A HEURISTIC TECHNIQUE FOR THE GENERATION OF NETWORK GEOMETRY FOR RURAL NATURAL GAS DISTRIBUTION SYSTEMS

A Thesis
Presented to
The Department of Civil Engineering
The Faculty of Engineering
The University of Manitoba

In Partial Fulfilment
of the Requirements for the Degree
Master of Science in Civil Engineering

by
James William Davidson
March 1990

ISBN 0-315-63248-8

A HEURISTIC TECHNIQUE FOR THE GENERATION OF NETWORK GEOMETRY FOR RURAL NATURAL GAS DISTRIBUTION SYSTEMS

BY

JAMES WILLIAM DAVIDSON

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

ACKNOWLEDGEMENTS

This work would not have been possible without the help of Dr. Ian Goulter. Special thanks are extended to him for his patience and support through the course of this project's development.

Much of the test data was provided by Rygh Engineering, Regina and the author wishes to thank the staff for their advice and support during the early part of the project.

ABSTRACT

An interactive computer program which provides a heuristic technique for the design of network geometry for rural natural gas distribution systems, was developed. The program is coded in two parts, a procedural component which combines the best features of two common network optimization algorithms - the Minimal Spanning Tree and Dijkstra's algorithm - and a cognitive component which incorporates rules derived from field experience for improving the layouts generated by the procedural component. Iteration between these two components can result in many near-optimal alternative solutions rather than convergence on a single solution.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

The model described in this thesis was developed to facilitate the design of rural natural gas distribution networks. This study extends previous work on the subject (Davidson and Goulter, 1989) and focuses specifically on the issue of the computer assisted design of the network geometry. The previous work by Davidson and Goulter (1989) considered the issues of hydraulic design and cost analysis in conjunction with geometric design. While it was relatively easy to achieve success in automating the hydraulic design and cost analysis components with that approach, the geometric design component proved to be considerably more difficult. Various strategies were tried but the problem was left substantially unresolved, and instead relied heavily on user judgment.

While geometric design is the primary emphasis in this work, it is not possible, or even desirable, to isolate this aspect of the problem from the physical reality for which the work is intended, namely the design of rural gas networks. This model is intended for practical application and therefore, "real world" concerns such as hydraulics, cost, physical and legal constraints on the system geometry are considered to be issues that are integral to the success of a particular system layout.

The proposed methods combine heuristics and user interaction.

This approach is preferred over conventional optimization. What is meant by conventional optimization is that approach which involves formulating the problem as an objective function and a set constraints for which a single optimal solution exists that may be obtained using some standard algorithmic technique such as linear programming. Experience has shown that the problem of proposing network geometry does not lend itself to conventional optimization techniques and there are three major reasons for this:

1. It is extremely difficult, if not impossible, to formulate mathematical expressions for the constraints for a problem of this type. The greatest difficulties would be encountered in articulating the physical and legal geography of the project area as mathematical expressions.
2. The cost of compiling all the required information for the entire project area would be prohibitive, possibly exceeding the cost of constructing the project.
3. There is no known algorithm that can solve the problem in a practical length of time.

Common engineering practice is to use successive refinements to obtain the final layout. The actual procedure is as follows. An initial layout is proposed before all the constraints are fully understood. Physical and legal barriers, which will require that revisions be made, will be imposed once this layout has been proposed. A project may have to undergo numerous revisions of this type before a final layout is obtained.

The procedure for designing these networks is very tedious, time consuming and prone to error. Davidson and Goulter (1989) showed that successive revisions could be accomplished more quickly and easily if the entire process of selecting the network geometry, hydraulic design and cost analysis could be integrated into a single computer program. The major weakness of their method was in the way in which the initial network was selected and the work presented here addresses this problem.

CHAPTER 2: PROBLEM DEFINITION

Physical Characteristics of the Problem

The natural gas distribution systems that are considered in this work are based on design standards for the Saskatchewan Natural Gas Distribution Program (SNGDP) initiated by the Saskatchewan Government and the Saskatchewan Power Corporation in the early 1980's. The networks themselves are constructed from polyethylene pipe. They are branched networks with a single gas source. Since redundancy is not a requirement there are generally no loops in these networks. The gas source, or regulator station, is assumed to operate continuously at a pressure of 550 kPa. The minimum allowable outlet pressure is specified as 140 kPa. This defines a range of pressure referred to as intermediate pressure. In addition a maximum hydraulic gradient of 20 kPa per km is recommended. Davidson (1988) describes the physical characteristics of these networks in more detail.

There are many natural and man-made obstacles to the construction of pipe networks of this type. The natural obstacles include rivers and lakes and the man-made obstacles include roads, railways, pipelines and cables. Some obstacles such as rivers can be crossed at an additional cost, while some obstacles such as lakes simply must be avoided. Experience has shown that the problem of obtaining land easement does more to determine the final layout than any of the other constraints.

Rectilinear geometry is also an important restriction. At a later date it may be necessary to locate components of the system, either to modify it or to allow some other underground utility to cross it. To facilitate locating the pipe, the requirement is made that the system be orthogonal to the survey grid. In practice this turns out to be an extremely costly restriction which has never been strictly adhered to. In recent years, this restriction has been relaxed substantially to reduce costs. Some recent work has shown some success in applying techniques involving fuzzy set theory to design networks in compliance with this relaxed constraint (Davidson, 1989). The use of these techniques is beyond the scope of this work, however it will be assumed throughout this work that the rectilinearity constraint must not be violated.

Graphic Interface and User Requirements

In the previous work by Davidson and Goulter (1989), an interactive graphic interface was found to be a very effective means to satisfy all the user requirements. A similar interface is used with the current program. The computer programs that performed the difficult tasks were non-interactive programs and could be called from the main program which was interactive and graphics oriented in nature. AutoCAD provided the graphics environment and the user interface was programmed in AutoLISP. LISP was found to be particularly well suited to this task because the language is interactive, extensible and handles dynamic memory easily. In this study the same interactive approach has been taken, but since fewer tasks are required to be performed, the user interface is scaled down slightly.

The interactive routines that comprise the user interface have been arranged in four different categories. The routines in the first of these categories are required for the input of data. The only input data required from the user are the set of nodes for a particular problem. The input of these data can be achieved easily and naturally using a digitizer. The connectivity between these nodes, which in effect, is the geometric design of the system, is created and maintained internally by the computer. All of the processes and data structures associated with the maintenance of the connectivity are deliberately hidden from the user. Any alterations in the connectivity must be accomplished by changing the configuration of nodes, essentially through the addition of so called dummy nodes. The process is described in more detail by Davidson and Goulter (1989).

While the process as described here may seem cumbersome, it is, in fact, extremely simple and easy to perform in practice. The objective is to achieve as simple an interface as possible by limiting the type of information that can be provided as input. In addition to achieving simplicity of input, the internal generation of connectivity by algorithms will ensure that the proper network structures are created. This feature cannot be ensured by manual input in the case of a careless or an inexperienced user.

Other routines allow the user to submit data to any of the non-interactive programs. Output data from these programs are automatically displayed in a graphical form when the program terminates. Another category of routines allows the user to control the graphic display. Successive revisions of a design may require the user to clear a previous design from the screen, or to overlay two designs for comparison.

The final category of routines is only required for larger projects. The nodespace, or memory reserved for the LISP interpreter, is 45 kilobytes which is unfortunately very small. Routines are provided to empty the nodespace and reload the LISP programs if the nodespace becomes full.

Network Theory Applications

A network of the type described in this thesis is generally referred to as a Rectilinear Steiner Tree (RST). In the field of Graph Theory the problem of finding the minimal cost RST is generally regarded as being a computationally hopeless task (Garey and Johnson, 1977). Additionally, as networks become more complex, the use of a mathematical model to select the best solution from a set of sufficiently good solutions may not be advisable. There is always some discrepancy between a mathematical model and the physical reality that is being modelled. Many of the feasible solutions that are to be assessed using the model may be so close to each other in their evaluation, that the difference between evaluations is smaller than the accepted precision of the mathematical model. It is also possible that the criterion used for evaluating solutions may be sufficiently imprecise and may not adequately reflect all aspects of a solution which determine that solution's desirability in the context of the physical reality. Dubois (1983) has suggested two reasons why a good (heuristic) solution may be preferable to the global minimum:

1. As the network size increases, the optimum ... degenerates into a group of equivalent solutions whose evaluations are very close to each other. The more complex the network, the more insensitive the criterion, when a link is dropped or added.
2. Owing to the uncertainties on the data, the system structure, and the criterion formulation, the optimal solution of the mathematical problem may not correspond with the optimal solution in the real world.

Rather than to attempt to find the global minimum, the approach that has been taken in this study is one of relying on heuristics to obtain a reasonably good solution.

For the sake of simplicity, in this study many of the accepted conventions of Graph Theory have been set aside or changed. Typically a graph is composed of a set of nodes, a set of edges and some function which describes the connectivity between the nodes and edges. The convention that has been adopted in this study is that a pipe network is described by a set of nodes and a set of lines.

The nodes are assumed to exist in two dimensional space. They are numbered and each node has an x and y coordinate. One node acts as a source which is assumed to be capable of supplying all the required gas at a consistent pressure. Other points act as sinks which determine the load throughout the system.

All lines are assumed to be straight lines and to have directions. Each line originates at one node and terminates at another. Since only straight lines are permitted, additional nodes are required to produce the tees and elbows found in a rectilinear network. These nodes do not contribute any load to the system and are referred to as dummy nodes.

The convention that has been adopted for the representation of the networks in the figures throughout this work is as follows. The source node is represented as a circle with the letter "S" drawn in the centre. All nodes are numbered and the source node is considered to be node 1, although this number is never displayed.

The sink nodes are represented by solid black circles. The number assigned to a sink node may be displayed to the immediate right of a node or concealed as required. Dummy nodes may be represented in a network or they may be left out to improve the clarity of a figure. If represented, dummy nodes appear as circles that are not solid and the node numbers may be represented to the right of the node. Connecting lines are simply represented as straight lines. Alternatively, an arrowhead may indicate the direction of flow of gas in a line.

Principal Objectives of Geometric Design

Experience shows that experts in the field of gas network design rely heavily on intuition when designing systems. Typically the designer's methods become less systematic as his ability improves. It would be extremely difficult to produce a set of rules or procedures that would emulate the entire design process from beginning to end, but three major issues, or objectives, have been identified based on the author's experience, as follows:

1. Eliminate as much of the parallel piping as possible to reduce the total system length.
2. Branch as "early" as possible without conflicting with the first objective.
3. Eliminate as many bends or elbows in the system as possible without conflicting with the first two objectives.

The first objective results from the fact that the cost-to-capacity ratio for larger diameter pipe is more favourable than for smaller diameter pipe. Parallel pipes should be replaced by a single pipe of larger diameter where possible. Figure 1a shows a system serving two sinks with parallel segments in the system. Figure 1b shows the same system with the parallel piping eliminated. The total cost of the system in Figure 1b will be lower than the cost of the system in Figure 1a even if the system in Figure 1b requires a larger diameter pipe for the common segment.

The total length of pipe in the systems shown in Figure 1c and Figure 1d is the same. The layout in Figure 1d is superior to the layout in Figure 1c because the branch occurs "earlier" (i.e., closer to the source). Placing the branch at node 5 rather than node 2 has decreased the flow of gas through the segment from node 5 to node 2 which may permit the use of a smaller diameter pipe for this segment.

The systems shown in Figure 1e and Figure 1f are equivalent in terms of total length and hydraulics, however, the system in Figure 1f has one less bend and is considered to be superior to the system in Figure 1e for this reason. While the elimination of unnecessary bends does not translate directly into an improved cost, the system with fewer bends has an improved general form and will be easier to install and survey.

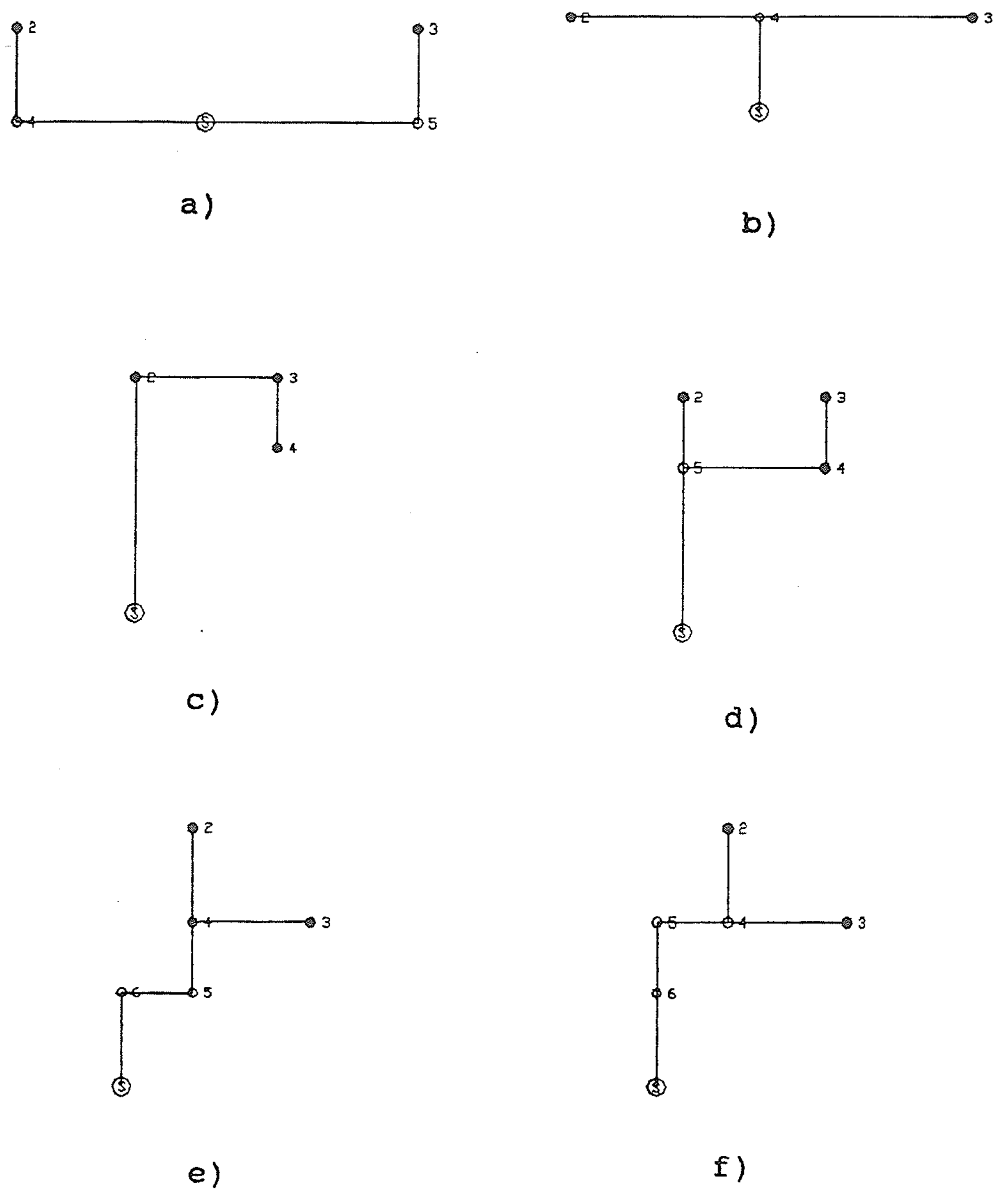


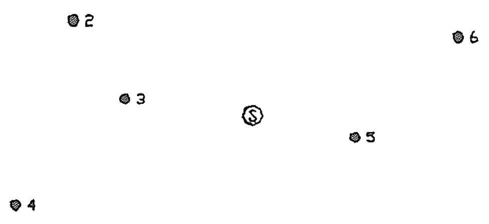
Figure 1: Examples illustrating three objectives

CHAPTER 3: PROCEDURAL COMPONENT OF THE PROGRAM

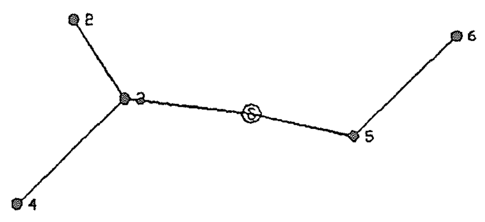
Automated Selection of Dummy Nodes

The procedural component of the program consists of a set of global data structures which can be operated on by a set of subroutines. The development strategy adopted was one of exploratory programming. The design of the data structures and subroutines evolved together. The mechanisms used in the procedure are best explained by tracing their development.

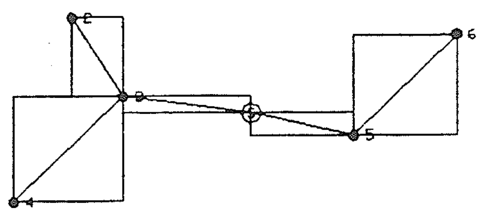
The previous work by Davidson and Goulter (1989) used the Minimal Spanning Tree (MST) algorithm to establish the connectivity of a set of input nodes. This approach had the advantage of ensuring that a relatively efficient network that was continuous and contained no cycles, would be produced. The major disadvantage of this method was that it produced trees composed of diagonal lines rather than rectilinear lines. One method that was explored to overcome this problem of diagonal lines was to enclose each diagonal line in a rectangle. If an additional node was incorporated at one of the corners of the rectangle a second iteration of the MST algorithm would replace that diagonal with a pair of rectilinear lines. This procedure is called the Boxplot Method and is described in Figure 2.



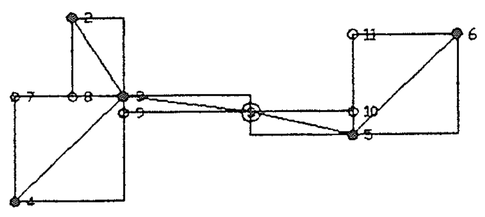
a) Source node and sink nodes



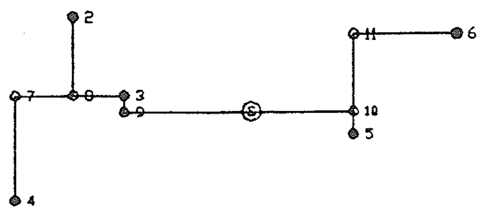
b) MST algorithm is used to create a tree network



c) Each diagonal line is enclosed in a rectangle



d) One corner of each rectangle is selected



e) A second iteration of MST produces a rectilinear layout

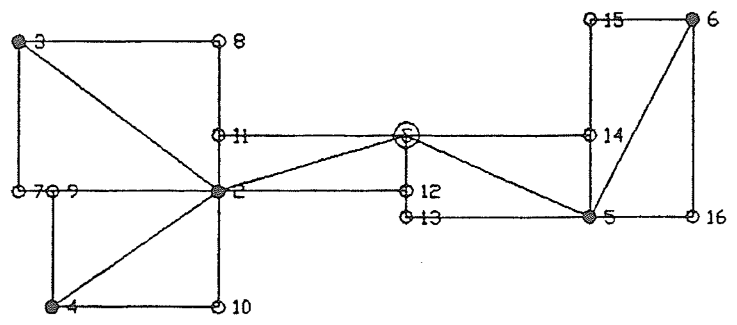
Figure 2: Example of the Boxplot Method

Several difficulties were encountered with the Boxplot Method, namely:

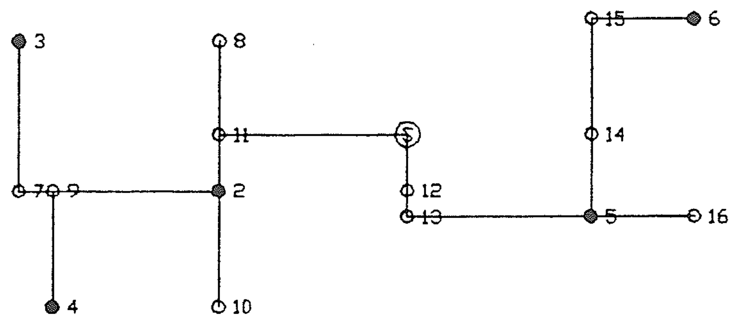
1. The method is time consuming for large networks.
2. Each rectangle offers two potential candidates for the location of a single dummy node. Since each diagonal line requires one dummy node there exists a very large set of combinations of corner nodes to choose from. The method relies on user judgment to select the appropriate set of corner locations.
3. Even the best selection of corner nodes does not ensure that the Minimal Rectilinear Steiner Tree (MRST) solution will be achieved.

The first step in the evolution of the present algorithm was to automate the selection of corner nodes. This was accomplished by including both candidate corner nodes for a given boxplot and running the MST algorithm for another iteration. The nodes that are added are flagged as dummy nodes. The tree network that results from the second iteration of the MST algorithm contains many lines that supply terminal dummy nodes. A terminal dummy node is a node that occurs at the end of a branch in a tree. Since a dummy node does not contribute any load to the system, the line supplying a terminal dummy node serves no purpose. A procedure was developed that identified and removed these terminal dummy nodes and the lines that supply them. Figure 3 shows the sequence of steps for a simple example.

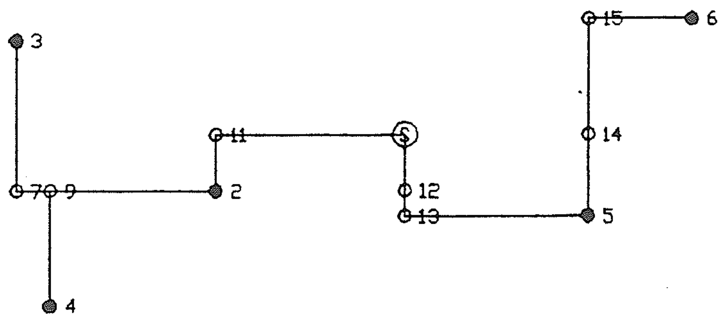
In general, the MST algorithm selects the shortest spanning tree



a) Both corner points are selected



b) A second iteration of the MST algorithm produces a rectilinear layout with terminal dummy nodes



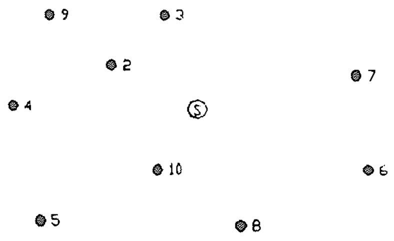
c) Terminal dummy nodes and supplying links are removed

Figure 3: Automated selection of dummy nodes

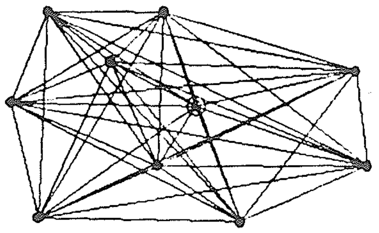
subgraph (or subset of lines) from a supergraph (or superset of all candidate lines). A spanning tree is a tree graph which connects all the nodes in the set. In this program the MST algorithm is implemented to select a spanning tree from a complete graph. A complete graph is the set of lines connecting every node to every other node in the set of nodes.

Dijkstra's algorithm can be used as an alternative to select a spanning tree from a supergraph (Bondy and Murty, 1976). Dijkstra's algorithm is a procedure that is used to find the shortest distance between two points in a graph. The algorithm generates the entire set of shortest paths from a single point to every other point on the graph. This set of shortest paths takes the form of a tree graph which is different from the Minimal Spanning Tree. Unlike the Minimal Spanning Tree, a different tree is produced when a different node is used as the origin or starting point. In this program Dijkstra's algorithm is implemented with the source as the starting point, since it is advantageous from the point of view of hydraulics to minimize line haul distances from the source to all sinks.

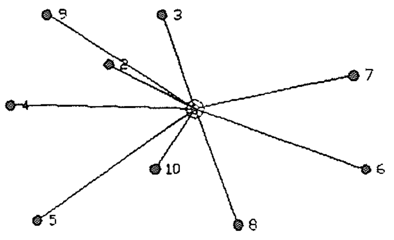
An example configuration of nodes is shown in Figure 4a. The complete graph formed from these nodes is shown in Figure 4b. If Dijkstra's algorithm is executed using Node 1 as the source node and the complete graph in Figure 4b as input, the graph in Figure 4c is the result. This is a star graph; it is composed of straight



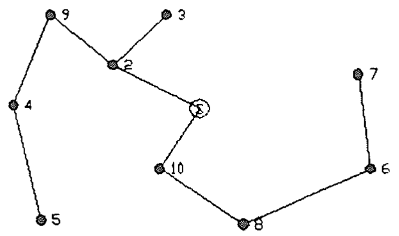
a) Set of nodes



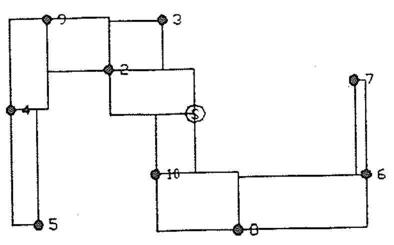
b) Complete graph



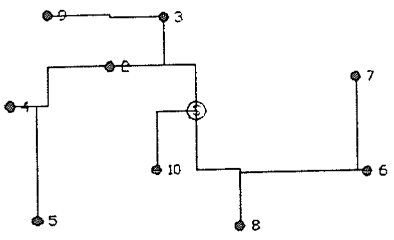
c) Star graph formed by Dijkstra's algorithm from (b)



d) Layout produced by MST algorithm



e) Supergraph produced by Boxplot Method



f) Dijkstra's algorithm is used with supergraph in (e)

Figure 4: The use of Dijkstra's algorithm

lines from the source to each of the sinks. This type of geometry is extremely impractical. Dijkstra's algorithm will always produce a star graph of this type from a complete graph.

It is necessary, therefore, to use some other type of supergraph with Dijkstra's algorithm. The following method combines the MST algorithm with Dijkstra's algorithm. The graph in Figure 4d results from the MST algorithm with the complete graph in Figure 4b as input. The graph in Figure 4e is composed of rectangles drawn from the diagonal lines of the graph in Figure 4d using the Boxplot Method. If the graph shown in Figure 4e is used as a supergraph for Dijkstra's algorithm the result is the graph in Figure 4f. The graph in Figure 4f has had the terminal dummy nodes and supplying links removed.

The graph in Figure 4f is a more practical solution than the graph in Figure 4c but some shortcomings are still evident in the parallel piping used to serve nodes 9 and 10. Dijkstra's algorithm typically produces layouts that have a greater overall length than layouts produced by the MST algorithm. However, the circuitous routings that the MST algorithm often produces can be avoided because Dijkstra's algorithm minimizes individual path lengths.

Use of Heuristic Techniques

Two heuristic techniques were developed to combine the best features of the MST algorithm and Dijkstra's algorithm. One of these techniques works by trying to eliminate the parallel lines produced by Dijkstra's algorithm. This heuristic is referred to as the "concept" heuristic and it will be explained in detail later.

The other heuristic technique will be described first. It can be illustrated by considering the following problem. A system is composed of three nodes, one source and two sinks. The MST algorithm is used to generate the graph in Figure 5a. The corner points of the rectangles enclosing the diagonal lines are added as dummy nodes and a supergraph is generated containing only the edges of the rectangles as shown in Figure 5b.

The four possible rectilinear solutions for this system are depicted in Figures 5c, 5d, 5e, and 5f. If minimum length is used as the criterion for optimality, the layout in Figure 5f is superior to the other three because the rectilinear pathways to each of the two sinks share a common line segment, namely the segment connecting node 5 and the source. It is this common line segment that results in the improved solution. The heuristic technique employed here is to find these common line segments in the supergraph and assign them artificially reduced lengths which are substantially shorter than their actual length. Dijkstra's

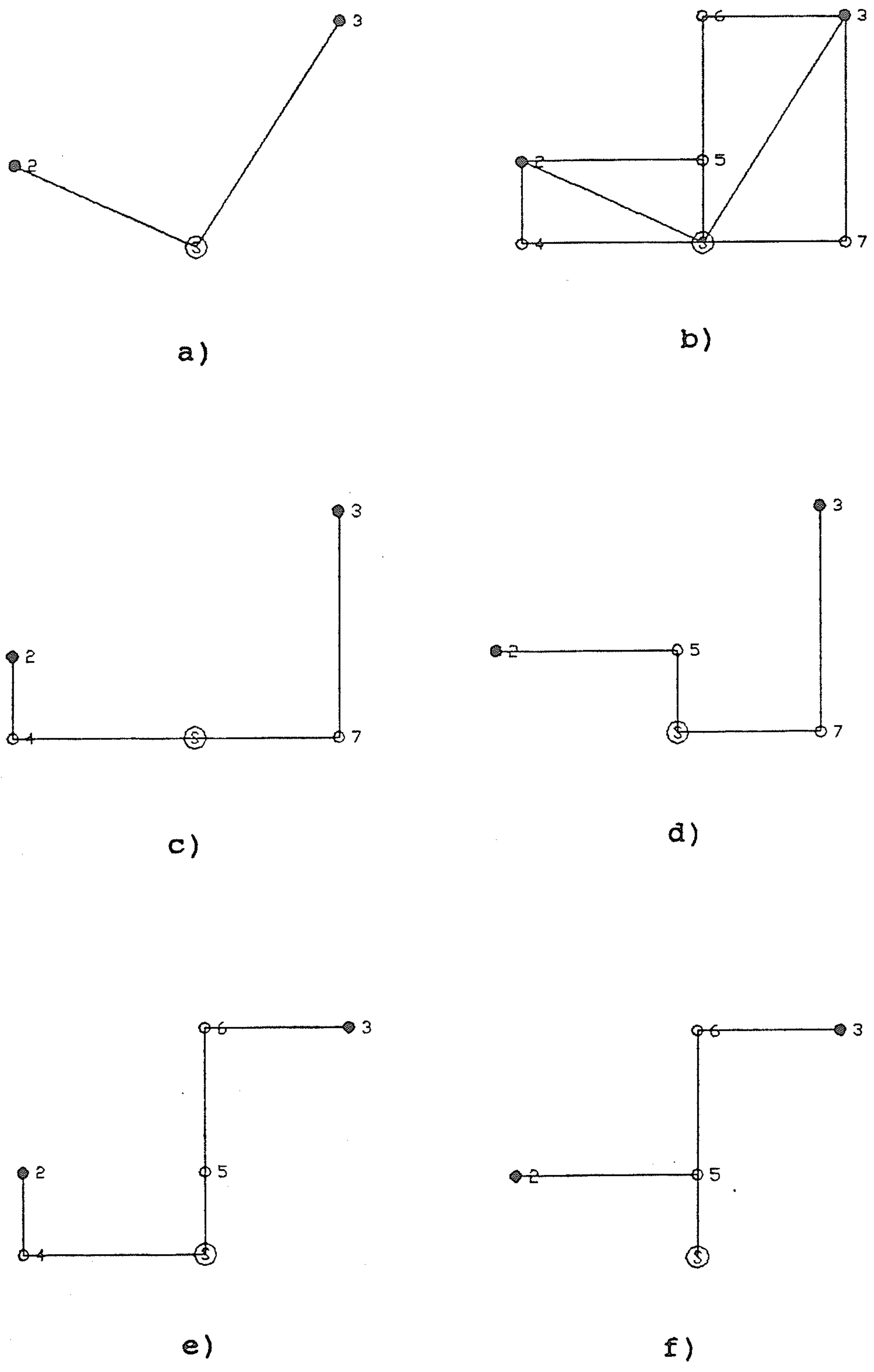


Figure 5: Three node problem

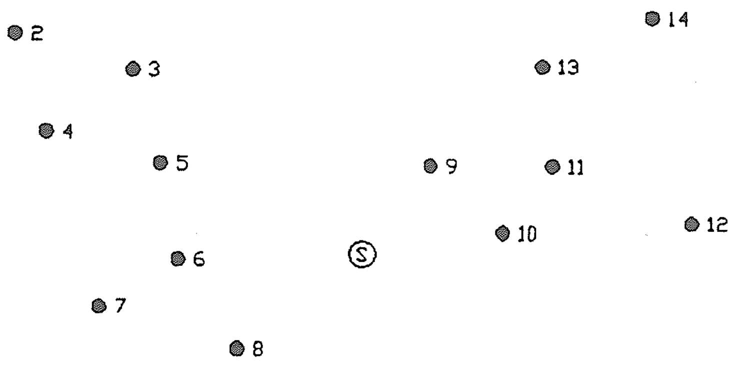
algorithm selects the shortest path to a node through a supergraph by minimizing individual path lengths. The distance transformation will ensure that Dijkstra's algorithm will choose the paths to the sinks which contain common segments in favour of the paths which do not contain these common segments. The MST algorithm on the other hand minimizes overall system length and will not ensure that segments with artificially reduced lengths will be incorporated along a path to a sink. It should be noted that the two rectilinear solutions corresponding to the graphs in Figures 5e and 5f can both be generated using the MST algorithm even when the distance transformation is performed.

Since the technique just described requires Dijkstra's algorithm to operate effectively another heuristic is employed to improve the inefficient layouts that Dijkstra's algorithm creates. Dijkstra's algorithm does not attempt to minimize the total length of the system in the same way as the MST algorithm does. As mentioned previously, Dijkstra's algorithm often selects layouts with many parallel lines which could be eliminated. The other heuristic technique identifies certain paths in the system as central "spines" and tries to connect the nodes directly to these paths by the shortest distance. It is possible to use Dijkstra's algorithm in successive iterations in combination with this technique to remove parallel lines.

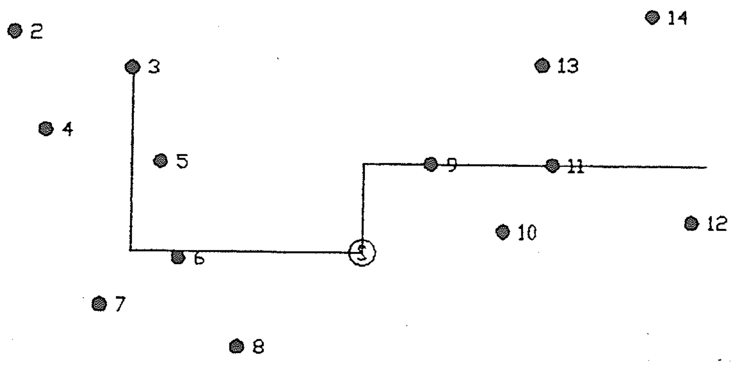
The technique is similar to a method that is used by engineers in practice. In proposing a layout an engineer will often select a very simple layout composed of a few lines which do not actually connect all the sinks but pass through the centres of clusters. This simple layout forms the structure, or concept, of the final layout. The final complete layout is created by connecting the sinks to this structure with short lines as shown in Figure 6.

The heuristic technique begins with a layout that has been generated from the modified supergraph of rectangles using Dijkstra's algorithm in the manner explained above. The longest path is identified by finding the node that is the most distant from the source. Each line on the path to this node has its length artificially reduced to zero in the supergraph. This single path of "zero length" lines designates, or defines, the initial "concept".

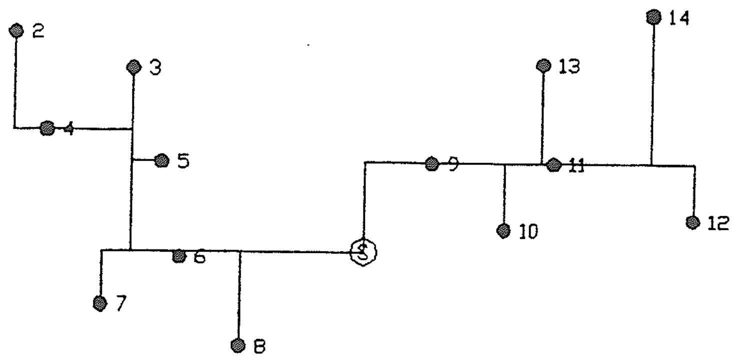
Dijkstra's algorithm is run again using the modified supergraph with this "concept" of a long path of "zero length". Since Dijkstra's algorithm minimizes the individual path lengths, nodes in the vicinity of the "zero length" path will be connected by the shortest line to that path in the same way the sinks are connected to the "concept" in practice. To determine if a node is in the vicinity of a "zero length" path the following rule can be used. If there is no path on the supergraph to the source that is shorter than the shortest path to a "zero length" path, then the node is in the vicinity of that "zero length" path and will be connected



a) Source and sink nodes



b) Concept is proposed



c) Remaining sinks are connected to the system

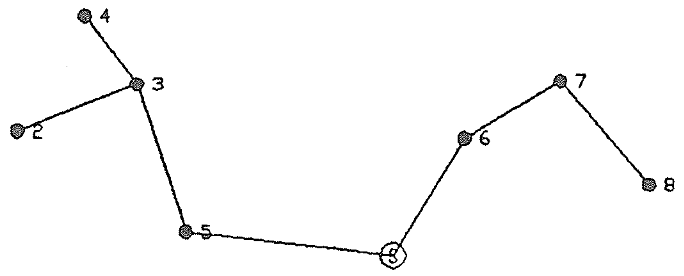
Figure 6: Design based on concept

to the "zero length" path. Otherwise the node will be connected to the source in the usual manner and not through the "zero length" path.

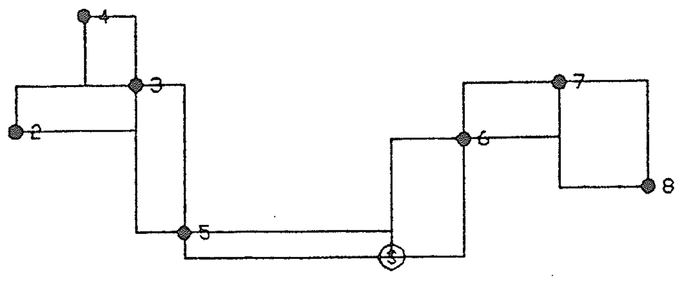
This process of identifying the longest path as the "concept" can be repeated for several iterations. Each iteration identifies a new "most distant" node and converts the path from the source to this node to a "zero length" path. Thus each iteration adds a new "zero length" path to the "concept". When a second iteration is tried the longest path is likely to be a long path proceeding in a different direction from the first path. This is because nodes in the vicinity of the previously selected path may be physically very distant from the source, but, due to their connection to a "zero length" path, appear to the algorithm to be only as far from the source as the distance to the nearest "zero length" path.

Figure 7 illustrates this point. In Figure 7c, node 4 is identified as being most distant from the source and the path to node 4 is identified as the "concept". During the second iteration node 8 is identified as the "most distant" node and node 8's path is included in the concept. Node 2 may actually be more distant than node 8, but to the algorithm the distance from the source to node 2 is only the length of the dashed line.

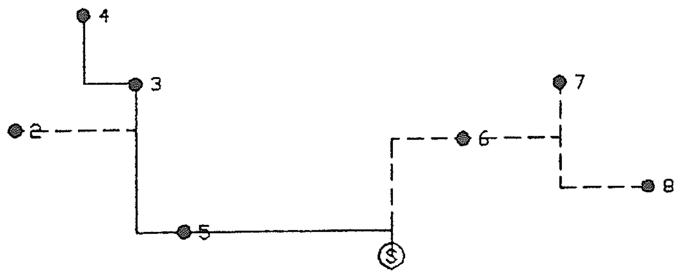
It is possible to iterate on this process until all the paths on the tree subgraph have a zero length, but in practice it is not



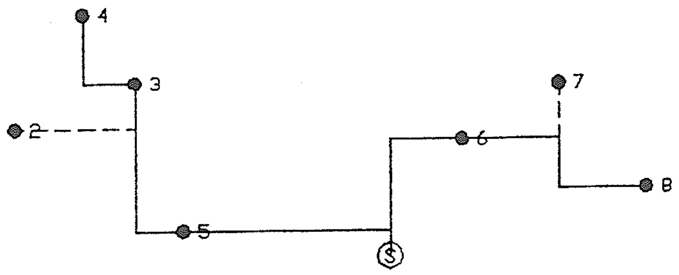
a) Minimal Spanning Tree



b) Modified supergraph of rectangles



c) Results from Dijkstra's algorithm
with longest path identified



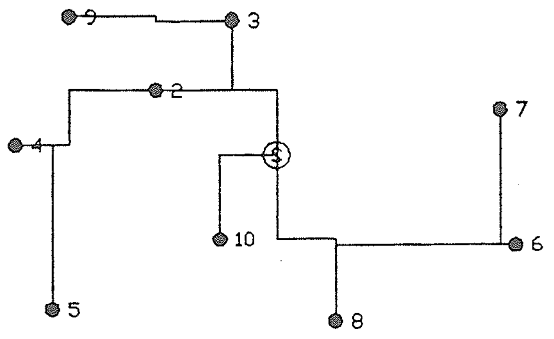
d) Results from second iteration
with new path added

Figure 7: The "concept" heuristic

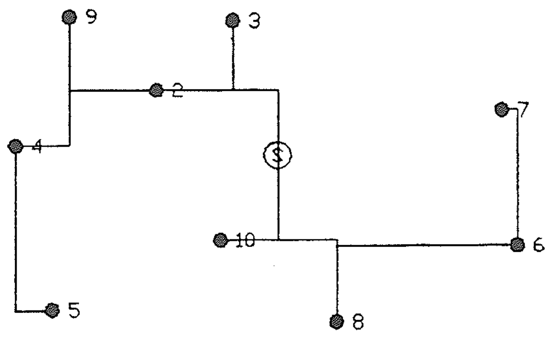
necessary. For the reason just explained, and on the basis of experience with the algorithm, four iterations are generally all that is required to "tighten-up" the most complex network.

Figure 8a shows the same layout as Figure 4f developed using Dijkstra's algorithm. The layout in Figure 8b is generated from the same set of nodes using both of the heuristic techniques that have been described. The "concept" heuristic was allowed to iterate to the completion point when all paths have become "zero length" paths. The parallel lines serving nodes 9 and 10 have been eliminated but new problems have resulted in the lines serving nodes 5 and nodes 7. These problems occurred because the dummy nodes at both the elbows adjacent to nodes 5 and 7 were found to be the end nodes of a "most distant" path. When these most distant paths were converted to "zero length" paths in subsequent iterations of the "concept" heuristic, the shortest path to the nodes 5 and 7 is to connect to the "zero length" paths. The problem is referred to as a "hook" in the next chapter and does not occur as frequently as this example might suggest. The rule based techniques that will be discussed in the next chapter are used to correct the problem.

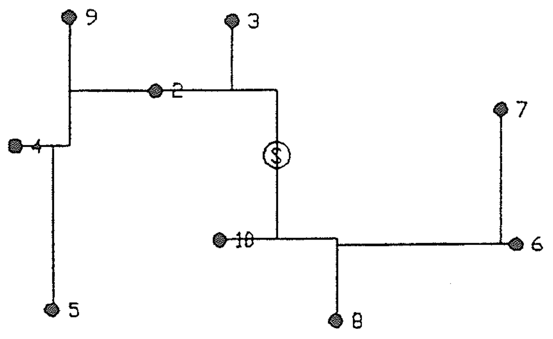
The graph in Figure 8c is the layout produced by the algorithm as it was finally implemented. In the final algorithm the "concept" heuristic is allowed to iterate to a maximum of four times. The following series of procedures is then performed. All terminal



a) Results from Dijkstra's algorithm without heuristics



b) Both heuristics are applied



c) Final results from the procedure

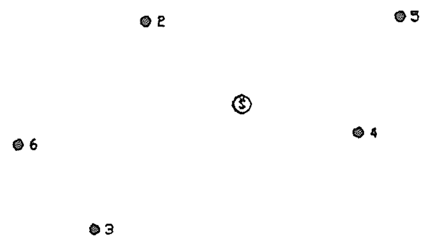
Figure 8: Results from various algorithms

dummy nodes and serving links are removed. A complete graph is constructed and a tree graph is extracted using the MST algorithm. If no diagonal lines occur in the tree graph then the process terminates. If diagonal lines are found, rectangles are substituted for the diagonals and Dijkstra's algorithm is run again. The process iterates until a layout containing no diagonal lines is produced by the MST algorithm.

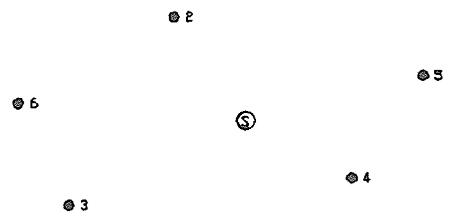
This final series of iterations corrected the problems associated with nodes 5 and 7 in Figure 8. Unfortunately in this instance the corrections of this type are due to chance alone and will not always occur.

In most cases the final series of iterations improves the layout. An improved layout may occur because the MST algorithm can find diagonal "short cuts" in the solution produced using Dijkstra's algorithm. However, it is not correct to assume that iterating between the MST and Dijkstra algorithms always reduces the system length. This point is explained further in more detail in Chapter 5.

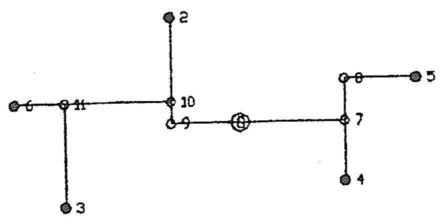
A more significant reason for this final iteration process is to produce skewed rectilinear layouts of the type shown in Figure 9. It is often desirable to have a set of sinks and dummy nodes that will produce a completely rectilinear system when the MST algorithm is run on a complete graph of these nodes. A set of



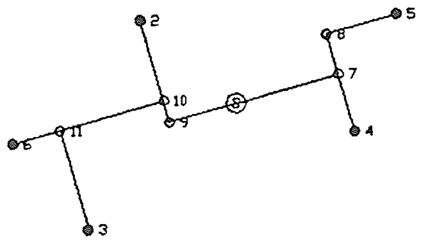
a) Original set of nodes



b) Nodes are rotated 15 degrees clockwise



c) The procedure is used to create a rectilinear system



d) The nodes are rotated 15 degrees counter-clockwise and connected using the MST algorithm

Figure 9: Creating a skewed layout

nodes, such as that shown in Figure 9, can be rotated to any angle. When the MST algorithm is run with a complete graph of these nodes as input, the result will be a rectilinear system skewed at the angle of rotation.

It may be necessary to design skewed layouts because the axes of the survey grid often do not correspond to the axes of the coordinate system used to produce CAD drawings. The example presented in Figure 29 in Chapter 5 uses the Universal Transverse Mercator (UTM) system of coordinates. At the particular location chosen as the example the survey grid is rotated 3 degrees counter-clockwise from the axes of the UTM coordinate system.

The technique that is employed to rotate the coordinates of nodes is to multiply the node vectors by a rotational transformation matrix. The network is designed on the basis of the rotated nodes. The design program produces the dummy nodes required to reproduce the final layout using the MST algorithm alone. All nodes including the dummy nodes are then rotated back to the original position by the reverse procedure of that used to rotate the layout originally. Running the MST algorithm alone produces a skewed layout. The example in Figure 9 shows how a grid skewed at 15 degrees can be produced.

Limits of the Procedural Approach

Figure 10 shows a flowchart of the procedural component of the program as it was finally implemented. After several experiments with alternative arrangements of the procedures described so far, it became apparent that there was no single arrangement of these procedures that could produce results that were consistently better than any other arrangement. The problems that resulted were always due to one of two reasons:

1. The program chose an inefficient layout because it was incapable of generating a dummy node at a particular location.
2. The program chose an inefficient layout because the program had generated a dummy node at the wrong location and was unable to eliminate it.

These problems may be due to inherent limitations in the approach taken. The approach has been to combine the MST and Dijkstra algorithms to exploit the best features of each of them. The two algorithms have distinctly different properties and behaviour. If the algorithms are examined carefully in terms of the way in which each of them handles the problem of selecting and incorporating dummy nodes into the network, the major weaknesses of either of the two algorithms become obvious.

The specific strengths and weaknesses of the two algorithms can be described as follows. If an arbitrary arrangement of nodes is

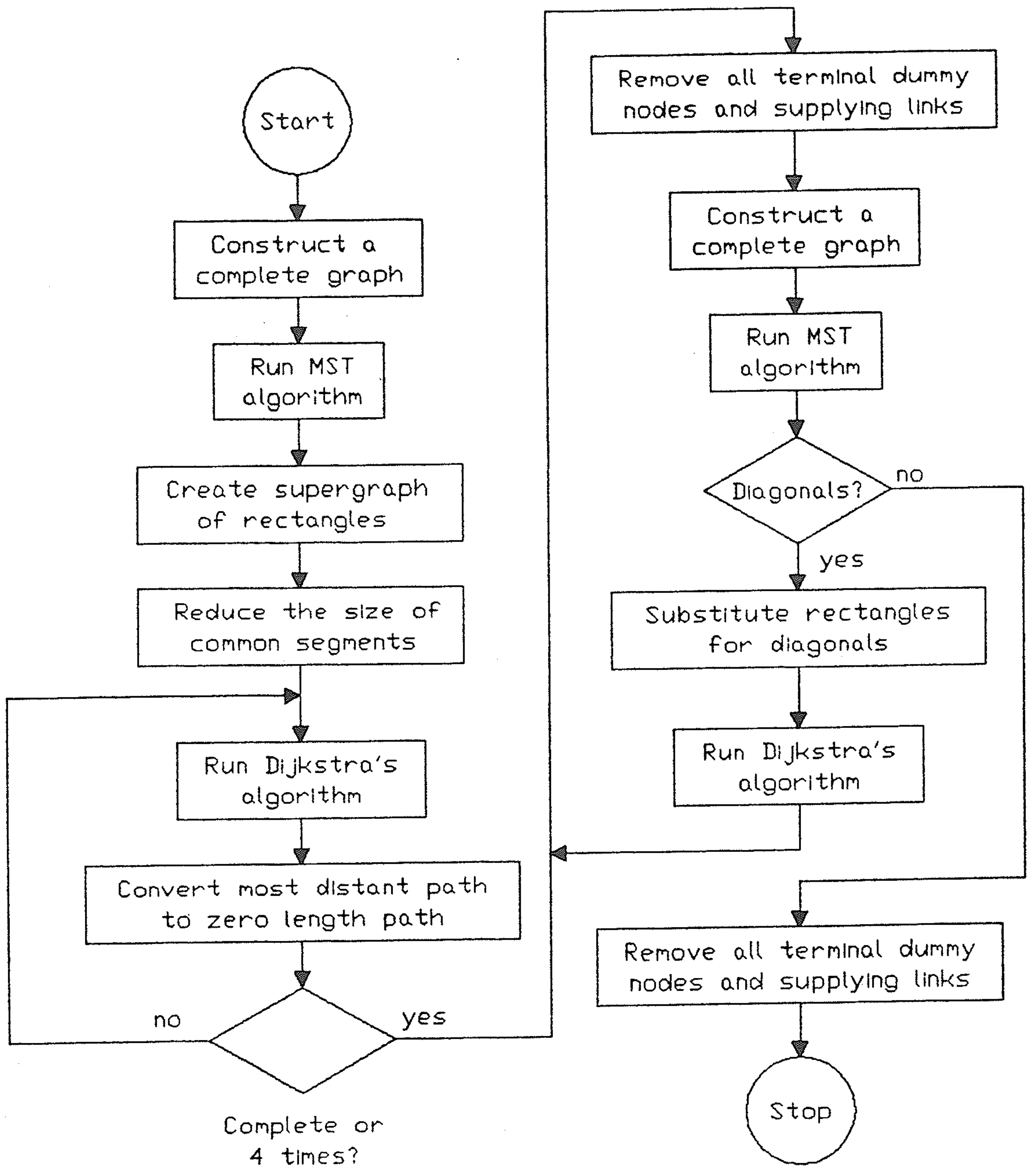
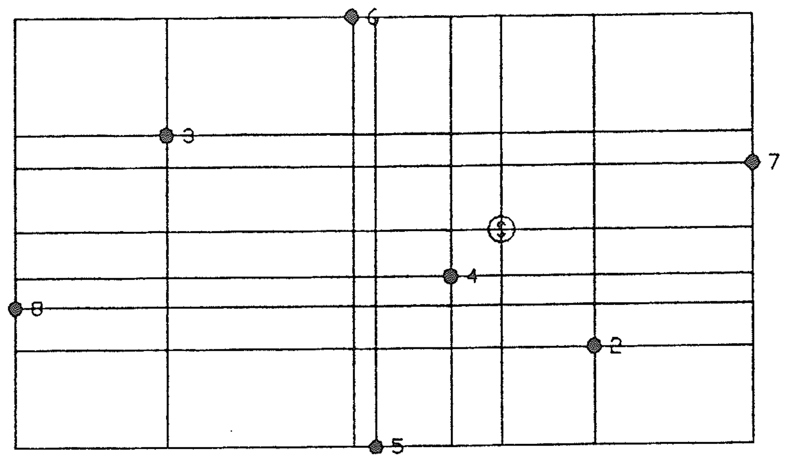

Start
Construct a complete graph
Run MST algorithm
Create supergraph of rectangles
Reduce the size of common segments
Run Dijkstra's algorithm
Convert most distant path to zero length path
no
yes
Complete or 4 times?
Remove all terminal dummy nodes and supplying links
Construct a complete graph
Run MST algorithm
Diagonals?
no
yes
Substitute rectangles for diagonals
Run Dijkstra's algorithm
Remove all terminal dummy nodes and supplying links
Stop

Figure 10: Flowchart of procedure

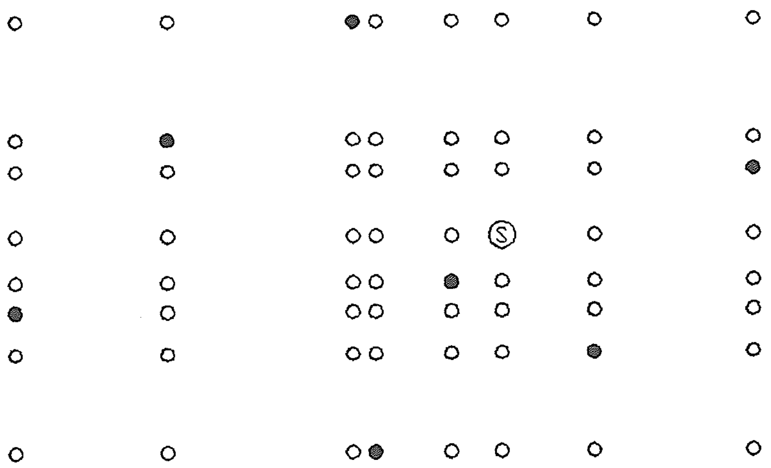
considered and orthogonal lines are extended from each node to the perimeter of the smallest rectangle enclosing all of the nodes, a grid is produced similar to the one shown in Figure 11a. It is not possible to prove at this time whether or not the minimal RST is a subset of this graph. It will be assumed however that the optimal solution, or a solution suitably close to the optimum, is contained within this graph. If this is correct, only the intersections of each of these lines need to be considered as potential candidates for the location of dummy nodes. Use of these intersections produces the complete set of potential dummy nodes shown in Figure 11b. Both the MST and Dijkstra algorithms produce very poor solutions if they are given the full set of potential dummy nodes. Yet, the algorithms behave poorly under these conditions for opposite reasons.

Both algorithms generate paths between the source and sink nodes. The MST algorithm will try to incorporate every dummy node along or near a path to a sink in accordance with its objective to produce the smallest network that includes all the nodes. The result is very circuitous paths as the algorithm detours to incorporate nearby dummy nodes whether they are required nodes or not.

Figure 12a shows the results of the MST algorithm using the set of nodes in Figure 11b as input. The system would be a very compact and efficient system if all the nodes shown were actually required

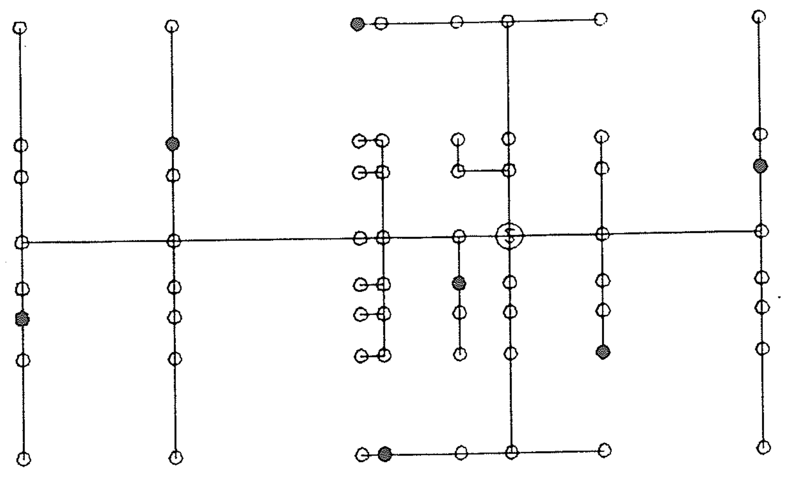


a) Grid formed by orthogonal lines drawn through each node

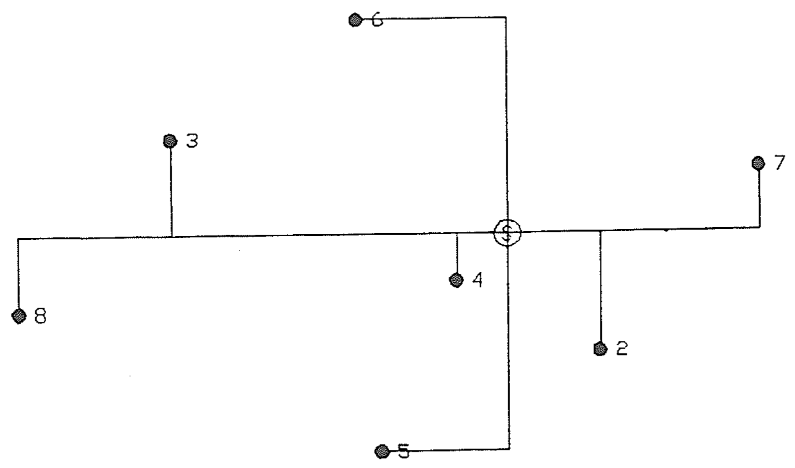


b) Dummy nodes are included at the intersections of lines

Figure 11: Including all potential dummy nodes



a) Results from MST algorithm with
dummy nodes shown in Figure 11(b)



b) Results from MST algorithm with
terminal dummy nodes and supplying
links removed

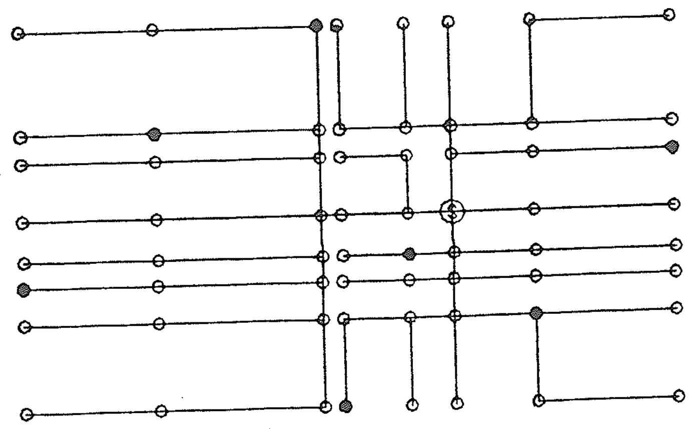
Figure 12: Results from MST algorithm

with all potential dummy nodes

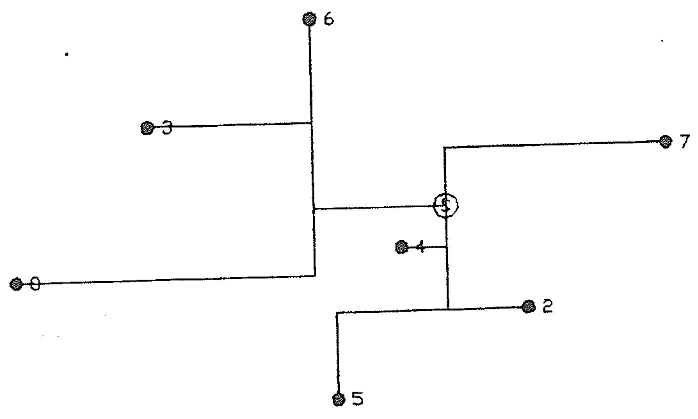
to be served. When all the dummy nodes and redundant links are removed, the system shown in Figure 12b is the result. The circuitous paths to nodes 5 and 6 are obvious shortcomings of this layout. From Figure 12a it is apparent that these paths were selected to incorporate nearby dummy nodes.

Dijkstra's algorithm behaves in the opposite manner to the MST algorithm. Its objective is to minimize path lengths regardless of overall system length. The algorithm generates paths with complete disregard to the location of other sinks. The length of the overall system would be reduced if the algorithm were capable of generating paths with an occasional detour to incorporate a nearby sink.

Figure 13a shows the results of Dijkstra's algorithm with the same set of input nodes. This system is much larger than the one generated by the MST algorithm and contains far more parallel lines. This would be a very inefficient layout if all the nodes were required to be served, but when the dummy nodes and redundant links are removed the remaining system, shown in Figure 13b, is not nearly as inefficient. However, there are still some obvious problems, particularly in the lines which encircle node 4. This node could have been incorporated at no additional cost in either of the paths to node 8 or nodes 5 and 2. Dijkstra's algorithm makes this mistake, because of its tendency to consider paths independently.



a) Results from Dijkstra's algorithm with dummy nodes shown in Figure 11(b)



b) Results from Dijkstra's algorithm with terminal dummy nodes and supplying links removed

Figure 13: Results from Dijkstra's algorithm with all potential dummy nodes

In summary, both algorithms require some preselection of potential dummy nodes. Both algorithms are just as incapable of producing good solutions when given too many candidates for dummy nodes, as when they are given too few candidates.

CHAPTER 4: COGNITIVE COMPONENT OF THE PROGRAM

Nature of the Control Strategy

Many of the limitations of the numerical approaches described in the previous chapter can be overcome by using an approach which is essentially cognitive in nature. The concept of coupling procedural and cognitive approaches has been used with success in the domain of floorplan design of integrated circuits (Jabri and Skellern, 1988).

The idea of combining these two approaches grew from the observation that deficiencies in the layouts that the procedural component produced occurred in certain patterns. Each of these patterns required a specific type of modification to correct the deficiency. The modification could be made once the type and the location of the deficiency had been identified. This "diagnose and correct" technique could best be accomplished with a rule based program. The formal structure that was adopted after some experimentation was a data driven production system.

A production system is a program composed in three parts:

1. Objects - data that symbolically represent the problem in a particular state.
2. Operators - rules that describe how the objects can be transformed from one state to another.

3. A control strategy - a program that translates the operators into procedures that the computer can perform.

The operators or rules are written in a declarative format. This means that the rules state the actions that the computer can perform on the objects but the rules do not state specifically what procedure is to be used to determine when a rule is to be used or not. The production system relies on the control strategy to translate the rules to a procedural form.

The term "data driven" means that the operators act on the objects which are in some initial state, transforming the objects to approach some predefined goal state, or in some cases, simply an improved state. The other approach is a goal driven system. With this approach the objects are assumed to be in the goal state and are transformed in reverse order to the initial state. In most expert system shells the data driven approach is referred to as forward chaining or forward reasoning. The goal driven approach is referred to as backward chaining. Since the goal state or optimal solution is unknown, the goal driven approach cannot be used for this problem.

The objects and operators for this instance of the Minimal Rectilinear Steiner Graph problem system are unusually simple. In many respects the problem resembles the "toy" problems used to illustrate the production system concept in textbooks on artificial intelligence. The objects are essentially of two types: the nodes and their coordinates; and the lines or connections between nodes.

Another important aspect of this problem is that it is possible to define the production rules so precisely that the effect of any rule can be known a **priori**, before that rule is applied. In other words it is possible to know if the rule will decrease the overall length of the system and, if so, by what amount; or if the rule will improve the system's hydraulics. It is not possible, however, to know a **priori** if the rule will allow further rules to be applied subsequent to its application. For this type of knowledge a "generate and test" procedure would be required, an approach which was not taken with this program but will be discussed later in Chapter 5.

Each rule is composed of two sets of clauses - antecedent clauses and consequent clauses. The rules are similar in structure to the "if ... then ... else ..." structures in conventional programming languages. The antecedent clauses are predicates, or statements that are to be verified as true. The antecedent clauses correspond to the "if" clauses in the conventional programming language structures. If the data describing the objects can be matched to the variables in the antecedent clauses such that all the antecedent clauses can be verified as true, the control passes to the consequent clauses. The consequent clauses perform the actions that transform the data. The consequent clauses correspond to the "then" clauses in the conventional programming structure. There is nothing that corresponds to the conventional "else" clauses in the type of rules implemented in this program.

Each rule in the rule base is, in effect, a small program which transforms the data describing the network. All of these small programs have to be coordinated in some way to produce the desired results. This coordinated action of rules is accomplished by a larger program under which the rules operate. This larger program is the control strategy mentioned earlier. In many rule based applications, such as expert systems, the control strategy is not designed by the programmer but is an integral part of the development tool, or shell. In this application the control strategy was largely custom designed. The portion of the program described here was written in PROLOG. PROLOG provides the advantage of the kind of control that a procedural language allows, and which is necessary in designing the control strategy, but still maintains the declarative syntax necessary for the rule base. Some aspects of the control strategy, namely unification and backtracking, were also provided by the PROLOG language. Forward-chaining and conflict resolution are not provided by the language and have to be included as part of the overall program.

Before the specific details of any of the rules in the rule base can be understood it is necessary to become familiar with the control strategy under which these rules operate. A rule is activated in two steps. The first step is to search the data to find every possible instance where a rule could be applied. This can be accomplished easily with PROLOG's unification and backtracking algorithms. If a particular combination of data fulfils a rule's antecedent clauses, the rule is said to have instantiated. If instantiation occurs a copy of the name of the rule and the data which instantiates the rule is recorded in an area of the computer memory designated as the conflict set. At this point the rule is said to have triggered. If a rule has triggered it may be used to alter the data, but at this stage it has not yet done so. The rule is being held on "stand by" while other instantiations of the rule, or other rules are being investigated.

Once all the successful instantiations of all the rules have been recorded in the conflict set the second step begins. The different instantiations of the rules are rated according to some predefined criteria and the best instantiation is chosen. This mechanism is referred to as conflict resolution. The chosen instantiation is then fired. Firing a rule means that the program described by the consequent clauses of the rule is executed. Typically only one instantiation of a rule in a conflict set fires. The rest of the triggered rules do not fire. However, in this case it was possible to adopt a slightly different strategy that resulted in a substantial improvement in the programs's execution time.

In most instances rules that are applied behave independently of each other. This is due to the fact that the problem is composed of several independent sub-problems. Another way to regard this is that the single object in the production system is actually a collection of connected yet independent objects. If the sub-problems are truly independent the order in which the rules are applied does not affect the final outcome.

Figures 14a and 14b represent examples of this type of problem. Rule 1 can be applied to one part of the network and Rule 2 can be applied to another. These two parts of the network are physically distant from each other and modifications in one part do not affect the other. Each rule can be applied only once to its respective area. Figure 14a shows the search space of a conventional production system. Since the results of the two paths are the same, the order in which the rules are applied is of no consequence and it would be simpler to think of the two rules as being applied simultaneously as shown in Figure 14b.

In most cases it is possible to apply the rules simultaneously. However, conflict will result when more than one rule applies to the same section of a network. In this case the order of application of the rules will effect the outcome. If one of the conflicting rules is applied the layout of the network will be transformed so that the other rule (or rules) cannot be applied. Therefore, a systematic conflict resolution mechanism should be

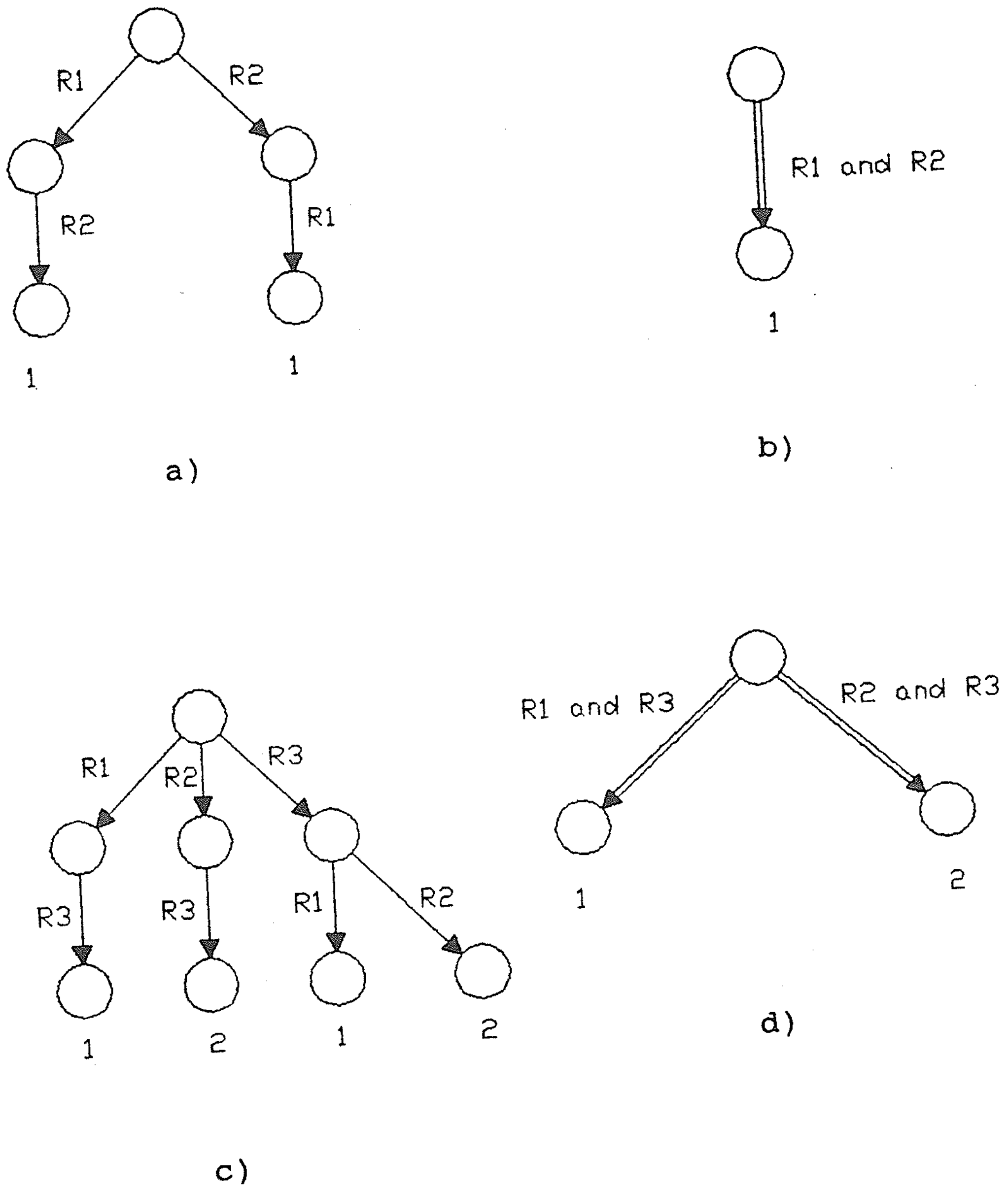


Figure 14: Production system models

applied in this conflicting situation.

Figures 14c and 14d illustrate the conflict problem. Rule 1 and Rule 2 affect the same area of the network while Rule 3 is independent of Rule 1 and Rule 2. Rule 1 and Rule 2 are mutually exclusive. The application of one rule precludes the use of the other. Expanding the system in the conventional manner produces the graph in Figure 14c. Four different paths are produced but only two different objects result. The graph in Figure 14d represents the problem in simpler terms as a conflict between two rule sets.

A simple but effective conflict resolution strategy can be achieved if each rule is assigned a level of priority as it is triggered. This level of priority can be determined from the a **priori** knowledge about the rule's effect on the network. Rules are to be fired in order of priority and each rule must be verified before it is fired. If verification fails the rule does not fire. Verification consists of checking to see if the portion of the network that triggered the rule has not been changed. If a rule of higher priority were triggered by the same portion of a network as a rule of lower priority, the rule of higher priority would be fired first, transforming the portion of the network that was also responsible for triggering the rule of lower priority. Verification of the rule of lower priority would fail and therefore the rule of lower priority would not fire.

Using the example in Figure 14d the procedure is as follows. Rule 1, Rule 2 and Rule 3 are triggered. In this example the levels of priority have been assigned so that Rule 1 has the highest priority followed by Rule 3 and then Rule 2. This creates the firing order. Rule 1 is to be fired first, but the rule must be verified. Since no rules have fired the network has not yet been modified. For this reason the verification of the rule of highest priority is not required, however, the conflict resolution strategy treats all rules in the same manner. The verification of Rule 1 naturally succeeds and Rule 1 fires.

Rule 3 is next to fire and similarly must be verified before firing. The network has been transformed by the firing of Rule 1, but since these two rules are independent the section of the network transformed by Rule 1 has not affected the section of the network that triggered Rule 3. The verification of Rule 3 succeeds and Rule 3 fires.

Now Rule 2 is to be fired, but Rule 1 has transformed the section of the network that triggered Rule 2. The verification of Rule 2 fails and Rule 2 will not fire. The conflict between Rule 1 and Rule 2 is resolved.

In summarizing the technique, a rule only transforms the part of the network that triggers it. If two rules are triggered by the same part of a network and the rule with the highest priority is fired first, the resulting transformation will not allow the rule of lower priority to fire.

Figure 15 shows the scheme that has been adopted. The triggering stage produces a set of rules to be fired. Conflict resolution will eliminate some of these rules so that the remaining rules act independently of each other and can be thought of as acting simultaneously as discussed previously. The effect of firing rules simultaneously is to reduce the number of iterations that would be required to reach the point where the rule base can no longer improve the system. If only one rule in the conflict set was allowed to fire during an iteration, the process of searching the rule base, creating a new conflict set, and firing the best rule would have to be repeated many times. If a conflict set initially contained five independent instantiations of rules, five iterations would be required for all the rules to fire. The process of grouping the instantiations into sets and firing the rules simultaneously will only require one iteration to achieve the same result. Clearly, grouping the rules in this way can achieve a significant improvement in the execution time of the program.

Often the firing of a rule will transform the network to allow new instantiations of rules. Therefore, it is necessary to perform a series of iterations even when rules are grouped in sets and fired simultaneously. The loop labelled "A" in Figure 15 performs these iterations which are analogous to the forward chaining

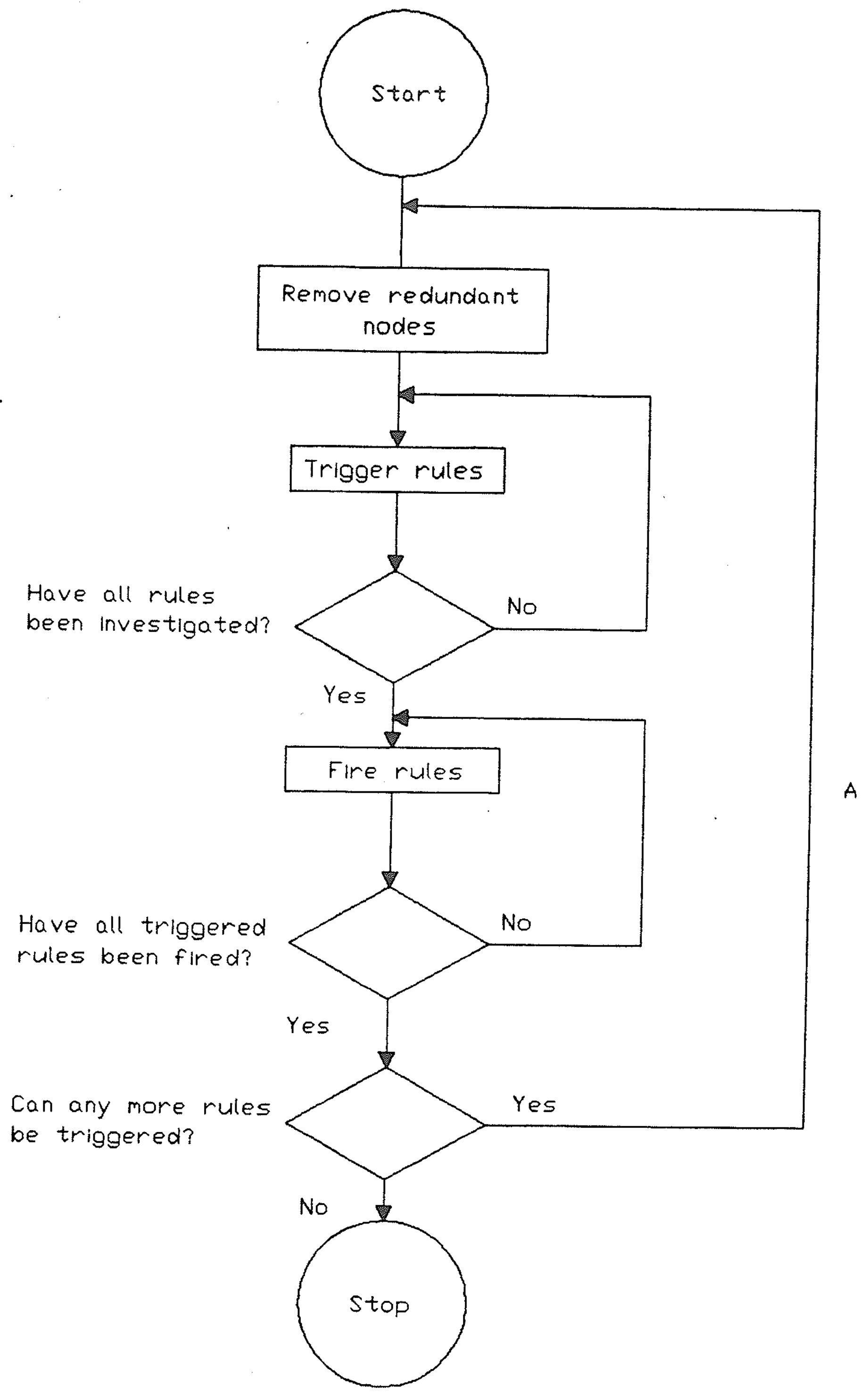

Start
Remove redundant nodes
Trigger rules
Have all rules been investigated?
No
Yes
Fire rules
A
Have all triggered rules been fired?
No
Yes
Can any more rules be triggered?
Yes
No
Stop

Figure 15: Flowchart of control strategy

techniques used in expert systems.

A special set of rules must be incorporated to ensure that the rules in the rule base can trigger correctly. This set of rules is necessary because dummy nodes with only two incident lines are often found such that the line directed toward the node and the line directed away from the node form a 180 degree angle. This type of dummy node does not define a tee or an elbow, but exists as a point on a straight line. Dummy nodes of this type result through the course of manipulations performed by the procedural and cognitive components of the program. During the execution of the procedural component of the program, these nodes may exist to ensure that the MST algorithm creates a desired rectilinear line, but during the execution of the cognitive component of the program these nodes serve no function and can prevent the triggering of rules. In the context of the cognitive component these nodes are referred to as redundant nodes. A simple rule based procedure that identifies and removes redundant nodes is incorporated before the loop that triggers the rules. The procedure is represented in Figure 15 as the box labelled "remove redundant nodes".

Nature of the Rules

It is now possible to describe the nature of the rules in the rule base in the context of the control strategy. The rule base has been designed to fulfil the three objectives described in Chapter 2, to reduce the total length of the system, to improve hydraulics by branching early, and to eliminate unnecessary bends. The system of priorities which are applied to the rules corresponds directly to these three objectives. The three levels of priority are positive, zero, and negative, with rules being fired in order from highest priority to lowest. There are three types of rules. Each rule type corresponds to one of the three objectives stated previously. The rule types have been tentatively referred to as "hooks", "slides" and "elbows".

"Hooks" are rules which decrease the overall system length. Figure 16a shows an example of a "hook". Figure 16b shows the solution to the "hook" problem. The rule will transform the layout in Figure 16a to the layout in Figure 16b if it is applied. It is possible to calculate the decrease in the system length before the rule is applied. In this instance the improvement is equal to the length of the segment connecting node 4 to node 2. The level of priority assigned to a "hook" rule when triggered is equal to the incremental decrease in the system length. The priority is assigned as a positive number. A separate number is calculated and assigned to each instantiation or triggering of a rule. "Hooks" are the only rules to receive a priority greater than zero. Since rules are fired in order of priority from highest to lowest, "hooks" are the most preferred rules, with those "hooks" which result in the greatest improvement in the system length being preferred over

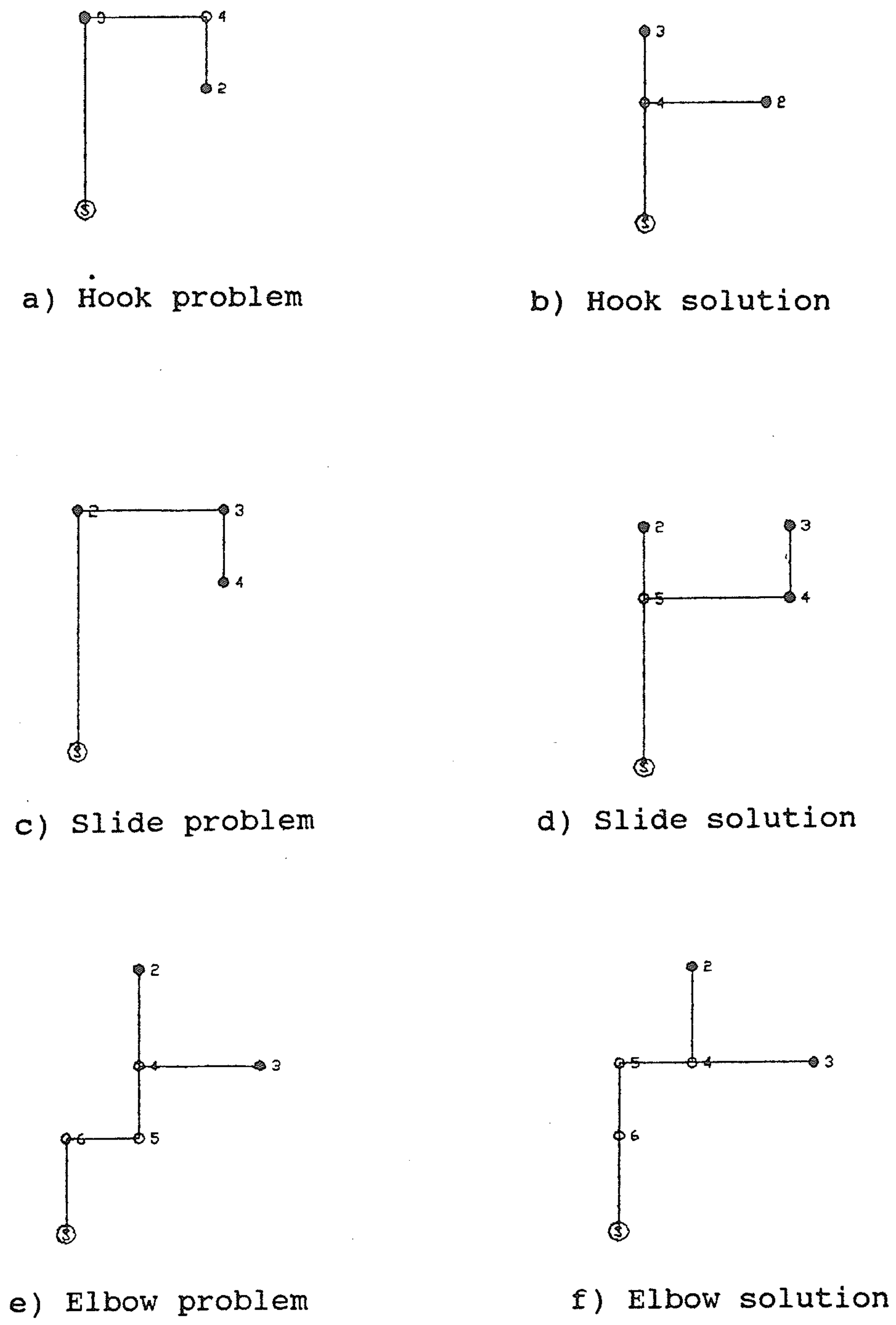


Figure 16: Three basic rule types

those which make smaller improvements.

Rules at the next level of priority are referred to as "slides". "Slides" improve the hydraulics but do not affect the overall system length. An example of a "slide" is shown in Figure 16c and the solution is shown in Figure 16d. This is the same example shown in Figure 1c and 1d and the resulting improvements in hydraulics are the same as explained previously. The term "slide" refers to the transformation process. Two parallel lines are connected by a transverse line. The solution is to slide the transverse line upstream as close to the source as possible to make the branch occur as "early" as possible.

"Slide" rules are all assumed to be equivalent. All such rules are triggered with zero priority. Conflict between rules of this type would be resolved arbitrarily in order of the instantiation of the rules. The conflict condition is unlikely to result due to the nature of the geometry of this pattern. More than one transverse line connecting a pair of parallel lines would result in a loop. Since these systems are not looped, only one transverse line can exist per pair of parallel lines.

Rules in the final category are those that remove superfluous bends. These rules are referred to as "elbows". The example problem shown in Figure 16e and Figure 16f is the same set of graphs in Figure 1e and 1f. The desired transformation is accomplished by "flipping" the bend over. The bend node is a dummy node of degree two. The node has the x coordinate of one the adjacent nodes and the y coordinate of the other adjacent node. In the example shown in Figure 16e, node 5 is the bend node with the x coordinate of node 4 and the y coordinate of node 6. To "flip" the elbow the x and y coordinates of the bend are reassigned to the other pair of coordinates of the adjacent nodes. In Figure 16f node 5 has the x coordinate of node 6 and the y coordinate of node 4.

The priority for "elbow" rules is calculated by taking the negative of the straight line distance from the source to the bend node with the transformed coordinates. The effect of this is to favour bends which "flip" in toward the source in the hope that the resulting layout will be more compact.

An Example Rule

The following example will illustrate how a simple rule is implemented with consideration to the points that have been discussed. Figure 17a shows a fragment of a network and Figure 17b shows the same fragment which has been improved by changing the location of point P2 and making the appropriate changes in the connecting lines. Arrows shown in Figure 17 represent the direction of fluid flow in the pipe segments.

The layout of the pipe network is described using three types of objects. These objects are maintained as relations in PROLOG's dynamic data base. The relations are "count", "point" and "lin" and are explained as follows:

1. count - This relation occurs only once in the data base. It has a single integer argument which is set equal to the number of nodes in a network.

2. point - This relation occurs once for every node in the network. The relation has five arguments which are as follows:

   i. node label - an integer number selected arbitrarily (each one must be unique).

   ii. X-coordinate - a real number.

   iii. Y-coordinate - a real number.

   iv. sink - an integer number used as a flag (1 indicates that the node is a sink and its position cannot be altered; 0 indicates that the node is a dummy and can be moved or deleted).

   v. degree - an integer number between 1 and 4 which indicates the number of lines incident at the node. If orthogonal networks are used the number of lines incident on a node has the maximum of four.

3. lin - This relation occurs once for every pipe segment or line in the network. There are two arguments to this relation. The arguments are the labels of the two points connected by the line. The direction of fluid flow in each segment is assumed to be from the node specified as the first argument to the node specified as the second.

The antecedent clauses of the example rule will describe the features of all layouts of the type shown in Figure 17a and the consequent clauses will transform that layout to the one shown in Figure 17b. The definition of the rule begins as follows. The point P1 is assumed to have the coordinates X1 and Y1 and the point P3

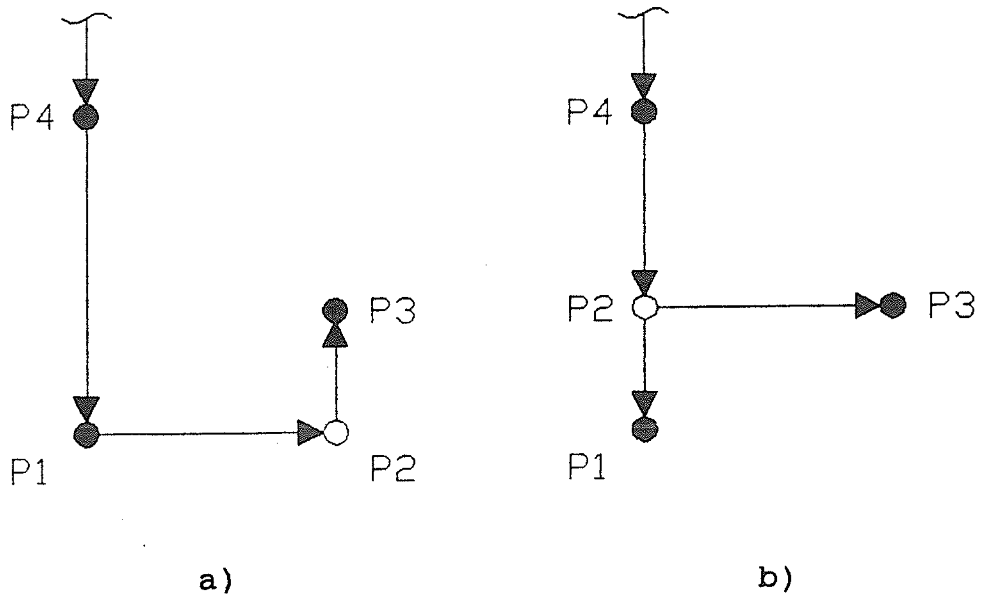


Figure 17: Example rule

has the coordinates X3 and Y3. On examining Figure 17a it is apparent that the point P2 has the x-coordinate of point P3 and the y-coordinate of point P1. Therefore, P2's coordinates are X3 and Y1. Similarly P4 has the same x-coordinate as P1 and a unique y-coordinate. P4's coordinates are X1 and Y4. The rule must determine whether P2 is a dummy node of degree two (having two incident lines). The degree and type of the other two nodes are not important.

The data that are required to describe the connecting lines are simply that a line connects P4 to P1, P1 to P2, and P2 to P3. A set of clauses must also determine if the y-coordinate of node P3 (Y3) falls between Y4 and Y1. The point and line data can be summarized in the PROLOG clauses below which constitute the complete set of antecedent clauses for this rule . These PROLOG clauses use the representation scheme discussed previously.

```
point(P1, X1, Y1, S1, D1),
point(P2, X3, Y1, 0, 2),
point(P3, X3, Y3, _, _),
point(P4, X1, Y4, _, _),
lin(P4, P1),
lin(P1, P2),
lin(P2, P3),
Y1 > Y3, Y3 > Y4,
```

It is necessary to obtain information about the type of node P1 (S1) and the degree of node P1 (D1). The consequent clauses will need this information to retract and reassert node P1 with the proper degree since P1 changes from degree 2 in Figure 17a to degree 1 in Figure 17b. The necessary consequent clauses for this rule are shown below:

```
retract(point(P2, X3, Y1, 0, 2)),
assertz(point(P2, X1, Y3, 0, 3)),
retract(lin(P4, P1)),
assertz(lin(P4, P2)),
retract(lin(P1, P2)),
assertz(lin(P2, P1)),
D1new = D1 - 1,
retract(point(P1, X1, Y1, S1, D1)),
assertz(point(P1, X1, Y1, S1, D1new)).
```

In order to permit separate triggering and firing of the rules, the antecedent and consequent clauses must be grouped as separate procedures. Data that was established by the antecedent clauses will be required by the consequent clauses. These data consist of the node number, coordinates, degree and sink flag for the different nodes in the pattern. The data are passed between the two procedures by means of a list. In this list the functor "i" is used for integer data types and "r" is used for real data. Only the x and y coordinates are represented as real data.

In addition to this list of parameters which must be passed to the consequent clauses, two pieces of information must be passed along another path between the antecedent clauses and the control strategy. The first of these pieces of information is a label that identifies which set of consequent clauses are to be executed, in other words, which set of consequent clauses correspond to this rule. The second piece of information is the priority that the rule is to have. In summary, three pieces of information must be generated by the antecedent clauses - a label used to select the proper set of consequent clauses, a level of priority for resolution of possible conflict, and a list of parameters to be used by the consequent clauses.

The procedure containing the antecedent clauses is called "find". The procedure "find" is required to calculate the reduction in length to determine the priority of this instance of the rule. By inspecting Figure 17a and 17b, it is clear that the reduction in length is equal to Y3 - Y1. "Find" will pass the priority (the length Y3 - Y1) along with the parameter list and the label of the required consequent clauses. In this example rule the label "rexample" (for Rule **EXAMPLE**) will be used. The label, the priority and the parameter list are passed to a procedure called "trigger". As the name implies, "trigger" triggers the rule by placing all of the data passed to it into the conflict set. Each entry passed to "trigger" appears as a relation, or fact, in the dynamic data base.

The consequent clauses for the example rule are grouped in a procedure called "execute" which is responsible for firing the rule. The consequent clauses for all rules are grouped in procedures that have the name "execute". This is why a label is required to select the correct set of consequent clauses. The label "rexample" identifies the proper set of consequent clauses in this instance.

As part of the conflict resolution strategy the procedure "execute" must establish that data describing the part of the network that triggered the rule have not changed since the rule was triggered. This process was referred to previously as verifying the rule. If a rule of higher priority was triggered by the data, a change of this type will occur and verification will not succeed. The antecedent clauses are placed at the beginning of the procedure "execute" to determine if a change has occurred. If any of these clauses are not satisfied the rule will not fire.

The rule has the following form:

```
find :-
     point(P1, X1, Y1, S1, D1),
     point(P2, X3, Y1, 0, 2),
     point(P3, X3, Y3, _, _),
     point(P4, X1, Y4, _, _),
     lin(P4, P1),
     lin(P1, P2),
     lin(P2, P3),
     Y4 > Y3, Y3 > Y1,
     Priority = Y3 - Y1,
     trigger(Priority, rexample,
          [i(P1), r(X1), r(Y1), i(S1), i(D1),
           i(P2),
           i(P3), r(X3), r(Y3),
           i(P4), r(Y4)]),
     fail.

find.

execute(rexample,
          [i(P1), r(X1), r(Y1), i(S1), i(D1),
           i(P2),
           i(P3), r(X3), r(Y3),
           i(P4), r(Y4)]) :-
     point(P1, X1, Y1, S1, D1),
     point(P2, X3, Y1, 0 2),
     point(P3, X3, Y3, _, _),
     point(P4, X1, Y4, _, _),
            lin(P4, P1),
            lin(P1, P2),
            lin(P2, P3),
            Y4 > Y3, Y3 > Y1,
            retract(point(P2, X3, Y1, 0, 2)),
            assertz(point(P2, X1, Y3, 0, 3)),
            retract(lin(P4, P1)),
            assertz(lin(P4, P2)),
            retract(lin(P1, P2)),
            assertz(lin(P2, P1)),
            D1new = D1 - 1,
            retract(point(P1, X1, Y1, S1, D1)),
            assertz(point(P1, X1, Y1, S1, D1new)),
            !.

    execute(_, _).
```

The second "find" clause and the second "execute" clause are required for similar reasons. The first "find" clause ends with a fail predicate. The procedure will fail even if all the sub-clauses can be satisfied. In this way the procedure will consider every possible combination that could satisfy it and every time the procedure is satisfied another instance of the rule is triggered. The second "find" clause allows "find" to finally succeed so the flow of control can proceed.

Unlike the first "find" clause which always fails, the first "execute" clause will only fail if a particular triggered instance of the rule is to be eliminated by the conflict resolution mechanism. The second "execute" is provided to ensure that "execute" always succeeds regardless of whether or not the rule actually fires. "Execute" must succeed in order for the program to continue, either to fire other rules, to perform an another iteration, or simply for the program to terminate properly. The first "execute" clause ends with a cut (!) predicate to ensure that the backtracking algorithm will not return control to this point at a later time.

If other rules are to be included in the rule base they should be grouped into a "find" procedure and an "execute" procedure. The "find" and "execute" procedures that always succeed must be placed physically last in the list of these clauses.

Structure of the Rule Base

The problems identified by rules in each of the three categories, "hooks", "slides" and "elbows", were found to occur in different variations. For example, "hooks" occur in eighteen different variations. The different variations can be organized into a tree structure. If the rule base is structured in accordance with this tree, it is possible to focus the search of the rule base which drastically reduces the program's execution time.

The concept of focusing the search was applied to all three categories of rules. The structure of only one of these categories, the "hooks", is described here, since the process is essentially the same for each category. The identification of a "hook" begins as follows. First, a dummy node of degree two must be found. The pattern described in the previous section as the example rule was a "hook". In the case of this pattern the dummy node of degree two is node P2 in Figure 17a. Since the node is of degree two there are two incident lines. One of these lines is directed to the node (i.e., P1 to P2 in Figure 17a) and the other line is directed away from the node (i.e., P2 to P3 in Figure 17a). These two lines must form a 90 degree angle. If all these conditions are met then the search begins at the root of the tree graph in Figure 18. The root of the graph is labelled "elbow". The first branch in the tree indicates two possible configurations of this 90 degree elbow - the line connecting P1 to P2 may be a vertical or a horizontal line.

The next branch occurs as a result of the third line that is required to repeat the pattern. This line connects point P4 to the elbow. The line can connect to point P1 or to point P3. The line can be directed toward the elbow - this is referred to as "in"; or the line can be directed away from the elbow - referred to as "out". With two possible points of connection and two possible directions of flow in the lines indicated, four combinations should exist. However, the combination of a line directed "in" at point P3 cannot exist. It has already been established that a line is directed at P3 from P2. In a tree network with a single source, two lines cannot be directed at a single point without forming a loop. Therefore only three combinations are possible: P1 "in", shown as i1 in Figure 18; P1 "out", shown as o1; and P3 "out", shown as o3.

The final consideration is the relative length of the line

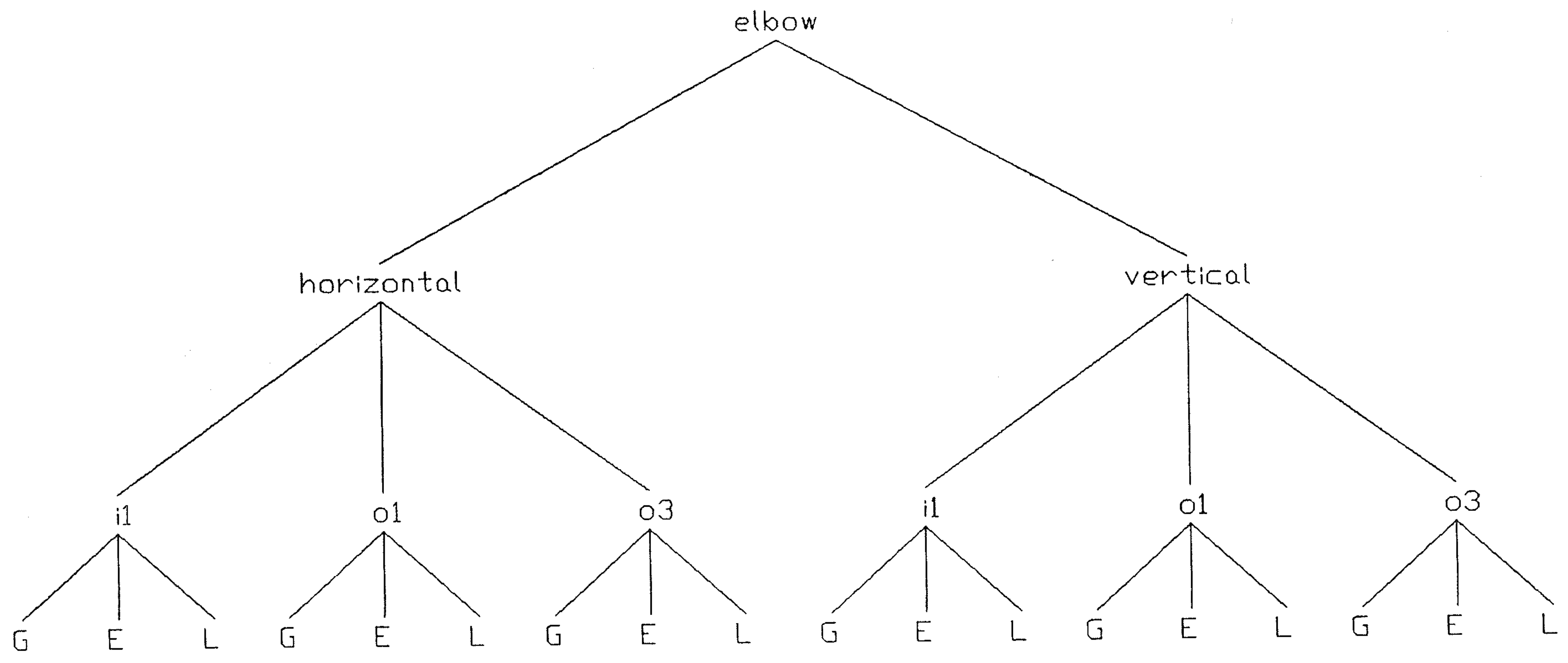


Figure 18: Structure of the rule base

connecting point P4. The corrective procedure that must be specified in the consequent clauses, depends on whether the length of this line is greater than (g), equal (e), or less than (l) the length of the line that is opposite and parallel to it. This completes the tree.

Each rule is given a name that is an abbreviation of its path on the tree. The name of the rule featured in the example in the previous section is "gilh". The name means that the rule identifies a horizontal elbow connected to a line directed in at P1 and that line is longer than the line that connects P2 to P3.

This tree structuring of the rules applies only to the antecedent clauses of the rules. It was discovered that the consequent clauses occur as distinct types and could be written generically so that fewer sets of consequent clauses were required than rules. For example, the eighteen rules required to find all the "hooks" required only seven sets of consequent clauses.

The search of the tree is accomplished by PROLOG's unification and backtracking algorithms. A search performed in this manner is highly focused and will avoid inapplicable rules, reducing the execution time of the program.

CHAPTER 5: DEMONSTRATION OF THE MODEL

The use of the model is demonstrated through the following two examples.

Example 1

Table 1 shows the set of x and y coordinates for this example. The problem is representative of a typical rural gas distribution system comprised of a single source and 28 sinks. The sinks are all considered to be the same type of load, namely, an intermittent load of 6 cubic meters per hour. The x and y coordinates in Table 1 are taken from the Universal Transverse Mercator System and given in meters. A "1" shown in the column labelled "sink" in Table 1 indicates that the node is a sink.

For the purpose of comparison, the layouts that follow will undergo hydraulic design and cost analysis using a computer program similar to the program used by Davidson and Goulter (1989). The program uses the MST algorithm to construct a layout from input data describing the nodes. The program selects the diameter of pipe for each segment with the objective of selecting the smallest diameter that will produce a pressure gradient below a certain specified allowable maximum. For this example 18 kPa/km was used as the maximum allowable gradient. While individual pipe segments are being sized, the hydraulic profile of the system is computed.

TABLE 1: NODE COORDINATES FOR EXAMPLE 1

NODE	X (m)	Y (m)	SINK
S	745967.5	5452799.0	
2	745351.1	5449946.0	1
3	743467.1	5448719.0	1
4	748789.2	5448974.0	1
5	748823.9	5448269.0	1
6	750196.5	5450686.0	1
7	751874.9	5449961.0	1
8	754009.4	5448723.0	1
9	753775.5	5446768.0	1
10	744538.9	5443649.0	1
11	743837.8	5443005.0	1
12	744960.2	5441010.0	1
13	745868.6	5441052.0	1
14	746816.3	5443452.0	1
15	747739.6	5442789.0	1
16	747439.9	5442146.0	1
17	747166.1	5439477.0	1
18	750561.6	5439203.0	1
19	749234.6	5438837.0	1
20	751255.2	5437691.0	1
21	748052.3	5437874.0	1
22	752144.3	5435890.0	1
23	751092.7	5434093.0	1
24	750284.9	5434061.0	1
25	746868.1	5432885.0	1
26	746238.7	5432853.0	1
27	743672.0	5437664.0	1
28	742793.8	5436483.0	1
29	741954.1	5437152.0	1

An estimate of the cost of the system is produced based on the total length of the various sizes of pipe that are required. The cost of crossings can also be estimated. Road and cable crossings are estimated on the basis of the total length of the system, while the number of river and railway crossings must be stated explicitly.

In previous work by Davidson and Goulter (1989), the cost analysis program established the number of river and railway crossings by grouping the nodes in sets. For example, the nodes on one side of a river will be grouped in one set and the nodes on another side will be grouped in another set. A crossing is indicated by a line with end points in different sets. In the previous work all the nodes in a system were entered manually. As each node was entered the user was prompted to provide the number of the set for that node.

A problem related to this set numbering is created by the approach described in the study through the automated selection of dummy nodes. Both the procedural and cognitive components of the program add dummy nodes to the system without any user interaction. In order to assess in which set a particular dummy node should belong, the program would have to know the side of the river, or railroad, where the dummy node is located. The task is particularly difficult in the case of rivers with many meander bends and nodes located within meander bends. The program based on the new approach does not possess the capability to recognise where a node is in relation to a river or railroad at this point.

Since no simple solution to this problem has been found, the method relies on user interaction to ensure that river and railway crossings have been accommodated appropriately. User interaction takes two forms, namely, modification and selection. Modification involves the manual input of additional dummy nodes to minimize the number of crossings in a layout. Selection involves selecting the most appropriate layout in the case where several alternatives have been generated. Of course, any layout that is selected from a set of alternatives can be modified as well. The example presented here is not a case in which multiple alternatives have been generated. Multiple alternatives are demonstrated in Example 2. In Example 1 the iterative processes converge on a single solution.

Figure 19 shows the nodes for Example 1 from Table 1 graphically and Figure 20 shows the Minimal Spanning Tree for this set of nodes. The results of the hydraulic design and cost analysis program for the MST layout are given in Table 2. The headings are explained as follows:

FROM	-	the origin node number of a pipe segment
TO	-	the destination node number of a pipe segment
LENGTH	-	length of the segment in m
NUM	-	the number of intermittent loads on that segment (used to calculate the coincidence factor that will

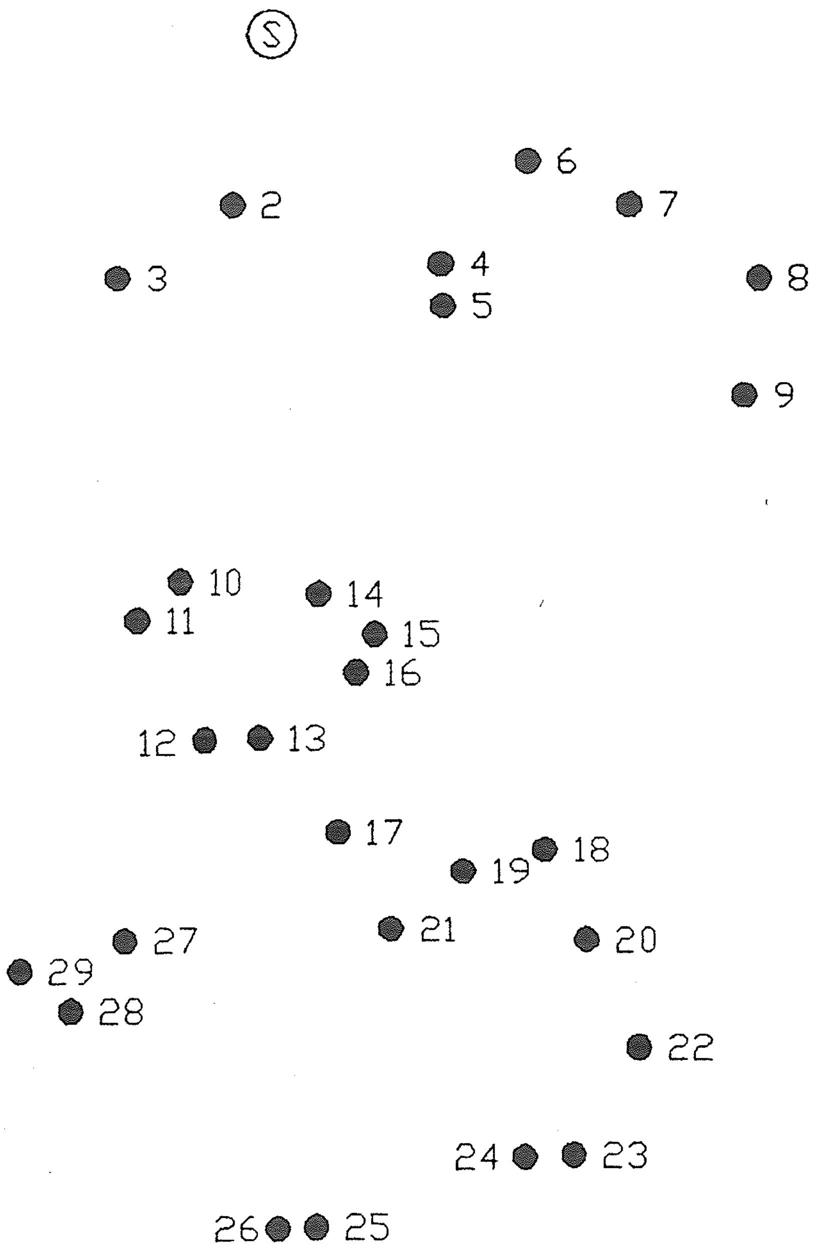


Figure 19: Example 1 nodes

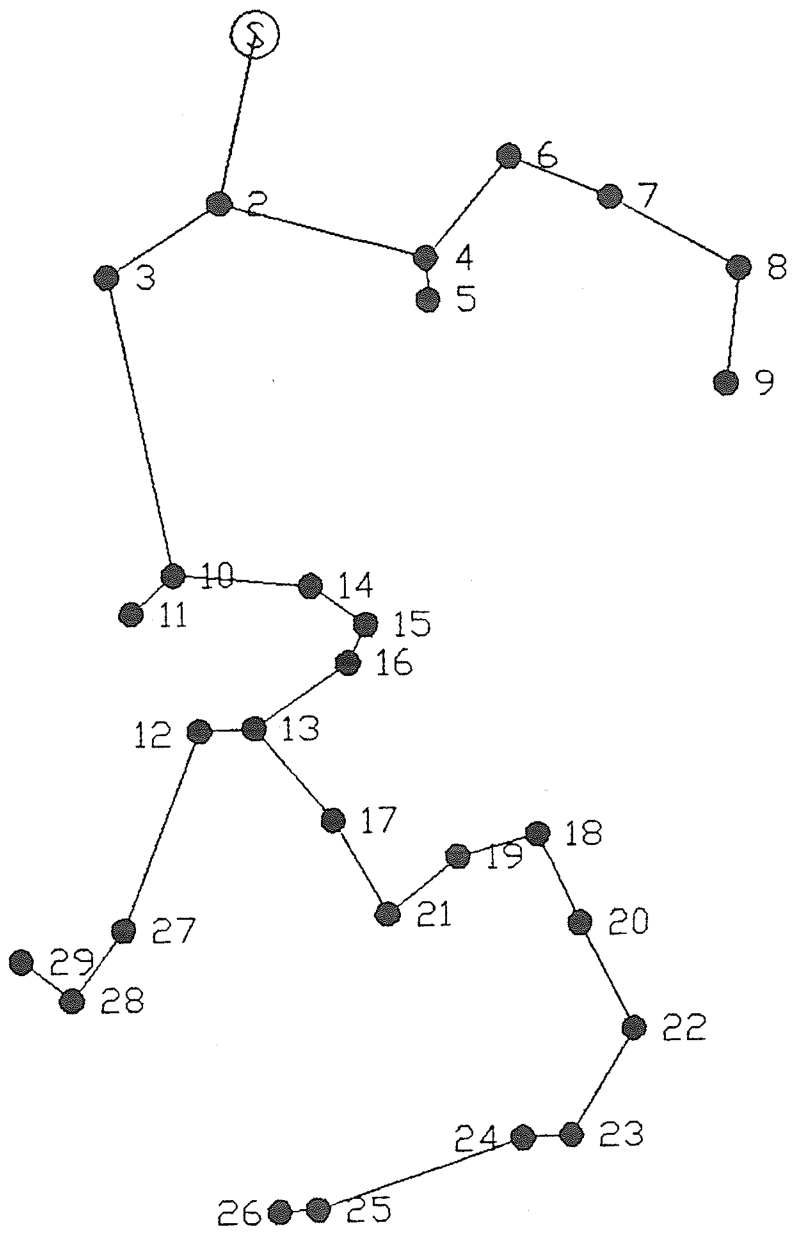


Figure 20: Minimal Spanning Tree for Example 1

		be applied to the load)
LOAD	-	the factored load in m^3/h
SIZE	-	nominal diameter of the pipe in mm (chosen by the program)
PI	-	inlet pressure of the pipe segment in kPa
PO	-	outlet pressure of the pipe segment in kPa
PD	-	average pressure gradient of the pipe segment in kPa/km
WO	-	warning flag (set to 1 if the outlet pressure falls below 140 kPa)
WD	-	warning flag (set to 1 if the pressure gradient is above the specified gradient)

Figure 21 shows the layout produced by the procedural component of the program as described in Chapter 3. To aid in the clarity of the figure the dummy nodes have not been plotted. Table 3 is a description of the locations of the dummy nodes, from 30 to 58, that are added by the program to make this layout rectilinear. Using the method described previously by Davidson and Goulter (1989) these 29 nodes would have to be entered manually and their locations would have to be determined using some combination of the Boxplot Method and user judgment - a process that would be both difficult and time consuming even though it might assist in locating the dummy nodes in the correct set as far as crossings of rivers and railways are concerned.

In Figure 22 the rectilinear layout is shown superimposed on the MST layout. It is clear from this figure that much of the general

TABLE 2: HYDRAULIC PROFILE FOR MST LAYOUT OF EXAMPLE 1

FROM	TO	LENGTH	NUM	LOAD	SIZE	PI	PO	PD	WO	WD
****	****	*******	***	*****	****	*****	*****	*****	****	****
1	2	2918.8	28	106.0	60.3	550.0	531.2	6.4	0	0
2	3	2248.3	21	82.6	48.3	531.2	503.6	12.3	0	0
2	4	3572.8	6	33.9	33.4	531.2	477.6	15.0	0	0
4	5	705.9	1	6.0	26.7	477.6	475.9	2.3	0	0
4	6	2216.2	4	24.0	33.4	477.6	458.4	8.7	0	0
6	7	1828.3	3	18.0	26.7	458.4	430.0	15.5	0	0
7	8	2467.5	2	12.0	26.7	430.0	409.0	8.5	0	0
8	9	1968.9	1	6.0	26.7	409.0	403.8	2.6	0	0
3	10	5182.0	20	79.2	48.3	503.6	439.3	12.4	0	0
10	11	952.0	1	6.0	26.7	439.3	436.9	2.5	0	0
10	14	2285.9	18	72.4	48.3	439.3	412.9	11.5	0	0
14	15	1136.7	17	68.9	48.3	412.9	400.6	10.9	0	0
15	16	709.4	16	65.3	48.3	400.6	393.4	10.1	0	0
16	13	1914.6	15	61.7	48.3	393.4	375.4	9.4	0	0
13	12	909.4	4	24.0	33.4	375.4	365.8	10.5	0	0
13	17	2040.6	10	42.9	48.3	375.4	365.0	5.1	0	0
17	21	1831.6	9	41.7	48.3	365.0	355.8	5.0	0	0
21	19	1524.9	8	39.8	48.3	355.8	348.7	4.7	0	0
19	18	1376.5	7	37.2	48.3	348.7	342.9	4.2	0	0
18	20	1663.5	6	33.9	48.3	342.9	337.0	3.6	0	0
20	22	2008.5	5	30.0	33.4	337.0	301.4	17.7	0	0
22	23	2082.1	4	24.0	33.4	301.4	274.9	12.7	0	0
23	24	808.4	3	18.0	33.4	274.9	268.5	7.9	0	0
12	27	3585.4	3	18.0	33.4	365.8	342.8	6.4	0	0
27	28	1471.7	2	12.0	26.7	342.8	327.8	10.2	0	0
28	29	1073.6	1	6.0	26.7	327.8	324.4	3.1	0	0
24	25	3613.5	2	12.0	26.7	268.5	221.9	12.9	0	0
25	26	630.3	1	6.0	26.7	221.9	219.2	4.2	0	0

PIPE SIZE	TOTAL LENGTH	COST
*********	************	*******
26.7 mm	14712.	23833.
33.4 mm	15183.	29910.
48.3 mm	21914.	63113.
60.3 mm	2919.	11880.
88.9 mm	0.	0.
TOTAL	54728.	128736.

TOTAL NUMBER OF ROAD CROSSINGS	31	COST	17471.
TOTAL NUMBER OF RAIL CROSSINGS	0	COST	0.
TOTAL NUMBER OF CABLE CROSSINGS	37	COST	3330.
TOTAL NUMBER OF CREEK CROSSINGS	0	COST	0.

TOTAL COST OF SYSTEM $149536.

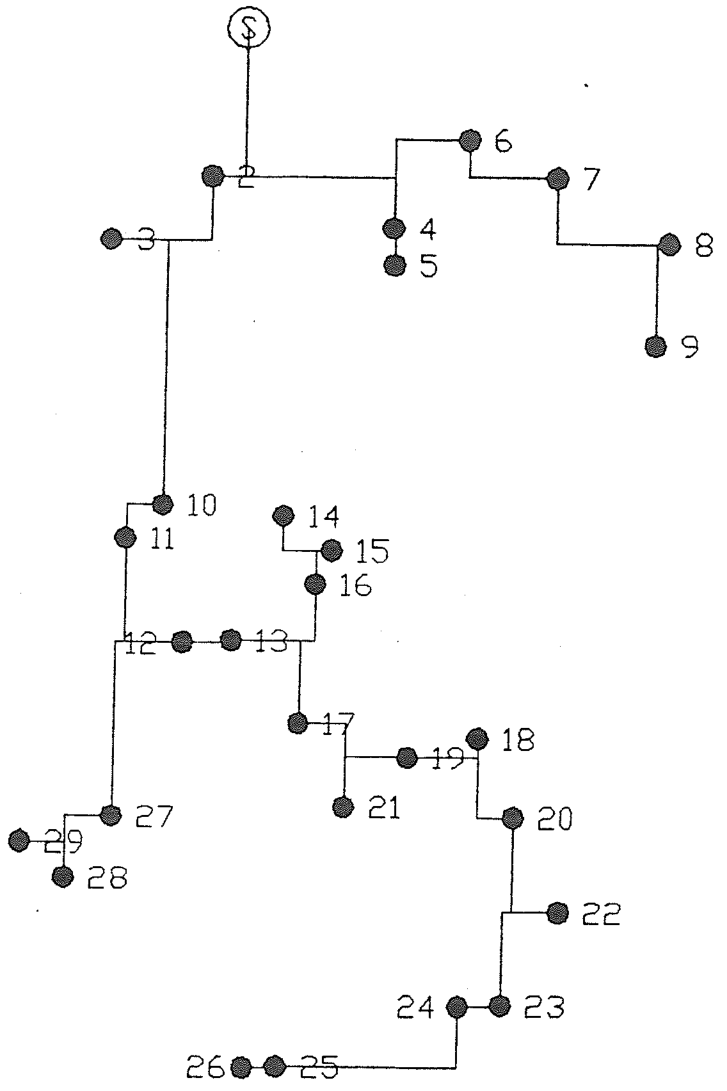


Figure 21: Rectilinear layout produced by the procedural component for Example 1

TABLE 3: DUMMY NODE COORDINATES FOR EXAMPLE 1

NODE	X (m)	Y (m)	SINK
30	745967.5	5449946.0	0
31	745351.1	5448719.0	0
32	745351.1	5448974.0	0
33	748789.2	5449946.0	0
34	748823.9	5448974.0	0
35	748789.2	5450686.0	0
36	750196.5	5449961.0	0
37	751874.9	5448723.0	0
38	753775.5	5448723.0	0
39	744538.9	5448719.0	0
40	743837.8	5443649.0	0
41	746816.3	5442789.0	0
42	747439.9	5442789.0	0
43	747439.9	5441052.0	0
44	745868.6	5441010.0	0
45	747166.1	5441052.0	0
46	748052.3	5439477.0	0
47	748052.3	5438837.0	0
48	750561.6	5438837.0	0
49	750561.6	5437691.0	0
50	751255.2	5435890.0	0
51	751092.7	5435890.0	0
52	751092.7	5434061.0	0
53	743672.0	5441010.0	0
54	742793.8	5437664.0	0
55	742793.8	5437152.0	0
56	750284.9	5432885.0	0
57	746868.1	5432853.0	0
58	743837.8	5441010.0	0

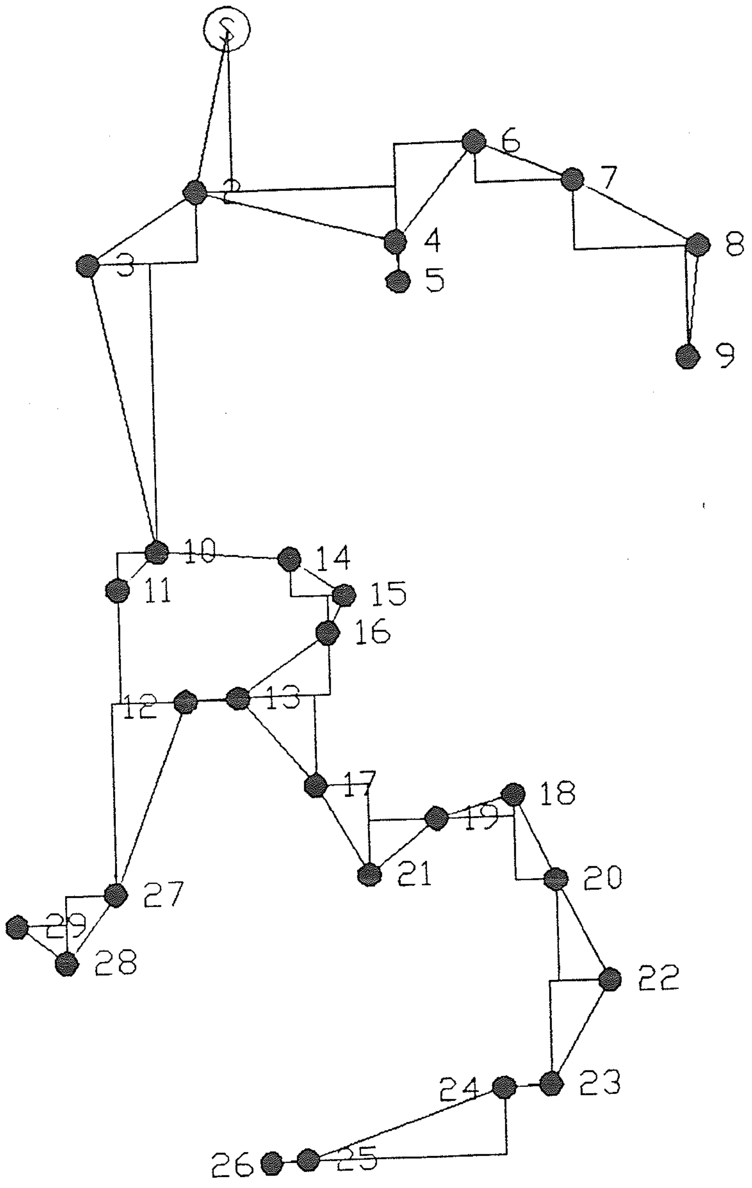


Figure 22: The rectilinear layout for Example 1 generated by the procedural component superimposed on the MST

form of the two layouts is similar. In general a rectilinear layout requires greater total system length than the MST. The results of the hydraulic design and cost analysis for this rectilinear layout are shown in Table 4. (The complete hydraulic profile is included in Appendix A.) In this case the change from a diagonal layout to a rectilinear layout involves an increase in total length of 13.8% with a corresponding increased cost is 10.0%.

TABLE 4: RESULTS FROM PROCEDURAL COMPONENT FOR EXAMPLE 1

PIPE SIZE	TOTAL LENGTH	COST
26.7 mm	23281.	37715.
33.4 mm	14283.	28138.
48.3 mm	21886.	63032.
60.3 mm	2853.	11612.
88.9 mm	0.	0.
TOTAL	62303.	140496.

TOTAL NUMBER OF ROAD CROSSINGS	36	COST	20289.
TOTAL NUMBER OF RAIL CROSSINGS	0	COST	0.
TOTAL NUMBER OF CABLE CROSSINGS	42	COST	3780.
TOTAL NUMBER OF CREEK CROSSINGS	0	COST	0.

TOTAL COST OF SYSTEM $164564.

Next, the cognitive component of the program is applied to improve the layout. For illustration purposes the control strategy of this portion of the program has been modified slightly to permit the results from each individual iteration to be examined and plotted.

An iteration of the cognitive component involves three stages. The first stage is to search for and remove all redundant nodes. The next stage is to trigger all rules that can be triggered. The final stage is to fire the rules in order of priority and resolve conflicts. Figure 23 is the contents of a spill file that traces the execution of the rule base. The file shows that one redundant node was removed; one "hook" was triggered and fired; four "slides" were triggered, only two of which fired; and ten "elbows" were triggered, only four of these fired. Figure 24 shows the changes generated by the cognitive component superimposed on the previous layout. The rules that caused each change are identified by their type and name. Table 5 shows the results from the hydraulic design and cost analysis of the improved layout which is shown in Figure 25. The result is a 1.4% improvement in cost and a 1.1% decrease in the total system length.

TABLE 5: RESULTS FROM THE FIRST ITERATION OF THE COGNITIVE COMPONENT FOR EXAMPLE 1

PIPE SIZE	TOTAL LENGTH	COST
26.7 mm	23648.	38309.
33.4 mm	13438.	26472.
48.3 mm	21640.	62322.
60.3 mm	2853.	11612.
88.9 mm	0.	0.
TOTAL	61578.	138715.

TOTAL NUMBER OF ROAD CROSSINGS	35	COST	19725.
TOTAL NUMBER OF RAIL CROSSINGS	0	COST	0.
TOTAL NUMBER OF CABLE CROSSINGS	42	COST	3780.
TOTAL NUMBER OF CREEK CROSSINGS	0	COST	0.

TOTAL COST OF SYSTEM $162220.

```
1 redundant node removed
Search terminated

1 e3i1v triggered    (elbow)
1 e1i1h triggered    (elbow)
1 lo3v triggered     (hook)
1 esi1v triggered    (elbow)
1 e3i1v triggered    (elbow)
1 e3i1h triggered    (elbow)
1 e3o3v triggered    (elbow)
1 e3o3h triggered    (elbow)
1 e3i1h triggered    (elbow)
1 esi1v triggered    (elbow)
1 e1i1v triggered    (elbow)
1 ghs triggered      (slide)
1 ghs triggered      (slide)
1 lhs triggered      (slide)
1 gvs triggered      (slide)
Search terminated
```

```
lo3 entered    (hook)
lo3 fired

gs entered     (slide)
rule is not fired

gs entered     (slide)
rule is not fired

ls entered     (slide)
ls fired

gs entered     (slide)
gs fired

ei1 entered    (elbow)
ei1 fired

ei1 entered    (elbow)
ei1 fired

esi1 entered   (elbow)
rule is not fired

ei1 entered    (elbow)
rule is not fired

ei1 entered    (elbow)
rule is not fired

eo3 entered    (elbow)
rule is not fired

eo3 entered    (elbow)
eo3 fired

ei1 entered    (elbow)
rule is not fired

esi1 entered   (elbow)
esi1 fired

ei1 entered    (elbow)
rule is not fired

All rules exhausted
Iterations complete
```

Figure 23: Spill file from the first iteration of the cognitive component for Example 1

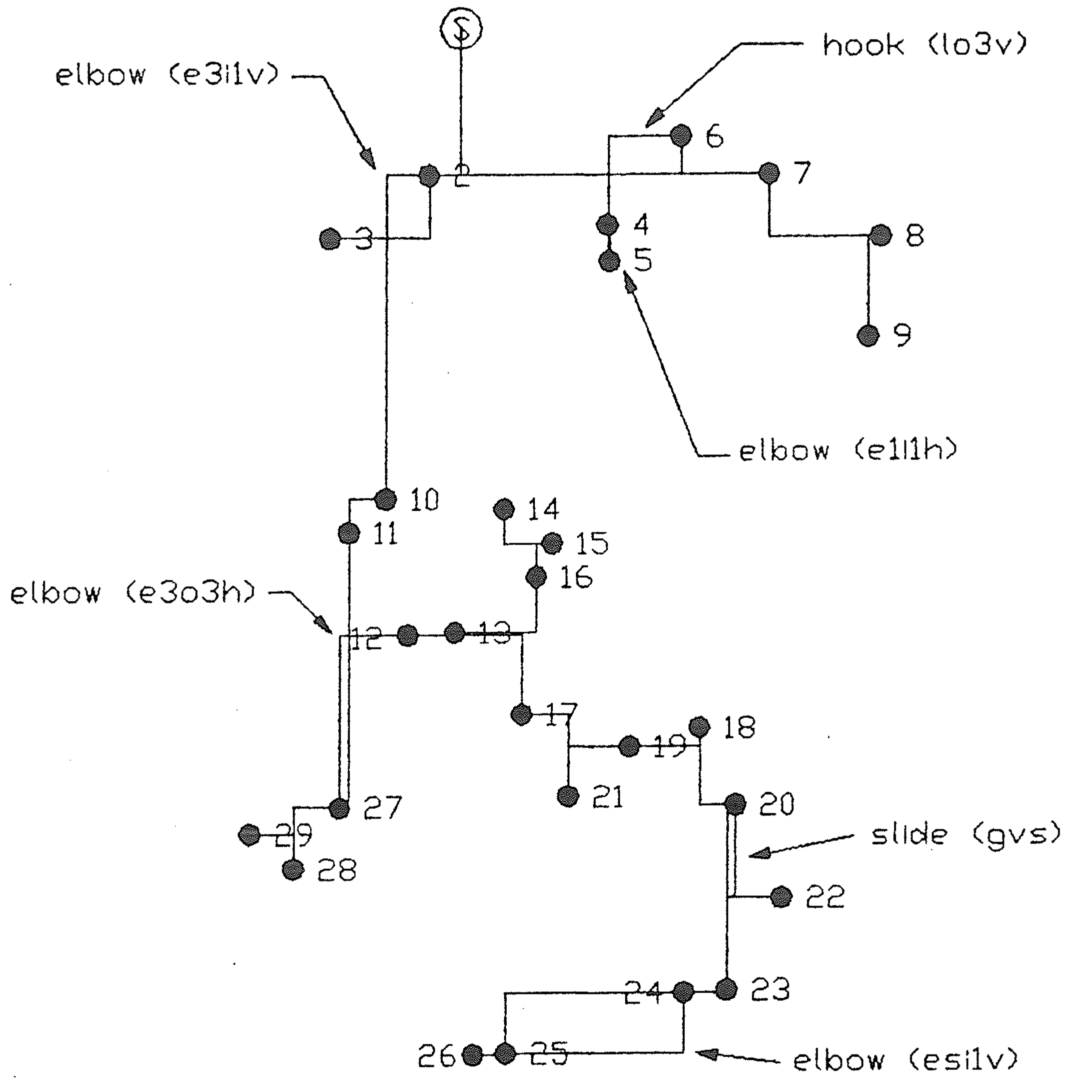


Figure 24: Modifications produced by the first iteration of the cognitive component for Example 1

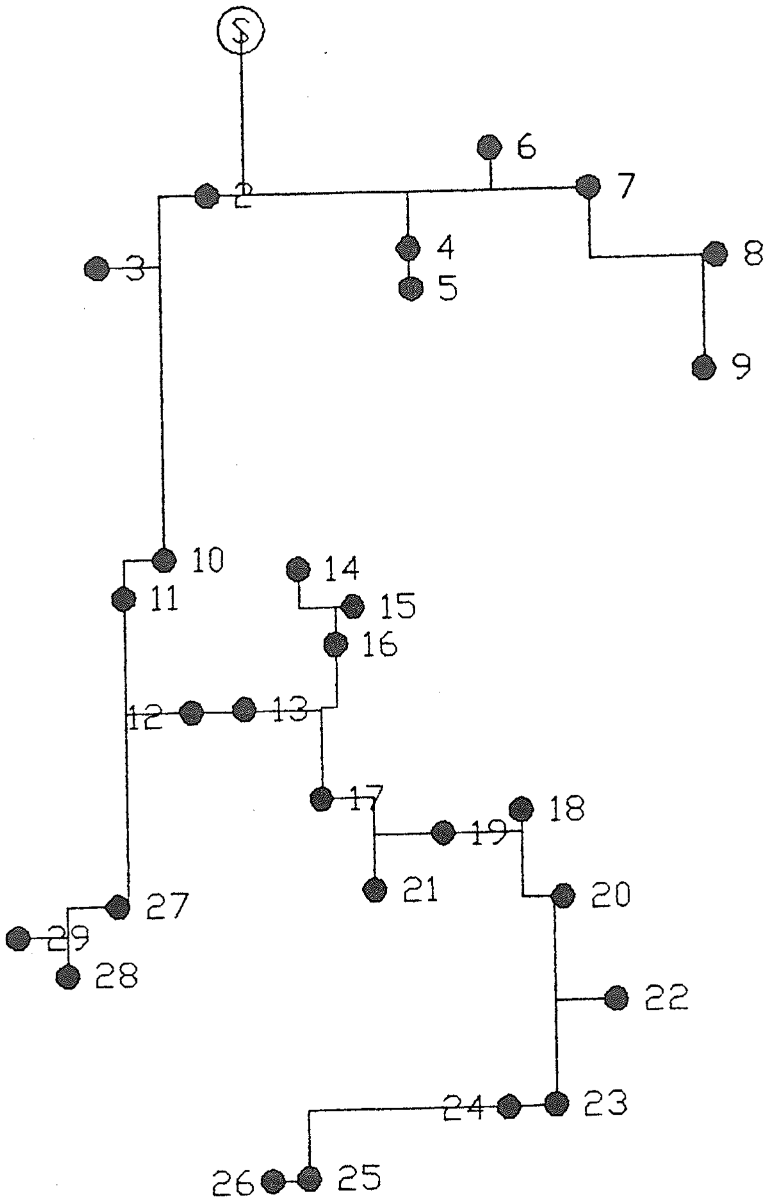


Figure 25: Layout produced by the first iteration of the cognitive component for Example 1

Figure 26 shows the spill file from the next iteration. A single redundant node is removed. No "hooks" are identified. One "slide" is triggered and fired. Three "elbows" are triggered and two are fired. Figure 27 shows the resulting modifications graphically. The "elbows" do not result in any improvement in the system length or cost. The "slide" which is not visible at the scale of Figure 27 contributes a small improvement in the hydraulics resulting in the highly insignificant improvement in cost of five dollars. The final layout is shown in Figure 28 and the results are shown in Table 6.

TABLE 6: RESULTS FROM THE SECOND ITERATION OF THE COGNITIVE COMPONENT FOR EXAMPLE 1

PIPE SIZE	TOTAL LENGTH	COST
*********	************	*******
26.7 mm	23663.	38333
33.4 mm	13423.	26442.
48.3 mm	21640.	62322.
60.3 mm	2853.	11612.
88.9 mm	0.	0.
TOTAL	61578.	138710

TOTAL NUMBER OF ROAD CROSSINGS	35	COST	19725.
TOTAL NUMBER OF RAIL CROSSINGS	0	COST	0.
TOTAL NUMBER OF CABLE CROSSINGS	42	COST	3780.
TOTAL NUMBER OF CREEK CROSSINGS	0	COST	0.

TOTAL COST OF SYSTEM $162215.

```
1 redundant node removed
Search terminated

1 e3i1v triggered   (elbow)
1 e3i1h triggered   (elbow)
1 e3o3v triggered   (elbow)
1 lhs triggered     (slide)
Search terminated

ls entered     (slide)
ls fired

ei1 entered    (elbow)
rule is not fired

eo3 entered    (elbow)
eo3 fired

ei1 entered    (elbow)
ei1 fired

All rules exhausted
Iterations complete
```

Figure 26: Spill file from the second iteration of the cognitive component for Example 1

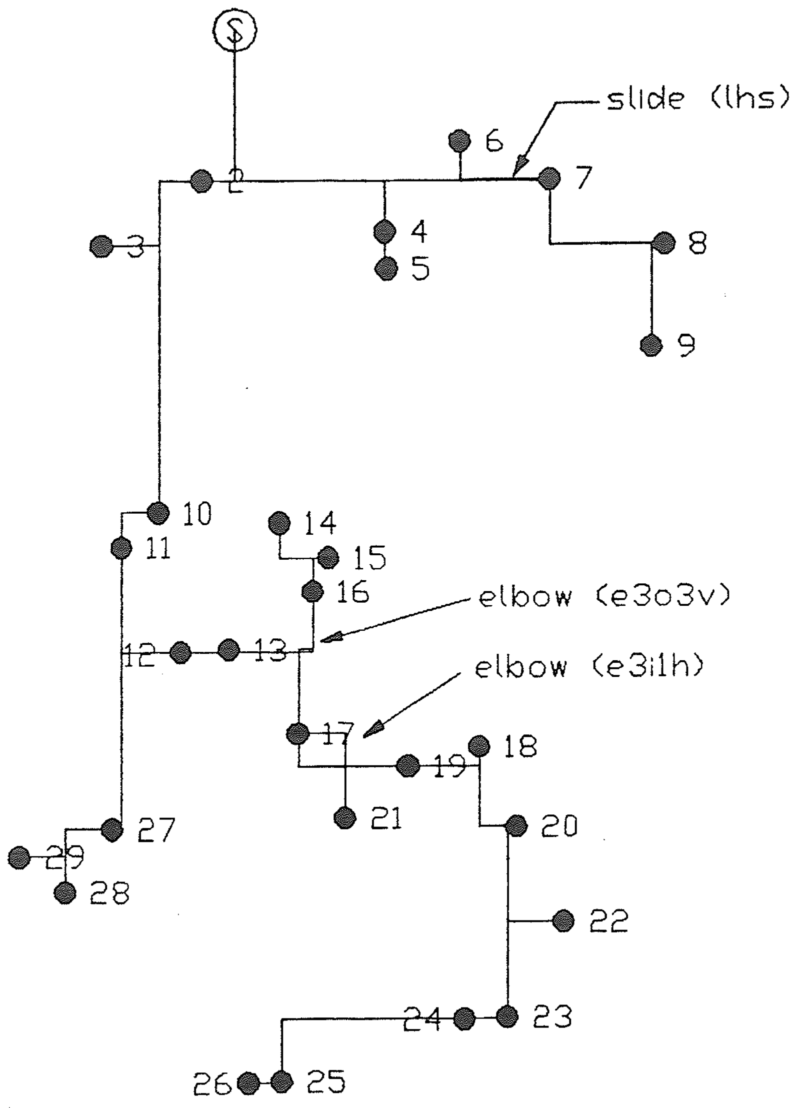


Figure 27: Modifications produced by the second iteration of the cognitive component for Example 1

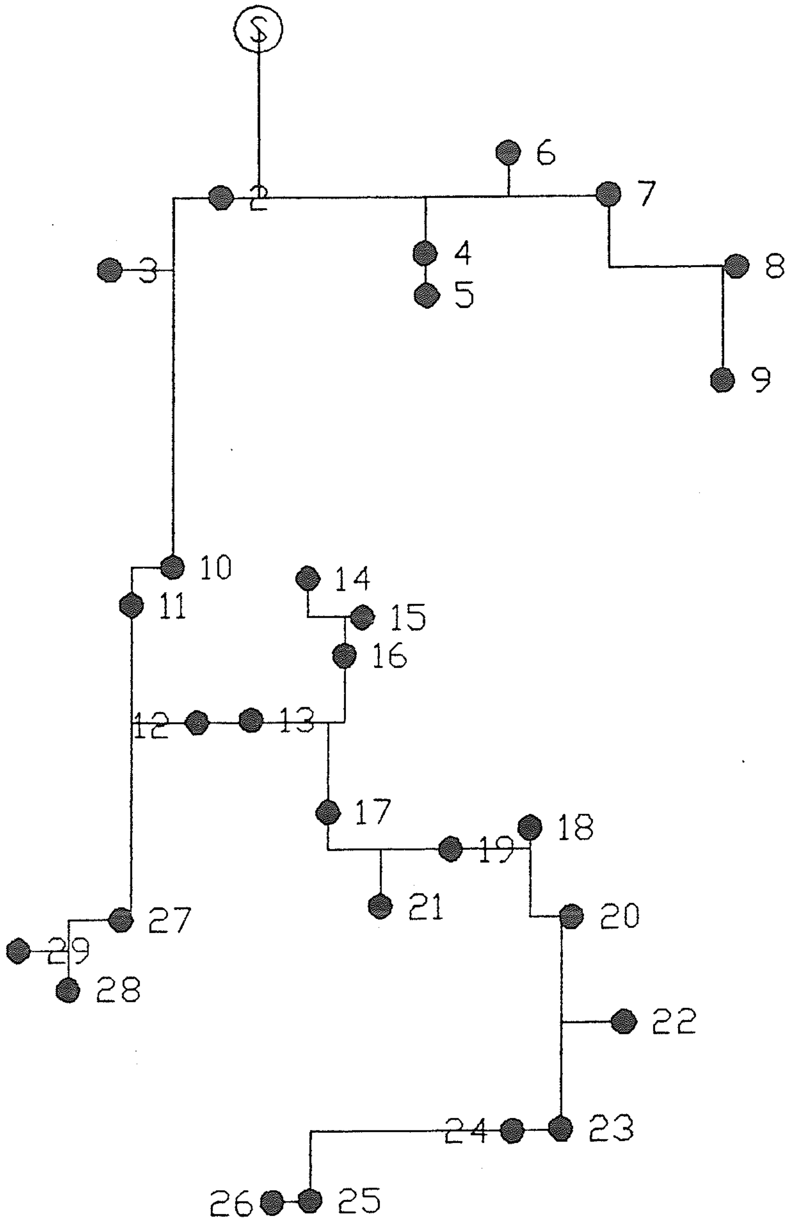


Figure 28: Layout produced by the second iteration of the cognitive component for Example 1

The results from this first example are summarized below.

TABLE 7: SUMMARY OF RESULTS FROM EXAMPLE 1

Method	Length (meters)	Cost (dollars)
Minimal Spanning Tree (not rectilinear)	54728	149536
Procedural Component	62303	164564
Cognitive Component		
First iteration	61578	162220
Second iteration	61578	162215

If the cognitive component is executed for another iteration, two redundant nodes are removed but no rules trigger and no subsequent modifications to the layout occur. In some cases it is possible that the procedural component can generate further improvements in a layout if the procedural component is executed using the layout modified by the cognitive component as input. For this example a further iteration of the procedural component did not result in any changes to the layout. The two components of the program have converged on a layout that neither component can improve. The next example will show that a series of iterations does not always end in this type of convergence.

Example 2

The second example is presented in Figures 29 through 34. The same nodes as the previous example are used. However, the survey grid is assumed to be at an angle three degrees counter-clockwise to the UTM coordinate system. Therefore, the problem requires a skewed rectilinear solution similar to those discussed in Chapter 3. In addition, the solution to this problem is considered with respect to a river that flows through the project site. The problem is exactly the same as the previous example in every other aspect, yet, as will be shown later, the small angle of rotation produces surprisingly different results.

The network in Figure 29 is the result of the first iteration of the procedural component of the program. The nodes are first submitted to a program that rotates their positions three degrees clockwise. The procedural component is used to create a rectilinear layout by adding dummy nodes to these rotated nodes. All the nodes, including the newly added dummy nodes are rotated three degrees counter-clockwise back to the original position. The layout in Figure 29 is created using the MST algorithm. Hydraulic design and cost analysis are performed on this layout in the same manner as the previous example with the maximum pressure gradient set at 18 kPa/km.

Next, the network is modified using the cognitive component and

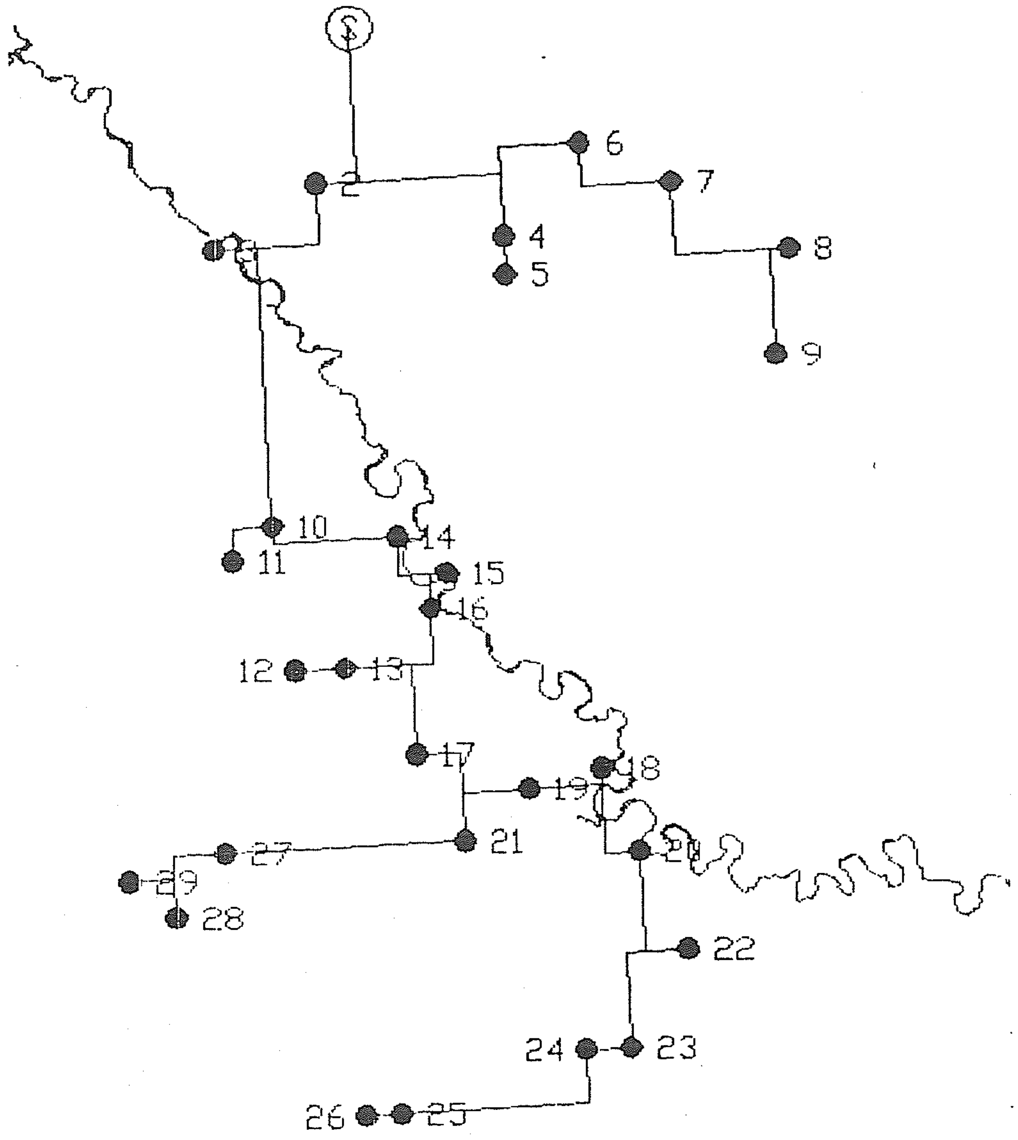


Figure 29: First iteration of the procedural component for Example 2

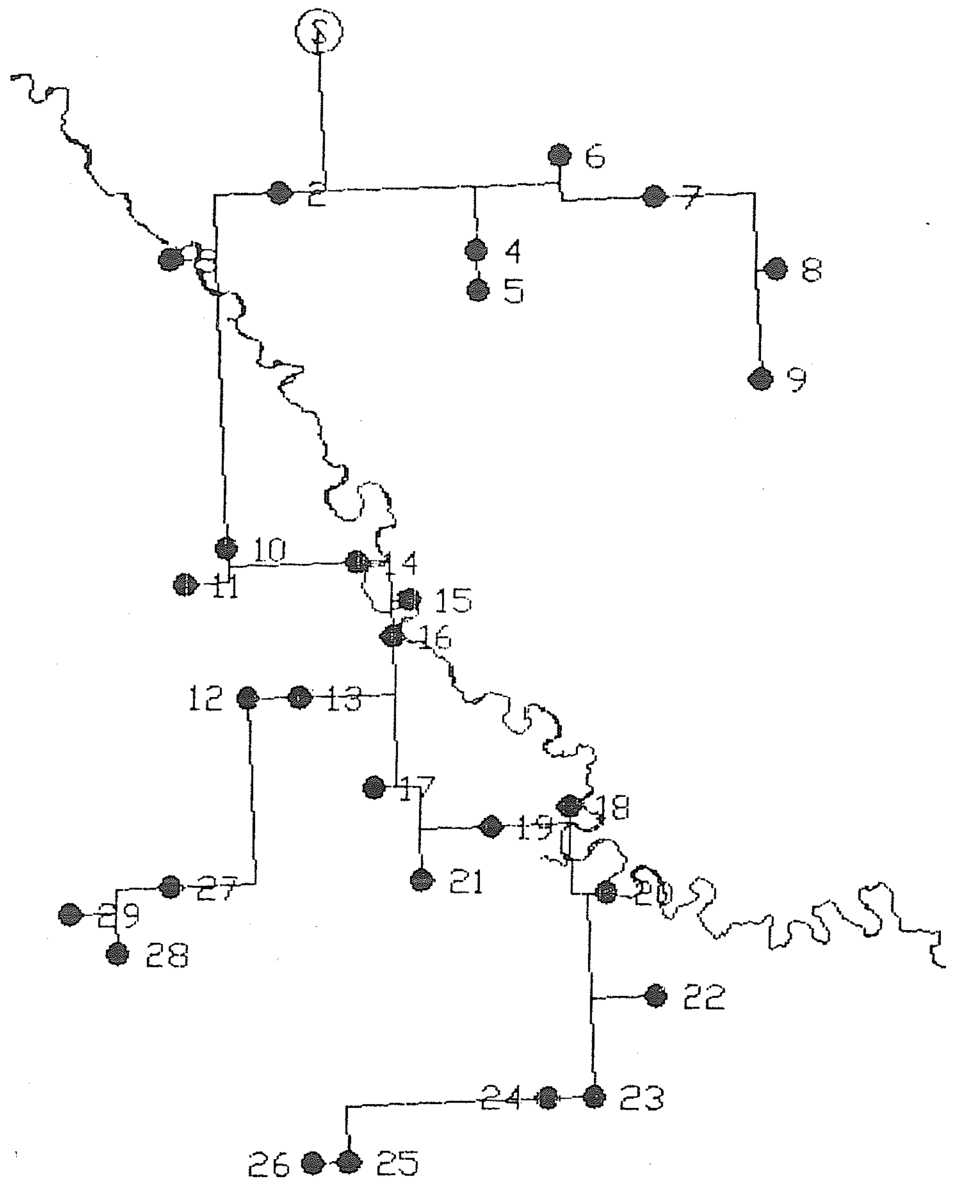


Figure 30: First iteration of the cognitive component for Example 2

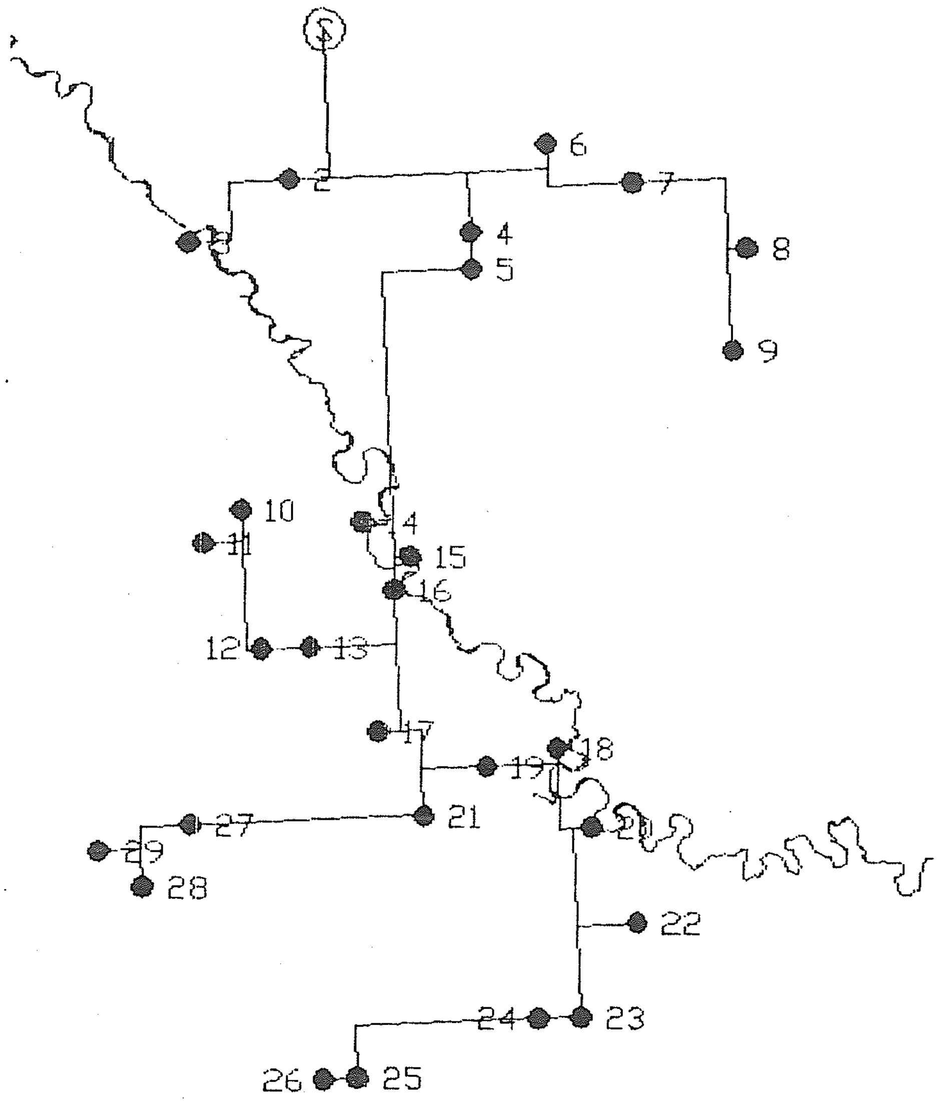


Figure 31: Second iteration of the procedural component for Example 2

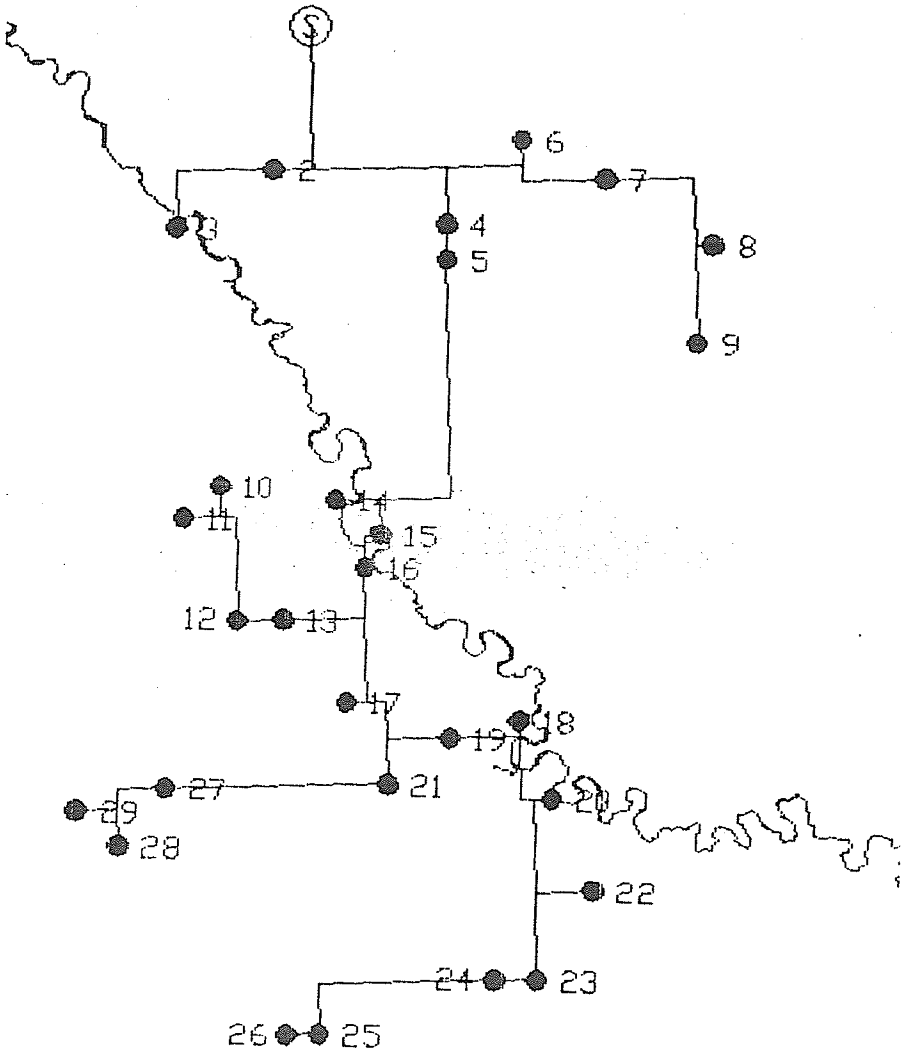


Figure 32: Second iteration of the cognitive component for Example 2

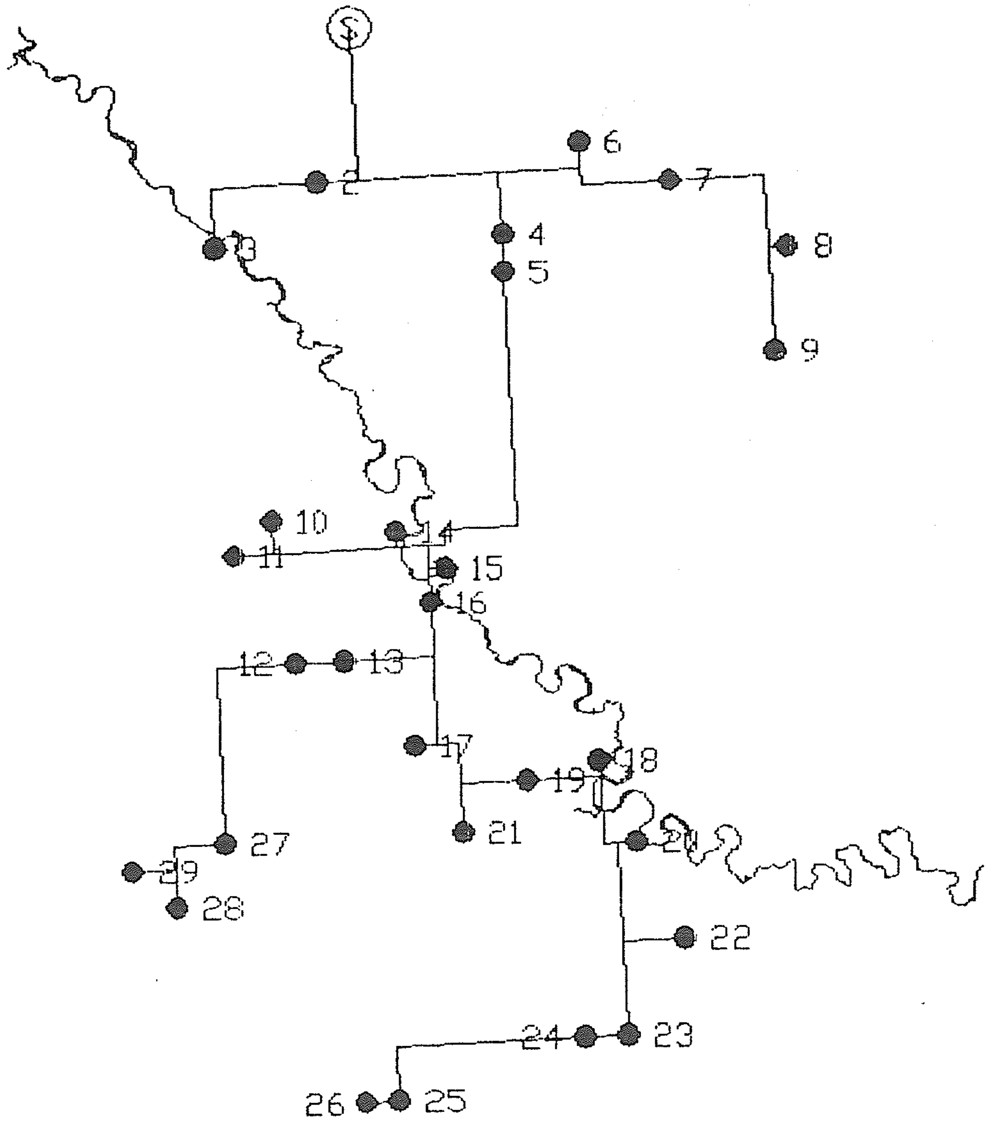


Figure 33: Third iteration of the procedural component for Example 2

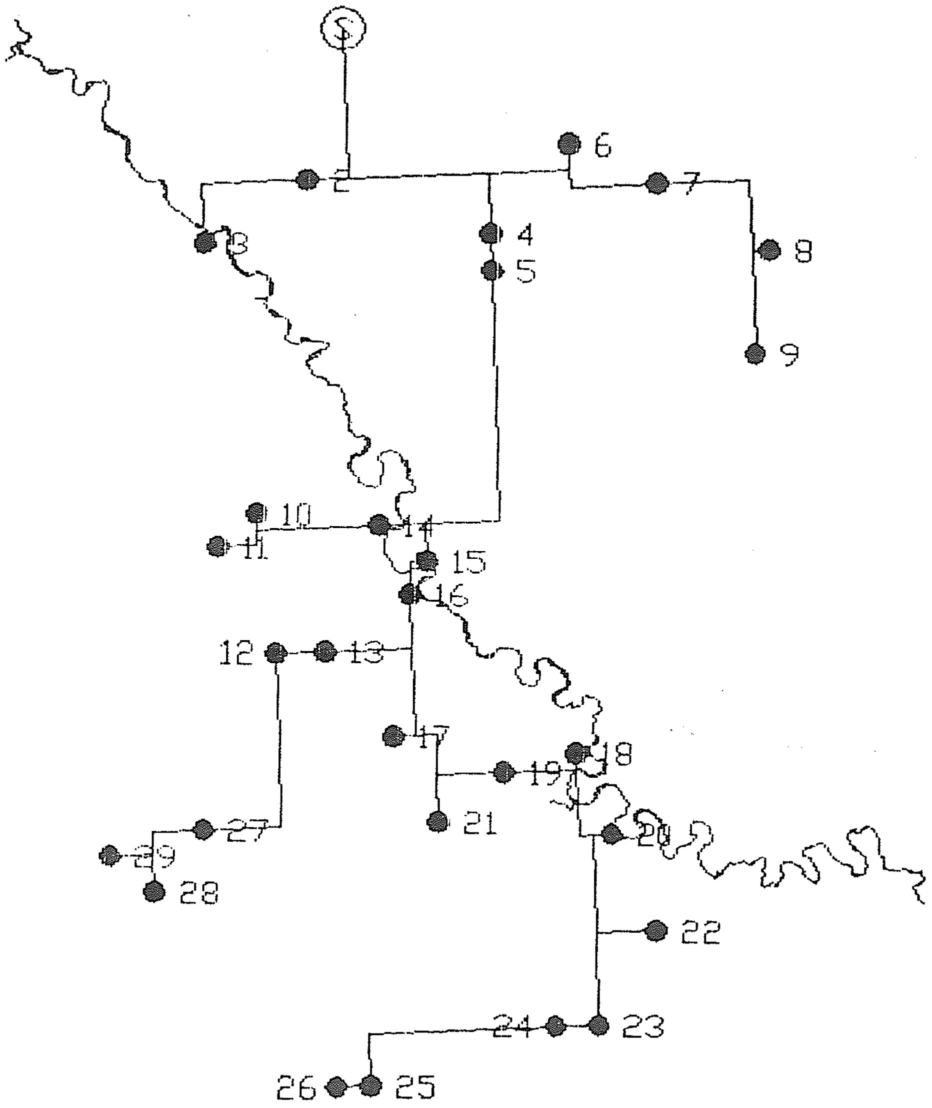


Figure 34: Third iteration of the cognitive component for Example 2

the modified network is shown in Figure 30. Hydraulic design and cost analysis were performed on this network. It is difficult in this instance to submit the output data from the cognitive component of the program to the program that is used to perform the hydraulic design and cost analysis. Some modification of the node data is required to evaluate the performance of the layout. The modifications are necessary because of a program design decision that was made during the development of the hydraulic design program.

The hydraulic design program is an older program that was developed before work began on the automated selection of dummy nodes. At the time of the older development all node data were entered manually and links were generated using the MST algorithm. The modifications that the cognitive component makes and redundant nodes that are removed at that stage can result in an efficient rectilinear layout that is not necessarily a Minimal Spanning Tree. In other words, there may exist diagonal links which can produce a more efficient layout. Since the hydraulic design program, as presently written, can only accept node data as input, and because the hydraulic design program always begins by connecting the nodes using the MST algorithm, the hydraulic design program often generates a layout that is different than the layout originally created by the cognitive component. A layout of this type, generated in error, will contain diagonal lines.

The problem does not occur with the procedural component because the output generated by the procedural component is always a Minimal Spanning Tree. The problem only occurs when a layout generated by the cognitive component is used as input to the hydraulic design program, and even then only in some cases - for instance, the problem did not occur in the previous example.

The proper solution to the problem would be to rewrite the hydraulic design program. The new program would not use the MST algorithm, or any other algorithm, to redesign the layout, i.e., connect the nodes. The new program would be capable of accepting as input, data describing connections between nodes that have been established previously using any network algorithm, not just the MST algorithm. Until this new program is created an alternative technique will be used in conjunction with the existing program. Dummy nodes can be added to force the MST algorithm in the hydraulic design program to generate the same layout as the layout produced by the cognitive component. This is the method that is used in this example. The nodes that were added during the first iteration of the cognitive component for Example 2 are listed below:

FIRST ITERATION

NODE NUMBER	X	Y
58	1028417.0	5399793.0
59	1029614.0	5396906.0
60	1028700.0	5393042.0

The cognitive component receives input from a file generated by the procedural component. This file is created before the nodes have been rotated back to their original position. It is essential that the nodes are in the same position as they were during the execution of the procedural component for the cognitive component to function properly. Once the additional nodes that must be added to generate the layout using the MST algorithm have been entered, the set of nodes can be rotated back to the original position to produce the final drawing shown in Figure 30. If a further iteration of the procedural component is to be attempted it is best not to rotate the nodes back to their original positions at this time.

In the next step, the output node data produced by the first iteration of the cognitive component are used as input for a second iteration of the procedural component. The network produced by the second iteration of the procedural component is shown in Figure 31. The process of iterating between the cognitive component and the procedural component is repeated to produce the networks shown in Figures 31 to 34. Each time the results from the cognitive component were to be evaluated additional dummy nodes were required. The coordinates of these dummy nodes are listed below. In the case of this example convergence was not reached after many iterations and, in fact, it is not even known whether the problem will converge.

SECOND ITERATION

NODE NUMBER	X	Y
58	1028700.0	5395713.0
59	1030824.0	5391291.0
60	1028693.0	5391291.0

THIRD ITERATION

NODE NUMBER	X	Y
58	1029442.0	5396906.0
59	1028700.0	5392901.0

It can be seen that a new set of dummy nodes is added at each iteration. For example, node 58 has different coordinates for the second and third iterations. This situation occurs because the additional dummy nodes are only required for rotating or for performing hydraulic designs on the results of the cognitive component. The results of the cognitive component that are used as input for a further iteration of the procedural component do not have any additional nodes added.

The results of an analysis of cost are summarized in Table 8. As mentioned previously it is not possible at this point for the cost analysis program to determine the number of river crossings that occur if the dummy nodes are generated internally by the computer. Given the small differences in cost between the six systems presented here, the relationship between the river and the different systems may be the deciding factor in selecting the final layout.

TABLE 8: SUMMARY OF EXAMPLE 2 RESULTS

METHOD	LENGTH	COST
PROCEDURAL COMPONENT		
first iteration	62452	166332
second iteration	63011	164926
third iteration	63574	165841
COGNITIVE COMPONENT		
first iteration	61996	162185
second iteration	63010	164924
third iteration	63286	165492

The results produced by the program are consistent with the assumptions made in Chapter 2 regarding the nature of complex problems. It was stated that many near-optimal solutions can be generated which are close enough to each other in their evaluations that the solutions should be regarded as equivalent given the precision of the model. It was also stated that criteria for the evaluation of solutions often exist which cannot be modelled easily. In this instance the criterion that could not be modelled was the number of river crossings. However, many other criteria of this type are certain to exist in complex real problems.

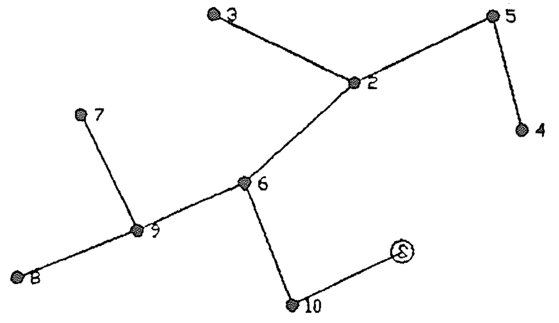
Example 1 and Example 2 differ from each other only in the angle of rotation. It is important to recognize that it is the change in the configuration or position of nodes that is brought about by rotation and not the process of rotation that is responsible for the phenomenon of non-convergence. The specific causes of non-convergence are examined in more detail in the next section.

Problems and Benefits of the Iterative Approach

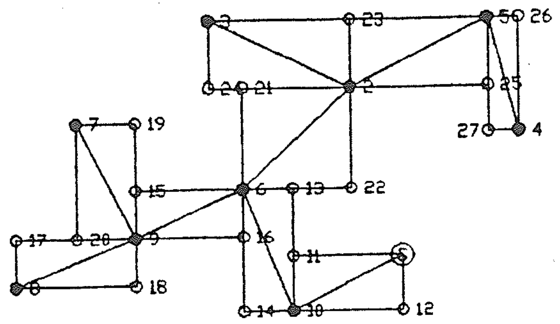
Iteration forms the basis of the organizational structure of the program. The iterative processes result in unpredictable behaviour both in terms of the execution time of the program and the quality of the results the program produces. Iteration occurs on many different levels in the program both within and between the two components of the program. Essentially the same processes are involved at any level. In some way, at any of these levels, the MST algorithm is used to determine a layout based on the proximity relationships between the nodes. The term "proximity relationships" is used to denote the spatial "closeness", or clustering of the nodes.

The cognitive component, in contrast to the procedural component, recognizes and manipulates patterns formed by connections that have been established between nodes previously. The cognitive component cannot recognize if these manipulations of patterns have changed the proximity relationships between the nodes within the pattern, or whether the manipulations have changed proximity relationships between the pattern and the surrounding nodes, to permit a more efficient layout to be produced. In other words, the cognitive component cannot perceive and take advantage of new clusters of nodes that may have been created by the firing of rules. The problem is not necessarily due to the rule based approach. It may be possible to formulate rules to perform this task. In the present form, the cognitive component relies on a further iteration of the procedural component to accomplish the task of reestablishing proximity relationships, hence the iteration between the two components. The difficulty with this approach is that the procedural component can react to the new placement of nodes in the wrong manner, generating a layout which is less efficient than the layout from the previous iteration, as occurs in Example 2.

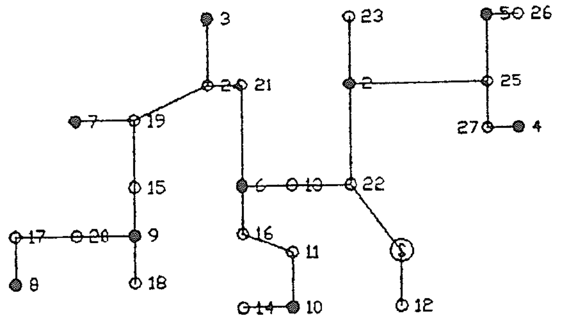
The series of graphs shown in Figure 35 illustrate how the generation of a less efficient layout can occur. The procedure that was used to generate the graphs in Figure 35 is not a procedure that is used in the program in its present form. The procedure is an early prototype of the procedural component of the program which provided some insight into problems with the iterative approach. The graphs are presented to illustrate these problems and explain the solution strategies that were adopted. The procedure consists of iterations of the MST algorithm in combination with a routine that produces supergraphs containing rectangles where diagonals



a) First iteration of MST

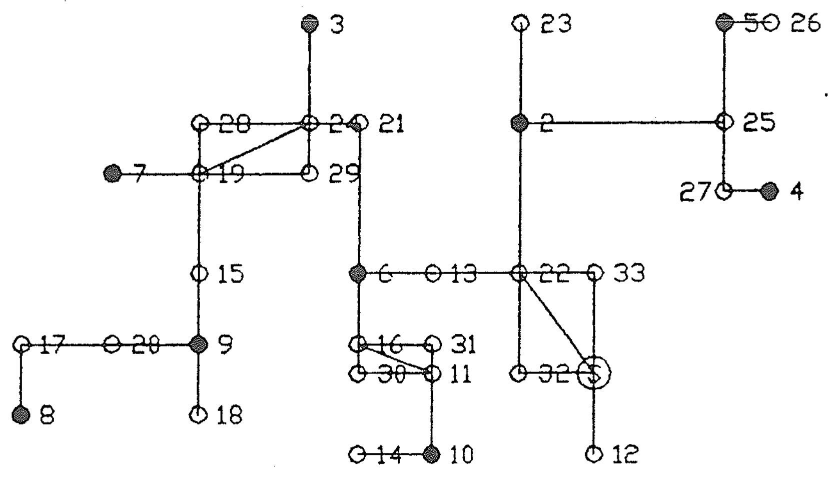


b) Dummy nodes are inserted at corners of rectangles

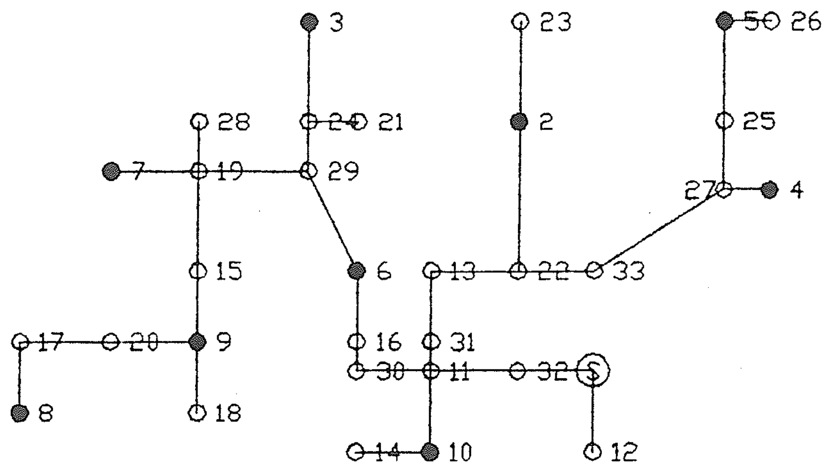


c) Second iteration of MST

Figure 35: Problems with the iterative approach for an example network

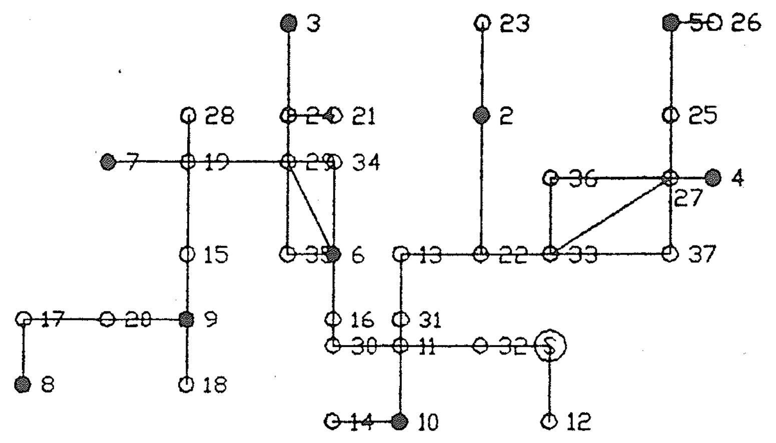


d) Dummy nodes are added
to remove diagonals

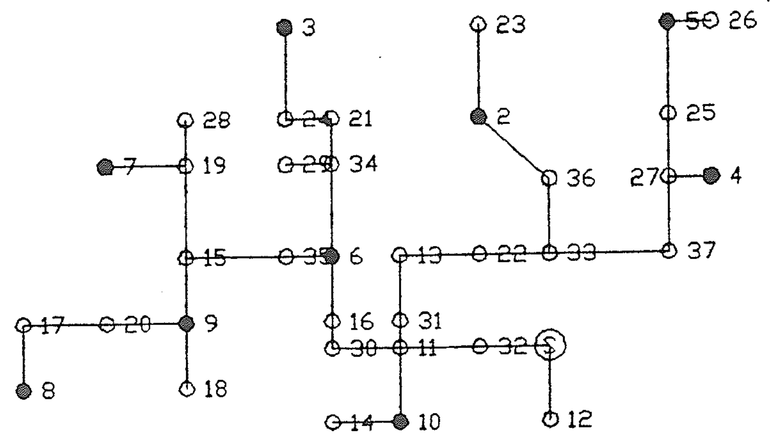


e) Third iteration of MST

Figure 35: Problems with the iterative approach
for an example network

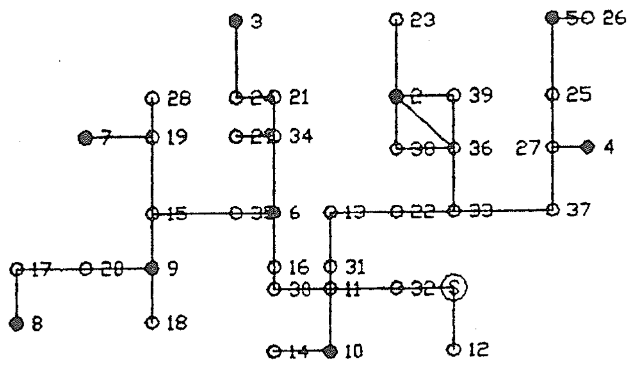


f) Dummy nodes added

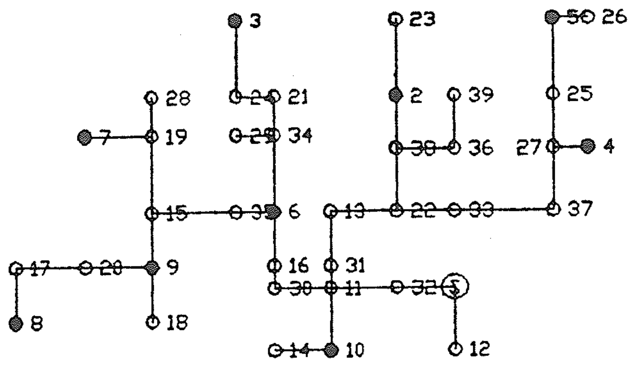


g) Fourth iteration of MST

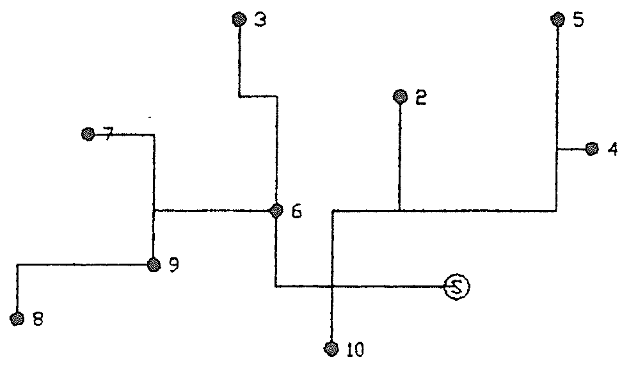
Figure 35: Problems with the iterative approach for an example network



h) Dummy nodes added



i) Fifth iteration of MST



j) Terminal dummy nodes removed

Figure 35: Problems with the iterative approach for an example network

were produced by the MST algorithm.

The MST algorithm is used to establish the initial layout shown in Figure 35a. In Figure 35b dummy nodes are added at the corners of rectangles to make the layout rectilinear as explained in previous examples. Occasionally a dummy node will be placed adjacent to another node and a diagonal line connecting the two nodes will be selected by the MST algorithm. The diagonal line is chosen over an competing rectilinear line because the diagonal line is shorter. Often diagonal lines of this type result in improved solutions when the diagonals are replaced with rectilinear lines in a subsequent iteration. However, in some cases the pair of rectilinear lines that replaces a diagonal on the subsequent iteration may be longer than the original rectilinear line that was replaced during the previous iteration. The design becomes less efficient rather than more efficient.

In Figure 35d a rectilinear line connects node 2 to node 25. In the Figure 35e this rectilinear line has been replaced by a shorter diagonal line from node 33 to node 27. In Figure 35g the diagonal has been replaced by a pair of lines one from node 33 to node 37 and one from node 37 to node 27, which are longer than the original line from node 2 to node 25.

This explains the fact that the subsequent iterations of the procedural component in Example 2 produced layouts with greater total length and higher costs than previous iterations of the cognitive component. A clear example of this can be seen by comparing Figure 30 with Figure 31. A straight line connects the tee adjacent to node 3 with node 10 in Figure 30. In Figure 31 this line has been replaced by an elbow that connects node 5 with the tee adjacent to node 14. This elbow has a greater total length than the line it replaces however the diagonal distance is shorter. It is this shorter diagonal distance that is responsible for the error.

One possible solution to this problem would be to search for all the diagonal lines in the supergraph and replace the actual lengths of each diagonal line with artificial lengths equal to the rectilinear distance of the diagonal line. However, this technique would not produce a final layout that can be generated by the MST algorithm from a complete graph of the nodes. Furthermore, it would not be possible to rotate the nodes to produce skewed networks.

It is preferable to leave the error in the algorithm since small errors of this type are useful. The error permits the algorithm to bypass local optima allowing the algorithm to generate many alternative layouts rather than simply converging to one layout that can no longer be improved. Non-convergence requires that not all the modifications made on the system during an iteration be improvements. Some modifications may slightly reduce the level of performance of the system, but these modifications are performed in combination with modifications that improve the system's performance. In this way an iteration may produce a system that is geometrically different from the previous system, yet more or less equivalent in terms of cost and other aspects of performance.

Example 2 illustrates this point. In the case of Example 2 it was clearly beneficial to have several layouts of equivalent performance and cost to choose from when considering the problems imposed by river crossings. The property of non-convergence was discovered by accident, as the result of a programming error. However, the error points to a direction for further program development. It is desirable to have many "errors" of this type to delay convergence in the case of problems that converge too rapidly, thereby generating more alternative layouts to choose from. This type of approach will require a new high level control strategy which is larger and more sophisticated, to guide the process of generating and evaluating alternatives.

Figure 35 illustrates another problem caused by successive iterations of this early prototype procedure. As more dummy nodes are added to make diagonal lines rectilinear, more diagonal "shortcuts" become possible. The figure shows that the process of adding nodes and rerunning the MST algorithm had to be repeated four times before an entirely rectilinear solution could be generated. By this time the number of input dummy nodes is so large that the MST algorithm cannot generate an efficient layout. The final layout, shown in Figure 35j, has inefficiencies similar to those shown in Figure 12b.

The problem of increasing numbers of dummy nodes and the consequent inefficient layouts does not occur to the same extent in the current version of the program. Applying the cognitive component with each successive iteration tends to "clean up" the system. As mentioned in Chapter 4, redundant nodes are removed, thus reducing the number of dummy nodes. Additionally, "hooks", "slides", and "elbows" are rules that identify many of the inefficiencies created through iteration. One way to regard the cognitive component is that it is used to ensure that a certain level of performance is maintained. In a similar way, an effective cognitive component appears to be the key to the success of any program that is developed to produce a large number of alternative layouts using the proposed method of non-convergence. In the proposed non-convergence method, the procedural component would generate sub-optimal "concepts" for layouts while the local optimal would be avoided. The fine tuning of the "concepts" would be performed by the cognitive component bringing the solutions to, or near, local optima.

CHAPTER 6: SUMMARY

A microcomputer based model for the design of rural natural gas distribution systems has been developed. The model,which is based upon mathematical network optimization algorithms, has been designed to accommodate as many "real world" concerns as could be identified while recognizing at the same time that many aspects of the problem are extremely difficult or impossible to model effectively. The model relies on user judgment to accommodate the aspects of the problem which cannot be modelled. As in previous work the use of an interactive, graphics-based user interface is considered to be essential to best facilitate user judgment.

The model combines programs designed for user interaction with non-interactive programs that generate layouts, perform hydraulic designs and analyze costs. Rectilinear layouts are generated by a procedure that combines two common algorithms, the Minimal Spanning Tree and Dijkstra's Algorithm, with two heuristic techniques to assist in the selection of dummy nodes. As development of the procedure progressed and its performance improved it became increasingly difficult to identify errors in the layouts created by the procedure and substantially more difficult to devise routines to ensure that these errors would not be repeated in an improved version of the program.

As a result another component of the program, a cognitive component, was created to diagnose and correct errors created by the procedural component. The majority of errors created by the procedure were found to belong to one of three categories, tentative referred to as "hooks", "slides", and "elbows". A rule base and control strategy were developed to identify and correct all possible occurrences of these three problems.

The program is presently composed of these two distinct components. The procedural component is written in C, which is a procedural language and the cognitive component is written in PROLOG, which is a declarative language. In early trials, the two components would be used in successive iterations to converge on a single layout that neither component could improve.

Later, it was discovered that a small error in programming logic existed in the procedural component, which, in certain cases, caused the program to produce many alternative layouts without converging. The ability to generate many alternatives is considered to be highly advantageous in view of the fact that certain aspects of the problem cannot be modelled and ultimately some user interaction will be required. In the previous work, user interaction took the form of repeated and extensive modification of a single layout based on the Minimal Spanning Tree. Using a program of the type based on the concept of non-convergence, the user would select a layout that is the best of the alternatives that the program generates. Some modification to this layout may be required but this modification would likely be very minor in contrast to the extensive manual work required by previous methods.

A means to ensure non-convergence has not been developed. In addition, some larger and more sophisticated control strategy will be required to manage the interaction of the two components of the program during the generation of alternatives. Future development of the network generating algorithms developed in this study should proceed in these two directions.

REFERENCES

Bondy, J. A. and Murty, U. R. S. (1976). Graph Theory with Applications. MacMillan Press, New York, 264 pages.

Davidson, J. W. (1989). A Fuzzy Decision Model for the Design of Rural Natural Gas Distribution Networks. Unpublished work.

Davidson, J. W. (1988). A Total Workstation for the Design of Rural Natural Gas Distribution Systems. Unpublished B.Sc. thesis, University of Manitoba, Winnipeg, MB.

Davidson, J. W. and Goulter, I. C. (1989). "Microcomputer Workstation for Design of Rural Natural Gas Distribution Systems". Microcomputers in Civil Engineering, 4(1).

Dubois, D. (1983). "A Fuzzy, Heuristic, Interactive Approach to the Optimal Network Problem". Advances in Fuzzy Sets, Possibility Theory and Applications, Paul p. Wang ed., Plenum Press, New York, pp 253-276.

Garey, M. R. and Johnson, D. S. (1977). "The Rectilinear Steiner Tree Problem is NP-Complete". SIAM J. Appl. Math, 32(4),:826-834.

Jabri, M. A. and Skellern, D. J. (1988). "PIAF: A KBS/algorithmic IC Floorplanner". Artificial Intelligence in Engineering Design, J. S. Gero ed. Elsevier, Amsterdam, pp 163-190.

Uhl, A. E., et al. (1965). Institute of Gas Technology Technical Report No. 10, Steady Flow in Gas Pipelines. American Gas Association, Inc., New York.

APPENDIX A: HYDRAULIC PROFILES FOR VARIOUS FIGURES

Hydraulic Profile for Figure 21

FROM	TO	LENGTH	NUM	LOAD	SIZE	PI	PO	PD	WO	WD
****	****	*******	***	*****	****	*****	*****	*****	****	****
1	30	2853.0	28	106.0	60.3	550.0	531.7	6.4	0	0
30	2	616.4	22	85.9	48.3	531.7	523.6	13.1	0	0
2	32	972.0	21	82.6	48.3	523.6	511.7	12.3	0	0
32	31	255.0	21	82.6	48.3	511.7	508.5	12.4	0	0
31	39	812.3	21	82.6	48.3	508.5	498.3	12.6	0	0
39	3	1071.8	1	6.0	26.7	498.3	495.9	2.2	0	0
30	33	2821.7	6	33.9	33.4	531.7	489.7	14.9	0	0
33	35	740.0	4	24.0	33.4	489.7	483.5	8.4	0	0
33	4	972.0	2	12.0	26.7	489.7	482.4	7.5	0	0
4	34	34.7	1	6.0	26.7	482.4	482.3	2.3	0	0
34	5	705.0	1	6.0	26.7	482.3	480.7	2.3	0	0
35	6	1407.3	4	24.0	33.4	483.5	471.5	8.5	0	0
6	36	725.0	3	18.0	26.7	471.5	460.7	14.9	0	0
36	7	1678.4	3	18.0	26.7	460.7	434.8	15.4	0	0
7	37	1238.0	2	12.0	26.7	434.8	424.5	8.4	0	0
37	38	1900.6	2	12.0	26.7	424.5	408.2	8.6	0	0
38	8	233.9	1	6.0	26.7	408.2	407.6	2.6	0	0
38	9	1955.0	1	6.0	26.7	408.2	403.0	2.6	0	0
39	10	5070.0	20	79.2	48.3	498.3	434.8	12.5	0	0
10	40	701.1	19	75.8	48.3	434.8	426.1	12.5	0	0
40	11	644.0	19	75.8	48.3	426.1	417.9	12.7	0	0
11	58	1995.0	18	72.4	48.3	417.9	393.9	12.0	0	0
58	53	165.8	3	18.0	26.7	393.9	391.1	17.2	0	0
58	12	1122.4	15	61.7	48.3	393.9	383.5	9.3	0	0
12	44	908.4	14	58.1	48.3	383.5	375.8	8.4	0	0
44	13	42.0	14	58.1	48.3	375.8	375.5	8.5	0	0
13	45	1297.5	13	54.4	48.3	375.5	365.5	7.7	0	0
45	43	273.8	3	18.0	33.4	365.5	363.7	6.3	0	0
43	16	1094.0	3	18.0	33.4	363.7	356.8	6.4	0	0
16	42	643.0	2	12.0	26.7	356.8	350.5	9.8	0	0
42	15	299.8	1	6.0	26.7	350.5	349.6	3.0	0	0
42	41	623.6	1	6.0	26.7	350.5	348.6	3.0	0	0
41	14	663.0	1	6.0	26.7	348.6	346.7	3.0	0	0
45	17	1575.0	10	42.9	48.3	365.5	357.3	5.2	0	0
17	46	886.2	9	41.7	48.3	357.3	352.8	5.1	0	0
46	47	640.0	9	41.7	48.3	352.8	349.5	5.1	0	0
47	21	963.0	1	6.0	26.7	349.5	346.6	3.0	0	0
47	19	1182.3	8	39.8	48.3	349.5	343.9	4.7	0	0
19	48	1327.0	7	37.2	48.3	343.9	338.3	4.3	0	0
48	18	366.0	1	6.0	26.7	338.3	337.1	3.1	0	0
48	49	1146.0	6	33.9	48.3	338.3	334.2	3.6	0	0
49	20	693.6	6	33.9	48.3	334.2	331.7	3.6	0	0
20	50	1801.0	5	30.0	33.4	331.7	299.5	17.9	0	0
50	51	162.5	4	24.0	33.4	299.5	297.5	12.4	0	0
50	22	889.1	1	6.0	26.7	299.5	296.5	3.4	0	0

```
FROM   TO    LENGTH   NUM  LOAD   SIZE   PI     PO     PD     WO   WD
****   ****  *******  ***  *****  ****   *****  *****  *****  **** ****
  51    23   1797.0    4   24.0   33.4   297.5  274.4  12.8    0    0
  23    52     32.0    3   18.0   33.4   274.4  274.2   7.8    0    0
  52    24    807.8    3   18.0   33.4   274.2  267.8   7.9    0    0
  24    56   1176.0    2   12.0   26.7   267.8  253.3  12.3    0    0
  53    27   3346.0    3   18.0   33.4   391.1  370.8   6.1    0    0
  27    54    878.2    2   12.0   26.7   370.8  362.4   9.5    0    0
  54    55    512.0    2   12.0   26.7   362.4  357.5   9.6    0    0
  55    28    669.0    1    6.0   26.7   357.5  355.5   2.9    0    0
  55    29    839.7    1    6.0   26.7   357.5  355.0   2.9    0    0
  56    25   3416.8    2   12.0   26.7   253.3  207.2  13.5    0    0
  25    57     32.0    1    6.0   26.7   207.2  207.1   4.4    0    0
  57    26    629.4    1    6.0   26.7   207.1  204.3   4.4    0    0


PIPE SIZE      TOTAL LENGTH      COST
*********      ************     *******
26.7   mm         23281.         37715.
33.4   mm         14283.         28138.
48.3   mm         21886.         63032.
60.3   mm          2853.         11612.
88.9   mm             0.             0.

TOTAL             62303.        140496.


TOTAL NUMBER OF ROAD  CROSSINGS  36   COST  20289.
TOTAL NUMBER OF RAIL  CROSSINGS   0   COST      0.
TOTAL NUMBER OF CABLE CROSSINGS  42   COST   3780.
TOTAL NUMBER OF CREEK CROSSINGS   0   COST      0.


TOTAL COST OF SYSTEM  164564.
```

Hydraulic Profile for Figure 25

FROM	TO	LENGTH	NUM	LOAD	SIZE	PI	PO	PD	WO	WD
****	****	*******	***	*****	****	*****	*****	*****	****	****
1	27	2853.0	28	106.0	60.3	550.0	531.7	6.4	0	0
27	2	616.4	22	85.9	48.3	531.7	523.6	13.1	0	0
2	56	812.3	21	82.6	48.3	523.6	513.6	12.3	0	0
56	31	1227.0	21	82.6	48.3	513.6	498.3	12.5	0	0
31	3	1071.8	1	6.0	26.7	498.3	495.9	2.2	0	0
27	28	2821.7	6	33.9	33.4	531.7	489.7	14.9	0	0
28	4	972.0	2	12.0	26.7	489.7	482.4	7.5	0	0
4	57	705.0	1	6.0	26.7	482.4	480.8	2.3	0	0
57	5	34.7	1	6.0	26.7	480.8	480.7	2.3	0	0
28	45	1407.3	4	24.0	33.4	489.7	477.9	8.4	0	0
45	47	15.0	4	24.0	33.4	477.9	477.7	8.5	0	0
47	46	725.0	1	6.0	26.7	477.7	476.1	2.3	0	0
47	6	1678.4	3	18.0	26.7	477.7	452.7	14.9	0	0
6	29	1238.0	2	12.0	26.7	452.7	442.7	8.1	0	0
29	30	1900.6	2	12.0	26.7	442.7	426.9	8.3	0	0
30	7	233.9	1	6.0	26.7	426.9	426.3	2.5	0	0
30	8	1955.0	1	6.0	26.7	426.9	422.0	2.5	0	0
31	9	5070.0	20	79.2	48.3	498.3	434.8	12.5	0	0
9	32	701.1	19	75.8	48.3	434.8	426.1	12.5	0	0
32	10	644.0	19	75.8	48.3	426.1	417.9	12.7	0	0
10	44	1995.0	18	72.4	48.3	417.9	393.9	12.0	0	0
44	11	1122.4	15	61.7	48.3	393.9	383.5	9.3	0	0
11	50	908.4	14	58.1	48.3	383.5	375.8	8.4	0	0
50	48	42.0	1	6.0	26.7	375.8	375.7	2.8	0	0
50	51	1297.5	13	54.4	48.3	375.8	365.8	7.7	0	0
51	49	42.0	3	18.0	33.4	365.8	365.6	6.3	0	0
49	35	273.8	3	18.0	33.4	365.6	363.8	6.3	0	0
35	14	1094.0	3	18.0	33.4	363.8	356.9	6.3	0	0
14	34	643.0	2	12.0	26.7	356.9	350.6	9.8	0	0
34	13	299.8	1	6.0	26.7	350.6	349.7	3.0	0	0
34	33	623.6	1	6.0	26.7	350.6	348.7	3.0	0	0
33	12	663.0	1	6.0	26.7	348.7	346.8	3.0	0	0
51	15	1533.0	10	42.9	48.3	365.8	357.9	5.2	0	0
15	36	886.2	9	41.7	48.3	357.9	353.4	5.1	0	0
36	37	640.0	9	41.7	48.3	353.4	350.1	5.1	0	0
37	18	963.0	1	6.0	26.7	350.1	347.2	3.0	0	0
37	17	1182.3	8	39.8	48.3	350.1	344.5	4.7	0	0
17	38	1327.0	7	37.2	48.3	344.5	338.9	4.3	0	0
38	16	366.0	1	6.0	26.7	338.9	337.8	3.1	0	0
38	39	1146.0	6	33.9	48.3	338.9	334.8	3.6	0	0
39	55	531.1	6	33.9	48.3	334.8	332.9	3.6	0	0
55	52	162.5	1	6.0	26.7	332.9	332.4	3.1	0	0
55	54	1801.0	5	30.0	33.4	332.9	300.8	17.8	0	0
54	53	162.5	1	6.0	26.7	300.8	300.3	3.3	0	0
53	19	889.1	1	6.0	26.7	300.3	297.3	3.4	0	0

```
FROM   TO   LENGTH   NUM  LOAD  SIZE   PI     PO     PD    WO   WD
****  ****  *******  ***  ***** ****  *****  *****  *****  **** ****
  54    20   1797.0    4  24.0  33.4  300.8  278.0  12.7    0    0
  20    40     32.0    3  18.0  33.4  278.0  277.7   7.8    0    0
  40    21    807.8    3  18.0  33.4  277.7  271.4   7.8    0    0
  44    58   3346.0    3  18.0  33.4  393.9  373.7   6.0    0    0
  58    24    165.8    3  18.0  26.7  373.7  370.8  17.9    0    0
  24    41    878.2    2  12.0  26.7  370.8  362.4   9.5    0    0
  41    42    512.0    2  12.0  26.7  362.4  357.5   9.6    0    0
  42    25    669.0    1   6.0  26.7  357.5  355.5   2.9    0    0
  42    26    839.7    1   6.0  26.7  357.5  355.0   2.9    0    0
  21    59   3416.8    2  12.0  26.7  271.4  227.9  12.7    0    0
  59    22   1176.0    2  12.0  26.7  227.9  211.5  13.9    0    0
  22    43     32.0    1   6.0  26.7  211.5  211.4   4.3    0    0
  43    23    629.4    1   6.0  26.7  211.4  208.6   4.3    0    0


PIPE SIZE     TOTAL LENGTH     COST
*********     ************    *******
26.7  mm         23648.        38309.
33.4  mm         13438.        26472.
48.3  mm         21640.        62322.
60.3  mm          2853.        11612.
88.9  mm             0.            0.

TOTAL            61578.       138715.


TOTAL NUMBER OF ROAD  CROSSINGS  35   COST  19725.
TOTAL NUMBER OF RAIL  CROSSINGS   0   COST      0.
TOTAL NUMBER OF CABLE CROSSINGS  42   COST   3780.
TOTAL NUMBER OF CREEK CROSSINGS   0   COST      0.


TOTAL COST OF SYSTEM  162220.
```

Hydraulic Profile for Figure 28

FROM	TO	LENGTH	NUM	LOAD	SIZE	PI	PO	PD	WO	WD
****	****	*******	***	*****	****	*****	*****	*****	****	****
1	26	2853.0	28	106.0	60.3	550.0	531.7	6.4	0	0
26	2	616.4	22	85.9	48.3	531.7	523.6	13.1	0	0
2	50	812.3	21	82.6	48.3	523.6	513.6	12.3	0	0
50	30	1227.0	21	82.6	48.3	513.6	498.3	12.5	0	0
30	3	1071.8	1	6.0	26.7	498.3	495.9	2.2	0	0
26	27	2821.7	6	33.9	33.4	531.7	489.7	14.9	0	0
27	4	972.0	2	12.0	26.7	489.7	482.4	7.5	0	0
4	51	705.0	1	6.0	26.7	482.4	480.8	2.3	0	0
51	5	34.7	1	6.0	26.7	480.8	480.7	2.3	0	0
27	56	1407.3	4	24.0	33.4	489.7	477.9	8.4	0	0
56	54	15.0	1	6.0	26.7	477.9	477.8	2.3	0	0
54	43	725.0	1	6.0	26.7	477.8	476.2	2.3	0	0
56	57	1678.4	3	18.0	26.7	477.9	452.8	14.9	0	0
57	55	15.0	1	6.0	26.7	452.8	452.8	2.4	0	0
57	28	1223.0	2	12.0	26.7	452.8	442.9	8.1	0	0
28	29	1900.6	2	12.0	26.7	442.9	427.2	8.3	0	0
29	6	233.9	1	6.0	26.7	427.2	426.6	2.5	0	0
29	7	1955.0	1	6.0	26.7	427.2	422.2	2.5	0	0
30	8	5070.0	20	79.2	48.3	498.3	434.8	12.5	0	0
8	31	701.1	19	75.8	48.3	434.8	426.1	12.5	0	0
31	9	644.0	19	75.8	48.3	426.1	417.9	12.7	0	0
9	42	1995.0	18	72.4	48.3	417.9	393.9	12.0	0	0
42	10	1122.4	15	61.7	48.3	393.9	383.5	9.3	0	0
10	45	908.4	14	58.1	48.3	383.5	375.8	8.4	0	0
45	44	42.0	1	6.0	26.7	375.8	375.7	2.8	0	0
45	46	1297.5	13	54.4	48.3	375.8	365.8	7.7	0	0
46	58	273.8	3	18.0	33.4	365.8	364.1	6.3	0	0
58	34	42.0	3	18.0	33.4	364.1	363.8	6.3	0	0
34	13	1094.0	3	18.0	33.4	363.8	356.9	6.3	0	0
13	33	643.0	2	12.0	26.7	356.9	350.6	9.8	0	0
33	12	299.8	1	6.0	26.7	350.6	349.7	3.0	0	0
33	32	623.6	1	6.0	26.7	350.6	348.7	3.0	0	0
32	11	663.0	1	6.0	26.7	348.7	346.8	3.0	0	0
46	14	1533.0	10	42.9	48.3	365.8	357.9	5.2	0	0
14	59	640.0	9	41.7	48.3	357.9	354.7	5.1	0	0
59	35	886.2	9	41.7	48.3	354.7	350.1	5.1	0	0
35	17	963.0	1	6.0	26.7	350.1	347.2	3.0	0	0
35	16	1182.3	8	39.8	48.3	350.1	344.5	4.7	0	0
16	36	1327.0	7	37.2	48.3	344.5	338.9	4.3	0	0
36	15	366.0	1	6.0	26.7	338.9	337.8	3.1	0	0
36	37	1146.0	6	33.9	48.3	338.9	334.8	3.6	0	0
37	49	531.1	6	33.9	48.3	334.8	332.9	3.6	0	0
49	47	162.5	1	6.0	26.7	332.9	332.4	3.1	0	0
49	48	1801.0	5	30.0	33.4	332.9	300.8	17.8	0	0
48	18	1051.6	1	6.0	26.7	300.8	297.3	3.4	0	0

```
FROM    TO   LENGTH   NUM  LOAD   SIZE   PI     PO     PD    WO   WD
****   ****  *******  ***  *****  ****  *****  *****  *****  ****  ****
  48    19   1797.0    4   24.0   33.4  300.8  278.0  12.7    0    0
  19    38     32.0    3   18.0   33.4  278.0  277.7   7.8    0    0
  38    20    807.8    3   18.0   33.4  277.7  271.4   7.8    0    0
  42    52   3346.0    3   18.0   33.4  393.9  373.7   6.0    0    0
  52    23    165.8    3   18.0   26.7  373.7  370.8  17.9    0    0
  23    39    878.2    2   12.0   26.7  370.8  362.4   9.5    0    0
  39    40    512.0    2   12.0   26.7  362.4  357.5   9.6    0    0
  40    24    669.0    1    6.0   26.7  357.5  355.5   2.9    0    0
  40    25    839.7    1    6.0   26.7  357.5  355.0   2.9    0    0
  20    53   3416.8    2   12.0   26.7  271.4  227.9  12.7    0    0
  53    21   1176.0    2   12.0   26.7  227.9  211.5  13.9    0    0
  21    41     32.0    1    6.0   26.7  211.5  211.4   4.3    0    0
  41    22    629.4    1    6.0   26.7  211.4  208.6   4.3    0    0


PIPE SIZE      TOTAL LENGTH       COST
*********      ************      *******
26.7  mm           23663.         38333.
33.4  mm           13423.         26442.
48.3  mm           21640.         62322.
60.3  mm            2853.         11612.
88.9  mm               0.             0.

TOTAL              61578.        138710.


TOTAL NUMBER OF ROAD   CROSSINGS  35    COST  19725.
TOTAL NUMBER OF RAIL   CROSSINGS   0    COST      0.
TOTAL NUMBER OF CABLE  CROSSINGS  42    COST   3780.
TOTAL NUMBER OF CREEK  CROSSINGS   0    COST      0.


TOTAL COST OF SYSTEM  162215.
```

Hydraulic Profile for Figure 29

FROM	TO	LENGTH	NUM	LOAD	SIZE	PI	PO	PD	WO	WD
****	****	*******	***	*****	****	*****	*****	*****	****	****
1	30	2816.9	28	106.0	60.3	550.0	531.9	6.4	0	0
30	2	765.0	22	85.9	48.3	531.9	521.9	13.1	0	0
2	31	1126.5	21	82.6	48.3	521.9	508.0	12.3	0	0
31	38	1141.1	21	82.6	48.3	508.0	493.6	12.6	0	0
38	3	805.0	1	6.0	26.7	493.6	491.8	2.3	0	0
30	32	2617.1	6	33.9	33.4	531.9	493.1	14.8	0	0
32	34	485.7	4	24.0	33.4	493.1	489.1	8.3	0	0
32	33	1150.5	2	12.0	26.7	493.1	484.5	7.5	0	0
33	4	1.9	2	12.0	26.7	484.5	484.5	7.6	0	0
4	5	704.9	1	6.0	26.7	484.5	482.9	2.3	0	0
34	6	1495.0	4	24.0	33.4	489.1	476.5	8.4	0	0
6	35	811.1	3	18.0	26.7	476.5	464.5	14.8	0	0
35	7	1637.9	3	18.0	26.7	464.5	439.4	15.3	0	0
7	36	1347.9	2	12.0	26.7	439.4	428.2	8.3	0	0
36	37	1730.9	2	12.0	26.7	428.2	413.5	8.5	0	0
37	8	336.0	1	6.0	26.7	413.5	412.6	2.6	0	0
37	9	1940.7	1	6.0	26.7	413.5	408.4	2.6	0	0
38	10	5119.0	20	79.2	48.3	493.6	428.9	12.6	0	0
10	40	315.4	18	72.4	48.3	428.9	425.2	11.5	0	0
10	39	734.0	1	6.0	26.7	428.9	427.0	2.5	0	0
39	11	605.8	1	6.0	26.7	427.0	425.5	2.5	0	0
40	14	2264.0	18	72.4	48.3	425.2	398.3	11.9	0	0
14	41	710.0	17	68.9	48.3	398.3	390.4	11.1	0	0
41	42	554.0	17	68.9	48.3	390.4	384.2	11.3	0	0
42	15	333.0	1	6.0	26.7	384.2	383.3	2.8	0	0
42	16	626.9	16	65.3	48.3	384.2	377.6	10.4	0	0
16	43	1010.4	15	61.7	48.3	377.6	367.9	9.7	0	0
43	45	413.0	15	61.7	48.3	367.9	363.8	9.8	0	0
45	13	1212.9	2	12.0	26.7	363.8	352.1	9.7	0	0
13	44	909.0	1	6.0	26.7	352.1	349.4	3.0	0	0
44	12	6.0	1	6.0	26.7	349.4	349.4	3.0	0	0
45	17	1641.3	13	54.4	48.3	363.8	350.8	7.9	0	0
17	46	801.0	12	50.6	48.3	350.8	345.1	7.2	0	0
46	47	747.0	12	50.6	48.3	345.1	339.6	7.3	0	0
47	60	880.2	4	24.0	33.4	339.6	329.6	11.4	0	0
60	21	20.0	1	6.0	26.7	329.6	329.6	3.1	0	0
60	59	801.0	3	18.0	33.4	329.6	324.1	6.9	0	0
47	19	1231.0	8	39.8	48.3	339.6	333.7	4.8	0	0
19	48	1345.1	7	37.2	48.3	333.7	327.8	4.4	0	0
48	18	295.4	1	6.0	26.7	327.8	326.9	3.1	0	0
48	49	1250.7	6	33.9	48.3	327.8	323.2	3.7	0	0
49	20	613.0	6	33.9	48.3	323.2	320.9	3.7	0	0
20	50	1844.5	5	30.0	48.3	320.9	315.3	3.0	0	0
50	51	350.0	4	24.0	33.4	315.3	311.1	12.0	0	0
50	22	794.0	1	6.0	26.7	315.3	312.7	3.2	0	0

```
FROM   TO    LENGTH    NUM   LOAD    SIZE    PI     PO     PD     WO    WD
****  ****  *******    ***  *****    ****  *****  *****  *****   ****  ****
  51    52   1729.4     4    24.0    33.4  311.1  289.8   12.3     0     0
  52    23     11.0     4    24.0    33.4  289.8  289.6   12.7     0     0
  23    53    808.8     3    18.0    33.4  289.6  283.5    7.6     0     0
  53    24     11.0     1     6.0    26.7  283.5  283.5    3.5     0     0
  53    57    985.3     2    12.0    26.7  283.5  271.9   11.8     0     0
  59    54   2122.0     3    18.0    33.4  324.1  309.2    7.0     0     0
  54    27   1462.0     3    18.0    33.4  309.2  298.6    7.2     0     0
  27    55    939.0     2    12.0    26.7  298.6  288.0   11.3     0     0
  55    56    421.6     2    12.0    26.7  288.0  283.2   11.5     0     0
  56    28    712.0     1     6.0    26.7  283.2  280.7    3.5     0     0
  56    29    803.0     1     6.0    26.7  283.2  280.3    3.5     0     0
  57    25   3473.0     2    12.0    26.7  271.9  227.7   12.7     0     0
  25    58      1.0     1     6.0    26.7  227.7  227.7    4.1     0     0
  58    26    631.0     1     6.0    26.7  227.7  225.1    4.1     0     0


PIPE SIZE      TOTAL LENGTH      COST
*********      ************     *******
26.7   mm         23354.         37833.
33.4   mm         12762.         25141.
48.3   mm         23519.         67734.
60.3   mm          2817.         11465.
88.9   mm             0.             0.

TOTAL             62452.        142173.


TOTAL NUMBER OF ROAD   CROSSINGS  36    COST  20289.
TOTAL NUMBER OF RAIL   CROSSINGS   0    COST      0.
TOTAL NUMBER OF CABLE  CROSSINGS  43    COST   3870.
TOTAL NUMBER OF CREEK  CROSSINGS   0    COST      0.


TOTAL COST OF SYSTEM  166332.
```

Hydraulic Profile for Figure 30

FROM	TO	LENGTH	NUM	LOAD	SIZE	PI	PO	PD	WO	WD
****	****	*******	***	*****	****	*****	*****	*****	****	****
1	23	2816.9	28	106.0	60.3	550.0	531.9	6.4	0	0
23	2	765.0	22	85.9	48.3	531.9	521.9	13.1	0	0
2	51	1141.0	21	82.6	48.3	521.9	507.8	12.3	0	0
51	26	1127.5	21	82.6	48.3	507.8	493.6	12.6	0	0
26	3	805.0	1	6.0	26.7	493.6	491.7	2.3	0	0
26	58	2547.5	20	79.2	48.3	493.6	462.3	12.3	0	0
58	38	2571.5	20	79.2	48.3	462.3	428.8	13.0	0	0
38	55	315.4	19	75.8	48.3	428.8	424.9	12.5	0	0
55	53	290.4	1	6.0	26.7	424.9	424.1	2.5	0	0
53	8	734.0	1	6.0	26.7	424.1	422.3	2.6	0	0
55	59	1196.9	18	72.4	48.3	424.9	410.8	11.7	0	0
59	54	1067.1	18	72.4	48.3	410.8	398.0	12.0	0	0
54	57	553.9	17	68.9	48.3	398.0	391.8	11.1	0	0
57	27	710.0	17	68.9	48.3	391.8	383.8	11.3	0	0
27	11	333.0	1	6.0	26.7	383.8	382.9	2.8	0	0
27	12	626.9	16	65.3	48.3	383.8	377.3	10.4	0	0
12	46	1010.4	15	61.7	48.3	377.3	367.5	9.7	0	0
46	10	1625.9	5	30.0	33.4	367.5	341.0	16.3	0	0
10	28	909.0	4	24.0	33.4	341.0	330.7	11.3	0	0
28	9	6.0	1	6.0	26.7	330.7	330.7	3.1	0	0
28	60	1517.1	3	18.0	33.4	330.7	320.2	6.9	0	0
46	47	1641.2	10	42.9	48.3	367.5	359.1	5.1	0	0
47	29	388.1	9	41.7	48.3	359.1	357.1	5.1	0	0
47	45	412.9	1	6.0	26.7	359.1	357.9	2.9	0	0
29	30	747.0	9	41.7	48.3	357.1	353.3	5.1	0	0
30	15	900.2	1	6.0	26.7	353.3	350.6	3.0	0	0
30	14	1231.0	8	39.8	48.3	353.3	347.5	4.7	0	0
14	31	1345.1	7	37.2	48.3	347.5	341.9	4.2	0	0
31	13	295.4	1	6.0	26.7	341.9	341.0	3.0	0	0
31	32	1250.7	6	33.9	48.3	341.9	337.4	3.6	0	0
32	50	263.0	6	33.9	48.3	337.4	336.5	3.6	0	0
50	48	350.0	1	6.0	26.7	336.5	335.4	3.1	0	0
60	56	1751.4	3	18.0	33.4	320.2	307.9	7.1	0	0
56	20	1462.0	3	18.0	33.4	307.9	297.2	7.3	0	0
20	34	939.0	2	12.0	26.7	297.2	286.6	11.3	0	0
34	35	421.6	2	12.0	26.7	286.6	281.7	11.6	0	0
35	21	712.0	1	6.0	26.7	281.7	279.2	3.5	0	0
35	22	803.0	1	6.0	26.7	281.7	278.9	3.5	0	0
50	49	1844.5	5	30.0	33.4	336.5	303.9	17.7	0	0
49	16	1144.0	1	6.0	26.7	303.9	300.1	3.3	0	0
49	33	1729.4	4	24.0	33.4	303.9	282.1	12.6	0	0
33	17	11.0	1	6.0	26.7	282.1	282.1	3.5	0	0
33	18	809.1	3	18.0	33.4	282.1	275.9	7.7	0	0
23	44	2615.1	6	33.9	33.4	531.9	493.2	14.8	0	0
44	43	1150.6	2	12.0	26.7	493.2	484.5	7.5	0	0

```
FROM    TO    LENGTH    NUM   LOAD   SIZE    PI     PO     PD    WO    WD
****   ****  *******    ***  *****   ****  *****  *****  *****  ****  ****
  43    42       1.9     2    12.0   26.7  484.5  484.5    7.6    0     0
  42     4     704.9     1     6.0   26.7  484.5  482.9    2.3    0     0
  44    36    1496.9     4    24.0   33.4  493.2  480.6    8.4    0     0
  36    24     325.5     3    18.0   26.7  480.6  475.9   14.6    0     0
  36    37     485.7     1     6.0   26.7  480.6  479.5    2.3    0     0
  24     5    1637.9     3    18.0   26.7  475.9  451.3   15.0    0     0
   5    52    1731.0     2    12.0   26.7  451.3  437.2    8.1    0     0
  52    25    1347.8     2    12.0   26.7  437.2  426.0    8.3    0     0
  25     6     336.0     1     6.0   26.7  426.0  425.2    2.5    0     0
  25     7    1940.7     1     6.0   26.7  426.0  421.1    2.6    0     0
  18    39    3473.0     2    12.0   26.7  275.9  232.2   12.6    0     0
  39    41     995.4     2    12.0   26.7  232.2  218.6   13.7    0     0
  41    40       1.0     1     6.0   26.7  218.6  218.6    4.2    0     0
  41    19     631.0     1     6.0   26.7  218.6  215.9    4.2    0     0


PIPE SIZE     TOTAL LENGTH       COST
*********     ************     *******
26.7  mm         22920.         37130.
33.4  mm         15760.         31048.
48.3  mm         20499.         59038.
60.3  mm          2817.         11465.
88.9  mm             0.             0.

TOTAL            61996.        138680.


TOTAL NUMBER OF ROAD  CROSSINGS  35    COST  19725.
TOTAL NUMBER OF RAIL  CROSSINGS   0    COST      0.
TOTAL NUMBER OF CABLE CROSSINGS  42    COST   3780.
TOTAL NUMBER OF CREEK CROSSINGS   0    COST      0.


TOTAL COST OF SYSTEM  162185.
```

Hydraulic Profile for Figure 31

FROM	TO	LENGTH	NUM	LOAD	SIZE	PI	PO	PD	WO	WD
****	****	*******	***	*****	****	*****	*****	*****	****	****
1	23	2816.9	28	106.0	60.3	550.0	531.9	6.4	0	0
23	2	765.0	2	12.0	26.7	531.9	526.5	7.0	0	0
23	61	912.1	26	99.3	48.3	531.9	516.5	16.9	0	0
2	51	1141.0	1	6.0	26.7	526.5	524.1	2.1	0	0
51	26	1127.5	1	6.0	26.7	524.1	521.7	2.1	0	0
26	3	805.0	1	6.0	26.7	521.7	520.0	2.1	0	0
61	44	1703.0	26	99.3	48.3	516.5	486.7	17.5	0	0
44	43	1150.6	22	85.9	48.3	486.7	470.4	14.2	0	0
43	42	1.9	22	85.9	48.3	470.4	470.3	14.4	0	0
42	4	704.9	21	82.6	48.3	470.3	460.9	13.4	0	0
44	36	1496.9	4	24.0	33.4	486.7	474.0	8.5	0	0
36	24	325.5	3	18.0	26.7	474.0	469.2	14.8	0	0
36	37	485.7	1	6.0	26.7	474.0	472.9	2.3	0	0
24	5	1637.9	3	18.0	26.7	469.2	444.4	15.2	0	0
4	58	1703.0	20	79.2	48.3	460.9	438.9	12.9	0	0
5	52	1731.0	2	12.0	26.7	444.4	430.1	8.2	0	0
52	25	1347.8	2	12.0	26.7	430.1	418.7	8.4	0	0
25	6	336.0	1	6.0	26.7	418.7	417.9	2.6	0	0
25	7	1940.7	1	6.0	26.7	418.7	413.7	2.6	0	0
58	57	4706.4	20	79.2	48.3	438.9	372.8	14.0	0	0
57	54	553.9	1	6.0	26.7	372.8	371.2	2.8	0	0
57	27	710.0	19	75.8	48.3	372.8	362.7	14.1	0	0
27	11	333.0	1	6.0	26.7	362.7	361.8	2.9	0	0
27	12	626.9	18	72.4	48.3	362.7	354.4	13.3	0	0
12	46	1010.4	17	68.9	48.3	354.4	342.0	12.3	0	0
46	10	1625.9	4	24.0	33.4	342.0	323.4	11.4	0	0
10	28	909.0	3	18.0	33.4	323.4	317.1	7.0	0	0
28	9	6.0	3	18.0	33.4	317.1	317.1	7.0	0	0
9	59	283.0	2	12.0	26.7	317.1	314.0	10.7	0	0
46	47	1641.2	13	54.4	48.3	342.0	328.3	8.3	0	0
47	29	388.1	12	50.6	48.3	328.3	325.4	7.6	0	0
47	45	412.9	1	6.0	26.7	328.3	327.0	3.1	0	0
29	30	747.0	12	50.6	48.3	325.4	319.7	7.6	0	0
30	62	880.2	4	24.0	33.4	319.7	309.2	11.9	0	0
62	15	20.0	1	6.0	26.7	309.2	309.1	3.3	0	0
62	60	801.0	3	18.0	33.4	309.2	303.4	7.2	0	0
30	14	1231.0	8	39.8	48.3	319.7	313.4	5.1	0	0
14	31	1345.1	7	37.2	48.3	313.4	307.3	4.6	0	0
31	13	295.4	1	6.0	26.7	307.3	306.3	3.3	0	0
31	32	1250.7	6	33.9	48.3	307.3	302.4	3.9	0	0
32	50	263.0	6	33.9	48.3	302.4	301.4	3.9	0	0
50	48	350.0	1	6.0	26.7	301.4	300.2	3.3	0	0
50	49	1844.5	5	30.0	48.3	301.4	295.5	3.2	0	0
49	16	1144.0	1	6.0	26.7	295.5	291.6	3.4	0	0
49	33	1729.4	4	24.0	33.4	295.5	273.2	12.9	0	0

```
FROM   TO    LENGTH    NUM   LOAD    SIZE    PI      PO      PD     WO    WD
****  ****  *******    ***  *****    ****   *****   *****   *****   ****  ****
 33    17     11.0      1     6.0    26.7   273.2   273.2    3.6     0     0
 33    18    809.1      3    18.0    33.4   273.2   266.8    7.9     0     0
 59    53   2050.8      2    12.0    26.7   314.0   291.4   11.0     0     0
 53    55    290.4      1     6.0    26.7   291.4   290.4    3.4     0     0
 55    38    315.4      1     6.0    26.7   290.4   289.4    3.4     0     0
 53     8    734.0      1     6.0    26.7   291.4   288.9    3.4     0     0
 60    56   2122.0      3    18.0    33.4   303.4   287.7    7.4     0     0
 56    20   1462.0      3    18.0    33.4   287.7   276.5    7.7     0     0
 20    34    939.0      2    12.0    26.7   276.5   265.2   12.0     0     0
 34    35    421.6      2    12.0    26.7   265.2   260.1   12.3     0     0
 35    21    712.0      1     6.0    26.7   260.1   257.4    3.7     0     0
 35    22    803.0      1     6.0    26.7   260.1   257.1    3.7     0     0
 18    39   3473.0      2    12.0    26.7   266.8   221.9   12.9     0     0
 39    41    995.4      2    12.0    26.7   221.9   207.8   14.2     0     0
 41    40      1.0      1     6.0    26.7   207.8   207.8    4.4     0     0
 41    19    631.0      1     6.0    26.7   207.8   205.1    4.4     0     0


PIPE SIZE      TOTAL LENGTH       COST
*********      ************      *******
26.7  mm          26413.          42789.
33.4  mm          11842.          23328.
48.3  mm          21940.          63186.
60.3  mm           2817.          11465.
88.9  mm              0.              0.

TOTAL             63011.         140768.


TOTAL NUMBER OF ROAD  CROSSINGS  36    COST  20289.
TOTAL NUMBER OF RAIL  CROSSINGS   0    COST      0.
TOTAL NUMBER OF CABLE CROSSINGS  43    COST   3870.
TOTAL NUMBER OF CREEK CROSSINGS   0    COST      0.


TOTAL COST OF SYSTEM  164926.
```

Hydraulic Profile for Figure 32

FROM	TO	LENGTH	NUM	LOAD	SIZE	PI	PO	PD	WO	WD
****	****	*******	***	*****	****	*****	*****	*****	****	****
1	22	2816.9	28	106.0	60.3	550.0	531.9	6.4	0	0
22	2	765.0	2	12.0	26.7	531.9	526.5	7.0	0	0
2	52	1946.0	1	6.0	26.7	526.5	522.4	2.1	0	0
52	3	1126.5	1	6.0	26.7	522.4	520.0	2.1	0	0
22	41	2615.1	26	99.3	48.3	531.9	486.7	17.3	0	0
41	40	1150.6	22	85.9	48.3	486.7	470.4	14.2	0	0
40	39	1.9	22	85.9	48.3	470.4	470.3	14.4	0	0
39	4	704.9	21	82.6	48.3	470.3	460.9	13.4	0	0
41	33	1496.9	4	24.0	33.4	486.7	474.0	8.5	0	0
33	23	325.5	3	18.0	26.7	474.0	469.2	14.8	0	0
33	34	485.7	1	6.0	26.7	474.0	472.9	2.3	0	0
23	5	1637.9	3	18.0	26.7	469.2	444.4	15.2	0	0
5	48	1731.0	2	12.0	26.7	444.4	430.1	8.2	0	0
48	24	1347.8	2	12.0	26.7	430.1	418.7	8.4	0	0
24	6	336.0	1	6.0	26.7	418.7	417.9	2.6	0	0
24	7	1940.7	1	6.0	26.7	418.7	413.7	2.6	0	0
4	53	4706.4	20	79.2	48.3	460.9	397.7	13.4	0	0
53	57	1370.0	20	79.2	48.3	397.7	377.8	14.6	0	0
57	56	710.0	19	75.8	48.3	377.8	367.8	14.0	0	0
56	55	333.0	18	72.4	48.3	367.8	363.5	13.1	0	0
55	11	626.9	18	72.4	48.3	363.5	355.2	13.2	0	0
57	50	887.0	1	6.0	26.7	377.8	375.3	2.8	0	0
11	43	1010.4	17	68.9	48.3	355.2	342.8	12.3	0	0
43	10	1625.9	4	24.0	33.4	342.8	324.3	11.4	0	0
10	25	909.0	3	18.0	33.4	324.3	317.9	6.9	0	0
25	9	6.0	3	18.0	33.4	317.9	317.9	7.0	0	0
9	58	1147.6	2	12.0	26.7	317.9	305.5	10.8	0	0
58	54	903.2	2	12.0	26.7	305.5	295.5	11.1	0	0
54	49	282.9	2	12.0	26.7	295.5	292.3	11.3	0	0
49	35	605.8	1	6.0	26.7	292.3	290.2	3.4	0	0
49	8	734.0	1	6.0	26.7	292.3	289.8	3.4	0	0
43	44	1641.2	13	54.4	48.3	342.8	329.1	8.3	0	0
44	26	388.1	12	50.6	48.3	329.1	326.2	7.5	0	0
44	42	412.9	1	6.0	26.7	329.1	327.8	3.1	0	0
26	27	747.0	12	50.6	48.3	326.2	320.5	7.6	0	0
27	51	880.2	4	24.0	33.4	320.5	310.0	11.9	0	0
51	14	20.0	1	6.0	26.7	310.0	310.0	3.3	0	0
51	59	799.0	3	18.0	33.4	310.0	304.3	7.2	0	0
27	13	1231.0	8	39.8	48.3	320.5	314.3	5.1	0	0
13	28	1345.1	7	37.2	48.3	314.3	308.1	4.6	0	0
28	12	295.4	1	6.0	26.7	308.1	307.1	3.3	0	0
28	29	1250.7	6	33.9	48.3	308.1	303.3	3.9	0	0
29	47	263.0	6	33.9	48.3	303.3	302.3	3.9	0	0
47	45	350.0	1	6.0	26.7	302.3	301.1	3.3	0	0
47	46	1844.5	5	30.0	48.3	302.3	296.4	3.2	0	0

```
FROM   TO    LENGTH   NUM  LOAD   SIZE   PI     PO     PD    WO    WD
****   ****  *******  ***  *****  ****  *****  *****  *****  ****  ****
 46     15   1144.0    1    6.0   26.7  296.4  292.5   3.4    0     0
 46     30   1729.4    4   24.0   33.4  296.4  274.2  12.8    0     0
 30     16     11.0    1    6.0   26.7  274.2  274.1   3.6    0     0
 30     17    809.1    3   18.0   33.4  274.2  267.8   7.9    0     0
 59     60   2131.0    3   18.0   33.4  304.3  288.5   7.4    0     0
 60     19   1455.0    3   18.0   33.4  288.5  277.4   7.7    0     0
 19     31    939.0    2   12.0   26.7  277.4  266.2  12.0    0     0
 31     32    421.6    2   12.0   26.7  266.2  261.0  12.2    0     0
 32     20    712.0    1    6.0   26.7  261.0  258.4   3.7    0     0
 32     21    803.0    1    6.0   26.7  261.0  258.0   3.7    0     0
 17     36   3473.0    2   12.0   26.7  267.8  223.0  12.9    0     0
 36     38    995.4    2   12.0   26.7  223.0  209.0  14.1    0     0
 38     37      1.0    1    6.0   26.7  209.0  209.0   4.3    0     0
 38     18    631.0    1    6.0   26.7  209.0  206.2   4.4    0     0


PIPE SIZE      TOTAL LENGTH      COST
*********      ************     *******
26.7  mm          26412.         42787.
33.4  mm          11842.         23328.
48.3  mm          21940.         63186.
60.3  mm           2817.         11465.
88.9  mm              0.             0.

TOTAL             63010.        140766.


TOTAL NUMBER OF ROAD  CROSSINGS  36    COST  20289.
TOTAL NUMBER OF RAIL  CROSSINGS   0    COST      0.
TOTAL NUMBER OF CABLE CROSSINGS  43    COST   3870.
TOTAL NUMBER OF CREEK CROSSINGS   0    COST      0.


TOTAL COST OF SYSTEM  164924.
```

Hydraulic Profile for Figure 33

FROM ****	TO ****	LENGTH *******	NUM ***	LOAD *****	SIZE ****	PI *****	PO *****	PD *****	WO ****	WD ****
1	22	2816.9	28	106.0	60.3	550.0	531.9	6.4	0	0
22	2	765.0	2	12.0	26.7	531.9	526.5	7.0	0	0
2	52	1946.0	1	6.0	26.7	526.5	522.4	2.1	0	0
52	3	1126.5	1	6.0	26.7	522.4	520.0	2.1	0	0
22	41	2615.1	26	99.3	48.3	531.9	486.7	17.3	0	0
41	40	1150.6	22	85.9	48.3	486.7	470.4	14.2	0	0
40	39	1.9	22	85.9	48.3	470.4	470.3	14.4	0	0
39	4	704.9	21	82.6	48.3	470.3	460.9	13.4	0	0
41	33	1496.9	4	24.0	33.4	486.7	474.0	8.5	0	0
33	23	325.5	3	18.0	26.7	474.0	469.2	14.8	0	0
33	34	485.7	1	6.0	26.7	474.0	472.9	2.3	0	0
23	5	1637.9	3	18.0	26.7	469.2	444.4	15.2	0	0
5	48	1731.0	2	12.0	26.7	444.4	430.1	8.2	0	0
48	24	1347.8	2	12.0	26.7	430.1	418.7	8.4	0	0
24	6	336.0	1	6.0	26.7	418.7	417.9	2.6	0	0
24	7	1940.7	1	6.0	26.7	418.7	413.7	2.6	0	0
4	53	4706.4	20	79.2	48.3	460.9	397.7	13.4	0	0
53	57	1370.0	20	79.2	48.3	397.7	377.8	14.6	0	0
57	61	289.4	20	79.2	48.3	377.8	373.4	14.9	0	0
61	60	333.0	20	79.2	48.3	373.4	368.4	15.1	0	0
60	55	419.6	17	68.9	48.3	368.4	363.5	11.8	0	0
55	56	333.0	1	6.0	26.7	363.5	362.5	2.9	0	0
60	58	554.1	3	18.0	33.4	368.4	365.0	6.3	0	0
58	50	290.4	1	6.0	26.7	365.0	364.1	2.9	0	0
55	11	626.9	16	65.3	48.3	363.5	356.6	10.9	0	0
11	43	1010.4	15	61.7	48.3	356.6	346.4	10.1	0	0
43	10	1625.9	5	30.0	33.4	346.4	318.5	17.2	0	0
10	25	909.0	4	24.0	33.4	318.5	307.6	12.0	0	0
25	9	6.0	1	6.0	26.7	307.6	307.6	3.3	0	0
25	59	1462.0	3	18.0	33.4	307.6	297.0	7.3	0	0
43	44	1641.2	10	42.9	48.3	346.4	337.6	5.4	0	0
44	26	388.1	9	41.7	48.3	337.6	335.5	5.3	0	0
44	42	412.9	1	6.0	26.7	337.6	336.3	3.1	0	0
26	27	747.0	9	41.7	48.3	335.5	331.5	5.4	0	0
27	51	880.2	1	6.0	26.7	331.5	328.8	3.1	0	0
51	14	20.0	1	6.0	26.7	328.8	328.7	3.1	0	0
27	13	1231.0	8	39.8	48.3	331.5	325.4	4.9	0	0
13	28	1345.1	7	37.2	48.3	325.4	319.4	4.4	0	0
28	12	295.4	1	6.0	26.7	319.4	318.5	3.2	0	0
28	29	1250.7	6	33.9	48.3	319.4	314.8	3.7	0	0
29	47	263.0	6	33.9	48.3	314.8	313.8	3.8	0	0
47	45	350.0	1	6.0	26.7	313.8	312.6	3.2	0	0
47	46	1844.5	5	30.0	48.3	313.8	308.0	3.1	0	0
46	15	1144.0	1	6.0	26.7	308.0	304.3	3.3	0	0
46	30	1729.4	4	24.0	33.4	308.0	286.5	12.4	0	0

```
FROM    TO    LENGTH    NUM   LOAD    SIZE    PI      PO      PD     WO     WD
****   ****   *******   ***   *****   ****   *****   *****   *****   ****   ****
  30    16      11.0     1     6.0    26.7   286.5   286.5     3.5    0      0
  30    17     809.1     3    18.0    33.4   286.5   280.3     7.6    0      0
  58    54    1980.9     2    12.0    26.7   365.0   345.7     9.7    0      0
  54    49     282.9     2    12.0    26.7   345.7   342.8    10.0    0      0
  49    35     605.8     1     6.0    26.7   342.8   341.0     3.0    0      0
  49     8     734.0     1     6.0    26.7   342.8   340.6     3.0    0      0
  59    19    3267.5     3    18.0    33.4   297.0   272.1     7.6    0      0
  19    31     939.0     2    12.0    26.7   272.1   260.7    12.1    0      0
  31    32     421.6     2    12.0    26.7   260.7   255.5    12.4    0      0
  32    20     712.0     1     6.0    26.7   255.5   252.8     3.8    0      0
  32    21     803.0     1     6.0    26.7   255.5   252.4     3.8    0      0
  17    36    3473.0     2    12.0    26.7   280.3   237.3    12.4    0      0
  36    38     995.4     2    12.0    26.7   237.3   223.9    13.5    0      0
  38    37       1.0     1     6.0    26.7   223.9   223.8     4.1    0      0
  38    18     631.0     1     6.0    26.7   223.9   221.2     4.2    0      0


PIPE SIZE      TOTAL LENGTH       COST
*********      ************     *******
26.7   mm         26965.         43683.
33.4   mm         11854.         23352.
48.3   mm         21939.         63183.
60.3   mm          2817.         11465.
88.9   mm             0.             0.

TOTAL             63574.        141683.


TOTAL NUMBER OF ROAD  CROSSINGS  36    COST  20289.
TOTAL NUMBER OF RAIL  CROSSINGS   0    COST     0.
TOTAL NUMBER OF CABLE CROSSINGS  43    COST   3870.
TOTAL NUMBER OF CREEK CROSSINGS   0    COST     0.


TOTAL COST OF SYSTEM  165841.
```

Hydraulic Profile for Figure 34

FROM	TO	LENGTH	NUM	LOAD	SIZE	PI	PO	PD	WO	WD
****	****	*******	***	*****	****	*****	*****	*****	****	****
1	22	2816.9	28	106.0	60.3	550.0	531.9	6.4	0	0
22	2	765.0	2	12.0	26.7	531.9	526.5	7.0	0	0
2	49	1946.0	1	6.0	26.7	526.5	522.4	2.1	0	0
49	3	1126.5	1	6.0	26.7	522.4	520.0	2.1	0	0
22	41	2615.1	26	99.3	48.3	531.9	486.7	17.3	0	0
41	40	1150.6	22	85.9	48.3	486.7	470.4	14.2	0	0
40	39	1.9	22	85.9	48.3	470.4	470.3	14.4	0	0
39	4	704.9	21	82.6	48.3	470.3	460.9	13.4	0	0
41	33	1496.9	4	24.0	33.4	486.7	474.0	8.5	0	0
33	23	325.5	3	18.0	26.7	474.0	469.2	14.8	0	0
33	34	485.7	1	6.0	26.7	474.0	472.9	2.3	0	0
23	5	1637.9	3	18.0	26.7	469.2	444.4	15.2	0	0
5	48	1731.0	2	12.0	26.7	444.4	430.1	8.2	0	0
48	24	1347.8	2	12.0	26.7	430.1	418.7	8.4	0	0
24	6	336.0	1	6.0	26.7	418.7	417.9	2.6	0	0
24	7	1940.7	1	6.0	26.7	418.7	413.7	2.6	0	0
4	50	4706.4	20	79.2	48.3	460.9	397.7	13.4	0	0
50	54	1370.0	20	79.2	48.3	397.7	377.8	14.6	0	0
54	52	710.0	17	68.9	48.3	377.8	369.5	11.6	0	0
52	51	333.0	16	65.3	48.3	369.5	366.0	10.7	0	0
51	11	626.9	16	65.3	48.3	366.0	359.2	10.8	0	0
54	53	887.0	3	18.0	33.4	377.8	372.3	6.2	0	0
11	43	1010.4	15	61.7	48.3	359.2	349.0	10.1	0	0
53	58	1239.1	2	12.0	26.7	372.3	360.5	9.5	0	0
58	56	1024.9	2	12.0	26.7	360.5	350.6	9.7	0	0
56	57	290.4	1	6.0	26.7	350.6	349.7	3.0	0	0
56	35	315.4	1	6.0	26.7	350.6	349.6	3.0	0	0
57	8	734.0	1	6.0	26.7	349.7	347.5	3.0	0	0
43	10	1625.9	5	30.0	33.4	349.0	321.3	17.1	0	0
10	25	909.0	4	24.0	33.4	321.3	310.5	11.9	0	0
25	9	6.0	1	6.0	26.7	310.5	310.5	3.3	0	0
43	44	1641.2	10	42.9	48.3	349.0	340.2	5.4	0	0
44	26	388.1	9	41.7	48.3	340.2	338.2	5.3	0	0
44	42	412.9	1	6.0	26.7	340.2	339.0	3.0	0	0
26	27	747.0	9	41.7	48.3	338.2	334.2	5.3	0	0
27	14	900.2	1	6.0	26.7	334.2	331.4	3.1	0	0
27	13	1231.0	8	39.8	48.3	334.2	328.1	4.9	0	0
13	28	1345.1	7	37.2	48.3	328.1	322.2	4.4	0	0
28	12	295.4	1	6.0	26.7	322.2	321.3	3.2	0	0
28	29	1250.7	6	33.9	48.3	322.2	317.5	3.7	0	0
29	47	263.0	6	33.9	48.3	317.5	316.6	3.7	0	0
47	45	350.0	1	6.0	26.7	316.6	315.4	3.2	0	0
25	59	1658.3	3	18.0	33.4	310.5	298.5	7.2	0	0
59	55	1610.2	3	18.0	33.4	298.5	286.5	7.5	0	0
55	19	1462.0	3	18.0	33.4	286.5	275.2	7.7	0	0

```
FROM   TO    LENGTH   NUM  LOAD   SIZE   PI     PO     PD    WO   WD
****  ****  *******   ***  *****  ****  *****  *****  *****  **** ****
  19    31    939.0    2   12.0   26.7  275.2  263.9  12.0    0    0
  31    32    421.6    2   12.0   26.7  263.9  258.7  12.3    0    0
  32    20    712.0    1    6.0   26.7  258.7  256.0   3.8    0    0
  32    21    803.0    1    6.0   26.7  258.7  255.7   3.8    0    0
  47    46   1844.5    5   30.0   48.3  316.6  310.9   3.1    0    0
  46    15   1144.0    1    6.0   26.7  310.9  307.1   3.3    0    0
  46    30   1729.4    4   24.0   33.4  310.9  289.5  12.4    0    0
  30    16     11.0    1    6.0   26.7  289.5  289.5   3.4    0    0
  30    17    809.1    3   18.0   33.4  289.5  283.4   7.6    0    0
  17    36   3473.0    2   12.0   26.7  283.4  240.7  12.3    0    0
  36    38    995.4    2   12.0   26.7  240.7  227.4  13.3    0    0
  38    37      1.0    1    6.0   26.7  227.4  227.4   4.1    0    0
  38    18    631.0    1    6.0   26.7  227.4  224.8   4.1    0    0

PIPE SIZE      TOTAL LENGTH      COST
*********      ************     *******
26.7  mm          26341.         42673.
33.4  mm          12188.         24010.
48.3  mm          21940.         63186.
60.3  mm           2817.         11465.
88.9  mm              0.             0.

TOTAL             63286.        141334.


TOTAL NUMBER OF ROAD  CROSSINGS 36   COST  20289.
TOTAL NUMBER OF RAIL  CROSSINGS  0   COST      0.
TOTAL NUMBER OF CABLE CROSSINGS 43   COST   3870.
TOTAL NUMBER OF CREEK CROSSINGS  0   COST      0.


TOTAL COST OF SYSTEM  165492.
```

APPENDIX B: SOURCE CODE FOR PROCEDURAL COMPONENT

```
#include <stdio.h>
#include "size.c"

       int n;                 /*   the number of nodes                  */
       float x[SIZE];         /*   the x-coordinate of a node           */
       float y[SIZE];         /*   the y-coordinate of a node           */
       int sink[SIZE];        /*   flag ( 0 = dummy node )              */
       int deg[SIZE];         /*   the degree of a node                 */
       float dist[SIZE];     /*    the distance to the source           */
       int l;                 /*   the number of lines                  */
       int ifr[SIZE];         /*   origin node of a line                */
       int ito[SIZE];         /*   destination node of a line           */
       float xo[SIZE];        /*   x-coord of origin node               */
       float yo[SIZE];        /*   y-coord of origin node               */
       float xd[SIZE];        /*   x-coord of destination node          */
       float yd[SIZE];        /*   y-coord of destination node          */
       float alen[SIZE];     /*    length of a line                     */
       float a[SIZE][SIZE]; /*    adjacency matrix                     */
       int lmax;      /*   the highest level in a tree          */
       int lev[SIZE];         /*   the level of a line                  */
       int diag;      /*   flag ( 0 = no diagonal lines )       */
       int overflow;          /*   flag ( 1 = too many nodes )          */
       int stopbu;            /*   flag ( 1 = backup complete)          */
       float rpi;             /*   performance index                    */
       int i;                 /*   counter                              */

#include "rstpro.c"

void main()
{
    nread(&n, x, y, sink);
    printf("First iteration\nBeginning MST\n");
    table(n, x, y, a);
    mst2(n, x, y, a, &l, ifr, ito, xo, yo, xd, yd, dist);
    printf("Boxes\n");
    boxes3(l, ifr, ito, xo, yo, xd, yd, &n, x, y, sink, a, &diag,
    &overflow);
    printf("Graph\n");
    graph(n, x, y, a, &l, ifr, ito, xo, yo, xd, yd);
    printf("Achange\n");
    achange(l, ifr, ito, xo, yo, xd, yd, a);
    printf("Beginning Dijkstra\n");
    dijk2(n, x, y, a, &l, ifr, ito, xo, yo, xd, yd, dist);
    printf("Entering Backup\n");
    backup(dist, l, ifr, ito, a, &stopbu);
    for (i = 1; i <= 4; i++) {
       if (stopbu == 1) break;
       dijk2(n, x, y, a, &l, ifr, ito, xo, yo, xd, yd, dist);
       backup(dist, l, ifr, ito, a, &stopbu);
       }
    printf("Backup complete\n");
    level(l, ifr, ito, &lmax, lev);
    cleanup(&n, x, y, sink, dist, &l, ifr, ito, xo, yo, xd, yd,
     &lmax, lev, a);
    table(n, x, y, a);
    mst2(n, x, y, a, &l, ifr, ito, xo, yo, xd, yd, dist);
    boxes3(l, ifr, ito, xo, yo, xd, yd, &n, x, y, sink, a, &diag,
    &overflow);
    while (diag == 1) {
       printf("New iteration  diag = %d\n", diag);
       graph(n, x, y, a, &l, ifr, ito, xo, yo, xd, yd);
       achange(l, ifr, ito, xo, yo, xd, yd, a);
       printf("Entering Dijkstra\n");
       dijk2(n, x, y, a, &l, ifr, ito, xo, yo, xd, yd, dist);
       printf("Dijkstra complete\n");
       level(l, ifr, ito, &lmax, lev);
       cleanup(&n, x, y, sink, dist, &l, ifr, ito, xo, yo, xd, yd,
        &lmax, lev, a);
       table(n, x, y, a);
       mst2(n, x, y, a, &l, ifr, ito, xo, yo, xd, yd, dist);
       boxes3(l, ifr, ito, xo, yo, xd, yd, &n, x, y, sink, a, &diag,
       &overflow);
    }
    printf("Iterations complete");
    level(l, ifr, ito, &lmax, lev);
    cleanup(&n, x, y, sink, dist, &l, ifr, ito, xo, yo, xd, yd,
     &lmax, lev, a);
    table(n, x, y, a);
    mst2(n, x, y, a, &l, ifr, ito, xo, yo, xd, yd, dist);
    rindex(n, x, y, sink, l, xo, yo, xd, yd, alen, &rpi);
    degree(n, l, ifr, deg);
    rbwrit(n, x, y, sink, deg, l, ifr, ito);
    nwrit(n, x, y, sink);
    lwrit2(l, xo, yo, xd, yd, rpi);
    con2(n, x, y, sink);
}
```

```
/* This line should be included in a separate file called "size.c" */
#define SIZE 120

/* The following should be included in a separate file called "rstpro.c"
*/
void main(void);
void nread(int*, float[SIZE], float[SIZE], int[SIZE]);
void table(int, float[SIZE], float[SIZE], float[SIZE][SIZE]);
void mst2(int, float[SIZE], float[SIZE], float[SIZE][SIZE], int*,
   int[SIZE], int[SIZE], float[SIZE], float[SIZE], float[SIZE],
   float[SIZE], float[SIZE]);
void boxes3(int, int[SIZE], int[SIZE], float[SIZE], float[SIZE],
     float[SIZE], float[SIZE], int*, float[SIZE], float[SIZE],
     int[SIZE], float[SIZE][SIZE], int*, int*);
void graph(int, float[SIZE], float[SIZE], float[SIZE][SIZE], int*,
    int[SIZE], int[SIZE], float[SIZE], float[SIZE],
    float[SIZE], float[SIZE]);
void achange(int, int[SIZE], int[SIZE], float[SIZE], float[SIZE],
      float[SIZE], float[SIZE], float[SIZE][SIZE]);
void dijk2(int, float[SIZE], float[SIZE], float[SIZE][SIZE], int*,
    int[SIZE], int[SIZE], float[SIZE], float[SIZE], float[SIZE],
    float[SIZE], float[SIZE]);
void backup(float[SIZE], int, int[SIZE], int[SIZE],
     float[SIZE][SIZE], int*);
void level(int, int[SIZE], int[SIZE], int*, int[SIZE]);
void cleanup(int*, float[SIZE], float[SIZE], int[SIZE], float[SIZE],
      int*, int[SIZE], int[SIZE], float[SIZE], float[SIZE],
      float[SIZE], float[SIZE], int*, int[SIZE],
      float[SIZE][SIZE]);
void rindex(int, float[SIZE], float[SIZE], int[SIZE], int,
     float[SIZE], float[SIZE], float[SIZE], float[SIZE],
     float[SIZE], float*);
void degree(int, int, int[SIZE], int[SIZE]);
void rbwrit(int, float[SIZE], float[SIZE], int[SIZE], int[SIZE],
     int, int[SIZE], int[SIZE]);
void nwrit(int, float[SIZE], float[SIZE], int[SIZE]);
void lwrit2(int, float[SIZE], float[SIZE], float[SIZE], float[SIZE],
     float);
void con2(int, float[SIZE], float[SIZE], int[SIZE]);
```

```
/**********************************************************************/
/*                                                                    */
/*   subroutine nread                                                 */
/*                                                                    */
/*       This subroutine reads in the values of the coorinates       */
/*       of nodes in the nodes file.                                  */
/*                                                                    */
/*                                                                    */
/**********************************************************************/

/*   nread(&n, x, y, sink);                                           */

#include <stdio.h>
#include <math.h>
#include "size.c"

void nread(int*, float[SIZE], float[SIZE], int[SIZE]);

void nread
(
/*  output parameters                                                 */
       int *n,                /*  the number of nodes                 */
       float x[SIZE],         /*  the x-coordinate of a point         */
       float y[SIZE],         /*  the y-coordinate of a point         */
       int sink[SIZE]         /*  flag ( 0 = dummy node )             */
)
{
/*  local variables                                                   */
       int i;                 /*  a counter                           */
       char line[80];         /*  input buffer                        */
       FILE *sysin;           /*  input stream                        */

   sysin = fopen("nodes2", "r");
   fgetc(sysin);
   fgets(line, 80, sysin);
   *n = atoi(line);
   for (i = 1; i <= *n; ++i) {
      fgets(line, 80, sysin);
      x[i] = atof(line);
      fgets(line, 80, sysin);
      y[i] = atof(line);
      fgets(line, 80, sysin);
      sink[i] = atoi(line);
   }
   fclose(sysin);
   return;
}
```

```
/************************************************************************/
/*                                                                      */
/*   subroutine table                                                   */
/*                                                                      */
/*       This subroutine computes the lengths in the adjacency          */
/*       matrix.                                                        */
/*                                                                      */
/*                                                                      */
/************************************************************************/

/*   table(n, x, y, a);                                                */

#include <math.h>
#include "size.c"

void table(int, float[SIZE], float[SIZE], float[SIZE][SIZE]);

void table
(
/*  input parameters                                                    */
       int n,                /*  the number of nodes                    */
       float x[SIZE],        /*  the x-coordinate of a node             */
       float y[SIZE],        /*  the y-coordinate of a node             */
/*  output parameters                                                   */
       float a[SIZE][SIZE]   /*  the adjacency matrix                   */
)
{
/*  local variables                                                     */
       float dx;               /*  x displacement of two nodes          */
       float dy;               /*  y displacement of two nodes          */
       int i, j;               /*  counters                             */

   for (i = 1; i <= n - 1; ++i)
      for (j = i + 1; j <= n; ++j) {
         dx = fabs(x[i] - x[j]);
         dy = fabs(y[i] - y[j]);
         a[i][j] = sqrt((dx * dx) + (dy * dy));
         a[j][i] = a[i][j];
      }
   for (i = 1; i <= n; ++i)
      a[i][i] = -1.0;
   return;
}
```

```
/*********************************************************************/
/*                                                                   */
/*   subroutine mst2                                                 */
/*                                                                   */
/*       This subroutine performs the minimum spanning tree          */
/*       algorithm using the indirection method.                     */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
#include "size.c"

void mst2(int, float[SIZE], float[SIZE], float[SIZE][SIZE], int*,
   int[SIZE], int[SIZE], float[SIZE], float[SIZE], float[SIZE],
   float[SIZE], float[SIZE]);

void mst2
(
/*  input parameters                                                 */
       int n,                /*  the number of nodes                 */
       float x[SIZE],        /*  the x-coordinate of a point         */
       float y[SIZE],        /*  the y-coordinate of a point         */
       float a[SIZE][SIZE],  /*  the adjacency matrix                */
/*  output parameters                                                */
       int *l,               /*  the number of lines                 */
       int ifr[SIZE],        /*  the origin node of a line           */
       int ito[SIZE],        /*  the destination node of a line      */
       float xo[SIZE],       /*  the x-coord of the origin           */
       float yo[SIZE],       /*  the y-coord of the origin           */
       float xd[SIZE],       /*  the x-coord of the destination      */
       float yd[SIZE],       /*  the y-coord of the destination      */
       float dist[SIZE]      /*  the distance from the source        */
)
{
/*  local variables                                                  */
       int nrow;             /*  the number of nodes in the tree     */
       int ncol;             /*  the number of nodes not in the tree */
       int row[100];         /*  the nodes in the tree               */
       int col[100];         /*  the nodes not in the tree           */
       float alow, comp;     /*  temporary variables                 */
       int il, jl;           /*  captures new line in the tree       */
       int i, j;             /*  counters                            */

   *l = n - 1;
   ncol = n;
   for (i = 1; i <= n; ++i)
      col[i] = i;
   jl = 1;
   dist[1] = 0.0;
   for (nrow = 1; nrow <= *l; ++nrow) {
      row[nrow] = jl;
      j = 1;
      for (i = 1; i <= ncol; ++i)
```

```
         if (col[i] != row[nrow]) {
            col[j] = col[i];
            ++j;
         }
      --ncol;
      alow = 1.0E30;
      for (i = 1; i <= nrow; ++i)
         for (j = 1; j <= ncol; ++j) {
            comp = a[row[i]][col[j]];
            if (comp < alow) {
               alow = comp;
               il = row[i];
               jl = col[j];
            }
         }
      ifr[nrow] = il;
      ito[nrow] = jl;
      dist[jl] = dist[il] + alow;
   }
   for (i = 1; i <= *l; ++i) {
      xo[i] = x[ifr[i]];
      yo[i] = y[ifr[i]];
      xd[i] = x[ito[i]];
      yd[i] = y[ito[i]];
   }
}
```

```
/***********************************************************************/
/*                                                                     */
/*   subroutine boxes3                                                 */
/*                                                                     */
/*       This subroutine finds the rectilinear cycles for each         */
/*       line in the minimum spanning tree. The node coordinates       */
/*       and the adjacency matrix are updated and returned. The        */
/*       flag "diag" is set to 1 if diagonal lines are found. The      */
/*       flag "overflow" is set to 1 if the maximum number of nodes    */
/*       specified by "SIZE" is exceeded.                              */
/*                                                                     */
/*       calls:  b_error      (integer function)                       */
/*                                                                     */
/*                                                                     */
/***********************************************************************/

/*   boxes3(l, ifr, ito, xo, yo, xd, yd, &n, x, y, sink, a, &diag,     */
/*        &overflow);                                                  */

#include <stdio.h>
#include <math.h>
#include "size.c"

void boxes3(int, int[SIZE], int[SIZE], float[SIZE], float[SIZE],
     float[SIZE], float[SIZE], int*, float[SIZE], float[SIZE],
     int[SIZE], float[SIZE][SIZE], int*, int*);

int b_error(int);

void boxes3
(
/*  input parameters                                                   */
       int l,                  /*  the number of lines                 */
       int ifr[SIZE],          /*  the origin node of a line           */
       int ito[SIZE],          /*  the destination node of a line      */
       float xo[SIZE],         /*  x-coord of origin node              */
       float yo[SIZE],         /*  y-coord of origin node              */
       float xd[SIZE],         /*  x-coord of destination node         */
       float yd[SIZE],         /*  y-coord of destination node         */
/*  output parameters                                                  */
       int *n,                 /*  the number of nodes                 */
       float x[SIZE],          /*  x-coordinate of a node              */
       float y[SIZE],          /*  y-coordinate of a node              */
       int sink[SIZE],         /*  flag ( 0 = dummy node )             */
       float a[SIZE][SIZE],    /*  adjacency matrix                    */
       int *diag,              /*  flag ( 0 = no diagonal lines )      */
       int *overflow           /*  flag ( 1 = too many nodes )         */
)
{
/*  local variables                                                    */
       float dx;               /*  x displacement                      */
       float dy;               /*  y displacement                      */
     int i, j, k;              /*  counters                                          */

   k = *n + (2 * l);
   if (k > SIZE - 1)
      k = SIZE - 1;
   for (i = 1; i <= k; ++i)
      for (j = 1; j <= k; ++j)
         a[i][j] = -1.0;
   *diag = 0;
   *overflow = 0;
   j = *n + 1;
   i = 1;
   while ((i <= l) && (*overflow == 0)) {
      if (xo[i] == xd[i]) {
         dy = fabs(yd[i] - yo[i]);
         a[ifr[i]][ito[i]] = dy;
         a[ito[i]][ifr[i]] = dy;
      } else {
         if (yo[i] == yd[i]) {
            dx = fabs(xd[i] - xo[i]);
            a[ifr[i]][ito[i]] = dx;
            a[ito[i]][ifr[i]] = dx;
         } else {
            *diag = 1;
            dx = fabs(xd[i] - xo[i]);
            dy = fabs(yd[i] - yo[i]);
            x[j] = xo[i];
            y[j] = yd[i];
            sink[j] = 0;
            a[ifr[i]][j] = dy;
            a[j][ifr[i]] = dy;
            a[ito[i]][j] = dx;
            a[j][ito[i]] = dx;
            j++;
            *overflow = b_error(j);
            x[j] = xd[i];
            y[j] = yo[i];
            sink[j] = 0;
            a[ifr[i]][j] = dx;
            a[j][ifr[i]] = dx;
            a[ito[i]][j] = dy;
            a[j][ito[i]] = dy;
            j++;
            *overflow = b_error(j);
         }
      }
      ++i;
   }
   *n = --j;
   return;
}
```

```
/*                                                                       */
/*   This function determines if the maximum number of nodes has         */
/*   been exceeded.                                                      */
/*                                                                       */
int b_error
(
   int i
)
{
   int j;

   if (i >= SIZE) {
      j = 1;
      printf("Error: Maximum number of nodes exceeded.\n");
   } else
      j = 0;
   return(j);
}
```

```
/*************************************************************************/
/*                                                                       */
/*   subroutine graph                                                    */
/*                                                                       */
/*       This subroutine returns the from-to table and the               */
/*       coordinates of lines given the coordinates of nodes             */
/*       and the adjacency matrix. Only half the adjacency               */
/*       matrix is used. This subroutine should only be used             */
/*       if directionality is not important.                             */
/*                                                                       */
/*                                                                       */
/*************************************************************************/

/*   graph(n, x, y, a, &l, ifr, ito, xo, yo, xd, yd);                    */

#include "size.c"

void graph(int, float[SIZE], float[SIZE], float[SIZE][SIZE], int*,
    int[SIZE], int[SIZE], float[SIZE], float[SIZE],
    float[SIZE], float[SIZE]);

void graph
(
/*  input parameters                                                     */
       int n,               /*  the number of nodes                      */
       float x[SIZE],       /*  the x-coordinate of a node               */
       float y[SIZE],       /*  the y-coordinate of a node               */
       float a[SIZE][SIZE], /*  the adjacency matrix                     */
/*  output parameters                                                    */
       int *l,              /*  the number of lines                      */
       int ifr[SIZE],       /*  the origin node of a line                */
       int ito[SIZE],       /*  the destination node of a line           */
       float xo[SIZE],      /*  x-coord of origin node                   */
       float yo[SIZE],      /*  y-coord of origin node                   */
       float xd[SIZE],      /*  x-coord of destination node              */
       float yd[SIZE]       /*  y-coord of destination node              */
)
{
/*  local variable                                                       */
       int i, j, k;         /*  counters                                 */


   k = 1;
   for (i = 1; i <= n - 1; ++i)
      for (j = i + 1; j <= n; ++j)
         if (a[i][j] >= 0.0) {
            ifr[k] = i;
            xo[k] = x[i];
            yo[k] = y[i];
            ito[k] = j;
            xd[k] = x[j];
            yd[k] = y[j];
            k++;
          }
   *l = k -1;
   return;
}
```

```
/************************************************************************/
/*                                                                      */
/*   subroutine achange                                                 */
/*                                                                      */
/*       This subroutine removes the double lines from the              */
/*       adjacency matrix of a rectilinear cycles supergragh.           */
/*       The double line segment is replaced by a line segment          */
/*       that is half the actual length. The procedure returns          */
/*       the adjacency matrix.                                          */
/*                                                                      */
/*       calls :  achange1      (procedure)                             */
/*                                                                      */
/*                                                                      */
/************************************************************************/

/*   achange(l, ifr, ito, xo, yo, xd, yd, a);                           */

#include <math.h>
#include "size.c"
void achange(int, int[SIZE], int[SIZE], float[SIZE], float[SIZE],
      float[SIZE], float[SIZE], float[SIZE][SIZE]);

void achange1(int, int, double, double, double, int, int, double,
       double, double, float[SIZE][SIZE]);

void achange
(
/*  input parameters                                                    */
       int l,               /*  the number of lines                     */
       int ifr[SIZE],       /*  the origin node of a line               */
       int ito[SIZE],       /*  the desination node of a line           */
       float xo[SIZE],      /*  the x-coord of the origin               */
       float yo[SIZE],      /*  the y-coord of the origin               */
       float xd[SIZE],      /*  the x-coord of the destination          */
       float yd[SIZE],      /*  the y-coord of the destination          */
/*  output parameters                                                   */
       float a[SIZE][SIZE]  /*  the adjacency matrix                    */
)
{
/*  local variables                                                     */
       float di;            /*  the displacement of the i line          */
       float dj;            /*  the displacement of the j line          */
       int i, j;            /*  counters                                */

   for (i = 1; i <= l -1; ++i)
      if (xo[i] == xd[i]) {
         di = fabs(yd[i] - yo[i]);
         for (j = i + 1; j <= l; ++j)
            if (xo[j] == xd[j] && xo[i] == xo[j]) {
               dj = fabs(yd[j] - yo[j]);
               achange1(ifr[i], ito[i], yo[i], yd[i], di,
                        ifr[j], ito[j], yo[j], yd[j], dj, a);
```

```
                    }
            } else {
               di = fabs(xd[i] - xo[i]);
               for (j = i + 1; j <= l; ++j)
               if (yo[j] == yd[j] && yo[i] == yo[j]) {
                  dj = fabs(xd[j] - xo[j]);
                  achange1(ifr[i], ito[i], xo[i], xd[i], di,
                           ifr[j], ito[j], xo[j], xd[j], dj, a);
               }
            }
         return;
      }
```

```
/**********************************************************************/
/*                                                                    */
/*   subroutine achange1                                              */
/*                                                                    */
/*       This subroutine determines the order that should be          */
/*       used to pass parameters to the subroutine achange2.          */
/*                                                                    */
/*       calls :  achange2        (procedure)                         */
/*                                                                    */
/*                                                                    */
/**********************************************************************/
#define I_POINTS_POS (i_origin < i_dest)
#define J_POINTS_POS (j_origin < j_dest)
#define COMMON(X, Y) (X == Y)
#define I_GREATER (zi > zj)
#define SAME 1
#define DIFF 2

void achange2(int, int, int, double, double, float[SIZE][SIZE]);

void achange1
(
/*  input parameters                                                  */
       int oi,               /*  the origin node of line i            */
       int di,               /*  the destination node of line         */
       double i_origin,      /*  the coord of the i origin            */
       double i_dest,        /*  the coord of the i destination       */
       double zi,            /*  the displacement of line i           */
       int oj,               /*  the origin node of line i            */
       int dj,               /*  the destination node of line         */
       double j_origin,      /*  the coord of the i origin            */
       double j_dest,        /*  the coord of the i destination       */
       double zj,            /*  the displacement of line j           */
/*  output parameters                                                 */
       float a[SIZE][SIZE]   /*  the adjacency matrix                 */
)
{
/*  local variables                                                   */
       int direct;           /*  the direction of lines i and j       */

   if I_POINTS_POS
      if J_POINTS_POS
         direct = SAME ;
      else
         direct = DIFF ;
   else
      if J_POINTS_POS
         direct = DIFF ;
      else
         direct = SAME ;

   switch (direct) {
      case SAME :
         if COMMON(i_origin, j_origin)
            if I_GREATER
               achange2(oi, dj, di, zj, zi, a);
            else
               achange2(oi, di, dj, zi, zj, a);
         else if COMMON(i_dest, j_dest)
            if I_GREATER
               achange2(di, oj, oi, zj, zi, a);
            else
               achange2(di, oi, oj, zi, zj, a);
         break;
      case DIFF :
         if COMMON(i_origin, j_dest)
            if I_GREATER
               achange2(oi, oj, di, zj, zi, a);
            else
               achange2(oi, di, oj, zi, zj, a);
         else if COMMON(i_dest, j_origin)
            if I_GREATER
               achange2(di, dj, oi, zj, zi, a);
            else
               achange2(di, oi, dj, zi, zj, a);
         break;
   }
   return;
}
```

```
/***********************************************************************/
/*                                                                     */
/*   subroutine achange2                                               */
/*                                                                     */
/*       This subroutine removes the line from the common point        */
/*       to the end point in the adjacenceny matrix. A line from       */
/*       the common point to the mid point is entered at half the      */
/*       actual length and a line from the mid point to the end        */
/*       is entered at the actual length.                              */
/*                                                                     */
/*                                                                     */
/***********************************************************************/
void achange2
(
/*  input parameters                                                   */
       int common,              /*  the number of the common point     */
       int mid,                 /*  the number of the mid point        */
       int end,                 /*  the number of the end point        */
       double short_seg,        /*  distance from common to end        */
       double long_seg,         /*  distance from common to mid        */
/*  output parameters                                                  */
       float a[SIZE][SIZE]      /*  the adjacency matrix               */
)
{
/*  local variables                                                    */
       float seg1;              /*  half the common distance           */
       float seg2;              /*  the distance from mid to end       */

   seg1 = short_seg / 2.0;
   seg2 = long_seg - short_seg;
   a[common][end] = -1.0;
   a[end][common] = -1.0;
   a[common][mid] = seg1;
   a[mid][common] = seg1;
   a[mid][end] = seg2;
   a[end][mid] = seg2;
   return;
}
```

```
/**********************************************************************/
/*                                                                    */
/*   subroutine dijk2                                                 */
/*                                                                    */
/*       This subroutine finds the shortest path from the source      */
/*       to each node using the indirection method.                   */
/*                                                                    */
/*                                                                    */
/**********************************************************************/

/*   dijk2(n, x, y, a, &l, ifr, ito, xo, yo, xd, yd, dist);            */

#include "size.c"

void dijk2(int, float[SIZE], float[SIZE], float[SIZE][SIZE], int*,
    int[SIZE], int[SIZE], float[SIZE], float[SIZE], float[SIZE],
    float[SIZE], float[SIZE]);

void dijk2
(
/*  input parameters                                                  */
       int n,                /*  the number of nodes                  */
       float x[SIZE],        /*  the x-coordinate of a point          */
       float y[SIZE],        /*  the y-coordinate of a point          */
       float a[SIZE][SIZE],  /*  the adjacency matrix                 */
/*  output parameters                                                 */
       int *l,               /*  the number of lines                  */
       int ifr[SIZE],        /*  the origin node of a line            */
       int ito[SIZE],        /*  the destination node of a line       */
       float xo[SIZE],       /*  the x-coord of the origin            */
       float yo[SIZE],       /*  the y-coord of the origin            */
       float xd[SIZE],       /*  the x-coord of the destination       */
       float yd[SIZE],       /*  the y-coord of the destination       */
       float dist[SIZE]      /*  the distance from the source         */
)
{
/*  local variables                                                   */
       int nrow;             /*  the number of nodes in the tree      */
       int ncol;             /*  the number of nodes not in the tree  */
       int row[100];         /*  the nodes in the tree                */
       int col[100];         /*  the nodes not in the tree            */
       float alow, comp;     /*  temporary variables                  */
       int il, jl;           /*  captures new line in the tree        */
       int i, j;             /*  counters                             */

   *l = n - 1;
   ncol = n;
   for (i = 1; i <= n; ++i)
      col[i] = i;
   jl = 1;
   dist[1] = 0.0;
   for (nrow = 1; nrow <= *l; ++nrow) {
      row[nrow] = jl;
      j = 1;
      for (i = 1; i <= ncol; ++i)
         if (col[i] != row[nrow]) {
            col[j] = col[i];
            ++j;
         }
      --ncol;
      alow = 1.0E30;
      for (i = 1; i <= nrow; ++i)
         for (j = 1; j <= ncol; ++j) {
            if (a[row[i]][col[j]] >= 0.0) {
               comp = dist[row[i]] + a[row[i]][col[j]];
               if (comp < alow) {
                  alow = comp;
                  il = row[i];
                  jl = col[j];
               }
            }
         }
      ifr[nrow] = il;
      ito[nrow] = jl;
      dist[jl] = alow;
   }
   for (i = 1; i <= *l; ++i) {
      xo[i] = x[ifr[i]];
      yo[i] = y[ifr[i]];
      xd[i] = x[ito[i]];
      yd[i] = y[ito[i]];
   }
   return;
}
```

```
/*********************************************************************/
/*                                                                   */
/*   subroutine backup                                               */
/*                                                                   */
/*       This subroutine assigns a length of zero to all segments    */
/*       in the longest leg of a tree.                               */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

/*   backup(dist, l, ifr, ito, a, &stopbu);                          */

#include "size.c"

void backup(float[SIZE], int, int[SIZE], int[SIZE],
     float[SIZE][SIZE], int*);

void backup
(
/*  input paramters                                                  */
       float dist[SIZE],       /*  the distance from the source      */
       int l,                  /*  the number of lines               */
       int ifr[SIZE],          /*  the origin node of a line         */
       int ito[SIZE],          /*  the destination node of a line    */
/*  output paramters                                                 */
       float a[SIZE][SIZE],    /*  the adjacency matrix              */
       int *stopbu             /*  flag ( 1 = stop iterations )      */
)
{
/*  local variables                                                  */
       int fpoint;             /*  furthest point from the source    */
       float maxdist;          /*  maximum distance                  */
       int line;               /*  temporary pointer to a line       */
       int i;                  /*  counter                           */


   fpoint = ito[l];
   maxdist = dist[fpoint];
   if (maxdist > 0.0)
      while (fpoint > 1) {
         for (i = 1; i <= l; ++i)
            if (ito[i] == fpoint) {
               line = i;
               break;
            }
         a[ifr[line]][ito[line]] = 0.0;
         fpoint = ifr[line];
      }
   else
      *stopbu = 1;
   return;
}
```

```
/***********************************************************************/
/*                                                                     */
/*   subroutine level                                                  */
/*                                                                     */
/*       This subroutine finds and returns the level of each line      */
/*       in the tree as well as the highest level in the tree.         */
/*                                                                     */
/*                                                                     */
/***********************************************************************/

/*   level(l, ifr, ito, &lmax, lev);                                   */

#include "size.c"

void level(int, int[SIZE], int[SIZE], int*, int[SIZE]);

void level
(
/*  input parameters                                                   */
       int l,                /*  the number of lines                   */
       int ifr[SIZE],        /*  the origin node of a line             */
       int ito[SIZE],        /*  the destination node of a line        */
/*  output parameters                                                  */
       int *lmax,            /*  the highest line level in the tree    */
       int lev[SIZE]         /*  the level of a line                   */
)
{
/*  local variables                                                    */
       int i, j;             /*  counters                              */

   *lmax = 1;
   for (i =1; i <= l; ++i)
      if (ifr[i] == 1)
  lev[i] = 1;
      else
  for (j = 1; j <= l; ++j)
     if (ito[j] == ifr[i]) {
        lev[i] = lev[j] + 1;
        if (lev[i] > *lmax)
           *lmax = lev[i];
        break;
     }
   return;
}
```

```
/***********************************************************************/
/*                                                                     */
/*   subroutine cleanup                                                */
/*                                                                     */
/*       This subroutine removes all redundant nodes and lines         */
/*       from the tree.                                                */
/*                                                                     */
/*                                                                     */
/***********************************************************************/

/*   cleanup(&n, x, y, sink, dist, &l, ifr, ito, xo, yo, xd, yd,      */
/*            &lmax, lev, a);                                          */

#include "size.c"

void cleanup(int*, float[SIZE], float[SIZE], int[SIZE], float[SIZE],
      int*, int[SIZE], int[SIZE], float[SIZE], float[SIZE],
      float[SIZE], float[SIZE], int*, int[SIZE],
      float[SIZE][SIZE]);

void cleanup
(
/*  input/output parameters                                            */
       int *n,                /*  the number of nodes                  */
       float x[SIZE],         /*  the x-coordinate of a node           */
       float y[SIZE],         /*  the y-coordinate of a node           */
       int sink[SIZE],        /*  flag ( 0 = dummy node )              */
       float dist[SIZE],      /*  distance from the source             */
       int *l,                /*  the number of lines                  */
       int ifr[SIZE],         /*  the origin node of a line            */
       int ito[SIZE],         /*  the destination node of a line       */
       float xo[SIZE],        /*  x-coord of the origin node           */
       float yo[SIZE],        /*  y-coord of the origin node           */
       float xd[SIZE],        /*  x-coord of the destination node      */
       float yd[SIZE],        /*  y-coord of the destination node      */
       int *lmax,             /*  the highest level in the tree        */
       int lev[SIZE],         /*  the level of a line                  */
       float a[SIZE][SIZE]    /*  the adjacency matrix                 */
)
{
/*  local variables                                                    */
     static int nincl[SIZE];  /*  flag ( 0 = redundant node )          */
     static int lincl[SIZE];  /*  flag ( 0 = redundant line )          */
       int i, j, k, m;        /*  counters                             */


   for (i = 1; i <= *n; ++i)
      if (sink[i] != 0)
         nincl[i] = 1;
      else
         nincl[i] = 0;
   for (i = 1; i <= *l; ++i)
   lincl[i] = 0;
for (k = *lmax; k >= 1; --k)
   for (i = 1; i <= *l; ++i)
      if (lev[i] == k)
         if (sink[ito[i]] != 0)
            lincl[i] = 1;
         else
            for (j = 1; j <= *l; ++j)
               if ((ifr[j] == ito[i]) && (lincl[j] != 0)) {
                  lincl[i] = 1;
                  nincl[ito[i]] = 1;
                  break;
               }
/*    remove all redundant rows    */
k = 1;
for (i = 1; i <= *n; ++i)
   if (nincl[i] != 0) {
      for (j = 1; j <= *n; ++j)
         a[k][j] = a[i][j];
      ++k;
   }
m = --k;
/*    remove all redundant columns    */
k = 1;
for (j = 1; j <= *n; ++j)
   if (nincl[j] != 0) {
      for (i = 1; i <= m; ++i)
         a[i][k] = a[i][j];
      ++k;
   }
/*    remove all redundant nodes        */
j = 1;
for (i = 1; i <= *n; ++i)
   if (nincl[i] != 0) {
      x[j] = x[i];
      y[j] = y[i];
      sink[j] = sink[i];
      dist[j] = dist[i];
      j++;
   }
*n = --j;
/*    remove all redundant lines        */
j = 1;
for (i =1; i <= *l; ++i)
   if (lincl[i] != 0) {
      ifr[j] = ifr[i];
      ito[j] = ito[i];
      xo[j] = xo[i];
      yo[j] = yo[i];
      xd[j] = xd[i];
      yd[j] = yd[i];
      lev[j] = lev[i];
```

```
         j++;
      }
   *l = --j;
   *lmax = 0;
   for (i = 1; i <= *l; ++i)
      if (lev[i] > *lmax)
         *lmax = lev[i];
   return;
}
```

```
/*********************************************************************/
/*                                                                   */
/*   subroutine rindex                                               */
/*                                                                   */
/*       This subroutine computes the efficiency of the              */
/*       Rectilinear Steiner Tree using Chung and Hwang's ratio.     */
/*       The true length of each line is computed as well.           */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

/*   rindex(n, x, y, sink, l, xo, yo, xd, yd, alen, &rpi);           */

#include <math.h>
#include "size.c"

void rindex(int, float[SIZE], float[SIZE], int[SIZE], int,
     float[SIZE], float[SIZE], float[SIZE], float[SIZE],
     float[SIZE], float*);

void rindex
(
/*  input paramters                                                  */
       int n,                /*  the number of nodes                 */
       float x[SIZE],        /*  the x-coordinate of a node          */
       float y[SIZE],        /*  the y-coordinate of a node          */
       int sink[SIZE],       /*  flag ( 0 = dummy node )             */
       int l,                /*  the number of lines                 */
       float xo[SIZE],       /*  the x-coord of the origin node      */
       float yo[SIZE],       /*  the y-coord of the origin node      */
       float xd[SIZE],       /*  the x-coord of the destination      */
       float yd[SIZE],       /*  the y-coord of the destination      */
/*  output paramters                                                 */
       float alen[SIZE],     /*  the length of a line                */
       float *rpi            /*  the performance index               */
)
{
/*  local variables                                                  */
       float dx;             /*  the x displacement of a line        */
       float dy;             /*  the y displacement of a line        */
       float xmax;           /*  the maximum x-coordinate            */
       float xmin;           /*  the minimum x-coordinate            */
       float ymax;           /*  the maximum y-coordinate            */
       float ymin;           /*  the minimum y-coordinate            */
       float ls;             /*  the length of the tree              */
       float lr;             /*  the length of the semiperimeter     */
       float ro;             /*  the ratio for the tree              */
       float romin;          /*  the calculated ratio                */
       int num;              /*  the number of sinks                 */
       int i;                /*  a counter                           */

   ls = 0.0;
   for (i = 1; i <= l; ++i) {
      dx = fabs(xd[i] - xo[i]);
      dy = fabs(yd[i] - yo[i]);
      alen[i] = sqrt((dx * dx) + (dy * dy));
      ls = ls + alen[i];
   }
   xmax = x[1];
   xmin = x[1];
   ymax = y[1];
   ymin = y[1];
   num = 1;
   for (i = 2; i <= n; ++i) {
      if (x[i] > xmax)
         xmax = x[i];
      if (x[i] < xmin)
         xmin = x[i];
      if (y[i] > ymax)
         ymax = y[i];
      if (y[i] < ymin)
         ymin = y[i];
      if (sink[i] == 1)
         ++num;
   }
   lr = (xmax - xmin) + (ymax - ymin);
   ro = ls / lr;
   romin = (sqrt(num) + 1.0) / 2.0;
   *rpi = (romin / ro) * 100.0;
   return;
}
```

```
/********************************************************************/
/*                                                                  */
/*   subroutine degree                                              */
/*                                                                  */
/*       This subroutine find the degree or the number of lines     */
/*       incident on a node.                                        */
/*                                                                  */
/*                                                                  */
/********************************************************************/

/*   degree(n, l, ifr, deg);                                        */

#include "size.c"

void degree(int, int, int[SIZE], int[SIZE]);

void degree
(                                                                   */
/*  input parameters                                                */
       int n,                  /*  the number of nodes              */
       int l,                  /*  the number of lines              */
       int ifr[SIZE],          /*  the origin node of a line        */
/*  output parameters                                               */
       int deg[SIZE]           /*  the degree of a node             */
)
{                                                                   */
/*  local variables
       int i, j;               /*  counters                         */

   deg[1] = 0;
   for (j = 1; j <= l; ++j)
      if (ifr[j] == 1)
  ++deg[1];
   for (i = 2; i <= n; ++i) {
      deg[i] = 1;
      for (j = 1; j <= l; ++j)
  if (ifr[j] == i)
     ++deg[i];
   }
   return;
}
```

```
/***********************************************************************/
/*                                                                     */
/*   subroutine rbwrit                                                 */
/*                                                                     */
/*       This subroutine creates the file "wm.dat" that the rule-      */
/*       based program uses as input.                                  */
/*                                                                     */
/*                                                                     */
/***********************************************************************/
/*   rbwrit(n, x, y, sink, deg, l, ifr, ito);                          */

#include <stdio.h>
#include "size.c"

void rbwrit(int, float[SIZE], float[SIZE], int[SIZE], int[SIZE],
     int, int[SIZE], int[SIZE]);

void rbwrit
(
/*  input parameters                                                   */
       int n,                  /*  the number of nodes                 */
       float x[SIZE],          /*  the x-coordinate of a node          */
       float y[SIZE],          /*  the y-coordinate of a node          */
       int sink[SIZE],         /*  flag ( 0 = dummy node )             */
       int deg[SIZE],          /*  the degree of a node                */
       int l,                  /*  the number of lines                 */
       int ifr[SIZE],          /*  the origin node of a line           */
       int ito[SIZE]           /*  the destination node of a line      */
)
{
/*  local variables                                                    */
       int i;                  /*  a counter                           */
       FILE *sysout;           /*  output stream                       */

   sysout = fopen("wm.dat", "w");
   fprintf(sysout, "%d\n", n);
   for (i = 1; i <= n; ++i) {
      fprintf(sysout, "%d\n", i);
      fprintf(sysout, "%.1f\n", x[i]);
      fprintf(sysout, "%.1f\n", y[i]);
      fprintf(sysout, "%d\n", sink[i]);
      fprintf(sysout, "%d\n", deg[i]);
   }
   fprintf(sysout, "0\n0.0\n0.0\n0\n0\n");
   for (i = 1; i <= l; ++i) {
      fprintf(sysout, "%d\n", ifr[i]);
      fprintf(sysout, "%d\n", ito[i]);
   }
   fprintf(sysout, "0\n0\n");
   fclose(sysout);
   return;
}
```

```
/***********************************************************************/
/*                                                                     */
/*   subroutine nwrit                                                  */
/*                                                                     */
/*       This subroutine writes the coordinates of nodes in the        */
/*       nodes.lsp file.                                               */
/*                                                                     */
/*                                                                     */
/***********************************************************************/

/*   nwrit(n, x, y, sink);                                             */

#include <stdio.h>
#include "size.c"

void nwrit(int, float[SIZE], float[SIZE], int[SIZE]);

void nwrit
(
/*  input parameters                                                   */
       int n,              /*  the number of nodes                     */
       float x[SIZE],      /*  the x-coordinate of a node              */
       float y[SIZE],      /*  the y-coordinate of a node              */
       int sink[SIZE]      /*  flag ( 0 = dummy node                   */
)
{
/*  local variables                                                    */
       int i;              /*  a counter                               */
       FILE *sysout;       /*  output stream                           */

   sysout = fopen("nodes2.lsp", "w");
   fprintf(sysout, " (setq nodes (quote ( %4d ( \n", n);
   for (i = 1; i <= n; ++i)
      fprintf(sysout, " ( %9.1f  %9.1f  %2d ) \n", x[i], y[i], sink[i]);
   fprintf(sysout, " )))) \n");
   fclose(sysout);
   return;
}
```

```
/**********************************************************************/
/*                                                                    */
/*   subroutine lwrit2                                                */
/*                                                                    */
/*       This subroutine writes the coordinates of the lines          */
/*       in the lines.lsp file.                                       */
/*                                                                    */
/*                                                                    */
/**********************************************************************/

/*   lwrit2(l, xo, yo, xd, yd, rpi);                                  */

#include <stdio.h>
#include "size.c"

void lwrit2(int, float[SIZE], float[SIZE], float[SIZE], float[SIZE],
     float);

void lwrit2
(
/*  input parameters                                                  */
       int l,                /*  the number of lines                  */
       float xo[SIZE],       /*  x-coord of origin node               */
       float yo[SIZE],       /*  y-coord of origin node               */
       float xd[SIZE],       /*  x-coord of destination node          */
       float yd[SIZE],       /*  y-coord of destination node          */
       float rpi             /*  the performance index                */
)
{
/*  local variables                                                   */
       int i;                /*  a counter                            */
       FILE *sysout;         /*  output stream                        */


   sysout = fopen("lines2.lsp", "w");
   fprintf(sysout, " (setq lines (quote ( \n");
   for (i = 1; i <= l; ++i)
      fprintf(sysout, " (( %9.1f  %9.1f ) ( %9.1f  %9.1f )) \n",
       xo[i], yo[i], xd[i], yd[i]);
   fprintf(sysout, " ))) \n");
   fprintf(sysout, " (setq rpi %6.2f ) \n", rpi);
   fclose(sysout);
   return;
}
```

```
/*********************************************************************/
/*                                                                   */
/*    subroutine con2                                                */
/*                                                                   */
/*        This subroutine creates the input file for the hydraulic   */
/*    design program "submit".                                       */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

/*   con2(n, x, y, sink);                                            */

#include <stdio.h>
#include "size.c"

void con2(int, float[SIZE], float[SIZE], int[SIZE]);

void con2
(
/*  input parameters                                                 */
       int n,                 /*   the number of nodes               */
       float x[SIZE],         /*   the x-coordinate of a node        */
       float y[SIZE],         /*   the y-coordinate of a node        */
       int sink[SIZE]         /*   flag ( 0 = dummy node )           */
)
{
/*   local variables                                                 */
 int i;                       /*   counter                           */
 FILE *sysout;                /*   input stream                      */

   sysout = fopen("nodes", "w");
   fprintf(sysout, "\n%d\n", n);
   fprintf(sysout, "%f\n%f\n", x[1], y[1]);
   fprintf(sysout, "0\n0.0\n");
   fprintf(sysout, "0.0\n0.0\n1\n1\n");
   for(i = 2; i <= n; ++i) {
      fprintf(sysout, "%f\n%f\n", x[i], y[i]);
      if (sink[i] == 0)
         fprintf(sysout, "0\n0.0\n");
      else
          fprintf(sysout, "1\n6.0\n");
      fprintf(sysout, "0.0\n0.0\n1\n1\n");
   }
   fprintf(sysout, "nil\n");
   fclose(sysout);
   return;
}
```

APPENDIX C: SOURCE CODE FOR COGNITIVE COMPONENT

```
code = 4096
project "RSTRB2"
include "global2.pro"

database - flags
   invoke

predicates
   main1
      enter
  enterpoint(integer, real, real, integer, integer)
  enterline(integer, integer)
      main2
            loop2
             set_invoke
/*           redun
        decount
     find
        dist
        trigger                 */
           clear_invoke
              add_priority(real)
              inc_trignum
     fire
        fire1(integer)
           repeat(integer)
           new_priority(real)
               maximum(plist, real, plist)
               max1(real, plist, plist, real, plist)
           new_action(real)
/*                    execute
                incount
                decount       */
   keep
      linwrit
      pointwrit

goal
   main1.

clauses
   main1 :-
  enter,
  main2,
  write("Iterations complete\n"),
  keep.


   enter :-
      openread(sysio, "wm.dat"),
      readdevice(sysio),
      readint(C),
   assertz(count(C)),
   C2 = C,
   assertz(count2(C2)),
   readint(N),
   readreal(X),
   readreal(Y),
   readint(S),
   readint(D),
   enterpoint(N, X, Y, S, D),
   readint(F),
   readint(T),
   enterline(F, T),
   readdevice(keyboard),
   closefile(sysio),
   write("Data loaded\n"),
   assertz(priorities([])),
   assertz(trig_num(0)),
   point(1, XS, YS, _, _),
   assertz(xsource(XS)),
   assertz(ysource(YS)),
   write("System initialized\n"),
   !.

enterpoint(N, _, _, _, _) :-
   N = 0,
   !.

enterpoint(N1, X1, Y1, S1, D1) :-
   assertz(point(N1, X1, Y1, S1, D1)),
   readint(N2),
   readreal(X2),
   readreal(Y2),
   readint(S2),
   readint(D2),
   enterpoint(N2, X2, Y2, S2, D2).

enterline(F, _) :-
   F = 0,
   !.

enterline(F1, T1) :-
   assertz(lin(F1, T1)),
   readint(F2),
   readint(T2),
   enterline(F2, T2).

main2 :-
   loop2,
      set_invoke,
      redun,
      find,
      fire,
      fail.

main2.

loop2.

loop2 :-
   not(invoke),
   loop2.

set_invoke :-
   invoke,
   !.

set_invoke :-
   assertz(invoke).

decount :-
   count(C),
   D = C - 1,
   retract(count(C)),
   assertz(count(D)),
   !.

dist(X, Y, Priority) :-
   xsource(XS),
   ysource(YS),
   DX = X - XS,
   DY = Y - YS,
   DXa = abs(DX),
   DYa = abs(DY),
   DX2 = DXa * DXa,
   DY2 = DYa * DYa,
   SUM = DX2 + DY2,
   D = sqrt(SUM),
   Priority = 0.0 - D.

trigger(Priority, Action, Params) :-
   clear_invoke,
   add_priority(Priority),
   inc_trignum,
   assertz(trig(Priority, Action, Params)).

clear_invoke :-
   invoke,
   retract(invoke),
   !.

clear_invoke :-
   not(invoke).

add_priority(P) :-
```

```
   priorities(Plst),
   retract(priorities(Plst)),
   assertz(priorities([P | Plst])),
   !.

inc_trignum :-
   trig_num(N),
   Nnew = N + 1,
   retract(trig_num(N)),
   assertz(trig_num(Nnew)),
   !.

fire :-
   trig_num(N),
   fire1(N),
   retract(trig_num(N)),
   assertz(trig_num(0)),
   priorities(P),
   retract(priorities(P)),
   assertz(priorities([])),
   !.

fire1(0) :-
   write("No rules triggered\n"),
   !.

fire1(N) :-
   repeat(N),
   not(priorities([])),
   new_priority(P),
   new_action(P),
   fail.

fire1(_) :-
   write("All rules exhausted\n").

repeat(_).

repeat(N) :-
   not(N = 0),
   Nnew = N - 1,
   repeat(Nnew).

new_priority(P) :-
   priorities(Plst),
   maximum(Plst, P, Plst2),
   retract(priorities(Plst)),
   assertz(priorities(Plst2)),
   !.

maximum([X | L1], Max, L2) :-
   max1(X, [], L1, Max, L2).
```

```
max1(X, L1, [], X, L1) :-
   !.

max1(X, L1, [X2 | L2], Max, L3) :-
   X2 > X,
   max1(X2, [X | L1], L2, Max, L3),
   !.

max1(X, L1, [X2 | L2], Max, L3) :-
   max1(X, [X2 | L1], L2, Max, L3).

new_action(P) :-
   trig(P, Action, Plst),
   retract(trig(P, Action, Plst)),
   execute(Action, Plst),
   !.

incount(Dnew) :-
   count(C),
   Cnew = C + 1,
   retract(count(C)),
   assertz(count(Cnew)),
   count2(D),
   Dnew = D + 1,
   retract(count2(D)),
   assertz(count2(Dnew)),
   !.

keep :-
   openwrite(sysio, "lines3.lsp"),
   writedevice(sysio),
   write("(setq lines (quote ("),
   nl,
   linwrit,
   write(")))"),
   nl,
   writedevice(screen),
   closefile(sysio),
   openwrite(sysio, "nodes3.lsp"),
   writedevice(sysio),
   count(C),
   writef(" (setq nodes (quote ( %4d ( \n", C),
   retract(count(C)),
   retract(count2(_)),
   point(1, X, Y, S, _),
   retract(point(1, X, Y, S, _)),
   writef(" ( %9.1f  %9.1f  %2d ) \n", X, Y, S),
   pointwrit,
   writef(" )))) \n"),
   writedevice(screen),
   closefile(sysio),
   write("Data written\n").

linwrit :-
   lin(F, T),
   point(F, X1, Y1, _, _),
   point(T, X2, Y2, _, _),
   writef("(( %9.1f %9.1f) ( %9.1f %9.1f))", X1, Y1, X2, Y2),
   nl,
   retract(lin(F, T)),
   fail.

linwrit.

pointwrit :-
   point(_, X, Y, S, _),
   retract(point(_, X, Y, S, _)),
   writef(" ( %9.1f  %9.1f  %2d ) \n", X, Y, S),
   fail.

pointwrit.
```

```
/* The following should be included in a separate file called
   "global2.pro" */

global domains
   file = sysio
   paramlst = param*
   param = i(integer) ; r(real)
   plist = real*

global database
   count(integer)
   count2(integer)
   point(integer, real, real, integer, integer)
   lin(integer, integer)
   xsource(real)
   ysource(real)
   trig_num(integer)
   priorities(plist)
   trig(real, symbol, paramlst)

global predicates
   nondeterm redun
   nondeterm decount
   nondeterm find
   nondeterm dist(real, real, real) - (i,i,o)
   nondeterm slide(integer,
            integer, real, real,integer, integer,
            integer) - (i,i,i,i,i,i,i)
   nondeterm trigger(real, symbol, paramlst) - (i,i,i)
   nondeterm execute(symbol, paramlst) - (i, i)
   nondeterm incount(integer) - (o)
```

```
code = 4096
project "RSTRB2"
include "global2.pro"

predicates
   rfix(integer, integer, integer, real, real)

clauses
   redun :-
      point(P2, Xm, Ym, 0, 2),
      lin(P1, P2),
      lin(P2, P3),
      rfix(P1, P2, P3, Xm, Ym).

   redun :-
      write("Search terminated\n"),
      !.

   rfix(P1, P2, P3, X1, Y2) :-
      point(P1, X1, _, _, _),
      point(P3, X1, _, _, _),
      retract(point(P2, X1, Y2, 0, 2)),
      decount,
      retract(lin(P1, P2)),
      retract(lin(P2, P3)),
      assertz(lin(P1, P3)),
      write("1 redundant node removed\n"),
      fail.

   rfix(P1, P2, P3, X2, Y1) :-
      point(P1, _, Y1, _, _),
      point(P3, _, Y1, _, _),
      retract(point(P2, X2, Y1, 0, 2)),
      decount,
      retract(lin(P1, P2)),
      retract(lin(P2, P3)),
      assertz(lin(P1, P3)),
      write("1 redundant node removed\n"),
      fail.
```

```
code = 4096
project "RSTRB2"
include "global2.pro"

predicates
   elbow(integer,
         integer, real, real,
         integer)
   horiz(integer, real, real, integer, integer,
         integer,
         integer, real, real, integer, integer)
   i1h(integer, real, real, integer, integer,
       integer,
       integer, real, real, integer, integer,
       integer,       real, integer, integer)
   o1h(integer, real, real, integer, integer,
       integer,
       integer, real, real, integer, integer,
       integer,       real, integer, integer)
   o3h(integer, real, real, integer, integer,
       integer,
       integer, real, real, integer, integer,
       integer, real,        integer, integer)
   vert(integer, real, real, integer, integer,
        integer,
        integer, real, real, integer, integer)
   i1v(integer, real, real, integer, integer,
       integer,
       integer, real, real, integer, integer,
       integer, real,        integer, integer)
   o1v(integer, real, real, integer, integer,
       integer,
       integer, real, real, integer, integer,
       integer, real,        integer, integer)
   o3v(integer, real, real, integer, integer,
       integer,
       integer, real, real, integer, integer,
       integer,       real, integer, integer)


clauses
   find :-
      point(P2, X2, Y2, 0, 2),
      lin(P1, P2),
      lin(P2, P3),
      elbow(P1,
            P2, X2, Y2,
            P3).

   find :-
      point(P2, X2, Y2, S2, D2),
      lin(P1, P2),
   lin(P2, P3),
   slide(P1,
         P2, X2, Y2, S2, D2,
         P3).

find :-
   write("Search terminated\n"),
   !.

elbow(P1,
      P2, X3, Y1,
      P3) :-
   point(P1, X1, Y1, S1, D1),
   point(P3, X3, Y3, S3, D3),
   horiz(P1, X1, Y1, S1, D1,
         P2,
         P3, X3, Y3, S3, D3).

elbow(P1,
      P2, X1, Y3,
      P3) :-
   point(P1, X1, Y1, S1, D1),
   point(P3, X3, Y3, S3, D3),
   vert(P1, X1, Y1, S1, D1,
        P2,
        P3, X3, Y3, S3, D3).

horiz(P1, X1, Y1, S1, D1,
      P2,
      P3, X3, Y3, S3, D3) :-
   lin(P4, P1),
   point(P4, X1, Y4, S4, D4),
   i1h(P1, X1, Y1, S1, D1,
       P2,
       P3, X3, Y3, S3, D3,
       P4,     Y4, S4, D4).

horiz(P1, X1, Y1, S1, D1,
      P2,
      P3, X3, Y3, S3, D3) :-
   lin(P1, P4),
   point(P4, X1, Y4, S4, D4),
   o1h(P1, X1, Y1, S1, D1,
       P2,
       P3, X3, Y3, S3, D3,
       P4,     Y4, S4, D4).

horiz(P1, X1, Y1, S1, D1,
      P2,
      P3, X3, Y3, S3, D3) :-
   lin(P3, P4),
   point(P4, X4, Y3, S4, D4),
   o3h(P1, X1, Y1, S1, D1,
       P2,
       P3, X3, Y3, S3, D3,
       P4, X4,     S4, D4).

ilh(P1, X1, Y1, S1, D1,
    P2,
    _,  X3, Y3, _,  _,
    P4,     Y4, _,  _) :-
   Y4 > Y3, Y3 > Y1,
   Priority = Y3 - Y1,
   trigger(Priority, rgil, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                            i(P2), r(X3), r(Y1), r(X1), r(Y3),
                            i(P4)]),
   write("1 gilh triggered\n"),
   fail.

ilh(P1, X1, Y1, S1, D1,
    P2,
    _,  X3, Y3, _,  _,
    P4,     Y4, _,  _) :-
   Y4 < Y3, Y3 < Y1,
   Priority = Y1 - Y3,
   trigger(Priority, rgil, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                            i(P2), r(X3), r(Y1), r(X1), r(Y3),
                            i(P4)]),
   write("1 gilh triggered\n"),
   fail.

ilh(P1, X1, Y1, S1, D1,
    P2,
    P3, X3, Y3, _,  _,
    P4,     Y4, S4, D4) :-
   Y4 = Y3, Y3 > Y1,
   Priority = Y3 - Y1,
   trigger(Priority, rel, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                           i(P2), r(X3), r(Y1),
                           i(P3),
                           i(P4), r(X1), r(Y4), i(S4), i(D4)]),
   write("1 eilh triggered\n"),
   fail.

ilh(P1, X1, Y1, S1, D1,
    P2,
    P3, X3, Y3, _,  _,
    P4,     Y4, S4, D4) :-
   Y4 = Y3, Y3 < Y1,
   Priority = Y1 - Y3,
   trigger(Priority, rel, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                           i(P2), r(X3), r(Y1),
                           i(P3),
                           i(P4), r(X1), r(Y4), i(S4), i(D4)]),
   write("1 eilh triggered"),
   fail.

ilh(P1, X1, Y1, S1, D1,
    P2,
    _,  X3, Y3, _,  _,
    P4,     Y4, S4, D4) :-
   Y3 > Y4, Y4 > Y1,
   Priority = Y4 - Y1,
   trigger(Priority, rl1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                           i(P2), r(X3), r(Y1), r(X3), r(Y4),
                           i(P4), r(X1), r(Y4), i(S4), i(D4)]),
   write("1 lilh triggered\n"),
   fail.

ilh(P1, X1, Y1, S1, D1,
    P2,
    _,  X3, Y3, _,  _,
    P4,     Y4, S4, D4) :-
   Y3 < Y4, Y4 < Y1,
   Priority = Y1 - Y4,
   trigger(Priority, rl1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                           i(P2), r(X3), r(Y1), r(X3), r(Y4),
                           i(P4), r(X1), r(Y4), i(S4), i(D4)]),
   write("1 lilh triggered\n"),
   fail.

ilh(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 1,
    P4,     Y4, S4, D4) :-
   Y4 > Y1, Y1 > Y3,
   dist(X1, Y3, Priority),
   trigger(Priority, rei1, [i(P1), r(X1), r(Y1), i(S1), i(2),
                            i(P2), r(X3), r(Y1), r(X1), r(Y3),
                            i(P3), r(X3), r(Y3), i(S3), i(1),
                            i(P4), r(X1), r(Y4), i(S4), i(D4)]),
   write("1 eli1h triggered\n"),
   fail.

ilh(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 1,
    P4,     Y4, S4, D4) :-
   Y4 < Y1, Y1 < Y3,
   dist(X1, Y3, Priority),
   trigger(Priority, rei1, [i(P1), r(X1), r(Y1), i(S1), i(2),
                            i(P2), r(X3), r(Y1), r(X1), r(Y3),
                            i(P3), r(X3), r(Y3), i(S3), i(1),
                            i(P4), r(X1), r(Y4), i(S4), i(D4)]),
   write("1 eli1h triggered\n"),
   fail.
```

```
i1h(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 3,
    P4,     Y4, S4, D4) :-
   Y4 > Y1, Y1 > Y3,
   dist(X1, Y3, Priority),
   trigger(Priority, rei1, [i(P1), r(X1), r(Y1), i(S1), i(2),
                            i(P2), r(X3), r(Y1), r(X1), r(Y3),
                            i(P3), r(X3), r(Y3), i(S3), i(3),
                            i(P4), r(X1), r(Y4), i(S4), i(D4)]),
   write("1 e3i1h triggered\n"),
   fail.

i1h(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 3,
    P4,     Y4, S4, D4) :-
   Y4 < Y1, Y1 < Y3,
   dist(X1, Y3, Priority),
   trigger(Priority, rei1, [i(P1), r(X1), r(Y1), i(S1), i(2),
                            i(P2), r(X3), r(Y1), r(X1), r(Y3),
                            i(P3), r(X3), r(Y3), i(S3), i(3),
                            i(P4), r(X1), r(Y4), i(S4), i(D4)]),
   write("1 e3i1h triggered\n"),
   fail.

i1h(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 2,
    P4,     Y4, S4, D4) :-
   lin(P3, P5),
   point(P5, X5, Y3, S5, D5),
   X1 > X3, X3 > X5,
   Y4 > Y1, Y1 > Y3,
   dist(X1, Y3, Priority),
   trigger(Priority, resi1, [i(P1), r(X1), r(Y1), i(S1), i(2),
                             i(P2), r(X3), r(Y1), r(X1), r(Y3),
                             i(P3), r(X3), r(Y3), i(S3), i(2),
                             i(P4), r(X1), r(Y4), i(S4), i(D4),
                             i(P5), r(X5), r(Y3), i(S5), i(D5)]),
   write("1 esi1h triggered\n"),
   fail.

i1h(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 2,
    P4,     Y4, S4, D4) :-
   lin(P3, P5),
   point(P5, X5, Y3, S5, D5),
   X1 > X3, X3 > X5,
   Y4 < Y1, Y1 < Y3,
   dist(X1, Y3, Priority),
   trigger(Priority, resil, [i(P1), r(X1), r(Y1), i(S1), i(2),
                            i(P2), r(X3), r(Y1), r(X1), r(Y3),
                            i(P3), r(X3), r(Y3), i(S3), i(2),
                            i(P4), r(X1), r(Y4), i(S4), i(D4),
                            i(P5), r(X5), r(Y3), i(S5), i(D5)]),
   write("1 esilh triggered\n"),
   fail.

ilh(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 2,
    P4,     Y4, S4, D4) :-
   lin(P3, P5),
   point(P5, X5, Y3, S5, D5),
   X1 < X3, X3 < X5,
   Y4 > Y1, Y1 > Y3,
   dist(X1, Y3, Priority),
   trigger(Priority, resil, [i(P1), r(X1), r(Y1), i(S1), i(2),
                            i(P2), r(X3), r(Y1), r(X1), r(Y3),
                            i(P3), r(X3), r(Y3), i(S3), i(2),
                            i(P4), r(X1), r(Y4), i(S4), i(D4),
                            i(P5), r(X5), r(Y3), i(S5), i(D5)]),
   write("1 esilh triggered\n"),
   fail.

ilh(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 2,
    P4,     Y4, S4, D4) :-
   lin(P3, P5),
   point(P5, X5, Y3, S5, D5),
   X1 < X3, X3 < X5,
   Y4 < Y1, Y1 < Y3,
   dist(X1, Y3, Priority),
   trigger(Priority, resil, [i(P1), r(X1), r(Y1), i(S1), i(2),
                            i(P2), r(X3), r(Y1), r(X1), r(Y3),
                            i(P3), r(X3), r(Y3), i(S3), i(2),
                            i(P4), r(X1), r(Y4), i(S4), i(D4),
                            i(P5), r(X5), r(Y3), i(S5), i(D5)]),
   write("1 esilh triggered\n"),
   fail.

olh(P1, X1, Y1, S1, D1,
    P2,
    _,  X3, Y3, _,  _,
    P4,     Y4, _,  _) :-
   Y4 > Y3, Y3 > Y1,
   Priority = Y3 - Y1,
   trigger(Priority, rgol, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                           i(P2), r(X3), r(Y1), r(X1), r(Y3),
                           i(P4)]),
   write("1 go1h triggered\n"),
   fail.

o1h(P1, X1, Y1, S1, D1,
    P2,
    _,  X3, Y3, _,  _,
    P4,     Y4, _,  _) :-
   Y4 < Y3, Y3 < Y1,
   Priority = Y1 - Y3,
   trigger(Priority, rgo1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                       i(P2), r(X3), r(Y1), r(X1), r(Y3),
                       i(P4)]),
   write("1 go1h triggered\n"),
   fail.

o1h(P1, X1, Y1, S1, D1,
    P2,
    P3, X3, Y3, _,  _,
    P4,     Y4, S4, D4) :-
   Y4 = Y3, Y3 > Y1,
   Priority = Y3 - Y1,
   trigger(Priority, re1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                      i(P2), r(X3), r(Y1),
                      i(P3),
                      i(P4), r(X1), r(Y4), i(S4), i(D4)]),
   write("1 eo1h triggered\n"),
   fail.

o1h(P1, X1, Y1, S1, D1,
    P2,
    P3, X3, Y3, _,  _,
    P4,     Y4, S4, D4) :-
   Y4 = Y3, Y3 < Y1,
   Priority = Y1 - Y3,
   trigger(Priority, re1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                      i(P2), r(X3), r(Y1),
                      i(P3),
                      i(P4), r(X1), r(Y4), i(S4), i(D4)]),
   write("1 eo1h triggered"),
   fail.

o1h(P1, X1, Y1, S1, D1,
    P2,
    _,  X3, Y3, _,  _,
    P4,     Y4, S4, D4) :-
   Y3 > Y4, Y4 > Y1,
   Priority = Y4 - Y1,
   trigger(Priority, rl1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                      i(P2), r(X3), r(Y1), r(X1), r(Y3),
                      i(P4), r(X1), r(Y4), i(S4), i(D4)]),
   write("1 lo1h triggered\n"),
   fail.
```

```
o1h(P1, X1, Y1, S1, D1,
    P2,
    _,  X3, Y3, _,  _,
    P4,     Y4, S4, D4) :-
   Y3 < Y4, Y4 < Y1,
   Priority = Y1 - Y4,
   trigger(Priority, rl1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                          i(P2), r(X3), r(Y1), r(X1), r(Y3),
                          i(P4), r(X1), r(Y4), i(S4), i(D4)]),
   write("1 lo1h triggered\n"),
   fail.

o3h(_,  X1, Y1, _,  _,
    P2,
    P3, X3, Y3, S3, D3,
    P4, X4,     _,  _) :-
   X4 > X1, X1 > X3,
   Priority = X1 - X3,
   trigger(Priority, rgo3, [i(P2), r(X3), r(Y1), r(X1), r(Y3),
                           i(P3), r(X3), r(Y3), i(S3), i(D3),
                           i(P4)]),
   write("1 go3h triggered\n"),
   fail.

o3h(_,  X1, Y1, _,  _,
    P2,
    P3, X3, Y3, S3, D3,
    P4, X4,     _,  _) :-
   X4 < X1, X1 < X3,
   Priority = X3 - X1,
   trigger(Priority, rgo3, [i(P2), r(X3), r(Y1), r(X1), r(Y3),
                           i(P3), r(X3), r(Y3), i(S3), i(D3),
                           i(P4)]),
   write("1 go3h triggered\n"),
   fail.

o3h(P1, X1, Y1, _,  _,
    P2,
    P3, X3, Y3, S3, D3,
    P4, X4,     S4, D4) :-
   X4 = X1, X1 > X3,
   Priority = X1 - X3,
   trigger(Priority, reo3, [i(P1),
                           i(P2), r(X3), r(Y1),
                           i(P3), r(X3), r(Y3), i(S3), i(D3),
                           i(P4), r(X4), r(Y3), i(S4), i(D4)]),
   write("1 eo3h triggered\n"),
   fail.

o3h(P1, X1, Y1, _,  _,
    P2,
     P3, X3, Y3, S3, D3,
     P4, X4,     S4, D4) :-
    X4 = X1, X1 < X3,
    Priority = X3 - X1,
    trigger(Priority, reo3, [i(P1),
                        i(P2), r(X3), r(Y1),
                        i(P3), r(X3), r(Y3), i(S3), i(D3),
                        i(P4), r(X4), r(Y3), i(S4), i(D4)]),
    write("1 eo3h triggered\n"),
    fail.

o3h(_,  X1, Y1, _,  _,
     P2,
     P3, X3, Y3, S3, D3,
     P4, X4,     S4, D4) :-
    X1 > X4, X4 > X3,
    Priority = X4 - X3,
    trigger(Priority, rlo3, [i(P2), r(X3), r(Y1), r(X1), r(Y3),
                        i(P3), r(X3), r(Y3), i(S3), i(D3),
                        i(P4), r(X4), r(Y3), i(S4), i(D4)]),
    write("1 lo3h triggered\n"),
    fail.

o3h(_,  X1, Y1, _,  _,
     P2,
     P3, X3, Y3, S3, D3,
     P4, X4,     S4, D4) :-
    X1 < X4, X4 < X3,
    Priority = X3 - X4 + 2,
    trigger(Priority, rlo3, [i(P2), r(X3), r(Y1), r(X1), r(Y3),
                        i(P3), r(X3), r(Y3), i(S3), i(D3),
                        i(P4), r(X4), r(Y3), i(S4), i(D4)]),
    write("1 lo3h triggered\n"),
    fail.

o3h(P1, X1, Y1, S1, 1,
     P2,
     P3, X3, Y3, S3, 2,
     P4, X4,     S4, D4) :-
    X1 > X3, X3 > X4,
    dist(X1, Y3, Priority),
    trigger(Priority, reo3, [i(P1), r(X1), r(Y1), i(S1), i(1),
                        i(P2), r(X3), r(Y1), r(X1), r(Y3),
                        i(P3), r(X3), r(Y3), i(S3), i(2),
                        i(P4), r(X4), r(Y3), i(S4), i(D4)]),
    write("1 elo3h triggered\n"),
    fail.

o3h(P1, X1, Y1, S1, 1,
     P2,
     P3, X3, Y3, S3, 2,
     P4, X4,     S4, D4) :-
```

```
   X1 < X3, X3 < X4,
   dist(X1, Y3, Priority),
   trigger(Priority, reo3, [i(P1), r(X1), r(Y1), i(S1), i(1),
                            i(P2), r(X3), r(Y1), r(X1), r(Y3),
                            i(P3), r(X3), r(Y3), i(S3), i(2),
                            i(P4), r(X4), r(Y3), i(S4), i(D4)]),
   write("1 e1o3h triggered\n"),
   fail.

o3h(P1, X1, Y1, S1, 3,
    P2,
    P3, X3, Y3, S3, 2,
    P4, X4,      S4, D4) :-
   X1 > X3, X3 > X4,
   dist(X1, Y3, Priority),
   trigger(Priority, reo3, [i(P1), r(X1), r(Y1), i(S1), i(3),
                            i(P2), r(X3), r(Y1), r(X1), r(Y3),
                            i(P3), r(X3), r(Y3), i(S3), i(2),
                            i(P4), r(X4), r(Y3), i(S4), i(D4)]),
   write("1 e3o3h triggered\n"),
   fail.

o3h(P1, X1, Y1, S1, 3,
    P2,
    P3, X3, Y3, S3, 2,
    P4, X4,      S4, D4) :-
   X1 < X3, X3 < X4,
   dist(X1, Y3, Priority),
   trigger(Priority, reo3, [i(P1), r(X1), r(Y1), i(S1), i(3),
                            i(P2), r(X3), r(Y1), r(X1), r(Y3),
                            i(P3), r(X3), r(Y3), i(S3), i(2),
                            i(P4), r(X4), r(Y3), i(S4), i(D4)]),
   write("1 e3o3h triggered\n"),
   fail.

vert(P1, X1, Y1, S1, D1,
     P2,
     P3, X3, Y3, S3, D3) :-
   lin(P4, P1),
   point(P4, X4, Y1, S4, D4),
   i1v(P1, X1, Y1, S1, D1,
       P2,
       P3, X3, Y3, S3, D3,
       P4, X4,     S4, D4).

vert(P1, X1, Y1, S1, D1,
     P2,
     P3, X3, Y3, S3, D3) :-
   lin(P1, P4),
   point(P4, X4, Y1, S4, D4),
   o1v(P1, X1, Y1, S1, D1,
       P2,
        P3, X3, Y3, S3, D3,
        P4, X4,     S4, D4).

vert(P1, X1, Y1, S1, D1,
     P2,
     P3, X3, Y3, S3, D3) :-
   lin(P3, P4),
   point(P4, X3, Y4, S4, D4),
   o3v(P1, X1, Y1, S1, D1,
       P2,
       P3, X3, Y3, S3, D3,
       P4,     Y4, S4, D4).

i1v(P1, X1, Y1, S1, D1,
    P2,
    _,  X3, Y3, _,  _,
    P4, X4,     _,  _) :-
   X4 > X3, X3 > X1,
   Priority = X3 - X1,
   trigger(Priority, rgi1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                            i(P2), r(X1), r(Y3), r(X3), r(Y1),
                            i(P4)]),
   write("1 gi1v triggered\n"),
   fail.

i1v(P1, X1, Y1, S1, D1,
    P2,
    _,  X3, Y3, _,  _,
    P4, X4,     _,  _) :-
   X4 < X3, X3 < X1,
   Priority = X1 - X3,
   trigger(Priority, rgi1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                            i(P2), r(X1), r(Y3), r(X3), r(Y1),
                            i(P4)]),
   write("1 gi1v triggered\n"),
   fail.

i1v(P1, X1, Y1, S1, D1,
    P2,
    P3, X3, Y3, _,  _,
    P4, X4,     S4, D4) :-
   X4 = X3, X3 > X1,
   Priority = X3 - X1,
   trigger(Priority, re1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                           i(P2), r(X1), r(Y3),
                           i(P3),
                           i(P4), r(X4), r(Y1), i(S4), i(D4)]),
   write("1 ei1v triggered\n"),
   fail.

i1v(P1, X1, Y1, S1, D1,
    P2,
     P3, X3, Y3, _, _,
     P4, X4,      S4, D4) :-
    X4 = X3, X3 < X1,
    Priority = X1 - X3,
    trigger(Priority, rel, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                            i(P2), r(X1), r(Y3),
                            i(P3),
                            i(P4), r(X4), r(Y1), i(S4), i(D4)]),
    write("1 eilv triggered\n"),
    fail.

i1v(P1, X1, Y1, S1, D1,
     P2,
     _, X3, Y3, _, _,
     P4, X4,      S4, D4) :-
    X3 > X4, X4 > X1,
    Priority = X4 - X1,
    trigger(Priority, rl1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                            i(P2), r(X1), r(Y3), r(X4), r(Y3),
                            i(P4), r(X4), r(Y1), i(S4), i(D4)]),
    write("1 lilv triggered\n"),
    fail.

i1v(P1, X1, Y1, S1, D1,
     P2,
     _, X3, Y3, _, _,
     P4, X4,      S4, D4) :-
    X3 < X4, X4 < X1,
    Priority = X1 - X4,
    trigger(Priority, rl1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                            i(P2), r(X1), r(Y3), r(X4), r(Y3),
                            i(P4), r(X4), r(Y1), i(S4), i(D4)]),
    write("1 lilv triggered\n"),
    fail.

i1v(P1, X1, Y1, S1, 2,
     P2,
     P3, X3, Y3, S3, 1,
     P4, X4,      S4, D4) :-
    X4 > X1, X1 > X3,
    dist(X3, Y1, Priority),
    trigger(Priority, rei1, [i(P1), r(X1), r(Y1), i(S1), i(2),
                             i(P2), r(X1), r(Y3), r(X3), r(Y1),
                             i(P3), r(X3), r(Y3), i(S3), i(1),
                             i(P4), r(X4), r(Y1), i(S4), i(D4)]),
    write("1 elilv triggered\n"),
    fail.

i1v(P1, X1, Y1, S1, 2,
     P2,
     P3, X3, Y3, S3, 1,
     P4, X4,      S4, D4) :-
```

```
   X4 < X1, X1 < X3,
   dist(X3, Y1, Priority),
   trigger(Priority, reil, [i(P1), r(X1), r(Y1), i(S1), i(2),
                            i(P2), r(X1), r(Y3), r(X3), r(Y1),
                            i(P3), r(X3), r(Y3), i(S3), i(1),
                            i(P4), r(X4), r(Y1), i(S4), i(D4)]),
   write("1 e1i1v triggered\n"),
   fail.

i1v(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 3,
    P4, X4,     S4, D4) :-
   X4 > X1, X1 > X3,
   dist(X3, Y1, Priority),
   trigger(Priority, reil, [i(P1), r(X1), r(Y1), i(S1), i(2),
                            i(P2), r(X1), r(Y3), r(X3), r(Y1),
                            i(P3), r(X3), r(Y3), i(S3), i(3),
                            i(P4), r(X4), r(Y1), i(S4), i(D4)]),
   write("1 e3i1v triggered\n"),
   fail.

i1v(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 3,
    P4, X4,     S4, D4) :-
   X4 < X1, X1 < X3,
   dist(X3, Y1, Priority),
   trigger(Priority, reil, [i(P1), r(X1), r(Y1), i(S1), i(2),
                            i(P2), r(X1), r(Y3), r(X3), r(Y1),
                            i(P3), r(X3), r(Y3), i(S3), i(3),
                            i(P4), r(X4), r(Y1), i(S4), i(D4)]),
   write("1 e3i1v triggered\n"),
   fail.

i1v(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 2,
    P4, X4,     S4, D4) :-
   lin(P3, P5),
   point(P5, X3, Y5, S5, D5),
   X4 > X1, X1 > X3,
   Y1 > Y3, Y3 > Y5,
   dist(X3, Y1, Priority),
   trigger(Priority, resil, [i(P1), r(X1), r(Y1), i(S1), i(2),
                             i(P2), r(X1), r(Y3), r(X3), r(Y1),
                             i(P3), r(X3), r(Y3), i(S3), i(2),
                             i(P4), r(X4), r(Y1), i(S4), i(D4),
                             i(P5), r(X3), r(Y5), i(S5), i(D5)]),
   write("1 esi1v triggered\n"),
   fail.
```

```
i1v(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 2,
    P4, X4,       S4, D4) :-
   lin(P3, P5),
   point(P5, X3, Y5, S5, D5),
   X4 > X1, X1 > X3,
   Y1 < Y3, Y3 < Y5,
   dist(X3, Y1, Priority),
   trigger(Priority, resi1, [i(P1), r(X1), r(Y1), i(S1), i(2),
                        i(P2), r(X1), r(Y3), r(X3), r(Y1),
                        i(P3), r(X3), r(Y3), i(S3), i(2),
                        i(P4), r(X4), r(Y1), i(S4), i(D4),
                        i(P5), r(X3), r(Y5), i(S5), i(D5)]),
   write("1 esi1v triggered\n"),
   fail.

i1v(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 2,
    P4, X4,       S4, D4) :-
   lin(P3, P5),
   point(P5, X3, Y5, S5, D5),
   X4 < X1, X1 < X3,
   Y1 > Y3, Y3 > Y5,
   dist(X3, Y1, Priority),
   trigger(Priority, resi1, [i(P1), r(X1), r(Y1), i(S1), i(2),
                        i(P2), r(X1), r(Y3), r(X3), r(Y1),
                        i(P3), r(X3), r(Y3), i(S3), i(2),
                        i(P4), r(X4), r(Y1), i(S4), i(D4),
                        i(P5), r(X3), r(Y5), i(S5), i(D5)]),
   write("1 esi1v triggered\n"),
   fail.

i1v(P1, X1, Y1, S1, 2,
    P2,
    P3, X3, Y3, S3, 2,
    P4, X4,       S4, D4) :-
   lin(P3, P5),
   point(P5, X3, Y5, S5, D5),
   X4 < X1, X1 < X3,
   Y1 < Y3, Y3 < Y5,
   dist(X3, Y1, Priority),
   trigger(Priority, resi1, [i(P1), r(X1), r(Y1), i(S1), i(2),
                        i(P2), r(X1), r(Y3), r(X3), r(Y1),
                        i(P3), r(X3), r(Y3), i(S3), i(2),
                        i(P4), r(X4), r(Y1), i(S4), i(D4),
                        i(P5), r(X3), r(Y5), i(S5), i(D5)]),
   write("1 esi1v triggered\n"),
   fail.

o1v(P1, X1, Y1, S1, D1,
     P2,
     _,  X3, Y3, _,  _,
     P4, X4,         _,  _) :-
   X4 > X3, X3 > X1,
   Priority = X3 - X1,
   trigger(Priority, rgo1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                       i(P2), r(X1), r(Y3), r(X3), r(Y1),
                       i(P4)]),
   write("1 go1v triggered\n"),
   fail.

o1v(P1, X1, Y1, S1, D1,
     P2,
     _,  X3, Y3, _,  _,
     P4, X4,         _,  _) :-
   X4 < X3, X3 < X1,
   Priority = X1 - X3,
   trigger(Priority, rgo1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                       i(P2), r(X1), r(Y3), r(X3), r(Y1),
                       i(P4)]),
   write("1 go1v triggered\n"),
   fail.

o1v(P1, X1, Y1, S1, D1,
     P2,
     P3, X3, Y3, _,  _,
     P4, X4,         S4, D4) :-
   X4 = X3, X3 > X1,
   Priority = X3 - X1,
   trigger(Priority, re1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                       i(P2), r(X1), r(Y3),
                       i(P3),
                       i(P4), r(X4), r(Y1), i(S4), i(D4)]),
   write("1 eo1v triggered\n"),
   fail.

o1v(P1, X1, Y1, S1, D1,
     P2,
     P3, X3, Y3, _,  _,
     P4, X4,         S4, D4) :-
   X4 = X3, X3 < X1,
   Priority = X1 - X3,
   trigger(Priority, re1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                       i(P2), r(X1), r(Y3),
                       i(P3),
                       i(P4), r(X4), r(Y1), i(S4), i(D4)]),
   write("1 eo1v triggered\n"),
   fail.

o1v(P1, X1, Y1, S1, D1,
     P2,
     _,  X3, Y3, _,  _,
     P4, X4,       S4, D4) :-
    X3 > X4, X4 > X1,
    Priority = X4 - X1,
    trigger(Priority, rl1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                           i(P2), r(X1), r(Y3), r(X3), r(Y1),
                           i(P4), r(X4), r(Y1), i(S4), i(D4)]),
    write("1 lo1v triggered\n"),
    fail.

o1v(P1, X1, Y1, S1, D1,
     P2,
     _, X3, Y3, _, _,
     P4, X4,       S4, D4) :-
    X3 < X4, X4 < X1,
    Priority = X1 - X4,
    trigger(Priority, rl1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                           i(P2), r(X1), r(Y3), r(X3), r(Y1),
                           i(P4), r(X4), r(Y1), i(S4), i(D4)]),
    write("1 lo1v triggered\n"),
    fail.

o3v(_,  X1, Y1, _,  _,
     P2,
     P3, X3, Y3, S3, D3,
     P4,     Y4, _,  _) :-
    Y4 > Y1, Y1 > Y3,
    Priority = Y1 - Y3,
    trigger(Priority, rgo3, [i(P2), r(X1), r(Y3), r(X3), r(Y1),
                            i(P3), r(X3), r(Y3), i(S3), i(D3),
                            i(P4)]),
    write("1 go3v triggered\n"),
    fail.

o3v(_,  X1, Y1, _,  _,
     P2,
     P3, X3, Y3, S3, D3,
     P4,     Y4, _,  _) :-
    Y4 < Y1, Y1 < Y3,
    Priority = Y3 - Y1,
    trigger(Priority, rgo3, [i(P2), r(X1), r(Y3), r(X3), r(Y1),
                            i(P3), r(X3), r(Y3), i(S3), i(D3),
                            i(P4)]),
    write("1 go3v triggered\n"),
    fail.

o3v(P1, _,  Y1, _,  _,
     P2,
     P3, X3, Y3, S3, D3,
     P4,     Y4, S4, D4) :-
    Y4 = Y1, Y1 > Y3,
    Priority = Y1 - Y3,
    trigger(Priority, reo3, [i(P1),
                      i(P2), r(X3), r(Y1),
                      i(P3), r(X3), r(Y3), i(S3), i(D3),
                      i(P4), r(X3), r(Y4), i(S4), i(D4)]),
   write("1 eo3v triggered\n"),
   fail.

o3v(P1, _,  Y1, _,  _,
    P2,
    P3, X3, Y3, S3, D3,
    P4,     Y4, S4, D4) :-
   Y4 = Y1, Y1 < Y3,
   Priority = Y3 - Y1,
   trigger(Priority, reo3, [i(P1),
                      i(P2), r(X3), r(Y1),
                      i(P3), r(X3), r(Y3), i(S3), i(D3),
                      i(P4), r(X3), r(Y4), i(S4), i(D4)]),
   write("1 eo3v triggered\n"),
   fail.

o3v(_,  X1, Y1, _,  _,
    P2,
    P3, X3, Y3, S3, D3,
    P4,     Y4, S4, D4) :-
   Y1 > Y4, Y4 > Y3,
   Priority = Y4 - Y3,
   trigger(Priority, rlo3, [i(P2), r(X1), r(Y3), r(X3), r(Y1),
                      i(P3), r(X3), r(Y3), i(S3), i(D3),
                      i(P4), r(X3), r(Y4), i(S4), i(D4)]),
   write("1 lo3v triggered\n"),
   fail.

o3v(_,  X1, Y1, _,  _,
    P2,
    P3, X3, Y3, S3, D3,
    P4,     Y4, S4, D4) :-
   Y1 < Y4, Y4 < Y3,
   Priority = Y3 - Y4,
   trigger(Priority, rlo3, [i(P2), r(X1), r(Y3), r(X3), r(Y1),
                      i(P3), r(X3), r(Y3), i(S3), i(D3),
                      i(P4), r(X3), r(Y4), i(S4), i(D4)]),
   write("1 lo3v triggered\n"),
   fail.

o3v(P1, X1, Y1, S1, 1,
    P2,
    P3, X3, Y3, S3, 2,
    P4,     Y4, S4, D4) :-
   Y1 > Y3, Y3 > Y4,
   dist(X3, Y1, Priority),
   trigger(Priority, reo3, [i(P1), r(X1), r(Y1), i(S1), i(1),
                      i(P2), r(X1), r(Y3), r(X3), r(Y1),
                      i(P3), r(X3), r(Y3), i(S3), i(2),
                     i(P4), r(X3), r(Y4), i(S4), i(D4)]),
   write("1 e1o3v triggered\n"),
   fail.

o3v(P1, X1, Y1, S1, 1,
    P2,
    P3, X3, Y3, S3, 2,
    P4,     Y4, S4, D4) :-
   Y1 < Y3, Y3 < Y4,
   dist(X3, Y1, Priority),
   trigger(Priority, reo3, [i(P1), r(X1), r(Y1), i(S1), i(1),
                     i(P2), r(X1), r(Y3), r(X3), r(Y1),
                     i(P3), r(X3), r(Y3), i(S3), i(2),
                     i(P4), r(X3), r(Y4), i(S4), i(D4)]),
   write("1 e1o3v triggered\n"),
   fail.

o3v(P1, X1, Y1, S1, 3,
    P2,
    P3, X3, Y3, S3, 2,
    P4,     Y4, S4, D4) :-
   Y1 > Y3, Y3 > Y4,
   dist(X3, Y1, Priority),
   trigger(Priority, reo3, [i(P1), r(X1), r(Y1), i(S1), i(3),
                     i(P2), r(X1), r(Y3), r(X3), r(Y1),
                     i(P3), r(X3), r(Y3), i(S3), i(2),
                     i(P4), r(X3), r(Y4), i(S4), i(D4)]),
   write("1 e3o3v triggered\n"),
   fail.

o3v(P1, X1, Y1, S1, 3,
    P2,
    P3, X3, Y3, S3, 2,
    P4,     Y4, S4, D4) :-
   Y1 < Y3, Y3 < Y4,
   dist(X3, Y1, Priority),
   trigger(Priority, reo3, [i(P1), r(X1), r(Y1), i(S1), i(3),
                     i(P2), r(X1), r(Y3), r(X3), r(Y1),
                     i(P3), r(X3), r(Y3), i(S3), i(2),
                     i(P4), r(X3), r(Y4), i(S4), i(D4)]),
   write("1 e3o3v triggered\n"),
   fail.
```

```
code = 4096
project "RSTRB2"
include "global2.pro"

predicates
   hs(integer, real, real, integer, integer,
      integer,                integer, integer,
      integer, real, real, integer, integer,
      integer,          real, integer, integer)
   vs(integer, real, real, integer, integer,
      integer,                integer, integer,
      integer, real, real, integer, integer,
      integer, real,          integer, integer)

clauses
   slide(P1,
         P2, X3, Y1, S2, D2,
         P3) :-
      point(P1, X1, Y1, S1, D1),
      point(P3, X3, Y3, S3, D3),
      lin(P4, P1),
      point(P4, X1, Y4, S4, D4),
      hs(P1, X1, Y1, S1, D1,
         P2,          S2, D2,
         P3, X3, Y3, S3, D3,
         P4,     Y4, S4, D4).

   slide(P1,
         P2, X1, Y3, S2, D2,
         P3) :-
      point(P1, X1, Y1, S1, D1),
      point(P3, X3, Y3, S3, D3),
      lin(P4, P1),
      point(P4, X4, Y1, S4, D4),
      vs(P1, X1, Y1, S1, D1,
         P2,          S2, D2,
         P3, X3, Y3, S3, D3,
         P4, X4,     S4, D4).

   hs(P1, X1, Y1, S1, D1,
      P2,          S2, D2,
      P3, X3, Y3, S3, D3,
      P4,     Y4, S4, D4) :-
      Y4 > Y3, Y3 > Y1,
      trigger(0, rgs, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                   i(P2), r(X3), r(Y1), i(S2), i(D2),
                   i(P3), r(X3), r(Y3), i(S3), i(D3),
                   i(P4), r(X1), r(Y4), i(S4), i(D4),
                          r(X1), r(Y3)]),
      write("1 ghs triggered\n"),
      fail.
```

```
hs(P1, X1, Y1, S1, D1,
   P2,            S2, D2,
   P3, X3, Y3, S3, D3,
   P4,     Y4, S4, D4) :-
   Y4 < Y3, Y3 < Y1,
   trigger(0, rgs, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                 i(P2), r(X3), r(Y1), i(S2), i(D2),
                 i(P3), r(X3), r(Y3), i(S3), i(D3),
                 i(P4), r(X1), r(Y4), i(S4), i(D4),
                        r(X1), r(Y3)]),
   write("1 ghs triggered\n"),
   fail.

hs(P1, X1, Y1, S1, D1,
   P2,            S2, D2,
   P3, X3, Y3, S3, D3,
   P4,     Y3, S4, D4) :-
   trigger(0, res, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                 i(P2), r(X3), r(Y1), i(S2), i(D2),
                 i(P3), r(X3), r(Y3), i(S3), i(D3),
                 i(P4), r(X1), r(Y3), i(S4), i(D4)]),
   write("1 ehs triggered\n"),
   fail.

hs(P1, X1, Y1, S1, D1,
   P2,            S2, D2,
   P3, X3, Y3, S3, D3,
   P4,     Y4, S4, D4) :-
   Y3 > Y4, Y4 > Y1,
   trigger(0, rls, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                 i(P2), r(X3), r(Y1), i(S2), i(D2),
                 i(P3), r(X3), r(Y3), i(S3), i(D3),
                 i(P4), r(X1), r(Y4), i(S4), i(D4),
                        r(X3), r(Y4)]),
   write("1 lhs triggered\n"),
   fail.

hs(P1, X1, Y1, S1, D1,
   P2,            S2, D2,
   P3, X3, Y3, S3, D3,
   P4,     Y4, S4, D4) :-
   Y3 < Y4, Y4 < Y1,
   trigger(0, rls, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                 i(P2), r(X3), r(Y1), i(S2), i(D2),
                 i(P3), r(X3), r(Y3), i(S3), i(D3),
                 i(P4), r(X1), r(Y4), i(S4), i(D4),
                        r(X3), r(Y4)]),
   write("1 lhs triggered\n"),
   fail.

vs(P1, X1, Y1, S1, D1,
   P2,            S2, D2,
   P3, X3, Y3, S3, D3,
   P4, X4,     S4, D4) :-
   X1 > X3, X3 > X4,
   trigger(0, rgs, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                 i(P2), r(X1), r(Y3), i(S2), i(D2),
                 i(P3), r(X3), r(Y3), i(S3), i(D3),
                 i(P4), r(X4), r(Y1), i(S4), i(D4),
                        r(X3), r(Y1)]),
   write("1 gvs triggered\n"),
   fail.

vs(P1, X1, Y1, S1, D1,
   P2,             S2, D2,
   P3, X3, Y3, S3, D3,
   P4, X4,     S4, D4) :-
   X1 < X3, X3 < X4,
   trigger(0, rgs, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                 i(P2), r(X1), r(Y3), i(S2), i(D2),
                 i(P3), r(X3), r(Y3), i(S3), i(D3),
                 i(P4), r(X4), r(Y1), i(S4), i(D4),
                        r(X3), r(Y1)]),
   write("1 gvs triggered\n"),
   fail.

vs(P1, X1, Y1, S1, D1,
   P2,             S2, D2,
   P3, X3, Y3, S3, D3,
   P4, X3,     S4, D4) :-
   trigger(0, res, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                 i(P2), r(X1), r(Y3), i(S2), i(D2),
                 i(P3), r(X3), r(Y3), i(S3), i(D3),
                 i(P4), r(X3), r(Y1), i(S4), i(D4)]),
   write("1 evs triggered\n"),
   fail.

vs(P1, X1, Y1, S1, D1,
   P2,             S2, D2,
   P3, X3, Y3, S3, D3,
   P4, X4,     S4, D4) :-
   X1 > X4, X4 > X3,
   trigger(0, rls, [i(P1), r(X1), r(Y1), i(S1), i(D1),
                 i(P2), r(X1), r(Y3), i(S2), i(D2),
                 i(P3), r(X3), r(Y3), i(S3), i(D3),
                 i(P4), r(X4), r(Y1), i(S4), i(D4),
                        r(X4), r(Y3)]),
   write("1 lvs triggered\n"),
   fail.

vs(P1, X1, Y1, S1, D1,
   P2,             S2, D2,
   P3, X3, Y3, S3, D3,
   P4, X4,     S4, D4) :-
```

```
X1 < X4, X4 < X3,
trigger(0, rls, [i(P1), r(X1), r(Y1), i(S1), i(D1),
            i(P2), r(X1), r(Y3), i(S2), i(D2),
            i(P3), r(X3), r(Y3), i(S3), i(D3),
            i(P4), r(X4), r(Y1), i(S4), i(D4),
                   r(X4), r(Y3)]),
write("1 lvs triggered\n"),
fail.
```

```
code = 4096
project "RSTRB2"
include "global2.pro"

clauses
   execute(rgi1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
           i(P2), r(X2), r(Y2), r(X2new), r(Y2new),
           i(P4)]) :-
      write("gi1 entered\n"),
      point(P2, X2, Y2, 0, 2),
      lin(P1, P2),
      point(P1, X1, Y1, S1, D1),
      lin(P4, P1),
      D1new = D1 - 1,
      retract(point(P2, X2, Y2, 0, 2)),
      assertz(point(P2, X2new, Y2new, 0, 3)),
      retract(lin(P1, P2)),
      assertz(lin(P2, P1)),
      retract(point(P1, X1, Y1, S1, D1)),
      assertz(point(P1, X1, Y1, S1, D1new)),
      retract(lin(P4, P1)),
      assertz(lin(P4, P2)),
      write("gi1 fired\n\n"),
      !.

   execute(re1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
           i(P2), r(X2), r(Y2),
           i(P3),
           i(P4), r(X4), r(Y4), i(S4), i(D4)]) :-
      write("e1 entered\n"),
      point(P2, X2, Y2, 0, 2),
      lin(P1, P2),
      lin(P2, P3),
      point(P1, X1, Y1, S1, D1),
      point(P4, X4, Y4, S4, D4),
      D1new = D1 - 1,
      D4new = D4 + 1,
      retract(point(P2, X2, Y2, 0, 2)),
      decount,
      retract(lin(P1, P2)),
      retract(lin(P2, P3)),
      assertz(lin(P4, P3)),
      retract(point(P1, X1, Y1, S1, D1)),
      assertz(point(P1, X1, Y1, S1, D1new)),
      retract(point(P4, X4, Y4, S4, D4)),
      assertz(point(P4, X4, Y4, S4, D4new)),
      write("e1 fired\n\n"),
      !.

   execute(rl1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
           i(P2), r(X2), r(Y2), r(X2new), r(Y2new),
           i(P4), r(X4), r(Y4), i(S4), i(D4)]) :-
   write("l1 entered\n"),
   point(P2, X2, Y2, 0, 2),
   lin(P1, P2),
   point(P1, X1, Y1, S1, D1),
   point(P4, X4, Y4, S4, D4),
   D1new = D1 - 1,
   D4new = D4 + 1,
   retract(point(P2, X2, Y2, 0, 2)),
   assertz(point(P2, X2new, Y2new, 0, 2)),
   retract(lin(P1, P2)),
   assertz(lin(P4, P2)),
   retract(point(P1, X1, Y1, S1, D1)),
   assertz(point(P1, X1, Y1, S1, D1new)),
   retract(point(P4, X4, Y4, S4, D4)),
   assertz(point(P4, X4, Y4, S4, D4new)),
   write("l1 fired\n\n"),
   !.

execute(rgo1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
        i(P2), r(X2), r(Y2), r(X2new), r(Y2new),
        i(P4)]) :-
   write("go1 entered\n"),
   point(P2, X2, Y2, 0, 2),
   lin(P1, P4),
   point(P1, X1, Y1, S1, D1),
   D1new = D1 -1,
   retract(point(P2, X2, Y2, 0, 2)),
   assertz(point(P2, X2new, Y2new, 0, 3)),
   retract(lin(P1, P4)),
   assertz(lin(P2, P4)),
   retract(point(P1, X1, Y1, S1, D1)),
   assertz(point(P1, X1, Y1, S1, D1new)),
   write("go1 fired\n\n"),
   !.

execute(rgo3, [i(P2), r(X2), r(Y2), r(X2new), r(Y2new),
        i(P3), r(X3), r(Y3), i(S3), i(D3),
        i(P4)]) :-
   write("go3 entered\n"),
   point(P2, X2, Y2, 0, 2),
   lin(P3, P4),
   point(P3, X3, Y3, S3, D3),
   D3new = D3 - 1,
   retract(point(P2, X2, Y2, 0, 2)),
   assertz(point(P2, X2new, Y2new, 0, 3)),
   retract(lin(P3, P4)),
   assertz(lin(P2, P4)),
   retract(point(P3, X3, Y3, S3, D3)),
   assertz(point(P3, X3, Y3, S3, D3new)),
   write("go3 fired\n\n"),
   !.
```

```
execute(reo3, [i(P1),
        i(P2), r(X2), r(Y2),
        i(P3), r(X3), r(Y3), i(S3), i(D3),
        i(P4), r(X4), r(Y4), i(S4), i(D4)]) :-
   write("eo3 entered\n\n"),
   point(P2, X2, Y2, 0, 2),
   lin(P1, P2),
   lin(P2, P3),
   lin(P3, P4),
   point(P3, X3, Y3, S3, D3),
   point(P4, X4, Y4, S4, D4),
   D3new = D3 - 1,
   D4new = D4 + 1,
   retract(point(P2, X2, Y2, 0, 2)),
   decount,
   retract(lin(P1, P2)),
   retract(lin(P2, P3)),
   assertz(lin(P1, P4)),
   retract(lin(P3, P4)),
   assertz(lin(P4, P3)),
   retract(point(P3, X3, Y3, S3, D3)),
   assertz(point(P3, X3, Y3, S3, D3new)),
   retract(point(P4, X4, Y4, S4, D4)),
   assertz(point(P4, X4, Y4, S4, D4new)),
   write("eo3 fired\n\n"),
   !.

execute(rlo3, [i(P2), r(X2), r(Y2), r(X2new), r(Y2new),
        i(P3), r(X3), r(Y3), i(S3), i(D3),
        i(P4), r(X4), r(Y4), i(S4), i(D4)]) :-
   write("lo3 entered\n"),
   point(P2, X2, Y2, 0, 2),
   lin(P2, P3),
   lin(P3, P4),
   point(P3, X3, Y3, S3, D3),
   point(P4, X4, Y4, S4, D4),
   D3new = D3 - 1,
   D4new = D4 + 1,
   retract(point(P2, X2, Y2, 0, 2)),
   assertz(point(P2, X2new, Y2new, 0, 2)),
   retract(lin(P2, P3)),
   assertz(lin(P2, P4)),
   retract(lin(P3, P4)),
   assertz(lin(P4, P3)),
   retract(point(P3, X3, Y3, S3, D3)),
   assertz(point(P3, X3, Y3, S3, D3new)),
   retract(point(P4, X4, Y4, S4, D4)),
   assertz(point(P4, X4, Y4, S4, D4new)),
   write("lo3 fired\n\n"),
   !.

execute(resi1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
         i(P2), r(X2), r(Y2), r(X2new), r(Y2new),
         i(P3), r(X3), r(Y3), i(S3), i(D3),
         i(P4), r(X4), r(Y4), i(S4), i(D4),
         i(P5), r(X5), r(Y5), i(S5), i(D5)]) :-
    write("esi1 entered\n"),
    point(P1, X1, Y1, S1, D1),
    point(P2, X2, Y2, 0, 2),
    point(P3, X3, Y3, S3, D3),
    point(P4, X4, Y4, S4, D4),
    point(P5, X5, Y5, S5, D5),
    lin(P1, P2),
    lin(P2, P3),
    lin(P4, P1),
    lin(P3, P5),
    retract(point(P2, X2, Y2, 0, 2)),
    assertz(point(P2, X2new, Y2new, 0, 2)),
    write("esi1 fired\n\n"),
    !.

execute(rei1, [i(P1), r(X1), r(Y1), i(S1), i(D1),
        i(P2), r(X2), r(Y2), r(X2new), r(Y2new),
        i(P3), r(X3), r(Y3), i(S3), i(D3),
        i(P4), r(X4), r(Y4), i(S4), i(D4)]) :-
    write("ei1 entered\n"),
    point(P1, X1, Y1, S1, D1),
    point(P2, X2, Y2, 0, 2),
    point(P3, X3, Y3, S3, D3),
    point(P4, X4, Y4, S4, D4),
    lin(P1, P2),
    lin(P2, P3),
    lin(P4, P1),
    retract(point(P2, X2, Y2, 0, 2)),
    assertz(point(P2, X2new, Y2new, 0, 2)),
    write("ei1 fired\n\n"),
    !.

execute(reo3, [i(P1), r(X1), r(Y1), i(S1), i(D1),
        i(P2), r(X2), r(Y2), r(X2new), r(Y2new),
        i(P3), r(X3), r(Y3), i(S3), i(D3),
        i(P4), r(X4), r(Y4), i(S4), i(D4)]) :-
    write("eo3 entered\n"),
    point(P1, X1, Y1, S1, D1),
    point(P2, X2, Y2, 0, 2),
    point(P3, X3, Y3, S3, D3),
    point(P4, X4, Y4, S4, D4),
    lin(P1, P2),
    lin(P2, P3),
    lin(P3, P4),
    retract(point(P2, X2, Y2, 0, 2)),
    assertz(point(P2, X2new, Y2new, 0, 2)),
    write("eo3 fired\n\n"),
    !.
```

```
execute(rgs, [i(P1), r(X1), r(Y1), i(S1), i(D1),
       i(P2), r(X2), r(Y2), i(S2), i(D2),
       i(P3), r(X3), r(Y3), i(S3), i(D3),
       i(P4), r(X4), r(Y4), i(S4), i(D4),
              r(X5), r(Y5)]) :-
   write("gs entered\n"),
   point(P1, X1, Y1, S1, D1),
   point(P2, X2, Y2, S2, D2),
   point(P3, X3, Y3, S3, D3),
   point(P4, X4, Y4, S4, D4),
   lin(P1, P2),
   lin(P2, P3),
   lin(P4, P1),
   D1new = D1 - 1,
   D2new = D2 - 1,
   D3new = D3 + 1,
   retract(point(P1, X1, Y1, S1, D1)),
   assertz(point(P1, X1, Y1, S1, D1new)),
   retract(point(P2, X2, Y2, S2, D2)),
   assertz(point(P2, X2, Y2, S2, D2new)),
   retract(point(P3, X3, Y3, S3, D3)),
   assertz(point(P3, X3, Y3, S3, D3new)),
   incount(P5),
   assertz(point(P5, X5, Y5, 0, 3)),
   retract(lin(P4, P1)),
   assertz(lin(P4, P5)),
   assertz(lin(P5, P1)),
   retract(lin(P1, P2)),
   assertz(lin(P5, P3)),
   retract(lin(P2, P3)),
   assertz(lin(P3, P2)),
   write("gs fired\n\n").

execute(res, [i(P1), r(X1), r(Y1), i(S1), i(D1),
       i(P2), r(X2), r(Y2), i(S2), i(D2),
       i(P3), r(X3), r(Y3), i(S3), i(D3),
       i(P4), r(X4), r(Y4), i(S4), i(D4)]) :-
   write("es entered\n"),
   point(P1, X1, Y1, S1, D1),
   point(P2, X2, Y2, S2, D2),
   point(P3, X3, Y3, S3, D3),
   point(P4, X4, Y4, S4, D4),
   lin(P1, P2),
   lin(P2, P3),
   lin(P4, P1),
   D1new = D1 - 1,
   D2new = D2 - 1,
   D3new = D3 + 1,
   D4new = D4 + 1,
   retract(point(P1, X1, Y1, S1, D1)),
   assertz(point(P1, X1, Y1, S1, D1new)),
   retract(point(P2, X2, Y2, S2, D2)),
   assertz(point(P2, X2, Y2, S2, D2new)),
   retract(point(P3, X3, Y3, S3, D3)),
   assertz(point(P3, X3, Y3, S3, D3new)),
   retract(point(P4, X4, Y4, S4, D4)),
   assertz(point(P4, X4, Y4, S4, D4new)),
   retract(lin(P1, P2)),
   assertz(lin(P4, P3)),
   retract(lin(P2, P3)),
   assertz(lin(P3, P2)),
   write("es fired\n\n").

execute(rls, [i(P1), r(X1), r(Y1), i(S1), i(D1),
       i(P2), r(X2), r(Y2), i(S2), i(D2),
       i(P3), r(X3), r(Y3), i(S3), i(D3),
       i(P4), r(X4), r(Y4), i(S4), i(D4),
              r(X5), r(Y5)]) :-
   write("ls entered\n"),
   point(P1, X1, Y1, S1, D1),
   point(P2, X2, Y2, S2, D2),
   point(P3, X3, Y3, S3, D3),
   point(P4, X4, Y4, S4, D4),
   lin(P1, P2),
   lin(P2, P3),
   lin(P4, P1),
   D1new = D1 - 1,
   D2new = D2 - 1,
   D4new = D4 + 1,
   retract(point(P1, X1, Y1, S1, D1)),
   assertz(point(P1, X1, Y1, S1, D1new)),
   retract(point(P2, X2, Y2, S2, D2)),
   assertz(point(P2, X2, Y2, S2, D2new)),
   retract(point(P4, X4, Y4, S4, D4)),
   assertz(point(P4, X4, Y4, S4, D4new)),
   incount(P5),
   assertz(point(P5, X5, Y5, 0, 3)),
   retract(lin(P1, P2)),
   assertz(lin(P4, P5)),
   retract(lin(P2, P3)),
   assertz(lin(P5, P2)),
   assertz(lin(P5, P3)),
   write("ls fired\n\n").

execute(_, _) :-
   write("rule is not fired\n\n").
```

APPENDIX D: SOURCE CODE FOR HYDRAULIC DESIGN COMPONENT

```
      PROGRAM SUBMIT
C     ******************************************************************
C     *                                                                *
C     *    VARIABLE DICTIONARY                                         *
C     *                                                                *
C     *    CONFIGURATION VARIABLES                                     *
C     *                                                                *
C     *       UC267  - THE UNIT COST OF 26.7 P.E. PIPE ($/m)           *
C     *       UC334  - THE UNIT COST OF 33.4 P.E. PIPE ($/m)           *
C     *       UC483  - THE UNIT COST OF 48.3 P.E. PIPE ($/m)           *
C     *       UC603  - THE UNIT COST OF 60.3 P.E. PIPE ($/m)           *
C     *       UC889  - THE UNIT COST OF 88.9 P.E. PIPE ($/m)           *
C     *       UCROAD - THE COST OF A ROAD CROSSING ($)                 *
C     *       UCRAIL - THE COST OF A RAILROAD CROSSING ($)             *
C     *       UCABLE - THE COST OF A CABLE CROSSING ($)                *
C     *       UCREEK - THE COST OF A CREEK/RIVER CROSSING ($)          *
C     *       RPERKM - THE NUMBER OF ROAD CROSSINGS PER KILOMETER      *
C     *       CPERKM - THE NUMBER OF CABLE CROSSINGS PER KILOMETER     *
C     *       RLEN   - THE EQUIVALENT LENGTH OF A RAILROAD CROSSING    *
C     *       CLEN   - THE EQUIVALENT LENGTH OF A CREEK CROSSING       *
C     *                                                                *
C     *                                                                *
C     *    NODE VARIABLES                                              *
C     *                                                                *
C     *       N      - THE NUMBER OF NODES                             *
C     *       X      - THE X-COORDINATE OF EACH NODE (m)               *
C     *       Y      - THE Y-COORDINATE OF EACH NODE (m)               *
C     *       NUMN   - THE NUMBER OF INTERMITTENT LOADS                *
C     *       QIN    - THE INTERMITTENT LOAD (m^3/hr)                  *
C     *       QCN    - THE CONTINUOUS LOAD (m^3/hr)                    *
C     *       QDN    - THE DRYER LOAD (m^3/hr)                         *
C     *       IRSET  - THE RAILROAD SET                                *
C     *       ICSET  - THE RIVER/CREEK SET                             *
C     *                                                                *
C     *                                                                *
C     *    LINE VARIABLES                                              *
C     *                                                                *
C     *       NL     - THE NUMBER OF LINES                             *
C     *       IFR    - THE ORIGIN NODE NUMBER                          *
C     *       ITO    - THE DESTINATION NODE NUMBER                     *
C     *       ALEN   - THE LENGTH OF A LINE (m)                        *
C     *       X1     - THE X-COORDINATE OF ORIGIN (m)                  *
C     *       Y1     - THE Y-COORDINATE OF ORIGIN (m)                  *
C     *       X2     - THE X-COORDINATE OF DESTINATION (m)             *
C     *       Y2     - THE Y-COORDINATE OF DESTINATION (m)             *
C     *       LMAX   - THE MAXIMUM LEVEL IN THE TREE                   *
C     *       LEV    - THE LEVEL OF A LINE                             *
C     *       DMAX   - THE LENGTH OF THE LONGEST LEG IN THE TREE (m)   *
C     *       DIST   - THE DISTANCE FROM DESTINATION TO SOURCE (m)     *
C     *       NUML   - THE NUMBER OF INTERMITTENT LOADS                *
C     *       QIL    - THE INTERMITTENT LOAD (m^3/hr)                  *
C     *       QCL    - THE CONTINUOUS LOAD (m^3/hr)                    *
```

```
C     *       QDL     - THE DRYER LOAD (m^3/hr)                              *
C     *       FC      - THE COINCIDENCE FACTOR                               *
C     *       QTL     - THE TOTAL LOAD OR DESIGN LOAD (m^3/hr)               *
C     *       PDMAX   - THE OPTIMUM PRESSURE DROP PER KM (kPa/m)             *
C     *       PIN     - THE INLET PRESSURE (kPa)                             *
C     *       POUT    - THE OUTLET PRESSURE (kPa)                            *
C     *       PD      - THE PRESSURE DROP PER KILOMETER (kPa/km)             *
C     *       SIZE    - THE DIAMETER OF A PIPE SEGMENT (mm)                  *
C     *                                                                      *
C     *                                                                      *
C     *    SYSTEM VARIABLES                                                  *
C     *                                                                      *
C     *       FAIL1   - FLAG ( TRUE IF LAYOUT FAILS )                        *
C     *       FAIL2   - FLAG ( TRUE IF MAX. CAPACITY EXCEEDED )              *
C     *       FAIL3   - FLAG ( TRUE IF INSUFFICIENT END PRESSURE)            *
C     *       FAIL4   - FLAG ( TRUE IF PRESSURE DROP .GT. 20 kPa/km)         *
C     *       AL267   - THE TOTAL LENGTH OF 26.7 P.E. PIPE (m)               *
C     *       AL334   - THE TOTAL LENGTH OF 33.4 P.E. PIPE (m)               *
C     *       AL483   - THE TOTAL LENGTH OF 48.3 P.E. PIPE (m)               *
C     *       AL603   - THE TOTAL LENGTH OF 60.3 P.E. PIPE (m)               *
C     *       AL889   - THE TOTAL LENGTH OF 88.9 P.E. PIPE (m)               *
C     *       IROAD   - THE NUMBER OF ROAD CROSSINGS                         *
C     *       IRAIL   - THE NUMBER OF RAIL CROSSINGS                         *
C     *       ICABLE  - THE NUMBER OF CABLE CROSSINGS                        *
C     *       ICREEK  - THE NUMBER OF CREEK CROSSINGS                        *
C     *       ALTOT   - THE TOTAL LENGTH OF PIPE (m)                         *
C     *       IWARN1  - OUTLET PRESSURE BELOW 140 kPa                        *
C     *       IWARN2  - PRESSURE DROP OF 20 kPa/km EXCEEDED                  *
C     *                                                                      *
C     *                                                                      *
C     *    COST VARIABLES                                                    *
C     *                                                                      *
C     *       C267    - THE COST OF 26.7 P.E. PIPE ($)                       *
C     *       C334    - THE COST OF 33.4 P.E. PIPE ($)                       *
C     *       C483    - THE COST OF 48.3 P.E. PIPE ($)                       *
C     *       C603    - THE COST OF 60.3 P.E. PIPE ($)                       *
C     *       C889    - THE COST OF 88.9 P.E. PIPE ($)                       *
C     *       CPIPE   - THE TOTAL COST OF ALL PIPE ($)                       *
C     *       CROAD   - THE COST OF ALL ROAD CROSSINGS ($)                   *
C     *       CRAIL   - THE COST OF ALL RAIL CROSSINGS ($)                   *
C     *       CCABLE  - THE COST OF ALL CABLE CROSSINGS ($)                  *
C     *       CCREEK  - THE COST OF ALL CREEK CROSSINGS ($)                  *
C     *       CTOT    - THE TOTAL COST OF THE SYSTEM ($)                     *
C     *                                                                      *
C     ************************************************************************
C
C     GLOBAL VARIABLES
C
C     CONFIGURATION VARIABLES
      REAL UC267,UC334,UC483,UC603,UC889,UCROAD,UCRAIL,UCABLE,UCREEK
      REAL RPERKM,CPERKM,RLEN,CLEN
C     NODE VARIABLES
```

```
      INTEGER N,NUMN(200),IRSET(200),ICSET(200)
      REAL X(200),Y(200),QIN(200),QCN(200),QDN(200)
C     LINE VARIABLES
      INTEGER NL,IFR(200),ITO(200),LMAX,LEV(200),NUML(200)
      REAL ALEN(200),DMAX,DIST(200),X1(200),Y1(200),X2(200),Y2(200)
      REAL QIL(200),QCL(200),QDL(200),FC(200),QTL(200)
      REAL PDMAX,PIN(200),POUT(200),PD(200),SIZE(200)
C     SYSTEM VARIABLES
      LOGICAL FAIL1,FAIL2,FAIL3,FAIL4
      INTEGER IROAD,IRAIL,ICABLE,ICREEK,IWARN1(200),IWARN2(200)
      REAL AL267,AL334,AL483,AL603,ALTOT
C     COST VARIABLES
      REAL C267,C334,C483,C603,C889,CPIPE,CROAD,CRAIL,CCABLE,CCREEK,CTOT
C
C     MAIN LINE
C
C     CONFIGURE THE PROGRAM
      CALL UCREAD (UC267,UC334,UC483,UC603,UC889,UCROAD,UCRAIL,UCABLE,
     *UCREEK,RPERKM,CPERKM,RLEN,CLEN)
C     READ NODES FILE
      CALL NDREAD (N,X,Y,NUMN,QIN,QCN,QDN,IRSET,ICSET)
C     WRITE NODES.LSP FILE
      CALL NDLRIT (N,X,Y,NUMN,QIN,QCN,QDN,IRSET,ICSET)
C     WRITE NODES.DAT FILE
      CALL NDDRIT (N,X,Y,NUMN,QIN,QCN,QDN,IRSET,ICSET)
C     FIND LAYOUT
      CALL ROUTE (RLEN,CLEN,N,X,Y,IRSET,ICSET,NL,IFR,ITO,ALEN,X1,Y1,X2,
     *Y2,IRAIL,ICREEK)
C     FIND THE LEVEL OF THE LINES IN THE TREE
      CALL LEVEL (NL,IFR,ITO,ALEN,LMAX,LEV,DMAX,DIST)
C     FIND THE LOAD ON EACH LINE
      CALL LOAD (NUMN,QIN,QCN,QDN,NL,IFR,ITO,LMAX,LEV,NUML,QIL,QCL,QDL)
C     FIND THE DESIGN LOAD FOR EACH LINE
      CALL FLOAD (NL,NUML,QIL,QCL,QDL,FC,QTL)
C     CHECK LAYOUT
      CALL CHECK1 (NL,QTL,FAIL1,FAIL2)
      IF (FAIL1.OR.FAIL2) GO TO 100
C     GET DESIRED PRESSURE DROP PER KM
      CALL MAXPD (DMAX,PDMAX)
C     FIND PIPE SIZES AND PRESSURES
      CALL PRESS (NL,IFR,ITO,ALEN,LMAX,LEV,QTL,PDMAX,PIN,POUT,PD,SIZE,
     *IWARN1,IWARN2)
C     CHECK FOR SUFFICIENT END PRESSURE AND PRESSURE DROP CONSTRAINT
      CALL CHECK2 (NL,IWARN1,IWARN2,FAIL3,FAIL4)
C     TOTAL PIPE LENGTHS
      CALL TOTAL (RPERKM,CPERKM,NL,ALEN,SIZE,AL267,AL334,AL483,AL603,
     *AL889,IROAD,ICABLE,ALTOT)
C     FIND COSTS
      CALL COSTS (UC267,UC334,UC483,UC603,UC889,UCROAD,UCRAIL,UCABLE,
     *UCREEK,AL267,AL334,AL483,AL603,AL889,IROAD,IRAIL,ICABLE,ICREEK,
     *C267,C334,C483,C603,C889,CPIPE,CROAD,CRAIL,CCABLE,CCREEK,CTOT)
C     WRITE LINES.DAT FILE
```

```
      CALL LNDRIT (NL,IFR,ITO,ALEN,NUML,QTL,PIN,POUT,PD,SIZE,IWARN1,
     *IWARN2,AL267,AL334,AL483,AL603,AL889,IROAD,IRAIL,ICABLE,ICREEK,
     *ALTOT,C267,C334,C483,C603,C889,CPIPE,CROAD,CRAIL,CCABLE,CCREEK,
     *CTOT)
C     WRITE LINES.LSP FILE
      CALL LNLRIT (NL,X1,Y1,X2,Y2,SIZE)
      GO TO 101
  100 CONTINUE
C     WRITE ERROR LINES.DAT FILE
      CALL FDRIT (NL,IFR,ITO,ALEN,NUML,QTL)
C     WRITE ERROR LINES.LSP FILE
      CALL FLRIT (NL,X1,Y1,X2,Y2,FAIL1)
  101 CONTINUE
      STOP
      END
      SUBROUTINE UCREAD (UC267,UC334,UC483,UC603,UC889,UCROAD,UCRAIL,
     *UCABLE,UCREEK,RPERKM,CPERKM,RLEN,CLEN)
C     ****************************************************************
C     *                                                              *
C     *   THIS SUBROUTINE READS IN DATA FROM THE FILE "UCOST"        *
C     *   AND CALCULATES THE LENGTH OF 26.7 mm PIPE EQUIVALENT       *
C     *   TO A RAILWAY CROSSING AND A CREEK CROSSING.                *
C     *                                                              *
C     *                                                              *
C     *   VARIABLE DICTIONARY                                        *
C     *                                                              *
C     *   OUTPUT PARAMETERS                                          *
C     *                                                              *
C     *       UC267  - THE UNIT COST OF 26.7 P.E. PIPE ($/m)         *
C     *       UC334  - THE UNIT COST OF 33.4 P.E. PIPE ($/m)         *
C     *       UC483  - THE UNIT COST OF 48.3 P.E. PIPE ($/m)         *
C     *       UC603  - THE UNIT COST OF 60.3 P.E. PIPE ($/m)         *
C     *       UC889  - THE UNIT COST OF 88.9 P.E. PIPE ($/m)         *
C     *       UCROAD - THE COST OF A ROAD CROSSING ($)               *
C     *       UCRAIL - THE COST OF A RAILROAD CROSSING ($)           *
C     *       UCABLE - THE COST OF A CABLE CROSSING ($)              *
C     *       UCREEK - THE COST OF A CREEK/RIVER CROSSING ($)        *
C     *       RPERKM - THE NUMBER OF ROAD CROSSINGS PER KILOMETER    *
C     *       CPERKM - THE NUMBER OF CABLE CROSSINGS PER KILOMETER   *
C     *       RLEN   - THE EQUIVALENT LENGTH OF A RAILROAD CROSSING  *
C     *       CLEN   - THE EQUIVALENT LENGTH OF A CREEK CROSSING     *
C     *                                                              *
C     ****************************************************************
C     OUTPUT PARAMETERS
      REAL UC267,UC334,UC483,UC603,UC889,UCROAD,UCRAIL,UCABLE,UCREEK
      REAL RPERKM,CPERKM,RLEN,CLEN
      OPEN(UNIT=1,FILE='UCOST')
      READ(1,150)UC267
      READ(1,150)UC334
      READ(1,150)UC483
      READ(1,150)UC603
      READ(1,150)UC889
      READ(1,150)UCROAD
      READ(1,150)UCRAIL
      READ(1,150)UCABLE
      READ(1,150)UCREEK
      READ(1,150)RPERKM
      READ(1,150)CPERKM
      CLOSE(UNIT=1)
      RLEN=UCRAIL/UC267
      CLEN=UCREEK/UC267
      RETURN
  150 FORMAT(20X,F8.2)
      END
      SUBROUTINE NDREAD (N,X,Y,NUMN,QIN,QCN,QDN,IRSET,ICSET)
C     *****************************************************************
C     *                                                               *
C     *   THIS SUBROUTINE READS IN DATA IN THE "NODES" FILE           *
C     *   CREATED BY THE AUTOLISP ROUTINE "SUBMIT".                   *
C     *                                                               *
C     *                                                               *
C     *   VARIABLE DICTIONARY                                         *
C     *                                                               *
C     *   OUTPUT PARAMETERS                                           *
C     *                                                               *
C     *       N       - THE NUMBER OF NODES                           *
C     *       X       - THE X-COORDINATE OF EACH NODE (m)             *
C     *       Y       - THE Y-COORDINATE OF EACH NODE (m)             *
C     *       NUMN    - THE NUMBER OF INTERMITTENT LOADS              *
C     *       QIN     - THE INTERMITTENT LOAD (m^3/hr)                *
C     *       QCN     - THE CONTINUOUS LOAD (m^3/hr)                  *
C     *       QDN     - THE DRYER LOAD (m^3/hr)                       *
C     *       IRSET   - THE RAILROAD SET                              *
C     *       ICSET   - THE RIVER/CREEK SET                           *
C     *                                                               *
C     *                                                               *
C     *   LOCAL VARIABLES                                             *
C     *                                                               *
C     *       I       - A COUNTER                                     *
C     *                                                               *
C     *****************************************************************
C     OUTPUT PARAMETERS
      INTEGER N,NUMN(200),IRSET(200),ICSET(200)
      REAL X(200),Y(200),QIN(200),QCN(200),QDN(200)
C     LOCAL VARIABLES
      INTEGER I
      OPEN(UNIT=1,FILE='NODES')
      READ(1,150)N
      DO 100 I=1,N,1
      READ(1,*)X(I),Y(I),NUMN(I),QIN(I),QCN(I),QDN(I),
     *IRSET(I),ICSET(I)
  100 CONTINUE
      CLOSE(UNIT=1)
      RETURN
  150 FORMAT(/I4)
      END
      SUBROUTINE NDLRIT (N,X,Y,NUMN,QIN,QCN,QDN,IRSET,ICSET)
C     ******************************************************************
C     *                                                                *
C     *   THIS SUBROUTINE CREATES A NEW "NODES.LSP" FILE               *
C     *   USED BY THE AUTOLISP ROUTINES.                               *
C     *                                                                *
C     *                                                                *
C     *   VARIABLE DICTIONARY                                          *
C     *                                                                *
C     *   INPUT PARAMETERS                                             *
C     *                                                                *
C     *       N       - THE NUMBER OF NODES                            *
C     *       X       - THE X-COORDINATE OF EACH NODE (m)              *
C     *       Y       - THE Y-COORDINATE OF EACH NODE (m)              *
C     *       NUMN    - THE NUMBER OF INTERMITTENT LOADS               *
C     *       QIN     - THE INTERMITTENT LOAD (m^3/hr)                 *
C     *       QCN     - THE CONTINUOUS LOAD (m^3/hr)                   *
C     *       QDN     - THE DRYER LOAD (m^3/hr)                        *
C     *       IRSET   - THE RAILROAD SET                               *
C     *       ICSET   - THE RIVER/CREEK SET                            *
C     *                                                                *
C     *                                                                *
C     *    LOCAL VARIABLES                                             *
C     *                                                                *
C     *       I       - A COUNTER                                      *
C     *                                                                *
C     ******************************************************************
C     INPUT PARAMETERS
      INTEGER N,NUMN(200),IRSET(200),ICSET(200)
      REAL X(200),Y(200),QIN(200),QCN(200),QDN(200)
C     LOCAL VARIABLES
      INTEGER I
      OPEN(UNIT=1,FILE='NODES.LSP')
      WRITE(1,150)N
      DO 100 I=1,N,1
      WRITE(1,151)X(I),Y(I),NUMN(I),QIN(I),QCN(I),QDN(I),IRSET(I),
     *ICSET(I)
  100 CONTINUE
      WRITE(1,152)
      CLOSE(UNIT=1)
      RETURN
  150 FORMAT('(setq nodes (quote ( ',I3,' (')
  151 FORMAT('( ',2(F9.1,2X),I3,3(2X,F6.1),2(2X,I2),' )')
  152 FORMAT(' ))))')
      END
      SUBROUTINE NDDRIT (N,X,Y,NUMN,QIN,QCN,QDN,IRSET,ICSET)
C     ******************************************************************
C     *                                                                *
C     *   THIS SUBROUTINE CREATES THE "NODES.DAT" FILE WHICH           *
C     *   IS PRINTED BY THE "NODEDATA" AUTOCAD MACRO.                  *
C     *                                                                *
C     *                                                                *
C     *   VARIABLE DICTIONARY                                          *
C     *                                                                *
C     *   INPUT PARAMETERS                                             *
C     *                                                                *
C     *       N       - THE NUMBER OF NODES                            *
C     *       X       - THE X-COORDINATE OF EACH NODE (m)              *
C     *       Y       - THE Y-COORDINATE OF EACH NODE (m)              *
C     *       NUMN    - THE NUMBER OF INTERMITTENT LOADS               *
C     *       QIN     - THE INTERMITTENT LOAD (m^3/hr)                 *
C     *       QCN     - THE CONTINUOUS LOAD (m^3/hr)                   *
C     *       QDN     - THE DRYER LOAD (m^3/hr)                        *
C     *       IRSET   - THE RAILROAD SET                               *
C     *       ICSET   - THE RIVER/CREEK SET                            *
C     *                                                                *
C     *                                                                *
C     *   LOCAL VARIABLES                                              *
C     *                                                                *
C     *       I       - A COUNTER                                      *
C     *                                                                *
C     ******************************************************************
C     INPUT PARAMETERS
      INTEGER N,NUMN(200),IRSET(200),ICSET(200)
      REAL X(200),Y(200),QIN(200),QCN(200),QDN(200)
C     LOCAL VARIABLES
      INTEGER I
      OPEN(UNIT=1,FILE='(C)NODES.DAT')
      WRITE(1,150)
      DO 100 I=1,N,1
      WRITE(1,151)I,X(I),Y(I),NUMN(I),QIN(I),QCN(I),QDN(I),IRSET(I),
     *ICSET(I)
  100 CONTINUE
      WRITE(1,152)
      CLOSE(UNIT=1)
      RETURN
  150 FORMAT(' POINT',6X,'X',10X,'Y',6X,'NUM',3X,'QIN',5X,'QCN',5X,
     *'QDN',4X,'RSET  CSET'/' *****',2('  *********'),'  ***',
     *3('  ******'),2('  ****'))
  151 FORMAT(' ',I5,2(2X,F9.1),2X,I3,3(2X,F6.1),3X,I2,4X,I2)
  152 FORMAT('1')
      END
      SUBROUTINE ROUTE (RLEN,CLEN,N,X,Y,IRSET,ICSET,NL,IFR,ITO,ALEN,X1,
     *Y1,X2,Y2,IRAIL,ICREEK)
C     *****************************************************************
C     *                                                               *
C     *   THIS SUBROUTINE FINDS THE MINIMUM SPANNING TREE             *
C     *   THROUGH THE SET OF NODES GIVEN. NODES ARE GROUPED           *
C     *   INTO RAIL AND CREEK SETS TO DETERMINE IF A RAILWAY          *
C     *   OR CREEK CROSSING OCCURS. IF A LINE IS FOUND TO             *
C     *   CROSS A RAILWAY OR CREEK THE LENGTH IS WEIGHTED IN          *
C     *   PROPORTION TO THE COST OF THAT CROSSING. THE TREE           *
C     *   IS DETERMINED ON THE BASIS OF THE WEIGHTED LENGTHS.         *
C     *                                                               *
C     *   THIS SUBROUTINE RETURNS THE ACTUAL LENGTHS OF LINES         *
C     *   RATHER THAN THE WEIGHTED LENGTHS.                           *
C     *                                                               *
C     *                                                               *
C     *   VARIABLE DICTIONARY                                         *
C     *                                                               *
C     *   INPUT PARAMETERS                                            *
C     *                                                               *
C     *       RLEN   - THE EQUIVALENT LENGTH OF A RAILROAD CROSSING   *
C     *       CLEN   - THE EQUIVALENT LENGTH OF A CREEK CROSSING      *
C     *       N      - THE NUMBER OF NODES                            *
C     *       X      - THE X-COORDINATE OF EACH NODE (m)              *
C     *       Y      - THE Y-COORDINATE OF EACH NODE (m)              *
C     *       IRSET  - THE RAILROAD SET                               *
C     *       ICSET  - THE RIVER/CREEK SET                            *
C     *                                                               *
C     *                                                               *
C     *   OUTPUT PARAMETERS                                           *
C     *                                                               *
C     *       NL     - THE NUMBER OF LINES                            *
C     *       IFR    - THE ORIGIN NODE NUMBER                         *
C     *       ITO    - THE DESTINATION NODE NUMBER                    *
C     *       ALEN   - THE LENGTH OF A LINE (m)                       *
C     *       X1     - THE X-COORDINATE OF ORIGIN (m)                 *
C     *       Y1     - THE Y-COORDINATE OF ORIGIN (m)                 *
C     *       X2     - THE X-COORDINATE OF DESTINATION (m)            *
C     *       Y2     - THE Y-COORDINATE OF DESTINATION (m)            *
C     *       IRAIL  - THE NUMBER OF RAIL CROSSINGS                   *
C     *       ICREEK - THE NUMBER OF CREEK CROSSINGS                  *
C     *                                                               *
C     *                                                               *
C     *   LOCAL VARIABLES                                             *
C     *                                                               *
C     *       B       - THE TABLE OF WEIGHTED LENGTHS (m)             *
C     *       ICOLS   - THE LIST OF COLUMNS (NODES) IN THE TREE       *
C     *       HIGH    - THE LONGEST LENGTH (m)                        *
C     *       ALOW    - THE SHORTEST LENGTH FOUND IN A SEARCH (m)     *
C     *       IL,JL   - THE INDICES OF LINE WITH THE SHORTEST LENGTH  *
C     *       I,J,K,L - COUNTERS                                      *
C     *                                                               *
```

```
C     ****************************************************************
C     INPUT PARAMETERS
      INTEGER N,IRSET(200),ICSET(200)
      REAL RLEN,CLEN,X(200),Y(200)
C     OUTPUT PARAMETERS
      INTEGER NL,IFR(200),ITO(200) ,IRAIL,ICREEK
      REAL ALEN(200),X1(200),Y1(200),X2(200),Y2(200)
C     LOCAL VARIABLES
      INTEGER ICOLS(200),IL,JL,I,J,K,L
      REAL B(200,200),HIGH,ALOW
      IRAIL=0
      ICREEK=0
C     BUILD TABLES
      NL=N-1
      HIGH=0.0
      DO 107 I=1,NL,1
      DO 106 J=I+1,N,1
      B(I,J)=SQRT((ABS(X(I)-X(J))**2.0)+(ABS(Y(I)-Y(J))**2.0))
      IF(IRSET(I)-IRSET(J))100,101,100
  100 CONTINUE
      B(I,J)=B(I,J)+RLEN
  101 CONTINUE
      IF(ICSET(I)-ICSET(J))102,103,102
  102 CONTINUE
      B(I,J)=B(I,J)+CLEN
  103 CONTINUE
      B(J,I)=B(I,J)
      IF(B(I,J)-HIGH)105,105,104
  104 CONTINUE
      HIGH=B(I,J)
  105 CONTINUE
  106 CONTINUE
  107 CONTINUE
      HIGH=HIGH+1.0
C     SET DIAGONAL TO HIGH
      DO 108 I=1,N,1
      B(I,I)=HIGH
  108 CONTINUE
C     START SEARCH IN COLUMN 1
      ICOLS(1)=1
      DO 118 L=1,NL,1
C     WIPE OUT SELECTED ROW
      I=ICOLS(L)
      DO 109 J=1,N,1
      B(I,J)=HIGH
  109 CONTINUE
C     INITIALIZE FOR SEARCH
      ALOW=B(1,1)
      IL=1
      JL=1
C     BEGIN SEARCH FOR SMALLEST DISTANCE
      DO 113 K=1,L,1
      J=ICOLS(K)
      DO 112 I=1,N,1
      IF (B(I,J)-ALOW) 110,111,111
  110 CONTINUE
      ALOW=B(I,J)
      IL=I
      JL=J
  111 CONTINUE
  112 CONTINUE
  113 CONTINUE
      IFR(L)=JL
      ITO(L)=IL
      ALEN(L)=B(IL,JL)
      IF(IRSET(JL)-IRSET(IL))114,115,114
  114 CONTINUE
      ALEN(L)=ALEN(L)-RLEN
      IRAIL=IRAIL+1
  115 CONTINUE
      IF(ICSET(JL)-ICSET(IL))116,117,116
  116 CONTINUE
      ALEN(L)=ALEN(L)-CLEN
      ICREEK=ICREEK+1
  117 CONTINUE
      X1(L)=X(JL)
      Y1(L)=Y(JL)
      X2(L)=X(IL)
      Y2(L)=Y(IL)
      ICOLS(L+1)=IL
  118 CONTINUE
      RETURN
      END
      SUBROUTINE LEVEL (NL,IFR,ITO,ALEN,LMAX,LEV,DMAX,DIST)
C     *****************************************************************
C     *                                                               *
C     *   THIS SUBROUTINE CALCULATES THE LEVEL OF EACH LINE           *
C     *   IN THE TREE AND THE DISTANCE OF THE DESTINATION NODE        *
C     *   FROM THE SOURCE.                                            *
C     *                                                               *
C     *   VARIABLE DICTIONARY                                         *
C     *                                                               *
C     *   INPUT PARAMETERS                                            *
C     *                                                               *
C     *       NL      - THE NUMBER OF LINES                           *
C     *       IFR     - THE ORIGIN NODE NUMBER                        *
C     *       ITO     - THE DESTINATION NODE NUMBER                   *
C     *       ALEN    - THE LENGTH OF A LINE (m)                      *
C     *                                                               *
C     *                                                               *
C     *   OUTPUT PARAMETERS                                           *
C     *                                                               *
C     *       LMAX    - THE MAXIMUM LEVEL IN THE TREE                 *
C     *       LEV     - THE LEVEL OF A LINE                           *
C     *       DMAX    - THE LENGTH OF THE LONGEST LEG IN THE TREE (m) *
C     *       DIST    - THE DISTANCE FROM DESTINATION TO SOURCE (m)   *
C     *                                                               *
C     *                                                               *
C     *   LOCAL VARIABLES                                             *
C     *                                                               *
C     *       I,J      - COUNTERS                                     *
C     *                                                               *
C     *****************************************************************
C     INPUT PARAMETERS
      INTEGER NL,IFR(200),ITO(200)
      REAL ALEN(200)
C     OUTPUT PARAMETERS
      INTEGER LMAX,LEV(200)
      REAL DMAX,DIST(200)
C     LOCAL VARIABLES
      INTEGER I,J
C     FIND THE CORRESPONDING LEVELS OF ALL LINES
C     FIND THE MAXIMUM LEVEL
      LMAX=0
      DMAX=0
      DO 100 I=1,NL,1
      IF(IFR(I)-1)102,101,102
  101 CONTINUE
      LEV(I)=1
      DIST(I)=ALEN(I)
      GO TO 103
  102 CONTINUE
      J=1
  104 CONTINUE
      IF(IFR(I)-ITO(J))105,106,105
```

```
106 CONTINUE
    LEV(I)=LEV(J)+1
    DIST(I)=DIST(J)+ALEN(I)
    GO TO 103
105 CONTINUE
    J=J+1
    GO TO 104
103 CONTINUE
    IF(LEV(I)-LMAX)107,107,108
108 CONTINUE
    LMAX=LEV(I)
107 CONTINUE
    IF(DIST(I)-DMAX)109,109,110
110 CONTINUE
    DMAX=DIST(I)
109 CONTINUE
100 CONTINUE
    RETURN
    END
```

```
      SUBROUTINE LOAD (NUMN,QIN,QCN,QDN,NL,IFR,ITO,LMAX,LEV,NUML,QIL,
     *QCL,QDL)
C     ******************************************************************
C     *                                                                *
C     *   THIS SUBROUTINE FINDS THE NUMBER OF LOADS AND THE            *
C     *   LOAD ON THE LINES IN THE TREE.                               *
C     *                                                                *
C     *   VARIABLE DICTIONARY                                          *
C     *                                                                *
C     *   INPUT PARAMETERS                                             *
C     *                                                                *
C     *       NUMN   - THE NUMBER OF INTERMITTENT LOADS                *
C     *       QIN    - THE INTERMITTENT LOAD (m^3/hr)                  *
C     *       QCN    - THE CONTINUOUS LOAD (m^3/hr)                    *
C     *       QDN    - THE DRYER LOAD (m^3/hr)                         *
C     *       NL     - THE NUMBER OF LINES                             *
C     *       IFR    - THE ORIGIN NODE NUMBER                          *
C     *       ITO    - THE DESTINATION NODE NUMBER                     *
C     *       LMAX   - THE MAXIMUM LEVEL IN THE TREE                   *
C     *       LEV    - THE LEVEL OF A LINE                             *
C     *                                                                *
C     *                                                                *
C     *   OUTPUT PARAMETERS                                            *
C     *                                                                *
C     *       NUML   - THE NUMBER OF INTERMITTENT LOADS                *
C     *       QIL    - THE INTERMITTENT LOAD (m^3/hr)                  *
C     *       QCL    - THE CONTINUOUS LOAD (m^3/hr)                    *
C     *       QDL    - THE DRYER LOAD (m^3/hr)                         *
C     *                                                                *
C     *                                                                *
C     *   LOCAL VARIABLES                                              *
C     *                                                                *
C     *       I,J,K  - COUNTERS                                        *
C     *                                                                *
C     ******************************************************************
C     INPUT PARAMETERS
      INTEGER NUMN(200),NL,IFR(200),ITO(200),LMAX,LEV(200)
      REAL QIN(200),QCN(200),QDN(200)
C     OUTPUT PARAMETERS
      INTEGER NUML(200)
      REAL QIL(200),QCL(200),QDL(200)
C     LOCAL VARIABLES
      INTEGER I,J,K
C     FIND LOAD FOR EACH LINE
      DO 100 I=LMAX,1,-1
      DO 101 J=1,NL,1
      IF(LEV(J)-I)102,103,102
  103 CONTINUE
      NUML(J)=NUMN(ITO(J))
      QIL(J)=QIN(ITO(J))
      QCL(J)=QCN(ITO(J))
      QDL(J)=QDN(ITO(J))
      DO 104 K=1,NL,1
      IF(IFR(K)-ITO(J))106,105,106
  105 CONTINUE
      NUML(J)=NUML(J)+NUML(K)
      QIL(J)=QIL(J)+QIL(K)
      QCL(J)=QCL(J)+QCL(K)
      QDL(J)=QDL(J)+QDL(K)
  106 CONTINUE
  104 CONTINUE
  102 CONTINUE
  101 CONTINUE
  100 CONTINUE
      RETURN
      END
      SUBROUTINE FLOAD (NL,NUML,QIL,QCL,QDL,FC,QTL)
C     ******************************************************************
C     *                                                                *
C     *   THIS SUBROUTINE COMPUTES THE FACTORED LOADS FOR ALL          *
C     *   THE LINES. COINCIDENCE FACTORS ARE OBTAINED FROM A           *
C     *   LOOK-UP TABLE "FLOOKUP".                                     *
C     *                                                                *
C     *   VARIABLE DICTIONARY                                          *
C     *                                                                *
C     *   INPUT PARAMETERS                                             *
C     *                                                                *
C     *       NL     - THE NUMBER OF LINES                             *
C     *       NUML   - THE NUMBER OF INTERMITTENT LOADS                *
C     *       QIL    - THE INTERMITTENT LOAD (m^3/hr)                  *
C     *       QCL    - THE CONTINUOUS LOAD (m^3/hr)                    *
C     *       QDL    - THE DRYER LOAD (m^3/hr)                         *
C     *                                                                *
C     *                                                                *
C     *   OUTPUT PARAMETERS                                            *
C     *                                                                *
C     *       FC     - THE COINCIDENCE FACTOR                          *
C     *       QTL    - THE TOTAL LOAD OR DESIGN LOAD (m^3/hr)          *
C     *                                                                *
C     *                                                                *
C     *   LOCAL VARIABLES                                              *
C     *                                                                *
C     *       FACT   - THE TABLE OF COINCIDENCE FACTORS                *
C     *       QWL    - THE WINTER LOAD OF A LINE                       *
C     *       QSL    - THE SUMMER LOAD OF A LINE                       *
C     *       I,J    - COUNTERS                                        *
C     *                                                                *
C     ******************************************************************
C     INPUT PARAMETERS
      INTEGER NL,NUML(200)
      REAL QIL(200),QCL(200),QDL(200)
C     OUTPUT PARAMETERS
      REAL FC(200),QTL(200)
C     LOCAL VARIABLES
      REAL FACT(36),QWL,QSL
      INTEGER I,J
C     READ LOOKUP TABLES
      OPEN(UNIT=1,FILE='FLOOKUP')
      DO 100 I=1,36,1
      READ(1,*)FACT(I)
  100 CONTINUE
      CLOSE(UNIT=1)
C     FACTOR THE LOAD OF EACH LINE
      DO 101 I=1,NL,1
      IF(NUML(I)-0)102,102,103
  102 CONTINUE
      J=1
      GO TO 106
```

```
103 CONTINUE
    IF(NUML(I)-36)105,105,104
104 CONTINUE
    J=36
    GO TO 106
105 CONTINUE
    J=NUML(I)
106 CONTINUE
    FC(I)=FACT(J)
    QWL=(QIL(I)*FC(I))+QCL(I)
    QSL=(0.4*FC(I)*QIL(I))+QCL(I)+QDL(I)
    IF(QSL-QWL)107,107,108
107 CONTINUE
    QTL(I)=QWL
    GO TO 109
108 CONTINUE
    QTL(I)=QSL
109 CONTINUE
101 CONTINUE
    RETURN
    END
```

```
      SUBROUTINE CHECK1 (NL,QTL,FAIL1,FAIL2)
C     ******************************************************************
C     *                                                                *
C     *   THIS SUBROUTINE CHECKS TO SEE IF A TERMINAL DUMMY NODE       *
C     *   HAS BEEN CREATED. IF A LINE IS FOUND WITH NO LOAD, THE       *
C     *   FLAG FAIL IS RETURNED AS TRUE.                               *
C     *                                                                *
C     *                                                                *
C     *   VARIABLE DICTIONARY                                          *
C     *                                                                *
C     *   INPUT PARAMETERS                                             *
C     *                                                                *
C     *       QTL     - THE TOTAL LOAD OR DESIGN LOAD (m^3/hr)         *
C     *       NL      - THE NUMBER OF LINES                            *
C     *                                                                *
C     *                                                                *
C     *   OUTPUT PARAMETERS                                            *
C     *                                                                *
C     *       FAIL1   - FLAG ( TRUE IF LAYOUT FAILS )                  *
C     *       FAIL2   - FLAG ( TRUE IF MAX. CAPACITY EXCEEDED )        *
C     *                                                                *
C     *                                                                *
C     *   LOCAL VARIABLES                                              *
C     *                                                                *
C     *       I       - A COUNTER                                      *
C     *                                                                *
C     ******************************************************************
C     INPUT PARAMETERS
      INTEGER NL
      REAL QTL(200)
C     OUTPUT PARAMETERS
      LOGICAL FAIL1,FAIL2
C     LOCAL VARIABLES
      INTEGER I
      FAIL1=.FALSE.
      FAIL2=.FALSE.
      DO 100 I=1,NL,1
      IF (QTL(I)-0.0)101,101,102
  101 CONTINUE
      FAIL1=.TRUE.
  102 CONTINUE
      IF (QTL(I)-600.0)103,103,104
  104 CONTINUE
      FAIL2=.TRUE.
  103 CONTINUE
  100 CONTINUE
      RETURN
      END
      SUBROUTINE MAXPD (DMAX,PDMAX)
C     ******************************************************************
C     *                                                                *
C     *   THIS SUBROUTINE DISPLAYS THE OPTIMUM PRESSURE DROP           *
C     *   PER KILOMETER AND GETS THE OVERRIDE FROM THE KEYBOARD.       *
C     *                                                                *
C     *   VARIABLE DICTIONARY                                          *
C     *                                                                *
C     *   INPUT PARAMETERS                                             *
C     *                                                                *
C     *       DMAX   - THE LENGTH OF THE LONGEST LEG IN THE TREE (m)   *
C     *                                                                *
C     *                                                                *
C     *   OUTPUT PARAMETERS                                            *
C     *                                                                *
C     *       PDMAX  - THE OPTIMUM PRESSURE DROP PER KM (kPa/m)        *
C     *                                                                *
C     ******************************************************************
C     INPUT PARAMETERS
      REAL DMAX
C     OUTPUT PARAMETERS
      REAL PDMAX
      OPEN(UNIT=5,FILE='TERMINAL')
      OPEN(UNIT=6,FILE='(C)TERMINAL')
      PDMAX=630000.0/DMAX
      IF(PDMAX-20.0)100,100,101
  101 CONTINUE
      PDMAX=20.0
  100 CONTINUE
      WRITE(6,150)DMAX
      WRITE(6,151)PDMAX
      READ(5,*)PDMAX
      PDMAX=PDMAX/1000.0
      CLOSE(UNIT=6)
      CLOSE(UNIT=5)
      RETURN
  150 FORMAT(' THE LONGEST LEG IS ',F8.1,' m.')
  151 FORMAT(' THE RECOMMENDED PRESSURE DROP PER KM = ',F4.1,' kPa'/
     *' INPUT THE DESIRED PRESSURE DROP PER KM IN KILOPASCALS')
      END
      SUBROUTINE PRESS (NL,IFR,ITO,ALEN,LMAX,LEV,QTL,PDMAX,PIN,POUT,PD,
     *SIZE,IWARN1,IWARN2)
C     ******************************************************************
C     *                                                                *
C     *   THIS PROGRAM CALCULATES THE APPROPRIATE DIAMETER             *
C     *   AS WELL AS THE INLET AND OUTLET PRESSURES FOR EACH           *
C     *   LINE SEGMENT. THE CALCULATIONS ARE PERFORMED WITH            *
C     *   THE AID OF A LOOK-UP TABLE OF GAS FLOW FORMULA               *
C     *   CONSTANTS.                                                   *
C     *                                                                *
C     *   VARIABLE DICTIONARY                                          *
C     *                                                                *
C     *   INPUT PARAMETERS                                             *
C     *                                                                *
C     *       NL      - THE NUMBER OF LINES                            *
C     *       IFR     - THE ORIGIN NODE NUMBER                         *
C     *       ITO     - THE DESTINATION NODE NUMBER                    *
C     *       ALEN    - THE LENGTH OF A LINE (m)                       *
C     *       LMAX    - THE MAXIMUM LEVEL IN THE TREE                  *
C     *       LEV     - THE LEVEL OF A LINE                            *
C     *       PDMAX   - THE OPTIMUM PRESSURE DROP PER KM (kPa/m)       *
C     *       QTL     - THE TOTAL LOAD OR DESIGN LOAD (m^3/hr)         *
C     *                                                                *
C     *                                                                *
C     *   OUTPUT PARAMETERS                                            *
C     *                                                                *
C     *       PIN     - THE INLET PRESSURE (kPa)                       *
C     *       POUT    - THE OUTLET PRESSURE (kPa)                      *
C     *       PD      - THE PRESSURE DROP PER KILOMETER (kPa/km)       *
C     *       SIZE    - THE DIAMETER OF A PIPE SEGMENT (mm)            *
C     *       IWARN1  - OUTLET PRESSURE BELOW 140 kPa                  *
C     *       IWARN2  - PRESSURE DROP OF 20 kPa/km EXCEEDED            *
C     *                                                                *
C     *                                                                *
C     *   LOCAL VARIABLES                                              *
C     *                                                                *
C     *       AK      - THE FLOW FORMULA CONSTANTS                     *
C     *       IQ      - THE INDEX TO THE TABLE OF FLOW CONSTANTS       *
C     *       PIA     - THE ABSOLUTE INLET PRESSURE                    *
C     *       AKMIN   - THE MINIMUM ALLOWABLE FLOW CONSTANT            *
C     *       I,J,K   - COUNTERS                                       *
C     *                                                                *
C     ******************************************************************
C     INPUT PARAMETERS
      INTEGER NL,IFR(200),ITO(200),LMAX,LEV(200)
      REAL ALEN(200),QTL(200),PDMAX
C     OUTPUT PARAMETERS
      INTEGER IWARN1(200),IWARN2(200)
      REAL PIN(200),POUT(200),PD(200),SIZE(200)
C     LOCAL VARIABLES
      INTEGER IQ,I,J,K
      REAL AK(120,5),PIA,AKMIN
```

```
C       READ IN LOOK-UP TABLE
        OPEN(UNIT=1,FILE='KLOOKUP')
        DO 100 I=1,120,1
        READ(1,*)(AK(I,J),J=1,5)
   100  CONTINUE
        CLOSE(UNIT=1)
C       SET WARNINGS TO 0
        DO 101 I=1,NL,1
        IWARN1(I)=0
        IWARN2(I)=0
   101  CONTINUE
C       DESIGN PIPES AT EACH LEVEL
        DO 102 I=1,LMAX,1
C       FIND ALL PIPES AT LEVEL I
        DO 103 J=1,NL,1
        IF(LEV(J)-I)104,105,104
   105  CONTINUE
C       FIND INLET PRESSURE
        IF(LEV(J)-1)107,106,107
   106  CONTINUE
        PIN(J)=550.0
        GO TO 108
   107  CONTINUE
        K=1
   109  CONTINUE
        IF(ITO(K)-IFR(J))110,111,110
   110  CONTINUE
        K=K+1
        GO TO 109
   111  CONTINUE
        PIN(J)=POUT(K)
C       CHECK TO SEE IF INLET PRESSURE IS ZERO
        IF(PIN(J)-0.0)117,117,108
   108  CONTINUE
C       FIND THE MINIMUM K REQUIRED
        IQ=INT((QTL(J)/5.0)+0.5)
        PIA=PIN(J)+94.1
        AKMIN=QTL(J)*SQRT(ALEN(J)/((PIA**2)-((PIA-(PDMAX*ALEN(J)))**2)))
C       FIND MINIMUM K POSSIBLE
        K=1
   112  CONTINUE
        IF(AK(IQ,K)-AKMIN)113,114,114
   113  CONTINUE
        IF(K-5)116,115,115
   115  CONTINUE
        IWARN2(J)=1
        GO TO 114
   116  CONTINUE
        K=K+1
        GO TO 112
   114  CONTINUE
C       CALCULATE OUTLET PRESSURE
```

```
      POUT(J)=SQRT((PIA**2)-(((QTL(J)/AK(IQ,K))**2)*ALEN(J)))-94.1
C     CHECK TO SEE IF OUTLET PRESSURE IS SUFFICIENT
      IF(POUT(J)-140.0)117,118,118
  117 CONTINUE
      POUT(J)=0.0
      PD(J)=0.0
      SIZE(J)=0.0
      IWARN1(J)=1
      GO TO 119
  118 CONTINUE
C     CALCULATE PRESSURE DROP PER KM
      PD(J)=(PIN(J)-POUT(J))/(ALEN(J)/1000)
C     PICK PIPE SIZE
      IF (K-1)120,120,121
  120 CONTINUE
      SIZE(J)=26.7
      GO TO 119
  121 CONTINUE
      IF(K-2)122,122,123
  122 CONTINUE
      SIZE(J)=33.4
      GO TO 119
  123 CONTINUE
      IF(K-3)124,124,125
  124 CONTINUE
      SIZE(J)=48.3
      GO TO 119
  125 CONTINUE
      IF(K-4)126,126,127
  126 CONTINUE
      SIZE(J)=60.3
      GO TO 119
  127 CONTINUE
      SIZE(J)=88.9
  119 CONTINUE
  104 CONTINUE
  103 CONTINUE
  102 CONTINUE
      RETURN
      END
      SUBROUTINE CHECK2 (NL,IWARN1,IWARN2,FAIL3,FAIL4)
C     ******************************************************************
C     *                                                                *
C     *   THIS SUBROUTINE CHECKS TO SEE IF SUFFICIENT END              *
C     *   PRESSURE HAS BEEN MAINTAINED OR THE MAXIMUM PRESSURE         *
C     *   DROP CONSTRAINT OF 20 kPa PER km HAS BEEN VIOLATED.          *
C     *                                                                *
C     *                                                                *
C     *   INPUT PARAMETERS                                             *
C     *                                                                *
C     *       NL     - THE NUMBER OF LINES                             *
C     *       IWARN1 - OUTLET PRESSURE BELOW 140 kPa                   *
C     *       IWARN2 - PRESSURE DROP OF 20 kPa/km EXCEEDED             *
C     *                                                                *
C     *                                                                *
C     *   OUTPUT PARAMETERS                                            *
C     *                                                                *
C     *       FAIL3  - FLAG ( TRUE IF INSUFFICIENT END PRESSURE)       *
C     *       FAIL4  - FLAG ( TRUE IF PRESSURE DROP .GT. 20 kPa/km)    *
C     *                                                                *
C     *                                                                *
C     *   LOCAL VARIABLES                                              *
C     *                                                                *
C     *       I      - A COUNTER                                       *
C     *                                                                *
C     ******************************************************************
C     INPUT PARAMETERS
      INTEGER NL,IWARN1(200),IWARN2(200)
C     OUTPUT PARAMETERS
      LOGICAL FAIL3,FAIL4
C     LOCAL VARIABLES
      INTEGER I
      FAIL3=.FALSE.
      FAIL4=.FALSE.
      DO 104 I=1,NL,1
      IF(IWARN1(I)-0)100,101,100
  100 CONTINUE
      FAIL3=.TRUE.
  101 CONTINUE
      IF(IWARN2(I)-0)102,103,102
  102 CONTINUE
      FAIL4=.TRUE.
  103 CONTINUE
  104 CONTINUE
      RETURN
      END
```

```
       SUBROUTINE TOTAL (RPERKM,CPERKM,NL,ALEN,SIZE,AL267,AL334,AL483,
      *AL603,AL889,IROAD,ICABLE,ALTOT)
C      *****************************************************************
C      *                                                               *
C      *   THIS SUBROUTINE FIND THE TOTAL LENGTHS OF PIPE USED         *
C      *   AND THE NUMBERS OF ROAD AND CABLE CROSSINGS.                *
C      *                                                               *
C      *                                                               *
C      *   VARIABLE DICTIONARY                                         *
C      *                                                               *
C      *                                                               *
C      *   INPUT PARAMETERS                                            *
C      *                                                               *
C      *       RPERKM - THE NUMBER OF ROAD CROSSINGS PER KILOMETER     *
C      *       CPERKM - THE NUMBER OF CABLE CROSSINGS PER KILOMETER    *
C      *       NL     - THE NUMBER OF LINES                            *
C      *       ALEN   - THE LENGTH OF A LINE (m)                       *
C      *       SIZE   - THE DIAMETER OF A PIPE SEGMENT (mm)            *
C      *                                                               *
C      *                                                               *
C      *   OUTPUT PARAMETERS                                           *
C      *                                                               *
C      *       AL267  - THE TOTAL LENGTH OF 26.7 P.E. PIPE (m)         *
C      *       AL334  - THE TOTAL LENGTH OF 33.4 P.E. PIPE (m)         *
C      *       AL483  - THE TOTAL LENGTH OF 48.3 P.E. PIPE (m)         *
C      *       AL603  - THE TOTAL LENGTH OF 60.3 P.E. PIPE (m)         *
C      *       AL889  - THE TOTAL LENGTH OF 88.9 P.E. PIPE (m)         *
C      *       IROAD  - THE NUMBER OF ROAD CROSSINGS                   *
C      *       ICABLE - THE NUMBER OF CABLE CROSSINGS                  *
C      *       ALTOT  - THE TOTAL LENGTH OF PIPE (m)                   *
C      *                                                               *
C      *   LOCAL VARIABLES                                             *
C      *       AKM    - TOTAL LENGTH OF PIPE (km)                      *
C      *       I      - A COUNTER                                      *
C      *                                                               *
C      *****************************************************************
C      INPUT PARAMETERS
       INTEGER NL
       REAL RPERKM,CPERKM,ALEN(200),SIZE(200)
C      OUTPUT PARAMETERS
       INTEGER IROAD,ICABLE
       REAL AL267,AL334,AL483,AL603,AL889,ALTOT
C      LOCAL VARIABLES
       INTEGER I
       REAL AKM
       AL267=0.0
       AL334=0.0
       AL483=0.0
       AL603=0.0
       AL889=0.0
       DO 109 I=1,NL,1
       IF(SIZE(I)-26.7)100,100,101
```

```
100 CONTINUE
    AL267=AL267+ALEN(I)
    GO TO 108
101 CONTINUE
    IF(SIZE(I)-33.4)102,102,103
102 CONTINUE
    AL334=AL334+ALEN(I)
    GO TO 108
103 CONTINUE
    IF(SIZE(I)-48.3)104,104,105
104 CONTINUE
    AL483=AL483+ALEN(I)
    GO TO 108
105 CONTINUE
    IF(SIZE(I)-60.3)106,106,107
106 CONTINUE
    AL603=AL603+ALEN(I)
    GO TO 108
107 CONTINUE
    AL889=AL889+ALEN(I)
108 CONTINUE
109 CONTINUE
    ALTOT=AL267+AL334+AL483+AL603+AL889
    AKM=ALTOT/1000.0
    IROAD=INT(RPERKM*AKM)
    ICABLE=INT(CPERKM*AKM)
    RETURN
    END
```

```
      SUBROUTINE COSTS (UC267,UC334,UC483,UC603,UC889,UCROAD,UCRAIL,
     *UCABLE,UCREEK,AL267,AL334,AL483,AL603,AL889,IROAD,IRAIL,ICABLE,
     *ICREEK,C267,C334,C483,C603,C889,CPIPE,CROAD,CRAIL,CCABLE,CCREEK,
     *CTOT)
C     *****************************************************************
C     *                                                               *
C     *   THIS SUBROUTINE TOTALS THE COSTS FOR THE SYSTEM             *
C     *                                                               *
C     *                                                               *
C     *   VARIABLE DICTIONARY                                         *
C     *                                                               *
C     *                                                               *
C     *   INPUT PARAMETERS                                            *
C     *                                                               *
C     *       UC267  - THE UNIT COST OF 26.7 P.E. PIPE ($/m)          *
C     *       UC334  - THE UNIT COST OF 33.4 P.E. PIPE ($/m)          *
C     *       UC483  - THE UNIT COST OF 48.3 P.E. PIPE ($/m)          *
C     *       UC603  - THE UNIT COST OF 60.3 P.E. PIPE ($/m)          *
C     *       UC889  - THE UNIT COST OF 88.9 P.E. PIPE ($/m)          *
C     *       UCROAD - THE COST OF A ROAD CROSSING ($)                *
C     *       UCRAIL - THE COST OF A RAILROAD CROSSING ($)            *
C     *       UCABLE - THE COST OF A CABLE CROSSING ($)               *
C     *       UCREEK - THE COST OF A CREEK/RIVER CROSSING ($)         *
C     *       AL267  - THE TOTAL LENGTH OF 26.7 P.E. PIPE (m)         *
C     *       AL334  - THE TOTAL LENGTH OF 33.4 P.E. PIPE (m)         *
C     *       AL483  - THE TOTAL LENGTH OF 48.3 P.E. PIPE (m)         *
C     *       AL603  - THE TOTAL LENGTH OF 60.3 P.E. PIPE (m)         *
C     *       AL889  - THE TOTAL LENGTH OF 88.9 P.E. PIPE (m)         *
C     *       IROAD  - THE NUMBER OF ROAD CROSSINGS                   *
C     *       IRAIL  - THE NUMBER OF RAIL CROSSINGS                   *
C     *       ICABLE - THE NUMBER OF CABLE CROSSINGS                  *
C     *       ICREEK - THE NUMBER OF CREEK CROSSINGS                  *
C     *                                                               *
C     *                                                               *
C     *   OUTPUT PARAMETERS                                           *
C     *                                                               *
C     *       C267   - THE COST OF 26.7 P.E. PIPE ($)                 *
C     *       C334   - THE COST OF 33.4 P.E. PIPE ($)                 *
C     *       C483   - THE COST OF 48.3 P.E. PIPE ($)                 *
C     *       C603   - THE COST OF 60.3 P.E. PIPE ($)                 *
C     *       C889   - THE COST OF 88.9 P.E. PIPE ($)                 *
C     *       CPIPE  - THE TOTAL COST OF ALL PIPE ($)                 *
C     *       CROAD  - THE COST OF ALL ROAD CROSSINGS ($)             *
C     *       CRAIL  - THE COST OF ALL RAIL CROSSINGS ($)             *
C     *       CCABLE - THE COST OF ALL CABLE CROSSINGS ($)            *
C     *       CCREEK - THE COST OF ALL CREEK CROSSINGS ($)            *
C     *       CTOT   - THE TOTAL COST OF THE SYSTEM ($)               *
C     *                                                               *
C     *****************************************************************
C     INPUT PARAMETERS
      INTEGER IROAD,IRAIL,ICABLE,ICREEK
      REAL UC267,UC334,UC483,UC603,UC889,UCROAD,UCRAIL,UCABLE,UCREEK
      REAL AL267,AL334,AL483,AL603,AL889
C     OUTPUT PARAMETERS
      REAL C267,C334,C483,C603,C889,CPIPE
      REAL CROAD,CRAIL,CCABLE,CCREEK,CTOT
      C267=AL267*UC267
      C334=AL334*UC334
      C483=AL483*UC483
      C603=AL603*UC603
      C889=AL889*UC889
      CPIPE=C267+C334+C483+C603+C889
      CROAD=IROAD*UCROAD
      CRAIL=IRAIL*UCRAIL
      CCABLE=ICABLE*UCABLE
      CCREEK=ICREEK*UCREEK
      CTOT=CPIPE+CROAD+CRAIL+CCABLE+CCREEK
      RETURN
      END
      SUBROUTINE LNDRIT (NL,IFR,ITO,ALEN,NUML,QTL,PIN,POUT,PD,SIZE,
     *IWARN1,IWARN2,AL267,AL334,AL483,AL603,AL889,IROAD,IRAIL,ICABLE,
     *ICREEK,ALTOT,C267,C334,C483,C603,C889,CPIPE,CROAD,CRAIL,CCABLE,
     *CCREEK,CTOT)
C      *****************************************************************
C      *                                                               *
C      *   THIS SUBROUTINE CREATES THE "LINES.DAT" FILE WHICH          *
C      *   IS PRINTED BY THE "LINEDATA" AUTOCAD MACRO.                 *
C      *                                                               *
C      *                                                               *
C      *   VARIABLE DICTIONARY                                         *
C      *                                                               *
C      *   INPUT PARAMETERS                                            *
C      *                                                               *
C      *       NL     - THE NUMBER OF LINES                            *
C      *       IFR    - THE ORIGIN NODE NUMBER                         *
C      *       ITO    - THE DESTINATION NODE NUMBER                    *
C      *       ALEN   - THE LENGTH OF A LINE (m)                       *
C      *       NUML   - THE NUMBER OF INTERMITTENT LOADS               *
C      *       QTL    - THE TOTAL LOAD OR DESIGN LOAD (m^3/hr)         *
C      *       PIN    - THE INLET PRESSURE (kPa)                       *
C      *       POUT   - THE OUTLET PRESSURE (kPa)                      *
C      *       PD     - THE PRESSURE DROP PER KILOMETER (kPa/km)       *
C      *       SIZE   - THE DIAMETER OF A PIPE SEGMENT (mm)            *
C      *       IWARN1 - OUTLET PRESSURE BELOW 140 kPa                  *
C      *       IWARN2 - PRESSURE DROP OF 20 kPa/km EXCEEDED            *
C      *       AL267  - THE TOTAL LENGTH OF 26.7 P.E. PIPE (m)         *
C      *       AL334  - THE TOTAL LENGTH OF 33.4 P.E. PIPE (m)         *
C      *       AL483  - THE TOTAL LENGTH OF 48.3 P.E. PIPE (m)         *
C      *       AL603  - THE TOTAL LENGTH OF 60.3 P.E. PIPE (m)         *
C      *       AL889  - THE TOTAL LENGTH OF 88.9 P.E. PIPE (m)         *
C      *       IROAD  - THE NUMBER OF ROAD CROSSINGS                   *
C      *       IRAIL  - THE NUMBER OF RAIL CROSSINGS                   *
C      *       ICABLE - THE NUMBER OF CABLE CROSSINGS                  *
C      *       ICREEK - THE NUMBER OF CREEK CROSSINGS                  *
C      *       ALTOT  - THE TOTAL LENGTH OF PIPE (m)                   *
C      *       C267   - THE COST OF 26.7 P.E. PIPE ($)                 *
C      *       C334   - THE COST OF 33.4 P.E. PIPE ($)                 *
C      *       C483   - THE COST OF 48.3 P.E. PIPE ($)                 *
C      *       C603   - THE COST OF 60.3 P.E. PIPE ($)                 *
C      *       C889   - THE COST OF 88.9 P.E. PIPE ($)                 *
C      *       CPIPE  - THE TOTAL COST OF ALL PIPE ($)                 *
C      *       CROAD  - THE COST OF ALL ROAD CROSSINGS ($)             *
C      *       CRAIL  - THE COST OF ALL RAIL CROSSINGS ($)             *
C      *       CCABLE - THE COST OF ALL CABLE CROSSINGS ($)            *
C      *       CCREEK - THE COST OF ALL CREEK CROSSINGS ($)            *
C      *       CTOT   - THE TOTAL COST OF THE SYSTEM ($)               *
C      *                                                               *
C      *                                                               *
C      *   LOCAL VARIABLES                                             *
C      *                                                               *
C      *       I,J    - COUNTERS                                       *
```

```
C      *                                                                *
C      ******************************************************************
C      INPUT PARAMETERS
       INTEGER  NL,IFR(200),ITO(200),NUML(200),IWARN1(200),IWARN2(200)
       REAL ALEN(200),QTL(200),PIN(200),POUT(200),PD(200),SIZE(200)
       INTEGER IROAD,IRAIL,ICABLE,ICREEK
       REAL AL267,AL334,AL483,AL603,AL889,ALTOT,C267,C334,C483,C603
       REAL C889,CPIPE,CROAD,CRAIL,CCABLE,CCREEK,CTOT
C      LOCAL VARIABLES
       INTEGER I,J
       OPEN(UNIT=1,FILE='(C)LINES.DAT')
       I=45
       DO 104 J=1,NL,1
       IF (I-45)101,100,100
   100 CONTINUE
C      TYPE HEADING
       WRITE(1,151)
       I=0
   101 CONTINUE
C      TYPE ONE LINE
       WRITE(1,152)IFR(J),ITO(J),ALEN(J),NUML(J),QTL(J),SIZE(J),PIN(J),
      *POUT(J),PD(J),IWARN1(J),IWARN2(J)
       I=I+1
       IF(I-45)103,102,102
   102 CONTINUE
       WRITE(1,150)
   103 CONTINUE
   104 CONTINUE
       WRITE(1,153)
       WRITE(1,154)AL267,C267
       WRITE(1,155)AL334,C334
       WRITE(1,156)AL483,C483
       WRITE(1,157)AL603,C603
       WRITE(1,158)AL889,C889
       WRITE(1,159)ALTOT,CPIPE
       WRITE(1,160)IROAD,CROAD
       WRITE(1,161)IRAIL,CRAIL
       WRITE(1,162)ICABLE,CCABLE
       WRITE(1,163)ICREEK,CCREEK
       WRITE(1,164)CTOT
       CLOSE(UNIT=1)
       RETURN
   150 FORMAT('1')
   151 FORMAT(' FROM   TO   LENGTH   NUM  LOAD   SIZE    PI     PO',
      *'      PD    WO    WD'/' ****   ****  *******  ***   *****  ****',
      *'  *****  *****  *****  ****  ****')
   152 FORMAT(' ',2(I4,2X),F7.1,2X,I3,2X,F5.1,2X,F4.1,3(2X,F5.1),4X,I1,
      *5X,I1)
   153 FORMAT('1PIPE SIZE     TOTAL LENGTH      COST'/
      *' *********     ************     *******')
   154 FORMAT(' 26.7  mm',8X,F7.0,8X,F8.0)
   155 FORMAT(' 33.4  mm',8X,F7.0,8X,F8.0)
```

```
156 FORMAT(' 48.3  mm',8X,F7.0,8X,F8.0)
157 FORMAT(' 60.3  mm',8X,F7.0,8X,F8.0)
158 FORMAT(' 88.9  mm',8X,F7.0,8X,F8.0/)
159 FORMAT(' TOTAL    ',8X,F7.0,8X,F8.0//)
160 FORMAT(' TOTAL NUMBER OF ROAD  CROSSINGS ',I3,'   COST ',F7.0)
161 FORMAT(' TOTAL NUMBER OF RAIL  CROSSINGS ',I3,'   COST ',F7.0)
162 FORMAT(' TOTAL NUMBER OF CABLE CROSSINGS ',I3,'   COST ',F7.0)
163 FORMAT(' TOTAL NUMBER OF CREEK CROSSINGS ',I3,'   COST ',F7.0//)
164 FORMAT(' TOTAL COST OF SYSTEM  ',F7.0)
    END
```

```
      SUBROUTINE LNLRIT (NL,X1,Y1,X2,Y2,SIZE)
C     ******************************************************************
C     *                                                                *
C     *   THIS SUBROUTINE CREATES THE FILE "LINES.LSP" WHICH IS        *
C     *   USED BY THE AUTOLISP ROUTINES TO PLOT THE SELECTED           *
C     *   LAYOUT AND DESIGN.                                           *
C     *                                                                *
C     *                                                                *
C     *   VARIABLE DICTIONARY                                          *
C     *                                                                *
C     *   INPUT PARAMETERS                                             *
C     *                                                                *
C     *                                                                *
C     *       NL      - THE NUMBER OF LINES                            *
C     *       X1      - THE X-COORDINATE OF ORIGIN (m)                 *
C     *       Y1      - THE Y-COORDINATE OF ORIGIN (m)                 *
C     *       X2      - THE X-COORDINATE OF DESTINATION (m)            *
C     *       Y2      - THE Y-COORDINATE OF DESTINATION (m)            *
C     *       SIZE    - THE DIAMETER OF A PIPE SEGMENT (mm)            *
C     *                                                                *
C     *                                                                *
C     *   LOCAL  VARIABLES                                             *
C     *                                                                *
C     *       I       - A COUNTER                                      *
C     *                                                                *
C     ******************************************************************
C     INPUT PARAMETERS
      INTEGER NL
      REAL X1(200),Y1(200),X2(200),Y2(200),SIZE(200)
C     LOCAL VARIABLES
      INTEGER I
C     WRITE LINES.LSP FILE
      OPEN(UNIT=1,FILE='LINES.LSP')
C     WRITE ALL 88.9 mm LINES
      WRITE(1,150)
      DO 100 I=1,NL,1
      IF (SIZE(I)-88.9)103,102,103
  102 CONTINUE
      WRITE(1,151)X1(I),Y1(I),X2(I),Y2(I)
  103 CONTINUE
  100 CONTINUE
      WRITE(1,152)
C     WRITE ALL 60.3 mm LINES
      WRITE(1,153)
      DO 104 I=1,NL,1
      IF (SIZE(I)-60.3)106,105,106
  105 CONTINUE
      WRITE(1,151)X1(I),Y1(I),X2(I),Y2(I)
  106 CONTINUE
  104 CONTINUE
      WRITE(1,152)
C     WRITE ALL 48.3 mm LINES
```

```
      WRITE(1,154)
      DO 107 I=1,NL,1
      IF (SIZE(I)-48.3)109,108,109
  108 CONTINUE
      WRITE(1,151)X1(I),Y1(I),X2(I),Y2(I)
  109 CONTINUE
  107 CONTINUE
      WRITE(1,152)
C     WRITE ALL 33.4 mm LINES
      WRITE(1,155)
      DO 110 I=1,NL,1
      IF (SIZE(I)-33.4)112,111,112
  111 CONTINUE
      WRITE(1,151)X1(I),Y1(I),X2(I),Y2(I)
  112 CONTINUE
  110 CONTINUE
      WRITE(1,152)
C     WRITE ALL 26.7 mm LINES
      WRITE(1,156)
      DO 113 I=1,NL,1
      IF (SIZE(I)-26.7)115,114,115
  114 CONTINUE
      WRITE(1,151)X1(I),Y1(I),X2(I),Y2(I)
  115 CONTINUE
  113 CONTINUE
      WRITE(1,152)
C     END OF FILE
      CLOSE(UNIT=5)
      RETURN
  150 FORMAT('(setq l889 (quote (')
  151 FORMAT('(( ',F9.1,2X,F9.1,' ) ( ',F9.1,2X,F9.1,' ))')
  152 FORMAT(' )))')
  153 FORMAT('(setq l603 (quote (')
  154 FORMAT('(setq l483 (quote (')
  155 FORMAT('(setq l334 (quote (')
  156 FORMAT('(setq l267 (quote (')
      END
      SUBROUTINE FDRIT (NL,IFR,ITO,ALEN,NUML,QTL)
C     *****************************************************************
C     *                                                               *
C     *   THIS SUBROUTINE CREATES THE "LINES.DAT" FILE WHICH          *
C     *   IS PRINTED BY THE "LINEDATA" AUTOCAD MACRO.                 *
C     *                                                               *
C     *                                                               *
C     *   VARIABLE DICTIONARY                                         *
C     *                                                               *
C     *   INPUT PARAMETERS                                            *
C     *                                                               *
C     *       NL     - THE NUMBER OF LINES                            *
C     *       IFR    - THE ORIGIN NODE NUMBER                         *
C     *       ITO    - THE DESTINATION NODE NUMBER                    *
C     *       ALEN   - THE LENGTH OF A LINE (m)                       *
C     *       NUML   - THE NUMBER OF INTERMITTENT LOADS               *
C     *       QTL    - THE TOTAL LOAD OR DESIGN LOAD (m^3/hr)         *
C     *                                                               *
C     *                                                               *
C     *   LOCAL VARIABLES                                             *
C     *                                                               *
C     *       I,J    - COUNTERS                                       *
C     *                                                               *
C     *****************************************************************
C     INPUT PARAMETERS
      INTEGER  NL,IFR(200),ITO(200),NUML(200)
      REAL ALEN(200),QTL(200)
C     LOCAL VARIABLES
      INTEGER I,J
      OPEN(UNIT=1,FILE='(C)LINES.DAT')
      I=41
      DO 100 J=1,NL,1
      IF (I-41)102,103,103
  103 CONTINUE
C     TYPE HEADING
      WRITE(1,150)NL
      WRITE(1,151)
      I=1
  102 CONTINUE
C     TYPE ONE LINE
      WRITE(1,152)IFR(J),ITO(J),ALEN(J),NUML(J),QTL(J)
      I=I+1
  100 CONTINUE
      CLOSE(UNIT=2)
      RETURN
  150 FORMAT('1 ',I3,' LINES TOTAL'//)
  151 FORMAT(' FROM   TO   LENGTH   NUM  LOAD'
     */' ****  ****  *******  ***  *****')
  152 FORMAT(' ',2(I4,2X),F7.1,2X,I3,2X,F5.1)
      END
      SUBROUTINE FLRIT   (NL,X1,Y1,X2,Y2,FAIL1)
C     ******************************************************************
C     *                                                                *
C     *   THIS SUBROUTINE CREATES THE FILE "LINES.LSP" WHICH IS       *
C     *   USED BY THE AUTOLISP ROUTINES TO PLOT THE SELECTED          *
C     *   LAYOUT AND DESIGN. IN THIS CASE AN ERROR HAS BEEN           *
C     *   FOUND IN THE DESIGN. IN THIS INSTANCE EITHER A LINE         *
C     *   WAS FOUND WITH NO LOAD OR THE MAXIMUM CAPACITY OF THE       *
C     *   SYSTEM HAS BEEN EXCEEDED.                                    *
C     *                                                                *
C     *                                                                *
C     *   INPUT PARAMETERS                                             *
C     *                                                                *
C     *       NL      - THE NUMBER OF LINES                            *
C     *       X1      - THE X-COORDINATE OF ORIGIN (m)                 *
C     *       Y1      - THE Y-COORDINATE OF ORIGIN (m)                 *
C     *       X2      - THE X-COORDINATE OF DESTINATION (m)            *
C     *       Y2      - THE Y-COORDINATE OF DESTINATION (m)            *
C     *       FAIL1   - FLAG ( TRUE IF LAYOUT FAILS )                  *
C     *                                                                *
C     *   LOCAL VARIABLES                                              *
C     *                                                                *
C     *       I       - A COUNTER                                      *
C     *                                                                *
C     ******************************************************************
C     INPUT PARAMETERS
      LOGICAL FAIL1
      INTEGER NL
      REAL X1(200),Y1(200),X2(200),Y2(200)
C     LOCAL VARIABLES
      INTEGER I
C     WRITE LINES.LSP FILE
      OPEN(UNIT=1,FILE='LINES.LSP')
      WRITE(1,150)
      DO 100 I=1,NL,1
      WRITE(1,151)X1(I),Y1(I),X2(I),Y2(I)
  100 CONTINUE
      WRITE(1,152)
      IF (FAIL1) GO TO 101
      WRITE(1,153)
      GO TO 102
  101 CONTINUE
      WRITE(1,154)
  102 CONTINUE
      CLOSE(UNIT=1)
      RETURN
  150 FORMAT('(setq lines (quote (')
  151 FORMAT('(( ',F9.1,2X,F9.1,' ) ( ',F9.1,2X,F9.1,' ))')
  152 FORMAT(' )))')
  153 FORMAT('(princ "ERROR: MAX. CAPACITY EXCEEDED   ")  600 ')
  154 FORMAT('(princ "ERROR: LINE WITH NO LOAD     ")  0 ')
      END
```

The following should be included in a separate file called "ucost".

```
0.75 IN PIPE         +0001.62
1.00 IN PIPE         +0001.97
1.50 IN PIPE         +0002.88
2.00 IN PIPE         +0004.07
3.00 IN PIPE         +0007.85
ROAD CROSSING        +0563.57
RAILWAY CROSSING     +1449.00
CABLE CROSSING       +0090.00
CREEK CROSSING       +0542.00
ROADS PER KM         +0000.58
CABLES PER KM        +0000.69
```

The following should be included in a separate file called "flookup".

```
1.0000
1.0000
1.0000
1.0000
1.0000
0.9430
0.8860
0.8291
0.7721
0.7151
0.7091
0.7032
0.6974
0.6917
0.6861
0.6806
0.6752
0.6700
0.6649
0.6601
0.6555
0.6511
0.6469
0.6431
0.6396
0.6365
0.6337
0.6312
0.6292
0.6274
0.6260
0.6250
0.6242
0.6236
0.6233
0.6232
```

The following should be included in a separate file called "klookup".

```
 3.69037    6.20531   14.73307   30.23703    83.32298
 4.05659    6.84478   16.20134   31.09201    83.32298
 4.25167    7.18541   17.29060   31.81488    83.32298
 4.40290    7.49313   17.82969   32.44107    83.32298
 4.51557    7.68815   18.42916   32.99340    83.32298
 4.62725    7.84122   18.91896   33.48747    83.32298
 4.69857    8.00610   19.33308   33.93442    84.50130
 4.75809    8.14893   19.47677   34.34245    85.57704
 4.83024    8.27492   19.79319   34.71780    86.56662
 4.89478    8.34725   20.07624   33.99955    85.83196
 4.95317    8.41030   20.33228   34.44650    87.01029
 5.00647    8.50337   20.56603   34.85453    88.08602
 5.05550    8.58899   20.78106   35.22988    88.57092
 5.10090    8.66825   20.87877   35.57740    89.48711
 5.12167    8.74205   20.96641   35.90094    89.85519
 5.14044    8.81109   21.13979   36.20359    90.65309
 5.17757    8.87593   21.30266   36.48788    91.40260
 5.21259    8.93707   21.45621   36.40839    92.10926
 5.24571    8.95737   21.60146   36.66194    92.77769
 5.27713    8.97597   21.73926   36.90247    93.41184
 5.30702    9.02816   21.87033   37.13127    94.01504
 5.33552    9.07792   21.99530   37.34942    94.59017
 5.36275    9.12547   22.11472   37.55787    95.13973
 5.38882    9.17099   22.22905   37.75745    95.66591
 5.41383    9.21466   22.33872   37.94888    95.70400
 5.43785    9.25661   22.34981   38.13280    96.18890
 5.46097    9.29698   22.45120   38.30978    96.20586
 5.46316    9.33588   22.45783   38.48032    96.65548
 5.48466    9.37341   22.55210   38.64488    97.08932
 5.48598    9.40968   22.64317   38.80386    97.50845
 5.50606    9.44475   22.73126   38.95762    97.91383
 5.52551    9.47871   22.81655   39.10651    98.30634
 5.54436    9.51163   22.89922   39.08625    98.68678
 5.56265    9.51541   22.97942   39.22624    99.05585
 5.58041    9.53949   23.05729   39.20320    99.41423
 5.59767    9.56962   23.13297   39.33530    99.76251
 5.61445    9.59205   23.20658   39.46379   100.10120
 5.63079    9.62058   23.27822   39.58884   100.43090
 5.64670    9.62128   23.34800   39.71066   100.75210
 5.66221    9.64836   23.41602   39.82938   101.06510
 5.67734    9.67477   23.48235   39.94517   101.37040
 5.69210    9.70055   23.54709   40.05817   101.66830
 5.70651    9.72572   23.61030   40.16852   101.95920
 5.72059    9.75031   23.67206   40.27633   102.24340
 5.73436    9.77435   23.73244   40.38171   102.52130
 5.74783    9.79786   23.79148   40.48478   102.79300
 5.76100    9.82086   23.84926   40.58563   102.62500
 5.77390    9.84338   23.90582   40.68436   102.88530
 5.78653    9.86544   23.96121   40.78105   103.14020
 5.79890    9.88705   23.92739   40.87579   103.39000
```

5.79591	9.90823	23.98059	40.96865	103.21570
5.80780	9.92900	24.03276	41.05971	103.45580
5.81575	9.94937	24.08393	41.14903	103.69130
5.82720	9.96937	24.04885	41.23669	103.92230
5.83474	9.98899	24.09815	41.32273	104.14920
5.84578	10.00827	24.14655	41.40723	104.37200
5.85662	10.02720	24.19410	41.49023	104.59080
5.86727	10.04580	24.24083	41.57179	104.80580
5.87774	10.06409	24.28675	41.65195	105.01720
5.87345	10.08206	24.33190	41.73077	105.22490
5.88358	10.09974	24.37631	41.80828	105.42930
5.89354	10.11714	24.41999	41.73077	105.63030
5.90334	10.13425	24.46297	41.80580	105.82810
5.91299	10.11818	24.50528	41.87965	106.02280
5.92249	10.13477	24.54693	41.95235	106.21450
5.93184	10.15110	24.58795	42.02395	106.40330
5.94105	10.16718	24.62835	42.09447	106.58920
5.95013	10.18303	24.66815	42.01506	106.77230
5.95907	10.19218	24.70737	42.08352	106.95280
5.96788	10.20757	24.74602	42.15099	107.13070
5.97657	10.19728	24.78413	42.21751	107.30610
5.98514	10.21224	24.82170	42.28310	107.47900
5.99359	10.22699	24.85876	42.34778	107.64950
6.00193	10.24154	24.89531	42.41158	107.81770
6.01015	10.25590	24.93137	42.47453	107.98370
6.01826	10.27007	24.96695	42.53664	108.14740
6.02627	10.28405	25.00207	42.59794	108.30910
6.03417	10.29786	25.03673	42.65845	108.46860
6.04198	10.31148	25.07095	42.71819	108.62610
6.04968	10.32494	25.10475	42.77718	108.78160
6.05729	10.33822	25.13812	42.83543	108.93520
6.06481	10.35135	25.17108	42.89297	109.08690
6.07224	10.36431	25.20364	42.94981	109.23670
6.07957	10.37712	25.23582	43.00597	109.38480
6.08682	10.38978	25.26761	43.06147	109.53110
6.09399	10.40229	25.29903	43.11631	109.67570
6.10107	10.41466	25.33009	43.17053	109.81860
6.10807	10.42688	25.36079	43.22412	109.95990
6.11499	10.43897	25.39115	43.27711	109.69420
6.12184	10.45092	25.42116	43.32951	109.83240
6.12861	10.46274	25.45085	43.38132	109.96900
6.13530	10.47443	25.39754	43.43258	110.10410
6.14192	10.48600	25.42659	43.48327	110.23770
6.14847	10.49743	25.45532	43.53342	110.37000
6.15496	10.50875	25.48375	43.58305	110.50080
6.14361	10.51995	25.51188	43.63215	110.63030
6.14996	10.53104	25.53972	43.68075	110.75840
6.15625	10.54201	25.56727	43.72884	110.88520
6.16246	10.55287	25.59454	43.77645	111.01070
6.16862	10.56362	25.62154	43.82358	110.74240
6.17472	10.57426	25.56808	43.87024	110.86550
6.18075	10.58480	25.59454	43.91645	110.98730

6.18673	10.59524	25.62075	43.96220	111.10790
6.18916	10.60557	25.64671	44.00750	111.22730
6.19502	10.61581	25.67242	44.05238	111.34560
6.20083	10.62595	25.69788	44.09683	111.46280
6.20658	10.63599	25.72311	44.14086	111.57890
6.21228	10.64594	25.74810	44.18448	111.69390
6.21792	10.65580	25.77286	44.22770	111.80790
6.22352	10.66557	25.79739	44.27053	111.92080
6.21529	10.67525	25.82170	44.31297	112.03270
6.22079	10.68484	25.84580	44.35503	112.14350
6.22623	10.69435	25.86968	44.39671	112.25340
6.23163	10.70377	25.89335	44.29373	112.36240
6.23698	10.71312	25.91681	44.33468	112.47030
6.24228	10.72238	25.94007	44.37528	112.57740
6.24754	10.73156	25.96313	44.41553	112.68350
6.25276	10.74066	25.98599	44.45544	112.78870
6.25793	10.74969	26.00866	44.49502	112.89300
6.26305	10.75864	26.03114	44.39427	112.99650

APPENDIX E: SOURCE CODE FOR LISP ROUTINES

```
(defun C:EMPTY ()
        (setq atomlist (member 'C:EMPTY atomlist)) 'DONE)

(defun C:SOURCE ()
        (setq scmde (getvar "cmdecho"))
        (setq sblip (getvar "blipmode"))
        (setq shigh (getvar "highlight"))
        (setvar "cmdecho" 0)
        (setvar "blipmode" 0)
        (setvar "highlight" 0)
        (command "layer" "set" "nodes" "")
        (setq pt1 (getpoint "\nEnter coordinates of source: "))
        (setq pt2 (list (car pt1) (- (cadr pt1) 200.0)))
        (command "circle" pt1 400.0)
        (command "text" "c" pt2 400.0 0.0 "S")
        (setq nodes (list 1 (list (append pt1 (list 1)))))
        (command "layer" "set" "0" "")
        (setvar "cmdecho" scmde)
        (setvar "blipmode" sblip)
        (setvar "highlight" shigh))

(defun C:SINK ()
        (setq scmde (getvar "cmdecho"))
        (setq sblip (getvar "blipmode"))
        (setq shigh (getvar "highlight"))
        (setvar "cmdecho" 0)
        (setvar "blipmode" 0)
        (setvar "highlight" 0)
        (command "layer" "set" "nodes" "")
        (setq num (+ 1 (car nodes)))
        (setq ndlst (cadr nodes))
        (print (quote NODE)) (print num)
        (setq pt1 (getpoint "\nEnter coordinates of node: "))
        (while pt1
                (setq pt2 (list (+ (car pt1) 500.0) (- (cadr pt1)
200.0)))
                (setq a (itoa num))
                (command "doughnut" 0.0 400.0 pt1 "")
                (command "text" pt2 400.0 0.0 a)
                (setq ndlst (append ndlst (list (append pt1 (list 1)))))
                (setq num (+ 1 num))
                (print (quote NODE)) (print num)
                (setq pt1 (getpoint "\nEnter coordinates of node: ")))
        (setq num (- num 1))
        (setq nodes (list num ndlst))
        (command "layer" "set" "0" "")
        (setvar "cmdecho" scmde)
        (setvar "blipmode" sblip)
        (setvar "highlight" shigh))


(defun C:DUMMY ()
        (setq scmde (getvar "cmdecho"))
        (setq sblip (getvar "blipmode"))
        (setq shigh (getvar "highlight"))
        (setvar "cmdecho" 0)
        (setvar "blipmode" 0)
        (setvar "highlight" 0)
        (command "layer" "set" "nodes" "")
        (setq num (+ 1 (car nodes)))
        (setq ndlst (cadr nodes))
        (print (quote NODE)) (print num)
        (setq pt1 (getpoint "\nEnter coordinates of node: "))
        (while pt1
                (setq pt2 (list (+ (car pt1) 500.0) (- (cadr pt1)
200.0)))
                (setq a (itoa num))
                (command "circle" pt1 200.0)
                (command "text" pt2 400.0 0.0 a)
                (setq ndlst (append ndlst (list (append pt1 (list 0)))))
                (setq num (+ 1 num))
                (print (quote NODE)) (print num)
                (setq pt1 (getpoint "\nEnter coordinates of node: ")))
        (setq num (- num 1))
        (setq nodes (list num ndlst))
        (command "layer" "set" "0" "")
        (setvar "cmdecho" scmde)
        (setvar "blipmode" sblip)
        (setvar "highlight" shigh))

(defun sub1 ()
        (setq nodes (list (- num 1) (append ndlst1 (cdr ndlst2))))
        (setq ndlst2 nil))

(defun sub2 ()
        (setq ndlst1 (append ndlst1
                             (list (car ndlst2))))
        (setq ndlst2 (cdr ndlst2)))


(defun C:SUBNODE ()
        (setq pt (getpoint "\nEnter coordinates of node to remove: "))
        (setq x (fix (car pt)))
        (setq y (fix (cadr pt)))
        (setq num (car nodes))
        (setq ndlst1 '())
        (setq ndlst2 (cadr nodes))
        (while ndlst2
                (if (and (= x (fix (caar ndlst2)))
                         (= y (fix (cadar ndlst2))))
                    (sub1)
                    (sub2))))
```

```
(defun nodesave ()
        (setq f (open "NODES2" "w"))
        (print (car nodes) f)
        (setq ndlst (cadr nodes))
        (while ndlst
                (print (caar ndlst) f)
                (print (cadar ndlst) f)
                (print (caddar ndlst) f)
                (setq ndlst (cdr ndlst)))
        (print nil f)
        (close f))


(defun rnodesave ()
        (setq ang (getreal "\nInput the angle of rotation: "))
        (setq f (open "RNODES" "w"))
        (print ang f)
        (print (car nodes) f)
        (setq ndlst (cadr nodes))
        (while ndlst
                (print (caar ndlst) f)
                (print (cadar ndlst) f)
                (print (caddar ndlst) f)
                (setq ndlst (cdr ndlst)))
        (print nil f)
        (close f))


(defun C:COMP ()
        (setq scmde (getvar "cmdecho"))
        (setvar "cmdecho" 0)
        (nodesave)
        (command "shell" "comp")
        (graphscr)
        (setvar "cmdecho" scmde))


(defun C:DIJK1 ()
        (setq scmde (getvar "cmdecho"))
        (setvar "cmdecho" 0)
        (nodesave)
        (command "shell" "dijk1")
        (graphscr)
        (setvar "cmdecho" scmde))


(defun C:DIJK2 ()
        (setq scmde (getvar "cmdecho"))
        (setvar "cmdecho" 0)
        (nodesave)
        (command "shell" "dijk2")
        (graphscr)
        (setvar "cmdecho" scmde))


(defun C:MST ()
        (setq scmde (getvar "cmdecho"))
        (setvar "cmdecho" 0)
        (nodesave)
        (command "shell" "mst_2")
        (graphscr)
        (setvar "cmdecho" scmde))


(defun C:RST2 ()
        (setq scmde (getvar "cmdecho"))
        (setvar "cmdecho" 0)
        (nodesave)
        (command "shell" "rst2")
        (graphscr)
        (setvar "cmdecho" scmde))


(defun C:ROT ()
        (setq scmde (getvar "cmdecho"))
        (setvar "cmdecho" 0)
        (rnodesave)
        (command "shell" "rotate")
        (graphscr)
        (setvar "cmdecho" scmde))


(defun C:LINEPLOT ()
        (setq scmde (getvar "cmdecho"))
        (setq sblip (getvar "blipmode"))
        (setvar "cmdecho" 0)
        (setvar "blipmode" 0)
        (command "layer" "set" "lines" "")
        (setq lnlst lines)
        (while lnlst
                (setq pt1 (caar lnlst))
                (setq pt2 (cadar lnlst))
                (command "line" pt1 pt2 "")
                (setq lnlst (cdr lnlst)))
        (command "layer" "set" "0" "")
        (setvar "cmdecho" scmde)
        (setvar "blipmode" sblip))


(defun C:NODEPLOT ()
        (setq scmde (getvar "cmdecho"))
        (setq sblip (getvar "blipmode"))
        (setq shigh (getvar "highlight"))
        (setvar "cmdecho" 0)
        (setvar "blipmode" 0)
        (setvar "highlight" 0)
        (command "layer" "set" "nodes" "")
        (setq ndlst (cadr nodes))
        (setq pt1 (car ndlst))
        (setq pt2 (list (car pt1) (- (cadr pt1) 200.0)))
        (command "circle" pt1 400.0)
        (command "text" "c" pt2 400.0 0.0 "S")
        (setq ndlst (cdr ndlst))
        (setq num 2)
        (while ndlst
                (setq a (itoa num))
                (setq temp (car ndlst))
                (setq pt1 (list (car temp) (cadr temp)))
                (setq flag (caddr temp))
                (setq pt2 (list (+ (car pt1) 500.0) (- (cadr pt1)
200.0)))
                (if (/= flag 0)
                        (command "doughnut" 0.0 400.0 pt1 "")
                        (command "circle" pt1 200.0 ))
                (command "text" pt2 400.0 0.0 a)
                (setq ndlst (cdr ndlst))
                (setq num (+ 1 num)))
        (command "layer" "set" "0" "")
        (setvar "cmdecho" scmde)
        (setvar "blipmode" sblip)
        (setvar "highlight" shigh))


(defun C:WIPE ()
        (setq scmde (getvar "cmdecho"))
        (setq sblip (getvar "blipmode"))
        (setq shigh (getvar "highlight"))
        (setvar "cmdecho" 0)
        (setvar "blipmode" 0)
        (setvar "highlight" 0)
        (setq pt1 (getvar "extmin"))
        (setq pt2 (getvar "extmax"))
        (command "erase" "w" pt1 pt2 "")
        (setvar "cmdecho" scmde)
        (setvar "blipmode" sblip)
        (setvar "highlight" shigh))


(defun C:SUBMIT ()
        (setq scmde (getvar "cmdecho"))
        (setvar "cmdecho" 0)
        (command "shell" "submit")
        (graphscr)
        (setvar "cmdecho" scmde))
```

```
(defun C:NODEDATA ()
        (setq scmde (getvar "cmdecho"))
        (setvar "cmdecho" 0)
        (command "shell" "type nodes.dat >lpt1")
        (graphscr)
        (setvar "cmdecho" scmde))


(defun C:LINEDATA ()
        (setq scmde (getvar "cmdecho"))
        (setvar "cmdecho" 0)
        (command "shell" "type lines.dat >lpt1")
        (graphscr)
        (setvar "cmdecho" scmde))


(defun lplot (lnlst)
        (while lnlst
                (setq pt1 (caar lnlst))
                (setq pt2 (cadar lnlst))
                (command "line" pt1 pt2 "")
                (setq lnlst (cdr lnlst))))


(defun C:DPLOT ()
        (setq scmde (getvar "cmdecho"))
        (setq sblip (getvar "blipmode"))
        (setvar "cmdecho" 0)
        (setvar "blipmode" 0)
        (command "layer" "set" "l1889" "")
        (lplot l889)
        (command "layer" "set" "l1603" "")
        (lplot l603)
        (command "layer" "set" "l1483" "")
        (lplot l483)
        (command "layer" "set" "l1334" "")
        (lplot l334)
        (command "layer" "set" "l1267" "")
        (lplot l267)
        (command "layer" "set" "0" "")
        (setvar "cmdecho" scmde)
        (setvar "blipmode" sblip))


(defun C:FAILPLOT ()
        (setq scmde (getvar "cmdecho"))
        (setq sblip (getvar "blipmode"))
        (setvar "cmdecho" 0)
        (setvar "blipmode" 0)
        (command "layer" "set" "lines" "")
        (lplot lines)
        (setvar "cmdecho" scmde)
(setvar "blipmode" sblip))
```

APPENDIX F: SOURCE CODE FOR MENU

```
***BUTTONS
;
$p1=*
^c^c
^B
^O
^G
^D
^E
^T
***AUX1
;
$p1=*
^C^C
^B
^O
^G
^D
^E
^T
***SCREEN
**TORNADO
[TORNADO ]
[********]
[  MAIN  ]

[ INPUT  ]$S=INPUT
[ LAYOUT ]$S=LAYOUT
[ DESIGN ]$S=DESIGN
[ DISPLAY]$S=DISPLAY
[ OUTPUT ]$S=OUTPUT
[ MEMORY ]$S=MEMORY


[ RSTRB  ](LOAD "LINES3")(LOAD "NODES3")(C:LINEPLOT)
[ FUZZY  ](LOAD "LINES4")(LOAD "NODES4")(C:LINEPLOT)
```

```
**INPUT
[TORNADO ]$S=TORNADO
[********]
[ INPUT ]

[ SOURCE](C:SOURCE)(C:SINK)
[  SINK ](C:SINK)
[ DUMMY ](C:DUMMY)
[SUBNODE](C:SUBNODE)

[  END  ]END;
```

```
**LAYOUT
[TORNADO ]$S=TORNADO
[********]
[ LAYOUT ]

[  COMP   ](C:COMP)(LOAD "LINES2")(C:LINEPLOT)
[  DIJK1  ](C:DIJK1)(LOAD "LINES2")(C:LINEPLOT)
[  DIJK2  ](C:DIJK2)(LOAD "LINES2")(C:LINEPLOT)
[  MST    ](C:MST)(LOAD "LINES2")(C:LINEPLOT)
[  RST2   ](C:RST2)(LOAD "LINES2")(LOAD "NODES2")(C:LINEPLOT)
[  ROT    ](C:ROT)(LOAD "NODES2")(C:WIPE)(C:NODEPLOT)
```

```
**DISPLAY
[TORNADO ]$S=TORNADO
[********]
[ DISPLAY]

[NODESON ]LAYER;ON;NODES;;
[NODESOFF]LAYER;OFF;NODES;;
[NODEPLOT](C:NODEPLOT)
[LINESON ]LAYER;ON;LINES;;
[LINESOFF]LAYER;OFF;LINES;;
[LINEPLOT](C:LINEPLOT)

[  WIPE  ](C:WIPE)
[ LWIPE  ]LAYER;OFF;NODES;;(C:WIPE);+
LAYER;ON;NODES;;
[ LWIPEP ]LAYER;OFF;NODES;;(C:WIPE);+
LAYER;ON;NODES;;(C:LINEPLOT)

[ RVALUE ](C:RVALUE)



**MEMORY
[TORNADO ]$S=TORNADO
[********]
[ MEMORY ]

[  LOAD  ](LOAD "TORNADO")
[ EMPTY  ](C:EMPTY)
```

```
**DESIGN
[TORNADO ]$S=TORNADO
[********]
[ DESIGN ]



[ SUBMIT ](C:SUBMIT)(LOAD "LINES")(C:DPLOT)










**DISPLAY
[ DESIGN ]$S=DESIGN
[********]

[ LAYERS ]$S=LAYERS

[LINEPLOT](C:LINEPLOT)
[NODEPLOT](C:NODEPLOT)
[FAILPLOT](C:FAILPLOT)






[  WIPE  ](C:WIPE)
```

```
**OUTPUT
[TORNADO ]$S=TORNADO
[********]
[ OUTPUT ]


[NODEDATA](C:NODEDATA)
[LINEDATA](C:LINEDATA)
[ PRPLOT ]PRPLOT
```