Genetic Algorithms for Drawing Directed Graphs

Lindsay J. Groves* Zbigniew Michalewicz† Paul V. Elia† Cezary Z. Janikow‡

Abstract

Genetic algorithms are adaptive algorithms that find solutions to problems by an evolutionary process based on natural selection. They can be used to find approximate solutions to optimization problems in cases where finding the precise optimum is prohibitively expensive. Drawing a directed graph can be viewed as a problem of optimizing the layout of nodes and arcs on a page according to certain aesthetic criteria characterizing "good" drawings of graphs. This paper discusses the use of genetic algorithms for drawing graphs, describing some experiments with two systems we have developed.

1 Introduction

There are many interesting optimization problems for which no reasonably fast algorithms are known. In some applications, however, a near optimal solution is acceptable if it can be computed reasonably quickly. Genetic algorithms ([Davis, 1987], [Goldberg, 1985], [Holland, 1975]) are a class of probabilistic algorithms that can find approximate solutions to optimization problems by starting with a population of randomly generated candidates and "evolving" towards a solution by applying genetic operators, modelled on genetic processes occurring in nature. Given sufficient time, a genetic algorithm can find solutions as close to the real optimum as desired.

Drawing a directed graph can be considered as a problem of finding a layout of nodes and arcs on a page which is optimized according to certain aesthetic criteria chosen to characterize "good" drawings of graphs. For most reasonable aesthetic criteria, however, finding an exact solution for large graphs turns out to be prohibitively expensive. The problem of drawing directed graphs has become of great practical importance with the recent development of interactive software tools. Such systems often use diagrams such as transition diagrams or structure diagrams, that are essentially some form of directed graph, to represent and illustrate various aspects of software systems. Since these diagrams are often generated or modified by the system, the system must be able to display the resulting diagrams in an intelligible fashion. In these systems, ensuring reasonable response time is important and a layout that is nearly optimal is generally quite acceptable. It is therefore desirable to investigate methods for finding approximate solutions.

In this paper we investigate the use of genetic algorithms for drawing directed graphs. Our motivation for this work is twofold. On one hand, we are interested in the design of interactive software tools of the kind mentioned above. On the other hand, we are interested in genetic algorithms and have recently been investigating the application of such to a variety of optimization problems ([Vignaux & Michalewicz, 1989a], [Vignaux & Michalewicz, 1989b], [Michalewicz, Vignaux & Groves, 1989], [Michalewicz *et al*, 1989]). This work has lead to a number of proposals

*Department of Computer Science, Victoria University, Wellington, New Zealand.
†Department of Computer Science, University of North Carolina, Charlotte, U.S.A.
‡Department of Computer Science, University of North Carolina, Chapel-Hill, U.S.A.

for improving the performance of genetic algorithms and a proposal for a genetic programming environment.

The paper is organised as follows. In Section 2 we briefly describe genetic algorithms, identifying the major components of a genetic algorithm and describing the representation and genetic operators used in classical work on genetic algorithms. In Section 3 we discuss the problem of finding a layout for drawing a directed graph, describing the approach taken in [Eades & Lin, 1989], on which our work is based, and presenting some general aspects of our formulation of graph drawing in terms of genetic algorithms. In Sections 4, 5 and 6 we describe two systems we have implemented that use genetic algorithms to find layouts for directed graphs. In Section 7 we discuss some experiments we have performed with these systems and present our results. Section 8 presents our conclusions.

2 Genetic Algorithms

The general principle underlying genetic algorithms is that we maintain a population of possible solutions (candidates) for a given problem, which undergoes a "natural" evolutionary process. In each generation, relatively "good" solutions reproduce, while relatively "bad" solutions die and are replaced. To distinguish between "good" and "bad" solutions we need some kind of evaluation function which plays the role of the environment. As stated in [Davis, 1987]:

> "... the metaphor underlying genetic algorithms is that of natural evolution. In evolution, the problem each species faces is one of searching for beneficial adaptations to a complicated and changing environment. The 'knowledge' that each species has gained is embodied in the makeup of the chromosomes of its members. The operations that alter this chromosomal makeup are applied when parents reproduce; among them are random mutation, inversion of chromosomal material, and crossover — exchange of chromosomal material between two parents' chromosomes."

The general structure of a simple genetic algorithm is shown in Figure 1, in which t is the generation number, and $P(t)$ is the population during generation t.

```
procedure genetic algorithm
begin
    t ← 0
    Initialize P(t)
    Evaluate P(t)
    while (not Termination-condition) do
        t ← t + 1
        Select P(t) from P(t − 1)
        Recombine P(t)
        Evaluate P(t)
    end
end
```

Figure 1: General structure of a genetic algorithm.

To construct a genetic algorithm for a particular problem, we need to specify the following four components:

1. A genetic representation for possible solutions to the problem.
2. A way to create an initial population of solutions.
3. An evaluation function that plays the role of the environment, rating solutions in terms of their "fitness".
4. Genetic operators that alter the composition of children during reproduction.

When the genetic algorithm is run, we also need to specify values for various parameters such as population size, mutation rate, probabilities of applying genetic operators, *etc.* In an experimental system, these can be changed for each run.

Most work on genetic algorithms (*e.g.* [Holland, 1975], [Holland, 1986]) uses bit strings (*i.e.* vectors of 0s and 1s) to represent solutions, and uses the genetic operators mutation, inversion and crossover which operate on bit string representations of candidate solutions.

The *mutation* operator arbitrarily alters one or more components of the chromosome of the selected candidate — this increases the variability of the population. Each bit position of each vector in the new population undergoes a random change with the probability specified by the mutation rate.

The *crossover* operator combines the features of two parents to form two similar offspring, by swapping corresponding segments of the parents' chromosomes.

This approach works well for many problems, such as the Prisoner's Dillema game discussed in [Axelrod, 1987]. In many of the problems we have investigated, however, we have discovered that more elaborate representations and genetic operators are required, and that additional constraints may need to be applied to candidates generated to ensure that valid solutions are obtained ([Vignaux & Michalewicz, 1989a], [Vignaux & Michalewicz, 1989b], [Michalewicz, Vignaux & Groves, 1989], [Michalewicz *et al*, 1989], [Janikow & Michalewicz, 1990]).

3 Drawing Directed Graphs

A large number of algorithms have been proposed for drawing graphs. The kinds of algorithms used, and their costs, vary according to the class of graph for which they are intended (*e.g.* trees, planar graphs, hierarchic graphs or general undirected graphs), the aesthetic criteria they consider, and the methods they use for optimizing the layout. In most cases, finding optimal layouts for large graphs is prohibitively expensive, so a number of heuristic methods have been investigated that find approximate solutions in a reasonable amount of time. A good discussion of the problem of drawing graphs, aesthetic criteria that have been considered, and various methods that have been proposed is given in [Tamassia, Di Battista & Batini, 1988]. A more extensive bibliography is given in [Eades & Tamassia, 1989].

Eades and Lin [1989] discuss an approach to drawing directed graphs, based upon the following three aesthetic criteria[1]:

C_1 Arcs pointing upward should be avoided.
C_2 Nodes should be distributed evenly over the page.
C_3 There should be as few arcs crossing as possible.

[1]The first criterion assumes that the preferred direction for arcs is downwards, this can easily be modified so that the preferred direction is left to right.

Eades and Lin's approach works in three stages, addressing each of these criteria in turn. At each stage the problem of satisfying the corresponding criterion is formulated as a graph theoretic problem.

In the first stage, the graph is turned into an acyclic directed graph (DAG) by reversing some of the arcs. The resulting DAG can be arranged so that all arcs point downwords by performing a topological sort. The DAG thus obtained is then used as the input to stages 2 and 3, and the arcs that were reversed are restored to their proper direction after all three stages have been completed. The major problem at stage 1 is to find a minimal set of arcs to reverse; this is called the "feedback arc set problem" and is known to be NP-hard [Carey & Johnson, 1979].

In the second stage, nodes are distributed evenly on the page. This is done by arranging the graph in layers and allocating nodes to layers in such a way as to match the height and width of the graph to the size of the page. This is equivalent to the "multiprocessor scheduling problem" and is also known to be NP-hard [Carey & Johnson, 1979].

In the third stage, nodes are ordered within layers so as to minimize the number of arc crossings. This problem is also NP-hard, even when there are only two layers in the graph.

Since all three stages are NP-hard, Eades and Lin discuss a number of heuristics that can be used at each stage and discuss some of their properties. At stage 3 they add dummy nodes to arcs spanning more than one layer, so that stage 3 can be done a layer at a time. In the resulting graph, any bends in arcs occur at dummy nodes.

4 Using Genetic Algorithms to Draw Directed Graphs

In order to investigate the applicability of genetic algorithms to the problem of drawing directed graphs, we have implemented two systems, called GRAPH-1 and GRAPH-2, which use genetic algorithms to find layouts for directed graphs. We have based our approach loosely on that described by Eades and Lin, but for our initial implementations we have only considered aesthetic criteria C_1 and C_3. The way in which we approach C_1 does not place all nodes on separate layers, so we do some of what their stage 2 does. At present, however, we do not attempt to spread nodes evenly on the page; we are currently developing an additional procedure to address this.

GRAPH-1 tries to optimize the layout of the graph according to both C_1 and C_3 at once, as in classical genetic algorithms. GRAPH-2 tackles the problem in two phases, first optimizing according to criteria C_1, then optimizing according to criteria C_3. This is closer to Eades and Lin's approach, and leads to a genetic algorithm for an optimization task with a decomposed evaluation function [Janikow & Michalewicz, 1990]. Both systems use "unconvential" genetic algorithms, with non bit-string representations and advanced genetic operators (see [Michalewicz, Vignaux & Groves, 1989]).

In formulating the graph drawing problem, we consider the page to be divided into a number of squares. The graph will be drawn with each node in one square and arcs drawn as straight lines between squares. The graph is assumed to have N nodes, numbered 1 through N, and the page is assumed to be H squares high and W squares wide. The size of square used is arbitrary, but it determines the "granularity" of layouts considered and hence the precision with which a near optimimal layout can be obtained.

In constructing evaluation functions for our systems, we need to consider, for a given candidate, the number of arcs that do not point downwards and the number of arc crossings. In order to allow more possible solutions to be explored, we consider arcs pointing upwards and arcs pointing horizontally separately. In GRAPH-2, we will also need to consider whether two

nodes occupy the same square on the page. These are calculated, for a candidate M, by the following functions:

- $n_u(M)$ is the number of upward pointing arcs in M,
- $n_h(M)$ is the number of horizontal arcs in M,
- $n_c(M)$ is the number of crossings between arcs in M, and
- $n_o(M)$ is the number of nodes in M which occupy the same square.

Each evaluation function will be a linear combination of some of these functions, with weightings chosen according to the penalty to be associated with each feature.

5 GRAPH-1

In GRAPH-1 we use perhaps the most natural data structure for representing a solution for the problem of graph layout — an $H \times W$ matrix whose elements correspond to the squares on the page. Each matrix element is set to n if node n is located in the corresponding square on the page, and 0 if that square is empty. Figure 2 shows a graph, G_1, used in our experiments; Figure 3 shows the genetic representation used by GRAPH-1 for this graph when the page size is 6×10.

Every candidate in the initial population has each natural number between 1 and N, inclusive, assigned to a random element in the matrix, and 0s in the rest. In order to avoid invalid solutions being generated, every candidate generated must satisfy the constraint that each natural number between 1 and N, inclusive, occurs exactly once in the matrix.

Figure 2: Example graph, G_1.

The evaluation function used in GRAPH-1, for matrix M, is:

$$Eval(M) = a_u \cdot n_u(M) + a_h \cdot n_h(M) + a_c \cdot n_c(M) + 1$$

Figure 3: Genetic representation of G_1 for a 6×10 page.

The coefficients a_u, a_h, and a_c determine the significance of upward pointing, horizonal and crossing arcs, respectively. The minimum value of this function is 1, which occurs when all arcs point downward and no arcs cross. In our experiments, we have used $a_u = 2$, $a_h = a_c = 1$.

The genetic operators used in GRAPH-1 are obtained by extending the ideas underlying the three classical bit-string operations described in section 2 to the matrix representation in various ways. All of the operators preserve the constraint mentioned above.

Swap-rows: Select two arbitrary rows in the matrix and swap them.

Swap-columns: The same operator for columns.

Swap-in-row: Select a row; if the row contains at least two nodes, select two nodes and swap them.

Swap-in-col: The same for columns.

Single-mutate: Relocate a single node in a matrix. This preserves the intuition behind the classical mutation operator which is to perform the smallest possible change.

Small-mutate: Select two entries in a matrix and swap them.

Large-cont-mutate: Select two disjoint parts of a matrix, each consisting of contiguous rows and columns, and swap them.

Invert-a-row: Reverse the order of elements in a row.

Invert-a-column: Reverse the order of elements in a column.

Invert-part-row: Reverse the order of some elements in a row.

Invert-part-column: Reverse the order of some elements in a column.

Crossover: Select some rows and columns; for each row i and column j selected, swap the corresponding elements of the two parent matrices. Following this operation, some repairs may be necessary to ensure that the resulting matrices are valid solutions (*i.e.* that they satisfy the constraint mentioned above).

Cont-crossover: Same as the previous operator, but selected rows and columns constitute contiguous sets.

6 GRAPH-2

In GRAPH-2 we split the optimization task into separate phases. The first phase only considers the directions of arcs — whether they point upwards, downwards or horizontally. Once a satisfactory population is found, it is used to generate the initial population for the second phase, which only considers whether arcs or nodes cross. Once a satisfactory population is found here, an evalution function which is a combination of the two previous ones is used to determine the best candidate for the ultimate layout for the output graph.

The data structure used in GRAPH-2 is a $2 \times N$ matrix, in which the i-th column represents the (x, y) coordinates of node i. Phase 1 ignores the x coordinates, while phase 2 utilizes both coordinates. Figure 4 shows another graph, G_2, used in our experiments; Figure 5 shows the genetic representation of this graph used by GRAPH-2 when the page size is 5×9.

Every candidate in the initial population has each node allocated to a random square on the page. At no point in our algorithm do we ensure that every node is allocated to a different square. Later, we will see why we do not need to explicitly preserve this constraint.

Figure 4: Example graph, G_2.

Figure 5: Genetic representation of G_2 for a 5×9 page.

6.1 Phase 1: Avoiding arcs that point upwards

As stated earlier, phase 1 only considers the direction of arcs (whether they point upwards, downwards or horizontally). Consequently, the evaluation function used in phase 1 is:

$$Eval_1(M) = a_u \cdot n_u(M) + a_h \cdot n_h(M) + 1$$

Again, in our experiments, we have used $a_u = 2$ and $a_h = 1$.

The genetic operators used at this stage are variations of the three classical operators:

Y-swap: Swap the y coordinates of 2 random nodes within a structure.

Y-crossover: Crossover the y coordinates of a random node from one structure with the corresponding node of another structure.

Y-mutate: Create a new random y coordinate for any node of a structure.

Note that these operators do nothing to guarantee that two nodes do not occupy the same square on the page. They simply help us to search for a low-cost arrangement of nodes along the vertical dimension. The Y-swap operator alters two nodes which, if their y coordinates were swapped, would reduce the evaluation cost. The Y-crossover operator is like the classical crossover operator, in that it exchanges useful information between structures. Likewise, the Y-mutate operator allows new points in the search space to be explored.

Phase 1 is repeated a number of times with different initial populations. Each time the best solution found is saved and the set of vertical arrangements thus obtained is used to generate the initial population for phase 2.

6.2 Phase 2: Avoiding arcs that cross

The initial population for phase 2 is constructed by using the good set of y coordinates from phase 1 and random x coordinates. This phase considers whether arcs cross and whether nodes overlap, since our representation allows this to occur. Consequently, the evaluation function used in phase 2 is:

$$Eval_2(M) = a_c \cdot n_c(M) + a_o \cdot n_o(M) + 1$$

In our experiments, we have used $a_c = 1, a_o = 2$.

In designing the set of genetic operators to be used at this stage we must take care not to undo the progress made in phase 1. Thus, these operators will not change y coordinates. The genetic operators used at this stage are, again, variations of the three classical operators:

X-swap: Swap the x coordinates of 2 random nodes within a structure.

X-crossover: Crossover the x coordinate of a random node from one structure with the corresponding node of another structure.

X-mutatation: Randomly change the x coordinate of a given structure's node.

Note again that these operators do not guarantee that two nodes will not occupy the same square on the page. We simply let the evaluation function and the power of the genetic algorithm search for the best solution.

Once phase 2 is complete, we evaluate each structure using a combination of the evaluation functions used previously and take the best solution found.

7 Experiments and Results

We have performed a number of experiments with the systems GRAPH-1 and GRAPH-2. In this section we present the results of applying these systems to the two graphs, G_1 and G_2, shown in Figures 2 and 4. In each case we used a page with 10 rows and 6 columns.

7.1 GRAPH-1

In the system GRAPH-1 we ran the genetic algorithm for 200 generations with each graph. The resulting layouts are shown in Figures 6 and 7.

Figure 6: The result of GRAPH-1 system for G_1.

These results are encouraging, given that the only factors taken into account in the evaluation function were the numbers of arcs that cross, point up or are horizontal. Both graphs display high regularity and constitute an excellent input for the algorithm to distribute nodes evenly on a page. Similar results were obtained in experiments with larger number of nodes and a larger page size.

7.2 GRAPH-2

In the system GRAPH-2, for each graph, we ran the genetic algorithm for 200 generations for each of 10 passes through phase 1, and 200 generations for phase 2. Using a population size of 50 meant that the initial population of phase 2 consisted of 5 structures for every y coordinate arrangement found by phase 1. The resulting layouts are shown in Figures 8 and 9.

Again, these results are encouraging, given the attributes taken into consideration by the evaluation functions.

Figure 7: The result of GRAPH-1 system for G_2.

7.3 Comparison

For G_1, the general shape of the layout produced by GRAPH-2 is probably better than that produced by GRAPH-1. The odd positioning of nodes 1, 8, 5 and 6 would be rectified by the node distribution algorithm. It does, however, unnecessarily have one arc pointing upwards and one intersection of two arcs.

For G_2, the general shape of the layout produced by GRAPH-1 looks marginally better than that produced by GRAPH-2, though this difference would probably be negligible once node distribution has been performed. Again, GRAPH-2 unnecessarily has one arc pointing upwards and one intersection of two arcs.

These two graphs are not sufficient to make any conclusive comparisons between the two systems. They do, however, suggest that GRAPH-1 is better at avoiding crossing arcs, while GRAPH-2 seems to be better at avoiding horizontal and upward arcs. Varying the number of generations, the probabilities associated with the various genetic operators and the weightings used for the various aesthetic features would clearly affect the results. Obviously many more tests need to be performed before more definite concusions can be drawn.

Figure 8: The result of GRAPH-2 system for G_1.

8 Conclusions

For several reasons, we are not yet able to fully evaluate the two presented approaches or compare them with other methods. Firstly, we only have results from few sample graphs; we need to test both systems on many other examples. Secondly, the algorithm to distribute nodes evenly on the page is still being implemented; we need to cmplete this before we can make meaningful comparisons with other methods. Thirdly, the two approaches presented in the paper are, in some sense, too different. They use different genetic representation, different operators, and additionally, the system GRAPH-2 is based on the idea of decomposed evaluation function.

To allow the various factors involved in these two systems to be compared and evaluated more precisely, we plan to implement a variation of GRAPH-1 which uses the same representation and genetic operators, but has a decomposed evaluation function, and a GRAPH-2 which uses the same representation and genetic operators, but has a single evaluation function. These should allow more meaningful comparisons to be made.

When we have these systems working well, we plan to incorporate the most successful system into an interactive programming tool. We could then allow the user to set various operational parameters, so the tradeoff between picture quality and speed is under the users' control. In particular, the user could ask for high quality drawings at the expense of extra time in order to generate publication quality diagrams from the system. We also plan to adding other aesthetic

Figure 9: The result of GRAPH-2 system for G_2.

criteria (as discused in [Tamassia, Di Battista & Batini, 1988]) and allow user to control which ones are used.

References

[**Axelrod, 1987**] Axelrod, *Genetic Algorithm for the Prisoner Dilemma Problem*, in [Davies, 1987].

[**Carey & Johnson, 1979**] Carey, M. R. and Johnson, D. S., *Computers and Intractibility – A Guide to the Theory of NP-Completeness*, Freeman, 1979.

[**Davis, 1987**] Davis, L. (editor), *Genetic Algorithms and Simulated Annealing*, Pitman, London, 1987.

[**Eades & Xuemin, 1989**] Eades, Peter and Lin Xuemin, *How to Draw a Directed Graph*, Technical Report, Department of Computer Science, Univeristy of Queensland, Australia, 1989.

[**Goldberg, 1985**] Goldberg, D.E., *Dynamic System Control Using Rule Learning and Genetics Algorithms*, in Proc. International Joint Conference on Artificial Intelligence, 9, pp.588–592.

[**Holland, 1975**] Holland, J., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[**Holland, 1986**] Holland, J., *Escaping Brittleness*, in *Machine Learning II*, ed. R. Michalski, J. Carbonell, T. Mitchel, Morgan Kaufmann Publ., Los Altos, CA.

[**Janikow & Michalewicz, 1990**] Janikow, C., Michalewicz, Z., *Hierarchical Genetic Algorithms*, submitted for publication.

[**Michalewicz** *et al*, **1989**] Michalewicz, Z., Jankowski, A., Vignaux, G. A., *The Constraints Problem in Genetic Algorithms*, submitted for publication.

[**Michalewicz, Vignaux & Groves, 1989**] Michalewicz, Zbigniew, Vignaux, G. A. and Groves, Lindsay J., *Genetic Algorithms for Optimization Problems* Proc. 11th New Zealand Computer Conference, August, 1989, pp211–223.

[**Tamassia, Di Battista & Batini, 1988**] Tamassia, R., Di Battista, G. and Batini, C., "Automatic Graph Drawing and Readability of Diagrams", *IEEE Trans. Systems, Man, and Cybernetics*, 18, 1 (Jan/Feb, 1988), pp61-79.

[**Vignaux & Michalewicz, 1989a**] Vignaux, G. A. and Michalewicz, Z., *Genetic Algorithms for the Transportation Problem*, Proc. 4th International Symposium on Methodologies for Intelligent Systems, Charlotte, NC, October 12–14, 1989.

[**Vignaux & Michalewicz, 1989b**] Vignaux, G. A. and Michalewicz, Z., *A Genetic Algorithm for the Nonlinear Transportation Problem*, in preparation.