AUTOMATED DISPLAY TECHNIQUES FOR LINEAR GRAPHS

R. Brien Maguire
Computer Science Department
University of Regina
Regina, Saskatchewan, CANADA

The development of display procedures for drawing pictures of linear graphs is described. The facility to model relationships pictorially has led to the use of graph theoretic techniques in many different applications. While computers normally work with a numeric representation of a graph such as its incidence matrix, manual transformation of such representations into pictures is a tedious process. An interactive graphics system has been developed which, through a combination of heuristic techniques and semi-automatic procedures, creates visual representations of graphs with a minimum of user intervention. The resultant pictures display mirror-image and rotational symmetries that occur within the graph. This very general approach of displaying symmetry in graphs has proven useful in studies of several classes of graphs. However, the system is primarily a research tool designed for use by mathematicians and graph theorists. Difficulties entailed in adapting the display procedures to more specific application areas are discussed.

Key Words and Phrases: computer graphics, graph theory, linear graph, automorphism group, symmetry and visual representation

CR Categories: 4.31, 4.9, 5.32, 8.2

1. Introduction

A wide variety of application areas employ graph theoretic techniques to model relationships which are best represented as pictures. Electrical networks, chemical structures, computational models and communication networks are examples of application areas where graph theoretic approaches have been used. One reason for this is the number of algorithms and techniques available for manipulating the graph models. Equally important, however, is the fact that the structure of the graphs can be represented pictorially.

Efficient computer manipulation of graphs requires that they have a numerical data representation. However, incidence or adjacency matrices and other numeric representations of graphs usually have little meaning for humans. Therefore, the engineer or mathematician attempts to translate the numeric results of his programs into pictures, seeking to obtain a deeper insight and understanding of the properties and structure of the results.

Conversion of numeric data into pictures can be a tedious trial and error process of drawing and redrawing pictures in order to achieve a suitable effect. A rough picture is drawn and then redrawn repeatedly, moving vertices and edges until the result appears satisfactory or more often until one runs out of time, paper or patience. Graph processing systems which allow the user to draw and manipulate graphs on a graphic display facilitate this process [9]. However, the user must still draw the pictures himself. All such systems do to aid in drawing the pictures is to provide a very expensive eraser. What is needed is a way to use the power of the computer to produce these pictures from a non-pictorial representation of the graph with minimal user interaction.

We have attempted to program a computer to do just that. Some of the problems involved in computer generation of pictures of graphs are discussed. The implementation of a semi-automated system for manipulating graphs which, through a combination of heuristic techniques and semi-automatic procedures, generates visual representations of graphs is described. This system allows a user to request the computer to draw pictures of a graph. The computer can produce these pictures without aid from the user, although in practice the user will direct the operation of the system as well as perform postediting of the pictures.

2. Graphics Support

An interactive graphics system called GSYM, for Graph SYMmety, was developed to aid in investigating visual representation techniques for graphs. GSYM allows a user to create, manipulate and display graphs using an IBM 2250 Graphic Display Unit [5]. GSYM is a special graphics display tool designed solely for investigating the problems involved in using a computer to generate visual representations of graphs. It should not be confused with existing graph processing languages [3,4] as it is not a programming language per se although it does provide macro-like commands for operating on graphs in certain situations. It also contains a list processing subsystem, but is not primarily intended to be a list processing tool.

2.1. System Operation

GSYM was designed to be used by individuals such as mathematicians and graph theorists that

Siggraph '77, July 20-22 San Jose, California

would have little, if any, experience with computers. The goal was to create a computer tool that could provide researchers with further insight into the structure of their graph theoretic models. However, we had to entice, for the most part, non-computer users to leave their offices and journey to the graphic display. The display, of course, was inconveniently located in the basement of the Mathematics and Computer building behind the closed doors of the Computer Centre. Thus, there was considerable pressure to make GSYM very easy to use. For example, we had to avoid the usual hurdles for a novice of learning awkward syntax rules for some programming language and being forced to stumble through a maze of job control language.

GSYM operates on what has been termed 'interacttion by anticipation' in that the interactive system attempts to guide the user through the sequence of events that causes a particular operation to be performed. The possible operations the user may initiate at any given moment are displayed and the user selects one of the current options using a light-pen. The system is said to be in the 'reset' state when the option list illustrated in Figure 1 is displayed. Once a basic option is selected, all the possible suboptions are given. After choosing the ADD option, for example, the user selects the element to be added to the graph from the list in Figure 2. If a vertex was being added, the system would request the user to position the vertex on the screen using a light-pen.

A similar process is followed for all operations. That is, GSYM displays a list or menu of the options available to the user at the moment and awaits the user's response. This type of format not only guides the user but also eliminates the possibility of system error owing to invalid user input. In order to reduce the frequency of user error the option lists contain a brief instructive note defining the nature of the current operation and the user action required. Moreover, should the user change his mind or wish to cancel an operation in the middle of a command sequence he may return to the reset state simply by pushing a programmed function key. Thus, the system is very forgiving, encouraging inexperienced users to become proficient in its operation through actual 'hands-on' experimentation. These features are especially important since as a research tool, GSYM is intended primarily for graph theorists and mathematicians. Such users understandably have little patience with the rigidity and poor man-machine interfaces that more experienced computer users seem to have accepted as the norm.

The wide variety of available operations includes addition, deletion and alteration of graph entities. The graphs are composed of vertices, edges and arrows. An arrow is always associated with an edge, that is, an undirected edge becomes a directed edge when the user adds an arrow (direction) to the edge. Move, rotation and translation commands manipulate the form of the graph as currently displayed on the screen. The user is able to move vertices, edges and arrows about the screen or, if he wishes, he may translate the whole graph. The rotation command rotates a graph about any point in the 3-dimensional cube in which it is defined. This cube corresponds roughly to the housing of the 2250 display screen.

2.2. GSYM Facilities

GSYM provides all of the graph manipulation operations allowed by earlier systems [9]. It also allows the user to associate lists of properties with vertices and edges. This facilitates the implementation of graph theoretic algorithms which operate on vertex and edge properties. The user is also able to create lists of vertices and edges by selecting list elements with a light-pen. This simple list-processing subsystem is adequate for most graph theoretic routines which operate on sets of graph elements. In anticipation of future expansion to allow command lists to be interpreted by the system, GSYM has a 'macro mode' whereby the user types in macros, containing all the information required by the system to perform the next operation. For example, to add a vertex one would type:

ADD V,(vertex coordinates),'label',(label coordinates),property list.

From an execution time standpoint macro mode is very efficient, but it requires a more knowledgeable user and more user time in order to type the command. However, a user written program running under the GSYM system may control the operation of the system by directing commands in the form of macros to the system. An ENQUIRE option permits users to quiz the system at any time concerning the current status of a graph, its properties and any graph related entities such as list pointers. It is also possible to have this information sent to a line printer for later reference and examination.

The user is able to save graphs on secondary storage and restore them later. Both the graph and the current status of the system are stored so that when a graph is restored the display as well as the data base associated with the graph is recreated. This feature allows the user to save displays created by visual representation routines and recover them at any time.

The graphs are stored as members of a file directory which the user creates before saving any graphs. There is no limit to the number of directories a user may have. In order to switch from one directory to another it is simply necessary to reference the new directory. Each graph in a directory is given a name when saved and may be renamed or replaced by another graph. The user does not become involved in the actual detailed creation and manipulation of the files containing these graphs. In particular, it is not necessary to know any command language statements for creating new files or referencing old files. All such file operations are handled dynamically by GSYM and are transparent to the user. The same standard control language is used to run GSYM regardless of what files will be used. The typical GYSM user is not likely to possess this knowledge and is unlikely to be inclined to acquire it in order to use the system.

Since GSYM is intended for designing and testing representation schemes, it has the ability to call user programs into execution. This feature has also been used to provide several special functions such as reading graph descriptions from input files, debug tracing of GSYM and smoothing

edges drawn with the light-pen. Although this feature is intended for testing visual representation routines, it is possible to incorporate any type of graph theoretic routine. For example, a program to calculate Hamiltonian paths could use the GSYM list processing subsystem to create lists representing any such paths in a graph. Space in the display processor's memory has been reserved to allow user programs to create their own displays if needed. All system functions for manipulating and interrogating a graph or its associated data base are available to user programs through the macro facility. The result is that it is a very simple matter to incorporate new representation routines. The user is isolated from the details of display file and data base manipulation.

2.3. Picture Format

The display screen layout for graphs in GSYM was designed in view of the effect it would have on pictures drawn by visual representation routines running under GSYM. Several screen layout problems and some of the steps taken to solve or avoid them are considered below.

Vertices and edges are displayed as points and straight lines, their natural representation. Arrows represent directions on directed edges and may be positioned anywhere along a directed edge. The positive and negative ends of an edge are indicated by arrow orientation. There are eight possible orientations corresponding to the eight major points on a compass.

The screen of a display unit is a two-dimensional entity and is therefore limited to two-dimensional pictures. If GSYM restricted itself to planar graphs with all visual representations being planar maps, such a two-dimensional display system would be quite adequate. However, GSYM is intended for use with both planar and non-planar graphs. While this did not preclude the possibility of generating pictures of non-planar graphs with intersecting edges, a more serious problem was the question of how to display multiple edges between vertices in a two-dimensional system. Thus, restricting GSYM to a two-dimensional coordinate system would severely constrain users' visual representations. For example, it would not be possible to generate a picture of a graph as a three-dimensional object and this is a natural representation for many graphs. For these reasons all GSYM graphs are defined using a three-dimensional coordinate system.

An edge is displayed as a straight line joining the x and y coordinates of the Cartesian (x,y,z) coordinates of its end-vertices. This representation implies that the user may need to rotate the graph in order to see more than just one of its faces. The GSYM rotation facility allows the user to rotate a graph about any point in the cube in which the graph is displayed.

There is one exception to the rule of three-dimensional graph entities, a special two-dimensional edge called a 'light-pen' edge. The user may specify that he wishes to draw a freehand representation of an edge using the light-pen. The edge is then displayed as a sequence of short straight line segments tracing the path of the light-pen. Light-pen edges solve the display problem caused by several edges being incident to the same pair of vertices since these edges may be drawn as curved lines. However, the primary importance of this feature is that it allows the user a wide range of choices in altering visual representations produced by the computer. Straight line edges can be replaced by curved edges which the user feels improve the picture.

3. Visual Representation Techniques

Our initial interest in computer generated representations of graphs was the result of an attempt to create pictures of a class of cubic, cyclically 4-connected graphs produced by Faulkner [2]. For every region size $n \geq 6$ in this class of graphs there exists a graph consisting of two polygons of n - 2 edges each such that every vertex in one of the polygons is adjacent to exactly one vertex in the other polygon. These graphs can be drawn as concentric polygons or rings of quadrilaterals as illustrated in Figure 3.

The ring structure was used as a basis for representations of more complicated graphs in the class. The ring structure is allowed to contain pentagons, hexagons and larger polygons. Once the ring structure is complete the remaining edges, if any, can easily be added to the interior of the ring. Figure 4 shows a 25 region graph. This display procedure produced reasonable pictures; however, excessive computer time was often necessary to calculate the ring structures. Moreover, the pictures could not be altered by the user. Finally, the representation routine was suited only to the one class of graphs. This lack of flexibility led directly to the development of GSYM and its use in the investigation of alternative representation techniques.

3.1. Symmetry

The most apparent and the most interesting feature of the pictures produced by this first representation routine was the frequent repetition of symmetry between elements of the graph. Symmetry, particularly mirror-image symmetry, occurs repeatedly in nature and in man's own creations. Weyl gives an excellent discussion of the nature of symmetry and its use in art and architecture through the ages [8]. The possibility of portraying symmetries in graphs suggested using the automorphisms of a graph as a basis for creating pictures of the graph. The cycles in an automorphism mapping of a graph can be interpreted visually as mirror-image and rotational symmetries.

Calculation of the automorphism mappings of most graphs is a long computational task. Heuristics have been used to determine whether graphs are isomorphic [7]. Similar heuristics were derived to calculate the automorphism mappings of a graph. Vertex valencies define an initial partition of the vertex set of a graph. Additional tests such as the number of vertices in each neighbourhood of a vertex and the number of polygons through a vertex are then used to attempt to refine the partition. Numerous such heuristics were investigated before selecting a set of heuristics which worked well with most graphs.

The amount of computation time required obtain the automorphisms depended on the heuristic tests and parameters applied to the graph. Experimentation with different tests and parameters on the same graphs indicated that a combination could usually be found that reduced the computation considerably. In order to make the calculation of the automorphism mappings feasible, the heuristics were implemented under GSYM. The operation of the heuristics could then be monitored on the display. The user interacts with the system by dynamically initiating and halting heuristics and varying parameters. Using this interactive approach it became practical to calculate automorphisms to use in generating visual representations of a graph. Different heuristics were easily implemented and tested as user programs under GSYM. The vertex set partition was handled using the list processing subsystem in GSYM.

3.2. General Representation Routine

Our first translation of the automorphisms of a graph into pictures was a very general representation routine. This routine attempts to handle all classes of graphs without any additional information about the graphs beyond the symmetries in the automorphism mappings.

Vertices fixed, that is, mapped onto themselves by an automorphism, are displayed as axes of symmetry around which are placed mirror-image vertex pairs, vertices interchanged by the automorphism. Automorphisms with cycles of one or two vertices become the mirror-image symmetries of Figure 5. Cycles of more than two vertices are treated as rotational symmetries. Each rotation is placed in a different plane in the three-dimensional cube in which GSYM graphs are defined. Figure 6 illustrates an example of rotational symmetry.

This approach tends to produce rather cluttered pictures as graphs become more complex. Poor placement of the mirror-image vertices or the rotations also has the same effect. Postediting facilities were therefore implemented to allow the user to modify and improve the generated pictures. The user may stop the representation routine at any point, modify the picture produced so far, and then allow the routine to continue to add further mirror-image and rotational symmetries. The user also controls the placement and size of the rotational symmetries. Finally, the user may edit symmetries as a unit. For example, moving a vertex in a mirror-image symmetry causes its corresponding vertex to move so as to maintain the original symmetry. This semi-automated display process produces representations of comparable quality to those generated by hand and does so in a fraction of the time and effort required to draw the same graphs manually.

The above approach has the fault that the quality of the pictures produced varies with the automorphism mapping used. Automorphisms with relatively few fixed vertices generally produce the best displays. A large number of vertices on an axis of symmetry obscures the structure of the graph because all the edges joining these vertices form a single line on the display. Moreover, large graphs or graphs with a large number of symmetries may require considerable postediting before a satisfactory picture is produced. The reason is that this representation routine uses only the symmetry information given by the automorphism mappings of the graph in order to display the graph. While the routine displays all the symmetries it lacks a frame around which to group these symmetries. The advantage of this approach is that the routine may be used with moderate success with many classes of graphs. The disadvantage is that this approach does not allow for situations where users are investigating problems where the graphs involved share common properties and structures.

3.3. Tree Representation

Trees are a class of graphs found in many applications. Trees also possess a high degree of known structure in that all vertices in a tree can be defined in terms of their distance from the centre or bicentre of the tree. Finally, the general symmetry routine, in ignoring the fact that a graph was a tree, produced pictures that rarely even hinted at the tree structure of the graph. It was apparent that the acceptability of the pictures produced by this general approach was provisional on the user not having a preconditioned idea of what the resultant picture should portray.

Not wishing to abandon the symmetry approach, a tree representation routine was designed which, in addition to the automorphism symmetries, considered the tree structure of the graph in the pictures generated. Mirror-image and rotational symmetries were combined with vertex distance from the centre or bicentre of the tree to produce pictures such as the one illustrated in Figure 7. Distance from the centre or bicentre determines the level at which a vertex is placed. Mirror-image and rotational symmetries are considered in ordering vertices on each level so as to illustrate the symmetry. The result was a representation routine that generates excellent pictures of trees and requires little or no postediting of the generated display.

4. Discussion

The pictures produced by the representation routines have been encouraging, even though the original objective of using a computer to automatically produce pictures of graphs has only partially been met. The representation methods tested required user postediting of the pictures. However, the need for postediting was considerably reduced when the representation routines considered more of the structure of the graph than the elementary automorphism mapping symmetries.

The two representation routines discussed above are radically different in nature. The first is totally naive about the structure of the graph while the other knows everything about it. Few classes of graphs lend themselves so well to a visual portrayal as do trees. However, just as with the development of specialized programming languages, we anticipate a parallel development in the creation of visual representation routines - a proliferation of specialized routines intended for specific classes of graphs or particular application areas. These routines would be based on properties that, when interpreted visually, determined the layout of the graph.

For example, one such property is planarity. All planar graphs can be embedded in a single plane without intersecting edges. Tutte gives an

algorithm for drawing such a graph in the plane [6]. Unfortunately, Tutte's algorithm tends to produce pictures where a large number of edges are drawn within a very small area. As a result, for most practical purposes, the algorithm proves to be unsatisfactory.

The GSYM system in combination with the representation routines is designed as a tool for graph theorists. It provides a means whereby a graph thoerist can easily investigate the structure of a graph or class of graphs in which he is interested. The representation routines should also prove useful for communicating information on classes of graphs between researchers. At present all that could be made available is a list of the numeric representations of the graphs. However, the representation routines could produce, in a systematic fashion, pictures of the graphs as well.

Applied use of automated representation procedures will require the implementation of more specialized routines than those described here. However, applications should be possible wherever structural symmetry in modelling relationships is important. For example, in the area of fault diagnosis functional simulation is one technique for digital fault simulation [1]. With this method a digital system compused of modules M_1, M2,...,M is to be simulated. Overall system behaviour is to be determined using the assumption that there are faults in only certain of the modules. The remaining modules need not be simulated at the gate level. It is only necessary to compute the output of these modules as a function of their input. If such a system were defined as a graph, then a representation routine could display the symmetries in the system. These pictures would indicate which modules should or should not be simulated in detail. For example, if modules M_i and M_j form a mirror-image pair, then if module M_i is not being simulated at the gate level the mirror-image symmetry implies that neither should module M_j be simulated at the gate level. That is, the symmetry in the system could help determine how to set up the simulation tests.

USE LIGHT-PEN TO SELECT OPTION:

* ADD
* ALTER
* BACKUP
* DECLARATION
* DELETE
* DISPLAY
* ENQUIRE
* HARD COPY

*** HALT ***

* MISCELLANEOUS FUNCTIONS
* MODE
* MOVE
* POINT
* ROTATION
* TRANSLATION

FIGURE 1: RESET STATE DISPLAY

USE LIGHT-PEN TO SELECT ELEMENT TO ADD:

* VERTEX
* EDGE
* DIRECTION
* DIRECTED EDGE
* LABEL

*** PFK 30 TO RESET ***

FIGURE 2: ADD OPTION DISPLAY

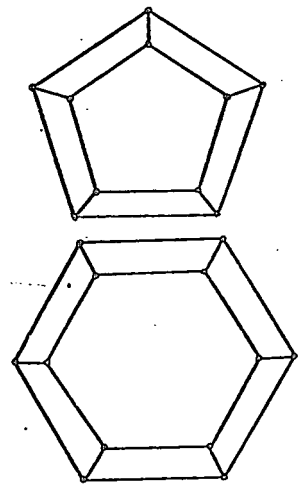

FIGURE 3: POLYGON RING STRUCTURE REPRESENTATIONS.

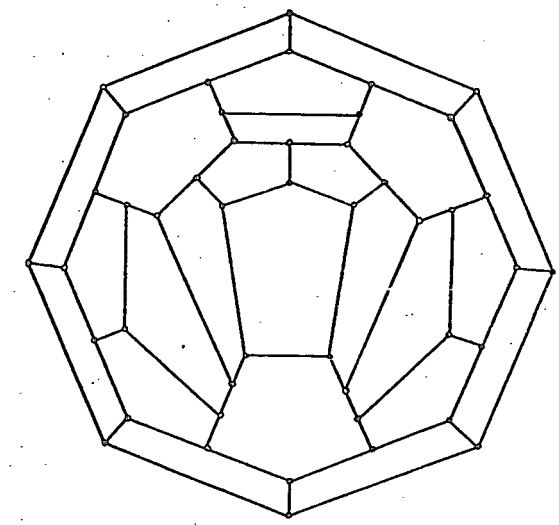

FIGURE 4:
MORE COMPLEX RING STRUCTURE REPRESENTATION.

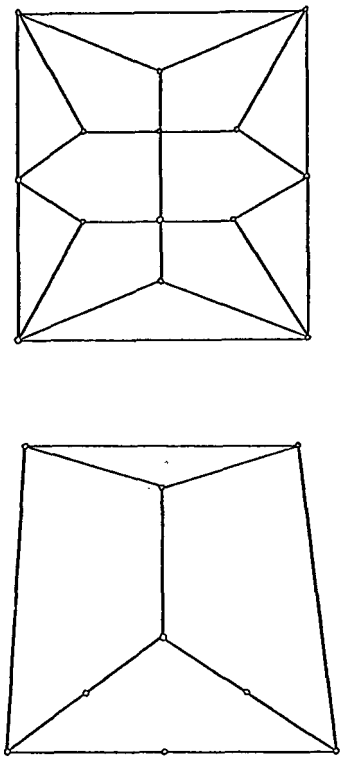

FIGURE 5: MIRROR-IMAGE SYMMETRIES.

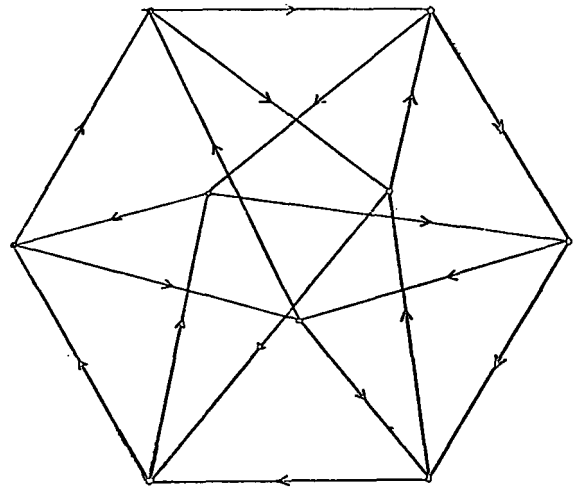

FIGURE 6: ROTATIONAL SYMMETRY.

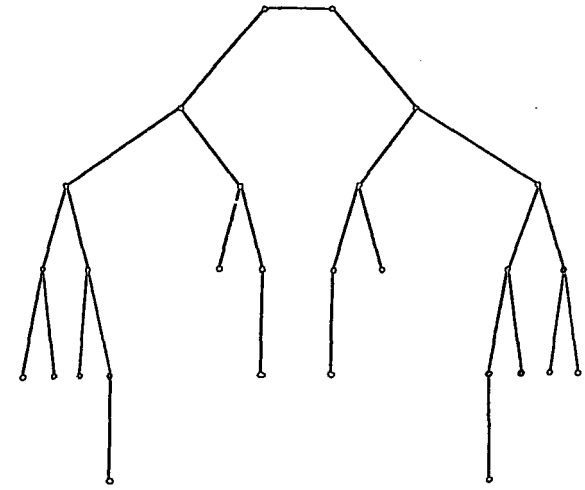

FIGURE 7: A TREE REPRESENTATION.

References

1. Chang, Manning and Metze. Fault Diagnosis of Digital Systems. Wiley, 1970.

2. Faulkner, Gary Bruce. Recursive Generation of Cyclically K-Connected Cubic Planar Graphs. Ph. D. Thesis, Department of Combinatorics and Optimization, University of Waterloo (1971).

3. Friedman, Daniel P., Dickenson, David C., Fraser, John J., and Pratt, Terrence W. GRASPE 1.5: A Graph Processor and its Applications. Department of Computer Science, University of Houston (1969).

4. Hurwitz and Citron. GRAF: Graphic Additions to FORTRAN. Proceeding SJCC 1967, 553-558.

5. IBM System/360 Component Description IBM 2250 Display Unit Model 1. Form A27-2701-2, IBM Data Processing Division, White Plains, N.Y.

6. Tutte, W. T. How to Draw a Graph. Proceedings London Mathematical Society 13 (1963), 743-767.

7. Unger, S.H. GIT-A Heuristic Program for Testing Pairs of Directed Line Graphs for Isomorphism. Comm. ACM 7, 1 (Jan. 1964), 26-34.

8. Weyl, Herman, Symmetry. Princeton University Press, 1952.

9. Wolfbert, Michael S. An Interactive Graph Theory System. Moore School Report No. 69-25, University of Pennsylvania (June 1969).