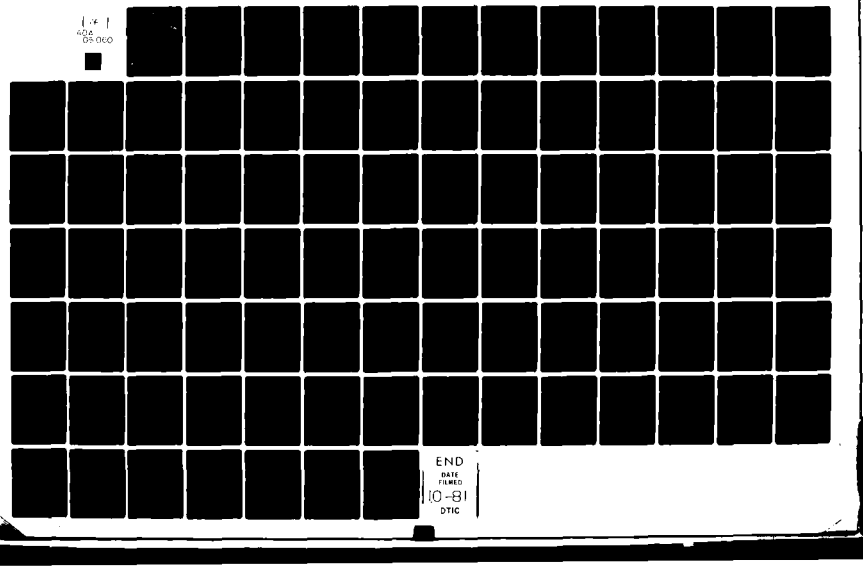
AD-A105 060 KANSAS STATE UNIV MANHATTAN DEPT OF COMPUTER SCIENCE F/G 9/2
A GRAPHICS SUPPORT SYSTEM FOR COMMUNICATIONS PROCESSES PROGRAMM--ETC(U)
AUG 80 R G SANDERS DAAG29-78-G-0200
UNCLASSIFIED TR-80-01 NL



END
DATE FILMED
10-81
DTIC

LEVEL

15

AD A105060

A GRAPHICS SUPPORT SYSTEM FOR COMMUNICATING PROCESSES PROGRAMMING

by

RICHARD GARY SANDERS

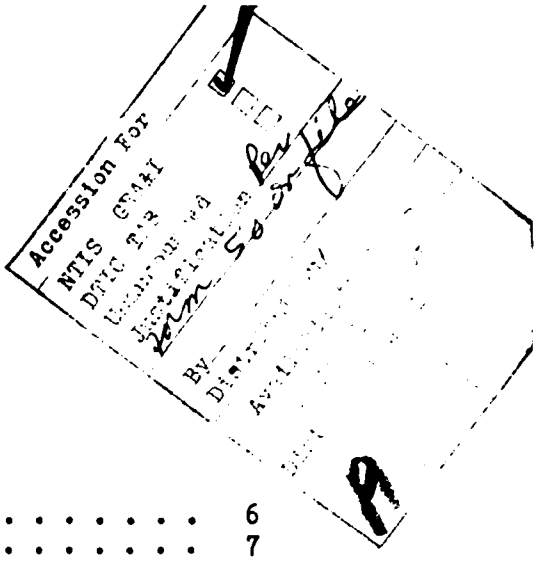
Technical Report
TR-80-01

14 TR 80-01

Department of Computer Science
Kansas State University
Manhattan, Kansas
August 1980

This research was supported in part by the Army Institute for Research in Management, Information, and Computer Systems under grant number DAAG 29-78-G-0200 from the Army Research Office.

TABLE OF CONTENTS



LIST OF ILLUSTRATIONS

ACKNOWLEDGEMENTS

This research was supported in part by the Army Institute for Research in Management, Information, and Computer Systems under grant number DAAG 29-78-G-0200 from the Army Research Office.

I would also like to express by gratitude to Dr. Virgil E. Wallentine who served as my major professor and provided valuable input into this document. I would also like to thank the other members of my thesis committee-- Dr. William J. Hankley and Dr Rod M. Bates for their assistance.

INTRODUCTION

The complexity of many sophisticated programming tasks requires a methodology to simplify and filter information to a manageable level. The GSS (Graphics Support System) described in this document will automatically draw pictures of software configuration networks of communicating sequential processes as they are constructed. A configuration, as defined in [5,3], is constructed of one to eight nodes. Each node can be hierarchical in nature and can itself be a configuration. Nodes communicate via ports through connections to other nodes. A node can be connected to as few as zero nodes to as many as seven nodes. A maximum of eight ports and connections is allowed per node, and more than one connection can exist between any two nodes. A connection can be specified to pass information in only one direction or in both directions. In addition, the nodes, ports, and connections in the picture must show the direction of information flow.

GSS does not build configurations although it could be modified to do so. It is meant as a documentation tool that assists in the understanding or explanation of a software configuration.

The most important contribution of GSS is the method used to determine the complex relationships that exist between the picture components. There are two basic relationships that must be analyzed before a picture can be drawn which are the relationship of one node to another as defined by connections, and the organization of the

nodes within the picture as defined by information flow.

Arbitrary configurations can be decomposed into three distinct types of objects: Hangers, pipes, and cycles. The decomposition is accomplished by following and analyzing all of the node connections and constructing patterns of linkage. A hanger is constructed of two nodes, A and B. Node A can only connect to node B, whereas node B can connect to nodes in addition to node A. A pipe is constructed of two nodes, A and B. Both node A and node B can both connect to nodes other than each other. A cycle is constructed of more than two nodes such that the nodes, when placed in a directed graph, form a cycle.

The purpose of building objects is to define a predictable, repeatable, heuristic that will draw pictures in the desired manner. The number of nodes in an object determines the shape of the object. An objects' shape is used to select a predefined pattern which defines how member nodes will be drawn relative to one another. Flow into and out of nodes is studied to determine where they should be placed relative to the picture and relative to other nodes within their parent object.

GSS allows the user to interact in drawing portions of a picture or it will draw the picture without user assistance. Once a picture has been drawn it can be saved and recalled later.

The graphics system described in this paper has been implemented in SEQUENTIAL PASCAL and is running on an INTERDATA 8/32. The output devices it is currently drawing pictures on include a dumb CRT, a lineprinter, a plotting printer, and a Chromatics Color Graphics System.

The algorithm is simple to alter and can be adapted with very

little trouble to devices with differing resolution or capabilites. If new object shapes are needed, the only required change is to add new patterns to accommodate the shape. Computing time for a picture with eight nodes and two ports per node is under two seconds. This is more than fast enough to support user interaction.

Chapter one documents the set of problems that drawing pictures of connected graphs presents and proposes solutions to the problems. Part one describes the software configuration that the GSS is designed to draw. Part two defines the graphics capabilities needed to draw a software configuration and discusses other design limitations. Part three discusses the heuristic that is proposed to draw configurations.

Chapter two documents the implementation of the heuristic. Part one defines high level algorithms that are used to determine relationships in the configuration and that are used to construct the actual picture. Part two discusses in general terms how the algorithms can be implemented in a high level language such as PASCAL. Part three examines the adapability of the algorithms with respect to adapting GSS to different display devices and to varying limit variables. The limit variables include such things as the number of pixels in the picture, the number of elements allowed, and the maximum number of ports per element. Part three also discusses how the use of a color display can effect the presentation of material.

Chapter three documents the conclusion. Part one presents a summary of this document. The performance of GSS is evaluated in part two and several proposed enhancements to GSS are presented in part three.

CHAPTER 1

1.1 Software Configuration

In [5] NADEX[7] is extended to support general graphs of communicating sequential programs. Using software tools described in [5,3], nodes can be distributed across a computer network. A software configuration is a graph of nodes of processes in a message based system. Nodes talk to one another by sending messages on Data Transfer Streams(DTSs). A DTS is connected to a node at a port. All nodes have at least one port. A DTS can carry information in only one direction. Therefore, the general graph that represents a software configuration is a directed graph. The nodes in the configuration are the nodes in the graph and the DTSs in the configuration are the arcs. A specific graph represents a specific software configuration.

Configurations can be either completely or partially defined. Partially defined configurations can later be combined to form complete configurations. Therefore, configurations can be defined hierarchically.

1.2 Requirements

When listing requirements for a computer graphics program, the designer must consider the input, the computing needs, and the output devices. The input for the GSS is from one of two types of files

which are a Partial Configuration Descriptor(PCD) file containing a record of the definition of a configuration, or a Picture(PIC) file containing the definition of a configuration picture which has already been drawn.

The computing needs are extensive. The primary objective of a graphics system is to draw pictures in a manner that the user expects. Figures 1, 2, and 3 show a number of sample configurations that the GSS is expected to draw. Note that the dining philosophers problem[4] is included as a prospective configuration. The problem of drawing a connected graph with cycles is the impetus for this thesis and the dining philosophers problem appears to be a good test for the heuristic since it is a complex cycle. Other examples show configurations made of hangers, pipes, and less complex cycles.

The GSS should be smart enough to draw a reasonably oriented picture without user intervention. This implies that the algorithm should be able to analyze the description of the graph, calculate possible ways to draw the configuration, select the best possibility, and then draw the picture. Flexibility for the user is important if the user is going to be able to draw a meaningful picture. Therefore, the user should be allowed to input recommendations as to how a picture should be arranged. This input could take the form of making suggestions to the GSS before the picture is drawn, or while the GSS is selecting the best possibility. The user should be able to take advantage of the fact that the GSS can faultlessly analyze the relationships between nodes and can in many cases draw a large number of different organizations of the configuration. This offers the possibility for the user to step through different computer generated

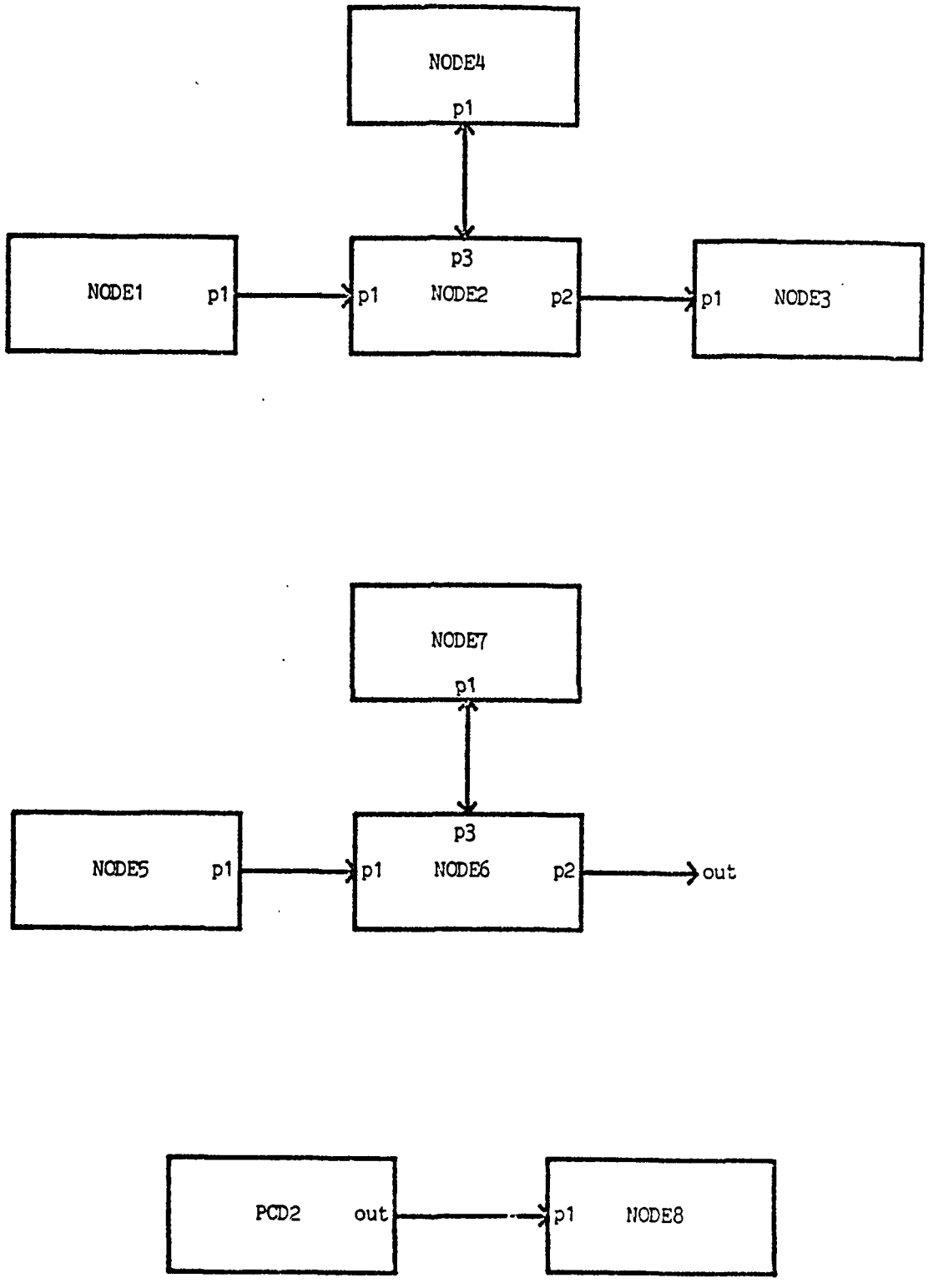


Fig. 1. Sample configurations

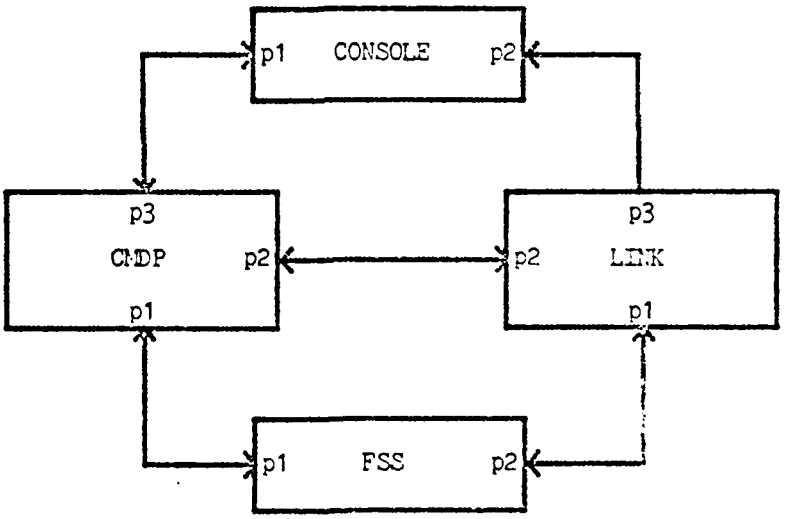


Fig. 2. Command processor configuration

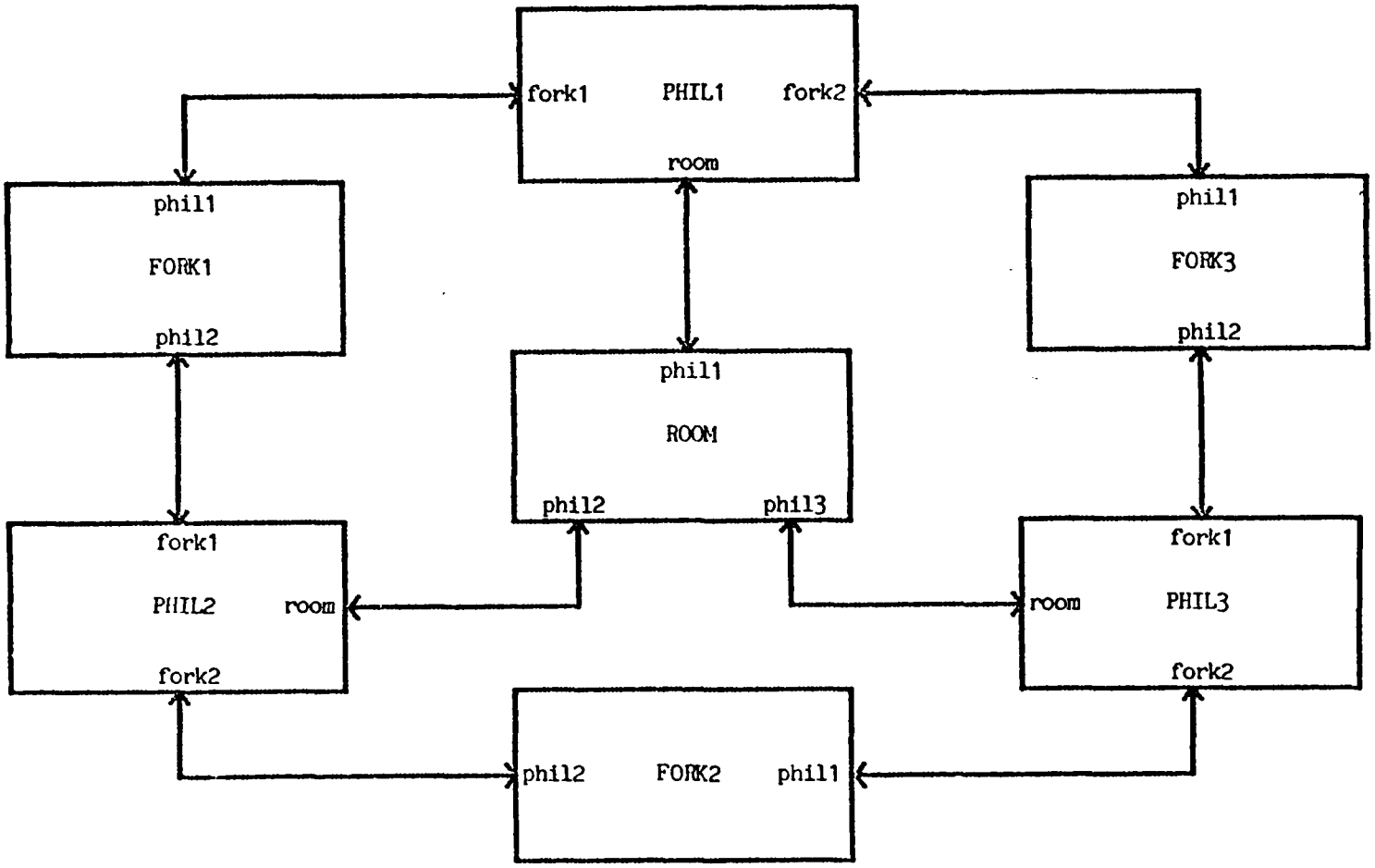


Fig. 3. Dining philosophers configuration

drawings of a configuration until the proper drawing is found.

Calculating the best way to draw a cycle is more difficult for a computer than for a user. However calculating all of the possible ways to draw a cycle is easier for a computer than for a user. Therefore, the computer should calculate all the possible ways to draw a cycle. The user should then be allowed to make a choice of a pattern. However, if the user does not want to make a choice, the computer should have the knowledge to make a selection on its own.

Elements must be plotted in such a way that crossing arcs are avoided thereby preventing the picture from getting too confusing. In many cases, rearranging the relative positions of the elements in the picture can reduce the number of crossed arcs. Therefore the GSS should attempt to select the pattern for plotting an object that will produce the least crossed arcs.

The algorithm should be readily adaptable to allow more than a certain number of nodes or ports to be drawn, or to adapt the GSS to new configuration types. Although the GSS is meant to run under NADEX which can run very large programs, the GSS should be designed in modules small enough so that it can be segmented to run on a small stand alone intelligent graphics device such as a Chromatics. This would also allow the segments to be split across the main computer and the intelligent output device thus gaining concurrent computing ability.

The user should be able to select from a variety of output devices including a dumb CRT, a line printer, a plotter, or a graphics display. Also, the sophistication of the picture output should reflect the sophistication of the output device. For instance, a CRT

has the resolution capability to display a configuration with eight nodes. While a plotter or graphics display can display sixteen nodes with ease. A color graphics display or color plotter also allows the user to display a more information with more clarity in less space.

The user should also have the capability to save and recall pictures that have been drawn correctly. Execution time is important since the system is interactive.

1.3 User Commands

The Graphics Support System may be accessed through the User Interface which allows the user to do the following types of commands:

a. Get PCD or PIC files that have been produced.

   The command is FILE(name: configuration name).

b. Display data concerning the picture components-nodes & ports.

   The command is DATA(name: element name).

c. Assist the program in drawing pictures by selecting object shapes.

   The command is ASSIST.

d. Manipulate a picture by shifting nodes or rotating objects.

   The commands are:

   TRANSLATE(direction:(up,down,left,right),unit:integer).

   ROTATE.

   SWITCH(name:node name,name:node name).

   GRID.

   MOVE(name:node;Row,Column:integer).

e. Move hierarchically through PCD file nodes to lower levels.

EXPAND(name: node name).

f. Select an output device upon which to draw the picture.

DRAW(device:(Console,Color,Plotter,Printer)).

g. Create a file and save a picture.

SAVE(name: file name);

The FILE command allows the user to search for an existing PIC file of the configuration that is to be drawn. If the PIC file does not exist then the requested PCD file is retrieved. This prevents the user from wasting unnecessary time redrawing a picture. If a PCD file is retrieved, it is automatically converted into a PIC record. The DATA command will display information relating to node requested. The information returned by the GSS includes the host name, the hierarchical name if the node is hierarchical, and a list and description of the internal and external ports in the node.

ASSIST supplies the user with the names of all nodes in the configuration and indicates if any objects in the picture are cycles. If cycles do exist, the user is asked if he wants to help the GSS to draw them. If the user wants to assist the computer the GSS displays the possible ways to draw each cycle and asks the user to make a choice. ASSIST can be repeated until the picture is formatted according to the user's expectations.

The next five commands allow the user to manipulate a drawn picture by moving nodes about. The picture is automatically redrawn on the current device after any one of these commands is executed. TRANSLATE will move the entire picture up, down, left, or right for the

designated number of units. ROTATE will rotate a particular cycle around its center in a clockwise or counterclockwise direction. SWITCH allows the user to switch the location of two nodes. The GRID command will draw an X,Y grid on the picture. This can assist the user by supplying the units needed to do a translate or a move. MOVE will move a node from one position to another without disturbing the rest of the picture.

EXPAND will take the desired node and determine if it is hierarchical. If it is, the hierarchical file name of the node is used to retrieve the file using the FILE command.

DRAW is used to actually display a configuration on a device. If no device name is supplied, the picture is drawn on the last device used or on the console.

The SAVE command allows the user to save a picture in a PIC file after a configuration has been drawn properly. The PIC file is not device dependent.

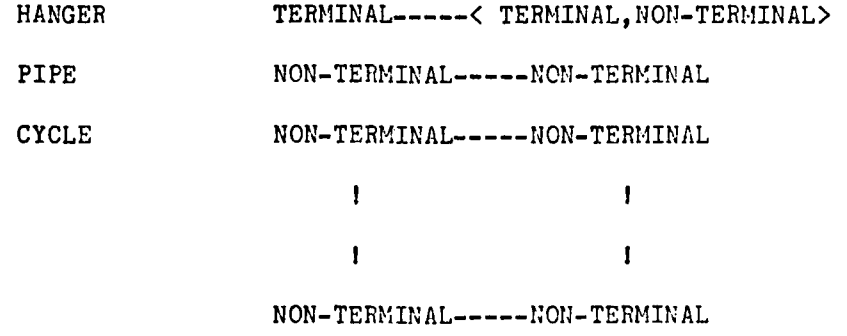
1.4 Heuristic

Graphics Systems typically permit a user to draw a picture by defining and plotting objects and arcs. However, as pictures get more complicated or as users begin to expect more from their system, the need for computer assistance in defining and drawing pictures becomes important. The requirement that the GSS should be able to draw pictures without user intervention made the algorithm a great deal more complicated since it now required a heuristic for determining how a picture should be organized. Also, the problem of drawing directed graphs with cycles is a great deal more complex than drawing graphs with

only pipes and hangers.

The possible object shapes are PIPE,CYCLE,and HANGER. The shapes may be differentiated by the type of elements they contain. Elements may be classified according to how many other elements they connect to, or touch. If an element touches only one other element, it is designated as a TERMINAL element. If an element touches two or more other elements then it is designated as a NON-TERMINAL. A HANGER contains two elements that touch each other and one element must be a TERMINAL. A PIPE contains two elements that touch one another and both elements must be NON-TERMINALS. A CYCLE contains three to eight elements that are related such that when all elements are connected according to touches they form a connected graph.

A graphic description of object shapes is shown below:



Objects are interconnected through common NON-TERMINALS. Elements and objects are located relative to one another according to their POSITION. The types of element positions are IN,MID,and OUT. Elements are classified according to the direction of the ARCs that connect to it. ARCs can have three possible directions. These are TO, BOTH, and FROM. TO indicates that information flows into the element through the ARC. BOTH indicates that information flows into and from the element

throught the ARC. FROM indicates that information flows from the element through the ARC. An element's POSITION is determined by counting the number of TOs, BOTHs, and FROMs that it contains and applying the following rules:

IN if TO > (BOTH and or FROM).

MID if BOTH only or TO = FROM.

OUT if FROM > (BOTH and or TO).

An object's position is determined by examining the member elements' positions and counting the number of IN elements, MID elements, and OUT elements. These counts are used to select the object's position according to the following rules:

IN if IN elements > (MID or OUT elements).

MID if no IN or OUT elements or if IN elements = OUT elements.

OUT if OUT elements > (MID of IN elements).

Objects are constructed by tracing and recording all possible non-redundant paths through the graph. A path is a list of elements that can be used to determine an object type. As paths are recorded they are analyzed to determine their type. If a path contains two elements, it is a pipe or a hanger. Then, if both of the elements are non-terminals, the object is a pipe. If one is a terminal, the object is a hanger. If the path contains two occurrences of the same element, the object is a cycle.

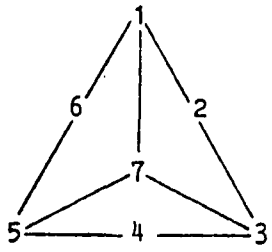
Two problems exist after all of the elements have been grouped into objects. A cycle is actually constructed of pipes which form a closed loop. Therefore, the pipes that are contained within the cycle

are redundant and must be discarded. The second problem is more complex. Cycles with more than 3 elements and 4 connections are actually formed by a combination of smaller cycles. Figure 4 shows that eight overlapping cycles exist in the dining philosophers problem.

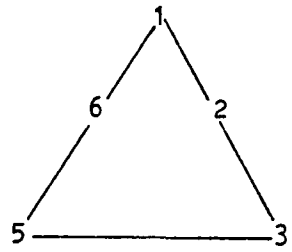
The method used to bind overlapping cycles to a common object is simple. If any two cycles have two non-terminal elements in common, they describe a common object. As cycles are linked to an object, they become the choices for ways to draw the object. Figure 5 gives an example of an object with four elements and six connections.

Figure 6 lists all of the cycles in Object A. Figure 7 shows the five possible ways to draw the object. When comparing Figure 6 to Figure 7 it can be seen that elements in Cycle 1 are the same elements drawn on the outside of the object in Picture 1. The same is true for Cycle 2 and Picture 2, Cycle 3 and Picture 3, Cycle 4 and Picture 4, and Cycle 5 and Picture 5. The list of cycles in Figure 6 form a list from which the computer or the user can select the desired way to draw the picture.

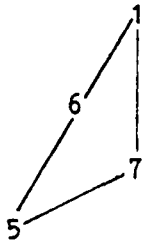
It is a simple matter to plot Objects of type HANGER or PIPE since these can be plotted one node at a time. There are two steps needed to plot a node relative to another node. The first step is to determine on which side of the key node the new node should be located. This can be done by analyzing the flow between the nodes. If the flow is from the new node to the key node then the new node should be placed on the input side of the key node. If flow is from the key node to the new node then the new node should be placed on the output side of the key node. If flow does not give a clear indication of a preferable location then the new node is placed in a middle location relative to the key node.



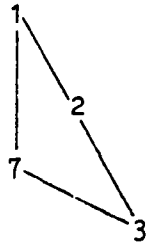
a.



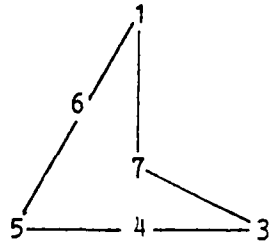
b.



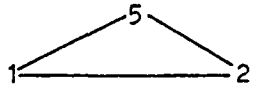
c.



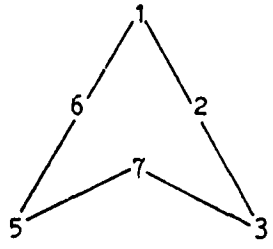
d.



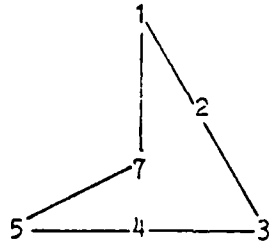
e.



f.



g.



h.

Fig. 4 Showing all cycles in dining philosophers problem

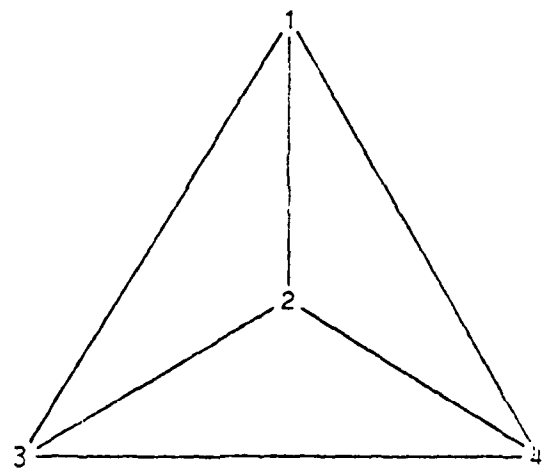
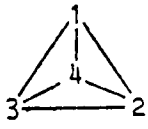


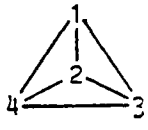
Fig. 5. Object with 4 elements and 6 connections

Cycle	Member Elements
1	1 2 3
2	2 3 4
3	1 3 4
4	1 2 4
5	1 2 3 4

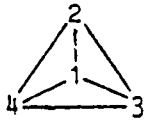
Fig. 6 The five possible cycle groups



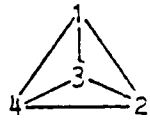
Picture 1



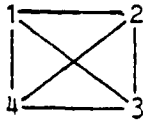
Picture 2



Picture 3



Picture 4



Picture 5

Fig. 7. The five possible ways to draw the cycle groups

The type of object in itself can also direct the placement of nodes. If the nodes are in a PIPE, they should be located linearly to one another. However, if the object is a HANGER then no clear pattern for location can be made. The plotting of a CYCLE is quite different from plotting the other shapes because three or more nodes must be located relative to one another.

A number of different methods could be used in plotting CYCLES. One such method would plot nodes one at a time, constantly shifting nodes in relation to each other until all of the nodes are plotted satisfactorily. This method would not suffer from the limitations of using only predefined patterns but it is more time consuming, more complicated, and more random in its results. Because of the limited number of nodes that the GSS is required to draw, the use of patterns seemed a logical solution. The patterns that were defined to support the GSS can be found in the appendices. An object is matched to a pattern according the number of outside and inside nodes in the selected cycle. Figure 8 shows a CYCLE with 3 outside nodes and 1 inside node. Once the pattern has been selected to draw the object, a clear space in the current picture must be found relative to the key node in which to place the object. This is done by successively overlaying the pattern on the picture until there are no collisions or overlapping nodes in the proposed pattern and the existing picture.

Depending on the shape, there can be several possible ways to connect the object to the key node and this in turn guides the placement of the pattern in the picture. Figure 9 shows the desired location of the current example depending on whether the object is of type IN, MID, or OUT. Once the object has been located in the picture, the nodes are

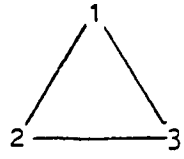


Fig. 8 A cycle with 3 outside nodes

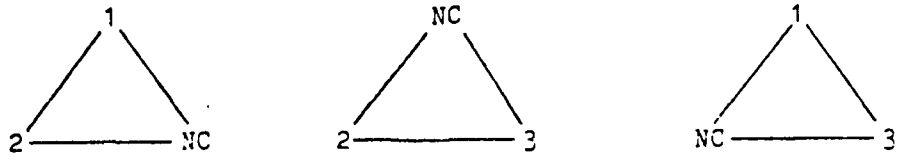


Fig. 9. Showing how cycle with 3 outside nodes can be plotted depending on its position type. NC represents the current node which has already been plotted.

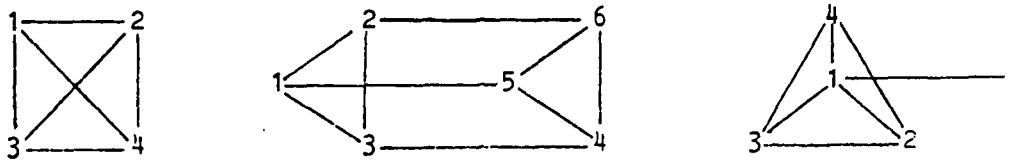
placed in the object. If possible, nodes of type IN are placed on the input side of the object and nodes of type OUT are placed on the output side of the object. Once the current object has been plotted and if more objects exist, a key node is found within the object and the next object is plotted.

After all of the elements have been positioned, the arcs and ports must be plotted. Drawing arcs is not very complicated as long as the nodes have been located correctly relative to each other, and if there are only one or two arcs between any two elements. Fortunately, most configurations have very few arcs between any two elements. Also, if the preceding heuristic is used to plot elements, any two elements will be oriented with a clear path for arcs between them.

In most cases, arcs will not have to cross one another. Several cases in which crossing arcs are unavoidable are shown in Figure 10. However, in some cases by selecting a different pattern to draw the same object, arc intersections can be reduced or eliminated. Figure 11 shows the same objects in Figure 10 redrawn using different patterns.

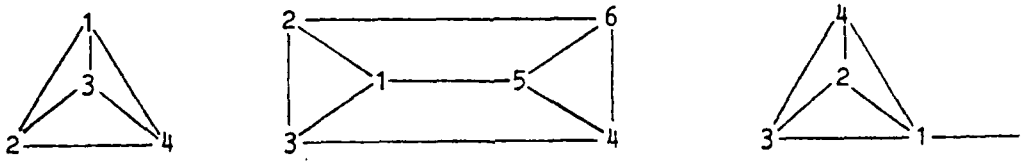
Arcs are drawn by selecting the correct free port in each element. Then the relationship between the ports is defined and the correct arc pattern is selected to guide the plotting of the arc. Finding the correct free ports and arc is a straightforward process of trying possible arc patterns until one that works is found. If two elements are oriented as in Figure 12.a, the best arc is shown in Figure 12.b.

If three elements are oriented as in Figure 13.a, the first arc choice might be Figure 13.b, the second choice might be Figure 13.c, and the last choice might be Figure 13.d.



Object 1 Object 2 Object 3

Fig. 10. Objects with crossing arcs



Object 1 Object 2 Object 3

Fig. 11. Objects redrawn without crossing arcs

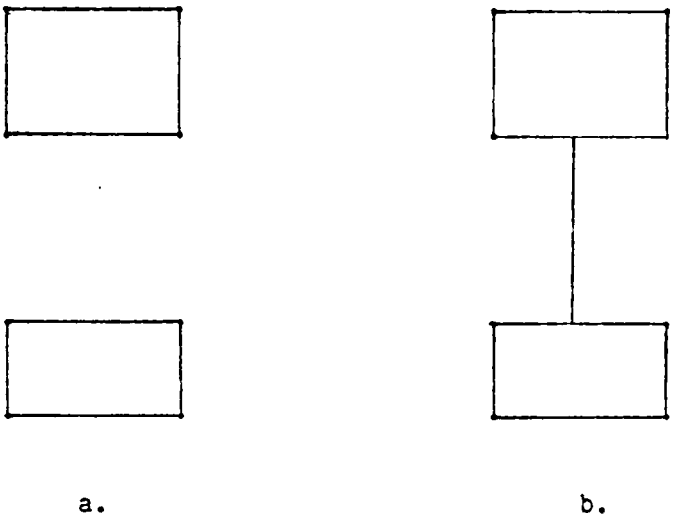


Fig. 12. Illustration of best arc between two nodes

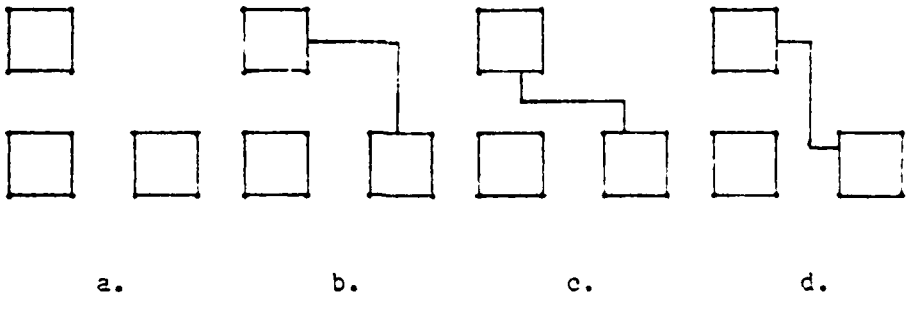


Fig. 13. Illustration of best arc for 3-node orientation

CHAPTER 2

2.1 Algorithms

The Graphics Support System can be subdivided into six submodules or classes:

1) User Interface.
2) Conversion of PCD file to a PIC file.
3) Analyze the PIC file to generate the relationships that will be used throughout the program to draw the pictures.
4) Determine object shapes, PIPE,HANGER,CYCLE.
5) Draw the picture in relative screen coordinates, independent of output device.
6) Convert the relative picture to an absolute picture destined to a specific device.

The main structure of the GSS is:

```
REPEAT
  Get a command from user.
  If valid command then call appropriate submodule.
UNTIL command = END.
```

The commands can be broken down into three subsets. The relationship between user commands and the submodules is shown in Figure 14. The first subset allows file manipulation and includes the commands FIle, EXpand, and SAve. The second subset allows picture manipulation and includes the commands GRid, SWitch, ROtate, TRanslate, and DRaw. The third subset supplies the user with information pertaining to elements or objects and allows the user to assist the computer in drawing the picture. The commands are ASsist, DAta, and HElp. The following discussion will present the algorithms that implement the

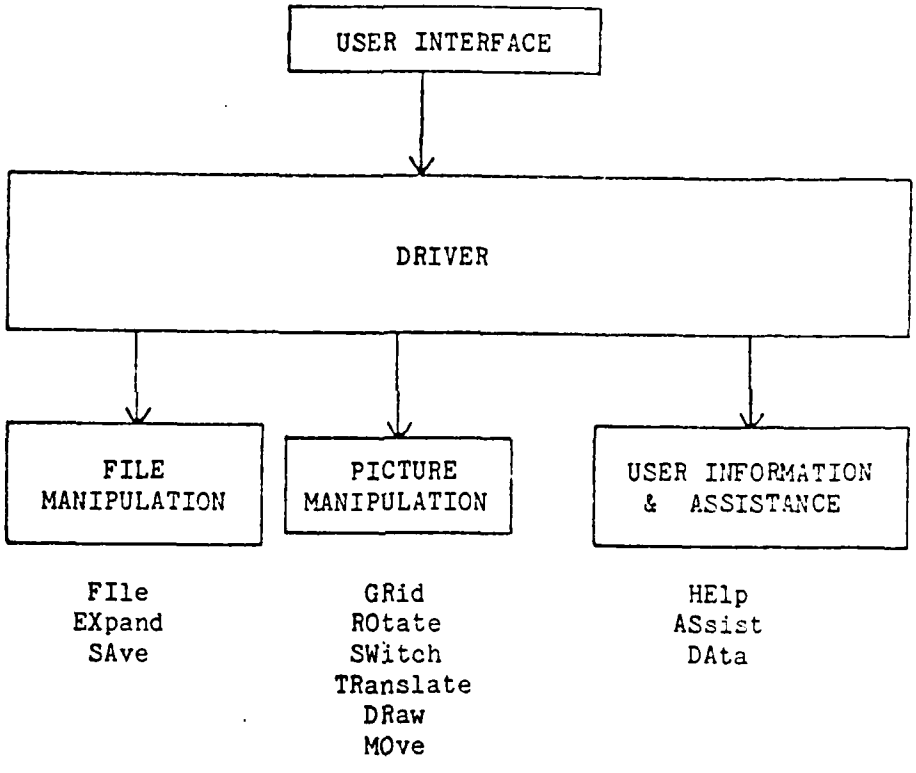


Fig. 14. The main structure of the GSS

commands in each of the subsets.

The commands in the first subset allow the user to open or close a file or to create a PIC file. The FILE command algorithm is described below:

```
CLASS FILE (name: filename)
 BEGIN
   IF name.PIC exists THEN OPEN name.PIC
                      ELSE OPEN name.PCD;
   IF file type PCD THEN CONVERT PCD file to PIC file;
 END;


CLASS EXPAND (name: element_name)
 BEGIN
   IF name <> hierarchical name THEN get hierarchical name;
   Call CLASS FILE (hierarchical name);
 END;


CLASS SAVE (name: filename)
 BEGIN
   IF name.PIC does not exist then CREATE a file name.PIC and
     WRITE (file);
 END;
```

The commands in the second subset support user manipulation a picture:

```
CLASS GRID (device: device_types)
 BEGIN
   Calculate an X and Y grid relative to a specific device;
   Add the grid to the display file;
   Call CLASS DRAW (current device);
 END;


CLASS SWITCH (element1,element2: element_name)
 BEGIN
   Switch the coordinates for element1 and element2;
   Call CLASS DRAW( current device );
 END;
```

```
CLASS ROTATE (direction: direction_types)
 BEGIN
   Switch the coordinates for a cycle in indicated direction;
   Call CLASS DRAW(current device);
 END;


CLASS TRANSLATE (direction: direction_types,unit: integer)
 BEGIN
   Shift all picture file coordinates in the indicated
     direction for the specified number of units;
   Call CLASS DRAW( current device);
 END;


CLASS DRAW (device: device_types)
 BEGIN
   Call CLASS DEFINE_OBJECTS;
   Call CLASS PLOT_NODES;
   Call CLASS DRAW_PICTURE(device);
 END;
```

The commands in the third subset provide the user with information concerning the GSS command set or pertaining to the picture organization:

```
CLASS ASSIST;
 BEGIN
   Display names of all elements.
   IF there are any cycle objects THEN ask user if he wants
     to assist in selecting shape of cycle objects;
   IF the user wants to interact
     THEN BEGIN
            List information about cycle choices;
            Ask user to make a selection of a shape;
          END
     ELSE the GSS selects the cycle choice;
 END;


CLASS DATA (name : element_name);
 BEGIN
   Print information about the element with id = name;
 END;
```

```
CLASS HELP
 BEGIN
   Print the annotated list of commands available in the GSS
 END
```

The algorithms described next support the GSS commands. They describe how the input file is transformed into objects, how the objects are located in the picture, and how the entire picture is drawn to a device.

```
CLASS DEFINE_OBJECTS;
 BEGIN
   Define connections between nodes.
   Combine nodes into objects.
   Combine common objects.
   Determine if nodes are of type IN, MID, or OUT.
   Determine if objects are of type IN, MID, or OUT.
   If any objects of type cycle exist determine all the possible
     ways to draw the object.
 END;
```

```
CLASS PLOT_NODES;
 BEGIN
   Select the first node.
   Select the first object.
   Plot the first node.
   REPEAT
     GET_NEXT_NODE in object;
     IF not another node in object
       THEN GET_NEXT_OBJECT
       ELSE PLOT_A_NODE(node);
   UNTIL no more nodes or objects to plot.
 END
```

```
CLASS GET_NEXT_NODE;
 BEGIN
   IF another node in current object
     THEN
       BEGIN
         Nodes are plotted clockwise around an object
           so the next node must satisfiy two conditions:
             1. Connect to current node.
             2. Be of the proper type of position if there
                is a choice.  This means that if the current
                node is a MID and the next position to be
                plotted in the object is on the OUT side of
                the current node, the next element to be
                selected should be of position OUT or MID
                if possible.
         Following these rules, select the next node;
       END;
 END;


CLASS GET_NEXT_OBJECT;
 BEGIN
   IF there is another object to plot THEN
     BEGIN
       Using the non_terminal external element in the current
         object select, select the next object.  If there is
         a choice of next objects always select the large one
         to do next.
     END;
 END;
```

```
CLASS SELECT_PATTERN;
 Using the current object
   BEGIN
     Select a choice if one has not been selected by the user;
     Count the number of outside and inside nodes in the
     object;
     IF the object is a cycle, count the number of outside
       and inside nodes;
     CASE object type OF
       PIPE or HANGER:  The object will be plotted using the
                          pattern for pipes and hangers;
       CYCLE :  The pattern will be chosen according to the
                  number of outside and inside elements;
     END;
     REPEAT
       CASE positon of object OF
         IN  :  Select the object orientation that is defined
                  for an input type object;
         MID :  Select the mid object orientation;
         OUT :  Select the out object orientation
       END;
       Overlay the selected pattern on the existing picture
         and check to see if there are any collisions with
         previously plotted node;
       IF a collision THEN chose another object orientation;
     UNTIL no collision OR no more patterns to try;
     IF no more patterns to try THEN error condition and
       another choice of pattern must be made;
 END;


CLASS DRAW_PICTURE(device: device_types);
 BEGIN
   Adjust limit variables to match device;
   Draw boxes;
   Draw text in boxes;
   Select arcs;
   Draw ports;
   Draw arcs;
 END;
```

2.2 Implementation

The GSS has been implemented in SEQUENTIAL PASCAL and is currently running on an Interdata 8/32. A detailed documentation of the GSS can be found in [6]. The following implementation details will discuss the heuristics that analyze directed graphs and select patterns.

2.2.1 PICTURE Data Structure

The PICTURE data structure holds all of the information that is needed by GSS to draw a picture of a configuration on any output device. Figure 15 is a drawing of a connected graph containing two cycles, A and B, and a hanger C. Figure 16 shows how the components of PICTURE relate to one another and demonstrates how they would be organized when defining the graph in Figure 15.

A picture as defined in PICTURE is composed of objects which can be of type HANGER, PIPE, or CYCLE. Each object is composed of one or more groups which can also be of type HANGER, PIPE, or CYCLE. If an object is of type HANGER or PIPE then it contains only one group. However, if the object is a CYCLE then it can contain one or more groups of type CYCLE. Each group is composed of one or more elements and multiple groups can point to the same element. If more than one object's groups point to a common element, that element is designated as being EXTERNAL and will be used to connect two objects together when drawing the picture. Each element connects to one or more ports. Ports are unique to an element and can be of two types, INTERNAL or EXTERNAL. INTERNAL ports connect to other ports within the current configuration. EXTERNAL ports, which only exist in partial configuration descriptions, do not connect to another port in the current picture and are drawn differently than INTERNAL ports.

The PICTURE record is described next. The constants used in the program are defined first. Notice that these are the values to change if any limit in the program is going to altered. For instance, if the number of nodes is to be changed, simply assign a different value to MAX_ELEMENT.

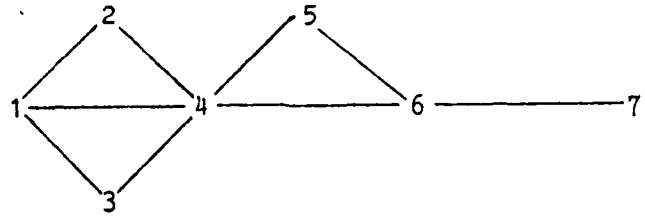


Fig. 15. Illustration of a connected graph

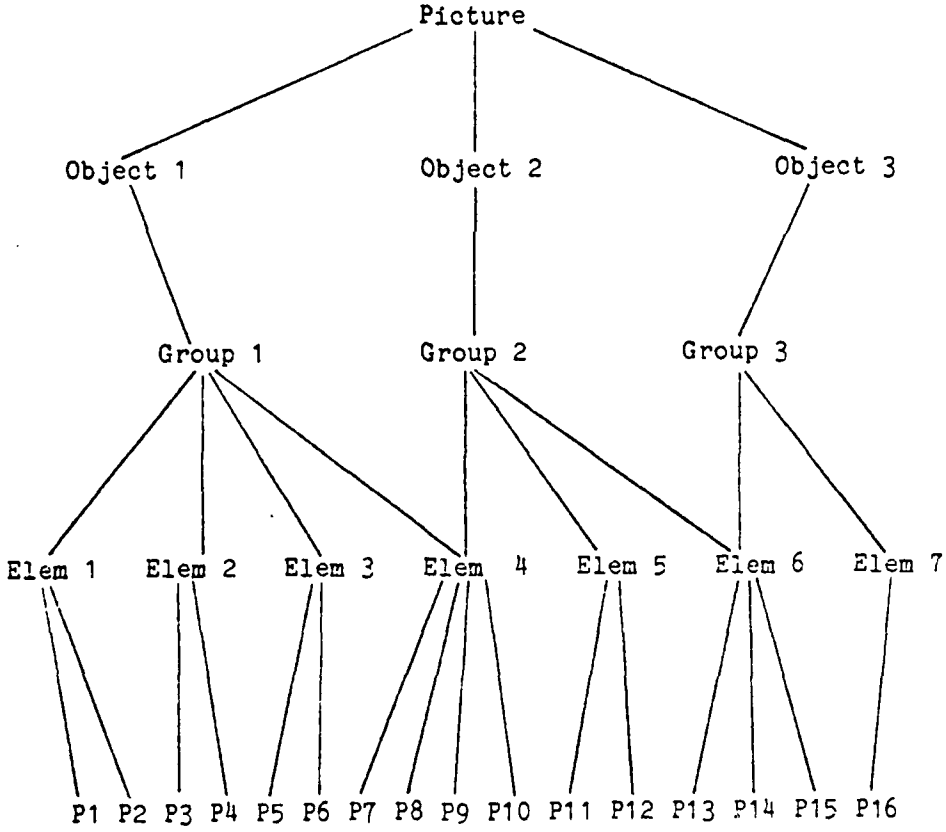


Fig. 16. Tree showing relation of picture components. P1 through P16 represent ports.

```
CONST
   MIN_ELEMENT = 1;    MAX_ELEMENT = 8;
   MIN_PORT = 1;       MAX_PORT = 32;
   MIN_ARC = 1;        MAX_ARC = 16;
   MIN_OBJECT = 1;     MAX_OBJECT = 7;
   MIN_GROUP = 1;      MAX_GROUP = 30;
   MIN_STACK = 1;      MAX_STACK = 30;
   MAX_COL = 79;
   FIRST_COL = 10;
   FIRST_ROW = 10;
   CONSOLE_MAX_WIDTH = 7;
   CONSOLE_MAX_HEIGHT = 5;
   DF_MIN_ROW = 1; DF_MAX_ROW = 21; DF_MIN_COL = 0; DF_MAX_COL = 79;
```

FIRST_ROW and FIRST_COL define where the first node will be plotted in relative space. When plotting pictures on the console, the two CONSOLE constants define the greatest number of pixel locations and how wide and high the picture may be. The four DF constants define how big the display file is when drawing pictures on the console.

The following types are used in the GSS:

```
TYPE
   STRING2 = ARRAY[1..2] OF CHAR;
   STRING8 = ARRAY[1..8] OF CHAR;
   STRING12 = ARRAY[1..12] OF CHAR;
   GROUP_INDX = MIN_GROUP..MAX_GROUP;
   ELEMENT_INDX = MIN_ELEMENT..MAX_ELEMENT;
   PORT_INDX = MIN_PORT..MAX_PORT;
   OBJECT_INDX = MIN_OBJECT..MAX_OBJECT;
   CONNECTION_TABLE_INDX = ELEMENT_INDX;
   STACK_INDX = MIN_STACK..STACK_LIMIT;
   GROUP_SET = SET OF GROUP_INDX;
   ELEMENT_SET = SET OF ELEMENT_INDX;
   PORT_SET = SET OF PORT_INDX;
   OBJECT_SET = SET OF OBJECT_INDX;
   FLOW_DEF = (TO_,FROM_,BOTH_);
   PORT_TYPE_DEF = (INTERNAL,EXTERNAL);
   POSITION_DEF = (IN_,MID_,OUT_);
```

The port record holds the information relative to each port in the configuration:

```
PORT_DEF = RECORD
             PORT_ID: STRING8;
             IN_ELEMENT : ELEMENT_INDX;
             FLOW : FLOW_DEF;
             CASE PORT_TYPE:PORT_TYPE_DEF OF
               INTERNAL : (TO_PORT: PORT_INDX);
               EXTERNAL : (EXTERNAL_NAME: STRING8);
           END;
```

PORT_ID has the name of the port. IN_ELEMENT is a pointer to the parent element. FLOW indicates the direction that information flows through the port. The PORT_TYPE indicates if the port is INTERNAL or EXTERNAL.

The definition of each element is contained in the following record:

```
ELEMENT_DEF = RECORD
                X: INTEGER;
                Y: INTEGER;
                LLC: BOOLEAN;
                IN_OBJECTS: OBJECT_SET;
                NODE_ID: STRING8;
                PORTS: PORT_SET;
                HIERARCHICAL: BOOLEAN;
                HIER_NAME : CHAR8;
                NODE_POSITION : POSITION_DEF
              END;
NODE_KIND_DEF = (TERMINAL,NON_TERMINAL);
```

X and Y hold the coordinates where the element is plotted. IN_OBJECTS is a pointer to parent objects and NODE_ID contains the name of the node. PORTS contains a set of pointers to all of the ports in the element and HIERARCHICAL specifies if the node is hierarchical and itself defines another configuration. HIER_NAME specifies the name of the hierarchy if the name is different than NODE_ID. NODE_POSITION

defines where the node should be plotted in relation to the picture. HOST_ID specifies the name of the host that the element resides in.

Connections between nodes are defined in the connection table:

```
CON_TABLE_DEF = RECORD
                  TOUCHES : ELEMENT SET;
                  NODE_KIND: NODE_KIND_DEF
                END;
KIND_DEF = (UNDEF_,HANGER_,PIPE_,CYCLE_);
```

There is an entry in the table for each element in the configuration and for each entry there is a set of elements that each element TOUCHES. An elements node kind is defined in NODE_KIND.

The description of each object is contained in the following record:

```
OBJECT_DESC = RECORD
               ALL_MEMBERS : ELEMENT_SET;
               GROUPS_NOT_TRIED: GROUP_SET;
               GROUPS_TRIED: GROUP_SET;
               OBJECT_POSITION: POSITION_DEF;
               KIND: KIND_DEF;
               EXTERNAL: ELEMENT_SET;
               CYCLE_CHOICE : NO_CHOICE..MAX_GROUP
              END;
ELEMENT_ARRAY= ARRAY[ELEMENT_INDX] OF ELEMENT_DEF;
PORT_ARRAY= ARRAY[PORT_INDX] OF PORT_DEF;
CONNECTION_TABLE_ARRAY= ARRAY[CONNECTION_TABLE_INDX]
                          OF CON_TABLE_DEF;
GROUP_ARRAY= ARRAY[GROUP_INDX] OF ELEMENT_SET;
OBJECT_ARRAY= ARRAY[OBJECT_INDX] OF OBJECT_DESC;
```

ALL_MEMBERS contains a set of pointers to all the elements in the object. GROUPS_NOT_TRIED is used in selecting an object pattern and to keep track of choices that have been tried. GROUPS_TRIED contains the set of all groups that have been tried. OBJECT_POSITION indicates the optimum relative location for the object in the picture. KIND indicates what kind of object it is, if it is a PIPE, HANGER, or CYCLE. EXTERNAL

contains the set of all EXTERNAL nodes in the object, and is used to connect objects together. CYCLE_CHOICE indicates the pattern that was last used to draw the object.

The picture description record defines all of the entities that make up the picture:

```
PICTURE_DESC = RECORD
                    ELEMENT: ELEMENT_ARRAY;
                    PORT: PORT_ARRAY;
                    CONNECTION : CONNECTION_TABLE_ARRAY;
                    GROUP: GROUP_ARRAY;
                    OBJECT: OBJECT_ARRAY;
                    SUCCESSFUL : BOOLEAN
                    NUM_NODES : INTEGER;
                    NUM_OBJECTS : OBJECT_INDX;
                    NUM_PORTS : PORT_INDX;
                    NUM_GROUPS  : GROUP_INDX;
END;
```

The SUCCESSFUL boolean is used through the program to indicate a successful plotting operation. If, at the end of the program, SUCCESSFUL is false then the picture can not be drawn for some reason that will be indicated by an error message.

The following global variables are used in the program:

```
VAR
  PICTURE : PICTURE_DESC;
 CD : UNIV_PCD_TYPE;
```

PICTURE is the data structure that holds the picture definition. CD is the record that holds the definition of a software configuration. Information in the CD is extracted to the PICTURE record if this is the first time this configuration is being drawn. After the PICTURE record has been initialized, the CD is not accessed again. A description of the UNIV_PCD_TYPE can be found in [5].

2.2.2 Graph Analysis and Object Definition

The first step in drawing a picture after the CD file has been converted is to analyze the graph to determine relationships and build objects. This is done in the class DEFINE_OBJECTS which will use several additional data structures:

```
CONST
 MIN_STACK  =  1;    STACK_LIMIT  = MAX_ELEMENT;
TYPE
 STACK_INDX = MIN_STACK..STACK_LIMIT;
 GROUP_DEF = RECORD
               KIND : KIND_DEF;
               COMMON : OBJECT_INDX;
             END;
 GROUP_DESC_DEF = ARRAY[GROUP_INDX] OF GROUP_DEF;
VAR
 ACTIVE_NODES : ARRAY[STACK_INDX] OF ELEMENT_INDX;
 TOS : STACK_INDX;
 GROUP_DESC : GROUP_DES_DEF;
 TEMP_CON_TABLE : CONNECTION_TABLE_ARRAY;
```

Figure 17 shows the procedure DEFINE_OBJECTS which analyzes the graph definition, then creates and describes objects, groups, elements, and ports.

INITIALIZE_TEMP_CON builds a duplicated copy of CONNECTION. SELECT_FIRST_NODE selects the first node to be located. It makes its selection by finding the node that connects to the most other nodes. This node is pushed on the stack. ANY_ACTIVE_NODES peeks in the stack to see if any elements are on the stack. If there are then the analysis is not complete.

The CURRENT_NODE is the one that is on the top of the stack. The NEXT_NODE is found by looking in the TEMP_CON_TABLE set for the CURRENT_NODE and pulling one out. The NEXT_NODE is then removed from the CURRENT_NODE's TEMP_CON_TABLE so that the same connection will not

```
PROCEDURE DEFINE_OBJECTS;
  VAR SAVE_CURRENT_NODE,CURRENT_NODE,
      NEXT_NODE: ELEMENT_INDX;
      DONE : BOOLEAN;
  BEGIN
   INITIALIZE_TEMP_CON;
   SELECT_FIRST_NODE;
   WHILE  (ANY_ACTIVE_NODES) DO
    BEGIN
     CURRENT_NODE := PEEK(TOP);
     SAVE_CURRENT_NODE := CURRENT_NODE;
     NEXT_NODE := NEXT(CURRENT_NODE);
     REMOVE(CURRENT_NODE,NEXT_NODE);
     CURRENT_NODE := NEXT_NODE;
     CASE NODE_TYPE(CURRENT_NODE) OF
      TERMINAL: BEGIN REMOVE(NEXT_NODE,SAVE_CURRENT_NODE);
                       HANGER(CURRENT_NODE);
                END;
       NON_TERMINAL: IF SEARCH(CURRENT_NODE) = NOT_FOUND
                     THEN
                       BEGIN
                        REMOVE(NEXT_NODE,SAVE_CURRENT_NODE);
                        PUSH(CURRENT_NODE);
                       END
                     ELSE CYCLE(CURRENT_NODE)
     END;
     DONE := FALSE;
     REPEAT
      IF ANY_ACTIVE_NODES THEN
       IF EMPTY(PEEK(TOP)) THEN
         BEGIN  REINIT(PEEK(TOP));
                IF TOP > MIN STACK THEN PIPE;
                POP;
         END  ELSE DONE := TRUE;
     UNTIL NOT(ANY_ACTIVE_NODES) OR  DONE;
    END;
   COMBINE;
END;
```

Fig. 17. The Procedure DEFINE_OBJECTS

be seen twice. Then the NEXT_NODE becomes the CURRENT_NODE.

If the CURRENT_NODE is a terminal then the previous CURRENT_NODE is removed from the CURRENT_NODE's TEMP_CON_TABLE and an object of type HANGER is created consisting of two elements, the CURRENT_NODE and the element on the top of the ACTIVE stack. If the element is a non_terminal then the first step is to search the ACTIVE stack to see if the CURRENT_NODE is already on the stack. If it is, a CYCLE group is created having the following members. The first group member is the CURRENT_NODE and the other members are found by popping elements off the stack until the element that matches the CURRENT_NODE is encountered. As each element is popped it is added to the CYCLE group. As a final step all of the elements that were popped off the stack are pushed back on the stack in the original order. The CURRENT_NODE is not pushed on the stack.

If the CURRENT_NODE was not found on the ACTIVE stack, it is pushed on the stack and the previous CURRENT_NODE is removed from the TEMP_CON_TABLE of the CURRENT_NODE. Before the loop is repeated the TEMP_CON_TABLE must be reinitialized for any elements that are on the ACTIVE stack that have had all of there connections analyzed. As they are reinitialized they are popped off the stack and PIPE objects are formed from the element on the top of the stack and the one just under it. If all of the elements get popped off of the stack then the procedure is done, if not, then a new CURRENT_NODE is found and the process begins again.

After all of the groups have been created the COMBINE procedure shown below is called:

```
PROCEDURE COMBINE;
 BEGIN
   GROUPS_TO_OBJECTS;
   ELIMINATE_REDUNDANT_CYCLES;
   MAKE_KIND;
   MAKE_GROUPS_NOT_TRIED;
   DETERMINE_EXTERNAL;
   DETERMINE_IN_OBJECTS;
   DETERMINE_POSITION;
 END;
```

GROUPS_TO_OBJECTS combines common groups together to form objects. HANGERs are not considered here because they cannot be combined. The method for combining groups is very simple. Compare every group with every other group. If any two groups contain two or more identical elements, they are placed in the same object. After all groups have been combined, the CYCLE objects will contain both CYCLE groups and PIPE groups. The PIPE groups are redundant information and must be deleted.

MAKE_ALL_MEMBERS simply makes a set of all the elements in all the groups the are contained within the object. MAKE_KIND determines the kind of object based on the type of groups found in the object. If an object contains a HANGER group the object type becomes HANGER. If an object contains a PIPE group the object type becomes PIPE. Finally, if the object contains a CYCLE group the object type becomes CYCLE.

GROUPS_NOT_TRIED is only done for CYCLE objects. It is a set of all the CYCLE groups in the object. DETERMINE_EXTERNAL is a set of all the elements in the object that are common to other objects. DETERMINE_IN_OBJECTS initializes the IN_OBJECTS variable of each element with a pointer to the object that it is in. DETERMINE_POSITION calculates the NODE_POSITION and OBJECT_POSTION for each element and

object. This is done for each element by counting the number of TO,BOTH,and FROM ports and then assigning the position according to the following rules:

IN if more FROMs than TOs.
OUT if more TOs than FROMs.
MID otherwise.

Then the object postion is determined by counting the number of IN, MID, of OUT elements and then assigning the position according to the following rules:

IN if more INs than OUTs.
OUT is more OUTs than INs.
MID otherwise.

2.2.3 Pattern Selection

The process of plotting objects has already been discussed in section 2.2 and a detailed explanation can be found in [6]. However, the process of selecting a pattern and locating elements in the picture is interesting and will be discussed next.

The method of selecting a pattern is based on the shape of the object to be plotted and can be determined using the object's kind definition. If the object is a PIPE or HANGER, the PATTERN_FOR_PIPES_AND_HANGERS is called. If the object is a cycle, a pattern for that particular shape of cycle is called. A cycle's shape is determined by the number of outside and inside elements in the cycle group that is the current choice. The method of placing an object in the picture will be discussed after the PIXEL_PAD has been defined.

The PIXEL is an entity that represents a location on a a PIXEL_PAD

shown in Figure 18. The PIXEL_PAD represents a window of a certain portion of plotted picture. The plotted picture is defined by the X and Y coordinates of elements that have already been plotted.

The PIXEL_PAD is oriented over the existing picture by placing a certain PIXEL, F5 in the case of plotting a hanger, over the current node. If the current node was in location 10,10, F5 would represent that location. Then if there is a plotted node in location 8,10, F3 would represent that location. A PIXEL can be converted into a a set of actual picture coordinates by converting the PIXEL into a row and a column as shown in Figure 18, then adding a row offset and a column offset to the row and column respectively. The offsets are calculated according to where the PIXEL_PAD is oriented over the picture. There are two types of object patterns defined for GSS. These are patterns for pipes and hangers and patterns for cycles. The two pattern types are very similar but they are used in slightly different fashons to select the next possible way to plot the pattern. The record defining a pattern is presented next:

```
PATTERN_DEF = RECORD
                LOCATION : ARRAY[1..8] OF
                             RECORD
                               LOC_PIXEL : PIXEL;
                               NEXT_OUTSIDE : INTEGER;
                               NEXT_INSIDE  : INTEGER;
                               XLOC : INTEGER;
                               YLOC : INTEGER;
                             END;
                INSERT_PTR : INTEGER;
                START_LOCATION : PIXEL;
                SPACE_NEEDED : ARRAY[1..8] OF PIXEL_SET;
                REQUIRED_SPACE : PIXEL_SET;
                NUM_LOCATIONS : INTEGER;
                LAST_CHOICE : (FIRST,SECOND,THIRD,FOURTH);
              END;
```

LOCATION is used as a circular linked list with NEXT_OUTSIDE

```
Row

+5  K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,

+4  J0,J1,J2,J3,J4,J5,J6,J7,J8,J9,J10,

+3  I0,I1,I2,I3,I4,I5,I6,I7,I8,I9,I10,

+2  H0,H1,H2,H3,H4,H5,H6,H7,H8,H9,H10,

+1  G0,G1,G2,G3,G4,G5,G6,G7,G8,G9,G10,

 0  F0,F1,F2,F3,F4,F5,F6,F7,F8,F9,F10,

-1  E0,E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,

-2  D0,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10,

-3  C0,C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,

-4  B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,

-5  AC,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10

Col  -5 -4 -3 -2 -1  0 +1 +2 +3 +4 +5
```

Fig. 18. Pixel Pad and the relationship of a pixel to a relative row and column.

pointing to the next location in the list. NEXT_OUTSIDE, as used in a the pipe-hanger pattern points to the next way to try to draw the pattern relative to the CURRENT_NODE. In a cycle, NEXT_OUTSIDE points the next node location on the perimeter of the cycle pattern. NEXT_INSIDE is not used in the pipe_hanger pattern but it points to the next node location on the inside of a cycle pattern. X_LOC and Y_LOC are coordinate locations which are calculated when a pattern orientation is selected. These locations are used in the class which plots nodes to locate nodes in the picture.

LOC_PIXEL specifies a pixel where a node in the object can be located. The INSERT_PTR points to the first location in the LOCATION list and is assigned according to the type of object position for the new object. The value in START_LOCATION specifies the pixel in the pattern which is used to orient the pattern to the current node. SPACE_NEEDED is used in the pipe-hanger pattern to designate the set of pixels that must be free when the pattern is overlayed on the picture. REQUIRED_SPACE serves the same function in the cycle patterns. LAST_CHOICE is used in the cycle patterns to designate the last valid way to try to overlay the pattern. For instance, if there are three ways to orient a certain pattern, LAST_CHOICE would be assigned THIRD.

The following example shows the values assigned to the pattern for pipes and hangers.

Figure 19 shows the pixel pad with the possible LOC_PIXELs indicated by brackets. Remember that the current node is located in the position indicated by pixel F5.

```
K1,K2,K3,K4,K5,K6,K7,K8,K9,K1
J1,J2,J3,J4,J5,J6,J7,J8,J9,J1
I1,I2,I3,I4,I5,I6,I7,I8,I9,I1
H1,H2, *,H4, *,H6, *,H8,H9,H1
G1,G2,G3,G4,G5,G6,G7,G8,G9,G1
F1,F2, *,F4,F5,F6, *,F8,F9,F1
E1,E2,E3,E4,E5,E6,E7,E8,E9,E1
D1,D2, *,D4, *,D6, *,D8,D9,D1
C1,C2,C3,C4,C5,C6,C7,C8,C9,C1
B1,B2,B3,B4,B5,B6,B7,B8,B9,B1
A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
```

Fig. 19. Shows the pixel pad with the possible LOC_PIXELs indicated by asterisks.

```
K1,K2,K3,K4,K5,K6,K7,K8,K9,K1
J1,J2,J3,J4,J5,J6,J7,J8,J9,J1
I1,I2,I3,I4,I5,I6,I7,I8,I9,I1
H1,H2,H3,H4, *,H6,H7,H8,H9,H1
G1,G2,G3,G4,G5,G6,G7,G8,G9,G1
F1,F2,F3,F4, *,F6,F7,F8,F9,F1
E1,E2,E3,E4,E5,E6,E7,E8,E9,E1
D1.D2, *,D4,D5,D6, *,D8,D9,D1
C1,C2,C3,C4,C5,C6,C7,C8,C9,C1
B1,B2,B3,B4,B5,B6,B7,B8,B9,B1
A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
```

Fig. 20. Shows pixel pad with the possible LOC_PIXELs indicated by asterisks.

The following code shows how the PATTERN_DEF record is initialized when doing the pipe-hanger pattern:

```
LOCATION[1].LOC_PIXEL := F3; LOCATION[1].NEXT_OUTSIDE := 2;
LOCATION[2].LOC_PIXEL := H3; LOCATION[2].NEXT_OUTSIDE := 3;
LOCATION[3].LOC_PIXEL := H5; LOCATION[3].NEXT_OUTSIDE := 4;
LOCATION[4].LOC_PIXEL := H7; LOCATION[4].NEXT_OUTSIDE := 5;
LOCATION[5].LOC_PIXEL := F7; LOCATION[5].NEXT_OUTSIDE := 6;
LOCATION[6].LOC_PIXEL := D7; LOCATION[6].NEXT_OUTSIDE := 7;
LOCATION[7].LOC_PIXEL := D5; LOCATION[7].NEXT_OUTSIDE := 8;
LOCATION[8].LOC_PIXEL := D3; LOCATION[8].NEXT_OUTSIDE := 1;
CASE OBJECT_POSITION OF
  IN  : INSERT_PTR := 1;
  MID_: INSERT_PTR := 5;
  OUT_: INSERT_PTR := 5
END;
START_LOCATION := F5;
SPACE_NEEDED[1] := [G2,G3,F2,F3,E2,E3];
SPACE_NEEDED[2] := [I2,I3,I4,H2,H3,H4,G2,G3];
SPACE_NEEDED[3] := [I4,I5,I6,H4,H5,H6];
SPACE_NEEDED[4] := [I6,I7,I8,H6,H7,H8,G7,G8];
SPACE_NEEDED[5] := [G7,G8,F7,F8,E7,E8];
SPACE_NEEDED[6] := [E7,E8,D6,D7,D8,C6,C7,C8];
SPACE_NEEDED[7] := [D4,D5,D6,C4,C5,C6];
SPACE_NEEDED[8] := [E2,E3,D2,D3,D4,C2,C3,C4];
```

The INSERT_PTR is initialized as a pointer to the first possible position in which to locate the node. In the case of a pipe of position IN, the first location choice would place the node in location F3. If F3 already contains an element, then the NEXT_OUTSIDE pointer is followed to the next choice, H3 in the current example. Successive locations are tried until an empty location is found.

The next example shows the values assigned to the pattern for a cycle with three outside nodes and one inside node. Figure 20 shows the pixel pad for this pattern with the possible LOC_PIXELS indicated by asterisks.

The following code shows how the PATTERN_DEF record is initialized when doing the pattern for a cycle with three outside nodes and one inside node:

```
WITH LOCATION[1] DO BEGIN
 LOC_PIXEL := D3;  NEXT_OUTSIDE := 2;  NEXT_INSIDE := 4; END;
WITH LOCATION[2] DO  BEGIN
 LOC_PIXEL := H5;   NEXT_OUTSIDE := 3;  NEXT_INSIDE := 4; END;
WITH LOCATION[3] DO  BEGIN
 LOC_PIXEL := D7 ;  NEXT_OUTSIDE := 1;  NEXT_INSIDE := 4; END;
WITH LOCATION[4] DO  LOC_PIXEL := F5;
CASE CHOICE OF
  FIRST_: IF OBJECT_POSITION = OUT_ THEN INSERT_PTR := 1
                                    ELSE INSERT_PTR := 3;
  SECOND_:INSERT_PTR := 2;
  THIRD_ :IF OBJECT_POSITION = OUT_ THEN INSERT_PTR := 3
                                    ELSE INSERT_PTR := 1;
END;
REQUIRED_SPACE := [C2,C3,C4,D2,D3,D4,D5,D6,D7] +
                  [D8,E2,E3,E4,E5,E6,E7,E8,C6] +
                  [C7,C8,F4,F5,F6,I4,I5,I6,G4] +
                  [G5,G6,H4,H5,H6];
CASE INSERT_PTR OF
  1: REQUIRED_SPACE := REQUIRED_SPACE -[D3];
  2: REQUIRED_SPACE := REQUIRED_SPACE - [H5];
  3: REQUIRED_SPACE := REQUIRED_SPACE - [D7]
END;
LAST_CHOICE := THIRD_;
CASE INSERT_PTR OF
  1: START_LOCATION:= D3;
  2: START_LOCATION:= H5;
  3: START_LOCATION:= D7
END;
```

The INSERT_PTR is initialized as a pointer to the first way to position the object relative to the current node. If this is the FIRST choice and the object position is OUT, the INSERT_PTR would be set to one. The INSERT_PTR now points to the node location which will serve as the START_LOCATION. This is the location which already contains the current node. REQUIRED_SPACE is initialized to hold the set of pixels which must be free if the object is to be plotted using this orientation of the object. If the FIRST choice is not possible, the SECOND choice

is tried and so on. Figure 21 shows the current node plotted in location 10,10. The FIRST, SECOND, and THIRD choices for locating the current pattern relative to the current node are shown in Figures 22, 23. and 24 respectively.

The only record of where an element has been plotted is kept in a record that describes each element. The location is recorded as an X coordinate and an Y coordinate.

The first step in selecting a pattern involves creating a set of locations where nodes have already been plotted. This set is constructed by converting the X,Y coordinates for each plotted node into a pixel location in the pixel pad. However, before this is done, a prospective pattern must be selected according to the description of the new object to be plotted. If it is a pipe, the pattern for pipes and hangers is created. If it is a cycle with three nodes on the outside and one node on the inside then the pattern for that shape is created. Patterns are created by initializing the PATTERN_DEF record for a certain shape.

There are two reasons for using patterns to position objects. First, a pattern provides a way to define an object shape and provides a way to position the nodes within the shape. Second, the pattern contains a list of locations or pixels in the current picture which must be empty if the shape is to be plotted in that space.

Pixels are relative locations within the pixel pad. The pixel pad is a window which can be moved about over the existing picture to give a snapshot of the nodes which have been plotted in the windowed part of the picture. The window is positioned according to the START_LOCATION pixel initialized in the current pattern. In the current example for a

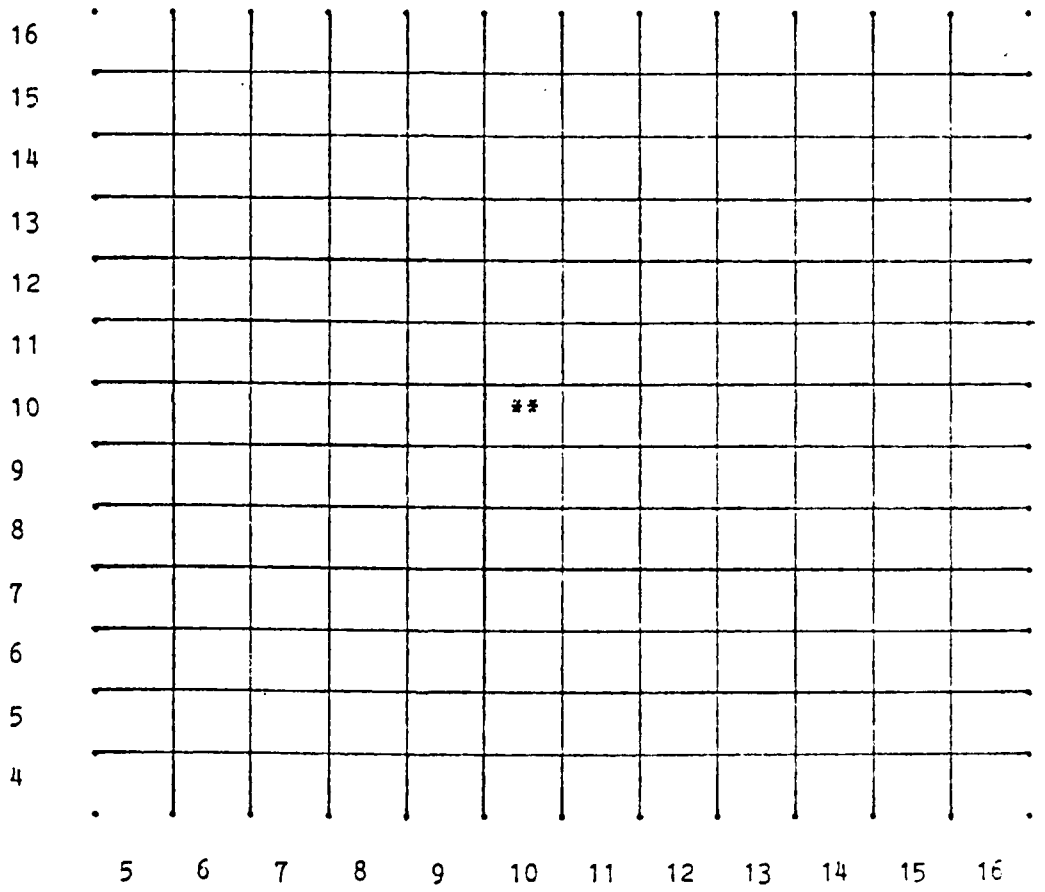


Fig. 21. Snapshot of current relative picture showing current node in location 10,10.
plotted.

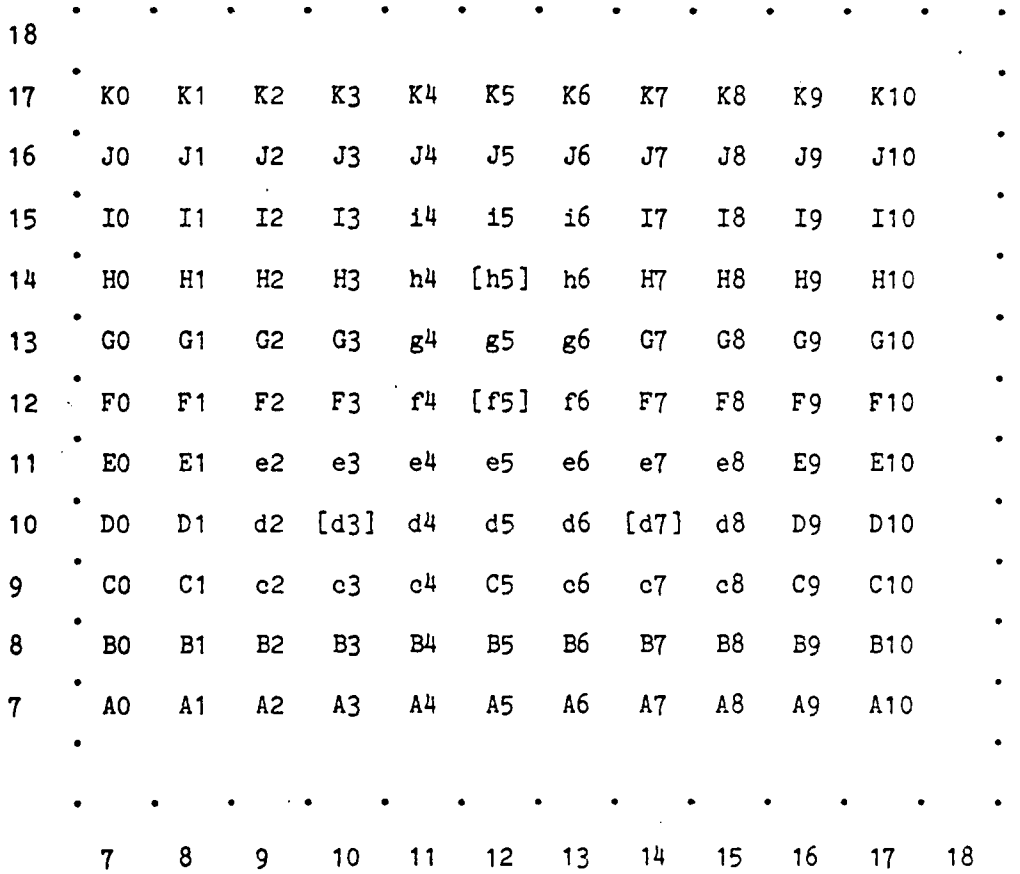


Fig. 22. Snapshot of pixel pad placed over portion of current relative picture. Node 1 is the current node and is located under pixel F7. A pair of brackets represent a prospective node location in the pattern. A lower case letter represents a pixel that must not contain a plotted node.

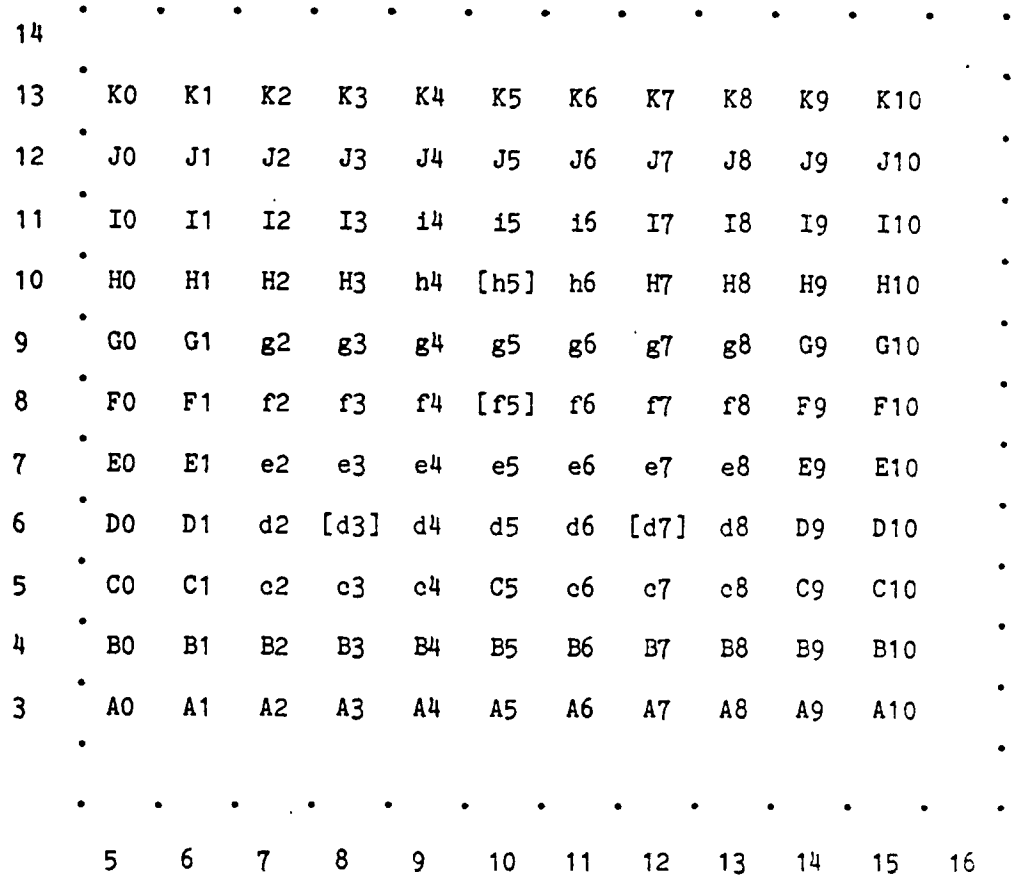


Fig. 23. Snapshot of pixel pad placed over portion of current relative picture. Node 1 is the current node and is located under pixel F7. A pair of brackets around a pixel indicates a prospective node location in the pattern. Lower case letters indicate pixels that must not contain a plotted node.

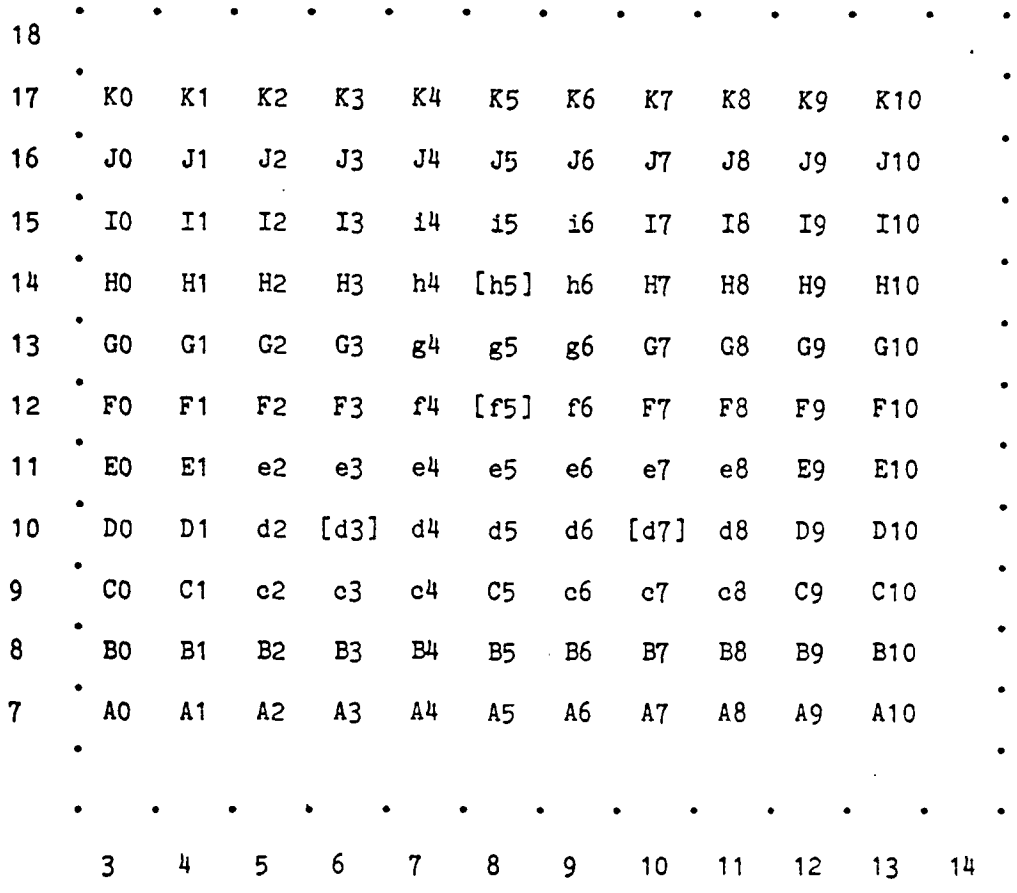


Fig. 24. Snapshot of pixel pad placed over portion of current relative picture. Node 1 is the current node and is located under pixel F7. A pair of brackets around a pixel indicate a prospective node location in the pattern. A lower case letter represents a pixel that must not contain a plotted node.

cycle, the first START_LOCATION is D3. Therefore, if the current node has been plotted in X = 10 and Y = 10, the window is positioned so that D3 rests over picture coordinates 10,10.

Then the locations of all the plotted elements that lay within the window are converted to pixels and stored in a COLLISION_SET of type pixel_set.

Once the COLLISION_SET is made it is compared to the REQUIRED_SPACE set for the pattern. If a pixel is in both sets there is a collision and the current choice for orientating the pattern cannot be used. If there is another choice of a way to lay out the pattern then that way is tried, otherwise the picture cannot be drawn using this pattern.

After a pattern orientation has been found, the locations for inserting nodes within the pattern are calculated by converting the value in each LOC_PIXEL into an X and Y coordinate and storing these coordinates in the respective XLOC and YLOC locations.

2.2.4 Locating a Node in a Pattern

The implementation of PLOT_A_NODE is now described. After an object has been selected to be plotted and a pattern has been chosen to guide the placement of nodes in the picture, PLOT_A_NODE is called repeatedly to plot each node in the object. Its purpose is to place the current node in the next empty location in the chosen pattern. Remember that the nodes in a cycle are always plotted in a clockwise manner around the outside of the object. Then the inside nodes are plotted in a clockwise direction around the inside of the object.

The method for placing the next node is very simple. The next

empty location in the pattern is indicated by the X_LOC and Y_LOC values pointed to by the NEXT_OUTSIDE value of the location where the last node was plotted. All that needs to be done to plot a node is to copy the X_LOC and Y_LOC values for the next location in the pattern into the X and Y coordinates location for the newly plotted node. If all of the outside locations are full, the next node being plotted is an inside node and its location is found by following the NEXT_INSIDE pointer rather than the NEXT_OUTSIDE pointer.

2.3 Adaptability

The GSS is constructed so that it can be easily adapted to define and add new object and arc patterns, and to change the limits for numbers of nodes, ports, and pixels.

Adding a new object shape to GSS requires the definition of a new pattern. To define a pattern the user calculates the relationship between each element location in the object. Then the new pattern is added to the existing set of possible patterns.

Adding a new arc shape to GSS can be done by building a set of subarcs that construct the arc. Then the new arc is added to the existing set of possible arcs.

The number of nodes or ports allowed in the picture can be increased or decreased by simply changing a MAX_NODES or MAX_PORTS constant. The number of nodes is limited only by the display resolution. The number of ports is limited by the size of an element. The maximum number of pixels in the display influences the organization of the elements in the final picture. Changing the number of pixels, therefore, changes the shape of the picture.

CHAPTER 3

3.1 Summary

A method for automatically drawing general directed graphs of communicating sequential programs has been presented. The primary reason for developing the heuristic was to create a user tool, called the Graphics Support System, for describing and documenting software configurations. These are graphs of nodes of processes in a message based system. A drawing of a configuration contains three main elements: nodes representing processes, arcs representing Data Transfer Streams which interconnect nodes, and ports where arcs connect to nodes.

The GSS was designed to display a configuration in an uncomplicated, meaningful way. This led to the development of a hierarchical picture definition which allowed the user to display a complete configuration or to expand a configuration node to display a partial configuration.

A heuristic for converting a graph description into a set of objects that form a picture is the major contribution of this document. The heuristic combines related nodes to form objects, then defines the different shapes that can be used to draw each object. After a shape for each object has been selected by the computer or the user, the picture is drawn by placing each object in the picture at the proper relative position. Once the objects have been arranged properly, the

GSS will automatically draw all of the arcs, ports, and text. Finally, the picture is displayed on the selected device and the user can save the picture by creating a PIC file.

The heuristic has been implemented and will draw a picture of a complete or partial configuration containing eight nodes on a variety of output devices.

3.3 Performance

The GSS has been implemented in about 3000 lines of PASCAL code. The algorithm generally takes less than one second of computing time and several seconds to a minute to plot a picture, depending on the output device. The use of hierarchical picture definitions produces pictures of the correct complexity. Drawing graphs of more than eight nodes increases the complexity of pictures and puts a limit on the types of output devices. A dumb terminal simply does not have the resolution to draw more that eight nodes. In comparing devices, the dumb terminal is least suitable and a high resolution color terminal is most suitable because it can reduce the graphic complexity.

The heuristic has no trouble in analyzing graphs and there is hardly a limit to the size of graphs that can be manipulated. However, the GSS connot draw graphs that contain objects that do not have a predefined set of possible shapes. Therefore, the user must know what shapes prospective graphs contain and add them to the GSS as they are need. Adding a new shape only requires defining an additional pattern. New arc shapes can be added by defining additinal arc patterns.

The GSS always draws graphs with the fewest crossing arcs, unless the user switches nodes around before the picture is drawn. It is very

difficult to always pick the arc that looks best. Also, it will blow up if no pattern exists for an object, or if there are too many arcs between two nodes.

3.4 Proposed Enhancements

The heuristic described in this document could be extended to draw graphs of things other than software configurations by writing a frontend conversion routine.

The GSS could be modified to build software configurations using a graphics CRT and a light pen. The only change would involve writing a routine that inserts into the PICTURE record the node connections as they are input by the user.

GSS can be modified to calculate the longest pipe in the graph. This can assist the computer in constructing the picture. Figure 25 shows how a a six node pipe is usually drawn. Figure 26 shows how the same graph could be draw if the length of the pipe is known and used to split the pipe in half.

If graphs are to be drawn that have large or complicated cycles, a possible addition to the GSS would allow it to draw cycles that it does not have predefined patterns for. However, this would require a more sophisticated heuristic and more computing time.

The ability to draw three dimensional graphs could be explored. A cycle with four or more elements and six or more connections could be drawn in 3-space and, in some cases, could present information more clearly than if the graph were drawn in 2-space. A very simple example is shown in Figure 27.

From an aesthetic point of view, it would be nice to be able to

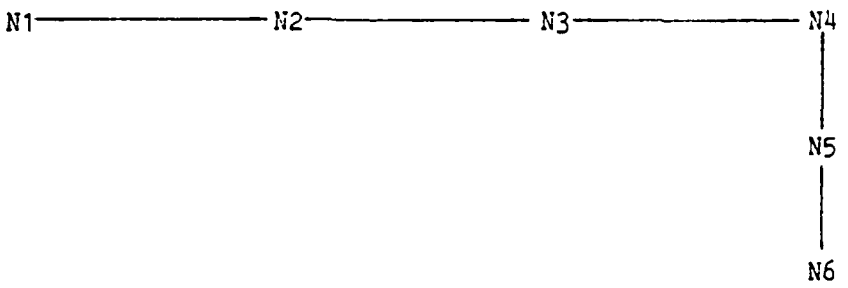


Fig. 25 This is the way the GSS draws 6 node pipe

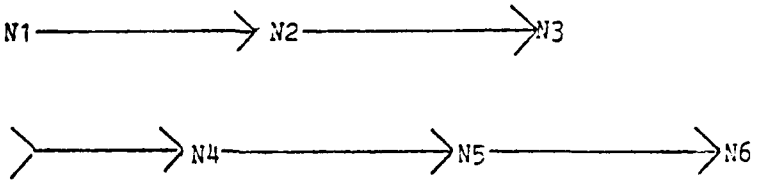
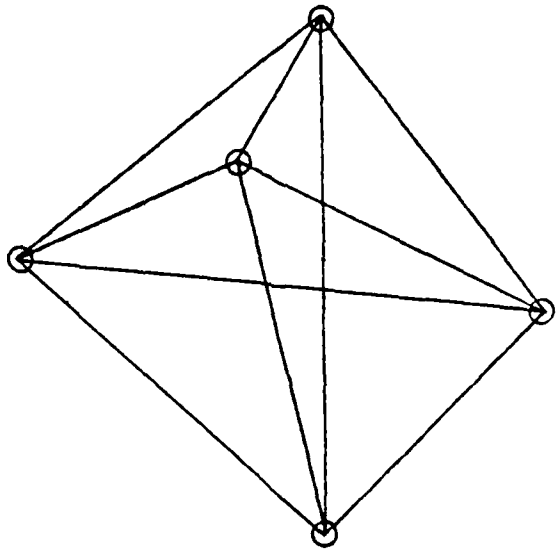
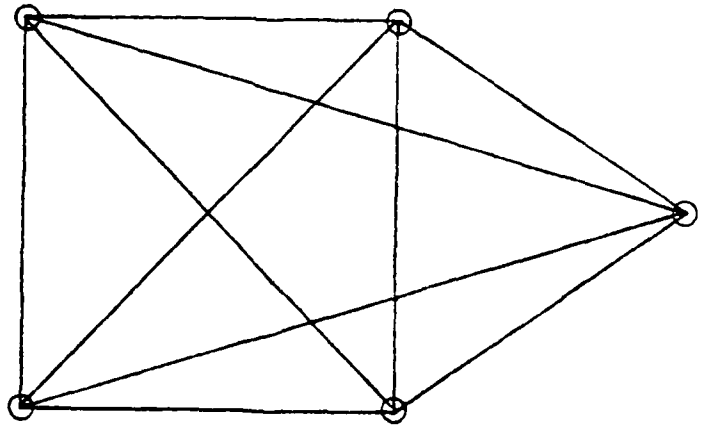


Fig. 26 This a clearer way to draw a 6 node pipe

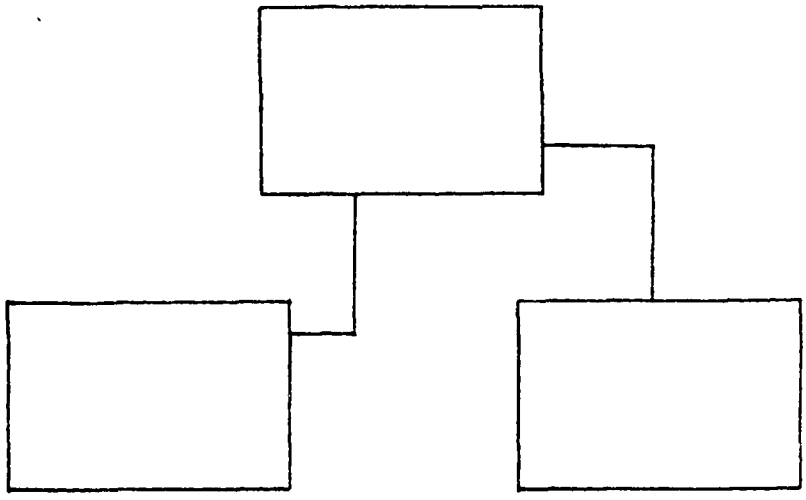


Graph drawn in 3-Space

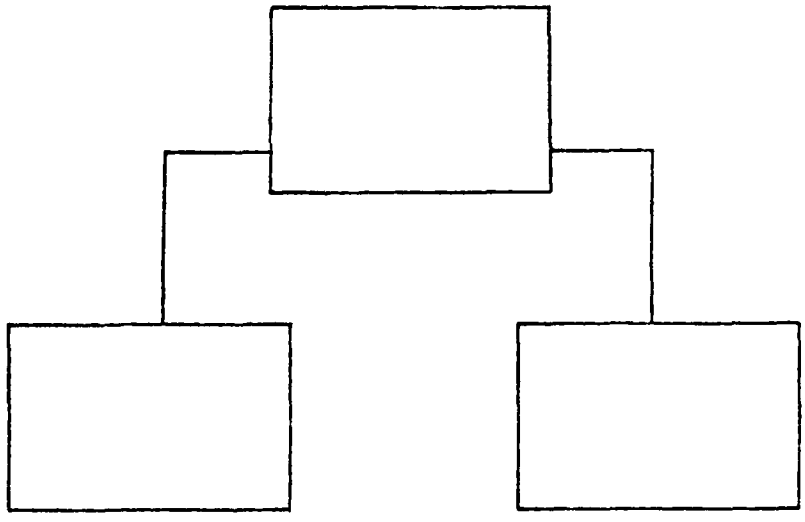


Graph drawn in 2-Space

Fig. 27. Graphs could be drawn in 3-space



Original Drawing



Modified Drawing

Fig. 28. Illustration showing modified area

alter drawn arc patterns as shown in Figure 28.

The proplem of having the GSS blow up if there are too many arcs between two nodes can be solved by defining a hierarchical arc structure. When the number of arcs between two nodes becomes critical, they are combined into a single arc and a description of the hierarchical arc is supplied elsewhere in the picture or on the user's console.

APPENDICES

A.1 Simulated Terminal Session

A sample session using the "Graphics Support System":

```
enter GSS                          "GSS is a CSS file that brings up the
                                    Graphics Support System"

GRAPHICS SUPPORT SYSTEM
*****************************
*  GRAPHICS SUPPORT SYSTEM *
*****************************

FOR ASSISTANCE TYPE                "HElp will list all the possible commands
                                    that may be used in GSS"

ENTER COMMAND ->
  enter FI FOURNODES               "The FIle command will cause GSS to search
                                     for the file UNIX.PCD.  If it is found
                                     it will open the file, read it, and
                                     determine relations.  If a PIC file does
                                     not exist then it will search for a PCD
                                     file"

ENTER COMMAND ->
  enter AS                         "ASsist will list the names of all
                                    elements and indicate if any of
                                    elements are hierarchical.  If any CYCLEs
                                    exist then data that can be used to help
                                    select a desired pattern can be entered"

NODE    NAME       HIER
1       ONE         NO
2       TWO         NO
3       THREE       NO
4       FOUR        YES

NO CYCLES EXIST                    "This will be displayed if there are no
                                    cycles"
CYCLES EXIST  DO YOU WISH TO MAKE A CHOICE?
                                   "This will be displayed if there are
                                    cycles"
enter NO                           "If the computer is to select the shape of
                                    the object.
enter YES                          "If the user wants to assist the computer
                                    in selecting the proper shape."
```

```
If YES then:

CYCLE   MEMBERS   CHOICE   OUTSIDE   INSIDE   TRIED   EXTERNAL
 1       1234       1       123         4               3
                    2       134         2               3
                    3       1234                        3
                    4       234         1               3
                    5       124         3               3

                                "This picture contains one CYCLE that
                                 has 4 elements.  It can be drawn 5
                                 different ways.  Choices 1,2,4, and 5
                                 have 3 nodes on the outside with one
                                 on the inside.  While choice 3 has 4
                                 elements on the outside and no center
                                 element.  The EXTERNAL field indicates
                                 that this cycle connects to another
                                 object
                                 through element 3 and thus CHOICE 5 might
                                 not be a good choice.  A much clearer
                                 picture can be drawn if EXTERNAL elements
                                 are located on the outside."
ENTER CYCLE ->
 enter 1                        "Select cycle number 1"
ENTER CHOICE ->
 enter 2                        "The cycle will be drawn with elements
                                 1,3, and 4 on the outside and 2 on the
                                 inside"
DO YOU WISH TO MAKE A CHOICE?  "If there is more than one cycle then
                                additional choices can be made."
 enter NO

ENTER COMMAND ->
 enter DA ONE                   "DAta will display information relating
                                 to the element ONE."

NODE_NAME IS ONE
PORT_ID   DIRECTION    EXTERNAL   TO_NODE   TO_PORT
 P2         FROM                    TWO       P1
 P3         TO                     THREE      P1
 P4         BOTH        MACCO       FOUR      P1

ENTER COMMAND ->
 enter DR CONSOLE               "DR will draw the picture on the
                                 console device."

ENTER COMMAND ->
 enter SW ONE,THREE             "SWitch will switch the locations of
                                 the elements ONE and THREE then
                                 redraw the picture"
```

```
ENTER COMMAND ->
 enter TR LEFT 2          "TRanslate will shift the entire picture
                           2 pixels to the left."

ENTER COMMAND ->
 enter SA DODA            "SAve will take the current picture and
                           save it in a new file called DODA.PIC."

ENTER COMMAND ->
 enter EX FOUR            "EXpand will make certain that FOUR is
                           a hierarchical element then perform a
                            FI on the file."

ENTER COMMAND ->
enter EN                  "ENd will terminate the session."
```

A.2 Patterns

The following pages contain the object patterns that have been implemented in the GSS. The patterns are shown imbedded in the pixel pad since this is the way that they are used when locating a shape. The LOC_PIXELs are shown surrounded with brackets. In all of the pipe-hanger patterns the key node is F5. However, this is not true in most of the cycle patterns. The pixels that are in the collision set are shown in lower case letters.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
H0	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
G0	G1	g2	g3	g4	g5	g6	G7	G8	G9	G10
F0	F1	f2	[f3]	f4	[f5]	f6	F7	F8	F9	F10
E0	E1	e2	e3	e4	e5	e6	E7	E8	E9	E10
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 29. Pattern for pipes and hangers. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	i2	i3	i4	I5	I6	I7	I8	I9	I10
H0	H1	h2	[h3]	h4	H5	H6	H7	H8	H9	H10
G0	G1	g2	g3	g4	g5	g6	G7	G8	G9	G10
F0	F1	F2	F3	f4	[f5]	f6	F7	F8	F9	F10
E0	E1	E2	E3	e4	e5	e6	E7	E8	E9	E10
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 30. Pattern for pipes and Hangers. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	I2	I3	i4	i5	i6	I7	I8	I9	I10
H0	H1	H2	H3	h4	[h5]	h6	H7	H8	H9	H10
G0	G1	G2	G3	g4	g5	g6	g7	g8	G9	G10
F0	F1	F2	F3	f4	[f5]	f6	F7	F8	F9	F10
E0	E1	E2	E3	e4	e5	e6	E7	E8	E9	E10
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 31. Pattern for pipes and Hangers. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	I2	I3	I4	I5	i6	i7	i8	I9	I10
H0	H1	H2	H3	H4	H5	h6	[h7]	h8	H9	H10
G0	G1	G2	G3	g4	g5	g6	g7	g8	G9	G10
F0	F1	F2	F3	f4	[f5]	f6	F7	F8	F9	F10
E0	E1	E2	E3	e4	e5	e6	E7	E8	E9	E10
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 32. Pattern for pipes and Hangers. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
H0	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
G0	G1	G2	G3	g4	g5	g6	g7	g8	G9	G10
F0	F1	F2	F3	f4	[f5]	f6	[f7]	f8	F9	F10
E0	E1	E2	E3	e4	e5	e6	e7	e8	E9	E10
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 33. Pattern for pipes and Hangers. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
H0	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
G0	G1	G2	G3	g4	g5	g6	G7	G8	G9	G10
F0	F1	F2	F3	f4	[f5]	f6	F7	F8	F9	F10
E0	E1	E2	E3	e4	e5	e6	e7	e8	E9	E10
D0	D1	D2	D3	D4	D5	d6	[d7]	d8	D9	D10
C0	C1	C2	C3	C4	C5	c6	c7	c8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 34. Pattern for pipes and Hangers. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
H0	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
G0	G1	G2	G3	g4	g5	g6	G7	G8	G9	G10
F0	F1	F2	F3	f4	[f5]	f6	F7	F8	F9	F10
E0	E1	E2	E3	e4	e5	e6	E7	E8	E9	E10
D0	D1	D2	D3	d4	[d5]	d6	D7	D8	D9	D10
C0	C1	C2	C3	c4	c5	c6	C7	C8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 35. Pattern for pipes and Hangers. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
H0	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
G0	G1	G2	G3	g4	g5	g6	G7	G8	G9	G10
F0	F1	F2	F3	f4	[f5]	f6	F7	F8	F9	F10
E0	E1	e2	e3	e4	e5	e6	E7	E8	E9	E10
D0	D1	d2	[d3]	d4	D5	D6	D7	D8	D9	D10
C0	C1	c2	c3	c4	C5	C6	C7	C8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 36. Pattern for pipes and Hangers. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
H0	H1	H2	H3	h4	h5	h6	H7	H8	H9	H10
G0	G1	G2	G3	g4	[g5]	g6	G7	G8	G9	G10
F0	F1	F2	f3	f4	f5	f6	f7	F8	F9	F10
E0	E1	E2	e3	[e4]	e5	[e6]	e7	E8	E9	E10
D0	D1	D2	d3	d4	d5	d6	d7	D8	D9	D10
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 37. Pattern for cycle with three outside nodes and no inside node. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	I2	I3	i4	i5	i6	I7	I8	I9	I10
H0	H1	H2	H3	h4	[h5]	h6	H7	H8	H9	H10
G0	G1	G2	G3	g4	g5	g6	G7	G8	G9	G10
F0	F1	F2	F3	f4	[f5]	f6	F7	F8	F9	F10
E0	E1	e2	e3	e4	e5	e6	e7	e8	E9	E10
D0	D1	d2	[d3]	d4	d5	d6	[d7]	d8	D9	D10
C0	C1	c2	c3	c4	C5	c6	c7	c8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 38. Pattern for cycle with three outside nodes and one inside node. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
H0	H1	H2	h3	h4	h5	h6	h7	H8	H9	H10
G0	G1	G2	g3	[g4]	g5	[g6]	g7	G8	G9	G10
F0	F1	F2	f3	f4	f5	f6	f7	F8	F9	F10
E0	E1	E2	e3	[e4]	e5	[e6]	e7	E8	E9	E10
D0	D1	D2	d3	d4	d5	d6	d7	D8	D9	D10
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 39. Pattern for cycle with four outside nodes and no inside node. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	i2	i3	i4	I5	i6	i7	i8	I9	I10
H0	H1	h2	[h3]	h4	h5	h6	[h7]	h8	H9	H10
G0	G1	g2	g3	g4	g5	g6	g7	g8	G9	G10
F0	F1	f2	f3	f4	[f5]	f6	f7	f8	F9	F10
E0	E1	e2	e3	e4	e5	e6	e7	e8	E9	E10
D0	D1	d2	[d3]	d4	d5	d6	[d7]	d8	D9	D10
C0	C1	c2	c3	c4	C5	c6	c7	c8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 40. Pattern for cycle with four outside nodes and one inside node. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixele that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	i1	i2	i3	I4	I5	I6	i7	i8	i9	I10
H0	h1	[h2]	h3	h4	h5	h6	h7	[h8]	h9	H10
G0	g1	g2	g3	g4	g5	g6	g7	g8	g9	G10
F0	f1	f2	f3	[f4]	f5	[f6]	f7	f8	f9	F10
E0	e1	e2	e3	e4	e5	e6	e7	e8	e9	E10
D0	d1	[d2]	d3	d4	d5	d6	d7	[d8]	d9	D10
C0	c1	c2	c3	C4	C5	C6	c7	c8	c9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 41. Pattern for cycle with four outside nodes and two inside nodes. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	I2	I3	i4	i5	i6	I7	I8	I9	I10
H0	H1	H2	H3	h4	[h5]	h6	H7	H8	H9	H10
G0	G1	g2	g3	g4	g5	g6	g7	g8	G9	G10
F0	F1	f2	[f3]	f4	f5	f6	[f7]	f8	F9	F10
E0	E1	e2	e3	e4	e5	e6	e7	e8	E9	E10
D0	D1	d2	[d3]	d4	d5	d6	[d7]	d8	D9	D10
C0	C1	c2	c3	c4	C5	c6	c7	c8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 42. Pattern for cycle with five outside nodes and no inside node. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	I2	I3	i4	i5	i6	I7	I8	I9	I10
H0	H1	h2	h3	h4	[h5]	h6	h7	h8	H9	H10
G0	G1	g2	[g3]	g4	g5	g6	[g7]	g8	G9	G10
F0	F1	f2	f3	f4	f5	f6	f7	f8	F9	F10
E0	E1	e2	[e3]	e4	e5	e6	[e7]	e8	E9	E10
D0	D1	d2	d3	d4	[d5]	d6	d7	d8	D9	D10
C0	C1	C2	C3	c4	c5	c6	C7	C8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 43. Pattern for cycle with six outside nodes and no inside node. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	I2	I3	i4	i5	i6	I7	I8	I9	I10
H0	H1	h2	h3	h4	[h5]	h6	h7	h8	H9	H10
G0	G1	g2	[g3]	g4	g5	g6	[g7]	g8	G9	G10
F0	F1	f2	f3	f4	[f5]	f6	f7	f8	F9	F10
E0	E1	e2	[e3]	e4	e5	e6	[e7]	e8	E9	E10
D0	D1	d2	d3	d4	[d5]	d6	d7	d8	D9	D10
C0	C1	C2	C3	c4	c5	c6	C7	C8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 44. Pattern for cycle with six outside nodes and one inside node. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	i2	i3	i4	I5	i6	i7	i8	I9	I10
H0	H1	h2	[h3]	h4	h5	h6	[h7]	h8	H9	H10
G0	G1	g2	g3	g4	g5	g6	g7	g8	G9	G10
F0	F1	f2	[f3]	f4	f5	f6	[f7]	f8	F9	F10
E0	E1	e2	e3	e4	e5	e6	e7	e8	E9	E10
D0	D1	d2	[d3]	d4	[d5]	d6	[d7]	d8	D9	D10
C0	C1	c2	c3	c4	c5	c6	c7	c8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 45. Pattern for cycle with seven outside nodes and no inside node. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

K0	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
J0	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
I0	I1	i2	i3	i4	i5	i6	i7	i8	I9	I10
H0	H1	h2	[h3]	h4	[h5]	h6	[h7]	h8	H9	H10
G0	G1	g2	g3	g4	g5	g6	g7	g8	G9	G10
F0	F1	f2	[f3]	f4	f5	f6	[f7]	f8	F9	F10
E0	E1	e2	e3	e4	e5	e6	e7	e8	E9	E10
D0	D1	d2	[d3]	d4	[d5]	d6	[d7]	d8	D9	D10
C0	C1	c2	c3	c4	c5	c6	c7	c8	C9	C10
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

Fig. 46. Pattern for cycle with eight outside nodes and no inside node. Brackets around a pixel represent a location in a pattern where a node can be placed. A lower case letter in a pixel location indicates a pixel that must not contain a plotted node.

REFERENCES

1. Bourne, S. R. The UNIX shell. The Bell System Technical Journal 57, 6, Part 2 (July-August 1978), 1971-1990.

2. Brinch Hansen, Per. The Architecture of Concurrent Programs. Prentice_Hall, Englewood Cliffs, N. J., 1977.

3. Fundis, R.M. Command processors for dynamic control of software configurations. M.S. Report, Dept. Computer Science, Kansas State University, Manhattan, Kansas (1980).

4. Hoare, C. A. R. Communicating sequential processes. COMM. ACM 21 , 8 (August 1978), 666-677.

5. Rochat, K. L. A software structuring tool for message-based systems. M.S. Thesis, Dept. of Computer Science, Kansas State University,Manhattan, Ks., 1980.

6. Sanders, R.G. Users guide to the Graphics Support System. Report TR-80-04, Department of Computer Science, Kansas State University, Manhattan, Ks., July 1980.

7. Young, Robert, and Wallentine, Virgil. The NADEX core operating system services, Technical Report TR-19-11, Department of Computer Science. Kansas State University, Manhattan, Ks., November 1979.