Drawing Undirected Graphs with Genetic Algorithms

Qing-Guo Zhang, Hua-Yong Liu, Wei Zhang, and Ya-Jun Guo

Department of Computer Science, Central China Normal University, Wuhan 430079, China
qgzhang@mail.ccnu.edu.cn

Abstract. This paper proposes an improved genetic algorithm for producing aesthetically pleasing drawings of general undirected graphs. Previous undirected graph drawing algorithms draw large cycles with no chords as concave polygons. In order to overcome such disadvantage, the genetic algorithm in this paper designs a new mutation operator single-vertex- neighborhood mutation and adds a component aiming at symmetric drawings to the fitness function, and it can draw such type graphs as convex polygons. The improved algorithm is of following advantages: The method is simple and it is easy to be implemented, and the drawings produced by the algorithm are beautiful, and also it is flexible in that the relative weights of the criteria can be altered. The experiment results show that the drawings of graphs produced by our algorithm are more beautiful than those produced by simple genetic algorithms, the original spring algorithm and the algorithm in bibliography [4].

1 Introduction

A number of data presentation problems involve the drawing of a graph on a limited two-dimensional surface, like a sheet of paper or a computer screen. Examples include circuit schematics, algorithm animation and software engineering. In almost all data presentation applications, the usefulness of a drawing of a graph depends on its readability, that is, the capability of conveying the meaning of the diagram quickly and clearly. Readability issues are expressed by means of aesthetics, which can be formulated as optimization goals for the drawing algorithms. Many aesthetic criteria can be conceived of and the generally accepted ones include:

(1) Uniform spatial distribution of the vertices.
(2) To minimize the total edge length on the precondition that the distance between any two vertices is no less than the given minimum value.
(3) Uniform edge length.
(4) To maximize the smallest angle between edges incident on the same vertex.
(5) The angles between edges incident on the same vertex should be as uniform as possible.
(6) Minimum number of edge crossings.
(7) To exhibit any existing symmetric feature.

While these criteria are useful measures of aesthetic properties of graphs, this is not an exhaustive list and there are other measures that can be used [1] .

L. Wang, K. Chen, and Y.S. Ong (Eds.): ICNC 2005, LNCS 3612, pp. 28 – 36, 2005.

It is not easy to locate the vertices of a general undirected graph so that it conforms to aesthetically pleasing principles of layout. There are many different strategies that can be used to draw a general undirected graph. One method is to use the spring model algorithm [2]. The algorithm likens a graph to a mechanical collection of rings (the vertices) and connecting springs (the edges). Two connected rings are attracted to each other or repelled by each other according to their distance and the properties of the connecting spring. A state with minimal energy in the springs corresponds to a nice drawing of the underlying graph. However, the spring method is likely to be trapped by local optima and thus obtains very poor drawings. Another method is to use simulated annealing algorithm [3]. Davidson and Harel have used the algorithm to draw undirected graphs. This algorithm produces drawings that are comparable to those generated by the spring model algorithm. However, the algorithm does not produce conventional looking figures for a large cycle with no chords. While this is normally drawn as a large circle, this algorithm tends to draw the cycle curled around itself and thus obtains a concave polygon but not a convex one. And also the simulated annealing algorithm is likely to be trapped by local optima and thus obtains very poor drawings. Eloranta and Mäkinen [4] present a GA for drawing graphs with vertices over a grid and use several operators but remark on the lack of a good crossover operator. And also the algorithm draws a large cycle with no chords curled around itself.

A graph G=(V, E) is formed by a set of vertices V and a set of edges E. It may be represented in different styles according to the purposes of the presentation. We are interested here in producing aesthetically-pleasing 2D pictures of undirected graphs. Vertices will be drawn as points in the plane inside a rectangular frame and edges will be drawn as straight-line segments connecting the points corresponding to the end vertices of the edges. So the problem of graph drawing reduces to finding the coordinates of such points. This paper concentrates on constructing the straight-line drawings of general undirected graphs with genetic algorithms. The algorithm has the following four advantages:

(1) The figures drawn by the algorithm are beautiful.
(2) it can draw large cycles with no chords as convex polygons.
(3) It is simple and it is easy to be implemented.
(4) It is flexible in that the relative weights of the criteria can be altered.

2 The Genetic Algorithm for Drawing Undirected Graph

The most important thing of solving graph drawing problems with genetic algorithms is to design fitness functions according to the adopted aesthetic criteria. The fitness function is given in section 2.2, and the various elements of the algorithm are illustrated in the following subsections.

2.1 Encoding

Let G=(V, E) be a finite, undirected, simple graph. Let n=|V| denote the number of vertices of G, and let m=|E| denote the number of edges of G. Suppose the vertices

sequence of a graph G is $v_1, v_2, \cdots, v_n$, and the coordinates assigned to them are $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$, respectively. The algorithm uses a real number vector $(x_1, y_1, x_2, y_2, \cdots, x_n, y_n)$ with the length of 2n to denote the solution to the problem. In order to draw graphs inside a rectangular frame in the plane, we add the following constraints:

$$a \le x_i \le b, \quad c \le y_i \le d$$

2.2 Fitness Function

The algorithm designs the following fitness function according to the aesthetic criteria (1)–(7), which are stated in Section 1. The fitness function is interpreted as follows:

$$f = \frac{w_1}{\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{1}{d_{ij}^2}} + \frac{w_2}{\sum_{(v_i,v_j)\in E} d_{ij}^2} + \frac{w_3}{\sum_{(v_i,v_j)\in E}\frac{(d_{ij} - ideal\_edgelength)^2}{m}} + \frac{w_4}{\sum_{\substack{i=1 \\ (v_i,v_j)\in E \\ (v_i,v_k)\in E}}^{n}\frac{1}{(\angle p_j p_i p_k)^2}} + \frac{w_5}{\sum_{\substack{i=1 \\ (v_i,v_j)\in E \\ (v_i,v_k)\in E}}^{n}\frac{(\angle p_j p_i p_k - \frac{2\pi}{\deg ree(p_i)})^2}{\deg ree(p_i)}} + \frac{w_6}{\sum_{\substack{(v_i,v_j)\in E \\ (v_k,v_l)\in E}} Cross(\overline{p_i p_j}, \overline{p_k p_l}) + 1} + \frac{w_7}{\sum_{i=1}^{n}\frac{1}{d_{ic}^2}} + \frac{w_8}{\sum_{i=1}^{n}\frac{(d_{ic} - \frac{1}{n}\sum_{i=1}^{n} d_{ic})^2}{n}} . \tag{1}$$

p_i is the position vector of vertex v_i, d_{ij} is the Euclidean distance between points p_i and p_j and d_{ic} is that between point p_i and the center of the rectangular frame. The first and the second terms make the points distributed evenly and minimize the total edge length of graph G. The value of the first term will decrease if the points in the plane get too close, while that of the second term will decrease if the points get too far. In the third term, $ideal\_edgelength = \sqrt{s/n}$ is the desired edge length, where $s = (d-c)(b-a)$ is the area of the rectangular frame in the plane. The length of each individual edge will be as close as possible to the parameter ideal_edgelength because of the third term, and thus be uniform. $\angle p_j p_i p_k$ in the fourth and the fifth term is the angle between edges incident on the point p_i, degree(p_i) in the fifth term is the vertex degree of point p_i. $Cross(p_i p_j, p_k p_l)$ in the sixth term is defined as formula (2). It can be calculated by means of analytic geometry according to the coordinates of end points of the two straight- line segments $\overline{p_i p_j}$ and $\overline{p_k p_l}$. The seventh and the eighth terms make drawings symmetric if such feature exists. w_i is the weight of criteria and it is a constant. They control the relative importance of the seven criteria and compensate for their different numerical magnitudes. The drawings produced by the algorithm can widely vary by modifying these constants.

$$Cross(\overline{p_i p_j}, \overline{p_k p_l}) = \begin{cases} 0 & \text{If straight-line segments } \overline{p_i p_j} \text{ and } \overline{p_k p_l} \text{ don't intersect .} \\ 1 & \text{If straight-line segments } \overline{p_i p_j} \text{ and } \overline{p_k p_l} \text{ intersect .} \\ \infty & \text{If straight-line segments } \overline{p_i p_j} \text{ and } \overline{p_k p_l} \text{ overlap .} \end{cases} \quad (2)$$

2.3 Selection

In order to avoid premature convergence, we perform a sigma proportional transformation on each individual's fitness value [5], i.e., for the fitness value f(i) of the i-th individual, at first we apply the following formula to f(i) to transform it into ExpVal(i):

$$ExpVal(i) = \begin{cases} 1 + (f(i) - f(t)) / 2\sigma(t), & if \quad \sigma(t) > 0. \\ 1, & if \quad \sigma(t) = 0. \end{cases} \quad (3)$$

where f(t) is the average fitness value of the t-th generation population, and $\sigma(t)$ is the standard deviation of the t-th generation population. After such transformation, the algorithm then uses elitist fitness proportionate selection mechanism for ExpVal(i) to select chromosomes for reproduction. The best individual in the population is always passed on unchanged to the next generation, without undergoing crossover or mutation.

2.4 The Design of Genetic Operator

The algorithm has three types of genetic operations: crossover, mutation and inversion. The crossover operator is defined as follows: The *single-point crossover* generates two new graph layouts by randomly selecting one vertex and exchanging the corresponding coordinates between the parent graphs. The mutation operators are applied sequential and independently from crossover. They are defined as follows:

The *non-uniform mutation*[6] – If $S=(v_1, v_2, \ldots, v_{2n})$ is a chromosome and the element v_k is selected for this mutation (the domain of v_k is $[a_k, b_k]$), the result is a vector $S' = (v_1, v_2, \ldots, v_{k-1}, v'_k, \ldots, v_{2n})$, with $k \in 1, \ldots, 2n$, and

$$v'_k = \begin{cases} v_k + \Delta(t, b_k - v_k), & if \ c = 0, \\ v_k - \Delta(t, v_k - a_k), & if \ c = 1, \end{cases} \quad (4)$$

Where c is a random number that may have a value of zero or one, and the function $\Delta(t, y)$ returns a value in the range [0, y] such that the probability of $\Delta(t, y)$ being close to 0 increases as t increases:

$$\Delta(t, y) = y(1 - r^{(1-t/T)^{\lambda}}) \quad (5)$$

where r is a random number in the interval[0, 1], t is the current generation number, T is the maximum number of generations, and λ is a parameter chosen by the user, which determines the degree of dependency with the number of iterations. This property causes this operator to make an uniform search in the initial space when t is small, and a very local one in later stages.

The *single-vertex-neighborhood mutation*-choose a random vertex and move it to a random point in the circle of decreasing radius around the vertex's original location. Suppose that vertex $v_i(x_i,y_i)$ is chosen for mutation, then the new coordinates (x_i', y_i') of v_i are defined as follows:

$$\begin{cases} x_i' = x_i + r\cos\theta \\ y_i' = y_i + r\sin\theta \end{cases} \tag{6}$$

where radius r=ideal_edgelength*(1-t/T);$\theta \in [0, 2\pi]$ is an angle randomly produced; The meanings of t, T and ideal_edgelength are the same as above. As can be seen, the radius r is decreasing as the algorithm proceeds.

The last genetic operation is inversion. Inversion works by randomly selecting two inversion points within a chromosome and inverting the order of genes between the inversion points, but remembering the gene's meaning or functionality. If

$$S=(v_1,\ldots,v_i,\ldots,v_j,\ldots,v_{2n})$$

is the parent vector and the two inversion points are i and j, then the offspring vector will be

$$S'=(v_1,\ldots,v_{i-1},v_j,v_{j-1},\ldots,v_{i+1},v_i,\ldots,v_{2n})$$

In order to avoid swapping x-coordinates for y-coordinates, we add the following constraint:

$$(j-i)\bmod 2=0$$

2.5 Determining the Termination Condition of the Algorithm

The termination condition is just a check whether the algorithm has run for a fixed number of generations.

3 Experimental Results and Analysis

The algorithm described above was implemented and run on a PC with Celeron 1.7 GHz CPU, 128MB RAM. The experimental parameters values are shown as table 1. The simple genetic algorithm and our algorithm were applied respectively to six test graphs with the number of vertices ranging from 4 to 28. For each class of graphs, the two algorithms were run 20 times, respectively. Table 2 shows the mean fitness value of the two algorithms. As can be seen, our algorithm is much better than the simple genetic algorithm under the same condition. The experimental results are shown as figures 1-8. Figure 1 shows three different outputs of three different algorithms for the same cycle. Figure 1 (a) is the output of the algorithm in bibliography [3]; Figure 1 (b) is that of the algorithm in bibliography [4]; Figure 1 (c) is that of our algorithm. Clearly, Figure 1 (c) is the best because it is a convex polygon while the other two are concave polygons.

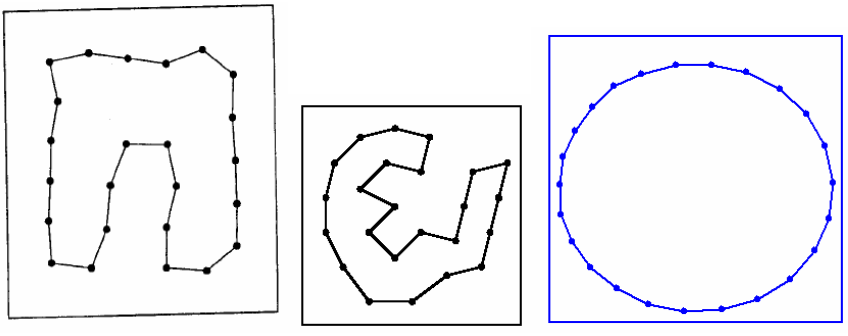
Figure 2 (a) shows a rectangular grid input graph with random locations for the vertices. Figure 2 (b) is the output of the simple genetic algorithm. Figure 2 (c) is that of our algorithm. Figure 2 (c) took 14.000547 seconds using 675 generation. As can be seen, the drawing produced by our algorithm is more beautiful than that produced by the simple genetic algorithm under the same condition.

Table 1. The parameters value of experiment

Parameters	Parameters value
Population size	30
Generation count	1000
Crossover probability	0.75
Mutation probability	0.25
Inversion probability	0.20

Table 2. Compare our algorithm with the simple genetic algorithm

Number of vertices	Simple GA	GA in this paper
4	0.106244	0.236976
7	0.106602	0.224305
11	0.104997	0.215806
16	0.103219	0.222807
25	0.062136	0.200684
28	0.054096	0.200018

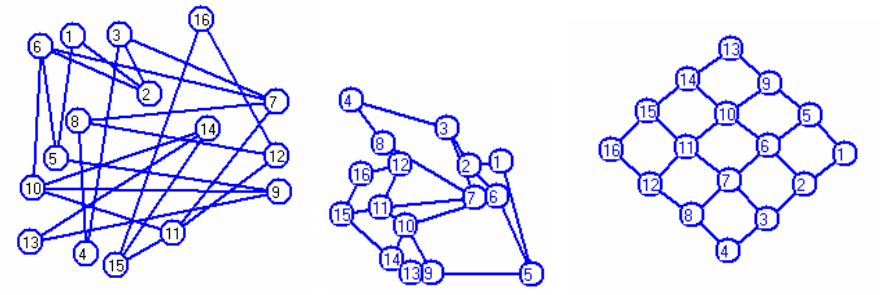


(a) (b) (c)

Fig. 1. Three different outputs of three different algorithms for the same cycle

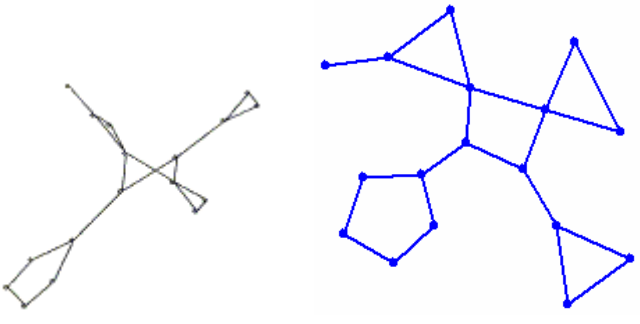
Figure 3 shows two different layouts of the same graph with bridges. Figure 3(a) is the layout produced by the original spring algorithm[2]; Figure 3(b) is the layout produced by our algorithm. Clearly, Figure 3(b) is better than Figure 3(a) for the uniform edge length.

Figure 4 shows two different drawing of the same graph produced by two different algorithms. Figure 4(a) is the drawing produced by the algorithm in bibliography [4]; Figure 4(b) is that produced by our algorithm. Clearly, Figure 4(b) is better than Figure 4(a) because Figure 4(b) has no crossing.



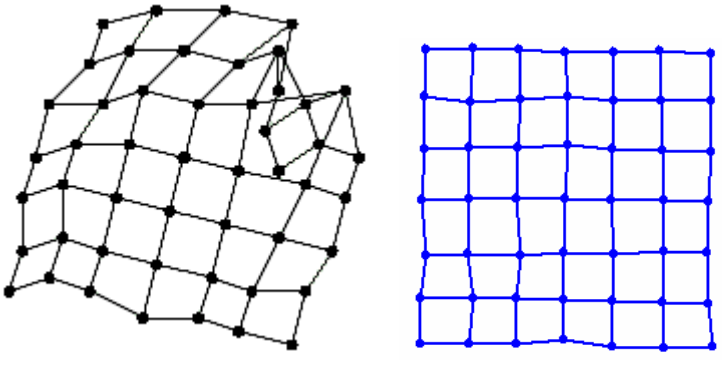
(a) (b) (c)

Fig. 2. A random input and the corresponding outputs of simple genetic algorithm and our algorithm



(a) (b)

Fig. 3. Two different layouts of the same graph with bridges



(a) (b)

Fig. 4. Two different outputs of two different algorithms for the same graph

Figure 5 is an output for a tree. Figure 6 is an output for a disconnected graph. Figure 7(a) shows an output for a rectangular grid graph with 25 vertices and 40 edges. Figure 7(b) is an output for a triangular grid graph with 15 vertices and 30 edges. Figure 8 is other sample outputs of our algorithm.

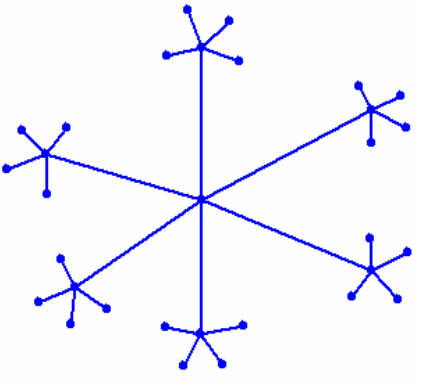


Fig. 5. An output for a tree

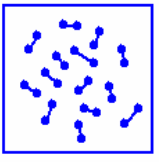
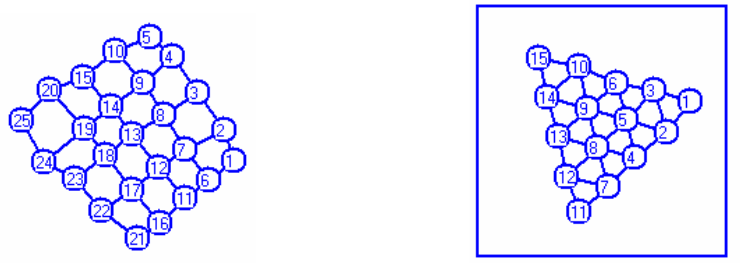


Fig. 6. An output for a disconnected graph



(a) (b)

Fig. 7. Outputs for two grid graphs

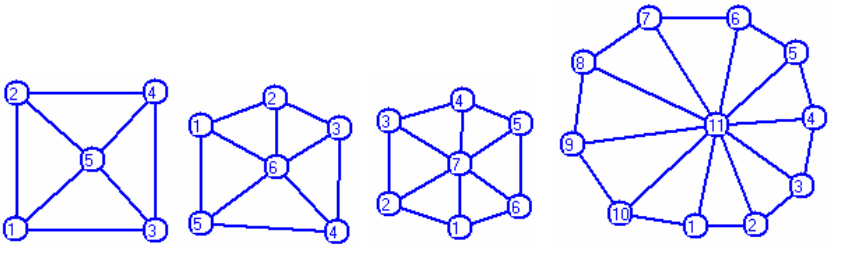


Fig. 8. Simple examples

4 Conclusions and Further Work

This paper has proposed an algorithm for producing aesthetically pleasing drawings of general undirected graphs. The primary advantage of our algorithm is that it can draw large cycles with no chords as convex polygons. This overcomes the disadvantage of previous undirected graph drawing algorithms drawing such type graphs as concave polygons. In addition, it is flexible in that the relative weights of the criteria can be altered. The experiment results show that the figures drawn by our algorithm are beautiful. The weakness of our algorithm is speed (like all that use GAs). The future research, based on bibliography [7], is the evolution (by a GA) of an ideal set of weights for the criteria - reflecting the aesthetic preferences of the user - by learning from examples. Those weights would then be used by the GA to layout graphs which will hopefully be more likely to please such users.

References

1. Tamassia, R., Di Battista, G., Batini, C.: Automatic graph drawing and readability of diagrams. IEEE Transactions on Systems, Man and Cybernetics, Vol. 18, no.1, (1988) 61-79
2. Eades, P.: A Heuristic for Graph Drawing. Congressus Numerantium, 42 (1984) 149-160
3. Davidson, R., Harel, D.: Drawing Graphs Nicely Using Simulated Annealing. ACM Transac-
4. tions on Graphics, Vol. 15, no.4, (1996)301-331
5. Eloranta, T., Mäkinen, E.: TimGA - a genetic algorithm for drawing undirected graphs. Tech-
6. nical report, Department of Computer Science, University of Tampere , December (1996)
7. Tanese, R.: Distributed Genetic Algorithms for Function Optimization. Unpublished doctoral dissertation, University of Michigan, Ann Arbor, MI.(1989)
8. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. 3rd edition. Springer-Verlag, Berlin Heidelberg New York (1996)
9. T.Masui.: Evolutionary learning of graph layout constraints from examples. In Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'94). ACM Press, November (1994) 103-108