DRAWING GRAPHS WITH EVOLUTIONARY ALGORITHMS

Andrea G. B. Tettamanzi[1,2]

[1] Università degli Studi di Milano, Dipartimento di Scienze dell'Informazione
Via Comelico 39, I-20135 Milano, Italy
E-mail: tettaman@dsi.unimi.it
[2] Genetica—Advanced Software Architectures S.r.l.
Viale Monte Nero 68, I-20135 Milano, Italy
E-mail: genetica@tin.it

Abstract. This paper illustrates an evolutionary algorithm for drawing graphs according to a number of esthetic criteria. Tests are carried out on three graphs of increasing difficulty and a comparison is made on the performance of three different recombination operators. The results are then briefly discussed.

1 Introduction

A number of data presentation problems involve the drawing of a graph on a two-dimensional surface, like a sheet of paper or a computer screen. Examples include circuit schematics, communication and public transportation networks, social relationships and software engineering diagrams. In almost all data presentation applications, the usefulness of a graph drawing depends on its readability, i.e. the capability of conveying the meaning of the diagram quickly and clearly. Readability issues are expressed by means of *esthetics,* which can be formulated as optimization criteria for the drawing algorithm [3].

An account of esthetic criteria that have been proposed and various heuristic methods for satisfying them can be found in [20]. An extensive annotated bibliography on algorithms for drawing graphs is given in [9] and [3]. The methods proposed vary according to the class of graphs for which they are intended and the esthetic criteria they take into account. For most reasonable esthetic requirements, however, it turns out that solving this problem exactly is prohibitively expensive for large graphs.

Evolutionary algorithms are a broad class of optimization methods inspired by Biology, that build on the key concept of Darwinian evolution [4, 6]. It is assumed that the reader is already familiar with the main concepts and issues relevant to evolutionary algorithms; good reference books are [19, 12, 5, 17, 16, 1]; [13, 10, 18] are more of a historical interest.

2 The Problem

Given a partially connected graph $G = (V, E)$, we want to determine the coordinates for all vertices in V on a plane so as to satisfy a certain number of esthetic requirements.

The esthetic criteria that were employed in this work are the following:

- there should be as few edge crossings as possible, ideally none;
- the length of each individual edge should be as close as possible to a parameter L;
- the angles between edges incident into the same vertex should be as uniform as possible (ideally the same).

A criterion that is usually considered requires that the vertices be evenly distributed on the available space. In fact, this criterion is entailed by the second and third criteria stated above, since all the edges should be approximately the same length and the edges departing from a vertex should spread as much apart from one another as possible.

2.1 Complexity

In general, the optimization problems associated with most esthetics are NP-hard [14, 15]. For instance, it has been proven that even just minimizing the number of edge crossings is an NP-hard problem [11]. Therefore, the problem described above, which requires in addition to satisfy two other criteria, is also NP-hard, thus providing a valid motivation for resorting to an evolutionary approach.

2.2 Related work

The use of evolutionary algorithms for drawing *directed* graphs, a problem of great practical importance in relation with interactive software tools that use diagrams such as transition or structure diagrams in the form of directed graphs (cf. for instance [8]), has already begun to be explored by Michalewicz [17]. Surprisingly, however, that application disappeared from subsequent editions of his book. Michalewicz's work considers only two criteria, namely that arcs pointing upward should be avoided and there should be as few arc crossings as possible.

Direction of edges is not addressed in the work described here, which deals with other types of graphs arising, for example, when we want to represent on paper telecommunication networks, where links are always bidirectional.

3 The Algorithm

The overall flow of the proposed evolutionary algorithm is the following, which operates on an array *individual*[*popSize*] of individuals (i.e. the population); even though this is not explicitly demonstrated in the pseudo-code for sake of readability, crossover and mutation are never applied to the best individual in the population.

```
SeedPopulation(popSize)
generation := 0
while true do
    for i := 1 to popSize do
        EvaluateFitness(i)
    end for
    Selection
    for i := 1 to popSize step 2 do
        Crossover(i, i + 1, p_cross, σ²_cross)
    end for
    for i := 1 to popSize do
        Mutation(i, p_mut, σ²_mut)
    end for
    generation := generation + 1
end while
```

The various elements of the algorithm are illustrated in the following subsections.

3.1 Encoding

How a graph is drawn on a plane, i.e. a candidate solution to our problem, is completely determined by the (x, y) coordinates assigned to each vertex.

Therefore, a genotype consists in a vector $((x_1, y_1), \ldots, (x_{\|V\|}, y_{\|V\|}))$, where (x_i, y_i) are the coordinates of the ith vertex of the graph, encoded as two integer numbers in $\{-4096, \ldots, 4095\}$, which can be considered the basic *genes* of an individual. This gives a virtual page of $8192 \times 8192 = 67,108,864$ pixels on which the graph can be drawn.

It is worth noticing that this is not a bit-string representation and thus genetic operators always act on pairs of coordinates (or vertex positions) according to their meaning.

3.2 Initialization

Initial vertex positions for each individual are randomly generated independently and according to the same uniform probability over the whole drawing page.

Generation of initial graphs could also be carried out with the help of greedy algorithms, which indeed have been tried, although they are not discussed in this paper.

3.3 Crossover

Three different types of crossover have been experimented with:

- *uniform* crossover, whereby the vertex positions that make up an individual's genotype have the same probability of being inherited from either parent;

- *single point* crossover, where the crossover point cannot split a vertex position into two;
- *convex hull* crossover, where the coordinates of each vertex in the offspring are a linear combination of the coordinates of the same vertex in its parents: suppose that (x_1, y_1) and (x_2, y_2) are the coordinates of a vertex in the two parents, then the same vertex in the offspring will have coordinates (X, Y), where X and Y are two independent, normally distributed random variables with mean respectively $\frac{x_1+x_2}{2}$ and $\frac{y_1+y_2}{2}$ and with the same variance σ^2_{cross}, which is a parameter of the algorithm. This kind of recombination operator is loosely inspired by the *intermediate recombination* operator widely used in evolution strategies [18, 19], which is just the averaging operator.

Whatever the type that is being used, crossover is applied with a given probability, p_{cross} to each couple of individuals in the population.

3.4 Mutation

Mutation perturbs individuals by adding independent Gaussian noise to each of their vertices. If (x, y) is the position of a vertex before mutation, the position of the same vertex after mutation will be given by (X, Y), where X and Y are two independent normally distributed random variables, with mean respectively x and y and with the same variance σ^2_{mut}, which is a parameter of the algorithm, and therefore it remains constant throughout evolution (unless the human operator changes it interactively, a possibility that is given by the software package implementing the evolutionary algorithm).

This mutation operator is very similar to the convex hull crossover described above, the main difference between the two operators being that in the case of crossover both parents participate in setting the mean of the new vertex position distribution; also, we have always used the mutation operator described above with a much bigger standard deviation than the convex hull crossover.

Self-adaptation of the mutation variance [1, 2], one of the crucial properties of evolution strategies, is likely to give good results in this setting and is certainly going to be implemented in the future.

3.5 Fitness

Three factors contribute to determining an individual's fitness, one for each esthetic criterion:

- the number of edge crossings, χ;
- the mean relative square error σ of edge lengths defined as

$$\sigma = \frac{1}{\|E\|} \sum_{e \in E} \left(\frac{\|e\| - L}{L} \right)^2, \tag{1}$$

  where $\|e\|$ is the length of edge e;

– the cumulative square deviation Δ of edge angles from their ideal values, defined as

$$\Delta = \sum_{v \in V} \sum_{k=1}^{N_v} \left(\psi_k(v) - \frac{2\pi}{N_v} \right)^2, \tag{2}$$

where N_v is the number of edges incident into vertex v and the $\psi_k(v)$, $k = 1, \ldots, N_v$, are the angles between adjacent vertices.

An individual's fitness, f, to be maximized, is then calculated as follows:

$$f = a\frac{1}{\sigma + 1} + b\frac{1}{\chi + 1} + c\frac{1}{\Delta + 1}, \tag{3}$$

where a, b and c are constants that control the relative importance of the three criteria and compensate for their different numerical magnitudes. Their values have been empirically determined as $a = 0.02$, $b = 0.8$ and $c = 0.18$ for the experiments described below; however, the reader should be aware that by modifying these constants the drawings produced by the algorithm can widely vary.

It can be easily verified that only in very few special cases can the fitness defined in Equation 3 approach the theoretical maximum of one. This is so because the three criteria employed conflict with each other and cannot in general be all completely satisfied at the same time.

3.6 Selection

Elitist fitness proportionate selection, using the roulette-wheel algorithm, was implemented to begin with, using a simple fitness scaling whereby the scaled fitness $\hat{f}$ is

$$\hat{f} = f - f_{\text{worst}}, \tag{4}$$

where f_{worst} is the fitness of the worst individual in the current population.

Linear ranking selection was tried as well, but the results were comparable to those obtained with fitness proportionate selection; therefore it was not taken into account in the experiments illustrated below.

Overall, the algorithm is elitist, in the sense that the best individual in the population is always passed on unchanged to the next generation, without undergoing crossover or mutation.

4 Experimental Results

The algorithm described above was implemented and run on several Pentium 120 workstations, under the Windows 95/NT operating systems.

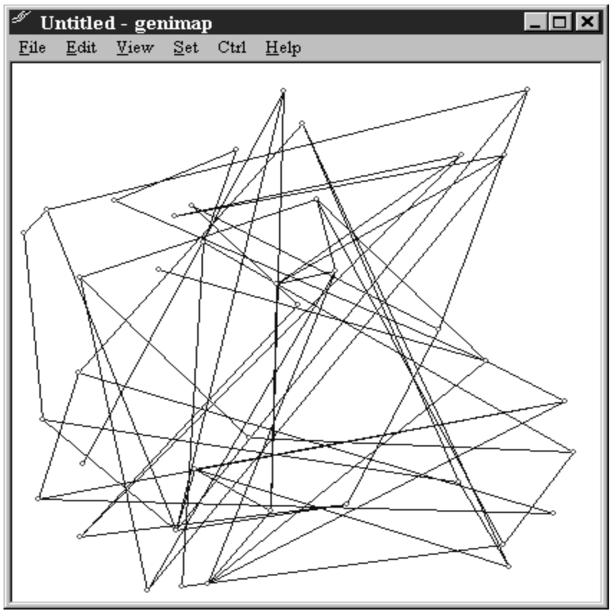

Fig. 1. An example of random layout for Graph G_1, having fitness 0.006193.

4.1 The Test Suite

Preliminary experiments were performed using three test graphs, that will be denoted G_1, G_2 and G_3.

- G_1 represents the road plan of a neighborhood of Milan and contains $\|V_1\| = 39$ vertices and $\|E_1\| = 58$ edges.
- G_2 is a less connected graph of a more irregular nature, having $\|V_2\| = 51$ vertices and $\|E_2\| = 58$ edges.
- G_3 is a graph describing a co-authorship relation among a number of researchers in the field of evolutionary computing. This graph is much bigger than the other two, having $\|V_3\| = 160$ vertices and $\|E_3\| = 165$ edges.

Both G_1 and G_2 were designed by hand so as to be planar and regular. Graph G_3, apart from being much bigger, results to be non-planar. Furthermore it has sets of non-connected vertices.

Figure 1 shows an example of how a randomly generated layout (i.e. a member of the initial population) for the simplest graph G_1 looks like.

The desired length for edges was set to

$$L = \frac{8192}{\sqrt{\|V\|}},$$

which is the average distance between closest vertices when they are uniformly distributed over the whole drawing plane.

4.2 Test Runs

After a first phase during which common optimal values for crossover and mutation rates were roughly established ($p_{\text{cross}} = 0.3$ and $p_{\text{mut}} = 0.15$) for a population size of 100 individuals, four sets of evolutionary algorithm runs were performed for each test graph. The characteristics of each set of runs are the following:

1. single-point crossover;
2. uniform crossover;
3. convex hull crossover with standard deviation $\sigma_{\text{cross}} = 100$;
4. convex hull crossover with standard deviation $\sigma_{\text{cross}} = 200$.

In all four runs, σ_{mut} was set equal to L. All runs were carried on for 10,000 generations, corresponding to an average execution time of about one hour.

4.3 Results

Throughout all the experiments performed, the set of runs featuring convex hull crossover proved significantly better than the other two, getting very close to finding a planar solution for graphs G_1 and G_2.

Tables 1 to 3 compare the performance of the different crossovers on the three test graphs.

Gen. no.	convex 100	convex 200	single-point	uniform
0	0.006133	0.006033	0.007258	0.006193
1000	0.042044	0.036540	0.077847	0.176708
2000	0.069960	0.052057	0.089722	0.217646
3000	0.070431	0.061988	0.091262	0.218769
4000	0.071041	0.063126	0.091565	0.219746
5000	0.071357	0.069572	0.091880	0.220178
6000	0.071468	0.069710	0.092377	0.220233
7000	0.071740	0.070211	0.092476	0.220305
8000	0.074128	0.070427	0.092660	0.220572
9000	0.074448	0.070489	0.092745	0.221003
10000	0.074633	0.070568	0.092809	0.221208

Table 1. Fitness of the best individual during four sample evolutions for Graph G_1 across 10,000 generations.

Gen. no.	convex 100	convex 200	single-point	uniform
0	0.005065	0.005285	0.005448	0.005459
1000	0.104270	0.148000	0.215165	0.283912
2000	0.212809	0.216858	0.216344	0.285629
3000	0.215021	0.284077	0.218658	0.285913
4000	0.416083	0.284694	0.219015	0.286115
5000	0.417016	0.417343	0.219184	0.286291
6000	0.418009	0.417981	0.219327	0.286588
7000	0.418725	0.418200	0.219455	0.286774
8000	0.418951	0.418361	0.219686	0.286876
9000	0.419220	0.418427	0.219897	0.286983
10000	0.419366	0.418585	0.219917	0.287002

Table 2. Fitness of the best individual during four sample evolutions for Graph G_2 across 10,000 generations.

Gen. no.	convex 100	convex 200	single-point	uniform
0	0.001045	0.001125	0.001087	0.001102
1000	0.017012	0.014273	0.017142	0.018259
2000	0.017805	0.015395	0.027558	0.040031
3000	0.018426	0.015815	0.041464	0.079277
4000	0.018536	0.016085	0.059769	0.093941
5000	0.018536	0.016444	0.066907	0.103466
6000	0.018870	0.016444	0.071187	0.115019
7000	0.019059	0.016602	0.071678	0.115365
8000	0.019059	0.016602	0.077160	0.115678
9000	0.019216	0.016882	0.104661	0.115932
10000	0.019404	0.017786	0.104922	0.149165

Table 3. Fitness of the best individual during four sample evolutions for Graph G_3 across 10,000 generations.

A quick inspection of the test data in Tables 1 to 3 points out that uniform crossover dominates single-point crossover. Convex hull crossover with a smaller standard deviation seems to dominate the one with a greater standard deviation. However, no simple ordering appears to hold between uniform (or single-point) crossover and convex hull crossover.

Uniform crossover exhibited a consistently good performance on all the three test graphs, and the best one on Graphs G_1 and G_3. On Graph G_2 convex hull crossover performed remarkably better than both single-point and uniform crossover, but its performance on the other two graphs was disappointing. This fact suggests that more study should be devoted to the tuning of standard deviation in convex hull crossover as a function of graph characteristics. If such a function exists, it is not a simple one, as is the case with mutation, where it was soon clear that standard deviation should be directly proportional to parameter L.

Figures 2 to 4 show the best results, in terms of fitness, obtained by the evolutionary algorithm applied to Graphs G_1, G_2 and G_3. However, compare Figure 3 with the less fit but more pleasing layout found in Figure 5.

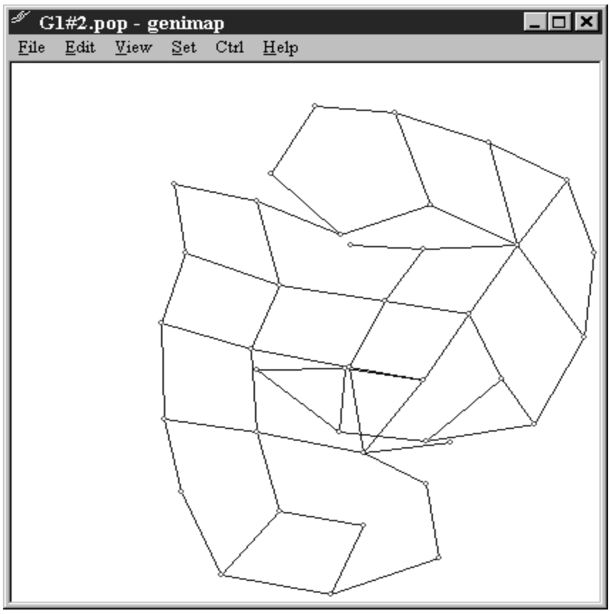

Fig. 2. A layout of Graph G_1 found by the algorithm with uniform crossover after 10,000 generations, having fitness 0.221208.

5 Discussion

There is a troubling aspect with convex hull crossover used in combination with mutation: when genetic diversity in the population is low, i.e. when it is highly likely that the same vertex is placed in the same position by both parents selected for sexual reproduction, that recombination operator turns into a kind of mutation, and it would thus be expected to contribute to maintaining a certain level of diversity, which in turn would avoid premature convergence and allow the algorithm to escape from local minima. Despite of this, however, runs using convex hull crossover got stuck in local minima in two of the three test cases. On the other hand, when genetic diversity in the population is high, convex hull

Fig. 3. A layout of Graph G_2 found by the algorithm with convex hull crossover with standard deviation 100 after 10,000 generations, having fitness 0.419366.

crossover is more likely to cram vertices in the center of the drawing plane; the algorithm then has a hard time spreading them back apart in the right way.

Even though one might think that using evolutionary algorithms for graph drawing represents a departure from classical algorithmic strategies found in computational geometry literature, a number of implicit similarities with well-known classical heuristics can be found.

For instance, including a term inversely proportional to σ as defined in Equation 1 in the fitness function recalls the so-called *spring embedder* force-directed method [7], whereby the drawing process is to simulate a mechanical system, where vertices are replaced by rings and edges are replaced by springs: the springs attract the rings if they are too far apart and repel them if they are too close.

The natural continuation of the work described here would consist in making comparisons with other stochastic algorithms based for example on taboo search or simulated annealing and with relaxation algorithms like the spring embedder mentioned above. For the latter, it would be easy to develop a hybrid evolutionary algorithm incorporating them either as Lamarckian mutation operators or as decoders of the genotype into the actual solution to be evaluated.

A further idea, since the settings of the parameters controlling the relative weights of the esthetic criteria are very arbitrary and the quality of a drawing is

Fig. 4. A layout of Graph G_3 found by the algorithm with uniform crossover after 10,000 generation, having fitness 0.149165.

inherently subjective, would be to allow the algorithm to interact with a human operator, using its responses to optimally tune the parameters relevant to the fitness function.

Acknowledgments

This work was partially supported by M.U.R.S.T. 60% funds. The author thanks Prof. Gianni Degli Antoni for his support and useful discussions and Genetica—Advanced Software Architectures for the extensive use of their computing equipment.

References

1. T. Bäck. *Evolutionary algorithms in theory and practice.* Oxford University Press, Oxford, 1996.
2. T. Bäck, U. Hammel, and H.-P. Schwefel. Evolutionary computation: History and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, 1997.
3. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: An annotated bibliography. Technical report, Available on the Internet, URL: `ftp://wilma.cs.brown.edu/pub/papers/compgeo/gdbiblio.ps.Z`, 1989.

Fig. 5. A layout of Graph G_2 found by the algorithm with uniform crossover after 10,000 generations, having fitness 0.287002.

4. C. Darwin. *On the Origin of Species by Means of Natural Selection.* John Murray, 1859.
5. L. Davis. *Handbook of Genetic Algorithms.* VNR Computer Library. Van Nostrand Reinhold, New York, 1991.
6. R. Dawkins. *The blind Watchmaker.* Norton, 1987.
7. P. Eades. A heuristics for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
8. P. Eades and X. Lin. How to draw a directed graph. Technical report, Department of Computer Science, University of Queensland, Australia, 1989.
9. P. Eades and R. Tamassia. Algorithms for drawing graphs: An annotated bibliography. Technical Report CS-89-09, Department of Computer Science, Brown University, 1989.
10. L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley & Sons, New York, 1966.
11. M. R. Garey and D. S. Johnson. Crossing number is np-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4(3):312–316, 1983.
12. D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley, Reading, MA, 1989.
13. J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, 1975.
14. D. S. Johnson. The np-completeness column: An ongoing guide. *Journal of Algorithms*, 3(1):89–99, 1982.

15. D. S. Johnson. The np-completeness column: An ongoing guide. *Journal of Algorithms*, 5(2):147–160, 1984.
16. J. R. Koza. *Genetic Programming: on the programming of computers by means of natural selection.* The MIT Press, Cambridge, Massachussets, 1993.
17. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Berlin, 1992.
18. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Fromman-Holzboog Verlag, Stuttgart, 1973.
19. H.-P. Schwefel. *Numerical optimization of computer models.* Wiley, Chichester; New York, 1981.
20. R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):61–79, 1988.

This article was processed using the LaTeX macro package with LLNCS style