Content-Based Retrieval using Heuristic Search

Dimitris Papadias, Marios Mantzourogiannis, Panos Kalnis, Nikos Mamoulis, Ishfaq Ahmad
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
+852-23586971

http://www.cs.ust.hk/{~dimitris, ~mantzour, ~kalnis, ~mamoulis, ~iahmad}

ABSTRACT

The fast growth of multimedia information in image and video databases has triggered research on efficient retrieval methods. This paper deals with structural queries, a type of content-based retrieval where similarity is not defined on visual properties such as color and texture, but on object relations in space. We propose the application of heuristic algorithms which provide good, but not necessarily optimal, solutions in a pre-determined time period, and compare our approach with systematic search methods which are guaranteed to find optimal solutions but require exponential time in the worst case. The quality of the output is calculated using a relation framework which is an extension of Allen's relations. With this framework our methods can be applied in multiple resolutions and dimensions, thus covering a wide range of applications in spatial, multimedia and video systems.

Keywords

MMIR, content-based indexing/retrieval, image indexing/retrieval

1. INTRODUCTION

Effective content-based retrieval of imagery and video can be performed at three abstraction levels [3] [7][15]:

Raw Data: At the lowest abstraction level, objects are simply aggregations of raw pixels. Comparison between objects or regions is done on a pixel-by-pixel basis using similarity measures such as the correlation coefficient and the euclidean distance.

Feature: A feature is a distinguishing primitive characteristic or attribute (e.g., luminance, shape descriptor, gray scale texture, color histogram, and spatial frequency).

Semantic: At the highest abstraction level, retrieval assumes that features have been grouped into meaningful objects and semantic descriptions have been attached to scenes. Search is performed on entities with well defined spatio-temporal properties.

For instance, a simple object is a connected region of raw pixels (at the lowest abstraction level), or where selected features are homogeneous (e.g., texture) at the second level, or with distinct semantics (e.g. human, building) at the semantic level. Existing content-based systems include MIT's PhotoBook [28], IBM's QBIC [22], VisualSeek [29] and the Multimedia Datablade from Infomix/Virage [2].

In the Proceedings of the *22nd ACM-SIGIR Conference on Research and Development in Information Retrieval*, Berkeley, August 15-19, 1999.

Here we deal with a form of content-based retrieval that can be applied at the feature and semantic level, and is based on configuration similarity and spatio-temporal structure. The corresponding *structural* (or, otherwise called, *configuration*) queries ask for a set of objects which satisfy some spatio-temporal constraints, e.g., "find all triplets of objects (v_1,v_2,v_3) such that v_1 is *northeast* of v_2 which is *inside* v_3" or "find all scenes which contain a picture of an island accompanied by a textual description on its *left*, immediately *followed* by a frame that *contains* both windows". Thus, structural queries pre-suppose an object-oriented architecture where pre-processing techniques have been applied to extract information about the objects in a spatial scene and their locations.

There exist two serious impediments for the efficient processing of configuration similarity. First, the complexity of the problem is, in general, exponential [25] and systematic search through the whole solution space does not guarantee worst case performance. In order to avoid this situation, most researchers [16][23] focus on a special case where all images/frames contain exactly the same set of labeled objects. Here we take a different approach and deal with the general case (i.e., we do not make any assumptions about data size and type of objects) by applying some non-systematic search heuristics which provide sub-optimal solutions in limited time.

The second impediment for structural queries is that, due to the inherent uncertainty in spatio-temporal relations, queries do not always have exact matches. Like text information retrieval techniques, the output should have an associated score to indicate its similarity to the input query. In order to provide a general application-independent solution for configuration similarity retrieval, we use a relation framework which can be easily extended to multiple resolutions and dimensions. Different users, even in the same system, may employ different sets of relations; the framework can be adjusted on-the-fly, automatically providing similarity measures depending on the resolution.

The rest of the paper is organized as follows: Section 2 describes the framework for spatio-temporal structure and the similarity measures used. Section 3 employs genetic algorithms for query processing and section 4 illustrates the application of iterative improvement and simulated annealing. Section 5 presents the experimental results comparing the above algorithms with random and systematic search techniques. Section 6 concludes the paper with a discussion.

2. CONFIGURATION SIMILARITY

Several approaches (see [23] for references) have used Allen's [1] relations to assess similarity of spatial scenes. The applicability of this framework in actual multi-dimensional systems, however, is restricted by its limited expressive power. For instance, Allen's relations do not capture the concept of distance; if two intervals are disjoint, their distance is not important in determining their relative relation. In order to overcome this deficiency, we employ a *binary string encoding* of relations [8] which defines spatio-temporal relations at various resolution levels providing means for the representation of distances and refined topological and direction information.

Figure 1 illustrates the *conceptual neighborhood* graph [11] for a distance-enhanced resolution scheme organized according to the binary string encoding (alternative schemes can be found in [8]). A reference interval [a,b] divides 1D space in nine regions (points or open intervals) of interest, (-∞,a-δ), [a-δ,a-δ], (a-δ,a), .., (b+δ,∞), each represented by a bit. The relation between a primary interval and [a,b] is then determined according to which of the regions are intersected; the corresponding bits are set to 1 defining a 9-bit string (i.e., a relation). For instance, $R_{100000000}$ denotes that the upper (primary) interval is to the left and more than δ distance units away from the leftmost point of the lower (reference) interval. $R_{110000000}$ is similar but implies that the upper interval ends exactly δ units before the beginning of the lower one.

The neighborhood graph has the property that the similarity between two relations is proportional to the proximity of the corresponding nodes. In general, each relation R_x may have up to four 1st degree neighbors, denoted *right*(R_x), *left*(R_x), *up*(R_x), *down*(R_x). *Right*(R_x) can be derived from R_x by finding the first "0" after the rightmost "1" and replacing it by a "1", while, *up*(R_x) can be derived from R_x by pumping an "1" from the left. The *distance* between two relations is defined as the length of the shortest path connecting them in the graph, and can be computed directly from their binary string encoding without the need of look-up tables.

The above concepts can be extended accordingly to multi-dimensional spaces. A *D*-dimensional relation is defined as a *D*-tuple of 1D *projections*, e.g., $R_{000001100\text{-}100000000}(v_1,v_2)$ implies $R_{000001100}(v_1,v_2)$ for axis *x*, and $R_{100000000}(v_1,v_2)$, for axis *y*. For *x* we assume a west-east direction, while for *y* north-south (according to the co-ordinate system used for the computer screenshots). In order to derive a 1st degree neighbor of a multi-dimensional relation we simply replace one of the constituent 1D projections with its neighbors. As a result, computing *D*-relation distances is reduced to calculating 1D distances. Time can be easily incorporated as an extra dimension with the same semantics.

The automatic calculation of similarity measures in multiple resolutions and dimensions allows users to ask structural queries by choosing their individual resolution schemes which may change for different queries. As an example consider the scheme of Figure 1 and the structural query of Figure 2a. The prototype configuration is drawn using a *query-by-sketch* language where the distance of the grid is set to δ (δ is user-defined). The goal is to find configurations of stored objects matching the input exactly or approximately.

Formally, a structural query can be described as a binary constraint satisfaction problem [25] which consists of:

- A set of *n* variables, $v_0,v_1,\ldots,v_{n-1}$ that appear in the query.
- For each variable v_i, a finite domain $D = \{u_0,\ldots, u_{N-1}\}$ of *N* values (we assume that all variables have the same domain). Values can be distinct objects in the case of semantic retrieval, or pseudo-objects (e.g., regions with a specific texture) for retrieval at the feature level.
- For each pair of variables (v_i,v_j), a constraint C_{ij} which is a disjunction of relations from the resolution scheme in use.

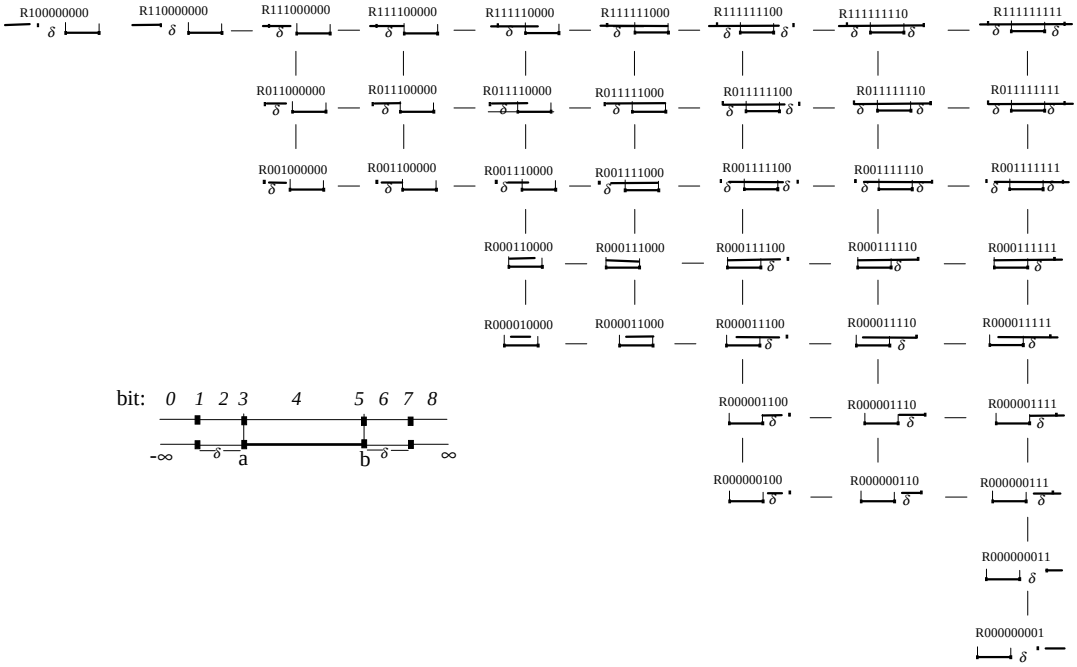
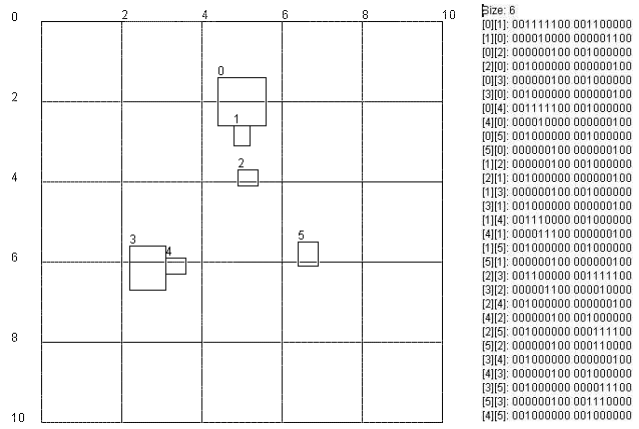


Figure 1 1D conceptual neighborhood graph for a distance-enhanced resolution scheme



(a) query (b)constraints

Figure 2 An example structural query

The query of Figure 2a contains six variables (v_0,…, v_5), one for every drawn object. The domain of each variable consists of all objects in the image to be searched for the particular configuration (i.e., all domains are identical). Figure 2b illustrates the set of binary constraints between all pairs of variables. For instance, the relation between query objects (variables) 0 (v_0) and 1 (v_1) is $R_{001111100\text{-}001100000}$. Alternatively the query could be expressed by an extended SQL language: *select* v_0,..,v_5, *from* ImageDB, *where* $R_{001111100\text{-}001100000}(v_0,v_1)$... Linguistic terms may be used instead of bit-strings e.g., *meets-north*(v_0,v_1) instead of $R_{001111100\text{-}001100000}$. Although the particular query specifies constraints between all pairs of variables, in some cases queries may be *incomplete* (some constraints may be unspecified) or *indefinite* (constraints may be disjunctions of relations). [27] describes a pictorial language for the expression of such queries.

In addition to content-based retrieval, structural queries can be applied with metadata, i.e., annotated images. Furthermore, in real applications some additional unary constraints may appear; these may specify object properties at the feature (e.g., v_0 is red) or semantic level (e.g., v_1 is a building). Although such constraints are easy to handle (provided that the corresponding properties have been extracted), for generality we omit them here and deal only with binary spatio-temporal ones.

Once the query is submitted, retrieval algorithms will attempt to find instantiations of query variables to stored objects such that the input binary constraints are satisfied to a maximum degree. A binary instantiation $\{v_i \leftarrow u_k, v_j \leftarrow u_l\}$ is *exact*, if $R(u_k,u_l) \subseteq C_{ij}$. If, for instance, the constraint C_{ij} between v_i and v_j is $R_{100000000} \vee R_{110000000}$ and the relation between u_k and u_l is one of these relations, then C_{ij} is exactly matched by the instantiation $\{v_i \leftarrow u_k, v_j \leftarrow u_l\}$. On the other hand, if the relation between u_k and u_l is $R_{111000000}$, the constraint is only approximately matched; its *inconsistency degree* d_{ij} equals the minimum distance between $R_{111000000}$ and $R_{100000000} \vee R_{110000000}$ (which is 1 because $R_{111000000}$ is a 1st degree neighbor of $R_{110000000}$).

The inconsistency degree d of a complete *solution* $S=\{v_0 \leftarrow u_p, .., v_i \leftarrow u_k, .., v_j \leftarrow u_l, .., v_{n-1} \leftarrow u_r\}$ is defined as the sum of inconsistency degrees of all binary constraints:

$$d = \sum_{\forall ij,\ i \neq j \text{ and } 0 \le i,j < n} d_{ij}(C_{ij}, R(u_k,u_l)) \quad where\{v_i \leftarrow u_k, v_j \leftarrow u_l\}$$

Given the inconsistency degree of a solution, we define its similarity f normalized within the range [0,1] in order to maintain uniformity over various problem sizes:

$$f = \frac{n(n-1)*MD - d}{n(n-1)*MD}$$

where $n(n\text{-}1)$ is the set of constraints between distinct variable pairs (including inverse and unspecified constraints) and MD is the maximum distance in the neighborhood graph; for the distance-enhanced scheme, MD is 16 in 1D and 32 for 2D space.

If N is the number of domain objects, and n the number of query variables, the total number of possible solutions is equal to the number of n-permutations of the N objects: $N!/(N\text{-}n)!$. Thus, systematic algorithms (e.g., backtracking), that search through the whole space, cannot guarantee acceptable worst case performance. In the rest of the paper we deal with an alternative form of processing where the goal is to retrieve the best possible solutions within a limited time. In this case, heuristic techniques yield, as we show in the experimental evaluation, better performance than systematic search. The next section illustrates the application of evolutionary methods (i.e., genetic algorithms).

3. GENETIC ALGORITHMS

Genetic algorithms (GA's), introduced in [17], are search methods based on the evolutionary concept of natural mutation and the survival of the fittest individuals. Given a well-defined search space, three different genetic search operations, *selection*, *crossover* and *mutation*, are applied to transform an initial population of chromosomes with the objective to improve their quality. A chromosome is an encoded representation of a feasible solution (i.e., in our problem an assignment of each query variable to an image object). Before the search process starts, a set of P chromosomes (called initial population) is initialized to form the first generation. Then the three genetic search operations are repeatedly applied in order to obtain a population (i.e., a new set of solutions) with better characteristics. This new population will constitute the next generation, at which the GA will perform the same actions and so on, until a stopping criterion is met. Next we demonstrate a *genetic configuration similarity algorithm* (GCSA), by presenting the encoding mechanism and then the selection, crossover and mutation operators.

Encoding mechanism: Each chromosome is simply an array S of n values, where S[i] is the instantiation of variable v_i in solution S. The quality of S is measured by its *fitness* f (i.e., its similarity). F is the average fitness of a population of chromosomes.

Selection mechanism: This operation consists of two parts: evaluation of a chromosome and offspring allocation. Evaluation is performed by measuring the above defined fitness value; offspring generation is then done by allocating to each chromosome, a number of offspring proportional to its fitness. GCSA implements the *stochastic remainder technique* [30]: a chromosome is assigned offspring according to the integer part of the proportionate fitness (f/F) value in a deterministic way and the fractional parts are put in a roulette wheel[1], for determining the

[1] *Roulette wheel selection* allocates a sector of the wheel equal to $2\pi f/F$ to every chromosome and then creates an offspring if a generated number in the range of 0 to 2π, falls inside the assigned sector of the chromosome.

remaining offspring. Thus, we restrict randomness to the fractional parts only and assure that a good chromosome will not vanish, which is possible, especially in the early generations.

Crossover mechanism is the driving force of *exploration* in GA's. In the simplest approach [17], pairs of chromosomes are selected randomly from the population. For each pair a crossover point is defined randomly, and the chromosomes beyond it are mutually exchanged, with probability μ_c (*crossover rate*), producing two new chromosomes. The rationale is that after the exchange of genetic materials, the two newly generated chromosomes are very likely to possess the good characteristics of their parents (*building-block hypothesis* [14]). In our case this corresponds to swapping of the assignments in two solutions after a selected point. One-point crossover seems to be inefficient for our application domain, since the probability of a bit to be swapped increases as we go to the end of the string. Instead we selected a two-point crossover mechanism for GCSA: after the pairing of chromosomes, two crossover points are randomly selected and the portion of the chromosome in between them is swapped. The whole operation is performed with probability μ_c.

Mutation mechanism: Mutation aims at restoring lost genetic material and is performed in GCSA by simply changing a variable instantiation with a probability μ_m, called the *mutation rate*. Although mutation is not the primary search operation and sometimes is omitted, it may be very useful for *exploitation*, i.e., cases where, through selection and crossover, all the chromosomes have converged to a local optimum for some variable.

GCSA starts with an initial population of *P* randomly generated chromosomes/solutions and terminates after the creation and evaluation of *G* generations. If only one solution is needed, then the best chromosome among all generations is returned. In the current problem, however, the user may require the best *K* solutions. In this case the *k* distinct chromosomes are extracted. If $k<K$ the algorithm is executed repeatedly with different initial populations. There is also the option of specifying a *target* fitness (i.e., retrieve the best *K* solutions where the similarity is greater than *target*), in which case only the *k* chromosomes that exceed the target are kept at each run of GCSA.

Several theoretical and empirical studies [9][13][14] have been carried out on tuning the control parameters, *P*, *G*, μ_c and μ_m. Most researchers suggest that the mutation and the crossover rate should be in the range of 0.001% - 0.05% and 0.60-0.95 respectively. We experimented with these values using various queries; the best results for most cases were achieved for μ_m =0.05% and μ_c=0.60. Intuitively, the population size *P* should increase with the domain size, since a larger population has more information capacity to provide accurate sampling for the larger domain. Unfortunately very large values cannot be applied in practice, because this would limit *G* which usually leads to poor results. We experimentally tested the behavior of the algorithm for *P* in the range 50 – 300 by using the queries and datasets described in section 5.

Figure 3 shows the fitness of the solution as a function of *P* and *G*. The different values of *P* do not affect fitness significantly; we chose *P*=50 because this value produces fair results for all cases and is small enough to allow the individuals to evolve through many generations within an acceptable running time. Combined

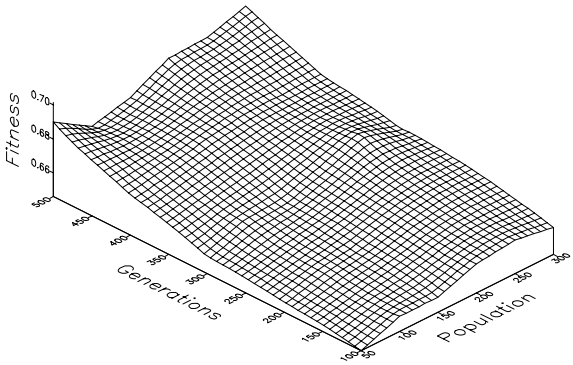


Figure 3 Parameter tuning for GCSA

with the relatively large value of μ_c (0.60) GCSA was able to exploit a large portion of the solution space.

4. HILL CLIMBING ALGORITHMS

The problem space for structural queries can be thought of as a graph, where each solution corresponds to a node associated with a similarity value. Our goal is to find the nodes with the globally maximum similarity, i.e. the best solutions. Hill climbing algorithms operate on such a graph, performing random walks between the nodes based on a certain movement (transition) mechanism. This transition mechanism defines a neighborhood for each node *S*, which consists of all the nodes that can be reached from *S* in one move. In our case, the neighbors of *S* are all the solutions that can be derived from *S* by changing the assignment of a single variable, i.e., a node has $n(N-1)$ neighbors (each variable can take $N-1$ values, excluding its current assignment). A move is called uphill, if it leads to a better solution and downhill if the destination node has lower similarity.

4.1 Iterative Improvement

Configuration similarity iterative improvement (CSII) starts with a randomly chosen initial solution and tries to find a better neighbor. If such a solution is found, the current one is replaced by the new one, otherwise the algorithm keeps the initial solution. The process continues until a local maximum is found. This iterative optimization is repeated a number of times, each time starting from a different random solution. As in the case of GCSA, the user defines the stopping criterion by specifying the running time, or providing the target similarity of the solutions to be retrieved.

As time approximates ∞, the probability that iterative improvement will find the global maximum approximates 1 [24]. However, given a finite amount of time, the algorithm terminates at a local maximum. In general, the execution time is proportional to the number of neighbors tested. Exhaustive search of all neighboring $n(N-1)$ solutions involves significant cost for large domains. In order to deal with this problem, CSII searches only a percentage (P_{neig}) of the neighbors (a similar approach is taken in [19]). Since the optimal value of P_{neig} is strongly related to each specific problem, we tested values ranging from 30% to 80% over several queries and datasets. In most cases, P_{neig}=60% gives the best results and we use this value in the experimental evaluation.

4.2 Simulated Annealing

Configuration similarity simulated annealing (CSSA), based on [21][5], performs random walks just like iterative improvement

but in addition to uphill, it also accepts downhill moves with a certain probability, trying to avoid local maxima. Figure 4 illustrates CSSA for the case where the user requires the best K solutions exceeding the similarity specified by *target*. *Solutions* is a Kxn array that stores the K current solutions.

```
CSSA(int K, target)
S = S0; T = T0;//S is initialized to random solution
store(S);
WHILE (not stopping criterion) {
    WHILE ( not equilibrium) {
        S' = random neighbor of S;
            IF (similarity(S')>similarity(solutions[K])) AND
                (similarity(S')>target) THEN
                    store(S');
                    Df = similarity(S') – similarity(S);
                    IF (Df >= 0) THEN S = S';
                        ELSE  IF (random[0,1) < exp(Df/T) )
                                THEN  S = S';
    }   //end while
  reduce T;
} //end while
```

Figure 4 CSSA

The inner for-loop is called *level*. Each level is executed with a fixed value of the parameter T. The starting value of T is such that the probability $\exp(D_f/T)$ at the first levels approximates 1, where D_f denotes the difference between the similarity of the current solution S and the new random neighboring solution S'. After the execution of each level, T is reduced according to some function, and the next level is performed using the new value of T. This means that the probability of accepting a downhill move is greater in the earlier levels and decreases in the subsequent ones. CSSA terminates when the value of T is very close to zero and thus the probability of accepting downhill moves is almost 0. Another way for the algorithm to stop is when a fixed criterion is reached; for example, when a solution with a given target similarity has been found.

As with previous algorithms, the quality of the output is strongly related to the values of some parameters. In order to define the initial value T we adopt the method of [20] [21]: a large value for T_0 is chosen and a number of transitions is performed. If the acceptance ratio x, defined as the number of accepted transitions divided by the number of proposed transitions, is less than a given value x_0 (in [21], x_0 = 0.8), T_0 is doubled. This procedure continues until the acceptance ratio exceeds x_0. Experimental evaluation suggests that x_0 = 0.8 and a T_0 equal to the similarity of the initial solution, is the best combination for the initial value of T. For decreasing the value of T, we apply the common (e.g., [19]) decrement rule: $T_{k+1} = a * T_k$, where a = 0.95.

The length of the inner while-loop is determined by the equilibrium condition. For a given value of T, an *equilibrium* is reached if all the neighbors of a solution S, have the same similarity with S. This parameter is, in general, the most complicated to adjust because it is closely related to the specific problem. We experimented using several queries with various sizes, over multiple datasets. The following formula provides a suitable value for the number of iterations:

$$\log(P(N,n)) = \log\left(\frac{N!}{(N-n)!}\right)$$

The logarithm of the number of solutions has also been used in the constraint satisfaction literature as a measure of the problem size [6].

5. EXPERIMENTAL EVALUATION

In order to evaluate performance, we constructed five sets of 20 queries each using the resolution scheme of Figure 1. The number of variables in each set was fixed to 3, 6, 9, 12 and 15. Query tightness varied from complete queries (where all pairs of variables are constrained as in Figure 2) to very loose ones involving only a few non-restrictive constraints. The value of δ was set to 1% of the global extent per axis. We used the three 2D datasets in Figure 5; the first one contains randomly generated rectangles according to a uniform distribution, while the second contains a VLSI circuit, and the third one road segments of Greece. Notice that the density (sum of all rectangle areas divided by the workspace) and distribution of the objects significantly affects the performance of algorithms since it determines the quality of solutions. For instance, queries involving constraints such as *overlap*, *inside* etc. are more easily satisfied in the second dataset due to its high density. Heuristic search is especially sensitive to the number of solutions [6]; if there exist only a few good solutions (e.g., for some restrictive large queries) it requires a significant amount of time to find them. The above datasets cover a wide range of cardinality values, data densities and distributions; thus they provide a good estimation for the performance of the algorithms on other domains.

As a benchmark for systematic search we used *forward checking* (FC) [18], because it is considered one of the most effective algorithms for general CSP problems [4], as well as for structural queries [26][27]. The current implementation of FC works in a *branch and bound* manner: (i) in case the user inputs a target similarity to be retrieved, instantiations are abandoned as soon as they cannot lead to solutions of similarity equal or higher than the target; (ii) if the user just wants the best K solutions with no similarity threshold, the target is always set as the similarity of the current K^{th} solution. In this way unsuccessful instantiations are rejected early and the search space is pruned effectively. We also compare performance with an algorithm (RND) that chooses solutions randomly and keeps the best ones. The experiments were run on a SUN UltraSparc2 (200MHz) with 256MB of RAM.

The first set of experiments measures the CPU time in milliseconds required to find one solution with similarity above a target of 0.7, 0.75 and 0.8. Each execution was allowed 400 seconds to complete; after this period it was terminated. Figure 5 illustrates the results for every query size/dataset combination (each row corresponds to one query size and each column to one dataset). CSII and CSSA clearly outperform the other algorithms for all cases, with CSII being the best option. Moreover, these two algorithms were the only ones to successfully terminate for all combinations; FC and RND exceeded the time threshold in most large queries, and their results are omitted from most graphs. RND, in general, outperformed FC; due to its simple implementation, it checks more instantiations per second than the other algorithms. GCSA had, on the average, slightly better performance than RND, but in comparison to CSII and CSSA it requires more time to reach a solution above the target similarity.

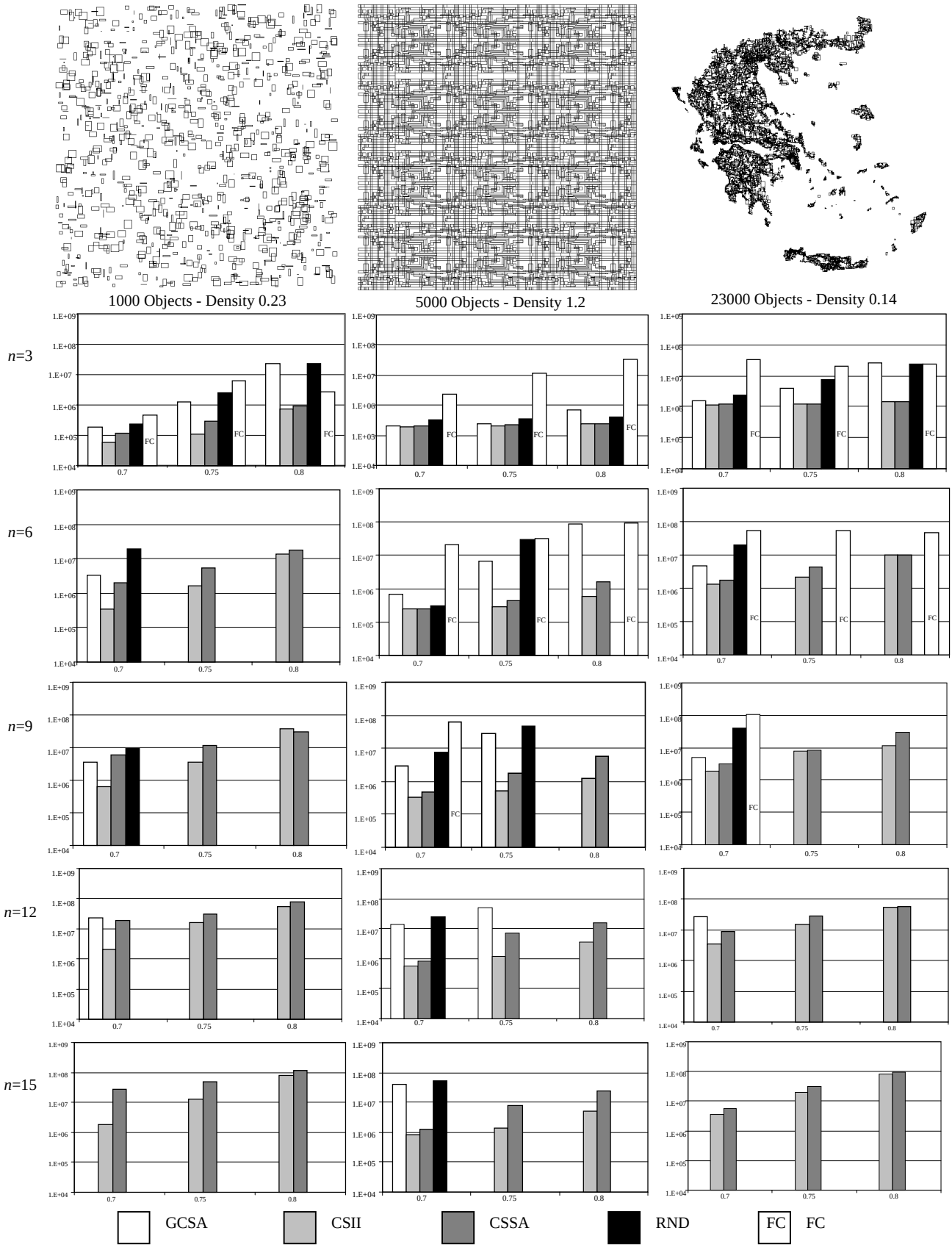
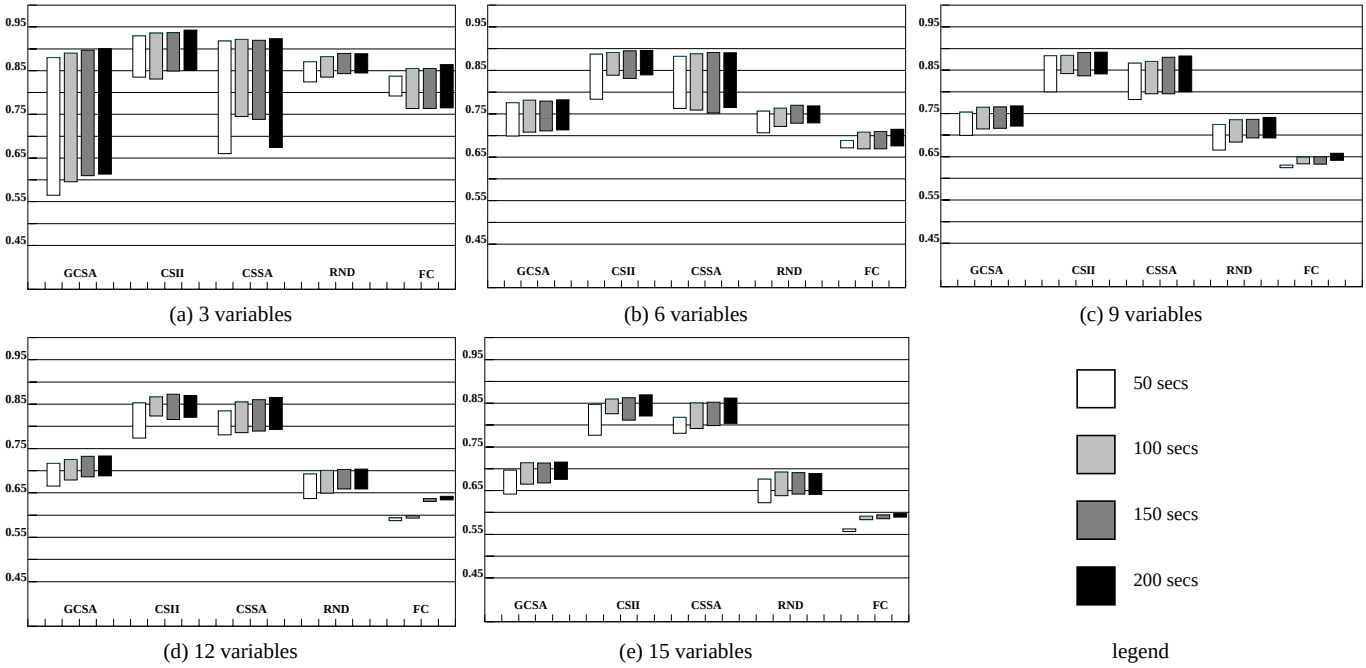


Figure 5 Time (msec) required to retrieve a solution with a given target similarity



(a) 3 variables (b) 6 variables (c) 9 variables

(d) 12 variables (e) 15 variables legend

Figure 6 Similarity range of 50 best solutions for pre-determined execution time

The performance of all algorithms degrades as the query size increases because large queries have, in general, few good solutions and a large part of the space has to be searched before they are found. The algorithms are most effective in the second dataset due to the high number of solutions, especially for small queries (notice that RND always finds solutions for target similarity 0.7).

The next set of experiments measures the similarity of the best 50 solutions retrieved by the algorithms (K=50) as a function of the execution time (50, 100, 150 and 200 seconds). Each diagram in Figure 6 corresponds to a different query size and shows the similarity ranges of all solutions averaged over the three datasets of Figure 5. In other words, the lowest (highest) value represents the average of all lowest (highest) similarities for queries of the given size in any dataset.

As expected, CSII and CSSA again outperform the other algorithms. The greater range of similarity values for CSSA can be explained by the fact that it starts from a random solution, which tends to have low similarity and remains in this region, until the temperature is reduced significantly. CSII also starts from a solution with low similarity, but very soon reaches a region with high similarity because it accepts only better solutions. Therefore, it has a better performance for the current problem because it can reach very quickly a local maximum while SA spends the initial stages exploiting low similarity regions.

GCSA performs better than RND but the quality of retrieved solutions with respect to CSII and CSSA drops for large queries (where the number of good solutions is small). The wide range of similarities for queries with three variables can be explained by the fact that if a single instantiation changes (e.g., due to mutation), it significantly affects (up to 33%) the fitness of the solution. FC is acceptable only for queries involving three variables (Figure 6a) where there is enough time to search a good part of the solution space. Its performance deteriorates significantly with the query size; notice that for large queries all solutions retrieved are in a narrow, low similarity range. This is explained by the fact that in restricted time periods, FC will find an area of the search space where some constraints are partially satisfied (while the rest totally violated) and retrieve all 50 solutions in this area. Most of these solutions will have the same instantiations for the partially satisfied constraints and differ only on the remaining variables.

6. CONCLUSIONS

This paper applies heuristic search algorithms in order to process structural queries. We develop three techniques based on genetic algorithms, iterative improvement and simulated annealing, and compare them against forward checking, a very effective systematic search algorithm, and random search. Extensive experimentation, with various query/dataset combinations, shows that heuristic search is the best way to process configuration similarity in cases where a near optimal solution is needed in restricted time.

The proposed methods have a wide range of applications in most modern spatial/multimedia database systems, which are increasingly vector-based, as well as the upcoming image/video compression methods (MPEG4). For the case of MPEG4, an object-oriented compression standard, [12] proposes an extension to the standard's specifications, in order to support an efficient way of indexing video objects. In addition, some query languages such as Query-by-Sketch [10] and VisualSeek [29] already provide facilities for the expression of structural queries.

In the future we plan to apply alternative search methods and combinations. For instance, we could first employ GCSA to find a set of widely spread solutions with relatively high similarities and use these solutions as the starting points for CSII. Another heuristic, which is expected to be very efficient, is based on *conflict minimization*: find the variable whose instantiation results in the highest degree of inconsistency, and re-assign it so that fitness is maximized.

Furthermore, in our implementation we don't use any indexing for the input datasets. The application of multi-dimensional data structures, such as R-trees, may improve the performance of heuristic search as it does for systematic algorithms [25]. In this way, the proposed algorithms will be applicable in domains where the number of objects is very large (e.g., 10^5 or 10^6).

ACKNOWLEDGMENTS

This work was supported by grant HKUST 6151/98E from Hong Kong RGC and grant DAG97/ 98.EG02. We would like to thank Jack Lee and Thanasis Loukopoulos for their comments on the paper.

REFERENCES

[1] Allen, J., "Maintaining Knowledge About Temporal Intervals", CACM, 26(11), 1983.

[2] Bach J., Fuller C., Gupta A., Hampapur A., Horowitz B., Humphrey R., Jain R. "The Virage Image Search Engine: An Open framework for Image Management". SPIE, Storage and Retrieval for Still Image and Video Databases, vol. 2670, pp. 76-87, 1996.

[3] Bergman, L. Castelli, V., Li C-S. "Progressive Content-Based Retrieval from Satellite Image Archives". D-Lib Magazine, October 1997.

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

[5] Cerny V. "Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm". J. Opt. Theory Appl., 45, 41-51, 1985.

[6] Clark, D., Frank, J., Gent, I., MacIntyre, E., Tomov, N., Walsh, T. "Local Search and the Number of Solutions". International Conference on Principles and Practice of Constraint Programming, 1998.

[7] Chang S.F, Smith J.R., Meng H.J, Wang H., Zhong D. "Finding Images/Video in Large Archives". CNRI Digital Library Magazine, Feb. 1997.

[8] Delis, V, Papadias, D., Mamoulis, N. "Assessing Multimedia Similarity: A Framework for Structure and Motion". ACM Multimedia, 1998.

[9] DeJong, K., Spears, W. "An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms". 1st Workshop on Parallel Problem Solving from Nature, Springer-Verlag, Berlin 1990.

[10] Egenhofer, M. "Query Processing in Spatial-Query-by-Sketch". Journal of Visual Languages and Computing, Vol. 8, 403-424, 1997.

[11] Freksa, C., "Temporal Reasoning based on Semi Intervals". Artificial Intelligence, Vol 54, pp. 199-227, 1992.

[12] Ferman A.M., Gunsel B., Tekalp A.M., "Object-Based Indexing of MPEG4 Compressed Video". Visual Communications and Image Processing, SPIE 1997, Vol. 3024, pp. 953-963.

[13] Grefenstette, J. "Optimization of Control Parameters for Genetic Algorithms". IEEE Trans. Systems, Man and Cybernetics, Vol.SMC-16, No.1, pp. 122-128, 1986.

[14] Goldberg, D.E. "Genetic Algorithms in Search, Optimization and Machine Learning". Addison-Wesley, Reading, Mass., 1989.

[15] Gupta A., Jain R., "Visual Information Retrieval". Communications of ACM, Vol. 40, No. 5, 70-79, May 1997.

[16] Gudivada, V., Raghavan, V. "Design and Evaluation of Algorithms for Image Retrieval by Spatial Similarity". ACM Transactions on Information Systems, 13(1):115-144, 1995.

[17] Holland, J.H. "Adaptation in Natural and Artificial Systems". University of Michigan Press, Ann Arbor, Mich., 1975.

[18] Haralick R.M., Elliot G.L. "Increasing Tree Search Efficiency for Constraint Satisfaction Problems". Artificial Intelligence, vol. 14, pp. 263-313, 1980.

[19] Ioannidis, Y., Kang, Y. "Randomized Algorithms for Optimizing Large Join Queries". ACM SIGMOD, 1990.

[20] Johnson, D., Aragon, C., McGeoch, L., and Schevon, C. "Optimization by Simulated Annealing: An Experimental Evaluation (Part I)". Operations Research, 37, 865-892, 1989.

[21] Kirkpatrick S., Gelat, C., Vecchi, M. "Optimization by Simulated Annealing". Science, 220, 671-680, 1983.

[22] Niblack W. Barber R., Equitz W., Flickner M., Glasman E., Petkovic D., Yanker P., Faloutsos C., Taubin G. "The QBIC Project: Querying Images by Content Using Color Texture, and Shape". SPIE, vol. 1908, Storage Retrieval for Image and Video Databases, pp. 173-187, 1993.

[23] Nabil, M., Ngu, A., Shepherd, J. "Picture Similarity Retrieval using 2d Projection Interval Representation". IEEE TKDE, 8(4), pp. 533-539, 1996.

[24] Nahar, S., Sahni, S., Shragowitz, E. "Simulated Annealing and Combinatorial Optimization". 23rd Design Automation Conference, pp. 293-299, 1986.

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

[26] Papadias, D., Mamoulis, N., Meretakis, D. "Image Similarity Retrieval by Spatial Constraints". ACM CIKM, 1998.

[27] Papadias, D., Karacapilidis, N., Arkoumanis, D. "Processing Fuzzy Spatial Queries: A Configuration Similarity Approach". International Journal of Geographic Information Science Vol.13(2), pp. 93-118, 1999.

[28] Pentland A., Picard R., Sclaroff S. "Photobook: Tools for Content-Based Manipulation of Image Databases". SPIE, vol. 2185, Storage and Retrieval for Image and Video Databases, pp.34-47, 1994.

[29] Smith J., Chang S.-F. "VisualSeek: A fully automated content-based image query system". International Conference on Image Processing, Lausanne, Switzerland, 1996.

[30] Srinivas, M., Patnaik, L.M. "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms". IEEE Trans. Systems, Man and Cybernetics, vol. 24(4), 656-667, 1994.