THE GGRAM SYSTEM:

AN INTERACTIVE GRAPHICS SYSTEM FOR GRAPH MANIPULATION

by

R. Doug Humphreys

B.Sc., University of British Columbia, 1972

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the Department

of

COMPUTER SCIENCE

We accept this thesis as conforming

to the required standard

The University of British Columbia

October 1974

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of COMPUTER SCIENCE

The University of British Columbia
Vancouver 8, Canada

Date OCT. 22, 1974.

ABSTRACT

The design and implementation of an interactive graphics system for graph manipulation are discussed. The motivation for such a system is examined, and the relevant literature is described and evaluated. A number of ways to improve and extend the system are presented.

The system provides the basic graph drawing operations of adding, deleting, labeling, and changing both vertices and edges. Also included are a number of graph manipulation operations which, among other things, allow a user to subdivide edges, associate vertices, reverse the direction of arcs, move vertices about the screen, or even move whole graphs about the screen.

A facility is provided whereby the screen can be divided into as many as four regions, thus allowing users to display more than one graph at a time. Graphs can be saved on disk and later restored. The image on the graphics screen can be easily plotted to obtain a hard copy of graphs.

A few routines which perform graph-theoretic operations have been implemented. Among these are a routine for finding the minimum and maximum degrees of a graph, and a routine for finding the blocks, cutnodes, and bridges of a graph. Moreover, the system is designed to allow users to add their own routines.

TABLE OF CONTENTS

CHAPTER I - GRAPHICS AND GRAPH THEORY 1

1.1 INTRODUCTION 1

1.2 MOTIVATION 2

1.2.1 Research 3

1.2.2 Education 4

1.2.3 Applications 5

1.3 RELEVANT LITERATURE 5

1.3.1 Graphics Systems For Graph Theory 6

1.3.2 Other Relevant Work 8

CHAPTER II - SYSTEM DESIGN 11

2.1 SYSTEM ORGANIZATION 11

2.2 SCREEN DESIGN 14

2.2.1 Menu 14

2.2.2 Regions 17

2.2.3 Message Area 17

2.2.4 Vertex And Edge Counts 18

2.3 FACILITIES 18

2.3.1 General Facilities 18

2.3.2 Graph Drawing Facilities 22

2.3.3 Graph-theoretic Operations 24

2.4 GENERAL CONSIDERATIONS 25

2.4.1 Free-hand Edges 25

2.4.2 Multiple Edges 26

2.4.3 Loops 26

CHAPTER III - IMPLEMENTATION 27

3.1 PROGRAMMING LANGUAGES 27

3.1.1 Choice Of Implementation Language 27

3.1.2 Effect Of Implementation Language On The System 31

3.2 HARDWARE CONFIGURATION AND SOFTWARE SUPPORT 32

3.2.1 The Adage Graphics Terminal 33

3.2.2 The GRAPH Supervisor Program 33

3.2.3 Effect Of Hardware And Software On The System 34

3.3 DATA STRUCTURES 37

3.3.1 Internal Representation Of Graphs 37

3.3.2 Shadow Buffer 44

3.3.3 Scratch Buffer 47

3.4 OVERVIEW OF THE PROGRAM 48

CHAPTER IV - IMPROVEMENTS AND EXTENSIONS 50

4.1 IMPROVING THE EXISTING SYSTEM 50

4.2 EXTENSIONS 51

4.3 SUMMARY 57

BIBLIOGRAPHY 59

APPENDIX I - USER'S GUIDE 61

LIST OF FIGURES

1 • The Fields in a GRAPH Record 38

2 • The Fields in a VERTEX Record 39

3 • The Fields in an EDGE Record 40

4 • The Fields in an EDGEDES Record 41

5 • The Fields in an CONSPT Record 41

6 • The Link Structure of a Small Graph 43

7 • The Screen When the System is Running 62

8 • The DRAW and FUNCTION Menus 69

9 • The Effect of SUBDIVIDE EDGE 87

10 • The Effect of REMOVE DEGREE 2 VERTEX 88

11 • The Effect of ASSOCIATE VERTICES 90

12 • The Effect of PERMUTE LABELS 95

13 • The Effect of PERMUTE COORDS 96

14 • The Effect of BLOCKS/CUTNODES/BRIDGES 98

ACKNOWLEDGMENT

I would like to express my sincere appreciation to Dr. Abbe Mowshowitz, not only for his valuable help during the course of this work, but also for his guidance during the past few years. I would also like to thank Dr. J.M. Kennedy for his helpful suggestions for improving this thesis.

CHAPTER I - GRAPHICS AND GRAPH THEORY

1.1 INTRODUCTION

The growing importance of graphics in computer science is certainly not surprising. Most people who have seen graphics terminals in operation can attest to their effectiveness in presenting visual information. For this reason, many graphics systems for particular applications have been developed. What is surprising, however, is that it appears that little work has been done in developing graphics systems for solving graph-theoretic problems.

The most natural way to present a graph for someone's inspection is by means of a diagram. Most people are likely to prefer a diagram rather than an adjacency matrix[1], for example. Thus a graphics terminal is well suited for displaying graphs.

This thesis describes the design and implementation of the GGRAM system, an interactive Graphics system for GRAph

---

[1]For graph-theoretic terms not explicitly defined here, see Harary [9].

Manipulation. The system is designed to allow a user to draw and manipulate graphs conveniently and quickly. It provides the basic operations of adding, deleting, labeling, and changing both vertices and edges, as well as a few special operations such as "subdivide edge" and "associate vertices". A number of facilities of general utility are also provided. These include commands for dividing the screen into as many as four regions, plotting the image on the screen, and saving graphs on disk and later restoring them. Furthermore, a few graph-theoretic algorithms have been implemented, and the system is designed in such a way as to allow users to easily add their own routines.

1.2 MOTIVATION

There are at least three reasons for developing a graphics system for manipulating graphs. First, a properly designed system could be useful to graph theory researchers. Second, such a system could help in the teaching of graph theory. And third, such a system could be used for solving specific classes of real-world problems.

1.2.1 Research

A graphics system for manipulating graphs could be used in its simplest form for investigating the properties of graphs. For example, a user could test for isomorphism between two graphs by displaying them both and then manipulating one to try to make it look like the other. Or a user might try to see if a graph is planar by trying to manipulate the graph so as to remove all edge-crossings. These operations make direct use of the intuition and experience of the user to help solve the problem.

The effectiveness of such procedures could be increased by providing the user with facilities for saving and later restoring graphs. Thus, the original graph and selected intermediate results could be saved, allowing the user to back up if necessary.

But while such procedures might be useful, they are clearly limited in what they can accomplish. What a system for graph manipulation really needs is a means for users to add their own routines. If this were possible, a user could tailor the system to meet his (or her) particular needs. For example, a user interested in planar graphs might implement an existing algorithm that tests for planarity. Moreover, he could use the system to test new algorithms.

Research using a graph manipulation system need not be

restricted to graph theory itself. The importance of graph theory stems partly from its wide range of applicability to real-world problems (for example, see [20]). There is no reason why these problems could not be attacked with the help of a graphics system.

1.2.2 Education

An interactive graphics system for graph manipulation has great potential for education. An annoying thing about learning graph theory is the seemingly endless number of definitions and concepts which must be learned before more interesting problems are considered. A graph manipulation system could make learning such things somewhat more enjoyable. In addition, a graph manipulation system, by its nature, would be very effective for illustrating concepts. For example, the concept of a cutnode could be illustrated by actually seeing what happens when a vertex is removed from a graph. Such a procedure is much more effective than, say, a blackboard demonstration simply because of its greater speed and visual impact.

As is the case with reasearh, the educational potential of a graph manipulation system is not limited to graph theory. Teachers in other disiplines could use such a system to illustrate concepts which can be formulated in terms of graphs. For instance, a probabilist might use it to demonstrate certain characteristics of Markov chains.

1.2.3 Applications

A graph manipulation system could also be used simply as a problem solver. For example, a user might implement a routine to find a critical path in a network, and then use this routine to solve real problems. The advantage in using a graphics system for this is that input is in the form of a diagram of the graph, rather than some other representation. This would likely be very helpful, especially since the user could also use the system to develop the model. A graphics system could also make changing the model easier.

1.3 RELEVANT LITERATURE

Although graph theory seems like a natural application for a graphics system, there appear to be few reports of such applications in the literature. Some work that has been done is described below. First, the work which is directly related to graph manipulation systems is examined. This is followed by a short discussion of indirectly related work.

1.3.1 Graphics Systems for Graph Theory

Two systems of special interest to the present study were developed by Wolfberg [20] and Maguire [12]. Only a general description of each is presented in this section as an indication of the nature and scope of the work. More detailed comments appear in Chapter II, where these systems are considered in relation to the GGRAM system.

A large part of Wolfberg's study encompasses the design and implementation of support systems to control the various devices on which the system runs, as well as consideration of programming languages for use on these devices. Of primary interest here, however, are the system's facilities as far as graph theory is concerned. The system provides a "complete graph drawing and editing facility" [20, p. 27]. Vertices and edges may be added, deleted, labeled, or changed. Vertices can have eight different shapes (including null), and vertices and edges can be dimmed or blinked. A light pen and menu are used to choose particular operations.

The system allows users to save any number of graphs and later restore selected ones. "A library of graph-theoretic algorithms is maintained so users may apply certain interactive algorithms to arbitrary graphs. Other algorithms aid in construction or recognition of particular types of graphs" [20, p. 27].

Graphs are drawn on what Wolfberg calls the "paper". One of the more interesting features of his system is the ability to look at the paper through various sizes of windows. The largest window is the same size as the paper. Three other smaller windows exist. As the windows become smaller, the visible portion of the graph also becomes smaller, but the graph in this area is magnified. In other words, a user can zoom in on a particular area of a graph.

Wolfberg's system is very comprehensive in terms of the facilities it offers. It is, however, sometimes cumbersome in its method of drawing graphs. More will be said about this in Chapter II.

Maguire [12] has developed "An Interactive System for Displaying Linear Graphs" called the GSYM (Graph Symmetry) System. It is a system "especially designed for working on the problem of generating displays of graphs" [12, p. 1], which allows a user to "create, manipulate, and display graphs" [12, p. 1]. Graphs are composed of vertices, edges, and arrows. A useful feature of this system is that vertices and edges can be assigned properties. For example, edges in a network might be assigned capacities. This feature is independent of labeling.

System operations are initiated by choosing, with a light pen, one of a number of options which the system displays. Again the basic operations are adding, deleting, labeling, and changing vertices and edges. Individual vertices, edges, and

arrows, or even a whole graph, can be moved about the screen. Graphs can be saved and restored, and GSYM provides for the addition of user-written routines.

Perhaps the most interesting part of GSYM is the way graphs are stored and displayed. The author states:

> GSYM graphs are defined using a three-dimensional co-ordinate system ... An edge is displayed on the screen as a straight line joining the x and y co-ordinates of the Cartesian (x,y,z) co-ordinates of its end-vertices. This three-dimensional Cartesian representation implies that the user must be able to rotate the graph in order to see more than just one face of the graph [12, p. 14].

The GSYM system offers a fairly complete set of facilities to the user. There are a few desirable facilities, however, which have not been included. Moreover, like the previous system, the GSYM system has a somewhat awkward procedure for drawing graphs. This is mentioned further in Chapter II.

1.3.2 Other Relevant Work

Di Giulio and Tuan [7] have developed a graphics system to draw and manipulate systems analysis networks. These networks are essentially directed graphs used to represent project scheduling problems, flows in networks, and so forth. Although their work encompasses both graphics and graph theory, it is included here rather than in the previous section because it is extremely specialized. Using their system, systems anaylsis networks may be drawn at a graphics terminal with a joystick and

a light pen. Graphs are stored as adjacency matrices and manipulations are performed using these matrices. Operations which can be applied to graphs include union, intersection, and a few others. The system is oriented towards displaying graphs, and it automatically generates diagrams of graphs from their adjacency matrices. An algorithm is presented for displaying acyclic directed graphs in such a way that all arrows point more or less in one direction and line-crossings are minimized.

A number of programming languages for graph theory have been developed by various people. These languages tend to be extensions of an existing language which enable graph-theoretic operations to be expressed naturally and conveniently. Crespi-Reghizzi and Morpurgo [6] have extended ALGOL 60 to include operations on graphs such as addition and deletion of vertices, the union of two graphs, or the intersection of two graphs. New data types are also provided for four different classes of graphs.

Pratt and Friedman [16] have developed a programming language extension for processing directed graphs. This extension is formulated in terms of the semantics of certain primitive operations which operate on graphs. It is not designed to be embedded in a specific language, although an example of an embedding in LISP is presented.

Another graph-theoretic language has been developed by Rheinboldt et al. [18]. It is an extension of ALGOL 60 for

"describing and implementing graph algorithms of the type arising in applications" [18, p. 220].

A similar language has been developed by King [11]. He has extended FORTRAN to include facilities to perform graph-theoretic operations.

CHAPTER II - SYSTEM DESIGN

In this chapter an attempt is made to justify the design of the GGRAM system by considering those features which are desirable in an interactive graphics system for graph manipulation. First, system organization is discussed. Then screen layout is considered, and the reasons for choosing certain system facitlities are looked at. Finally, a number of general considerations are examined. Throughout the discussion the GGRAM system is compared to the two similar existing systems which were described in Chapter I, Section 1.3.1. For parts of this chapter, readers may find it helpful to refer to figure 7, a plot of the screen while the system is running.

2.1 SYSTEM ORGANIZATION

The GGRAM system has been implemented within the framework of what might be called a typical hardware configuration (c.f. [20, p. 6]). The graphics facility consists of a graphics (CRT) terminal driven by a general purpose graphics computer. Various input devices, such as a light pen, pushbuttons, and variable control dials, are attached to the graphics terminal. The graphics computer is connected to a

large main computer by way of a communication line. Associated with the graphics terminal, but connected to the main computer, is a normal (CRT) terminal. Programs being run at this (normal) terminal are allowed to communicate with the graphics computer using the communication line.

When designing a system using such a configuration, a number of decisions must be made. One of the most important decisions is how to divide the computing duties between the graphics computer and the main computer. This decision can be influenced by many factors, including the desired response time, time constraints, cost constraints, or the scope of the work, to name a few.

In most graphics systems there are certain functions which can be handled locally by the graphics computer. For example, in a graph manipulation system information about the layout of displayed graphs, and changes to this layout, can be handled in this way (see [20, p.21]). This approach, however, necessitates the implementation of a special supervisor program for the graphics computer. But the development of such a supervisor program is usually only necessary if communication with the main computer is too slow for a particular application. Thus, for special applications like computer animation it may be necessary to perform local processing if frames cannot be received from the main computer at a high enough rate. On the other hand, graph manipulation requires only that the

communication time be reasonable (see [13]).

In the GGRAM system all computing is done in the main computer. A general purpose graphics supervisor [4] is used at the graphics computer to display images on the graphics screen, read the pushbuttons and dials, and detect light pen hits (see also Chapter III). Special routines [5] are provided to communicate with the graphics supervisor.

Within the main computer the GGRAM system is divided into components in a straightforward and perhaps obvious manner. One important component is the graph data area which contains the internal representation of graphs. Information from this component is used by display routines to construct images to be sent to the graphics computer. A group of construction routines are used for changing these structures.

The remainder of the system consists of routines which roughly fall into three categories: control routines, command routines, and utility routines. At the top level, control routines communicate with a user to determine what operation is desired. When an operation is specified, the control routines call the command routine(s) necessary to accomplish the operation. Certain utility routines are also available to perform special functions.

2.2 SCREEN DESIGN

The method of presentation of information to a user is usually very important in a graphics system. Some of the methods which seem appropriate to a graph manipulation system are described in this section.

2.2.1 Menu

Perhaps the most effective way for a user to control the operation of a graphics system, in most cases, is by way of a menu. This usually means that all the operations that are possible at a given time are displayed on the graphics screen and a user chooses the desired operation with a light pen. Generally the menu is frequently changed, depending on the state of the system. Wolfberg and Maguire both use this approach. Another approach is to use a static menu, that is, all system facilites are always displayed. With this approach, it may not be possible to initiate some displayed operations at a given point in time.

Both approaches have their drawbacks. The first method is sometimes unsatisfactory when a user is frequently changing between operations, especially if there are many choices in each state. One problem is that it can take too long, relative to a user's speed of thought, to display a new menu. A similar problem is encountered with the GGRAM system when the variable menu is changed. Users might also be distracted by the visual

readjustment needed for a new menu. Part of the reason for this is that a new menu must be displayed sufficiently far from the location where the light pen may remain after pointing at a previous menu [20, p.172]. On the other hand, a static menu also presents problems. Since all operations are always displayed, the amount of information may distract (or even confuse) a user. A novice may have to search for the operation he wants. Furthermore, a user must be aware of which operations are allowable at a given point in time.

Of these two approaches, the first is unacceptable for the GGRAM system because it would slow down the graph drawing process too much. There are simply too many options to make a continually changing menu feasible. The second method, therefore, is the method that is used. This is not as bad as it might appear at first, for two reasons. First, both the objections to this approach become less important as a user becomes more familiar with the system. And second, the menu can be designed in such a way as to help the user find operations and know which are available at each point in time.

At all levels the menu is divided into logical sections. This helps a user find the command he is looking for. At the top level are the fixed and variable menus. The fixed menu contains commands of general utility. The variable menu can contain one of two menus: the draw menu or the function menu. The draw menu contains commands for drawing and altering graphs,

while the function menu contains commands for initiating graph-theoretic algorithms. The variable menu can be changed using a command in the fixed menu. This menu must be changed in this way because there is not enough room on the screen to allow both the draw and function menus to be displayed at the same time. A solid dividing line across the menu serves to physically separate the fixed and variable portions.

There is also a logical ordering within each menu. Closely related commands, such as the commands for saving and restoring graphs, are displayed on the same line. The draw menu is further subdivided into sections by two dashed dividing lines. The top section contains commands for changing the appearance of vertices and edges. The middle section contains the vertex and edge prototypes which show what vertices and edges will look like when added to a graph. The bottom section contains commands for actually drawing and altering graphs.

Some important commands are displayed in large letters. This not only serves to emphasize these commands, but also helps to orient a user by providing "landmarks". Certain commands, such as "add", can be applied to both vertices and edges. So, instead of cluttering the screen by displaying both an "add vertex" and "add edge" command, a single ADD command is displayed. A square to the left of ADD is for adding edges; a square to the right is for adding vertices. Since there are a number of commands of this type, two columns of squares are

formed. The left column, in line with the edge prototype, initiates operations on edges; the right column, in line with the vertex prototype, initiates operations on vertices.

2.2.2 Regions

The system allows the screen to be broken into as many as four regions. There are two major reasons for allowing this. First, it is often useful to be able to visually compare a number of graphs. For example, a user may want to determine by inspection whether or not two graphs are isomorphic; or he may want to see a graph before and after an operation such as associating vertices is performed. Second, a user may want to draw two or more graphs in parallel. A number of different yet related graphs might be drawn in this way.

2.2.3 Message Area

A narrow area at the bottom of the screen, called the message area, is reserved for messages to the user. A separate area is allocated for this function to emphasize the messages and also to prevent messages from interfering with graphs displayed on the screen. Messages are blinked to make them even more noticeable.

2.2.4 Vertex and Edge Counts

As a convenience to the user, for each graph on the screen the number of vertices and edges are displayed toward the lower left corner of the region. This is information that is frequently wanted. It also serves as a double-check when drawing specific graphs by allowing a user to make sure that all desired components have been drawn.

2.3 FACILITIES

In this section the reasons for including certain facilities in the system are examined. The three subsections correspond to the fixed menu, the draw menu, and the function menu, respectively. Detailed descriptions of the individual commands can be found in Appendix I.

2.3.1 General Facilities

Since the menu does not change, a user needs some way to tell when the system is expecting input. An indicator which blinks when the system is ready serves this purpose. To make this indicator highly visible, it is placed at the top of the menu and displayed in large characters.

A command is necessary to change how the screen is divided into regions. In the GGRAM system eight possible patterns are

allowed, with the number of regions being either one (one choice), two (two choices), three (four choices), or four (one choice). This provides a high degree of flexibility for displaying graphs.

Two commands are supplied to set the variable menu. The system is designed in such a way that, should the function menu become full from the addition of user-written routines, additional menus can be added.

Another command changes the system into keyboard mode[1]. In keyboard mode, the keyboard rather than the light pen is used to control the system. This mode is provided for three reasons. First, some people simply prefer typing to using the light pen. Second, it can be used to rapidly initiate a group of commands. This could be used, for example, to change a number of system defaults at the beginning of a run. To do this, the appropriate commands would be put into a file, and then this file used as input to the system. The file could be kept and used each time the system is run. Third, keyboard mode allows the system to be used even if the light pen hardware becomes inoperable.

Both Wolfberg and Maguire have similar facilities. Wolfberg displays a number or a letter beside each menu item. Typing this character is equivalent to choosing the item with

---

[1]Keyboard mode has not yet been implemented in the GGRAM system.

the light pen[2]. Maguire has implemented what he calls "macro mode", in which the keyboard is used for communication with the GSYM system.

A "help" command[3] is supplied to act as an interactive user's guide. When this command is activated, the user is asked a series of questions to determine what information is desired. This feature is most useful when a user gets into trouble or forgets something while running the system.

The system allows a user to change various global parameters. All parameters, of course, have defaults, but a user may wish to change some of these. For example, unless otherwise specified, the system places labels at a default (relative) location, but this location can be changed if desired.

The system allows users to selectively display labels. This means that some labels that are defined may not be displayed. Some graphs may be more readable when vertex and edge labels are not displayed simultaneously. A user may want to erase all labels for some reason. Wolfberg's system has a similar feature which allows labels to be defined but not

---

[2]The UBC graphics system rules out implementing keyboard mode in this way because it is set up to read from only one device at a time.
[3]The "help" command has not yet been implemented in the GGRAM system.

displayed.

An obviously useful feature for a graph manipulation system to have is the ability to produce a hard copy of a particular graph. For this reason a command is included in the GGRAM system for producing a plot of the whole screen.

Another obviously useful feature is an ability to save and later restore graphs. Commands for these two functions are also included in the GGRAM system.

The graph titling feature of the GGRAM system allows a user to annotate graphs with a title up to forty characters long. This is most useful when restoring graphs that have been saved. It is also convenient to be able to title a graph for display purposes. A user, however, is never forced to title a graph, even if the graph is to be saved. This is unlike both Wolfberg and Maguire, who insist a graph be titled before it is saved.

The system maintains a list of graphs that have been defined to it (either by drawing of restoring). Thus, commands to display and rub out graphs are necessary to control which graphs are displayed.

A command is supplied which allows a user to move graphs between regions. This involves rubbing out the graph and then re-displaying it in the new region.

The system automatically prevents a particular internal

copy of a graph from being displayed in more than one region. If this were allowed, it would create havoc if the graphs were altered. Sometimes, though, the same image is desired in more than one region. To do this, distinct internal copies must be made and the copies displayed in the other regions. A graph duplicating command is provided for this purpose.

It is clearly desirable to allow a user to cancel (where possible) an operation. A command in the menu serves this purpose when the GGRAM system is reading from the graphics terminal. A user can also cancel when the system is expecting input from the keyboard. This is accomplished by entering the current contents of a global parameter which is initially "CANCEL", but it can be changed if desired.

2.3.2 Graph Drawing Facilities

It is very useful for display purposes to allow the physical characteristics of vertices and edges to be varied. Thus, the GGRAM system allows vertices and edges to be normal intensity or bold face, solid or dashed. The size and shape (number of sides) of vertices can also be changed. In order that a user can actually see what the vertices and edges that he will draw will look like, an edge prototype and vertex prototype are displayed in the menu.

This procedure is somewhat faster than the one employed by Wolfberg. He allows the appearance of vertices and edges to be

varied, but they are always initially created with the same characteristics. A user can then change them if desired. Maguire does not provide for vertices and edges to vary in appearance.

The edge prototype also serves as a convenient indication as to whether or not the system is in directed mode. In this mode, all edges that are drawn are considered to be directed. When the system is in directed mode, the edge prototype is displayed with an arrow.

The systems of both Wolfberg and Maguire require a user to add an arrow to an existing edge in order to make it into a directed edge. Thus, two operations are necessary to draw such an edge. This seems to be an unnecessary waste of effort.

The standard graph drawing operations, of course, are provided in the GGRAM system. These allow both vertices and edges to be added, deleted, labeled, or changed.

A number of special graphical operations are provided, some of which are related to particular graph-theoretic concepts. Homeomorphisms between graphs can be visually investigated using features which allow an edge to be subdivided, and a vertex of degree two to be removed. A command for associating vertices allows users to perform elementary homomorphisms on selected graphs. Directed graphs can be manipulated using commands which reverse the direction of an arc, and change an undirected edge

into a directed edge, or vice versa. A graph's layout can be changed by moving vertices around the screen or even moving the whole graph.

2.3.3 Graph-theoretic Operations

Only four graph-theoretic routines have been implemented to date. Their selection depended on personal preference and time constraints, and they are only mentioned here for completeness. The first is a routine for determining the minimum and maximum degreees in a graph. If the graph is directed, both the minimum and maximum indegrees and the minimum and maximum outdegrees are determined.

Another routine will permute the vertex labels of a selected graph, while a third routine will permute vertex coordinates. The latter routine is particularly useful for testing for and demonstrating isomorphism.

The final routine will find the blocks, cutnodes, and bridges of an arbitrary graph. It also finds a spanning tree, if one exists. The algorithm used in this routine is due to Read [17].

2.4 GENERAL CONSIDERATIONS

Three general considerations which are important to a graph manipulation system are discussed in this section. These are curved edges, multiple edges, and loops.

2.4.1 Free-hand Edges

The ability to draw bent or curved edges is essential to a truly flexible graph-drawing system. For this reason, the GGRAM system allows a user to draw edges which are made up of any number of line segments, rather than a straight line between vertices. These edges are easily drawn using the tracking pattern and the light pen.

Wolfberg allows a user to bend edges, but it appears that only one bend is allowed. Maguire allows "light pen edges" to be drawn. These edges are temporary curved replacements for existing straight edges. Unfortunately, these edges disappear when a graph is rotated within the three-dimensional space in which the graph is defined. After such a rotation, all light pen edges must be re-drawn.

2.4.2 Multiple Edges

Whenever a multiple edge is drawn, the GGRAM system displays a warning message to the user. This is done because it is possible to draw one of a pair of multiple edges on top of the other, in which case the pair looks like a single edge. A warning is particularly important if a user program causes a multiple edge, since such an occurence could easily go unnoticed by the user. But it is even quite easy to unintentionally draw a multiple edge with the light pen.

Maguire's system does not check for multiple edges. He leaves it up to the user to notice their existence. Wolfberg does not seem to indicate whether or not multiple edges are checked for or even allowed in his system.

2.4.3 Loops

Loops are allowed by the GGRAM system under the proviso that they be free-hand edges. Otherwise the edge could not be seen. Wolfberg allows loops, but they are always the same shape and must have either an east, west, north, or south orientation. This seems like an unnecessary restriction. Maguire does not indicate if loops are allowed in his system.

CHAPTER III - IMPLEMENTATION

This chapter is a discussion of the implementation of the GGRAM system. First, the criteria used to decide on the implementation language are discussed, and the effects of this choice on the system are examined. The hardware configuration and software support are then described, and their effects on the system are considered. Next, the major data structures used by the system are described. Finally, an overview of the program is presented.

3.1 PROGRAMMING LANGUAGES

3.1.1 Choice of Implementation Language

Most of the program is written in ALGOLW. There are also about a dozen routines written in FORTRAN, and one routine written in 360 ASSEMBLER.

There are a number of considerations which led to the choice of ALGOLW as the primary implementation language. First is the requirement that a structured data type be available to use for the internal representation of graphs. The

representation chosen makes extensive use of pointers (see DATA STRUCTURES below), and would be extremely cumbersome if implemented by means of arrays.

The ALGOLW record [1] is a one-level structured data type consisting of a number of fields which can be of different (simple) types. Records are pointed to by reference variables.

Closely related to structured data types, and equally important in terms of this implementation, is the requirement that dynamic storage allocation be a feature of the language. This is not only essential for record allocation, but is also useful for array allocation. In fact, dynamic array allocation is used quite often in the system.

A third requirement is that a convenient and efficient interface must exist (or be possible) between the implementation language and FORTRAN. In particular, the language must be able to call FORTRAN routines and accept returned values from these routines, and also must be able to handle FORTRAN arrays. The reason for this is that the library routines supplied for use with the Adage Graphics Terminal [5] either are written in FORTRAN or use FORTRAN calling conventions. Furthermore, these routines often have arrays as parameters.

To call FORTRAN routines from ALGOLW, one simply declares an external procedure and specifies that FORTRAN calling conventions are to be used [8]. A routine declared in this way

is used in an ALGOLW program in the same way as a normal ALGOLW procedure. The internal representation of ALGOLW (one-dimensional) arrays is exactly the same as the FORTRAN representation, and so arrays do not present a problem.

Two other requirements are also of some importance. First, the language should be known, or easily learned, by people who might use the system. Second, the language should be efficient both in terms of program development and program usage, that is, it should have an efficient compiler which produces efficient code. These two points are important primarily because the system is intended to be used, in part, as a research tool. Researchers who know the implementation language will be able to modify or extend the system to meet their own particular needs. If the system is expensive to run, researchers will be discouraged from using it. Furthermore, since a significant portion of research in graph theory relates to algorithm development, the compiling efficiency of the language must also be acceptable.

Those who are not familiar with ALGOLW but who know ALGOL 60, PL/I, FORTRAN, or a similar language, should not have any trouble learning ALGOLW. Furthermore, it is relatively efficient both at compile time and run time.

There are also several requirements which, although not essential, nevertheless make programming easier. First, the language should have reasonable string handling facilities.

This allows easy manipulation of graph titles, labels, messages, and so on. Second, the language should be recursive. This allows a few facilities to be provided which would be difficult in a non-recursive language. For example, the execution of commands is controlled by large CASE statements, and it is sometimes convenient to be able to call these CASE statements recursively. Such is the case when a certain command is to be executed while still executing another command. This is done, for instance, when the appearance of one of the prototypes is to be changed while "add vertex" is still activated. Finally, the language should be easy to write in. There are many language features that could be considered in this regard, many of which are simply personal preference. However, some features that could be included are a block structure, the existence of IF-THEN-ELSE, WHILE, and CASE statements, and the ability to use long (and meaningful) identifiers.

Three languages other the ALGOLW were seriously considered as the primary implementation language: PL/I, LISP, and FORTRAN. FORTRAN was rejected almost immediately since the only two requirements it meets are that it "interfaces to FORTRAN" and it is known by many people. LISP was considered somewhat more seriously, especially in light of the fact that a LISP interface to the Adage Graphics Terminal has already been written. As well as meeting most of the requirements mentioned above, LISP also provides excellent debugging facilities, and allows a user to easily modify existing routines and write new ones without

having to re-compile. LISP was rejected, however, mainly because of its inefficiency and its lack of widespread appeal to the class of users for whom the system was designed.

Both PL/I and ALGOLW met all the criteria described above; however, the PL/I compiler is somewhat less efficient than the ALGOLW compiler. Moreover, although PL/I is easy to write in, it does not have the consistency and inherent simplicity of ALGOLW; and despite the fact that linkage to FORTRAN from PL/I is possible, it is subject to special rules that are, at the very least, a nuisance. Thus, ALGOLW was chosen over PL/I for the implementation of the system.

3.1.2 Effect of Implementation Language on the System

The version of ALGOLW adopted by the Computing Centre of the University of British Columbia [8] certainly is not perfect. Although the language itself is excellent, the implementation is in many ways poor. Most of the problems have to do with size limitations of one sort or another. These limitations have had a number of effects on system development.

The restriction that "the data area for each PROCEDURE or BEGIN block with declarations is limited to 4096 bytes" [10, p. 30] has meant that any large arrays must be handled in FORTRAN. This involves setting up the arrays in COMMON blocks and writing FORTRAN routines (which can be called from ALGOLW) to manipulate the arrays. The "shadow buffer" and the "scratch buffer" are

handled in this way (see below).

The restriction that "only 255 different procedures or BEGIN blocks with declarations are allowed by the compiler" [10, p. 28] has meant that a number of related procedures have often been combined into one larger procedure in order to save procedure calls. For example, a procedure called DELETE_ITEM does the work of DELETE_VERTEX and DELETE_EDGE. This limitation, of course, has hampered efforts to write a truly structured program, and also partially restricts future expansion of the system.

A third restriction that has affected the program is that "the total amount of machine code and constants for any PROCEDURE or other BEGIN block with declarations must be less than 8192 bytes" [10, p. 30]. This has made it necessary to split a few procedures that are too large. For example, CONSTRUCT_DRAW_MENU was split into CONSTRUCT_DRAW_MENU1 and CONSTRUCT_DRAW_MENU2.

3.2 HARDWARE CONFIGURATION AND SOFTWARE SUPPORT

Two important influences on the design of a graphics system are the hardware that is available and the software that is supplied (if any) for use with this hardware. This aspect of the design of the GGRAM system is discussed in this section.

3.2.1 The Adage Graphics Terminal

The GGRAM system is implemented on an Adage Corporation Model 10 Graphics Terminal, which is connected to a general purpose graphics computer. An IBM 3270 Display Station next to the graphics terminal is connected to an IBM 370/168, and programs being run from the 3270 can communicate with the graphics terminal. The normal mode of operation is for a program being run at the 3270 to generate data which is transmitted to the graphics computer for display. Refer to reference [4] for more details.

3.2.2 The GRAPH Supervisor Program

When the GGRAM system is running, local control of the graphics terminal is handled by the GRAPH[1] supervisor program [4]. A number of FORTRAN-callable routines [5] have been written to allow users to communicate with this supervisor program.

The supervisor program displays an image on the graphics screen by continuously scanning a 6000 word display buffer. Each word in the display buffer contains an instruction. These instructions include ones for moving the light beam to a certain point, "drawing" the beam to a certain point, controlling dashing and scaling, enabling or disabling the light pen, and so

---

[1] from graphics, not graph theory.

forth.

To produce an image on the graphics screen special routines must be used to produce words which are properly formatted for the display buffer. One routine is used for control words (e.g. intensity), another is used for draws and moves, and a third is used for text. This information is put into an array as an image is being built up. When a group of instructions is ready to be transferred to the display buffer, a routine is called to carry out the transfer. Another routine is supplied to detect pushbutton depressions and light pen hits, and read the dials.

3.2.3 Effect of Hardware and Software on the System

As far as the acutal program is concerned, the hardware and supplied software have in some ways dicatated the design of certain internal structures. For example, the method of returning the location of a light pen hit suggests the necessity of structures like the "shadow buffer" and the "scratch buffer" (see below). Also, in order to erase an image from the screen, the system must know the location, in the display buffer, of the instructions that are producing that image. Thus, for each item that might be erased, the system must keep track of a starting and ending location. This has meant that such values are kept for vertices, edges, labels, and various other displayed objects.

The only major problem with the graphics terminal, at least

as far as this implementation is concerned, is that a character generator is not included. This means that all characters must be generated by software as a series of short lines. Characters generated in this way tend to require a relatively large number of display instructions. Since the system menu consists in large part of characters, the display buffer contains a significant number of display instructions even before any graphs are drawn. This, combined with the fact that the Adage uses a refreshed screen, causes the screen to begin to flicker when large graphs are drawn. In fact, the size limit of graphs seems to depend more on this factor than on other factors such as memory availability; the flicker becomes too objectionable long before the display buffer becomes full.

For users who can stand the flicker, the size limit for graphs is very reasonable. The largest graph actually drawn to date contained 200 vertices and 100 edges. An approximate upper bound on the size of undirected and unlabeled graphs is implied by the inequality

$$(3+s)V + (3+e)E \leq 3500,$$

where $\underline{V}$ is the number of vertices, $\underline{E}$ is the number of edges, $\underline{s}$ is the average number of sides to a vertex, and $\underline{e}$ is the average number of edge sections in an edge. So, for example, if vertices were drawn as triangles (s=3) and all edges were drawn as one straight line between vertices (e=1), then it would be possible to draw a graph with 300 vertices and 425 edges!

This limit is reduced considerably if labels are included, since, as mentioned above, characters tend to require a large number of display instructions. A further reduction is encountered if dircted edges (with arrows) are considered. For example, if all vertices and edges in a graph are labeled (and displayed), and there are an average of two characters per label, and we assume that vertices are triangles and edges are directed but not free-hand, then a graph with 60 vertices and 74 edges could be drawn.

System response time is usually excellent, and is even very good when the operating system is busy. This allows graphs to be drawn and manipulated very quickly, especially by users familiar with the system.

The GGRAM system is relatively expensive to use, but not excessively expensive. During a typical one hour run a user can expect that about 32 per cent of the cost will be for connect time, 2 per cent for signing on to the operating system and loading the system, and 66 per cent for CPU time and memory usage. The actual cost of preparing and plotting figure 7, for example, was 18 computer dollars. This can be compared with a cost of 12 computer dollars for compiling the system itself.

3.3 DATA STRUCTURES

There are three important data structures within the system. They are the internal representation of graphs, the shadow buffer, and the scratch buffer.

3.3.1 Internal Representation of Graphs

The most important structures, of course, are those structures used for storing graphs. There are five types of record classes associated with graphs. Figures 1 through 5 show the contents of each record type and what the fields are used for. At the top level in the structure is the GRAPH record, which contains global information about the graph such as the title, the number of vertices, and so on. It also contains a pointer to a list of VERTEX records.

Each VERTEX record contains information local to the vertex as well as three pointers to incident edges: the undirected edges, the indirected edges, and the outdirected edges (c.f. [12]). These pointers, however, do not point directly to EDGE records, but instead point to EDGEDES (edge descriptor) records. Since each edge is connected to two (not necessarily distinct) vertices, it would be wasteful to have two copies of all the information for each edge. Having two copies would also complicate changing an edge. Therefore, a VERTEX points to a list of EDGEDES records, and each EDGEDES record in turn points to its associated EDGE. Each EDGEDES record also has a pointer

INTEGER GNUM	graph number
STRING(40) GTITLE	graph title
INTEGER NVERTICES	no. of vertices
INTEGER NEDGES	no. of edges
INTEGER NDIREDGES	no. of directed edges
INTEGER NLOOPS	no. of loops
INTEGER NMULTEDGES	no. of multiple edges
LOGICAL CHANGED	changed? flag
INTEGER LABDISP	label display indicator (label mode)
INTEGER COUNTER	count of components
REFERENCE(VERTEX) VLIST	pointer to vertex list
REFERENCE(GRAPH) NEXTGRAPH	pointer to next graph in graph list
REFERENCE(GRAPH) LASTGRAPH	pointer to previous graph in graph list
INTEGER TSTART	start of title in buffer
INTEGER TEND	end of title in buffer

Figure 1 - The Fields in a GRAPH Record

INTEGER VNUM	vertex number
STRING(8) VLABEL	vertex label
INTEGER DEGREE	degree
REFERENCE(EDGEDES) UNEDGES	pointer to list of undirected edges
INTEGER INDEGREE	indegree
REFERENCE(EDGEDES) INEDGES	pointer to list of indirected edges
INTEGER OUTDEGREE	outdegree
REFERENCE(EDGEDES) OUTEDGES	pointer to list of outdirected edges
REAL VINTENSITY	intensity
LOGICAL VSOLID	solid (or dashed)?
INTEGER VSHAPE	shape (no. of sides)
REAL VSIZE	size ("radius")
REAL XVER	x-coordinate of vertex
REAL YVER	y-coordinate of vertex
REAL XVLAB	relative x-coordinate of label
REAL YVLAB	relative y-coordinate of label
REFERENCE(VERTEX) NEXTVERTEX	pointer to next vertex in vertex list
REFERENCE(VERTEX,GRAPH) LASTVERTEX	pointer to previous vertex in vertex list (points back to GRAPH if first VERTEX in list)
REFERENCE(VERTEX) DUPTEMP	"scratch" field used by certain commands
INTEGER VSTART	start of vertex in buffer
INTEGER VEND	end of vertex in buffer
INTEGER VLSTART	start of label in buffer
INTEGER VLEND	end of label in buffer

Figure 2 - The Fields in a VERTEX Record

INTEGER ENUM	edge number
STRING(8) ELABEL	edge label
INTEGER ETYPE	edge type
REAL EINTENSITY	intensity
LOGICAL ESOLID	solid (or dashed)?
INTEGER NCONSPTS	no. of construction points
REFERENCE(CONSPT) CONSLIST	pointer to list of construction points
REAL XELAB	relative x-coordinate of label
REAL YELAB	relative y-coordinate of label
REFERENCE(EDGEDES) POSDES	pointer to positive edge descriptor
REFERENCE(EDGEDES) NEGDES	pointer to negative edge descriptor
INTEGER ESTART	start of edge in buffer
INTEGER EEND	end of edge in buffer
INTEGER ELSTART	start of label in buffer
INTEGER ELEND	end of label in buffer

Figure 3 - The Fields in an EDGE Record

REFERENCE(EDGE) EPTR	pointer to edge
REFERENCE(VERTEX) VPTR	pointer to other adjacent vertex
REFERENCE(EDGEDES) NEXTDES	pointer to next edge descriptor in the list
REFERENCE(EDGEDES) LASTDES	pointer to previous edge descriptor in the list

Figure 4 - The Fields in an EDGEDES Record

REAL XCONS	x-coordinate of construction point
REAL YCONS	y-coordinate of construction point
REFERENCE(CONSPT) NEXTCONSPT	pointer to next construction point in the list

Figure 5 - The Fields in a CONSPT Record

to the second incident vertex.

The EDGE records contain information local to edges, including the edge label, the edge type (undirected or directed), and so on. If the edge is a free-hand edge, then the EDGE record also contains a pointer to a list of CONSPT (construction point) records. Each CONSPT record in this list simply contains the coordinates of one of the intermediate points in the free-hand edge.

Figure 6 shows the complete link structure for a small graph.

All GRAPHs that have been defined to the system are kept on the "graph list". There are two ways to put a GRAPH on the graph list: by using RESTORE or by simply starting to draw in a blank region. The system keeps track of which graphs have been changed (a new graph is considered to be changed), since these are the graphs a user will likely consider saving at the end of a run.

Graphs in the system are stored as if they had been drawn in a unit square. When they are displayed they are scaled according to the size of the appropriate region. The internal coordinates of vertex labels are relative to the centre of the vertex; the internal coordinates of edge labels are relative to

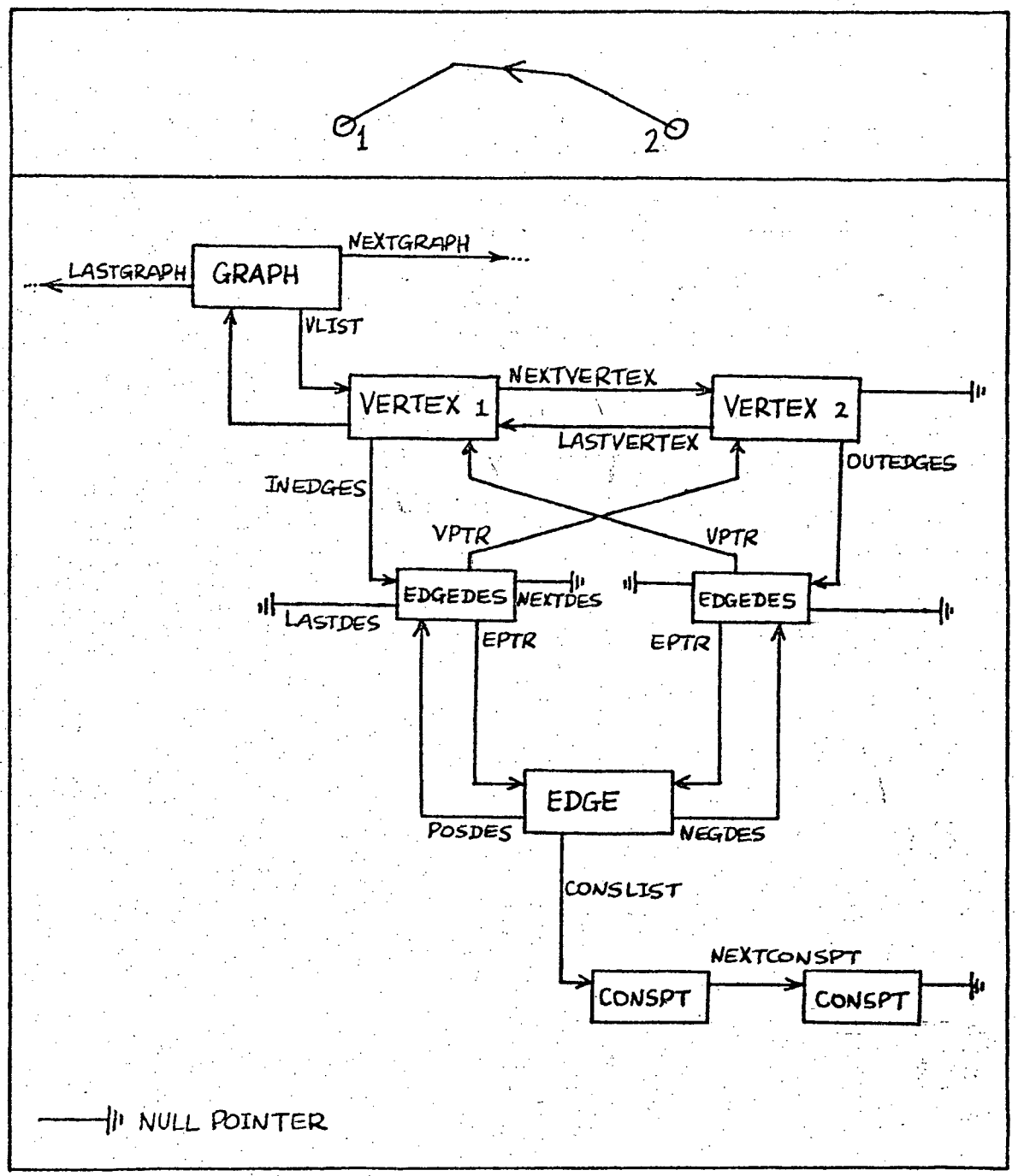

Figure 6 - The Link Structure of a Small Graph

the middle[2] of the edge.

3.3.2 Shadow Buffer

The image on the graphics terminal's screen is produced from a 6000 word display buffer (or display program) which is contained in the graphics computer connected to the graphics terminal. This display program consists of instructions for moving the light beam, enabling and disabling dashing, enabling and disabling the light pen, and so forth. Thus, to produce a particular image on the graphics screen, the appropriate instructions must be inserted into this buffer.

When a light pen "hit" is detected by the graphics terminal, the value that is returned to the system is the index in the display buffer of the vector that was being drawn when the hit occurred. So, in order to be able to tell what has been hit, the system must somehow keep track of what is in each buffer location. Furthermore, it is desirable to have a convenient method of finding holes in the buffer, that is, sequences of elements which are not being used. The shadow buffer serves both these purposes.

The shadow buffer is a series of three parallel arrays, each of length 6001 (logical index 0 to 6000). They are implemented in FORTRAN since ALGOLW will not support arrays this

---

[2]refer to the LABEL command description in Appendix I.

long. The first array holds the type of the entity, the second holds the number of the entity, and the third holds a pointer (where one exists) to the entity. The following table shows the contents of the arrays for each entity type:

ENTITY	TYPE	NUMBER	POINTER
hole	≤ 0	(see below)	(see below)
(button)[3]	1	button no.	NULL
fixed menu item	2	item no.	NULL
variable menu item	3	item no.	NULL
CANCEL	4	0	NULL
region mark	5	region no.	NULL
vertex	6	region no.	pointer to VERTEX
edge	7	region no.	pointer to EDGE
vertex label	8	region no.	pointer to VERTEX
edge label	9	region no.	pointer to EDGE
message	10	0	NULL
misc.	999	0	NULL

So, for example, if a light pen hit occurs at location 500 in the display buffer, and if location 500 in the type array contains 6 and location 500 in the number array contains 2, then we know that a vertex displyed in region two is the entity that was hit. Moreover, location 500 in the pointer array contains a pointer to that VERTEX.

Holes in the buffer are treated in a special way. The first element of a hole holds the following information: the type is the negative of the length of the hole; the number is

---

[3]logical element 0 of the buffer is used to communicate information about buttons.

the index of the start of the next hole; and the pointer is the index of the start of the previous hole. All other elements of the hole are set to zero.

Thus, the holes are connected together by way of a doubly linked list. Initially the list simply consists of one entry which is 6000 elements long. When a hole of a particular length is needed, the list is scanned and the first hole which is long enough is used. If the hole is longer than is needed, a new shorter hole is created.

As holes are used and created in this way, the buffer eventually becomes fragmented. This fragmentation is minimized, however, in the following way: when a new hole is created (by the deletion of an edge, for example), the system checks to see if a hole already exists either in front of or following the new hole. If either or both of these holes are present, all of the holes are combined into one large hole.

If the system cannot find a hole of the appropriate size when it needs one, then it simply terminates execution. The system is designed in such a way that the buffer can be compactified, and in fact the dummy procedure COMPACTIFY exists and is called in the system. As it turns out, though, a compactification has never been needed, and so COMPACTIFY has not been implemented.

3.3.3 Scratch Buffer

The scratch buffer is used to temporarily hold data that will eventually be sent to the display and shadow buffers. The scratch buffer is a set of four parallel arrays of length 500. Like the shadow buffer, it is implemented in FORTRAN.

As an image to be displayed is being constructed, information is put into the scratch buffer. One array holds the words that will be sent to the display buffer; the other three arrays hold the type, number, and pointer that will be sent to the shadow buffer. For example, when a vertex is to be displayed the following instuctions are put into the first array of the scratch buffer: set intensity; enable light pen; enable dashed lines if a dashed vertex; move to the start of the vertex; draw each side of the vertex; and disable dashed lines if a dashed vertex. The appropriate type, number, and pointer are put in the corresponding elements of the other three arrays.

At this point the image is ready to be transferred to the display buffer, so an appropriate hole is found and the transfer is made. At the same time the contents of the other arrays are transferred to the shadow buffer.

3.4 OVERVIEW OF THE PROGRAM

The structure of the program is quite straightforward. At the top level is a driver routine which simply initializes the global variables, sets up the screen, and then goes into a loop which first determines the control mode (light pen or keyboard), and then calls the appropriate routine. Looping continues until the system is stopped.

At the present time, only light pen mode is implemented. The routine which handles this mode reads the graphics screen and then waits until a command is specified. When one is specified, the appropriate routine is called. It should be noted that, whenever the graphics screen is read, care must be taken not to accept invalid or unwanted input. Therefore, the light pen mode control routine will not do anything unless a command is specified.

Aside from a number of utility routines, the remainder of the program is made up primarily of routines for the individual commands. There are also a group of routines associated with displaying and constructing graphs. These include routines for scaling graphs for display in a particular region, and routines for adding vertices and edges to the internal graph structures, among others.

The program listing has not been included with this thesis because it is too long. It is available on request from the

Department of Computer Science of The University of British Columbia.

CHAPTER IV - IMPROVEMENTS AND EXTENSIONS

The primary goal throughout the development of the GGRAM system has been to provide a system for quickly and conveniently drawing and manipulating graphs. Although this goal has been achieved, there are still a number of refinements which could be made to improve the system's performance. Some of these improvements are mentioned below. More importantly, though, there are a number of ways the system could be extended in order to increase its usefulness. Some of these extensions are also discussed below.

4.1 IMPROVING THE EXISTING SYSTEM

An annoying characteristic of the system is that an objectionable flicker develops on the screen as graphs are drawn. As mentioned in Chapter III, this is made worse than it might be because the menu needs a large number of display instructions. One way to delay the onset of flickering is to reduce the number of characters in the menu. This could be done by simply abbreviating commands. For example, DUPLICATE might become DUP. This would have an additional advantage. If names in the menu were shortened, then the width of the menu area

could be decreased, resulting in a larger drawing area.

Abbreviated commands, of course, would compound the problems of learning to use the system. To overcome this, it would be fairly easy to allow two forms of the menu, one abbreviated and one not. If the abbreviated menu were the default, the system could allow the user to change to the full menu using, say, the SET command.

Another slightly annoying system characteristic is that when REPEAT is on, an activated menu item must be explicitly turned off before a new item is activated. It would be nicer if the act of choosing a new command implicitly released the repeat status of any other command which happened to be activated. This would speed up the graph drawing process slightly.

4.2 EXTENSIONS

There are two distinct problems which a system for "graph theory" might be concerned with. One is graph manipulation, and this is the problem at which the GGRAM system is aimed. An equally important problem is the automatic display of arbitrary abstractly defined graphs. An abstractly defined graph is one where logical connections of vertices are defined, but layout is not. For instance, an adjacency matrix represents an abstractly defined graph. As mentioned in Chapter I, however, the most natural way to present a graph to a person is by way of a

diagram. It would be convenient, therefore, to be able to produce a suitable diagram from an abstract definition of a graph.

Of course, the question immediately arises as to just what the properties of a "suitable" graph are. One common criterion used is that such a graph should have a minimum of edge-crossings. But the layout of vertices in itself can be important. For example, many bipartite graphs are best displayed as two rows of vertices, not necessarily with edge-crossings minimized. The implication, then, is that the automatic display of of arbitrary graphs is a difficult problem.

But how could a facility for automatically displaying graphs be embedded in, or combined with, a graph manipulation system? For certain classes of graphs, such as cycles or complete graphs, a simple generation algorithm could be devised and simply added as a new system command. The only input needed for such commands would be the number of vertices desired. For more complicated graphs, the system would first have to have a facility for entering the abstract graphs. This would be relatively easy to provide. Also, since the most suitable diagramatic representation of a graph depends on its properties, there must be a way to determine the properties of a given (abstract) graph. This would involve providing property recognizers for a large number of properties, but such recognizers would be easy to incorporate into a graph

manipulation system.

It is an open question as to where to proceed from this point. Perhaps the most straightforward way, but not necessarily the best, would be to implement a number of algorithms for displaying certain specific classes of graphs. An abstract graph to be displayed could then be passed to each in turn. Each would test to see if the graph belonged to its class and, if so, would process the graph. Of course, something would have to be done if the graph did not fit into any of the classes for which a display algorithm existed. An unsophisticated system might simply reject the graph.

Whatever the approach, though, it does not seem likely that many difficulties would be encountered incorporating automatic graph display facilities into a graph manipulation system, assuming the graph-theoretic problems could be overcome. After all, the automatic display routines would simply be programs operating on some sort of an internal representation, likely a matrix.

One important reason for combining graph manipulation and graph display facilities into one system is that the two would complement each other. For example, a user might want to draw a graph which is "almost" a complete twenty vertex graph. In a combined system, he could automatically display a complete twenty vertex graph and then use the manipulation facilities to change the graph slightly. On the other hand, a user might draw

a rough version of a graph, and then use the graph display facilities to generate a neater version.

A number of possible extensions concern the addition of user-written routines to a graph manipulation system. The ability to add new routines is crucial to a flexible system. The GGRAM system, as well as the systems of Wolfberg [20] and Maguire [12], all provide this facility at a very low level. This simply involves allowing a user to link a new routine into the system by re-compiling some or all of the system. No debugging facilities, except perhaps a command for dumping graph structures, are provided.

An important extension to a graph manipulation system would be a set of comprehensive facilities for debugging user-written routines. Many facilities of this type can be envisioned. One is that a user might be provided with a means of scanning internal structures (like graphs) in a way more flexible than a simple dump. Of course, with a graphics system a displayed image can be a good indicator for telling whether or not a particular routine is working. Often, however, the exact cause of trouble cannot be pinpointed by looking at the image.

Another extremely useful debugging facility would be the provision of an interpreter for the language in which the system is implemented. With such a facility, a user could interpret (rather than compile and execute) new routines, and gain the associated advantages. These advantages include the prospect of

better run-time diagnostics, and the ability to dynamically alter routines without having to re-compile. Of course, the addition of such a facility would be a major project in its own right.

One way to make it more convenient to add user-written routines is to make the internal design of the system as transparent as possible to the user. This can be done by providing a set of appropriate (well chosen) system routines to perform graph-theoretic and utility functions. Another way is to extend the implementation language to include graph-theoretic operations and structures. This is what Wolfberg [20] has done in his graphics system. Other language extensions, without graphics, have also been developed [11, 18, 16, 6].

The extensions which follow are more specific in nature than the previous ones. These are modifications which could be made, without too much trouble, to increase system performance.

At the moment, only one of the sixteen pushbuttons at the graphics terminal is used. It would be convenient if the remaining buttons could be defined to be equivalent to an arbitrary menu item. In other words, pushing such a button would have the same effect as hitting the associated menu item with the light pen. The "add vertex" and "add edge" commands are prime candidates for such a feature.

Another improvement could be made by providing a routine

for the input of characters using the light pen rather than the keyboard. Thus, when a title or label is to be entered, a "keyboard" could be displayed on the screen and the light pen used to choose particular characters. This would be particularly useful with the existing hardware because the keyboard is physically situated to the right of the user. This means that right-handed users must put down the light pen and reach over to enter a label or title. Naturally this is a nuisance if it is necessary to repeat it more than a few times.

It would also be desirable if a routine were written to smooth free-hand edges. This idea could be extended by providing an internal switch which could be turned on or off. When on, all free-hand edges would automatically be smoothed when drawn.

One of the interesting features of the system developed by Maguire [12] is the ability to associate property lists with vertices and edges (see Chapter I). The number and types of these properties can be changed dynamically as the system runs. This feature can be partially simulated with labels, but such an approach essentially limits a component to one property. Numerical labels on arcs could be converted to numbers to be used as capacities in networks, for example. But the ability to associate a number of properties with a vertex or an edge would significantly improve the system for certain applications.

Finally, it is interesting to consider whether or not a

facility analogous to Wolfberg's "windows" [20] (see also Chapter I) would be worthwhile implementing. It appears that it would be worthwhile for at least two reasons. First, the ability to look at a magnified portion of a graph can be very useful. It can, for example, let users draw large, detailed graphs which would be difficult otherwise. This could be done by drawing a number of highly magnified subgraphs which would then be combined to form the total graph. Second, windowing could serve to reduce the effect of screen flicker by allowing users to display only the portion of a graph which is in use. This, again, would be useful for large graphs.

4.3 SUMMARY

In this study the design and implementation of an interactive graphics system for graph manipulation have been examined. The principal aim of the system is to facilitate the drawing and manipulation of graphs. To this end, a comprehensive graph drawing capability is provided. This includes the basic operations of adding, deleting, labeling, and changing both vertices and edges. Also included are a group of graph manipulation operations. These operations, among other things, allow a user to subdivide edges, associate vertices, reverse the direction of arcs, move vertices about the screen, or even move whole graphs about the screen.

A facility is provided whereby the screen can be divided into as many as four regions, thus allowing users to display more than one graph at a time. Graphs can be saved on disk and later restored. The image on the graphics screen can be easily plotted to obtain a hard copy of graphs.

A number of routines which perform graph-theoretic operations have been implemented. Among these are a routine to find the minimum and maximum degrees of a graph and a routine to find the blocks, cutnodes, and bridges of a graph. Moreover, the system is designed to allow users to add their own routines.

BIBLIOGRAPHY

[1] Bauer, H.R.; et al., ALGOLW Programming Manual, Univ. of Newcastle Upon Tyne Computing Laboratory, 1970.

[2] Choit, M.D. (ed.), The MTS Commands Manual, Univ. of British Columbia Computing Centre, 1970.

[3] Coulthard, W.J., Operator's Manual for the Adage Graphics Terminal, Univ. of British Columbia Computing Centre, 1970.

[4] Coulthard, W.J., A Simple Interactive Graphics Package, Univ. of British Columbia Computing Centre, 1971.

[5] Coulthard, W.J.; DeKleer, J., Basic Communications Package for the Adage Graphics Terminal, Univ. of British Columbia Computing Centre, 1971.

[6] Crespi-Reghizzi, S.; Morpurgo, R., A language for treating graphs, Comm. A.C.M. 13, 5 (May 1970), 319-323.

[7] Di Giulio, H.A.; Tuan, P.L., A graph manipulator for on-line network picture processing, Proc. AFIPS 1969 FJCC, 387-398.

[8] Dixon, W.R., ALGOLW Compilers, Univ. of British Columbia Computing Centre, 1971.

[9] Harary, F., Graph Theory, Addison-Wesley, Reading, Massachusetts, 1969.

[10] Higginson, M., Extensions to ALGOLW, Tech. Report TR73-19, Dept. of Computing Science, Univ. of Alberta, 1973.

[11] King, C.A., A graph-theoretic programming language, in Graph Theory and Computing, by R.C. Read (ed.), Academic Press, New York, 1972.

[12] Maguire, R.B., The GSYM System: An Interactive System for Displaying Linear Graphs, Tech. Report CS-73-24, Univ. of Waterloo, 1973.

[13] Miller, R.B., Response time in man-computer conversational transactions, Proc. AFIPS 1968 FJCC, 267-277.

[14] Newman, W.M.; Sproull, R.F., Principles of Interactive Computer Graphics, McGraw-Hill, New York, 1973.

[15] Osborne, E.J., Using MTS from the IBM 3270 Display Station, Univ. of British Columbia Computing Centre, 1973.

[16] Pratt, T.W.; Friedman, D.P., A language extension for graph processing and its formal semantics, Comm. A.C.M. 14, 7 (July 1971), 460-467.

[17] Read, R.C., Teaching graph theory to a computer, in Recent Progress in Combinatorics, by W.T. Tutte (ed.), Academic Press, New York, 1969.

[18] Rheinboldt, W.C.; Basili, V.R.; Mesztenyi, C.K., On a programming language for graph algorithms, BIT 12, (1972), 220-241.

[19] Smith, L.B., A survey of interactive graphical systems for mathematics, Computing Surveys 2, 4 (Dec. 1970), 261-301.

[20] Wolfberg, M.S., An Interactive Graph Theory System, Doctoral Disseration, Univ. of Pennsylvania, 1969.

APPENDIX I - USER'S GUIDE

INTRODUCTION

This Guide presupposes familiarity with MTS [2], the 3270 terminal [15], the Adage graphics terminal [3], and the GRAPH supervisor program for the Adage graphics terminal [4].

The system can be run by issuing the following MTS command:

$SOURCE HUMP:GRAPH

The message READY, which appears at the top of the menu, begins blinking when the system is ready. Users sould note that the GRAPH supervisor program may have to be loaded into the graphics computer before running the system. Refer to reference [3] for details.

Users who wish to experiment with the system without first reading all of the material that follows in this Guide are advised to read the introductory remarks and the following command descriptions: READY, STOP, RUB OUT, CANCEL, ADD, DELETE, and REPEAT. With this minimal amount of information a user should be able to draw graphs, erase graphs, and stop when finished.

SCREEN LAYOUT

Figure 7 shows what the Adage screen looks like when the system is running, including some sample graphs. The largest portion of the screen is the drawing area . This is the area where graphs are actually drawn. In the figure, the drawing area has been divided into three regions by using the REGIONS command. In its initial state, the drawing area is not divided, that is, it is simply treated as one, large region. The size of the regions can be varied by changing the region intersection point. This is one of the functions of the SET command.

One of the regions on the screen is considered to be the current region or current area. When there is only one region, then that region, of course, is the current area. When the drawing area is divided into two or more regions, the current area is surrounded by bold face lines. The current area is used by certain functions when a particular region must be specified. For example, when RUB OUT is activated, the graph in the current area is erased.

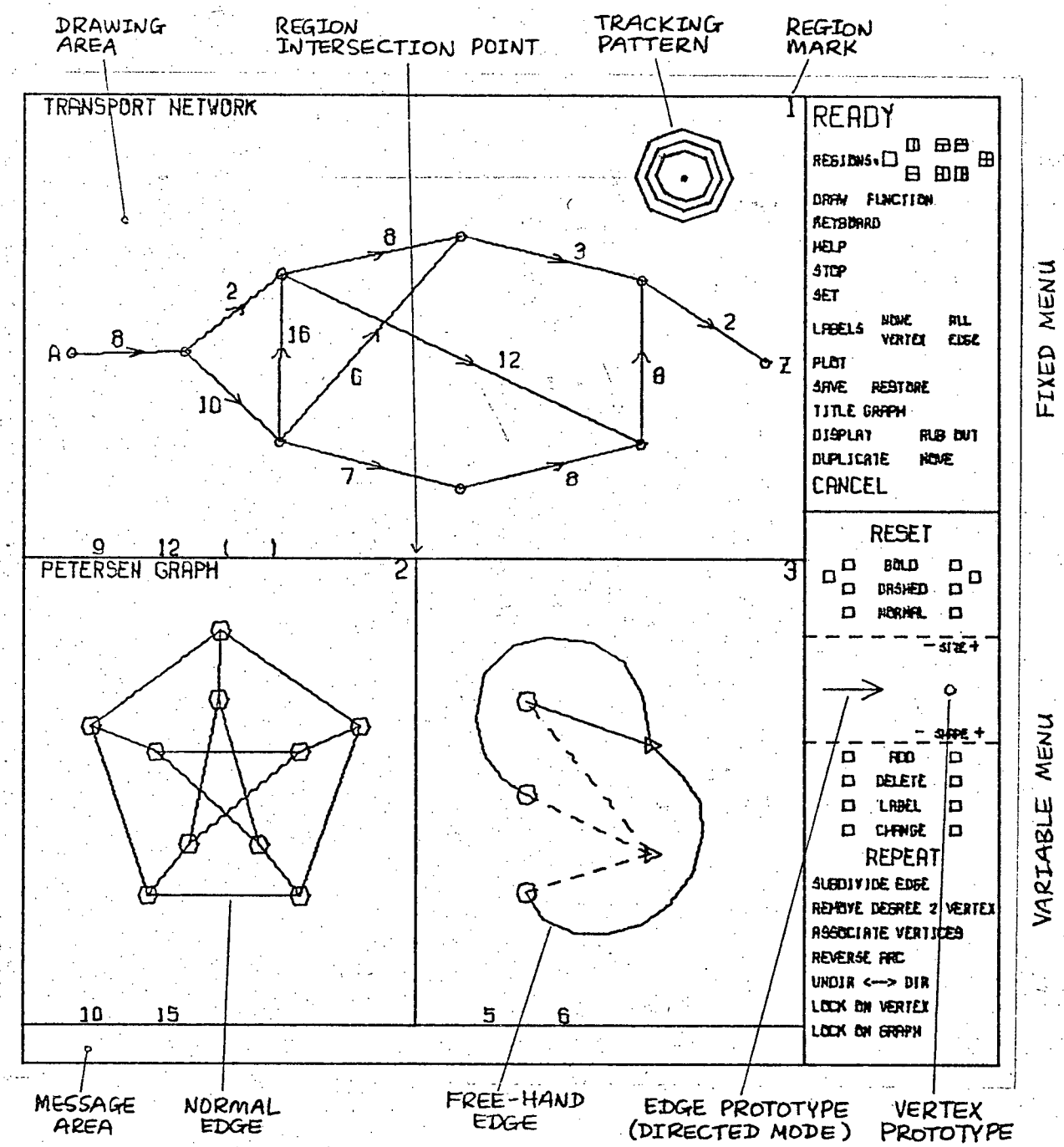

Figure 7 - The Screen When the System is Running

When there are two or more regions, a region number or region mark appears in the top right corner of each region. The region mark is used both to change the current area and to specify a particular region for certain commands. To change the current area, a user simply "hits", with the light pen, the region mark of the region which is desired.

The right hand side of the screen is reserved for the menu. The upper part of the menu area contains the fixed menu. This menu does not change as the system operates. It includes commands that are of general utility, such as PLOT (for plotting the screen) and SAVE and RESTORE (for saving graphs on disk and later restoring them). The lower part of the menu area is for the variable menu. The variable menu can be changed so that it contains either the draw menu or the function menu.

As its name implies, the draw menu includes commands for drawing and altering graphs, including such things as adding and deleting both vertices and edges. The function menu contains commands for invoking graph-theoretic algorithms. These, for example, include a command for determining the maximum and minimum degrees of a graph, and a command for finding the blocks, cutnodes, and bridges of a graph. The variable menu can be changed using the DRAW or FUNCTION commands which appear in the fixed menu. Menu items that are in effect or in use are displayed in bold face.

At the bottom of the screen is a narrow area called the message area. This area is used for displaying instructions, warnings, error messages, and so on.

When the system is started, the drawing area is empty except for the tracking pattern, which appears in the centre of the screen. The tracking pattern is a set of three concentric octagons with a dot in the centre. This pattern can be moved around the screen using the light pen. When the light pen touches one of the lines on the perimeter of one of the octagons, the pattern will move in that direction. The system can determine where the centre of the pattern is at any given time, and thus the tracking pattern is a convenient means to indicate such things as the locations of vertices.

As graphs are drawn, a user will notice numbers appearing near the lower left corner of the regions. The leftmost number is the number of vertices. The second number is the number of edges. For directed graphs this number includes a count of two for each undirected edge in the graph. If there are no edges, this number does not appear. A third number, enclosed in parentheses, appears for directed graphs. This number is a count of the number of undirected edges. It is blank if there are no undirected edges.

I/O METHODS

There are two devices with which a user communicates with the system: the Adage graphics terminal itself, and the IBM 3270 terminal which is set up as an interface between the graphics terminal and the operating system. The primary output device is, of course, the screen of the graphics terminal. A small amount of information also appears from time to time on the 3270 screen.

Most of the input to the system is supplied by way of the light pen. Some input, however, is supplied with pushbuttons and dials connected to the Adage, and the keyboard connected to the 3270. Since the system cannot read from both devices at the same time, a user must be able to tell where the system is expecting input from. When the system is expecting input from the light pen (or the pushbuttons) the READY at the top of the menu starts blinking. On the other hand, when input is expected from the 3270 keyboard, a blinking message is displayed on the Adage screen and a message is also displayed on the 3270 screen.

It is desirable to be able to issue a "cancel" indication at both devices. This is useful, for example, when a user changes his mind after initiating an operation. The cancel indication for input from the Adage is the menu item CANCEL, at the bottom of the fixed menu. To issue a cancel indication at the 3270, a user must enter the current contents of the parameter CANCELSTRING. Initially this is "CANCEL", but it can be changed with the SET command.

PROTOTYPES

Since the physical appearance of vertices and edges can be varied, a user should be able to see easily just what their current characteristics are. For this reason, a <u>vertex prototype</u> and an <u>edge prototype</u> are displayed in the draw menu (see figure 7). These prototypes reflect the current settings of the intensity, dashing, and size parameters associated with vertices and/or edges. Vertices and edges drawn on the screen will look like the prototypes. As explained below in the command descriptions, vertices and edges can be normal intensity, bold face, dashed, or dashed and bold. Furthermore, the size and shape of vertices can be varied.

The edge prototype also serves to indicate whether or not the system is in <u>directed mode</u>. In directed mode, all edges that are drawn are considered to be directed edges (or arcs). The edge prototype is a bistable switch. If hit with the light pen when in undirected mode, the prototype will change into a directed edge and the system will go into directed mode; if hit

when in directed mode, the prototype will change back to an undirected edge and the system will return to undirected mode.

DIALS

It is desirable to allow the user control over the relative intensity between normal and bold face. This is accomplished using the variable control dials. Dial A (top left) controls normal intensity; dial D (top right) controls bold face. So, if dial A is "turned up", all vertices and edges of normal intensity will increase in intensity. Dial D does the same for bold face vertices and edges. If vertices or edges do not appear on the screen when expected, it may be because the dials are set too "low". It is advised that they be turned fully clockwise initially, and then adjusted as desired.

DRAWING GRAPHS

In order to get a flavour as to how the system is used, the following description is included to indicate how a graph is drawn. In particular, the steps necessary to draw a graph like the one in region 3 of figure 7 are described. Of course, the "feel" of the system can only be experienced by actually using it [20, p. 172].

The first step is to draw the vertices. Since a number of vertices are to be drawn, REPEAT is activated by hitting it with the light pen (at which time it changes to bold face). When REPEAT is on, certain commands (including "add vertex") repeat until explicitly turned off. Now, by activating the small square to the right of ADD (underneath the vertex prototype), the system is put into "add vertex mode". At this point, to add vertices the tracking pattern is simply moved to positions where vertices are desired and the draw button[1] pressed. The number of sides in a vertex can be decreased by hitting the "-" to the left of SHAPE, and can be increased by hitting the "+" to the right.

The second step is to draw the edges. To do this, "add vertex" must first be turned off by hitting it again with the light pen. Note that REPEAT remains on. "Add edge" can then be activated using the square to the left of ADD (underneath the edge prototype). To draw the normal edges, the procedure is to first hit a "from" vertex (a blinking cross will be displayed at the centre of the vertex), and then hit a "to" vertex. When both vertices have been specified, the edge is drawn. To draw

---

[1]The draw button is initially button 1, at the top left corner.

the free-hand edge, a "from" vertex is first specified in the normal way. Each edge section can then be drawn by moving the tracking pattern in turn to each intermediate point and pressing the draw button. The edge is completed by specifying a "to" vertex. Dashed edges can be drawn after first hitting the small square to the left of DASHED, above the edge prototype.

A user who is familiar with the system could draw this graph in about one minute.

COMMANDS IN THE FIXED MENU

- READY

Purpose:

To indicate that the system is ready to accept input (from the Adage).

Description:

When READY is blinking, the system is waiting for input from the Adage. This means that a light pen hit or a button depression will be detected, but that keyboard input will not be detected.

READY will appear with constant intensity when the system is not waiting for Adage input.

- REGIONS

Purpose:

To change how the drawing area of the screen is divided into regions.

Description:

The drawing area can be divided into one, two, three, or four regions by choosing one of the patterns that appear on the menu immediately to the right of REGIONS. For example, the screen can be divided into two regions either vertically or horizontally. When there are two or more regions, the region number (or region mark) is displayed in the top right corner of the region, and the current area is outlined in bold face.

To change how the screen is divided, a user simply chooses a new pattern with the light pen. When this is done, all graphs being displayed are rubbed out, and new region lines are drawn. The system then re-displays, on region one, the graph (if any) which was in the current area when the new pattern was chosen.

The coordinates of the point of intersection of the horizontal and vertical dividing lines are system parameters which can be changed using the SET command. The default location is the centre of the drawing area.

- DRAW-FUNCTION

Purpose:

To put the system into either draw or function menu mode, that is, to set the variable menu.

Description:

When the system is in draw mode, the draw menu is displayed in the variable menu area; similarly, when the system is in function mode, the function menu is displayed. To change to one of these modes a user simply activates either DRAW or FUNCTION.

The draw menu contains facilities for drawing and altering graphs. For example, "add vertex", "delete edge", and "reverse arc" are included in this menu.

The function menu contains facilities for applying graph-theoretic algorithms to a given graph. These operations include such things as permuting the vertex labels of a graph and finding the minimum and maximum degrees of a graph.

Figure 8 shows the menu in both draw and function mode.

- KEYBOARD[2]

Purpose:

To put the system into keyboard mode.

Description:

In keyboard mode, a user communicates with the system using the keyboard rather than the light pen. All regular facilities of the system are available in keyboard mode through the use of typed commands.

- HELP[3]

Purpose:

To supply a user with information about the system.

Description:

When HELP is activated, the user is asked a series of questions to determine what information is desired. So, for example, if a user wants to know how the DUPLICATE command works, HELP will supply the necessary information.

---

[2]KEYBOARD has not been implemented.
[3]HELP has not been implemented.

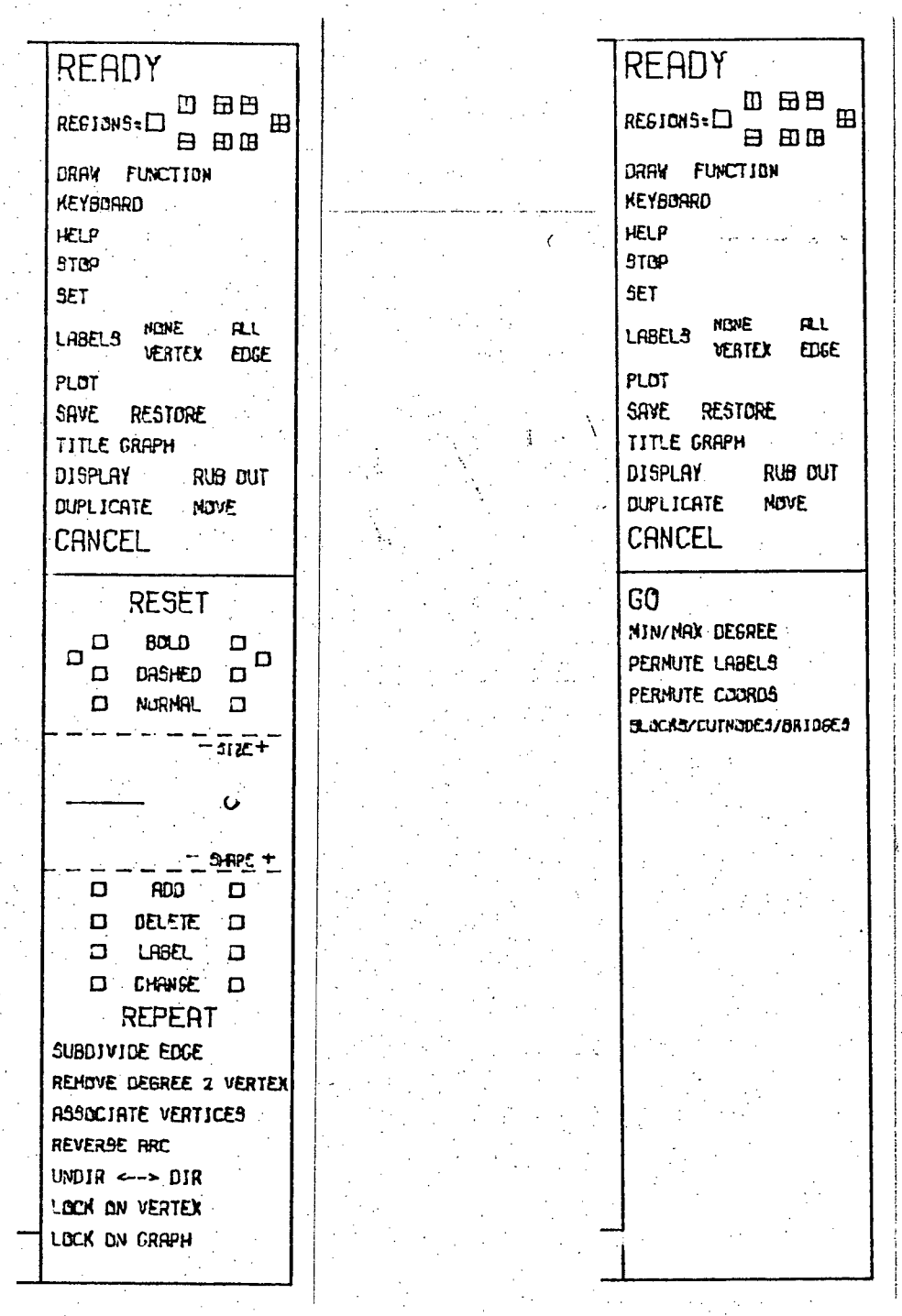

Figure 8 - The DRAW and FUNCTICN Menus

- STOP

Purpose:

To stop the system.

Description:

When STOP is activated, the system first asks for confirmation. If confirmation is received, the system does some "bookkeeping" associated with plots and then execution is terminated.

If the stop is not confirmed, the system will continue as if nothing happened.

- SET

Purpose:

To change various system parameters.

Description:

When SET is activated, the names of a number of system parameters are displayed on the screen. The light pen is then used to choose the parameter to be changed.

With one exception, the system then prints the current value of the parameter on the 3270 screen, and waits for the user to input a new value at the keyboard. If at this point the user enters "DEFAULT" or "#", the parameter will be set to its default value.

The following list contains those parameters which can be changed using SET. Unless otherwise indicated or implied, the default values are expressed in "normalized screen units", that is, in relation to a unit (one inch) square. These values will be scaled appropriately when a graph is displayed on a particular region.

XSCREEN/YSCREEN (default: centre of drawing area)

These are the coordinates of the point of intersection of the vertical and horizontal dividing lines. The setting of these parameters is different from the setting of the other parameters.

For these parameters, the new values are not typed at the keyboard, but instead the tracking pattern is used to locate the new intersection point. To accomplish this, the word "PATTERN" (or abbreviation "P") is used when the system asks for a new value.

Thus, to set XSCREEN/YSCREEN a user inputs "DEFAULT" (or "#") if the default location is desired, or "PATTERN" (or "P") if the tracking pattern location is desired.

When XSCREEN and YSCREEN are changed, all graphs on the screen are rubbed out, and the graph which was in the current area is re-displayed in region one.

ARRLEN (default: 0.02)

This parameter helps determine the size of the arrows used to show the orientation of directed edges (arcs). ARRLEN (arrow length) is the length of the projection of the arrow's side onto the associated arc.

ARRWID (default: 0.01)

This parameter helps determine the size of the arrows used to show the orientation of directed edges (arcs). ARRWID (arrow width) is the length of the projection of the arrow's side onto a line perpendicular to the associated arc.

CANCELSTRING (default: "CANCEL")

The value of this parameter is used to cancel a operation when the system is reading from the keyboard. If, for example, the system is waiting for a user to enter a graph title, then by entering the current value of CANCELSTRING the operation will be cancelled, and the graph's title will be unchanged.

If a user wishes to use "CANCEL" as a title or label, then CANCELSTRING can be changed to allow this. CANCELSTRING can be any non-blank character string of length less than or equal to 8.

LABDIS (default: 1.0)

This is the length (in inches) that is used to determine whether or not the default position is used

for labels. When labeling a vertex or an edge, if the tracking pattern is less than LABDIS inches from the labeling point[4] then the label is placed at the centre of the tracking pattern. On the other hand, if the tracking pattern is more than LABDIS inches from the labeling point (or in a different region), then the label is placed at the default location. The default location is determined using the parameters XLABLOC and YLABLOC, which can also be changed using SET (see below).

XLABLOC (default: -0.025)

This parameter is the relative x-coordinate of the default label location (see also the parameter LABDIS). It is added to the x-coordinate of the vertex or edge labeling point to obtain the x-coordinate of the (lower left corner of the) label.

YLABLOC (default: 0.015)

This parameter is the relative y-coordinate of the default label location (see also the parameter LABDIS). It is added to the y-coordinate of the vertex or edge labeling point to obtain the y-coordinate of the (lower left corner of the) label.

LABELHT (default: 0.15)

This is the height (in inches) of labels that are displayed on the screen.

DRAWBUT (default: 1)

This parameter is the number of the "draw button". It can be any value from 1 to 18 inclusive.

The draw button is used for two purposes. First, it is the button that is depressed when in "add vertex" mode to indicate that a vertex should be drawn at the tracking pattern location. Second, it is used in "add

---

[4]The labeling point of a vertex is the centre of the vertex; the labeling point of an edge is the "middle" of the edge. Refer to the LABEL command description for a more detailed explanation.

edge" mode to indicate that a free-hand edge is being drawn.

(Note: the foot switches are treated as "buttons" 17 and 18.)

READWAIT (default: 75)

This parameter controls the delay between reads of the Adage screen. It is expressed as an integral number of 1/300's of a second. Thus the default wait is 75/300 = 0.25 seconds.

Every time the system attempts to read from the Adage screen, it first performs a wait calculated using READWAIT. This prevents multiple light pen hits which can occur if a user accidentally holds the light pen to an item too long.

VINCREMENT (default: 0.05)

This is the amount by which the size of the vertex prototype is incremented or decremented when the SIZE command is used in draw mode.

PLOTSIZE (default: 10.0)

This parameter is the size (in inches) of any plots that are produced. The plots are scaled so that the enclosing square is PLOTSIZE X PLOTSIZE inches. The default is the actual size of the image as it appears on the graphics screen.

Error Messages:

INVALID REGION - TRY AGAIN

Attempt to move (XSCREEN,YSCREEN) out of the drawing area.

INVALID PARAMETER - TRY AGAIN

An invalid parameter of some sort has been input. The possibilities are: a real number instead of an integer, or vice versa; an attempt to set CANCELSTRING to blank; an attempt to set DRAWBUT to a number out of the allowable range.

- LABELS

Purpose:

To selectively display the labels of the graph in the current area.

Description:

When LABELS is activated, the labels on the graph in the current area are displayed according to the current "label mode". The label mode is one of NONE, ALL, VERTEX, or EDGE. The label mode can be changed by activating one of the menu items which appear immediately to the right of LABELS. The initial label mode is ALL.

Labeling with NONE activated will cause all displayed labels (if any) to be rubbed out. Labeling with ALL will cause all labels to be displayed. To display only vertex labels VERTEX is used, and to display only edge labels EDGE is used.

When a graph is labeled in this way, an internal indicator associated with that graph is changed to correspond with the current label mode. This then becomes a property of that graph, and if the graph is, say, rubbed out and then later re-displayed, the displayed image will reflect the new label mode.

- PLOT

Purpose:

To plot the image which appears on the Adage screen.

Description:

When PLOT is activated, a "plot file" is prepared for later entry into the operating system's plot queue. PLOT can be activated as many times a desired during a particular run of the system. On each activation, an exact copy of the Adage screen (including menu) is "plotted". (The tracking pattern is not included in plots.)

When the system is stopped (see STOP) all plots for that

run are automatically sent to the plot queue.

- SAVE

Purpose:

To save the graph in the current area in a disk file.

Description:

When SAVE is activated, the system responds by asking the user for the name of the file in which the graph is to be saved. If the attempt to open this file is successful, the system saves the graph in the current area in that file. As many graphs as desired can be saved in a particular file.

A graph need not have a title in order to be saved, but the presence of a title sometimes makes restoration easier.

Error Messages:

NO GRAPH IN CURRENT REGION

FILE SPECIFIED DOES NOT EXIST

CANNOT OPEN FILE - RETURN CODE nn

The attempt to open the file was unsuccessful. The return code can be used to determine the exact cause.

WARNING: NOT A SEQUENTIAL FILE

If the file is not a sequential file then new data put into the file will be overwritten rather than appended. Thus, if graphs are added to a line file that already contains graphs, the old graphs will be partially or totally destroyed, and therefore are lost.

If a line file is specified, the system will ask for confirmation of the SAVE. If the user confirms the SAVE, the specified line file will be used; if not,

the SAVE will be cancelled.

FILE SPECIFIED IS FULL

The saved graph will be incomplete is this case.

ERROR IN WRITE - RETURN CODE IS nn

This is caused by an error condition occuring in the MTS WRITE routine. The return code can be used to determine the exact cause.

- RESTORE

Purpose:

To restore one or more graphs that have been saved in a disk file.

Description:

When RESTORE is activated, the system responds by asking the user for the name of a file from which restoration will take place. If this file is opened successfully, the user is asked to enter a request.

RESTORE operates in one of three request modes: "all", "select", or "title" mode. To restore in all mode the request should be the word "ALL" (or abbreviation "A"). If this request is entered, then all graphs in the specified file are restored. They are not displayed on the screen immediately, but instead are brought into the system and can later be displayed using DISPLAY.

To restore in select mode, the request "SELECT" (or "S") is used. In select mode, for each graph in the specified file the user is queried as to whether of not that graph should be restored. For each graph, the graph title is displayed (or the graph number if the graph does not have a title), as well as three choices: "CANCEL", "RESTORE", and "IGNORE". If CANCEL is chosen, then restoration stops and the system begins waiting for a new command. If RESTORE is chosen then the graph is restored (but not displayed), and if IGNORE is chosen then the graph is not restored. For both RESTORE and IGNORE the system will continue looking for graphs in the file after taking the appropriate action.

If the request is not "ALL", "A", "SELECT", or "S" then the system assumes it is a graph title. In this case the specified file is searched for the first graph with that title, and if such a graph is found it is restored and then displayed in the current area.

Whenever a graph is restored its number and title are printed on the 3270 screen.

Error Messages:

FILE SPECIFIED DOES NOT EXIST

CANNOT OPEN FILE - RETURN CODE IS nn

The attempt to open the file was unsuccessful. The return code can be used to determine the exact cause.

WARNING: FIRST RECORD IN FILE NOT A GRAPH

This probably indicates that a file has accidentally been specified which does not contain saved graphs. When this warning occurs the system asks the user whether or not he or she wishes to continue.

UNEXPECTED RECORD TYPE - ERROR RETURN

UNEXPECTED EOF - ERROR RETURN

Unexpected end-of-file.

LENGTH INCOMPATIBILITY - ERROR RETURN

A record was read that was the correct type but the incorrect length.

SYSTEM I/O ERROR - ERROR RETURN nn

This is caused by an error condition occuring in one of the MTS I/O routines. The return code "nn" can be used to determine the exact cause.

- TITLE GRAPH

Purpose:

To title the graph in the current area.

Description:

When TITLE GRAPH is activated, the user is asked to enter a title at the keyboard. A title can be up to 40 characters in length.

TITLE GRAPH can also be used to replace an existing title.

- DISPLAY

Purpose:

To display a graph in the current area.

Description:

When DISPLAY is activated, the system responds by displaying a list of the titles of the graphs which have been defined to the system. (If a graph does not have a title, its number is displayed.) The user then uses the light pen to choose one of the graphs which is then displayed in the current area.

Graphs which have been introduced into the system by drawing (as opposed to introduction by RESTORE) and graphs which have been restored but altered, are indicated by an asterisk ("*") to the left of their name. These are the graphs which a user may be interested in saving at the end of a run.

Error Messages:

NO GRAPHS TO BE DISPLAYED

No graphs have been defined to the system.

THAT GRAPH IS ALREADY DISPLAYED

A particular graph can only be displayed in one region

on the screen. If the same image is desired in two or more regions then DUPLICATE should be used.

- RUB OUT

Purpose:

To rub out the graph in the current area.

Description:

When RUB OUT is activated, the graph in the current area (if any) is rubbed out. A graph that is rubbed cut is not destroyed, and can be re-displayed using DISPLAY.

- DUPLICATE

Purpose:

To duplicate both the internal structure and the image of the graph in the current area.

Description:

When DUPLICATE is activated, the system responds by asking "ON WHICH REGION?". The user then indicates a region by choosing a region mark with the light pen.

If the region specified is not the current region, then DUPLICATE proceeds to make a distinct internal copy of the graph, and then displays the copy in the specified region.

- MOVE

Purpose:

To move the graph in the current area to another region.

Description:

When MOVE is activated, the system responds by asking "TO WHERE?". The user then indicates a region by choosing a region mark with the light pen.

If the region specified is not the current region, then the graph in the current region is rubbed out and re-displayed in the specified region.

- CANCEL

Purpose:

To cancel an operation when the system is reading from the Adage screen.

Description:

If CANCEL is activated when the system is waiting for input from the Adage screen, then (if possible) the system will cancel the current operation and return to its previous state. Usually this previous state is the state of waiting for a new command.

COMMANDS IN THE DRAW MENU

- RESET

Purpose:

To reset the vertex and edge prototypes to their default characteristics.

Description:

When RESET is activated, the vertex and edge prototypes are reset to their default intensities and dashing is turned off. The vertex prototype is also reset to its default size and shape.

- BOLD-DASHED-NORMAL

Purpose:

To change the intensities and dashing of the vertex and edge prototypes.

Description:

The small squares to the left of BOLD, DASHED, and NORMAL are used to change the edge prototype; the squares to the right are used to change the vertex prototype. For example, the square to the left of BOLD is used to change the edge prototype to bold face. The two boxes on the extreme left and the extreme right ("between" BOLD and DASHED) will change the appropriate prototype to dashed bold face.

Thus, the intensities and dashing of the two prototypes can be changed independently using the squares. The prototypes can also be changed simultaneously by activating the words "BOLD", "DASHED", and "NORMAL". So, for example, if "BOLD" is activated, then both the prototypes are changed to bold face.

- SIZE

Purpose:

To change the size of the vertex prototype.

Description:

When the "-" to the left of SIZE is activated, the size of the vertex prototype is decreased by the parameter VINCREMENT; when the "+" to the right of SIZE is used, the size is increased by VINCREMENT.

The parameter VINCREMENT can be changed using SET.

- SHAPE

Purpose:

To change the shape of the vertex prototype.

Description:

When the "-" to the left of SHAPE is activated, the number of sides of the vertex prototype is decreased by 1; when the "+" to the right of SHAPE is used, the number of sides is increased by 1.

The number of sides is not allowed to go below 3 or above 9.

- ADD

Purpose:

To add vertices and edges to graphs on the screen.

Description:

When the box to the right of "ADD" is activated, the system is ready to add vertices; when the box to the left is activated, the system is ready to add edges.

To add a vertex, the user simply moves the tracking pattern to the desired position and presses the draw button (initially button 1, at the top left corner). A vertex which looks like the vertex prototype is then created, added to the graph in the appropriate region, and displayed.

To add an edge, the user first specifies a "from" vertex. When this is done, a blinking cross is displayed at the centre of the vertex. The user is then expected to either specify a "to" vertex or press the draw button.

If a vertex is specified, then an edge which looks like the edge prototype is created, added to the graph in the appropriate region, and displayed. The edge will be a straight line between the centres of the two specified vertices.

Sometimes, however, a straight edge is not suitable, but instead a "curved" edge is desirable. Such an edge, a "free-hand edge", can be drawn using the draw button and

the tracking pattern. If the system has had a "from" vertex specified to it, then if the draw button is depressed a section of an edge is drawn from the "from" vertex to the centre of the tracking pattern. The tracking pattern can be moved and the draw button depressed again, in which case another section of an edge is added, this time from the end of the last section to the new location of the tracking pattern. As many sections as desired can be added in this way. The process is terminated by specifying a "to" vertex as the second end of the final edge section.

If the system is in directed mode when an edge is drawn, the edge will be a directed edge with its tail at the "from" vertex and its head at the "to" vertex.

Error Messages:

TRACKING PATTERN NOT IN A REGION (vertex)

WRONG REGION (edge)

The "from" vertex and the "to" vertex must be in the same region. Also, free-hand edges cannot cross region boundaries.

LOOP MUST BE A FREE-HAND EDGE (edge)

When the "from" and "to" vertices are the same vertex, the edge must be free-hand (otherwise the edge cannot be seen).

WARNING - MULTIPLE EDGE (edge)

This message appears whenever a multiple edge is drawn. This is done because it is possible to draw multiple edges which are impossible to distinguish (that is, the second is drawn "on top" of the first).

- DELETE

Purpose:

To delete vertices and edges from graphs on the screen.

Description:

When the box to the right of "DELETE" is activated, the system is ready to delete vertices; when the box to the left is activated, the system is ready to delete edges.

To delete a vertex or an edge the user simply hits the desired object with the light pen. When a vertex is deleted, then all incident edges are also deleted.

- LABEL

Purpose:

To label vertices and edges.

Description:

When the box to the right of "LABEL" is activated, the system is ready to label vertices; when the box to the left is activated, the system is ready to label edges.

At this point the user hits either a vertex or an edge with the light pen, and a blinking cross is displayed at the "labeling point" (see below). The system then asks the user to "ENTER LABEL AT KEYBOARD", and reads the label (up to 8 characters). The label is then displayed.

Labels can be positioned explicitly or a default location can be used. The parameter LABDIS controls this feature. If the tracking pattern is less than LABDIS inches (initially 1 inch) from the labeling point then the label is placed at the centre of the tracking pattern. If, however, the tracking pattern is more than LABDIS inches from the labeling point, or in a different region, then the label is placed at the default location. The relative coordinates of the default label location are the parameters XLABLOC and YLABLOC. LABDIS, XLABLOC, and YLABLOC all can be changed using SET. Refer to the description of SET for more details.

The "labeling point" is the point relative to which labels are both stored in the internal representation and

displayed. The labeling point of a vertex is the centre of the vertex. The labeling point of an edge is the "middle" of the edge. For simple edges (edges which are one straight line between two vertices) the middle is simply the geometric mid-point of the line. For free-hand edges, the middle is determined by finding an edge section which is "roughly equidistant" from the incident vertices, and then using the mid-point of that section as the middle.

- CHANGE

Purpose:

To change the appearance of vertices and edges.

Description:

When the box to the right of "CHANGE" is activated, the system is ready to change vertices; when the box to the left if activated, the system is ready to change edges.

To change a vertex or an edge, the desired object is hit with the light pen. The appearance of the vertex or edge is then changed to look like the appropriate prototype.

- REPEAT

Purpose:

To allow automatic repetition of various system functions.

Description:

All system commands and functions normally execute only once before control is returned to the top level of the system. By enabling REPEAT, however, certain commands can be made to automatically repeat. The commands affected are those in the lowest section of the draw menu (below the lower dashed dividing line). These commands will continue to repeat until they are explicitly turned off.

To turn off a menu item that is being repeated, a user must hit either REPEAT, CANCEL, or the menu item itself. If REPEAT is hit, REPEAT is also turned off. Only one command can be in use at one time, so if the system does not respond to an attempt to turn on a particular command, it

may be because another command is being held on by REPEAT. The one exception to this rule is that the prototypes can be adjusted while "add edge", "add vertex", "change edge", or "change vertex" is turned on.

- SUBDIVIDE EDGE

Purpose:

To subdivide an edge by creating a new vertex in the middle of the edge.

Description:

When SUBDIVIDE EDGE is activated, the system waits for an edge to be specified with the light pen. When an edge is chosen, the edge is replaced with two new edges and a new vertex in the way indicated by figure 9. Directed edges can also be subdivided, with the obvious result.

- REMOVE DEGREE 2 VERTEX

Purpose:

To remove a vertex of degree 2 and add an edge between the two vertices to which it was connected.

Description:

When REMOVE DEGREE 2 VERTEX is activated, the system waits for a vertex to be specified with the light pen. When a vertex is chosen, the vertex and its two incident edges are deleted, and an edge is added between the two vertices to which the deleted vertex was connected. Refer to figure 10 for some examples.

Error Messages:

VERTEX MUST BE DEGREE 2

This message appears whenever the vertex chosen is not acceptable, including such cases as a vertex with two inpointing arcs.

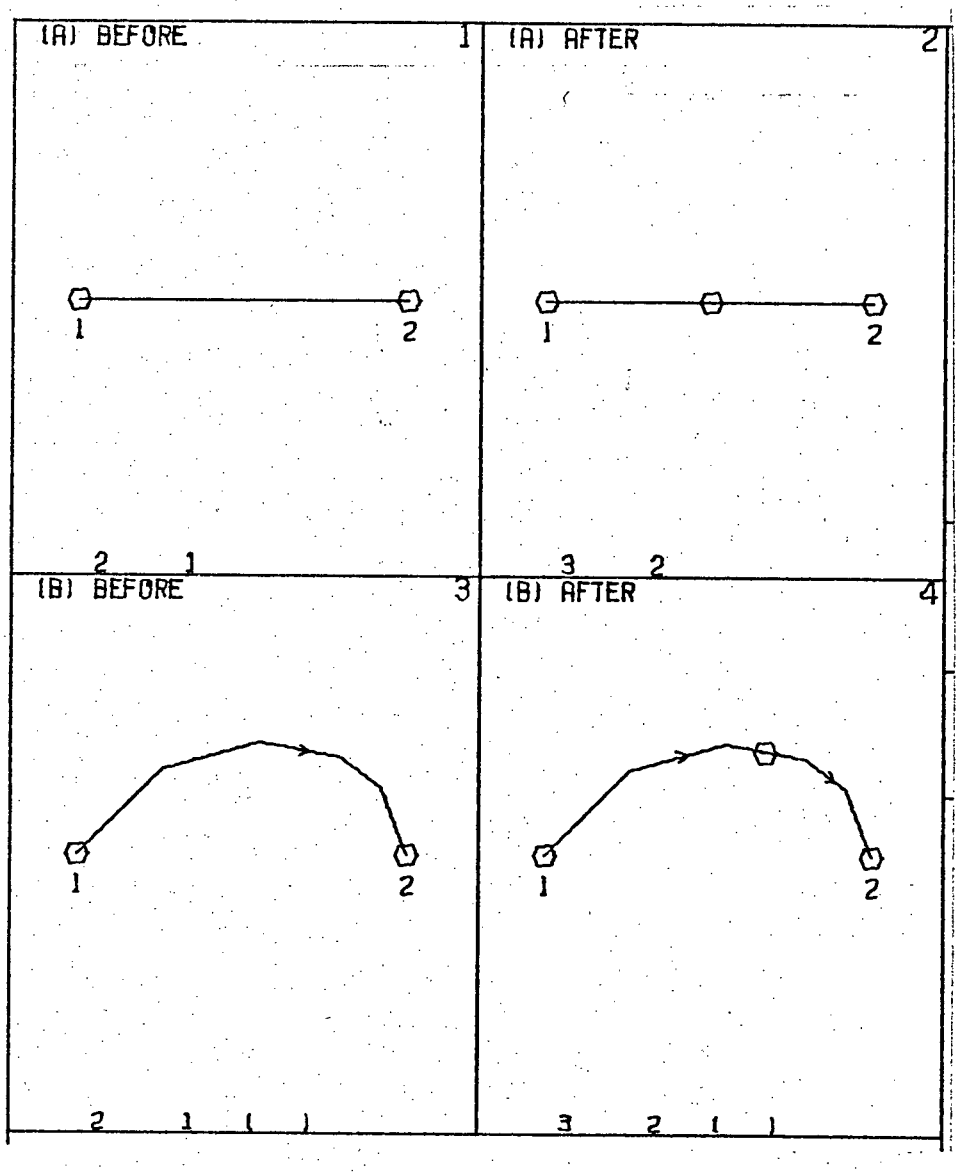

Figure 9 - The Effect of SUBDIVIDE EDGE

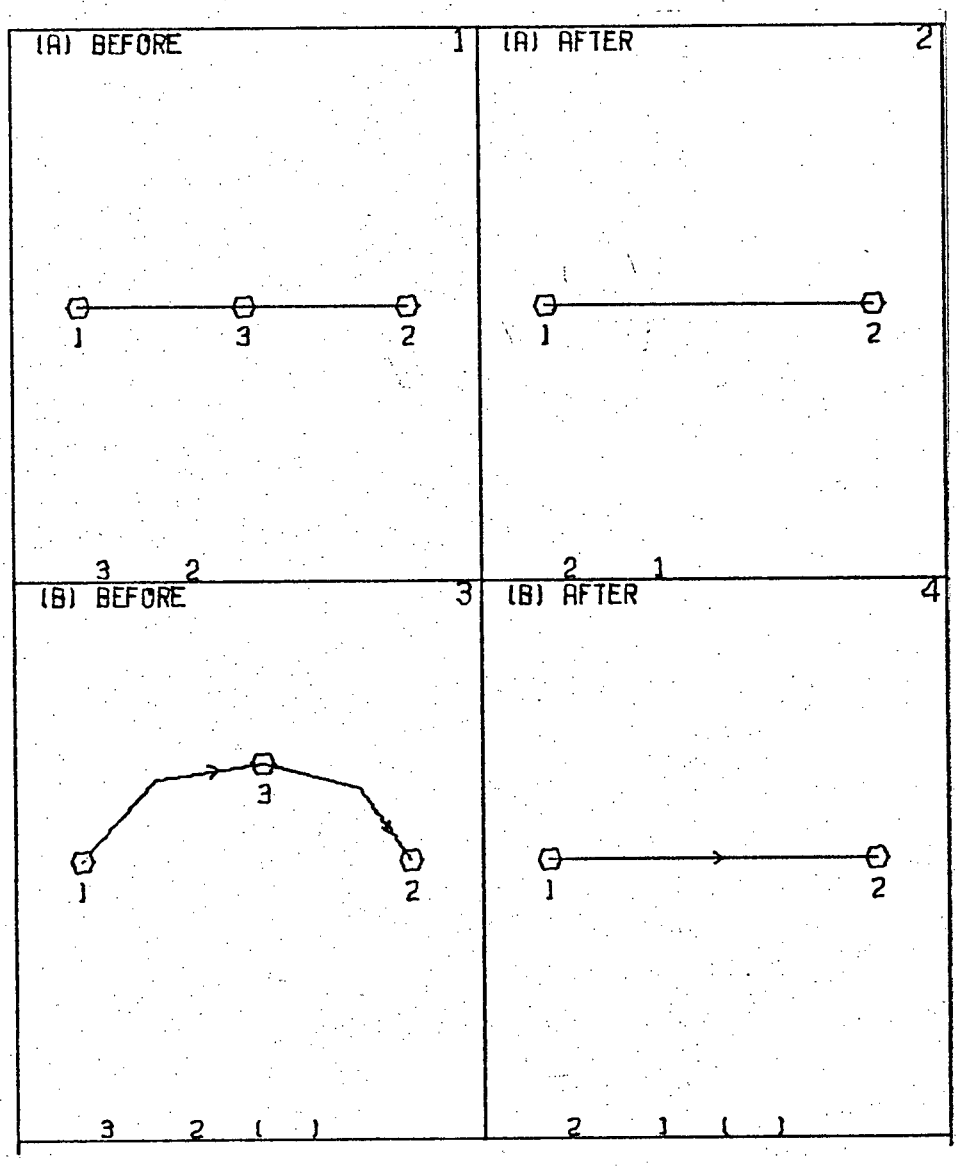

Figure 10 - The Effect of REMOVE DEGREE 2 VERTEX

- ASSOCIATE VERTICES

Purpose:

To associate two vertices, that is, to perform an elementary homomorphism.

Description:

When ASSOCIATE VERTICES is activated, the system waits for the user to specify the first vertex. When this is done, a blinking cross is displayed at the centre of the vertex. The user then must specify a second vertex. When this second vertex is specified, the first vertex is deleted (including incident edges) and edges are added so that the second vertex is adjacent to all vertices that were adjacent to the first vertex. Multiple edges are not created during this operation, so if both the first vertex and the second vertex are adjacent to the same third vertex, only one edge remains in the final graph.

If two edges must be combined, and one of the edges is directed while the other is undirected, then the resulting edge will be undirected. If both edges are directed, and they have opposing directions, then the resulting edge will be undirected, and a warning message will be displayed along with a blinking cross on the edge in question.

Figure 11 shows some examples of the results of associating vertices.

Error Messages:

WRONG REGION

The vertices to be associated must be in the same graph.

SAME VERTEX

The vertices to be associated must be distinct.

CANNOT ASSOCIATE ADJACENT VERTICES

NOTE EDGES CHANGED TO UNDIRECTED

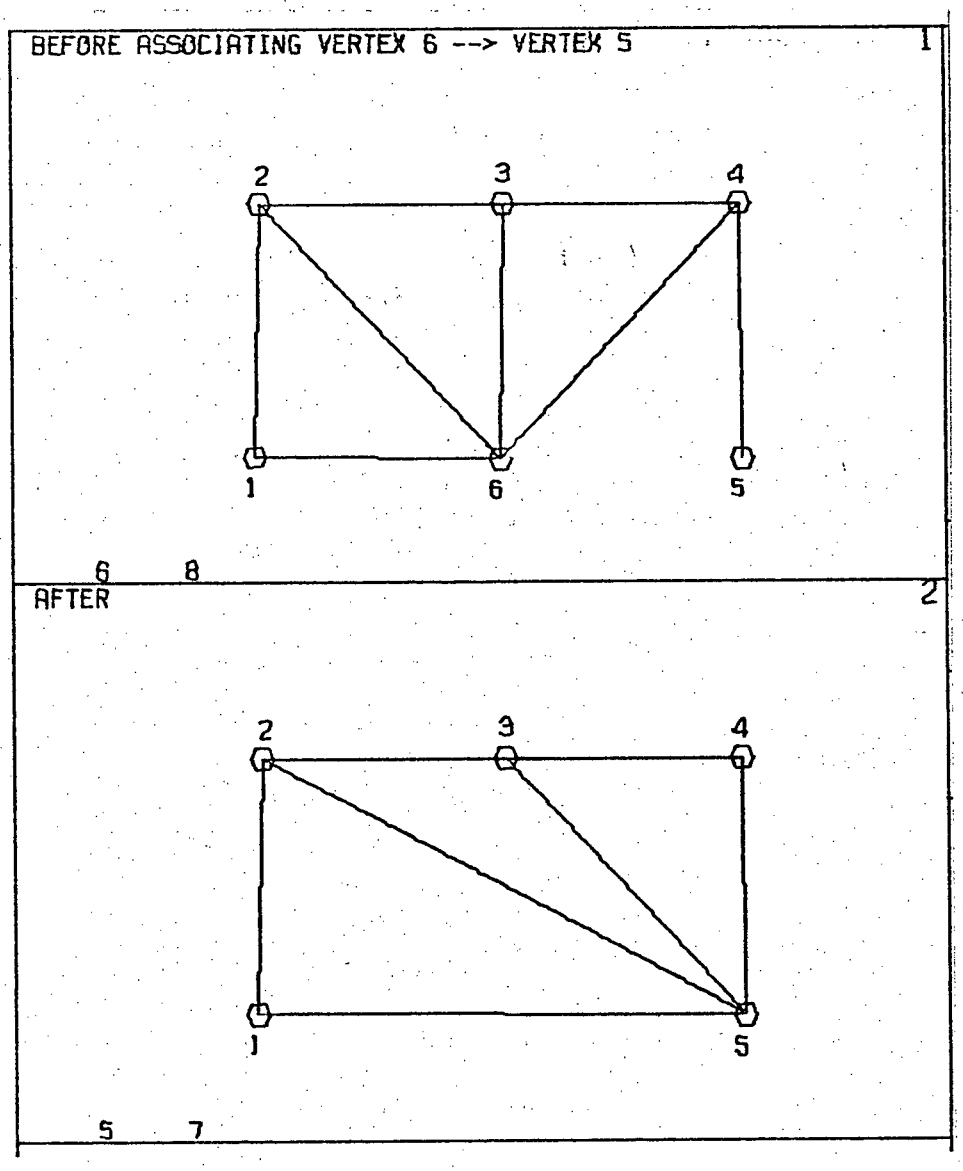

Figure 11 - The Effect of ASSOCIATE VERTICES

All undirected edges which resulted from a pair of oppositely directed multiple edges are indicated by a blinking cross.

- REVERSE ARC

Purpose:

To reverse the direction of an arc.

Description:

When REVERSE ARC is activated, the system waits for the user to specify an arc. When this is done, the orientation of the arc is reversed.

Error Messages:

THAT EDGE IS NOT DIRECTED

- UNDIR <--> DIR

Purpose:

To change an undirected edge into a directed edge, and vice versa.

Description:

When UNDIR <--> DIR is activated, the system waits for an edge to be specified. If the edge is undirected, it is changed to directed; if directed, it is changed to undirected.

- LOCK ON VERTEX

Purpose:

To move a vertex to a new location.

Description:

When LOCK ON VERTEX is activated, the system waits for the user to specify a vertex to be moved. When a vertex is chosen, it is moved to the current location of the tracking pattern. Of course, all of its incident edges are also redrawn.

Error Messages:

WRONG REGION

A vertex cannot be moved to a different region.

- LOCK ON GRAPH

Purpose:

To move a whole graph to a new location.

Description:

When LOCK ON GRAPH is activated, the system waits for the user to specify a vertex. When a vertex is chosen, that vertex is moved to the current location of the tracking pattern, and all other vertices in that graph are moved the same relative distances in the x-direction and the y-direction.

No part of the graph will be moved out of the containing region.

Error Messages:

WRONG REGION

The specified vertex cannot be moved to a different region.

COMMANDS IN THE FUNCTION MENU

- GO

Purpose:

To signal the start of certain operations.

Description:

Some operations use GO as a signal to continue to the next step. For example, when permuting labels (see PERMUTE LABELS below), the first step is to enter a permutation. When this is done, GO is activated and the labels are actually permuted.

- MIN/MAX DEGREE

Purpose:

To determine the minimum and maximum degrees of a graph.

Description:

When MIN/MAX DEGREE is activated, the system determines and displays the minimum and maximum degrees of the graph in the current area. If the graph is directed, both the minimum and maximum indegree and the minimum and maximum outdegree are displayed.

Error Messages:

NO GRAPH IN THE CURRENT AREA

GRAPH HAS NO VERTICES

GRAPH HAS NO EDGES - ALL DEGREES ARE ZERO

- PERMUTE LABELS-PERMUTE COORDS

Purpose:

To permute vertex labels or vertex coordinates.

Description:

When PERMUTE LABELS or PERMUTE COORDS is activated, the system waits for a cyclic permutation to be entered. The labels or coordinates will be changed according to this permutation.

To enter a permutation, the user simply chooses a number of vertices with the light pen. As each vertex is chosen, its sequence number in the permutation is displayed as a blinking number adjacent to the vertex. Vertices can appear in the permutation only once. If a vertex is chosen a second time, an error message is generated, unless the vertex has the highest sequence number. In this latter case the vertex is removed from the permutation, thus allowing a user to back up.

When the permutation is complete, the user must hit GO with the light pen, at which time either labels or coordinates will be permuted, depending on the type of permutation in effect.

Examples of the effect of PERMUTE LABELS and PERMUTE COORDS are shown in figures 12 and 13.

Error Messages:

MUST SPECIFY AT LEAST TWO VERTICES

There must be at least two vertices in the permutation.

WRONG REGION

All the vertices must be in the same graph.

VERTEX ALREADY IN PERMUTATION

PERMUTE CANCELLED

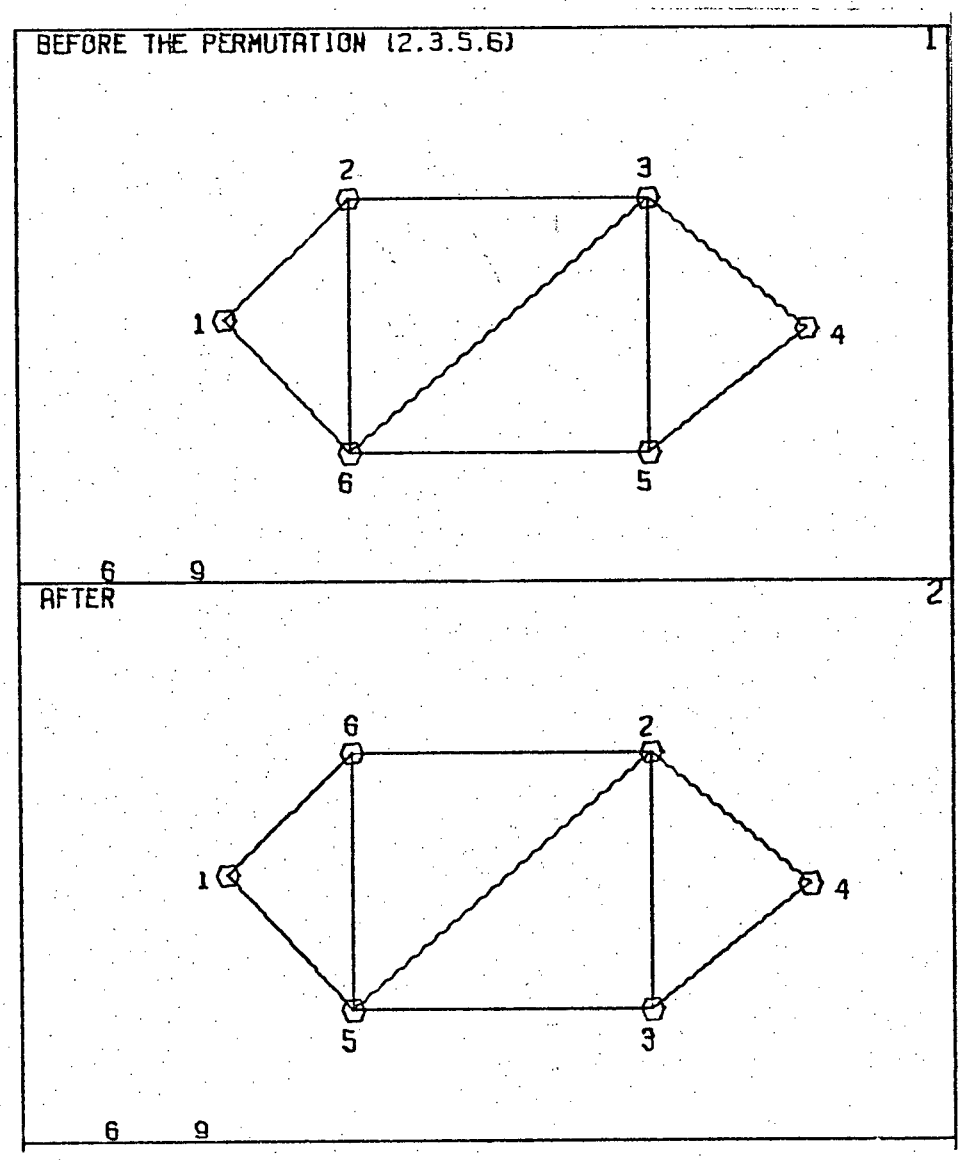

Figure 12 - The Effect of PERMUTE LABELS

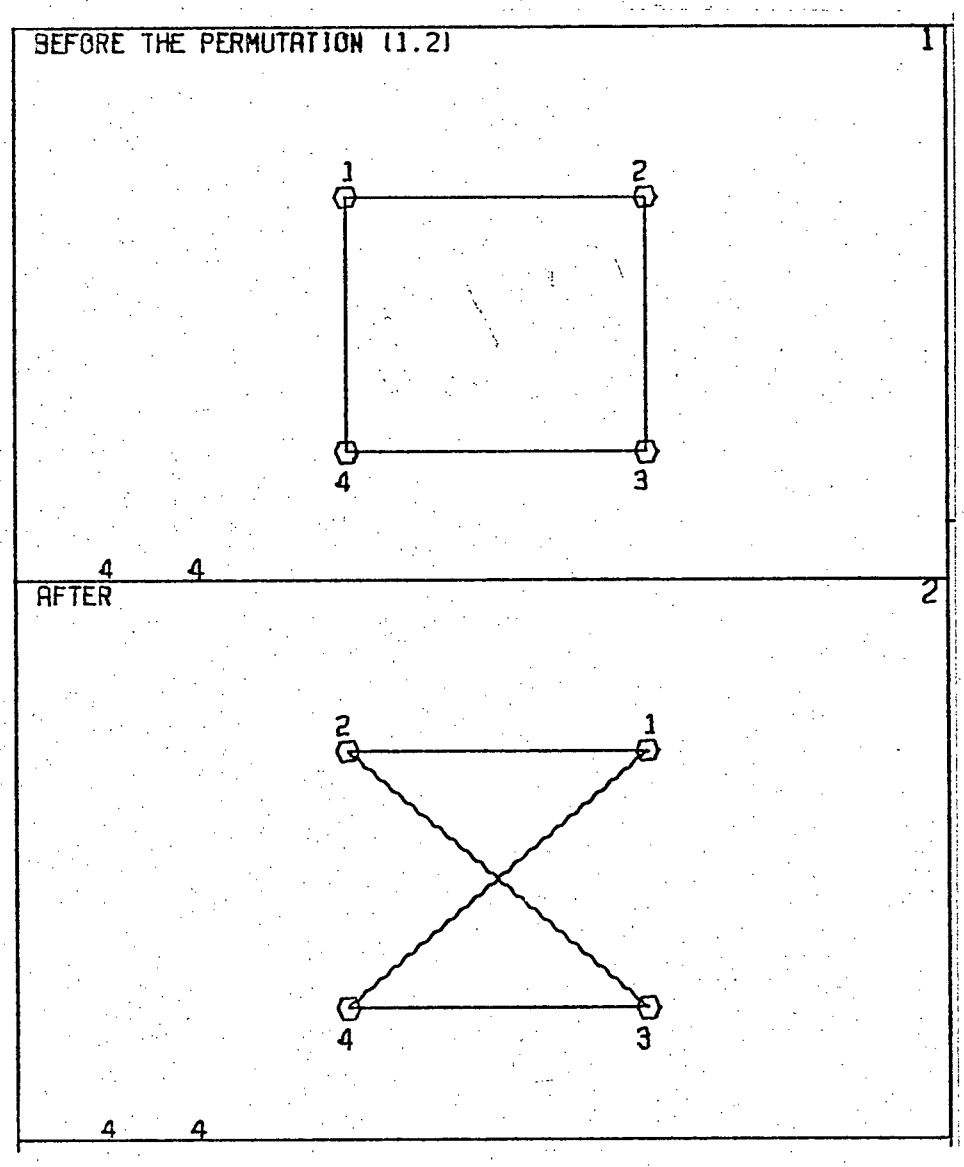

Figure 13 - The Effect of PERMUTE COORDS

All vertices in the permutation have been "removed".

- BLOCKS/CUTNODES/BRIDGES

Purpose:

To determine and display the blocks, cutnodes, and bridges of the graph in the current area. A spanning tree is also found, if one exists.

Description:

When BLOCKS/CUTNODES/BRIDGES is activated, the system will perform its calculations on the graph in the current area. When completed, the different parts of the graph will be indicated as follows: cutnodes will be bold face vertices; bridges will be bold face edges; blocks will be those edges labeled with the same label; a spanning tree (if any) will be indicated by edges of normal intensity (in addition to the bridges). All other vertices will be of normal intensity and all other edges will be dashed.

Figure 14 shows an example of the application of BLOCKS/CUTNODES/BRIDGES. The algorithm used for this operation is taken from [17].

Error Messages:

NO GRAPH IN THE CURRENT AREA

GRAPH HAS NO EDGES

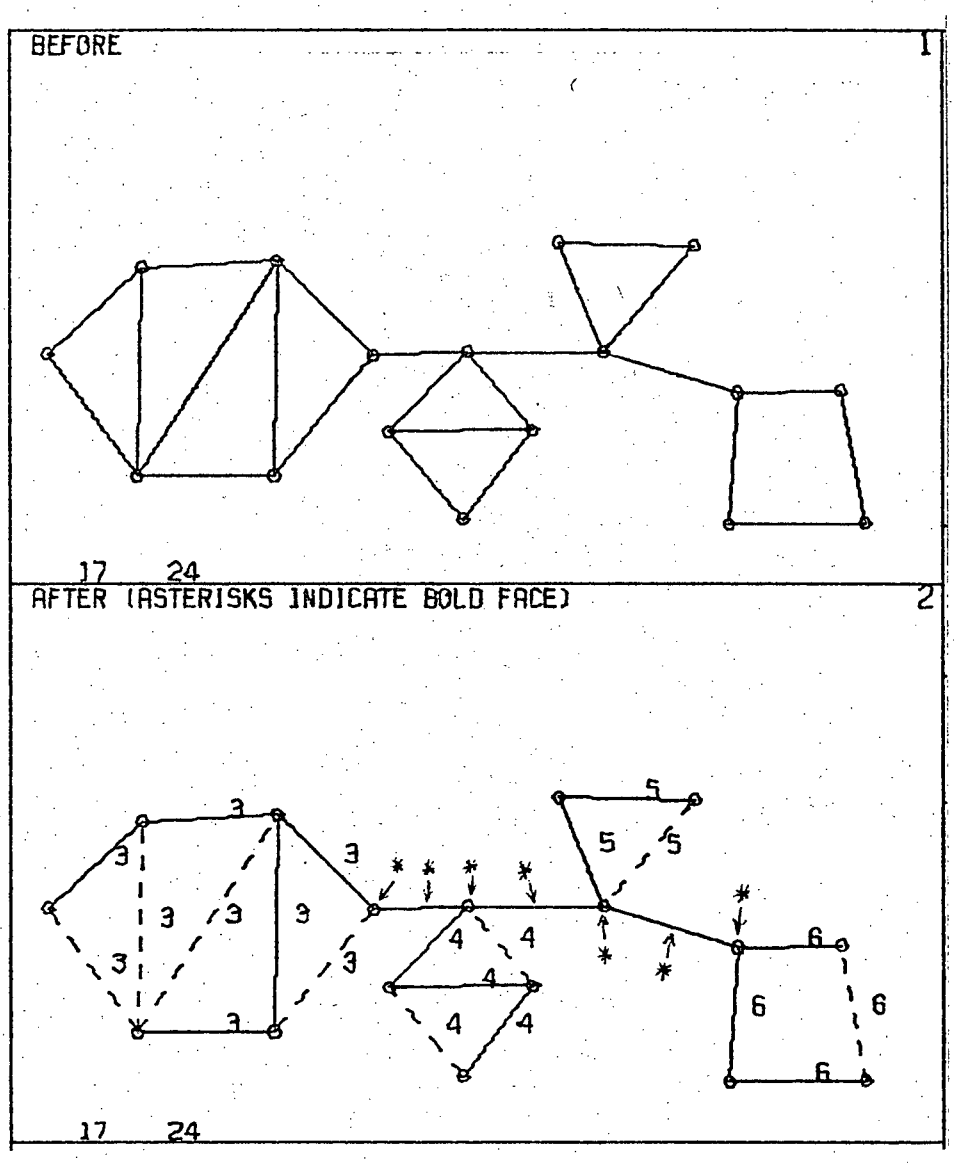

Figure 14 - The Effect of BLOCKS/CUTNODES/BRIDGES

INDEX TO COMMANDS

ADD	82
ASSOCIATE VERTICES	89
BLOCKS/CUTNODES/BRIDGES	97
BOLD	81
CANCEL	80
CHANGE	85
DASHED	81
DELETE	84
DISPLAY	78
DRAW	67
DUPLICATE	79
FUNCTION	67
GO	93
HELP	68
KEYBOARD	68
LABEL	84
LABELS	74
LOCK ON GRAPH	92
LOCK ON VERTEX	92
MIN/MAX DEGREE	93
MOVE	79
NORMAL	81
PERMUTE COORDS	94
PERMUTE LABELS	94
PLOT	74
READY	66
REGIONS	67
REMOVE DEGREE 2 VERTEX	86
REPEAT	85
RESET	80
RESTORE	76
REVERSE ARC	91
RUB OUT	79
SAVE	75
SET	70
SHAPE	82
SIZE	81
STOP	70
SUBDIVIDE EDGE	86
TITLE GRAPH	78
UNDIR <--> DIR	91