GENETIC ALGORITHMS FOR GRAPH LAYOUTS WITH GEOMETRIC CONSTRAINTS

Dana Vrajitoru
Intelligent Systems Laboratory
Computer and Informations Sciences
Indiana University South Bend
danav@cs.iusb.edu

Boutros R. El-Gamil
Department of Computer Science, Faculty of Science,
Minia University, El-Minia, Egypt
boutrosgameil@yahoo.com

ABSTRACT
In this paper we introduce an application of real-coded genetic algorithms to the problem of consistent graph layouts and explore the potential contribution of the genetic algorithms for this particular problem. Given a weighted graph, we want to find a geometric position of every vertex (layout) consistent with the weights in the graph. Among the possible solutions to this problem, we would also like to find one that follows a particular given shape or has some geometric properties. Our paper shows that the genetic algorithm can help improve the quality of the solution based on the geometric criteria and not only on precision.

KEY WORDS
genetic algorithms, graph theory, geometric modeling

1 Introduction

The problem we focus on for this study is building consistent graph layouts for weighted graphs, in particular following a specified geometric shape. In this paper we explore the potential use of genetic algorithms to this problem and various implementation aspects related to it. The graph theory represents an interesting challenge for the genetic algorithms because many of the graph problems are NP-complete or generally hard to solve. The genetic algorithms can be a viable alternative to more traditional approaches.

Let us start from the assumption that a weighted graph represents a discretization of a geometrical object with a specific shape and topological properties. If we don't know the exact coordinates of every vertex in the graph, and its general shape, is it possible to deduce them from the distribution of weights in the graph? Thus, given a weighted graph, we must derive a three dimensional spatial representation of the graph (a layout) such that the distances between the vertices are consistent with the weights of the corresponding edges.

Extensive work was accomplished on drawing unweighted graphs with emphasis on showing the structure of the graph in the geometrical representation [8]. Layouts presenting some aesthetic qualities are also appreciated [6]. The problem is even more interesting and challenging when the graphs to be drawn are large [3].

The best-known heuristic for generating graph layouts is certainly the spring algorithm [5] that considers the edges in the graph as springs connecting the vertices, pulling them closer if the distance between them is greater than the optimum, and pushing them apart in the opposite case. In addition, non-connected vertices often repel each other. In the usual implementation, the edges are expected to have the same length. Part of the research on graph layouts has also focused on weighted graphs and the best methods seems to be force-oriented [8], [1].

An interesting model [4] combines this method with the use of genetic algorithms. The authors have applied their model to graph layouts for unweighted graphs with a fitness reflecting some visual, aesthetic properties of the layout. We extend our approach to weighted graphs and focus on reducing the discrepancy between the weights in the graph and the distance between the vertices. We combine this measure with others reflecting geometric properties that can help the algorithm in the search for particular interesting shapes.

The paper is structured the following way: Section 2 introduces the graph layout problem and two force-based heuristics. Section 3 introduces the problem representation and the genetic operators we used. Section 4 presents some experimental results with the genetic algorithm focusing on the precision of the solution. Section 5 introduces a new approach combining the precision with geometric properties. We finish with conclusions.

2 Problem Description and Force Based Algorithms

The first aspect we try to address is, given an undirected and weighted graph, we must assign a geometric point to each of the vertices in the graph (a layout) such that for every edge AB, the distance between the vertices A and B is equal to the weight of the edge. The second aspect we are interested in is, given that for many graphs, there is an infinity of possible solutions to the first question, is it possible to impose some constraints on the solution to find the shape of the original graph layout? Among the applications of these algorithms we can cite designing electronic circuits [8], designing web sites and visualizing the content

of the World Wide Web [2], parallel computing and VLSI.

Definition. Let $G = \{\mathcal{V}, \mathcal{E}\}$ be a graph where $\mathcal{V}$ is the set of vertices, $|\mathcal{V}| = n$, and $\mathcal{E}$ is the set of edges, $|\mathcal{E}| = m$. A *layout* for the graph is a function $P : \mathcal{V} \rightarrow \mathbf{R}^p$ that maps each vertex $v \in V$ to a geometric point in $\mathbf{R}^p$, where usually $p = 2$ or 3. The edges are represented as line segments connecting the points associated with the vertices.

We will denote the undirected edges by uv, where $u, v \in \mathcal{E}$.

Problem. Let $G = \{\mathcal{V}, \mathcal{E}, W\}$ be an undirected, weighted graph where the weights of the edges are given by the function $W : \mathcal{E} \rightarrow \mathbf{R}^+$. We must find a layout $P : \mathcal{V} \rightarrow \mathbf{R}^3$ such that $\forall u, v \in \mathcal{V}$, $\mathrm{d}\,(P(u), P(v)) = W_{uv}$, the weight of the edge uv. A layout with this property will be called a *consistent layout* for this graph.

If $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, then we must find a set of points $\{P_1, P_2, \ldots, P_n\}$ such that if there is an edge between two vertices v_i and v_j, $v_i v_j \in \mathcal{E}$, then the points associated with these vertices are situated at a distance from each other equal to the weight of the edge.

$$\mathrm{d}\,(P_i, P_j) = W_{v_i v_j} \tag{1}$$

There exist graphs with no solution, others that have a family of isomorphic solutions, or others with an infinity of non-isomorphic solutions. For example, a graph containing a triangle where the weight contradict the triangulation property will have no solution. A triangle has a unique solution if we ignore the isometric transformations. A higher order polygon has an infinity of non-equivalent solutions. In the latter case, we would like to find a particular solution resembling a specified shape, as for example, a regular grid, an ellipsoid, or a torus.

2.1 Breadth-First Order Algorithm

To generate graphs for which a consistent layout can be found, we can start with any unweighted graph, build a layout by any method, then assign weights to the edges according to the Euclidean distance between the corresponding points.

Let u and v be two adjacent vertices in the graph. Suppose that we have associated the geometric points P_u and P_v with the vertices u and v respectively. Let us denote by err_{uv} the error on the edge uv computed as the difference between the weight of the edge and the Euclidean distance between the two points:

$$err_{uv} = W_{uv} - d(P_u, P_v). \tag{2}$$

This error gives us an estimation of how much the points are misplaced with respect to each other considering that the weight of the edge represents the ideal distance between them. If the error is positive, then the points are too close to each other. If the error is negative, the points are too far apart.

We would like to find a layout that minimizes the total absolute error in the graph:

$$total\_error = \sum_{\forall uv \in \mathcal{E}} |err_{uv}| \tag{3}$$

The algorithm presented in this subsection follows the ideas from the the spring algorithm and most of the force-oriented methods. The graph forms a dynamic system in which each element (vertex) is attracted or repelled by its neighbors according to the discrepancy between the length of the line segment connecting them and the weight of the edge. In our model, if two vertices are not neighbors in the graph, then they do not interact directly with each other.

The algorithm starts with a random layout that is adjusted in a number of iterations to obtain one that is consistent. For every iteration, the algorithm repositions one vertex at a time such as to reduce the error on an adjacent edge. The second point on the edge is considered as a reference. The point is moved on the line segment representing the edge, further away from the reference point if the distance is smaller that the weight of the edge, and closer to the reference point if the distance between them is greater than the weight of the edge.

Let u and v be the two vertices that have been randomly selected and P_u and P_v the points assigned to them in the current layout. If the error on the edge uv is not equal to 0, we will adjust the position of the vertex v by assigning it a new point P'_v determined in the following way:

$$P'_v = P_v + \varepsilon \cdot \frac{err_{uv}}{d(P_u, P_v)} \cdot (P_v - P_u), \tag{4}$$

where ε is a constant, $0 < \varepsilon < 1$.

An iteration of the *breadth-first order (BF)* algorithm adjusts all but one vertex in the graph according to (Equation 4). The algorithm starts with a randomly chosen vertex (origin), and it adjusts all the other vertices in the graph starting from this origin in a breadth-first order. The parameter ε allows us to control the amount of adjustment that is performed at each step and thus, decide on the convergence rate.

2.2 Tension Vector Heuristic

Let us suppose that we can construct a physical representation of the graph using interconnecting springs for the edges, as in the spring method [5]. Each spring corresponding to an edge has an initial length equal to the weight of the edge and a section much smaller than the length. When extended, the springs tend to contract to their initial length, and when compressed, they tend to extend. Moreover, each spring creates a contracting or extending force along the main direction proportionate to the amount of deformation that was applied to it.

The *tension vector (TV)* algorithm computes the total deformation forces in each point of the graph in the current configuration, and then moves all of the points in one step

according to these forces. This will result in an equilibrium solution.

We define the deformation force on the edge AB depends on the error on this edge, err_{AB}, as defined by Equation 2. Then we can define the deformation force applied to the point B as

$$\vec{F}_{AB} = err_{AB} \frac{\vec{AB}}{\|\vec{AB}\|} \quad (5)$$

The resulting force applied to the point A is then defined as:

$$\vec{R}_A = \sum_{\forall AB \in \mathcal{E}} \vec{F}_{BA} \quad (6)$$

If P_A is the point associated with the vertex A in a particular iteration and $\vec{R}_A$ is the force exerted on it, the algorithm moves the point to a new location P'_A defined as follows:

$$P'_A = P_A + \varepsilon \vec{R}_A \quad (7)$$

where ε is a constant, $0 < \varepsilon \leq 1$.

The algorithm starts again with a random layout and moves the points according to Equation 7 in a given number of iterations or until the layout convergences to an equilibrium. In each iteration, all of the tension forces are computed in the first step, then all of the points are moved in the next step without recomputing the forces.

In this algorithm, the tension force in every vertex is computed based on the current layout before any of them is moved. This is the major difference between this algorithm and the breadth-first order method introduced in the previous subsection, which moves one point at a time.

The preliminary results of these methods presented in [12] are encouraging. Both methods have converged to a solution very close to an exact one. The tension vector algorithm can derive solutions of higher precision in many cases, but can also diverge in some situations. The BF algorithm is more robust from this point of view.

An iteration of each of the algorithms is linear over the number of edges in the graph. The execution time in the experiments presented here thus depends on the total number of iterations.

3 Genetic Representation

In this section we introduce the genetic representation of the consistent graph layout problem focusing on the precision of the solution.

3.1 Chromosome Representation and Genetic Operations

Since a layout is composed of points with real coordinates, a real-encoded genetic representation is appropriate for this problem. Several real-encoded models for genetic algorithms have been proposed [7], with more focus on the crossover operator [9], or the search space [11]. Our study focuses on a different aspect than these other approaches.

To apply the GAs to to our problem, we represent a graph layout as a chromosome. A layout is a string of points in the three dimensional Euclidean space. The length of the string is equal to the number of vertices m. Each of the points is composed of three genes taking real values. The genes are initially generated in a given boundary which is a 3D box of dimensions depending on the weights in the graph.

The fitness function is based on the total error in the graph as computed in Equation 3 and is given by the formula

$$f = \frac{1}{1 + total\_error}$$

Since our chromosomes are of fixed length, any of the classical crossover operators can be applied without modification. We have chosen the uniform crossover [10] with a swap probability established experimentally.

The fitness function introduced in Equation 3.1 is linear over the number of edges in the graph. Thus, the execution time of the algorithm depends linearly on the size of the graph, the size of the population, and the number of generations. So, even if we chose a number of generations equal to the number of iterations of the heuristics presented in Section 2, the GAs require more execution time.

3.2 Real-Encoded Gene Mutation

The mutation has always played an important role for the genetic algorithms, but usually has less impact on the population evolution than the crossover. Practically, the mutation acts by modifying a randomly selected gene, for example by flipping the bit in the binary gene encoding. The real-coded representation for our problem presents more possibilities for the mutation. We are interested in the impact of the various mutation forms on the achieved fitness. This part of the paper represents an extension and continuation of [13].

The mutation operator is more easily defined in the context of binary genes. The genetic representation of a problem with floating-point genes offers more possibilities for defining the mutation. Several forms have been proposed in the literature, and some of our operators are inspired from them.

The *uniform* mutation replaces the value of the chosen gene with a uniform random value within the range specified by the user [7]. The *mirror* mutation replaces a gene with it's mirror value with respect to the middle point of the boundary interval for the gene. This is the closest operator to the binary bit-flipping usual mutation. The *percentage* mutation replaces a gene with a random percentage of its value within the interval $[80\%, 120\%]$.

These three types of mutation are not specific to our problem. The following two forms of mutation represent a combination of the heuristics presented in Section 2.

Table 1. Total error as percentage of the total weight in 1000 generations

Size	Uniform	Mirror	Percentage	Edge	TVM
50	93.64%	94.45%	68.14%	60.35%	21.02%
60	93.63%	94.33%	69.07%	58.99%	19.63%
70	94.14%	94.69%	74.02%	64.97%	08.82%
80	93.74%	94.32%	71.61%	61.80%	15.48%
90	94.18%	94.67%	74.12%	62.69%	16.59%
100	93.95%	94.46%	75.02%	64.17%	11.23%
125	94.19%	94.67%	76.64%	64.15%	13.60%
150	94.18%	94.65%	79.11%	66.85%	14.15%
175	94.30%	94.72%	79.46%	65.60%	14.64%
200	94.33%	94.74%	81.12%	67.12%	22.87%

Table 2. Average total error in 1000 iterations and 500 generations plus 500 iterations

Size	BF	TV	GA+BF	GA+TV
50	0.012%	0.024%	1.145%	2.115%
60	0.045%	0.042%	2.187%	2.000%
70	9.45e-04%	1.06e-03%	0.100%	0.046%
80	1.20e-06%	2.39e-07%	0.000%	0.0
90	1.11e-06%	2.02e-07%	0.004%	0.399%
100	5.71e-03%	0.031%	0.168%	0.390%
125	9.81e-07%	0.003%	0.0%	0.0%
150	1.07e-06%	0.019%	0.0%	0.0%
175	9.83e-07%	0.020%	0.0%	0.0%
200	1.05e-06%	0.019%	0.0%	0.0%

The *edge* mutation is moving the 3 genes of a randomly selected point on a randomly selected edge starting from that vertex according to Equation 4 to reduce the error on that edge. The *tension vector* mutation (TVM) is moving the 3 genes of a randomly selected point based on the tension vector resulting from all edges starting from that vertex as described in Equation 7.

4 Experimental Results

We have conducted our experiences with the set of graphs introduced in Section 2. These graphs were generated such that there exists at least one solution to the consistent layout problem. The number of vertices varies from 50 to 200.

Based on some preliminary tests, a swap probability of 0.45 for the uniform crossover seems to be the most appropriate for this problem, and we have used this value of the parameter for the rest of the experiments.

Table 1 shows the results of the five variations of the mutation operator on the ten graphs with exact solution. The first column represents the number of vertices in the graph. All of these results represent the total error as a percentage of the total weight of the edges in the graph, as an average over 50 trials. The number of generations is 1000 and the population size is 50. The ϵ parameter is equal to 0.005 for these results.

We can notice that there is a considerable difference in performance between the forms of mutation we have introduced. The uniform and mirror mutations are the only ones that seem to give similar results, even though the uniform mutation is slightly better. In comparison, the percentage mutation is doing visibly better than both of them, with more than 10% of improvement in all the cases. The edge mutation is a further clear improvement, showing that taking advantage of the specificity of the problem can have a positive impact on the GAs.

Last, the tension vector mutation is substantially better than even the edge mutation. The difference between the two heuristics is larger than in the case were they were applied exclusively in part because by combining the tension vector method with the genetic algorithms, we avoid the divergence issues previously encountered.

As an additional experiment, we combined the execution of the GAs with the force-based methods. We chose the tension-vector mutation and evolved 500 generation of the genetic algorithms followed by the BF or TV heuristic. Table 1 presents the total error as percentage of the total weight in the graph, first for the two force-based methods in 1000 iterations, then for the combination of 500 generations of the genetic algorithm with 500 iterations of the force-based methods with $\varepsilon = 0.05$.

This table shows that the results of the combination of the two approaches are comparable to the force-based methods, and even exceeds it in some cases. The genetic algorithm also solved the divergence problem that the tension vector methods display sometimes.

The numbers in the second and third columns represent the total error as percentage of the total weight in the graph achieved in 1000 iterations . Higher values have caused the tension vector algorithm to diverge. We computed the average over 50 trials with the same parameters but different sequences of random numbers.

Comparing the results in Tables 2 and 1, we can conclude that the force-based heuristics perform better than the GAs on this particular problem. Even if the actual precision of the solution is not the goal of this paper, we can still wonder if the GAs are appropriate for solving the graph layout problem. In the context of simply generating a precise and consistent solution, the force-based heuristics seem like a better approach indeed.

5 Geometric Constraints

The experiments presented in the previous section seem to indicate that the force-based algorithms perform better in general than the GAs for this kind of problem. Still, the GAs have the advantage of a providing a great flexibility in the choice of the fitness function, and we can use this feature to derive solutions with interesting geometric shapes.

Many of the known geometric shapes present such properties as maximizing the enclosed surface or volume, as it is the case for spheres of any dimension. In this section we add such properties to the fitness function to see how they change the general shape of the solution.

We performed our experiences with three types of constraints: angle, surface, and volume. In the case of the *angles*, we added a component to the fitness function minimizing the standard deviation of the angles, maximizing their uniformity. For the *surface* component, we try to maximize the surface occupied by the graph layout in a particular projection. And last, for the *volume* component, we try to maximize the product of the standard deviation of the points from the geometric center of the layout in the 3 dimensions.

The actual fitness function represents, in all cases, a combination of the component minimizing the error presented in Equation 3 and one of the tree shape components in equal measures.

In the literature it was suggested that adding *repulsive vectors* to the spring algorithm can help building layouts with nicer geometric shapes, showing the graph structure more clearly. Specifically, if A and B are two non-adjacent vertices, then a repulsion force is established between them, inversely proportionate to the distance between them. More precisely, the repulsion force R_F has the following expression:

$$R_F = \frac{1}{d(A,B)^p}$$

where p is an integer number, the most often equal to 2.

In the following experiments, we have combined the genetic algorithm with the force-based algorithms for two graphs with a recognizable structure. The first one is a 2x2 grid, and the second one a 4x4 grid, both made of squares. Figure 1 shows the original layout of the two graphs.

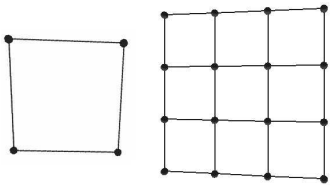

Figure 1. The two graphs for the experiments

First we applied three forms of force-based algorithms, breadth-first based (BF), tension vector TV, and a combination of the tension vector method with the repulsion forces (RV). Next, we applied a genetic algorithm for 500 generation using a fitness function representing the sum of f in Equation 3.1 and the measure for the angle, for the surface, and for the volume respectively. We then continued for another 500 iterations using the RV method. The next four figures will show the shapes derived by these methods.

Figure 2 shows the shape derived for the grid 2x2 by the force-based methods respectively. Figure 3 shows the shapes derived by the GAs using the three fitness measures combined with the RV method. From these figures we can see that both the RV method by itself, and the GAs with a fitness maximizing the volume achieve the closest shape to the original layout, but the GAs are somewhat better.

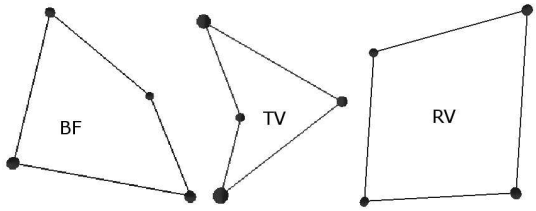

Figure 2. The shapes derived by force-based algorithm for the grid 2x2

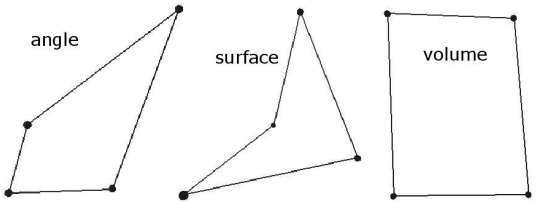

Figure 3. The shapes derived by GAs and RV for the grid 2x2

Figure 4 shows the shape derived for the grid 4x4 by the force-based methods respectively. Figure 5 shows the shapes derived by the GAs using the three fitness measures combined with the RV method. This shape constitutes more of a challenge for all algorithms, and even the RV method by itself doesn't achieve the desired shape. The closest shape to the original design is the one derived by the GAs with a fitness maximizing the volume.

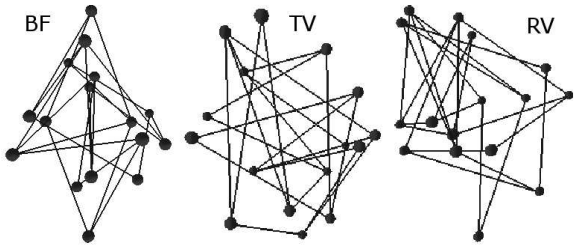

Figure 4. The shapes derived by force-based algorithm for the grid 2x2

We can deduce from these experiments that the genetic algorithms can contribute indeed to deriving better shapes for the graph layouts while preserving the consistency properties. The measure aiming to maximize volume seems to work better as the fitness than the corresponding

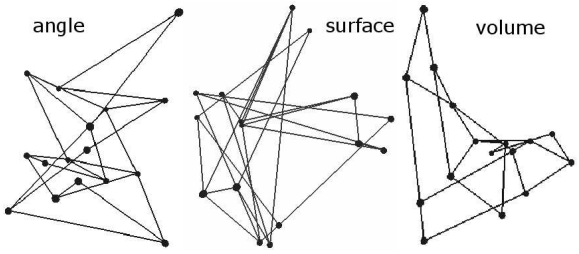

Figure 5. The shapes derived by GAs and RV for the grid 2x2

one for the angles or for the surface. Experiments with more complex shapes and larger graphs are not yet conclusive.

6 Conclusions

In this paper we introduced a real-coded genetic algorithm applied to the consistent graph layout problem. The focus of this study has been on two aspects, the precision of the solution derived by force-based methods as compared to the GAs, and the use of the GAs to build layouts following a particular geometric shape.

Our experiments have shown that the GAs can be quite successful in deriving a precise solution to the problem, even if the force-based algorithms are a viable alternative. In addition, the choice of the mutation operatorfor the GAs can have a large impact on a real-encoded genetic algorithm. Last, a combination of the force-based algorithms and the GAs can avoid some of the divergence problems of the tension vector method while achieving similar levels of precision for the solution.

By introducing a new constraint in the derived solution concerning not only the precision, but the actual geometric shape, we have shown that the GAs present a clear advantage over the other methods. In this approach, the fitness function reflects both the precision of the solution and some geometric properties, for example, maximizing the surface or the volume. The results presented in this section show that the solutions derived by the GAs are closer to the desired geometrical shape than the force-based algorithms alone.

The conclusion is that even though in simple terms of precision, the force-based heuristics are more efficient in finding consistent graph layouts, the genetic algorithms can be used to help achieve a different set of properties in the solution. Last, the combination between the two types of algorithms seems to be the most promising approach for both aspects we have studied.

7 Acknowledgments

This work has been in part conducted under the IUSB Faculty Fellowship Grant, 2004.

References

[1] H.L. Bodlaender, M.R. Fellows, and D.M. Thilikos. Derivation of algorithms for cutwidth and related graph layout problems. Technical Report UU-CS-2002-032, Institute for Information and Computing Sciences, Utrecht University, 2002.

[2] U. Brandes, V. Kääb, A. Löh, and D. Wagner. Dynamic WWW structures in 3d. *Journal of Graph Algorithms and Applications*, 4(3):183–191, 2000.

[3] U. Brandes and D. Wagner. Using graph layout to visualize train interconnection data. *Journal of Graph Algorithms and Applications*, 4(3):35–155, 2000.

[4] J. Branke, F. Bucher, and H. Schmeck. Using genetic algorithms for drawing undirected graphs. In J.T. Allen, editor, *The Third Nordic Workshop on Genetic Algorithms and their Applications*, pages 193–205, 1997.

[5] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[6] P. Gajer and S.G. Kobourov. Grip: Graph drawing with intelligent placement. *Journal of Graph Algorithms and Applications*, 6(3):203–224, 2002.

[7] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading (MA), 1989.

[8] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. An Alan R. Apt Book. Prentice Hall, Upper Saddle River, NJ, 1999.

[9] I. Ono and S. Kobayashi. A real-coded genetic algorithm for function optimization. In *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 246–253, 1997.

[10] G. Syswerda. Uniform crossover in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the International Conference on Genetic Algorithms*, San Mateo (CA), 1989. Morgan Kaufmann Publishers.

[11] S. Thutsui and D. Goldberg. Search space boundary extension method in real-coded genetic algorithms. *Information Sciences*, 133(3-4):229–247, 2001.

[12] D. Vrajitoru and J. DeBoni. Consistent graph layout for weighted graphs. In *The 3rd ACS/IEEE International Conference on Computer Systems and Applications*, Cairo, Egypt, 2005.

[13] D. Vrajitoru and J. DeBoni. Hybrid real-coded mutation for genetic algorithms applied to graph layouts. In *Proceeding of the Genetic and Evolutionary Computation Conference (GECCO'05 and SIGEVO 1)*, pages 1563–1564, Washington, DC, June 25-29 2005.