

Query Processing and Optimization for Pictorial Query Trees

Aya Soffer¹ * and Hanan Samet² **

¹ Computer Science Department, Technion City, Haifa 32000, Israel and
Institute for Advanced Computer Science University of Maryland at College Park
E-mail: ayas@cs.technion.ac.il

² Computer Science Department, Institute for Advanced Computer Science
University of Maryland at College Park, College Park, Maryland 20742
E-mail: hjs@umiacs.umd.edu

Abstract. Methods for processing of pictorial queries specified by *pictorial query trees* are presented. Leaves of a pictorial query tree correspond to individual pictorial queries while internal nodes correspond to logical operations on them. Algorithms for processing individual pictorial queries and for parsing and computing the overall result of a pictorial query tree are presented. Issues involved in optimizing query processing of pictorial query trees are outlined and some initial solutions are suggested.

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

1 Introduction

Image databases must be capable of being queried pictorially. The most common method of doing this is via an example image. The problem with this method is that in an image database we are usually not looking for an exact match. Instead, we want to find images similar to a given query image. One of the main issues is how to determine if two images are similar and whether the similarity criteria that are used by the database system match the user's notion of similarity. Another difficulty with pictorial queries is that they are usually not very expressive in terms of specifying combinations of conditions and negative conditions. A good pictorial query specification method should leverage on the expressiveness of pictorial queries in terms of describing what objects the target images should contain and their desired spatial configuration, resolve the ambiguities inherent to pictorial queries, and enable specifying combinations of conditions.

In our previous work [8], we devised a pictorial query specification technique for formulating queries that specify which objects should appear in a target image

* The support of the Lady Davis Foundation and 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.

as well as how many occurrences of each object are required. Moreover, the minimum matching certainty between query-image and database-image objects can be specified, and spatial constraints that specify bounds on the distance between objects and the relative direction between objects can be imposed. Expressive power is achieved 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 [9] we extended the pictorial query specification technique to allow the formulation of complex pictorial queries via *pictorial query trees* where leaves correspond to individual pictorial queries while internal nodes represent logical operations on the set of pictorial queries (or subtrees) represented by its children.

In this paper we describe in detail the algorithm for processing individual pictorial queries. It can handle multiple instances of each symbol in the query image as well as in the database image and can also handle wild cards. In addition, we present an algorithm for parsing and computing the overall result of a pictorial query tree. We also discuss some issues that are involved in query optimization of pictorial query trees (i.e., their simplification).

2 Related Work

Most image database research deals either with global image matching based on color and texture features [4, 5, 10] or with the ambiguity associated with matching one query-image object to another [3]. These methods do not address the case of images that are composed of several objects and their desired spatial configuration. There has been some work on the specification of topological and directional relations among query objects [1, 2]. These studies only deal with tagged images. Furthermore, it is always assumed that the goal is to match as many query-image objects to database-image objects as possible.

A limited form of spatial ambiguity is allowed in pictorial queries based on the *2D-string* and its variants [2]. The spatial logic described in [1] also allows specification of query images in terms of spatial relations between objects and permits users to select the level of spatial similarity. In some cases (e.g., [6]), the images are segmented into regions either automatically or semi-automatically and some queries involving spatial constraints specifying the desired arrangement of these regions can be performed. However, the issue of the distance between objects is not addressed by these or any other method. Furthermore, none of these methods provide Boolean combinations or negations of query images.

3 Building Pictorial Query Trees

An individual pictorial query is specified by selecting the required query objects and positioning them according to the desired spatial configuration. Next, the similarity level in terms of three parameters is specified. 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 and thus match. *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 use

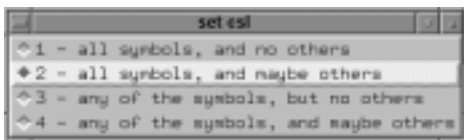


Fig. 1. Contextual similarity levels (csl).

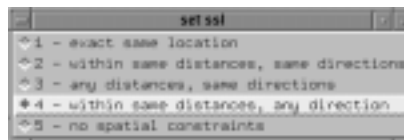


Fig. 2. Spatial similarity levels (ssl).

four levels of contextual similarity (see Figure 1). *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. We use five levels of spatial similarity (see Figure 2). For more details and examples, see [8].

Complex pictorial queries are composed of combinations of individual pictorial queries and are specified via *pictorial query trees*. Leaves of a pictorial query tree correspond to individual pictorial queries. A negated leaf node (NOT) yields the set of all images that do not satisfy the pictorial query. Internal nodes in the tree represent logical operations (AND, OR, XOR) and their negations on the set of images that satisfy the pictorial query (or query subtree) represented by its children. The root of the tree is either a pictorial query or a logical operator, while an internal node corresponds to a logical operator and can have one or more children. For a 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. This is termed *object binding*.

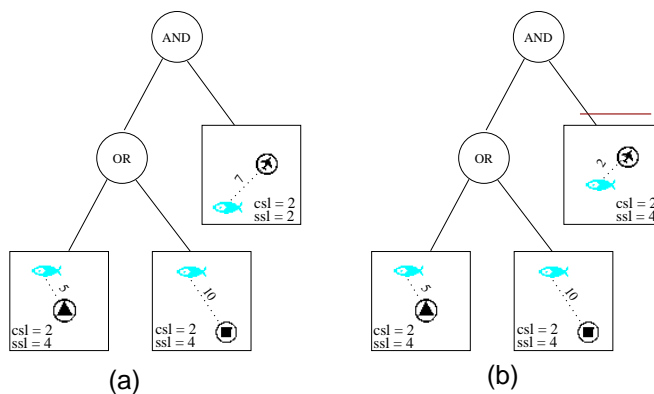


Fig. 3. Images with a (a) camping site within 5 miles of a fishing site OR a hotel within 10 miles of a fishing site AND an airport northeast of and within 7 miles of the fishing site . (b) camping site within 5 miles of a fishing site OR with a hotel within 10 miles of a fishing site AND with no airport within 2 miles of the fishing site (the line above the query denotes negation).

Figures 3 and 4 are examples of pictorial query trees. Figure 3a specifies

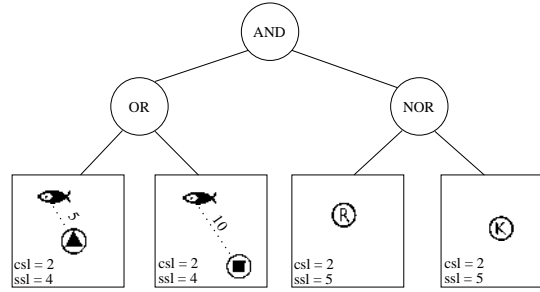





















Fig. 4. Images with a camping site  within 5 miles OR with a hotel  within 10 miles of a fishing site  AND with neither a restaurant  nor a cafe .

more than one acceptable spatial constraint (i.e., a camping site  within 5 of a fishing site  or a hotel  within 10 of a fishing site ) with an OR. We also specify that there is an airport  northeast of and within 7 miles of the fishing site  with an AND. Figure 3b shows negation of a pictorial query specifying that there is no airport  within 2 miles of the fishing site . Figure 4 shows negation of logical operators in internal nodes. Here we are seeking images with a camping site  within 5 miles of a fishing site  OR a hotel  within 10 miles of a fishing site  but without either a restaurant  or a cafe .

4 Pictorial Query Processing

In this section, we present an algorithm for retrieving all database images that conform to a given pictorial query tree specification.

4.1 Processing individual pictorial queries

The first step in the algorithm processes each pictorial query image (i.e., each leaf) individually using function *GetSimilarImagesM* that takes as input a query image (QI), the matching similarity level ($mssl$), 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. Figure 5 summarizes this algorithm. *GetSimilarImagesM* handles wildcards as well as multiple instances of each class in the query image and in the database image.

First, for each symbol in the query image it finds all database images, DI , that contain this symbol with certainty $\geq mssl$. Next, it handles the contextual constraints. If csl is 1 or 2 (images should contain all symbols in QI), then it intersects the set of result images from the first step. If csl is 3 or 4 (any one symbol from QI is enough), then it takes the union of the result images. If the contextual similarity level is 1 or 3, then it avoids including images containing symbols that are not present in QI . Next, it checks the case of multiple instances of query symbols in the query image. If csl is 1 or 2, then for every instance of each symbol in QI , it checks whether there exists an instance of the symbol in DI .

```

GetSimilarImagesM(logical image  $QI$ ,  $mssl$ ,  $csl$ ,  $ssl$ )
 $m \leftarrow 0$ 
/* check matching similarity */
foreach  $el \in QI$ 
  if ( $el = wildcard$ ) then
     $r_m \leftarrow$  set of all images stored in the database
  else
     $r_m \leftarrow$  set of all images containing  $C(el)$  with certainty  $\geq mssl$ 
      (use index on class)
   $m \leftarrow m + 1$ 
/* 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
   $RI \leftarrow RI - \{I \text{ s.t. } I \text{ includes symbols not in } QI\}$ 
  (use index on image_id)
/* check multiple symbol instance conditions */
if ( $csl = 1$ )  $\vee$  ( $csl = 2$ ) then
   $RI \leftarrow RI - \{I \text{ s.t. } \exists k : n_k^{QI} > n_k^I\}$ ,
  i.e. some ( $k$ -th) symbol of  $QI$  is underrepresented in  $I$ 
/* check spatial similarity */
foreach  $I \in RI$ 
  for every possible matching of symbols between  $QI$  and  $I$ 
    check feasibility of this matching w.r.t. spatial constraints
  if all matchings are infeasible
     $RI \leftarrow RI - I$ 
return  $RI$  ordered by average certainties

```

Fig. 5. Algorithm to retrieve all database images similar to a query image (QI) conforming to constraints dictated by $mssl$, csl , and ssl . n_k^I denotes the number of occurrences of the k^{th} symbol in image I

Finally, it checks whether the spatial constraints are satisfied for each candidate image I in the candidate image list RI . Since multiple instances of symbols are allowed in QI and in I , this step needs to check many possible matchings. It can be that some mappings between QI symbols and I symbols create feasible configurations while others do not. For each QI symbol create a set of possible matches in I . Selecting one element from each of these sets generates one possible matching. If none of the possible matchings pass the spatial constraints test, then remove the image from the candidate result set. The spatial similarity between any two matchings is calculated using algorithm *CheckSsl* [8] which determines whether the spatial constraints dictated by a query image QI and spatial similarity level ssl hold in a logical image DI . Images that pass all of the

tests are ordered by the average matching certainty of all matching symbols and returned as the result of the query.

4.2 Parsing and evaluating pictorial query trees

```

                                ProcessQueryTree(query tree node: N)
S ← set of all images in the database (global variable)
if (isLeaf(N))
  NR ← GetSimilarImagesM(QI(N), msl(N), csl(N), ssl(N))
  if (hasNegationFlag(N))
    NR ← S - NR
else
  n ← 0
  foreach M ∈ sons(N)
    rn ← ProcessQueryTree(M)
    n ← n + 1
  NR ← OP(N)i=0n-1ri (OP(N) can be  $\cup$ ,  $\cap$ , or  $\oplus$ , possibly inverted)
return NR

```

Fig. 6. Algorithm to retrieve all images satisfying the query represented by node N of a pictorial query tree.

Procedure *ProcessQueryTree* parses and evaluates the result of a pictorial query tree. Figure 6 summarizes the algorithm. *ProcessQueryTree* takes as input a node N in the query tree, and returns the set of images that satisfy the query tree rooted at N . If N is a leaf node, then it checks whether the results of this query are cached from earlier invocations. If they are not, then algorithm *GetSimilarImagesM* is invoked. If the leaf node is negated in the tree, then the complement of the result images set returned by *GetSimilarImagesM* is taken. The final result image set is returned. If N is an internal node in the query tree, then *ProcessQueryTree* is called recursively on each child of N , followed by applying the appropriate logical operation on the results of these calls. The whole query tree is evaluated in this recursive manner by invoking algorithm *ProcessQueryTree* with the root of the query tree as an argument.

Recall, that users can specify object binding. That is, whether the same instance of an object is to be used when it appears in more than one of the pictorial query images that make up the pictorial query tree. The following is an outline of the additions to our algorithms that are necessary for handling object binding. Algorithm *ProcessQueryTree* receives as additional input a global set of constraints that stipulates the bindings that were specified as part of the query. This set consists of groups of symbols, where all of the symbols in the same group should be matched to the same symbol instance in the database







image. To filter out database images that are incompatible with respect to the binding conditions, we combine these binding constraints with information that is provided by the algorithm *GetSimilarImagesM*, which is augmented to return for each database image that was found similar to the query image, the mapping between query symbols and matched database symbols.

5 Query Optimization Issues

Several optimization techniques can be applied to improve the efficiency of processing pictorial query trees. These include methods designed for optimization of individual pictorial query processing and optimization of query tree processing. Individual pictorial query processing may be made more efficient by handling spatial and contextual constraints simultaneously rather than one followed by the other as we do now. We addressed this issue in [7].

Two optimizations are possible for computing the result of the pictorial query tree. The first optimization is to change the order of processing individual query images in order to execute the parts that are more selective (i.e., result in fewer images) first. The selectivity of a pictorial query is based on three factors. *Matching selectivity* estimates how many images satisfy the matching constraint as specified by *mssl*. *Contextual selectivity* estimates how many images satisfy the contextual constraint as specified by the query image and *csl*. *Spatial selectivity* estimates how many images satisfy the spatial constraint as specified by *ssl*. Depending on *ssl*, either distance, direction, both, or neither are constrained. Matching and contextual selectivity factors are computed based on statistics stored as histograms in the database which indicate the distribution of classifications and certainty levels in the images. These histograms are constructed when populating the database. Computing spatial selectivity is much more complex. One approach to measuring the distance aspect of the spatial selectivity calculates some approximation of the area spanned by the symbols in the query image. This can be estimated, for example, using an approximation of the convex hull of the symbols in the query image. Details of this method are beyond the scope of this paper. Selectivity of an individual pictorial query (leaf) is computed by combining these three selectivity factors.

The query tree selectivity is computed using a recursive algorithm similar to the one executing the query. If an individual pictorial query is negated in the tree, the selectivity is 1 - the selectivity of the query. The selectivity of a subtree is as follows. For OR or XOR, take the sum of the selectivities of the subtrees minus the probability that a combination of cases occurred. For AND, take the product of the selectivities of the subtrees.

To illustrate the general use of this optimization method, consider the query trees in Figure 3. In both queries the left side of the tree requests images with a camping site  within 5 miles of a fishing site  OR a hotel  within 10 miles of a fishing site . In query (a), we add the constraint that there exists an airport  northeast of and within 7 of fishing site . In our database, we have very few airfields and thus the right side is more selective and it will be processed first. On the other hand in query (b), we add the constraint that

there is no airport \textcircled{A} within 2 miles of the fishing site \textcircled{F} . Clearly, in most cases there will be no such airport \textcircled{A} , and thus in this case the right side is not selective and the left side should be processed first.

The second form of optimization is to combine individual query images and to process them together. To see its usefulness, we study how the query in Figure 4 is processed using the current algorithm. First, find $\{CF\}$ all images with a camping site \textcircled{C} within 5 of a fishing site \textcircled{F} . Next, find $\{HF\}$ all images with a hotel \textcircled{H} within 10 of a fishing site \textcircled{F} . Then, take the union of these two sets: $\{LS\} = \{CF\} \cup \{HF\}$. Now, find the set $\{R\}$: images with a restaurant \textcircled{R} and the set $\{C\}$: images with a cafe \textcircled{C} and compute the set $RS = I - (R \cup C)$. The final result is the intersection of the two sets: $LS \cap RS$. A more sensible way to compute this query is as follows. For each fishing site \textcircled{F} , find the nearest neighbors up to distance 5 in incremental order. If the next nearest neighbor is a camping site \textcircled{C} or a hotel \textcircled{H} , then add this image to the candidate list. Continue retrieving nearest neighbors in incremental order up to distance 10. If the next nearest neighbor is a hotel \textcircled{H} , then add this image to the candidate list. For each image I in the candidate list, examine all of the objects in I . If there is a restaurant \textcircled{R} or a cafe \textcircled{C} in I , then remove I from the candidate list.

References

1. A. Del Bimbo, E. Vicario, and D. Zingoni. A spatial logic for symbolic description of image contents. *Jour. of Vis. Lang. and Comp.*, 5(3):267–286, Sept. 1994.
2. S. K. Chang, Q. Y. Shi, and C. Y. Yan. Iconic indexing by 2-D strings. *IEEE Trans. on Patt. Anal. and Mach. Intel.*, 9(3):413–428, May 1987.
3. W. I. Grosky, P. Neo, and R. Mehrotra. A pictorial index mechanism for model-based matching. *Data & Know. Engin.*, 8(4):309–327, Sept. 1992.
4. 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 *Proc. of the SPIE, Storage and Retrieval of Image and Video Databases*, vol. 1908, pp. 173–187, San Jose, CA, Feb. 1993.
5. A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Content-based manipulation of image databases. In *Proc. of the SPIE, Storage and Retrieval of Image and Video Databases II*, vol. 2185, pp. 34–47, San Jose, CA, Feb. 1994.
6. J. R. Smith and S.-F. Chang. VisualSEEK: a fully automated content-based image query system. In *ACM Int. Conf. on Multimedia*, pp. 87–98, Boston, Nov. 1996.
7. A. Soffer and H. Samet. Pictorial queries by image similarity. In *13th Int. Conf. on Patt. Recog.*, vol. III, pp. 114–119, Vienna, Austria, Aug. 1996.
8. A. Soffer and H. Samet. Pictorial query specification for browsing through spatially-referenced image databases. *Jour. of Vis. Lang. and Comp.*, 9(6):567–596, Dec. 1998.
9. A. Soffer, H. Samet, and D. Zotkin. Pictorial query trees for query specification in image databases. In *14th Int. Conf. on Patt. Recog.*, vol. I, pp. 919–921, Brisbane, Australia, Aug 1998.
10. M. Swain. Interactive indexing into image databases. In *Proc. of the SPIE, Storage and Retrieval for Image and Video Databases*, vol. 1908, pp. 95–103, San Jose, CA, Feb. 1993.

This article was processed using the L^AT_EX macro package with LLNCS style