Image Similarity Retrieval by Spatial Constraints

Dimitris Papadias, Nikos Mamoulis and Dimitris Meretakis
Department of Computer Science
Hong Kong University of Science and Technology
Clearwater Bay, Hong Kong
+852-23586971
{dimitris, mamoulis, meretaks}@cs.ust.hk

1. ABSTRACT

This paper deals with queries involving the retrieval of images that contain certain object configurations. Consider, for instance, that a user wants to "find all images where there exists a building *adjacent* to the *west* side of a park which is *southwest* and *near* a commercial center". This query can be formulated as a constraint satisfaction problem (*CSP*) where the query variables are nodes of the corresponding constraint network and the image objects constitute the domain of each variable. The arcs of the network correspond to spatial constraints (e.g., adjacent $\wedge$ west (X_1,X_2), southwest $\wedge$ near (X_2,X_3)). Problems of the above nature are, in general, intractable. In addition, spatial constraints (e.g., southwest, near) lack universally accepted semantics and cannot always be modeled by crisp relations; a fact that further complicates query processing. This paper focuses on the development of effective methods that take advantage of the special structure of the spatial domain to achieve good average performance even for large images and queries.

1.1 Keywords

Spatial Constraints, Image Similarity Retrieval

2. INTRODUCTION

Image similarity based on visual characteristics such as shape, color and texture has been a topic of active research for many years [19]. *Configuration similarity* emerged in the context of spatial databases and geographic information systems (GIS) in order to complement existing methods of content based retrieval. Configuration queries retrieve all tuples of objects that satisfy a set of spatial constraints. Spatial constraints can be classified in three main categories: topological (e.g., inside, overlap), direction (e.g., north) and distance.

Most existing methods for configuration similarity retrieval focus on the special case where all images contain the same set of objects, possibly in different locations. [12] assumes that queries specify only direction relations between individual objects and applies symbolic object *projections* encoded in 2D strings and string matching algorithms for retrieval. [14] describes another projection-based technique that uses *conceptual neighborhoods* [11] to define relation similarity. [8] employs the angles between object centroids to define image similarity based on directions.

This paper addresses the general problem of configuration similarity where images contain arbitrary objects and queries refer to object variables rather than instances. This is, in general, a hard combinatorial problem [6] which can be thought of as a special case of subgraph isomorphism; queries correspond to subgraphs which are matched with stored image graphs. We deal with two instances of the problem: one where the whole process can take place in main memory, and one where the size of the images necessitates the application of secondary indexing methods. The structure of the paper is as follows: Section 3 describes spatial relations and similarity measures based on them. Section 4 proposes query and image pre-processing techniques, while Section 5 discusses algorithms for main memory retrieval. Section 6 combines search algorithms with spatial indexing to deal with disk retrieval, and Section 7 concludes the paper.

3. SPATIAL SIMILARITY

Topological constraints express the concepts of inclusion and neighborhood. A large body of the related work has focused on the *intersection* model [5] which describes relations using intersections of object's interiors and boundaries. The model defines the following set of 8 pairwise disjoint topological relations between planar regions: *T={disjoint, meet, overlap, covers, contains, equal, covered_by, inside}*. Figure 1 illustrates these relations in the form of a conceptual neighborhood graph. Nodes in the graph denote relations that are linked through an edge if they can be directly transformed to each other by continuous deformations (e.g., enlargement, movement). For instance, starting from relation *meet* and extending (or moving) one of the objects, we derive relation *overlap*. Depending on the allowed deformation and the relations of interest, several graphs may be obtained (e.g., [11] [14]). We employ the distance between two relations R_k and R_l in the graph to define their *similarity* σ as follows: (i) $\sigma(R_k,R_l)=1$ if $R_k=R_l$, (ii) $\sigma(R_k,R_l)=\tau$ $(0<\tau<1)$ if $R_k \neq R_l$ and distance$(R_k,R_l)=1$, (iii) $\sigma(R_k,R_l)=0$ otherwise.

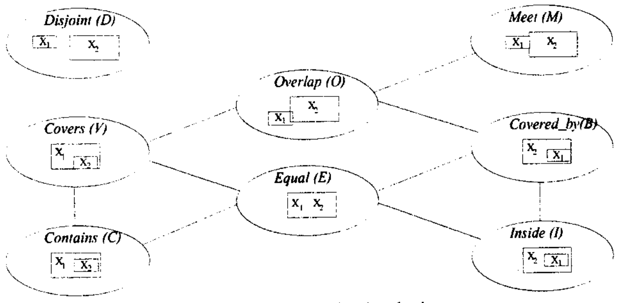


Figure 1 Topological relations

For direction relations we follow a centroid-based approach, where the direction between two objects is determined by the angle between their centroids. We use a set of 8 cardinal directions: northeast (NE), north (N), northwest (NW), west (W), southwest (SW), south (S), southeast (SE) and east (E). The similarity of each direction with a given angle θ between two centroids is illustrated in Figure 2 as a trapezoidal membership function of the angle. In case of northeast, for instance, (i) $\sigma(NE,\theta)=1$ if $\theta \in [45^o-\alpha, 45^o+\alpha]$, (ii) $\sigma(NE,\theta)=0$ if $360^o \geq \theta \geq 90^o$, (iii) $\sigma(NE,\theta)=\theta/(45-\alpha)$ if $0<\theta<45^o-\alpha$, and (iv) $\sigma(NE,\theta)=(90-\theta)/(45-\alpha)$ if $45^o+\alpha<\theta<90$.

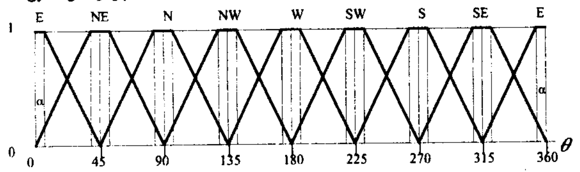


Figure 2 Directions relations

Object centroids are also considered for distances. A distance relation $R_{d_1 d_2}$ can be described by a range $[d_1,d_2]$ (meaning that the distance between two centroids is between d_1 and d_2 distance units). The similarity $\sigma(R_{d_1 d_2},d)$ between $R_{d_1 d_2}$ and a given distance d is given in the form of a membership function similar to the one used for directions: (i) $\sigma(R_{d_1 d_2},d)=1$ *if* $d \in [d_1,d_2]$, (ii) $\sigma(R_{d_1 d_2},d)=0$ *if* $d \geq d_2+\delta$ or $d \leq d_1-\delta$ (iii) $\sigma(R_{d_1 d_2},d)=(d+\delta-d_1)/\delta$ *if* $d_1-\delta<d<d_1$ and (iv) $\sigma(R_{d_1 d_2},d)=(\delta+d_2-d)/\delta$ if $d_2<d<d_2+\delta$.

The previous relations constitute a comprehensive way to express spatial queries; in addition to topology and directions, they capture distances and, according to our knowledge, this is the first approach that combines the three types of constraints in image similarity retrieval. The parameters, τ, α and δ, can be tuned to match different application or user needs providing flexibility to the model. However, we do not argue that the above relations and similarity measures exhaust all possibilities. The methods proposed in the paper can be used with alternative definitions that match the requirements of specific domains.

A configuration similarity query can be represented as a set of clauses in conjunctive normal form. Each clause corresponds to a constraint which can be one of the above relations, or a set (disjunction) of homogeneous relations from the respective domain (e.g., $NE \vee N$ or $D \vee M \vee O$); disjunctions of heterogeneous relations (e.g., $NE \vee M$) are not allowed. When a relation is left unspecified, the corresponding constraint is called *universal* (*U*) and it represents all relations (e.g., $U_T = D \vee M \vee O \vee V \vee C \vee E \vee I \vee B$, $U_D=R_{0\infty}$).

Consider an example query "retrieve all configurations of three objects where the first object *overlaps* the second one to the *southwest*, and their centroids are 2-3 distance units apart. The second object is *disjoint* and *northwest* of the third one, and their centroids' distance is 2-4". In order to find a triplet of objects that satisfy the above query within a database image, we have to instantiate each of the three query variables to image objects so that all input constraints are satisfied. Figure 3 illustrates two solutions <a,b,l> and <f,g,i> within an image containing 11 objects.

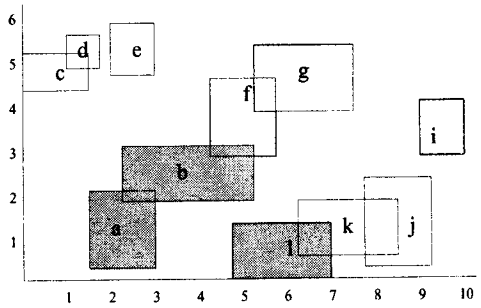


Figure 3 Example image and solutions

Let $M^{I,Q}$ be the set of *mappings* (1-1 functions) from query (*Q*) variables to image (*I*) objects. A *complete mapping* $m \in M^{I,Q}$ is one where all the query variables have been instantiated. Object $x_j = m(X_j)$ is called an *instantiation* of variable X_j under *m*. Let C_{ij}, be a spatial constraint between variables X_i and X_j, and $x_i = m(X_i)$, $x_j = m(X_j)$. If R is the relation between x_i and x_j in *I*, then the *degree of satisfaction* of C_{ij} is: $S(C_{ij})=max\{\sigma(R_x,R) / R_x \in C_{ij}\}$. The degree of satisfaction *S(m)* of a complete mapping *m* is computed from the satisfaction degrees of individual constraints according to various potential metrics [18]. Here we assume the *average combination*, i.e., *S(m)* is equal to the sum of satisfaction degrees of individual constraints divided by the total number of constraints. Similarity retrieval algorithms will find the best *K* complete mappings (*K* is user defined) according to the average combination metric. We use three types of image retrieval:

- *Hard retrieval* in which all complete mappings to be retrieved should totally satisfy all constraints (i.e., $S(m)=1$ for all solutions). Hard retrieval will not return any solutions if some constraint is not fully satisfied, even though there may exist images that match the query very closely. The following two methods overcome this problem.
- *Soft retrieval* will find all complete mappings that are good on the average even though they may totally or partially violate some constraints. Soft is considerably more expensive than hard retrieval because it has to generate more instantiations before it rejects a partial solution.
- Between soft and hard, there is *semi-hard retrieval* which excludes solutions that totally violate some constraint. It is generally a good trade-off because it reduces execution time compared to soft retrieval while missing only few solutions.

If N is the number of objects in image I, then the total number of complete mappings equals the number of n-permutations of the N objects: $N!/(N-n)!$. For most applications where $N>>n$, the number of permutations is $O(N^n)$. In the worst case all these mappings have to be searched, a fact that renders the problem intractable. Despite the exponential nature, we have developed methods which yield good average performance for considerable image and query sizes.

4. PRE-PROCESSING TECHNIQUES

The example query of the previous section (Overlap $\wedge$ SW $\wedge$ $R_{2\text{-}3}(X_1,X_2)$, Disjoint $\wedge$ NW $\wedge$ $R_{2\text{-}4}(X_2,X_3)$) contains several implicit constraints, i.e., relations between query variables X_1 and X_3 not explicitly stated. For instance, given the constraints between (X_1,X_2) and (X_2,X_3) the allowed topological constraint between X_1 and X_3 is Disjoint $\vee$ Meet $\vee$ Covers $\vee$ Contains $\vee$ Overlap. In order to explicate such relations in queries, we need *composition tables* that encode rules about the permissible relation between (X_i,X_j), given the constraints for (X_i,X) and (X,X_j). Two tables are required: one for topological relations and one for combined distances and directions.

For composition of topological relations we use the composition table presented in [5]. Because (in the context of this work) topological relations are defined on extended objects while distances and directions on centroids, no conclusion can be drawn about the direction and the distance between the centroids of two objects given their topological relation (and vice versa) unless we take advantage of domain knowledge (e.g., buildings 5 kms apart are *disjoint*). Since we provide a general framework of retrieval, not tied to any specific application, we assume independence of topological relations.

On the other hand, directions and distances defined on centroids are interrelated. For instance, in the previous query from SW$\wedge R_{2\text{-}3}(X_1,X_2)$ and NW$\wedge R_{2\text{-}4}(X_2,X_3)$ can be inferred that: (NW$\vee$W$\vee$SW)$\wedge R_{2,82\text{-}5}(X_1,X_3)$ (NW, W, SW are the only relations with potentially non-zero memberships). Constraints are refined when the composition constraint contains a proper subset of relations of the original one. If the intersection of the composition and original constraint is empty there is an inconsistency. During the first pre-processing step, a path consistency algorithm [13] extracts the implicit constraints according to composition tables and computes the transitive closure of the query. Composition tables and details about the explication of query constraints for image similarity retrieval can be found at [15].

The second step of pre-processing involves the re-ordering of query variables. In general, a good order is one where the most constrained variables are instantiated first because bad instantiations are detected and abandoned early in the search. We use a weight $W(R,I)$ to denote the strictness of a relation R in a particular image I. In order to calculate $W(R,I)$ we perform an exhaustive search in each image I and count the pairs of objects N_R that satisfy relation R in I. Then the weight of R is calculated as: $W(R,I)=N(N-1)/N_R$ (where $N(N-1)$ is the number of distinct object pairs).

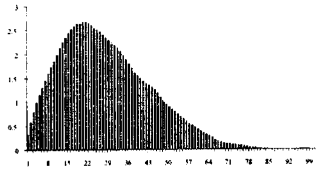
Relations that occur rarely get high weights because they have high discriminative value (this is similar to *inverse term frequency* used by information retrieval techniques). For instance, if a query specifies that two variables are *equal*, then these two variables should be instantiated first in order to prune the search space as early as possible. On the other hand, *disjoint* has a very small weight since for normal data density it is satisfied by more than 99% of object pairs; therefore a *disjoint* constraint is not significant. Figure 4 illustrates the weights for two images used in the following experiments. Given the pre-computed weights of relations, weights for constraints (i.e., disjunctions of relations of the same type) are calculated by the following equation which captures the property that the weight of a constraint is smaller than the weight of any (tighter) constraint with a proper subset of its relations:

$$\frac{1}{W(C,I)}=\sum_{\forall R\in C}\frac{1}{W(R,I)}$$

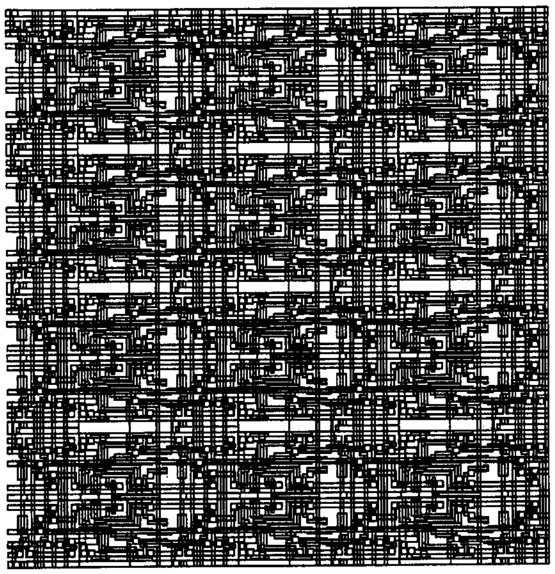
For instance, the weight of a constraint $C = NE$ in the image of Figure 4a would be 5.857, while the weight of another C' = N would be 13.949. The weight of a (less restrictive) constraint $NE\vee N$ would be: 5.857*13.949/(5.857+13.949) = 4.124. For distance constraints, we computed the distribution of distances normalized by maximum distance for all pairs of objects (Figure 4). Weights of ranges were calculated from these distributions using the above equation since a range can be thought of as disjunction of precise distances (the longer the range the smaller its weight).

Image meta-data (weights, distance distribution) is calculated only once and stored with each image. When an

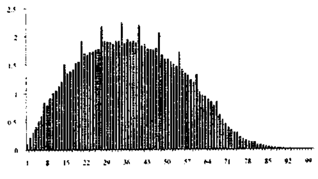
D	M	E	I
1.004002	289.8436	509545	63693.13
B	C	V	O
40763.6	63693.13	40763.6	2205.823
E	NE	N	NW
5.410793	5.857685	13.94977	13.73956
W	SW	S	SE
5.410793	5.857685	13.94977	13.73956



Weights and distance distributions - roads of Long Beach county



D	M	E	I
1.000942	1392.863	1038960	1038960
B	C	V	O
74211.43	1038960	74211.43	5171.1
E	NE	N	NW
8.104869	7.916698	7.804625	8.185119
W	SW	S	SE
8.104869	7.916698	7.804625	8.185119



Weights and distance distributions-VLSI Image

Figure 4 Images and their characteristics

image I is searched for a particular configuration expressed by Q, its associated meta-data is retrieved and each query variable is assigned a weight which is equal to the sum of weights of constraints in which it participates. Instantiation order of variables in Q for retrieval from I is then determined according to variable weights: the heaviest variable first and the least constrained last. In the next sections we show the significant effects of pre-preprocessing on the efficiency of retrieval.

5. MAIN MEMORY RETRIEVAL

This section describes configuration similarity retrieval when the image size permits the entire process to take place in main memory. In this case the whole image to be searched is retrieved and its *attributed relational graph (ARG)* is constructed. Each node in the ARG corresponds to an image object and each arc between x_i and x_j indicates: (i) the topological relation between (x_i,x_j), (ii) the angle and (iii) the distance between their centroids. Search is performed solely using the ARG which is maintained in memory. The *K* best complete mappings are retrieved and shown to the user. We experimented with four algorithms:

• The first algorithm is a non-recursive variation of backtracking (BT). After the ARG has been constructed, every query variable is instantiated to an image object according to the order determined during query pre-processing. When a variable is instantiated to some object, this object is "*locked*", i.e., it is removed from the domain of current and future variables and cannot be mapped to another query object. The new similarity is calculated by adding to the previous one the satisfaction degrees of the constraints that relate the new object with already instantiated objects. Depending on the new similarity and the type of retrieval used, the mapping proceeds forward (to the next variable) or backward. The algorithm proceeds forward if the instantiations so far constitute a *partial solution*; i.e., if all similarity degrees are 1 (for hard retrieval), or if the current similarity can exceed the target similarity of the K^{th} solution (for soft retrieval). The condition for semi-hard is the same as for soft, provided that no constraints have been totally violated. When the algorithm goes backward, another mapping for the same variable is chosen and the previous object is unlocked. If the variable domain is empty, the algorithm proceeds another step back and re-instantiates the previous variable after releasing all objects locked by subsequent variables.

A simplified version of the algorithm is illustrated below. *Instantiations* is an 1D array of *n* elements that holds the current values of variables (*instantiations[i]* stores the current value of x_i). *S[i]* holds the current similarity at the instantiation level *i* (variables up to the ith one have been instantiated). *Solutions* is a *Kxn* array that holds the *K* instantiations that have the highest similarity. *Target* is the similarity of the K^{th} solution; an instantiation will be included in the solutions only if its similarity is greater than *target*. When a variable is to be instantiated, BT chooses a value from its domain and calculates the *similarity* produced by the new instantiation and the already instantiated variables. If the new instantiation results in a partial solution, the algorithm proceeds forward, or outputs a solution if there are no other un-instantiated variables. Otherwise, it chooses a new value for the current variable, after restoring *new_value* to the domains of next variables.

```
BT (Query q, Image I, int K )
Preprocess(Q) /* compute query closure, retrieve I and metadata, and
determine variable ordering (most constrained first) */
FOR j := 1 TO n DO domain[j] = all objects in image I
S := target := 0;
i := 1; /* index to the current variable */
WHILE (TRUE) {
  similarity := 0;
  new_value := chooseNextValue(domain[i]);
  IF new_value = NULL THEN /* end of domain */
      IF i=1 THEN RETURN; /* end of domain for first variable*/
      ELSE i:=i-1; CONTINUE; /*Backtrack*/
  ELSE
      instantiations[i] := new_value; /*store instantiation*/
      FOR j:=i to n DO domain[j]:=domain[j]-{new_value}  /*locking*/
  FOR j:=1 to i -1 DO
     similarity:=similarity +σ(C_ij,R(instantiations[i],instantiations[j]));
   S[i] := S[i-1] + similarity;
   IF S[i] can exceed target THEN /* instantiated variables 1,..,i
    constitute a partial solution (depending on the type of retrieval) */
         IF i < n THEN /* intermediate variable instantiated */
                  i := i+1; /* successful instantiation: go forward */
           ELSE /*last variable instantiated*/
                  store(instantiations, solutions);
                  target = solutions[K];
   ELSE /*new instantiation does not result in a partial solution */
     FOR j:=i+1 to n DO domain[j]:=domain[j]∪{new_value}/*unlock*/
}
```

• The second retrieval algorithm is based on *backjumping* (BJ) [4]. BJ instead of going back to the previous variable when a deadlock occurs, jumps back to the last variable that precluded a candidate value from the current domain. Assume, for example, the instantiation order X_1, X_2, X_3, X_4, X_5, and that there does not exist a value in the domain of X_5 which is consistent with the instantiations of the previous variables. Furthermore, this inconsistency is solely due to the constraint between and X_5 and X_2 (e.g., a restrictive constraint such as *covers*). Backtracking to X_4 or X_3 will not solve the problem (the constraints between X_5 and X_4 or X_3 may be non-restrictive or even universal); backjumping, on the other hand, will re-instantiate the variable (X_2) that caused the problem, thus reducing the number of consistency checks. In order to do this, a pointer has to be kept for each variable to the last variable that caused an inconsistency. The forward move is the same as in backtracking, checking the constraints of the current variable with respect to previously instantiated ones.

• In the above example, BJ after re-instantiating X_2, would move forward, re-instantiate X_3 and so on. Assume that X_3 and X_4 are related by a restrictive constraint satisfied only by a few object pairs. Therefore, finding a good instantiation pair for X_3 and X_4 may require a significant amount of search which is redundant since such a pair was already found before the deadlock at X_5 occurred. Unlike BJ, *dynamic backtracking* (DBT) [7], after re-instantiating X_2, would keep the existing instantiations of X_3 and X_4, and directly attempt to find a new value for X_5. This means that the instantiation order is changed dynamically, i.e., from X_1, X_2, X_3, X_4, X_5, it becomes X_1, X_3, X_4, X_2, X_5, so DBT can be thought of BJ with *dynamic variable ordering* [2].

• The fourth algorithm is based on *forward checking* (FC) [3]: whenever a query variable is instantiated to an image object, the domain of remaining variables is searched and all image objects that cannot lead to a solution with similarity above the current threshold are removed. In order to achieve this, FC uses a three dimensional *domain table* which keeps track of the consistent values for each variable at every instantiation level. Whenever an instantiation causes the domain of a future variable to become empty the algorithm immediately backtracks, in contrast to the previous algorithms which would continue the search. This ensures that a query variable is never instantiated to an object if the resulting partial solution cannot lead to a complete mapping with a similarity above the current target. Furthermore, since the domain of the last variables is pruned by previous instantiations, fewer instantiations have to be done to find a solution (but more checks for each instantiation). A description of FC for similarity retrieval can be found in [16].

In order to test the performance of the algorithms we used real geographic and VLSI data sets of various sizes. In particular, we constructed images of 100, 200, ... and 500 objects using portions of the map in Figure 4a, and the VLSI image of 4b (a total of ten images). The parameters for relations were set to: $\tau=0.33$, $\alpha=5$ and $\delta=0$, while K=100. Because actual queries may vary significantly depending on the domain, we constructed an artificial set of 70 queries each consisting of 3 to 9 variables (10 queries of 3 variables, 10 of 4, ..., 10 of 9 variables). Query tightness varies from complete queries created using a query-by-sketch language to very loose queries involving only a few non-restrictive constraints. The implementation was done using Java Symantec JIT compiler and the experiments were run on several Pentium PCs 133MHz with 64M Ram.

We executed all queries (70), for the ten images using the four algorithms (BT, BJ, DBT and FC) with and without pre-processing, for the three retrieval types (hard, soft, semi-hard) (i.e. a total of 16800 runs). Each execution was allowed 10 minutes to complete; after this period it was terminated. Figure 5 illustrates the average times (in sec) for each algorithm and retrieval type combination for queries with four variables and images of 300 objects. FC has the best performance for all types of retrieval (the other combinations of query/image sizes yield very similar relative performance).

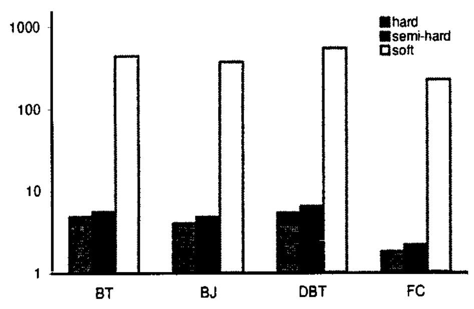


Figure 5 Main memory retrieval

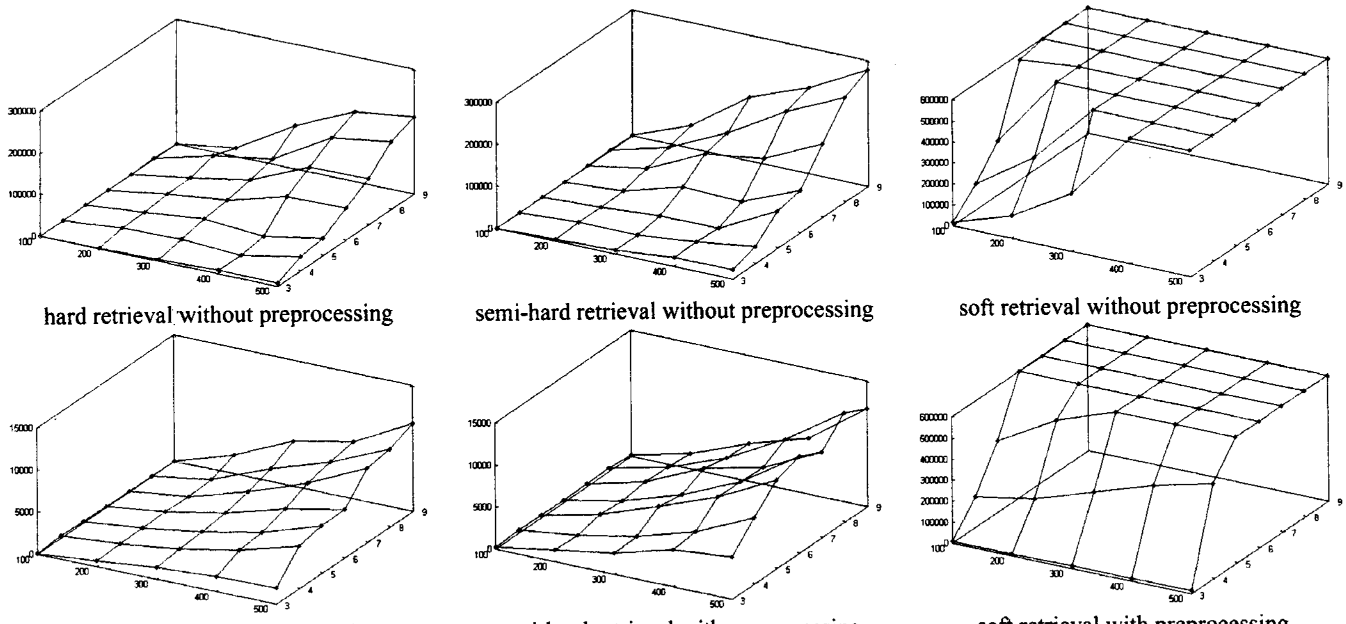
An interesting observation from the above graph refers to the satisfiability types. Soft retrieval is 1-2 orders of magnitude slower than semi-hard and prohibitively expensive for real-time applications. On the other hand, semi-hard retrieval is only about 10% slower than hard, while in 90% of the cases it retrieved the same solutions as soft (solutions that are good on the average while totally violating some constraint are rare, especially for large images). In general, semi-hard seems to be a very good trade-off for applications involving large images and require approximate retrieval.

Figure 6 illustrates the effect of pre-processing on the performance of FC. Each chart shows the response time (in milliseconds) as a function of the number of query variables (3,..,9) and the image size (100,…,500). Soft retrieval did not terminate successfully for most queries involving more than four variables. The experiments illustrate that pre-processing speeds up query processing more than an order of magnitude for all types of satisfiability. The same is true for the rest of the algorithms. For most queries, semi-hard FC with pre-processing will find solutions in less than 10 seconds even if image sizes reach 500 objects.

In actual applications, much larger images could be effectively processed since often queries involve some other features that prune the search space. GIS queries, for instance, usually include properties of objects (e.g., "find all maps where there is a river crossing a large city" etc.) that restrict variable domains to a small percentage of the image size. For spatial applications involving very large images, it is infeasible to construct and maintain in main memory the ARGs of the images, in order to solve the configuration similarity problem. A secondary memory data structure should be incorporated in the search algorithms, to overcome the main memory restrictions. The next section shows how *R-trees* can be combined with search techniques to speed up retrieval from the disk.

6. SECONDARY MEMORY RETRIEVAL

The R-tree [10] is a multidimensional extension of the height-balanced B-tree, used to store large numbers of objects in space. The minimum bounding rectangles (MBRs) of the actual data objects are stored in the leaf nodes of the tree. Intermediate nodes are built by grouping rectangles at the lower level. Each node of the tree corresponds to a disk page and is associated with some rectangle which encloses all rectangles that correspond to lower level nodes. Figure 7 illustrates the construction of an R-tree using the image of Figure 3 and assuming a capacity of 3 rectangles per page (normally the number of MBRs per page is in the range 30-100). Object MBRs (denoted with small letters) are grouped together in four intermediate nodes (A,B,C,D) which are in turn grouped in two larger ones (1 and 2). For our experiments we re-implemented in Java one of the most successful variations of R-trees, namely the R*-trees [3].



hard retrieval without preprocessing semi-hard retrieval without preprocessing soft retrieval without preprocessing

hard retrieval with preprocessing semi-hard retrieval with preprocessing soft retrieval with preprocessing

Figure 6 FC performance

The processing of a traditional overlap query (light gray window) in R-trees involves the following procedures: Starting from the top node, exclude the nodes that are disjoint with the query window, and recursively search the remaining ones (gray nodes in the tree of Figure 7). Among the entries of the leaf nodes retrieved, select the ones that overlap the query window. Two types of query windows may arise in similarity queries: rectangular ones for topological constraints between MBRs [17], and angular/ring windows for directions/distances between centroids.

In case of hard retrieval only the objects that fall inside the range for total similarity are retrieved. In addition, semi-hard retrieval finds the ones that fall in the area of partial satisfaction. The ring window of Figure 7, for instance, corresponds to semi-hard search for NE $\wedge$ $R_{2\text{-}3}$ (X,*b*); the window for hard retrieval would include only the slice of angle 2α where NE is totally satisfied. Soft retrieval is not implemented; it would not take advantage of R-trees since it requires retrieval of objects even if they do not overlap the query window. However, because of the size of the images, semi-hard retrieval will not miss any solutions in the vast majority of the cases.

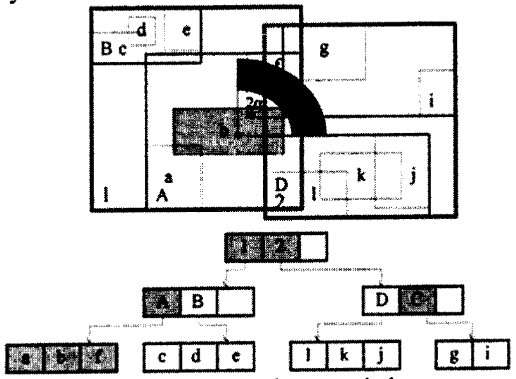


Figure 7 R-trees and query windows

The algorithm for configuration similarity retrieval using R-trees works as follows. In order to instantiate the first query variable X_i (the one with the largest weight according to query constraints and image meta-data) a potential query window is calculated from the given constraints. For instance, if there is a constraint N $\wedge$ $R_{5\text{-}6}(X_k,X_i)$, then X_i should lie at least 5 distance units south of the north border of the total space (otherwise there cannot not exist any object for the instantiation of X_k). The second variable X_j to be instantiated is the one that has the largest weight taking into account only the constraints C_{ji} with respect to the first one. This is because restrictive constraints imply that relatively few objects can be in the domain of the second variable (notice that the ordering strategy is slightly different from the one applied for main memory retrieval). Let x_i be the instantiation of X_i. A window query is performed using x_i as the reference object and C_{ji} as the spatial condition. The image objects that fall inside this query window are retrieved and constitute the domain of X_j. As the algorithm proceeds forward the query windows become gradually smaller because the new constraints add more restrictions.

Consider again the query Overlap $\wedge$ SW $\wedge$ $R_{2\text{-}3}(X_1,X_2)$, Disjoint $\wedge$ NW $\wedge$ $R_{2\text{-}4}(X_2,X_3)$ and that X_1 is the first variable to be instantiated to object *b* of the image in Figure 7. If the second object to be instantiated is X_2, the R tree is used to search the potential instantiations using the constraints between X_2 and X_1 (Overlap, NE, $R_{2\text{-}3}$). Two searches will be performed: Overlap(X_2,*b*) which corresponds to the light gray window in Figure 7, and NE $\wedge$ $R_{2\text{-}3}$ (X_2,*b*) that corresponds to the dark gray one. The actual domain for X_2 consists of the intersection of objects found in the two searches (i.e., only object *f*). Therefore, the instantiations for the first two variables are: *b*=*m*(X_1) and *f*=*m*(X_2).

When the third variable (X_3) is instantiated, its potential domain is restricted by its constraints with respect to both previously instantiated variables. These constraints produce two sets of query windows: one corresponding to Disjoint $\wedge$ SE $\wedge$ $R_{2\text{-}4}(X_1,f)$ (due to the explicit constraint between X_3 and X_2) and (Disjoint $\vee$ Meet $\vee$ Covered_by $\vee$ Inside $\vee$ Overlap) $\wedge$ (NE$\vee$E$\vee$SE)$\wedge R_{2,82\text{-}5}(X_3,b)$ (due to the constraint between X_3 and X_1 generated by path consistency). Since there do not exist any object centroids in the intersection of the query windows (see Figure 8), X_3 cannot be instantiated. Consequently, the algorithm backtracks and attempts to find another instantiation for X_2. Because *f* was the only object in the domain of X_2, the algorithm will backtrack again and attempt to re-instantiate X_1.

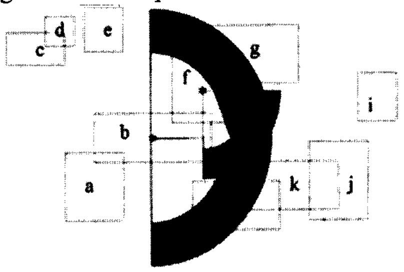
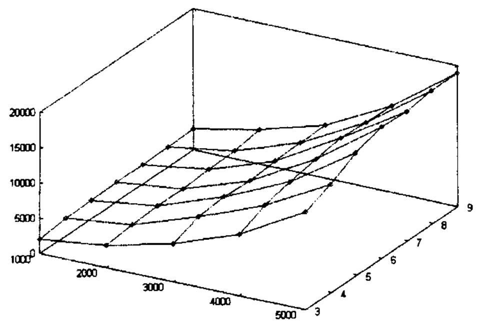
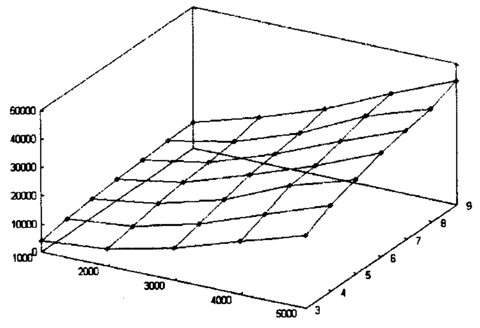


Figure 8 Instantiation of X_3

We tested the performance of the secondary memory retrieval algorithm, with the queries of the previous section and R-trees of 1K, 2K, ... 5K objects (created from the images of Figure 4). Figure 9 illustrates the mean response time (for hard and semi-hard retrieval), as a function of the query and image size (preprocessed queries). The diagrams show that, as opposed to main memory retrieval, the image size is the most decisive factor for the response time, especially for hard retrieval. This is because query windows prune the search space significantly; once the first few variables have been instantiated, the domains of future variables are significantly restricted to small areas of space or totally eliminated, thus avoiding useless forward checks.



hard retrieval



semi-hard retrieval

Figure 9 Secondary memory retrieval

In general, the above algorithm can be thought of as a form of forward checking with depth 1, since once a variable X_i is instantiated the domain of the subsequent variable X_{i+1} is restricted to the image objects that fall inside the query window that is specified by X_i and the constraints between X_{i+1} and X_i. [16] provides a detailed description and experimental evaluation of several algorithms for secondary memory retrieval using an alternative model of spatial relations.

7. CONCLUSION

This paper focuses on image similarity retrieval based on topological, direction and distance constraints. Despite the exponential nature of the problem we propose techniques that take advantage of the special structure of space and address the problem even for large images. In particular we use two methods: one for images that can be handled entirely in main memory and one for large images that need spatial access methods for efficient disk retrieval. Pre-processing, which involves computing the transitive closure of the query and reordering of variables based on strictness of constraints, facilitates efficiency.

We are currently experimenting with advanced spatial database techniques such as spatial joins [9]. Future work will attempt to combine alternative search algorithms and spatial indexing methods to enhance performance. Finally, reasoning mechanisms can be incorporated in the system. Assume that the three objects that match the description of the example query are not in the same image, but two objects x_1, x_2 that match the constraints between X_1 and X_2 exist in image I_1, while x_2 (common object) and x_3 match the constraints between X_2 and X_3 but exist in I_2. In order to retrieve a complete solution we need reasoning to combine information found in the two images (similar to a map overlay operation).

8. REFERENCES

[1] Bacchus, F., Grove, A. On the Forward Checking Algorithm. International Conference on Principles and Practice of Constraint Programming, 1995.

[2] Bacchus, F., Van Run, P. Dynamic Variable Ordering for CSPs. International Conference on Principles and Practice of Constraint Programming, 1995.

[3] Beckmann, *N*., Kriegel, H.P., Schneider, R., Seeger, B. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. SIGMOD, 1990.

[4] Dechter, R. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. Artificial Intelligence 41, 273-312, 1990.

[5] Egenhofer, M., Sharma, J. Assessing the Consistency of Complete and Incomplete Topological Information. Geographical Systems, Vol. 1, No 1, pp. 47-68, 1993.

[6] Grigni, M., Papadias, D., Papadimitriou, C. Topological Inference. IJCAI, 1995.

[7] Ginsberg, M. Dynamic Backtracking. Journal of Artificial Intelligence Research, 1, 25-46, 1993.

[8] Gudivada, V., Raghavan, V. Design and Evaluation of Algorithms for Image Retrieval by Spatial Similarity. ACM TOIS, 13(1):115-144, 1995.

[9] Guenther, O. Efficient Computation of Spatial Joins. *IEEE 9th ICDE*, 1993.

[10] Guttman, A. R-trees: a Dynamic Index Structure for Spatial Searching. SIGMOD, 1984.

[11] Hernandez, D. Qualitative Representation of Spatial Knowledge. Springer Verlag LNAI, 1994.

[12] Lee S-Y., Hsu F-J Spatial Reasoning and Similarity Retrieval of Images Using 2D C-string Knowledge Representation. Pattern Recognition, 25(3):305-318, 1992.

[13] Mackworth, A, Freuder, E. The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. Artificial Intelligence,25,65-74, 1985.

[14] Nabil, M., Ngu, A., Shepherd, J. Picture Similarity Retrieval Using 2d Projection Representation. IEEE TKDE, 8(4),1996.

[15] Papadias, D., Arkoumanis, N., Karacapilidis, N. On The Retrieval of Similar Configurations. 8th Symposium on Spatial Data Handling (SDH), 1998.

[16] Papadias, D., Mamoulis, N., Delis, B. Algorithms for Querying by Spatial Structure. VLDB, 1998.

[17] Papadias, D., Theodoridis, Y. Spatial Relations, Minimum Bounding Rectangles, and Spatial Data Structures. International Journal of GIS, 11(3) , pp. 111-138, 1997.

[18] Ruttkay Z. Fuzzy Constraint Satisfaction. IEEE International Conference on Fuzzy Systems, 1994.

[19] Santini S., Jain, R. Similarity Searching. TR VCL-95-110, Visual Computing Laboratory, UCSD, 1995.