Journal of Visual Languages and Computing, 9(6), pp. 567–596, Dec. 1998

Pictorial Query Specification for Browsing Through Spatially-Referenced Image Databases [*]

Aya Soffer [†]
Computer Science Department and
Center for Automation Research and
Institute for Advanced Computer Science
University of Maryland at College Park
College Park, Maryland 20742
E-mail: aya@umiacs.umd.edu

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

December 26, 1999

Abstract

A pictorial query specification technique that enables the formulation of complex pictorial queries for browsing through a collection of spatially-referenced images is presented. It is distinguished from most other methods by the fact that in these methods the query image specifies a target database image in its entirety whereas in our approach the query image specifies the combination of objects that the target database image should contain rather than being treated as a whole image. The query objects are represented by shape features although other features such as color, texture, or wavelets could also be used. Using our technique, it is possible to specify which particular objects should appear in the target images as well as how many occurrences of each object are required. Moreover, it is possible to specify the minimum required certainty of matching between query-image objects and database-image objects, as well as to impose spatial constraints that specify bounds on the distance between objects and the relative direction between them. These spatial constraints can also be used to specify other topological relations such as enclosure, intersection, overlap etc. Each pictorial query is composed of one or more query images. Each query image is constructed by selecting the required query objects and positioning them according to the desired spatial configuration. Boolean combinations of two or more query images are also possible by use of AND and OR operators. A query image may be negated in order to specify conditions that should not be satisfied by the database images that are retrieved successfully. In addition, a capability is provided to specify whether the same instance of an object is to be used when it appears in more than one of the query images that make up the pictorial query, or whether two different instances are allowed. Several example queries are given that demonstrate the expressive power of this query specification method. An algorithm for retrieving all database images that conform to a given pictorial query specification is presented. The user interface for using this pictorial query specification method to browse the results in a map image database application is described and illustrated via screen shots.

[*]This paper is an extended version of [31].
[†]The support of the National Science Foundation under Grant CDA-950-3994 is gratefully acknowledged.
[‡]The support of the National Science Foundation under Grant IRI-97-12715 is gratefully acknowledged.

1 Introduction

Consider a collection of spatially-referenced images composed of several objects (or symbols) where both the topological layout and the distances among the objects are significant (e.g., maps, satellite images, aerial photos, floor plans, blueprints, etc.). The collection of images can be preprocessed so that the objects are classified according to their types and then stored in a database. In this case, queries would be made on the basis of the classifications. Alternatively, the collection of images can be left unprocessed and the queries would actually look for the objects (using sample objects) in the image thereby making use of computer vision, image processing, and pattern recognition techniques. Regardless of how the collection of images is represented, we may want to query them looking for particular objects in specific locations and/or relative spatial positions with respect to one another. One method to deal with such a query is by an SQL extension with additional predicates corresponding to spatial relationships. Unfortunately, this solution is only applicable to the first method of representing the collection of images — that is, the objects in the image must be preclassified so that the user can specify them by some alphanumeric tags. This solution is not applicable to the second method of representing the collection of images. In addition, if we want to find more complex images that involve several objects that must satisfy a particular spatial configuration or a choice among objects that satisfy some spatial configuration, then the corresponding SQL query would be very complex.

An alternative method is to specify the queries pictorially. This is a more "natural" method that facilitates the use of more complex constraints based on the implicit characteristics of the pictorial query (i.e., the particular objects in the pictorial query and their spatial arrangement). There are, however, several difficulties associated with pictorial query specifications. First of all, pictorial queries are inherently ambiguous which gives rise to several questions. In particular, what criteria should be used in order to determine that an object in a database image is the same as a particular object in the query image (termed *matching ambiguity*)? In addition, when query images are composed of several objects, are we looking for images that contain all of these objects, or would we be satisfied with any subset of these objects (termed *contextual ambiguity*)? Finally, is the spatial arrangement of the query objects of significance? For example, if one object in the query image is placed above and within 30 units of another object, what database images satisfy this query? One possibility is that only database images with exactly the same spatial configuration satisfy the query. However, the intent may be that only the distance must be the same, or maybe that any configuration may suffice (termed *spatial ambiguity*).

Another difficulty with pictorial queries is that they are not always as expressive as textual queries in terms of specifying combinations of conditions and negative conditions. For example, how do we specify pictorially images that contain beaches but do not contain camping sites within 3 miles of these beaches? It is desirable to have a pictorial query specification method that leverages on the expressiveness of pictorial queries in terms of describing what objects the target images should contain and their desired spatial configuration, while simultaneously resolving the matching, contextual, and spatial ambiguities as well as the limited expressiveness of pictorial query specifications.

This paper presents a pictorial query specification technique for image databases that we have developed that addresses the issue of matching, contextual, and spatial ambiguity inherent in pictorial queries. This method enables the formulation of complex pictorial queries that describe the target images in terms of their required contextual and spatial properties. The desired objects can be specified as well as how many occurrences of each object are required in the target images. Moreover, spatial constraints can be imposed that specify bounds on the distance between objects, as well as the relative direction between objects. These spatial constraints can also be used to specify other topological relations such as enclosure, intersection, overlap etc. We can handle objects with extent such as lines and regions in addition to point objects. Expressive power is achieved in our approach by allowing a pictorial query specification to be composed of one or more query images and by allowing a query image to be negated in order to specify conditions that should not be satisfied by the database images that are retrieved successfully. In addition, our technique provides a capability to specify whether the same instance of an object is to be used when it appears in more than one of the query images that make up the pictorial query specification (termed *object binding*), or whether two different instances are allowed.

Matching, contextual, and spatial ambiguity are resolved in our approach by providing a mechanism to specify the desired level of similarity in these three domains. The *matching similarity level* specifies a lower bound on the certainty that is required in order to match a query-image object to a database-image object. The *contextual similarity level* specifies how good a match is required between the query and database image in terms of overall content (i.e., collection of objects). For example, should the database image contain all of the objects in the query image or may it just contain some of these objects. The *spatial similarity level* specifies how good a match is required in terms of the relative locations and orientation of the matching symbols in the two images.

Using our pictorial query specification technique, we can specify a complex query such as “get all images that have at least two beaches in them with no restaurant within 5 miles of either beach, but with a two-lane road within 2 miles of both beaches”. Once we have the ability to formulate such complex queries pictorially, we must also address the issue of how to process these queries efficiently. In particular, which indexing structures are required and in what order should they be used when evaluating a query. Finally, the issue of how to display the results of a pictorial query so that the user can quickly browse through them is also important. All of these issues are addressed in this paper. It is important to note that although we describe a pictorial query specification tool, we do not address issues of usability (e.g., [23]) nor do we make any claims about its completeness (e.g., [34]). These issues are beyond the scope of this paper. As mentioned above, our goal is to explore the issues involved in providing a pictorial query specification tool for an image database. We do not claim to have solved all problems. Instead, we point out some of the issues that we encountered and outline our approaches to resolve them. Clearly, work remains to be done in this field.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the user interface for specifying pictorial queries in an example application as well as the process that we use for matching query-image objects to database-image objects in this application. In Section 4 we show how to resolve the matching, contextual, and spatial ambiguities inherent in pictorial queries. This includes a definition of the various levels of similarity that can be specified by the user and several sample queries that demonstrate how to use them. Section 5 shows how individual pictorial queries are combined to form compound queries along with examples of their use. In Section 6 we present an algorithm for retrieving all database images that conform to a given pictorial query specification. Section 7 contains concluding remarks.

2 Related Work

There have been a number of studies of pictorial queries for spatial and image databases in recent years [26]. Most of the image database research has dealt either with global image matching based on color and texture features [10, 12, 17, 20, 32, 33] or with the ambiguity associated with matching one query-image object to another [1, 8, 11]. For example, a method for searching an image database using a query image that is similar to the intended target was presented in [10]. The query image may be a hand-drawn sketch or a scan of the image to be retrieved. The result images are the ones that are most similar to the query image as a whole based on a multiresolution wavelet decomposition of the query and database images. This method (and the other global database search methods) do not address the case of images that are composed of several objects and their desired spatial configuration. In other words, these methods only address the problem of matching ambiguity and do not deal with contextual and spatial ambiguity at all.

There has also been some work on the specification of topological and directional relations among query objects [2, 4, 6, 7, 9, 14, 19, 24]. The focus of this work has been on defining spatial relations between objects and efficiently computing them when the objects are stored in a database. These studies only deal with tagged images (images in which the objects have already been recognized and tagged with their semantic meaning). Therefore, they do not address the issue of matching ambiguity. Furthermore, it is always assumed that the goal is to match as many query-image objects to database-image objects as possible, and, in most cases, it is also assumed that the relative locations of the objects must be exactly as specified by the query image or as close to that as possible (e.g., [9]).

A limited form of spatial ambiguity is allowed in pictorial queries based on the *2D-string* and its variants [4, 14] via a parameter that defines the type of subsequence matching that is required between the 2D-string representation of the query image and that of the database image. The spatial logic described in [2] also allows specification of query images in terms of spatial relations between objects and permits users to select the level of spatial similarity. However, the issue of the distance between objects is not addressed by these or any other method. In addition, it is assumed that the database images must contain all objects in the query image. Thus, the question of contextual and spatial ambiguity in its full extent is not considered. Furthermore, none of these methods provide Boolean combinations or negations of query images.

Another data structure called the *spatial orientation graph* is introduced in [9] and used for spatial similarity based retrieval of symbolic images. This representation does not capture any information about the distance between objects either. In addition, many assumptions are made about what a user might define as image similarity. These assumptions are used in computing the similarity value between the query image and the database images.

PQBE (a pictorial query-by-example language) [19] attempts to address the problem of the limited expressiveness of pictorial queries. PQBE can be used to express more complex queries by allowing pictorial queries that are composed of several query images joined by conjunctions and disjunctions, and by use of variable objects. PQBE, however, is a rather complex language that may not be easy to master (although it is probably simpler than SQL extensions dealing with spatial relations). While it is possible to specify directional constraints using PQBE, the distance between objects is ignored in PQBE as it is in all other methods dealing with spatial similarity. Furthermore, PQBE does not address the question of contextual and spatial ambiguity, and since it assumes that the objects in the pictorial queries are already classified, it does not address matching ambiguity either.

Spatial-Query-by-Sketch [7], a query language for geographic information systems, allows users to formulate a spatial query by drawing the desired configuration with a pen on a touch-sensitive computer screen. The results are ranked based on the similarity in terms of the spatial configuration. It considers mainly the topological configuration of the query objects. Distance metrics are used implicitly in order to relax the topological constraints. For example, if two query objects are very close to each other but disjoint, then a database image where the objects touch may also be considered as a match. In addition, distance metrics are used to rank results that are equivalent in terms of the topological configuration. However, distance is not considered as a condition for matching and thus bounds on distances can not be specified. Furthermore, the user cannot specify the desired spatial ambiguity. The system assumes that the topological relations sketched by the user should all hold. Spatial-Query-by-Sketch does not address contextual ambiguity. It is assumed that the results must contain all of the objects that are in the query sketch. Finally, Spatial-Query-by-Sketch does not allow Boolean combinations or negations as part of the query.

Another method for similarity based picture retrieval based on spatial relationships is described in [24, 25]. This method also allows for specifying query images pictorially. The main concern of this work however is how to map spatial relationships that are derived from the pictorial query to other comparable spatial relationships that have been used to describe the images in the database. For example, $leftOf(a, b) = rightOf(b, a)$. This method does not allow various levels of contextual similarity. That is, it is assumed that the goal is always to match as many query-image objects to database-image objects as possible. Furthermore, as in all the other cases, this method does not deal with spatial-locational information (e.g., distance) and does not allow pictorial queries that are composed of more than one query image.

In addition to this work in the field of pictorial query specification for image databases, there has been a large body of research in the field of graphical query languages for traditional (i.e., alphanumeric) database systems based on the relational, E-R, and object-oriented models. These include DOODLE [5], RBE [13], STBE [18], and GRAQULA [27]. Some of these languages (e.g., RBE [13]) support dynamic construction of the user interface for querying the database using a set of predefined widgets (e.g., sliders, scatter plots, and tables). However, all of these languages require knowledge of the underlying schemas that are used to model the data. Furthermore, these languages are complex since they need to support advanced features that traditional query languages such as SQL provide (e.g., aggregation, nested queries, etc.). On the other hand, these languages do not support the paradigm of similarity queries and query by sketch. In other words, it is not possible to draw (or construct from icons) a query image and request images that are similar to it. Finally, since these methods do not deal with images directly, the issue of matching ambiguity is not considered. Thus, they would need to assume that the images are already preprocessed and objects have been recognized and stored in the database as such.

In contrast to these methods, our approach handles queries that deal with both spatial-relational and spatial-locational data, as well as contextual information. Thus we can deal with the distance between objects as well as with their topological configuration. In addition, as part of the pictorial specification, the user indicates the degree of desired similarity, and thus the results are not subjective. Furthermore, we allow compound queries (via conjunction and disjunction of query image) with object binding, and thus provide a more expressive and comprehensive pictorial query specification method than any previously described methods.

3 Pictorial Query Specification and Browsing Results in an Example Application

In our approach, a pictorial query is composed of one or more query images. Each query image is constructed by selecting the symbols that should appear in the database images from a menu of symbols and by positioning these symbols so that the desired spatial constraints hold. In addition, the user must specify the image similarity level required to satisfy the matching, contextual, and spatial constraints between the query image and the required database images. The individual query images may be composed via AND and OR operators. In addition, a query image can be negated with the NOT operator in order to specify conditions that should not be satisfied by the database images that are retrieved successfully. In the case of the conjunction of query images where the same symbol appears in both query images, the user may specify whether the two query-symbols must match (i.e., be bound to) the same instance of the symbol in the database image, or whether two different instances are allowed (i.e., the instances may be different but need not be so)[1]. In order to bind two query-symbols to the same database symbol, the user selects the symbol for the second query image from the first query image, rather than selecting it from the menu of symbols.

[1]Binding is irrelevant in the case of disjunction of query images, since only one part of the clause needs to hold for the query to be satisfied.

We have implemented this pictorial query specification as a query interface for a map image database system developed by us [22] named MARCO (denoting MAp Retrieval by COntent). The input to MARCO are raster images of separate map layers (*map layer images*) and raster images of *map composites* (the maps that result from composing the separate map layers). Map layer images are processed in order to extract contextual cues from the map layer that can be used to index the composite images. This process utilizes the symbolic knowledge found in the legend of the map to drive geographic symbol recognition. In particular, we focused on symbol layers which contain geographic symbols that represent campsites, hotels, recreation areas, etc.

This input process requires some user intervention in order to build an initial training set. Once this is done, the current training set library is used to assign candidate classifications to each symbol using a *weighted bounded several-nearest neighbor classifier* [3]. A certainty value (between 0 and 1) is attached to each classification indicating how certain this classification is. In cases where there is more than one possible classification, all candidate classifications are returned by the classifier with their associated certainty value and stored in the database. Classification is based on a set of *features* that describe the symbol's shape. Each layer is first split into several tiles (since the whole map is too large to process). Each tile is segmented into its constituent elements using a connected component labeling algorithm (e.g., [21]). For each region in the labeled image, a set of *features* based on its shape is computed. These features include some global (e.g., first invariant moment, circularity, eccentricity, rectangularity) and some local shape descriptors (e.g., intersections, gaps) [15] 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. 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. Many of the geographic symbols that we classify are composed of a circle (or rectangle) enclosing one or more small shapes. We use a representation termed *negative symbol* that is based on the interior of these symbols with the shapes considered as holes [30]. Note, that in our application we we use shape features to resolve the matching ambiguity (i.e., to match query-image symbols to database-image symbols). However, for other applications we could use different features for this purpose. For example, we could use color or texture features which are commonly used in image databases, or wavelet features as described in [10].

3.1 Pictorial Query Builder

Figure 1 shows the pictorial query builder used by MARCO. The user has constructed a query to retrieve all database images that contain a hotel ◙ within 6 miles of a beach ⊜ and do not have an airport ✈ within 1 mile of the beach ⊜ (see Figure 2 for a description of the symbols used in this query and in the rest of this paper). Furthermore, the certainty that the database-image symbols are in fact a hotel ◙ , beach ⊜ , and airport ✈ is ≥ 0.5 (determined by *msl*, matching similarity level). The symbols are "dragged and dropped" from the menu of symbols displayed in the bottom of the window. The query builder constructs this menu of symbols directly from the database which stores one example of each symbol relevant for the application at hand. These example symbols are taken from the legend of the map in our example application. Alternatively, provisions exist for the user to import examples of symbols directly. Thus, the interface can automatically adjust to a different set of symbols. We use a color coding scheme to denote that two query-image symbols are bound to the same instance in the database image. That is, two symbols that have the same non-black color are bound, whereas black symbols are not bound. For example, in the query in Figure 1, the two beach ⊜ symbols are a lighter color (blue in our system), and thus they are bound. That is, the same instance of the database-image beach ⊜ symbol must be matched to the query-image beach ⊜ symbols in both clauses of

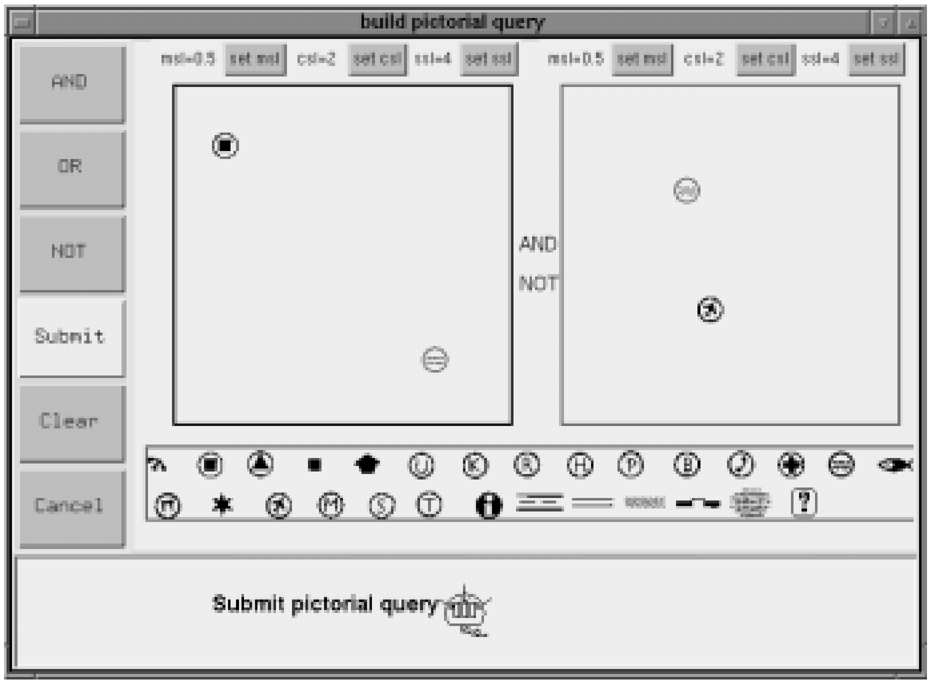


Figure 1: Tool for constructing pictorial queries.The user has constructed a query to "retrieve all database images that contain a hotel ▣ within 6 miles of a beach ⊜ and do not have an airport ⊛ within 1 mile of the beach ⊜ , where the certainty that the found symbols are in fact a hotel ▣ , beach ⊜ , and airport ⊛ is ≥ 0.5."

the pictorial query. Matching, contextual, and spatial similarity levels are set via menu buttons "set msl", "set csl", and "set ssl" respectively (see Figures 3– 5 for the corresponding menus).

⊛	airport	◀●	fishing site	▲	camping site	▨	local road
Ⓑ	gas station	⊜	beach	▣	hotel	═	one-lane road
Ⓚ	cafe'	⊕	first aid	↷	scenic view	☰	two-lane road
Ⓡ	restaurant	Ⓜ	museum	✶	site of interest	━	railroad
Ⓣ	picnic site	Ⓟ	post office	?	wild card	▦	open field

Figure 2: Symbols and their semantic meaning.

In our example application we have limited ourselves to symbolic images since this enabled us to use rather simple pattern recognition methods to resolve the matching ambiguity (i.e., how to match symbols in the query image to the database images). However, our pictorial query specification technique can easily be used for other images as well. The prerequisites for handling such images are: 1) segment the image into separate objects (or entities). 2) compute some features that characterize each object (e.g., color, texture, shape, wavelet coefficients). 3) provide a similarity measure that approximates the certainty that two objects given by their characteristic features are the same. Once these three prerequisites have been satisfied, the query specification and processing would remain the same. In terms of the user interface, a replacement for the menu of symbols is required since we can no longer assume that we have an example bitmap of each

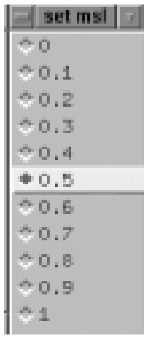


Figure 3: Menu for setting matching similarity level (msl).

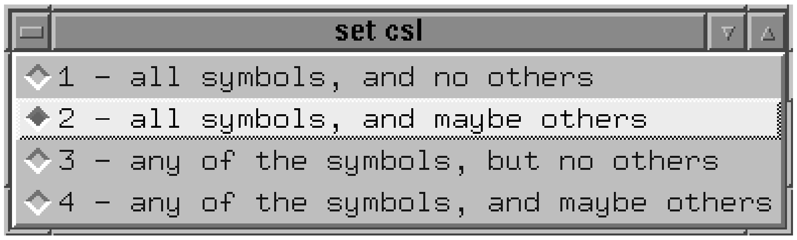

set csl
◇1 – all symbols, and no others
◆2 – all symbols, and maybe others
◇3 – any of the symbols, but no others
◇4 – any of the symbols, and maybe others

Figure 4: Menu for setting contextual similarity level (csl).

object that may appear in the database images. The alternatives are either a sketch tool that would let the user draw exemplars of the required objects, a tool that would let users provide samples of objects to be put into a menu, or enabling users to import images that already contain all of the required objects in the desired spatial configurations. For example, in a database of satellite images we could segment the image into objects based on texture features or based on multispectral signatures. This representation would be used to resolve the matching ambiguity between query and database objects. A sample object could be identified to the database by an image "patch". A query image could then be composed of several such objects, and our pictorial query specification method could be used to specify matching, contextual, and spatial constraints among these objects, as well as to specify compound queries and negative constraints. For example, find places where there is no road within 1 mile of a forest.

3.2 Browsing Results of Pictorial Queries

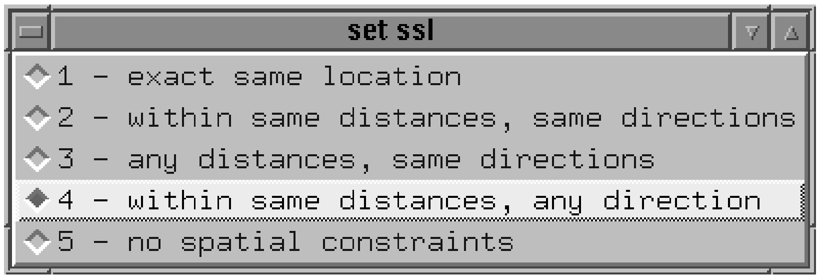

set ssl
◇1 – exact same location
◇2 – within same distances, same directions
◇3 – any distances, same directions
◆4 – within same distances, any direction
◇5 – no spatial constraints

Figure 5: Menu for setting spatial similarity level (ssl).

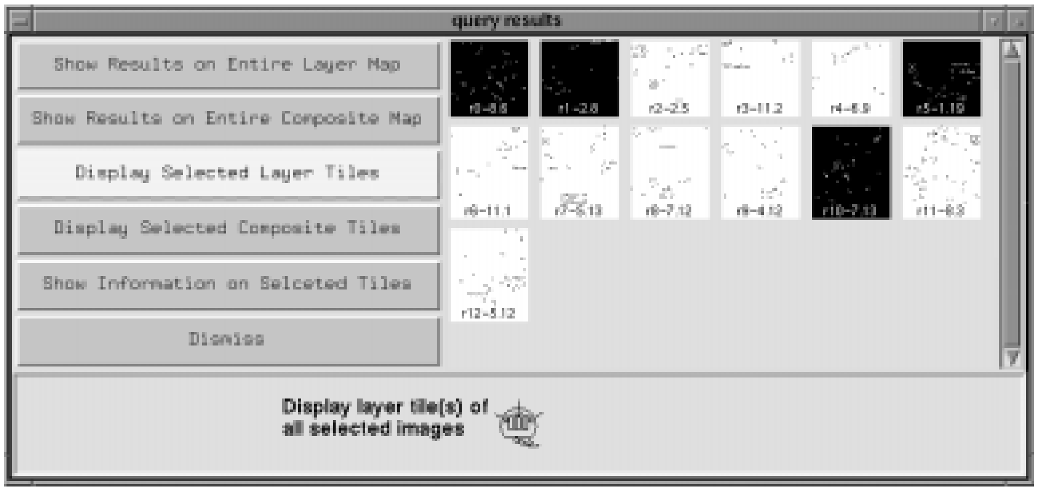

query results
Show Results on Entire Layer Map
Show Results on Entire Composite Map
Display Selected Layer Tiles
Display Selected Composite Tiles
Show Information on Selceted Tiles
Dismiss
Display layer tile(s) of all selected images

Figure 6: Results of query computation. The user has selected to display the layer tiles of four results.

The example database that we have used to test our system consists of the red sign layer and the composites (all layers) of the GT$_3$ map of Finland, which is one of a series of 19 GT maps that cover the whole area of Finland. The red sign layer contains geographic symbols that mostly denote tourist sites. The map was scanned at 240dpi. The layer was split into 425 tiles of size 512 $\times$ 512. These tiles were automatically processed and symbol recognition was performed as outlined above. The logical representation of these tiles as well as the physical (raw) images of the layer and composites are stored in the database. After the user poses

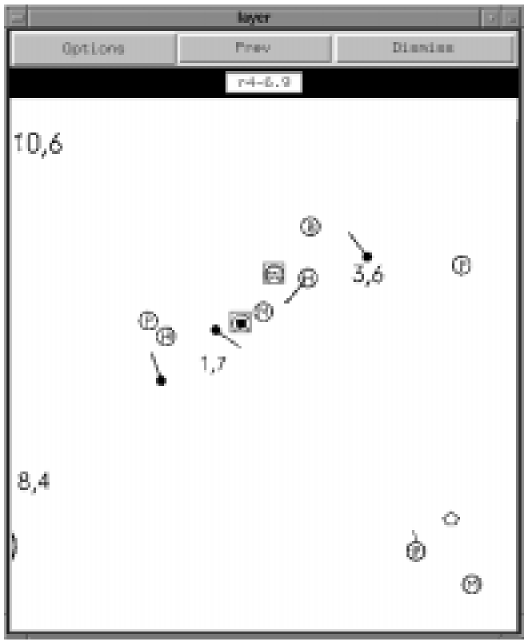


Figure 7: Displaying the selected layer tiles (the query symbols are surrounded by a square).

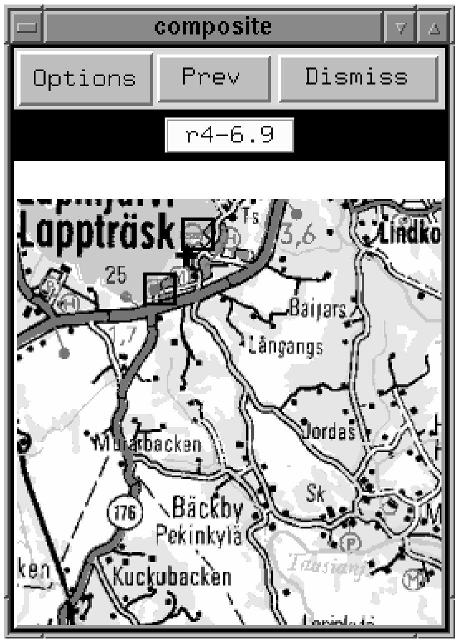


Figure 8: Displaying the selected composite tiles (the query symbols are surrounded by a square).

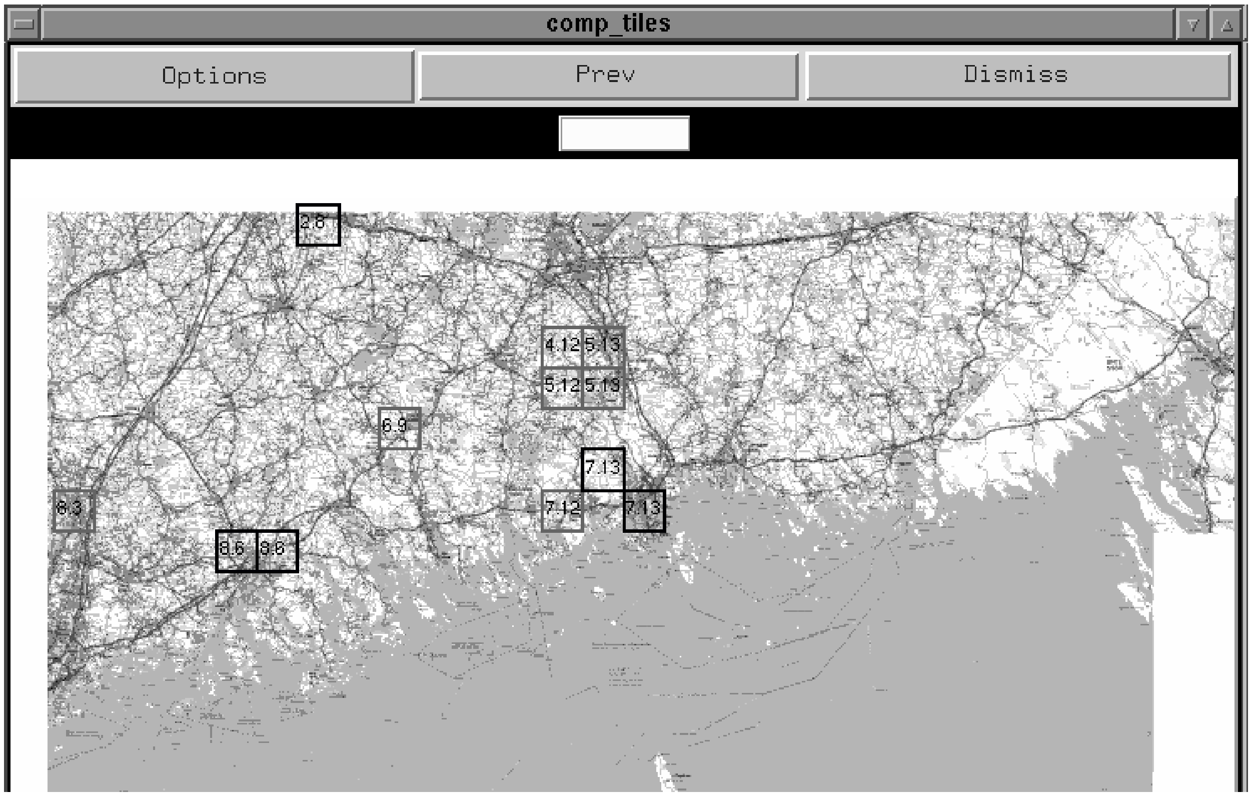


Figure 9: Showing the result tiles on the non-tiled map (selected tiles are a lighter color).

the pictorial query, the result of this query is displayed in a window as seen in Figure 6. A thumbnail (i.e., a reduced bitmap of the whole tile) is displayed for each tile that was found that meets the query specification. The result tiles are displayed in decreasing order (from left to right and top to bottom) of the average certainty value of the matches between the query-image symbols and database-image symbols. 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. Figures 7 and 8 show the results of clicking these two buttons, respectively. The “prev” button in Figure 7 and 8 is used to step through the selected tiles. A square is drawn around the symbols that were part of the pictorial specification. By clicking the “Information” button in Figure 6, the user can see the information stored regarding each of these tiles in the database and the exact 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 (e.g., Figure 9). The tiles corresponding to thumbnail images that are selected in the results window are highlighted in red (dark in Figure 9), while the remaining tiles are highlighted in green (light in Figure 9). If the result of a query lies in two tiles (e.g., the beach ⊜ is in one tile and the hotel ▣ is in another tile), then the result is composed of both thumbnails. Selecting either one of them for display will show both tiles.

4 Resolving Matching, Contextual, and Spatial Ambiguity

In this section we describe how the matching, contextual, and spatial ambiguity inherent in pictorial queries is resolved by explicitly specifying the required level of similarity between query image *QI* and database image

DI in these three domains. We first define the various levels of similarity. This is followed by several examples of pictorial queries that demonstrate the use of our pictorial specification method. We define similarity using the following definitions. A *symbol* s is a group of connected pixels that together have a common semantic meaning. A *class* C is a group of symbols that all have the same semantic meaning.

4.1 Matching Similarity

Matching similarity specifies how close a match between a symbol s_1 in the query image *QI* and a symbol s_2 in the database image *DI* is required in order to consider them to be the same. The matching similarity level *msl* is a number between 0 and 1 that specifies a lower bound on the certainty that two symbols are from the same class. In other words, if the certainty that s_1 and s_2 are from the same class $\geq$ *msl*, then s_1 and s_2 will be considered a match. Note that the certainty that two symbols are from the same class can be derived from precomputed certainties output by a classifier as in our example application (see Section 3). Alternatively, if symbols are represented in the database by a characteristic feature vector, this certainty can be computed at query time based on the distance in feature space between the feature vectors representing the symbols. For more details on the similarity features that we used, see [30].

4.2 Contextual Similarity

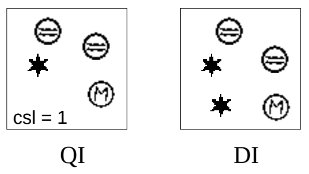
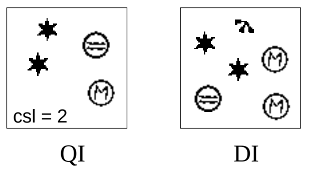
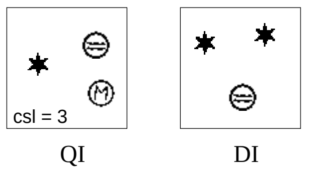
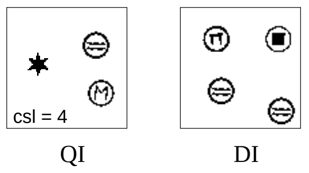
1. Every symbol in QI has a distinct matching symbol in DI, and every symbol in DI has a matching symbol in QI	
2. Every symbol in QI has a distinct matching symbol in DI (DI may contain additional symbols from any class)	
3. Every symbol in DI has a matching symbol in QI	
4. At least one symbol in QI has a matching symbol in DI (DI may contain additional symbols from any class)	

Figure 10: Four levels of contextual similarity (csl).

Contextual similarity specifies how well the content of database image *DI* matches that of query image *QI* (e.g., do all of the symbols in *QI* appear in *DI*). We measure contextual similarity by varying two parameters.

Each parameter has two possible values. The first parameter indicates if all of the symbols of *QI* must have matching symbols in *DI* or if a subset sufficient (i.e., whether an AND or OR of the symbols is required). The second parameter indicates whether *DI* may contain symbols that do not match any symbol in *QI*. Considering all the combinations of these two parameters, we define the following four levels of contextual similarity (see Figure 10 for examples):

1. Every symbol in *QI* has a distinct matching symbol in *DI*, and every symbol in *DI* has a matching symbol in *QI*.
2. Every symbol in *QI* has a distinct matching symbol in *DI* (*DI* may contain additional symbols from any class).
3. Every symbol in *DI* has a matching symbol in *QI* (*QI* may contain additional symbols from any class).
4. At least one symbol in *QI* has a matching symbol in *DI* (*DI* may contain additional symbols from any class) and vice versa.

Note that the definition of *csl* is not symmetric. When matching *QI* symbols to *DI* symbols, we require a match for each distinct symbol in *QI*. On the other hand, when matching *DI* symbols to *QI* symbols, we only require a match for each symbol class in *QI* (not for each distinct symbol). The rationale for this asymmetry is to support an intuitive interpretation of multiple symbols from the same class in both *QI* and *DI*. If a user specifically places more than one occurrence of the same symbol in *QI*, then most likely the intention is to search for database images with as many (or maybe more) occurrences of this symbol. However, if a user places only one occurrence of a symbol in *QI*, then the user most likely does not mind if there is more than one potential match for this symbol in the retrieved database image. These cases are illustrated in the examples in Figure 10. Nevertheless, it is possible to specify an exact number of desired symbols in a database image using compound queries and negation as described in Section 5 and demonstrated in Figure 24.

4.3 Spatial Similarity

Spatial similarity specifies how good a match is required in terms of the relative locations and orientation of the matching symbols between the query and database image. In order to define spatial similarity levels we need to distinguish between various spatial symbol types. In our application, a symbol may correspond to a point (e.g., a museum Ⓜ), a line (e.g., a local road ▨), or a polygon (e.g., an open field ▦). The location of a symbol $loc(s)$ is defined as follows:

$$loc(s) = \begin{cases} \text{the } (x,y) \text{ coordinate values of the center of gravity of } s, & \text{when } s \text{ is a point symbol} \\ \text{the } (x,y) \text{ coordinate values of the end points of } s, & \text{when } s \text{ is a line symbol} \\ \text{the } (x,y) \text{ coordinate values of the upper left and} & \\ \text{bottom right corners of the minimum bounding} & \\ \text{rectangle of } s \text{ whose sides are parallel to the axes}, & \text{when } s \text{ is a polygon symbol} \end{cases}$$

The distance between two symbols $dist(s_1, s_2)$ is defined as the Euclidean distance between s_1 and s_2. $dist(s_1, s_2) = 0$ when the two symbols intersect (e.g., a line symbol intersects a polygon symbol). $dist(s_1, s_2) = -\infty$ if one symbol is totally enclosed in the other (e.g., a line symbol is inside a polygon symbol). For example, the distance between a line l_1 and a polygon p_1 represented by its minimum bounding rectangle r_1 is defined as follows:

$$\textit{dist}(\text{line } l_1, \text{ rect } r_1)\text{:} \begin{cases} 0, & \text{when } l_1 \text{ intersects } r_1 \\ -\infty, & \text{when } l_1 \text{ is inside } r_1 \\ dist(l_1, l_2), \text{ where } l_2 \text{ is the edge of} & \text{otherwise} \\ r_1 \text{ closest to } l_1 & \end{cases}$$

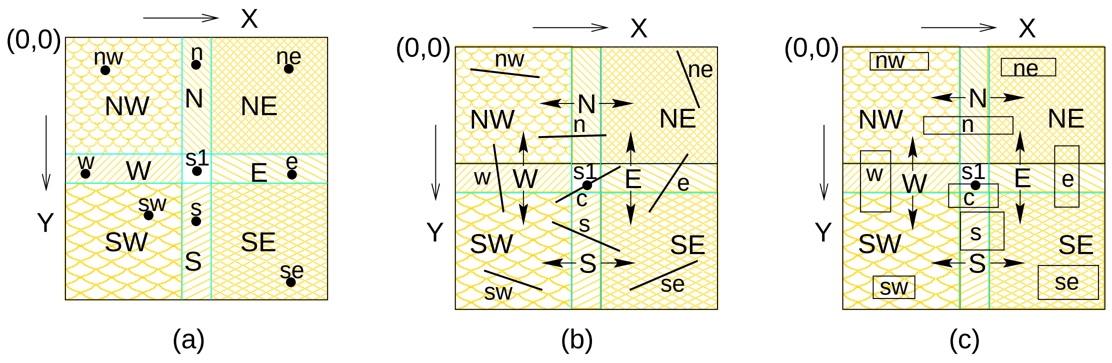


Figure 11: Possible directional relations with respect to point symbol s_1 for: (a) point symbols, (b) line symbols, (c) polygon symbols represented by their minimum bounding rectangles.

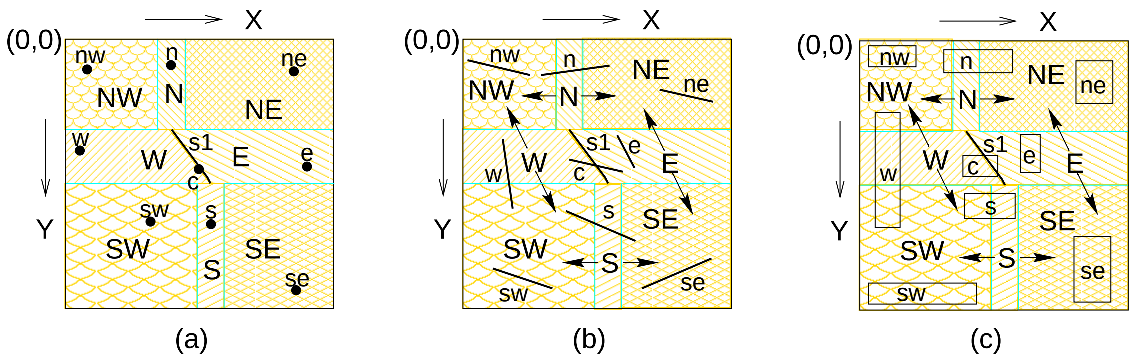


Figure 12: Possible directional relations with respect to line symbol s_1 for: (a) point symbols, (b) line symbols, (c) polygon symbols represented by their minimum bounding rectangles.

Let $rel(s_1, s_2)$ denote the relative position of symbol s_2 with respect to symbol s_1. In our implementation, the function $rel(s_1, s_2)$ can take on one of the following values: N,NW,W,SW,S,SE,E,NE,C where N,W,S,E are the four cardinal directions, NW,NE,SW,SE are the diagonal directions, and C denotes coincidence. The definition of $rel(s_1, s_2)$ is in terms of $loc(s_1)$ and $loc(s_2)$, and varies depending on the types of the argument symbols. Figures 11, 12, and 13 illustrate these relations for point, line, and polygon symbols (represented by their minimum bounding rectangles), respectively. The figures show the region covered by each direction as well as one example symbol in each direction. Note that in the case of lines and polygons, any symbol that passes through a region labeled N,W,E,S is considered to satisfy that relation with the reference symbol, whereas to be considered NE,SW,SE, or SW of the reference symbol, the symbol must be totally enclosed in the corresponding region. All of these relations can be easily computed based on the $loc(s)$ attribute of each symbol. For example, assuming an origin in the upper left corner, for point symbols: $(loc_x(s_j) < loc_x(s_i) \wedge loc_y(s_j) < loc_y(s_i)) \Rightarrow NW(s_j, s_i)$. Whenever the two symbols coincide,

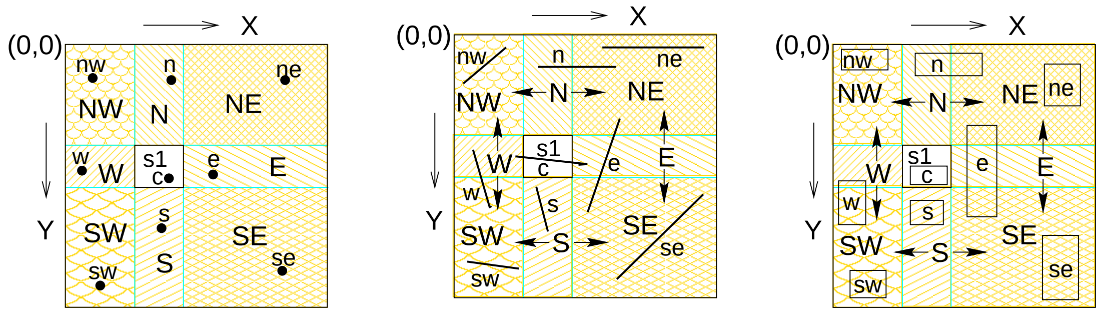


Figure 13: Possible directional relations with respect to polygon symbol s_1 for: (a) point symbols, (b) line symbols, (c) polygon symbols represented by their minimum bounding rectangles.

$rel(s_1, s_2) = C$. This definition can be refined to allow more detailed cardinal directions as defined in [7]. Furthermore, a wider variety of topological relations can also be expressed by adding cases that distinguish between the various relations (i.e. overlap, meet, etc.), rather than just using C to denote any form of coincidence.

In defining spatial similarity, we vary two parameters, each has two possible values. The first parameter indicates whether or not there exists a constraint on the distance between symbols. The second parameter

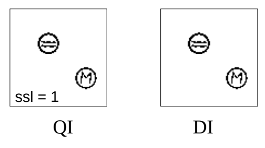
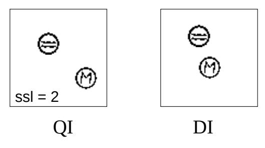
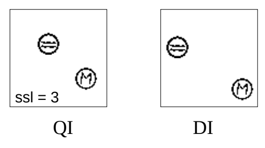
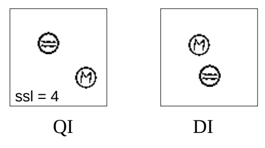
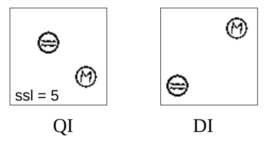
1. Matching symbols are in the same location	
2. Same relative position of matching symbols and distance between symbols in DI > L and <= distance between matching symbols in QI (L=0 by default)	
3. Same relative position of matching symbols but distance between matching symbols may vary	
4. Relative position of matching symbols varies but distance between symbols in DI > L and <= distance between matching symbols in QI (L=0 by default)	
5. Distance and relative position between matching symbols may vary (no spatial constraints)	

Figure 14: Five levels of spatial similarity (ssl).

indicates whether or not there exists a restriction on the relative direction between symbols. Considering all the combinations of these two parameters yields four spatial similarity levels. In addition to these four cases, we allow the restriction that the matching symbols must be in the exact same locations. Thus, the following five levels of spatial similarity are defined (see Figure 14 for examples):

1. The matching symbols of *QI* and *DI* are in the exact same locations in both images.
2. The relative position of the matching symbols of *QI*and *DI* is the same, and the distance between them is bounded from below by some given value L and bounded from above by the distance between the symbols in *QI*. By default $L = 0$. If $L = 0$, then for any symbols $s_i, s_j \in DI$, $s_k, s_l \in QI$, where s_i matches s_k and s_j matches s_l, $0 \leq dist(s_i, s_j) \leq dist(s_k, s_l)$ (i.e., it is a range search). If $L = dist(s_k, s_l)$, then $dist(s_i, s_j) = dist(s_k, s_l)$ (i.e., it is an exact distance search).
3. The relative position of the matching symbols of *QI* and *DI* is the same, but the distance between them may vary.
4. The relative position of the matching symbols of *QI* and *DI* may vary, but the distance between them is bounded from below by some given value L and bounded from above by the distance between the symbols in *QI*. By default $L = 0$.
5. The location of the matching symbols, the distance between them, and the relative position of these symbols may vary (i.e., no spatial constraints).

4.4 Total Image Similarity

The total similarity between *QI* and *DI* is defined by combining the three similarity factors. For example, $DI \equiv_{0.5,2,3} QI$ specifies that the matching, contextual, and spatial similarity of the two images is at levels 0.5, 2, and 3, respectively. That is, for each symbol in *QI* there is a matching symbol with a certainty ≥ 0.5 from the same class in *DI*, the location of the symbols and the distance between them may vary, but the inter-symbol spatial relationship between them is the same. In general, if $DI \equiv_{msl,csl,ssl} QI$ and if S' is the set of all the symbols of DI that match some symbol in QI with a certainty $\geq msl$, then the set of classes of the symbols of S' is a subset of the set of classes of the symbols of QI. Furthermore, for every pair of symbols s_1 and $s_2 \in S'$, the spatial constraints dictated by ssl and the positions of the matching symbols in QI hold. In other words, the spatial constraints must simultaneously hold between all of the matching symbols that appear in both query and database images. Note however, that S, the set of all symbols of DI, is not necessarily a subset of the set of classes of symbols of QI.

4.5 Example Queries varying csl

Figure 15 demonstrates the use of different contextual similarity levels for query specification. In all of these queries we assume that $ssl = 5$ (i.e., no spatial constraints are imposed). We do not specify *msl* in these or any other example queries since its use is straightforward and does not require further illustration. Query Qa requests all images that contain a site of interest ✱, a beach ⊜, a museum Ⓜ, and no other symbols. Query Qb requests all images that contain a site of interest ✱, a two-lane road ═, and at least one other symbol (there may be more). Query Qc requests all images that contain a beach ⊜, or a scenic view ↷, or an open field ▦ (an image may contain both) but no other symbols. Query Qd requests all images that contain a site of interest ✱, or a beach ⊜, or a museum Ⓜ (an image may contain one or all of them as well as other symbols).

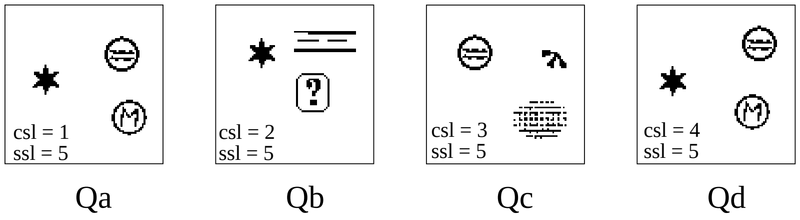


Figure 15: Pictorial queries demonstrating the use of different contextual similarity levels. The question mark $\boxed{?}$ symbol denotes a wild card (i.e., any symbol matches it).

4.6 Example Queries varying ssl

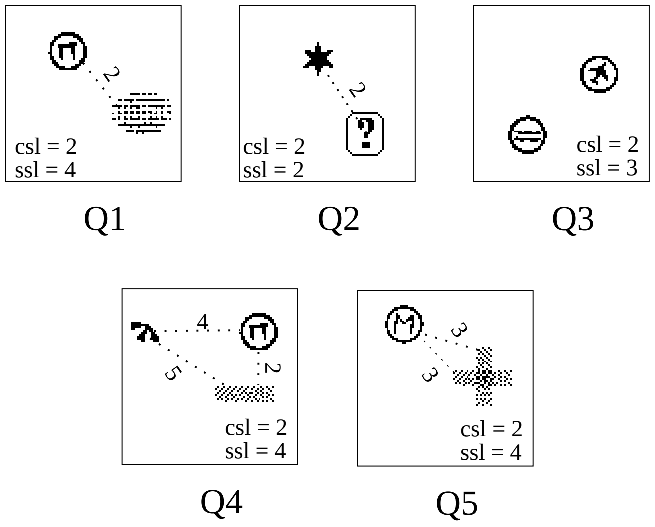


Figure 16: Pictorial queries demonstrating the use of different spatial similarity levels. "csl" denotes contextual similarity level, "ssl" denotes spatial similarity level. The question mark $\boxed{?}$ symbol denotes a wild card (i.e., any symbol matches it).

Figure 16 demonstrates the use of different spatial similarity levels for query specification. In all of these queries we assume $csl = 2$ (i.e., every symbol in the query image has a distinct matching symbol in the database image). Query Q1 requests all images that contain a picnic site within 2 miles of an open field . Query Q2 requests images with a site of interest ✱ and any symbol within 2 miles and southeast of the site of interest ✱ . Query Q3 requests all images that contain an airport northeast of a beach . Query Q4 requests images that contain a picnic site within 2 miles of a local road and within 4 miles of a scenic view , and the scenic view is within 5 miles of the local road . Query Q5 requests images that contain a museum within 3 miles of of two local roads that intersect. Observe that the condition that the two local roads intersect is specified by positioning the corresponding symbols in the query image so they intersect and setting $ssl = 4$ (i.e., the distance in the database image is bound from above by the distance in the query image). Since the distance between the two roads in the query image is 0, the distance in the database image must also be 0. Therefore, local roads in the database image must also intersect. Note that the dotted lines with the distance label that appear in the query images in Figure 16 are only used to denote the distance between symbols in the figure; they are not actually part of the query image. The query image only contains symbols. The distance (and relative directions) between the symbols is specified implicitly in the query image *QI* by the actual distance (and relative direction) between the symbols in *QI* provided that the spatial similarity level specifies that they are to be taken into account in computing the response to the query. Since the distances are inferred from the query image, it is currently up to the user to gauge the required distance in image space in order to specify a query such as find images with a hotel within 1 mile of a beach. We are currently working on incorporating scale into our tool and displaying the “real world” distances as the symbols are moved when the distance constraint is set.

5 Expressive Power: Compound Queries and Negation

So far we have described how to construct an individual query image and resolve the matching, contextual, and spatial ambiguity by specifying *msl*, *csl*, and *ssl*. In this section we show how we add expressive power to our graphic specifications via compound queries and negation. A pictorial specification may be composed of several query images. The query images can be joined with AND and OR operators. In addition, a query image can be negated with the NOT operator in order to specify negative conditions. The semantic meaning of these operators is given by taking the conjunction, disjunction, or negation of the conditions specified by each individual query image as follows:

OR: $[DI \equiv (QI_1 \text{ OR } QI_2)] \Longrightarrow (DI \equiv QI_1) \vee (DI \equiv QI_2)$

AND: $[DI \equiv (QI_1 \text{ AND } QI_2)] \Longrightarrow (DI \equiv QI_1) \wedge (DI \equiv QI_2)$

NOT: $[DI \equiv \text{NOT } (QI)] \Longrightarrow \neg(DI \equiv QI)$[2]

In the case of the conjunction of query images where the same symbol appears in both query images, the user may specify whether the two query-symbols must match the same instance of the symbol in the database image, or whether two different instances are allowed. As mentioned in Section 3, in order to bind two query-symbols to the same database symbol, the user selects the symbol for the second query image from the first query image, rather than selecting it from the menu of symbols. Let $bound(s_i, s_j)$ denote that symbols s_i and s_j have been bound to each other in this way, and let $s_i \doteq s_j$ denote that s_i is the symbol in query image *QI* found to match symbol s_j in database image *DI*. That is, when determining which symbols of *DI* are from the same class as the symbols of *QI*, s_i was determined to be equivalent to s_j. The semantic meaning of the AND operator is now augmented with the following condition: $\forall i, j, k, l (s_i \in QI_1 \wedge s_j \in QI_2 \wedge s_k \in DI \wedge s_l \in DI \wedge bound(s_i, s_j) \wedge s_i \doteq s_k \wedge s_j \doteq s_l) \Longrightarrow s_k = s_l$.

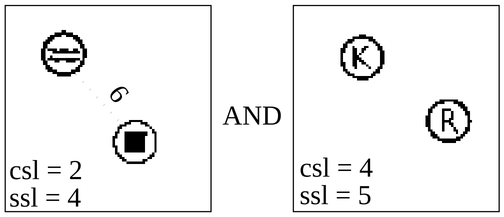


Figure 17: A pictorial query to “display all images with a hotel ▣ within 6 miles of a beach ⊜ and with a cafe Ⓚ or a restaurant Ⓡ ”.

Compound queries can be used to specify more complex queries. In particular, two separate query images with different values of *csl* and *ssl* can be combined via the AND operator to specify a query with spatial

[2]Although the use of NOT, in general, may lead to unsafe queries (i.e., queries that explode with infinitely many answers), this is not the case here since the set of all possible answers is finite as it corresponds to all of the images in the database

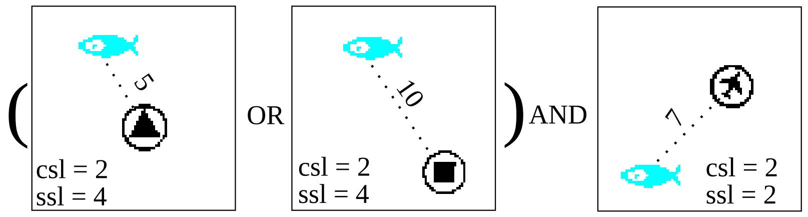


Figure 18: A pictorial query to "display all images with a camping site ▲ within 5 miles of a fishing site 🐟 OR with a hotel ▣ within 10 miles of a fishing site 🐟 AND with an airport ✈ northeast of and within 7 miles of the fishing site 🐟 ".

constraints between some symbols, but with no spatial constraints between other symbols. For example, consider the query in Figure 17 which requests "all images with a hotel ▣ within 6 miles of a beach ⊜ and with a cafeⓀ or a restaurant Ⓡ ". No spatial constraints are specified for the restaurant Ⓡ and cafeⓀ symbols; however, the hotel ▣ must be within 6 miles of a beach ⊜ . Notice that each query image component has a different *csl* value associated with it. Thus, the first component requests images containing both symbols, whereas the second component requests images containing either symbol. Compound queries can also be used to specify more than one acceptable spatial constraint. For example, the query "display all images with a camping site ▲ within 5 miles of a particular fishing site 🐟 OR with a hotel ▣ within 10 miles of the same fishing site 🐟 AND with an airport ✈ northeast of and within 7 miles of the same fishing site 🐟 " can be specified as shown in Figure 18. Recall that we use a color coding scheme to denote that two query-image symbols are bound to the same instance in the database image (i.e., the fishing site 🐟 in this example). That is, two symbols that have the same non-black color are bound, whereas black symbols are not bound.

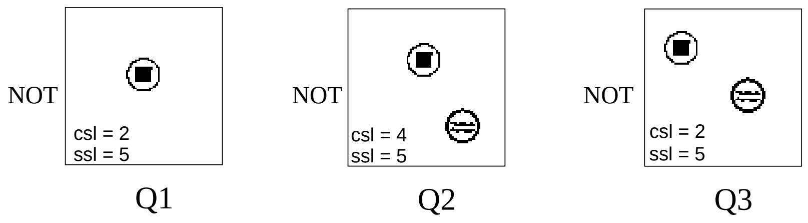


Figure 19: Pictorial queries using negation. Q1: "images with no hotel▣ "; Q2: "images that have neither a beach ⊜ nor a hotel ▣ "; Q3: "images that do not have a beach ⊜ or do not have a hotel ▣ ".

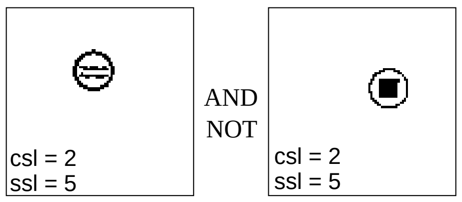


Figure 20: A pictorial query to "display all images with a beach ⊜ but with no hotel ▣ ".

Negation of queries can be used in order to specify conditions that should not be satisfied by the database images that are retrieved successfully. Figure 19 demonstrates how negation can be used to express retrieval of images that do not contain a particular symbol, a pair of symbols, or one of two symbols. Query Q1 requests "images with no hotel ▣ ", Query Q2 requests "images that have neither a beach ⊜ nor a hotel ▣ ", while query Q3 requests "images that do not have a beach ⊜ or do not have a hotel ▣ ". Negation in

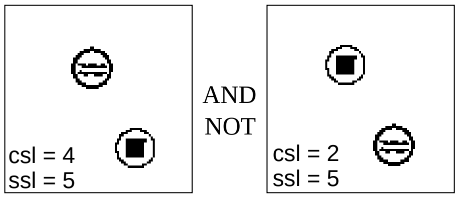


Figure 21: A pictorial query to “display all images with either a beach ⊜ or a hotel ▣ but not both”.

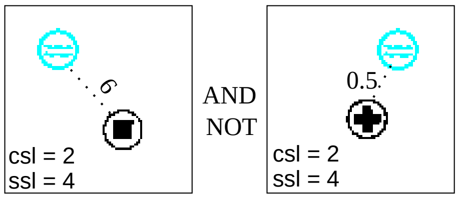


Figure 22: A pictorial query to “display all images with a hotel ▣ within 6 miles of a beach ⊜ and with no first aid station ⊕ within 0.5 mile of the beach ⊜ ”.

conjunction with compound queries can be used to specify both positive and negative conditions as is the case for the query in Figure 20 which requests “images that do have a beach ⊜ , but do not have a hotel ▣ ”. Using negation in conjunction with varying *csl* values makes it possible to specify an XOR condition as is the case for the query in Figure 21 which requests “images that have either a beach ⊜ , or a hotel ▣ , but not both”. Compound queries with symbol binding and negation can be used to specify more than one spatial condition for the same symbol as is the case for the query in Figure 22 which requests “all images with a hotel ▣ within 6 miles of a beach ⊜ and with no first aid station ⊕ within 0.5 mile of the beach ⊜ ”. Another application of compound queries with symbol binding and negation is to to specify distance constraints in terms of an upper bound. For example, the query in Figure 23 requests “all images with a camping site ▲ further than 1 mile from a beach ⊜ ”. The first component of the query requests images with a beach ⊜ and a camping site ▲ . The second component of the query requests images with a camping site ▲ within 1 mile of the beach ⊜ . The two components of the query are composed with the “AND NOT” operator, thus the entire query requests images that have a beach ⊜ and a camping site ▲ , but not within 1 mile of each other. Notice that the beach ⊜ and camping site ▲ symbols of the second component of the query image are bound to their counterparts in the first component of the query. This is necessary in order to ensure that the two query components will match the same symbol instance and thus every pair of beach ⊜ and camping site ▲ symbols must be further than 1 mile apart.

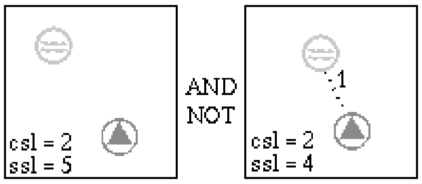


Figure 23: A pictorial query to “display all images with a camping site ▲ further than 1 mile from a beach ⊜ ”.

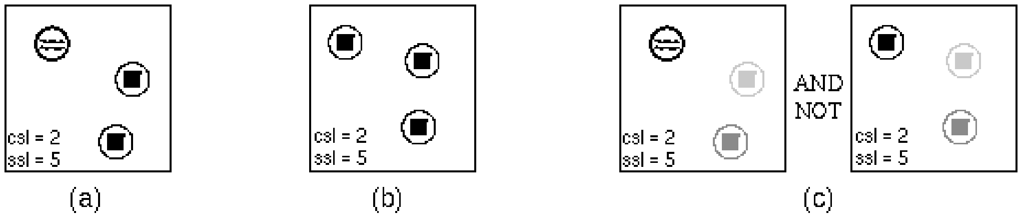


Figure 24: A graphical query to (a) " display all images that contain a beach ⊜ and at least two hotels ▣ ", (b) " display all images that contain at least three hotels ▣ ", (c) " display all images that contain a beach ⊜ and exactly two hotels ▣ "

In the example queries that we have seen so far, there is only one instance of each symbol in each query image. Our pictorial specification method does, however, allow multiple instances of each symbol. This is useful in order to specify the number of occurrences of a particular symbol that are required in a target database image. According to the definition of *csl* (see Figure 10), if *csl* is set to 1 or 2, then every symbol in *QI* has a distinct matching symbol in *DI*. Thus, if there are two instances of a symbol in *QI*, and *csl* is set to 1 or 2, then there must be at least two instances of this symbol in *DI* for it to satisfy the query. For example, the query in Figure 24a requests "images that contain a beach ⊜ and at least two hotels ▣ ". If we want to restrict ourselves to a beach ⊜ and two hotels ▣ , then we create a compound query that combines finding a beach ⊜ with at least two hotels ▣ with the negation of a query that finds all "images with at least three hotels ▣ " (Figure 24b). The resulting query is shown in Figure 24c and corresponds to the query that finds all "images that contain a beach ⊜ and exactly two hotels ▣ ".

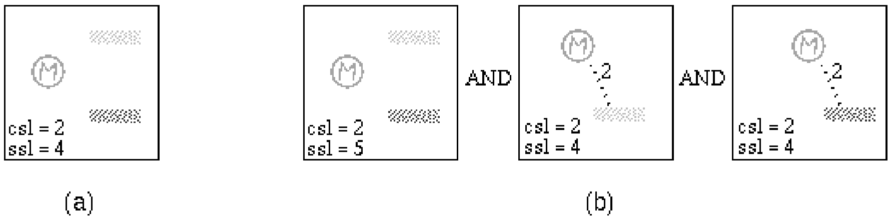


Figure 25: A pictorial query to "display all images with two different local roads ▨ within 2 miles of a museum Ⓜ ".

Another use of compound queries with multiple instances of symbols is one where we impose different constraints on the spatial relationship between the different symbols. In particular, in the current implementation, given a query image and a spatial constraint, all symbols in the image must satisfy it. For example, suppose that want to find all "images with two different local roads ▨ so that each of the roads is within 2 miles of a museum Ⓜ ". At a first glance, it would appear that we can specify this query by use of the graphical query given in Figure 25a. In this case, *csl* is set to 2 as we allow the databases images to contain symbols from other classes and *ssl* is set to 4 as we are restricting the distance between the two local roads ▨ and the museum Ⓜ . Unfortunately, the graphical query given in Figure 25a will not satisfy our desired query. The problem is that the graphical query in Figure 25a places a distance constraint on the distance between the two local roads ▨ as well whereas there is no such constraint in our query. In essence, we have encountered one of the shortcomings of our query language in the sense that the spatial constraints must either hold for all of the symbols in the query image or for none of them. What we want is a partial specification of the spatial constraints.

The above shortcoming can be overcome by decomposing the query into three components as shown in Figure 25b and using object binding to ensure that the components use the same instances of the various symbols. In this case, the first component specifies the contextual similarity condition that the image contain a museum Ⓜ and at least two local roads ▨ (as well as possibly other symbols) with no spatial similarity constraints (i.e., *csl* is set to 2 and *ssl* is set to 5). The remaining two components correspond to the two spatial conditions. In particular, one component specifies that the distance between the museum Ⓜ and one of the local roads ▨ , while the second component specifies the distance between the museum Ⓜ and the other local road ▨ . Notice the use of different colors (dark gray and light gray) for the local roads ▨ in the first component and gray for the museum Ⓜ . The museums Ⓜ in both the second and third components are shown in gray indicating that they are bound to the same instance of the museum Ⓜ in the first component. The local roads ▨ in the second and third components are shown in light gray and dark gray respectively, indicating that they are bound to the corresponding different instances in the first component.

6 Pictorial Query Processing

In this section we describe how pictorial queries can be processed efficiently in an image database. In particular, we present an algorithm for retrieving all database images that conform to a given pictorial query specification. In order to execute the algorithm efficiently, the image database must have indexes that enable the following operations: (i) retrieve all images that contain symbols of a given class; (ii) retrieve all symbols in a given image; (iii) retrieve all symbols within a given distance or direction from a given point. In our database, the first two indices are realized with a B-tree. The ability to retrieve all symbols within a given distance or direction from a given point is achieved by use of an index on the locations of the set of all symbols in all of the images. This index is implemented using a PMR quadtree for points [16].

The first step in finding all database images that conform to a pictorial query specification is to process each component query image *QI* that is part of the pictorial query specification individually. This is done by a function called *GetSimilarImages* that takes as input a query image (*QI*), the matching similarity level (*msl*), the contextual similarity level (*csl*), and the spatial similarity level (*ssl*) associated with *QI*. It returns the set of database images RI such that each image $DI \in RI$ satisfies the pictorial query (i.e., $DI \equiv_{msl,csl,ssl} QI$ for all *DI* $\in$ *RI*). Figure 26 summarizes this algorithm. The algorithm assumes only one instance of each class in the query image as well as in the database image. We briefly discuss how to deal with other cases at the end of this section. For the purpose of simplicity, we assume that in all queries that involve spatial distance constraints, L, the lower bound for the distance allowed between symbols in the database image, is 0. That is, $0 \leq dist(s_i, s_j) \leq dist(s_k, s_l)$, where s_i and s_j are database image symbols and s_k and s_l are query image symbols, respectively.

In the following algorithms, LI is the logical image representation of the query image *QI*. The logical image representation LI of an image I, is a list of elements for each symbol $s \in I$. Each element is of the form: $\{(C, certainty), loc\}$ where C is the classification of s, *loc* is the location of s in I, and $0 < certainty \leq 1$ indicates the certainty that $s \in C$. The classification, C, of a specific element $el \in LI$ is denoted by $C(el)$. The location of a specific element $el \in LI$ is denoted by $loc(el)$. The matching, contextual, and spatial similarity levels are denoted by *msl*, *csl*, and *ssl* respectively. $|I|$ denotes the number of elements in the logical image I (i.e., its cardinality). *GetSimilarImages* constructs a set of candidate images from the database in which the symbols match those of the query image with a certainty $\geq msl$ and the contextual constraints hold and then invokes function *CheckSsl* for each candidate image to determine if the spatial constraints dictated by *ssl* hold in it. An image in which the spatial constraints do not hold is removed from the candidate-image set. Compound queries are resolved by combining the result image sets $\{RI_i\}$ of each query component QI_i

according to the operators that join the query components. Letting $\{RI_1\}$ and $\{RI_2\}$ denote the results of query components QI_1 and QI_2, respectively, and letting $\{A\}$ denote the set of all images in the database, we have:

QI$_1$ **AND** *QI*$_2$ $= \{RI_1\} \cap \{RI_2\}$

QI$_1$ **OR** *QI*$_2$ $= \{RI_1\} \cup \{RI_2\}$

NOT *QI*$_1$ $= \{A\} - \{RI_1\}$

Finally, we need to check for the bound symbol condition. That is, $\forall i,j,k,l(s_i \in QI_1 \wedge s_j \in QI_2 \wedge s_k \in DI \wedge s_l \in DI \wedge bound(s_i,s_j) \wedge s_i \doteq s_k \wedge s_j \doteq s_l) \Longrightarrow s_k = s_l$. This is done by comparing the logical image representation of the result images when computing set intersections, and only including images in which the same database-image symbol was matched to the two bound query-image symbols.

```
GetSimilarImages(logical image QI, similarity level $msl$,$csl$,$ssl$)

$n \leftarrow 0$
/* check matching similarity */
foreach $el \in QI$
  $r_n \leftarrow$ set of all images containing $C(el)$ with certainty $\geq$ msl
      (use index on class)
  $n \leftarrow n+1$
/* check contextual similarity */
if $(csl=1) \vee (csl=2)$ then
  $RI \leftarrow \bigcap_{i=0}^{n-1} r_i$
elseif $(csl=3) \vee (csl=4)$
  $RI \leftarrow \bigcup_{i=0}^{n-1} r_i$
if $(csl=1) \vee (csl=3)$ then
  $RI \leftarrow RI - \{I$ s.t. set of all elements of $I$ (use index on image_id)
         includes symbols not from classes in QI$\}$
/* check spatial similarity */
$RI \leftarrow RI - \{I$ s.t. spatial constraints dictated by $ssl$ do not hold
      (call CheckSsl) $\}$
/* order by closeness of matching */
return $RI$ ordered by average certainties
```

Figure 26: Algorithm *GetSimilarImages* to get all database images that are similar to a given query image (*QI*) following the constraints dictated by the matching (*msl*), contextual (*csl*), and spatial (*ssl*) similarity levels. Assumes one instance of each class in *QI* and *DI*.

Algorithm *CheckSsl* determines whether the spatial constraints dictated by a query image *QI* and spatial similarity level *ssl* hold in an image *DI*. Figure 27 summarizes this algorithm. The input to *CheckSsl* is *QI'* and *DI'* which are sub-images of the original *QI* and *DI* that contain only those symbols that were matched to each other by *GetSimilarImages* when checking the contextual constraints. Thus, the set of classes of the symbols of *DI'* and *QI'* is identical.

```
CheckSsl(logical image DI, QI, similarity level $ssl$)

  if $ssl = 5 \vee |DI| = 1$ then /* no need to check anything */
    return TRUE
  /* compute distances and relative location between QI symbols*/
  foreach $qel_1 \in QI$
    foreach $qel_2 \in QI - \{qel_1\}$
      if $(ssl = 2) \vee (ssl = 4)$ then
        $dists[cl(qel_1), cl(qel_2)] \leftarrow getDist(loc(qel_1), loc(qel_2))$
      if $(ssl = 2) \vee (ssl = 3)$ then
        $relDirs[cl(qel_1), cl(qel_2)] \leftarrow getReldir(loc(qel_1), loc(qel_2))$
  /* now check that these hold in the input image */
  foreach $del_1 \in DI$
    foreach $del_2 \in DI - \{del_1\}$
      if $(ssl = 2) \vee (ssl = 4)$ then
        if $getDists(loc(del_1), loc(del_2)) > dists[cl(del_1), cl(del_2)]$ then
          return FALSE
      if $(ssl = 2) \vee (ssl = 3)$ then
        if $getReldirs(loc(del_1), loc(del_2)) \neq relDirs[cl(del_1), cl(del_2)]$ then
          return FALSE
  return TRUE /* everything is OK */
```

Figure 27: Algorithm *CheckSsl* to determine whether the spatial constraints dictated by a query image *QI* and spatial similarity level *ssl* hold in a logical image *DI*. *CheckSsl* assumes one instance of each class in *QI* and *DI* and that L, the lower bound on distance between symbols, is $= 0$.

Figure 28 is an example of the use of *CheckSsl*. Note that the dotted lines in *QI* represent the maximal distance between the symbols as specified implicitly by *QI*, while the solid lines in *DI* represent the actual distance between the symbols in a particular database image. In this case, query *QI'* requests images with a beach ⊜, hotel ▣, picnic site ⊓, and site of interest ✱ (the image may contain one or all of them as well as other symbols since $csl = 4$). In addition, the distances between the subset of these symbols that appear in the database image must be less than or equal to those specified in *QI* as indicated by the labels on the dotted

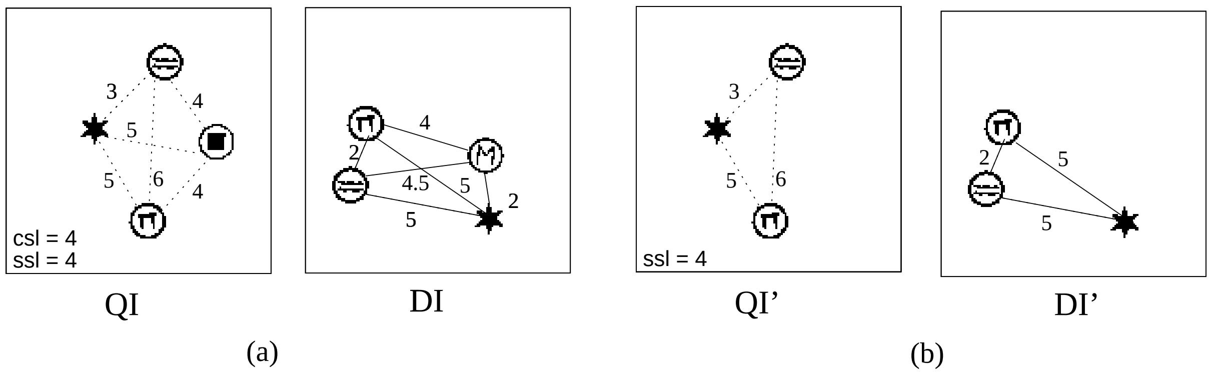


Figure 28: (a) Query image *QI* and database image *DI*. (b) The corresponding sub-images *QI'* and *DI'* that contain only the matching symbols.

lines (e.g., the beach ⊜ must be within 3 of the site of interest ✱, within 4 of the hotel ◙, and within 6 of the picnic site ⓜ). There is a match in *DI* only for query-image symbols beach ⊜, picnic site ⓜ, and site of interest ✱. Thus, *QI'* and *DI'* only contain these symbols, and *CheckSsl* will check the spatial constraints among them. The algorithm first computes the distance and/or the relative directions between the symbols of *QI'*. It then computes them for *DI'* and checks whether the required spatial constraints between each symbol pair in *DI'* hold. For the example in Figure 28, *CheckSsl* would return FALSE since the constraint that the distance between the site of interest ✱ and the beach ⊜ is $\leq$ 3 imposed by *QI* does not hold in *DI* (where the distance between these two symbols is 5), and since according to the definition of $\equiv_{msl,csl,ssl}$ in Section 4 the spatial constraints must hold simultaneously between all of the symbols that appear in both query and database images. In order to specify that just some of the spatial constraints hold, we would need to split the query into a disjunction of several queries that specify the permissible combinations.

In the algorithms, as described in Figures 26 and 27, we only allowed one instance of each class in both the query image and the database image. If we allow more than one instance of each class in the database image, then *CheckSsl* as we have presented it is incorrect. We have devised a method that allows more than one instance of each class in the database image (while still requiring only one instance from each class in the query image). This method makes use of an auxiliary graph data structure. The running time of this algorithm is exponential in the number of symbols in the query image. It is not influenced by the number of symbols in the database image. We do not describe this method here; see [28] for the details.

The algorithm presented above is a relatively naive solution to pictorial query processing. This algorithm deals with each constraint imposed by the pictorial query individually. Namely, it first performs symbol matching and then it resolves the contextual constraints. Next it resolves the spatial constraints. Finally, it applies the operators and checks for symbol binding. The formal analysis of the complexity of this approach is beyond the scope of this paper. There are clearly more efficient ways to proceed in pictorial query processing. In [29], four additional algorithms for function *GetSimilarImages* are discussed. These algorithms handle the contextual and spatial constraints simultaneously in order to achieve better pruning of the search space in the early stages of query processing. Other optimization techniques that may be applied to improve the efficiency of processing pictorial queries involve changing the order of processing of the individual query images in order to execute the parts that are more selective first, and combining individual query images and processing them together. These and other query optimization issues are the subject of future research.

7 Concluding Remarks

A pictorial query specification technique that enables the formulation of complex pictorial queries for image databases has been described. Using this technique, it is possible to specify which objects should appear in the target images as well as how many occurrences of each object are required. Moreover, spatial constraints can be imposed that specify bounds on the distance between objects, as well as the relative direction or orientation between objects. As part of the pictorial specification, the user indicates the degree of desired similarity, and thus the results of the image retrieval are not subjective. Expressive power is achieved by combining several query images into a compound pictorial query specification and by providing the capability of object binding in order to specify whether the same instance of an object is to be used in the case of a conjunction of two query images. An algorithm for processing such pictorial queries has been outlined. The efficiency of this algorithm can be improved by employing some of the suggested query optimization techniques. We have used this pictorial query specification method to build a query interface for a map image database system.

While it is possible to express rather complex queries using our method, there are some conditions that cannot be specified. In particular, we cannot specify conditions involving the location of certain events between objects. For example, in Figure 16, we showed how to specify the condition “museum Ⓜ within 3 miles of two local roads ▨ that intersect”. However, we cannot specify that we want the museum Ⓜ to be within 3 miles of the point where these two local roads ▨ intersect. In addition, although we take the extent of objects into account in distance and relative position computations, we do not consider the size or direction of the object itself. For example, we cannot specify “an open field ▦ whose area is at least 1 square mile” or “a local road ▨ that goes from north to south”. Finally, we cannot qualify objects in terms of non-spatial conditions. For example, we would like to specify “hotels whose price is less than $80 per night”. Incorporating these features into our pictorial query specification method is a subject for future research.

8 Acknowledgments

We are grateful to Karttakeskus, Map Center, Helsinki, Finland for providing us the map data. We have benefited from discussions with Walid G. Aref.

References

[1] A. Del Bimbo and P. Pala. Image indexing using shape-based visual features. In *Proceedings of the 13th International Conference on Pattern Recognition*, volume III, pages 351–355, Vienna, Austria, August 1996.

[2] A. Del Bimbo, E. Vicario, and D. Zingoni. A spatial logic for symbolic description of image contents. *Journal of Visual Languages and Computing*, 5(3):267–286, September 1994.

[3] 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.

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

[5] I. F. Cruz. DOODLE: A visual language for object-oriented databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 71–80, San Diego, CA, June 1992.

[6] M. J. Egenhofer. A formal definition of binary topological relationships. In W. Litwin and H. J. Schek, editors, *Proceedings of the Third International Conference on Foundations of Data Organization and Algorithms*, pages 457–472, Paris, June 1989. (Lecture Notes in Computer Science 367, Springer-Verlag, Berlin, 1989).

[7] M. J. Egenhofer. Query processing in spatial-query-by-sketch. *Journal of Visual Languages and Computing*, 8(4):403–424, August 1997.

[8] W. I. Grosky, P. Neo, and R. Mehrotra. A pictorial index mechanism for model-based matching. *Data & Knowledge Engineering*, 8(4):309–327, September 1992.

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

[10] C. E. Jacobs, A. Finkelstein, and D. H. Salesin. Fast multiresolution image querying. In *Proceedings of the SIGGRAPH'95 Conference*, pages 277–286, Los Angeles, CA, August 1995.

[11] T. Joseph and A.F. Cardenas. PICQUERY: A high level query language for pictorial database management. *IEEE Transactions on Software Engineering*, 14(5):630–638, May 1988.

[12] T. Kato, T. Kurita, N. Otsu, and K. Hirata. A sketch retrieval method for full color image databases – query by visual example. In *Proceedings of the 11th International Conference in Pattern Recognition*, pages 530–533, The Hague, The Netherlands, August 1992.

[13] R. Krishnamurthy and M. Zloof. RBE: Rendering by example. In *Eleventh International Conference on Data Engineering*, pages 288–297, Taipei, Taiwan, March 1995.

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

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

[16] 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).

[17] 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.

[18] G. Özsoyoğlu, V. M. Matos, and Z. M. Özsoyoğlu. Query processing techniques in the summary-table-by-example database query language. *ACM Transactions on Database Systems*, 14(4):526–573, December 1989.

[19] D. Papadias and T. K. Sellis. A pictorial query-by-example language. *Journal of Visual Languages and Computing*, 6(1):53–72, March 1995.

[20] 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.

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

[22] H. Samet and A. Soffer. MARCO: MAp Retrieval by COntent. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):783–798, August 1996.

[23] B. Shneiderman. *Designing the User Interface*. Addison-Wesley, Reading, MA, second edition, 1992.

[24] A. P. Sistla, C. Yu, and R. Haddad. Reasoning about spatial relationships in picture retrieval systems. In J. Bocca, M. Jarke, and C. Zaniolo, editors, *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 570–581, Santiago, Chile, September 1994.

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

[26] A. W. M. Smeulders and R. Jain, editors. *Proceedings of the First International Workshop on Image Databases and Multi Media Search*, Amsterdam, The Netherlands, August 1996.

[27] G. H. Sockut, L. M. Burns, A. Malhotra, and K. Y. Whang. GRAQULA: A graphical query language for entity-relationship or relational databases. *Data and Knowledge Engineering*, 11(2):171–202, October 1993.

[28] A. Soffer and H. Samet. Handling multiple instances of symbols in pictorial queries by image similarity. In *Proceedings of the First International Workshop on Image Databases and Multi Media Search*, pages 51–58, Amsterdam, The Netherlands, August 1996.

[29] A. Soffer and H. Samet. Pictorial queries by image similarity. In *Proceedings of the 13th International Conference on Pattern Recognition*, volume III, pages 114–119, Vienna, Austria, August 1996.

[30] A. Soffer and H. Samet. Negative shape features for image databases consisting of geographic symbols. In *3rd International Workshop on Visual Form*, Capri, Italy, May 1997.

[31] A. Soffer and H. Samet. Pictorial query specification for browsing through image databasess. In *Proceedings of the Second International Conference on Visual Information Systems*, pages 117–124, San Diego, California, Dec. 1997.

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

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

[34] J. D. Ullman. *Database and Knowledge-Base Systems*, volume I. Computer Science Press, Rockville, MD, 1988.