An Evolutionary Algorithm for Drawing Directed Graphs

Article · December 1999

Source: CiteSeer

CITATIONS
32

READS
928

5 authors, including:

Juergen Branke
The University of Warwick
217 PUBLICATIONS **12,668** CITATIONS
SEE PROFILE

Hartmut Schmeck
Karlsruhe Institute of Technology
272 PUBLICATIONS **5,682** CITATIONS
SEE PROFILE

Peter Eades
The University of Sydney
286 PUBLICATIONS **10,563** CITATIONS
SEE PROFILE

An Evolutionary Algorithm for Drawing Directed Graphs

J. Utech, J. Branke, H. Schmeck *
Institut für Angewandte Informatik und
Formale Beschreibungsverfahren
University of Karlsruhe
76128 Karlsruhe, Germany

P. Eades †
Department of Computer Science
and Software Engineering
University of Newcastle
Callaghan, NSW 2308, Australia

Abstract *The Sugiyama heuristic and its variants are the predominant methods for drawing directed acyclic graphs. This heuristic decomposes the drawing problem into three sub-problems: layering, ordering of the nodes in each layer, and fine-tuning. These sub-problems are then addressed independently. In this paper, we propose an evolutionary algorithm to optimize the layering and the ordering step simultaneously. We show that there are significant interdependencies between those two steps, and that by addressing them together, the evolutionary algorithm is able to produce highly superior results when compared to the Sugiyama heuristic.*

Keywords: evolutionary algorithm, graph drawing, crossing minimization

1 Introduction

The problem of drawing a graph nicely can be regarded as searching for an optimal layout of a given graph according to some measurable aesthetics. However, solving this problem to optimality seems to be computationally infeasible even for relatively simple aesthetic criteria [5]; thus heuristics and stochastic search methods must be used. In recent years, a considerable amount of research on the problem of supporting the automatic construction of pleasing layouts has been done. An extensive survey can be found in [1].

*{jut, branke, schmeck}@aifb.uni-karlsruhe.de
†eades@cs.newcastle.edu.au

The graph layout problem is e.g. motivated by visualization problems in Software Engineering; the graphs are mostly directed and acyclic, since they arise as dependencies between modules.

Since Evolutionary Algorithms (EAs) have shown to be good global optimizers for a broad range of optimization problems, it seems natural to try to apply them to graph drawing as well. And indeed, some examples can be found in the literature [2, 7, 9, 10, 11, 12], although very little of this work mentions directed acyclic graphs.

This paper focuses on drawing *directed acyclic* graphs by evolutionary algorithms. So far, the most widespread heuristic used for drawing acyclic directed graphs is the so called Sugiyama-Heuristic [4]. This heuristic decomposes the problem into three sub-problems that are then solved independently:

1. layering of the nodes and introduction of dummy nodes where edges cross a layer,
2. crossing minimization by ordering the nodes in each layer, and
3. minimizing bends in the edges by fine-tuning the position of the nodes.

For the layering step (1), there are three main methods: the longest path layering, in which the nodes are placed in the lowest possible layer, the Coffman-Graham Algorithm that tries to minimize the height of the drawing given a maximum width of the graph, and a

linear programming method for finding a layering with minimal vertical edge length.

For crossing minimization, usually the Barycenter (BC) heuristic is applied: in each layer, the nodes are positioned according to the average position of their adjacent nodes. Sometimes, this is followed by a greedy heuristic (Greedy Switching (GS)) for local fine-tuning. Due to the space limit, we refer to the literature [4] for a more detailed description of the Sugiyama Method.

Note that most of the optimization problems used in the Sugiyama method are NP complete. Further, they are actually not independent from each other, as can be seen in Figure 1: only if three layers are used, the graph can be drawn crossing-free.

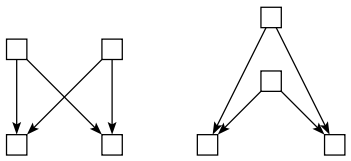

Figure 1: A graph with different layerings

The EA presented in this paper is powerful enough to simultaneously address the layering of nodes as well as the crossing minimization. Therefore it can exploit the interdependencies between these two problems and produce much better layouts.

The outline of the paper is as follows: Section 2 begins with a short introduction to evolutionary algorithms, followed by the details of our approach. Some results are reported in Section 3. The paper concludes with a summary and some suggestions for future work in Section 4.

2 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are stochastic global search methods that have proven to be successful for many kinds of optimization problems.

They work with a population of candidate solutions (individuals) and try to optimize these by means of three basic principles: selection, recombination, and mutation. The initial population is chosen randomly. Then, in every subsequent generation, a parent population is selected from the current population, with higher probability of selection for better individuals. From this parent population, offspring are generated by recombination (combining the genetic information of two parents) and mutation (slight changes of the individual). The offspring population then becomes the current population and the next generation may begin. This basic loop is depicted in Fig. 2.

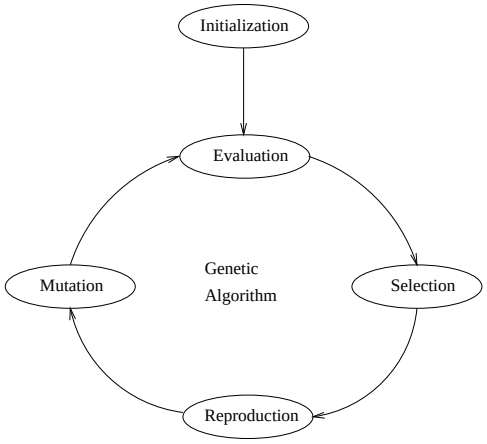

Figure 2: The basic loop of an evolutionary algorithm

Note that mutation and crossover are done on a "genetic" representation of the individual. For most problems, a large variety of reasonable representations and genetic operators exist, and the proper choice can have a significant impact on the optimization behaviour. Thus, when designing an EA for a specific application, the most crucial aspect is to choose a proper representation and suitable genetic operators. In the following subsections we will therefore discuss representation and accompanying operators in more detail. Further parameters of the used EA were: distributed island model (c.f. [13]) with 16 islands, 30 individuals each, linear ranking selection, and generational replacement. For more information on Evolutionary Algorithms in general, the reader is referred to e.g. [6, 11].

2.1 Edge Length Representation

To address the layering problem and the ordering problem together, a chromosome (i.e. the genetic representation) has to determine the layers and the positions within a layer for all nodes, including dummy-nodes (nodes that are inserted wherever an edge crosses a layer). As the work of Michalewicz [7, 11] has shown, a direct encoding of the layer number in the chromosome leads to problems with the construction of genetic operators. To overcome these problems we propose a new representation, the *Edge Length Representation*.

In this representation, a chromosome consists of two parts:

- a part containing all information concerning the nodes of the original graph and
- a part containing the position of necessary dummy nodes within a layer.

The first part is an array of fixed length, which contains an integer value $l(v)$ (the "edge length"), and a real value $p(v)$ (the "position" value) for each node in the graph. $l(v)$ determines the difference between the layer of v ($\lambda(v)$) and the layer of the lowest predecessor of v, and thus uniquely defines the layer of node v. The layers are assumed to be labeled from top to bottom, starting with layer 1. The final layering λ, which assigns to every node v its layer $\lambda(v)$, can then be computed from $l(v)$ by visiting the nodes in topological order:

$$\lambda(v) \quad := \quad l(v) + \max_{u \in N_G^-(v)} \lambda(u)$$

where $N_G^-(v)$ denotes the *inset* of node v, i.e. the set of all direct predecessors of v.

Note that for $l(v) = 1$ the node is placed at the highest layer possible given the precedence constraints.

The position value $p(v)$, a real value between 0 and 1, determines the position of a node within a layer. Once the layering has been constructed from the information given in $l(v)$, all nodes in the same layer are sorted according to their $p(v)$. In the rare case that two nodes in a chromosome have the same position value, they are ordered by their internal node number.

The second part of the chromosome contains the positioning information for the dummy nodes. For each dummy node the source and target nodes of its corresponding edge, its position value $p(v)$ and its position (number) in the source-target path are stored. Note that the second part of a chromosome is of variable length since its length depends on the number of dummy-nodes that have to be created according to the layering information in the first part of the chromosome.

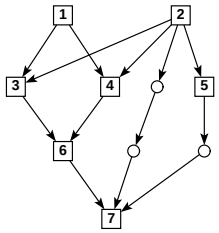

Figure 3: A graph with dummy nodes

An example is given in Figure 3: the dummy node in between nodes 4 and 5 is denoted as $D_7^{2,(1)}$, because it is the first node in the dummy path from node 2 to node 7. Its successor is denoted as $D_7^{2,(2)}$.

The following chromosome is a possible genotype for the graph in Fig. 3:

$$\{\overbrace{(1,0.1)}^{Node1}\overbrace{(1,0.5)}^{Node2}\overbrace{(1,0.2)}^{Node3}\overbrace{(1,0.7)}^{Node4}\overbrace{(1,0.8)}^{Node5}\overbrace{(1,0.3)}^{Node6}\overbrace{(1,0.2)}^{Node7}$$
$$(D_7^{2,(1)}:0.75),(D_7^{2,(2)}:0.4),(D_7^{5,(1)}:0.8)\}$$

The only nodes which need a special treatment are the source nodes of the graph, because they do not have predecessors. There are two different possibilities:

1. The source nodes are all by definition placed in the highest layer. They are omitted in the first part of the chromosome and all algorithms have to take into consideration that $\lambda(v)$ is the highest layer if v is a source node.
2. The source node is placed in layer $l(v)$.

This allows sources, which sit in others than the topmost layer. Since $\max_{u \in N_{\bar{G}}^-(v)} \lambda(u) = 0$ for $N_{\bar{G}}^-(v) = \emptyset$ this alternative is naturally included in the formula for $\lambda(v)$.

In the experiments presented below, alternative 2 is used.

2.1.1 Crossover Operators

The first part of this representation allows classic one- and two-point crossovers without any special repair operators. In fact, every sequence of integers $l(i)$ and real numbers $p(i)$ of length $|V|$ represents a valid chromosome and therefore a valid layering of the graph.

However, after each crossover step, the set of dummy nodes in the variable-length part of the chromosome has to be recomputed. This is done by the following operation:

For each edge, which crosses a layer, add the necessary dummy nodes to the child chromosome.

If at least one parent has a corresponding dummy node (same label), inherit the position value of this node from a parent. If neither of the parents has a matching dummy node (same label), initialize the weight of the new dummy node to a random real value between 0 and 1.

2.1.2 Mutation Operator

For all results shown in Section 3 two different mutation operators were used, independently with probability of 20% each. The first mutation operator – it is called "Shake" – chooses a few nodes of the graph randomly and slightly modifies their $l(v)$ and $p(v)$ values. If necessary, corresponding dummy nodes are inserted into or deleted from the second part of the chromosome.

The second mutation operator is designed to incorporate the knowledge of the well-known barycenter heuristic into the genetic algorithm. This operator, which we call "BC-Walk", does not change the assignment of nodes to layers, but moves nodes within a layer closer to their successors or predecessors in the graph.

MUTATION BC-WALK: Choose a direction (either Up or Down) randomly. Choose a node v randomly. If the chosen direction is Up, set node v's position to the arithmetic mean of the position of all its successors, otherwise set its position to the arithmetic mean of the position of all it predecessors. Randomly choose a non-dummy predecessor (Up-Walk) or successor (Down-Walk) and continue until a source node (or a sink) has been found.

3 Results

To test our approach, we compared it systematically on a variety of graphs against a number of common heuristics available.

Apart from our EA, in our tests the following two combinations of heuristics performed best:

- longest path layering (LP) and barycenter (BC), followed by greedy switching (GS) for ordering
- Coffman-Graham (CG) for layering and BC, followed by GS for ordering

As baseline for comparison, the results of LP and BC alone are also reported.

As test graphs, we generated graphs with the number of nodes ranging from 15 to 40 nodes, and densities of 2%, 5%, and 10% (density is defined here as the actual number of edges divided by the maximum number of edges). For each class of graphs, 6 random instances were created. Each EA-run has been repeated 4 times with different random seeds.

Table 1 shows the relative number of produced crossings of the three tested approaches, with a combination of LP and BC as baseline (100%).

As can be seen, the EA is able to cut down on the number of crossings by about 65% on average, ranging form 33% on large, high-density graphs, up to 98% on small, low-density graphs, when it is compared to the standard LP+BC heuristic. Even when compared to the best (so far) heuristic over all tests

Table 1: Crossing numbers of various algorithms (relative to longest path layering and barycenter heuristic (=100%))

		density 2%	5%	10%	(all)
n=15	LP & BC+GS	80.8	80.1	64.1	75.0
	CG & BC+GS	117.3	54.2	132.9	101.5
	EA	1.9	6.5	40.3	16.3
n=20	LP & BC+GS	47.1	87.7	74.5	69.8
	CG & BC+GS	200.2	158.5	107.1	155.3
	EA	19.9	33.9	56.1	36.6
n=25	LP & BC+GS	92.0	78.1	81.9	84.0
	CG & BC+GS	85.0	75.2	93.6	84.6
	EA	3.5	36.0	74.4	37.9
n=30	LP & BC+GS	82.2	87.0	86.3	85.0
	CG & BC+GS	80.9	113.7	83.5	90.8
	EA	31.8	46.4	67.4	48.7
total	LP & BC+GS	75.5	83.22	76.6	
	CG & BC+GS	120,85	100,4	132,9	
	EA	14.3	30.7	59.6	

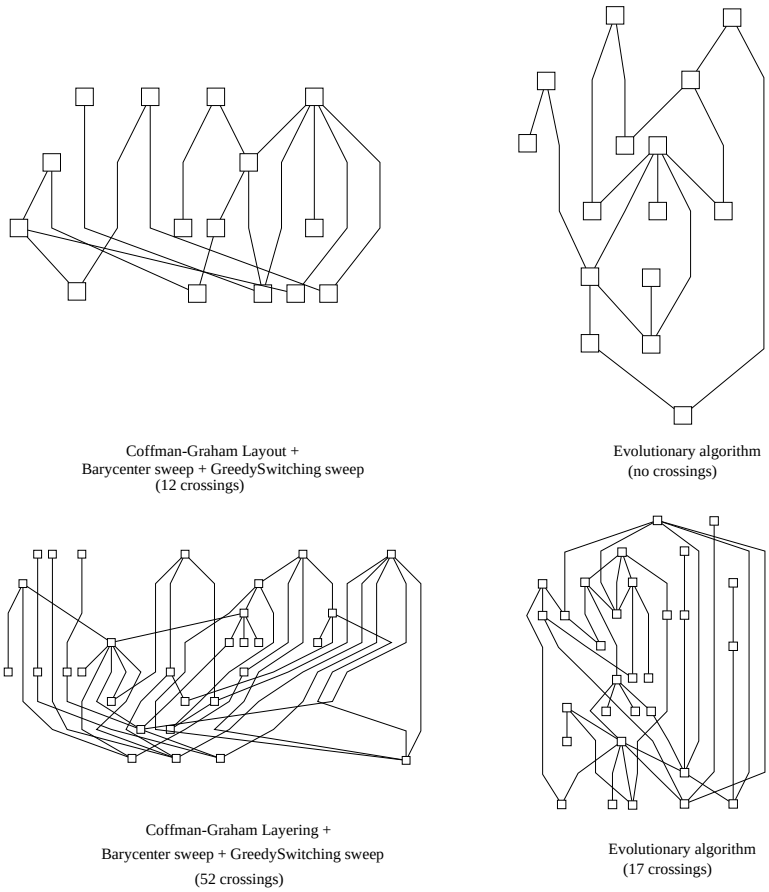

Figure 4: Sample graphs, produced by CG+BC+GS and the EA respectively

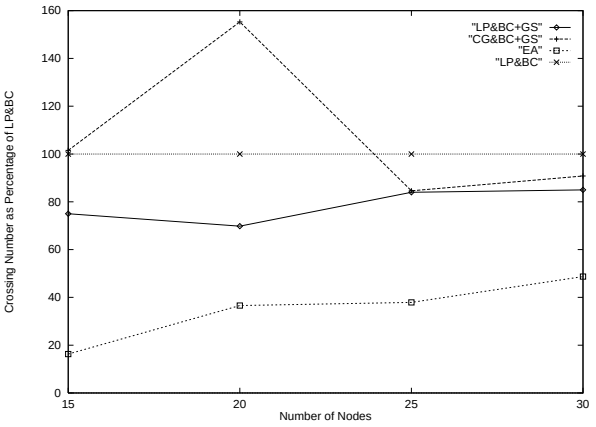

Figure 5: Performance of heuristics depending on the graph's size

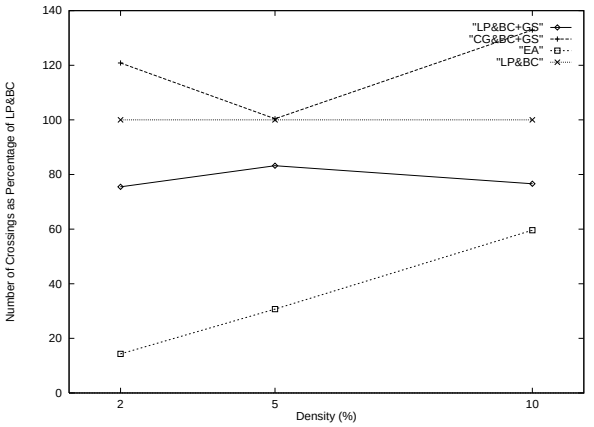

Figure 6: Performance of heuristics depending on the graph's density

(LP&BC+GS), the EA produced 45% fewer crossings on average.

The EA's superiority is also clearly visible in the produced final layouts. Figure 4 compares the results of CG+BC+GS and of the EA, for a smaller and a larger graph.

The results of Table 1 are visualized again in Fig. 5 and 6, where the different heuristics are compared depending on the graph's size resp. density. Note that for growing edge densities, the EA has less potential to further reduce the number of crossings. One reason is the high number of unavoidable crossings (which makes relative improvements more difficult). Also, high densities for the overall graph imply high densities in each layer, and at least for the two-layer case it has been shown [3, 8] that with growing density the solutions of all heuristics converge towards the optimum.

4 Conclusion

We have proposed an evolutionary algorithm for the combined problem of layering and ordering the nodes of an acyclic, directed graph. To our knowledge, this has been the first attempt to address these problems simultaneously, and to exploit the interdependencies between them.

Systematic experiments were made on a variety of randomly generated graphs. On all test instances, when compared to other heuristics, the proposed EA was able to vastly reduce the number of crossings, sometimes even to eliminate them completely.

The main disadvantage of the presented approach is the comparatively long running time, so there is the usual tradeoff between quality and running time.

For future work, we are planning to exploit another advantage of evolutionary algorithms: the fact that they can work with almost arbitrary objective functions. Although minimizing the number of crossings in the drawing seems to be the most important optimization criterion, other criteria may be taken into account. For example, we could additionally focus on the total path length, or the deviation from a given width-to-height ratio. Such aesthetics may improve the resulting drawing at no significant computational cost. In addition, user specific preferences could be incorporated easily by a mere change of the objective function.

Also, since the greedy switching heuristic seemed to achieve some improvements when applied after LP or CG, we hope to get some further improvements by using it after running the EA.

References

[1] G.D. Battista, P. Eades, R. Tamassia, and I.G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Discrete Geometry: Theory and Applications*, 4:235–282, 1994.

[2] J. Branke, F. Bucher, and H. Schmeck. A genetic algorithm for drawing undirected graphs. In J. T. Alander, editor, *Proceedings of the Third Nordic Workshop on Genetic Algorithms and their Applications*, pages 193–206, ftp://ftp.aifb.uni-karlsruhe.de/pub/jbr/gagd.ps.gz, 1997. University of Vaasa.

[3] P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *ARS Combinatoria*, 21-A:89–98, 1986.

[4] Peter Eades and Koizo Sugiyama. How to draw a directed graph. *Journal of Information Processing*, 13(4):424 – 437, 1990.

[5] M. R. Garey and D. S. Johnson. Crossing number is np-complete. *SIAM J. Alg. Disc. Methods*, 4:312–316, 1983.

[6] D. E. Goldberg. *Genetic Algorithms*. Addison-Wesley, 1989.

[7] L. J. Groves, Z. Michalewicz, P.V. Elia, and C.Z. Janikow. Genetic algorithms for drawing directed graphs. In *Proceedings of the Fifth Int. Symposiom on Methodologies for Intelligent Systems*, pages 268–276. Elsevier North-Holland, 1990.

[8] J. Jünger and P. Mutzel. 2-layer straigtline crossing minimization: Performance of exact and heuristic algorithms. In F. J. Brandenburg, editor, *Symposium on Graph Drawing*, number 1027 in LNCS, pages 337–348. Springer Verlag, 1995.

[9] C. Kosak, J. Marks, and S. Shieber. A parallel genetic algorithm for network-diagram layout. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 458–465, 1991.

[10] Corey Kosak, Joe Marks, and Stuart Shieber. Automating the layout of network diagrams with specified visual organization. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(3):440–454, 1994.

[11] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*, pages 239–250. Springer Verlag, 2nd edition, 1994.

[12] A. O. Rodriguez and A. R. Suarez. Automatic graph drawing by genetic search. In *ISPE/IEE/IFAC International Conference on CAD/CAM, Robotics and Factories of the Future, Pereira, Colombia*, August 1995.

[13] T. Starkweather, D. Whitley, and K. Mathias. Optimization using distributed genetic algorithms. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, pages 176–185, Berlin, 1990. Springer Verlag.