A Map Acquisition, Storage, Indexing, and Retrieval System *

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

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
aya@umiacs.umd.edu

Abstract

A system for the acquisition, storage, indexing, and retrieval of map images is presented. The input to this system are raster images of separate map layers and map composites. A legend driven map interpretation system converts layer images from a physical to a logical representation. This logical representation is used to automatically index both the composite and the layer images. Methods for incorporating logical and physical layers as well as composite images into the framework of a relational database management system are described. An example query and a corresponding query processing strategy that uses these indices is presented. The user interface is demonstrated via an example query execution.

1 Introduction

The paper map has long been the traditional representation of spatial data. Today, we are seeing the emergence of geographic information systems (GIS) as a replacement. One of the central issues in this field is how to integrate paper maps into a GIS. In particular, we would like to store scanned images of paper maps (termed *map images*) and 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".

In order to support retrieval by content, 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 also preserve the spatial information inherent in the map image. Both the logical and the physical representation of the map images are stored in the database. An index mechanism for the logical representation can then be used to retrieve map images based on both contextual and spatial information in an efficient way.

The process of converting a map image from its physical to its logical representation is the subject of the field of map interpretation. There has been some research in recent years on automating this process. Most researchers have focussed on skeletonization and vectorization methods [4, 8]. Unfortunately, this does not always yield accurate and useful results. One problem in performing this conversion is that a paper map is nothing more than 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 color and size of city names on the map convey information about the population of a city. Many graphical symbols are used to indicate the location of various sites such as post offices, scenic areas etc. The key to this symbolic information may be found on the map itself in the form of the legend. In [7], we described a map interpretation system that was built by us that performs legend-driven geographic symbol recognition.

In this paper, we present a system for acquisition, storage, indexing, and retrieval of map images. The input to this system 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 interpreta-

*The support of the National Science Foundation under Grant IRI-9017393 and HPCC/GCAG Grant BIR-9318183, and the National Aeronautics and Space Administration under Grant NGT-30130 are gratefully acknowledged.

tion system described in [7] is used to convert map layer images from their physical to logical representation. This logical representation is then used to automatically index both the composite and layer images. In this paper, 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. The logical map images are stored as tuples in a relation. Indices are constructed on both the contextual and the spatial data. Although this system was designed for maps, it can be adapted easily to handle many other types of documents that are of a symbolic nature. These include CAD/CAM documents, engineering drawings, floor plans, etc.

2 Map Acquisition and Conversion

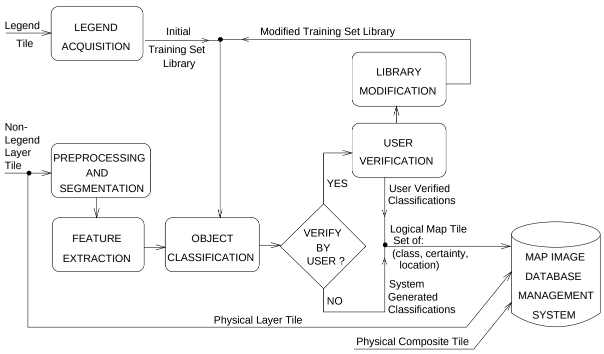

Figure 1: Map interpretation system

Map layers and composites are scanned and divided into small *tiles* (i.e., of size 512×512 pixels). Figure 1 is a block diagram of the map interpretation system that we have developed. 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. See [7] for a complete description of this process.

The output of the conversion is a logical map tile. It consists of the candidate classifications that were made, the certainty of the classifications, and the corresponding location of the symbols found in the map tile. The logical map tile, physical layer tiles, and composite tiles are input to the map image database management system.

3 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 [1] (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.

3.1 Schema Definitions

```
(create table       (create table       (create table
   classes             phys_mi             log_mi
   class CHAR[30],     img_id INTEGER,     class CHAR[30],
   seman CHAR[50],     desc CHAR[50],      cert FLOAT,
   bitmap IMAGE);      low_left POINT,     limg_id INTEGER,
                       raw IMAGE);         cimg_id INTEGER,
                                           l_loc POINT,
                                           c_loc POINT);
```

Figure 2: schemas for the relations `classes`, `phys_mi`, and `log_mi`.

The schema definitions given in Figure 2 define the relations that are used by the map image database system. We use an SQL-like syntax. The `classes` relation has one tuple for each classification used by the system. The `class` field stores the name of the classification (e.g., star), the `seman` 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.

The `phys_mi` relation has one tuple per map tile T in the database. These include both layer tiles and composite tiles. The `desc` field stores an alphanumeric description of the tile T that the user gives when inserting T. The `raw` field stores the actual tile T in its physical representation. It is an attribute of type `IMAGE`. The `low_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.

The `log_mi` 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. The `class` and `cert` fields store the name of the class C to which the map tile conversion system classified s and the certainty that $s \in C$. The `limg_id` and `cimg_id` fields are the integer identifiers assigned to the corresponding layer tile LT and composite tile CT, respectively, that contain symbol s. The `l_loc` and `c_loc` fields store the (x, y) coordinate values of the center of gravity of s relative to the non-tiled layer and composite images, respectively.

Alphanumeric and spatial indices are defined on the schemas of the map image database. These indices include an alphanumeric index that is used to search the `log_mi` relation by the `class` attribute, and a spatial index that is used to search the `log_mi` relation by location. The spatial indices are implemented by a PMR quadtree for points [2].

4 Map Image Retrieval

An example query seeks to "display all layer and composite tiles that contain a beach within 5 miles of a hotel." This query can be given in SQL format

```
display PI1.raw PI2.raw
  from log_mi LI1, log_mi LI2, classes C1,
        classes C2, phys_mi PI1, phys_mi PI2
  where C1.seman = "beach" and
      C2.seman = "hotel" and
      C1.class = LI1.class and
      C2.class = LI2.class and
      distance(LI1.l_loc,LI2.l_loc) < 15 and
      LI1.limg_id = LI2.limg_id and
      PI1.img_id = LI1.limg_id and
      PI2.img_id = LI1.cimg_id;
```

The following execution plan outlines how a response to this query is computed using the available indexing structures. See [6] for a more detailed plan and for more 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. The strategy is to search for hotel tuples using an alphanumeric index on `class` and search for site of interest tuples using a spatial index on `l_loc`.

```
get all tuples of log_mi which correspond to "hotel"
  (use index on the class attribute)
for each such tuple t
  get all points within 5 miles of t.l_loc
    (use spatial index on the l_loc attribute)
  for each one of these points p
    if p is a "beach" and in same map tile then
      display the corresponding physical
        map layer and composite tiles
```

4.1 User Interface

Queries may be given to our system using either an SQL-like language or a graphical user interface (GUI). The SQL-like language used by our system 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 a symbol from a given *class*.

One within Query find all map tiles in which a symbol from *class2* is within a given *distance* from a symbol from *class1*.

All Within Query find all map tiles in which a symbol from any class is within a given *distance* from a symbol from *class1*.

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

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

An additional difference between queries specified using SQL and queries specified using the GUI is in the cost (in terms of time) of responding to the queries. For queries that are specified using SQL, the query plan is generated automatically. Thus the cost of computing the result is determined by the quality of SAND's query optimizer. The problem of writing a query optimizer for a spatial database is very complex [1], 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.

5 Implementation and Tests

The 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. The map layer was scanned at 240dpi. The layer was split into 425 tiles of size 512×512. The map composite was scanned at 160dpi. The layer was split into 551 tiles of size 256×256. The composites were scanned at a lower resolution in order to reduce the space required to store these tiles.

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 the map interpretation system outlined in Section 2. 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. The results of this conversion (i.e., the logical tiles) were input to SAND and inserted into the `log_mi` relations as defined in Section 3. The physical layer and composite tiles were also input to SAND and inserted into the `phys_mi` relation. The GUI and the plans for the five query categories were implemented using Tcl (short for Tool Command Language), and Tk, a toolkit for the X window system [3].

5.1 An Example Query Execution

The following scenario describes how the example query, "display all layer and composite tiles that contain a beach within 5 miles of a hotel", is specified using the GUI and how the results are presented. Figure 3 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 `seman` fields of the tuples in the `classes` relation. To perform a "one within" query, the user first selects the icons of the two required classes followed by pressing 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 the plan that was outlined in Section 4. The result of this query is displayed in a window as seen in Figure 4. 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. The result tiles are displayed in decreasing order of the certainty of the response. The user may now display any of the result tiles by selecting the corresponding thumbnails. A square is drawn around the two symbols that were given to the query. The user may also choose to display the corresponding composite tile. In addition, the user may choose to display the non-tiled map with the query result tiles highlighted.

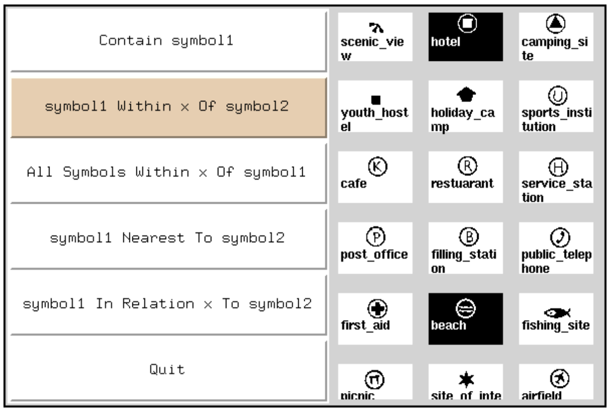

Figure 3: Graphical User Interface for query initiation. User has selected a "One Within Query" between a "hotel" and a "beach".

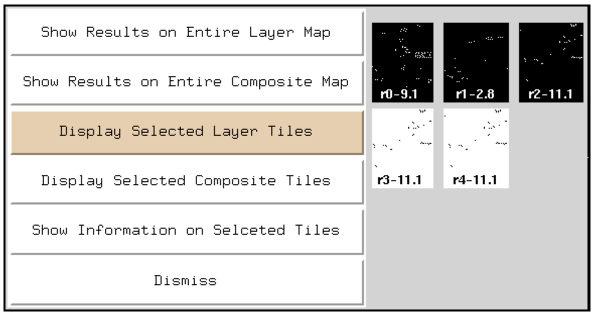

Figure 4: Results of query computation. The user has selected to display the layer tiles of the first three results.

5.2 Evaluating the System

We have identified two error types in order to evaluate the system in terms of accuracy.

Type I A tile that meets the query specification was not retrieved by the system (a miss).

Type II A tile that the system retrieved for a given query does not meet the query specification (a false hit).

Type II errors are counted by visual inspection of the result tiles. Each result tile that does not meet the query specification, is counted as a Type II error. We performed a "contain query" for each of the symbols in our application. 92% of the result tiles did in fact contain the desired symbol (a Type II error rate of 8%). Furthermore, 98% of those result with a certainty value over 0.5 contained the required symbol.

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. 94% of the tiles that should have been retrieved were in fact retrieved by the system. Thus we have a 6% type I error rate.

6 Concluding Remarks

Although our system was designed for maps it can easily be adapted to many other types of documents that are of a symbolic nature. These include CAD/CAM documents, engineering drawings, floor plans, etc.. Note that we have used a similar system for the interpretation of floor plans [5]. The results of this interpretation could be incorporated into a similar image database management system. 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.

In our system, 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.

7 Acknowledgements

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 Seventeenth International Conference on Very Large Data Bases*, pages 81–90, Barcelona, September 1991.

[2] 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.

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

[4] 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.

[5] 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.

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

[7] 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.

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