Hill Climbing Algorithms for Content-Based Retrieval of Similar Configurations

Dimitris Papadias
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
+852-23586971
http://www.cs.ust.hk/~dimitris

ABSTRACT

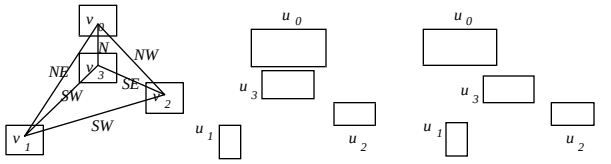
The retrieval of stored images matching an input configuration is an important form of content-based retrieval. Exhaustive processing (i.e., retrieval of the best solutions) of configuration similarity queries is, in general, exponential and fast search for sub-optimal solutions is the only way to deal with the vast (and ever increasing) amounts of multimedia information in several real-time applications. In this paper we discuss the utilization of hill climbing heuristics that can provide very good results within limited processing time. We propose several heuristics, which differ on the way that they search through the solution space, and identify the best ones depending on the query and image characteristics. Finally we develop new algorithms that take advantage of the specific structure of the problem to improve performance.

Keywords

MMIR (general), content-based indexing/retrieval (general), image indexing/retrieval, efficient search over non-textual information

1. INTRODUCTION

The large availability of visual content in emerging multimedia applications and the WWW triggered significant advances in content-based retrieval mechanisms. Such mechanisms, sometimes in conjunction with traditional information retrieval techniques for text, allow the user to access a variety of information sources. A special form of content-based retrieval is *configuration* similarity, otherwise called *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 that of Figure 1a. Such a query could be expressed by one of the existing pictorial languages that permit configuration similarity retrieval, e.g., *VisualSeek* [23], *Query by Sketch* [6], *PQBE* [20], *Safe* [24], 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).



(a) query (b) a perfect match (c) an approximate match

Figure 1 Query example and solutions

Formally, a configuration similarity query can be described by: (i) A set of n variables, $v_0,v_1,\ldots,v_{n-1}$ that appear in the query, (ii) For each variable v_i, a finite domain $D_i = \{u_0,\ldots, u_{Ni-1}\}$ of N_i values, (iii) 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. Nabil et al. [16] use Allen's [1] relations in multidimensional space and conceptual neighborhoods. The idea is extended in [19] with the incorporation of binary string encoding to automate similarity calculations. Conceptual neighborhoods, but this time for topological relations (e.g., inside, overlap), are also applied in [6]. Gudivada and Raghavan [9] use angular directions (e.g., northeast is defined as an angle of 45 degrees) and fuzzy similarity measures. A related approach, which also includes distances between object centroids, is followed in [18].

Independently 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*

Proceedings of the ACM Conference on Information Retrieval (SIGIR), Athens, July 24-28, 2000

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* $S=\{v_0 \leftarrow u_p, .., v_i \leftarrow u_k, .., v_j \leftarrow u_l, .., v_{n-1} \leftarrow u_r\}$ can be defined as:

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

Figures 1b and 1c 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 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 reasonable time. Actually in some cases, even the retrieval of the best solutions in a single large image may take hours to complete.

An alternative 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 hill climbing techniques that can quickly provide good, but not necessarily optimal, solutions. The rest of the paper is organized as follows: Section 2 outlines previous processing approaches and classifies them according to their applicability. Section 3 describes several hill climbing algorithms by exploiting various search strategies and unifying the different approaches under one framework. A detailed study of the solution space provides significant insight for the performance of query processing. The results of this study are used in Section 4 for the development of improved algorithms that take advantage of spatial order to accelerate search. Section 5 concludes the paper with a discussion.

2. QUERY PROCESSING TECHNIQUES

Several query processing techniques have been proposed for configuration similarity retrieval in multimedia databases. 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 variable 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.

The first 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 where 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: (i) locate the query objects in the image (possibly using an index on object id), (ii) for each object pair compute its similarity to the corresponding query constraint, and (iii) calculate the total similarity of the configuration using the pairwise similarities. Gudivada and Raghavan [9] follow this approach to answer configuration similarity queries involving angular directions including rotation invariants. Nabil et al., [16] deal with projection directions and topology. Algorithms that combine pairwise matching with contextual similarity (i.e., based on object features) can be found in [25]. Assuming that image objects are stored using absolute coordinates, pairwise matching can be applied with variable relation schemes. Its disadvantage is its very limited applicability due to the fixed nature of query variables.

Petrakis and Faloutsos [21] solve 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 unlabelled objects are decomposed into combinations of images with fixed size. An enhanced version that reduces the number of dimensions is proposed in [22]. Performance could be further improved by employing more efficient high dimensional indexing methods, such as M trees [5], the pyramid technique [2] etc. Nevertheless the method (like all techniques based on high dimensional indexing and search) is applicable only for fixed relation schemes and databases with small images of mainly labeled objects; otherwise, it is not possible to pre-determine dimensions and build indexes.

A number of methods are based on several variations of 2D strings, which encode the arrangement of objects on each dimension into sequential structures. *2DB* strings [13] capture the object projections, effectively approximating each object by its MBR. *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][12]. Every database image is indexed by a 2D string; queries are also transformed to 2D strings and 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 linear, 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 and use their own relations but only the scheme according to which 2D strings are built.

Another approach is motivated by spatial databases and geographic information systems. In this case, very large images (in the order of 10^5-10^6) contain objects with well-defined semantics (e.g., maps created through topographic surveys). Each map is not stored as a single entity, but information about objects is kept in relational tables with a spatial index for each type of objects covering the same area (e.g., an R-tree for the roads of California, another for residential areas etc). This facilitates the processing of traditional spatial selections (e.g., find all roads inside a query window) and spatial joins (e.g., find all pairs of intersecting roads and railroad lines in California). The same organization can be used to answer configuration queries using cascaded spatial joins.

This technique is applied in [14] for queries where each variable is restricted to an object type (e.g., v_1 must be a road) and the constraint can only be *overlap*. The generalization to arbitrary queries requires the extension of spatial join algorithms to various predicates and approximate retrieval. For most algorithms (e.g., spatial hash joins [11] for intermediate non-indexed results), this is a difficult problem.

Papadias et al., [18] deal 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 medium size images (10^2 - 10^3 objects) and can be employed with variable relation schemes. In [17], 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* [8]. For instance, the running time for the same query in two images of the same size may be order of magnitudes 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.

Consider now a database with numerous, medium or large images where users can ask any type of queries (i.e., with non-fixed variables) using variable relation schemes. The only approach that could be employed is systematic search [17][18], 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, Papadias et al. [19] apply several local search techniques for the retrieval of sub-optimal solutions. Their experimental evaluation reveals that one of these techniques, hill climbing, clearly outperforms the rest (genetic algorithms and simulated annealing) for configuration similarity retrieval.

The good performance of hill climbing motivates the current work, since fast search for sub-optimal solutions is the only way to deal with the vast amounts of multimedia information in several applications. In the sequel we describe several alternatives of hill climbing and identify the problem properties that determine performance by a thorough investigation of the search space. For the following discussion, we assume medium or large non-indexed images and unrestricted variables. In the experimental evaluations we employ the relation scheme of [19], but the algorithms could be used with any type of spatial constraints.

3. HILL CLIMBING AND THE PROBLEM SPACE

The problem space of configuration similarity retrieval can be visualized as a graph, where each solution S corresponds to a node (i.e., the graph has $N!/(N-n)!$ nodes where N is the image cardinality, and n the number of query variables). Two solutions are connected through an edge, if one can be derived from the other by changing the instantiation of a single variable, i.e., the neighborhood of S consists of $n \cdot N$ nodes. Hill climbing algorithms operate on such a graph; starting with a random solution (called *seed*), they improve it by performing *uphill moves*, i.e., by visiting neighbors with higher similarity. Each uphill move (otherwise called a *step*) may involve a number of unsuccessful attempts (i.e., visits to nodes of lower similarity). A solution is called a *local maximum* if no uphill moves can be performed starting from the corresponding node. If hill climbing reaches a local maximum, it restarts the process with a different seed in search of better solutions in other areas of the space. Several variations of hill climbing can be developed depending on the mechanisms for visiting neighboring solutions.

The straightforward approach for generating neighbors is to select a random variable and change its instantiation. This *random variable selection* is followed by the algorithm for configuration similarity retrieval in the implementation of [19]. An alternative approach, motivated by conflict minimization algorithms [15] 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:

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

Worst variable selection re-instantiates 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 algorithm considers the second worst; if it cannot be improved either, the third worst, 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 a local maximum.

Once a variable is chosen for re-instantiation, the value selection mechanism determines its new assignment. The first variation, *all-best value selection* (sometimes called *steepest ascent*), systematically tries all possible values in the domain of the variable to be re-instantiated and assigns the one that results in the solution with the highest similarity. The second variation, *first-better selection*, assigns 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. If after N unsuccessful assignments no better neighbor has been 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, while others not at all.

In order to comprehend the behavior of hill climbing under different combinations of search strategies, we first study the search space for configuration similarity. We produce five queries with 9 variables, and five with 12, and for each query we generate 500 random solutions in a dataset of 1,000 uniformly distributed rectangles with density 0.5 (density is defined as the sum of all rectangle areas divided by the workspace). Figure 2 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 random 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.

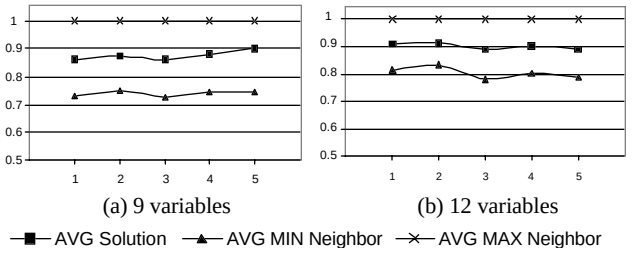


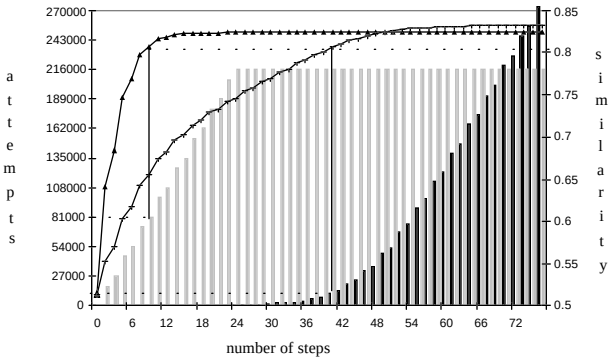
Figure 2 Similarity ranges around random solutions

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) we replace a solution by the best of all its neighbors, which is equivalent to applying all-best value selection to all variables, or (ii) we accept the first better neighbor found by changing the instantiations of random variables, which is equivalent to applying first better value selection to all variables. We refer to the local maxima obtained using these approaches as *All-maximum* and *First-max* respectively. When searching for *All-max*, each step tries all possible values for each variable, i.e., a total of $\Theta(n \cdot N)$ attempts. For *First-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.

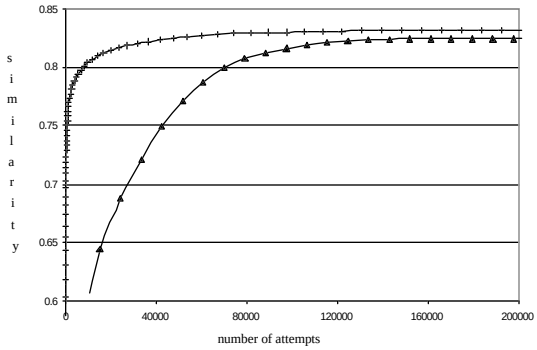
Figure 3a shows how these maxima are reached (attempts, similarity) as a function of the number of steps, for queries of size 9 over the 0.5 density datasets (n=9, N=10^3). The horizontal axis corresponds to the number of steps, the left y axis to the total number of attempts (including unsuccessful instantiations) and the right y axis to similarity. *All-max* (similarity 0.824) is reached after 24 steps; 9,000 instantiations are tested in each step, resulting in 216,000 attempts. *First-max* (similarity 0.831) is reached after 77 steps and 273,408 attempts. Search for *All-max* is deterministic, meaning that starting with one solution, we always reach the same local maximum. On the other hand, the value of *First-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 *First-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 *All-max* approach, the solution will be found after 9 steps and 81,000 attempts (see Figure 3a). On the other hand, if the *First-max* approach is employed, the solution will be found in about 40 steps. However, the total number of attempts is less than 15,000 because uphill moves are easily performed from solutions of low similarity. Each attempt involves a similarity computation; thus the number of attempts (rather than steps) determines the cost of search. Figure 3b illustrates the similarity achieved as a function of the number of attempts for the above query set (9 variables) and dataset (density 0.5) combination. The advantage of the *First-max* search approach is clear, since it converges much faster to high similarity solutions.

According to these 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



(a) Similarity and attempts as a function of steps



(b) Similarity as a function of the number of attempts

	All-max	*First*-max
total attempts		
similarity		

Figure 3 Comparison of *All-max* and *First-max*

searched. Near the local maximum, 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 query sets of 6 and 15 variables over datasets of 1000 uniformly distributed rectangles with densities of 0.1 and 1. In general, the quality of solutions increases with density. Small rectangles are most often disjoint, but as they get larger the diversity of relations between them also grows. Disjoint object pairs can be effortlessly found in datasets with any density. So similarity is determined by tight constraints like *overlap* or *inside* which are more easily satisfied in dense datasets. Figure 4 shows the average similarity (of 25 queries in each set) retrieved over the two datasets every 50 seconds (using a SUN Ultrasparc 2, 200 MHz, with 256MB of RAM).

As expected, first-better value selection quickly (within the first 50 seconds) finds good solutions even for the large queries. Among the two variable selection mechanisms, random selection (R-F) is faster 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

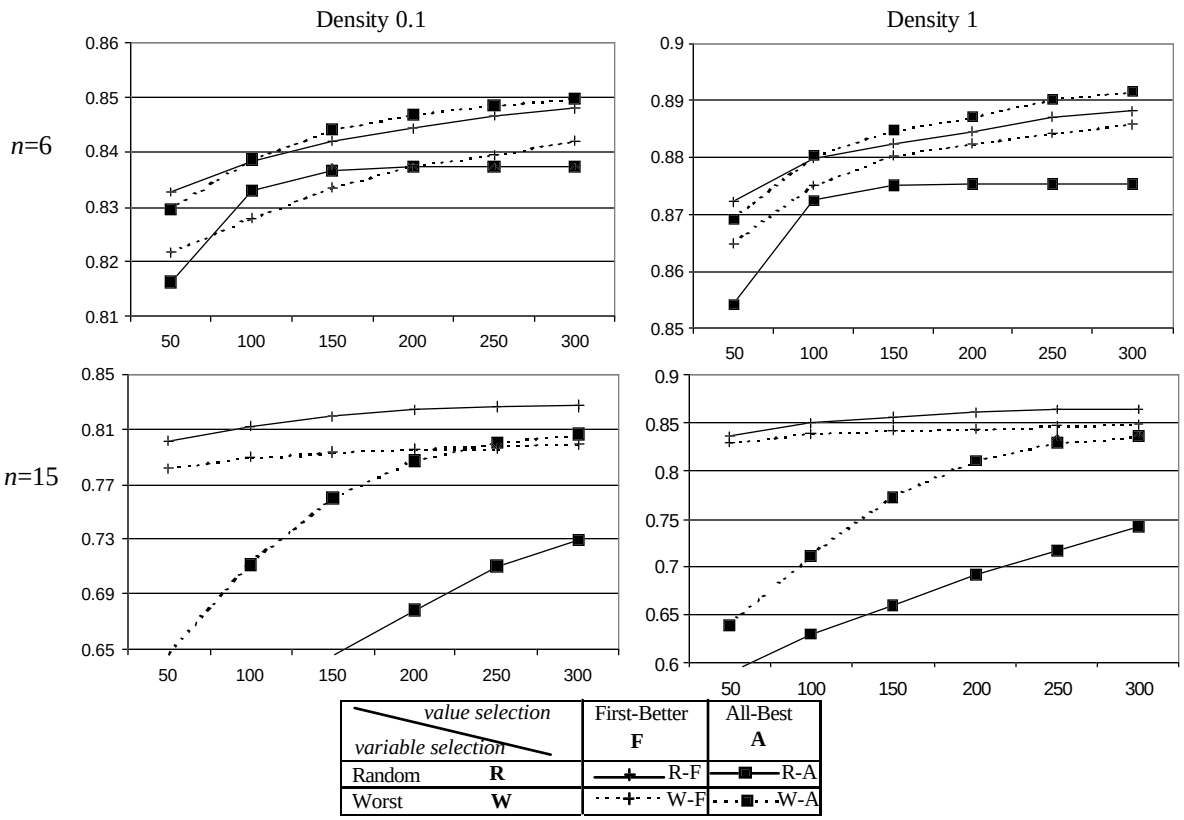


Figure 4 Similarity retrieved as a function of the execution time

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 secs, while R-A does not converge at all within the 300 secs 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.

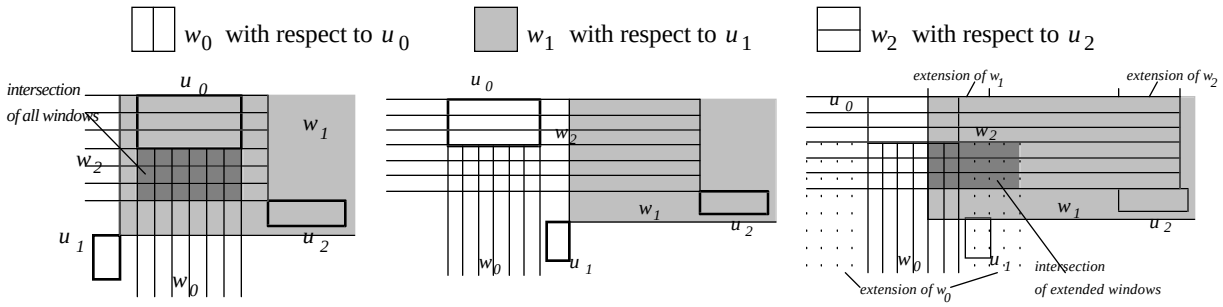
For 6-variable queries, however, W-A (worst variable selection, all best value) outperforms R-F after 100 secs and achieves the highest similarity. This is due to a combination of reasons: for small queries, finding the best possible instantiation for a variable may increase the similarity of the solution 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, in the next section we propose an algorithm that can outperform the previous ones in all cases. The idea is to start with R-F which quickly reaches an area of high similarity. In subsequent steps (when R-F 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 instantiations of a variable.

4. IMPROVED ALGORITHMS

Consider the example query of Figure 1a where the first three variables are instantiated to objects u_0, u_1, and u_2, as shown in Figure 5a. 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) 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 Figure 5a, there is no need to search the whole domain of v_3.

Although, in the example of Figure 5a, 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 Figure 5b, 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.



(a) exact windows (b) exact windows for inexact solution (c) extended windows

Figure 5 Value selection using windows

Window value selection (WVS) applies this idea. Once the variable v_i for re-instantiation has been chosen, the appropriate windows w_j ($0 \le j < n$, $i \ne j$) 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 Figure 5c, 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 which 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, 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^o-50^o in case of a low value of d, or a window 30^o-60^o for higher inconsistency.

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 pre-processing 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 co-ordinates 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. Figure 6 illustrates the results for query-sets (average of 25 queries per set) with 6 and 15 variables over the dataset with density 0.5. WVS always outperforms exhaustive search; as in the case of all-best value selection (see experiments in Figure 4), WVS performs best with worst variable selection (W-WVS) since each re-instantiation may improve the solution significantly.

Despite its good performance, WVS (even with worst variable selection) does not converge fast to high similarity regions, due to the large windows in the initial phases of the algorithm. So, we propose a *two phase search* (2PS) algorithm that first uses R-F to quickly find a good solution, which is then improved by W-WVS. R-F is executed for a time analogous to the size of the problem (for this implementation $n \cdot N$ milliseconds), and W-WVS for the remaining time. For instance, for a query with 15 variables over a 1,000 objects dataset, the running time of R-F is 15 seconds. During this time R-F has performed enough steps to improve the seed significantly. Thus, in most cases the initial windows of W-WVS restrict search in a relatively small portion of the space.

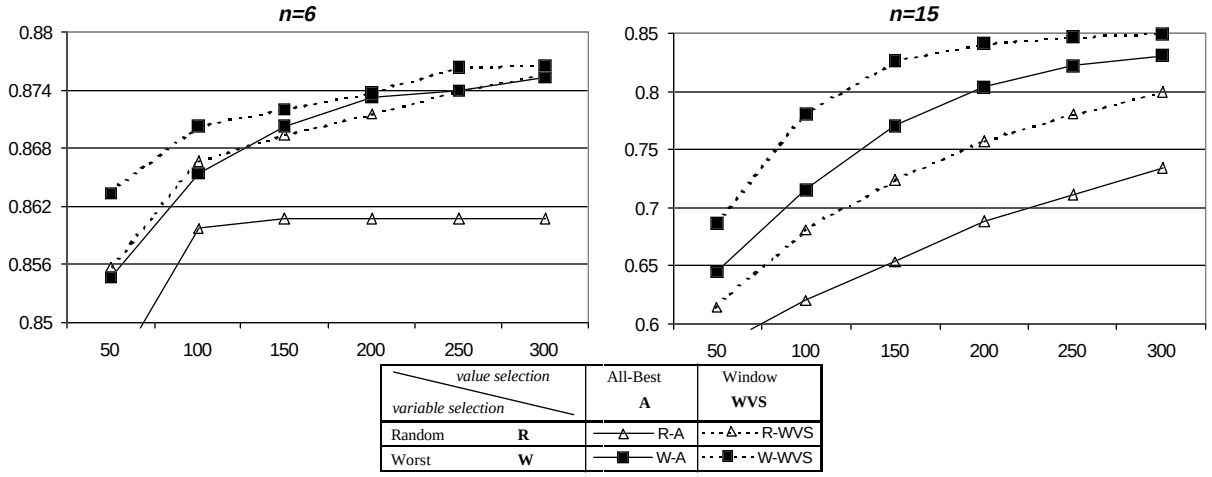


Figure 6 WVS versus exhaustive search - Similarity as a function of time

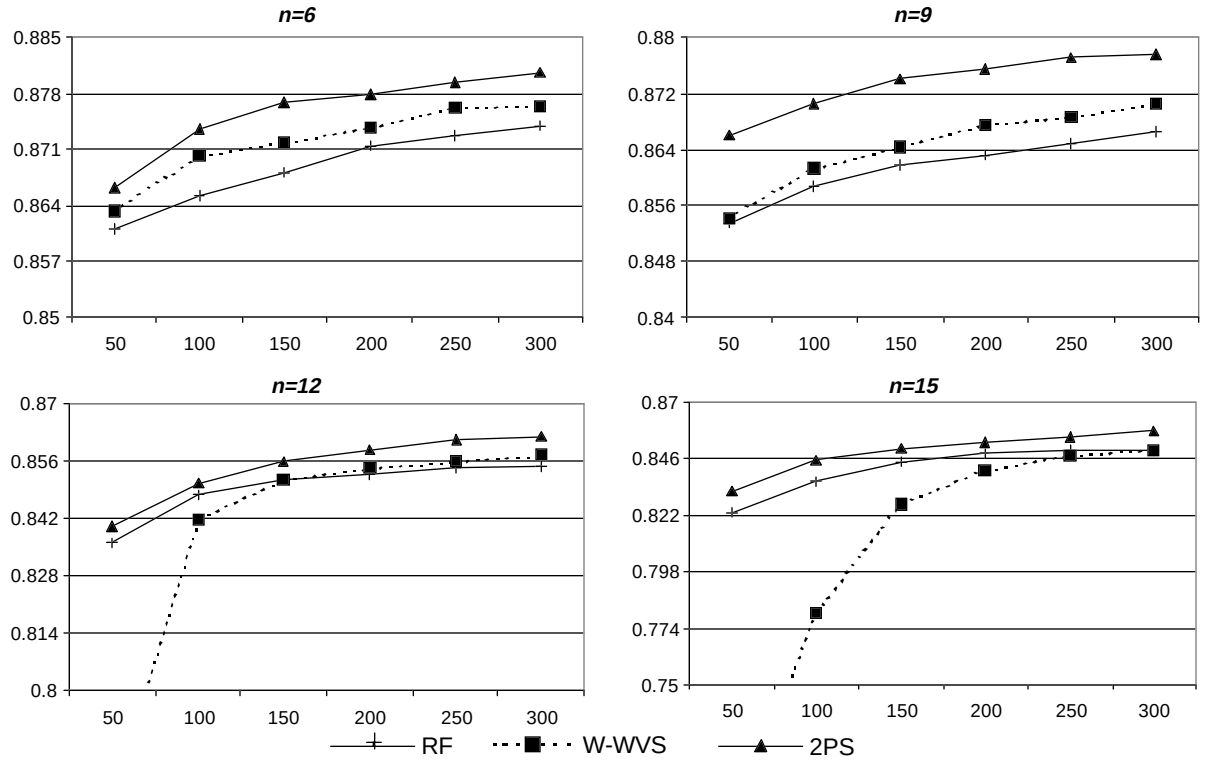


Figure 7 2PS versus WVS and RF - Similarity as a function of time

2PS is tested against W-WVS and R-F using query sets with 6, 9, 12 and 15 variables (25 queries per set) over the 0.5 density dataset. Figure 7 shows the highest similarity retrieved by the algorithms as a function of time. For small queries (6 and 9 variables), WVS produces better solutions than R-F even in the first 50 seconds. As the query size increases, WVS slows down significantly and for 15 variables, it catches up with R-F only at 300 seconds. 2PS outperforms both algorithms since it combines their best characteristics. In general, 2PS has consistently the best performance of all hill climbing variations for all combinations of queries/datasets tested (including real data). In addition to its robustness, another advantage of 2PS, and hill climbing in general, with respect to other local search algorithms, is that it does not require the complicated tuning of parameters (e.g., population size and generations in genetic algorithms, or temperature and equilibrium conditions in simulated annealing) which significantly affect efficiency.

In order to evaluate the effectiveness of our methods with respect to other query processing techniques, we replicate the above experiments using random sampling (RND) and forward checking (FC). RND just picks solutions at random and returns the best one. It has been shown that, depending on the structure of the search space, in some applications it may outperform techniques based on local search [7]. FC [10] is one of the most efficient systematic search algorithms, traditionally used for constraint satisfaction problems. We implement a branch and bound version of FC, which backtracks when a partial solution cannot reach the similarity of the best solution found so far. A similar implementation outperforms several other systematic search algorithms (e.g., backjumping, dynamic backtracking) for configuration similarity queries when the goal is to find the best solution with no time limit [18].

Table 1 shows the best similarity retrieved over time for queries with 6 and 15 variables for the 0.5 density dataset. In general, both algorithms provide very low similarity when compared with the corresponding values in Figure 7. RND produces better results than FC within the 300 seconds but quality does not increase much over time, implying that only a small percentage of solutions have similarity close to a local maximum. Although FC, as expected, improves gradually with time it does not find good solutions even for 6 variables within the available time. The situation is worse for 15 variables due to the significant increase in the search space; FC remains in the neighborhood of the initial assignments, which in most cases have low quality.

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

Table 1 Similarity as a function of time for RND and FC

These results, which were also validated with various geographic datasets, clearly motivate the need for fast retrieval of sub-optimal solutions. Notice that, due to the unavailability of representative actual application queries, we do not use real datasets in the experimental evaluation, since datasets by themselves do not determine performance. Nevertheless, the large variance among data densities, query sizes and constrainedness allows for general conclusions about performance.

5. CONCLUSION

This paper applies hill climbing algorithms for the effective retrieval of configuration similarity. A thorough investigation of the search space of the problem leads to the development of algorithms that can find good solutions even if very limited time is available for query processing. The most efficient algorithm is 2PS, which first applies random variable, first-better value selection to reach an area of high similarity. Then, window value selection employs spatial order to identify good values, without searching the whole domain of the variable to be re-instantiated. 2PS has a very robust performance, quickly locating very good solutions for all types of queries and datasets tested. Since the algorithm does not require the tuning of special parameters we expect similar behavior for several types of applications.

In the future we plan to extend our work for spatio-temporal objects and queries. Consider, for instance, a database with satellite images of weather patterns. Since temporally adjacent images are usually similar, solutions of previous images can be used to guide search for subsequent ones. Queries can also be extended to capture motion, e.g., find the set of images containing a movement similar to that of a given weather pattern. In this case solutions are not found independently in each image, but they are interrelated as specified by the motion constraints.

The increasing amount of visual content and the emergence of real-time multimedia applications (e.g., WWW visual search engines) provide a significant motive for the development of fast retrieval techniques. Since other existing methods are either inapplicable for general queries, or not guaranteed to terminate within reasonable time (possibly missing a large part of the database), hill climbing algorithms constitute one of the most important alternatives for configuration similarity processing.

ACKNOWLEDGMENTS

The paper was funded by RGC grants HKUST 6158/98E and 6090/99E. Thanks to Marios Mantzourogiannis for the implementation. Part of this work was done while the author was fulfilling his military service requirements at the Department of Communications, island of Chios (96 ΛΔΒΕΘ), Greece. I would like to thank all the people in the camp and especially captain Nikos Delis and soldier Panos Nikakis for their encouragement.

REFERENCES

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

[2] Berchtold S., Boehm C., Kriegel H. The Pyramid-Technique: Towards Breaking the Curse of Dimensionality. *ACM SIGMOD*, 1998.

[3] Chang S., Jungert E., Li. T. Representation and Retrieval of Symbolic Pictures Using Generalized 2D Strings. *SPIE Visual Communications and Image Processing IV*, 1989.

[4] Chang S., Shi Q., Yan C. Iconic Indexing by 2-D String. *IEEE PAMI*, 9(3), 413-428, 1987.

[5] Ciaccia P., Patella M., Zezula P. M-tree: An efficient access method for similarity search in metric spaces. *VLDB*, 1997.

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

[7] Galindo-Legaria C., Pellenkoft A., Kersten M. Fast, Randomized Join-Order Selection - Why Use Transformations ?. *VLDB*, 1994.

[8] Gent I., MacIntyre E., Prosser P., Walsh T. The Constrainedness of Search. *AAAI*, 1996.

[9] 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.

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

[11] Koudas N., Sevcik K. Size Separation Spatial Join. *ACM SIGMOD*, 1997.

[12] Lee S, Hsu F. Spatial Reasoning and Similarity Retrieval of Images using 2D C-Strings Knowledge Representation. *Pattern Recognition*, 25(3), 305-318, 1992.

[13] Lee S, Yang M, Chen J. Signature File as a Spatial Filter for Iconic Image Database. *Journal of Visual Languages and Computing*, 3, 373-397, 1992.

[14] Mamoulis N., Papadias D. Integration of Spatial Join Algorithms for Processing Multiple Inputs. *ACM SIGMOD*, 1999.

[15] Minton S., Johnston M., Philips A., Laird P. Minimizing Conflicts: A Heuristic Method for Constraint-Satisfaction and Scheduling Problems. *Artificial Intelligence*, 58, 161-205, 1992.

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

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

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

[19] Papadias D., Mantzourogiannis M., Kalnis P., Mamoulis N., Ahmad I. Content-Based Retrieval Using Heuristic Search. *ACM SIGIR*, 1999.

[20] Papadias D., Sellis T. A Pictorial Query-By-Example Language. *Journal of Visual Languages and Computing*, 6(1), 53-72, 1995.

[21] Petrakis E., Faloutsos C. Similarity Searching in Medical Image Databases. *IEEE TKDE*, 9(3) 435-447, 1997.

[22] Petrakis E., Faloutsos C., Lin K. ImageMap: An Image Indexing Method Based on Spatial Similarity. *IEEE-TKDE*, (in press).

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

[24] Smith J., Chang S-F. Integrated Spatial and Feature Image Query. *Multimedia Systems*, 7, 129-140, 1999.

[25] Soffer A., Samet H. Pictorial Queries by Image Similarity. *13th International Conference on Pattern Recognition*, 1996.