A heuristic for graph drawing

Peter Eades
University of Queensland

Abstract

This note reports on a heuristic method for drawing graphs. The method is successful in producing good layouts for many graphs with less than 30 vertices.

1. Introduction

Many data presentation problems involve the layout of a graph whose vertices represent entities and whose edges represent relationships between the entities. Examples are database schema, PERT networks,organisation charts, and logic circuit diagrams. The TYGES system developed at the University of Queensland assists with the practical physical layout of such diagrams on limited two dimensional surfaces such as plotters and CRT screens. A description of TYGES is given in [EHIL83].

The crucial part of TYGES is the embedder, the program which assigns locations to vertices in such a way that the resulting layout is in some sense aesthetically pleasing. The design of an embedder is a formidable task, since "aesthetically pleasing" is a subjective concept.

In this note we report on a method for drawing graphs to meet two criteria: all the edge lengths ought to be about the same, and the layout should display as much symmetry as possible. These criteria form a part of "aesthetically pleasing" in a wide variety of application areas. Further, we aim to produce layouts which conform to the author's somewhat subjective sense of aesthetics.

It is NP-hard to draw a graph so that all edge lengths are the same (see [J82]), and displaying symmetry is at least as difficult as graph isomorphism. Hence a heuristic method is justified.

2. The algorithm

The basic idea is as follows. To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system, as in figure 0. The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state. The algorithm outputs the positions of the vertices in this stable state.

Two practical adjustments are made to this idea: firstly, logarithmic strength springs are used; that is, the force exerted on a ring by a spring is

$$C1*\log(d/C2),$$

where d is the length of the spring, and C1 and C2 are constants. Experience shows that Hookes Law (linear) springs are too strong when the vertices are far apart; the logarithmic force solves this problem. Note that the springs exert no force when d=C2. Secondly, we make nonadjacent vertices repel each other. An inverse square law force,

C3/sqr(d)

where C3 is constant and d is the distance between the vertices, is suitable.

The mechanical system is simulated by the following algorithm.

```
algorithm SPRING(G:graph);
place vertices of G in random locations;
repeat M times
    calculate the force on each vertex;
    move the vertex C4*(force on vertex)
draw graph on CRT or plotter.
```

The values C1=2.0, C2=1.0, C3=1.0, C4=0.1, are appropriate for most graphs. Almost all graphs achieve a minimal energy state after the simulation step is run 100 times, that is, M=100.

Calculating the force on each vertex takes time proportional to the square of the number of vertices. An implementation of the algorithm on a VAX11/780 (in unoptimised Pascal) is fast (in fact, I/O bound) if the number of vertices in G is less than 30.

3. Examples

Figures 1 to 6 show both the successes and limitations of the method. Some remarks on the figures are appropriate.

Figure 1. These examples show that symmetries, i.e., graph automorphisms, are often displayed. Note that 1(a) is the same graph as 1(b); the different layout is due to different starting positions.

Figure 2. The method is particularly successful with regular grid like structures.

Figure 3. A few extra edges added to the figure 2(a) makes figure 3(a). Although it is a little distorted, it is acceptable. Further edges are added in 3(b), and even more in 3(c). This distorts the layout badly, and there is a wide variance in edge length. The extra edges are all in one area of the grid, and this concentration of springs holds this area together tightly, and the vertices not adjacent to this group are pushed far away by the accumulated (inverse square law) repulsion force. The only saving feature of figure of 3(c) is that the 6-clique is displayed clearly.

Figure 4. The algorithm consistently produces good layouts for trees. The edges are sometimes rather longer near the center of the tree

than near the leaves, but symmetries are displayed well and the overall effect is not unpleasant. The algorithm is fast for trees; M=30 is sufficient to reach equilibrium for most trees with less than 20 vertices.

Figure 5. Similarly, the algorithm produces good layouts for most sparse graphs.

Figure 6. Some problems with the method are illustrated. In 6(a) there is a triangle with vertices that are almost co-linear. This problem can be averted by a different choice of constants C1, C2, C3. In 6(b), the random initial layout has the central 4-cycle crossed, and the algorithm does nothing that would untangle it. Further, 6(a) and (b) illustrate a common problem for graphs with a small number of bridges (edges whose deletion disconnects the graph): the bridges become rather too long. Figure 6(c) shows another unnecessary crossing; in this case, the crossing is forced by the attempt to make all edge lengths the same, and will occur no matter what the initial layout is. Some problems with the layout of dense graphs are illustrated in 6(d). There are many vertices which lie close to edges, and (even with a screen or plotter of good resolution) it is difficult to distinguish whether the vertex is actually incident with the edge or merely placed near it.

4. Conclusions

The algorithm produces good layouts for many graphs, in keeping edge lengths about the same and in displaying symmetries.

There are several classes of graphs for which the algorithm produces poor layout: dense graphs, or graphs with dense subgraphs, graphs with a small number of bridges. However, for a wide class of graphs, the method can be used as a first approximation to a layout; the approximation can be "fine tuned" using a graph editor such as that provided by TYGES (see [EHIL83]).

The algorithm has an acceptable running time for graphs with less than 50 vertices. In applications, graphs with more vertices are usually broken up into small subgraphs, because more vertices will not fit on the output device.

We conclude that the method shows promise as an embedder for practical graph layout problems.

5. Acknowledgement

We would like to thank Leo Bradley for his preliminary work in this area, and for many stimulating discussions. We also acknowledge the support of the University of Virginia and the Computational Geometry Laboratory at McGill University.

6. References

[EHIL83] Peter Eades, John Hynd, Adrian Lee and Don Isdale, "Interim Report on TYGES", Technical Report 47, Department of Computer Science, University of Queensland, 1983.

[J82] David S. Johnson, "The NP-Completeness Column: An Ongoing Guide", Journal of Algorithms 3, 89-99, (1982).

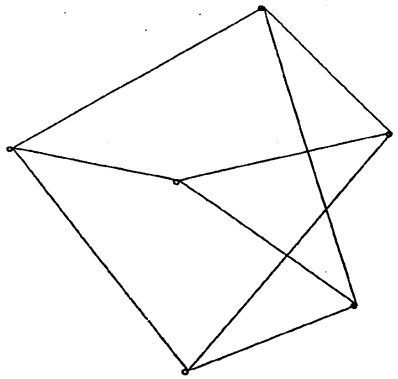

Figure 1(a)

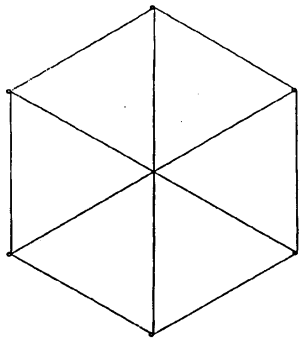

Figure 1(b)

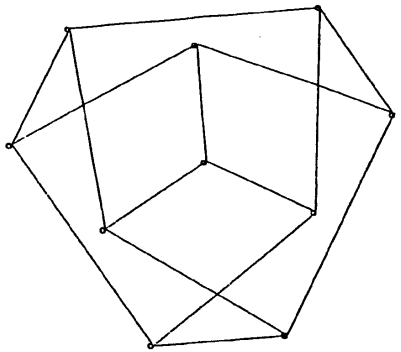

Figure 1(c)

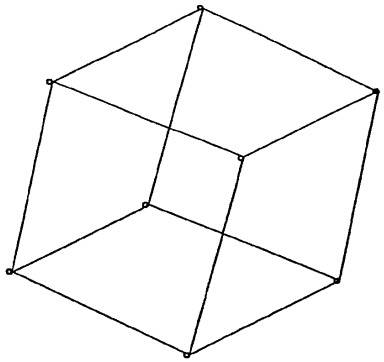

Figure 1(d)

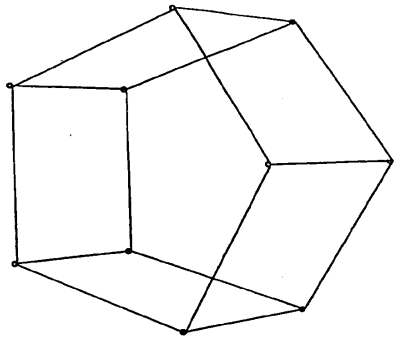

Figure 1(e)

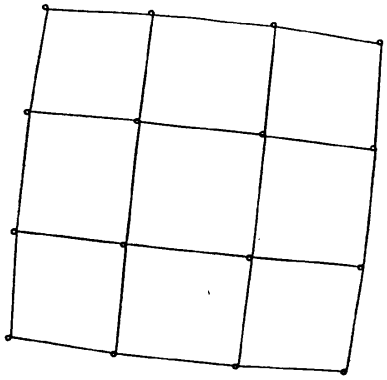

Figure 2(a)

Figure 2(b)

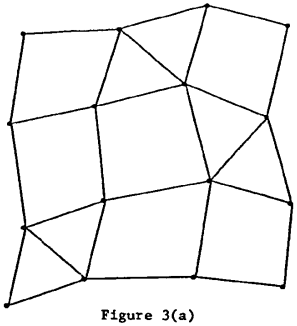

Figure 3(a)

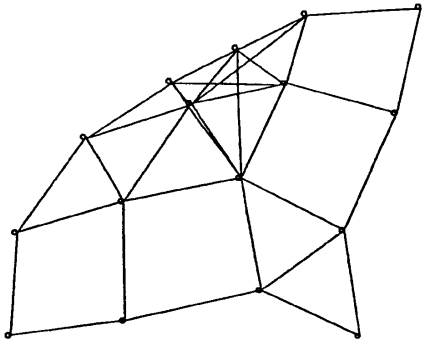

Figure 3(b)

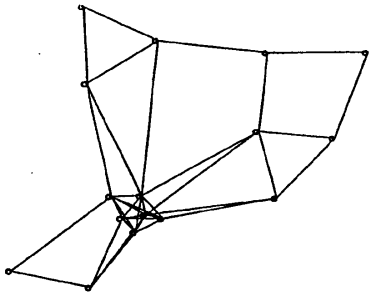

Figure 3(c)

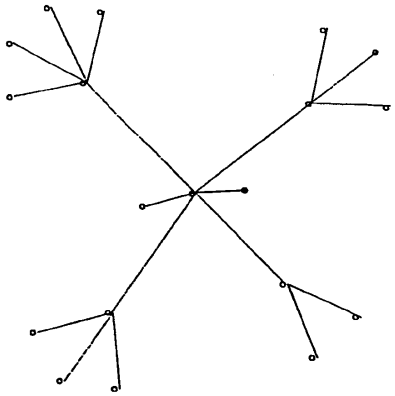

Figure 4(a)

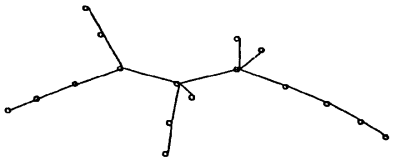

Figure 4(b)

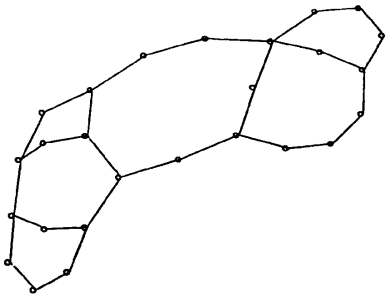

Figure 5(a)

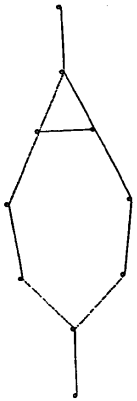

Figure 5(b)

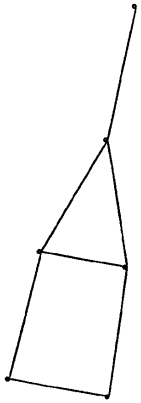

Figure 5(c)

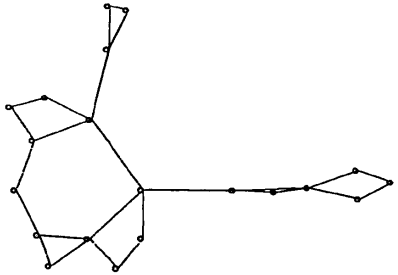

Figure 6(a)

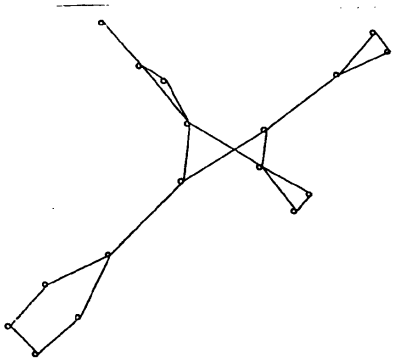

Figure 6(b)

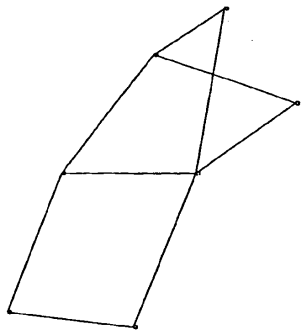

Figure 6(c)

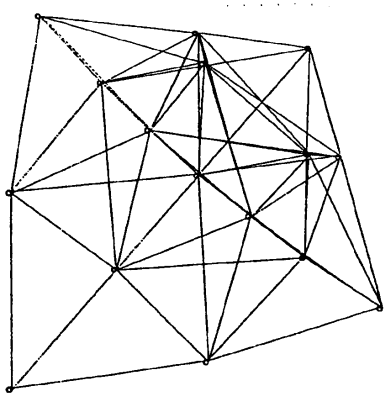

Figure 6(d)