Evolutionary algorithm for minimizing the number of edge crosses in the graph

Rafał Biedrzycki
Institute of Electronic Systems
Warsaw University of Technology
Nowowiejska 15/19, 00-665 Warszawa, Poland
email: rbiedrzy@elka.pw.edu.pl

April 28, 2005

Abstract

In the article some results of the work on the genetic algorithms applied to graph-drawing problem have been presented. The algorithm has been introduced and the results of its work compared to the standard graph drawing programs. The new approach to the fitting function criterion has been proposed.

1 Introduction

Graph-drawing problems are problems how to draw a graph correctly. One of the most important criterion is to minimize the number of edges cutting each other.

Graph-drawing problems are being studied from a long time, are well documented and have the use in many important practical fields like designing electronic circuits. A lot have been done; however, existing solutions do not fulfill all the requirements: they are not universal enough or are too slowly.

Graph-drawing problems are NP-problems, that is why heuristics is used to help solving them. The most common approach is to replace a graph-drawing problem by a well-defined physics problem. In the work by Kamada and Kawai [7], or [4], [5] the nodes are being replaced by monomial electric charges and edges by springs. The results of their work is that there is less crossings, but they do not focus on that. None of those algorithms even calculates the number of crossings. Those action result that for the same problem the algorithms give different number of crossings as presented in [4], for example for the $K_{8,8}$ problem algorithm [4] achieved 596, algorithm [7] 686 and algorithm [5] 624 crossings. Popular graph-drawing applications like Pajek [1], or GraphViz [8] used those algorithms. As shown on the figure 1 the physical model gave ,,nice-looking" picture, however even for the very simple problem there are a lot more crossings than needed. They exists other approaches, but are less effective than the physics-model approach.

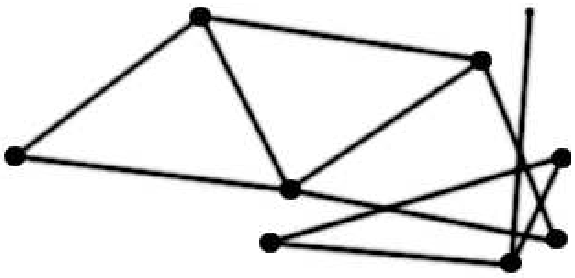

Figure 1: Example output of Pajek software

In this work new method have been introduced without prior knowledge of existing solutions, which enables a fresh view of the field.

2 Assumptions

The assumptions presented in this paper are slightly differ than those made in [5]. The minimization of the edges cutting each other, are stated as the most important. Less important is the assumption of ,,nice-looking" image, which means that the graph should be esthetic for the human eye and properly distributed on the available space. Such defined problem is surely a difficult problem, that is why genetic algorithms were chosen to apply. Because the goal was to get an algorithm as fast and simple to implement as possible, so the decision was not to apply methods considered as proper but used for other different problems.

3 Presented approach

Genetic algorithm has been used to solve problem introduced in previous chapter. Graph is drawn in the space defined by a window in standard graphic interface. Therefore the available space is limited by the size of those window (800 x 600 pixels in the presented experiment). Every node of a graph from this space is defined by a pair of integer numbers. The genotype of each individual represents a specific distribution of nodes in the graph and is implemented as a table of nodes. The individual is being drawn simply by connecting those nodes by lines. The initial population has been stated by experiment as 40 individuals, its nodes coordinates are randomly drawn with the uniform distribution.

3.1 Mutation

All individuals in the basic population are exposed to mutation. With the probability of 1/5 the X coordinate is being changed, exactly as the coordinate Y, both of them at the same time with the probability of 3/5. The value of the perturbation is being chosen with the uniform distribution from the range +/- 0,5 of available space. When attempting of leaving the available space the coordinates are projected to limits.

3.2 Crossover

Every individual from the ,,better" half of the population is exposed to crossover. Its pair is chosen randomly from the whole population. The parents chromosomes are cut randomly into two parts. First part of the first chromosome is being stuck together with the second part of the other chromosome. The same procedure for the other parts. In result we get two descendant individuals. The idea of such crossover is that two ,,well" sub-graph will be connected together. The set of descendants creates the temporary population.

3.3 Succession

The number of surviving individuals is equal to the basic population. Considered are: ,,old" basic population and populations generated by mutation and crossover. Additionally the population is diversified by the age of the individuals. Those older than three generations are dismissed. The method is implemented by the quicksort algorithm.

3.4 Fittness Function

The main criterion is the number of crossings. Therefore it should be accurately computed. The standard approach is to check for each edge if it has any crossings with non-checked edges. The complexity of such problem is $O(n^2)$. Moreover such method gives us no information how the graph will look like. The graph could focus in small part of the available space, the lines could be so close one to another that distinguishing between them would be impossible, the angles could be to small, etc. There are algorithms of time complexity better than $O(n^2)$, but they need more memory and are much more difficult to implement.
To draw graph nicely we should also compute:

- minimal distance between nodes,
- minimal distance between nodes and non-incidental edges,
- minimal angles between incidental edges.[3]

The presented approach is much more simple. To know how many crossings have been generated we only need to draw the graph and compute all the pixels with value grater than zero. The time needed for drawing the graph is a function of total length of edges to draw. The very important result of this approach is that we do not have to worry about complexity.

To draw the graph Bereshman[2] algorithm has been used. To clear the bitmap the graph is drawn using the same color as the screen itself. That is the fastest known approach (table 1).

Table 1: Comparison of time needed for all computations depending on bitmap clearing methods

Bitmap clearing	Time
for loop	> 4 min
operating system function	2 min 40 s
graph the same color as screen	36 s
graph the same color as screen - optimized algorithm	34 s

Because according to our assumptions the most important thing was to avoid crossings, the mechanism for distinguishing between real crossings and ,,false" – those being the result of incidental edges drawn one on another has been developed. An unique number is assigned to each line. By every crossing those numbers are being checked, therefore the false crossings could be detected. The real crossings are punished with much higher value.
Another type of false crossings are the result of meeting many edges in one node. In order to avoid rating those false crossings into result a correction function has been proposed. The correction function is computed as the difference between the number of nodes and the sum of the nodes ranks.

Other advantages being the result of presented method of calculating the fitness function:

- there are no edges drawn one upon another – highly punished, because in such situation there are a lot of pixels with value higher than zero;
- the whole available space is being used, because more densely graphs have more edges very close one to another and those could result in false crosses, therefore such graph are much more punished.

4 Results

The graphs up to 357 nodes and 410 edges have been studied (DNA [1]). However for such big graphs Pajek or GraphViz achieved better results. The presented approach was dealing better with small graphs. The problems are mainly the result of random generation of initial population. As we can see on figure 2, there are many chaotic lines, which are very difficult to improve. Figure 3 presents result made by Pajek, GraphViz and presented approach for the cube. For $K_{3,3}$ problem: figure 4 – drawn by GraphViz, figure 5 – presented approach. For Gr3_44 problem figure 6 drawn by Pajek (left) and drawn by presented algorithm (right). Fitness function without crossover for the best individual – figure 7.

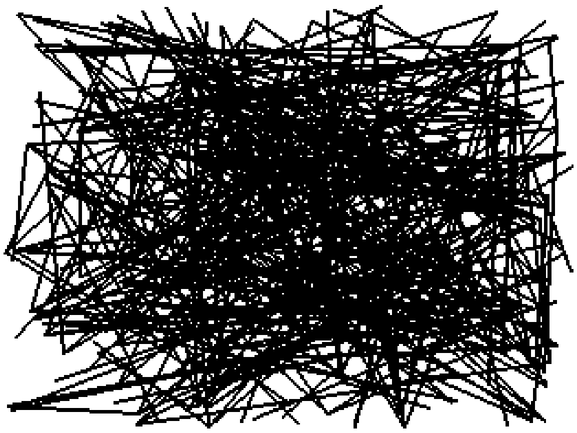

Figure 2: DNA - best individual after first iteration of presented algorithm

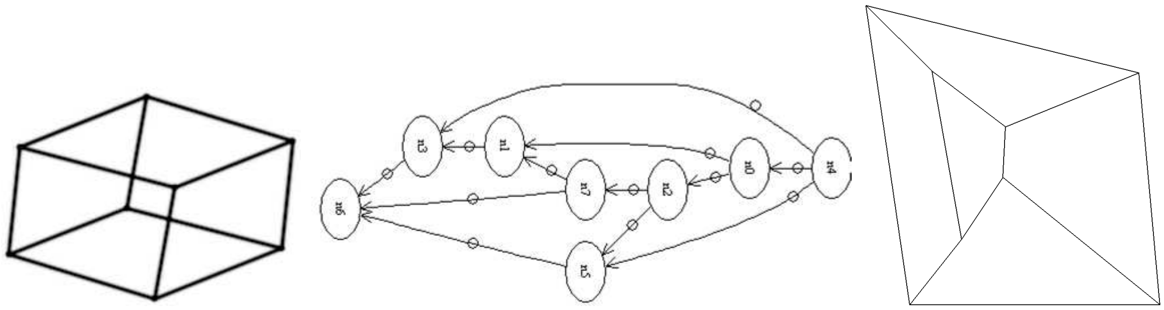

Figure 3: The cube drawn by Pajek (left) GraphViz (center) and presented approach after 30 iterations (right)

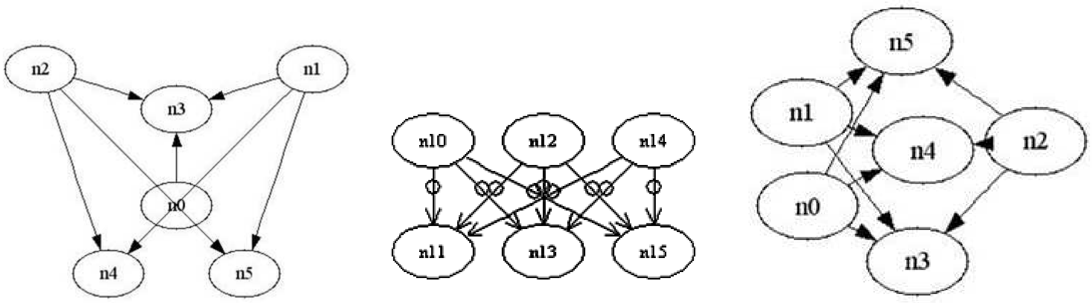

Figure 4: K3,3 – drawn by GraphViz

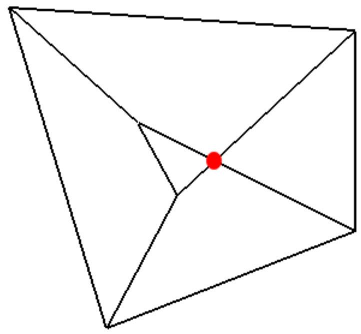

Figure 5: K3,3 – presented approach. Dot marks crossover point

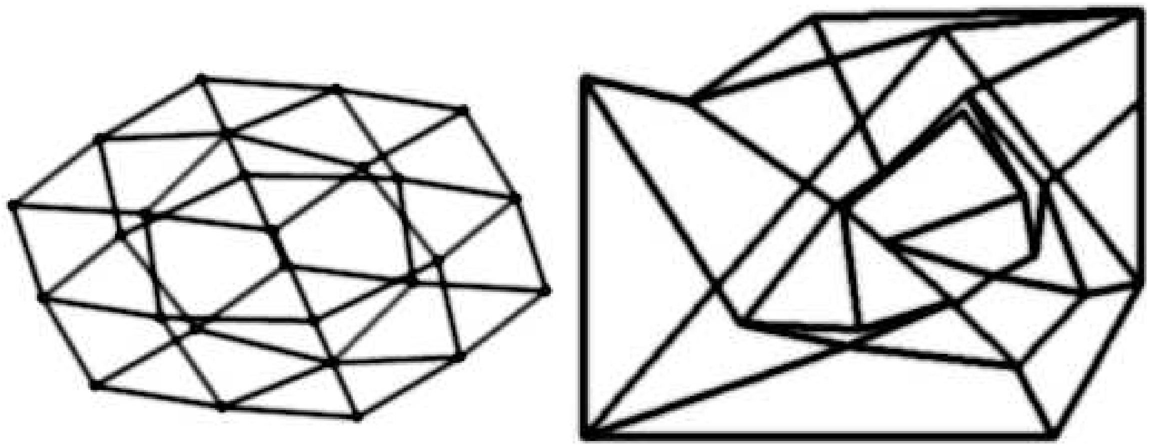

Figure 6: Gr3_44 drawn by Pajek (left) and by presented algorithm (right)

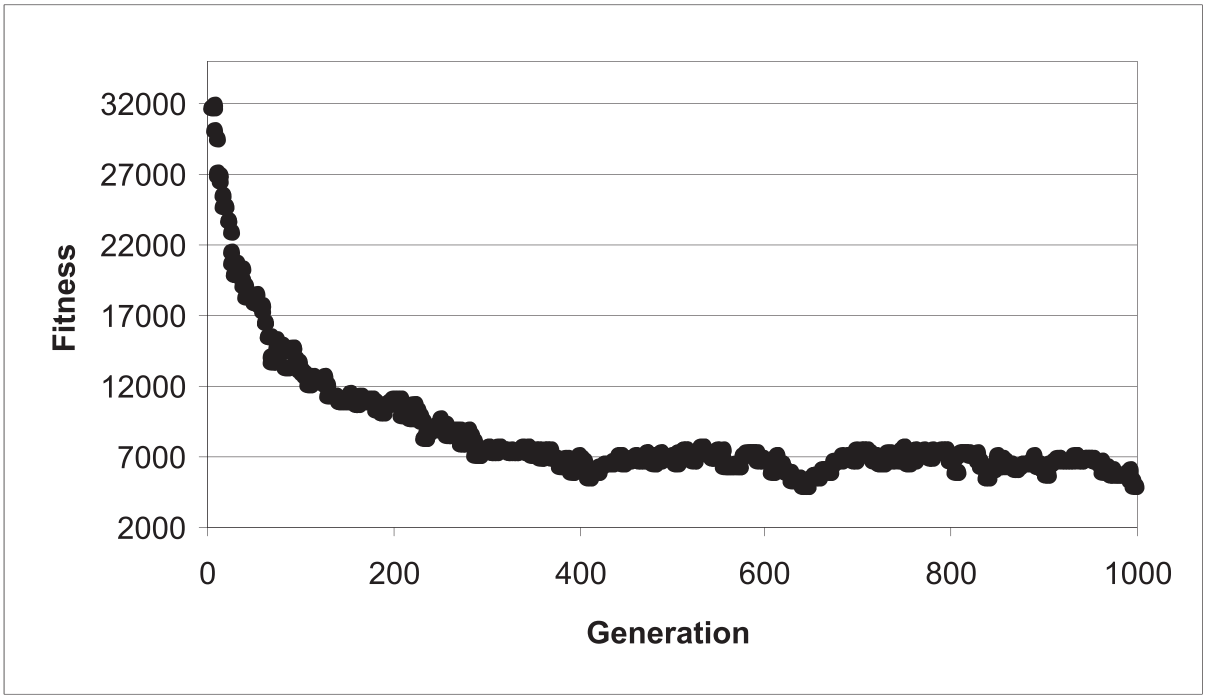

Figure 7: Fitness function without crossover

5 Conclusions

The presented approach is a result of my interest in graph-drawing and is far from being completed. The developed algorithm has many interesting characteristics. I would like to emphasize the original method of calculating fitness function.

Expanding the available space for the graph does not lead to better results. Enabling crossover leads to slightly better results, but it significantly slows down the computations. The best results were achieved for only one mutation at a time even for the big graphs.

Potentially interesting area of future developments could be use some basic graph-drawing algorithms to initialize the starting population. This way the chaotic lines in the first iteration, which are very difficult to improve, will be avoided.
Other possible improvements:

- the physics model may be used for local optimization or for generation of initial population;
- dividing graphs into smaller parts, and optimizing them separately may lead to better results;
- speeding up the computation of individual fitness function by taking into account changes made by mutation without drawing the whole graph.

References

[1] V. Batagelj, A. Mrvar: *Pajek - Program for Large Network Analysis.* http://vlado.fmf.uni-lj.si/pub/networks/pajek

[2] Bereshman Algorithm. http://www.republika.pl/wmula/prog/bereshman.html

[3] F. Cruz, R. Tamassia: *Graph Drawing Tutorial*

[4] A. Frick, A. Ludwig, H. Mehldau: *A Fast Adaptive Layout Algorithm for Undirected Graphs (Extended Abstract and System Demonstation)*

[5] T.M.J. Fruchterman, E.M. Reingold: *Graph draving by force-directed placement.* Software-Practice and Experience, 21, 1991

[6] I. Herman, M. S. Marshall: *Graph Visualization and Navigation in Information Visualization: A Survey.* IEEE Transactions on Visualization and Computer Graphics, vol. 6, no. 1, January-March 2000

[7] T. Kamada, S.Kawai: *An algorithm for drawing general undirected graphs.* Information Processing Letters, 31,1989.

[8] S. North: *GraphViz.* http://www.research.att.com/sw/tools/graphviz/