2nd Hellenic Conf. on AI, SETN-2002, 11-12 April 2002, Thessaloniki, Greece,
Proceedings, Companion Volume, pp. 141-152

Heuristic Algorithms for Similar Configuration Retrieval in Spatial Databases

Dinos Arkoumanis, Manolis Terrovitis,Lefteris Stamatogiannakis

Dept. of Electrical and Computer Engineering
National Technical University of Athens
Zographou 157 73 Athens, Greece
{dinosar,mter,estama}@dblab.ntua.gr

Abstract. The search for similar configurations is an important research topic for content-based image retrieval in G.I.S. and spatial databases. Due to the complexity of the problem, finding the fittest solution in a large database is computationally intractable. Our work is focused on designing, implementing and experimentally evaluating two heuristic algorithms, an evolutionary and a hill-climbing one, that provide an approximate solution. With the use of spatial indexes we manage to efficiently deal with considerably large queries. We utilize a similarity framework that addresses topological, directional and distance relations. In this framework the problem of retrieving similar configurations is defined as a binary constraint satisfaction problem. Our work complements the existing work on similarity retrieval with two efficient, stochastic, algorithms.

1 Introduction

A user of a G.I.S. system usually searches for configurations of spatial objects on a map that match some ideal configuration or are bound by a number of constraints. For example, a user may be looking for a place to build a house. He wishes to have a house A north of the town that he works, in a distance no greater than 10km from his child's school $B$and next to a park C. Moreover, he would like to have a supermarket D on its way to work. Modern GIS systems look for a solution that satisfies all query conditions. In the case of complex queries, like the above, it is very unlikely that a solution in the database exists. An answer that indicates that no solution has been found is not very helpful. In these cases the user is interested to know the closest possible answer. The way to overcome this is either by having the user to relax the constraints he sets on his ideal house configuration, or having an algorithm that returns the fittest configurations rated by some similarity metric. The problem of *configuration similarity retrieval* in spatial databases expresses the latter approach of the problem. It has been an active area of research in the recent years [8]. Similar problems are very common in information retrieval areas (WWW search engines are a good example of this), where the user has an overview of the alternatives if a perfect solution is not found.

Our work focuses on similar configurations retrieval. Retrieving the fittes t configuration is a computationally intensive procedure. We use the similarity framework of [8]. We address three kinds of relations: topological, directional and distance.

Related work is presented by Roussopoulos et. al. [12]. They propose a technique for answering nearest neighbor queries with the help of an R-tree index. Papadias et. al in [9], reduce the configuration similarity retrieval to several smaller ones with the use of a spatial index. [9] uses only topological relations. Moreover, the authors investigate variations of a forward checking and a minimum conflict local search algorithm. Finally, Papadias et. al., in [10], propose a few algorithms for a similar problem, but deal with small queries and with a limited version of the problem.

Related work partially deals with the problem as defined in [8] or address rather easy versions of it. In this paper we design, implement and evaluate two heuristic algorithms; a Hill-Climbing and an evolutionary one, and we investigate the successful integration of the R*-tree index in them. We find an approximate solution to the most computationally intensive version of the problem, using at the same time significantly larger queries than most in approaches that have appeared in the past. We experimentally evaluate the effect of several parameters on the algorithms' performance and study the behavior of several of their variations.

The rest of the paper is organized as follows: In Sections 2 and 3 we present the similarity retrieval framework. Section 4 proposes a Hill-Climbing and an evolutionary algorithm. In Section 7 we present the results of the experimental evaluation of the algorithms' performance. Our conclusions are presented in Section 8.

2 Spatial Relations

As mentioned in Section 1, we use a similarity framework that relies on three kinds of spatial relations, namely topological, directional and distance. Let us now present a short description of each one.

Topological relations express the concepts of inclusion and neighborhood. We use the topological relations of the 9-intersection model [2]. This model identifies the following 8 pair-wise disjoint topological relations (Figure 1):

{*Disjoint*, *Meet*, *Overlap*, *Inside*, *Covered_by*, *Equal*, *Contains, Covers*}

Two topological relations T_i and T_j are first degree neighbors iff there is an edge (T_i, T_j) in the graph of Figure 1. For instance, *Overlaps* and *Covers* are first degree neighbors while *Overlaps* and *Equals* are not. The similarity $sigma_T$ of two topological relations T_i and T_j is defined as follows [PAK98]:

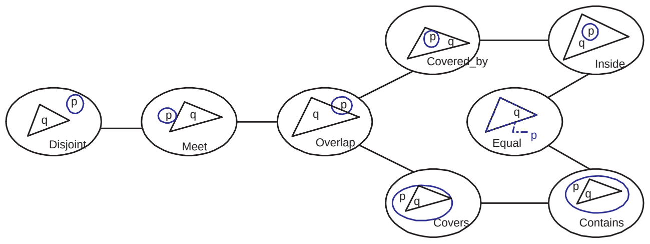

Fig. 1. Topological relations and 1^{st} degree neighbors graph

$$\sigma_T(T_i, T_j) = \begin{cases} 1 & T_i = T_j \\ \tau \ (0 < \tau < 1) & \text{if } T_i \text{ and } T_j \text{ are } 1^{\text{ st}} \text{ degree neighbors} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where τ is an application defined constant.

relative to one another (e.g., object a is *north* of object a). We follow a centroid-based approach where the direction between objects is determined by the angle between their centroids (a more expressive model is discussed in [14]). This models identifies the following 8 directions:

$$\{NorthEast, North, NorthWest, West, SouthWest, South, SouthEast, East\}.$$

The similarity σ_A of each direction with a given angle θ is the following trapezoid function [8]:

$$\sigma_A = (A_i, \theta) = \begin{cases} \theta/(i45^o - a) & (i-1)45^o < \theta < i45^o - \alpha \\ 1 & \text{if } i45^o - \alpha \leq \theta \leq i45^o + \alpha \\ ((i+1)45^o - \theta)/(i45^o - \alpha) & i45^o + \alpha < \theta < (i+1)45^o \\ 0 & otherwise \end{cases} \tag{2}$$

where α is a constant that expresses the tolerance in our directional relations, and i is an integer in the range 1..8. For $\theta = 0$, we have an eastbound relation.

A *distance* relation $D_{[d_1,d_2]}$ between two objects indicates that the distance between the centeroids of the two objects is no less than d_1 and no more than d_2. For instance, relation $D_{[0,d]}$ means "closer than d" and relation $D_{[d,\infty]}$ means "further than d".

The similarity measure of a distance d_x with the relation $D_{[d_1,d_2]}$ is given by the following equation [8]:

$$\sigma_D(D_{[d_1,d_2]}, d_x) = \begin{cases} 1 & \text{if } d_1 \leq d_x \leq d_2 \\ 0 & otherwise \end{cases} \tag{3}$$

3 Similarity Retrieval as a Constraint Satisfaction Problem

Let us assume that we have a database storing the spatial objects $O$of an image I. Let also C be a set of query conditions. A typical database query Q looks

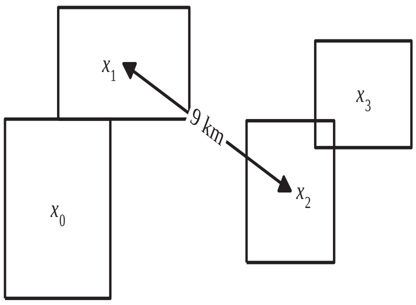

Fig. 2. Prototype

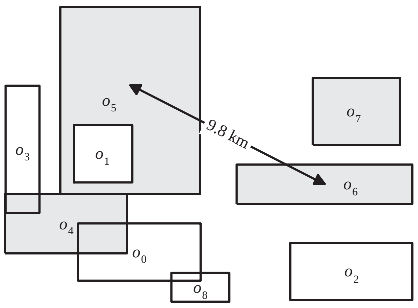

Fig. 3. Solution

for objects in the database that satisfy *all* conditions C. A *similarity query* is something more general. It considers that conditions C describe a *prototype* and looks for objects that are similar to it. In this section, we formalize such queries as constraint satisfaction problems and we define a metric (the satisfaction degree) that can be used to express similarity.

Example 1. Let us now consider the following similarity query over the spatial objects $\{o_0, ..., o_8\}$ of the image of Figure 3:

"Retrieve all configurations that are similar to the scene where there are four objects x_0, x_1, x_2 and $x_3$such that object x_0 *meet*s object x_1, object x_1 is *disjoint* to object x_2 and it is in a *distance* of 8km to 10km from object x_2 and object x_2 *overlap*s with object x_3, which is *north* from object x_2."

Figure 2 illustrates the prototype of the example query. The assignment $\{x_0 \leftarrow o_4,\ x_1 \leftarrow o_5,\ x_2 \leftarrow o_6,\ x_3 \leftarrow o_7\}$, illustrated in Figure 3, is similar to the prototype; only one condition is lost since x_2 does not *overlap* with x_3.

A configuration similarity query Qcan be formalized as a *binary constraint satisfaction problem* (BCSP), which is verbally defined in [6] as:

- A set of n variables $\{x_0, \ldots, x_{n-1}\}$.
- For each variable x_i a finite domain $D_i$of potential values.
- For each pair of variables x_i, x_j a set of binary constraints C_{ij}, where C_{ij} is a subset of $D_i \times D_j$.

Thus, a query $Q$is a triplet $(V,\ D,\ C)$ where V =$\{x_0,\ \ldots,\ x_{n-1}\}$ is a set of variables, $D = \{D_0,\ \ldots,\ D_{n-1}\}$ is the set of their domains and C is a set of constraints over $x_0,\ \ldots,\ x_{n-1}$. Let $\{x_0 \leftarrow o_o,\ \ldots,\ x_{n-1} \leftarrow o_{n-1}\}$ be an assignment of variables (where $o_i \in D_i$). The *satisfaction degree* of an assignment $\{x_0 \leftarrow o_0,\ \ldots,\ x_{n-1} \leftarrow o_{n-1}\}$, expresses how close this configuration is to the query's prototype and is defined as follows [8].

$$\sigma = \frac{\sum\limits_{i \neq l, 0 \leq i,j < n} \sigma_T(C_T(x_i,x_j),T(o_i,o_j)) + \sigma_A(C_A(x_i,x_j),\theta(o_i,o_j)) + \sigma_D(C_D(x_i,x_j),d(o_i,o_j))}{3n(n-1)} \qquad (4)$$

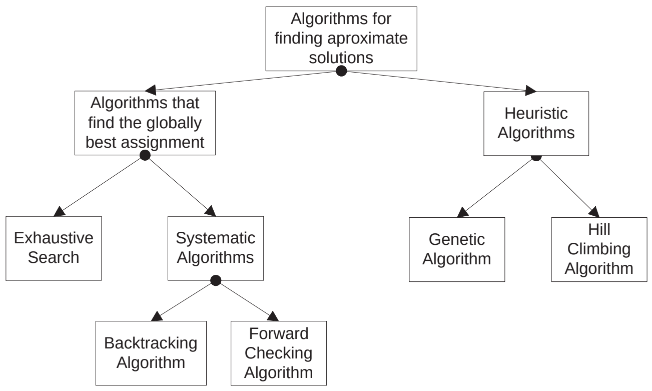

Fig. 4. Algorithms for finding approximate solutions to similarity retrieval BCSP

where C_T, C_A, C_D are the topological, directional and distance constraints respectively between the variables x_i, x_j, and o_i, o_j are objects that are assigned to these variables.

The *satisfaction degree* of an assignment can be calculated using several other possible metrics. [13] describes *conjunctive combination* (i.e., the minimum degree of satisfaction of individual constraints) and *productive combination* (i.e., the product of satisfaction degrees of individual constraints). The problem with conjunctive combination is that it does not distinguish between assignments that contain equally "bad" binary instantiations, while productive combination does not differentiate between instantiations that fully violate some constraint(s).

Here we use the average combination metric that was proposed by [8], which is the sum of all pair-wise satisfaction degrees divided by the total number of constraints. This avoids the drawbacks of the other metrics (i.e., conjunctive and productive combination) but is more computationally expensive (instantiations cannot be abandoned as early in the search process). We choose this metric for our work since it allows to sort assignments that are not perfect a lot easier than the others. For the rest of the paper when we say an assignment a is better than another assignment b, it will mean that assignment a has a greater degree of satisfaction than assignment b.

4 Finding Approximate Solutions

In Section 3, we have seen that the similarity retrieval problem can be formalized as a BCSP. BCSP is an NP-complete problem, thus, even with the use of indexes, finding the best global solution is a computationally intensive procedure. Our work is not focused in finding a solution for the BCSP but rather at finding an approximate one. The algorithms that have been proposed [Ark01] for this problem can be classified as shown in Figure 4. They are briefly sketched in the sections 4.1 , 4.2.

4.1 Exhaustive search

Exhaustive search examines all possible assignments of query variables and evaluates their satisfaction degree in order to find the best possible assignment. The above procedure is performed without pruning any part of the search space. As a result it is highly inefficient, especially for large queries and large databases.

4.2 Systematic algorithms

Systematic algorithms also find the best assignment of query variables but they avoid examining areas of the search space that do not lead to better assignments. They take advantage of the data index (R*-tree for our data) and they perform well for large queries and databases provided that we are interested in assignments with high satisfaction degree. Unfortunately, if we are also interested in assignments with lower satisfaction degree the performance of systematic algorithms deteriorates a lot, because in this case they cannot avoid examining many areas of the search space. [Ark01] proposed two systematic algorithms a backtracking and a forward checking. Backtracking does not check the areas of the search space that do not lead to assignments with higher satisfaction degree than the one it has already found. Forward checking, on the other hand, limits the set of potential values of each variable of the query, according to the constraints between the variables and the current partial instantiations.

4.3 Heuristic algorithms

The main difference between systematic and heuristic algorithms lies in the fact that the latter do not always find the overall optimal assignment. Typically they find an assignment with a high satisfaction degree quickly. Their performance varies since they approximate the solution in a non-deterministic way.

Evolutionary Algorithms. Evolutionary algorithms [Hol62][5], instead of evolving a single assignment to increase the satisfaction degree, they evolve a population of assignments with competitive criteria. A population p, of random assignments is created. On this population a series of variation operators is applied. In our implementation we use the mutation and the crossover operator. The application of these operators is probabilistic; there is a $m_p$probability that the mutation operator will be applied and an m_c probability that the crossover operator will be applied, for each assignment. After the application of the variation operators the degree of satisfaction of each assignment is calculated according to Equation 4 and a selection operator is applied to determine the assignments that are going to survive to the next iteration (or generation if we use the evolutionary term). The selection procedure is probabilistic too. The best assignments are more likely to survive, and the worst less. The selection is done is such a way that the population size of each generation remains the same. This means that in many cases some assignments are duplicated, sometimes more than once, at the next generations, whereas others do not appear at all. We present below a short description of each operator.

Mutation. Mutation is the most basic operator of an evolutionary algorithm. It helps introduce new elements into an assignment and it enables the assignments' population to evolve. The basic idea is to replace an object of the current assignment with another one from the database. The techniques we use to choose the variable to be re-instantiated and the new object to be assigned are described in more details in Sections 5 and 6.

Crossover. The 2-point crossover technique [PMK+99][11], is used. Two points of an assignment are chosen and the objects between them are exchanged with those from another random assignment.

Tournament Selection. Three different ways of selecting the assignments that will survive at the next generation, were examined:

– Proportional selection, where the chance of each assignment to survive is proportional to its similarity.
– Ranking, where all assignments of the initial population are ordered according to their similarity score and the best are chosen to survive. In variant of this technique the objects that will survive at the next generation are chosen with probability proportional to their rank.
– Tournament selection, where a subpopulation t of the initial assignments is selected randomly and the best assignment of this population is chosen for the next generation. This is repeated for each assignment of the next generation.

After testing the three variants we chose the last one because it is the most flexible. By changing the size of the subpopulation, the tournament size, the randomness of the selection procedure can be adjusted. This way the randomness degree that gives the best performance to the algorithm can be found by experimental methods.

The basic function of the evolutionary algorithms is described by the following equation [3]:

$$x(t+1) = s(v(x(t))) \tag{5}$$

where $x(t)$ is the population of the t generation and $v()$ and $s()$ are variation and selection operators respectively.

Our evolutionary algorithms' performance is affected greatly by several factors: the probability p_m of each assignment to be mutated, the probability p_c of the crossover operator to be applied, the size of the tournament population t_s and the size of the generation population p_s. After implementing the algorithms we did extensive experiments to achieve a fine-tuning. In the procedure of fine-tuning the four previous factors were adjusted in order to maximize the satisfaction degree of the best assignment found in a time period of 2 minutes.

Local Search – Hill Climbing Algorithm Hill-Climbing uses an iterative improvement technique. A single random assignment is created and a variable is chosen to be re-instantiated (*current variable*). During each iteration, a new object is selected from the domain of the current variable. If that object provides a better similarity score, it is assigned to the current variable, and a new variable is chosen to be re-instantiated. Otherwise some other object is selected and tested on the current variable. The method terminates if no further improvement is possible. The details of the algorithm are discussed in the following Section.

It is clear that Hill-Climbing algorithm highly depends on the selection of the starting point. Moreover, it can only provide locally optimum assignments. In such cases, no further improvement can be made, and the process will re-start from another random assignment.

5 Variable selection

The choice of the variable that will be re-instantiated has a significant effect on the performance of our algorithms. In both of the previous algorithms we used the minimum conflict heuristic [7], which suggests the replacement of the "worst" variable. Each variable in an assignment contributes a certain percentage in the total inconsistent degree. The inconsistence degree of a variable x_i in a assignment S is defined as follows:

$$dgr(x_i, S) = \frac{2}{n} - \sum_{i \neq l, i=k, 0 \leq j < n} \sigma(C(x_i,x_j),(o_i,o_j)) - \sum_{i \neq l, j=k, 0 \leq i < n} \sigma(C(x_i,x_j),(o_i,o_j)) \qquad (6)$$

where $\sigma(C(x_i, x_j), (o_i, o_j))$ is the sum of the similarity measures given by equations 1, 2 and 3, C is the conjunction of C_T, C_A and C_D and o_i, $o_j \epsilon S$, are the objects that are assigned to variables x_i, x_j respectively. This metric expresses the difference that the satisfaction degree would have, if instead of the object that is currently assigned to variable x_i, an object that satisfied all the constraints that bound the variable x_i, were assigned.

In the Hill-Climbing algorithm the object that violates the most constraints, as calculated by Equation 6, is selected to be substituted. After a variable is re-instantiated, the same procedure is repeated with the rest of the variables until new objects are assigned to all of them. When this occurs the algorithm outputs the result and continues with the same assignment but with *all* variables being considered, again, for substitution. A new random assignment is created when the similarity of the assignment does not improve after trying to re-instantiate all the variables.

In evolutionary algorithms our approach is slightly different. The object with the highest inconsistence degree is always chosen to be re-instantiated with all variables being considered for re-instantiation. If a local maximum has been reached and no better assignment can be found a random object is chosen to be re-assigned to the same variable. In this case, even if the variable that violates the greatest number of constraints remains the same, the decreased similarity of the assignment renders unlike the survival of the assignment in the next generation.

After providing the description of the algorithms we to discuss the improvement that spatial indexing techniques can bring to them.

6 Value Selection

The choice of the objects that will be assigned to the variables of the query is identical in both of our algorithms. We studied two basic strategies: a) To select one object randomly b) to select the object that would give the highest satisfaction degree to our assignment (*fittest object*). Selecting the fittest object without the use of spatial indexes is a slow procedure, especially for large databases. In order to overcome this, in our implementation, we used the R*-tree [1], which is variant of the R-tree [4].

6.1 The R-tree

The R-tree is a height balanced tree and it can be considered as the extension of the B+-tree in multiple dimensions. R-trees are used for the dynamic organization of a set of k-dimensional geometric objects (2-dimensional in our case) representing them by the Minimum Bounding k-dimensional Rectangle (MBR). Each R-tree node corresponds to the MBR that contains its children. The leaves contain pointers to the objects of the database.

There are many variations of the R-trees. For our work we use a popular variation, the R*-tree, which has the same structure as the R-tree but it uses a more efficient clustering algorithm.

6.2 Using the R*-tree

Due to the increased computational cost of search for the fittest object, for an existing assignment, a random substitution is the best choice if a spatial index is not available. In the worst case, with an n variable query in an N object database, 3(n-1) constraints for N objects must be examined. Using an R*-tree index over the spatial data for the search procedure, enables the algorithms to limit the search space significantly.

Before examining a sub-tree of the R*-tree, the algorithms check if its MBR violates any constraints with the existing assignment. The similarity of a relation between an MBR M_2 and an object o_1, is defined as the maximum similarity of the relation of an object that may exist in M_2 with o_1. In this way we can translate the relations between two objects o_1 and $o_2$to a set of prerequisite relations between o_1 and an intermediate node M_2, where M_2 is an ancestor of o_2, as illustrated in Table 1. The first column shows the topological relations between two objects o_1, o_2 and the second column shows the relations that o_1 and M_2 must have in order for o_2 to be a child of M_2.The correspondence of directional and distance relations is simpler. Whereas the similarity of a relation between two objects o_1 and o_2 is actually the similarity of the relation of the two

Relation(x_i, x_j)	*Condition for intermediate nodes*(X_i, x_j)
equal	equal∨cover∨contain
contain	Contain
inside	overlap∨covered_by∨inside∨equal∨cover∨contain
cover	cover∨contain
covered_by	overlap∨covered_by∨equal∨cover∨contain
disjoint	disjoint∨meet∨overlap∨cover∨contain
meet	meet∨overlap∨cover∨contain
overlap	overlap∨cover∨contain

Table 1. Conditions for intermediate nodes

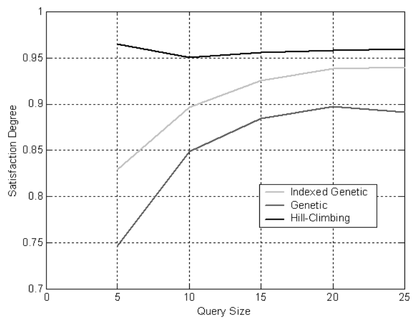

Fig. 5. Prototype

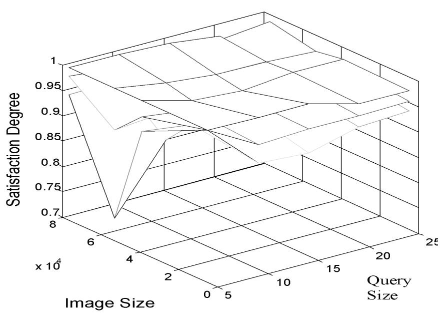

Fig. 6. Solution

objects' centeroids, the similarity of the relation of o_1 and M_2 is the maximum similarity or the relation between the centeroid of o_1 and any point of M_2.

Calculating the similarity of an MBR with an existing object, in this way, enables the algorithm to prune all the R*-tree branches that cannot lead to assignments with a higher satisfaction degree.

7 Experiments

We performed our experiments on four images concerning geographical areas (such as the Long Beach roads) or VLSI circuits. The sizes of the images vary from 5000 to 75000 data objects. In each image, we performed 5 sets of queries in 5, 10, 15, 20 and 25 variables respectively.

We have constructed our queries from actual configurations on each image. We chose a set of objects in each image and we created the queries in such a way so that the relations between the objects in the images would be the constraints between the variables in the derived queries. This method of construction produced queries with some special characteristics. Each topological constraint, between two variables, is satisfied for just one topological relation. The distance tolerance is 10% for all distance constraints (the minimum and maximum distances defined by the constraint are ±10% of the actual distance between the objects). The directional constraint is satisfied by only one direction or by two neighbor ones (for example N, NE). The queries produced do not have any universal constraints and they have only few, or just one, solutions on each image. At the same time though, we are certain that a perfect solution exists for each query, thus we can observe how close to the optimal are the assignments that the algorithms find.

We tested 3 algorithms: The Evolutionary Algorithm, the indexed Evolutionary algorithm, where the mutation is done by using the R*-tree, and the Hill-climbing Local Search Algorithm. All algorithms were left to run for 2 minutes and the satisfaction degree of the best assignment found at the end of this time period, was recorded. They were tested with 5 queries of each size on 4 images, and they performed 5 iterations of each run. As a result, for each query size, we had $5 \times 4 \times 5 = 100$ runs. The results for each algorithm can be seen in Figure 5.

From the Figure 5 it is evident that the use of R*-tree increases the performance of the algorithms, since the simple evolutionary algorithm is the worst in all cases. Still, though, it performs quite well in comparison with the others, for large queries. Both evolutionary algorithms perform better as the size of the query grows. We conjecture that our genetic algorithm gets trapped easier at a local maximum and always loses some constraints. In small queries, losing just a few constraints has a grater impact on the satisfaction degree than in larger queries where the total number of constraints is polynomially larger. It is, also clear, that the Hill-Climbing Local Search outperforms both other algorithms for all query sizes and at the same time demonstrates a robust behavior by performing equally well in all cases. This is more visible in Figure 6 where the average, the best and the worst performance for each query size and each image are shown for the Hill-climbing Local Search algorithm. As the size of the query grows the derivation of the satisfaction degree of the assignment found, is decreased.

8 Conclusions

This paper applies heuristic algorithms in order to process spatial configuration similarity queries. The goal is to retrieve the best assignment of database objects to query variables resembling a given spatial configuration. We investigate a version of the Hill-Climbing algorithm and two evolutionary algorithms. With the use of spatial indices, the exhaustive domain search is avoided, giving us the possibility to handle large queries, up to 25 variables, over large databases, up to 75000 objects. The results, we obtain, show that our algorithms can find satisfying approximations to the queries (i.e., assignments with high satisfaction degree), with the assignments being more than 95% accurate even at the hardest cases.

We believe these methods have a wide range of applications in spatial and multimedia systems, as well as the upcoming video compression methods. Another area of application is the emerging spatiotemporal databases, where scheduling problems requiring acceptable solutions fast, will be common.

Further experimenting with variations and similar algorithms, but mainly with their tuning over the indexing techniques, will make the topological configuration similarity problem solvable, for even larger domains, aiming at millions of objects.

References

1. N. Beckmann, H. Kriegel, and B. Schneider, R.and Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990*, pages 322–331. ACM Press, 1990.
2. M. J. Egenhofer and R. D. Franzosa. Point-set topological spatial relations. *International Journal on Geographical Information systems*, 5(2):161–174, 1991.
3. D. Fogel and A. Ghozeil. Using fitness distributions to design more efficient evolutionary computations. In *International Conference on Evolutionary Computation*, pages 11–19, 1996.
4. A. Guttman. R-trees: A dynamic index structure for spatial searching. In Beatrice Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.
5. Holland. Outline for a logical theory of adaptive systems. In *Essays on Cellular Automata, ed. Arthur W. Burks, University of Illinois Press, Urbana, Chicago, London.* 1970.
6. A. Mackworth and E. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25(1):65–74, 1985.
7. S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.
8. D. Papadias, N. Arkoumanis, and N. Karacapilidis. On The Retrieval of Similar Configurations. In *Proceedings of 8th International Symposium on Spatial Data Handling (SDH)*, 1998.
9. D. Papadias, P. Kalnis, and N. Mamoulis. Hierarchical Constraint Satisfaction in Spatial Databases. In *Proccedings of AAAI Conference*, pages 142–147, 1999.
10. D. Papadias, N. Mamoulis, and V. Delis. Algorithms for querying by spatial structure. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 546–557, 24–27 1998.
11. D. Papadias, M. Mantzourogiannis, P. Kalnis, N. Mamoulis, and I. Ahmad. Content-based retrieval using heuristic search. In *SIGIR '99: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 15-19, 1999, Berkeley, CA, USA*, pages 168–175. ACM, 1999.
12. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 71–79. ACM Press, 1995.
13. Z. Ruttkay. Fuzzy constraint satisfaction. In *Proceedings 1st IEEE Conference on Evolutionary Computing*, pages 542–547, Orlando, 1994.
14. S. Skiadopoulos and M. Koubarakis. Composing Cardinal Directions Relations. In *Proceedings of the 7th International Symposium on Spatial and Temporal Databases (SSTD-01)*, volume 2121 of *LNCS*, pages 299–317, July 2001.