# APPLICATIONS

# OF DYNAMIC PROGRAMMING

# TO

# PROBLEMS OF ROUTING PIPEWORK AND CABLES

S B KAPUR

B Sc (Eng),A.C.G.I.

Submitted in the partial fulfilment of the Degree of Doctor of Philosophy of the University of London and for the Membership of the Diploma of Imperial College London.

# ABSTRACT

Various branches of Engineering applications involve the problem of deciding the physical positioning of components and their interconnections which could be classed under the general title of 'Layout' problems. The layout problems in Precedence type Management Networks, Printed Circuit Boards and Pipe routing are considered here followed by consideration of the physical contraints to be observed in their solution. Finally, together with existing algorithms, new and more optimum algorithms are discussed and these are presented as interactive graphics systems.

ACKNOWLEDGEMENTS

Special thanks are due to the British Ship Research Association, Wallsend Research Station, Newcastle, for sponsoring most of this work at Imperial College, London.

I wish to acknowledge with deepest appreciation the suggestions, counsel and encouragement from Professor P J Grant, who has been a great source of inspiration with his never ending enthusiasm throughout this project. I also wish to thank Dr A Jebb and Dr C B Besant for their help in obtaining the objectives of this project.

I am highly grateful to my friends, without whose help and enthusiasm this project would never have been successful. Special thanks are due to Mrs Margaret Bradstreet for typing this thesis so painstakingly.

- S B Kapur.

TO ERR IS HUMAN -

But to make a real mess,
It takes a Computer.

- Anonymous.

CONTENTS PAGE NO.

# 1. INTRODUCTION

As industry takes a further step in the direction of automatic design and production, an interactive graphics computer system provides a very versatile and powerful system for problem solving throughout the spectrum of applications in design, draughting, production and quality assurance. This research was primarily aimed at the more classical problem of pipe and conduit routing in ships. The principles involved in pipe routing, considered here in the light of their application to ships, buildings and chemical plants, share the layout problem with printed circuit boards and management networks.

Precedence type management networks lend themselves relatively easily to simulation in 2 dimensions aiding to good pre-planning of a project and an optimum flow of information at all levels during the life of the project. Management networks were simulated consistent with the British Standard 4335 (1972)[1] and the result provided with a technique for fast data input, analysis, interactive modifications and data extraction in a variable format for further analysis. An algorithm was developed for minimising cross overs in a given network. This was used in management networks for clarity and understanding and a further application was discovered in optimising cost of printed circuit boards by minimising the number of conducting wire holes in printed circuit boards.

Various graph theoretical and branch-and-bound methods were considered for the automatic layouts of printed circuit boards but it was found that no significant cost optimisation could be carried out using the existing algorithms. The automatic and interactive methods finally implemented for the printed circuit board layouts were based on an algorithm which was specially designed for this particular application.

The afore mentioned algorithm when extended to 3-dimensional layouts led to the solution of the general pipe routing problem. Various constraints on the system were considered together with preset priorities for different pipes and conduits. The algorithm which was used carried out the cost optimisation on the basis of shortest path analysis combined with minimisation of bends.

It is clear that the layout problems involved in management networks, printed circuit boards and pipe routing can be solved using the same general algorithms with special optimisation techniques built in and these are presented as independent graphics systems.

## 2. SYSTEM HARDWARE

The graphics station in use consisted of a PDP 11/45 operating under DOS ( Disk Operating System ), a Tectronix-611 graphics display and a Kennedy magtape unit together with a Decwriter and a Logabax line printer. The main peripheral was a cursor and grid based digitising table. The control program used to bring the application programs into core for execution was operated from a "menu" area on the digitising table.

"Menu" based systems have become quite popular in computer applications to graphics in various fields. Menus can be displayed on a screen and then a light pen can be used for selecting the operation to be performed. The menu in use here was based on sensing the position of the 'bug' or the cursor on the digitising table and deciphering the co-ordinates in terms of an application program for carrying out a specific function. The digitising table provided a good means of conversion of picture data into digital (X,Y) co-ordinates. It is thought that the solid state tablet offers more in terms of ease of use and more consistent accuracy as compared with the digitising table.

Work is continuing at Imperial College on automatic conversion of mechanical engineering drawing data into digital (X,Y) data.

Three dimensional data input on the digitising table is carried out using separate perspective planes, XY, YZ and XZ. C Yi[2] in his thesis has covered various aspects of this subject.

## 3. GRAPHICS SOFTWARE

An attempt is made here to describe various facets of the graphics software available as concisely as possible.

The philosophy behind the graphics in 2-D was that every point would carry (X,Y) co-ordinates together with a flagging code indicating the state of the 'beam' or the 'pen' in generating the vector $\overline{XY}$. The same principle was extended to 3-D graphics with the vector under consideration being $\overline{XYZ}$.

Both systems, i.e. 2-D and 3-D, were based on an overlay system where a background program was used to select the required application program and control being returned to the background program once the function of the application program had been completed. Hence, the background program operated in an interminable loop, always resident in core, only to relinquish control when an application program was requested.

Though the 'beam' would only have 'on' or 'off' conditions, i.e. the pen could only be up or down, but more than two flagging codes were employed. So the flagging codes were not only used to indicate the state of the beam but also to identify different types of data. The basic applications on the pure graphics could be listed as follows:

### 3.1. 2-D GRAPHICS:

### 3.1.1 Initial Set Up Routine:

This routine was used to set up grid parameters, input and output scales, alphanumerics size etc.

### 3.1.2 Background Digitising Overlay:

This module was set up to read tabel co-ordinates and check for the pen up or down condition. This also provided the controlled movement of the cursor and was useful in selecting the particular application programs to be executed.

3.1.3. Symbol Generating Routine:

This routine was used to generate any standard symbols such as circles, rectangles and arcs etc.

3.1.4. Windowing Routine:

This routine provided the magnification facility to be able to concentrate on any fraction of the screen and magnifying it to fill the whole screen.

3.1.5. Alphanumeric Generation Routine:

This overlay was used to convert alpha-numeric characters into their equivalent screen vectors thus making it possible to afford storage of alphanumeric data in a more simple and realistic form.

3.1.6. Various Editing Facilities:

This suite of programs formed the backbone of interaction of the computer and the operator. Facilities included deletion of lines, symbols or alphanumerics and provided for any correction to input data on-line.

3.1.7. Find or Locate routines:

This suite of 3 programs was used to aid the operator in determining the 'exact' positions of already input points, lines or intersections of lines under program control.

3.1.8. File Manipulation Routines:

These routines were used for storing picture data on disk, copying such files, initialising them and displaying already stored files.

3.2. 3 - D Graphics:

The 3-D graphics system provides for all the facilities mentioned under the 2-D system together with the following additional operational routines:

3.2.1. Simulated Joy-stick Program:

This program provided the facility for viewing a 3-D perspective picture from any angle in 3-dimensions and also provided the facility of positive or negative 'zoom' effect.

3.2.2. Solids of Revolution Generator:

This module made it possible to be able to generate a solid of revolution of any profile round any axis.

3.2.3. Perspective Transformation Routine:

This program was used for conversion of 3-D picture data into perspective planes i.e. XY, YZ and XZ planes, and vice-versa.

The overall systems described by the above mentioned modules provided a very versatile arrangement for 2-D and 3-D systems. The application programs could be added to either of the systems. The programs that took care of management networks and of printed circuit boards were added to the 2-D system, whereas the pipe routing and the design system was added on to the 3-D system. This was made particularly easy by the modular structure of the available systems.

Further details of the 2-D graphics system exist in the works of A. Hamlyn[3] and R. C. Edney[4] whereas the 3-D system is covered by C. Yi.[2]

## 4. NATURE OF LAYOUT PROBLEMS

The layout problem is that of deciding the positioning of components, whether electronic, mechanical or chemical, and their interconnections. A combination of factors cause the problem. Firstly the increasing complexity of equipment means that there are many components to be interconnected. As an example, in a powerful computer, there can be 4o,ooo logic gates and upto 25o,ooo interconnections. An average tower block may contain 3,ooo electrical conduits or a ship may have a few thousand service routes. Secondly, the factors involved in transmission require that the routes be as small as possible. These include the related flow problem as well as the cost optimisation problem.

For these reasons and others given later it has become necessary to employ automatic procedures to assist in the design of equipment layout. Each of the various aspects of the layout problem may be regarded as a constraint to an overall optimisation problem. However, it is not always possible to find a useful mathematical formulation describing the problem. If one is found, it usually takes the form of a function to be minimised.

For instance, in placing n integrated circuit modules on a plug-in board, the total inter-connecting conductor length L may be expressed as :

$$L = \tfrac{1}{2}\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \; d_{p(i)} \; d_{p(j)}$$

where

$C_{ij} = C_{ji}$ = no. of connections between modules i and j.

and

$d_{p(i)}d_{p(j)}$ = distance between modules i and j.

One way to minimise this expression would be to evaluate it for every possible module placement. There are n! ways of placing n modules on a board; for n=24, there are 24! = 612 x $10^{21}$ arrangements. Clearly, it is out of the question to attempt complete enumeration even with the most powerful computers available.

It might be possible to eliminate many of the combinations immediately. For instance, two modules with a large number of connections might always be placed adjacently if it could be shown initially that any other positioning would contribute a greater amount to the total length than any possible reduction caused by the consequent repositioning of other modules. However, even large reductions in the number of combinations would still leave an enormous number to be investigated.

Some aspects of the layout problem cannot be formulated in any meaningful way at all - e.g. the positioning of a conductor on a printed circuit board. Although a conductor path could be described by a sequence of adjacent 'cells' by dividing the board into a grid, such a description would yield an astronomical number of combinations for, say, specifying the number of possible postionings for 50 different conductors. Same applies to pipes or conduits in 3-dimensional space. Most combinations would of course involve crossings and could be rejected on that account, but there would still remain an enormous number of feasible solutions, some better than others.

Thus, although layout problems could be conceived as combinatorial ones, such an approach would be useless in finding solutions. It should, therefore, be appreciated that in discussing layout problems, remarks concerning feasibility or true optimisation are always made in the context of 'practicable' methods. It is always possible to find a placement by enumerating all possibilities.

The problem is not therefore one of finding solutions but rather 'ways' of finding solutions. Solutions are usually defined as optimum to a given procedure and are not true minimum answers. What is required is a 'good' answer i.e. one that is satisfactory and comes arbitrarily close to a true optimum. This will normally require a compromise between computation time and the 'quality' of the solution. Different algorithms will yield different efficiencies in this respect.

It will be seen that layout problems are not of the type in which formulae and mathematical results are established by researchers which can be used in turn as starting points by new researchers in the field - 'Developments' are concerned with new and more efficient algorithms. Occasionally, these may supersede earlier procedures but in many instances have varying advantages and disadvantages according to the particular problem and constraints applicable in its solution.

# 5. GRAPH THEORY APPLICATIONS.

Having established in the previous chapters that layout problems occur in almost all branches of engineering applications and that there is a lot in common in Management Network graphics, Printed Circuit Board layouts and Pipe routing, a common basis for analysis is not only desirable, it becomes necessary. To deal with any problems of interconnected components, Graph Theory provides many answers in the form of computer oriented algorithms.

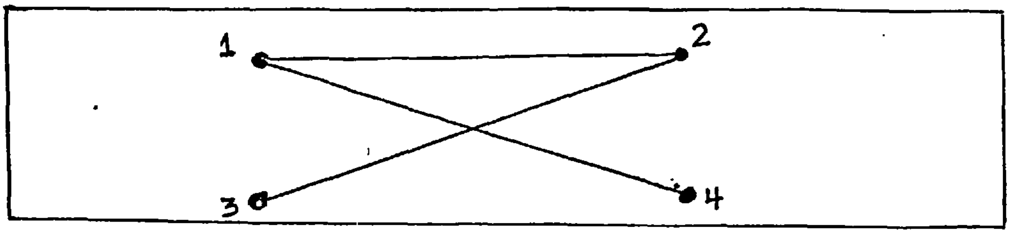
The concepts required are quite simple and have a characteristic 'yes - no' flavour and for the most part require no complicated arithmetical operations. The manipulations involved are ideally suited to the use of digital computers and very large and complex systems can be investigated efficiently. It is often quite simple to build a model for a system in terms of graphs. Presently, the techniques are used to study networks of telephones, satellites, computers, roads, pipe lines, production facilities and many other components of technology. The view usually taken is 'macroscopic' rather than detailed.

A 'Graph' is defined as a collection of points, called nodes, some of which may be inter-connected by lines, called branches. So, a graph can be defined as two sets, a set N of nodes and a set B of branches. Thus the mathematical notation of graph G is

$$G = ( N,B )$$

The set B contains certain unordered

pairs of nodes: those pairs connected by branches. For example,



This graph has 4 nodes and 3 branches. The mathematical notation for this graph is then

G = (N,B)

where N = (1,2,3,4)

and B = ((1,2), (2,3), (1,4)) .

Although a graph will usually be represented by a picture, or by some arrays of numbers in a computer, this mathematical notation illustrates some important points. For example, it would be correct to say

N = (3,2,1,4)

i.e. the sets of nodes and branches are not in any order. Also, the pair of nodes representing any branch is un-ordered, so B could be written as

B = ((3,2), (4,1), (1,2))

i.e. there is no particular direction associated with a branch, and the term undirected graph is sometimes used to emphasize this. In some applications, of course, directed graphs with ordered branches would have to be considered.

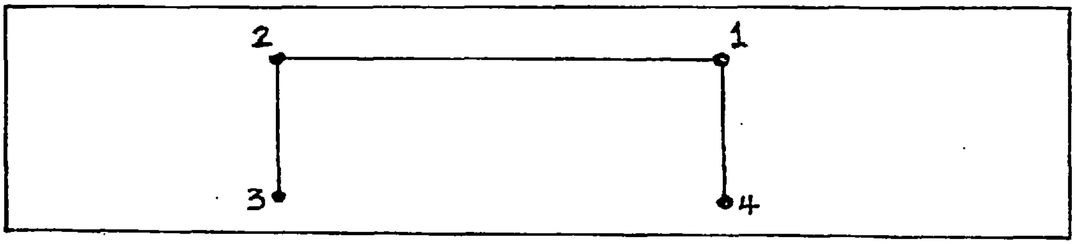
There are two special situations that might arise. First, a node may be connected to itself. This corresponds to a branch of the form (a,a), which connects node a with itself. Such a branch is called a self-loop. Second, there may be more than one branch connecting a particular pair of nodes.

This corresponds to a set B like the following:

$$B = (\ldots\ (a,b),\ (a,b)\ldots)$$

Such branches are called multiple-branches.

Nothing has been said about where the nodes are to be placed when the graph is drawn or what shape lines are used to indicate branches. These decisions are determined by system constraints but do not affect the identity of the graph being represented. The graph shown previously could be re-drawn as:



If it is possible to draw a graph on a plane so that no branches cross, the graph is called "planar". It often clarifies the picture of a graph if it is drawn so as to minimise the number of branch crossings. This has widespread implications in management networks and printed circuit boards.

On the other hand, it might be a constraint to preserve the relative position of the nodes, e.g. steel works in a chemical plant, then the branches would represent relative distances.

Thus a graph is thought of abstractly as being a set of nodes and a set of branches, and the picture is drawn so that it is easy to understand. More often than not, the nodes are placed on a regular grid and this offers a great deal of advantages as will be seen later.

As an example of a graph, the collection of natural gas from offshore drilling platforms, and the transmission of the gas through pipelines to an onshore point, may be considered. Such systems may cost millions of pounds and sophisticated techniques have been developed for designing them efficiently.

Sometimes, merely drawing the right graph helps a great deal towards solving a problem. The problem of collecting garbage in a city provides another good example of the application of graph theory. The cartographic colouring of maps represents another graph theory problem and is usually grouped with "chromatic" problems. Nicos Christofides[5] has done a great deal of work in this field. The main application of graph theory that interests the layout engineer, however, is that of constructing the shortest path connecting two nodes in a graph.

The next problem would be to consider "directed" graphs and each branch would have a direction associated with it. This feature would make it possible to send something from node i to node j but not in the reverse direction. This would obviously find application in pipe lines where the direction of flow would be predefined.

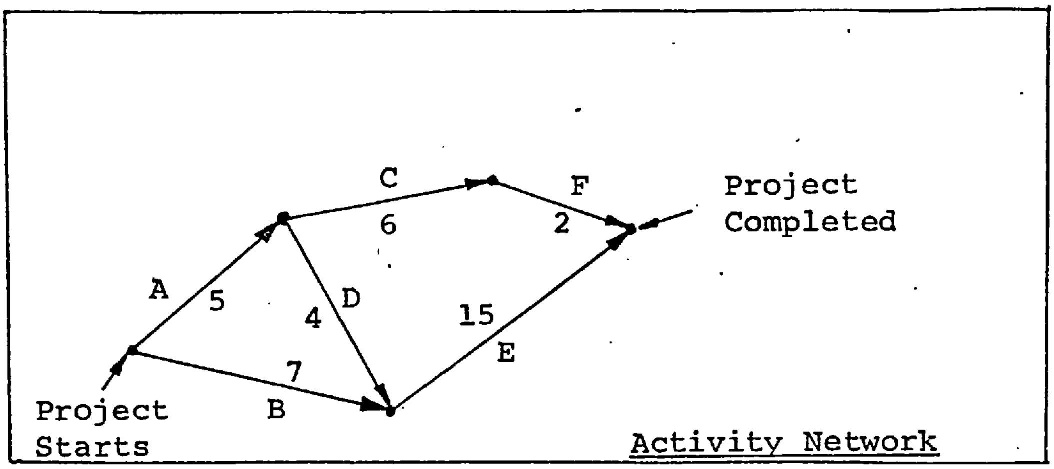
The particular algorithms that have been used to solve management network diagrams, printed circuit board and pipe layouts are described later and proposed algorithms are compared with the graph theoretical algorithms.

## 5.1. ACTIVITY NETWORKS:

One of the most popular and successful applications of networks in operations research is in the planning and scheduling of large complicated projects. The two best known names in this connection are CPM (Critical path method)[6] and PERT (Program evaluation and review technique)[7]. A project is divided into many well-defined and non-overlapping individual jobs, called activities. Due to technical restrictions, some jobs must be finished before others can be started such as putting foundations before erecting walls etc. In addition to this precedence relationship among the activities, each activity also requires a certain time, called the duration of the activity. Given the list of activities in a project, the list of immediate pre-requisites (i.e. predecessors) for each activity, and the duration, a weighted graph can be drawn to depict the project, as follows:

Each edge represents an activity, and its weight represents the duration of the activity. The vertices represent beginnings and endings of activities and are called events in the project. An activity (i, j) cannot be started before all activities leading to the event i have been completed. Each event in the project is a well-defined vertex. Such a weighted, connected graph representing activities in a project is called an activity network.

Taking an extremely simple example, suppose that a project consists of six activities, A,B,C,D,E and F, with the restriction that A must precede C and D;B and D must precede E; and C must precede F. The durations for the activities may be taken as 5, 7, 6, 4, 15 and 12 days respectively. The activity network of this project is as follows:



Activity Network

It can be observed that an activity network must be acyclic; otherwise, an impossible situation would arise in which no activity in the directed circuit could be initiated.

# 6. MANAGEMENT NETWORK ANALYSIS

## 6.1. GRAPHICS:

In this section, a computer aided data preparation system for Precedence type Management Networks is presented containing all the facilities for up-dating the Network in its active phase. The network data for further project analysis could be transferred to a main frame computer on punch cards, magtape or paper tape.

The PM (Project Management) system contained modules to perform the following operations inter-actively:

6.1.1. A module to create rectangles with the option of placing them on a grid. The rectangles would identify with "activities".

6.1.2. A program to input alphanumeric data at pre-defined locations in a selected rectangle. This data consisted of activity codes, descriptions of the activity in terms of resources required and any special treatments required, together with the duration of the activity.

6.1.3. A program to input connecting arrows automatically by selecting any two rectangles. The arrows would represent "events" connecting various activities.

6.1.4. A module was required to specify the "delay" times on events in order to monitor the flow through the network.

6.1.5. A facility was provided to attain even distribution of activities horizontally or vertically for the sake of clarity. In fact, if the activities were placed on a pre-defined grid under program control, then this facility was practically redundant.

6.1.6. A program to magnify i.e. "window" any specific part of the network was included. In a large network, this would be very useful in concentrating only on the required part of the network.

6.1.7. Data extraction routines were included for further project analysis. This data was directly written to the system disk and then could be transferred to any device, e.g. papertape, magtape or punch cards.

6.1.8. The different editors provided, for correction of mistakes in the initial inputting of data or for modification of the network during its active phase, included:

6.1.8.1. An arrow editor to remove any events unnecessary to the network or for inputting new events.

6.1.8.2. A rectangle editor for activities in a network in case any of the activities were redundant.

6.1.8.3. An alphanumerics editor for texts and codes which could be removed or new ones added in.

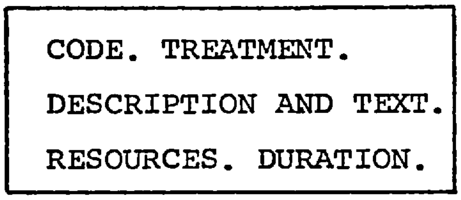
6.1.8.4. An editor to move any form of data, whether graphic or alphanumeric, in two dimensions.

The application programs were added to an existing modular graphics system which allowed the input and editing of drawings within the computer system.

Management Systems are usually designed to assist Management in their control of planning, progress and cost information of production; throughout all stages of construction. Various systems exist but the one described here would contribute to data preparation and extraction from precedence type of activity networks. Further analysis on the network is usually carried out on a main-frame computer but data preparation is a long and tedious process. The PM system is capable of very fast and accurate data preparation for such an analysis. It can generate data for a medium size network in a matter of minutes as compared with manual techniques presently in use which can take up to a few weeks for the data preparation and checking and so it represents a vast saving in the cost of such procedures.

One of the core points of the real time system is its capability to provide man-machine interaction. The user is given maximum possible freedom and is not asked to follow a long set of standard procedures. Mistakes are quickly spotted and corrected by the editors provided.

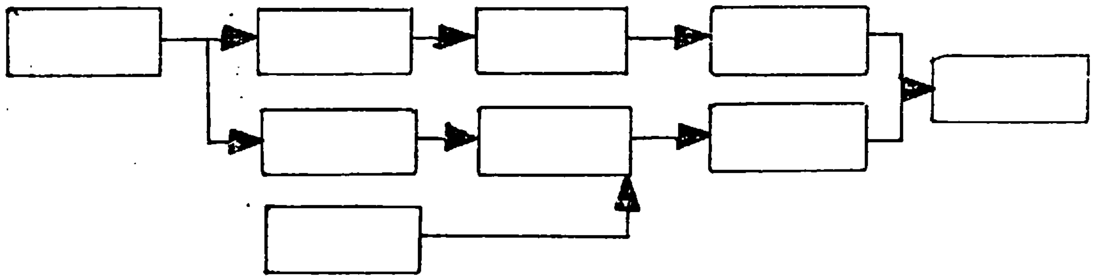
The use of the mini computer requires that a minimum of data is brought into core and this prompted the need to process any network file point by point. All activities of a network are stored in standard size rectangles, and the size is consistent right through any network. The master rectangle is treated as a picture component or a "macro" and the facility is provided to add a master rectangle wherever required. A standard activity may contain information of the type indicated below:



DESCRIPTION OF AN ACTIVITY

Various activities are now generated by first placing the rectangles wherever required and then alphanumerics can be input by just digitising a point in any rectangle to identify it and typing the codes and descriptions etc. as required.

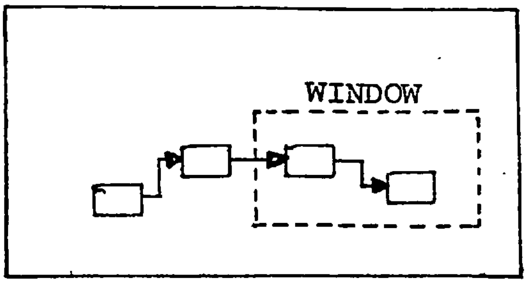
AN EXAMPLE OF A NETWORK CREATED USING 'PM' SYSTEM



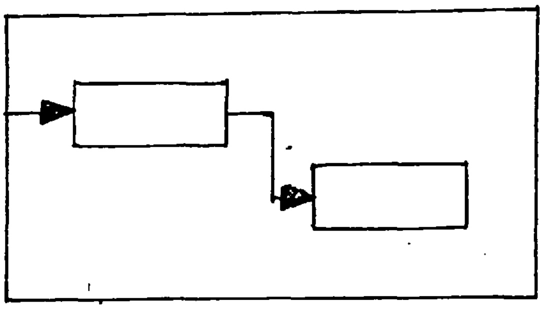
The logical connections representing events can be input next by selecting any pair of activities and any of the following types of connections:

1) Finish to start arrow
2) Finish to finish arrow
3) Start to start arrow

After carrying out the above operations, a complete network is now ready. It is possible to magnify any part of the network to fill up the whole screen by placing a window of variable size around the area of interest.

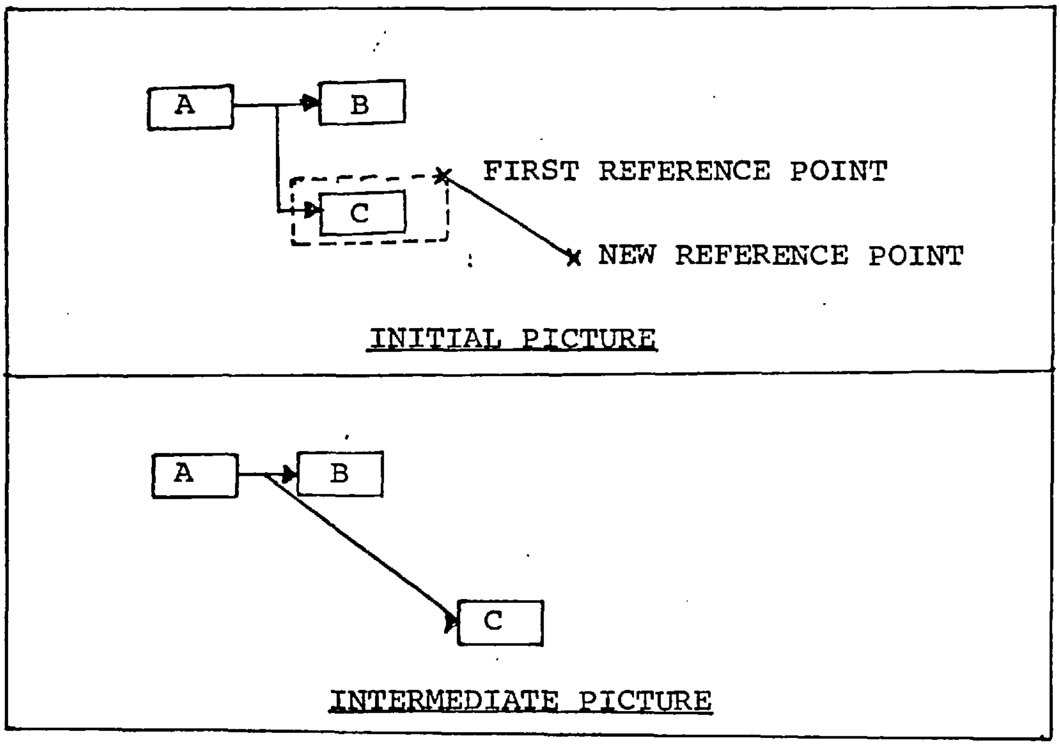


SCREEN FORMAT FOR MAGNIFICATION FACILITY



MAGNIFIED PICTURE

Although it is not always necessary, facility exists for placing activities on a grid, the size of which is pre-specified. It is possible to drag any data in two-dimensional space and any individual activities or combinations of activities may be moved to new locations if so desired.

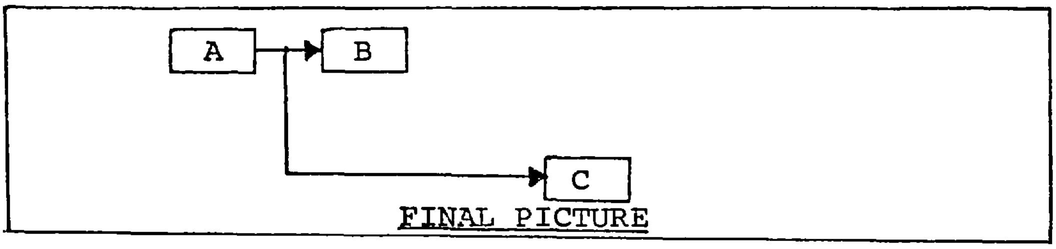


INITIAL PICTURE

INTERMEDIATE PICTURE

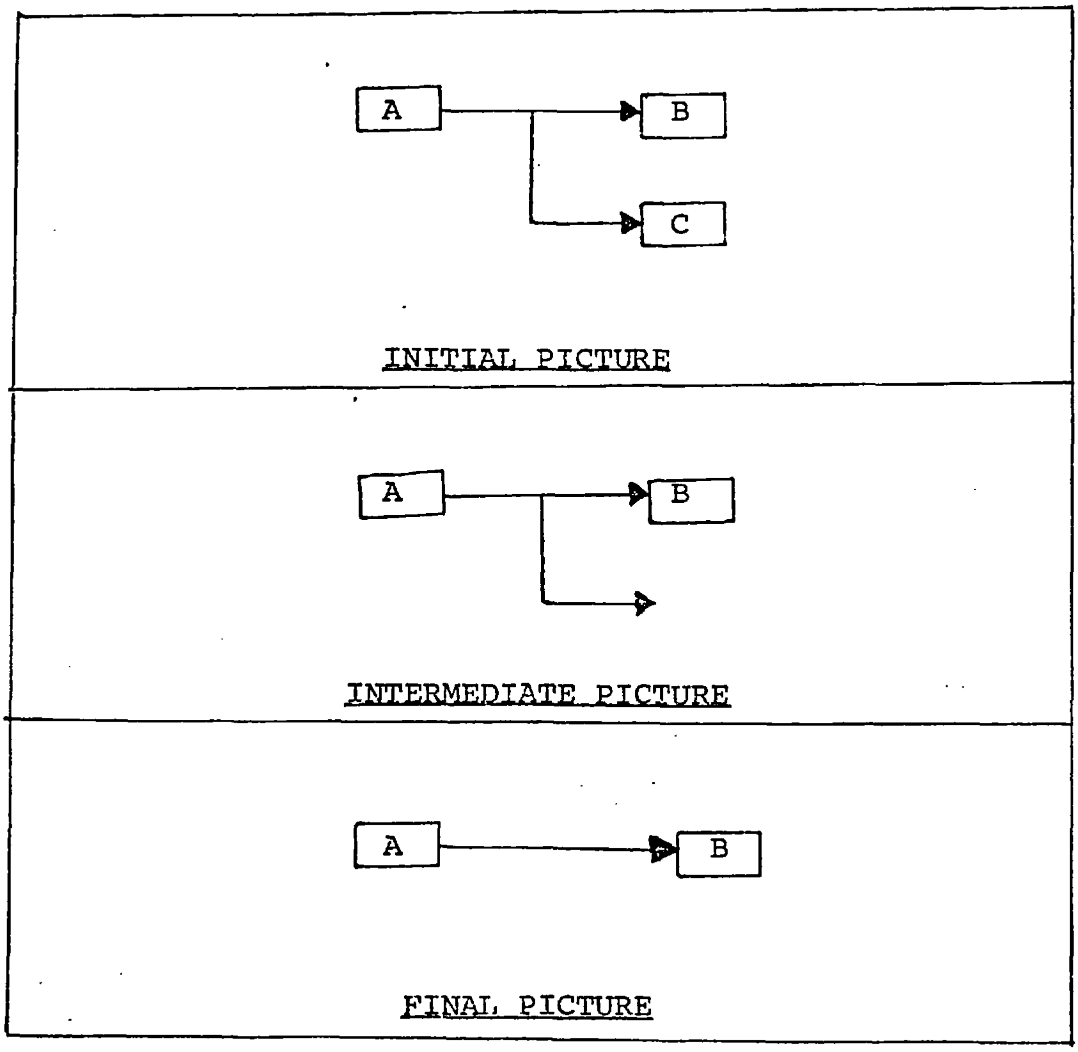
After any of these operations, the geometrical shapes of the arrows can get distorted, though the logical connections do not change. These however, for aesthetic reasons and to clarify the picture, can be reformed.

The old geometrical arrows are deleted and new arrows are added to the picture file to maintain an accurate graphical description of the network.



FINAL PICTURE

If any of the data in the network file needs to be changed, various editors for activities, events and text have been designed. So it is possible to add as well as subtract activities or parts of activities.

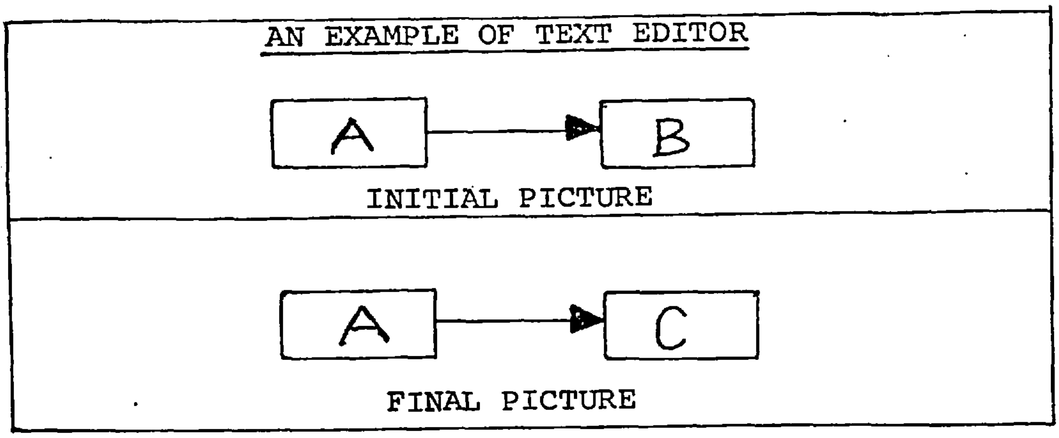


INITIAL PICTURE

INTERMEDIATE PICTURE

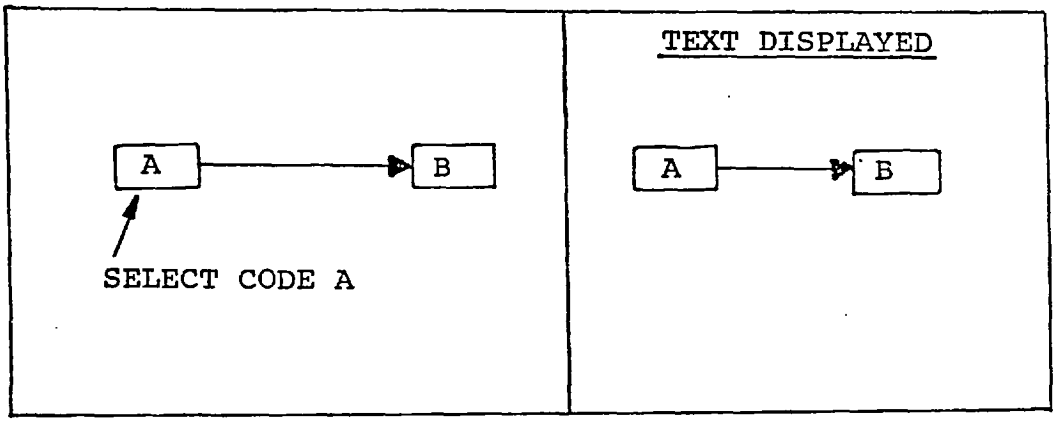
FINAL PICTURE

AN EXAMPLE OF 'ACTIVITY' EDITOR

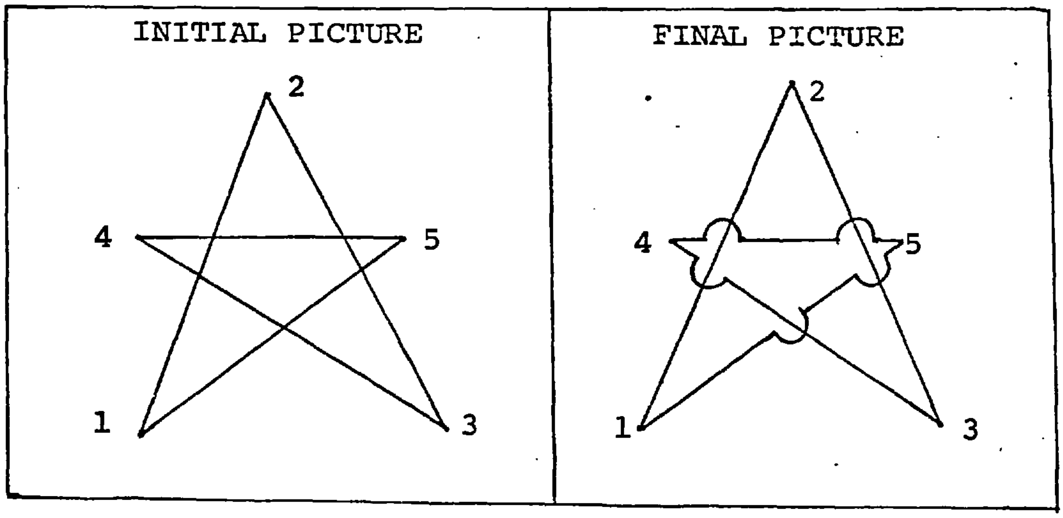


AN EXAMPLE OF TEXT EDITOR

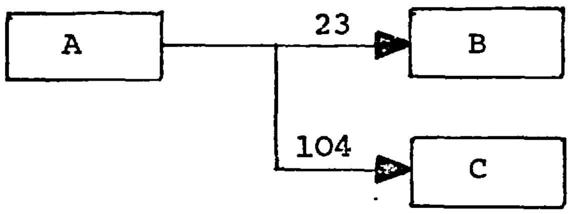
To hold large amounts of text related to a code, a library is stored on the disk. The code can be selected from the screen or just typed in from the keyboard and related text of upto 256 alphanumeric characters is displayed on the screen.



To indicate the continuity of events, in case, two arrows cross-over each other, loops are drawn to indicate which one of them continues.



It is sometimes required to add delay times on certain events and these are placed at the arrow ends for correct representation.



DELAY TIMES ON EVENTS

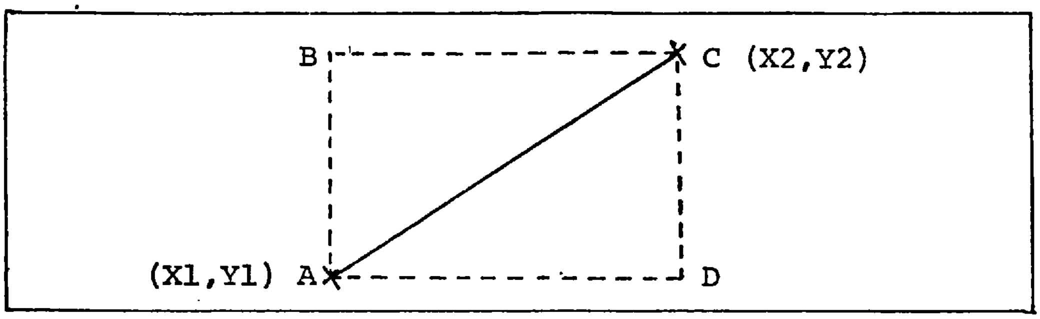
General data extraction routine works on the final network and transfers logical precedence information onto any selected device i.e. paper punch, cards or magtape.

An algorithm has been developed for minimising the number of cross-overs in a project network and is described later in Section 6.3. The network data, once extracted, can be transferred to a mainframe computer and used for Critical Path applications or for generating progress reports for management control.

## 6.2 ALGORITHMS FOR MANAGEMENT NETWORK ANALYSIS:

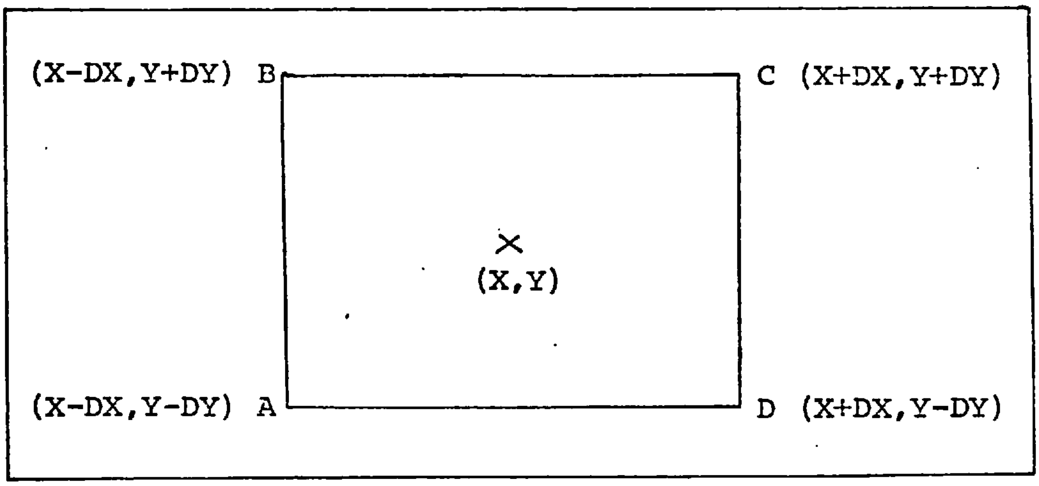
A description of the general graphics principles involved in the management network analysis system is given here. The application programs were described in Section 6.1. under the general title of graphics for networks. The various mathematical concepts were developed on the basis of the ease from the user point of view and interaction was considered to be the leading criterion.

To generate a rectangle, a horizontal rectangle is completely and sufficiently defined by specifying one of its diagonals. Given the points (X1, Y1) and (X2, Y2) as below define



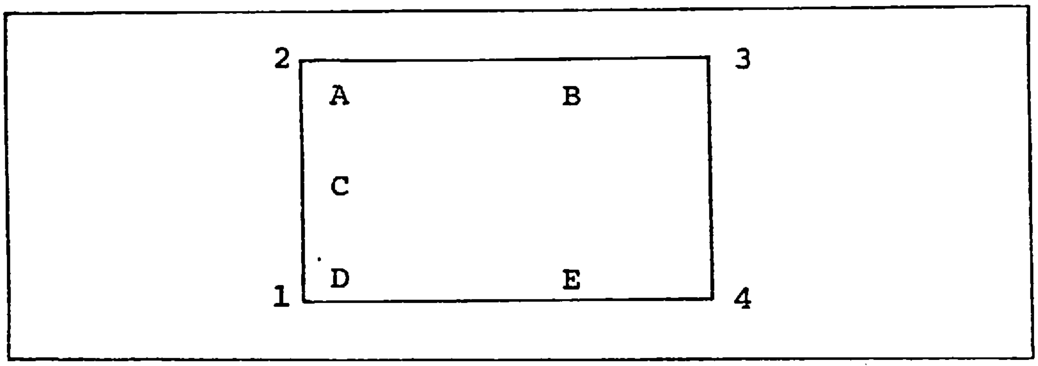
the rectangle ABCD where the co-ordinates of B are given by (X1, Y2) and those of D as (X2, Y1).

To locate a given point (X, Y) on a non-uniform grid of size DX, DY, it is important to appreciate that X, Y must lie within the grid box as defined below by A, B, C, D.



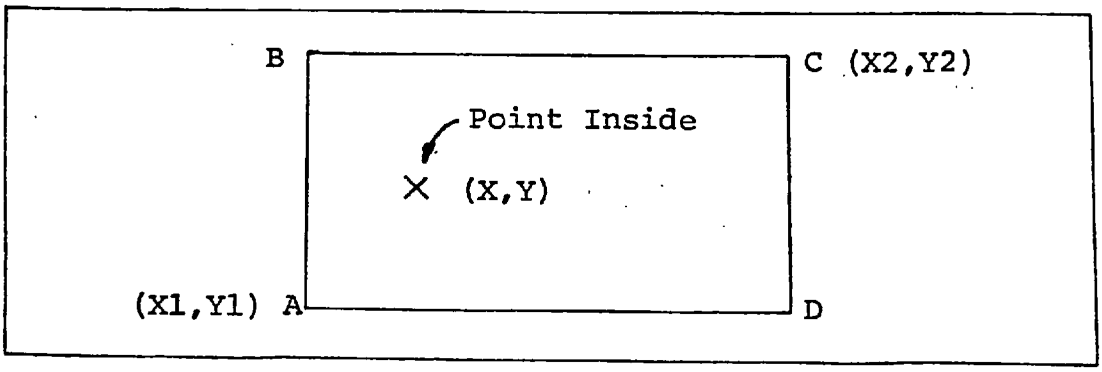
Then the offsets are calculated from the nearest points on the grid and hence the cursor is controlled to move on a grid.

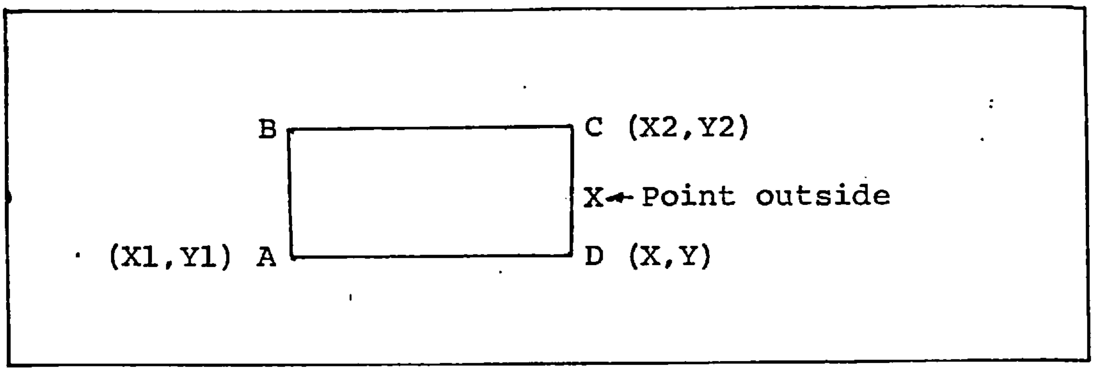
To place different alphanumeric data within a given activity, it would be desirable to set up some rules that would be followed. This was achieved by taking proportions of the sides of the rectangle and then calculating the start and end points of alphanumeric data which in this case would always be horizontal. The taking of proportions implies normalised sides i.e. the size of the rectangle becomes immaterial and data retrieving eventually becomes relatively simple e.g. if the standard activity is taken to contain five codes as follows:



Then side 12 would be sub-divided into 5 sections and the codes A and B would start at the beginning and in the middle of the rectangle respectively.

The best means of locating an existing rectangle in a network was considered to be defining one point anywhere inside it and then carrying out the following analysis:-





The condition to extract the correct values of (X1, Y1) and (X2, Y2), the necessary and sufficient condition would be that

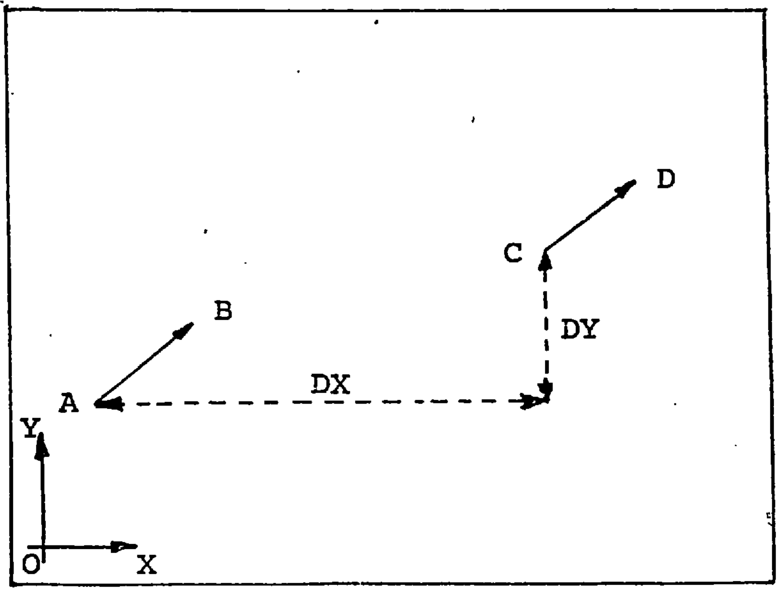
$$X1 \leqslant X \leqslant X2 \quad - ①$$

and $$Y1 \leqslant Y \leqslant Y2 \quad - ②$$

The criterion is the intersection sub-set of ① and ② above.

This method of locating rectangles provided a means of inputting connecting events where the start and finish activities could be located and a logical event connection put in automatically.

To allow for general translation, scaling and rotation of picture components or macros, the following principles were employed:



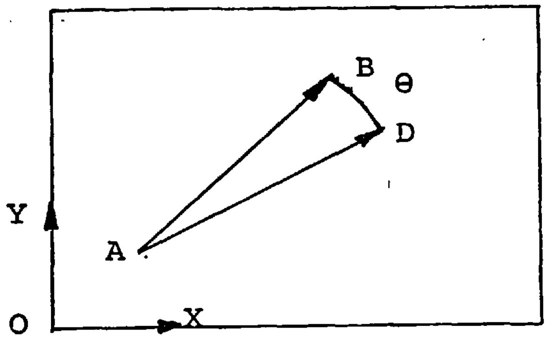
The translation of vector AB to new position as vector CD may be carried out by the following calculation:

XC=XA+DX

YC=YA+DY

and so on for all the co-ordinates within the picture component.

b) Rotation:



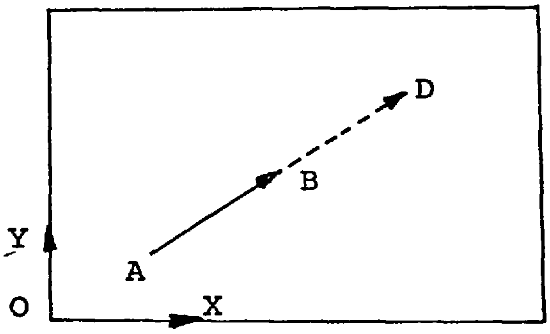
The rotation of a vector from AB to AD by an angle θ is

$XD=XB*COS\theta + YB* SIN\theta$

$YD=YB*COS\theta - XB* SIN\theta$

if the origin is taken to be at A.

c) Scaling:

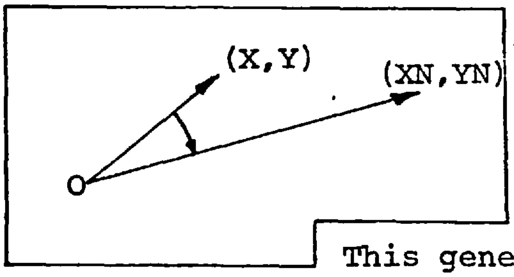


The magnification of a vector AB to AD, considering the origin to be at A, would be given by:

XD=XB*SCALE

YD=YB*SCALE

Hence to perform a general translation by DX, DY followed by a rotation clockwise through angle θ and magnification by a scaling factor M for any co-ordinate vector (X,Y) in two-dimensional space would be carried out as follows:

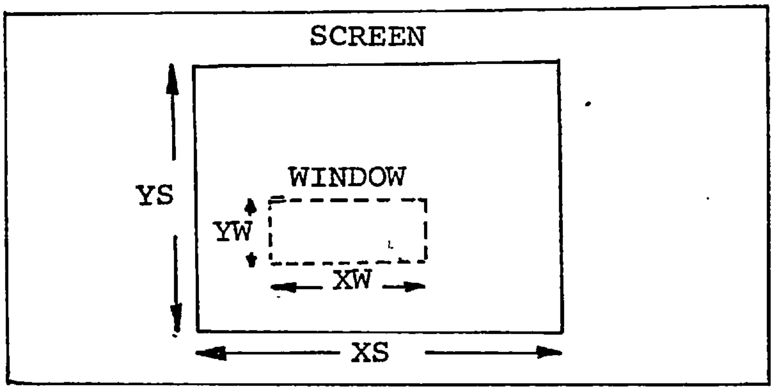


$XN=(X+DX)M\ COS\theta + (Y+DY)M\ SIN\theta$

$YN=(Y+DY)M\ COS\theta - (X+DX)M\ SIN\theta$

This general calculation formed the basis of all picture component manipulations.

The magnification or the windowing routine was based on taking proportions of the window with respect to the screen size.



The window co-ordinates were so constrained that XW and corresponding YW held the same ratio as the screen dimensions

i.e. $\frac{XS}{XW} = \frac{YS}{YW} = M$

and the constant of proportionality thus was the magnification factor. The vector magnification was then carried out as described before taking offsets from the origin and then multiplying the offset by the scale.

The co-ordinate manipulations described led to the basic graphics package for the management networks. One of the leading requirements from the system was minimisation of cross-overs of different events, i.e. a network which would approach as near as possible to a "planar" network. This would provide for graphic clarity as well as better visualisation of the different events. Various algorithms for the same were considered and are described in the next section.

## 6.3 MINIMISATION OF CROSS OVERS IN A NETWORK:

The problem of embedding a graph in a plane arises in several fields. In engineering, discovering whether a given circuit may be laid out in a plane is of interest in integrated circuit design. In chemistry, determining isomorphism of chemical structures may be made much easier if the structures are planar.

The earliest characterization of planar graphs was given by Kuratowski[8], who showed that every non-planar graph contains a subgraph which upon removal of a branch is planar. However, searching for such subgraphs may require an amount of time at least proportional to $n^6$ where n is the total number of vertices in the graph. It is clear that more efficient procedures are needed to analyse large graphs.

Using list-processing and various programming tricks, the search time may be shown to be proportional to $n^3$. Tarjan[9] has programmed an algorithm giving a time bound of $n^2$. Hopcroft and Tarjan[10] have proposed, what is considered to be probably the fastest, an algorithm which carried out planarity testing in n log n steps.

The proposed algorithm, in this thesis was, developed, not only to check for planarity, but rather to find a solution as near to planar as possible. This would obviously be the more practical approach. Almost all the algorithms published so far indicate how to test for planarity but the proposed algorithm would provide an optimum near-planar solution if it was found that the graph could not be embedded in a plane.

The basic concepts originated while considering management networks. A few examples here clarify some of the ideas:

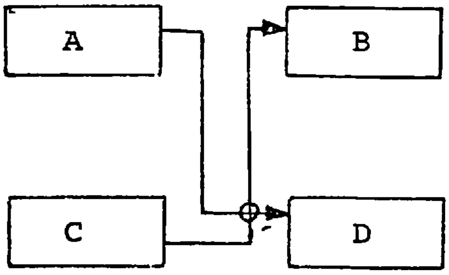


Fig. 6.3.1.

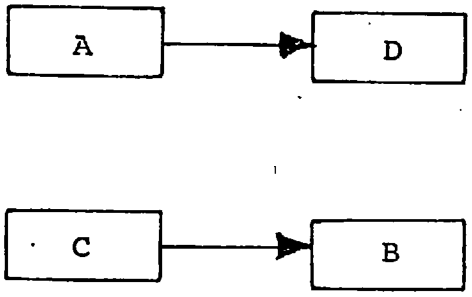


Fig. 6.3.2.

If this network in 6.3.1. was re-drawn as shown in fig.6.3.2., obviously the picture would be much clearer. One fact that becomes clear is that the vertical offset is zero for the planar solution i.e. zero cross over.

Now considering the situation as follows, it becomes obvious that the problems get more complicated but it is still possible to find a planar solution which would be as in fig 6.3.4.

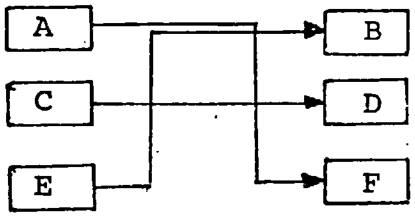


Fig. 6.3.3.

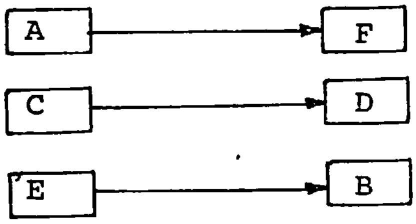


Fig. 6.3.4.

A few constraints of the system deserve a mention here:

1) The activities are allowed to exchange places with each other only vertically. They probably fall on a time-scale horizontally.

2) The logical connections indicating the precedence of the activities have to be maintained.

3) The graphical shapes of the connecting events have to be redrawn every time for the sake of clarity.

Next, consider a situation where it will not be possible to reach a planar solution.

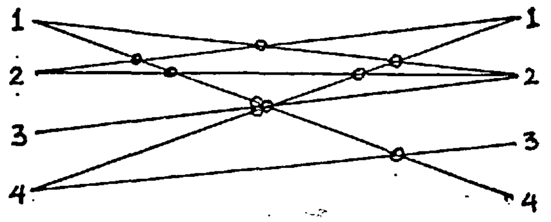
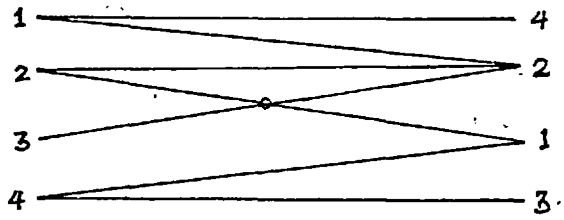


FIG. 6.3.5.

The near-optimum solution of fig 6.3.5 could be as follows:



The problem of finding a solution of near-minimum crossovers in a network has wider possibilities. Most printed circuit boards use both sides of the board and use copper plated conducting wire holes for transmission of signals. Whereas, the signal paths on either side can be etched rather inexpensively, the wire holes require special procedures for drilling and plating. So a method of minimising the conducting wire holes in a P.C.B. layout is desirable within the system of constraints imposed by shortest path analysis and signal transmission times. This also finds applications in the layout of pipes and conduits where the bends require special treatment and are obviously much more expensive than the straight lengths.

From cost optimisation angle, cost is usually a function of path length and the number of cross-overs or bends that exist in a layout. In this section, the minimisation of cross-overs only is considered with the direct application to management networks where the path lengths hold no physical meaning. Later the combination of optimisation of shortest path lengths and minimum cross-overs will be considered as applied to P.C.B.s and pipe routing.

### 6.3.1 PLANARITY SOLUTION:

The integrated circuit designers have endeavoured to answer the following question: Is it possible to find procedures which would enable a computer to solve efficiently path connection problems inherent in logical drawing, wiring diagramming, and optimal route finding?

The following types of problems need solving:

1) To find a path between two points so that it crosses the least number of existing paths.

2) To find a path between two points so that it avoids as much as possible preset obstacles such as edges.

3) To find a path between two points so that the path is optimal with respect to several properties; for example, a path which is not only one of those which cross the fewest number of existing paths, but, among these, is also one of the shortest.

Various methods for solving the problem mentioned in 1) are described in this section.

In processing information consisting of patterns, rather than numbers or symbols, on a digital computer, the problem to be solved is, how a computer, without sight and hearing, can be made to deal competently with situations which appear to require co-ordination, insight and perhaps intuition.

### 6.3.1.1 ALGORITHMS:

Most of the layout problems require algorithms; given a network, one may ask if the network has a certain property, and an algorithm is to provide the answer. Since graphs are widely used as models of real phenomena, it is important to discover efficient algorithms for answering theoretical questions. Hopcroft and Tarjan[10] have presented an algorithm to determine whether a network can be embedded, without any crossing edges, in a plane. The planarity algorithm may be viewed as an iterative version of a recursive method originally proposed by Auslander & Porter and correctly formulated by Goldstein. The algorithm uses depth-first search to order the calculations and thereby achieve efficiency. Depth first search or back-tracking has been widely used for finding solutions to problems in combinatorial theory and artificial intelligence. Recently, this type of search has been used for solving several problems in graph theory, including finding biconnected components, finding triconnected components, finding strongly connected components, finding dominators and determining whether a directed graph is reducible.

Embedding a network in a plane has several applications. The design of integrated circuits requires knowing when a circuit may be embedded in a plane. The importance of the problem is suggested by the number of published planarity algorithms. The earliest characterisation of planar networks was given by Kuratowski.[8]

### 6.3.2 PROPOSED ALGORITHM:

A heuristic approach is taken for finding an optimum solution to the minimum cross-over problem. The algorithm is based on a simple list processing technique and the result in mind is; given a management network layout, how to modify the network, within a given set of logical constraints, so as to achieve minimum cross-overs. This has wider applications in the field of layout of printed circuit boards. The algorithm attempts to reduce the degree of non-planarity in a given network.

Most layouts, it is expected, would be defined in a grid based system, though that in no way is a hinderance to the working of the algorithm. The network is reduced to a simple connectivity diagram from which the relevant data for various lists is acquired. The diagram is iteratively re-drawn as the end nodes exchange positions in 2-D space until a planar or an optimal non-planar solution is found.

### 6.3.2.1 THE PROBLEM:

The nature of the problems involved in the minimisation of cross overs is directly dependent on the need for optimisation. The need exists because the minimisation would aid clarity in a network. It would also help a smoother flow of the network. It provides for the ease in management control of a given project and would aid a more accurate estimation of the resources required for the project.

The minimisation problem is further defined by the fact that the connected activities are constrained to move only vertically because they probably lie on a time scale horizontally, e.g. they might comply with a bar chart. The other constraint is that logical connections, e.g. finish-to-finish or finish-to-start etc., must be maintained. Correct graphical representation is required. Hence the main requirement for a minimisation algorithm would be to exchange positions of activities vertically in order to achieve a near-optimum solution.

The minimisation problem may then be defined as describing a means of exchanging any given activities vertically maintaining the connections and the graphics so as to achieve minimum cross overs, given a general set of activities connected by a given set of events.

### 6.3.2.2 THE SOLUTION: *

Assuming that most of the layout would be done on a grid, though that in no way is a constraint, it is possible to draw a simple connectivity diagram indicating the activities as nodes and the events as branches. If the start and end nodes of a branch are numbered logically in descending order, then Fig.A network is fully represented by a list diagram as in Fig.B.

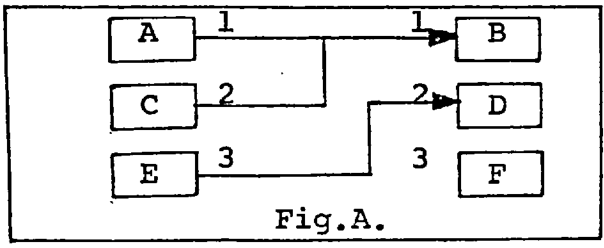


Fig.A.

From a connection representation of the type shown in Fig.B, a list processor can be designed to obtain a minimum cross over situation.

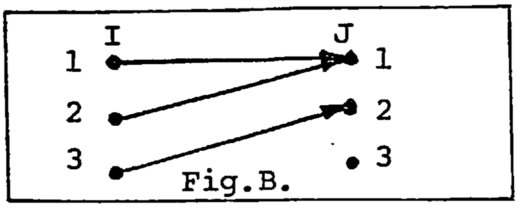


Fig.B.

* For specification of the Algorithm, See Appendix 'B'.

The constraint put on the system is that Array I would represent the fixed components in space whereas Array J, where the connections end will be the one which could be swapped around for optimisation.

The data used in lists is extracted in the form of a list J of end nodes, a list JPT of pointers to J indicating the number of branches terminating at any particular node in J and a list I of pointers to the start nodes which end at any given J. Hence it is concluded that list J will have as many components as there are end points, list JPT will have the same number of entries as list J, each entry indicating the number of branches and list I will have the same number of entries as list J since every end point corresponds to a start point.

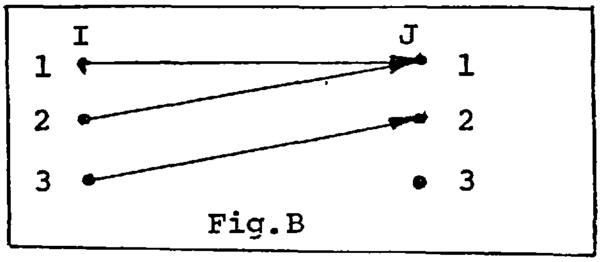


Fig.B

As an example, consider the connectivity diagram of Fig.B. The following list can then be extracted:

The arrows in Fig.C. represent the logical linkage of the lists. List J indicates that there are 3 end nodes in the diagram.

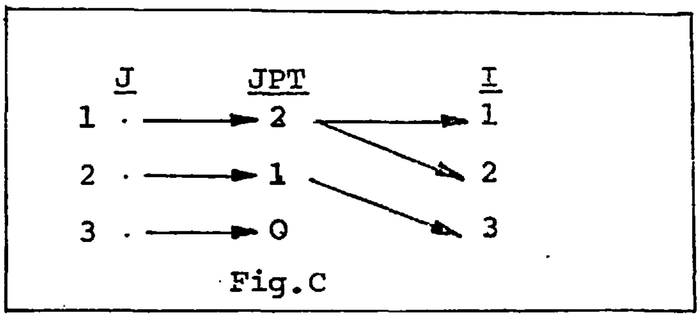


Fig.C

List JPT indicates that 2 branches end at node 1, 1 branch ends at node 2 and no branches come to node 3. Furthermore, list I indicates that the two branches that end at node 1 start at nodes 1 and 2 and that the branch that ends at node 2 started at node 3 and there is no entry for end node 3 since, as the list JPT indicates, there is no branch connected to that node.

In fact, the structure of the lists is almost as if the connectivity diagram was reversed. The main part of the list processor is that everytime an end node is swapped in list J, corresponding adjustments are made to the lists JPT and I.

A cross-over in this case is defined as a four-bounded condition as explained with reference to Fig.D. below:

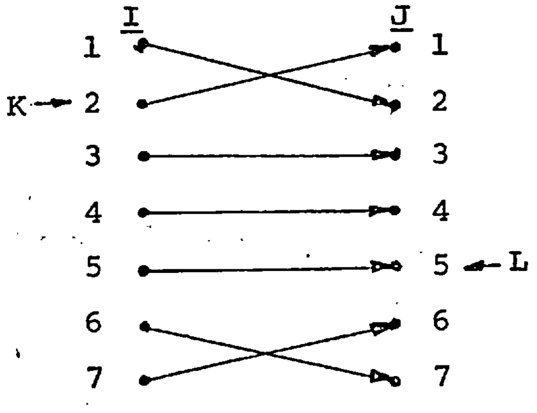


Fig.D.

Two counters, K and L are maintained along the lists I and J corresponding to a connectivity diagram. It is also ensured that K and L never equal each other, then a cross over is defined by the condition that if I(K), i.e. starting node I corresponding to counter K, is greater than I(L) and if J(K) is less than J(L), then J(L) has to be greater than or equal to I(K) and that J(K) has to be lesser than or equal to I(L), for a cross-over to occur.

The basic technique used is that every element of list J is swapped "sequentially" with every other element of list J, the corresponding modifications made to JPT and I, and unless the new number of intersections is less than before the swap, the lists are swapped back to the original status. So if there are n components in the list J, this involves n(n-1) number of computations but since the data structure is very simple and the check for intersections is a single statement, the algorithm provides very fast computation.

Considering some of the algorithms in the literature, which take, just for planarity testing, computations proportional to $n^2$ and higher powers of n, the proposed algorithm is much faster. As stated before, the real advantage lies in leaving a minimal cross-over situation if the planarity test failed. The algorithm works even faster in reality because searching is terminated as soon as a planar graph is found. It is expected that for most networks, to reach planarity, the average number of computations required would be of the order of $\frac{n(n-1)}{2}$.

The solution as found using the proposed algorithm for the connections depicted in Fig.D. was as shown in Fig.E. below:

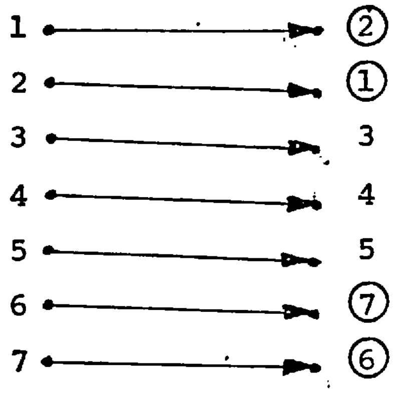


Fig.E.

The main points to note about the solution are:

a) the encircled nodes represent the end points which have moved from their previous position.

b) the minisation may be seen as decreasing the vertical offset of any given branch between its starting and end nodes.

c) the solution for this particular case was planar, i.e.

The final number of intersections = Ø.

d) the planar solution was found in 20 steps of 'swapping' instead of n(n-1), i.e. 42 steps for a 7 node network.

The speed of the algorithm cannot be increased by reducing the processing, i.e. for example, an alternative approach to the algorithm might have been to proceed to a succeeding node everytime a positive swap was done to obtain a new minimum. This could reduce the number of computations far below the stated value of n(n-1). Enough special cases could be thought up where this approach would not lead to the required optimisation.

In its application, the algorithm would be applied to "column-pairs" of activity networks to make as thorough a search for minimum cross overs as possible. The search could be carried out by traversing the network from left to right or from right to left or both. This process does imply its own disadvantages.

During processing, the algorithm would proceed along a minimisation curve as illustrated in Fig.A.

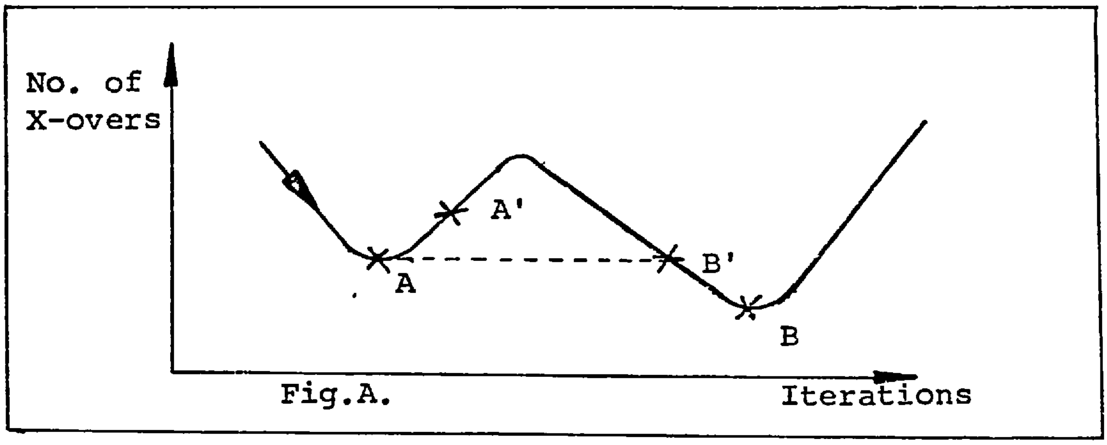


Fig.A.

Main disadvantage of the sequential processing as proposed is that, taking the Fig.A. as an example, during processing, the algorithm might arrive at a false minimum such as A, try another iteration represented by A' and give up. Obviously, the solution is not an optimum one because the real solution is at stage B.

The requirement is that A should be translated into B', i.e. over the top of the curve. This could be easily done by making the system interactive since the human eye is a very good judge of patterns. The other automatic alternative is to take the nodes in pairs and swap as node-pairs to try to achieve a minimum. Once the algorithm reaches on or below point B', it will definitely get to B which in this case is the real optimum solution.

Great advantage could be taken of an interactive system in such an optimisation algorithm. Drastic changes can be carried out to the layout very quickly interactively and the new data thus could be fed to the algorithm for retrial and a better solution. The main advantage of the algorithm is that planar solution would terminate any further trial and greatly aid the speed of execution of the algorithm.

6.3.2.3 THE GRAPHICS:

The graphics application to the interactive graphics system 'PM' could be broken down into two parts:

1) Conversion of activity precedency diagram into linked lists consisting of start and end points of connecting events.

The next step would be to carry out the minimisation based on these lists.

2) The conversion of the optimised lists has to be reflected into the diagram. So the activities, as represented by nodes in the lists, are translated vertically into their new positions with cross overs minimised.

1) Conversion of precedence network into linked lists:

For a network of activities and events, the principles involved can be illustrated by taking the following diagram as an example:

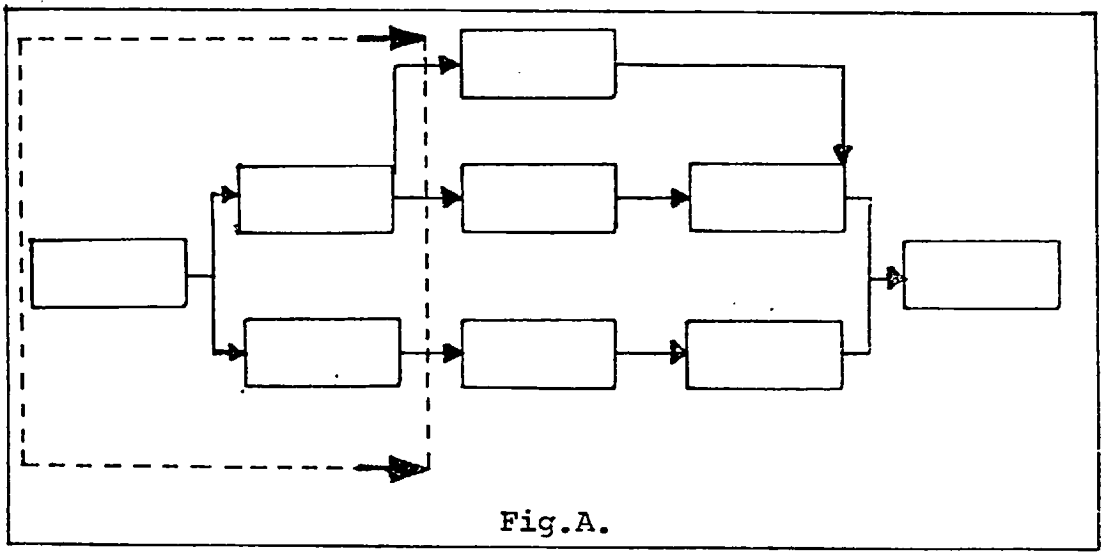
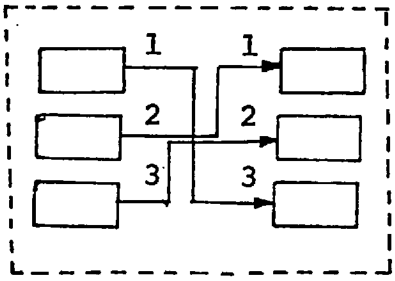


Fig.A.

An imaginary window may be placed around two concurrent columns of inter connected activities. Considering the horizontal flow of the network, the window could travel towards the right going down through various sets of columns of activities. To extract the three lists as already described, consider a diagram of the following type within the imaginary window: .



The data structure in which the activities and events were stored on the disk file within the computer system were in order in which they were generated. Hence the order was not compatible with the ordering of the lists. So the first thing to do was to extract the start and end co-ordinates of all arrows as two variable size arrays, in the order in which they were generated.

| Start points | End points |
|---|---|
| X1, Y1 | X3, Y3 |
| X2, Y2 | X1, Y1 |
| X3, Y3 | X2, Y2 |

The suffices reflect the order in which the lists would be required. Hence, a "sort" was carried out on the arrays thus formed of start and end points, independently for both the arrays. The algorithm used for this sort may be defined as follows:

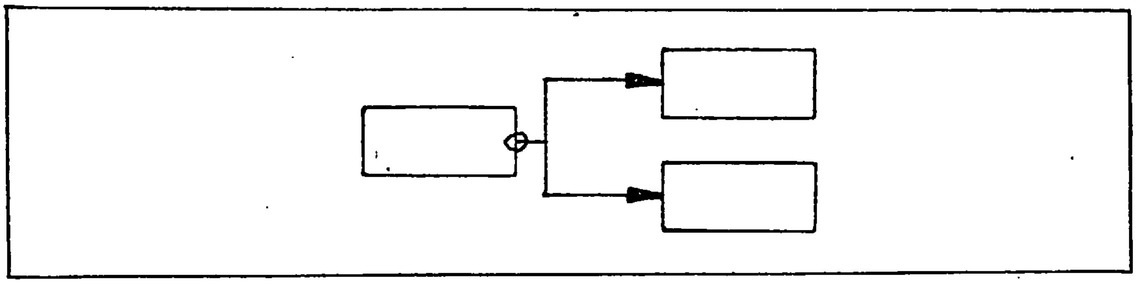
* Scan the array and extract the largest element of the array.

* Store this quantity away. Initialise this element to the smallest possible number.

* Scan the array in a loop according to the steps defined above.

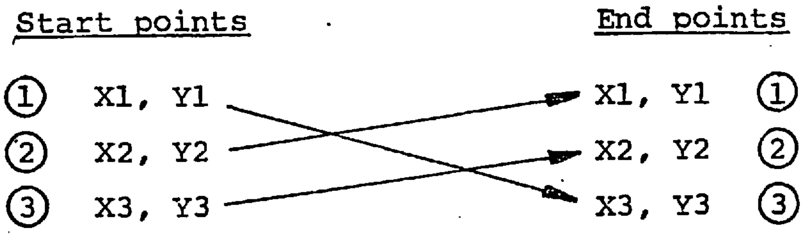
However, there is a special case caused by two machine input start points being in exactly the same position. The same applies to end points as well.

This is best illustrated by the following diagram:



The y co-ordinate for both the arrows is exactly the same and the algorithm fails when it carries out a "greater than" comparison. The technique used to get over this problem was to use "perturbation". This process may be described as going through the array, before submitting it to "sort", looking for any elements which may be exactly equal and adding a small quantity like $10^{-10}$mm to it. This would ensure that no two elements would ever be equal during array processing for ordering. Also, the graphics displays would not be affected because such a small quantity as $10^{-10}$mm would be impossible to detect. Hence perturbation technique was of great use and combined with ordered lists formed the basis of minimisation algorithm.

So, after the perturbation first and then the ordering, the following lists could be extracted:



The start and end co-ordinate lists now could be submitted directly to the minimisation algorithm:

| J | JPT | IPT |
|---|---|---|
| 1 → | 1 → | 2 |
| 2 → | 1 → | 3 |
| 3 → | 1 → | 1 |
| End points | No of arrows | Start points |

The final output from the proposed algorithm would be:

| J | JPT | IPT | |
|---|---|---|---|
| 1 → | 1 → | 1 | ② |
| 2 → | 1 → | 2 | ③ |
| 3 → | 1 → | 3 | ① |
| End points | No of arrows | Final Start points | Initial Start points |

So, the final lists achieved with the pointers leading from the end points to the start points lead to what is a network with minimum cross-overs. As far as the graphics display is concerned, this is done "backwards" i.e. in fact, the start points remain fixed and the end points are swapped around in "Move Rectangles" mode.

The arrow that was connected to the start point marked by suffix 2, i.e. X2, Y2 would have its end point moved to the end point related to the start point marked by suffix 1. The final result would look like:

|  | Start points |  | Final<br>End points | Initial<br>End points |
|---|---|---|---|---|
| ① | X1, Y1 | → | X'1, Y'1 | (X3, Y3) |
| ② | X2, Y2 | → | X'2, Y'2 | (X1, Y1) |
| ③ | X3, Y3 | → | X'3, Y'3 | (X2, Y2) |

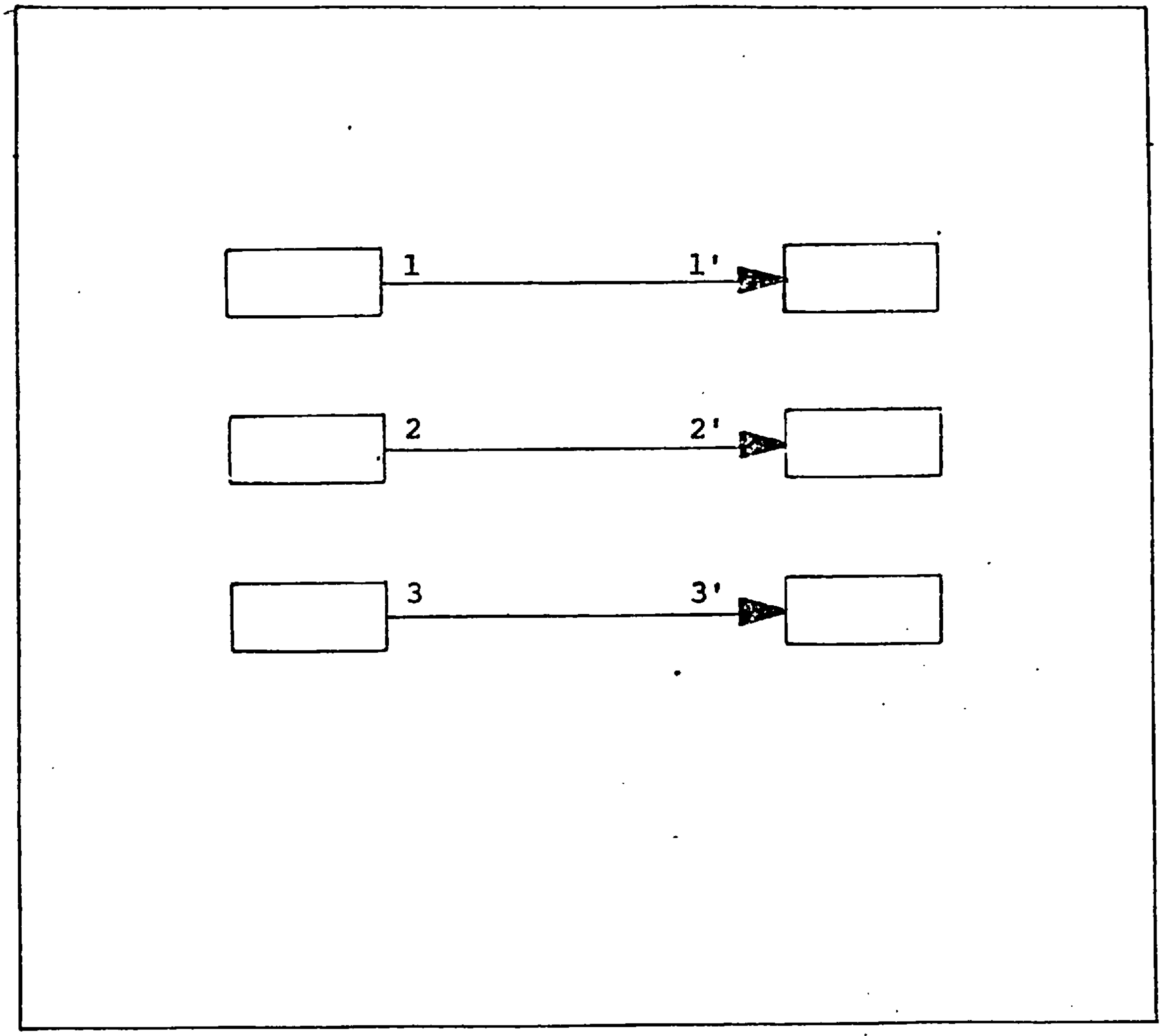
The translation in graphics terms is easily carried out by calculating offsets in the X and Y directions:

e.g. $DX = X3 - X1'$

$DY = Y3 - Y1'$

This would lead to the correct placement of the activity as it "should" be connected to the preceding activity marked by X1, Y1.

The final graphics display could be easily extracted now from the final lists. Hence the solution is:



However, there are special cases to be considered. The main problem arises in splitting of the arrows vertically as shown below diagrammatically:

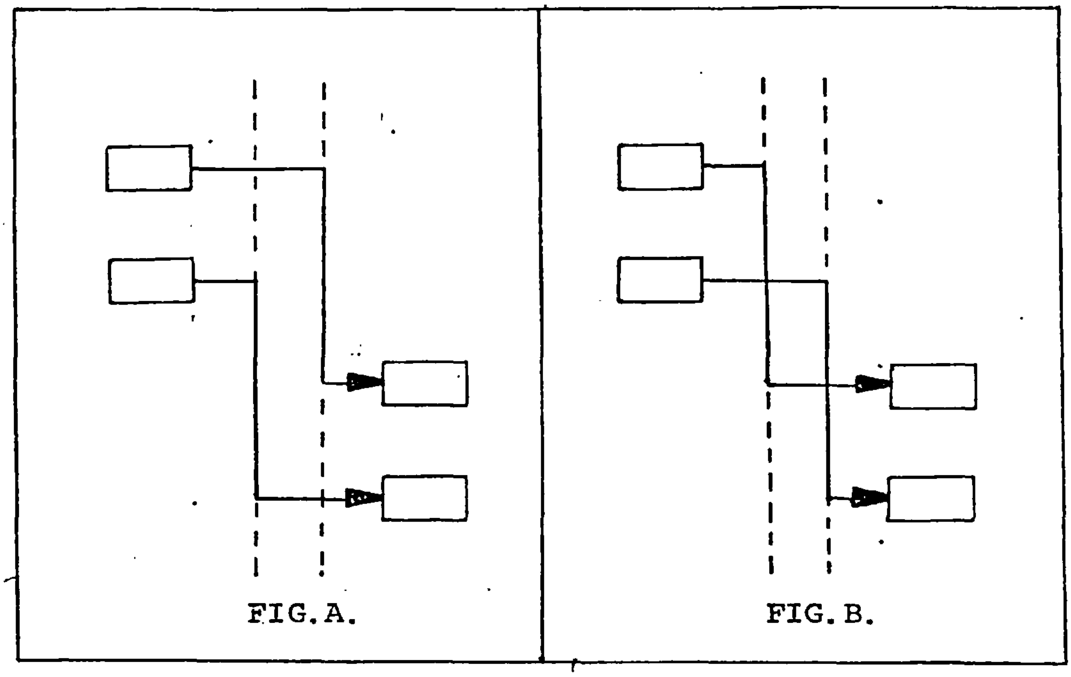
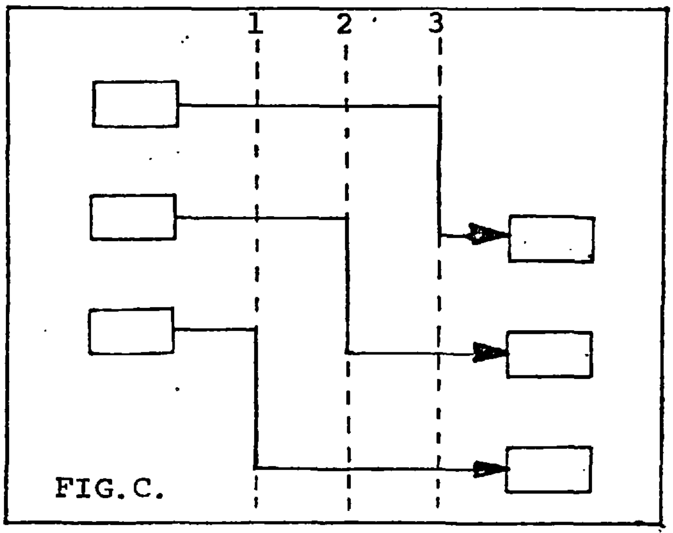


Fig A. indicates the correct positioning of events where no "false" crossover is caused whereas in Fig B. two crossovers are caused by the reverse position. Herein comes the other application of graph theory where the vertical subdivision of the event has to be a "descending" order to avoid any "false" crossovers.



Another fact that is clarified from the Fig C. is that the space between activities should be divided up into the number of events and then vertical layout may be done in the descending order, but that the activities can then be compressed together.

Certain columns from the grid on which the events are placed can be taken away without affecting the overall picture. Fig.C. illustrates this where the columns ① and ② can be dispensed with without any loss of clarity or any adverse effect on the layout of the network.

### 6.3.3 DISCUSSION:

Various minimisation algorithms were considered in this section, in connection with their specialised application to the minimisation of crossovers in Precedence Type Management Networks. Compared with the existing algorithms, a minimisation algorithm was proposed which had the following main advantages:

1. It was particularly suitable for the interactive real time system, as in this case, the user would always have full control of the progress of the algorithm and the computer would aid the design process.

2. The algorithm was very suitable for the type of co-ordinate data that formed the basis of the graphical picture for the network system.

3. Another advantage of the algorithm, by the very nature of linked lists that it uses, the processes involved in converting the co-ordinate data structure into linked list data structure and vice-versa were relatively easy.

4. The correlations of the lists with the pointers helped general physical appreciation of the events connecting various activities.

5. The list processor required for the minimisation was simple to comprehend. Since most of the processing was done in core, small amounts of disk I/O were required within the computer system, and this reduced processing time considerably.

6. Since the algorithm was in generalised list data structure, it could easily find applications in other fields of engineering like minimisation of conducting wire holes in Printed Circuit Boards and the same for number of bends in Pipe Layout Problems. Some of these applications would be considered in greater detail later.

The main disadvantage of the system was considered to be slightly larger number of iterations required to reach the solution as compared with some of the algorithms described in the literature. This was more than compensated by the fact that the algorithm provided a "measure" of the degree of non-planarity which other methods did not. Furthermore, since the computations were straight conditional directives, although large in number, were very fast and hence it did not really affect the overall performance. Combined with various other advantages mentioned above, the algorithm was found to be very effective for the minimisation of crossovers in management network diagrams.

Having successfully generated the management network diagrams and extracted the relevant precedence information for further analysis, the next step was to improve the quality of the graphical displays by minimising the crossovers in the network. Since, so far, only the two-dimensional layout problem had been considered, the time had come to generalise the layout further.

Double sided Printed Circuit Boards, such as those used in modern electronic equipment and sophisticated computers, posed what may be loosely termed as the 2½-dimensional layout problem: Not only do the printed circuit boards have conductors running on both sides of the board, the conductors are also allowed to "cross over" from one side to the other.

The conductor paths are usually drawn on a board and then 'etched' into position but the crossovers or the 'conducting wire holes' require more expensive processes. The holes have to be drilled in and then plated on the inside to provide for conduction. Having already solved the general problem of minimisation of crossovers in layouts, it was considered viable to stretch the algorithm to 2½-dimension cost optimisation of printed circuit boards as described in the next chapter.

# 7. PRINTED CIRCUIT BOARD LAYOUTS

## 7.1 2-D AND 2½-D PROBLEMS:

There are two dimensional layout problems which can be clearly defined within the biaxial co-ordinate system (X,Y) and these can be usefully solved in relation to engineering problems such as management networks, single sided printed circuit boards or layouts of various shapes within a given area such as layouts of rooms in a building or layouts of roads or train lines or aircraft routes. Such problems can be classified in various classes and solutions exist for most of these problems, either in Graph Theory or in Classical Mathematics.

The three dimensional problems are slightly different in nature but they lend themselves to a co-ordinate system of (X, Y, Z) and probably reflect more of the general engineering applications. The major layout problems in this field relate to pipe layout problems in big buildings, chemical plants or ships. These problems will be considered in more detail in Chapter 8.

A special class of problems arises in layouts, which though lends itself to (X, Y) co-ordinate system but cannot be fully defined without interaction of various planes. For example, take a pipe layout of a multi-storey building. If various floors are considered independent of each other, the layout problem can be reduced to two dimensions but for the interaction between different floors.

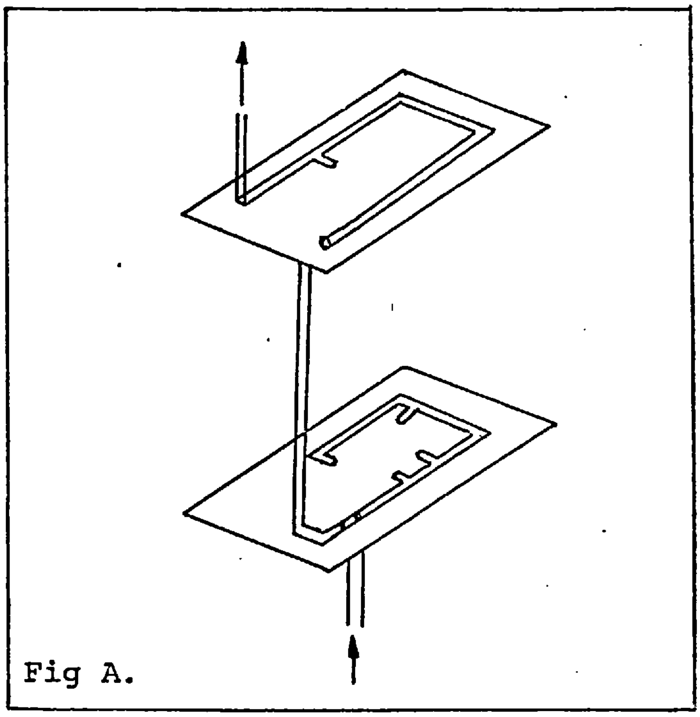


Fig A.

Fig A indicates how the 3-dimensional problem can be split up into parts and solved as a 2-dimensional problem. This, however, becomes difficult as the problem grows in size and complexity such as major pipe layouts in modern chemical plants etc.

A simpler problem and more suitable for 'layer' analysis is the one that would define the layout of a double sided printed circuit board with conductors on either side connected by conducting holes drilled through the board or in some cases, by making 'flyover' connections.

Thus, the 2½-dimensional layout problem may be defined as the problem associated with finding paths between a given set of nodes subject to a set of optimisation conditions. The nodes would completely lie within a number of plane surfaces forming parallel or unparallel layers depending on the application.

## 7.2 ALGORITHMS:

The algorithms described in this chapter are the outcome of an endeavour to answer the following question: Is it possible to find procedures which could enable a computer to solve efficiently path-connection problems inherent in logical drawing, wiring diagraming and optimal route finding? The results are highly encouraging. The problem further breaks down into the following subsets:

1. To find a path between two points so that it crosses the least number of existing paths. This problem has already been considered in Chapter 6 and solutions proposed.

2. To find a path between two points so that it avoids as much as possible preset obstacles such as edges. This part of the problem has wide implications.

3. To find a path between two points so that the path is optimal with respect to several properties; for example, a path which is not only one of those which cross the fewest number of existing paths, but among these, is also one of the shortest.

Various existing algorithms are considered and a new, more practical one perhaps, is proposed. In processing information consisting of patterns, rather than numbers or symbols, on a digital computer, the computer, without sight and hearing, has to be made to deal competently with situations which appear to require co-ordination, insight and intuition. The problem is to find efficient procedures which, if followed by the machine, would lead to an optimal solution. With sufficient care, it is possible to make a problem such as this unambiguous. In most cases, however, it would be too great a struggle just to present the problem in a way that is completely and consistently stated.

Within this class of problems is the shortest-path problem on which there have been earlier definitive algorithms for finding shortest paths by Dantzig[11], Ford and Fulkerson[12] and Moore[13].

The layout problem for pipes and printed circuit boards has the following important points in common:

1. The paths have to be as short as possible. In pipes, pipe lengths contribute directly to the total cost, the cost is, in fact, linearly proportional to the pipe lengths. In printed circuit boards, however, it is not so much the cost that is related to the lengths, but the fact that signals could be delayed or weakened by long conductors.

2. The number of bends have to be minimum. In pipes, certain bends have to be heat bent or need special processing and cost a lot of money. Similarly, conducting wire holes in printed circuit boards are expensive.

3. The paths usually have a certain order of priority. In pipes, some pipes may be made of more expensive material than others and so it would be important in selecting the priority levels. Similarly, conductors may have ordering in terms of layouts.

4. There is the criterion of obstacles in the path. Some pipes would have to go through prespecified ducts or others may have obstacles or no-go areas before the pipes can be laid out, for example, in a building, pipes would be restricted to going through the walls and they could not go through the rooms. Similar conditions would apply to conductor paths where conductors would not be allowed to touch.

## *The Definition of the 'Minefield'

1. The minefield is the area forbidden for the path finding algorithm i.e. a step must not finish within the "mined" area.

2. The minefield can have any shape or size.

3. The minefield is only effective on a given plane. Pathfinding can continue normally on any other plane.

4. The path can be around a minefield or under/over it in a different plane.

5. The cost minimisation, the main criterion of an algorithm for layouts, would be a consequence of the interaction between the above limitations specified together with any other constraints placed on an individual application.

The following problem is "proposed" which combines the above constraints and will eventually lead to a practical optimisation algorithm for path layouts:

1. Consider a man in a mine-field with a fixed step-length.

2. He is only allowed to travel along X or Y axis.

3. He has to traverse a given number of paths joining any two nodes except that he has to traverse some paths before others.

4. Obviously, he is not allowed to step into the mine-field which will explode.* He is, however, allowed to take flying jumps or dig underground holes to take him distances multiples of his step-length.

5. He has to keep to the shortest possible path and satisfy the above conditions. He is to avoid taking jumps or digging in relation to a factor, "the optimisation factor", in preference to taking slightly longer paths.

6. The above problem simulates any layout problem whether it be printed circuit board layout in 2½ dimensions or a 3-dimensional pipe layout.

## 7.3. COST OPTIMISATION CRITERIA:

A large number of optimisation problems are mathematically equivalent to finding shortest paths in a graph. Consequently, shortest path algorithms have been worked over more thoroughly than any other algorithm in graph theory. Some of the proposed algorithms are better than others, some are more suited for a particular structure than others, and some are only minor variations of earlier algorithms. For a good comparative study of various shortest path algorithms, a survey paper by Dreyfus can be recommended.

There are different types of shortest path problems. Most frequently encountered among these are the following:

1. Shortest path between two specified vertices.
2. Shortest paths between all pairs of vertices.
3. Shortest paths from a specified vertex to all others.
4. Shortest path between specified vertices that passes through specified vertices.
5. The second, third and so on shortest paths.

In the worst case, type 1 is identical to type 3, because in the process of finding the shortest path from a specified vertex to another specified vertex, the shortest paths to all other vertices may have to be determined.

The problem of finding the shortest path from a specified vertex 's' to another specified vertex 't', can be stated as follows:

A graph G of n vertices is described by an n by n matrix D = [dij], where

dij = length of the path from vertex i
to vertex j ; dij $\geqslant \emptyset$,
dii = $\emptyset$
dij = $\infty$, if there is no path from i so j.

In general, dij $\neq$ dji, and the triangle inequality need not be satisfied, i.e. dij + djk may be less than dik. In fact, if the triangle inequality is satisfied for every i,j and k, the problem would be trivial because the direct path (x,y) would be the shortest path from vertex x to vertex y.

The distance of a directed path P is defined as the sum of the lengths that make up P. The problem is to find the shortest possible path and its length from a starting vertex 's' to a terminal vertex 't'. Among several algorithms that have been proposed for the shortest path between a vertex pair, perhaps the most efficient one is due to Dijkstra.[14]

Dijkstra's algorithm labels the vertices of the given graph. At each stage in the algorithm, some vertices have permanent labels and other temporary labels. The algorithm begins by assigning a permanent label $\emptyset$ to the starting vertex s, and a temporary label $\infty$ to the remaining n-1 vertices. From then on, in each iteration, another vertex gets a permanent label, according to the following rules:

1. Every vertex j that is not yet permanently labelled gets a new temporary label whose value is given by

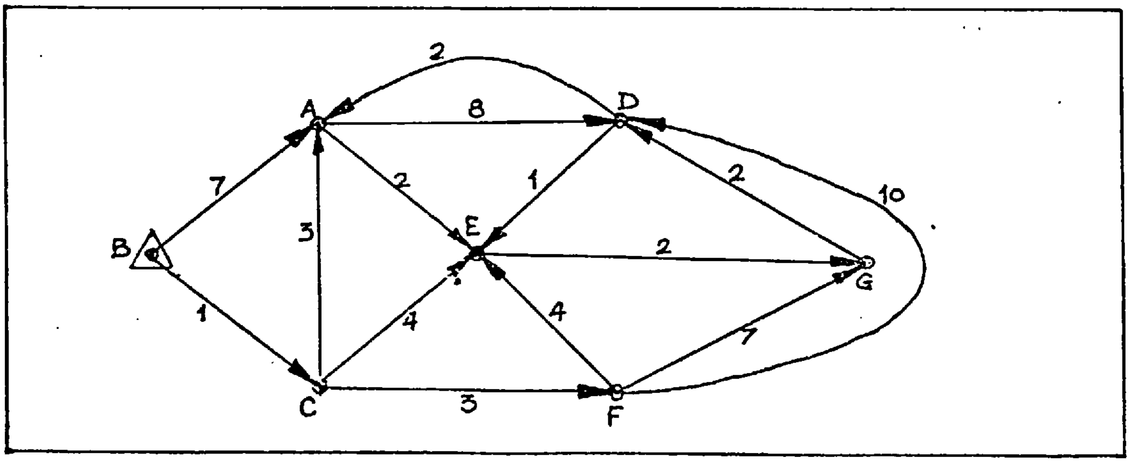
Min $\left[\text{Old label of j, (Old label of i+dij)}\right]$

Where i is the latest vertex permanently labelled, in the previous iteration, and dij is the direct distance between vertices i and j. If i and j are not joined by an edge, then dij $=\infty$.

2. The smallest value among all the temporary labels is found, and this becomes the permanent label of the corresponding vertex. In case of a tie, select either one for permanent labelling.

Steps 1 and 2 are repeated alternately until the destination vertex t gets a permanent label. The first vertex to get a permanent label is at a distance zero from s. The second vertex to be permanently labelled, out of the remaining n-1 vertices, is the vertex closest to s. From the remaining n-2 vertices, the next one to be permanently labelled is the second closest vertex to s. And so on. The permanent label of each vertex is the shortest distance of that vertex from s. This statement can be proved by induction.

As an illustration of Dijkstra's procedure, the distance from vertex B to G in the following diagram may be found:

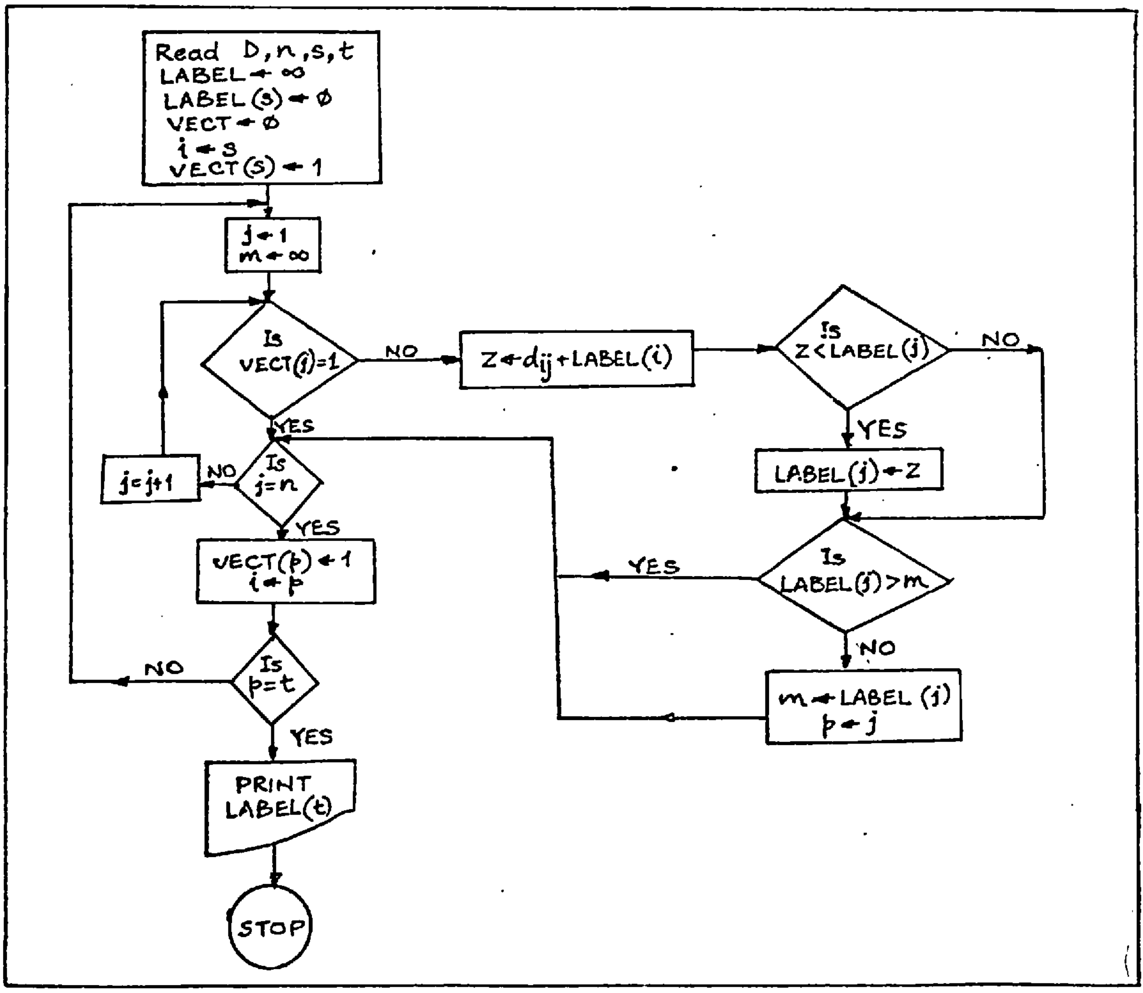


Let a vector of length 7 indicate the temporary and permanent labels of the vertices as the solution is discovered. The permanent labels will be shown enclosed in a square, and the most recently assigned permanent label in the vector is indicated by a tick. The labelling proceeds as follows:

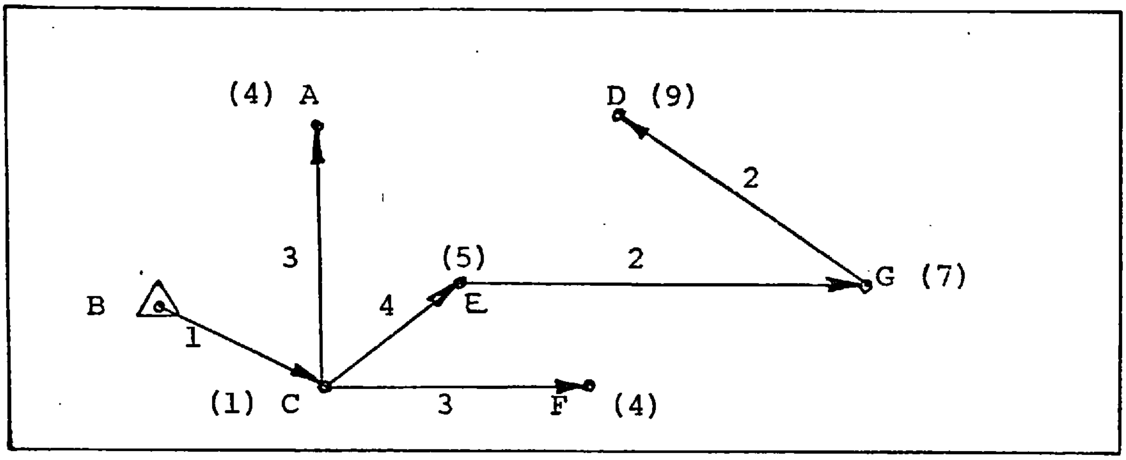
| A | B | C | D | E | F | G | |
|---|---|---|---|---|---|---|---|
| ∝ | $\boxed{0}$ ✓ | ∝ | ∝ | ∝ | ∝ | ∝ | : Starting vertex B is labelled O. |
| 7 | $\boxed{0}$ | 1 | ∝ | ∝ | ∝ | ∝ | : All successors of B get labelled. |
| 7 | $\boxed{0}$ | $\boxed{1}$ | ∝ | ∝ | ∝ | ∝ | : Smallest label becomes permanent |
| 4 | $\boxed{0}$ | $\boxed{1}$ ✓ | ∝ | 5 | 4 | ∝ | : Successors of C get labelled |
| 4 | $\boxed{0}$ | $\boxed{1}$ | ∝ | 5 | $\boxed{4}$ ✓ | ∝ | |
| $\boxed{4}$ | $\boxed{0}$ | $\boxed{1}$ | 14 | 5 | $\boxed{4}$ | 11 | |
| $\boxed{4}$ ✓ | $\boxed{0}$ | $\boxed{1}$ | 14 | 5 | $\boxed{4}$ | 11 | |
| $\boxed{4}$ | $\boxed{0}$ | $\boxed{1}$ | 12 | 5 | $\boxed{4}$ | 11 | |
| $\boxed{4}$ | $\boxed{0}$ | $\boxed{1}$ | 12 | $\boxed{5}$ ✓ | $\boxed{4}$ | 11 | |
| $\boxed{4}$ | $\boxed{0}$ | $\boxed{1}$ | 12 | $\boxed{5}$ | $\boxed{4}$ | 7 | |
| $\boxed{4}$ | $\boxed{0}$ | $\boxed{1}$ | 12 | $\boxed{5}$ | $\boxed{4}$ | $\boxed{7}$ ✓ | : Destination vertex gets permanently labelled. |

The algorithm described does not actually lis t the shortest path from the starting vertex to the terminal vertex; it only gives the shortest distance. The shortest path can be constructed by working backwards from the terminal vertex such that the path goes through that predecessor whose label differs exactly by the length of the connecting edge. (A tie indicates more than one shortest paths). Alternatively, the shortest path can be determined by keeping a record of the vertices from which each vertex was labelled permanently.

In this algorithm, if labelling was continued until every vertex got a permanent label, the shortest paths from starting vertex s to all other vertices could be found. A binary vector VECT of order n could be maintained to indicate whether a label was permanent or temporary. A flowchart of the algorithm follows:



A short distance tree for the case under consideration would take the following form:



Some conclusions may be drawn from the above diagram, e.g. the shortest path from B to G was 7 units long and went through B-C-E-G. In this algorithm, as more vertices acquire permanent labels, the number of additions and comparisons needed to modify the temporary labels continues to decrease. In the case where every vertex gets permanently labelled, n(n-1)/2 additions and 2n(n-1) comparisons are needed. Thus the computational time is proportional to $n^2$. The main disadvantage of the algorithm is that if some of the distances are negative, the algorithm will not work. Negative distances in a network may represent costs and the positive ones profits. The reason for the failure is that once a vertex has been permanently labelled, its label cannot be altered. Shortest path algorithms have, however, been proposed that will solve this problem, provided the sum of all dij around every circuit is positive.* The computation time of the existing algorithm that can handle negative dij is $n^3$ and not $n^2$.

* See References Section.

It was suggested by Nicholson[15] that carrying the shortest path algorithm from both ends s and t would improve the speed. Dreyfus,[16] however, has shown that the double-ended procedure would improve the efficiency only in certain types of graphs.

### 7.3.1 BRANCH AND BOUND METHODS:

Branch-and-bound methods use the concepts of trees, logic trees, and bounds to solve combinatorial problems. The method is a powerful alternative to exhaustive enumeration on a computer, since the time and storage requirements for exhaustive enumeration increase exponentially with the number of variables, and even large and fast machines can only handle very small problems. The name 'Branch and Bound' comes from the particular approach used by Little, et.al.[17] in their attempt to solve the famous operations research problem of the travelling salesman.

Branch and Bound methods use "search trees", each node of which represents a class of possible solutions to the problem. The union of all the pending nodes represents the class of all possible solutions. A cost is computed for each pending node. The cost bounds induce an ordering of desirability on the pending nodes, which determines the branching in subsequent steps. The algorithm stops when it is not possible to generate any new node, or a feasible solution with associated cost less than the lower bounds of the pending nodes has been found.

### 7.3.1.1 FORMAL DEFINITION OF THE BRANCH AND BOUND TECHNIQUE[1]

"Let $S = [\sigma j]$ be the set of possible solutions to a problem P of interest. Let $|S|$, the modulus of S be a finite number and f be a function defined on the elements $\sigma j$ of subsets of S. The solution to be found is $\sigma^* \in S$ which minimises the function f, and is feasible i.e. satisfies a set of conditions [C] ".

Suppose the problem has a property which allows for making a partition $\pi$ of a subset $S_{olm\ldots p}^{(i-1)}$ of S

$$\pi = \left[ S_{olm\ldots p1}^{(i)},\ S_{olm\ldots p2}^{(i)}, \ldots, S_{olm\ldots pq}^{(i)}, \right], \quad q > 1 \qquad (11.1)$$

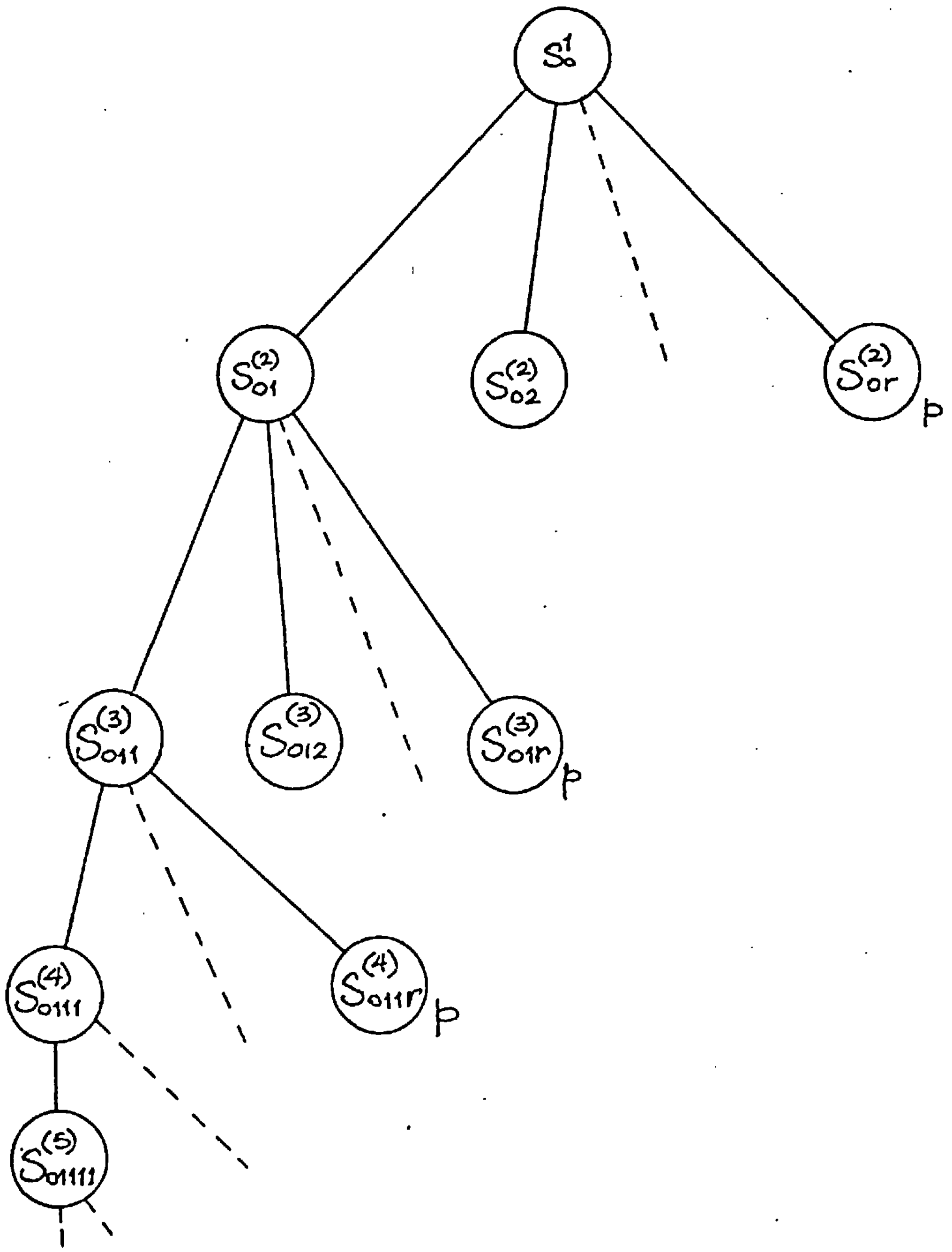
where the subsets are defined by:

$$S_{olm\ldots pk}^{(i)} \neq \emptyset \quad , \quad K=1,2,\ldots,q \qquad (11.2)$$

with the initial condition

$$S_o^{(1)} = S$$

A search tree may be built as follows. Each node bears the name of a subset $S_{olm\ldots pk}^{(i)}$ of solutions of P. The set of level i indices olm...pk indicates a path from the node to which they belong to the root of the tree.



SEARCH TREE FOR PROBLEM P

(Each partition contains r subsets; p indicates pending nodes)

In the above diagram, node $\text{Sollr}^{(4)}$ is at level 4 and the path to the root of the tree is

$\text{Sollr}^{(4)}$, $\text{Soll}^{(3)}$, $\text{Sol}^{(2)}$, $\text{So}^{(1)}$

A closed node in the search is a node that can no longer be partitioned, i.e. cannot have followers. A pending node is one which is not closed. For example, in the above figure, $\text{So}^{(1)}$, $\text{Sol}^{(2)}$ are closed and $\text{Sollr}^{(4)}$ is pending.

Since each subset corresponding to each node is partitioned into two or more non-empty subsets, the modulus of the subsets is monotonically decreasing along a branch of the search tree, i.e.

$$|\text{Sollr}^{(4)}| < |\text{Soll}^{(3)}| < \ldots < |\text{SO}^{(1)}| = |S|$$

$|S|$ is finite, so we will eventually reach a level at which one of the pending nodes $\text{Solm...pq}^{(j)}$ contains only one element of S, i.e., contains a solution to P. This is a terminal node.

The problem is to get the minimal solution $\sigma^*$ by enumerating as few nodes as possible. To do so, at each node, over the subset assigned to the node, an upper and lower bound for function f must be calculated.

The strategy consists of branching from the pending node having the least lower bound. In other words, one uses a property inherent in the nature of the problem to make a partition of the most promising pending node. For a terminal node, the upper and lower bounds collapse to the value of f for the solution assigned to this node. The search is ended when a node contains a feasible solution, the value of which is less than the smallest value of the lower bounds of the pending vertices.

The solution proposed to the "Man in the minefield" problem is a more practical approach to the "Branch and Bound" technique and the function evaluated at every node is related to the length of the path and the number of bends involved.

Several examples of branch and bound methods exist in the literature to emphasise the fact that branch and bound is not one method but a class of methods. The classical Travelling Salesman problem may be taken as an example:

"A travelling salesman, starting in one city, wishes to visit each of n-1 cities once and only once and return to the start. In what order should he visit the cities to minimise the total distance travelled?"

A feasible solution to the problem is a Hamiltonian Cycle which contains every vertex. One of these solutions was total enumeration by Berge.[18] A more sophisticated version of this solution has been put forward by Eastman.[19]

A slightly more complex and a bit more heurestic approach was proposed by Little et al.[17] Many combinatorial problems that arise in engineering or management may be formulated either as integer programming problems; or in the context of decision making as pseudo-Boolean problems. Land and Doig[20] have proposed a branch and bound solution for the related optimisation problem. The program described by Benayoun et.al.[21] for mixed integer programming can handle several thousand constraints and a few hundred discrete variables. (It was written for a CDC 66ØØ). Branch and bound is a structural method containing the seeds of many possible generalisations.

The travelling salesman problem illustrates the compromises between the length of computations at each node (sub-optimisation) and the number of nodes of the search tree that have to be generated in order to obtain an optimal solution. It is also important to understand the structure of a particular problem before designing a branch and bound scheme to solve it. If the structure is poorly understood, this may lead to prohibitive amount of data to be stored.

Heuristic methods differ from the branch and bound algorithms in the sense that one is not sure of having the optimal solution at the end of the search, only a 'good' solution. Heuristic searches have been tried on mathematical models as well as real life problems. Sometimes this is the only way to obtain a solution. Heuristic criteria are often used to accelerate the branch and bound procedure. In the last few years, many applications have been reported in the literature where branch and bound techniques were applied successfully: synthesis of integrated process design; computer aided synthesis of chemical plants; ordering of recycle calculations; project scheduling; modular design; generation of NAND structures.[22]

7.3.1.2 DEPTH-FIRST SEARCH ON A GRAPH:

Depth-first search is a powerful technique of systematically traversing the edges of a given graph such that every edge is traversed exactly once and each vertex is visited at least once. Depth-first search or back-tracking on a graph was first formalised and used by Hopcroft and Tarjan[10] and was subsequently studied by Tarjan.[9] To answer questions of separability, planarity and the like, every edge and every vertex would have to be examined at least once. There are two ways of scanning or searching the edges of a graph:

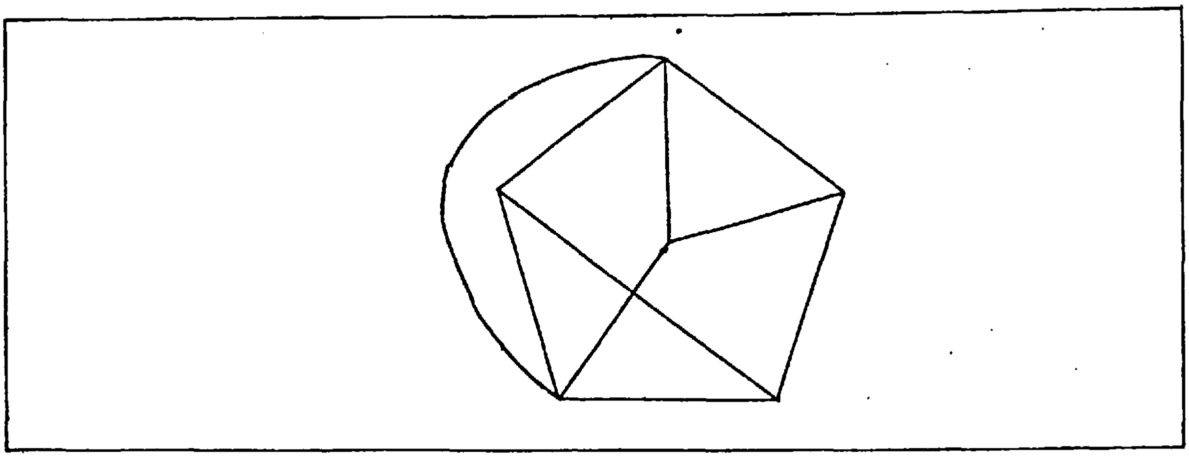
1) Once at vertex v, all edges incident on v could be examined and then the adjacent vertex w could be considered. At w, all edges incident on w could be scanned. This method of considering each vertex in turn is referred to as Breadth-first search.

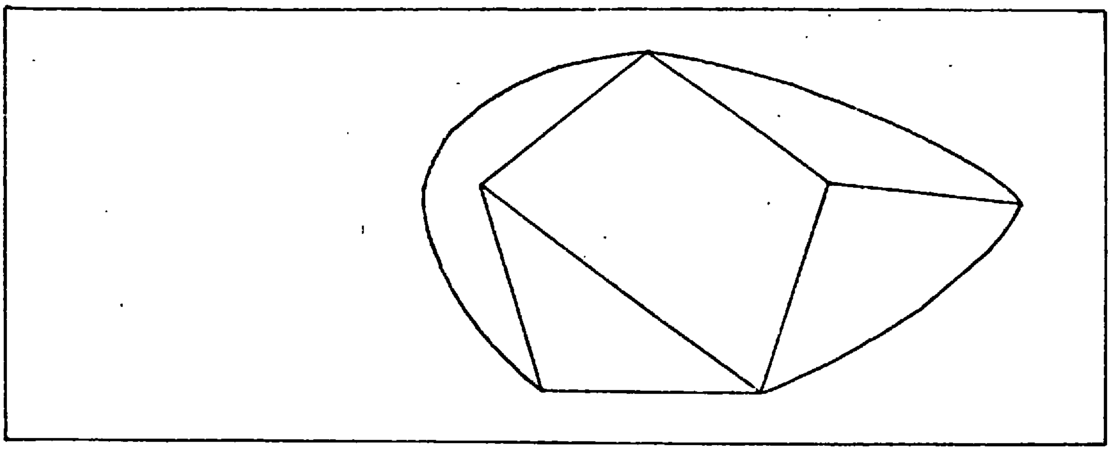
2) An opposite approach is, instead of scanning every edge incident on vertex v, move to an adjacent vertex w, a vertex not visited before, as soon as possible, leaving v with possibly unexplored edges for the time being. In other words, a walk is traced through the graph going on to a new vertex whenever possible. This method of traversing the graph, called the Depth-first search, has been found to be very useful in simplifying graph-theoretic algorithms.

7.3.1.3 PLANARITY TESTING:

The problem of determining whether or not a given graph is planar, is an important one. The planarity characterisations of Kuratowski[8], Whitney[23] or MacLane[24], although theoretically elegant, are unsuitable for testing by a computer. They are difficult to implement; besides, if a graph is planar, these methods do not yield a plane representation, which is often what is needed. It has been shown, for example, that if Kuratowski's characterisation is used to test planarity of an n-vertex graph $(n>5)$, the computation time is at least proportional to $n^6$.

In recent years, many algorithms for planarity testing have been proposed and programmed on computers. Most of these methods employ the map construction approach, which works as follows:
A planar subgraph is first selected and mapped on a plane. Then gradually, the remaining edges are added on, such that no crossings occur. If the reconstruction succeeds, the graph is obviously planar, and a plane representation has been achieved. The only difficult part of such an algorithm is that in the early stages of adding edges, there may be choices available, i.e. ambiguity in placement of edges. A wrong choice made earlier may later prevent addition of an edge, even if the graph is planar. Two mappings of graph represent this problem as follows:





TWO-MAPPINGS OF A GRAPH.

The two mappings represent the problem in the map-construction method of planarity testing, and different methods have been devised to solve it. One such algorithm is due to Bruno, Steiglitz and Weinberg.[25] Probably the most efficient algorithm is the one suggested by Hopcroft and Tarjan.[10] To solve the problem of ambiguity, the following two options are available:

1) Continue adding paths to the basic planar circuit till no path can be added. Then backtrack to explore the alternative choices.

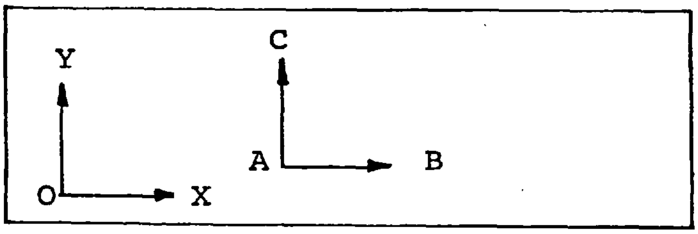
2) Continue to look at different paths but not add them to the basic circuit, till it is found which face a path must be placed in, or it is ascertained that it does not matter which face the path is placed in.

Some algorithms use approach 1, but Hopcroft and Tarjan have used approach 2 and have shown that their algorithm is more efficient because of it. The basis of their algorithm is a list processing technique.

### 7.3.2 PROPOSED ALGORITHM:*

The following set of rules provides the solution to the above defined "Man in the Mine-field". problem:-

1. The man would travel, i.e. take steps only in accordance with a fixed 2-co-ordinate axes, i.e. for the example shown, he may only go to B or C from any given position A. This would define a grid of the size AB.

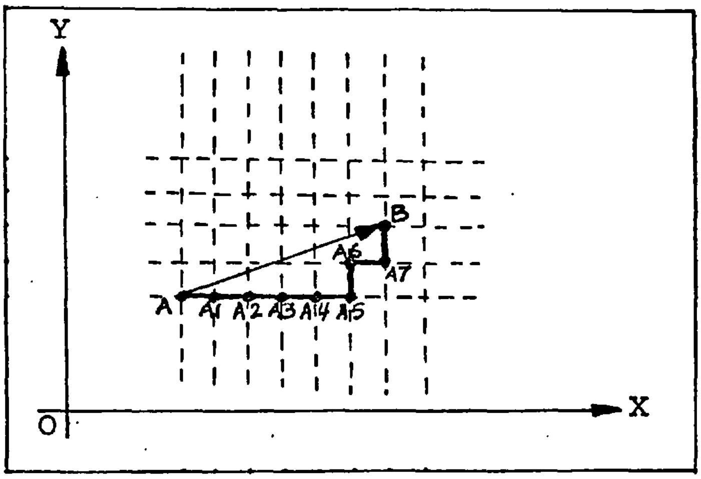


For simplicity, the grid may be assumed to be square, i.e. AB = AC. This would be provide for linearity in the problem, i.e.

Distance travelled along an axis
=(Number of steps) X(step length).

2. The man would consistently take steps along the dominant axis from his start to the destination point. The new dominant axis will be worked out after every step.

This rule may be described graphically as:



The path traced by the man to get from A to B will be A,A1,A2,A3, A4,A5,A6,A7,B. Since at A the dominant direction of travel would be along the X-axis. This would hold good until the man got to A5. From A6 to B, both X and Y directions have equal dominance.

* For Specification of the Algorithm, see Appendix 'B'.

A "preferred" direction has to be chosen which in this case was taken to be the X-direction. Same condition was applied at A6.

3. Within the structure of the rules defined above, the man would have traced the shortest "possible" path. Obviously the shortest path would be the straight line joining A and B but the grid based solution provides many other advantages as seen later.

If the offsets ABx and ABy are taken into consideration:

By Pythagoras $AB = \sqrt{(ABx^2 + ABy^2)}$

The actual path length, however, is

$$= ABx + ABy$$

Proportion of extra path length $= \dfrac{(ABx + ABy) - (ABx^2 + ABy^2)^{\frac{1}{2}}}{(ABx^2 + ABy^2)^{\frac{1}{2}}}$

$$= \frac{ABx + ABy}{(ABx^2 + ABy^2)^{\frac{1}{2}}} - 1$$

For minimisation of the extra path length,

$$\frac{ABx + ABy}{(ABx^2 + ABy^2)^{\frac{1}{2}}} - 1 = \emptyset$$

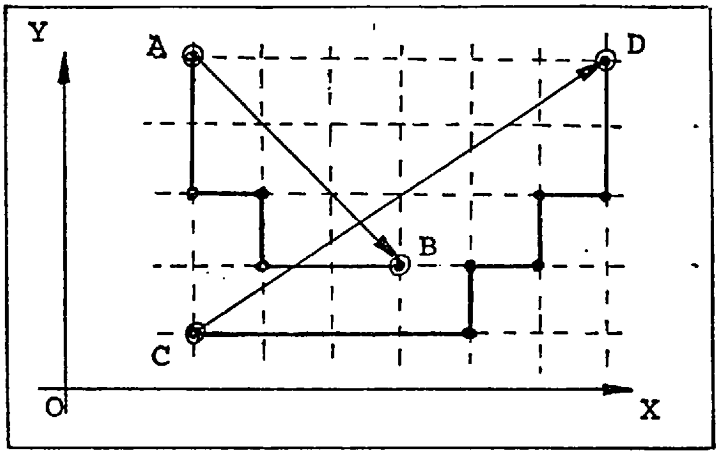
i.e. $ABx + ABy = (ABx^2 + ABy^2)^{\frac{1}{2}}$

i.e. Either $ABx = \emptyset$

or $ABy = \emptyset$

This leads to the conclusion that the above rule would actually lead to the shortest path when the path to be traced is horizontal or vertical.

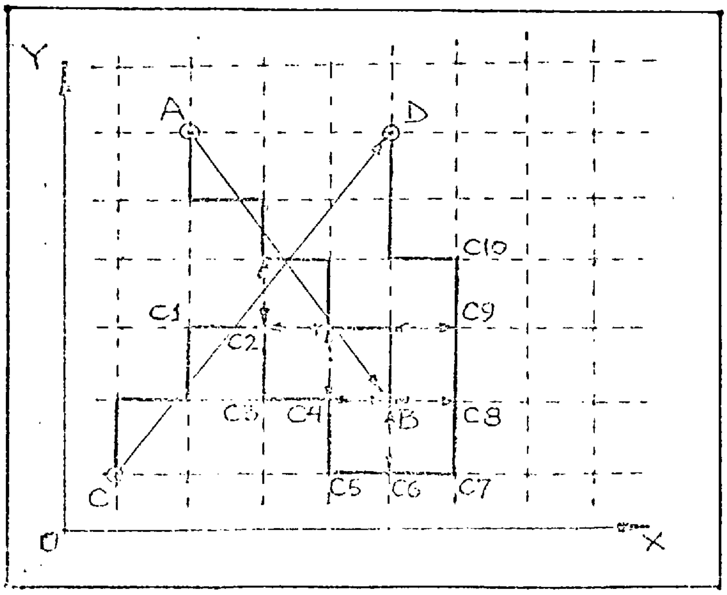
4. To travel along a given number of paths, the man may be made to traverse the paths in the order in which they were indicated to him. This may be graphically seen as follows:



Say the order in which the start and end points are indicated is A,B and C,D. So the man would have to traverse AB first. The solution is suggested as in the diagram.

5. The interaction of the paths has been carefully avoided in the above example but in fact, the first path is an 'obstacle' to the second path. Following is a detailed analysis of the obstacle interaction and how the solution would vary depending on the priority in which they might be traversed.

First take the case where path AB is traversed before the path CD. Traversing the path AB is no problem and is a straight forward calculation of checking the dominance and taking the steps. The problem arises when the man gets to point C2 following the dominant axis.



At this stage, y-axis is the dominant axis and the direction of travel is upwards but the path found for AB poses an obstacle. So the man has to step back to C2 after having attempted to travel in the 'correct' direction. Instead, he is forced to take a step in the minor or non-dominant direction.

When the man gets to C4 which is directly below B, there is no more distance to travel in the x-direction but he still cannot travel directly upwards to get to point D. Two solutions can be taken into consideration here:

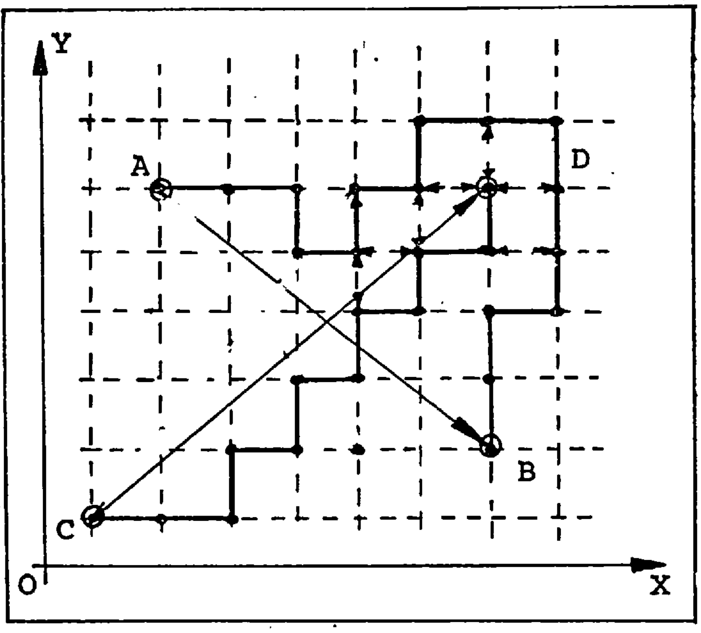
1) The man takes a jump from C4 to C8 or digs a tunnel to get from C4 to C8. The points that lie on the path found for AB represent mines or walls. Then he can travel straight up to D.

2) The other alternative depends on the "risk" or the "cost" of taking jumps or digging tunnels. He may be allowed to travel on in the 'negative-dominant' direction of C4 to C5 depending on this "risk factor". From C5 he continues to strive to get to D in the shortest distance possible. The number of steps that he is allowed to take depends on the relative risk of taking jumps to tracing a longer path. This risk factor defines the "optimisation factor".*

The solution suggested in 1) above is the "2½D layer" solution and that in 2) is the "planar" solution. The decision as to which solution is the "better" of the two depends upon the optimisation factor.

* OPTIMISATION FACTOR = $\frac{\text{COST OF MAKING A HOLE}}{\text{COST OF TAKING A STEP}}$.

If the path CD was laid out before the path AB, i.e. their priorities were reversed, the solution might be different. Considering the original set of co-ordinates again; the path along CD would be traced by travelling along the dominant direction and taking steps along the grid. As described before, there will again be two options for tracing the path AB i.e. one to take a long route around D or the other one to take a bridge to maintain the shortest path.



6. The above possibilities cater for both printed circuit boards which is a 2½-D problem and for pipes in 3-D. In both cases, "priorities" are attached to certain paths.

In printed circuit boards, the time taken for a signal to travel path can be critical. Also different combinations of paths may create 'inductance' problems. Hence it becomes necessary to lay some paths before others and different solutions may be found as described.

However, in pipe layouts the priorities may be assigned to routes for different reasons. For example, in a chemical plant, the material cost of certain pipes may be very high and so it may be decided to lay them out before any of the others for cost optimisation reasons.

The constraints imposed by the above two cases are taken care of in the design of the proposed algorithm. The actual implementation and the graphics of the application of the proposed algorithm to the PCB layout in 2½-D is discussed in the next section.

### 7.3.3. GRAPHICS:

The branch-and-bound detailed programming technique had a lot in common with commercial plotter software. In a controlling program for the pen of a plotter, fastest speeds would be achieved if the pen travelled the shortest possible paths between different points. The pen-up and pen-down conditions would correspond to start and end points of a path on the printed circuit boards. The major difference, however, was in the handling of obstacles or existing paths on a printed circuit board which would not exist on a plotter. Existing lines on a plot obviously do not present any obstruction to the drawing of new lines. The technique for programming could, however, be divided up into two independent approaches:

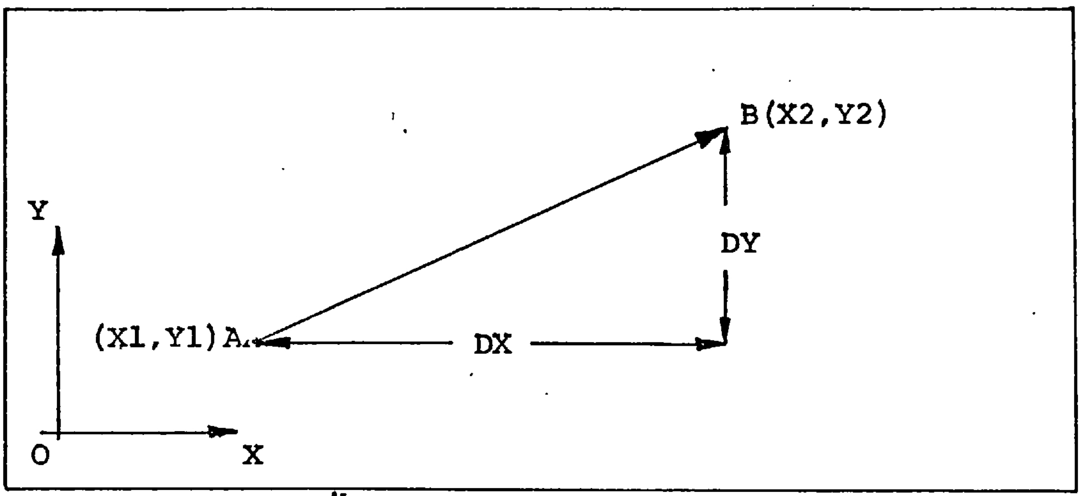
1. Fully-automatic approach: This could be used in a batch processing environment on a large computer.

2. On-line Interactive approach: This would be ideal in a mini computer graphics environment and the operator could interact with the algorithm to produce optimal solutions. The intuitive and pattern recognition qualities of the operator would help the algorithm achieve a better solution than a fully-automatic one.

Both the approaches are considered in detail and the main similarities and differences are pointed out in the following sections:

1. Fully-automatic layout technique:

To find a path from point A (X1,Y1) to B (X2,Y2), the only assumption made is



that both A and B lie on a grid. If the grid is equally spaced in X and Y, let the grid size be GR. The following calculation may then be carried out:

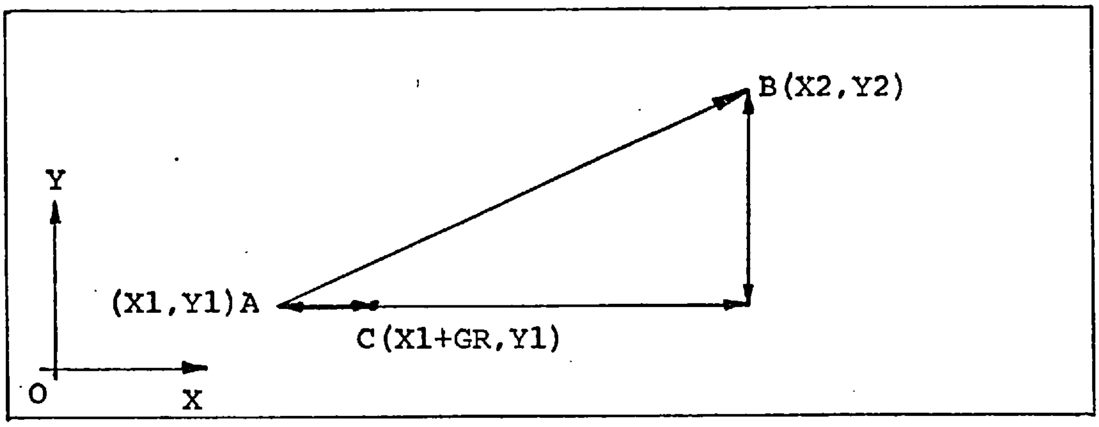
$$DOMX = \frac{DX}{GR}$$

$$DOMY = \frac{DY}{GR}$$

Now if DOMX $\geqslant$ DOMY, the condition would be that the dominant direction is along the X-axis.
A few properties of the above calculation, without the need for pattern recognition, are:

a) The grid was assumed to be equi-spaced only for convenience, in fact, the grid sizes along X and Y axes could be different without affecting the processing as long as the step length in each direction was equal to the grid size in that direction. Thus normalised calculations could be carried out independent of the grid.

Having decided upon the direction of dominance, a step equal to the grid size could be taken in that direction from the start point. So the point found would be C, whose co-ordinates would be (X1+GR,Y1).



The check would have to be carried out to see if the procedure had arrived at the end point B. This condition would only be satisfied if both the offsets in X and Y directions between B and C would be zero.

i.e. If X1+GR = X2

and Y1 = Y2 .

If this condition is not satisfied, in other words, if the end of the path has not been reached, the condition to check next is that the point reached does not lie in an inaccessible area. This can be checked by using a search routine to see if the new co-ordinates lie on an existing grid node, i.e. one which already has a branch beginning or ending on it. There is no possibility of crossing over an existing path since the step length has already been fixed by the grid size.

If the point does lie on an existing path, the step is traced back to the original one, otherwise the new point is stored away and path finding continued from this as the old point. However, if the point does lie on an existing path and has been traced back, the next step is to swap the dominances of X and Y axes. Therefore, if the step was initially taken in the X axis, and retracted, the new step is taken along the Y-axis. If the same happens again, i.e. the new point still lies on an existing path, the step is traced back.

Depending upon the optimisation factor already input, the path can go back towards its origin. However, given an optimisation factor of 1, i.e. conducting wire holes cost the same as a unit path length, the decision made is to go through the plane onto the other side. In pipe routing, this would be equivalent to a pipe bend and the path could be traced along a different axis.

However, if the new point does not lie on an already existing path, it can be stored as the next valid point on the path and the search continued till the path from start to end has been fully traced.

2. On Line Interactive approach:

The basic needs of the graphics system for the inter-active approach would be almost identical to the fully automatic approach. The main·advantage of this technique is that an operator can "help" the algorithm along to achieve, perhaps, a better solution than the fully automatic approach.

The usefulness of this technique is that, effectively, it provides for a continuously variable optimisation factor, i.e. the operator would always have the choice whether to select a conducting wire hole or a longer path. The path finding search would be carried out in the same way as before except that when a new point lay on an existing path, the operator would be able to choose whether to retract on the already found path, i.e. take a longer route or to go across to the other plane or layer.

It may then be concluded that fairly simple graphics overlays are required to use the proposed algorithm for path finding application to Printed Circuit Board layouts.

## 8. PIPE ROUTING AND DESIGN

### 8.1. COST OPTIMISATION IN PIPE LAYOUTS:

In all engineering design problems, the final problem is always the cost evaluation. A good design would have to be optimum with respect to development and production costs. Some of the basic requirements of such a cost analysis are considered here as applied to pipe routing.

1. Pipe Length:

Different types of pipes cost different sums of money. Pipe cost per unit length would be a single largest contributory factor to the cost. The basic pipe cost would then depend upon the following factors:

a) Material of the pipe.
b) Diameter of the pipe.
c) Wall thickness.
d) Stock Quantities.
e) Handling Costs.

Information about pipe lengths could be held as cost per unit length of the pipe (say £ C/meter). This could be held as a data file within a computer system. All pipe lengths would be calculated in metres (PL metres) as

$$PL = \sqrt{(X2-X1)^2 + (Y2-Y1)^2 + (Z2-Z1)^2}$$

Then the total cost of a pipe system would be

$$TC = \sum C_1 * PL_1 + C_2 * PL_2 + \ldots.$$
$$= \sum_{n=1}^{n} C_n \, PL_n$$

2. Machine Time:

Every pipe, once its length has been calculated, will go through a standard machining process which might consist of:

a) Retrieving pipe from store.
b) Pipe cleaning.
c) Measure the length and saw.
d) Pipe bending.
e) Welding.
f) Finishing processes.

A rough cost estimate is required for all the above processes and cost may be assigned to:

| | |
|---|---|
| Feed | (£ CF/m) |
| Rotation | (£ CR/degree) |
| Bending | (£ CB/degree) |

These costs would be worked out for every pipe and added to give a total cost for machining time (£ CM/pipe).

3. Heat Treatments:

This cost may be split into two parts:

a) If a pipe requires heat treatment before installation, this must be assigned a cost, say £ CHP/m, and taken into account. For most pipes, this cost would be zero.

b) A pipe may have to be heat bent because of radii involved in connection with pipe diameter and wall thickness. This cost may be proportional to the degree of bending required or may in fact be independent of the actual bending angle.

For this simplified analysis, we may assume that all heat bends cost the same money, say £ CHB/number of bends or if it is important to take into account the bending angle, then consider the cost of heat treatment per degree, £ CHT/degree and multiply by bending angle BA.

:. Total heat treatment cost may be calculated as:

$$£CH = £CHP * PL + £CHB * NB + £CHT * BA$$

This cost will be added to the total cost of the designed pipe.

4. Cable Radii:

There may be costs directly proportional to the cable radii depending on what type of cables are required. One factor may be £ CC/m i.e. cost of cable as a function of its diameter. The other cost may be dependent on the type of insulation required, say £ CI/m. For a conduit length PL, the cost due to different cable radii may be calculated as:

$$£CCR = £CC * PL + £CI * PL$$

This again has to be added to the total cost for minimum cost conduit to be designed.

5. Diameter/Bending Radii Relationships:

It is assumed that small diameter pipes can be bent to almost any radius, whereas, there would be a limit on how small a radius a large pipe can be bent to. Say the pipe diameter is PD and the bending radius is BR, then the ratio PD/BR may have an upper limit, UL.

So $\frac{PD}{BR} \leqslant$ UL for satisfactory bends.

There may be costs associated with this, e.g. a pipe may have to be redesigned to satisfy the above criterion.

So if $\frac{PD}{BR}$ = PBR (Pipe Bend Ratio),

the cost £CPDB = £CPBR * PBR.

This cost would not be significant in most pipes.

6. Flange Joints:

Various types of flanges are in use and each has a different cost associated with it. These costs would be held as data files within the computer system. The cost of every flange, £CF multiplied by the number of flanges NF would be added to the total cost of a pipe in the form of cost of flange joints:

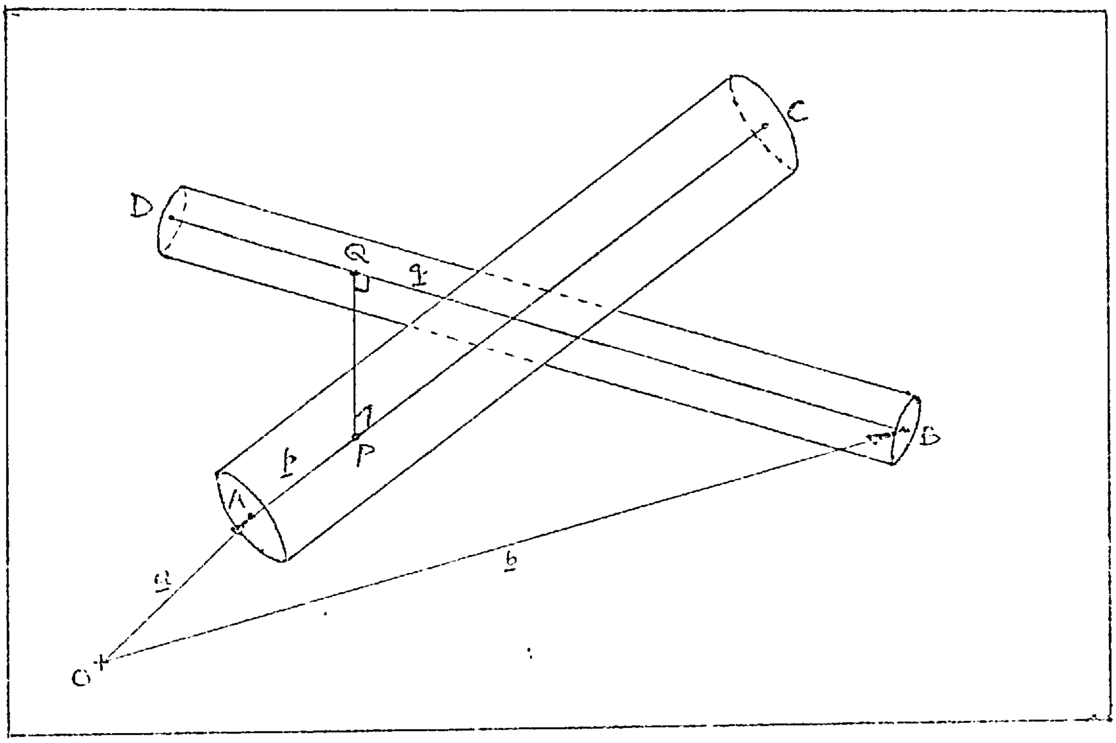
£CFJ = £CF * NF

Similar criterion would hold for pipe junctions. The total cost of the final pipe system would be thus calculated and minimised.

## 8.2 MATHEMATICS OF PIPE ROUTING

### 8.2.1 CALCULATION OF INTERSECTION CO-ORDINATES

This section shows how the intersection co-ordinates are calculated between two pipe lengths A and B.



PQ is perpendicular to A and B. Vectors $\underline{a}$, $\underline{b}$ are the vectors from the origin to A and B respectively and $\underline{p}$ and $\underline{q}$ are unit vectors in the direction of pipes AC and BD. Since PQ is perpendicular to $\underline{p}$ and $\underline{q}$

$$\underline{a} + \lambda\underline{p} - (\underline{b} + \mu\underline{q}) = \lambda'(\underline{p} \times \underline{q}) \qquad (1)$$

where λ and μ are constants of proportionality.

To eliminate μ, take the dot product of (1) with $\underline{q} \times (\underline{p} \times \underline{q})$. This leads to:

$$\underline{a}.\underline{q} \times (\underline{p}\times\underline{q}) - \underline{b}.\underline{q} \times (\underline{p} \times \underline{q}) + \lambda\underline{p}.(\underline{q} \times \underline{p} \times \underline{q}) = 0 \qquad (2)$$

$$\therefore \quad \lambda = -\frac{(\underline{a} - \underline{b}).\underline{q} \times \underline{p} \times \underline{q}}{\underline{p}.\underline{q} \times \underline{p} \times \underline{q}} \qquad P = \underline{a} + \lambda\underline{p}$$

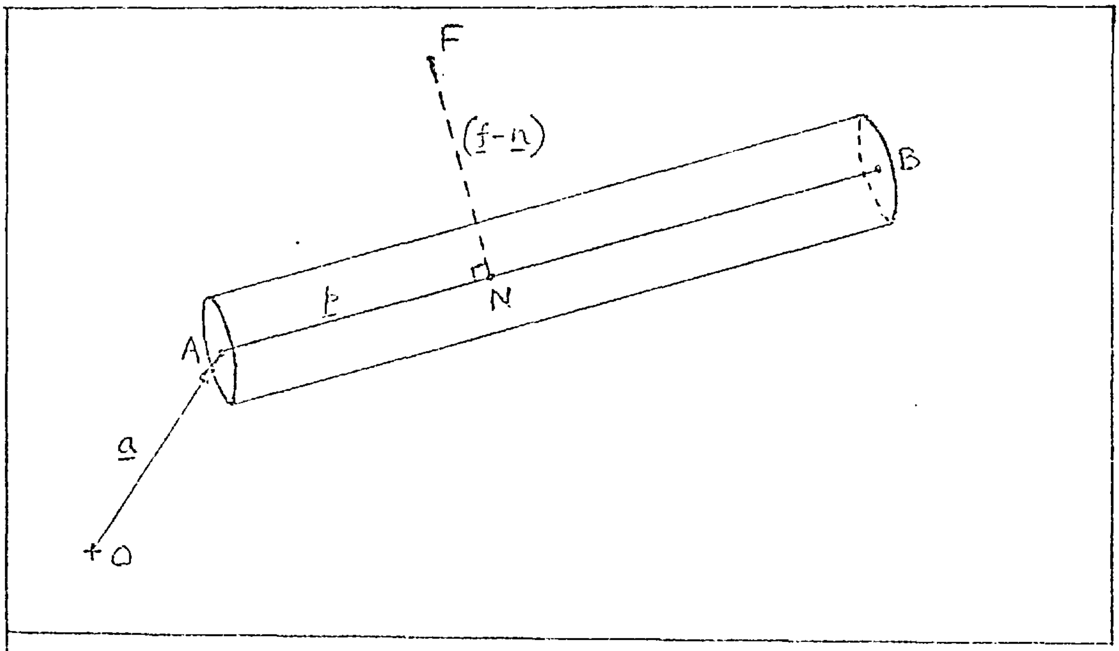
and

$$\mu = \frac{(\underline{a} - \underline{b}).\underline{p} \times \underline{q} \times \underline{p}}{\underline{q}.\underline{p} \times \underline{q} \times \underline{p}} \qquad Q = \underline{b} + \mu\underline{q}$$

The point of intersection is taken to be P or Q depending on the properties of the pipe lengths AC or BD whether the pipe length AC or BD is regarded as the more significant (or the one currently under study).

### 8.2.2 CALCULATION OF EXACT CENTRE LINE POINT

During the detailing stage of the program, the 3-D co-ordinates of the digitised position of the fitting F are calculated. However, this point may not lie mathematically on the line defined to be the centre line of the pipe AB. Hence the base of the normal N from the point to the centre line needs to be calculated.



Since FN is perpendicular,

$$\underline{n} = \underline{a} + \lambda\underline{p} \qquad (1)$$

where $\lambda$ is the constant of proportionality.

Also, $(\underline{f} - \underline{n}).\underline{p} = 0$ since they are perpendicular.

$$\therefore \quad \underline{f} \,.\, \underline{p} = \underline{a}.\underline{p} \times \lambda\underline{p}^2$$

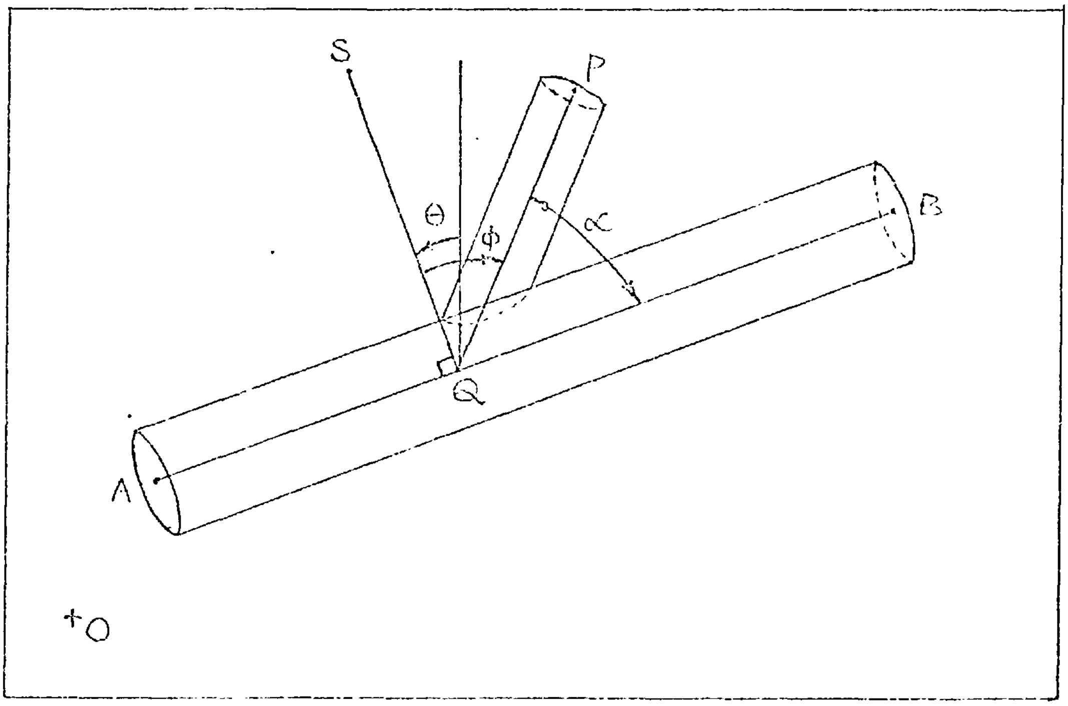
$$\therefore \quad \lambda = \frac{(\underline{f} - \underline{a}) \,.\, \underline{b}}{p^2}$$

This leads to the normal as, from (1)

$$\underline{n} = \underline{a} + \frac{(\underline{f} - \underline{a}) \cdot \underline{p}}{p^2}$$

### 8.2.3 CALCULATION OF ROTATION AND INCLINATION ANGLES

It is necessary when detailing branches to calculate the rotation ($\theta$) and inclination ($\emptyset$) angles. The direction cosines of the pipe length AB are known as (l,m,n). The end co-ordinates of the branch are known as P and the position of the branch on the pipe centreline is known as Q.



$\underline{AB}$ = (l,m,n)

Sine l,m,n are the direction cosines,

$$l^2 + m^2 + n^2 = 1$$

Let |AB| = a

$\underline{QP}$ = (h,i,j) as the direction cosines.

Let |QP| = k

Then $\cos\alpha = \frac{\underline{AB} \cdot \underline{QP}}{|\underline{AB}||\underline{QP}|} = \frac{lh + im + jn}{ak}$

and $\therefore$ $\varnothing = 90 - \alpha$. gives the inclination angle.

The angle $\theta$ is found by determining the direction cosines of the normals to the planes AQS and AQP. AQS is by definition the vertical plane through AB.

Hence the normal to AQS implies,

$\underline{AB} \times \underline{QS}$ (0,0,1)

and the direction cosines are (0,0,1).

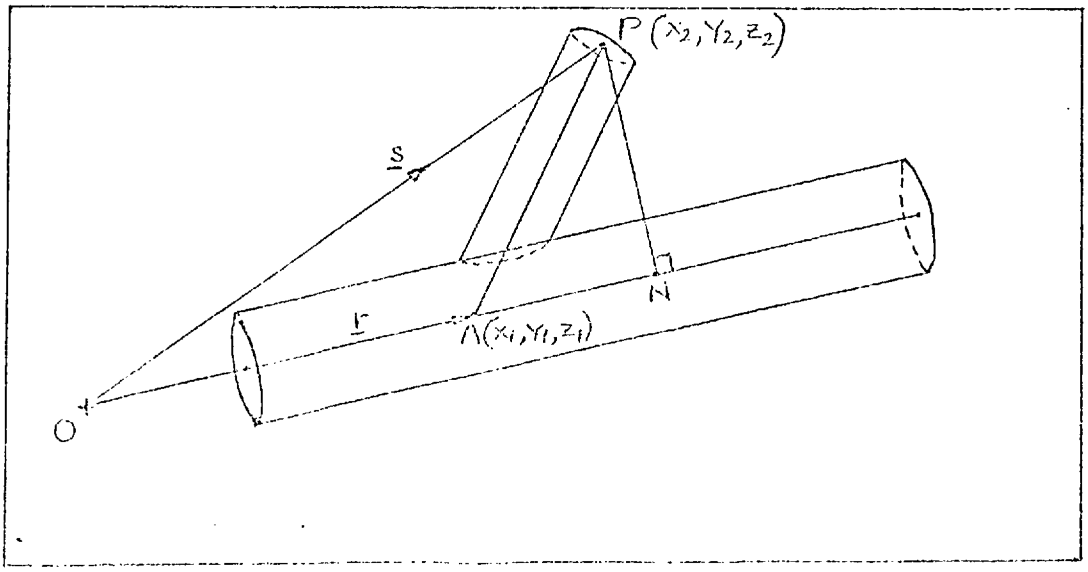
The normal to AQP gives

$\cos\theta = \underline{QP} \times \underline{AQ}$

This physically means that the zero rotation angle is along the z-axis.

8.2.4 CALCULATION OF THE ROTATION MATRIX

The problem is to define the rotation matrix necessary to re-orientate the pipe such that the first pipe length (OA) is parallel to the x axis and the next length (AP) is in the xy plane. Referring to the diagram below, this means that the direction cosines of OA, PN and a line perpendicular to the plane OPN, relative to the original axes need to be known.



The co-ordinates of $A(x_1, y_1, z_1)$ and $P(x_2, y_2, z_2)$ are given. If $\underline{i}$, $\underline{j}$, $\underline{k}$ are unit vectors along x, y, z respectively,

$$\underline{OA} = x_1\underline{i} + y_1\underline{j} + z_1\underline{k} = \underline{r}$$

$$\text{Also } |\underline{r}| = \sqrt{x_1^2 + y_1^2 + z_1^2} = r$$

$\therefore$ The direction cosines of OA are

$$\frac{x_1}{r}, \frac{y_1}{r}, \frac{z_1}{r}$$

Since $\underline{PN}$ is perpendicular to $\underline{ON}$,

$\underline{PN} \cdot \underline{ON} = 0$ by definition of scalar product.

Also $\underline{ON} = \lambda.\underline{OA}$ by constant of proportionality.

$$\therefore \underline{S} \cdot (\lambda\underline{r}) = 0$$

or

$$(x_1\lambda - x_2)x_1 + (y_1\lambda - y_2)y_1 + (z_1\lambda - z_2)z_1 = 0$$

$$\therefore \lambda = \frac{x_1x_2 + y_1y_2 + z_1z_2}{x_1^2 + y_1^2 + z_1^2}$$

Let $x_1x_2 + y_1y_2 + z_1z_2 = t^2$

Then $\lambda = \dfrac{t^2}{r^2}$

Thus the direction cosines of PN are

$$\frac{x_1 - x_2}{\sqrt{(x_1\lambda - x_2)^2 + (y_1\lambda - y_2)^2 + (z_1\lambda - z_2)^2}},$$

$$\frac{y_1 - y_2}{\sqrt{(x_1\lambda - x_2)^2 + (y_1\lambda - y_2)^2 + (z_1\lambda - z_2)^2}}$$

$$\frac{z_1 - z_2}{\sqrt{(x_1\lambda - x_2)^2 + (y_1\lambda - y_2)^2 + (z_1\lambda - z_2)^2}}$$

or

$$\frac{x_1t^2 - x_2r^2}{\sqrt{r^2(r^2s^2 - t^4)}}, \quad \frac{y_1t^2 - y_2r^2}{\sqrt{r^2(r^2s^2 - t^4)}}, \quad \frac{z_1t^2 - z_2r^2}{\sqrt{r^2(r^2s^2 - t^4)}}$$

The direction cosines of the normal to plane OPN, plane through $\underline{OA}$ and $\underline{OP}$, are given by

$\underline{OP} \times \underline{OA} = 0$ by definition of cross product.

$$(x_1\underline{i} + y,\underline{j} + z_1\underline{k}) \times (x_2\underline{i} + y_2\underline{j} + z_2\underline{k}) = 0$$

or

$$x_1y_2 \cdot \underline{i} \times \underline{j} + x_1z_2\ \underline{i} \times \underline{k} + y_1x_2\ \underline{j} \times \underline{i} + y_1z_2\ \underline{j} \times \underline{k}$$

$$+ z_1x_2\ \underline{k} \times \underline{i} + z_1y_2\ \underline{k} \times \underline{j} = 0$$

$$\therefore \quad (y_1z_2 - z_1y_2)\underline{i} + (z_1x_2 - z_2x_1)\underline{j} + (x_1y_2 - x_2y_1)\underline{k} = 0$$

If $f = y_1z_2 - z_1y_2$

$g = z_1x_2 - z_2x_1$

$h = x_1y_2 - x_2y_1$

Then the direction cosines of the normal are:

$$\frac{f}{\sqrt{f^2 + g^2 + h^2}}, \quad \frac{g}{\sqrt{f^2 + g^2 + h^2}}, \quad \frac{h}{\sqrt{f^2 + g^2 + h^2}}$$

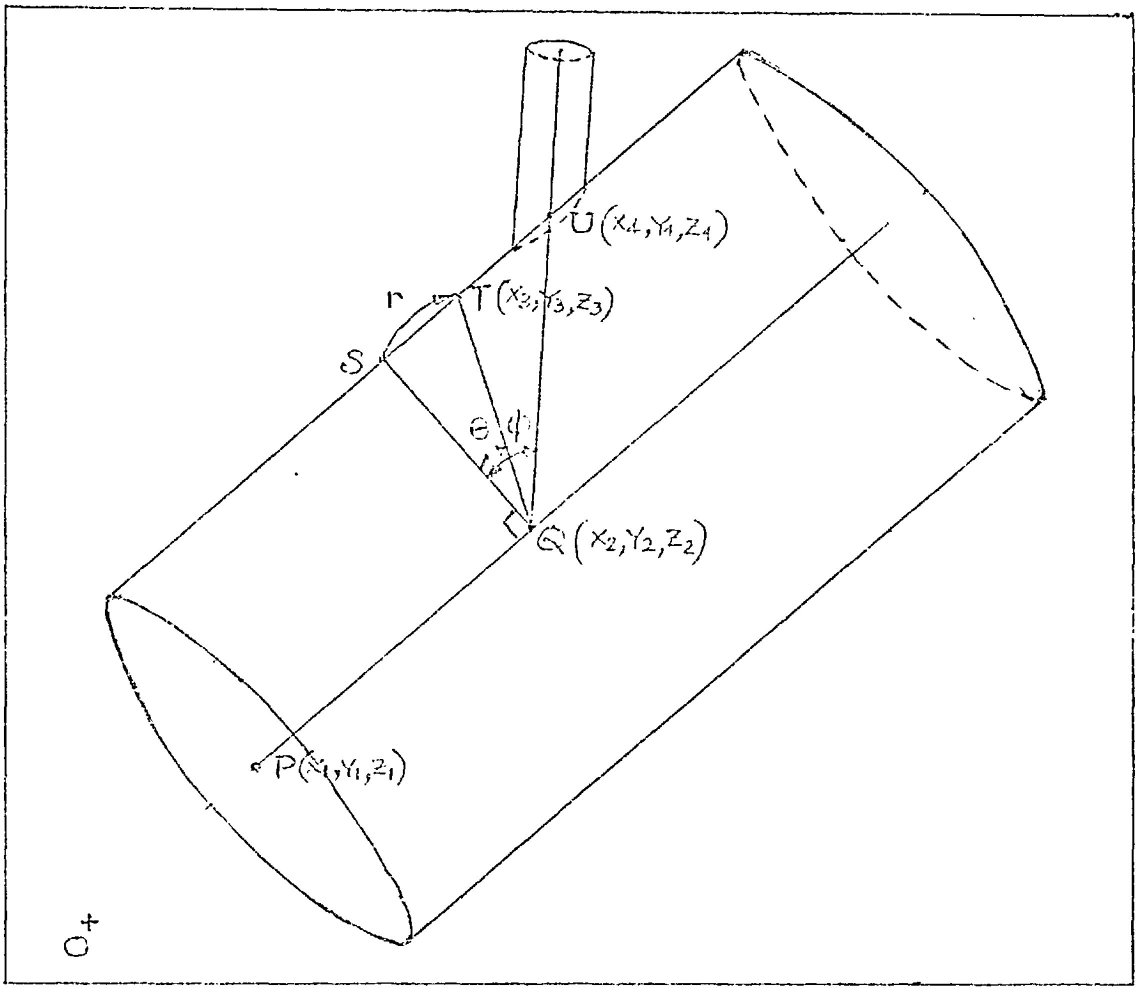
Hence the elements of the required rotation matrix are:

$$\begin{bmatrix} \frac{x_1}{r} & \frac{y_1}{r} & \frac{z_1}{r} \\ \frac{x_1t^2 - x_2r^2}{\sqrt{r^2(r^2s^2 - t^4)}} & \frac{y_1t^2 - y_2r^2}{\sqrt{r^2(r^2s^2 - t^4)}} & \frac{z_1t^2 - z_2r^2}{\sqrt{r^2(r^2s^2 - t^4)}} \\ \frac{f}{\sqrt{f^2 + g^2 + h^2}} & \frac{g}{\sqrt{f^2 + g^2 + h^2}} & \frac{h}{\sqrt{f^2 + g^2 + h^2}} \end{bmatrix}$$

where r,s,t,f,g and h are fully known.

### 8.2.5 CALCULATION OF THE OUTER WALL OF A PIPE

Once the rotation and inclination angles $(\theta,\emptyset)$ have been determined either by selection or calculation, it is necessary to calculate the co-ordinates of the mark (U) which must be made on the outside of the pipe. It is sufficient to find this centreline point of the feature to be placed on the outside of the pipe.



The outside radius of the pipe (r) is known as well as the direction cosines of the pipe length PQ, the co-ordinates of a point $P(x_1,y_1,z_1)$ on the pipe, and the co-ordinates of the feature on the centreline of the pipe $Q(x_2,y_2,x_2)$.

Let l, m, n be the direction cosines of PQ. for the plane PQS to pass through $(x_1,o,z_1)$ to make it vertical, the direction ratios of the normal to this plane are :

$n, o, -1.$ Let $b^2 = 1^2 + n^2$.
Then the direction cosines are $\frac{n}{b}, o, -\frac{1}{b}$.

Also $\underline{OT} - \underline{OQ} = \underline{TQ}$
$= (x_3 - x_2)\underline{i} + (y_3 - y_2)\underline{j} + (z_3 - z_2)\underline{k}$

$\therefore$ The direction cosines of $\underline{TQ}$ are

$$\frac{(x_3 - x_2)}{r}, \frac{(y_3 - y_2)}{r}, \frac{(z_3 - z_2)}{r}$$

where

$$r^2 = (x_3 - x_2)^2 + (y_3 - y_2)^2 + (z_3 - z_2)^2 \qquad (1)$$

Now the angle between the line $\underline{TQ}$ and the plane $\underline{PQS}$ is the complement of the angle between the line and the normal to the plane,

$$\therefore \quad \theta = \frac{\pi}{2} - \emptyset$$

By dot product,

$$\cos \emptyset = \frac{x_3 - x_2}{r} \cdot \frac{n}{b} - \frac{z_3 - z_2}{r} \cdot \frac{1}{b} \qquad (2)$$

To determine T $(x_3,y_3,z_3)$ uniquely, another condition is required. This is the condition that $\underline{TQ}$ is perpendicular to $\underline{PQ}$ or

$\underline{TQ} \cdot \underline{PQ} = 0$ taking the scalar product.

or

$$1(x_3 - x_2) + m(y_3 - y_2) + n(z_3 - z_2) = 0 \qquad (3)$$

From (1), (2) and (3),

T is fully known as,

$$x_3 = x_2 + \frac{r}{ab}(1m \sin\emptyset + an \cos\emptyset)$$
$$y_3 = y_2 - \frac{rb}{a}\sin\emptyset$$
$$z_3 = z_2 + \frac{r}{ab}(nm \sin\emptyset - a1 \cos\emptyset)$$

where $a^2 = 1^2 + m^2 + n^2$

Now the direction cosines of $\underline{TU}$ are

$$\frac{x_4 - x_3}{|\underline{TU}|} , \quad \frac{y_4 - y_3}{|\underline{TU}|} , \quad \frac{z_4 - z_3}{|\underline{TU}|}$$

but

$$|\underline{TU}| = \sqrt{(x_4 - x_3)^2 + (y_4 - y_3)^2 + (z_4 - z_3)^2}$$

$$= r \tan\emptyset$$

and $\underline{TU}$ is parallel to $\underline{PQ}$, therefore their direction cosines are equal.

Hence

$$\frac{x_4 - x_3}{r \tan\emptyset} = \frac{1}{a}$$

$$\frac{y_4 - y_3}{r \tan\emptyset} = \frac{m}{a}$$

$$\frac{z_4 - z_3}{r \tan\emptyset} = \frac{n}{a}$$

$\therefore$ Co-ordinates of U are known as

$$x_4 = \frac{r\ 1 \tan \emptyset}{a} + x_3$$

$$y_4 = \frac{r\ m \tan \emptyset}{a} + y_3$$

$$z_4 = \frac{r\ n \tan \emptyset}{a} + z_3$$

## 9. CONCLUSIONS:

Computer applications to general layout problems constituted the main flow of this thesis. An overwhelmingly large number of engineering layout problems, which actually exist in 3-dimensions, can be meaningfully translated into 2-dimensional or layer problems. The two major applications considered were:

1. Data preparation of Precedence type Management Networks.

2. Path finding in Printed Circuit Board Layouts.

Specialised and original algorithms were developed to overcome two main problems of optimisation. The first one was a computer orientated algorithm to minimise the number of branch cross-overs in a given network. List-processing techniques were employed to achieve planar solutions. This led to the basis of optimisation in Management Networks. Similar problems existed in the field of Printed Circuit Board Layouts.

The more useful solution to the optimisation problem in Printed Circuit Boards could only be achieved if the minimisation of cross-overs was combined with a shortest path finding algorithm. This provided for the definition of the Man in the Minefield problem, the solution of which led to the overall optimisation. Various Branch and Bound techniques, and Planarity testing algorithms were considered but it was concluded that the proposed algorithms were more suited to the practical problems in hand, and for implementation on a digital computer.

The system was developed as a real-time system, so the operator-machine interaction carried a high priority at all levels; from design through to implementation stages. Thorough Mathematical analysis of the pipe-routing and the linked design problem was carried out and it is recommended that this be installed as a satellite of a major 3-Dimensional Graphics System.

## FUTURE WORK

This section describes the recommended research which would act as a continuation in the following three areas:

### (1) Management Network Analysis

The algorithm described works well in an inter-active environment but is limited in its scope by the size of the networks that can be handled. This algorithm can be automated fully by simulating the existing techniques of data extraction and manipulation. First of all, an overall analysis can be done and a matrix containing the picture data can be set up. Then areas in the network which contain the largest number of crossovers can be concentrated upon.

This new technique should be implemented on the London University's CDC computers because of their larger storage capacity. The matrix properties can be analysed further and a new algorithm can be devised to work on this matrix to achieve fast and automatic optimisation in large networks. The means of transferring data from the PDP to the CDC is available using magnetic tape. The new algorithm for achieving minimum crossover situation in the matrix solution is expected to be an upper triangular matrix with unity along the diagonal of the matrix.

The data interchange between the connectivity diagram and the matrix representation can be easily achieved and it is expected that the matrix algorithm will provide a fast and efficient solution to the problem of minimising crossovers in large management networks.

(2) Printed Circuit Board Layouts

The routing algorithm described in this thesis had a disadvantage arising out of the lack of preparation of the input data before submitting it to the algorithm. It would be possible to carry out an initial analysis of the start and end points of the different paths to be found together with the priorities assigned to them. Core tables could be formed including such data.

The core table would be a 3-element list indicating all the paths to be found. Every element of this list could then be examined individually to check whether or not a planar solution existed for a given set. Different paths could then be mapped leading to an optimum path layout. Clash or crossover situations could be easily analysed using the core table and an overall function (i.e. cost) could be minimised.

Further work needs to be done on the final layout of the paths as found by the algorithm. A post-processor should be developed which would examine every path in turn and check for self-loops of the type occurring in the figure on page 77. The post-processing algorithm should be able to remove any anomalies caused by the dominance-driven nature of the existing algorithm. There is further scope here for designing an efficient new algorithm,which would calculate the planar and non-planar costs for any given path and compare them to obtain an optimal solution.

(3) General 3-Dimensional Routing

The future work required for the routing algorithm for 2-dimensions also applies to that in 3-dimensional routing. Many more design constraints can be combined together with routing in 3-dimensional analysis. During pipe routing, it should be possible to find multiple paths because of the 3-dimensional nature of the problem. Each solution can then be examined further by applying the design criteria. For example, if the pipe contains a fluid, then wall friction analysis should be done to calculate the overall pressure drop in a pipe. If the pipe is an electrical conduit, then the voltage drop along the conduit should be calculated. Therefore, a path found using the routing algorithm should only be accepted if it also meets the pre-specified design criteria.

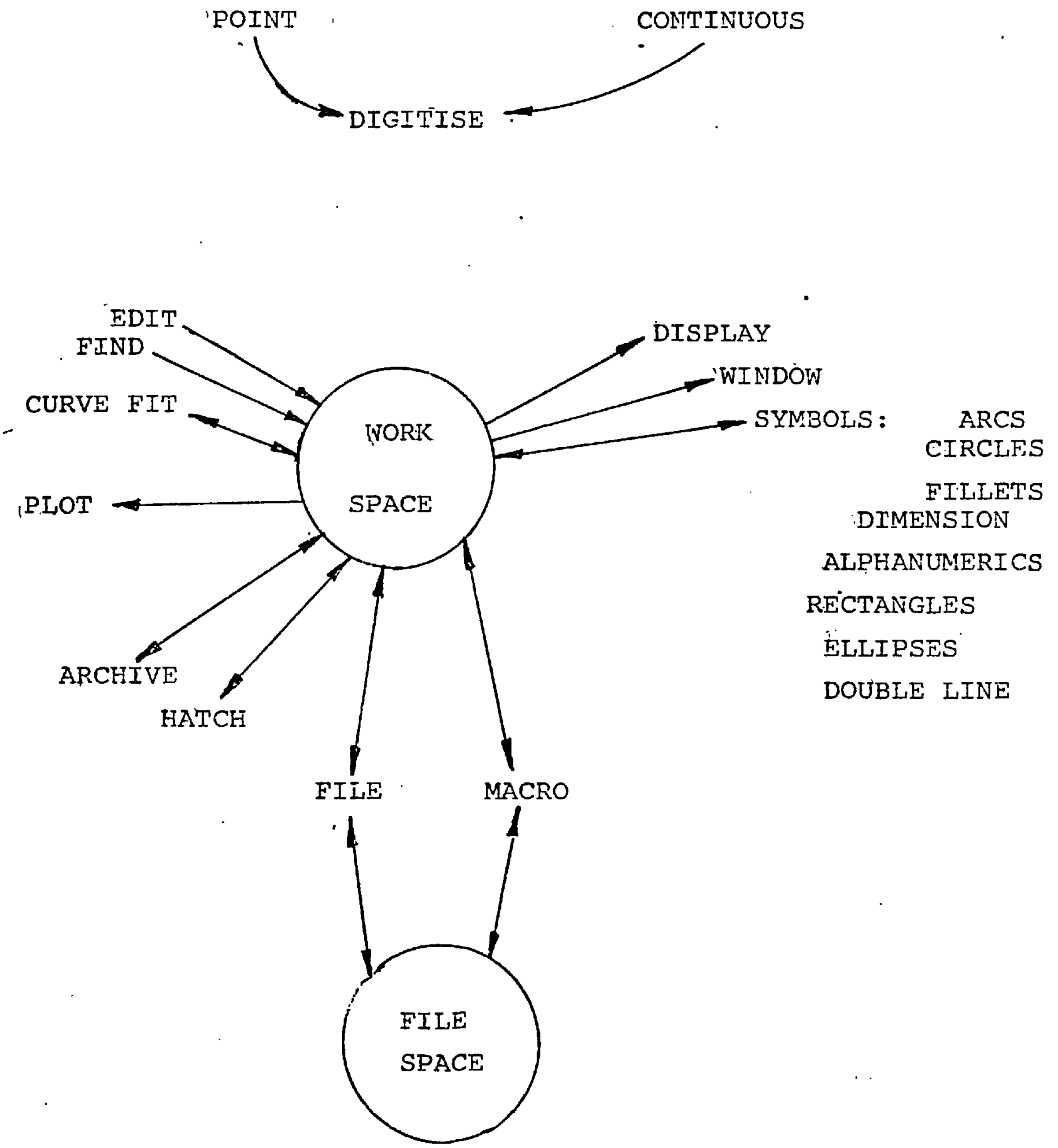
The way in which this can be achieved could be as follows. Submit the co-ordinates of the path to be found to the routing algorithm, which also takes care of the existing obstacles. Then, if the path found does not satisfy the design criteria, it should itself be flagged as a temporary obstacle and re-submitted to the path finding algorithm. This process should be repeated until a satisfactory path is found, which also meets the design criteria. Before the submission of the next set of co-ordinates to the path finding algorithm, the paths flagged as temporary obstacles should be removed. This method would combine an optimisation based upon design and routing.

There is further scope for developing an algorithm which would carry out grouping analysis on different paths in 3-dimensions. Again the technique of re-defining certain paths as temporary obstacles could be used to group together certain types of pipes e.g. electrical conduits could be grouped separately from fuel pipes.

The above mentioned future work is an expansion of the algorithms described in this thesis. The reader should be able to think of other applications where these algorithms could be usefully employed in minimising other functions.

APPENDIX 'A'

EXAMPLE OF CENTRAL GRAPHICS SYSTEM:



CONVENTION:

→ AFFECTED BY

← AFFECTS

## APPENDIX B

This appendix contains definitive specification of the two algorithms:

(a) Activity Network Algorithm.

(b) Routing Algorithm.

### (a) ACTIVITY NETWORK ALOGORITHM

1. The network is reduced to a simple connectivity diagram. The start and end points of an event are redefined from $(x_1 y_1)$ and $(x_2, y_2)$ to 1 and 2 respectively.

2. Lists containing start and end points are extracted in an ascending order.

3. The lists are used to examine the number of existing cross overs.

4. The number of cross overs is examined as every end point exhanges position with all the others sequentially.

5. Whenever a lower number of cross overs is achieved, the end point is fixed.

6. In the list representation, this leads to a minimum cross over situation.

7. The final lists are transferred back to the activity network by moving the activities, using the connectivity diagram.

### (b) ROUTING ALGORITHM

1. To find a path from a start point to an end point, it is assumed that both the points lie on a grid.

2. The dominant direction along the path is calculated.

3. A step equal to the grid size is taken from the start point along the dominant direction.

4. A check is made to see if the new point is the end point.

5. If the end point is reached, then the procedure stops. Otherwise, it is checked that the new point does not lie in a forbidden area.

6. If the point is not forbidden, then the path finding continues till the end.

7. If the point does lie in a forbidden area, then the step is traced back to the previous point. A step is then taken along the non-dominant direction.

8. The already specified checks are made. If this step is unsuccessful, then the path is made to go through to the other plane and path finding continued on the other side, when cost factor is unity.*

9. If, however, the step was successful, then path finding is continued normally till the end point is reached.

*For a cost factor of n, the algorithm attempts n steps before going to the other plane.

# REFERENCES

1. British Standard 4335: 1972
"Project Network Techniques"
.British Standards Institution

2. YI, C.
"The Use of Computer Aided Design Techniques in Dynamic Graphical Simulation".
Ph.D. Thesis, University of London (1977)

3. HAMLYN, A.
"The Application of C.A.D. Techniques to Building Engineering Design".
Ph.D. Thesis, University of London (1974)

4. EDNEY, R.C.
"An Application of Computer Aided Design Techniques to Mechanical Engineering".
Ph.D. Thesis, University of London (1977)

5. CHRISTOFIDES, N.
"An Algorithm for the Chromatic Number of a Graph".
Imperial College Management Science Section
Report No. 69/15 (1969)

6. MODER, J.J. and PHILLIPS, C.R.
"Project Management with CPM and PERT".
Van Nostrand Reinhold Company, New York (1970)

7. BATTERSBY, A.
"Network Analysis for Planning and Scheduling".
MacMillan & Co., (1970)

8. DEO, N.
"Graph Theory with Applications to Engineering and Computer Science".
PRENTICE-HALL, INC. Englewood Cliffs, N.J.
PP 90-93 (1974)

9. TARJAN, R.
"An Efficient Planarity Algorithm".
Computer Science Department, Report No. CS-244-71.
Stanford University (1971)

10. HOPCROFT, J.E. and TARJAN, R.
"Planarity Testing in Vlog V Steps".
Journal of the Association for Computing Machinery
Vol.21, No.4. PP 549-568. (1974)

11. DANTZIG, G.B.
"All Shortest Routes in a Graph".
Proceedings of the International Symposium on Graph Theory, Rome, Italy. Published by Dunod Editeur, Paris (1966)

12. FORD, R and FULKERSON, D.R.
"Flow in Networks".
Princeton University Press. (1962)

13. MOORE, E.F.
"The Shortest Path through a Maze".
Proc. Int. Symp. on Theory of Switching. (1957)

14. DIJKSTRA, E.W.
"A Note on Two Problems in Connection with Graphs".
Numerische Math, Vol.1. PP 269-271. (1959)

15. NICHOLSON, T.A.J.
"Finding the Shortest Route Between Two Points in a Network".
The Computer Journal, Vol.9. PP 275-288. (1966)

16. DREYFUS, S.E.
"An Appraisal of Some Shortest Path Algorithms".
J. Operations Research, Vol.17 No.3 PP 395-412 (1969)

17. LITTLE, J.D.C. et.al.
"An Algorithm for the Travelling Salesman Problem".
Operations Research. Vol.11 No.6 PP 972-989 (1963)

18. BERGE, C.
"The Theory of Graphs and its Applications".
London: Methuen (1962)

19. EASTMAN, W.L.
"A Solution to the Travelling Salesman Problem".
American Summer Meeting Econometrics Soc. (1958)

20. LAND, A.H. and DOIG, A.
"Automatic Method for Solving Discrete Programming Problems".
Econometrica. Vol 28 PP 497 (1960)

21. BENAYOUN, J.H. et.al.
"An Integer Program for Solving Branch and Bound Problems".
J. Operations Research. Vol.15 No.5 pp 402-438 (1965).

22. DAVIDSON, E.S.
"An Algorithm for NAND Decomposition Under Network Constraints".
IEEE Trans. Comput. Vol.12 PP 1098 (1969)

23. WHITNEY, H.
"Planar Graphs".
Fund. Math. Vol.21 PP 73-84 (1933)

24. MaCLANE, S.
"A Combinational Condition for Planar Graphs".
Fund. Math. Vol.28 PP 22-32 (1937)

25. BRUNO, J., STEIGLITZ, K. and WEINBERG, L.
"A New Planarity Test Based on 3 - Connectivity".
IEEE Trans. on Circuit Theory. Vol.CT-17 May (1970)

# BIBLIOGRAPHY

The books and papers in the following bibliography are wholly or partly related to the subjects covered in this thesis.

1. BESANT, C.B. et al
"CADMAC - A Fully Interactive Computer Aided Design System".
Computer Aided Design, Vol.4, No.2 (1972)

2. HENLEY, E.J. and WILLIAMS, R.A.
"Graph Theory in Modern Engineering".
Academic Press (1973)

3. STEIGLITZ, K.
"An Introduction to Discrete Systems"
John Wiley & Sons, Inc. (1974)

4. KNUTH, D.E.
"The Art of Computer Programming; Vol.1 - Fundamental Algorithms".
Addison-Wesley (1968)

5. DEO, N.
"Graph Theory with Applications to Engineering and Computer Science".
Prentice-Hall (1974)

6. MASSEY, B.S.
"Mechanics of Fluids".
Van Nostrand (1970)

7. BESANT, C.B. et al
"The Use of Computer Aided Design Techniques in
Printed Circuit Layouts".
CAD Vol.5 (1973)

8. The Concise Oxford Dictionary.