Fast Retrieval of Similar Configurations

Dimitris Papadias, Marios Mantzourogiannis, and Ishfaq Ahmad, *Member, IEEE*

***Abstract*—Configuration similarity is a special form of content-based image retrieval that considers relative object locations. It can be used as a standalone method, or to complement retrieval based on visual or semantic features. The corresponding queries ask for sets of objects that 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." Exhaustive processing (i.e., retrieval of the best solutions) of configuration similarity queries, in general, has exponential complexity and fast search for sub-optimal solutions is the only way to deal with the vast amounts of multimedia information in several real-time applications. In this paper we first discuss the utilization of nonsystematic search heuristics, based on genetic algorithms, simulated annealing and hill climbing approaches. An extensive experimentation with real and synthetic datasets reveals that hill climbing techniques are the best for the current problem; therefore, as a subsequent step we study the search space, and develop improved variations of hill climbing that take advantage of the special structure of the problem to enhance speed. The proposed heuristic methods significantly outperform systematic search when there is only limited time for query processing.**

***Index Terms*—Content-based retrieval, local search algorithms, spatial similarity.**

I. Introduction

The large availability of visual content in emerging multimedia applications and the worldwide web (WWW) has triggered significant advances in content-based retrieval mechanisms. Such mechanisms, sometimes in conjunction with traditional information retrieval techniques for text, allow users to access a variety of information sources. A special form of content-based retrieval is *configuration* similarity, also known as *spatial*, *structural*, or *arrangement* similarity. The corresponding queries describe some prototype configuration and the goal is to retrieve all images containing arrangements of objects matching the input exactly or approximately. As an example consider that the user is looking for all images (video frames, html pages, VLSI circuits) containing arrangements similar to those of Fig.1(a). Such a query could be expressed by one of the existing pictorial languages that permit configuration similarity retrieval, e.g., *VisualSeek* [34], *Query by Sketch* [8], *PQBE* [29], *Safe* [35], or extended SQL commands, e.g., Select v_0, v_1, v_2, v_3, From ImageDB, Where NE(v_0, v_1), NW(v_0, v_2), N(v_0, v_3), ... (NE means *northeast*, NW *northwest*, and so on).

Formally, a configuration similarity query can be described by 1) a set of n variables, $v_0, v_1, \ldots, v_{n-1}$ that appear in the query; 2) for each variable v_i, a finite domain $D_i = \{u_0, \ldots, u_{Ni-1}\}$ of N_i values; and 3) for each pair of variables (v_i, v_j), a constraint C_{ij} which can be a simple spatio-temporal relation or a disjunction of relations. The example query contains four variables $(v_0, \ldots, v_3)$, one for every drawn object. The domain of each variable consists of the objects in the image(s) to be searched for the particular configuration. The input constraints restrict the possible assignments of variables to subsets of the domains. In addition to binary spatio-temporal relations, some query languages allow the user to specify unary constraints in the form of object properties at the feature (v_0 is a red square) or the semantic level (v_0 is a building). In this case, appropriate retrieval algorithms (e.g., for color matching) must be integrated with the ones for configuration similarity.

As in most forms of information retrieval, a scoring mechanism should be employed for inexact matches. Depending on the types of constraints allowed in the expression of queries, several types of similarity measures have been proposed. [22] uses Allen's [1] relations in multidimensional space and conceptual neighborhoods [10]. The idea is extended in [28] with the incorporation of binary string encoding to automate similarity calculations. Conceptual neighborhoods for topological relations (e.g., inside, overlap) are also applied in [8]. Reference [15] uses angular directions (e.g., northeast is defined as an angle of 45○) and fuzzy similarity measures. A related approach, which also includes distances between object centroids, is followed in [27].

Irrespective of the relations employed and the similarity measures used, the goal of query processing is to find instantiations of variables to image objects so that the input constraints are satisfied to a maximum degree. The *inconsistency degree* d_{ij} of a binary instantiation $\{v_i \leftarrow u_k, v_j \leftarrow u_l\}$ is defined as the dissimilarity between the relation $R(u_k, u_l)$ (between objects u_k and u_l in the image to be searched) and the constraint C_{ij} (between v_i and v_j in the query). Given the inconsistency degrees of binary constraints, the inconsistency degree $d(S)$ of a complete *solution* $S = \{v_0 \leftarrow u_p, \ldots, v_i \leftarrow u_k, \ldots, v_j \leftarrow u_l, \ldots, v_{n-1} \leftarrow u_r\}$ can be defined as

$$d(S) = \sum_{\forall i,j,i \neq j \ and \ 0 \leq i,j<n} d_{ij}\left(C_{ij}, R(u_k, u_l)\right) \quad \text{where } \{v_i \leftarrow u_k, v_j \leftarrow u_l\}. \tag{1}$$

Fig. 1(b) and (c) illustrate two solutions for the example query where $v_i \leftarrow u_i$, $0 \leq i < 4$. The first solution corresponds to a perfect match, while the second is inexact since some binary

Manuscript received May 11, 2000; revised December 12, 2001. This work was supported by Grants HKUST 6070/00E and HKUST 6081/01E from Hong Kong RGC. Portions of this paper were presented at the ACM SIGIR [25], [28]. The associate editor coordinating the review of this paper and approving it for publication was Prof. Shih-Fu Chang.

D. Papadias and M. Mantzourogiannis are with the Department of Computer Science, Hong Kong University of Science and Technology, Clearwater Bay, Hong Kong (e-mail: dimitris@cs.ust.hk; mantzour@cs.ust.hk).

I. Ahmad was with the Department of Computer Science, Hong Kong University of Science and Technology, Clearwater Bay, Hong Kong. He is now with the Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019 USA.

Digital Object Identifier 10.1109/TMM.2003.811629

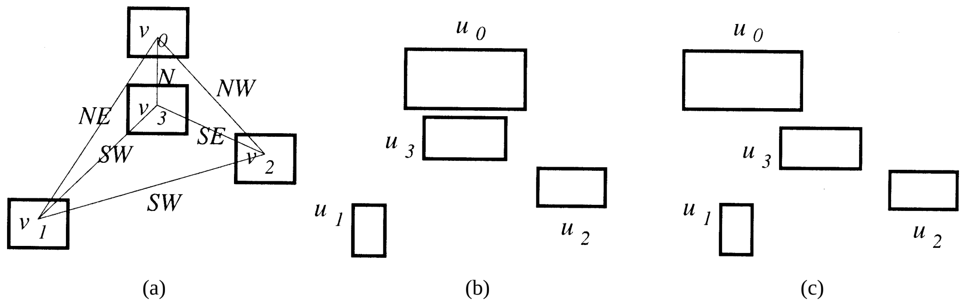


Fig. 1. Query example and solutions.

constraints (e.g., between v_0 and v_1) are not totally satisfied. If N is the image cardinality, the total number of possible solutions that have to be considered in each image is equal to the number of n-permutations of the N objects: $N!/(N-n)!$. Due to the high cost of query processing, it is not always possible to search all database images within a reasonable amount of time. In some cases, systematic search for the best solutions even in a single large image may take hours to complete [27].

An alternative approach is to compromise quality in order to achieve speed; in other words, we could assign a certain amount of processing to each image (possibly proportional to its size or importance) so that the whole database can be searched within the available time. In this paper we follow this approach and exploit nonsystematic search heuristics that can quickly provide good, but not necessarily optimal, solutions. The contributions of this work can be summarized as follows: a) we first apply three search methodologies, genetic algorithms, hill climbing and simulated annealing, to the problem of configuration similarity retrieval and identify the best one (hill climbing) using a variety of experimental settings; b) we perform a study of the solution space and evaluate alternative search strategies for hill climbing; and c) based on our study, we develop improved algorithms that take advantage of the spatial structure of the problem to enhance performance.

The rest of the paper is organized as follows. Section II outlines previous processing approaches and discusses their advantages and shortcomings. Section III describes the application of genetic algorithms, hill climbing and simulated annealing, to configuration similarity retrieval. Section IV compares systematic search with nonsystematic search, using experiments with both synthetic and real datasets. Section V, studies the problem space and proposes improved variations of hill climbing. Section VI concludes the paper with a discussion.

II. Query Processing Techniques

The problem of configuration similarity retrieval is similar to scene matching, which has been extensively studied in computer vision and pattern recognition [32], [33]. In the multimedia databases literature, several forms of processing configuration similarity, based on different assumptions and algorithms, have been proposed. The various approaches can be classified according to the size of database images for which they can be applied, the form of relations permitted, and the type of query variables. The form of relations, otherwise called *relation scheme*, can be *static* or *dynamic*. Static methods assume a predefined set of relations to be used by all users in all queries. Dynamic methods can be employed with any type of relations (assuming of course that the query language allows different sets of relations for different queries). Query variables can be *fixed* or *unrestricted*: a fixed variable can be instantiated to at most one object in each image, while an unrestricted one can range within the whole domain.

A class of methods, which can be grouped under a general category called *pairwise matching*, assumes that all query variables are fixed (e.g., find all images in which George is left of Mary). Thus, an image has at most one configuration matching the query which can be found in polynomial time as follows: a) locate the query objects in the image (possibly using an index on object id), b) for each object pair compute its similarity to the corresponding query constraint, and c) calculate the total similarity of the configuration using the pairwise similarities. [15] follows this approach to answer configuration similarity queries involving angular directions including rotation invariants. [22] deals with projection directions and topology. Algorithms that combine pairwise matching with contextual similarity (i.e., based on object features) can be found in [37]. Assuming that image objects are stored using absolute coordinates, pairwise matching can be applied with dynamic relation schemes. Its disadvantage is its limited applicability due to the fixed nature of query variables.

[30] solves configuration queries for medical images (X-rays) that contain a constant number of labeled/expected objects (e.g., stomach, heart) and a small number of unlabeled ones (e.g., tumors). Every image is mapped onto a point in multi-dimensional space, where each dimension corresponds to a relation between a specific pair of objects; i.e., if N is the number of image objects and r the number of relations in the relation scheme, the number of dimensions is $O(rN^2)$. Queries, which are also X-ray images containing mostly labeled (i.e., fixed) variables, are processed by multidimensional nearest neighbor search using R-trees. In order to keep the number of dimensions stable, images with unlabeled objects are decomposed into combinations of images with fixed size. An enhanced version that reduces the number of dimensions is proposed in [31]. Performance could be further improved by employing more efficient high dimensional indexing methods, such as M trees [6], the pyramid technique [2] etc. Nevertheless, the method (like all techniques based on high dimensional indexing and search) is applicable only for static relation schemes (otherwise it is not possible to pre-determine the dimensions) and databases with small images (fewer than

ten objects) of mainly labeled objects (otherwise, the number of dimensions and images explodes to unmanageable levels).

A number of methods are based on several variations of two-dimensional (2-D) strings, which encode the arrangement of objects on each dimension into sequential structures. $2DB$ strings capture the object projections, while $2DC$ and $2DG$ strings decompose objects in entities with disjoint convex hulls, allowing the representation of more detailed spatial information at the expense of storage [3], [19], [20]. Every database image is indexed by a 2-D string, and queries are also transformed to 2-D strings while configuration similarity retrieval is performed by applying appropriate string matching algorithms [4]. If the query contains only fixed variables, the cost of processing each image is polynomial, while in the general case it is exponential since matching has to be performed for multiple instantiations of the variables to different image objects. Users are not allowed to define their own relation scheme, but are restricted to the relations captured by the 2-D strings.

[27] deals with configuration similarity without any restriction on the type of variables or relations. Approximate retrieval is modeled and solved as a constraint satisfaction problem by applying branch and bound algorithms that stop searching once a partial solution cannot lead to a desired target. The method is applicable for images of 10^2–10^3 objects and can be employed with variable relation schemes. In [26], the incorporation of spatial indexing (R-trees) enables retrieval from much larger images (10^4–10^5 objects). Although this approach works well in most cases, systematic search algorithms do not have a predictable behavior depending on the problem size. Different query/image combinations, even with the same number of variables and image objects, may yield vast variances in cost depending on *constrainedness* [12]. For instance, the running time for the same query in two images of the same size may be orders of magnitude different. As a consequence, a large part of query processing may be devoted to a few images, while other images may not be searched at all within the available time.

III. Heuristic Search for Configuration Similarity

Consider a database with numerous, large images where users can ask any type of queries (i.e., with nonfixed variables) using variable relation schemes. The only approach that could be employed here is systematic search [26], [27], which due to the worst-case exponential cost is not guaranteed to terminate within reasonable time. In order to deal with configuration similarity under limited time, we employ search heuristics based on genetic algorithms, hill climbing and simulated annealing, which are explained as follows.

A. Genetic Algorithms

Genetic algorithms [13] are based on the concepts of natural mutation and the survival of the fittest individuals. Given a well-defined search space, three different genetic operations, *selection*, *crossover* and *mutation*, are applied to transform an initial population of chromosomes to next generation 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, on which the genetic algorithm performs 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/solution is simply an array S of n values, where $S[\mathrm{i}]$ 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* [36]: a solution 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 remaining offspring. Thus, we restrict randomness to the fractional parts only and assure that a good chromosome will not vanish.

Crossover mechanism is the driving force in the *exploration* part of a genetic algorithm. In the simplest approach, 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 two newly generated chromosomes are very likely to possess the good characteristics of their parents (*building-block hypothesis* [13]). In our case this corresponds to swapping of the assignments in two solutions after a selected point. One-point crossover is inefficient for our application domain, since the probability of a bit to be swapped increases as we move 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 entire 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. The option of specifying a *target* fitness also exists (i.e., retrieve

[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.

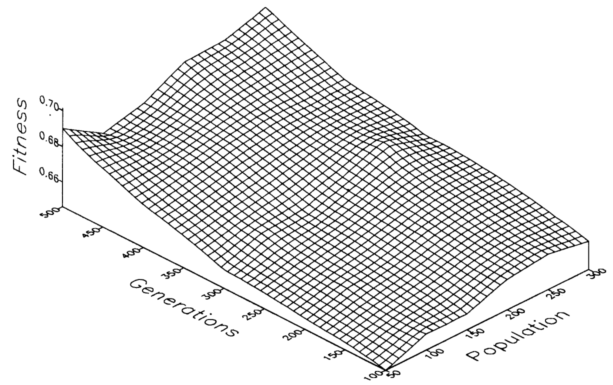


Fig. 2. Fitness (similarity) as a function of P and G in GCSA.

the best K solutions where the similarity is greater than *target*), in which case only chromosomes that exceed the target are kept at each run of GCSA.

Several theoretical and empirical studies [14], [36] have been carried out on the control parameters, P, G, μ_c and μ_m. Most results 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 the queries and datasets described in Section IV. The best results for most cases were achieved for $\mu_m = 0.05\%$ and $\mu_c = 0.60$. We measured the average similarity of the best solutions (for all combinations of queries and datasets) found by GCSA for several values of P in the range 50–300 and G in the range 500–100 (as P increases, G has to decrease in order to keep the execution time constant). Fig. 2 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 a sufficient number of generations even for large problem instances. Combined with the relatively high value of μ_c (0.60) GCSA was able to exploit a large portion of the solution space.

B. Hill Climbing

The problem space for configuration queries can be thought of as a graph, where each solution corresponds to a node associated with a similarity value. The 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, where n is the number of query variables and N the image cardinality (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.

Configuration similarity iterative improvement (CSII) starts with a randomly chosen initial solution (*seed*) and tries to find a better one by visiting random neighbors. If such a solution is found, it replaces the previous one. The process continues until a local maximum is reached. This iterative optimization is repeated a number of times, each time starting from a different seed. 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 [23]. However, given a finite amount of time, the algorithm terminates at a local maximum.

```
CSSA(target)
$S = S_0$; $T = T_0$;//S is initialized to random solution
WHILE (not stopping criterion) {
    WHILE ( not equilibrium) { //level
        $S'$ = random neighbor of $S$;
        IF (similarity($S'$)>target) THEN store($S'$);
            $D_f$= similarity($S'$) – similarity($S$);
          IF ($D_f$ >= 0) THEN $S = S'$;
          ELSE  IF (random[0,1) < exp($D_f/T$) ) THEN  $S = S'$;
    }  //end level
  reduce $T$;
} //end while
```

Fig. 3. Outline of CSSA.

C. Simulated Annealing

Configuration similarity simulated annealing (CSSA), based on [5], [18], also performs random walks, but in addition to uphill, it also accepts downhill moves with a certain probability. The intuition behind accepting downhill moves is led by the fact that some local maxima may be close to each other, separated by a small number of downhill moves. If only uphill moves were accepted (as in CSII) the algorithm would stop at the first local maximum visited, missing a subsequent (and possibly better) one. Fig. 3 illustrates CSSA for the case where the user requires solutions exceeding the similarity specified by *target*.

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 neighbor 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 at 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 zero. 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 GCSA, the quality of the output is strongly related to the choice parameter values. In order to define the initial value T we adopt the method of [18]: a large value for T_0 is chosen and a number of transitions are 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 [18], $x_0 = 0.8$), 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., [17]) 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:

$$\log(P(N,n)) = \log\left(\frac{N!}{(N-n)!}\right) \tag{2}$$

where $P(N,n)$ is the number of n-permutations of the N objects (i.e., the number of possible solutions). (2) has been widely used in the constraint satisfaction literature as a measure of the problem size [12].

IV. Experimental Evaluation of Search Heuristics

In order to evaluate performance, we constructed five sets, each containing 25 queries using the relation scheme and similarity measures[2] of [7]. The number of variables in each set was fixed to three, six, nine, 12, and 15. Query tightness varied from complete queries (where all pairs of variables are constrained) to very loose ones involving only a few nonrestrictive constraints. We used the three 2-D datasets in Fig. 4; the first set contains randomly generated rectangles according to a uniform distribution, while the second set contains a VLSI circuit, and the third set contains road segments of Greece. Note 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 [12]; 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 most problem instances.

As a benchmark for systematic search we used *forward checking* (FC) [16], because it is considered to be one of the most effective algorithms for general CSP problems, as well as for configuration similarity [26], [27]. The current implementation of FC works in a *branch and bound* manner, i.e., a partial solution is abandoned if it cannot lead to similarity equal or higher than the best already found. In this way unsuccessful instantiations are rejected early and the search space is pruned effectively. We also compare performance with random sampling (RND), which chooses solutions randomly and keeps the best ones—for some optimization problems, random sampling outperforms other search heuristics due to its simplicity [11]. The experiments were run on a SUN UltraSparc2 (200 MHz) with 256 MB of RAM.

The first set of experiments measures the CPU time (in milliseconds) required to find one solution with similarity[3] above a target of 0.7, 0.75 and 0.8. Each execution was allowed 400 s to complete; after this period it was terminated. Fig. 4 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 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. This is because RND, due to its simple implementation, checks more instantiations per second than the other algorithms. GCSA, on the average, yields slightly better performance than RND, but in comparison to CSII and CSSA requires more time to reach a solution above the target similarity.

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. The algorithms are most effective in the second dataset due to the presence of a high number of solutions, especially for small queries (note that RND always finds solutions for target similarity 0.7). Obviously, if the available processing time increases, FC will eventually outperform nonsystematic search—but for the current, rather long (400 s) processing limit, it is inefficient.

The next set of experiments measures the similarity of the best 50 solutions retrieved by the algorithms as a function of the execution time (50, 100, 150, and 200 s). Each diagram in Fig. 5 corresponds to a different query size and shows the similarity ranges of the 50 solutions averaged over the three datasets of Fig. 4. 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

[2]The algorithms can be employed with any type of spatial constraints. Due to the absence of a standard benchmark, we use the relation scheme in [7] since a) it is general, in the sense that it can express all types of common constraints (i.e., direction, topological and distance), b) it permits the automatic calculation of similarity measures, and c) it has been used for systematic search algorithms [26].

[3]Similarity values range between 0 and 1. Some queries (especially large ones) do not have perfect matches.

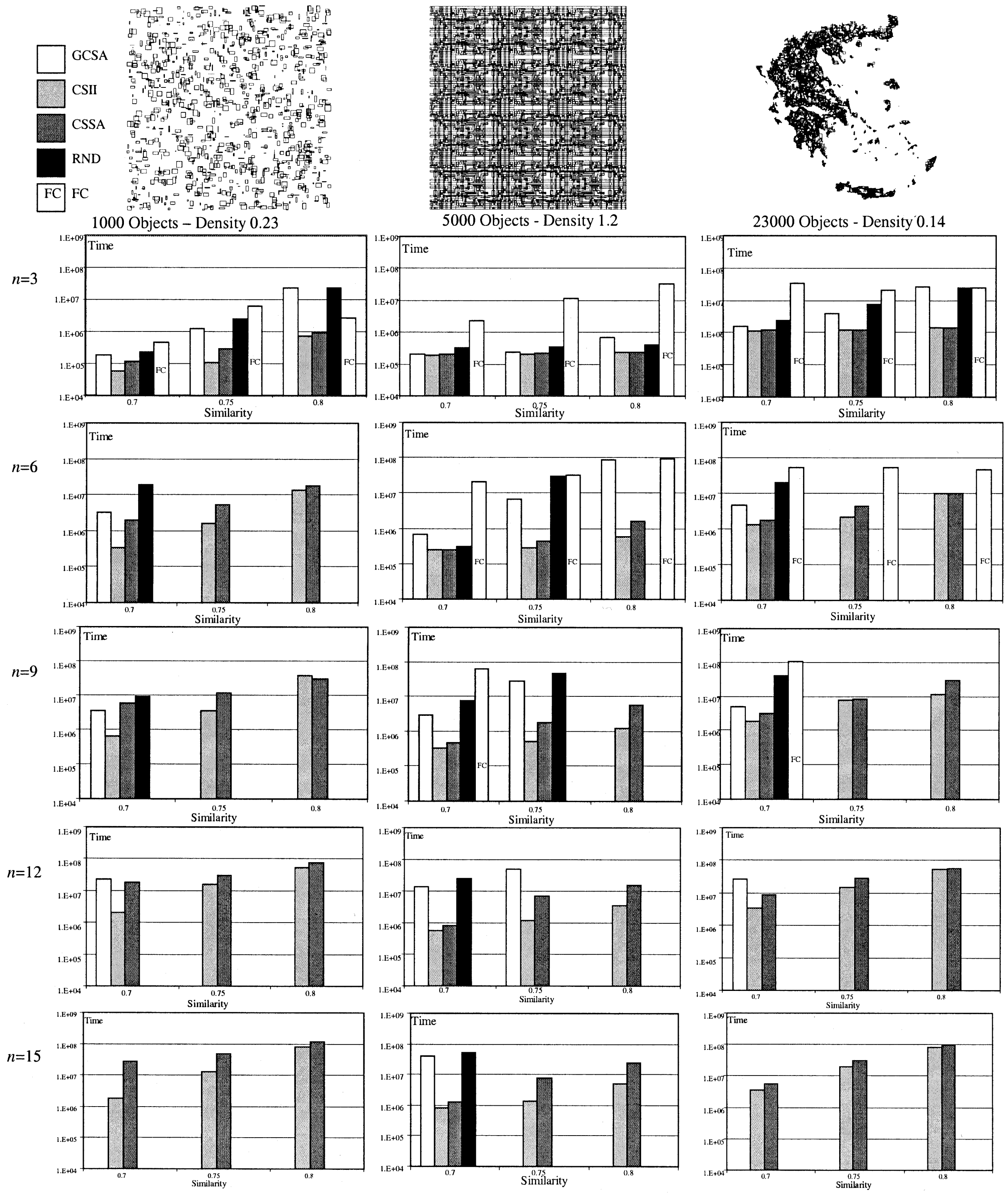


Fig. 4. Time (in milliseconds) required to retrieve a solution with a given target similarity.

where there is enough time to search a good part of the solution space. Its performance deteriorates significantly with the query size. It can be observed that that for large queries all solutions retrieved are in a narrow and low similarity range. This is because 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 tend to have the same instantiations for the partially satisfied constraints and differ only on the remaining variables.

In summary, nonsystematic clearly outperforms systematic search when the processing time is limited with respect to the problem size. Among the heuristics tested, CSII yields the best performance. Furthermore, it does not require parameter tuning. This is a significant advantage because tuning of GCSA and CSSA is rather complicated. Moreover, the parameter values

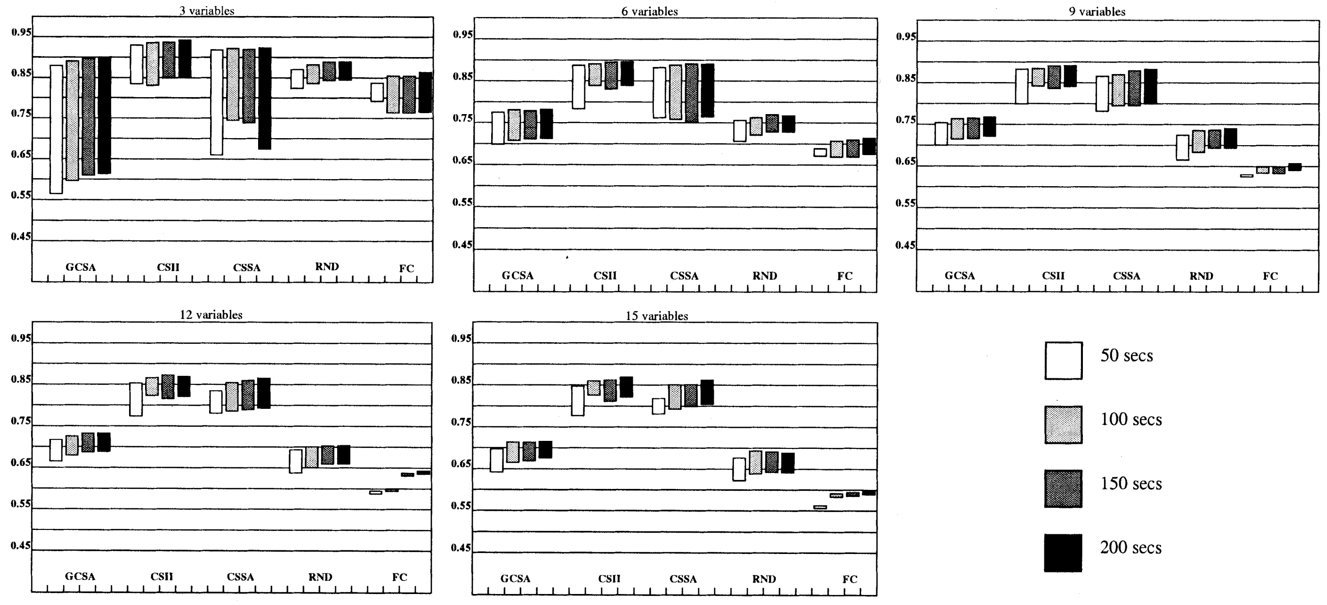


Fig. 5. Similarity range of 50 best solutions for predetermined execution time.

depend on the query and dataset characteristics (i.e., a set of values that yield optimal results for a query/image combination, may provide very poor results for another combination). Subsequently, we further improve the performance of hill climbing by proposing alternative search strategies.

V. Improved Hill Climbing Algorithms

CSII generates neighbors by selecting a random variable and changing its instantiation. An alternative approach, motivated by conflict minimization algorithms [21] is to select the "worst" variable. The inconsistency degree of a variable v_i (currently instantiated to value u_k) in a solution S is defined as the sum of inconsistency degrees of all binary instantiations involving v_i:

$$d(v_i, S) = \sum_{\forall j, i \neq j \ and \ 0 \leq j < n} d_{ij}\left(C_{ij}, R(u_k, u_l)\right) \quad \text{where } \{v_j \leftarrow u_l\}. \tag{3}$$

Worst variable selection reinstantiates the variable with the highest inconsistency degree, so that the similarity of the specific solution may be increased significantly. If the worst variable cannot be improved, the second worst will be considered and so on. If one variable can be improved, the next step will consider again the new worst one; otherwise, if all variables are exhausted with no improvement, the current solution is considered to be a local maximum.

Once a variable is chosen for reinstantiation, CSII determines its new value by applying *first-better value selection*, i.e., by assigning values to the specific variable randomly, until a better instantiation is found. When the similarity of a solution is very low, first-better selection performs just a few attempts before it finds a better solution. As the quality increases, it becomes more difficult for the solution to be improved by random re-instantiations. If after N unsuccessful assignments no better neighbor is found, the solution is considered a local maximum. Notice, however, that due to the random nature of search, better neighbors may be missed since some instantiations are tried multiple times, whereas others not at all. Another option is *all-best value selection* (sometimes called *steepest ascent*), which systematically tries all possible values in the domain of the variable to be re-instantiated and assigns the one that results in the highest similarity.

A. Search Space Analysis

In order to comprehend the behavior of hill climbing under different search strategies, we first study the search space for configuration similarity. The goal is to identify how easily are local maxima found and how much they differ from random solutions. For the first experiment, we randomly selected five queries with nine variables, and five with 12, and for each query we generated 500 random solutions in a dataset of 1,000 uniformly distributed rectangles with density 0.5. Fig. 6 shows the *average* similarity of a solution and the *average maximum* and *minimum* similarities of the neighbors that can be reached with a single move. The similarity values are scaled, i.e., they are divided by the average maximum similarity found in each case. The x-axis represents the five different queries, with no specific significance in the placement. According to the diagrams, there is a considerable difference between the similarity of a solution and the maximum and minimum similarity of its neighbors. For queries of size 9, this difference is around 15%, while for queries involving 12 objects about 10%. Thus, even a single move can have a significant effect on the quality of the solution especially in small queries (because each variable has a higher contribution to the total similarity).

The second experiment studies the number of *steps*, i.e., uphill moves that must be performed in order to reach a local maximum. We use two approaches for identifying local maxima: i) in the first one we replace a solution with the best of all its neighbors and ii) in the second one we accept the first better neighbor found by changing the instantiations of random variables. We

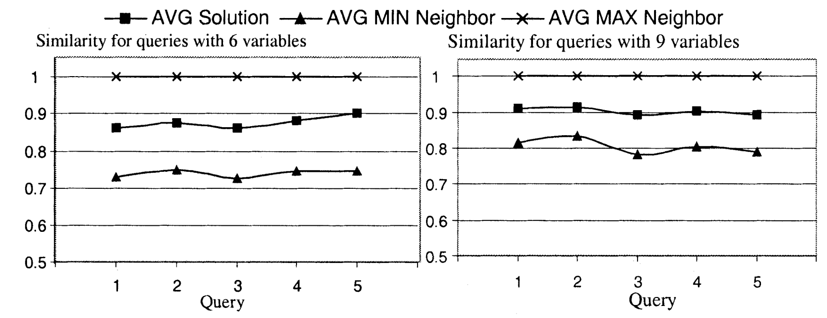


Fig. 6. Similarity ranges around random solutions.

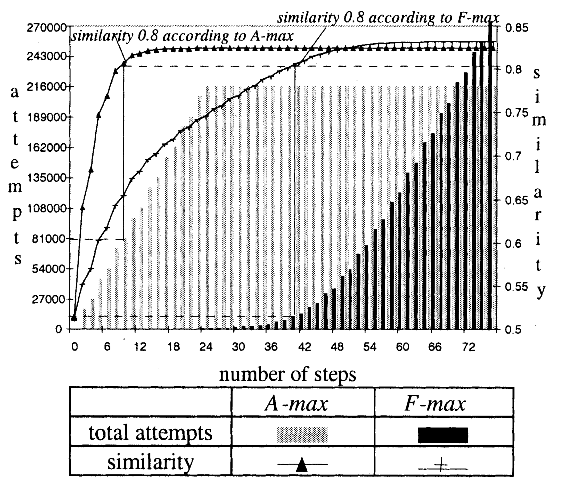


Fig. 7. Similarity and total number of attempts versus the number of steps.

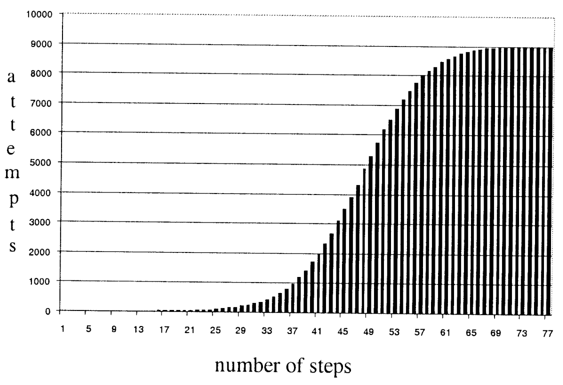


Fig. 8. Attempts per step for the F-max.

refer to the local maxima obtained using these approaches as *All-maximum* (*A-max* for short) and *First-maximum* (*F-max*) respectively. When searching for *A-max*, each step tries all possible values for each variable, i.e., a total of $\Theta(n \cdot N)$ attempts. For *F-max* this number differs in each step; in the best case an uphill move can be found with the first attempt, while in the worst, even $O(n \cdot N)$ attempts may fail to find a better neighbor (some instantiations may be missed).

Fig. 7 shows how these maxima are reached (attempts, similarity) as a function of the number of steps, for a query with nine variables over the 0.5 density dataset ($n = 9$, $N = 10^3$). The horizontal axis corresponds to the number of steps, the left y-axis to the total number of attempts (including unsuccessful instantiations) and the right y-axis to similarity. *A-max* (similarity 0.824) is reached after 24 steps; 9000 instantiations are tested in each step, resulting in 216 000 attempts. *F-max* (similarity 0.831) is reached after 77 steps and 273 408 attempts. Search for *A-max* is deterministic, meaning that starting with one seed, we always reach the same local maximum. On the other hand, the value of *F-max* and the steps required to reach it change depending on the order that neighbors are visited. In most cases the two maxima are close to each other.

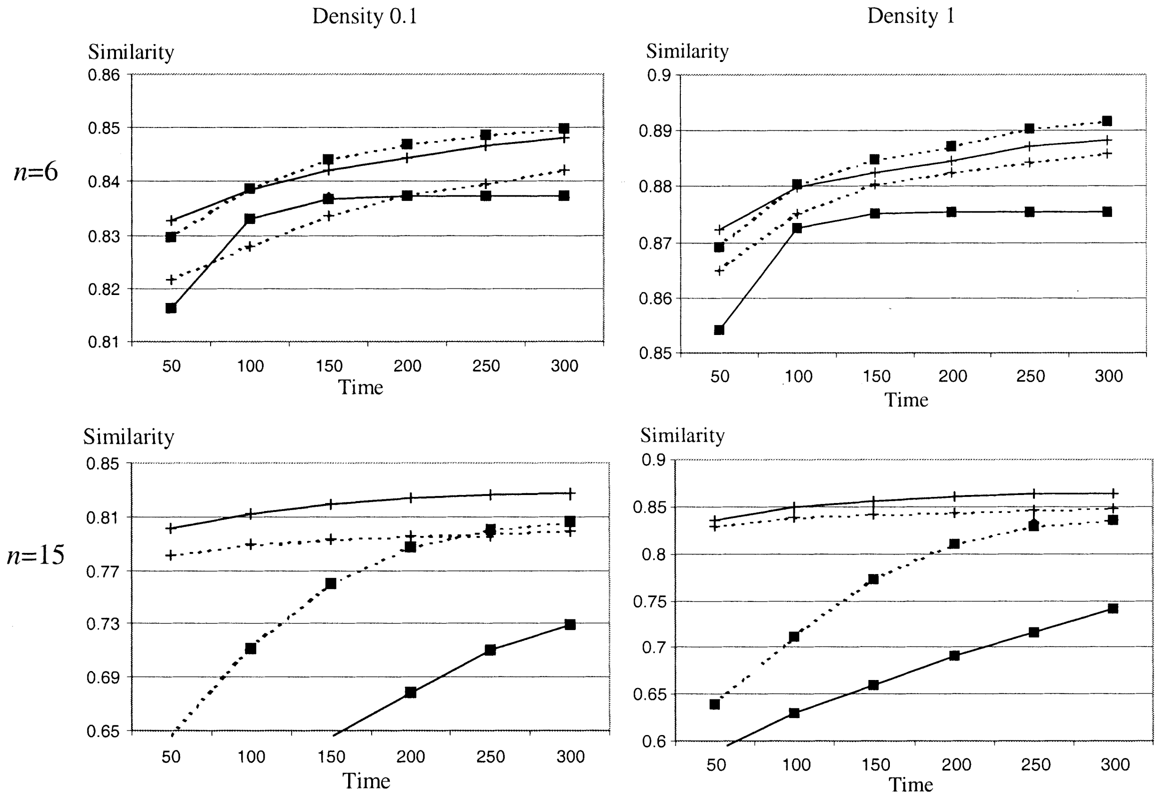
Although search for *F-max* finds the highest similarity using a longer path (77 steps as opposed to 24), it reaches high quality solutions faster. Consider, for instance, a solution with similarity around 0.8. If search is performed according to the *A-max* approach, the solution will be found after nine steps and 81 000 attempts (see Fig. 7). On the other hand, if the *F-max* approach is employed, the solution will be found in about 40 steps. However, the total number of attempts is lower than 15 000 because uphill moves are easily performed from solutions of low similarity. The average number of attempts per step when searching for *F-max* is shown Fig. 8. Up to the 37th step, fewer than 1000 instantiations are needed before each uphill move is found.[4] At that point the similarity is close to 0.8. Every attempt involves a similarity computation; thus the number of attempts (rather than steps) determines the cost of search. The advantage of the *F-max* search approach is clear, since it converges much faster to high similarity solutions.

B. Performance of Hill Climbing Alternatives

According to the previous results, first-better value is expected to outperform all-best selection. However, as the quality of the solution increases, improvement by random re-instantiations becomes more difficult and large parts of the domains are searched (see Fig. 8). Near local maxima, first-better behaves likes all-best value selection, but unlike exhaustive search it may miss some good neighbors. Consequently, in some cases where there is enough time for processing (small queries and/or datasets), all-best may eventually yield better solutions than first-better selection.

In order to test this observation we ran experiments with the four variations of hill climbing (2 variable selection · 2 value selection mechanisms) using the query sets of six and 15 variables

[4]The maximum number of attempts per step for this experiment is 9000, since there exist nine variables, each taking 1000 values.



value selection / *variable selection*	First-Better **F**	All-Best **A**
Random **R**	—+— R-F	—■— R-A
Worst **W**	···+··· W-F	···■··· W-A

Fig. 9. Similarity retrieved as a function of the execution time.

over datasets of 1000 uniformly distributed rectangles with densities of 0.1 and 1.[5] Fig. 9 shows the average similarity (of 25 queries in each set) retrieved over the two datasets every 50 s.

Observe that the relative performance of the algorithms is the same for both datasets, despite the large density difference, implying that their effectiveness is independent of density. First-better value selection quickly (within the first 50 s) finds good solutions even for the large queries. Among the two variable selection mechanisms, random selection (R-F) is faster[6] than worst variable (W-F) since random variables are more easily improved than the worst one. All-best value selection is ineffective for large queries because the number of neighbors, as well as the cost of similarity computations increases with the number of variables. Thus, W-A and R-A take a long time to converge to high similarity regions. Notice that for 15 variables W-A converges after 200 s, while R-A does not converge at all within the 300-s limit. R-A is worse than W-A, because some variables, especially if the solution is good, contribute little, or not at all, to the total degree of inconsistency. Therefore, spending a long time to improve these variables does not pay-off.

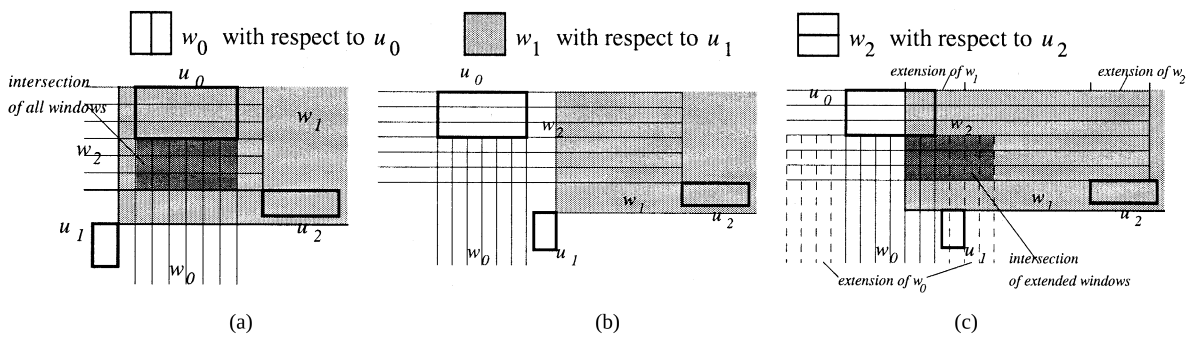
[5]Most real-life datasets have densities between 0.1 and 1, with a density around 0.5 considered common/average. The datasets used in this section are synthetic (so we can vary the densities) and relatively small (10^3 objects—so we can run numerous experiments). Nevertheless, similar results are obtained with the real datasets of Section IV.

[6]R-F corresponds to CSII algorithm.

For 6-variable queries, however, W-A (worst variable selection, all best value) outperforms CSII after 100 s and achieves the highest similarity. This is due to a combination of reasons: for small queries, finding the best possible instantiation for a variable may increase the similarity of the solution significantly, especially if the variable chosen is the worst one. Furthermore, due to the small problem size, there is enough time to search extensively within the neighborhood of a solution, identifying good local maxima. Motivated by these observations, we propose an algorithm that can outperform the previous ones in all cases. The idea is to start with CSII that quickly reaches an area of high similarity. In subsequent steps (when CSII starts behaving like exhaustive search), a deterministic value selection technique locates the good neighbors, using the spatial structure of the problem to avoid the expensive search for all possible variable instantiations.

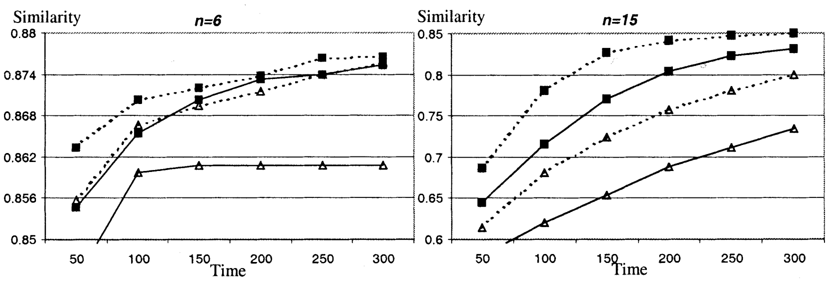
C. Window Value Selection

Consider the example query of Fig. 1(a), where the first three variables are instantiated to objects u_0, u_1, and u_2, as shown in Fig. 10(a). Assume that these three instantiations perfectly match the query constraints. The fourth variable (v_3) is chosen for re-instantiation and the goal is to find the best value for it. Variable v_3 is related with the other ones by the following projection-based constraints: south(v_3, v_0), northeast(v_3, v_1)



w_0 with respect to u_0
w_1 with respect to u_1
w_2 with respect to u_2
intersection of all windows
extension of w_1
extension of w_2
extension of w_0
intersection of extended windows
(a)
(b)
(c)

Fig. 10. Value selection using windows.



value selection / variable selection		All-Best A	Window WVS
Random	R	—△— R-A	- - -△- - - R-WVS
Worst	W	—■— W-A	- - -■- - - W-WVS

Fig. 11. WVS versus exhaustive search: similarity as a function of time.

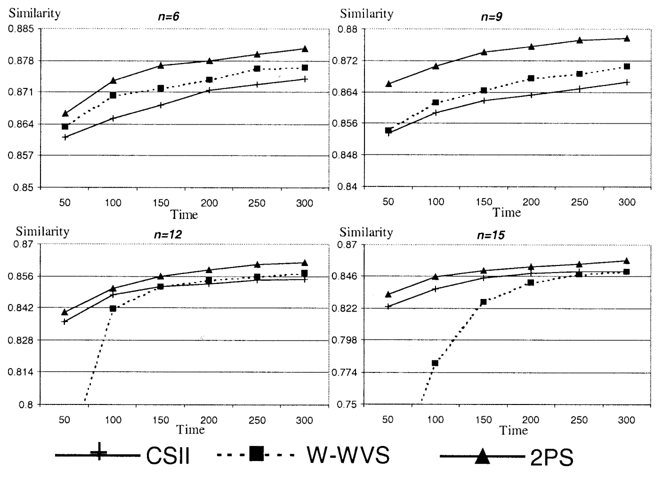


Fig. 12. 2PS versus WVS and CSII—Similarity as a function of time.

and northwest(v_3, v_2). Each of the constraints, in combination with the current value of the corresponding variable, defines a window in space containing all consistent values for v_3 (e.g., all objects south of v_0 are inside w_0). The best values of v_3 (i.e., satisfying all constraints) are the ones that lie in the intersection of all windows. In other words, if a value is found in the dark gray area of Fig. 10(a), there is no need to search the whole domain of v_3.

Although, in the example of Fig. 10(a), we assume that the first three variables are instantiated to objects that result in a perfect match, in most cases the partial solution after the removal of a single variable, is only approximate. As an example consider the partial solution of Fig. 10(b), where u_0 has been shifted to the left. The instantiation $\{v_0 \leftarrow u_0, v_1 \leftarrow u_1, v_2 \leftarrow u_2\}$ has some inconsistency degree on the x axis (the positions of the objects on the y axis are the same). As a result, the intersection of w_0, w_1 and w_2 is empty and therefore cannot contain any objects. Intuitively, the good instantiations for v_3, are still found somewhere in the area between u_0, u_1, and u_2. In order to continue improving the solution, we should extend the windows so that a new value for v_3 can be chosen.

Window value selection (WVS) applies this idea. Once the variable v_i for reinstantiation has been chosen, the appropriate windows w_j $(0 \leq j < n, i \neq j)$ are computed. Then each window is extended according to the maximum inconsistency degree d of the partial solution (where all variables except for v_i have been instantiated) on each dimension; the higher the value of d, the larger the extension on the corresponding axis. In the example of Fig. 10(c), w_0, w_1 and w_2 are only extended on the x-axis because there is no inconsistency on the y-axis. Although the objects in the intersection (dark gray area) do not result in perfect matches (e.g., the constraint between v_0 and v_1 is still violated), they provide good solutions that can be further improved in subsequent steps. The window extension method depends on the relation scheme in use. In the current implementation, which is based on conceptual neighbors [7], in addition to the original constraint, its d neighbors are taken into account when generating the window. If angular directions were used, a *northeast* constraint, for instance, could generate an angular window 40°–50° in case of a low value of d, or a window 30°–60° for higher inconsistency.

TABLE I
SIMILARITY AS A FUNCTION OF TIME RND AND FC

n	**Method**	**Time (seconds)**					
		50	**100**	**150**	**200**	**250**	**300**
6	RND	0.742521	0.749104	0.752542	0.755187	0.756688	0.758604
	FC	0.6475	0.654167	0.691563	0.698229	0.703313	0.708438
15	RND	0.665443	0.672321	0.678756	0.682307	0.683283	0.683923
	FC	0.566101	0.56692	0.56869	0.572113	0.575488	0.576732

In order to be able to search fast within such windows, all objects within an image are sorted according to the x-coordinate of the lower left point (this preprocessing takes place when the image is inserted in the database). The objects that fall inside the window (and potentially some false hits) are found by a simple range query in this sorted list. The other three coordinates of each retrieved object are checked and, if they also fall within the specified window, the object is kept as a good value. Initially, due to the low similarity of random solutions (seeds), the windows and their intersection usually cover the whole workspace. In this case WVS behaves like all-best value selection (with the additional overhead of computing the windows). As the inconsistency degree of the solution drops, the windows in some projections decrease restricting the search space.

We compare WVS against best-value selection (i.e., exhaustive domain search) for worst and random variable selection. Fig. 11 illustrates the results for query-sets (average of 25 queries per set) with six and 15 variables over the dataset with density 0.5 $(N = 10^3)$. WVS always outperforms exhaustive search; as in the case of all-best value selection (see experiments in Fig. 9), WVS performs best with worst variable selection (W-WVS) since each re-instantiation may improve the solution significantly.

D. *Two-Phase Search*

Despite its good performance, W-WVS does not converge fast to high similarity regions, due to the large windows in the initial phases of the algorithm. To circumvent this problem, we propose a *two-phase search* (2PS) algorithm that first uses CSII to quickly find a good solution, which is then improved by W-WVS. CSII is executed for a time analogous to the size of the problem (for this implementation $n \cdot N$ ms), and W-WVS for the remaining time. For instance, for a query with 15 variables over a 1000 objects dataset, the running time of CSII is 15 s. During this time CSII has performed enough steps to improve the seed significantly. Thus, in most cases the initial windows of W-WVS restrict the search to a relatively small portion of the space.

We compared 2PS with W-WVS and CSII using all query sets over the 0.5 density dataset $(N = 10^3)$. As shown in Fig. 12, 2PS outperforms both algorithms in all cases since it combines their best characteristics. Observe that for small queries (six and nine variables), W-WVS produces better solutions than CSII even at the first 50 s. As the query size increases, W-WVS slows down significantly and for 15 variables, it catches up with CSII only at 300 s.

We also repeated the same experiment using RND and FC. Table I shows the best similarity retrieved over time for the smallest (six variables) and largest (15 variables) query sizes. In general, both algorithms provide very low similarity when compared with the local search (see Fig. 12). RND produces better results than FC within the 300 seconds but quality does not increase much over time, implying that that only a small percentage of solutions have similarity close to a local maximum. Even though FC expectedly improves gradually with time it does not find good solutions even for six variables within the available limit. The situation is worse for 15 variables due to the significant increase of the search space; FC remains in the neighborhood of the initial assignments, which in most cases have low quality.

These results motivate the need for fast retrieval of suboptimal solutions in large multimedia databases. Among all techniques tested, 2PS consistently yielded the best performance for all combinations of queries/datasets (including real data). In addition to its robustness, another advantage of 2PS with respect to genetic algorithms and simulated annealing is that it does not require the complicated tuning of parameters that significantly affect efficiency.

VI. CONCLUSION

This paper applies nonsystematic search algorithms for processing configuration similarity queries. We first apply 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 an effective way to process configuration similarity in cases that a near optimal solution is needed in restricted time. The best performance is consistently achieved by CSII, which is based on hill climbing; thus, as a second step we try to enhance its efficiency by studying alternative variable and value selection mechanisms. Next we present window value selection, a technique that quickly identifies the best values of a variable while avoiding exhaustive search in its domain. Finally, the best overall algorithm is shown to be two-phase search, a method that first applies CSII to quickly find a good solution and then improves it by window value selection.

To our knowledge, currently there do not exist other methods that can solve arbitrary queries (with no restrictions on the size of the problem or the type of the variables), given a time limit. 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 such as MPEG-4. An efficient way of indexing MPEG-4 video objects is proposed in [9]. In addition, some query languages such as *Query-by-Sketch* [8] and *VisualSeek* [34] already provide facilities for the expression of configuration similarity queries.

In our implementation we do not use any indexing for the input datasets. The application of a multi-dimensional data structure, such as R-trees, may improve the performance of heuristic search as it does for systematic algorithms [26]. In this way, the proposed algorithms are applicable in domains where the number of objects is very large (e.g., in the order of 10^6). Future work can be carried out on the application of other sub-optimal algorithms (for a bibliography see [24]) on configuration similarity. In order to achieve efficiency, however, such algorithms need to be fine-tuned and modified for the particular, structure of the problem.

REFERENCES

[1] J. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[2] S. Berchtold, C. Boehm, and H. Kriegel, "The pyramid technique: toward breaking the curse of dimensionality," in *Proc. ACM SIGMOD*, 1998, pp. 142–153.

[3] S.-K. Chang, E. Jungert, and T. Li, "The design of pictorial databases based upon the theory of symbolic projections," in *Proc. 1st Symp. Large Spatial Databases*, 1989, pp. 303–323.

[4] S.-K. Chang, Q. Shi, and C. Yan, "Iconic indexing by 2-D string," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-9, no. 3, pp. 413–428, 1987.

[5] V. Cerny, "Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm," *J. Opt. Theor. Applicat.*, vol. 45, pp. 41–51, 1985.

[6] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: an efficient access method for similarity search in metric spaces," in *Proc. Very Large Databases Conf.*, 1997, pp. 426–435.

[7] V. Delis, D. Papadias, and N. Mamoulis, "Assessing multimedia similarity: a framework for structure and motion," in *Proc. ACM Multimedia*, 1998, pp. 333–338.

[8] M. Egenhofer, "Query processing in spatial-query-by-sketch," *J. Vis. Lang. Comput.*, vol. 8, pp. 403–424, 1997.

[9] A. Ferman, B. Gunsel, and A. Tekalp, "Object-based indexing of MPEG4 compressed video," *Proc. SPIE, Vis. Commun. Image Process.*, pp. 953–963, 1997.

[10] C. Freksa, "Temporal reasoning based on semi intervals," *Artif. Intell.*, vol. 54, pp. 199–227, 1992.

[11] C. Galindo-Legaria, A. Pellenkoft, and M. Kersten, "Fast, randomized join-order selection—why use transformations?," in *Proc. Very Large Databases Conf.*, 1994, pp. 85–95.

[12] I. Gent, E. MacIntyre, P. Prosser, and T. Walsh, "The constrainedness of search," in *Proc. AAAI*, 1996, pp. 246–252.

[13] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[14] J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, no. 1, pp. 122–128, 1986.

[15] V. Gudivada and V. Raghavan, "Design and evaluation of algorithms for image retrieval by spatial similarity," *ACM Trans. Inform. Syst.*, vol. 13, no. 1, pp. 115–144, 1995.

[16] R. Haralick and G. Elliot, "Increasing tree search efficiency for constraint satisfaction problems," *Artif. Intell.*, vol. 14, pp. 263–313, 1980.

[17] Y. Ioannidis and Y. Kang, "Randomized algorithms for optimizing large join queries," in *Proc. ACM SIGMOD*, 1990, pp. 312–321.

[18] S. Kirkpatrick, C. Gelat, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[19] S. Lee and F. Hsu, "Spatial reasoning and similarity retrieval of images using 2D C-strings knowledge representation," *Pattern Recognit.*, vol. 25, no. 3, pp. 305–318, 1992.

[20] S. Lee, M. Yang, and J. Chen, "Signature file as a spatial filter for iconic image database," *J. Vis. Lang. Comput.*, vol. 3, pp. 373–397, 1992.

[21] S. Minton, M. Johnston, A. Philips, and P. Laird, "Minimizing conflicts: a heuristic method for constraint satisfaction and scheduling problems," *Artif. Intell.*, vol. 58, pp. 161–205, 1992.

[22] M. Nabil, A. Ngu, and J. Shepherd, "Picture similarity retrieval using 2d projection interval representation," *IEEE Trans. Knowl. Data Eng.*, vol. 8, no. 4, pp. 533–539, 1996.

[23] S. Nahar, S. Sahni, and E. Shragowitz, "Simulated annealing and combinatorial optimization," in *Proc. 23rd Design Automation Conf.*, 1986, pp. 293–299.

[24] I. Osman and G. Laporte, "Metaheuristics: a bibliography," *Ann. Oper. Res.*, vol. 63, pp. 513–623, 1996.

[25] D. Papadias, "Hill climbing algorithms for content-based retrieval of similar configurations," in *Proc. ACM SIGIR*, 2000, pp. 240–247.

[26] D. Papadias, N. Mamoulis, and V. Delis, "Algorithms for querying by spatial structure," in *Proc. Very Large Databases Conf.*, 1998, pp. 546–557.

[27] D. Papadias, N. Mamoulis, and D. Meretakis, "Image similarity retrieval by spatial constraints," in *Proc. ACM CIKM*, 1998, pp. 289–296.

[28] D. Papadias, M. Mantzourogiannis, P. Kalnis, N. Mamoulis, and I. Ahmad, "Content-based retrieval using heuristic search," in *Proc. ACM SIGIR*, 1999, pp. 168–175.

[29] D. Papadias and T. Sellis, "A pictorial query-by-example language," *J. Vis. Lang. Comput.*, vol. 6, no. 1, pp. 53–72, 1995.

[30] E. Petrakis and C. Faloutsos, "Similarity searching in medical image databases," *IEEE Trans. Knowl. Data Eng.*, vol. 9, no. 3, pp. 435–447, 1997.

[31] E. Petrakis, C. Faloutsos, and K. Lin, "ImageMap: an image indexing method based on spatial similarity," *IEEE Trans. Knowl. Data Eng.*, to be published.

[32] K. Price, "Relaxation matching techniques—a comparison," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, pp. 617–623, 1981.

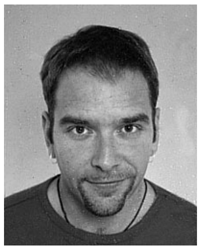
[33] L. Shapiro and R. Haralick, "Structural descriptions and inexact matching," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-3, pp. 504–519, 1981.

[34] J. Smith and S.-F. Chang, "VisualSEEk: a fully automated content-based image query system," in *Proc. ACM Multimedia*, 1996, pp. 87–98.

[35] ——, "Integrated spatial and feature image query," *Multimedia Syst.*, vol. 7, pp. 129–140, 1999.

[36] M. Srinivas and L. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, pp. 656–667, Apr. 1994.

[37] A. Soffer and H. Samet, "Retrieval by content in symbolic image databases," *Proc. SPIE, Storage and Retrieval for Image and Video Databases*, pp. 144–155, 1996.



Dimitris Papadias is an Associate Professor, Department of Computer Science, Hong Kong University of Science and Technology (HKUST). Before joining HKUST in 1997, he worked at the German National Research Center for Information Technology (GMD), Bonn, the National Center for Geographic Information and Analysis (NCGIA), Orono, ME, the University of California at San Diego, the Technical University of Vienna, the National Technical University of Athens, and Queen's University, Kingston, ON, Canada. His main interests are in spatial and spatio–temporal databases.

Marios Mantzourogiannis received the B.Eng degree from the University of Patras, Greece, and the M.Phil. degree from the Hong Kong University of Science and Technology under the supervision of Dimitris Papadias and Ishfaq Ahmad. He is currently serving in the Greek army.

Ishfaq Ahmad (M'92) received the B.Sc. degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1985, and the M.S. degree in computer engineering and the Ph.D. degree in computer science from Syracuse University, Syracuse, NY, in 1987 and 1992, respectively.

He is currently a professor of computer science and engineering in the CSE Department of the University of Texas at Arlington. His recent research focus has been on developing parallel programming tools, scheduling and mapping algorithms for scalable architectures, heterogeneous computing systems, distributed multimedia systems, video compression techniques, and web management. His work in the above areas is published in over 100 technical papers in refereed journals and conferences. He is an associate editor of *Cluster Computing* and *Journal of Parallel and Distributed Computing*.

Dr. Ahmad received the best paper awards at Supercomputing'90 (New York), Supercomputing'91 (Albuquerque, NM), and 2001 International Conference on Parallel Processing (Spain). He is an associate editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, *IEEE Concurrency*, and *IEEE Distributed Systems Online*.