IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 18, No.8, pp. 783–798, Aug. 1996.

MARCO: MAp Retrieval by COntent

Hanan Samet [1]
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
Telephone: (301) 405-1755 Fax: (301) 314-9115

and

Aya Soffer [2]
Computer Science and Electrical Engineering Department
University of Maryland Baltimore County
5401 Wilkens Avenue
Baltimore, MD 21228-5398 *and*
Center of Excellence in Space Data and Information Sciences
NASA Goddard Space Flight Center
E-mail: soffer@cs.umbc.edu
Telephone: (301) 286-2439

Abstract

A system named MARCO (denoting MAp Retrieval by COntent) that is used for the acquisition, storage, indexing, and retrieval of map images is presented. The input to MARCO are raster images of separate map layers and raster images of map composites. A legend-driven map interpretation system converts map layer images from their physical representation to their logical representation. This logical representation is then used to automatically index both the composite and the layer images. Methods for incorporating logical and physical layer images as well as composite images into the framework of a relational database management system are described. Indices are constructed on both the contextual and the spatial data thereby enabling efficient retrieval of layer and composite images based on contextual as well as spatial specifications. Example queries and query processing strategies using these indices are described. The user interface is demonstrated via the execution of an example query. Results of an experimental study on a large amount of data are presented. The system was evaluated in terms of accuracy and in terms of query execution time.

Keywords: Map storage and retrieval, Document storage, Digital libraries, Automated indexing, Retrieval by content, Map interpretation, Geographic Information Systems (GIS)

[1] the support of the National Science Foundation under Grant IRI-92-16970 is gratefully acknowledged.
[2] The support of USRA/CESDIS and NASA Goddard Space Flight Center is gratefully acknowledged.

1 Introduction

The paper map has long been the traditional storage and retrieval medium for geographic data. Today, we are seeing the emergence of geographic information systems (GIS) as a replacement. One of the important issues in this field is how to integrate existing paper maps into a GIS. In particular, we would like to store scanned images of paper maps (termed *map images*) and to be able to retrieve portions of these maps based on the information that they convey, termed *retrieval by content*. An example query is "find all map images containing camping sites within 3 miles of fishing sites". We refer to such a system as a map image information system.

In this paper, we present a system for acquisition, storage, indexing, and retrieval of map images named MARCO (denoting MAp Retrieval by COntent). MARCO does not attempt to fully convert paper maps into one of the common GIS data formats (e.g. vector), and then let the GIS handle the queries and discard the map images. Such a conversion is very time consuming and labor intensive and is not feasible for a very large collection of maps such as those stored in map libraries [12]. Instead, MARCO's goal is to retrieve map images based on content from a database that contains a large number of such maps.

The input to MARCO are raster images of separate map layers and raster images of *map composites* (the maps that result from composing the separate map layers). We refer to these raster images as *layer images* and *composite images*, respectively. The map composite may be composed of layers that are input to the system as separate map layers as well as of layers that are not input to the system as separate map layers. Map layer images are processed with a system named MAGELLAN (denoting Map Acquisition of GEographic Labels by Legend ANalysis) [27]. The goal of this process is to extract contextual cues from the map layer that can be used to index the composite images.

MARCO uses the logical representation of a map image that is output by MAGELLAN to automatically index both the composite and layer images. We describe how to incorporate both layer and composite images into an existing spatial database. Our emphasis is on extracting and storing both contextual and spatial information from the layer images. Indices are constructed on both the contextual and the spatial data thereby enabling efficient retrieval of layer and composite images based on contextual as well as spatial specifications. Additional meta-data (e.g., how the maps were formed, scanning resolution, scale, etc) is also stored as attributes in the same relations. Indices may be constructed on these attributes to provide efficient access by meta-data. Queries may be posed to MARCO using either an SQL-like language or a graphical user interface (GUI). Queries are processed using advanced techniques from the field of spatial databases that utilize the available indices [23]. These include incremental nearest neighbor operators [9], and spatial join operators [21].

The rest of this paper is organized as follows. Section 2 lays out the background for this problem and discusses related work. Section 3 gives a short description of MAGELLAN (for a full description see [27]). Section 4 describes how layer and composite images are stored and indexed in a relational database management system including schema definitions and example relations. Section 5 presents methods for retrieving map images and some sample queries. Section 6 describes our implementation of MARCO along with some snapshots of the system performing an example query. Section 7 contains an experimental evaluation of MARCO in terms of accuracy and in terms of query execution time. Section 8 contains concluding remarks.

2 Background and Related work

In order to support retrieval by content of map images, the maps should be interpreted to some degree when they are inserted into the database. This process is referred to as converting a map image from a *physical* representation to a *logical* representation. It is desirable that the logical representation preserve the spatial information inherent in the map image (i.e., the spatial relationship between the objects found in the map image). We refer to the information regarding the objects found in a map image as *contextual information*, and to the information regarding the spatial relationship between these objects as *spatial information*.

The process of converting a map image from its physical to its logical representation is the subject of the field of map interpretation. This subject has been studied both in the context of the data conversion process in the field of GIS research [18], and in the context of document analysis in computer science research [11]. One of the most common means of performing this conversion is by use of a digitizing tablet. This is a very time-consuming and expensive process. Optical scanners have also been put to use for this purpose. The maps are first scanned resulting in a raster image of the map. This raster data is then usually converted into vector format with very heavy user intervention in order to assure the quality of this conversion [28].

Most research that deals with automating this process has focussed on skeletonization and vectorization methods [1, 30, 32]. The goal of these methods is to translate the raster scanned image of the map into a complete vector description (i.e., a description in terms of primitive geometric features such as point, lines, and polygons). The problem is that these geometric primitives do not necessarily correspond to geographic entities of interest to the application on hand. The reason is that a paper map is really an abstraction. The information found in maps is mainly symbolic rather than an accurate graphical description of the region covered by the map. For example, the width of a line representing a road has little to do with the road's actual width. Instead, most often the width of the road on the map is determined by the nature or type of the road (i.e., highway, freeway, rural road, etc.). As another example, graphical symbols are often used to indicate the location of various sites such as hospitals, post offices, recreation areas, scenic areas etc. The actual key to interpreting this symbolic information is usually found on the map itself in the map's legend. In contrast, in our approach map recognition is driven by the legend. In particular, the user identifies those symbols in the legend that are relevant for their application, and the acquisition process extracts this information from the raster scanned maps.

Much attention has been given to the subject of retrieval by content of images in the emerging field of *image databases* [10]. Most research in this area has concentrated on retrieval of photographs, where the image is treated as a whole and is indexed based mainly on color and texture. QBIC [15], Photobook [17], and FINDIT [31] are examples of systems that use such methods. Thus the issue of spatial information such as the relative position of different components of the image is not considered in these systems. In contrast, our approach combines indexing on spatial information as well as permitting retrieval on the basis of contextual (i.e., semantic) information.

Some issues that deal with storing spatial information in a relational database have been discussed as part of the SEQUOIA 2000 project [29]. However, this work did not address the image interpretation and contextual indexing aspects involved in map images (rather than just pure spatial data) into a relational database management system (DBMS).

3 Map Acquisition and Conversion

Map acquisition and conversion is performed by a system named MAGELLAN (denoting Map Acquisition of GEographic Labels by Legend ANalysis). The output of MAGELLAN is the logical representation of the layer tiles. These logical layer tiles, along with the physical layer tiles and composite tiles serve as input to MARCO. We first outline the MAGELLAN system, followed by a detailed description of the symbol classification process.

3.1 MAGELLAN System Description

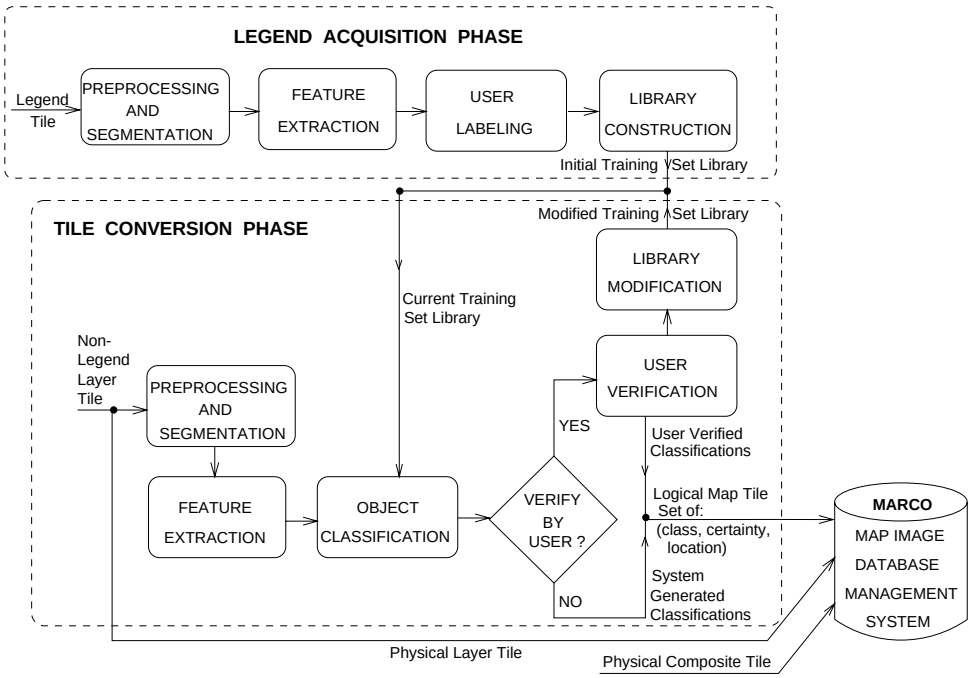

Figure 1: Block diagram of MAGELLAN (Map Acquisition of GEographic Labels by Legend ANalysis)

Figure 1 is a block diagram of MAGELLAN. Map layers and composites are scanned and divided into small *tiles* (i.e., of size 512×512 pixels). Note that it is possible for a symbol to lie in two tiles. In order to simplify our presentation, here we assume that this does not occur. If this is a concern, then an overlap of half the size of the largest symbol is required between tiles. The map layer tiles are processed one-by-one. Legend tiles are used to create an initial training set library. Non-legend tiles are converted from a physical to a logical representation. The system has two phases, the legend acquisition phase and the conversion phase, corresponding to the processing of legend and non-legend tiles, respectively.

The purpose of the legend acquisition phase is two-fold. The first is for the user to indicate which symbols of the legend are of importance to the application. These symbols are termed *valid symbols*. Any other symbols that are found in map tiles but were not pointed out by the user at this stage are termed *invalid symbols*. The second purpose of the legend acquisition phase is to construct an initial training set library that is subsequently used in the map tile conversion phase. This initial training

set library contains a feature vector corresponding to one instance of each valid symbol along with its semantic meaning (also termed *classification*). This is the classification that the classifier should assign to each instance of this valid symbol that is subsequently found in a map tile. For example "fishing site" for a fish symbol. For invalid symbols, a special classification termed *undefined* is used (i.e., invalid symbols should be classified as undefined by the system). All other classifications (i.e., those that correspond to valid symbols) are termed *valid classifications*.

Each non-legend tile is processed in the conversion phase. This phase may operate in two modes. In the *user verification mode*, the user verifies the classifications before they are input to the database. The training set library is modified to reflect the corrections that the user made for the erroneous classifications. Only feature vectors of symbols that could not be classified correctly using the current training set library are added to the library. Feature vectors of correct classifications are not added to it. The training set library is stored as an adaptive k-d tree [7, 22]. In the *automatic mode*, the classifications are generated by the system and input directly to the database. The user determines the mode in which the system operates. In general, the first tiles will be interpreted in user verification mode. Once the user is satisfied with the recognition rate achieved, the system is placed in automatic mode to process the remaining tiles.

Classification is performed using non-parametric statistical pattern recognition [6]. Each physical layer tile is first segmented into its constituent elements using a connected component labeling algorithm (e.g., [20]). This results in a labeled image in which each pixel has a region number as its value. For each region in the labeled image, a set of *features* is computed. These features include some global (e.g., first invariant moment, circularity, eccentricity, rectangularity) and some local shape descriptors (e.g., intersections, gaps) [13] that we empirically identified as useful features in discriminating between geographic symbols. The results of the feature computation are composed into a *feature vector*. The center of gravity (i.e., centroid) of each region is also computed. The x and y coordinate values of this location are termed a *location vector*. The current training set library is used to assign candidate classifications to each feature vector using a *weighted bounded several-nearest neighbor classifier* [4]. A value approximating the certainty of the correctness of these candidate classifications is attached to each classified symbol (see Section 3.2 for more details). Note that while a symbol may be composed of more than one connected component, we assume that the symbols may be distinguished from each other by one of these connected components. This component is identified in the legend acquisition phase, and is subsequently used to classify the symbol in the classification phase.

The output of the conversion phase is a logical map tile, consisting of the candidate classifications that were made, the certainty of the classifications, and the corresponding location of the symbols found in the physical map tile. Formally, the *logical map tile* representation of a physical map tile T is a list of tuples, one for each valid symbol $s \in T$. The tuples are of the form: $(C, certainty, (x, y))$ where C is a valid classification (i.e. $\neq undefined$) , $0 < certainty \leq 1$ indicates the certainty that $s \in C$, and (x, y) is the location of s in T. Using the terminology of spatial databases, the output is point data where the classifications and certainty values are alphanumeric attributes of the point. For more information about this system, see [26]. The logical map tile, physical layer tiles, and composite tiles are input to MARCO.

3.2 Symbol Classification

Geographic symbols identified in the map tiles are classified using a modification of the *weighted*

WBSNN-classify(feature vector F^I, training set library TSL)

1. Find $F^{L_N} \in TSL$ nearest to F^I
2. $D \leftarrow \mathbf{dist}(F^{L_N}, F^I) = \sqrt{\sum_{i=1}^{N} w_i (F_i^{L_N} - F_i^I)^2}$
3. $\varphi_I^L \leftarrow \{F^L \mid F^L \in TSL \wedge \mathbf{dist}(F^L, F^I) < min(\alpha \times D, \beta)\}$
4. **if** $\varphi_I^L = \emptyset$
   return(undefined)
5. **else begin**
6. **for** each $F^L \in \varphi_I^L$ **do begin**
   $Vote_{F^L} \leftarrow \frac{1}{\mathbf{dist}(F^L, F^I)}$
   end
7. **for** each C_i s.t. $\exists F^L \in \varphi_I^L \wedge class(F^L) = C_i$ **do begin**
   $Votes_{C_i} \leftarrow \sum_{F^L \in \varphi_I^L \wedge class(F^L) = C_i} Vote_{F^L}$
8. **if** $Votes_{C_i} > \mathbf{MaxVotes_C}$
   $certainty(I \in C_i) \leftarrow 0.9999$
9. **else**
   $certainty(I \in C_i) \leftarrow \frac{Votes_{C_i} - \mathbf{MinVotes_C}}{\mathbf{MaxVotes_C} - \mathbf{MinVotes_C}}$
   end
   end

Figure 2: Algorithm *WBSNN-classify* to classify an input symbol I using a weighted bounded several-nearest neighbor classifier. Four constants: α, β, $\mathbf{MinVotes}_C$, $\mathbf{MaxVotes}_C$, are used.

several-nearest neighbor classifier [4] termed a *weighted bounded several-nearest neighbor classifier*. This classifier makes use of two pre-defined constants: α, which is a neighborhood-size factor that determines the search range for nearest neighbors as a multiple of the distance between an input feature vector and the nearest feature vector in the training set library, and β, which is a bound that determines the maximum distance allowed between the feature vector of an input symbol and its several-nearest neighbors in the training set library. Figure 3.2 summarizes the algorithm for classifying an input symbol I using this classifier. This classifier first finds the feature vector F^{L_N} in the training set library (TSL) that is nearest to the feature vector of the input symbol F^I. Letting D be the weighted Euclidean distance between F^{L_N} and F^I given by

$$D = dist(F^{L_N}, F^I) = \sqrt{\sum_{i=1}^{N} w_i (F_i^{L_N} - F_i^I)^2}.$$

where $F_i^{L_N}$ is the i^{th} feature of the training set library vector F^{L_N}, F_i^I is the i^{th} feature of the feature vector of the input symbol I, and w_i is the weighting factor of the i^{th} feature of the feature vector. The weighing factor is computed so that features with a smaller variance have a larger weight as described in [5]. Next, the classifier finds the set φ_I^L of all training set library feature vectors F^L

whose distance to F^I is less than the smaller of α times D, and β. Formally:

$$\varphi_I^L = \{F^L \mid F^L \in TSL \wedge dist(F^L, F^I) < min(\alpha \times D, \beta)\}.$$

The range defined by min($alpha \times D, \beta$) is termed the *$\alpha\beta$-neighborhood.* Each feature vector $F^L \in \varphi_I^L$ is given a vote, whose strength is inversely proportional to its distance from the feature vector of the input symbol F^I, given by

$$Vote_{F^L} = \frac{1}{dist(F^L, F^I)}.$$

The votes of all feature vectors that belong to the same classification C_i are summed giving:

$$Votes_{C_i} = \sum_{F^L \in \varphi_I^L \wedge class(F^L) = C_i} Vote_{F^L}.$$

If $\varphi_I^L = \emptyset$ (i.e., the distance to the nearest neighbor was $> \beta$), then the input symbol is classified as undefined. A certainty value between 0 and 1 is computed for each candidate classification C_i found in the $\alpha\beta$-neighborhood of the input feature vector. This value approximates the certainty that the input vector belongs to C_i. The certainty value is calculated by normalizing the value of $Votes_{C_i}$ with respect to some minimal and maximal acceptable values of $Votes_C$ for any of the possible candidate classifications C. The maximal acceptable vote value is determined by selecting a minimal required distance d_{min}, for a "sure" classification (i.e., if $dist(F^L, F^I) < d_{min}$, then F^I will be assigned the training set library classification corresponding to F^L with certainty 0.999). Hence, the maximal value for $Votes_C$ is $1/d_{min}$. The minimal acceptable vote value is determined by selecting a maximal allowed distance d_{max} for a classification to be considered as a candidate (i.e., if $dist(F^L, F^I) > d_{max}$, then the training set library classification corresponding to F^L will not be considered as a candidate classification for F^I at all). Hence, the minimal value for $Votes_C$ is $1/d_{max}$. These two constants are denoted as $MaxVotes_C$ and $MinVotes_C$, respectively. The motivation for calculating the certainty values in this manner is that the certainty values must rank the candidate classifications with respect to one another not only in one invocation of the classifier, but must do so with respect to the candidate classifications of all other invocations of the classifier, also. Therefore, some global method of calculating certainty values is required.

The nearest neighbors of the training set library that are within the $\alpha\beta$-neighborhood are found using a modification of the priority k-d tree search algorithm [3]. The classification module outputs all of the candidate classifications along with their certainty. The classification with the highest certainty value is considered the best classification for the input vector using the weighted bounded several-nearest neighbor classifier.

Figure 3 demonstrates the classification process. It uses a sample set of symbol instances in 2-space as the training set library. Let $X = (34, 31)$ be the feature vector (in 2-space) of an input symbol. Let $\alpha = 2$, $\beta = 11$, and $w_1 = w_2 = 1$ (i.e., both features are assigned an equal weight in the distance computation). The classifier assigns candidate classifications to X as follows. First, the feature vector F^{L_N} in the training set library (TSL) that is nearest to the feature vector of X (F^I) is found. In this case, $F^{L_N} = (35, 40), D = \sqrt{82} \approx 9.055$. The classifier next finds the set φ_I^L of all library feature vectors F^L whose distance to F^I is less than the smaller of α times D, and β (i.e., in the $\alpha\beta$-neighborhood). In this case, $sizeof(\alpha\beta\text{-neighborhood}) = min(18.11, 11) = 11$. Thus, φ_I^L = {(35,40),(25,35)}, where (35,40) is an instance of the symbol "arrow" (holiday camp)

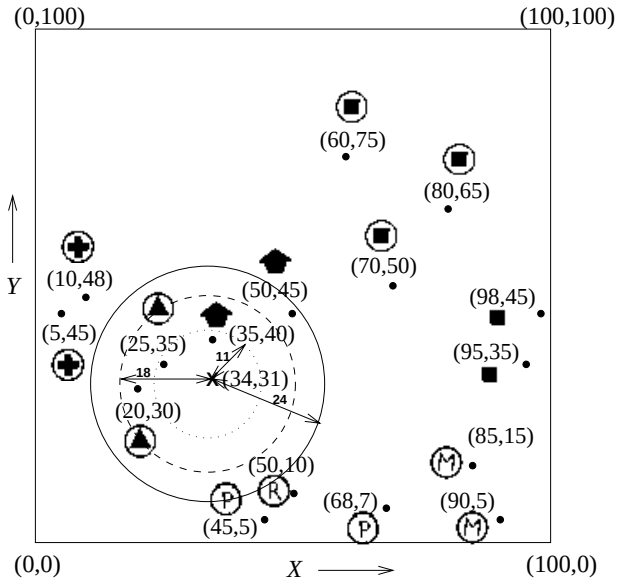

Figure 3: Example classifion of a symbol X (using a sample set of symbols in 2-space as the training set) with a weighted bounded several-nearest neighbor classifier. For $\alpha = 2$, $\beta_1 = 11$, $\beta_2 = 24$, the dotted circle is the $\alpha\beta_1$-neighborhood, the dashed circle is the $\alpha\beta_2$-neighborhood, and the solid circle is the range defined by β_2.

and (25,35) is an instance of the symbol "triangle" (camping site). $Votes_{arrow} = 1/\sqrt{82} \approx 0.11$, $Votes_{triangle} = 1/\sqrt{97} \approx 0.10$. Thus, X will be assigned classification "arrow" with a higher certainty than classification "triangle". Recall that the certainty value is calculated by normalizing the value of $Votes_{C_i}$ with respect to some minimal and maximal acceptable values of $Votes_C$ for any of the possible candidate classifications C. These values are determined according to d_{min} and d_{max}, the minimal and maximal acceptable values for $dist(F^L, F^I)$. Thus, to calculate the certainty values for this example we must assign some values to these parameters. Using d_{min} and d_{max} values of $\sqrt{8}$ and 24, respectively, we get $MaxVotes_C = 1/\sqrt{8} \approx 0.353$ and $MinVotes_C = 1/24 \approx 0.042$. Therefore, $certainty(X \in arrow) = 0.218$ and $certainty(X \in triangle) = 0.186$ since we normalized 0.10 and 0.11, respectively.

However, if we let $\beta = 24$, then $sizeof(\alpha\beta\text{-neighborhood}) = min(18.11, 24) = 18.11$. Point (20,30) which is another instance of "triangle" will now also be included in φ_I^L. Thus, $Votes_{arrow} = 1/\sqrt{82} \approx 0.11$, $Votes_{triangle} = 1/\sqrt{97} + 1/\sqrt{197} \approx 0.17$. In this case, X will be assigned classification "triangle" with a higher certainty than classification "arrow". Using the same d_{min} and d_{max}, we get $certainty(X \in arrow) = 0.218$ and $certainty(X \in triangle) = 0.411$. Note that in this example there is really no clear cut classification for the input symbol. Therefore the classifier outputs two candidates with relatively low certainty values. The purpose of the certainty values is to rank the classifications on a global scale of $0 - 1$. It is not a probability function (i.e., the sum of all certainties for a given input symbol need not equal 1). In practice, with a large training set, and if the input symbol clearly falls into a particular classification, most of its neighbors in the $\alpha\beta$-neighborhood will belong to the same classification. Furthermore, the distance between them and the feature vector of the input symbol will be small. Thus, the certainty value will be higher.

4 Map Image Storage

Map images and other information pertaining to the application are stored in relational tables. The database system that we use for this purpose is SAND [2] (denoting spatial and non-spatial database) developed at the University of Maryland. It is a home-grown extension to a relational database where 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 non-locational nature. Both types of attributes may be designated as indices of the relation. For indices built on locational attributes, SAND makes use of suitable spatial data structures. Attributes of type image are used to store physical images. Query processing and optimization is performed following the same guidelines of relational databases extended with a suitable cost model for accessing spatial indices and performing spatial operations.

4.1 Schema Definitions

```
(create table classes
   class CHAR[30],
   semantics CHAR[50],
   bitmap IMAGE);
```

```
(create table physical_map_images
   image_id INTEGER,
   descriptor CHAR[50],
   lower_left POINT,
   raw IMAGE);
```

```
(create table logical_map_images
   l_image_id INTEGER,
   c_image_id INTEGER,
   class CHAR[30],
   certainty FLOAT,
   l_location POINT,
   c_location POINT);
```

Figure 4: Schemas for the relations `classes`, `physical_map_images`, and `logical_map_images`.

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

Figure 5: Example instance of the `classes` relation.

The schema definitions given in Figure 4 define the relations that are used by MARCO. We use an SQL-like syntax. The `classes` relation has one tuple for each classification that was identified in the legend acquisition phase. The `class` field stores the name of the classification (e.g., star). The `semant` field stores the semantic meaning of the classification in the map (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 library for the map interpretation system in the legend acquisition phase (i.e.,

image_id	descriptor	raw	lower_left
image_1	red sign layer tile 003.012 of Finish road map	Figure 7	(6144,7168)
image_2	red sign layer tile 003.013 of Finish road map	N/A	(6656,7168)
image_4	composite tile 1.13 of Finish road map	Figure 8	(3328,4608)
image_5	composite tile 1.14 of Finish road map	N/A	(3584,4608)

Figure 6: Example instance of the physical_map_images relation.

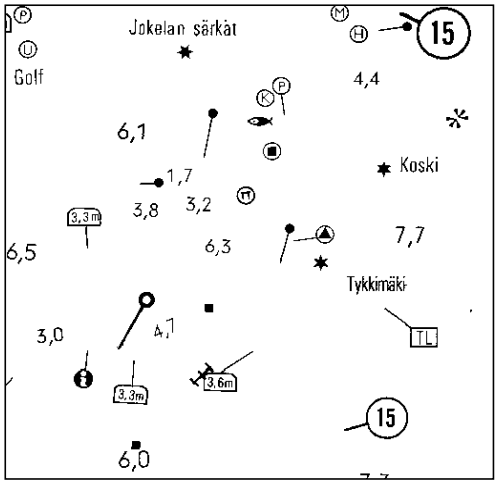

Figure 7: Example layer image1.

Figure 8: Example composite image4.

one example symbol for each classification found in the legend along with its name and semantic meaning). See Figure 5 for an example instance of the classes relation.

The physical_map_images relation has one tuple per map tile T in the database. These include both layer tiles and composite tiles. The image_id field is an integer identifier assigned to the tile T when it is inserted into the database. The descriptor field stores an alphanumeric description of the tile T that the user gives when inserting T (this is meta-data). The raw field stores the actual

l_image_id	c_image_id	class	certainty	l_location	c_location
image_1	image_4	fish	1	(6411,7043)	(3438, 4605)
image_1	image_4	scenic	0.99	(6612,7042)	(3572, 4605)
image_1	image_4	square	1	(6422,7011)	(3445, 4584)
:					
image_1	image_4	cross	0.79	(6291,6850)	(3358, 4477)
image_2	image_5	pi	0.99	(6849,6948)	(3730, 4542)
image_2	image_5	R	0.71	(6849,6948)	(3730, 4542)
:					

Figure 9: Example instance of the logical_map_images relation. The tuples correspond to some of the symbols in the tile of Figure 7.

tile T in its physical representation. It is an attribute of type `IMAGE`. The `lower_left` field stores an offset value that locates the lower left corner of map tile T with respect to the lower left corner of the non-tiled map image M. Subtracting this offset value from the absolute location of s in the non-tiled map image M yields the location of s in the map tile T that contains it. It is an attribute of type `POINT`. Any additional meta-data that the user may wish to store about the map tiles such as how they were formed, scanning resolution, scale, etc. can be added as fields of this relation. See Figure 6 for an example instance of the `physical_map_images` relation corresponding to the layer tile given in Figure 7 and to the composite tile in Figure 8.

The `logical_map_images` relation stores the logical representation of the map tiles. It has one tuple for each candidate classification output by the map interpretation system for each valid symbol s in each layer tile LT. Each tuple has six fields. The `l_image_id` field is the integer identifier assigned to LT when it was inserted into the database. The `c_image_id` field is the integer identifier assigned to the corresponding composite tile CT that contains symbol s. The values of these fields are equal to the `image_id` field of the tuples representing LT and CT in the `physical_map_images` relation. The `class` and `certainty` fields store the name of the class C to which MAGELLAN classified s and the certainty that $s \in C$. The `l_location` field stores the (x, y) coordinate values of the center of gravity of s relative to the non-tiled layer image. The `c_location` field stores the (x, y) coordinate values of the center of gravity of s relative to the non-tiled composite image. Note that `l_location` and `c_location` are not necessarily the same since the scanning resolution of the non-tiled layer images and the non-tiled composite images may differ. Therefore, since the size of the entire non-tiled map image may vary, the location of the symbol in the non-tiled map image will also vary. See Figure 9 for an example instance of the `logical_map_images` relation corresponding to the layer tile given in Figure 7.

The coordinate values that are stored in the `lower_left` attribute of the `physical_map_images` relation and the `l_location` and `c_location` attributes of the `logical_map_images` relation are absolute pixel values (with the lower left corner of the entire map at 0,0). These values can easily be converted into long/lat values or any other required coordinate system. We chose to ignore this issue since it is a simple transformation and not in the scope of this research.

Alphanumeric indices `cl_semant` and `cl_class` are constructed to search the `classes` relation by the `semant` and `class`, attributes respectively. An alphanumeric index `pi_imid` is used to search the `physical_map_images` relation by the `image_id` attribute. A spatial index on points `pi_ll` is used to search the `physical_map_images` relation by the coordinates of the lower left corner of the map tiles. Alphanumeric index `li_cl` is used to search the `logical_map_images` relation by the `class` attribute. It has a secondary index on attribute `certainty`. Spatial indices `li_l_loc` and `li_c_loc` are used to search the `logical_map_images` relation by location in the non-tiled layer images and non-tiled composite images, respectively (i.e., to deal with spatial queries such as distance and range queries on the locations of the symbols in the map images). The spatial indices are implemented using a PMR quadtree for points [14].

4.2 Partitioning the Logical Map Images Relation

Many queries in a map image database application need to access all symbols that are assigned the same classification. The part of the query that selects all tuples that belong to the same classification is repeated each time such a query is posed. In order to make this repetitive selection at query time unnecessary, we propose to partition the `logical_map_images` relation. We refer to

the case where the `logical_map_images` are in one relation as the *integrated organization*, and to the case where the `logical_map_images` relation is partitioned as the *partitioned organization*. In the partitioned organization, the tuples are partitioned into separate relations resulting in a one-to-one correspondence between relations and the classifications that are present in the application (as identified in the legend acquisition phase). 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. Although this may be inappropriate when the number of different classes is large, this is not the case in a map image database system since the number of classifications corresponds to the number of different symbols found in the legend of the map, which is typically not very large. The motivation for the partitioned organization is that it enables efficient use of spatial indices while processing spatial queries by using a spatial join operator (e.g., [21]). For more details, see Section 5.

```
for each class CL in map
    (create table CL_class
     l_image_id INTEGER,
     c_image_id INTEGER,
     certainty FLOAT,
     l_location POINT,
     c_location POINT);
```

Figure 10: Schemas for the `CL_class` relations in a partitioned organization.

star_class:

l_image_id	c_image_id	certainty	l_location	c_location
image_1	image_4	0.99	(6540,6992)	(3524, 4571)
image_1	image_4	1	(6474,6890)	(3480, 4503)

pi_class:

l_image_id	c_image_id	certainty	l_location	c_location
image_1	image_4	0.99	(6395,6963)	(3427, 4552)
image_2	image_5	0.99	(6849,6948)	(3730, 4542)
image_2	image_5	0.99	(6800,6897)	(3697, 4508)

Figure 11: Example instances of `class` relations using the partitioned organization. The tuples correspond to the symbols in the image of Figure 7.

Figure 10 gives the schema definitions for the relations of the partitioned organization that correspond to the `logical_map_images` relation. Both the `classes` and `physical_map_images` definitions are identical to those in the integrated organization. The only difference between the two organizations is in the way the logical representation of the map tiles is stored. In the partitioned organization, there is one relation, `CL_class` for each class CL in the map. Each relation `CL_class` contains the `logical_map_images` tuples $(C, certainty, (x, y))$ for which $C = CL$. This is equivalent to the result of applying a selection operation, `class` $= CL$, on relation `logical_map_images`. See Figure 11 for example instances of relations `star_class` and `pi_class` for the image given in Figure 7. Each relation `CL_class` has an alphanumeric index on `certainty` and a spatial index on `l_loc` and `c_loc`. The spatial index is used to deal with queries of the type "find all images with sites of interest within 10 miles of a picnic area" by means of a spatial join operator.

5 Map Image Retrieval

In order to describe the methods that we use for retrieving map tiles by content, we first present some example queries. Next, we demonstrate a few possible strategies that can be used to process one of these queries.

5.1 Example Queries

The example queries in this section are first specified using natural language. This is followed by an equivalent SQL-like query for the first two queries. An SQL-like query for the other queries can be created in a similar manner.

Query Q1: display all layer tiles that contain a beach.

```
display PI.raw
    from logical_map_images LI, classes C, physical_map_images PI
    where C.semant = "beach" and C.class = LI.class
        and PI.image_id = LI.l_image_id
```

Query Q2: display all layer and composite tiles that contain a site of interest within 15 miles of a hotel.

```
display PI1.raw PI2.raw
    from logical_map_images LI1, logical_map_images LI2, classes C1,
         classes C2,  physical_map_images PI1, physical_map_images PI2
    where C1.semant = "site of interest" and C2.semant = "hotel"
        and C1.class = LI1.class and C2.class = LI2.class
        and distance(LI1.l_location,LI2.l_location) < 15
        and  PI1.image_id = LI1.l_image_id
        and  PI2.image_id = LI1.c_image_id;
```

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

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

Query Q4: display all composite and layer tiles that contain an airfield north of a beach.

5.2 Query Processing

The problem of how to use the indices that have been constructed on the spatial, contextual, and meta data in an efficient matter is very complex. This is a question of query optimization in a spatial database [2], a subject that is only recently receiving attention in the database literature. To illustrate just how complex this issue may be, we present four different strategies (plans) for computing an answer to query Q2. The first three plans are suitable for the integrated organization. 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. The fourth plan assumes that a partitioned organization exists. We only sketch these plans here; see [25] for more detailed plans for all of the example queries. 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. Direct addressing of a tuple within a relation is possible by means of a tuple identifier (or *tid* for short). All index structures have an implicit attribute that stores this tid.

Query Q2: display all layer and composite tiles that contain a site of interest within 15 miles of a hotel.

Plan P2A Search for "hotel" and "site of interest" tuples using an alphanumeric index on `class`. For each "hotel" tuple, check all "site of interest" tuples to determine which ones are within the specified distance.

Plan P2B Search for hotel tuples using an alphanumeric index on `class`. Search for all tuples within 15 miles of each hotel using a spatial index on `l_location`. Determine which of these tuples are "sites of interest" by checking the `class` attribute.

Plan P2C Search for "site of interest" tuples using an alphanumerical index on `class`. Build a temporary spatial index on the `l_location` attribute of these tuples. Search for "hotel" tuples using an alphanumeric index on `class`. For each hotel, search for all tuples within 15 miles in the temporary spatial index on `l_location` (these all correspond to "site of interest" tuples).

Plan P2D Search the "hotel" partition sequentially. For each hotel, search the "site of interest" partition for all tuples within 15 miles using the spatial index on `l_location`.

5.3 User Interface

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

Contain Query: find all map tiles that contain symbol(s) from some given *class(es)* (query Q1 is in this category).

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

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

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

Directional Location Query: find all map tiles in which a symbol from *class1* is located in a given *direction* relative to a symbol from *class2* (query Q4 is in this category).

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

6 Implementation

The MARCO system was tested on the red sign layer and the composite of the GT_3 map of Finland. This map is one of a series of 19 GT maps that cover the whole area of Finland. The red sign layer contains geographic symbols that mostly denote tourist sites. This layer was chosen because it contains the symbols on which we wanted to base the map image indexing and because MAGELLAN was designed to recognize such symbols. The map layer was scanned at 240dpi. This layer was split into 425 tiles of size 512×512. See Figure 7 for an example layer tile.

Of these 425 tiles, 280 contained at least one valid symbol. These 280 tiles contained 1093 valid symbols. Thus, there were a total of 280 logical images corresponding to 1093 tuples in the `logical_images` relation. The map composite was scanned at 160dpi. The layer was split into 551 tiles of size 256×256. See Figure 8 for an example composite tile. The composites were scanned at a lower resolution in order to reduce the space required to store these tiles. The red sign layer tiles are binary images, whereas the composite tiles are full color images. Thus each composite tile requires much more storage space. In addition, the red sign layer tiles are used for indexing purposes – that is, the conversion from physical to logical representation is performed on them, and thus a high resolution is desired in order to get good recognition rates. The composite tiles, on the other hand, are only used for the purpose of display and thus the lower resolution is adequate for this use.

The initial training set was created by using one example symbol of each class as taken from the legend of the map. There were 22 classes in the map. The tiles were input in random order to the map image database via MAGELLAN as outlined in Section 3. MAGELLAN was implemented using Khoros [19], an integrated software development environment for information processing and visualization. The first 50 tiles 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 tiles were processed automatically. In [26] we reported results of experimenting with various values for β, and tested whether considering more than just the classification with the highest certainty improves the recognition rates. A β value of 0.1 was found to be best for this data set. A valid symbol recognition rate of 91% and an invalid symbol recognition rate of 99% were achieved with this value. In addition, 10% of the symbols that were assigned valid classifications were results of multiple classifications for a valid symbol. We also found that only the candidate classifications with the highest and second-highest certainty values should be considered (i.e., transferred to the database). The improvement in the valid symbol recognition rate when considering all candidate classifications rather than just the first two was very small while resulting in numerous cases of multiple classifications for the same input symbol. Therefore, considering all candidate classifications does not seem to be beneficial.

We thus ran MAGELLAN with a β value of 0.1, and inserted the candidate classifications with the highest and second-highest certainty values into the logical map image. These logical images were input to SAND and inserted into the `logical_map_images` relations as defined in Section 4. The physical layer and composite tiles were also input to SAND and inserted into the `physical_map_images` relation. The GUI and the plans for the five query categories were implemented using Tcl (short for Tool Command Language), an interpreted scripting language, and Tk, a toolkit for the X window system, developed by Ousterhout [16]. For query Q2, we created plans following all of the strategies outlined in Section 5.2. These plans were executed on a Sparc 10 running UNIX, and statistics regarding the execution were recorded.

We currently have only scanned one small portion of the map of Finland. Thus, our data set is not very large. In order to test our system on a larger and more realistic data set, we derived semi-synthetic data from the original data set. This was done by replicating the map tiles by a constant factor M. Each original tile was replicated M times. The lower-left coordinate of these new images was computed so that it seems that the entire map is duplicated M times. The objects inside each image were relocated randomly within the 512×512 area of the image. The values selected for M were 2, 4, 8, 16, 32, and 64, yielding map image multiples of the original map 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 map images, corresponding to 1093, 2184, 4368, 8736, 17472, 34984, and 69952 tuples in the `logical_images` relation (since the basic data set has 1093 valid symbols).

6.1 An Example Query Execution

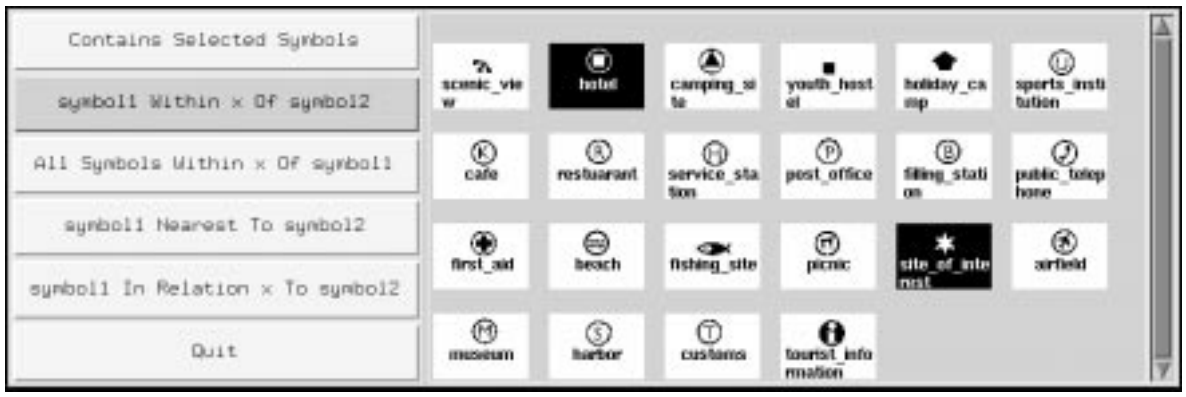

Figure 12: Graphical User Interface for query initiation. User has selected a "One Within Query" between a "hotel" and a "site of interest".

The following scenario describes how example query Q2 is specified using the GUI and how the results are presented. Recall, that Q2 was the query "display all layer and composite tiles that contain a site of interest within 15 miles of a hotel". Figure 12 shows the GUI for initiating a query. It consists of a button for each query category and an icon for each of the symbol classes. The icon is composed of the `bitmap` and `semant` fields of the tuples in the `classes` relation. Recall that these fields are populated with data from the map legend. Thus the GUI can automatically adjust to a different set of symbols simply by updating the `classes` relation.

To perform a "one within" query, the user first selects the icons of the two required classes

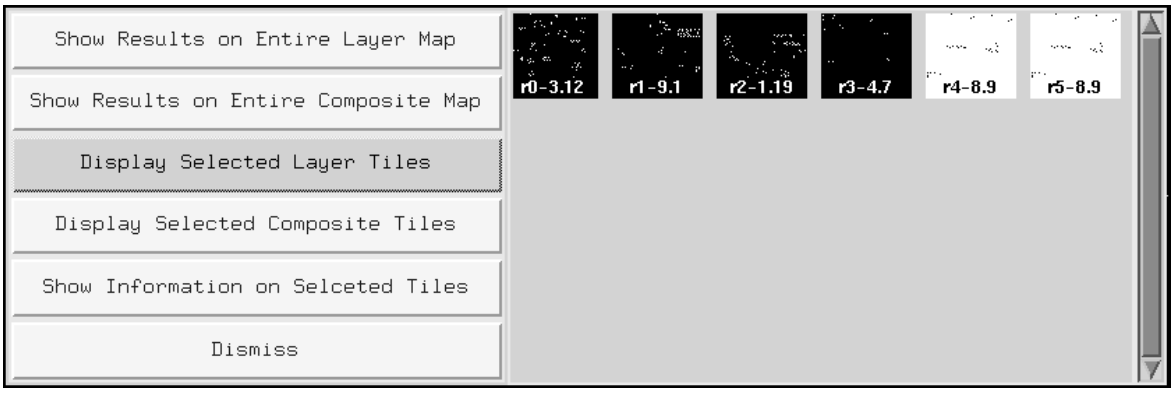

Figure 13: Results of query computation. The user has selected to display the layer tiles of the first four results.

followed by clicking the "symbol1 Within x of symbol2" button. The user is then prompted for the required distance. Once the user enters the required distance, the result of the query is computed using one of the four plans outlined in Section 5.2. The result of this query is displayed in a window as seen in Figure 13. A thumbnail (i.e., a reduced bitmap of the whole tile) is displayed for each layer tile that was found that meets the query specification. Recall, that each tuple in the `logical_map_images` relation has a certainty value that estimates the certainty that the symbol in the location corresponding to the tuple belongs to the class corresponding to the tuple. The result tiles are displayed in decreasing order of the certainty value. Therefore, the first result tiles are more likely to be correct (i.e., meet the query specification) and the last tiles are more likely to be incorrect. The user may now display any of the result tiles by selecting the corresponding thumbnails followed by clicking either the "Display Layer" or "Display Composite" buttons. A square is drawn around the two symbols that were given to the query. By clicking the "Information" button, the user can see the information stored regarding each of these tiles in the `physical_map_images` relation and the locations of the symbols in these tiles. In addition, the user may choose to display the non-tiled map with the query result tiles highlighted.

7 Evaluation Procedure and Results

Our experimental evaluation of MARCO was designed to test two performance metrics. The first is accuracy of the results. That is, how many result tiles are missed and how many result tiles do not meet the query specification. The second performance metric is query execution time. In particular, we performed an empirical comparison in terms of execution time between the four plans for queries that fall into the "One Within Query" category such as query Q2.

7.1 Accuracy

We evaluated MARCO in terms of accuracy using two error types that are commonly used in document analysis studies. 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.

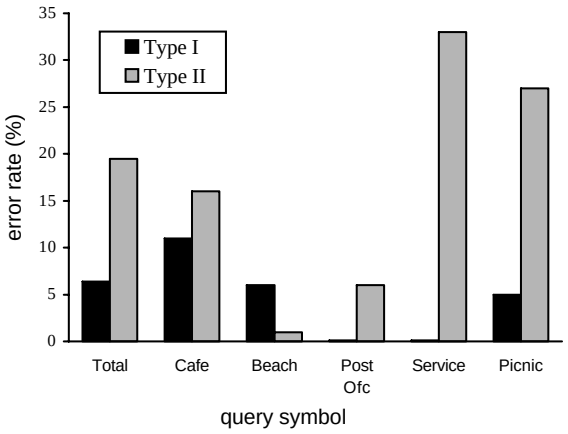

Figure 14: Type I and Type II error rates when considering all certainty values.

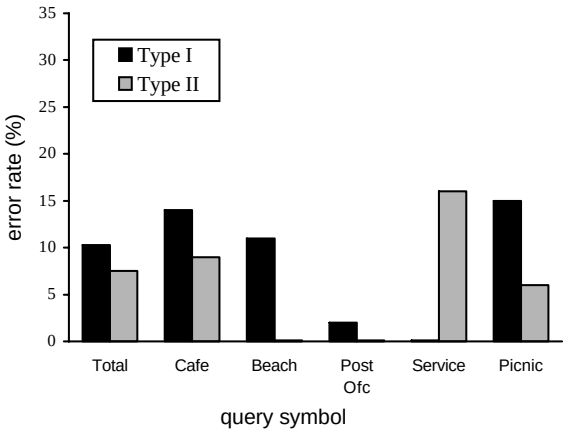

Figure 15: Type I and Type II error rates when considering only results with certainty value > 0.25.

Type II errors are detected by visual inspection of the result tiles. Each result tile that does not meet the query specification, is counted as a Type II error. In order to calculate the type II error rate in MARCO, we performed a "contain query" for each of the symbols in our application. We counted how many results did not meet the query specification for each symbol. We computed the type II error rate for each symbol. In addition, we computed the total type II error rate as the total number of incorrect results divided by the total number of results.

In order to count the Type I errors, we need to visually inspect the physical map tiles (in contrast to just looking at the result tiles as we did for Type II errors) or the paper map and look for all required results in order to determine whether any result tiles were missed (since we do not have ground truth for this data set). We did this for 50 tiles (out of the 425 tiles) chosen at random and for each one of the symbols. Once again, we computed the type I error rate for each symbol in addition to a total type I error rate which is the total number of missed results divided by the total number of results we should have had.

Figure 14 reports the total type I and type II error rates, as well as these error rates for a few of the symbols. The total type I error rate was 6% (i.e., 94% of the tiles that should have been retrieved were in fact retrieved by the system). Note however, that this rate varies for the different symbols. The rates varied from 0% for the "post office" symbol to 11% for the "cafe" symbol. The total type II error rate was 19% (i.e., 81% of the tiles that were retrieved did in fact contain the desired symbol). The type II error rates varied from 1% for the "beach" symbol to 33% for the "service station" symbol. We attribute the variance in the error rates between different symbols to the specific classification method and to the contents of the training set that is used when inserting the map images into the database. In particular, 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 configurations, and these results were consistent in all cases. In order to achieve lower error rates, more features would be required.

Recall, that the results are ranked by certainty. The type II error rate can be improved by considering only those results with a certainty value greater than some cutoff value. This, of course, 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. Figure 15 shows the type I and type II error rates for a cutoff certainty value of 0.25. The total type I error rate went up to 10% while the total type II error rate went down to 7.5%. 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. Even for a the lowest setting of the certainty cutoff value, only 6% of the images in the database were retrieved on average for the queries that we performed.

Note that these results are a direct consequence of the settings chosen for the parameters of MAGELLAN, the map input system. In [26] we saw that we can achieve higher valid symbol recognition rates by increasing the values of the search range, β. However, this results in lower invalid symbol rates. Thus, by changing this value we could improve our type I error rates, while degrading our type II error rates. In addition, with more user intervention (i.e., running MAGELLAN in user verification mode), the valid and invalid symbol recognition rates can be improved, thus resulting in better type I and type II error rates. The ideal search bound value and amount of intervention should be selected according to the requirements of the application. If it is critical for the map image database not to miss any tiles when responding to a query, then a larger search bound value should be selected. If accuracy is not as important as the time required to weed out the incorrect tiles manually, then a smaller search bound value should be selected.

7.2 Query Execution Time

We performed an empirical comparison in terms of execution time between the four query plans for queries that fall into the "One Within Query" category such as query Q2. This was done by executing numerous variations of this query using the plans as outlined in Section 5.2. An important aspect in selecting the the appropriate query 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. This was achieved by varying parameters that affect these selectivity factors. 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). 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.

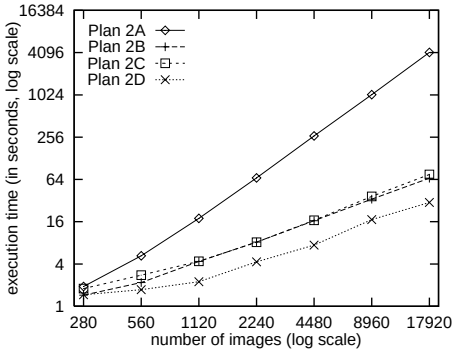

Figure 16: Retrieval time in seconds for various data set sizes for various plans and organizations search radius 2 miles; small difference in contextual selectivity.

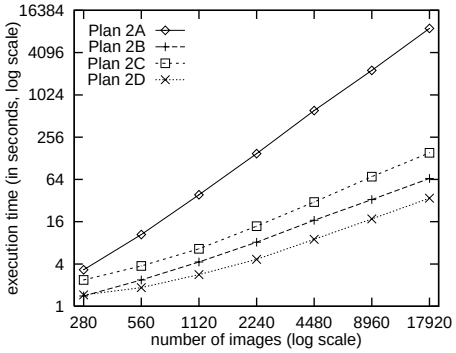

Figure 17: Retrieval time in seconds for various data set sizes for various plans and organizations search radius 2 miles; large difference in contextual selectivity.

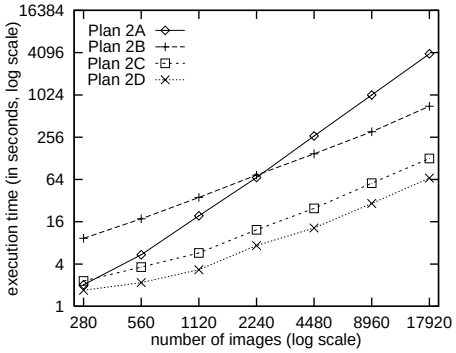

Figure 18: Retrieval time in seconds for various data set sizes for various plans and organizations search radius 16 miles; small difference in contextual selectivity.

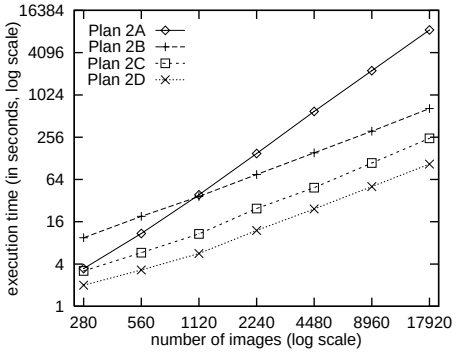

Figure 19: Retrieval time in seconds for various data set sizes for various plans and organizations search radius 16 miles; large difference in contextual selectivity.

Figures 16 – 21 report the retrieval time in seconds for query Q2 using the four plans for various data set sizes. Figures 16, 18, and 20 correspond to the case where the difference in the contextual selectivity between the two symbols involved in the query is small, for progressively smaller spatial selectivities (i.e., larger search radii). Similarly, Figures 17, 19, and 21 correspond to the case where the difference in the contextual selectivity between the two symbols involved in the query is large for progressively smaller spatial selectivities (i.e., larger search radii). From these figures, it is apparent that plan P2D, which uses a partitioned organization, is almost always the best plan. Only in the case of a relatively small data set and a low spatial selectivity (64 mile search radius) is plan P2A better. Note however, that for queries where the contextual selectivity is low such as query Q3 the partitioned organization performs very poorly (see Figure 29).

Between the plans that utilize an integrated organization, plan P2C (which builds an intermediate spatial data structure before performing the spatial join) is best in most cases. The cost of building the temporary spatial data structure pays off in the much more efficient spatial join due to the fact

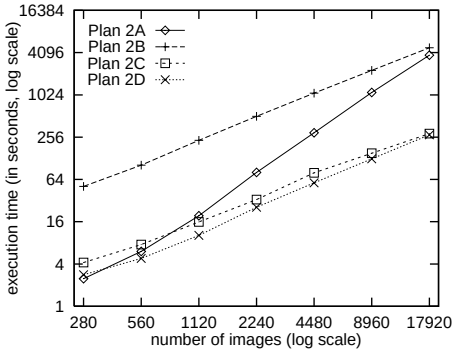

Figure 20: Retrieval time in seconds for various data set sizes for various plans and organizations search radius 64 miles; small difference in contextual selectivity.

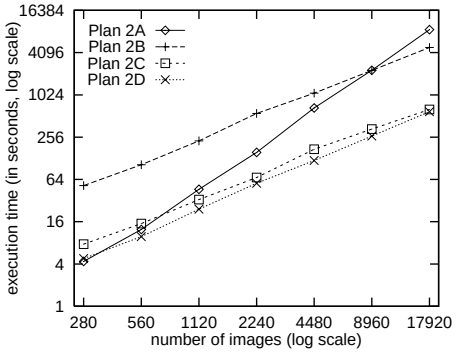

Figure 21: Retrieval time in seconds for various data set sizes for various plans and organizations search radius 64 miles; large difference in contextual selectivity.

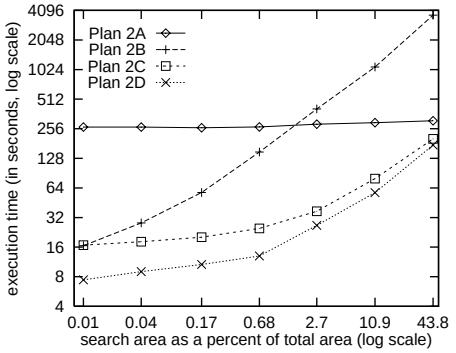

Figure 22: Retrieval times in seconds for various plans and organizations varying the search range, for the 4480 image data set — small difference in contextual selectivity.

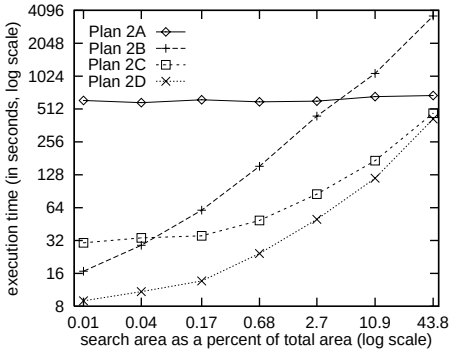

Figure 23: Retrieval times in seconds for various plans and organizations varying the search range, for the 4480 image data set — large difference in contextual selectivity.

that the tree is smaller and contains only symbols of the desired class. However, when the spatial selectivity is high (small search radius) and the difference in contextual selectivities is large, plan P2B outperforms plan P2C (see Figure 17). The reason for this is that the number of symbols that need to be inserted into the temporary spatial data structure is relatively large, while the area of the search range that is searched in the spatial index that holds all of the symbols (as is done in plan P2B) is small and thus the search is relatively inexpensive. Therefore, the overhead incurred when building the temporary spatial data structure does not pay off in this case.

Figures 22 and 23 show the execution time in seconds for query Q2 using the four plans varying the spatial selectivity for a constant dataset size. Figure 22 corresponds to the case where the difference in the contextual selectivity between the two symbols involved in the query is small, while Figure 23 corresponds to the case where the difference in the contextual selectivity between the two symbols involved in the query is large. Here we can see that when the spatial selectivity is low (i.e.,

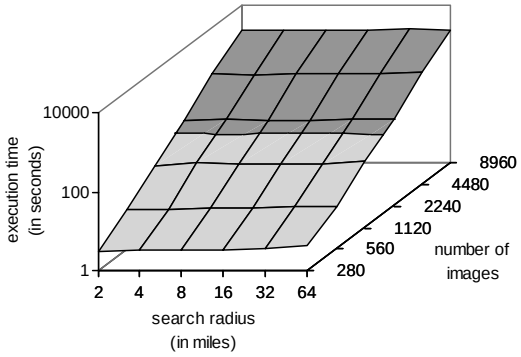

Figure 24: Retrieval times in seconds for Plan P2A for various search radii and various data set sizes.

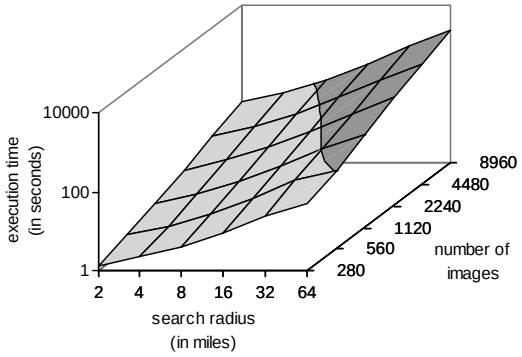

Figure 25: Retrieval times in seconds for Plan P2B for various search radii and various data set sizes.

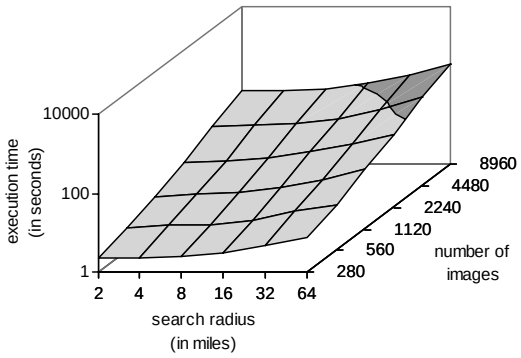

Figure 26: Retrieval times in seconds for Plan P2C for various search radii and various data set sizes.

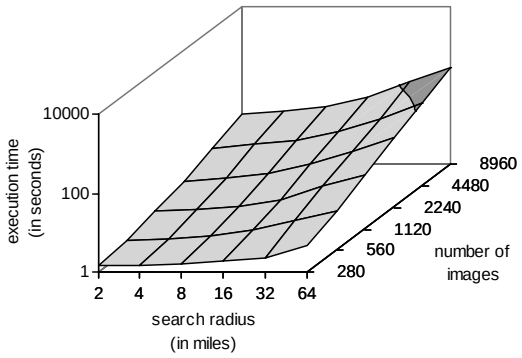

Figure 27: Retrieval times in seconds for Plan P2D (partitioned organization) for various search radii and various data set sizes.

a large search area), plan P2B becomes very inefficient. This can also be seen in Figure 25 which shows the execution time of plan P2B as both the search radius and the data set grows. Notice the steep slope as the size of the radius and the data set increases. In contrast, in Figure 24, which shows the execution time of plan P2A as both the search radius and the data set grows, the size of the search radius does not effect the execution time. The reason for this is that plan P2A does not use the spatial index at all. It computes the distance between every two symbols conforming to the query specification. Thus, it is not sensitive to the changing distance and it becomes attractive when the spatial selectivity is low.

Plan P2C seems to be the best compromise. It makes use of both a contextual and spatial index. 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 data structure, 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

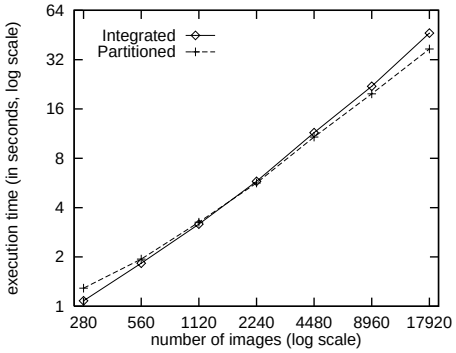

Figure 28: Retrieval time in seconds for various data set sizes for query Q1 using the integrated and partitioned organizations.

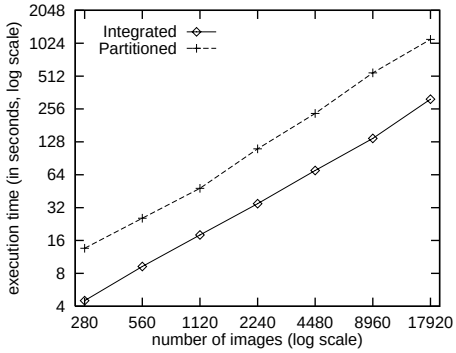

Figure 29: Retrieval time in seconds for various data set sizes for query Q3 using the integrated and partitioned organizations.

the moderate slopes in Figure 26. The partitioned organization serves the same purpose; however, there is no need to build the spatial index on the fly when processing the query, since this spatial index already exists as part of the database in the partitioned organization. Thus, plan P2D, which utilizes this organization, usually outperforms plan P2C. Notice, however, that as the search radius increases and as the size of the data set increases, the difference between plan P2C and plan P2D becomes small (see Figures 23, 26 and 27).

Finally, we compare the execution time of query Q1 and Q3 using both organizations. Figures 28 and 29 report the retrieval times that were required to process queries Q1 and Q3 for various sizes of the data set. The retrieval time for Q1 is almost identical for both organizations. However, for query Q3 which has a low contextual selectivity, the partitioned organization performs very poorly.

8 Concluding Remarks

MARCO, a system for retrieving map tiles by content, has been presented. The input to MARCO are physical images of separate map layers, physical images of map composites, and their logical representations. MAGELLAN, a legend-driven map interpretation system, converts map layer images from their physical to their logical representation. This logical representation is then used to automatically index both the composite and the layer images. Indices are constructed on both the contextual and the spatial data thereby enabling efficient retrieval of layer and composite images based on contextual as well as spatial specifications. Users may fine-tune the performance MARCO in terms of the accuracy that they require by setting the search bound value and minimum certainty values to fit their particular application. In terms of query execution times, the performance varies according to the contextual and spatial selectivity factors of the query, the plan chosen to execute the query, and the organization of the logical images. By choosing the appropriate query processing strategy based on the results of the empirical comparison presented in this paper, MARCO can be very efficient.

The small amount of user intervention required during map image input, the advanced indexing mechanisms on contextual, spatial, and meta data, and the efficient query processing strategies employed by MARCO, make it highly suitable to handle large repositories of maps such as those

that will be abundant in digital libraries. Although MARCO was designed for maps it can easily be adapted to many other types of images that are of a symbolic nature (termed *symbolic images*). These include CAD/CAM documents, engineering drawings, floor plans, etc. Note that we have used a similar system for the interpretation of floor plans [24]. The results of this interpretation could be incorporated into a database in a similar manner. The main difference would be in the graphical query interface that would need to be adapted to query categories suitable for such a floor plan application.

Another complication that could arise in these applications is that the spatial extent of the symbol may be of importance. In MARCO, the symbols found in an image were represented by a point (the center of gravity of the symbol relative to the entire image). In other applications, we may need to use bounding boxes or other geometric entities to represent the symbols in the logical image. However, by using the methods suggested in this paper we can handle objects with spatial extent just as easily. The only difference would be in the selection of the spatial data structure that is used to index the locational information in the logical images. For example, if we chose to represent the symbols by bounding boxes, then we could use any data structure that is suitable for indexing a large number of rectangles such as an R-tree [8]. We could then utilize the well-known algorithms that exist for range queries on these data structures.

In the system that we describe in this paper, the automatic indexing of map composites is done according to a map layer that contains geographic symbols. In order to index by other layers that contain additional types of symbolic information such as roads, bodies of water, etc., other methods that are suitable for interpreting this kind of symbolic information need to be developed. The results of such an interpretation can then be integrated into the map image database system using spatial indexing methods that are suitable for corresponding data types such as lines, polygons, etc. This would enable the system to provide a comprehensive tool to utilize the vast amount of data that is found in paper maps.

9 Acknowledgements

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

References

[1] S. V. Ablameyko, B. S. Beregov, and A. N. Kryuchkov. Computer-aided cartographical system for map digitizing. In *Proceedings of the Second International Conference on Document Analysis and Recognition.*, pages 115–118, Tsukuba Science City, Japan, October 1993.

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

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, Arlington, VA., January 1994.

[4] 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.

[5] G.L Cash and M. Hatamian. Optical character recognition by the method of moments. *Computer Vision, Graphics, and Image Processing*, 39(3):291–310, September 1987.

[6] P. Devijver and J. Kittler. *Statistical Pattern Recognition*. Prentice-Hall, Englewood-Cliffs, NJ, 1982.

[7] 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.

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

[9] 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*, number 951 in Lecture Notes in Computer Science, pages 83–95, Portland, ME, August 1995.

[10] R. Jain. NSF workshop on visual information management systems. *SIGMOD RECORD*, 22(3):57–75, September 1993.

[11] R. Kasturi, R. Raman, and C. Chennubhotla. Document image analysis an overview of techniques for graphics recognition. In *Proceedings of the IAPR Workshop on Syntactic and Structural Pattern Recognition*, pages 192–230, Murray Hill, New Jersey, June 1990.

[12] M. L. Larsgaard. *Map Librarianship: an Introduction*. Libraries Unlimited, Littleton, CO, 2nd edition, 1987.

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

[14] 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).

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

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

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

[18] D. J. Peuquet. An examination of techniques for reformatting cartographic data part 1: The raster-to-vector process. *Cartographica*, 18(1):34–48, January 1981.

[19] J. Rasure and C. Williams. An integrated visual language and software development environment. *Journal of Visual Languages and Computing*, 2(3):217–246, September 1991.

[20] A. Rosenfeld and A.C. Kak. *Digital Picture Processing*. Academic Press, New York, second edition, 1982.

[21] D. Rotem. Spatial join indices. In *Proceedings of the Seventh International Conference on Data Engineering*, pages 500–509, Kobe, Japan, April 1991. IEEE Computer Society, IEEE Computer Society Press.

[22] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1990.

[23] H. Samet. Spatial data structures. In W. Kim, editor, *Modern Database Systems, The Object Model, Interoperability and Beyond*, pages 361–385. ACM Press and Addison-Wesley, 1995.

[24] 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.

[25] H. Samet and A. Soffer. Integrating images into a relational database system. Technical Report CS-TR-3371, University of Maryland, College Park, MD, October 1994.

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

[27] H. Samet and A. Soffer. Magellan: Map acquisition of geographic labels by legend analysis. Technical Report CS-TR-3386, University of Maryland, College Park, MD, December 1994.

[28] J. Star and J. Estes. *Geographic Information Systems*, chapter 6, pages 85–91. Prentice-Hall, Englewood Cliffs, NJ, 1990.

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

[30] S. Suzuki and T. Yamada. MARIS: Map recognition input system. *Pattern Recognition*, 23(8):919–933, August 1990.

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

[32] N. Tanaka, T. Kamimura, and J. Tsukumo. Development of a map vectorization method involving a shape reforming process. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, pages 680–683, Tsukuba Science City, Japan, October 1993.