

CHAPTER 1

SOFTWARE SYSTEMS FOR IMPLEMENTING GRAPH ALGORITHMS FOR LEARNING AND RESEARCH

JAY BAGGA

*Department of Computer Science, Ball State University
Muncie, Indiana 47306, USA*

ADRIAN HEINZ

*Department of Computer Science, Ball State University
Muncie, Indiana 47306, USA*

Graph algorithms have several important applications in fields such as computer science, software engineering, mathematics, engineering, business, and bioinformatics. Researchers and practitioners in these disciplines often need to experiment with empirical data about graphs to gain deeper insights into their properties, which may lead to general proofs. Students also need to learn and be able to implement graph algorithms for their applications. However, these individuals often have varying backgrounds and training, and they may not have a working knowledge of programming tools to implement graph algorithms. The goal of our research is to create a software system which allows users to easily create and implement graph algorithms through a simple graphical user interface, without any coding. Towards this goal, we have developed several systems that can be used to draw and manipulate graphs as well as to execute graph algorithms. The development of the general system raises a number of interesting research questions that we will discuss. To illustrate the need for experimentation with different graph algorithms, we describe some examples of our research in different areas of graph theory and computational geometry. We also describe some features of our systems that we have found useful in teaching graph theory.

1. Introduction

In the last few decades the fields of graph theory and graph algorithms have experienced a very rapid growth. Much of this can be attributed to applications of graph theory and graph algorithms to a large number of areas such as computer science, computer engineering, mathematics, business, sciences, bioinformatics, global information systems and several others. Applications of graphs are especially pervasive in almost all areas of computer science and information technology. This has fueled an explosive growth in research in this

area. A large number of books, research journals, web sites and other online resources are now available.

Graph theory, graph algorithms and their applications are routinely included in courses in undergraduate and graduate programs in computer science, mathematics, engineering, business and several others. Students in these courses, researchers in pure and applied graph theory, and professionals who need to experiment with graph algorithms often have varying backgrounds. In particular, such persons need to work with graphs and implement graph algorithms, but they may not have the necessary programming skills to do so.

The goal of our project is to create software systems for easy manipulation and experimentation with graphs and graph algorithms. Such software systems can be used for learning and teaching, for implementing graph algorithms, for applications, and for conducting research where experimentation with graphs and empirical evidence are needed. Our objective is to develop general systems which allow students and practitioners to experiment with graphs and construct and implement graph algorithms without programming. Such development raises a number of important research questions. We discuss these below.

In this paper we describe our recent work and progress of our project. In Section 2, we describe some of these research topics and describe how our software systems have been used in this research. The research topics discussed in Section 2 are samples that are intended to illustrate the use of software systems that we are developing. In Section 3, we describe a number of systems that we have developed and used in teaching and research. In Section 4 we present a summary and discuss plans for our future work.

2. Some Research Topics in Graph Theory

In this section we describe how our software systems are helping us in teaching and research in graph theory graph algorithms and their applications. We developed these systems to help us investigate questions that arise in our research, and to give our students tools which help them gain a better understanding and experience with the complexity of the concepts and algorithms. While some of our tools are specialized to areas of our research interest, others are more general and extensible. Our goal is to develop a general tool which is extensible in the sense that new graph operations and graph algorithms can be easily added without any significant programming. Users from a variety of different background should be able to fully utilize the system with little preparation.

In the following subsections, we introduce three areas of active research. We give only a brief introduction and some definitions for each area and provide references where more details can be found. Our purpose is to describe how the use of our software tools is an integral and indispensable part of our research and teaching.

2.1. Computational Geometry – Visibility Graphs

The study of visibility graphs is an active area of research in computational geometry. Visibility graphs have applications in art gallery guard placement problems, robot path planning in the presence of obstacles, and computer graphics, among others [1]. We have been interested in the study of polygon and segment endpoint visibility graphs.

Definition 1 Let P be a polygon on n vertices v_1, v_2, \dots, v_n . The *polygon visibility graph* $G(P)$ of P has vertices v_1, v_2, \dots, v_n , and two of these are adjacent if the line segment joining them is either an edge of P or an internal diagonal of P .

Definition 2 Let S be a set of n disjoint line segments $x_1y_1, x_2y_2, \dots, x_ny_n$ in the plane, in general position. The *segment endpoint visibility graph* $G(S)$ has $2n$ vertices $x_1, y_1, x_2, y_2, \dots, x_n, y_n$ and two of these are adjacent if the line segment connecting them is either in S or does not (internally) intersect any of the segments in S .

Visibility graphs have been extensively studied [2, 3, 4, 5, 6, 7, 8, 9, 10]. The breadth and depth of the results demonstrates that this is a rich field of study. According to O'Rourke [2], there are four goals of research in visibility graphs:

- **Characterization.** Characterize the class of graphs realizable by certain classes of geometric objects.
- **Recognition.** Find algorithms to recognize a graph as a visibility graph.
- **Reconstruction.** Find algorithms to output a geometric realization of visibility graphs.
- **Enumeration.** Find the number of visibility graphs under various restrictions.

Our research is in the area of the characterization of visibility graphs. In general, most of the problems in these four areas of research are hard. Visibility graphs of even small sizes are difficult to draw by hand. One of our software systems has helped us visualize such graphs and determine some properties. This system, Colossus is described in Section 3.2.

2.2. Graph Drawing

Graph drawing is an active area of research. Here we are concerned with efficient and aesthetic presentations of graphs (usually in the plane). See [11] for an excellent introduction to graph drawing. An annual conference on graph drawing has been held since 1992. See www.graphdrawing.org.

A graph is called *planar* if it can be drawn in the plane without any edges crossing. There are several well-known efficient algorithms to check a graph for planarity. We have implemented one such algorithm in our system JGraph. We describe JGraph in detail in Section 3.1. A well known theorem of Fáry [12] states that a planar graph can be drawn in the plane such that all edges are straight lines. In applications such as integrated circuit design, more specialized drawings of planar graphs might be required. An algorithm of de Fraysseix et al [13] draws a planar graph with straight line edges such that the vertices are points on a grid. We have implemented this algorithm in JGraph. See Section 3.1 for details.

2.3. Graceful Labeling of a Graph

Given a graph G with q edges a labeling of the nodes with distinct integers from the set $\{0, 1, 2, \dots, q\}$ induces an edge labeling where the label of an edge is the absolute difference of the labels of the two nodes incident to that edge. Such a labeling is *graceful* if the edge labels are distinct. This concept was introduced in 1967 by Rosa [14]. See Gallian [15] for an excellent survey of graceful labelings. Graceful labelings have applications in coding theory, x-ray crystallography, radar, astronomy, circuit design and communication networks, addressing and data base management [15].

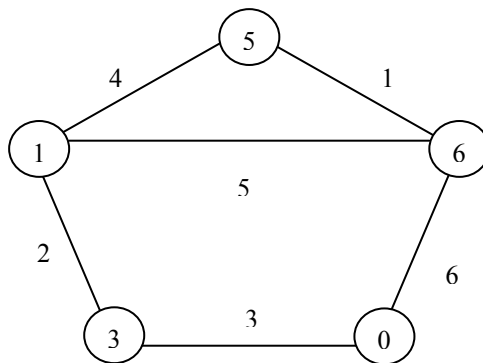


Figure 1. A graceful labeling of a graph.

We are interested in particular in graceful labelings of trees. Figure 1 shows a graceful labeling of a small graph.

The Ringel-Kotzig-Rosa conjecture [14] states that every tree has a graceful labeling. This almost forty year old conjecture is still open. Certain special classes of graphs such as paths, caterpillars, symmetrical trees and some others have shown be graceful. A computer search by Aldred and McKay [16] was used to show that all trees with up to 27 vertices are graceful. We have investigated different types of graceful labelings of paths. It was proved in [17] that the number of graceful labelings of the path of length n grows asymptotically at least as fast as $(5/3)^n$. We have used our system *Manohar* to investigate graceful labelings of caterpillars and lobsters, and also to investigate different patterns of graceful labelings of paths. See Section 3.3 for details.

2.4. Teaching of Graph Theory and Graph Algorithms

The first author has been involved in research and teaching of graph theory for more that twenty-five years. Over the last several years, we have used our software system JGraph in our graph algorithms classes. The majority of students in this class are computer science students, but some others such as those from mathematics and physics also enroll. Thus they come with a variety of backgrounds. JGraph can be used as a graph editor where users can draw, manipulate and experiment with different graphs. The extensive graphical user interface allows for graph manipulations with mouse clicks. Students can also execute and visualize (with animation) many well known graph algorithms. Students have found the animation feature especially useful since it helps them understand the workings of complex algorithms. New algorithms can be easily added to the system and algorithms in the system can be removed from the menu of available algorithms. Students have also used the system for classroom and term assignments and projects. JGraph is described in more detail in Section 3.1. The reader is invited to visit www.cs.bsu.edu/homepages/gnet for a demonstration version (JEdit 4.2) of this system. See [18] for a detailed discussion of an earlier version of JGraph.

3. Software Systems

In this section we describe in some detail our software systems. While we have found these systems useful in our research and teaching, it must be mentioned that these systems are themselves research projects in varying stages of development.

3.1. JGraph

JGraph is a system for creating graphs and running graph algorithms. The system has been used for research and teaching of graph theory in undergraduate and graduate level courses.

The system provides an easy-to-use graphical user interface in which the user can create graphs by adding vertices and connecting them by edges. It is possible to move individual vertices and edges around the window or to move multiple elements by selecting a component of the graph. The graph can also be manipulated by changing the colors of its elements. Rotation features are also available by selecting the rotation pivot point and the rotation angle.

Another feature of JGraph is that it contains drawings of certain special graphs. These are graphs that have been the focus of attention of the research community. These special graphs include: complete graphs K_n , complete bipartite graph $K_{m,n}$, hypercube graphs, platonic graphs, cycle, wheel, the Petersen graph and the Headwood graph. Graphs can be saved in a special format .GPH files for later retrieval.

One of the most important features of JGraph is its ability to execute graph algorithms. These algorithms can be applied to the currently displayed graph. Algorithms can be executed in an optional animation mode. This allows the user to clearly see each step of the execution of an algorithm. Algorithms are classified by the input required from the user. *Automatic algorithms* are those that can be directly applied to the current graph. For instance, the *blocks finding algorithm* finds all the blocks of a graph and uses different colors to display each block as well as cut-vertices. The second group of algorithms is referred to as *one-click* algorithms. In this group, the user is required to click on a vertex of the graph which becomes the input of the algorithm. *Prim's minimum spanning tree algorithm* falls into this category. Once the user has drawn a graph and selected a vertex, the algorithm will find the minimum spanning tree for the given graph and color its vertices and edges. The next category refers to algorithms which require a graph and two vertices as input. This category is called *two-click*. The *Floyd-Warshall shortest path algorithm* executes on the current graph and two vertices selected by the user. It then proceeds to calculate the shortest path between the two selected vertices and colors the vertices and edges on that path.

Some of the most complex algorithms implemented are the automatic algorithms *planarity testing* and *planarity drawing*. Planar graphs were defined in Section 2.2. The planarity testing algorithm receives a graph as input and determines whether or not it is planar. It also displays a message to the user with

the result. The planarity drawing algorithm also receives a graph as input. If the input graph is non-planar, the algorithm displays a message informing the user that the graph cannot be drawn on a plane without edges crossing. On the other hand, if the graph is planar, the algorithm proceeds to arrange the vertices on a grid creating a planar drawing of the graph with non-crossing straight line edges.

Figure 2 displays a drawing of a graph with several edges crossing, and the result of running the planarity testing algorithm.

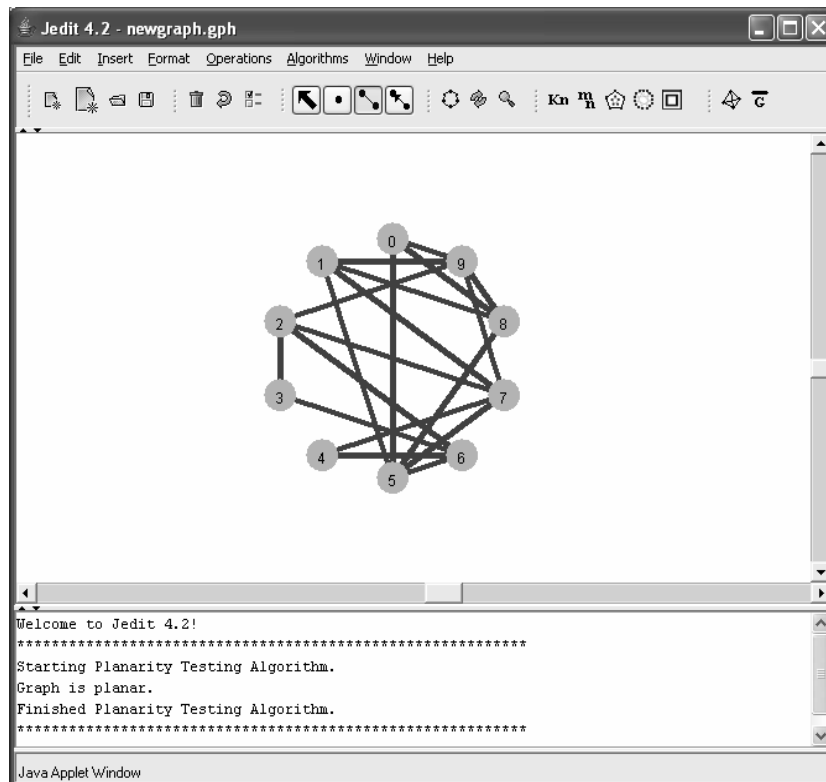


Figure 2. A graph drawn with several edge crossings and the result of running the planarity testing algorithm.

Figure 3 displays a planar drawing (of the graph of Figure 2) generated by the planarity drawing algorithm.

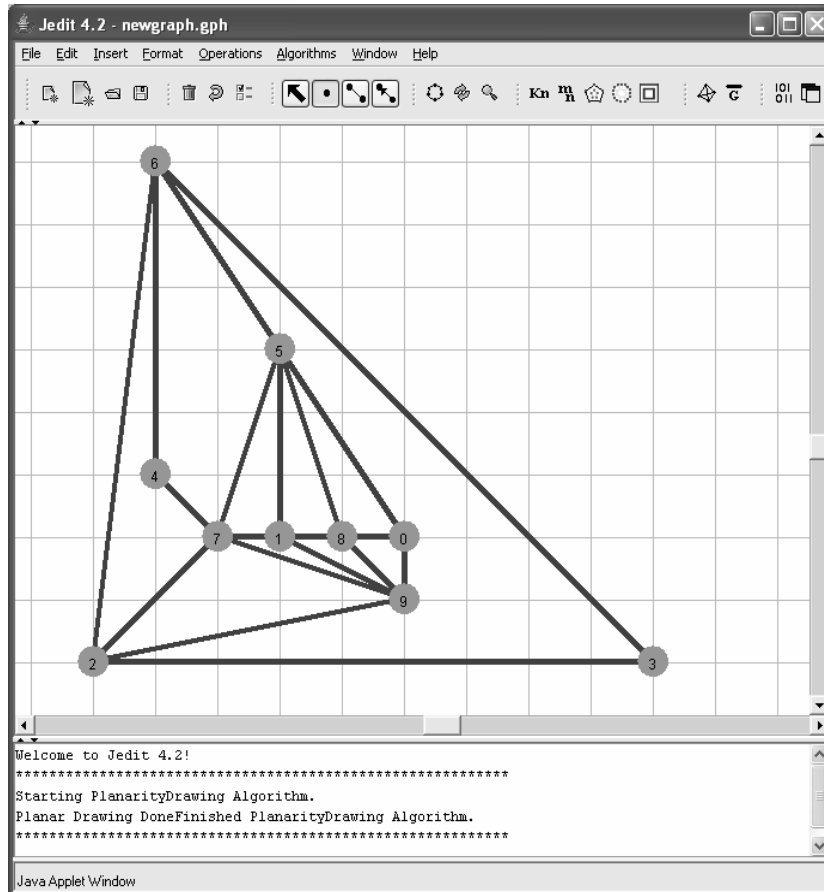


Figure 3. Output generated by the planarity drawing algorithm using the graph of Figure 2 as input. A planar embedding on a grid is displayed.

JGraph allows the user to easily add new algorithms. When selecting *new algorithm* from the menu, the user is prompted for the algorithm name, a description and the category of algorithm (automatic, one-click or two-click). After the input is completed, the system generates a template file in which the user can edit and add the new algorithm source code. Once the source code is completed and the file is saved, the next time JGraph runs, the new file will be compiled with the rest of the system and the new algorithm automatically added to the JGraph menu.

JGraph has been developed entirely under Java and therefore it can be run on any platform.

3.2. *Colossus*

A visibility graph is a graph of intervisible locations. Each node or vertex in the graph represents a point location, and each edge represents a visible connection between them (that is, if two locations can see each other, an edge is drawn between them). Colossus is a system for determining visibility graphs. The system includes a graphical user interface in which the user can draw a simple polygon. Colossus displays its visibility graph of the polygon and colors the ears and mouths. See [19] for definitions. The system performs the computation in real-time. Any changes to the configuration of the original graph are immediately reflected in its visibility graph. A screenshot of Colossus is shown in Figure 4.

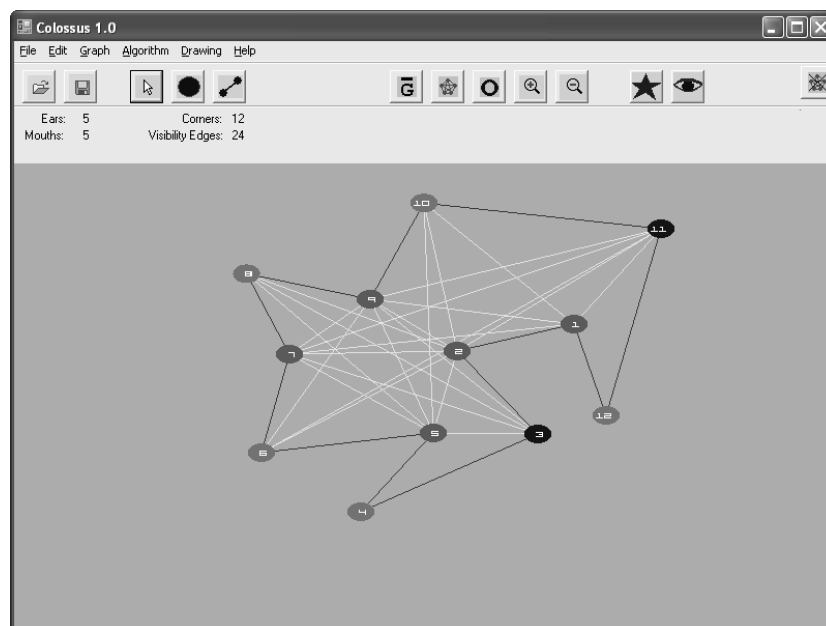


Figure 4. Screenshot of a visibility graph generated by Colossus. Ears and mouths correspond to vertices 4, 6, 8, 10, 12 and 1, 2, 5, 7, 9 respectively.

Another feature of Colossus is the ability to work with orthogonal graphs. When working in orthogonal mode, Colossus only allows horizontal or vertical edges among orthogonally aligned vertices.

3.3. *Manohar*

Manohar is a system for computing graceful labeling of graphs. See Section 2.3 for definitions. The system consists of a graphical user interface in which the user can draw a graph. When the input graph is completed, the system computes a graceful labeling of the graph and if it exists, it displays the graceful labeling on the screen. A screenshot of Manohar is illustrated in Figure 5.

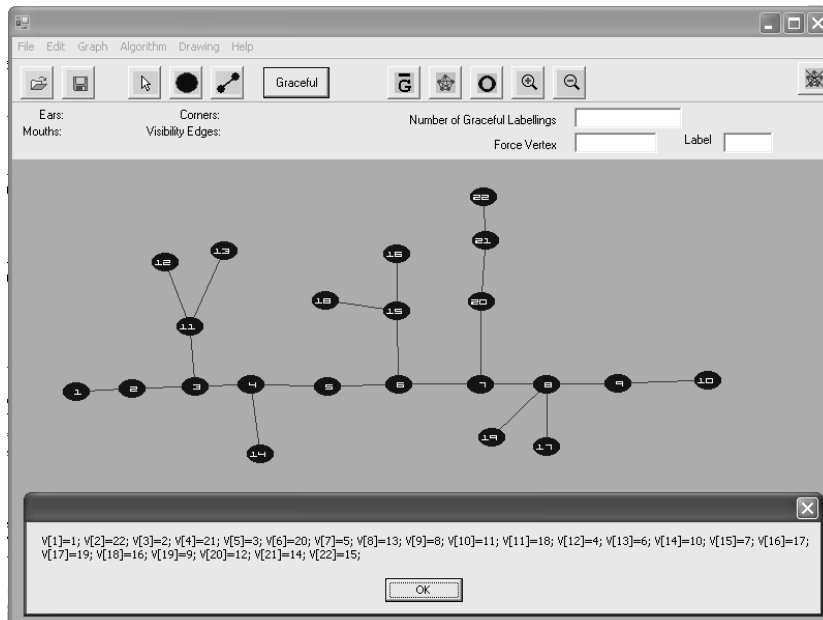


Figure 5. Screenshot of the output generated by Manohar for a tree with 22 vertices.

Even though Manohar can be used with many simple undirected graphs, it has been primarily used as a research tool for studying graceful labeling of trees. In particular, we have used Manohar as a research tool to investigate graceful labeling of certain special trees such as caterpillars and lobsters. See [15] for definitions.

3.4. Graph Algorithm Constructor

Graph Algorithm Constructor is our system for creating and running graph algorithms. The main feature of this system is that algorithms are created by drawing *flow diagrams* instead of writing source code and therefore no knowledge of programming is necessary to use the application. The system also incorporates a graph editor for drawing and manipulating graphs.

A flow diagram is a directed graph in which every vertex is an *entity*. Users can create a flow diagram by adding entities and connecting them. Entities consist of *elements*, *operations*, *control structures* and *algorithms*. An element is an atomic structure that can be randomly generated or input by the user. Examples of elements include graphs, trees and vertices. An operation is an action on an element or a group of elements. Operations require input and output. Examples of operations include *mark vertex*, *color vertex* and *find neighbor*. A control structure determines the execution flow. Examples of control structures include *loops*, *counters* and *conditionals*. An algorithm is defined by the flow diagram. Algorithms can be saved and later embedded in flow diagrams of other algorithms. A screenshot of Graph Algorithm Constructor is illustrated in Figure 6.

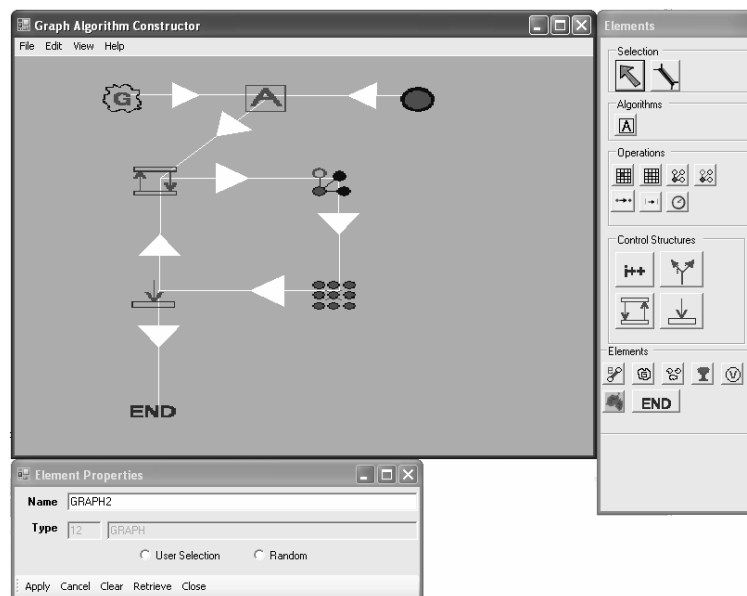


Figure 6. Screenshot of Graph Algorithm Constructor.

The flow diagram of an algorithm can be saved in a file for later use. The file format used is XML.

An algorithm created by the Graph Algorithm Constructor can be loaded and executed by the *Graph Algorithm Runner*. This is a part of the system which executes algorithms. It interprets the XML file created by the constructor and executes step by step showing the progress on the screen. The Graph Algorithm Runner has a graph editor where input graphs for an algorithm can be drawn or loaded from a directory of existing graphs. It also handles the display of the output graph and or other textual output as appropriate. A screenshot of the graph algorithm runner is shown in Figure 7. It shows an output graph on the left side and the XML tree in the panel on the right. The XML tree is obtained by parsing the XML file of the flow diagram. It allows the user to quickly see the structure of the file before running the algorithm.

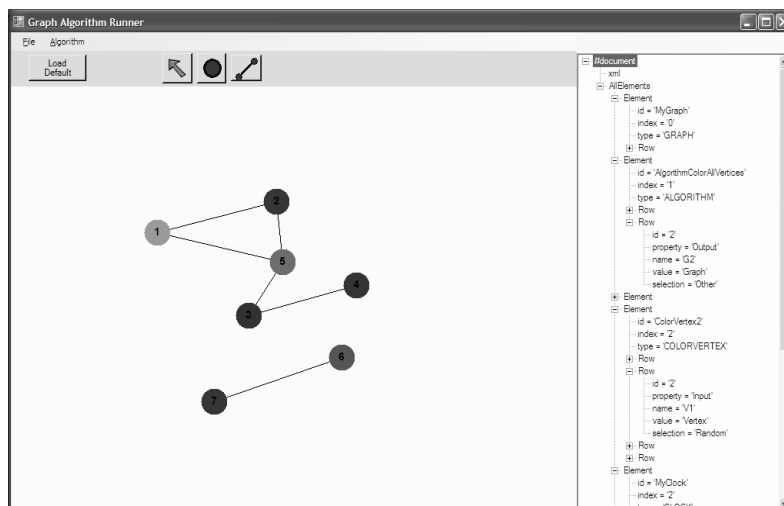


Figure 7. Screenshot of Graph Algorithm Runner.

The development of the Graph Algorithm Constructor system is an ongoing research project. We have investigated a number of interesting research questions that arose during this development process. Some of these questions are:

- **Classification of entities.** We are interested in using a minimal but complete set of entities as defined above. The classification of entities into

four distinct types – elements, operations, control structures and algorithms – serves this purpose.

- **List of elements.** The elements are the building blocks of graphs such as vertices, and edges. These can also be special graphs such as trees and cycles.
- **List of operations.** We seek a sufficient set of independent graph operations that are needed for the most common tasks in graph algorithms.
- **List of control structures.** This list includes most of the control structures of a high-level programming language. The goal is to find a set of primitive operations to create a flow diagram.

4. Summary and Future Work

In this paper we have described several software systems that we have developed for use in research and teaching of graph theory, graph algorithms, and their applications. We described some areas of our research where our systems have been particularly helpful. Students in our graph algorithms classes have used these systems to experiment with graphs and execute graph algorithms. A general graph algorithm construction system is currently under development. This system allows the user to create and implement graph algorithms without much coding. We have investigated a number of research questions that arise during the process of such a system.

Our immediate goal is to complete the development of the algorithm constructor system. We will also continue to upgrade the software systems described in this paper. New features will be added as needed in research and teaching of graph algorithms and their applications.

References

1. O'Rourke. Art Gallery Theorems and Algorithms. Oxford University Press, 1987.
2. J. O'Rourke. Visibility. In Handbook of Discrete and Computational Geometry, CRC Press, 1997.
3. X. Shen and H. Edelsbrunner. A tight lower bound on the size of visibility graphs. *Inform. Process. Lett.* **26**, pages 61-64, 1987/88.
4. J. Bagga, J. Emert, M. McGrew, and W. Toll. On the Sizes of Some Visibility Graphs. In *Congressus Numerantium*, **104**, pages 25-32, 1994.
5. J. Bagga, L.Gewali, and S.Ntafos. Visibility Edges, Mixed Edges, and Unique Triangulation. In *Proceedings of the 13th European Conf. on Comput. Geom.* page 11, 1997.

6. J. Bagga, S. Dey, L. Gewali, J. Emert, and M. McGrew. Contracted Visibility Graphs of Line Segments. In Proc. 9th Canadian Conf. Comput.Geom., pages 76-81, 1997.
7. J. Bagga, J. Emert, and M. McGrew. Directed Polygon Visibility Graphs. In *Congressus Numerantium*, **132**, pages 61-67, 1998.
8. J. Bagga, J. Emert, and M. McGrew. Directed Polygons as Boundaries of Visibility Graphs. In *Congressus Numerantium*, **142**, pages 57-63, 2000.
9. J. Bagga, J. Emert, and M. McGrew. Directed Polygons as Boundaries of Visibility Graphs. In *Congressus Numerantium*, **142**, pages 57-63, 2000.
10. J. Bagga, J. Emert, and M. McGrew. Connectivity Properties of Visibility Graphs. In *Congressus Numerantium*, **165**, pages 189-194, 2003.
11. Graph Drawing, Algorithms for the Visualization of Graphs. Giuseppe Di Battista, Peter Eades, Roberto Tamassia, Ioannis G. Tollis. Prentice Hall, 1999. ISBN 0-13-301615-3
12. I. Fáry, On straight line representations of planar graphs, *Acta Sci. Math.* **11**(1948) 229-233.
13. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid". *Combinatorica*, **10**(41-51), 1990.
14. A. Rosa, On Certain Valuations of the Vertices of a Graph", *Theory of Graphs (Proc. Internat. Symposium, Rome, 1966)*, Gordon and Breach, N. Y. and Dunod Paris, 349-355, 1967.
15. Joseph A. Gallian, A Dynamic Survey of Graph Labeling, *The Electronic Journal of Combinatorics*, **5** (2005), #DS6.
16. R. E. Aldred and B. D. McKay, Graceful and harmonious labellings of trees, *Bull. Inst. Combin. Appl.* **23**, 69-72, 1998.
17. R. E. Aldred, J. Siran and M. Siran, A note on the graceful labelings of path. *Discrete Math*, **261** 27-30, 2003.
18. J. Bagga and A. Heinz. JGraph - A Java based system for drawing graphs and running graph algorithms, *Lecture Notes in Computer Science*, **2265** (2002), Springer Verlag.
19. G. H. Meisters. Polygons have ears. *Amer. Math. Monthly* **82** pages 648-651, 1975.