Using A Heuristic Algorithm to Help Analyze Graph Problems

LieJune Shiau

Department of Mathematics and Computer Science
University of Houston - Clear Lake
Houston, TX 77058

Introduction

The study of graphs in most books implies that graphs can be investigated and analyzed by simple analytic techniques. When students begin to explore even moderately complicated problems that arise in real-world applications, such as the traveling salesman, transportation system, and network problems, they soon realize this is not true. Very often, useful information about a graph is concealed and can be found visually only after the vertices are rearranged. My main point in this paper is that computer graphics, combined with a simple heuristic algorithm, can provide a powerful new tool for exploring graphs. This new tool is easy to use, and also gives insight not readily attainable by traditional analytic techniques. Furthermore, this tool can provide geometric views which motivate analytic arguments.

A Heuristic Algorithm

A graph can be represented as an adjacency matrix. An adjacency matrix gives the relations of all the vertices and edges of a graph. However, with various ways of labeling vertices, there are many different appearances of isomorphic graphs corresponding to an adjacency matrix. The important thing is how to select the one with the fewest unwanted intersections and most logical simplification.

Here, we will introduce a heuristic algorithm for rearranging the vertices in a graph. The computation is performed by an iterative search which starts with a poor layout of a graph and progressively improves it by reducing its cost function. This cost function measures the extent to which the current layout of the graph varies from the desired result. Each possible layout has an associated cost, computed by its cost function. The employed heuristic for rearranging the vertices is chosen so that this cost keeps being reduced. Once the cost reaches its minimal, the layout of the graph is the desired one.

The cost function is defined as $E=\sum|d_{ij}-r|$. Here d_{ij} is the length of the edge between vertices i and j, and r is a constant. It is obvious that E is minimal if, and only if all edges are of the equal length r.

The algorithm begins with randomly assigning vertices inside a square of width r, then gradually readjusting the position of each vertex to reduce the cost determined by the function E. This algorithm treats adjacent and non-adjacent vertices separately. For adjacent vertices, if their distance is less than r, the algorithm pushes them away to increase their distance. If their distance is greater than r, this algorithm pulls them toward each other to reduce their distance. After a few iterations, the vertices will come to the vicinity of their final positions. In the process, if a vertex has been pushed or pulled too drastically, it may oscillate around its final position and never settle. To avoid this, the algorithm slowly reduces the displacement along each iteration. To be more specific, for the next iteration, if adjacent vertices i and j are too close, i.e. $d_{ij}<r$, let them both be moved away by a displacement of $\alpha(d_{ij}-r)^2$. If they are too far away, i.e. $d_{ij}>r$, let them both be moved closer by a displacement of $\alpha(d_{ij}-r)^2$. For non-adjacent vertices, the algorithm separates them as far as possible. In each iteration of this algorithm, let the vertices push each others away by a displacement of βd_{ij}.

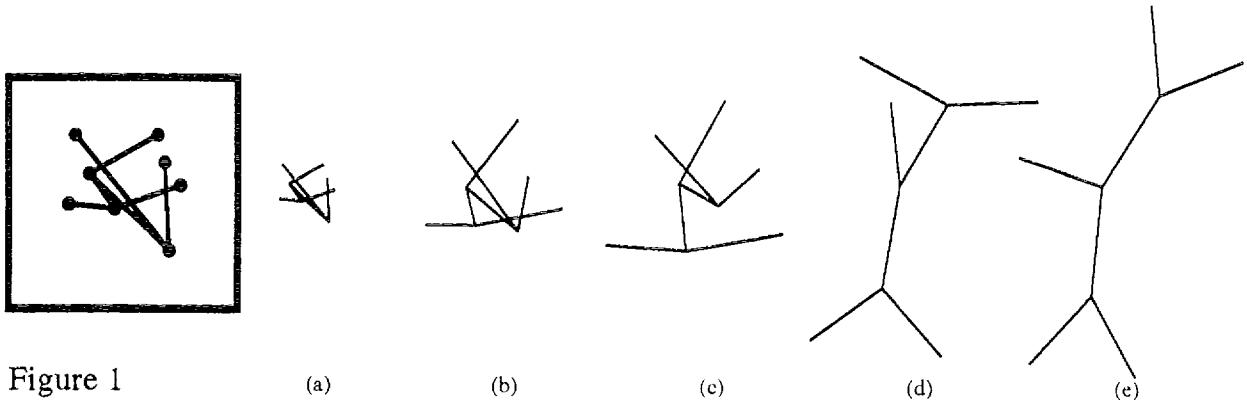


Figure 1 (a) (b) (c) (d) (e)

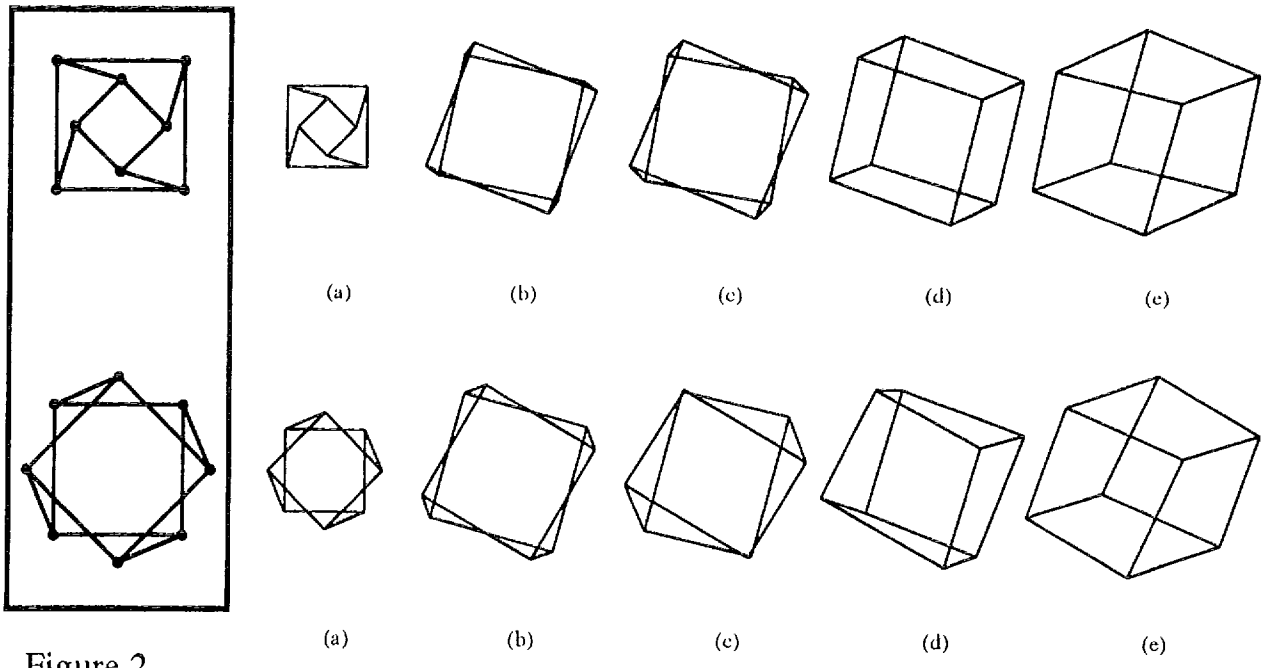
Experience has shown that the values of α=0.25 and β=0.1 favorite most situations, and give very satisfactory results.

It is conceivable that a vertex will be pulled and pushed simultaneously by several forces from all vertices. The net result should be the combination of all the applicable forces. In order to reflect this fact, this algorithm requires that all the coordinate adjustments be computed first, then the accumulated adjustments be applied to the vertices.

Applications

Although the idea of this heuristic algorithm is very simple, a number of applications may utilize it. Since every model has relationships between items that lend themselves to a graphical representation, this algorithm would apply. For instance, this algorithm will provide a clearer and more desirable layout by getting rid of unwanted intersections in a graph. In Figure 1, we used a series of animation to depict the sequence of unfolding a graph. Starting with a few undesired intersections, this graph gets transformed into a graph which is much easier to handle. Once the transformation is completed, we can easily answer questions such as whether this graph is connected or which is the shortest path from one particular node to another.

Another application is to determine graph isomorphism. We can investigate the layouts of two transportation systems using this algorithm, which helps in detecting if they are actually isomorphic transportation systems. (See Figure 2.) It



(a) (b) (c) (d) (e)

(a) (b) (c) (d) (e)

Figure 2

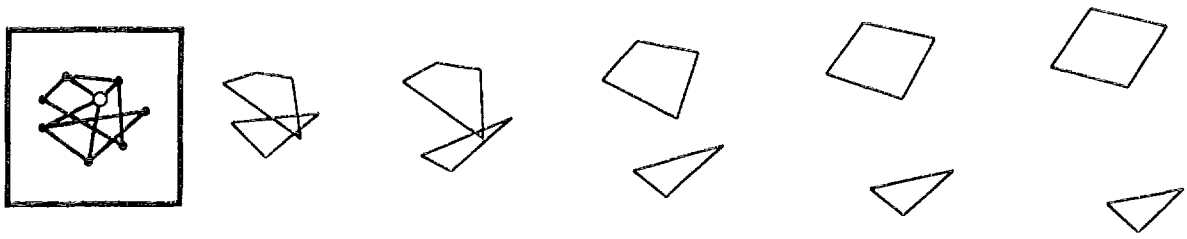


Figure 3 (a) (b) (c) (d) (e)

may not be easy to find a one-on-one function between the two graphs on the left, especially when the existence of such A function is still unknown. These two graphs seem unlikely to be related. However, after being unfolded by this algorithm, it is clear that they are isomorphic.

In a graph that represents a computer or communication network, the vertices denote the communication entities (computers, telephones, etc.) and the edges denote the communication medium (coaxial cables, telephone lines, etc.). Such a graph should be a connected graph, so that there is a path between every pair of nodes. However, even in a graph sufficiently rich in edges, removing a single node may cause the graph to become disconnected. Such a node is referred to as an articulation point. We can apply this algorithm to such a network for one node being removed (as the circled node in Figure 3). The result of connectivity shows if that particular node is an articulation point. In Figure 3, it is not initially clear which node is an articulation point; but application of this algorithm results in a disconnected transformation, and identifies an articulation point.

Conclusion

This algorithm is easy to implement on a computer. The details depend on the programming language chosen. I used Turbo Pascal because of its capability in supporting graphical presentations. The results can be presented as computer animation which is not only interesting but also concise.

If this heuristic algorithm is assigned as a programming project, students should be encouraged to find other plausible applications of this program. They often find more fundamental issues about graph this way, which they had never before appreciated.

**********DISCRETE MATHEMATICS II Continued From Page 17**********

2. Brookshear, J. Glenn, Theory of Computation: Formal Languages, Automata, and Complexity, The Benjamin/Cummings Publishing Company, Inc., New York, 1989

3. Gries, David, "Teaching Calculation and Discrimination: A More Effective Curriculum", Communications of the ACM, volume 43, number 3, March 1991, pages 44-55

4. Gries, David, The Science of Programming, Springer-Verlag, New York, 1981

5. Rosen, Kenneth H., Discrete Mathematics and its Applications, The Random House, New York, 1988

6. Wainwrite, Roger, "Introducing Functional Programming in Discrete Mathematics", SIGSCE Bulletin, volume 24, number 1, March 1992, pages 147-152