstraction approach in a database system that stores symbolic images. We show how they can be integrated into a relational database management system and discuss the issues that are involved. This includes query processing strategies to take advantage of the presence of relevant indices (i.e, on the classes and the spatial locations of the objects). We also report the results of a performance comparison of the execution time needed to respond to a number of different queries using a database of map images containing tourist symbols that is organized using these two approaches. Although, the main conceptual difference between the two approaches is in the time at which the classification is made, the two approaches also differ in numerous performance aspects (e.g., image insertion time, storage space, retrieval accuracy, and retrieval time).

The main contribution of this paper a detailed presentation of all the necessary steps required to integrate symbolic images into a database system using these two approaches. In addition, we provide a comprehensive comparison of the classification and abstraction approaches by applying both of them to the same application domain. Although methods using similar approaches have been described separately (e.g., [2, 8, 25]), to the best of our knowledge they have never been applied simultaneously to one dataset so that such a comparison could be conducted. Furthermore, the method that we present for integrating symbolic images into a database system enables queries

[1]In the interest of brevity, we use the more general term *image* although we mean a symbolic image unless there is some possibility for misinterpreting our discussion.

involving spatial constraints that include distance between objects. In most previous work (e.g., [7, 20, 34]), the relation between objects could only be described in terms of relative locations between objects (i.e., left of, north of, etc.). Finally, as part of the comparison we discuss query execution plan generation and evaluation for a mixture of spatial and non-spatial queries. This is an important topic in itself which has not received much attention. While query optimization for relational databases has been studied extensively (e.g. [33]) and some work exists regarding query optimization for object-oriented databases (e.g. [40]), the issue of query optimization in a spatial database has hardly been studied. Note that our system performs automatic image input and classification and thus we also demonstrate how to deal with uncertainty in the classification process and how to test the system in terms of accuracy.

The rest of this paper is organized as follows. Section 2 lays out the background for this problem and discusses related work. Section 3 outlines the image input methodologies that are used to convert images from a physical representation to a logical representation as they are input to the symbolic-image database. 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 and estimated costs for each approach. Section 6 describes our experimental study and the results of a quantitative comparison of the two proposed approaches for integrating symbolic images into a database. Section 7 contains concluding remarks.

2 Background and Related work

In order to support retrieval by content in symbolic-image databases, the 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. 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 contents of the image in terms of the objects found in it 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 (locational and relational) information in an efficient way.

Very few commercial systems support retrieval by image content. The INFORMIX-Universal Server [17] supports DataBlade modules that enable adding custom data types for supporting non-traditional kinds of data, such as full text, images, video, spatial data, and time series. The DataBlade concept is based on Illustra object relational database [43]. In particular, Informix offers the Virage [44] Visual Information Retrieval (VIR) DataBlade Module for managing images in its database server. VIR defines images by four important attributes-color, composition, structure, and texture. These attributes are quantitative measurements that can be used to compare images based upon their visual characteristics. IBM's UltiMedia Manager offers content-based image query (based on QBIC [25] technology) in conjunction with standard search. Excalibur Technologies' [42] Visual RetrievalWare product is a comprehensive application development software which provides content-based, high-performance retrieval for multiple types of digital visual media. Content-based image search is performed based on composition, color, and contrast. In all of these systems, image content means similarity in terms of global feature similarity between a query image and the database images. Thus they do not deal with semantic-like features and spatial constraints.

Numerous prototype research image database management systems (IDMS) have been reported in recent years that address the issue of how to index tagged images (images in which the objects have already been recognized and associated with their semantic meaning) in order to support retrieval by image content [2, 6, 9, 13, 14, 19, 27]. These systems do not specify how these tagged images were created. They assume either manual tagging or some unspecified (and most likely non-existing) automatic process that performs this function. Therefore, they do not deal with the uncertainty and errors that are involved in truly automatic image input procedures. Furthermore, these systems are mainly concerned with the relative spatial relationship between the objects in the images (i.e., relational-spatial information). They do not deal with locational-spatial information (e.g., distance between objects).

The most common data structure used in these systems is the *2-D string* and its variants [7, 20]. Distance is not preserved at all in 2-D strings. Thus, only a limited subset of spatial queries that do not deal with distances between objects can be handled. A framework which supports querying on spatial arrangements of objects, is presented in [4]. It enables comparing objects which may be arbitrary regions without replacing the regions by a centroid or bounding rectangle as is usually done. Distance between objects is not accounted for in this method either. Another data structure called the *spatial orientation graph* is introduced in [13] and used for spatial similarity based retrieval of symbolic images. This method can handle slope between objects, however it cannot handle distances. In [34] a method for similarity based retrieval of pictures using indices on spatial relationships is described. This method can only deal with relational-spatial information, it can not handle locational-spatial information. Furthermore, the relationship between objects needs to be explicitly defined using one of a set of predefined relations (e.g., left of, overlap, inside, etc.). In order to allow queries involving other relationships that are not part of this predefined set such as "diagonal of" or "close to", the logical image representation of all images stored in the database would need to be modified.

Other prototype research IDMS's treat the image as a whole, and index the images based mainly on color and texture. QBIC [25], Photobook [29], and FINDIT [41] are examples of systems that use such methods. In these systems, images can be retrieved on the basis of similarity to a query image as a whole. There is no notion of an image that is composed of several objects. Thus the issue of spatial information is not considered in these systems. In some cases [3, 10, 22, 35], the images are segmented into regions either automatically or semi-automatically and some queries involving spatial constraints specifying the desired arrangement of these regions can be performed. A few systems [18, 23] 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 [39]. However, this work did not address the image interpretation and contextual indexing aspects involved in integrating images (rather than just spatial data) into a DBMS.

In contrast, our approach combines indexing on spatial information (both locational and relational) as well as permitting retrieval on the basis of contextual (i.e., semantic) information. The conversion from the physical to the logical representation is part of the system, and the database can accommodate uncertainty in the conversion process. The method that we use for storing and indexing spatial information is flexible and new spatial relationships between objects can be defined at query time with no need to refine the logical representation of the images already stored in the database.

In our work, we 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 image, 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. The reason that we have chosen to concentrate on such images is that there are well-known and effective methods to automatically convert such images from a physical to a logical representation. In contrast, for other types of images such as photographs, satellite images, and medical images, such robust methods do not exist. 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. We have conducted a preliminary study for some of them (e.g., floor plans [31]).

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: $s : (C, certainty, (x, y))$ 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: $s : (\vec{s}, (x, y))$ 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 of the *weighted several-nearest neighbor classifier* [5] termed 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. The feature vector that is computed for each of these samples constitutes an initial training set. 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.

The feature vector is composed of four global descriptors (first invariant moment, circularity, eccentricity, and rectangularity) and three local shape descriptors (horizontal gaps per total area, vertical gaps per total area, and ratio of hole area to total area). These features are commonly used to describe shapes [21], and we found them to be effective in discriminating between different geographic symbols. These features are all invariant to scale and translation. In addition, most of them are also invariant to orientation.

More than one candidate classification may be output for each symbol. Symbols that are classified as undefined are not inserted into the database. Figure 1 is a block diagram of the image input system for the classification approach. See [32] for more details.

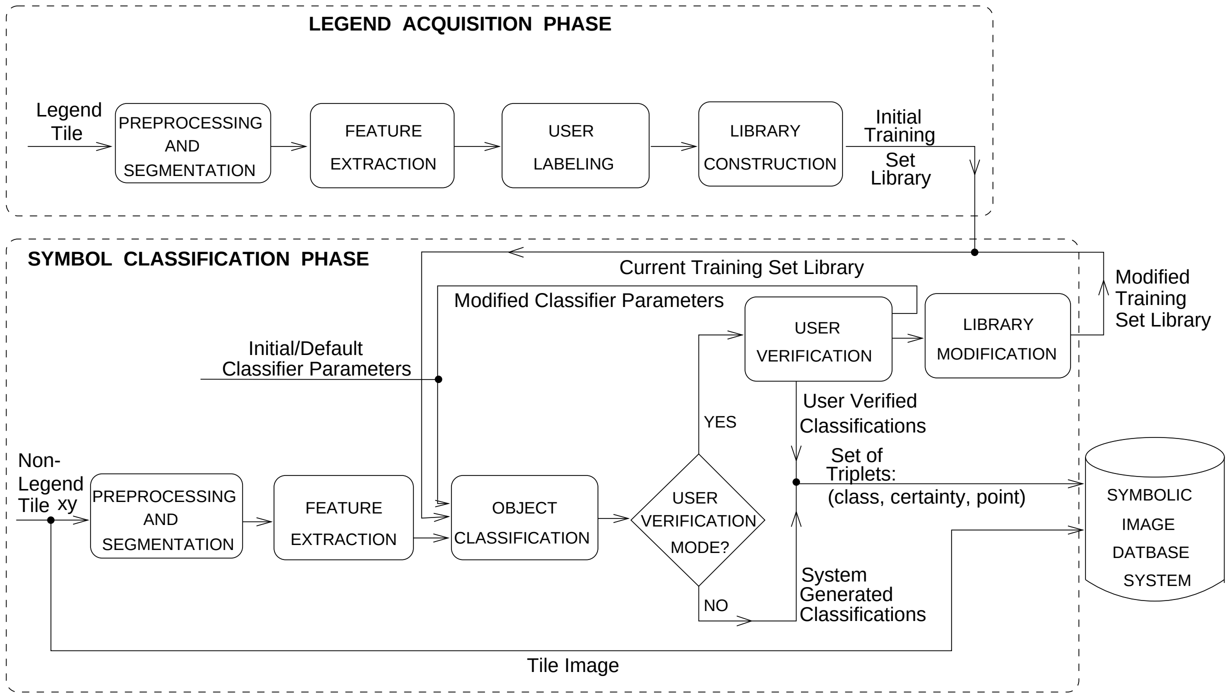

LEGEND ACQUISITION PHASE
Legend Tile
PREPROCESSING AND SEGMENTATION
FEATURE EXTRACTION
USER LABELING
LIBRARY CONSTRUCTION
Initial Training Set Library
SYMBOL CLASSIFICATION PHASE
Current Training Set Library
Modified Classifier Parameters
Modified Training Set Library
USER VERIFICATION
LIBRARY MODIFICATION
Initial/Default Classifier Parameters
YES
User Verified Classifications
Set of Triplets: (class, certainty, point)
Non-Legend Tile xy
PREPROCESSING AND SEGMENTATION
FEATURE EXTRACTION
OBJECT CLASSIFICATION
USER VERIFICATION MODE?
SYMBOLIC IMAGE DATBASE SYSTEM
NO
System Generated Classifications
Tile Image

Figure 1: Image input system (classification approach)

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 symbolic-image 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.

As mentioned above, the database that we used is based on the relational model. The methods that we employ can however be used in an object-oriented database as well. In fact, SAND has many characteristics of an object relational database in the sense that new data types can be defined along with methods that access, index, and display these types. We have utilized this feature to implement the high dimensional data type used for the feature vector attribute along with a k-d tree index for this data type.

4.1 Classification Approach

```
(CREATE TABLE classes
  name STRING PRIMARY KEY,
  semant STRING,
  bitmap IMAGE);

(CREATE TABLE physical_images
  img_id INTEGER PRIMARY KEY,
  descriptor STRING,
  upper_left POINT,
  raw IMAGE);

(CREATE TABLE logical_images
  img_id INTEGER REFERENCES physical_images(img_id),
  class STRING REFERENCES classes(name),
  certainty FLOAT (CHECK certainty BETWEEN 0 AND 1),
  loc POINT,
  PRIMARY KEY (img_id,class,loc);
```

Figure 2: Schemas for the relations classes, physical_images, and logical_images when using the classification approach.

name	semant	bitmap
S	harbor	Ⓢ
square	hotel	▣
scenic	scenic view	⌒
T	customs	Ⓣ
R	restaurant	Ⓡ
P	post office	Ⓟ
M	museum	Ⓜ
K	cafe	Ⓚ
waves	beach	⊜
triangle	camping site	▲
B	filling station	Ⓑ
arrow	holiday camp	⬟
cross	first aid station	⊕
fish	fishing site	🐟
H	service station	Ⓗ
inf	tourist information	ℹ
pi	picnic site	Ⓟ
air	airfield	✈
star	sight of interest	✱
telephone	public telephone	☎
box	youth hostel	■
U	sports institution	Ⓤ

Figure 3: 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 5
image_2	tile 003.013 of Finish road map	(6656,1536)	Figure 6

Figure 4: Example instance for the physical_images relation in the map domain using the classification approach.

The schema definitions given in Figure 2 define the relations in the DBMS when using 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 3 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 assigned 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 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 4 for an example instance of the `physical_images` relation in the map domain.

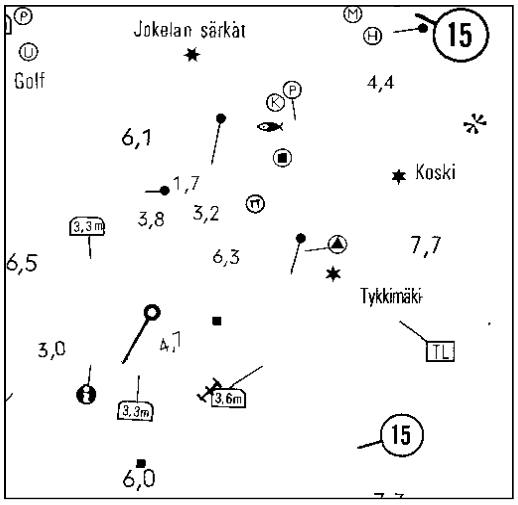


Figure 5: Example image 1 (`image_1`).

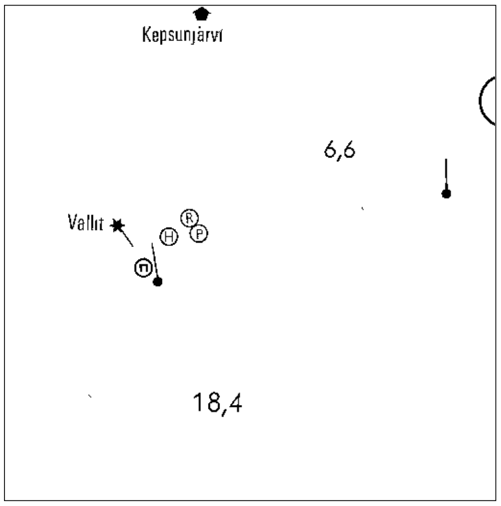


Figure 6: Example image 2 (`image_2`).

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 assigned to I when it was inserted into the database. It is a foreign key referencing the `img_id` field of the tuple representing I in the `physical_images` relation. 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. The primary key of the `logical_images` relation is the combination of the `img_id`, `class`, and `loc` fields. Recall, that a symbol may be assigned more than one classification. Therefore there can be two tuples with the same `img_id`,`loc` values, but with a different `class`, and thus all three attributes are required to uniquely identify a tuple. See Figure 7 for an example instance of the `logical_images` relation in the map domain for the images given in Figures 5 and 6.

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 7: 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 5 and 6.

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, range, and directional relationship queries). The spatial indices are implemented using a PMR quadtree for points [24].

4.2 Abstraction Approach

```
(CREATE TABLE classes
 name STRING PRIMARY KEY,
 semant STRING,
 bitmap IMAGE,
 fv POINT);

(CREATE TABLE physical_images
 img_id INTEGER PRIMARY KEY,
 descriptor STRING,
 upper_left POINT,
 raw   IMAGE);

(CREATE TABLE logical_images
 img_id INTEGER REFERENCES physical_images(img_id),
 fv POINT,
 loc POINT,
 PRIMARY KEY (img_id,loc));
```

Figure 8: Schemas of the relations `classes`, `physical_images`, and `logical_images` when using the abstraction approach.

The schema definitions given in Figure 8 define the relations for the abstraction approach. These relations 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

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 9: Example instance for `logical_images` relation in the map domain using the abstraction approach. The tuples correspond to a subset of the symbols in the images of Figures 5 and 6 (the full relation would have 78 tuples).

of relation `logical_images`. In the abstraction approach, the attributes `img_id` and `loc` uniquely identify a tuple as there can be only one symbol in a given location. Thus, the primary key of relation `logical_images` is (`img_id`,`loc`). Multidimensional indices `cl_fv` and `li_fv` are constructed for both the `classes` and the `logical_images` relations on the basis of the `fv` field. These indices are realized with a disk-based version of an adaptive k-d tree [12], a data structure that has been shown to be effective for indexing points in high dimensions in contrast to variants of R-trees [45]. 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 most discriminating 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 in this work.

5 Retrieving Images by Content

In order to describe the methods that we use for retrieving images by content, and in order to compare the cost of processing queries using the two suggested approaches, we present two example queries and demonstrate the strategies used to process these queries. The first query only deals with contextual information. The second query specifies constraints both on contextual and spatial information.

5.1 Example Queries

The example queries are specified using natural language and SQL. For each query, we present two SQL formulations. The first formulation uses the schemas as defined by the classification approach, and the second formulation uses the schemas as defined by the abstraction approach.

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, that 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, for any two symbols s_1 and s_2, $(wdist(\vec{s_1}, \vec{s_2}) \leq MD) \wedge s_1 \in C \Rightarrow s_2 \in C$, where $\vec{s_1}$ and $\vec{s_1}$ are the feature vectors representing symbols s_1 and s_2, respectively.

Query Q2: display all images that contain a scenic view within 5 miles of a picnic site.

```
display PI.raw
    from logical_images LI1, logical_images LI2, classes C1,
        classes C2,  physical_images PI
    where C1.semant = "scenic view" and C2.semant = "picnic site"
        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;

display PI.raw
    from logical_images LI1, logical_images LI2, classes C1,
        classes C2,  physical_images PI
    where C1.semant = "scenic view" and C2.semant = "picnic site"
        and wdist(C1.fv,LI1.fv) <= MD
        and wdist(C2.fv,LI2.fv) <= MD
        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.

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 the items in increasing order of their distance from a given point (this is termed an *incremental nearest neighbor operation*) [16]. This operation may optionally receive a maximum distance, D, in which case it will stop when the distance to the next nearest neighbor is greater than D. Thus, it returns all neighbors within D of a query point in increasing distance.

The x^{th} plans, labeled Px_C and Px_A, use the classification approach and the abstraction approach, respectively. In this section we only sketch the plans. For detailed plans for these queries, see the Appendix where associated with each line is the cost of the appropriate operations.

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 multidimensional 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 scenic view within 5 miles of a picnic site.

Finding a plan for Q2 gives rise to many query optimization issues within the domain of symbolic-image databases. Most of these issues are also applicable to spatial databases [1]. While query optimization for relational databases has received much attention (e.g. [33]) and some work exists regarding query optimization for object-oriented databases (e.g. [40]), the issue of query optimization in spatial database has hardly been studied. To illustrate just how complex this issue may be, and to see how it is effected by the two approaches, 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 "picnic" and "scenic view" tuples using the alphanumeric index on `class`. For each "picnic" tuple, check all "scenic view" tuples to see which ones are within 5 miles.

```
get all tuples of logical_images corresponding to "picnic" (use index li_cl)
for each such tuple t1
   get all tuples of logical_images corresponding to "scenic view" (use index li_cl)
   for each such tuple t2
      if distance between t1 and t2 $\leq$ 5 miles
         and they are in the same image then
         display corresponding physical image
```

Plan P2B$_C$ Search for "picnic" tuples using the alphanumeric index on `class`. Search for all tuples within 5 miles of each "picnic" tuple using the spatial index on `loc`, and see which are "scenic views".

```
get all tuples of logical_images corresponding to "picnic" (use index li_cl)
for each such tuple t
   get all points within 5 miles of t.loc
      (using the incremental nearest neighbor operation)
   for each one of these points p
      if p is a ''scenic view'' and in the same image as t then
         display the corresponding physical image
```

Plan P2C$_C$ Search for "scenic view" tuples using the alphanumeric index on `class`. Build a temporary spatial index on the `loc` attribute of these tuples. Search for "picnic" tuples using the alphanumeric index on `class`, and search for "scenic view" tuples within 5 miles of each "picnic" using the temporary index.

```
create a PMR quad tree for points tmp_loc
   having the same properties as the index on loc
```

```
get all tuples of logical_images corresponding to "scenic view" (use index li_cl)
for each such tuple t
    insert it into tmp_loc
get all tuples of logical_images corresponding to "picnic" (use index li_cl)
for each such tuple t
    get all points within 5 miles of t.loc in tmp_loc
    for each one of these points p that are in the same image as t
        display the corresponding physical image
```

Plan P2A$_A$ Search for tuples corresponding to "picnic" and "scenic view" feature vectors using the multidimensional index on `fv`. For each "picnic" tuple, check all "scenic view" tuples to see which ones are within 5 miles.

```
get all tuples of logical_images whose fv field is within MD
    of the feature vector of "picnic" (use index li_fv)
for each such tuple t1
    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 t2
        if distance between t1 and t2 $\leq$ 5 miles
            and they are in the same image then
            display the corresponding physical image
```

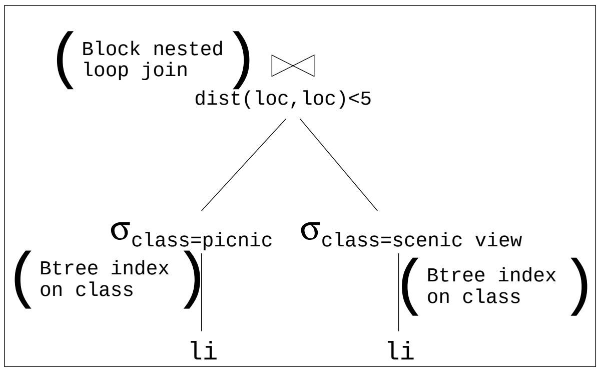
Plan P2B$_A$ Search for tuples corresponding to "picnic" feature vectors using the multidimensional index on `fv`. Search for all tuples within 5 miles of each "picnic" tuple using the spatial index on `loc`, and see which ones correspond to "scenic view" feature vectors.

```
get all tuples of logical_images whose fv field is within MD
        of the feature vector of "picnic" (use index li_fv)
for each such tuple t
    get all points within 5 miles of t.loc
        (using the incremental nearest neighbor operation)
    for each one of these points p
        if p.fv is within MD of ''scenic view'' feature vector
            and in the same image as t then
            display the corresponding physical image
```

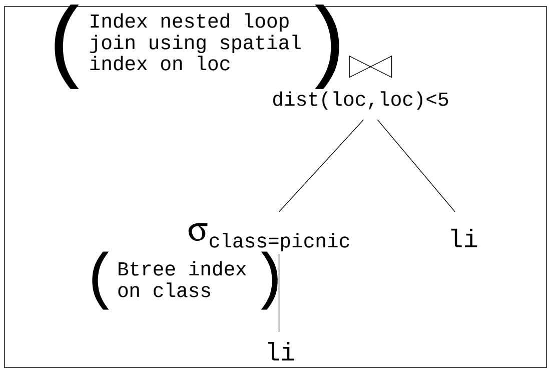
Plan P2C$_A$ Search for tuples corresponding to "scenic view" feature vectors using the multidimensional index on `fv`. Build a temporary spatial index on the `loc` attribute of these tuples. Search for tuples corresponding to "picnic" tuples using the multidimensional index on `fv`, and search for tuples corresponding to "scenic view" feature vectors within 5 miles using the temporary index.

```
create a PMR quad tree for points tmp_loc
    having the same properties as the index on loc
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
    insert it into tmp_loc
get all tuples of logical_images whose fv field is within MD
    of the feature vector of "picnic" (use index li_fv)
for each such tuple t
    get all points within 5 miles of t.loc in tmp_loc
```

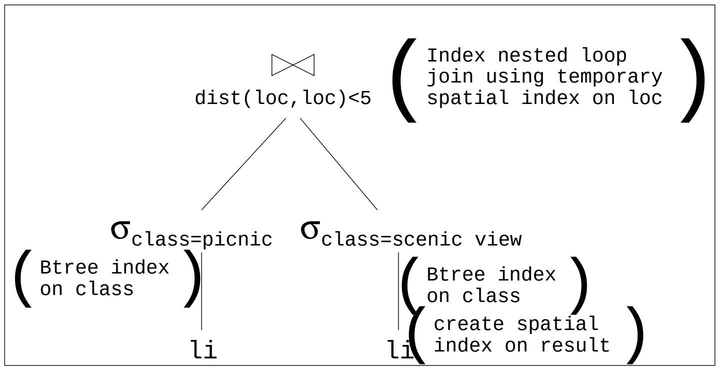
```
for each one of these points p
    if it is in the same image as t then
        display the corresponding physical image
```



(a)



(b)



(c)

Figure 10: Query evaluation plans for query Q2 using the classification approach. (a) plan P2A$_C$, (b) plan P2B$_C$, (c) plan P2C$_C$.

Figure 10 depicts these plans using Relational Algebra expression trees annotated with the processing strategy for each operation. Note that these query evaluation plans only contain the portion of the query that accesses the `logical_images` relation in order to retrieve the tuples that satisfy the query. The access to the `physical_images` relation in order to get the actual image for the "display" operation is not included since it is not considered part of processing the query. It is only a mechanism to output the answer to the query and is always the same regardless of the selected execution plan.

5.3 Cost Analysis

In this section we present a cost analysis of the various query execution plans given in the previous section. The purpose of this analysis is to compare the two approaches analytically as well as to demonstrate the difference in performance while processing queries that have both spatial and non-spatial components using different query processing strategies. As part of this analysis, we point out several important factors in query execution plan generation for such queries. In particular,

Name	*Meaning*
c_r	accessing a tuple by tid (random order)
c_{sq}	accessing a tuple in sequential order
c_{sqf}	accessing the first tuple of a relation
c_{af}	"find first" operation on an alphanumeric index
c_{an}	"find next" operation on an alphanumeric index
c_{lsf}	"find nearest neighbor" operation on a location space index
c_{lsn}	"find next nearest neighbor" operation on a location space index
c_{fsf}	"find nearest neighbor" operation on a feature space index
c_{fsn}	"find next nearest neighbor" operation on a feature space index
c_{sc}	string comparison
c_{lsd}	distance computation in location space
c_{fsd}	weighted distance computation in feature space
c_{mi}	inserting a tuple into a memory-resident relation
c_{msq}	accessing a tuple in a memory-resident relation sequentially
c_{lsc}	creating a location space index
c_{lsi}	inserting a point into a location space index

Table 1: Costs of basic operations used in query processing.

we outline how to compute spatial selectivity factors and and how to use them for selecting the appropriate plan for each query.

In order to estimate the costs of each plan, we must make some assumptions about the data distribution and the costs of the various operations. Table 1 contains a tabulation of the costs of basic operations that are used in processing queries. The cost of many of these operations is a function of the particular relation on which they operate. $c_{x(y)}$ denotes the cost of performing operation x on relation or index y. `li` stands for `logical_images`. The cost of accessing the physical_images relation in order to retrieve the result image and the cost of the "display" operation are not included since it is always the same regardless of the selected execution plan. Let N_{pic} and N_{sv} denote the number of tuples from class "picnic" and "scenic view", respectively. Let B_{pic} and B_{sv} denote the number of disk blocks containing tuples from class "picnic" and "scenic view", respectively.

Equations 1, and 2 estimate the cost of responding to query 1 using the classification approach and the abstraction approach, respectively.

$$C_{1_C} = c_{af(li\_cl)} + N_{sv} \times (c_{r(li)} + c_{an(li\_cl)}) \qquad (1)$$
$$C_{1_A} = c_{fsf(li\_fv)} + N_{sv} \times (c_{r(li)} + c_{fsn(li\_fv)}) \qquad (2)$$

The difference between C_{1_C} and C_{1_A} is that in the abstraction approach a location-space spatial "find first" operation replaces the alphanumeric "find first" operation of the classification approach. Likewise, a location-space spatial "find next" operation replaces the alphanumeric "find next" operation. The reason for this is that in the classification approach the "scenic view" tuples are found by searching the `logical_images` relation using the alphanumeric index on the `class` attribute. On the other hand, in the abstraction approach the "scenic view" tuples are found by searching the `logical_images` relation using the spatial index on the `fv` attribute.

Equations 3 and 4 estimate the cost of responding to query 2 with plan P2A using the classification approach and the abstraction approach, respectively. N_{InC_2} denotes the average number of tuples in the circular range specified in query 2 (C_2). $N_{sv\_InC_2}$ denotes the average number of scenic view tuples in C_2.

Assuming a uniform distribution of symbols in space (i.e., there is an equal number of symbols in any given area), then $N_{InC_2} = \frac{area(C_2)}{A} \times N$, where N is the total number of tuples in the `logical_images` relation, A is the area covered by these tuples, and C_2 is the circular range specified in query 2. Assuming a uniform distribution of classifications among the symbols (i.e., there is an equal number of symbols from each classification in any group of symbols), then $N_{sv\_InC_2} = \frac{area(C_2)}{A} \times \frac{N}{CL}$, where CL is the number of different classifications in the database.

If these assumptions about the distribution of the classifications among symbols do not hold, then other methods are required to estimate the number of scenic view tuples in a given area. The portion of all tuples that belong to each classification can be recorded when populating the database by checking the `class` attribute and tallying the number for each classification. This data can then be used to estimate the distribution of the classifications among the symbols. Assuming that the distribution of classifications among any group of symbols is equal to the the total database distribution (i.e., the portion of tuples from each classification among any given group of symbols is equal to the portion of tuples from each classification in the entire database), then $N_{sv\_InC_2} = \frac{area(C_2)}{A} \times sv_p \times N$, where sv_p is the portion of the database tuples that belong to the "scenic view" class.

Plan P2A$_C$ performs a spatial join operation on the results of two selection operations on relation `logical_images`. The first select operation extracts all tuples of the relation that are of class "picnic", while the second select operation extracts all tuples of the relation that are of class "scenic view". The results of these two select operations are then joined according to a predicate based on the `loc` attribute. In our implementation of plan P2A$_C$, we perform the select and join operations simultaneously using a block nested loop join algorithm as follows. One of the classes is designated as the *inner class*, and the other is designated as the *outer class*. One block of tuples belonging into the outer class are read into a memory-resident buffer (using the index on attribute `class`). All tuples of the inner class are then read (one block at a time using the index on attribute `class`) and spatially joined with all tuples of the outer class that are in memory (by computing the predicate on the spatial attribute). This process is repeated with the next block of tuples of the outer class, until all tuples of the outer class have been read.

$$\begin{aligned}
\mathrm{C_{2A_C}} &= c_{af(li\_cl)} + N_{pic} \times (c_{r(li)} + c_{an(li\_cl)}) + && \text{read all pic tuples} \quad (3)\\
&\quad B_{pic} \times [c_{af(li\_cl)} + N_{sv} \times (c_{r(li)} + c_{an(li\_cl)})] + && \text{for each pic block, read all sv tuples}\\
&\quad N_{pic} \times N_{sv} \times c_{lsd} && \text{check distance betweem each (pic, sv) pair}\\
C_{2A_A} &= c_{fsf(li\_fv)} + N_{pic} \times (c_{r(li)} + c_{fsn(li\_fv)}) + && \text{read all pic tuples} \quad (4)\\
&\quad B_{pic} \times [c_{fsf(li\_fv)} + N_{sv} \times (c_{r(li)} + c_{fsn(li\_fv)})] + && \text{for each pic block, read all sv tuples}\\
&\quad N_{pic} \times N_{sv} \times c_{lsd} && \text{check distance between each (pic, sv) pair}
\end{aligned}$$

The difference between C_{2A_C} and C_{2A_A} is that in the the classification approach, the `logical_images` relation is searched using the alphanumeric index on the `class` attribute, whereas in the abstraction approach, the `logical_images` relation is searched using the spatial index on the `fv` attribute (as in the case of query 1). As a result, in the abstraction approach a location-space spatial "find first" operation replaces the alphanumeric "find first" operation of the classification approach. Likewise, a location-space spatial "find next" operation replaces the alphanumeric "find next" operation.

Notice that the choice of which class is designated as the outer class and which class is designated as the inner class is significant. In our case we arbitrarily designated class "picnic" as the outer class. However, by examining equation 3 it is apparent that the class with less instances in the database should be designated as the outer class. Similarly, from Equation 4 it is apparent that for plan

P2A$_A$ (the abstraction approach) once again the class with less instances in the database should be designated as the outer class. Note, however, that in the abstraction approach the number of instances of each class is not as readily available as it is in the classification approach. The assignment to a particular class is only determined at query execution time by comparing the `fv` attribute of a tuple to that of an example feature vector of a symbol belonging to the required classification. Thus, some external statistics or estimates of the number of instances of each classification in the application is required.

Equations 5 and 6 estimate the cost of responding to query 2 with plan P2B using the classification approach and the abstraction approach, respectively.

$$\begin{aligned} \mathrm{C}_{2\mathrm{B}_\mathrm{C}} &= c_{af(li\_cl)} + N_{pic} \times [c_{r(li)} + c_{an(li\_cl)}] && \text{read all pic tuples} \quad (5)\\ &\quad N_{pic} \times && \text{for each pic tuple,}\\ &\quad [c_{lsf(li\_loc)} + N_{InC_2} \times (c_{r(li)} + c_{sc} + c_{lsn(li\_loc)})] && \text{find sv tuples in range using index li\_loc}\\ C_{2B_A} &= c_{fsf(li\_fv)} + N_{pic} \times [c_{r(li)} + c_{fsn(li\_fv)}] && \text{read all pic tuples} \quad (6)\\ &\quad N_{pic} \times && \text{for each pic tuple,}\\ &\quad [c_{lsf(li\_loc)} + N_{InC_2} \times (c_{r(li)} + c_{fsd} + c_{lsn(li\_loc)})] && \text{find all sv tuples in range} \end{aligned}$$

One difference between C_{2B_C} and C_{2B_A} is that in the classification approach, the `logical_images` relation is searched using the alphanumeric index on the `class` attribute, whereas in the abstraction approach, the `logical_images` relation is searched using the spatial index on the `fv` attribute (as in the case on plan P2A). However, in this case, there is another significant difference between the two. In the classification approach, once a tuple `t` is retrieved a string comparison (cost c_{sc}) is required in order to determine if the `class` attribute of `t` corresponds to a "scenic view". On the other hand, in the abstraction approach, a weighted distance computation (cost c_{fsd}) between the `fv` field of `t` and the example feature vector of "scenic view" is required. The cost of this computation (c_{fsd}) is much higher than the cost of a string comparison (c_{sc}), and thus C_{2B_A} will be significantly greater than C_{2B_C}. In addition, N_{InC_2} (the average number of tuples in the circular range specified in query 2) is larger in the abstraction approach because the `logical_images` relation in the abstraction approach contains tuples that correspond to both valid and invalid symbols, while in the classification approach only symbols that were classified by the image input system as valid are inserted into the database. Recall, that N_{InC_2} is estimated by $\frac{area(C_2)}{A} \times N$, where N is the total number of tuples in the `logical_images` relation, A is the area covered by these tuples, and C_2 is the circular range specified in query 2. Since C_2 and A are the same in both the classification approach and the abstraction approach, and since N is larger in the abstraction approach, N_{InC_2} is larger in the abstraction approach.

It is interesting to compare the costs of answering query 2 for each approach using plans P2A and P2B. For the classification approach, we compare equations 3 and 5. In plan P2A$_C$, both relations are scanned sequentially via the alphanumeric index `li_cl`. For each picnic tuple, each scenic view tuple is checked to determine whether or not it is within the specified range. Thus, the total number of distance computations is $N_{pic} \times N_{sv}$. In addition, the same number of random access operations are also required in order to get the locations from the `logical_images` relations. In plan P2B$_C$, the spatial index is used and thus only tuples that are within the specified range need to be examined. The cost of this is the overhead involved in using the spatial index. In this case, this cost is N_{pic} location-space "find first" operations, and $N_{pic} \times N_{InC_2}$ location-space "find next" operations. These spatial operations involve distance computations as part of the incremental nearest neighbor operation. However, there is no need for any distance computations as part of the plan itself. Whether plan P2A$_C$ or plan P2B$_C$ is better depends on the size of the data set, the portion of these tuples that belong to each classification (termed the *contextual selectivity*), and on the portion of all tuples that fall in the range specified by the spatial component (termed the *spatial selectivity*). Assuming a high spatial selectivity (i.e., that the number of tuples in the spatial range is much smaller than the total number of tuples in the data set), plan P2B$_C$ should prove to be much more efficient than plan P2A$_C$. However, if the spatial selectivity is low, then plan P2A$_C$ may prove to be better.

In the abstraction approach, the difference between the cost of plan P2A$_A$ and plan P2B$_A$ may not be as significant. The reason for this is that in plan P2A$_A$ the spatial index on the `fv` field is used to extract all the "scenic view" tuples, whereas in plan P2B$_A$ a relatively expensive weighted distance operation is required in order to determine if a given tuple is a "scenic view". However, if the spatial selectivity is high (i.e., a small number of tuples fall in the range specified by the query), and the contextual selectivity is low (i.e., a large number of the database tuples are of the required classification), then plan P2B$_A$ should outperform plan P2A$_A$ in the abstraction approach as well.

Equations 7, and 8 estimate the cost of responding to query 2 with plan P2C using the classification approach and the abstraction approach, respectively. The difference between plan P2C and plan P2B is that in plan P2C an intermediate spatial data structure that contains only those tuples that are of the desired classification is built before performing the spatial join.

$$\begin{aligned}
\mathrm{C}_{2\mathrm{C}_C} = \; & c_{lsc} + & & \text{create temporary spatial index} \quad (7)\\
& c_{af(li\_cl)} + N_{sv} \times (c_{r(li)} + c_{lsi(tmp\_loc)} + c_{an(li\_cl)}) + & & \text{insert sv tuples into temp index}\\
& c_{af(li\_cl)} + N_{pic} \times [c_{r(li)} + c_{an(li\_cl)}] + & & \text{get all pic tuples}\\
& N_{pic} \times & & \text{for each pic tuple}\\
& [c_{lsf(tmp\_loc)} + N_{sv\_InC_2} \times (c_{r(li)} + c_{lsn(tmp\_loc)})] & & \text{get tuples in range using temp index}\\
C_{2C_A} = \; & c_{lsc} + & & \text{create temporary spatial index} \quad (8)\\
& c_{fsf(li\_fv)} + N_{sv} \times (c_{r(li)} + c_{lsi(tmp\_loc)} + c_{fsn(li\_fv)}) + & & \text{insert sv tuples into temp index}\\
& c_{fsf(li\_fv)} + N_{pic} \times [c_{r(li)} + c_{fsn(li\_fv)}] + & & \text{get all pic tuples}\\
& N_{pic} \times & & \text{for each pic tuple}\\
& [c_{lsf(tmp\_loc)} + N_{sv\_InC_2} \times (c_{r(li)} + c_{lsn(tmp\_loc)})] & & \text{get tuples in range using temp index}
\end{aligned}$$

The difference in the cost of executing plan P2B and P2C can be seen by comparing C_{2B_C} and C_{2C_C} (Equations 5 and 7). In C_{2C_C} there is an added cost associated with creating and populating the spatial data structure with "scenic view" tuples. The cost of this is $c_{lsc} + c_{af(li\_cl)} + N_{sv} \times (c_{r(li)} + c_{lsi(tmp\_loc)} + c_{an(li\_cl)})$. On the other hand, the term $N_{InC_2} \times (c_{r(li)} + c_{sc} + c_{lsn(li\_loc)})$ is replaced with $N_{sv\_InC_2} \times (c_{r(li)} + c_{lsn(tmp\_loc)})$. Since the the number of "scenic view" tuples is most likely much smaller than the total number of tuples in the `logical_images` relation, the cost of building the temporary spatial data structure should pay off in the much more efficient spatial join due to the fact that the spatial index is smaller and contains only symbols of the desired class.

A more precise comparison is obtained as follows. Recall, that assuming a uniform distribution of symbols in space (i.e., there is an equal number of symbols in any given area), then $N_{InC_2} = \frac{area(C_2)}{A} \times N$, where N is the total number of tuples in the `logical_images` relation, A is the area covered by these tuples, and C_2 is the circular range specified in query 2. Assuming that the distribution of classifications among any group of symbols is equal to the the total database distribution (i.e., the portion of tuples from each classification among any given group of symbols is equal to the portion of tuples from each classification in the entire database), then $N_{sv\_InC_2} = \frac{area(C_2)}{A} \times sv_p \times N$, where sv_p

is the portion of the database tuples that belong to the "scenic view" class. Under these assumptions plan P2C will outperform plan P2B if the following holds:

$$N_{pic} \times [N_{InC_2} \times (c_{r(li)} + c_{sc} + c_{lsn(li\_loc)}) - N_{sv\_InC_2} \times (c_{r(li)} + c_{lsn(tmp\_loc)})] >$$

$$c_{lsc} + c_{af(li\_cl)} + N_{sv} \times (c_{r(li)} + c_{lsi(tmp\_loc)} + c_{an(li\_cl)})$$

Substituting values for N_{InC_2}, $N_{sv\_InC_2}$, and N_{sv} we get:

$$N_{pic} \times [\frac{area(C_2)}{A} \times N \times (c_{r(li)} + c_{sc} + c_{lsn(li\_loc)}) -$$

$$\frac{area(C_2)}{A} \times sv_p \times N \times (c_{r(li)} + c_{lsn(tmp\_loc)})] >$$

$$c_{lsc} + c_{af(li\_cl)} + sv_p \times N \times (c_{r(li)} + c_{lsi(tmp\_loc)} + c_{an(li\_cl)})$$

This inequality will not hold when sv_p is large (i.e., a low contextual selectivity), and when the search range C_2 is small (i.e., a high spatial selectivity). Therefore, when the contextual selectivity is high or the spatial selectivity is low, then plan P2C will outperform plan P2B.

When using the classification approach, Query Q2 can be further optimized by using a data organization in which the tuples are partitioned into separate relations resulting in a one-to-one correspondence between relations and classes of the application. For example, tuples $(C, certainty, (x, y))$ of a logical image for which $C = C_1$ are stored in a relation corresponding to C_1. Each such partition has a spatial index on the locations of the symbols from the corresponding class. Query $Q2$ is then computed by a spatial join operation between the two spatial indices with no need for creating temporary indices as is done in Plan P2C. This is termed a *partitioned organization* in contrast to the approach discussed in this paper which we term an *integrated organization.* See [38] for more details about the partitioned organization as well as a comparison with the integrated organization. The paritioned organization can not be used with the abstraction approach, since there is no mapping between tuples and classes using this approach.

Note that the cost estimates presented in this section are at the low level and quite specific for the particular queries and plans that we use. The purpose of these cost estimates was to compare the two approaches as well as to demonstrate the difference in performance while processing queries that have both spatial and non-spatial components using different query processing strategies. A more general cost analysis is a very complex issue and is beyond the scope of this paper.

6 Experimental Study

The symbolic-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 22 different 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 an image. Figure 5 is an example image. Of these 425 tiles, 280 contained at least one symbol. These 280 images contained 4111 symbols (both valid and invalid). Of these 4111 symbols, 1093 were valid symbols, and 3018 were invalid symbols.

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 (see

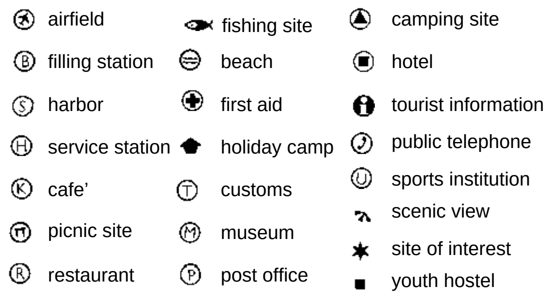


Figure 11: Symbols and their semantic meaning.

Figure 11 for a description of these symbols). 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 were inserted into the symbolic-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.

In order to compare the classification and abstraction approaches in terms of query execution time and in order to compare the various execution plans foe a given query, we created query execution plans for each one of the queries listed 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 [28] which is the query language used by SAND [11]. They were executed on a SPARC 10 running UNIX, and statistics regarding the execution were recorded.

Currently, we have only obtained data and scanned one portion of the map. While this data set was sufficient for testing the symbol recognition component of our system, .a larger data set is required in order to evaluate the system in terms of storage and retrieval from the database. In order to do this, we derived semi-synthetic data from the original data set for this purpose. This was done by replicating the map tiles by a constant factor M. Each original tile was replicated M times. The upper-left coordinate of these new tiles was computed so that it seems that the entire non-tiled image is replicated. The objects inside each image were relocated at random within the 512×512 area of the image. The values selected for M were 2, 4, 8, 16, 32, and 64, yielding multiples of the original non-tiled image of size 1×2, 2×2, 2×4, 4×4, 4×8, and 8×8, respectively. Therefore, the data sets with which we experimented consisted of 280, 560, 1120, 2240, 4480, 8960, and 17920 logical images. This corresponds to 1093, 2184, 4368, 8736, 17472, 34984, and 69952 tuples in the `logical_images` relation using the classification approach (since the basic data set yielded 1093 candidate classifications). When using the abstraction approach, the same data set had 4111, 8222, 16444, 32888, 65776, 131552, and 263104 tuples in the `logical_images` relation (since the basic data set has 4111 valid and invalid symbols).

In this section, we report results for various quantities that measure database storage and retrieval performance. These include image insertion time, storage space, retrieval accuracy, and query execution time. The results were collected for both the classification approach and the abstraction approach. In the section dealing with query execution time (i.e., Section 6.4) we compare the different approaches as well as the various execution plans for each one separately. Throughout the following discussion, CLA denotes the classification approach, and ABA denotes the abstraction approach.

6.1 Insertion Times

Figure 12 compares the time to insert the images into the database for various data-set sizes following the classification and abstraction approaches. These results are only for inserting the data. They do not include the time required for converting from physical to logical images, which is a time-consuming process (i.e., it took approximately 4 hours for the basic data set).

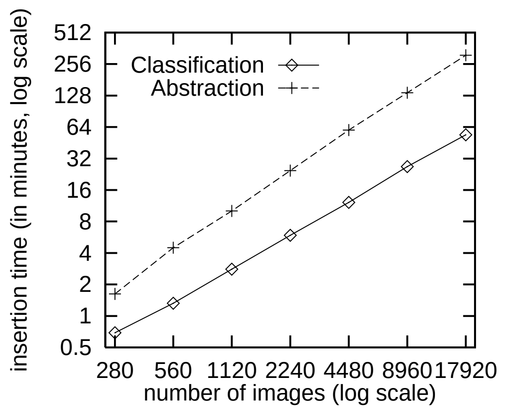


Figure 12: Time (in seconds) for insertion of the physical and logical images into the database for various sizes of the data set.

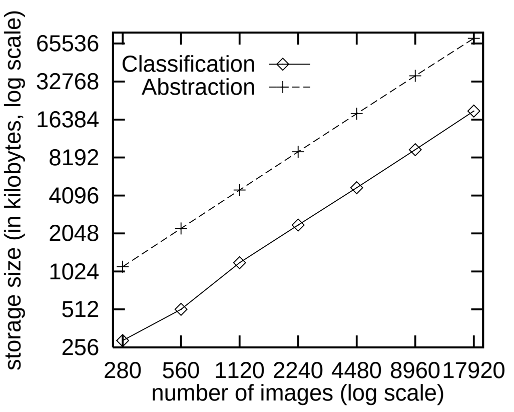


Figure 13: Sizes (in Kilobytes) of the relations for various data set sizes.

Inserting the data using ABA is slower than CLA. Inserting the entire data set takes 2.3 times longer for small data sets and as much as 5.7 times longer for large data sets. The reason for this is that there are four times as many tuples in ABA compared to CLA, since tuples corresponding to both valid and invalid symbols are stored in the symbolic-image database using ABA. Furthermore, constructing the k-d tree that is used to index the feature vectors in ABA is much more time-consuming than constructing the B-tree that is used to index the class names in CLA, especially for large data sets. Recall that an adaptive k-d tree is used for this purpose. The adaptive k-d tree is optimized for quick searches by the nature of its construction process. However, the cost of this is a quite complex as it involves selecting the best feature and computing the median of the value of this feature for all points each time the tree is split.

6.2 Storage Space

Figure 13 compares the sizes of the files that store and index the logical images in CLA and ABA for various data set sizes. The total space required for ABA is about four times as much as the space required for CLA. This corresponds precisely to the difference in the number of logical image tuples between CLA and ABA (1093 compared to 4111 in the basic data set).

6.3 Retrieval Accuracy

We evaluate the symbolic-image database system in terms of accuracy using two error types that are commonly used in document analysis studies [26]. Type I error occurs when an image that meets the query specification was not retrieved by the system (a *miss*), and a Type II error occurs when an image that the system retrieved for a given query does not meet the query specification (a *false hit*). Note that Type I and Type II errors correspond to the recall and precision metrics, respectively, used in information retrieval experiments [30]. Figure 14 compares the Type I and Type II error rates for the classification approach and the abstraction approach.

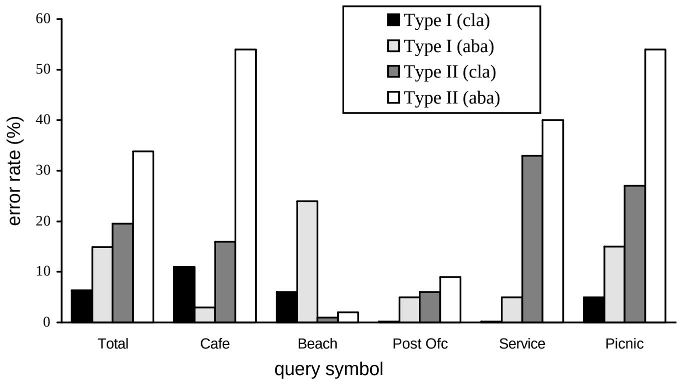


Figure 14: Type I and Type II error rates for the classification approach and the abstraction approach.

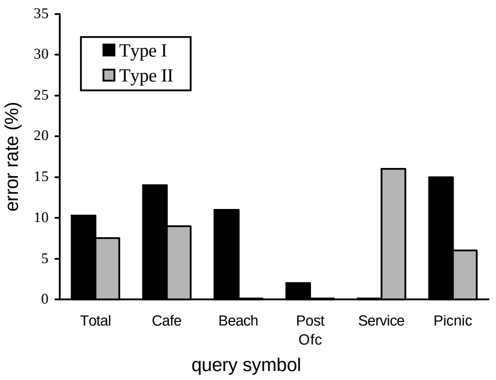


Figure 15: Type I and Type II error rates when considering only results with certainty value > 0.25 (classification approach).

The error rates were computed by performing a query that requests all images that contain one particular symbol. This was repeated for each of the symbols in our application. We counted how many results did not meet the query specification for each symbol, and how many result images were missed. We computed the error rates for each symbol. In addition, we computed the total Type I error rate as the total number of missed results divided by the total number of expected results. The total Type II error rate, which is the total number of incorrect results divided by the total number of results, was also computed.

The total Type I error rate using CLA was 6% (i.e., 94% of the images that should have been retrieved were in fact retrieved by the system). Note however, that this rate varies for the different symbols. The total Type II error rate was 19% (i.e., 81% of the images that were retrieved did in fact contain the desired symbol). Recall, that for the classification approach the results are ranked by certainty. The Type II error rate for the classification approach can be improved by considering only results with a certainty value greater than some cutoff value. This may increase the Type I error rate as some results that were correct may have a certainty value that is smaller than the chosen cutoff certainty value. See Figure 15 for a cutoff certainty value of 0.25 for the classification approach. In contrast, the certainty cutoff rate is irrelevant for the abstraction approach. Instead, here, the factor that effects the error rates is the search bound distance. Figure 16 reports the total Type I and Type II error rates, as well as these error rates for a few of the symbols using the abstraction approach, while varying the search bound value. Figure 17 plots the typeII error rate versus the type I error rate for various search bound values. From this graph it is apparent that the type II error rate decreases as the type I error rate increases.

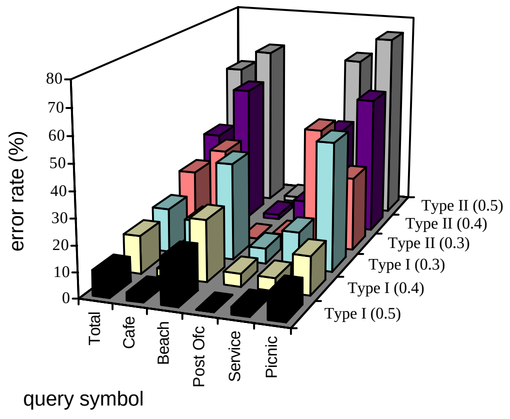


Figure 16: Type I and Type II error rates for symbols using the abstraction approach (the numbers in parenthesis are the search bound value).

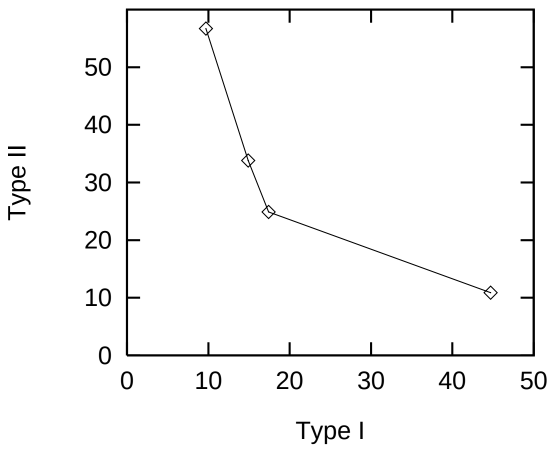


Figure 17: Total Type II error rate versus Type I error rate using the abstraction approach varying the search bound value.

The cause for these errors can only be attributed to the image input process (i.e., symbol classification or abstraction). Our image storage, indexing, and retrieval methods do not introduce any errors or approximations. The variance in the error rates between different symbols results from the specific classification method and from the contents of the training set that is used when inserting images into the database using CLA. In ABA this variance is attributed to the particular sample feature vector that is used for the search. In particular, in both approaches, the variance is attributed to the ability of the system to differentiate between different symbols. The results that we report here are for one particular training set (feature vector). However, we experimented with various training sets and sample feature vectors, and these results were consistent in all cases. In order to achieve lower error rates, more features would be required to describe each symbol.

Although the Type II error rates may seem rather high (i.e, a relatively large percent of the images retrieved did not conform to the query specification), the ranking of the results was very good – that is, the first result images in almost all cases did conform to the query specification. If the goal of the search is to find a few good result images (rather than finding all of them), then setting a high certainty value or a low search bound value and checking the first few results should give good results. If the goal is to find as many results as possible, then a low certainty value or a high search bound is required followed by manually weeding out the erroneous results. In any case, the number of result images as a fraction of the total number of images in the database was very low. Even for a the largest search bound value, only 6% of the images in the database were retrieved on average for the queries that we performed.

6.4 Query Execution Time

In order to compare the two approaches as well as the competing plans we performed an empirical comparison in terms of execution time between query Q1 using both approaches and between the three query execution plans for query Q2 (display all images that contain a scenic view within 5 miles of a picnic site) using the two approaches. This query has both a spatial and a non-spatial component and is thus a good basis for evaluation. The comparison was done by executing numerous variations of this query using the plans as outlined in Section 5. An important aspect in selecting the appropriate query execution plan is the selectivity of the various parts of the query. That is, what percent of the tuples of the relation conform to the spatial and non-spatial components of the query (we refer to these as the *spatial* and *contextual* selectivity factors, respectively). The variations of the query were selected so that we could test its execution under different levels of spatial and contextual selectivities. The spatial selectivity was changed by varying the search radius (i.e. the "within" distance). The contextual selectivity was varied by the particular selection of the symbols specified by the query. We repeated the queries for two cases. In the first case, the two symbols were chosen so that the contextual selectivity of their respective classes is similar (i.e., about the same percent of tuples of the entire data set belong to both symbol classes). In the second case, the symbols were chosen such that there are significantly more tuples that belong to one symbol class than to the other (i.e., one symbol had a very low contextual selectivity). Note that although the times that we report here are for a specific combination of symbols, we performed similar queries for numerous combinations and observed the same behavior. Therefore, the general trend will be the same, although the exact values of the execution times may differ.

We repeated these variations of this query for the various data-set sizes to see how they scale up. Two steps were taken in order to ensure that the data was in fact read from disk every time a query was executed (i.e., neutralize the effects of any buffering that the file system may perform). The first step was to clear the machine's memory by calling a routine that fills the entire memory with the value 0. The second step was to alternate the queries so that they were posed to different data sets. This ensured that the same data set was never referenced consecutively thereby requiring that it be read from the disk.

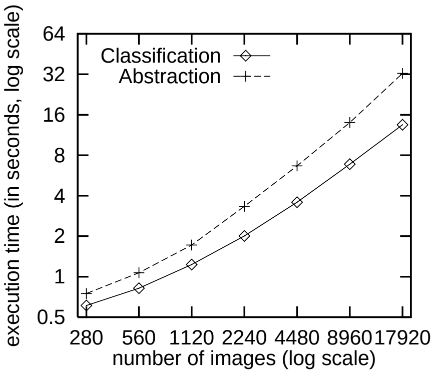


Figure 18: Retrieval times in seconds for Query Q1 for various data set sizes.

Figure 18 compares the retrieval times that were required to process query Q1 for various sizes of the data set. We repeated this query for each one of the symbols in our application. The results that we report are the average for all symbols. The execution time using ABA is approximately 1.2 times the execution time using CLA for small data sets, and approximately 2.4 times that for the large data set. There are two reasons for this difference in performance. One reason is that since ABA stores tuples that correspond to both valid and invalid symbols, the database is larger and thus the total number of operations that are required in order to find all symbols of a particular class is larger. The second reason is that in the case of ABA, this search is performed on the adaptive k-d tree and involves many weighted feature vector distance computations, while in the case of CLA, the search is performed on the B-tree and it involves string comparisons, which are a less costly operation.

Figures 19 and 20 report the retrieval time in seconds for query Q2 using the three plans for

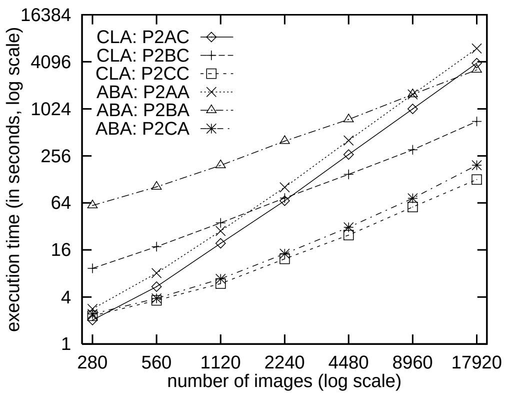


Figure 19: Retrieval time in seconds for Q2 with search radius 16 varying the data set size; small difference in contextual selectivity.

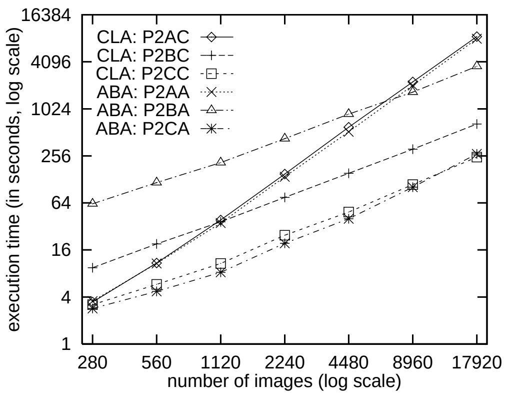


Figure 20: Retrieval time in seconds for Q2 with search radius 16 varying the data set size; large difference in contextual selectivity.

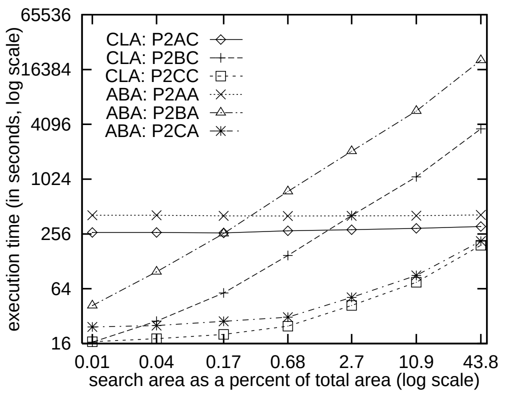


Figure 21: Retrieval time in seconds varying the search radius, for the 4480 image data set; small difference in contextual selectivity.

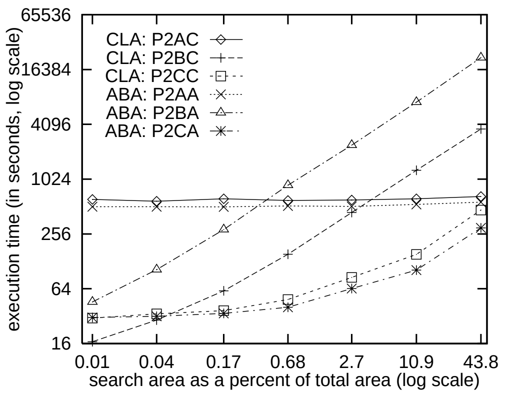


Figure 22: Retrieval time in seconds varying the search radius, for the 4480 image data set; large difference in contextual selectivity.

various data-set sizes for both approaches. Figure 19 corresponds to the case where the difference in the contextual selectivity between the query symbols is small, for progressively lower spatial selectivities (i.e., larger search radii). Similarly, Figure 20 corresponds to the case where the difference in the contextual selectivity between the query symbols is large. From these figures, it is apparent that plan P2C (which builds an intermediate spatial data structure) is best in most cases for both approaches. The cost of building the temporary spatial index pays off in the much more efficient spatial join needed to execute the query due to the fact that the spatial index results in far fewer pairs being compared.

Figures 21 and 22 show the retrieval time in seconds for query Q2 using the three plans varying the spatial selectivity for a constant data-set size. Here we can see that when the spatial selectivity is low (i.e., a large search area), plan P2B becomes very inefficient. This can also be seen in Figures 24 and 27 which show the retrieval time of plan P2B as both the search radius and the data set grows for CLA and ABA, respectively. Notice the steep slope as the size of the radius and the data set increases. In contrast, in Figures 23 and 26, which show the retrieval time of plan P2A as both

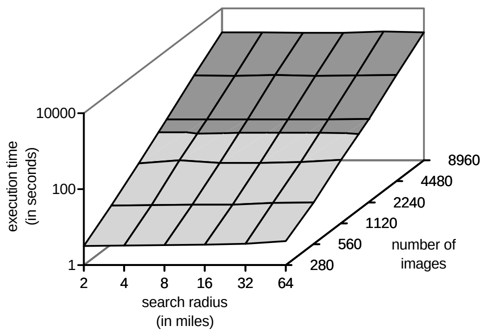


Figure 23: Retrieval time in seconds for Plan P2AC varying the search radius and the data set size (classification approach).

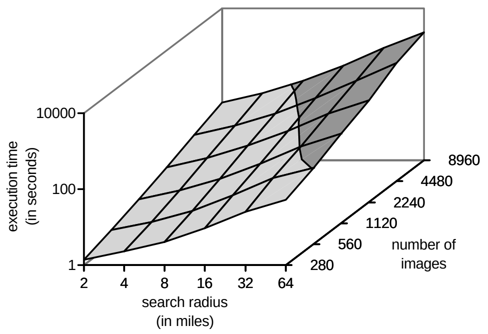


Figure 24: Retrieval time in seconds for Plan P2BC varying the search radius and the data set size (classification approach).

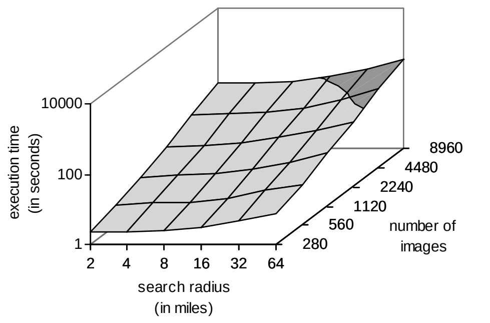


Figure 25: Retrieval time in seconds for Plan P2CC varying the search radius and the data set size (classification approach).

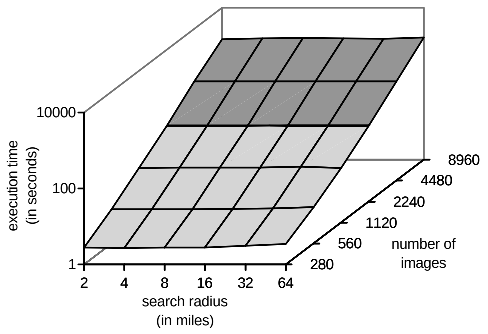


Figure 26: Retrieval time in seconds for Plan P2AA varying the search radius and the data set size (abstraction approach).

the search radius and the data set grows, the size of the search radius does not effect the retrieval time. The reason for this is that plan P2A simply computes the distance between every two symbols conforming to the contextual part of the query specification. Thus, it is not sensitive to the particular distance and it is attractive for low spatial selectivities.

Plan P2C seems to be the best compromise. It first uses the contextual index to only get the tuples that correspond to the classes specified by the query. It then builds a spatial index on these tuples and performs a spatial join between them. Even though some overhead is incurred when building this temporary index, in most cases this pays off. Plan P2C is sensitive to increasing both the search radius and the size of the data set. However, the changes are much more subtle compared to those of plan P2B as can be seen from the moderate slopes in Figures 25 and 28.

Finally, we compare the retrieval times for query Q2 using CLA and ABA. For plan P2A, the ratio of retrieval time using ABA over the retrieval time using CLA is about 1.5. For plan P2B, the ratio of retrieval time using ABA over the retrieval time using CLA varies between 3 and 7 depending on the value of the search radius, the size of the data set, and whether there is a large or small difference in the contextual selectivities. There are two reasons for the poor relative performance of P2B using ABA. One reason is that since ABA stores tuples that correspond to both valid and invalid symbols, more tuples are found in each range search, and thus the total number of comparison operations that are required in order to determine whether the symbol is of the desired class is larger.

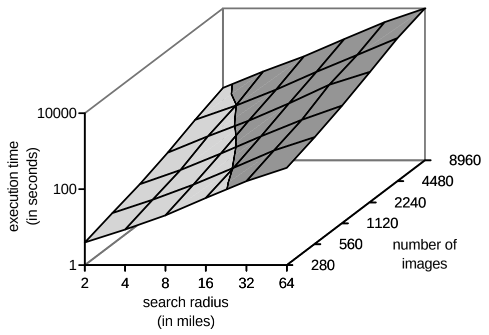


Figure 27: Retrieval time in seconds for Plan P2BA varying the search radius and the data set size (abstraction approach).

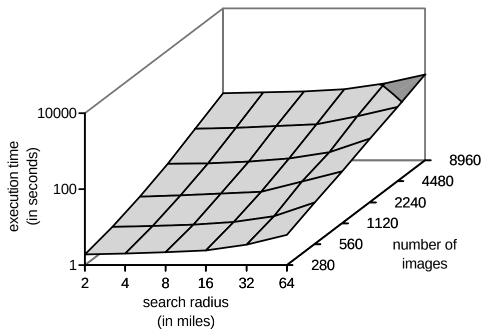


Figure 28: Retrieval time in seconds for Plan P2CA varying the search radius and the data set size (abstraction approach).

The second reason is that this comparison, in the case of ABA, involves a weighted feature vector distance computation while in the case of CLA, it involves a string comparison, which is a less costly operation. For plan P2C, the ratio of retrieval time using ABA over the retrieval time using CLA is a little more than 1. In plan P2C, only tuples whose classification matches the classes of the query-symbols are retrieved. Therefore, the impact of the larger number of total tuples in the database in ABA, is very small, and the total retrieval time is very similar.

6.5 Discussion

Our results show that for both data organizations (i.e, CLA and ABA), plans that use a spatial index perform much better than plans that do not make use of such an index. In addition, for all but the smallest data sets, it was beneficial to build a temporary spatial index that contained only those tuples that belonged to the required classification, as is done by plan P2C. If only query retrieval time is considered, then clearly CLA is always better than ABA. However, in comparing these two approaches one must consider a broad spectrum of factors, where query retrieval time is only one of them. Table 2 outlines these differences.

	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 2: Comparison of the classification approach and the abstraction approach.

The image preprocessing workload in CLA is heavier than in ABA. In both cases, the images are segmented, and a feature vector is computed for each connected component. In CLA, this is followed by a classification process resulting in a more time-consuming preprocessing step in CLA compared to ABA. In terms of the time to insert the images into the database after the preprocessing is complete, ABA is slower than CLA. In terms of user interaction during the image interpretation process, in ABA there is no need for human interaction. On the other hand, in CLA the user must assist in creating the training set which is used to automatically classify the images.

Retrieving queries purely on the basis of content (with no spatial constraints, e.g., example query Q1) is slightly slower in ABA than in CLA. Our results varied from a retrieval time that was greater by a factor of 1.2 for the small data set, to a retrieval time that was greater by a factor of 2.4 for the large data set. A similar observation was made for hybrid queries (e.g., query Q2).

The empirical results presented here were for two generic types of queries. The first query searches for images based on content only, while the second query searches for symbolic images based on both contextual and spatial constraints. In particular, the second query searches for two given symbols within a certain distance of each other. In addition to these queries, we have also experimented with queries that involve relational-spatial constraints such as display all images that contain an airfield northeast of a beach. Spatial indices are not as efficient for such queries, as the spatial selectivity of the relational-spatial constraint is rather small. That is, on the average a relatively large number (1/4th for northeast) of the symbols will satisfy the relational-spatial constraint. Therefore, unless there is an additional locational-spatial constraint, it is most likely best to compute such a query based on the contextual constraints. In fact, we observed performance similar to pure contextual queries for this query. That is, CLA had a slight advantage over ABA. In general, most queries will be a variation of these two basic queries; that is, they are either based on content, or they have an additional spatial constraint. Thus, the results presented here should hold for other applications that deal with symbolic images.

In terms of flexibility, ABA has a few advantages. The first advantage is that it is applicable for a larger number of applications. CLA is only applicable when all the classes of the application are known in advance. In addition, a classifier that can classify these symbols is required. In contrast, ABA is also applicable in cases where all the classes of the application are not known in advance (although a sample feature vector is still required to execute queries). In addition, there is no need for a classifier. All that is needed is a function that returns the certainty that two symbols belong to the same class. This function can be realized simply via the Euclidean distance between two feature vectors with an appropriate normalization.

The second advantage of ABA in terms of flexibility is that it is more adaptive at run time. In CLA, the parameters that determine the size of the window that is used to search the feature space are set at image insertion time. The results of the classification using these particular settings are inserted into the database, and there is no record of the raw data that was used to make these classifications. In ABA, on the other hand, this decision is only made at run time. The user may vary the size of the search window in feature space for different queries.

Finally, in terms of accuracy, CLA outperforms ABA in both Type I and Type II error rates. In our results, the difference was approximately twofold. One reason for this is that CLA's user involvement in the early stages of populating the symbolic-image database (when constructing the training set) pays off in terms of accuracy. Another reason is that CLA uses a sophisticated classifier that is presented with several example instances of each symbol. In contrast, ABA simply considers any symbol whose feature vector is within a predefined distance of one sample feature vector as belonging to the same class. This is in effect like a classifier that just sets a constant search bound for neighbors and has no voting process, with a training set that consists of only one instance of each class. Note, that the same features are used for both approaches. The only difference is in the number of instances in the training set and in the voting process.

The comparison of the CLA and ABA methods depends on the quality of both the particular features that are used, and the classification algorithm. However, assuming that both methods use the same features, these results should hold for other features as well. In our application we selected some global and some local shape descriptors that were empirically shown to effectively discriminate between geographic symbols [36]. Note that the quality of the classification is the only factor that effects accuracy. The indexing methods and the query execution plans that we employ do not use any approximations. Their results are always accurate.

7 Concluding Remarks

Throughout this paper we discussed how to input, store, index, and retrieve symbolic images in a database following the classification and the abstraction approaches. It is apparent that the choice of which approach to use depends on the application at hand. If the application consists of fully-symbolic images, and it is important to have more accurate results and quicker query response times, then CLA should be used. On the other hand, if the application consists of partially-symbolic images, and it is important to have flexibility at run time, then ABA should be used. For applications that fall in between these two extremes, the performance study that was presented in this paper can be used as a guideline for the appropriate selection.

The queries in this paper were specified textually. The system also has a graphical user interface (GUI) that we did not describe here (see [37]). In addition, we have incorporated a pictorial query specification that can deal with both contextual and spatial constraints [37]. In this case, a sample symbol for each class will not be required when using ABA, as it can be taken directly from the pictorial query. The example spatial queries that we presented in this paper mainly involved distance. However, by explicitly storing the locations of objects we can easily derive any required relative spatial relation such as “left of”, “above”, etc. at query time. Furthermore, using the spatial index we can devise optimized plans for such queries.

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 (i.e, storing locations and using spatial data structures to index the locational data) we can handle objects with spatial extent just as easily. The only difference would be in the selection of the spatial data structure. For example, if symbols are represented by bounding boxes, then any data structure that is suitable for indexing rectangles such as an R-tree [15] 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 [7]. Note that we have used similar methods for the interpretation of floor plans [31].

8 Acknowledgments

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

References

[1] W. G. Aref and H. Samet. Optimization strategies for spatial query processing. In G. Lohman, editor, *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 81–90, Barcelona, September 1991.

[2] Y. A. Aslandogan, C. Thier, C. T. Yu, C. Liu, and K. R. Nair. Design, implementation and evaluation of SCORE (a System for COntent based REtrieval of pictures). In *Eleventh International Conference on Data Engineering*, pages 280–287, Taipei, Taiwan, March 1995.

[3] A. Del Bimbo, M. Mugnaini, P. Pala, and F. Turco. PICASSO: Visual querying by color perceptive regions. In *Proceedings of the Second International Conference on Visual Information Systems*, pages 125–131, San Diego, California, December 1997.

[4] A. Del Bimbo and E. Vicario. Weighting spatial relationships in retrieval by visual contents. In *Proceedings of the IFIP 2.6 4th Working Conference on Visual Database Systems (VDB-4)*, pages 277–292, L'Aquila, Italy, May 1998.

[5] J.L. Blue, G.T. Candela, P.J. Grother, R. Chellappa, and C.L. Wilson. Evaluation of pattern classifiers for fingerprints and OCR applications. *Pattern Recognition*, 27(4):485–501, April 1994.

[6] S. K. Chang, E. Jungert, and Y. Li. The design of pictorial databases based upon the theory of symbolic projections. In A. Buchmann, O. Günther, T. R. Smith, and Y. F. Wang, editors, *Design and Implementation of Large Spatial Databases — First Symposium, SSD'89*, pages 303–323, Santa Barbara, CA, July 1989. (Also Springer-Verlag Lecture Notes in Computer Science 409).

[7] 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.

[8] S. K. Chang, C. W. Yan, D. C. Dimitroff, and T. Arndt. An intelligent image database system. *IEEE Transactions on Software Engineering*, 14(5):681–688, May 1988.

[9] T. Dalamaga, T. Sellis, and L. Sinos. A visual database system for spatial and non-spatial data management. In *Proceedings of the IFIP 2.6 4th Working Conference on Visual Database Systems (VDB-4)*, pages 105–122, L'Aquila, Italy, May 1998.

[10] C. Djeraba, M. Bouet, and H. Briand. Content-based query and indexing. In *Proceedings of the Second International Conference on Visual Information Systems*, pages 69–76, San Diego, California, December 1997.

[11] C. Esperança and H. Samet. Spatial database programming using SAND. In M. J. Kraak and M. Molenaar, editors, *Proceedings of the Seventh International Symposium on Spatial Data Handling*, volume 2, pages A29–A42, Delft, The Netherlands, August 1996. International Geographical Union Comission on Geographic Information Systems, Association for Geographical Information.

[12] 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.

[13] 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.

[14] 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.

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

[16] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In M. J. Egenhofer and J. R. Herring, editors, *Advances in Spatial Databases — Fourth International Symposium, SSD'95*, pages 83–95, Portland, ME, August 1995. (Also Springer-Verlag Lecture Notes in Computer Science 951).

[17] Informix. Informix web site. http://www.informix.com.

[18] 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.

[19] S. Kaushik and E. A. Rudensteiner. Direct manipulation spatial exploration using SVIQUEL. In *Proceedings of the IFIP 2.6 4th Working Conference on Visual Database Systems (VDB-4)*, pages 179–185, L'Aquila, Italy, May 1998.

[20] S. Y. Lee and F. J. Hsu. 2D C-string: a new spatial knowledge representation for image database systems. *Pattern Recognition*, 23(10):1077–1088, October 1990.

[21] M.D. Levine. *Vision in Man and Machine.* McGraw-Hill, New York, 1982.

[22] M. S. Lew, K. Lempinen, and N. Huijsmans. Webcrawling using sketches. In *Proceedings of the Second International Conference on Visual Information Systems*, pages 77–84, San Diego, California, December 1997.

[23] 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.

[24] 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).

[25] 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 *Proceedings of the SPIE, Storage and Retrieval of Image and Video Databases*, volume 1908, pages 173–187, San Jose, CA, February 1993.

[26] L. O'Gorman and R. Kasturi, editors. *Document Image Analysis.* IEEE Computer Society Press, Los Alamitos, CA, 1994.

[27] V. Oria, B. Xu, and M. T. Tamer. VisualMOQL: A visual query language for image databases. In *Proceedings of the IFIP 2.6 4th Working Conference on Visual Database Systems (VDB-4)*, pages 186–191, L'Aquila, Italy, May 1998.

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

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

[30] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1st edition, 1983.

[31] 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.

[32] H. Samet and A. Soffer. Magellan: Map acquisition of geographic labels by legend analysis. *International Journal of Document Analysis and Recognition*, 1(2):89–101, June 1998.

[33] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database system concepts.* McGraw-Hill, New York, 3rd edition, 1996.

[34] A. P. Sistla, C. Yu, C. Liu, and K. Liu. Similarity based retrieval of pictures using indices on spatial relationships. In *Proceedings of the 21st International Conference on Very Large Data Bases*, pages 619–629, Zurich, Switzerland, September 1995.

[35] J. R. Smith and S.-F. Chang. VisualSEEk: a fully automated content-based image query system. In *ACM International Conference on Multimedia*, pages 87–98, Boston, MA, November 1996.

[36] A. Soffer and H. Samet. Negative shape features for image databases consisting of geographic symbols. In C. Arcelli, L. P. Cordella, and G. Sanniti di Baja, editors, *Advances in Visual Form Processing*, pages 569–581, Singapore, 1997. World Scientific.

[37] A. Soffer and H. Samet. Pictorial query specification for browsing through spatially-referenced image databases. *Journal of Visual Langauges and Computing*, pages –, 1998. To appear.

[38] A. Soffer and H. Samet. Two data organizations for storing symbolic images in a relational database system. In *Semantic Issues in Multimedia Ststems*, Norwell, MA, 1999. Kluwer Academic Press. To Appear.

[39] 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*, pages 397–412, Singapore, June 1993. (Also Springer Verlag Lecture Notes in Computer Science 692).

[40] D.D. Straube and M.T. Özsu. Query optimization and execution plan generation in object-oriented database systems. *EEE Transactions on Knowledge and Data Engineering*, 7(2):210–227, April 1995.

[41] M. Swain. Interactive indexing into image databases. In *Proceedings of the SPIE, Storage and Retrieval for Image and Video Databases*, volume 1908, pages 95–103, San Jose, CA, February 1993.

[42] Excalibur Technologies. Excalibur technologies web site. http://www.excalib.com.

[43] 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.

[44] Virage. Virage web site. http://www.virage.com.

[45] D. White and R. Jain. Similarity indexing: Algorithms and performance. In *Proceedings of the SPIE, Storage and Retrieval of Still Image and Video Databases IV*, volume 2670, pages 62–73, San Jose, CA, February 1996.

9 Appendix: Sample Plans

The following are programs for the queries described in Section 5. Notice that the letters in italic at the end of each line represent the cost of executing this line in terms of the constants defined in Table 1. `NN` denotes "nearest neighbor". `li`, `pi`, and `cl` denote `logical_images`, `physical_images`, and `classes`, respectively. The cost of the "display" operation is not included since it is not considered part of processing the query. It is only a mechanism to output the answer to the query and is always the same regardless of the selected execution plan. In addition, no cost is associated with operations that compare two values (e.g., =, <).

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

```
cd ← first tuple t of cl_sem such that
          t.semant ≥ "scenic view"                              $c_{af(cl\_sem)}$
c ← tuple of classes corresponding to cd.tid                    $c_{r(cl)}$
lc ← first tuple t of li_cl such that t.class ≥ c.name          $c_{af(li\_cl)}$
while lc ≠ ERROR and lc.class = c.name do
    li ← tuple of logical_images corresponding to lc.tid        $c_{r(li)}$
    pi ← tuple t of physical_images such that
              t.img_id = li.img_id                              $c_{af(pi\_id)} + c_{r(pi)}$
    Display pi.raw
    lc ← next tuple of li_cl in alphabetic order                $c_{an(li\_cl)}$
endwhile
```

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

```
cd ← first tuple t of cl_sem such that
          t.semant ≥ "scenic view"                              $c_{af(cl\_sem)}$
c ← tuple of classes corresponding to cd.tid                    $c_{r(cl)}$
lv ← first tuple t of li_fv such that
          t.fv is the NN and within MD of c.fv                  $c_{fsf(li\_fv)}$
while lv ≠ ERROR do
    li ← tuple of logical_images corresponding to lv.tid        $c_{r(li)}$
    pi ← tuple t of physical_images such that
              t.img_id = li.img_id                              $c_{af(pi\_id)} + c_{r(pi)}$
    Display pi.raw
    lv ← tuple t of li_fv such that
              t.fv is the next NN and within MD of c.fv         $c_{fsn(li\_fv)}$
endwhile
```

Plan P2A$_C$: Search using alphanumeric indices on **class** for all picnic tuples and all scenic view tuples.

```
cds ← first tuple t of cl_sem such that
            t.semant ≥ "scenic view"                                    $c_{af(cl\_sem)}$
cs ← tuple of classes corresponding to cds.tid                          $c_{r(cl)}$
cdp ← first tuple t of cl_sem such that
            t.semant ≥ "picnic"                                         $c_{af(cl\_sem)}$
cp ← tuple of classes corresponding to cdp.tid                          $c_{r(cl)}$
lcp ← first tuple t of li_cl such that t.class ≥ cp.name                $c_{af(li\_cl)}$
create temporary memory-resident relations lcp_buf and lcs_buf
while lcp ≠ ERROR and lcp.class = cp.name do
    i ← 0
    while i < BS and lcp ≠ ERROR and
            lcp.class = cp.name do
        lip ← tuple of logical_images corresponding to lcp.tid          $c_{r(li)}$
        insert lip into lcp_buf                                         $c_{mi}$
        lcp ← next tuple of li_cl in alphabetic order                   $c_{an(li\_cl)}$
        i ← i + 1
    endwhile
    lcs ← first tuple t of li_cl such that t.class ≥ cs.name            $c_{af(li\_cl)}$
    while lcs ≠ ERROR and lcs.class = cs.name do
        j ← 0
        while j < BS and lcs ≠ ERROR and
                lcs.class = cs.name do
            lis ← tuple of logical_images corresponding to lcs.tid      $c_{r(li)}$
            insert lis into lcs_buf                                     $c_{mi}$
            lcs ← next tuple of li_cl in alphabetic order               $c_{an(li\_cl)}$
            j ← j + 1
        endwhile
        for all tuples cp of lcp_buf                                    $c_{msq}$
            for all tuples cs of lcs_buf                                $c_{msq}$
                if dist(cs.loc, cp.loc) ≤ 5 and                         $c_{lsd}$
                        cs.img_id = cp.img_id then
                    pi ← tuple t of physical_images such that
                        t.img_id = lis.img_id                           $c_{af(pi\_id)} + c_{r(pi)}$
                    Display pi.raw
                    endif
            endfor
        endfor
    endwhile
endwhile
```

Plan P2A$_A$ Search for picnic tuples and scenic view tuples using the spatial index on `fv`. For each picnic tuple, check all scenic view tuples to determine which ones are within the specified distance.

```
cds ← first tuple t of cl_sem such that
            t.semant ≥ "scenic view"                                    $c_{af(cl\_sem)}$
cs ← tuple of classes corresponding to cds.tid                          $c_{r(cl)}$
cdp ← first tuple t of cl_sem such that
            t.semant ≥ "picnic"                                         $c_{af(cl\_sem)}$
cp ← tuple of classes corresponding to cdp.tid                          $c_{r(cl)}$
lcp ← first tuple t of li_fv such that
            t.fv is the NN and within MD of cp.fv                       $c_{fsf(li\_fv)}$
```

```
create temporary memory-resident relations lcp_buf and lcs_buf
while lcp $\neq$ ERROR do
    i $\leftarrow$ 0
    while i < BS and lcp $\neq$ ERROR do
        lip $\leftarrow$ tuple of logical_images corresponding to lcp.tid          $c_{r(li)}$
        insert lip into lcp_buf                                                    $c_{mi}$
        lcp $\leftarrow$ tuple t of li_fv such that
                    t.fv is the next NN and within MD of cp.fv                     $c_{fsn(li\_fv)}$
        i $\leftarrow$ i + 1
    endwhile
    lcs $\leftarrow$ first tuple t of li_fv such that
            t.fv is the NN and within MD of cs.fv                                  $c_{fsf(li\_fv)}$
    while lcs $\neq$ ERROR do
        j $\leftarrow$ 0
        while j < BS and lcs $\neq$ ERROR do
            lis $\leftarrow$ tuple of logical_images corresponding to lcs.tid      $c_{r(li)}$
            insert lis into lcs_buf                                                $c_{mi}$
            lcs $\leftarrow$ tuple t  of li_fv such that
                        t.fv is the next NN and within MD of cs.fv                 $c_{fsn(li\_fv)}$
            j $\leftarrow$ j + 1
        endwhile
        for all tuples cp of lcp_buf                                               $c_{msq}$
            for all tuples cs of lcs_buf                                           $c_{msq}$
                if dist(cs.loc, cp.loc) $\leq$ 5 and                               $c_{lsd}$
                        cs.img_id = cp.img_id then
                    pi $\leftarrow$ tuple t of physical_images such that
                        t.img_id = lis.img_id                                      $c_{af(pi\_id)} + c_{r(pi)}$
                    Display pi.raw
                endif
            endfor
        endfor
    endwhile
endwhile
```

Plan P2B$_C$: Search for picnic tuples using an alphanumeric index on **class** and search for scenic view tuples using a spatial index on **loc**.

```
cds $\leftarrow$ first tuple t of cl_sem such that
            t.semant $\geq$ "scenic view"                                          $c_{af(cl\_sem)}$
cs $\leftarrow$ tuple of classes corresponding to cds.tid                          $c_{r(cl)}$
cdp $\leftarrow$ first tuple t of cl_sem such that
            t.semant $\geq$ "picnic"                                               $c_{af(cl\_sem)}$
cp $\leftarrow$ tuple of classes corresponding to cdp.tid                          $c_{r(cl)}$
lc $\leftarrow$ first tuple t of li_cl such that t.class $\geq$ cp.name            $c_{af(li\_cl)}$
while lc $\neq$ ERROR and lc.class = cp.name do
    lip $\leftarrow$ tuple of logical_images corresponding to lc.tid               $c_{r(li)}$
    ll $\leftarrow$ tuple t of li_loc such that
            t.loc is the NN and within 5 of lip.loc                                $c_{lsf(li\_loc)}$
    while ll $\neq$ ERROR do
        lis $\leftarrow$ tuple of logical_images corresponding to ll.tid           $c_{r(li)}$
        if lis.class = cs.name and lis.img_id = lip.img_id then                    $c_{sc}$
            pi $\leftarrow$ tuple t of physical_images such that
```

```
                  t.img_id = lip.img_id                          $c_{af(li\_id)} + c_{r(pi)}$
        Display pi.raw
      endif
      ll ← tuple t of li_loc such that
                t.loc is the next NN and within 5 of lip.loc      $c_{lsn(li\_loc)}$
    endwhile
    lc ← next tuple of li_cl in alphabetic order                  $c_{an(li\_cl)}$
endwhile
```

Plan P2B$_A$ Search for picnic tuples using spatial index on `fv` and search for scenic view tuples using a spatial index on `loc`.

```
cds ← first tuple t of cl_sem such that
          t.semant ≥ "scenic view"                                $c_{af(cl\_sem)}$
cs ← tuple of classes corresponding to cds.tid                    $c_{r(cl)}$
cdp ← first tuple t of cl_sem such that
          t.semant ≥ "picnic"                                     $c_{af(cl\_sem)}$
cp ← tuple of classes corresponding to cdp.tid                    $c_{r(cl)}$
lc ← first tuple t of li_fv such that
        t.fv is the NN and within MD of cp.fv                     $c_{fsf(li\_fv)}$
while lc $\neq$ ERROR do
    lip ← tuple of logical_images corresponding to lc.tid         $c_{r(li)}$
    ll ← tuple t of li_loc such that
        t.loc is the NN and within 5 of lip.loc                   $c_{lsf(li\_loc)}$
    while ll $\neq$ ERROR do
      lis ← tuple of logical_images corresponding to ll.tid       $c_{r(li)}$
      if wdist(lis.fv,cs.fv) ≤ MD and lis.img_id = lip.img_id then  $c_{fsd}$
        pi ← tuple t of physical_images such that
                  t.img_id = lip.img_id                           $c_{af(pi\_id)} + c_{r(pi)}$
        Display pi.raw
      endif
      ll ← tuple t of li_loc such that
                t.loc is the next NN and within 5 of lip.loc      $c_{lsn(li\_loc)}$
    endwhile
    lc ← tuple t of li_fv such that
            t.fv is the next NN and within MD of cp.fv            $c_{fsn(li\_fv)}$
endwhile
```

Plan P2C$_C$ Search for "scenic view" tuples using the alphanumeric index on `class`. Build a temporary spatial index on the `loc` attribute of these tuples. Search for "picnic" tuples using the alphanumeric index on `class`, and search for "scenic view" tuples using the temporary spatial index on `loc`.

```
cds ← first tuple t of cl_sem such that
          t.semant ≥ "scenic view"                                $c_{af(cl\_sem)}$
cs ← tuple of classes corresponding to cds.tid                    $c_{r(cl)}$
cdp ← first tuple t of cl_sem such that
          t.semant ≥ "picnic"                                     $c_{af(cl\_sem)}$
cp ← tuple of classes corresponding to cdp.tid                    $c_{r(cl)}$
create a PMR quad tree for points tmp_loc
    having the same properties as li_loc                          $c_{lsc}$
```

```
lc ← first tuple t of li_cl such that t.class ≥ cs.name                    $c_{af(li\_cl)}$
while lc ≠ ERROR and lc.class = cs.name do
    lip ← tuple of logical_images corresponding to lc.tid                  $c_{r(li)}$
    insert lip.loc into tmp_loc                                            $c_{lsi(tmp\_loc)}$
    lc ← next tuple of li_cl in alphabetic order                           $c_{an(li\_cl)}$
endwhile
lc ← first tuple t of li_cl such that t.class ≥ cp.name                    $c_{af(li\_cl)}$
while lc ≠ ERROR and lc.class = cp.name do
    lip ← tuple of logical_images corresponding to lc.tid                  $c_{r(li)}$
    ll ← tuple t of tmp_loc such that
                t.loc is the NN and within 5 of lip.loc                    $c_{lsf(tmp\_loc)}$
    while ll ≠ ERROR do
        lis ← tuple of logical_images corresponding to ll.tid              $c_{r(li)}$
        if lis.img_id = lip.img_id then
            pi ← tuple t of physical_images such that
                        t.img_id = lip.img_id                              $c_{af(pi\_id)} + c_{r(pi)}$
            Display pi.raw
        endif
        ll ← tuple t of tmp_loc such that
                    t.loc is the next NN and within 5 of lip.loc           $c_{lsn(tmp\_loc)}$
    endwhile
    lc ← next tuple of li_cl in alphabetic order                           $c_{an(li\_cl)}$
endwhile
```

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

```
cds ← first tuple t of cl_sem such that
            t.semant ≥ "scenic view"                                       $c_{af(cl\_sem)}$
cs ← tuple of classes corresponding to cds.tid                             $c_{r(cl)}$
cdp ← first tuple t of cl_sem such that
            t.semant ≥ "picnic"                                            $c_{af(cl\_sem)}$
cp ← tuple of classes corresponding to cdp.tid                             $c_{r(cl)}$
create a PMR quad tree for points tmp_loc
    having the same properties as li_loc                                   $c_{lsc}$
lc ← first tuple t of li_fv such that
            t.fv is the NN and within MD of cs.fv                          $c_{fsf(li\_fv)}$
while lc ≠ ERROR do
    lip ← tuple of logical_images corresponding to lc.tid                  $c_{r(li)}$
    insert lip.loc into tmp_loc                                            $c_{lsi(tmp\_loc)}$
    lc ← tuple t of li_fv such that
                t.fv is the next NN and within MD of cs.fv                 $c_{fsn(li\_fv)}$
endwhile
lc ← first tuple t of li_fv such that
            t.fv is the NN and within MD of cp.fv                          $c_{fsf(li\_fv)}$
while lc ≠ ERROR do
    lip ← tuple of logical_images corresponding to lc.tid                  $c_{r(li)}$
    ll ← tuple t of tmp_loc such that
                t.loc is the NN and within 5 of lip.loc                    $c_{lsf(tmp\_loc)}$
    while ll ≠ ERROR do
        lis ← tuple of logical_images corresponding to ll.tid              $c_{r(li)}$
        if lis.img_id = lip.img_id then
            pi ← tuple t of physical_images such that
                        t.img_id = lip.img_id                         $c_{af(pi\_id)} + c_{r(pi)}$
            Display pi.raw
        endif
        ll ← tuple t of tmp_loc such that
                    t.loc is the next NN and within 5 of lip.loc      $c_{lsn(tmp\_loc)}$
    endwhile
    lc ← tuple t of li_fv such that
                t.fv is the next NN and within MD of cp.fv            $c_{fsn(li\_fv)}$
endwhile
```

Plan P3$_C$ Search for site of interest tuples using an alphanumeric index on **class** and search for points in the given range using a spatial index on **loc**.

```
cds ← first tuple t of cl_sem such that
            t.semant ≥ "site of interest"                             $c_{af(cl\_sem)}$
cs ← tuple of classes corresponding to cds.tid                        $c_{r(cl)}$
lc ← first tuple t of li_cl such that t.class ≥ cs.name               $c_{af(li\_cl)}$
while lc ≠ ERROR and lc.class = cs.name do
    lis ← tuple of logical_images corresponding to lc.tid             $c_{r(li)}$
    ll ← first tuple t of li_loc such that
                t.loc is the NN and within 2 of lis.loc               $c_{lsf(li\_loc)}$
    while ll ≠ ERROR do
        lic ← tuple of logical_images corresponding to ll.tid         $c_{r(li)}$
        if lic.img_id = lis.img_id then
            pi ← tuple t of physical_images such that
                        t.img_id = lic.img_id                         $c_{af(pi\_id)} + c_{r(pi)}$
            cd ← tuple t of classes such that
                        t.name = lic.class                            $c_{af(cl\_name)} + c_{r(cl)}$
            Display pi.raw, Print cd.semant
        endif
        ll ← tuple t of li_loc such that
                    t.loc is the next NN and within 2 of lis.loc      $c_{lsn(li\_loc)}$
    endwhile
    lc ← next tuple of li_cl in alphabetic order                      $c_{an(li\_cl)}$
endwhile
```

Plan P3$_A$ Search for site of interest tuples using a spatial index on `fv` and search for points in given range using a spatial index on `loc`.

```
cds ← first tuple t of cl_sem such that
            t.semant ≥ "site of interest"                             $c_{af(cl\_sem)}$
cs ← tuple of classes corresponding to cds.tid                        $c_{r(cl)}$
lc ← first tuple t of li_fv such that
        t.fv is the NN and within MD of cs.fv                         $c_{fsf(li\_fv)}$
while lc ≠ ERROR do
    lis ← tuple of logical_images corresponding to lc.tid             $c_{r(li)}$
    ll ← first tuple t of li_loc such that
                t.loc is the NN and within 2 of lis.loc               $c_{lsf(li\_loc)}$
    while ll ≠ ERROR do
```

```
        lic ← tuple of logical_images corresponding to ll.tid                    $c_{r(li)}$
        if lic.img_id = lis.img_id then
            cd ← tuple t of classes such that
                        t.fv is the NN and within MD of lic.fv                   $c_{fsf(cl\_fv)} + c_{r(cl)}$
            if cd ≠ ERROR then
                pi ← tuple t of physical_images such that
                            t.img_id = lic.img_id                                $c_{af(pi\_id)} + c_{r(pi)}$
                Display pi.raw, Print cd.semant
            endif
        endif
        ll ← tuple t of li_loc such that
                    t.loc is the next NN and within 2 of lis.loc                 $c_{lsn(li\_loc)}$
    endwhile
    lc ← tuple t of li_fv such that
                t.fv is the next NN and within MD of cs.fv                       $c_{fsn(li\_fv)}$
endwhile
```