ALGORITHMS FOR THE DESIGN OF CHEMICAL PLANT LAYOUT AND PIPE ROUTING

By

R.G. Newell, BSc(Eng),ACGI,DIC

A thesis submitted for the degree of Doctor of Philosophy.

Department of Chemical Engineering and Chemical Technology,
Imperial College, London, November 1974.

CONTENTS.

LIST OF TABLES AND ILLUSTRATIONS.

ACKNOWLEDGEMENTS.

I am indebted to Professor R.W.H.Sargent for originally suggesting the subject of this research and for his extreme patience and encouragement during the course of preparing this thesis. My thanks are due to the SRC and to ICL who supported me, and also to the CAD Centre for allowing the research to be continued while in their employ.

SUMMARY.

The use of computers in chemical plant design has largely centred around process flow sheet design, vessel design and the output of production documentation from the layout and routing phases. The work reported in this thesis identifies a number of problems concerned with the layout and routing phases and as such attempts to bridge the gap in the design process.

Three separate problem areas are tackled; firstly the vessel layout problem, secondly, the pipe routing problem; and thirdly, the visualisation of a digital model of a chemical plant.

The vessel layout problem is tackled by modelling each vessel as a collection of simple modules which have relative positional constraints imposed in order to represent differing vessel sizes and shapes. The same system of constraints is used in order that relative or absolute constraints can be imposed on the layout as a whole. Intelligent search procedures are used to optimise the layout, and because of the combinatorial explosion inherent in such problems, facilities are provided for the user to specify additional constraints which reduce the solution time.

The pipe routing problem is tackled by using a shortest route algorithm that generates minimal routes on an automatically generated network. This network is so constructed that all obstacles in space are avoided by the

routed pipes. Algorithms for routing branched pipes with three or more terminal nozzles are presented. The routing algorithms are designed such that constraints on a routing scheme can be incorporated by the user defining favourable or unfavourable volumes for routing pipes. An algorithm for jointly minimising the cost of pipe and bends is also presented.

The chapter concerned with visualisation describes a novel hidden-line algorithm which is capable of handling a reasonably complex scene. A chemical plant is defined as a collection of polygons which approximate the surfaces of the vessels and other items in the scene. This algorithm has wider application than the chemical engineering industry and as such is significantly superior to other published hidden-line algorithms. The computation time increases a little more than linearly with scene complexity.

CHAPTER 1

THE USE OF COMPUTERS IN CHEMICAL ENGINEERING DESIGN.

1.1 Introduction.

It has long been realised that the design process of a chemical plant contains many tasks which are amenable to the application of computer techniques. In particular some aspects of the design process amount to well defined numerical problems which can be translated in a straight forward manner into a program suitable for a computer. Other areas of the design process are not so easy to automate, either because the numerical problems are difficult to solve as a problem in numerical analysis, or because the task is not well specified and relies on such factors as the engineers inherent knowledge of the problem. Into this latter category falls the area of plant layout and pipe routing, which is the subject of this thesis. Although we are concerned with plant layout problems in general, it is relevant to examine the whole of the design process with respect to process plants.

It is possible to segment the design process of a chemical plant into five main phases. The reason for this segmentation is in order to break down a large problem into a series of smaller problems which can be tackled in sequence. This therefore assumes that every phase can be tackled independently provided the required results of earlier phases have been determined. In practice, any design solution is iterative, there being a requirement for feedback from later phases to an earlier phase, but the amount of iteration permitted to the designer has to be limited by deadlines imposed on the design task which are dictated by the overall requirements of getting a new plant on stream. Therefore any contribution that can shorten the

time taken for each design step will allow more design alternatives to be evaluated and hence a better solution should result.

The five phases of chemical plant layout design can be categorised as follows:

1) Process flow sheet and vessel design.
2) Layout of main plant items and support structure.
3) Pipe routing.
4) Detailed positioning of pipe fittings and instrumentation.
5) Documentation.

The two areas that have been tackled using computer techniques are phases 1 and 5. A number of systems now exist to evaluate process flow sheets and many companies have their own vessel design programs. The area that has received most publicity is the automatic production of isometric pipe drawings, bills of quantities and fabrication schedules. These are areas many companies have found possible to isolate and replace by computer methods, but the input to such systems has to be coded manually and hence much of the potential profitability is lost. Here then lies one of the advantages of providing a computer based vessel layout and pipe routing system, all the input which is now coded manually for the documentation systems can be replaced by automatic procedures.

1.2 The Scope of this Thesis.

The layout and routing problems are special in one sense, in that design constraints are largely geometric and the problems to be solved are combinatorial in nature, as opposed to functional. In no sense is it possible to write down a solution of the problem as a function of a few input parameters, as one can with some vessel design procedures. Thus the problems to be solved are severe, however three main problem areas are tackled here. All three are considered as self contained problems and in all cases wholly automatic solutions have been sought.

Chapters 2,3 and 4 attempt to provide solutions to the vessel layout problem, the pipe routing problem and the visualisation problem of a design held by the computer. A fifth chapter is included to assess the importance and practicability of the results of chapters 2 to 4 and to suggest ways in which the algorithms devised can be incorporated in an integrated layout,routing and detailing system.

CHAPTER 2

ALGORITHMS FOR PLANT LAYOUT.

2.1 Introduction.

The vessel layout problem is a particular kind of assignment problem. Assignment problems arise in many fields, for example in the electrical industry where it is required to arrange electrical components on a printed circuit board or in the ship-building industry where it is required to layout shapes on sheets of metal in order to produce hull components. Layout problems in general have a number of features in common. They deal with the geometric arrangement of conponents in two or three dimensions. The resulting arrangement must satisfy certain constraints, and some objective function must be minimized. Layout problems differ from one another because the constraints are different or the objective function is different. For example, Vergin and Rogers (1) describe a procedure that takes no account of clashing items. Nearly all solutions to layout problems are based on some combination of exhaustive search techniques and heuristic programming. For example Nemelak (2) gives a very involved recipe as opposed to Hillier and Connors (3) who use a search approach in a manner similar to the methods described in this thesis. Vollmann and Buffa (4) identified many of the fundamental problems and common misconceptions concerned with solving layout problems automatically. We will outline here what particular features are characteristic of vessel layout in the chemical industry.

A chemical plant is a three dimensional arrangement of components consisting of the main plant items. A main plant item is usually a vessel, but could just as easily be any large item which is not permitted to occupy the same

space as another component. The components in the layout are connected by pipes which abut onto defined nozzle positions and it is required to minimize the cost of piping required to connect up the plant. Other costs might also be included in the objective function for example the cost of support structures, erection costs and running costs.

The essential thing about any cost is that it should be computable and if automatic layout procedures are to be used, then it is necessary that the computation can be carried out quickly, since a large number of alternative solutions will need to be evaluated in the optimisation procedure.

The constraints imposed by the vessel layout problem affect the permitted position of the vessels. For example a vessel might require to be fixed at a certain location or restricted to a certain defined area of the plant. A relative constraint might be imposed affecting the relative positions of two items. For example two vessels might not be permitted to be placed within a certain distance of one another.

In this chapter, approaches are proposed for solving these problems which are efficient provided that the size of problem tackled is not too large. Simplifications are made in defining the problem before it can be tackled and in doing this there is the inherent risk of solving a different problem from the one intended.

The following basic information is assumed at the start of the vessel layout phase:

(a) the process flow sheet which defines schematically the

manner in which the vessels are to be connected

(b) vessel dimensions

(c) site dimensions

(d) terminal positions of pipes on vessels

(e) estimates of the cost per unit length of pipes

(f) relative and absolute constraints imposed on the vessel positions

2.2 Backgound and Basic Assumptions.

The vessels of a chemical plant comprise an assortment of object types that vary in size and shape. Although the problem might be considered as a three dimensional one, in practice many problems are essentially two dimensional since the level of each vessel is fixed in order to satisfy pressure drop conditions. When units are permitted to vary in level then they can either be considered as being independant of the two dimensional layout problem or the problem can be discretized in layers corresponding to the various levels of support structure in many plants. In this case we have a two and a half dimensional problem. There is in general an infinite number of possible ways to place a set of vessels on a given site, but it is convenient to adopt the use of a grid such that each vessel can only be placed at a discrete number of positions on the site. Thus, if for simplicity's sake, we can assume that all vessels are the same size then each can be represented by a square unit which is permitted to occupy any of the discrete positions on the grid.

If each grid position is called a node and if there

are m nodes and n units ($m \geq n$) then there are $m!/(m-n)!$ ways of arranging the units on the nodes.

For each configuration of units there is an associated cost function which is taken to be the weighted sum of the piping lengths between all the units. In the 2D case, the pipe length between two units occupying node positions i, and j (coordinates X_i, Y_i, X_j, Y_j) is taken to be

$$|X_i - X_j| + |Y_i - Y_j| + D$$

where 'D' is some correction factor that could be included to account for the amount of extra piping that might be needed to avoid other units or to account for the fact that pipes do not emerge from a grid point but from the sides of the units.

At this stage no account is taken of the number of bends, since it is impossible to estimate unless a detailed design of each pipe is separately undertaken.

Since in general we are considering placing 'n' units on 'm' nodes $m \geq n$, then we have $(m - n)$ dummy units occupying the unused nodes and these have no pipework associated with them. The problem is to determine that configuration of units that has the minimum value of the cost function associated with it.

In order to formalise the above outline then:

let the set of units be; $U = [u_1, u_2, \ldots, u_n]$

let the set of nodes be; $N = [n_1, n_2, \ldots, n_m]$

consider any placement p of the set of units U on the set of nodes N.

$P = [p_1, p_2, \ldots, p_m]$ $p_i \in N$ and is the position of unit i.

P is a permutation of N.

Once the number of units is more than about eight the idea of finding the optimum permutation by means of an exhaustive search becomes impractical. A number of approaches have been suggested which are based on taking a subset of the placement and improving the global costs by rearranging the units in the subset. If one chooses a number of subsets and proceeds by improving the layout in an iterative manner, eventually a stage will be reached where no further improvement can be made within the framework of the method used and a local optimum will be reached.

Now let us consider any subset of the units:

$U_l = [u_{l1}, u_{l2}, \ldots, u_{lr}]$ $r<n$

and the placement of U_l is

$P_l = [p_{l1}, p_{l2}, \ldots, p_{lr}]$

then the idea is to rearrange the units of U_l on the node position of P_l such that the resulting global costs are minimized.

Let us examine how the cost of the piping is made up. First, the cost of any one pipe running between unit i and unit j is $C_{ij} \cdot L(p_i, p_j)$ where C_{ij} is the cost per length of the pipe and $L(p_i, p_j)$ is the length of pipe needed to go

man i for job j is R_{ij}. With one man per job, the idea is to maximize $\sum R_{ij}$.

The problem can be cast in this form since the cost of piping for each unit $u \in U_i$ can be computed for each node belonging to N_i irrespective of the positions of the other members of U_i. There is an efficient algorithm for solving the assignment problem for numbers of units up to at least twelve (see 7).

The important thing to remember with these approaches is that the optimum solution is not necessarily a global solution, and that the final solution will depend on the initial configuration and the order in which the subsets are taken for rearranging.

However, it is intuitively obvious that a better solution is more likely to result if larger subsets are chosen rather than smaller ones, thus such methods depend on the existence of efficient algorithms for finding the minimum placement of a subset.

The above discussion has not taken account of positional constraints. The method of unconnected subsets no longer applies in the presence of constraints. Pair swopping is very inefficient for constrained problems because it is necessary to permute larger subsets of units than two in order to maintain satisfaction of the constraints. The ability to include constraints is important, since among other things it allows the facility to handle vessels of different shapes and sizes without further complication.

between p_i and p_j. If the expression $C_{ij}.L(p_i,p_j)$ is denoted K_{ij}, then the total cost is:

$$\sum_{\substack{ij\in U \\ i<j}} K_{ij} = \sum_{\substack{ij\in U_1 \\ i<j}} K_{ij} + \sum_{\substack{ij\in U_2 \\ i<j}} K_{ij} + \sum_{\substack{j\in U_1 \\ i\in U_2}} K_{ij}$$

$$= \quad C_1 \quad + \quad C_2 \quad + \quad C_3$$

Hence if we are only rearranging the elements of U_1, we need not consider C_2 in the process of finding a minimum since it remains constant. Thus in the general case it is only necessary to find the minimum of (C_1+C_3) in order to find a new placement of U_1.

The simplest application of this approach is to take a pair of nodes, and find the minimum of the two alternative placements (see 5). If this process is continued for all possible pairs of nodes until no further improvement can be made no matter what pair of units is switched then some local optimum solution will have been reached. This solution might depend on the initial starting configuration and the order in which the pairs are switched. An extension of this idea is to iterate through all possible groups of three or more in a like manner. The larger the subsets taken, the longer it will take, but of course a better solution is to be expected. Another approach is to choose the subsets of units U_1 such that its elements are unconnected to each other (see 6). Now the problem of finding the best placement of U_1 becomes the classical assignment problem. This problem can be cast as follows. There are 'n' men for 'n' jobs and the rating of

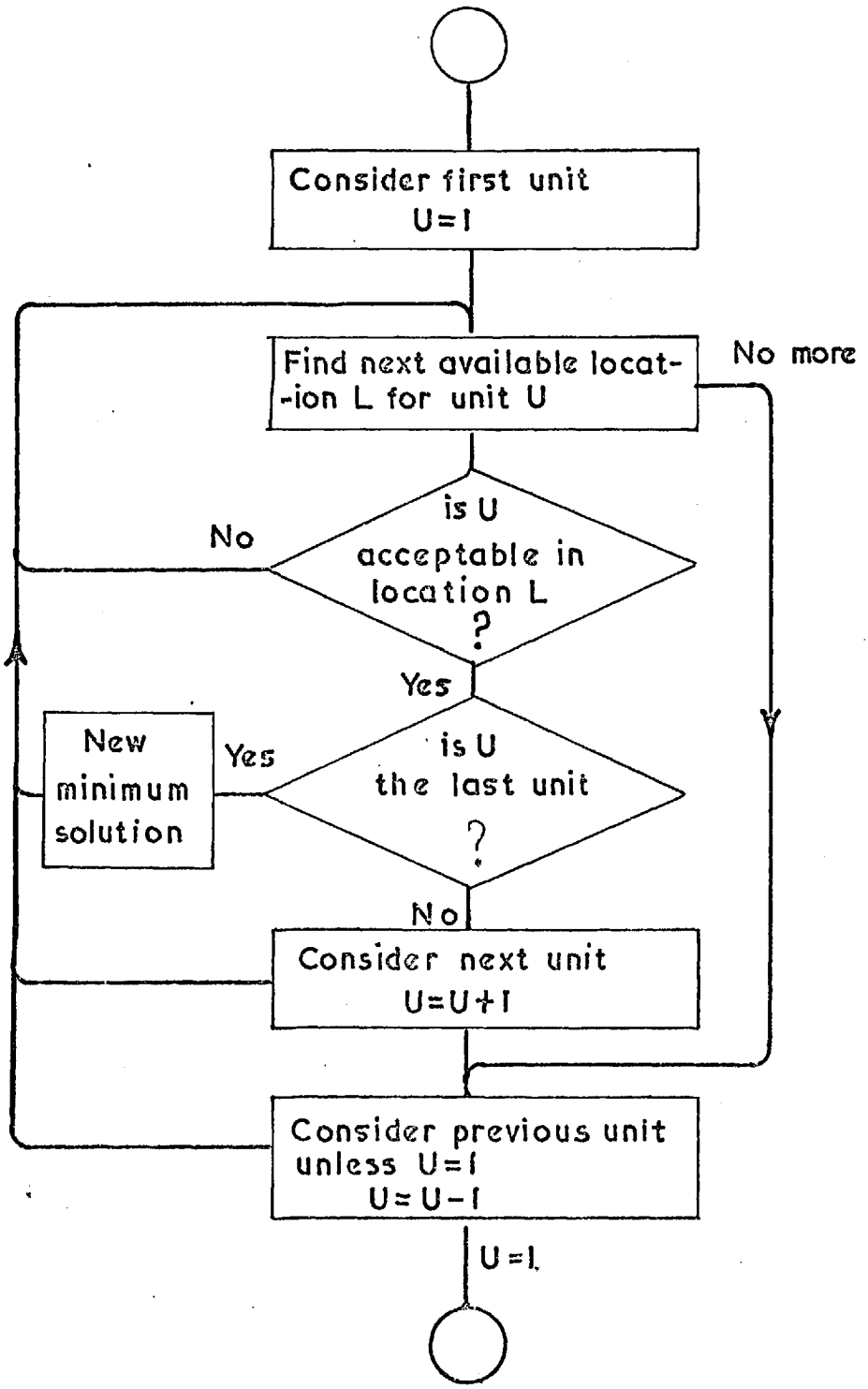


FIG. 2-1 : FLOW CHART OF SEARCH PROCEDURE

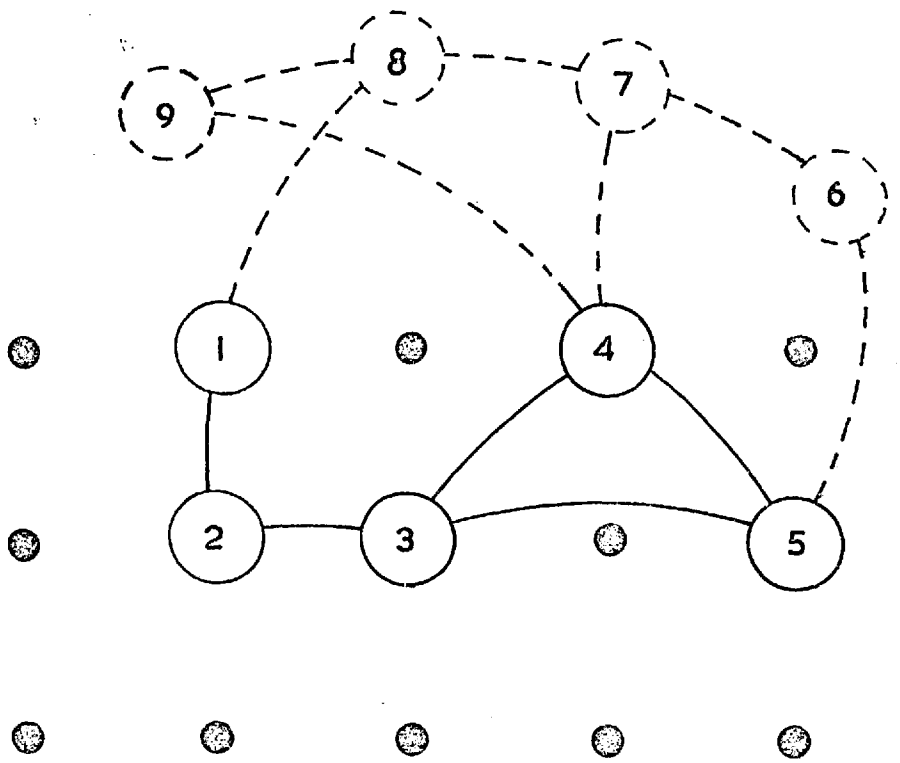


FIG.2-2: INTERMEDIATE STAGE OF BASIC SEARCH ALGORITHM

2,3 A Tree-structured Search Algorithm.

Although it is just conceivable that for some small problems, an exhaustive search of all permutations could be undertaken, it is obvious that a more sophisticated, faster approach is needed for larger problems. Therefore a way is needed of accelerating the search to such an extent that the procedure will terminate in a reasonable time. As a basis, a straight forward method of building up a layout scheme is considered and then various methods are proposed for accelerating the procedure. A flowchart of the basic search procedure appears in figure 2-1.

Units are numbered sequentially before the search begins, this being the order in which they will be brought into the solution. This order is important, because it can have a marked effect on the speed of solution. Imagine the procedure at an intermediate stage where we have placed four units on four nodes and anticipate placing the fifth unit as shown in figure 2-2.

If L_i is a prospective location for placing the fifth unit, then we can calculate the cost of all pipes, shown full, running between the first five units. In addition we can also obtain some lower bound for the costs of all pipes, shown dotted, connecting units 1 to 5 and units with sequence numbers greater than 5 (i.e. those units which are yet to be placed). A simple lower bound could be the shortest length which each pipe could possibly take, this being the distance between adjacent grid locations.

We now have a lower bound estimate of the total cost of the optimal solution with the first five units placed in

these locations. Should this estimate be greater than some solution previously found to the whole problem, then it is clearly not necessary to leave unit 5 in this position, and so it can be moved to the next available position and the procedure repeated. If however, the lower bound estimate is less than the previous estimate, then we can carry on and try to place the 6th unit. When all available locations for unit 5 have been investigated, then we step back, to unit 4 and advance its position in a similar manner.

This procedure amounts to a tree-structured search of all possible solutions with tests conducted at each node of the tree in order to decide whether to pursue a particular branch further.

Positional constraints can be easily incorporated in the above scheme by testing whether the position of a unit is acceptable at the same time as testing the cost.

If the constraint is an absolute one, then this is very straight forward indeed, however, if the constraint is relative then it depends on whether the other unit involved in the constraint has been placed or not. Constraints could involve more than two units, and in general such constraints can only be tested for the last member of the group placed.

If we examine the efficiency of the procedure, with reference to figure 2-2, then it can be seen that unit 5 has 11 availble locations. It is desirable that most of these 11 options are rejected without proceding to unit 6. The lower bound test is one way of achieving this reduction, but also constraints can become very helpful

here. In fact the more constraints imposed on the layout, then in general a quicker solution time is to be expected. In the extreme, when all units are constrained to lie in particular grid locations, the above procedure takes virtually no time at all, but then it would be a poor show if it did not.

In order that the solution be obtained quickly, then it is advantageous if those units which are tightly constrained are placed before those which are loosely constrained. Thus those units whose position is fixed should be placed first. The criteria for which units should follow is not so clear, but good rules for choosing an ordering depend on the cost of the connections with units already placed. In particular if a rigid sub-assembly of units is indicated by constraining the distances between units in each sub-assembly to have a fixed value, then a good rule is to place large sub-assemblies prior to small ones.

A further point which needs clarifying and which also has an important bearing on the speed of solution is the target cost. One approach is to commence with a very large target cost. If this is done, then the first solution found by the search will usually not be optimal, but it will be a solution satisfying the constraints. This first solution gives a new, more meaningful, target cost. The target cost gets smaller as more new solutions are found and so the lower bound test becomes more effective and the progression of the search accelerates.

A way of deliberately accelerating the solution is to

sub-optimise. In other words, whenever a new solution is found, a target cost is set which is less than the new minimum value. This approach is acceptable if one can say that a solution within say 5% of the minimum is satisfactory. These possibilities are illustrated in the test examples in section 2.5

An alternative approach to the above is to start with a very low target cost. This might be the sum of all the lower bound costs for each pipe. However, there is a risk that this target cost is less than the minimal solution, in which case no solution will be found, but the search may be accomplished quickly. If this happens, then the target cost can be raised by a small amount and the search recommenced. By adopting low target values, the effectiveness of the lower bound test is maximised, however, for each increment in the target cost, the search time gets longer.

So far we have discussed the effects on the speed of solution of the placement order of the units, the positional constraints, and the size of the target cost. A further factor which affects speed is the quality of the lower bounds used. The lower bounds suggested have the advantage of being simple to compute, but in many cases are too low and hence lead to long search times. The following section shows how higher lower bounds can be obtained.

2.4 Computing Improved Lower Bounds.

The method adopted for computing improved lower bounds is recursive in nature. Having placed a subset of the units we require a lower bound estimate of the cost of pipes connected to those units not yet placed. Thus one

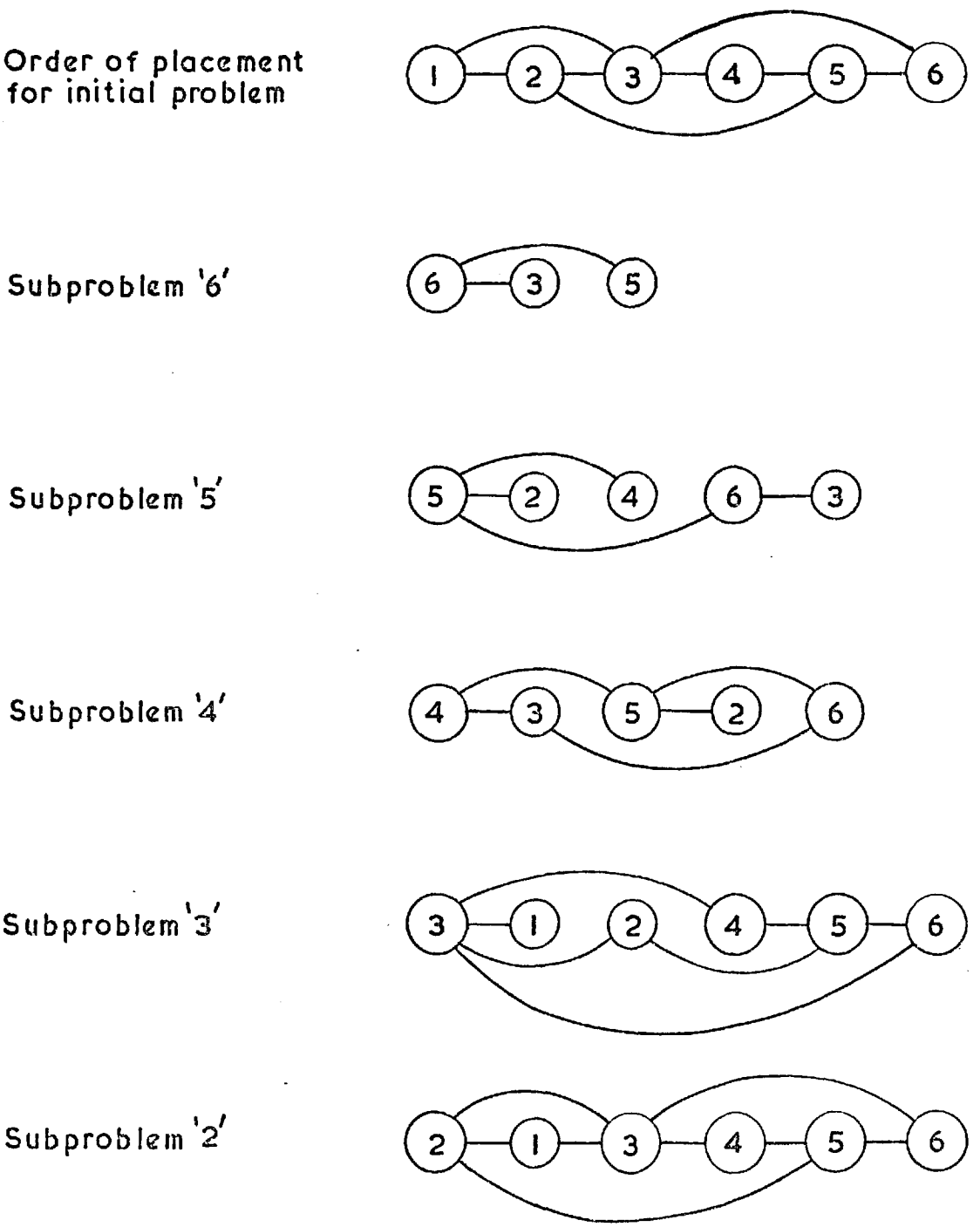


FIG.2-3: UNIT ORDERINGS FOR IMPROVED LOWER BOUND CALCULATIONS.

could consider the problem of computing this lower bound as a sub-problem which in order to be solved generates its own sub-problem and so on. Therefore in order to solve the initial problem it is required that all sub-problems of that problem have already been solved. It is then to be hoped that the lower bounds resulting from this lead to a smaller number of allowable placements at each stage of the algorithm.

The example in figure 2-3 illustrates the sequence of sub-problems solved. The solution to sub-problem '2' is in fact the solution to the original problem since all units and pipes are present. The order in which the units are placed in each sub-problem is important if the lower bounds computed from previously solved sub-problems are to be used. For example, the ordering for sub-problem '4' was generated as follows. Sub-problem '4' is required in order to generate the lower bound to be used when trying to place units 2 and 3 in sub-problems '3' and '2' respectively, so unit 4 is the first unit. Following unit 4 is unit 3 which although it has a smaller sequence number is connected to unit 4 and is therefore included with this sub-problem. The sequence is terminated by vessels 5,2 and 6. Note that the pipe connecting units 2 and 3 is not included. If the ordering of sub-problem '4' is now examined it can be seen that the lower bounds found in sub-problems '5' and '6' are used when placing units 3 and 2 respectively. We still need a lower bound for placing unit 5 in this sub-problem. This is calculated as the minimum for sub-problem '6' plus the lower bound estimate for the pipe connecting unit 5 to unit 2.

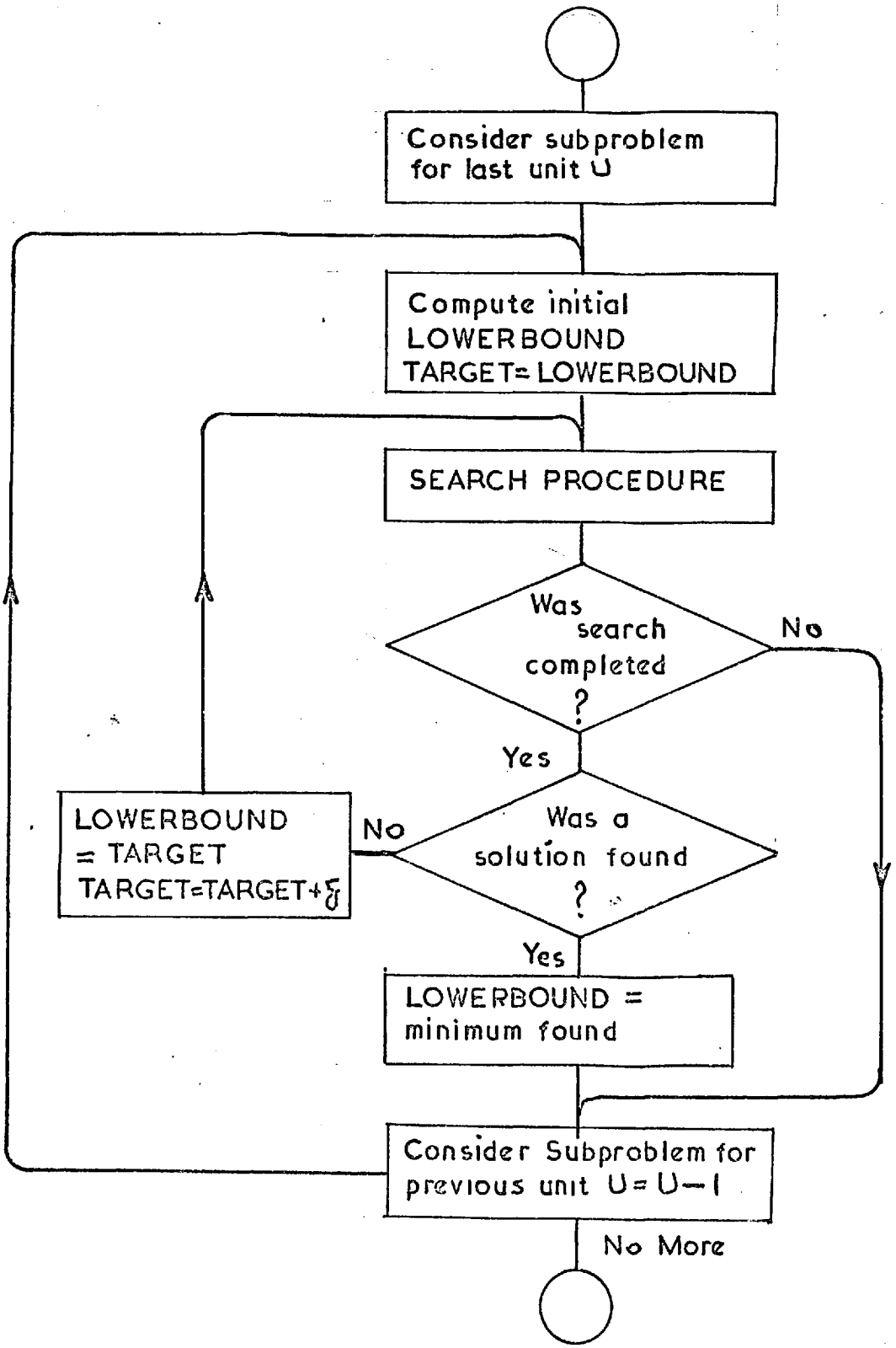


FIG. 2-4 : STRATEGY FOR FINDING IMPROVED LOWER BOUNDS

The strategy for solving the complete problem is shown in figure 2-4. A particular feature of this strategy is that it caters for the eventuality of a sub-problem being too difficult to solve, and when this arises, the highest target value which resulted in a completed search is used. In order to measure whether a sub-problem is too difficult, a count is incremented every time the solution moves forward after a successful unit placement. Before each sub-problem is commenced, a limit is set on the number of forward steps allowed in the basic search algorithm. If this limit is exceeded then the search procedure exits before completion. This feature is important because in some cases, bad orderings can arise within a sub-problem, and it is not worth spending an excessive amount of time finding a solution.

It is clear that each sub-problem is free of any constraints and costs which may be imposed on it by the rest of the layout and hence the lower bound computed may well be less than the contribution of the sub-problem in the completed layout.

2.5 Results of Test Runs.

In order to provide experience of the use of the techniques described above, the search procedure was implemented, and then this was embedded in a second system to investigate the acceleration obtained by computing improved lower bounds. The input to these systems comprises a list of pipe connection costs followed by lists of relative and absolute constraints. Pipe connections and relative constraints are only permitted between pairs of

units. Branched pipes are analogous to constraints involving more than two units and can only be handled in the test program by replacing the branched pipe by a set of pipes connecting pairs of units. In some cases, this representation can give a good approximation, but in others it is poor. Vessels of differing size and shape are handled by rigid sub-assemblies of units, with contraints imposed to maintain a fixed distance between units, however pipe terminal points are restricted to be from the centre of the units only.

Layouts are generated by the test program on a two dimensional rectangular grid. The extension to three dimensions is straight forward, but this would lead to a large increase in problem size. This increase could be offset to some extent by constraining many vessels to be on specific levels of the grid.

There are seven types of constraint that are permitted between pairs of units. If 'd' is the distance between units, x_1, y_1 and x_2, y_2 are the coordinates of the two units, then the seven constraints are:

d = specified value

d < specified value

d > specified value

$x_1 = x_2$

$x_1 < x_2$

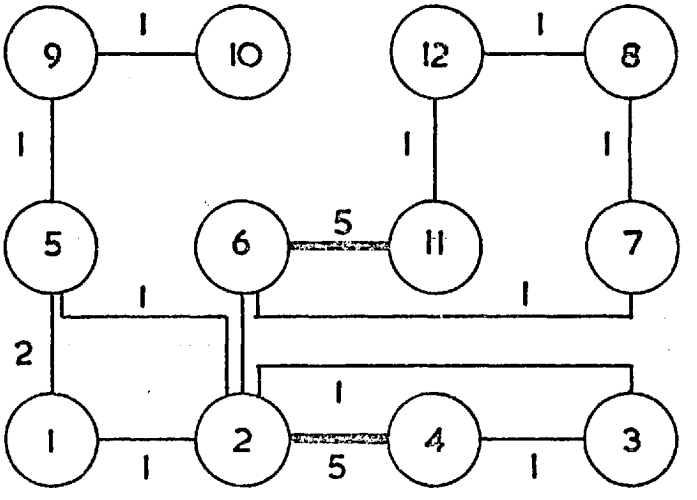
$y_1 = y_2$

$y_1 < y_2$

In addition, six types of absolute constraint can be imposed on unit positions:

x = specified value

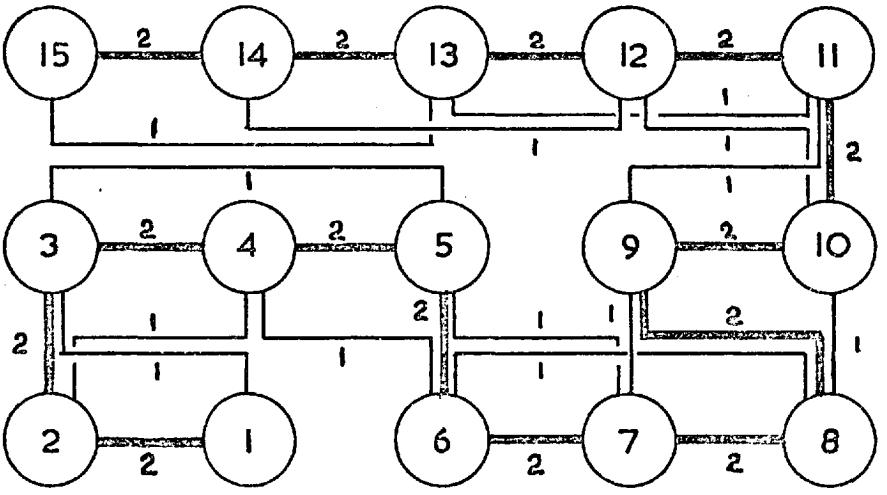
Minimum cost = 26



Constraints : $X_1 < 3$; $Y_1 < 3$

FIG. 2-5: EXAMPLE 1

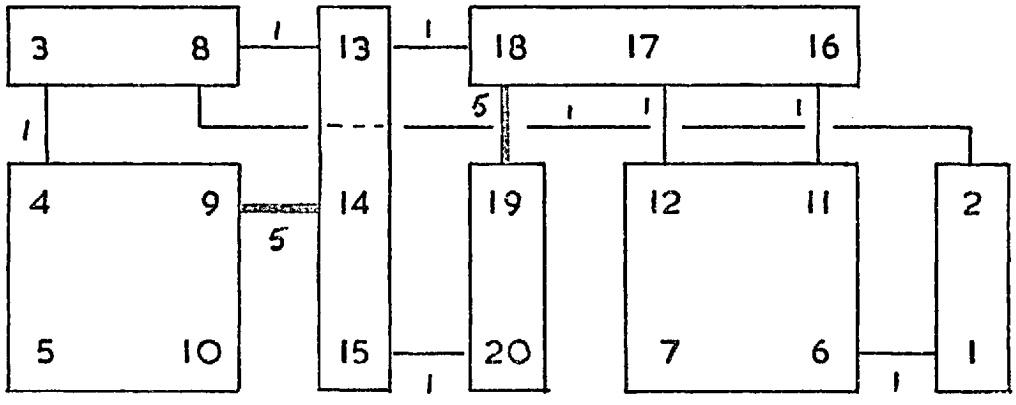
Minimum cost=52



Constraints : $X_1 < 4$; $Y_1 < 3$

FIG. 2-6 : EXAMPLE 2

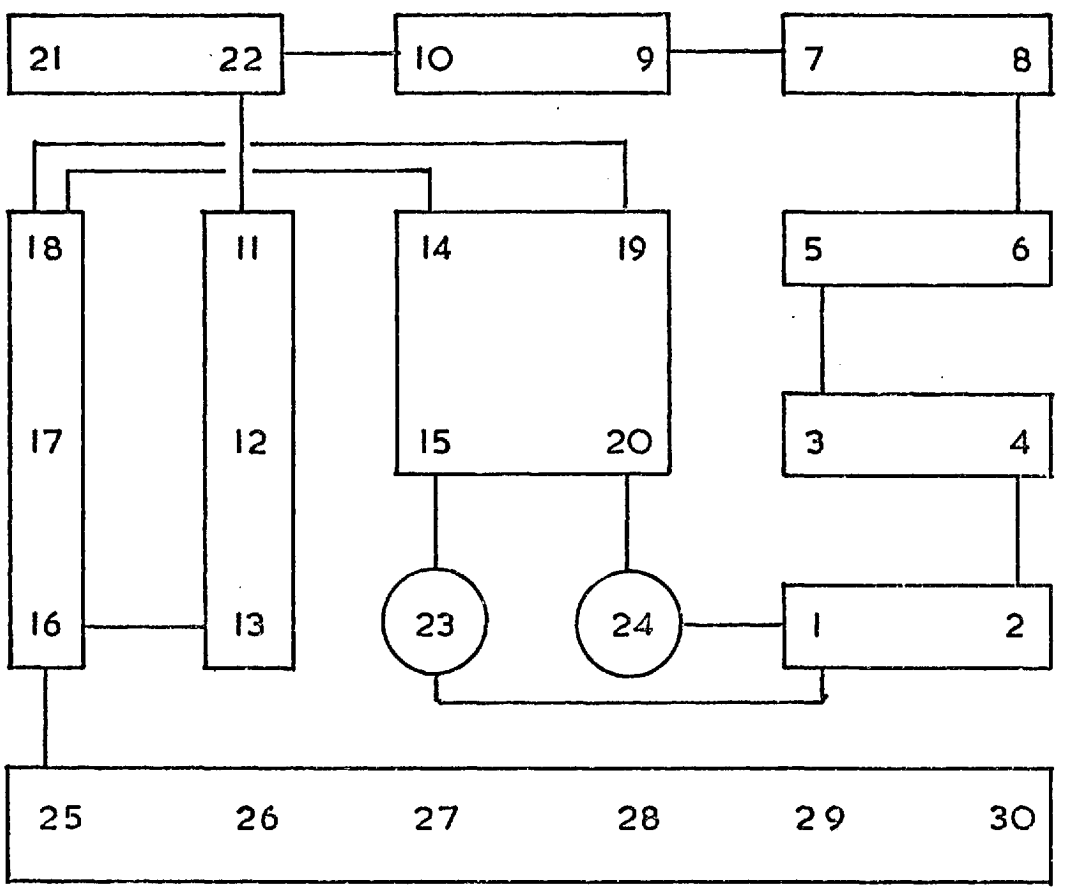
Minimum cost = 23



Constraints : X9 <5; Y4 <3

FIG. 2-7 : EXAMPLE 3

Minimum cost = 18

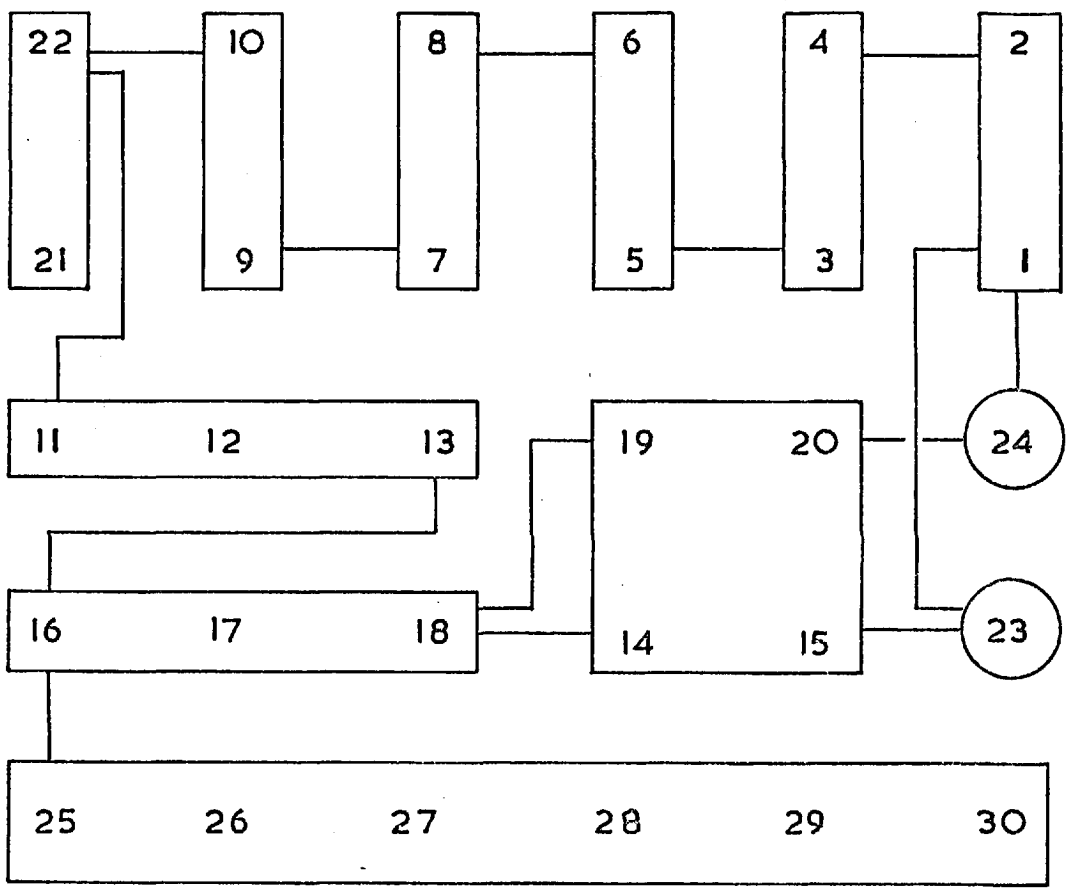


Constraints : $X_{25} < 4$; $Y_{25} < 4$

$X_{11} = X_{13}$; $X_{16} = X_{18}$

FIG. 2-8 : EXAMPLE 4 (a)

Minimum cost = 19



Constraints :

$Yn > 3$ for $n = 1,2,3,4,5,6,7,8,9,10,21,22$

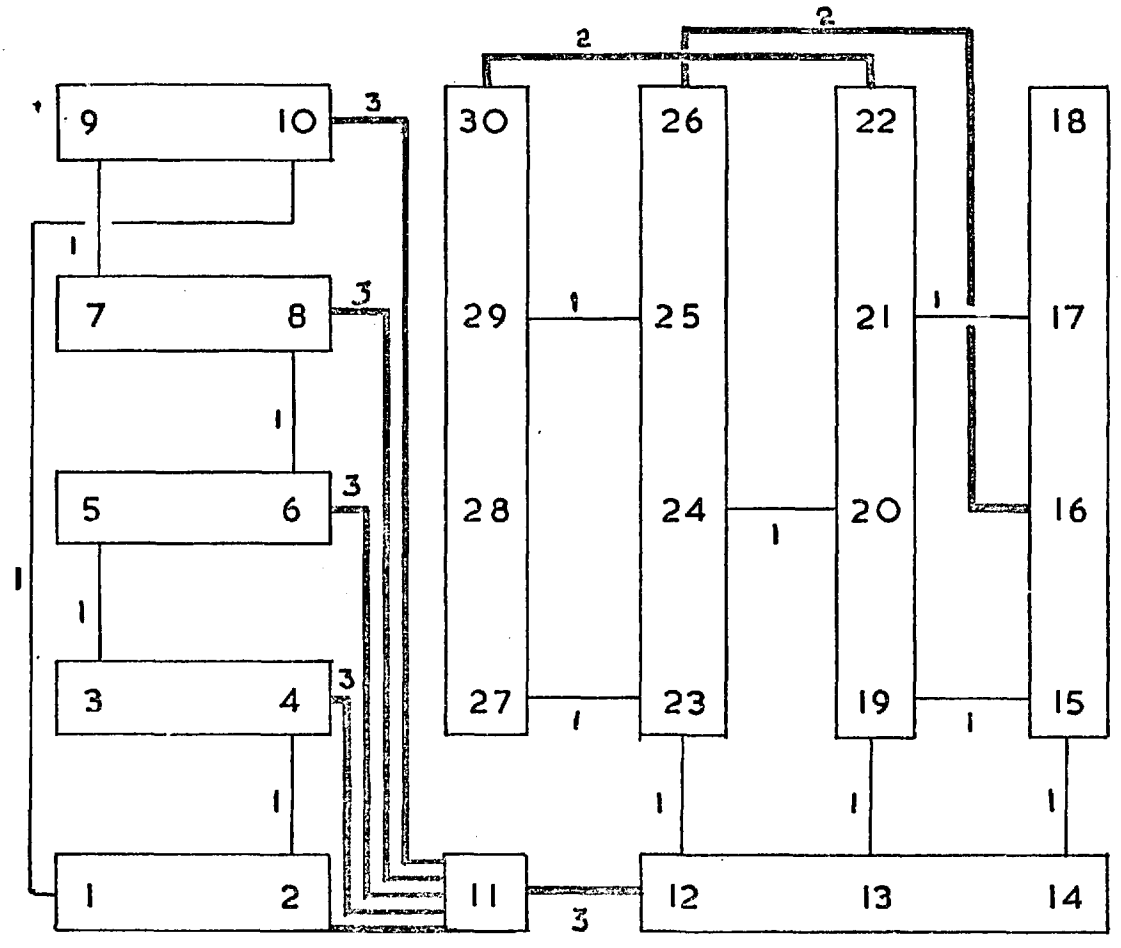
$Xi = Xj$ for $(i,j) = (1,2),(3,4),(5,6),(7,8),(9,10),(21,22)$

$X25 < 4$

$Yn < 4$ for $n = 11,12,13,16,17,18,14,15,19,20,23,24,25,26,27,28,29,30$

FIG. 2-9: EXAMPLE 4(b)

Minimum cost = 77



Constraints :

$X_i < 3$ for $i = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$

$Y_i = Y_j$ for $(i, j) = (1, 2), (3, 4), (5, 6), (7, 8), (9, 10)$

$X_i > 2$ for $i = 11, 12, 13, 14$

$Y_i < 2$ for $i = 11, 12, 13, 14$

$Y_i < 4$

FIG. 2-10 : EXAMPLE 5

x < specified value

x > specified value

y = specified value

y < specified value

y > specified value

In all the test examples, the grid size is taken to be unity and pipe lengths are computed as the sum of the distances between unit centres along the grid directions.

The five test problems are illustrated in figures 2-5 to 2-10. In some cases, constraints are imposed on the layout to avoid searching for symmetrical solutions. For example in example 1, unit 1 is constrained to lie within one quadrant of the layout. When vessels are introduced comprising more than one unit then these are handled by fixed distance constraints. In example 3, the vessel comprising units 13,14 and 15 is specified by fixing the three distances (13,14)(14,15) and (13,15). As already mentioned, the basic measure of efficiency is taken to be the number of steps forward in the search procedure.

Example 1: This example was tackled for two orderings of the units as follows:

(a) 1 2 3 4 5 6 7 8 9 10 11 12

(b) 6 11 2 4 3 1 5 7 8 12 9 10

When the simple search procedure was applied to this problem, (a) took 2552 steps and (b) took 4085 steps, thus illustrating the marked effect of the ordering on the solution time. In both cases, the initial target cost was high. As an experiment, (b) was solved given the minimum target of 26, but the number of steps were reduced by a small amount to 3973.

When the method of improved lower bounds was applied to the two orderings, the final problem took 229 and 2189 steps respectively. In the case of (a) this represents a marked acceleration in the solution speed, however one sub-problem of (a) took over 6000 steps because the ordering was particularly bad. Overall, the method of improved lower bounds took significantly longer for (a) and slightly longer for (b).

Example 2: This example was deliberately chosen as a case which would benefit from the method of improved lower bounds. When the simple search method was applied to this problem, it was fairly quick to find a minimal solution, but this was lucky because there are many alternative solutions with the same minimum value. However, after finding the minimum, the search was nowhere near terminating after a further 150000 steps, when the program was halted. When the method of improved lower bounds was tried, the final sub-problem took only 2857 steps to complete the search. This acceleration is easily explained by comparing the lower bounds used:

simple lower bounds:

41 39 36 33 30 27 24 21 18 15 12 9 6 3

improved lower bounds:

51 49 45 41 37 33 29 25 21 17 14 11 7 3

For the first few units placed, the simple lower bounds are 9 or 10 lower than the improved lower bounds, which means many more fruitless placements for these units.

Example 3: This example illustrates the use of fixed distance constraints in order to handle vessels of differing sizes and shapes. The simple search procedure

took 100354 steps when starting with a high target cost, but it found the solution in 32879 steps when given the minimum cost of 23 as a target. The method of improved lower bounds took 29465 steps for the final stage, however, the total steps taken for all sub-problems leading up to the final problem was very much more than this.

Example 4: This example is tackled for two differing sets of constraints, the constraints in 4(b) being more severe than in 4(a). In neither case did the simple search terminate after 150000 steps, even with a low target value. However, both problems were solved by the method of improved lower bounds, taking 33511 and 57767 steps respectively for the final stage. For many of the sub-problems tackled, no solution was found within the prescribed limit of 25000 steps. The improved lower bounds were only marginally above the simple lower bounds, but this was enough to cause a significant acceleration in the search.

Example 5: This example is a heavily constrained problem where the final pipe costs are very much higher than the simple lower bound estimates. The simple search procedure took a total of 95091 steps, but the final stage of the method of improved lower bounds took only 2477 steps.

These results show that for simple problems, intelligent search methods can find optimal solutions in a reasonable time. In some cases, the method of improved lower bounds takes longer overall than does the simple search procedure, but in those cases where it takes less, the simple search procedure could take hours or even days

to find a solution. A useful point in favour of these methods is that the designer can provide 'help' in the form of additional constraints whenever a problem is encountered which takes an excessive amount of time to solve. These additional constraints have the effect of restricting the search to a smaller number of options.

CHAPTER 3

ALGORITHMS FOR PIPE ROUTING.

3.1 Introduction.

Chapter 2 was concerned with finding the optimal layout of vessels on a site, and in so doing it was found necessary to make simplifications to the costing functions in order to gain maximum speed of solution. At the end of this first stage, very little is known about where each individual pipe will run, although crude estimates have been made of the costs of each pipe, based on the fact that most pipes run orthogonally in directions parallel to the coordinate axes of the plant. The first stage assumed that estimating piping costs was essentially a two dimensional problem, but to obtain an absolute estimate of costs as opposed to a relative one for different two dimensional layout representations, it is of course necessary now to find optimal pipe routes in three dimensions. In the layout stage costs were simplified in the hope that an 'optimal' layout so found would be close to the true optimum if more sophisticated cost procedures are used.

This chapter is concerned with obtaining a better estimate of the cost of pipes associated with a fixed layout of vessels. The resulting output of this phase will give the broad scheme of pipe routes such that the basic geometry of each route is known, and hence the positions of pipe bridges and racks can be determined, and a cost estimated for them. Procedures have been developed which will route simple pipes and branch pipes optimally such that the pipes avoid any prescribed obstacles in space.

The criteria for assessing a good piping configuration are not always defined by engineers in

economic terms. However the engineer will often wish to incorporate certain desirable features in the design which he himself can only express in a qualitative way. It would in fact seem more logical if costs could be ascribed to these various alternative qualitative desires since overall cost is what we are trying to minimise. To take this to extremes would mean that if any mandatory constraint is broken, then this will incur an infinite cost. It has therefore been attempted to develop pipe routing procedures such that qualitative constraints can be interpreted by adjusting the costing mechanisms of the routing procedures. Using this notion it is found that by adjusting costs, either automatically or interactively, the routing algorithms can be influenced so as to choose the most desirable of a number of otherwise equal alternatives and by extending this principle the problem of minimising support costs can be tackled.

Again in this second phase of the overall design it has been necessary to make certain simplifications and assumptions before tackling the problem. Thus the problem is presented as finding optimal solutions to an approximate representation of the true problem in the real world. In some cases procedures have been found which find the true global optimum to these approximate problems, but in others it has been found necessary to devise heuristic procedures, or recipes, which find an optimal or near optimal solution for most of the time. The methods used involve a form of hill climbing by improving and updating an existing solution until no further improvement can be made, thus finding a local optimum. In these cases the excessive

amount of computation which obtaining a guaranteed global optimum might take cannot be justified.

The procedures used to determine optimal pipe routes depend to a large extent on the shortest route algorithm. That is, given any network whose elements have positive lengths, then find the shortest path through the network from a start node s to a terminal node t. However the problem of routing a single pipe optimally is presented as finding the shortest route in space between two terminal points. The first stage of setting up the problem is to replace the spatial representation of the plant by a three dimensional network, the elements of which are oriented in one of the three orthogonal directions which constitute the basis of coordinates for the site. This constraint is imposed for practical reasons, but should a more general form of network be employed, then the procedures used to find optimal routes are still valid.

It is fortunate that the shortest route problem is a solved problem for which solutions can be obtained quickly, and it is therefore very useful to exploit. By a simple extension of this algorithm we can find the minimal spanning tree of a subset of three nodes of a graph and by a further extension of this second procedure we can derive a procedure to find approximate optimal solutions for the minimal spanning tree for larger subsets of nodes. Further, by a simple extension of the network it is possible to find the minimal route between two nodes with respect to pipe length and the cost of bends. This bend minimising algorithm can be extended in the same way to find minimal spanning trees for more than two nodes.

The basic tools used in this chapter are a three dimensional grid or network on which the pipes are to be routed and a series of pipe routing algorithms. The network is set up such that any route is guaranteed to miss any prescribed obstacles in space and such that the basic topology of the route in space has a unique mapping onto the corresponding route in the network, (the converse mapping is not unique). However it is permitted at this stage for there to be mutual clashing between the pipes themselves, that is routes may cross or run concurrently.

The latter of these two violations is equivalent to saying in the real world context that the two pipes run along the same pipe bridge or pipe rack. Thus at this stage of the design it is assumed that for each element of the network there is adequate space to accomodate all the pipes that run along it. It is therefore foreseen that in the case of a compact plant where space is at a premium, these procedures may become inadequate.

The discussion so far has only been concerned with the optimal routing of a single pipe, whether it has two or more terminals, and no mention has been made of the optimisation of the piping configuration as a whole. At the present time a realistic economic assessment of any particular piping configuration is hard to come by. However rules can be formulated which impose constraints on individual pipe runs, for example stress criteria must be satisfied. Also the overall scheme must be satisfactory from the point of view of pipes running along similar routes, such that support costs can be minimised. In addition it may be known before any routing commences that

certain parts of the plant are more favourable for routes than others, because a support already exists there, for example there may be an existing pipe bridge or the pipe may be supported by a vessel. These possibilities can be expressed quantitatively by assigning artificial costs to the elements of the basic three dimensional network; in other words elements which are considered favourable for pipe routes can have low costs assigned to them and elements which are unfavourable have high costs. Thus any a priori assignment of costs to the elements of the network will influence the routing of each pipe such that it tends to take the more favourable route. Of course the difficulty now arises as to just what quantitative values to assign to these qualitative desires. This gets us back to the problem of what is the objective function, what are we trying to optimise? In its simplest form we wish to minimise the cost of each individual pipe run together with the costs of supporting those pipes. With regard to the first of these items it is assumed that a subroutine exists which, given the topology of a pipe route and other necessary information (e.g. desirable pressure drops, and piping costs) will assign sizes to the various branches of the route.

The problem of minimising support costs is a little more nebulous and no overall algorithm which finds a joint global optimum has been derived. However it has been found that pipes can be encouraged to follow similar routes to each other by reducing the costs of elements along which pipes have already been routed. The size of such reductions may change from problem to problem and this is possibly a

case where intervention in an interactive manner may be useful. Some experiments for this research have been conducted by reducing costs automatically and the solutions derived can be seen to route many of the pipes together. However by manipulating these elemental costs interactively, the user could in a short space of time evaluate a number of different alternative solutions, and could intervene in the design process whenever he considers the machine to have produced an unsatisfactory configuration.

3.2 Outline of Related Problems and Methods.

There are a number of problems posed in the literature which are of a similar nature to finding a shortest pipe route when posed in the manner given in the introduction. The shortest route problem has already been mentioned and the methods used to solve this problem can clearly be used directly in the case of a pipe with two terminals only(e.g. 8). Nicholson (9) has proposed an efficient method of solving the shortest route problem which compares very favourably with alternative methods. In particular his method is particularly efficient in cases where one requires to find the shortest route between two nodes which may be fairly close together in a very large network, since the method only searches that part of the network near the terminal nodes. Farbey et al (10) proposed an efficient method for finding all the shortest routes in a network, but it is too time consuming for finding only a few shortest routes.

The shortest route problem needs extending if it is

to cater for the wider range of problems posed in the routing of branch pipes, usually the piping configuration needed to span several terminals is in the form of an undirected tree, i.e. there are several branches, but no loops in the pipe. Thus it would seem that the work done by Kruskal (sec 11) (and later repeated by Loberman and Weinberger (12)) on minimal spanning trees should be of relevance. Kruskal's constructions find the minimal spanning tree of all the nodes of an undirected graph.

Loberman and Weinberger, although setting out to find the minimal total wire length for connecting terminals, in fact only do so provided that wires are restricted to go directly from one terminal to another. In short, they are finding the minimal spanning tree of the complete graph of all the terminal nodes. If one allowed auxiliary terminals to be included in the graph (whose position needs be determined by the minimisation procedure) then wiring configurations could be found which require less wire than their method.

In the case of pipe routing one requires to find a minimal spanning tree of a subset of nodes of a finite graph. Similarly, the problem of Loberman and Weinberger could have been posed as finding the minimal length of wire required to connect a finite subset of nodes of an infinite graph, the graph being the complete graph of all points in space, i.e. an infinite number of nodes and elements.

Therefore because Kruskal's constructions are not sufficiently general, it is not possible to apply the method to the problem of finding minimal branch pipes.

One other related problem which is worth exploration is the Transhipment problem (see 11), in which it is required to ship goods from a set of nodes called origins to another set of nodes called destinations. Goods are permitted to be 'transhipped' via other nodes of the network to accomplish this task. The branch pipe problem has a number of similarities to the transhipment problem. Firstly both problems have their solutions in the form of a tree. Secondly, the cost of each element of the tree is the product of the original network element length and a cost factor. However, in order to cast the branch pipe problem as a transhipment problem, one needs to assume that unit pipe costs are proportional to flow rate. The complete solution is then a tree which embraces all the nodes of the network, however, many branches of the tree will have zero valued elements and are not really part of the required solution. In addition, this tree solution, although being topologically connected, may in practice be disconnected due to some basic variables having zero values (degeneracy). Even if the above assumption and inadequacies are accepted the size of problem generated is likely to be prohibitive, for example Dantzig gives an example with 8 nodes, typically a chemical plant may generate networks of 500 nodes leading to a 500x500 tableau. This tableau would consist largely of null entries but the search time needed to move from one basic solution to the next would be large.

The brief conclusion to this section is that it does not seem possible to solve the branch pipe problem by applying any published methods directly.

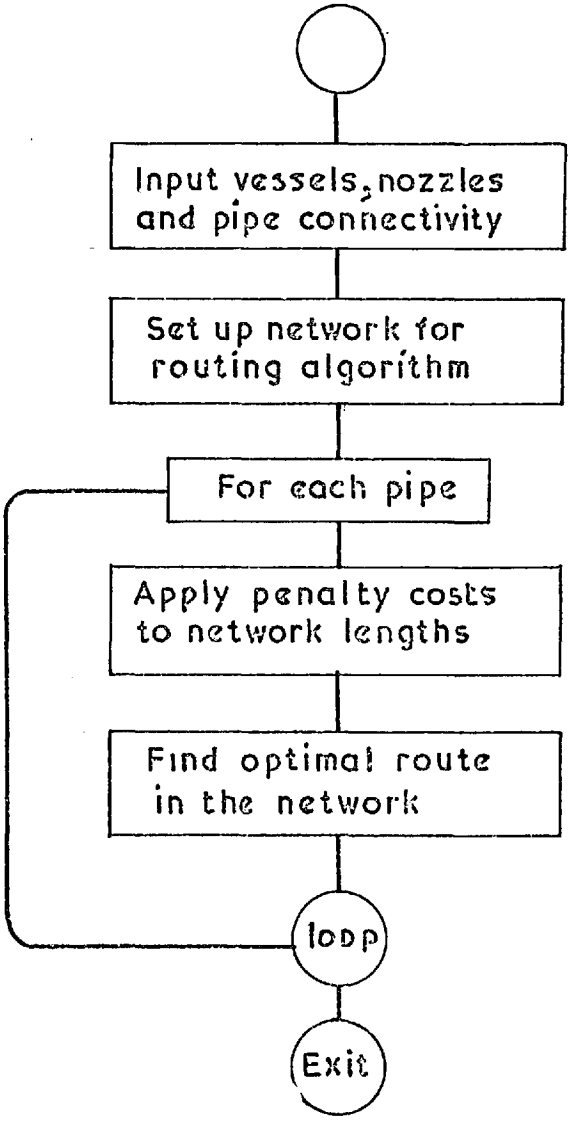


FIG. 3-1: FLOW CHART OF SIMPLE PIPE ROUTING SYSTEM

3.3 Structure of Simple Pipe Routing System

In order to introduce the various stages of the routing algorithms we refer to the flow chart in figure 3-1. This illustrates the simple pipe routing system that was implemented to test the algorithms. The remainder of this chapter will be devoted to describing in some detail the workings of each stage of the flowchart. General requirements of each stage and the particular attributes and shortcomings of the working program will be given.

3.4 Input Definitions.

The unit definitions used have been restricted for the sake of simplicity. The vessels are represented by cylinders of given length and radius. The position of the cylinder in space is given by the coordinates of the end points of its central axis, and the direction of this axis is constrained to lie parallel to one of the three coordinate directions.

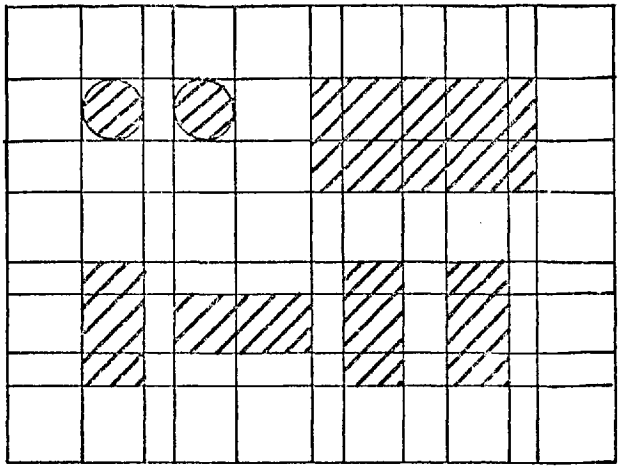
Any given pipe is specified by a list of terminal points, each terminal being associated with a vessel. Terminal points of pipes are always assumed to start on the central axis of the vessel. The vessel has ends 1 and 2 (i.e. botton and top) and the pipe is defined by two orientations, the first orientation is relative to the vessel and can be one of three options, either the pipe emerges from the top, bottom, or side of the vessel. The second orientation is the spatial one and the pipe is constrained to go up, down, or horizontally. This simple system does not cater for all the cases that would arise in real situations but has flexibility to enable the

formulation of realistic problems. In specifying the two orientations of the pipe nozzles, care must be taken to ensure that these are compatible, e.g. a pipe coming out of the side of a vessel whose axis is vertical must be oriented horizontally in space, thus yielding four possible directions in which the pipe may emerge from the vessel.

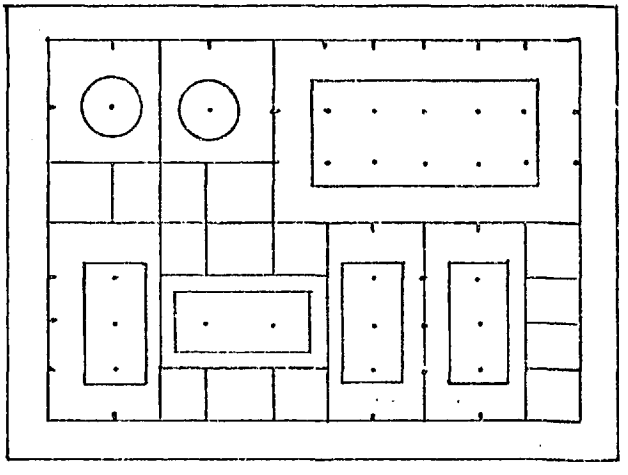
It will be assumed that sufficient information is available for the desired performance of the pipe, such that for a given tree topology, it is possible to determine pipe costs per unit length for each branch of the pipe. In this research this has been done by assuming that a pipe cost depends only on the quantity of fluid flowing through it, and thus for each terminal, the quantity of fluid entering or leaving the pipe at that point is given. Thus the sum of all flows over all the terminals should be zero. Clearly this is a simplification, but the routing procedures themselves do not depend on the method of costing used.

3.5 Algorithm to Set up the Network of Permissible Routes.

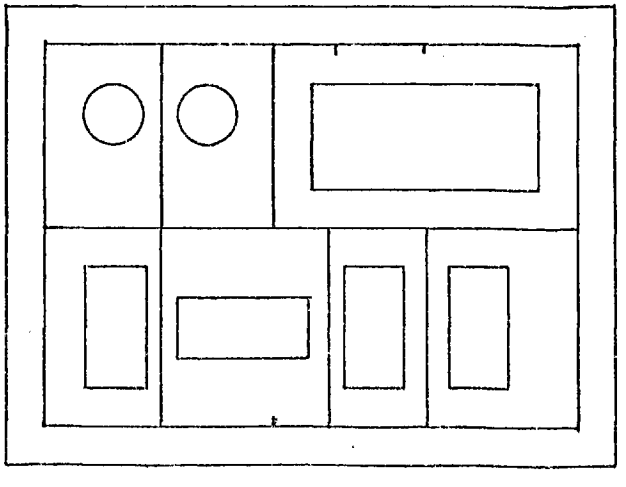
The basis of nearly all that is described subsequently in this chapter is the network of permissible routes. If one describes two points in space which have to be connected optimally, subject to avoiding any number of obstacles, then the solution space for searching is infinite. As mentioned before, in the case of chemical plant layout, there is usually the practical restriction that the majority of pipes run in the three orthogonal directions. However, even with this restriction, the number of alternatives is still infinite. Hence there is a need to



a: The Site Divided into 'Space Boxes'



b: Network Generated from the Space Boxes



c: Superfluous Elements Removed from Network

FIG. 3-2: STAGES IN THE GENERATION OF THE ROUTING NETWORK

discretise space in the form of a network, such that this network represents adequately the space available for routing pipes. However, there is no unique way of representing space in this manner.

The network generation procedure now proposed is based on logical considerations and usually produces a sensible network. Any inadequacies in the algorithm would best be remedied by allowing the user to intervene in an interactive manner, adding, deleting and moving network elements at will.

The procedure is probably best described with reference to figure 3-2. Although these are two dimensional diagrams, they should serve to illustrate the workings of the algorithm in three dimensions.

The first stage of the algorithm is to take every vessel (i.e. cylinder) and enclose it in a rectangular box which just contains it. Therefore the extreme coordinates of the box in the X,Y & Z directions are the same as the extreme coordinates of the vessel. It is assumed that the space occupied by this box is inviolable.

The second stage of the algorithm is to set up three sets of planes, each set being orthogonal to X,Y or Z. Each plane corresponds to one of the two extremum values of each box in each direction. Redundant planes (i.e. those which are coplanar with previously generated planes) are removed. In addition to these planes, six boundary planes are inserted, two for each coordinate direction and these are arranged at a specified distance from the smallest and largest extremum values in each direction.

The whole plant is now confined in a box, bounded by the six boundary planes as shown in figure 3-2(a). In addition this box is divided into a finite number of smaller boxes (henceforth termed 'space boxes'), each bounded by six of the internal planes inserted at the extremum values of the vessels. Thus we have succeeded in discretising the space occupied by the plant into a set of boxes, some of which are wholly inviolable and others which contain only vacant space.

The network is now generated from the discrete set of boxes by representing each box by a node at its centre point and each face separating two adjacent boxes by an edge joining the centre points of the two adjacent boxes. Clearly any node positioned within an inviolable box is never used by the routing algorithm, the reason for including such nodes is to save on computer storage space. It is more economical to store the complete network in a regular and well structured form than to store only the relevant part of the network in the form of a generalised list structure.

A problem which arises at this stage is the number of superfluous planes of network elements caused by vessel extremum points not quite lining up. The original purpose of the network was to represent the main corridors of space running between plant items by elements of a network, thus a procedure is required which eliminates all planes of network elements which are considered redundant in this context. Hence it might first be useful to define what is meant by a corridor.

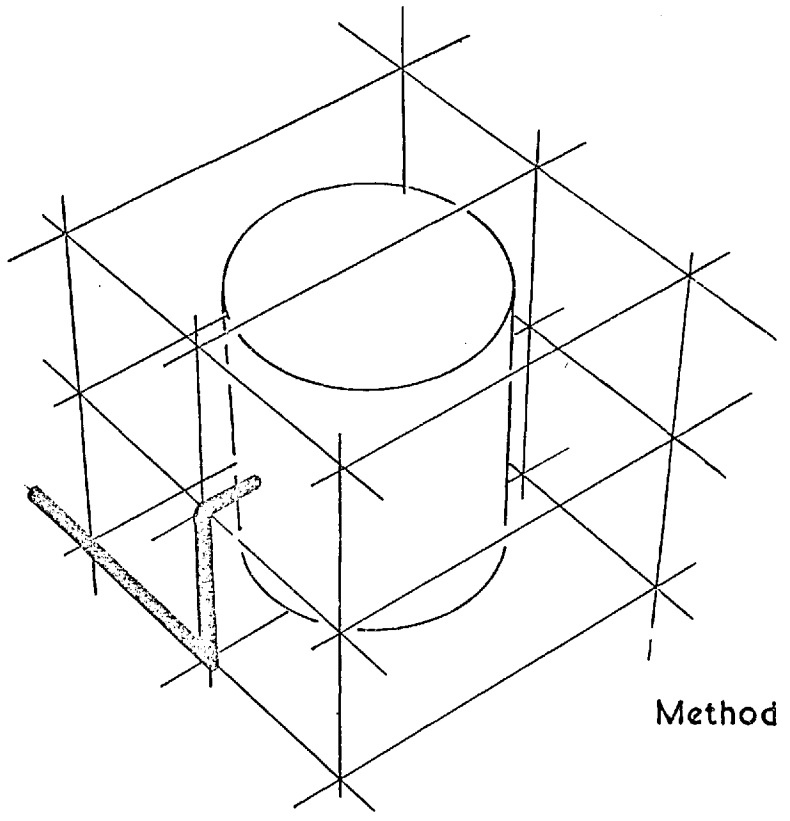
A corridor consists of all the space boxes bounded by any two adjacent parallel planes, provided that any two vessels lying on each side of the corridor have extremum values lying in the boundary planes of the corridor.

Using this definition, it is therefore necessary to remove from further consideration all network elements which correspond to a plane of space boxes not constituting a corridor. The above definition can be expressed mathematically in another manner, thus yielding a simple procedure for selecting those parts of the network representing corridors and rejecting the rest. To simplify the formulation it is assumed that the six bounding planes are generated from a large 'negative' box which encloses the plant. The term 'negative' is here used to mean that the exterior only of the box is inviolable.

If we wish to examine the set of space boxes between two adjacent planes orthogonal to one of the three coordinate axes, then consider the ordered set A of bits, where each bit represents one of the space boxes. A bit is set to 1 if it corresponds to an inviolable box and 0 if it is vacant space. Now let A1 and A2 be the corrsponding bit patterns for the two sets of space boxes on either side of A (the one being tested), then A corresponds to a corridor if and only if

$$A \vee A1 \neq A; \quad A \vee A2 \neq A$$

This provides a very simple method of detecting a corridor. The bit patterns for each potential corridor can be stored compactly in machine words and these can be tested very efficiently with the machine 'OR' function.



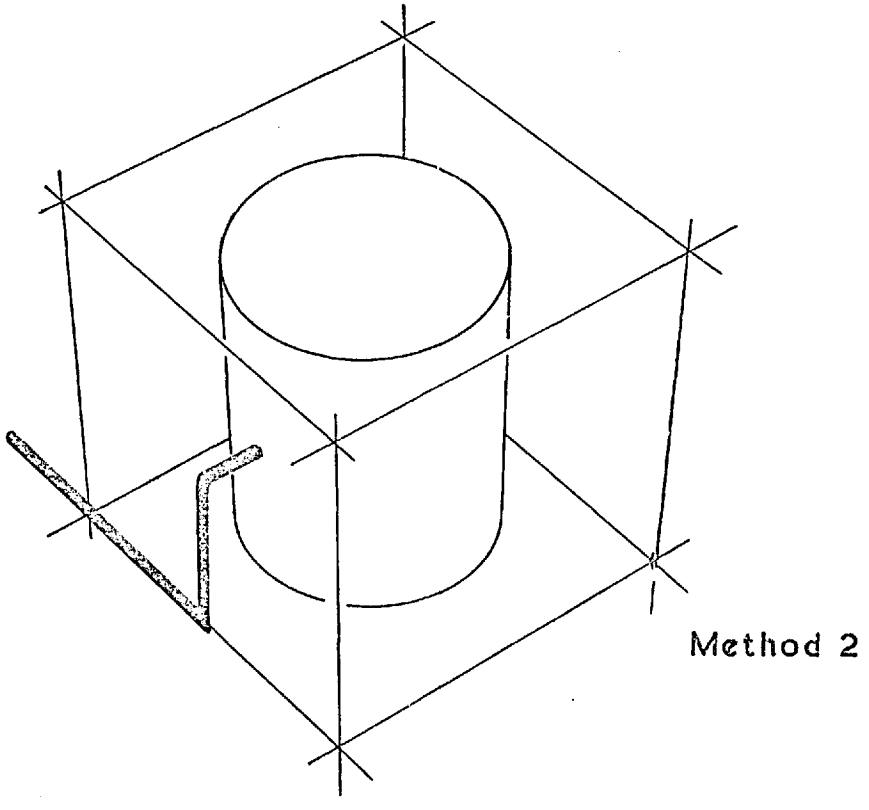


FIG. 3-3: METHODS OF CONNECTING NOZZLES TO THE NETWORK

The stage has now been reached whereby all planes of network elements corresponding to a corridor have been detected. If these elements only are included in the network of allowable routes then at least one is assured that all vessels are confined in a box which contains no other vessel and the edges of this box are elements of the network of feasible routes. It might be considered at this stage that if all other network elements are eliminated then the resulting network may be too coarse a representation of the space available for routing pipes. For this reason it is possible to arrange that some of the planes of elements are left in provided they lie at a distance greater than a specified minimum from the nearest adjacent corridor.

There is now only one problem left to complete this spatial representation and that is if one is going to route pipes along the elements of the network then one needs to connect the nozzle positions on the vessels to the network. Two methods of providing the link between nozzles and network have been investigated and these are described with reference to figure 3-3.

The first method leads to the simpler solution which enables all cases to be catered for. This method is simply to insert two planes for each nozzle position. For example if the nozzle is constrained to point in the direction X, then planes are inserted orthogonal to Y and Z such that the Y and Z coordinates of the nozzle lie in the inserted planes. Doing this ensures that there is a node directly in line with the nozzle with no obstruction between it and the nozzle. To route a pipe from a nozzle to the network

becomes trivial since it requires a single straight piece of pipe. Of course adding network elements in this manner enlarges the storage required for holding the network. This does not arise in method two.

Method two is less elegant than method 1, but although it is not always sufficiently general, it does have the advantage of not requiring so much store. It was mentioned above that every vessel is confined in a box. Thus if one considers a pipe emerging from a nozzle it must meet one of the six faces of the enclosing box. If one now considers the four corner nodes of this face then the problem is to route the pipe from the nozzle to these corner nodes (the case might arise where by coincidence the centre of the nozzle passes through a network element or node). This is done by generating a small piece of local network which is then interfaced with the main network.

An example of the inadequacy of this second method is the case where the nozzles of a two terminal pipe are located in adjacent space boxes such that they both impinge on the same face. Clearly the pipe should go straight through the face instead of via a corner node. To cater for this situation renders the method even less elegant and in the case of routing branch pipes it becomes totally impractical. For this reason the first method is greatly to be preferred and subsequent references to the network assume that this is the method implemented in the test program.

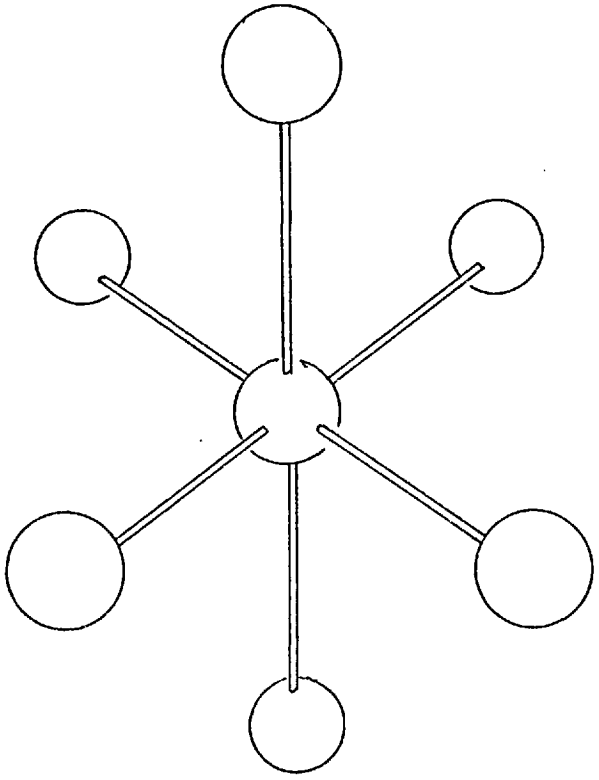


FIG. 3–4: INTERNAL NODE OF NETWORK WITH 6 ADJACENT NODES

3.6 Storage Structure for Networks.

It was mentioned above that some of the nodes stored in the network will never be used by the routing procedures. The reason for including them is economy of storage. This section gives the details of this advantage. There are three sets of information relating to the network which need to be stored, firstly the positions of the nodes in space, secondly the connectivity, and thirdly the costs or lengths ascribed to the network elements.

In general, any node of the network is connected to up to six adjacent nodes. Because the complete network is held in store such that all nodes are numbered in a logical sequence, it is possible to calculate from the node number the row, column and level in which the node lies, (row, column and level being associated with X,Y and Z axes respectively). Therefore in order to determine the positions of the nodes, it is only necessary to store the single coordinate of each row, column and level.

Each node in the network is connected to up to 6 adjacent nodes, as shown in figure 3-4. It is possible to calculate all six adjacent node numbers without storing the adjacency relationships explicitly (in the case of boundary nodes some of these don't exist). Therefore if these six adjacent nodes have a relative numbering system such that in the positive X,Y and Z directions they are numbered 1,2,3 and in the negative X,Y and Z directions 4,5 and 6, then all that need be stored for each direction is a single bit, set to 1 or 0 depending on whether or not it is possible to go to the adjacent node. Thus in its most

compact form the connectivity of the network can be held in six bits for each node. Bit manipulation is not very efficient in most high level computer languages and so the following alternative scheme was devised. Each of the six directions is assigned one of the first six primes (2,3,5,7,11 and 13). The connectivity of any one node is then held in one integer word which contains the product of the primes corresponding to directions in which it is not possible to move from the node. If it is permitted to move in all six directions then the number held is unity. If the node itself is not part of the usable network (e.g. it lies inside a vessel) then the number held is zero. The maximum possible number generated in this manner is 2x3x5x7x11x13 = 30030.

The connectivity information can easily be extracted as follows. If for example it is desired to test whether the element in direction Z is usable then, perform an integer division by the third prime (five) and test whether the remainder is zero.

Each element of the network is connected to two nodes at its end points. Hence of the six possible elements connected to a node, it is only necessary to store three of these to avoid duplication of information. The element lengths or costs are held in a 3xn array (n = number of nodes), each column of the array holding the elemental costs in the positive X,Y and Z directions. If the cost of an element in a negative direction is required it is necessary to refer to the array column associated with the adjacent node in the corresponding negative direction.

The above scheme could of course be made more efficient from the point of view of retrieval time or more compact. The storage methods proposed are thought to be a good compromise.

3.7 The Shortest Route Algorithm Applied to Pipe Routing.

The basic theory of finding the shortest route is the following. If c_i is the minimum distance on the network between the start node 's' and node 'i', and d_{ij} is the length of element connecting the adjacent nodes 'i' and 'j' then:

$$c_i = \min_j (c_j + d_{ij})$$

where the minimisation is carried out over all nodes 'j' adjacent to 'i' on the network. In fact if this relationship is applied iteratively to all nodes in the network, then all costs c_i will converge to their minimal values. Such a method, although simple to implement, is not very efficient. Nicholson's method avoids this brute force appoach by performing the minimisations in an ordered manner.

Shortest route algorithms are normally designed for finding solutions in directed networks, but in the case of routing pipes it is only necessary to consider the particular case of undirected networks,

i.e. $d_{ij} = d_{ji}$

There is also usually a restriction that all elemental lengths d_{ij} should be non-negative, violation of this constraint immediately renders Nicholson's method

invalid. However, provided that no cycles exist in the network where the distance around the cycle is negative, then a solution to the shortest route problem exists. In the case of an undirected network, each element is really two elements pointing in opposite directions. If such an undirected element has negative length then there exists a cycle in the network consisting of the two corresponding elements and the above constraint is violated. Therefore in the case of routing pipes, the network will always be constrained to have non-negative elements. This is not a serious restraint since it is difficult to see a practical need for negative elements.

The basis of being able to find any minimal piping configuration for a number of nozzles is the ability to find the shortest distance between a start node 's' and a terminal node 't' on the network. Since a nozzle can emerge from a number of different directions then this implies that there is more than one possible start node corresponding to a nozzle. This difficulty is easily overcome by notionally extending the network by connecting the several alternative start nodes corresponding to the nozzle to a single extra node. The elements used for this purpose may have zero costs associated with them or a value related to the amount of pipe required to connect the nozzle on the vessel to its corresponding start node on the network. Hence we may always consider having only one start node and one terminal node in all shortest route problems.

The method of finding a shortest route consists of two stages. The first stage consists of finding the minimal distance of all nodes in the network from the start node

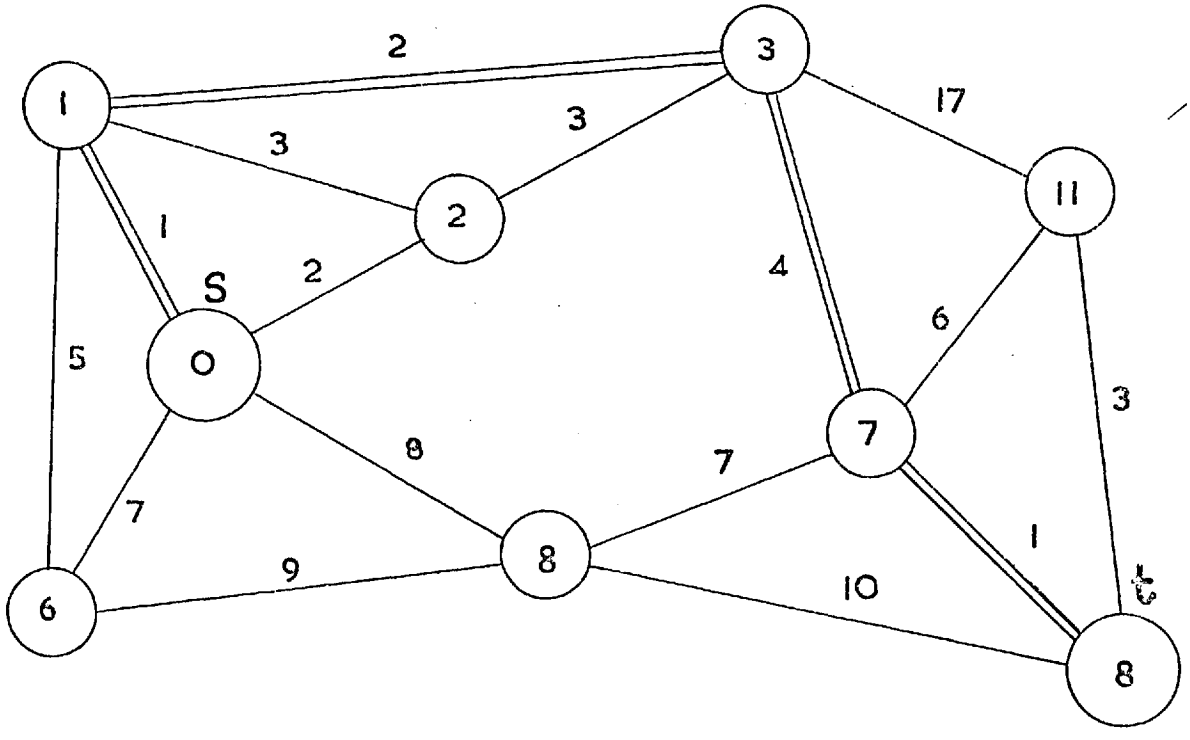


FIG. 3-5: THE SHORTEST ROUTE ALGORITHM

and the second stage consists of threading a route through these costs from the terminal node back to the start node. This is illustrated in figure 3-5 where the network was costed from node 's' and then the shortest route was found from 't' back to 's'. As Nicholson pointed out, it is not necessary to cost up the whole network provided this is done in an orderly fashion. Also it is not necessary to store the sequence in which each node is added to the list of costed nodes (as Nicholson does) since the shortest route is easily found knowing the minimal costs of all nodes from the start node. This saving in space may lead to a very small increase in computing time.

The method used to cost up the network is that of Nicholson. If one wishes to determine the shortest route from any node 'i' in the network back to the start node 's' then examine all the adjacent nodes 'j' of 'i' and if S_i and S_j are the minimal distances of nodes i and j from s then $S_i - S_j = d_{ji}$ means node j lies on the shortest route from s to i. In this manner it is possible to find the sequence of all nodes on the shortest route back to s. In some cases, there may be more than one shortest route solution, in which case, an arbitrary choice is made.

3.8 The Minimal Tree Spanning Three Nodes of a Graph.

If there are n nozzles then there are at most n-2 internal junction points in the tree. Each branch of the tree consists of a path of elements of the network of feasible routes. The end points of a branch can be a nozzle or a junction. If the tree is minimal, then each branch of the tree must be the shortest route between its end points.

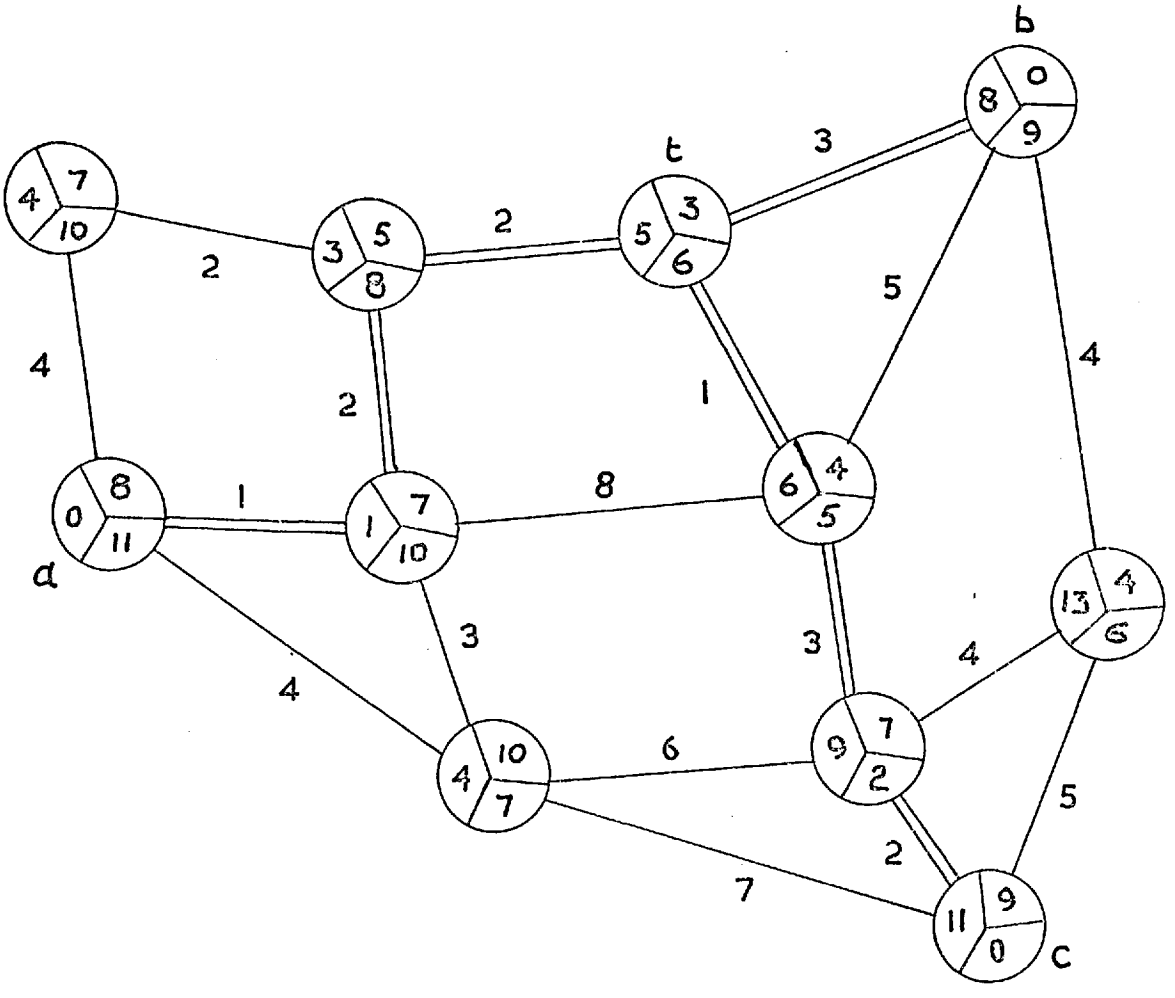


FIG. 3-6 : MINIMISATION PROCEDURE FOR THREE – BRANCHED PIPE

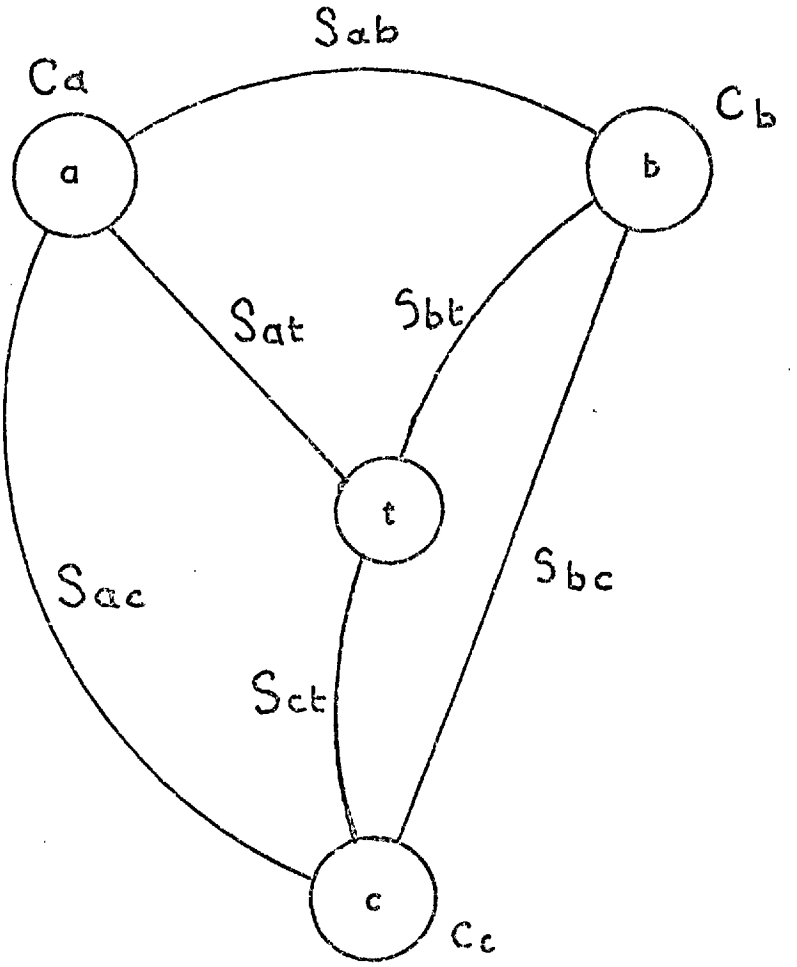


FIG. 3-7 : NOTATION USED FOR THREE-BRANCHED PIPE

Thus in the case of a pipe with three terminals there must be one junction point whose position in the network remains to be found. (In order to be consistent with the terminology, a 'nozzle' is defined to be a 'terminal' which abutts on to a vessel, and a 'junction' is a 'terminal' which is an internal point in the branch pipe. Thus a branch always connects two terminals).

In order to find the position of this junction point, consider the following procedure, illustrated in figure 3-6. If the three nozzles are labelled a,b,c then determine the shortest distances of all nodes i in the network from each nozzle S_{ai} , S_{bi} , S_{ci} . (Note $S_{ai} = S_{ia}$ etc.) If one considers the cost of pipe needed to place the junction at a particular node t, then

$$C(t) = C_a . S_{at} + C_b . S_{bt} + C_c . S_{ct}$$

where C_a , C_b , C_c are the unit costs of pipe connected to each nozzle. In order to find the junction point corresponding to the minimal branch pipe spanning a,b,c we merely have to find the node t for which $C(t)$ is a minimum.

As in Nicholson's method for two nozzles, it is again not necessary to find the shortest distance of all nodes from the nozzles. In figure 3-7, all edges shown are shortest routes in the network between their end points, which are nodes in the network. If we now consider t as the junction corresponding to the minimal tree then it may be noted that all triangles satisfy the triangular inequality constraint, also since the tree is minimal then

$$C_a . S_{at} + C_b . S_{bt} + C_c . S_{ct} \leq C_b . S_{ab} + C_c . S_{ac}$$

$$\leq C_b . (S_{at} + S_{bt}) + C_c . S_{ac}$$

$$C_a . S_{at} + C_c . S_{ct} \leq C_b . S_{at} + C_c . S_{ac}$$

if $C_a \geq C_b$ then $S_{ct} \leq S_{ac}$

and by symmetry

if $C_b \geq C_a$ then $S_{ct} \leq S_{bc}$

in other words the junction node t always lies within a shorter distance from any nozzle than the more expensive of the two remaining nozzles. Hence for any one terminal it is only necessary to examine routes as far as the more costly nozzle in order to find the minimal position of the junction.

It is also of interest to note

$$C_a . S_{at} + C_c . S_{ct} < C_b . S_{at} + C_c . (S_{at} + S_{ct})$$

thus S_{at} can only be greater than zero if

$$C_a < C_b + C_c$$

which means that if the unit pipe cost associated with one nozzle is greater than the sum of the other two, then the cheapest solution is to connect the other two branches directly to the nozzle.

Once the optimal position of the junction 't' has been found, it is merely a matter of solving three shortest route problems in order to find the actual routes taken by the three branches of the tree. This is easily done since the network has already been costed up from the three nozzles.

3.9 Minimal Spanning Trees of Larger Subsets of Nodes of a Graph.

To extend the procedure outlined above to larger subsets of nodes leads to an intolerably large combinatorial problem. Hence it is proposed that the

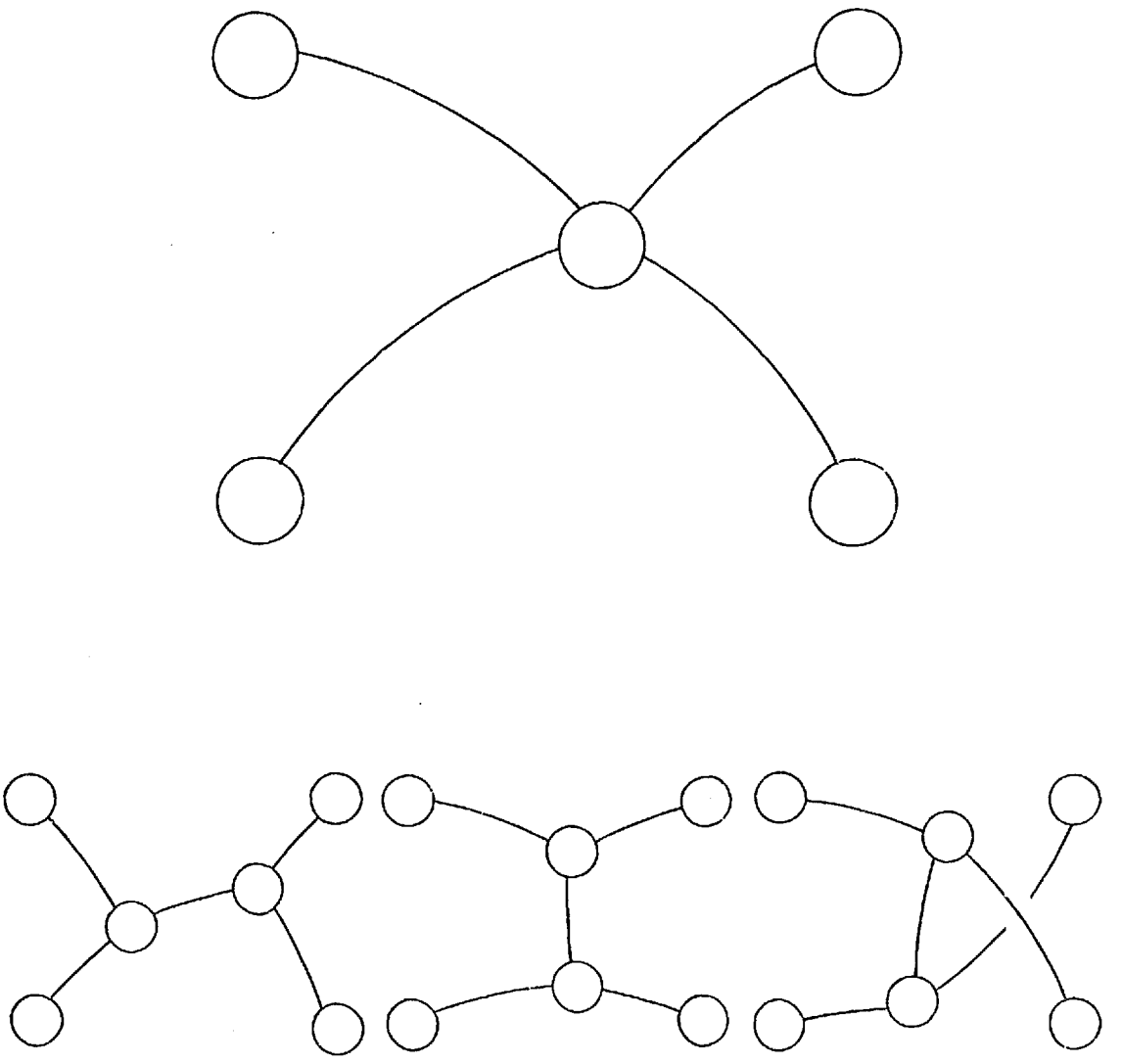


FIG.3-8: THREE WAYS OF DECOMPOSING A FOUR-BRANCHED JUNCTION

solution to these higher ordered problems should be based on an approximate method using the result obtained for three nozzles. If we consider that a tree spanning n nozzles has n-2 internal junctions, then each junction will be associated with three branches of the tree. It occurs in practice that a junction can be associated with more than three branches and in this case it is convenient to consider such a junction as being a number of junction points which happen to be coincident, e.g. a junction associated with 4 branches would be considered as 2 coincident junctions connected to one another by a branch of zero length. The manner in which a junction can be subdivided is not unique, as shown in figure 3-8. There are two features of a tree which must be varied in order to find an optimal solution. First the connectivity relationship between the junctions themselves and the nozzles (henceforth termed topology), and secondly the positions of the junctions in the network.

For a given topology, and an initial feasible set of junction positions, the solution can be improved iteratively by applying the three nozzle procedure to each junction. This is done by removing a junction and its 3 associated branches, and finding a new optimal position of the junction assuming the branches are to be connected to the same nodes in the remaining parts of the tree.

This clearly allows the junction positions to be varied for a given topology, but what about varying the topology? It was mentioned above that a node adjoining more than 3 branches cannot be uniquely decomposed into a number of 3 branch nodes, i.e. its topology is not uniquely

defined in this sense. The existence of such nodes in the tree enables changes to occur in the topology as follows.

Consider all the branches associated with a junction and of these consider any pair of terminals at the other end of a pair of branches. Then an improvement in cost may possibly be obtained by deleting these two branches and introducing an extra 3 branch junction node somewhere else in the network, thus reducing the order of the original junction by 1. The problem remains of deciding which pair of branches to choose in the first place. This could be arbitrary or, as in the case of the test program, a pair is chosen which leads to the greatest improvement in cost. This is done by evaluating all pairs. It will be noted that replacing any junction node by another junction node leaves the topology unchanged, thus one computation of the unit pipe costs for each of the 3 branches involved in the local optimisation is all that is needed.

The above procedure requires to be given an initial solution in order to commence the optimisation. This initial solution is generated in the test program by connecting all the nozzles to 1 internal junction node placed in the vicinity of the 'centre of gravity' of the nozzle positions.

In some cases, the topology of a branched pipe may be fixed before routing commences, in these cases, the routing procedure becomes a much simplified version of the procedure described above, comprising a simple iterative scheme through all the internal junction points of the pipe.

3.10 Manipulating network costs manually.

If 'l' is the effective length of a network element and 'c' is the cost of a unit length of pipe running along that element, then the total cost of the section of pipe is cxl. It is the sum of such terms that is minimised by the shortest route algorithm. Normally, the effective length will be the actual length of the element, but facilities have been implemented in the test program which allow the user to impose penalty functions on the network elements as follows.

The network is rectilinear, hence if one selects any pair of nodes in the network, then a box of space is uniquely defined whose extreme coordinates are those of the pair of defining points. Since, in general a box has 4 pairs of opposite corners, then the same box can be defined arbitrarily by any of these 4 pairs. This method of selecting a part of the network allows anything from a single element to the whole network to be selected.

A node is defined in the network by three integers giving the row, column and level of the node. Hence 6 integers define a box. Further one can select any combination of elements running in the x,y and z directions. If the x,y,z axes are numbered 1,2,4 then any digit between 1 and 7 will give the combination of axes to be affected. Having selected a subset of elements of the network in this manner, it is now possible to multiply the current effective lengths of all the selected elements by a penalty factor, e.g. the sequence of numbers 1 1 3 4 5 3 3 10 means define a box by the pair of points (1,1,3) and

(4,5,3) , select those elements running in the x and y directions (3 = 1+2) and use a penalty factor of 10%, this being a penalty factor which is meant to encourage pipes to run along the selected elements. A whole set of boxes can be selected in this way so that all favourable and unfavourable parts of the network can be designated before routing commences.

If the user is to influence the routing algorithm in this way, then it is necessary for him to ascertain from the program the network scheme generated and the numbering system used to define the nodes. It is thus only practical to use this system using a graphics display where a user can examine a picture of the network on the screen.

3.11 Manipulating network costs automatically.

In the absence of any means of interaction with the program, it is still possible to obtain some effects automatically. Thus the user may not specify any penalty functions, but he may wish the program to bunch pipes together as much as possible, because if pipes run along similar routes then they will presumably cost less to support.

An early version of the test program carried out this function by routing all the pipes using unit penalty functions. New penalty functions were then computed depending on the amount of pipe running along each element, the more pipe on an element, then the cheaper the penalty function. The pipes were then rerouted using these new penalty functions. Thus each pipe was routed twice in order to achieve this bunching effect. However, it was found that

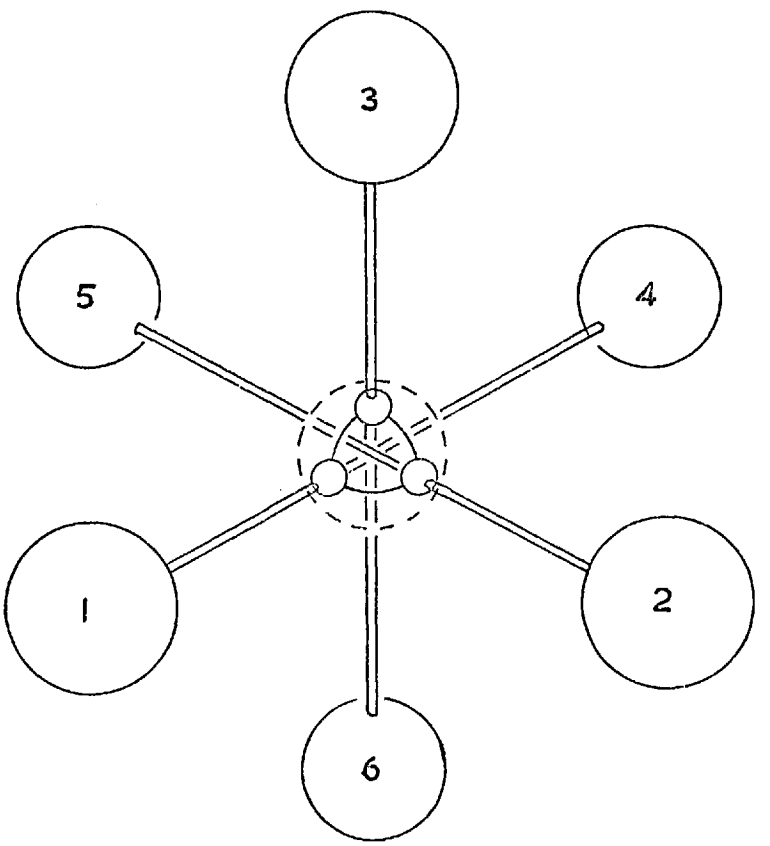


FIG. 3-9 : NETWORK STRUCTURE USED TO ACCOUNT FOR BENDS

very similar results could be obtained in one iteration by adjusting the penalty function along the route of each pipe, immediately after it is routed, such that the new penalty functions would effect all pipes subsequently routed.

To say the least, both methods are 'ad hoc' and it is thought of little practical use in a system where the user is allowed to specify his own penalty functions. Usually, the designer has a shrewd idea where he wants the pipe racks to go.

3.12 Joint Minimisation of Pipe Costs and Bends.

In all that has been said, the only contribution to pipe costs that has been considered is the length of pipe in the route, whereas in practice, bends would contribute a significant amount to the overall cost. This section is included to show that bends can be included in the overall cost function.

For a particular branch of a pipe being routed, if the size and cost of the straight pipe is known, then also the cost of an elbow is known. The routing algorithm previously described was such that each node could be considered to have 6 adjacent nodes. If the network is modified such that each node becomes 3 nodes all lying at the same point in space and also introduce 3 new network elements to represent the cost of an elbow, then routing pipes on this new network will include the cost of bends. Therefore one can go from node 1 to node 4 in figure 3-9, without going via an elbow element, but if one wishes to connect 1 to any of the remaining 4 elements then the cost

of a bend will be incurred. Thus the same shortest route methods can be used to include the costs of bends.

The big drawback of including bends in the cost function is the large increase in storage needed to cater for them, and since even without bends, the networks get very large, a further increase is not felt to be practicable.

3.13 Results Obtained from the Test Program.

Figures 3-10 to 3-17 show various routing schemes derived from the test program. In all cases, these should be self explanatory. In no case did it require more than three penalty volumes to be defined to obtain the desired effects. The pictures are trimetric projections, either based on a plan view or an elevation looking in the positive 'y' direction. Each pipe is shifted bodily in the picture by a small amount in order to avoid concurrent pipes being displayed on top of one another.

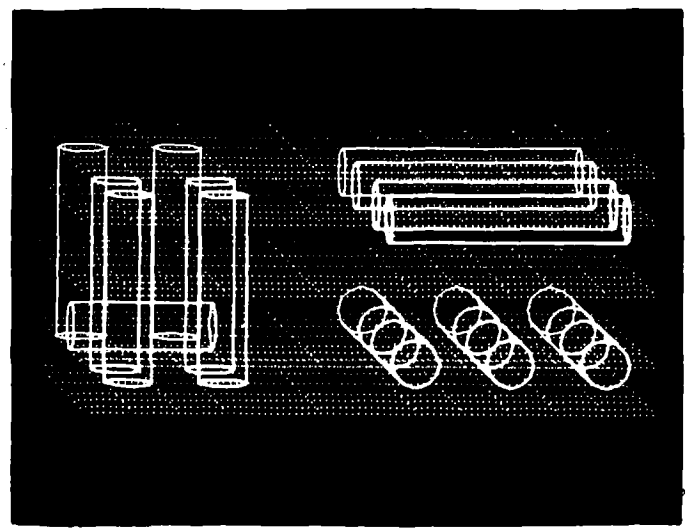


FIG. 3-10 : VESSEL LAYOUT WITH FIVE LAYERS OF THE ROUTING NETWORK

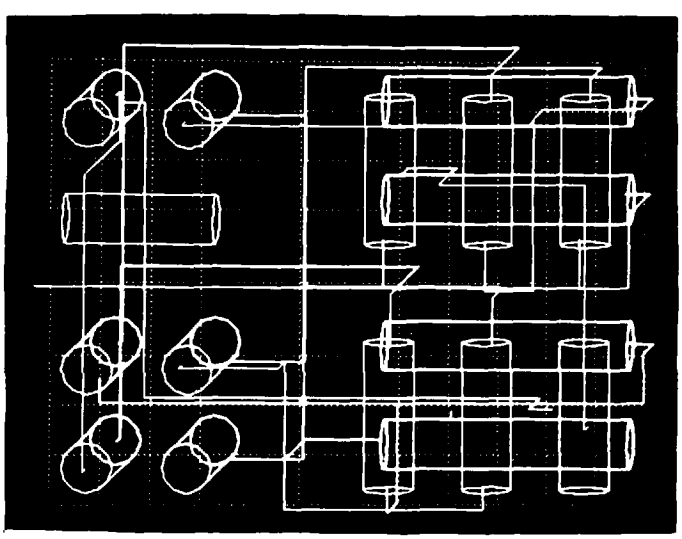


FIG. 3-11: PIPES ROUTED WITH NO PENALTY CONSTRAINTS

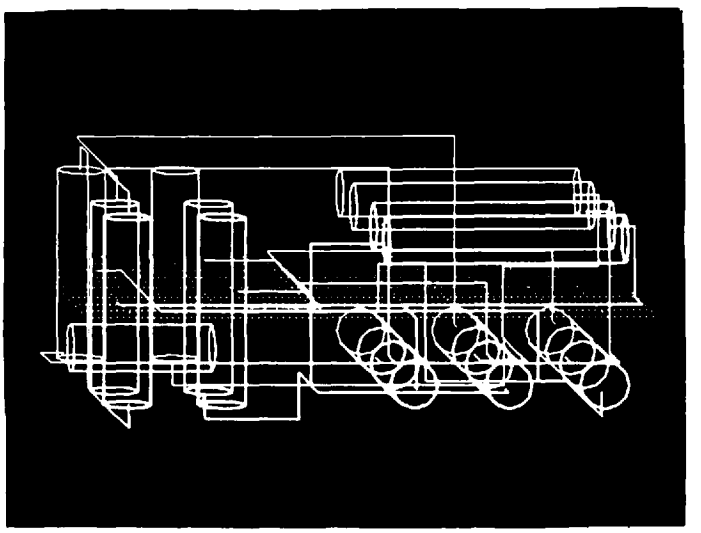


FIG. 3-12: 'ELEVATION' OF FIGURE 3-11.

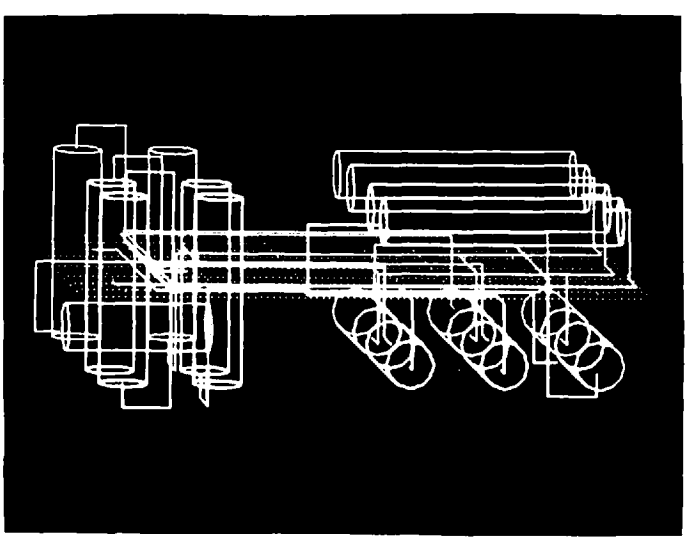


FIG. 3-13: PIPES ENCOURAGED TO RUN ON THE THIRD NETWORK LEVEL

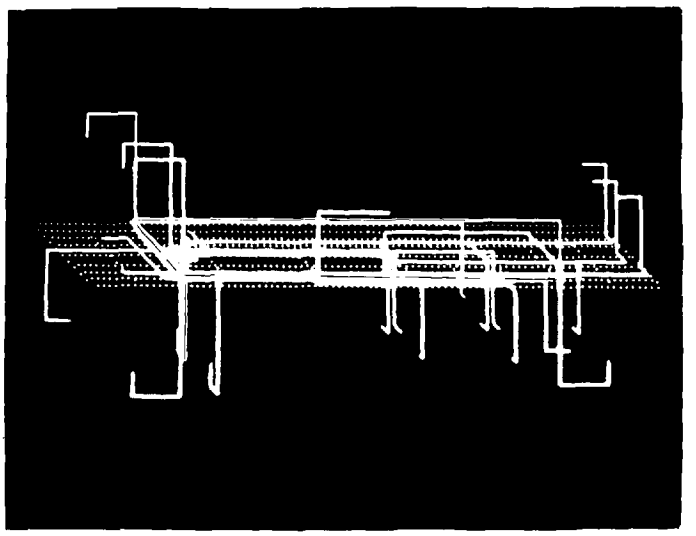


FIG. 3-14: AS 3-13 WITH VESSELS 'SWITCHED OFF'

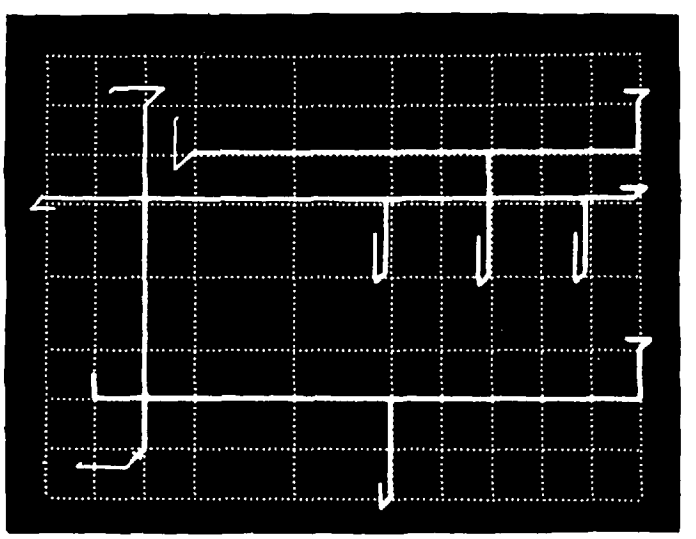


FIG. 3-15: EXAMPLE OF ROUTES PRODUCED FOR 2,3 AND 4 NOZZLED PIPES

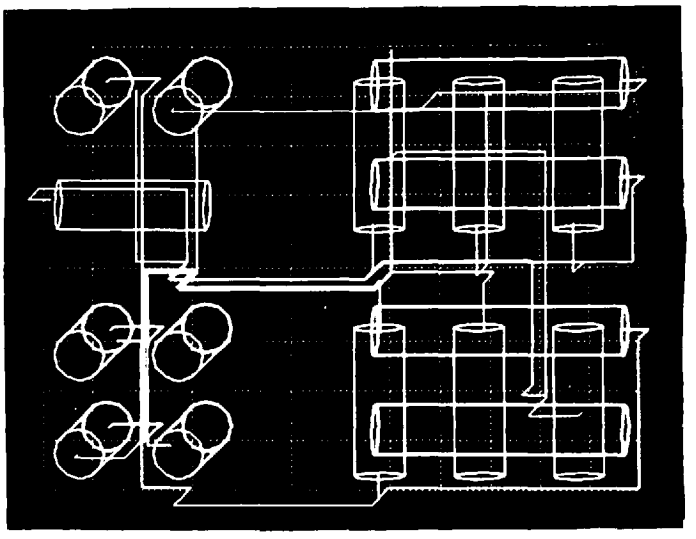


FIG. 3-16 : PIPES ROUTED TO AVOID WALKWAY BETWEEN VESSELS

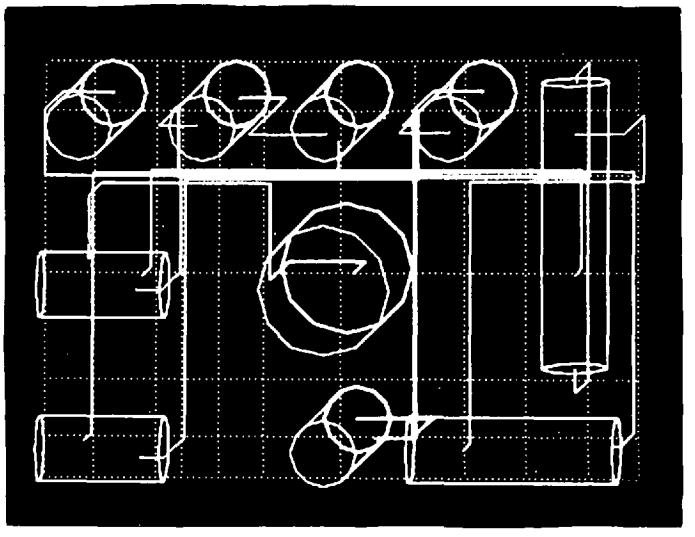


FIG. 3-17 : EXAMPLE OF PIPES ENCOURAGED TO RUN ALONG THE SAME PIPE TRACK

CHAPTER 4

3-D VISUALISATION OF A CHEMICAL PLANT - THE HIDDEN-LINE PROBLEM.

4.1 Introduction.

We have seen in chapter's 2 and 3 that, although algorithms can contribute to the design of good plant layouts, they are not sufficient to solve all problems in complete generality. Thus there is still a requirement for manual intervention in the design process. Wholly manual methods of design are sometimes centred around a model and it would be desirable to provide a similar facility using a computer-based design system. For effective communication to be made between designer and computer, there is a requirement for being able to visualise designs held by the computer.

There are various techniques for representing a 3 dimensional object on a 2-dimensional surface. These range from providing orthogonal views in the form of line drawings to a complete shaded picture representation with a large degree of realism. An object can be represented as a set of closed volumes, such as cylinders, truncated cones, half spheres and boxes. These basic solids can be represented by drawing lines to represent their edges, such a drawing is termed a 'wireframe' representation. 'Wireframe' drawings are cheap to compute and are ideally suited to a largely interactive form of design.

'Wireframe' drawings can be enhanced in a number of ways to provide depth clues to the 3D shape that they represent. Among these techniques are, brightness modulation with depth and stereo pairs. Both of these techniques are computationally cheap, the reason for their cheapness being that each edge in the picture can be

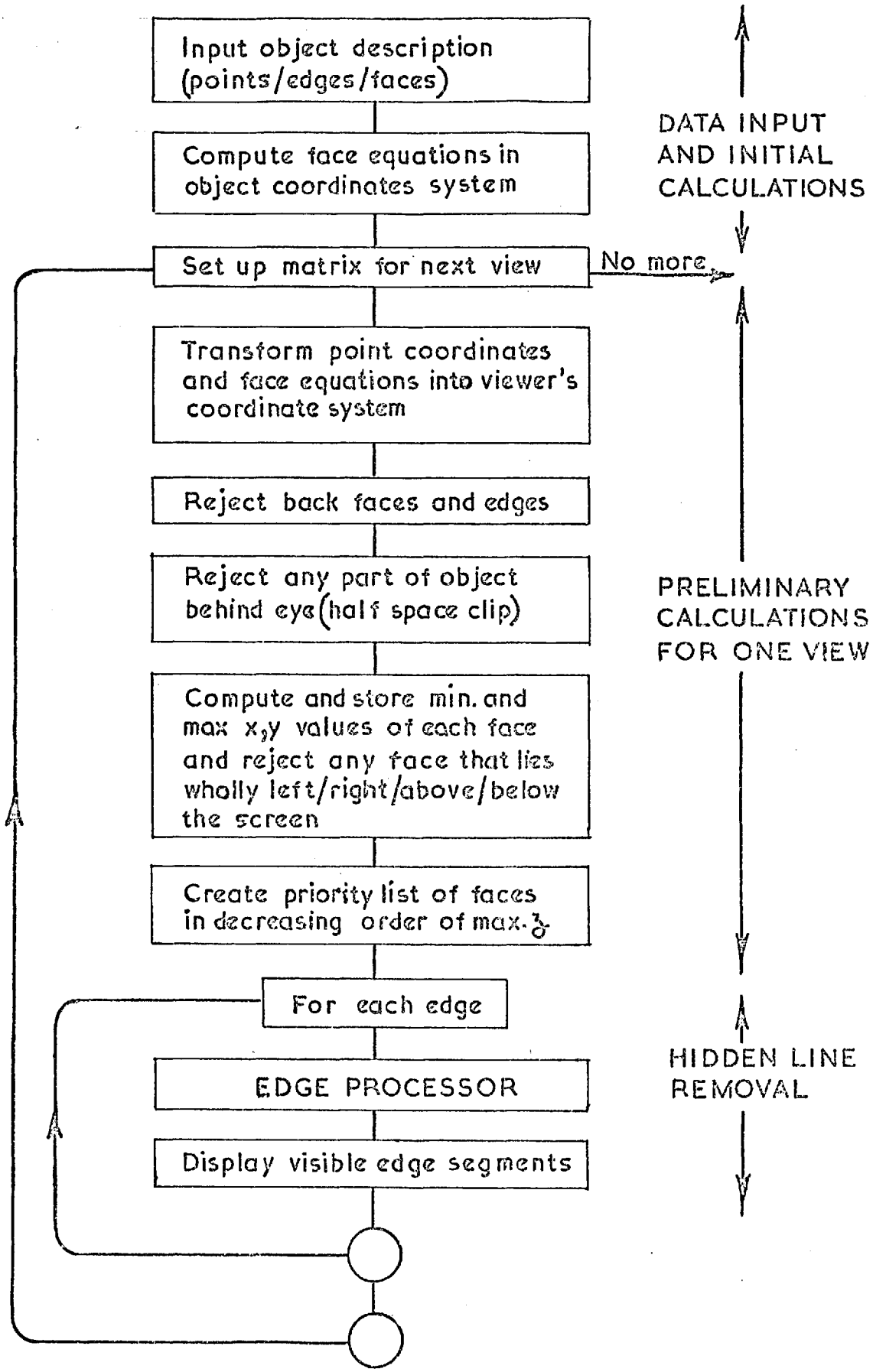


FIG. 4-1: OVERALL FLOW CHART OF HIDDEN-LINE PROGRAM

computed and displayed independently of any other part of the picture. The fundamental problem that remains in the display of wireframe representations is the removal of hidden lines. A number of authors have dealt with the hidden-line problem (13,14,15,16,17), but we here consider a method that is simple in its approach and it also has the advantage that it is quick in its execution in comparison with other methods.

4.2 The Hidden Line Problem.

The solution of the hidden-line problem requires a knowledge of the opaque parts of the object, as well as its vertices and edges. Here we will consider objects defined with straight line edges and planar faces. Curved surfaces can be incorporated by using an approximation consisting of a number of planar faces and straight edges.

The procedure used is to process each edge in the image, finding those faces that occlude all or part of the edge, and then to display the visible edge segments. Thus at first sight, since each edge may be tested against each face, the computation time could increase as the square of the number of objects in the scene. It is shown that using the method described here, the computation time increases a little more than linearly with the number of faces or edges in the scene. The crux of the method is a short sequence of exceedingly simple tests that allow much computation to be avoided. In addition, a number of features are incorporated in the algorithm that are considerably simpler and more economic than other methods.

The flow chart in figure 4-1 shows all stages

necessary for computing a sequence of hidden-line pictures of a scene. The flow chart consists of three main parts; first, the input stage; second, the processing required to set up a particular view; and third, the removal of hidden lines for all edges in the scene. The functions of all parts of this flowchart are described in the following sections.

4.3 Definition of an Input Object.

An object to be processed comprises a list of points, edges and faces. Each point consists of its name and coordinates. Edges are defined in terms of two named end points and each face is defined in terms of its closed boundaries, each boundary being defined as an ordered sequence of named points. (For example, a square face with two square holes in it would comprise three boundaries of four points each).

The internal representation of the object consists of linear arrays. The coordinates of each point are held in one array and each edge is held as a pair of pointers to its end points. However, although faces are defined by the user in terms of boundary points, the internal representation of each face is a list of boundary edges. The reason for this difference is that it is easier for a user to define boundary points, but it is easier computationally if the boundary edges are held explicitly. User defined edges can be defined which do not belong to the boundary of a face. This can be useful for defining pipe centre lines.

4.4 Calculation of Face Equations.

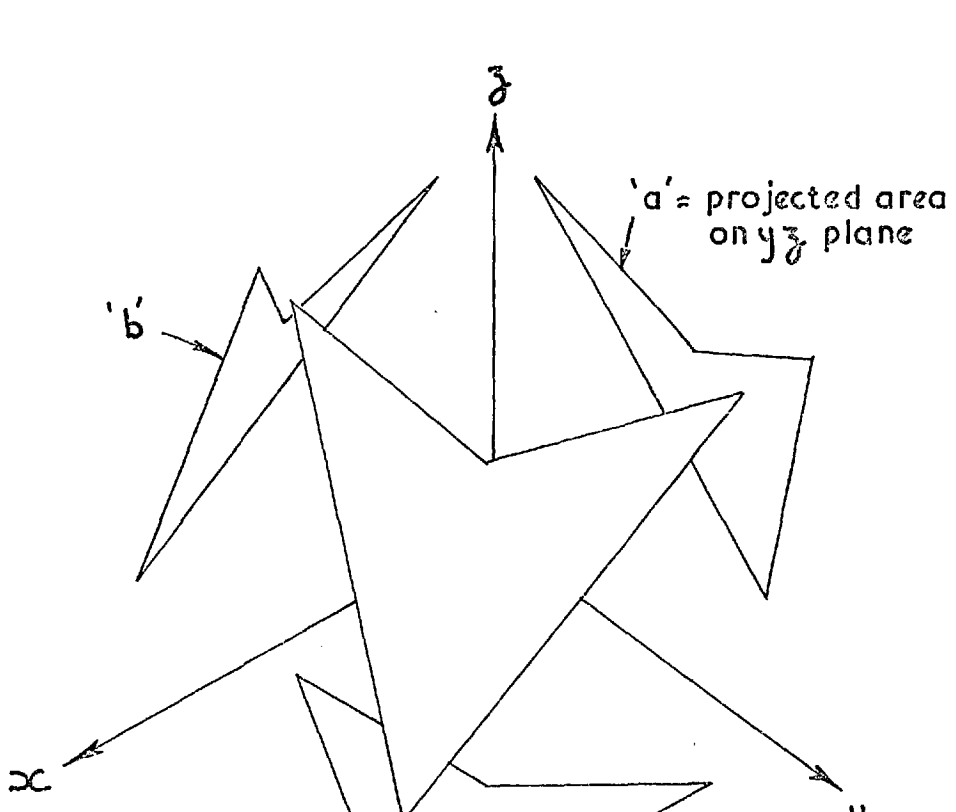


FIG. 4–2: CALCULATION OF FACE EQUATIONS

As will be seen later, face equations are needed for a number of reasons in the solution of the hidden-line problem, and hence a simple efficient method of computing face equations is required. It is generally assumed that three points are sufficient to define a plane, and one could take any three points on the face boundary, however, there is a risk that the three points chosen might be collinear or even coincident, in which case special tests are required to find three non-collinear points.

A face equation can be written

$$ax + by + cz + d = 0$$

the coefficients a,b,c are in proportion to the direction cosines of the face normal. It will be seen later that it is sometimes required to know the direction of the outward normal of a face. The user uses a convention to indicate the direction of the outward normal by defining the face boundary points in a clockwise manner about it (anti-clockwise for holes). It is important when computing a face equation that the coefficients a,b,c have a sign corresponding to the outward normal, and it would be convenient if this could be achieved without the need for special tests, which might arise with faces with a non-convex boundary.

The following method of computing face equations solves this problem and is very compact. The coefficients a,b,c are obtained by computing the area of the face projected onto the yz,zx,xy coordinate planes respectively, as illustrated in figure 4-2. If the face boundary consists of

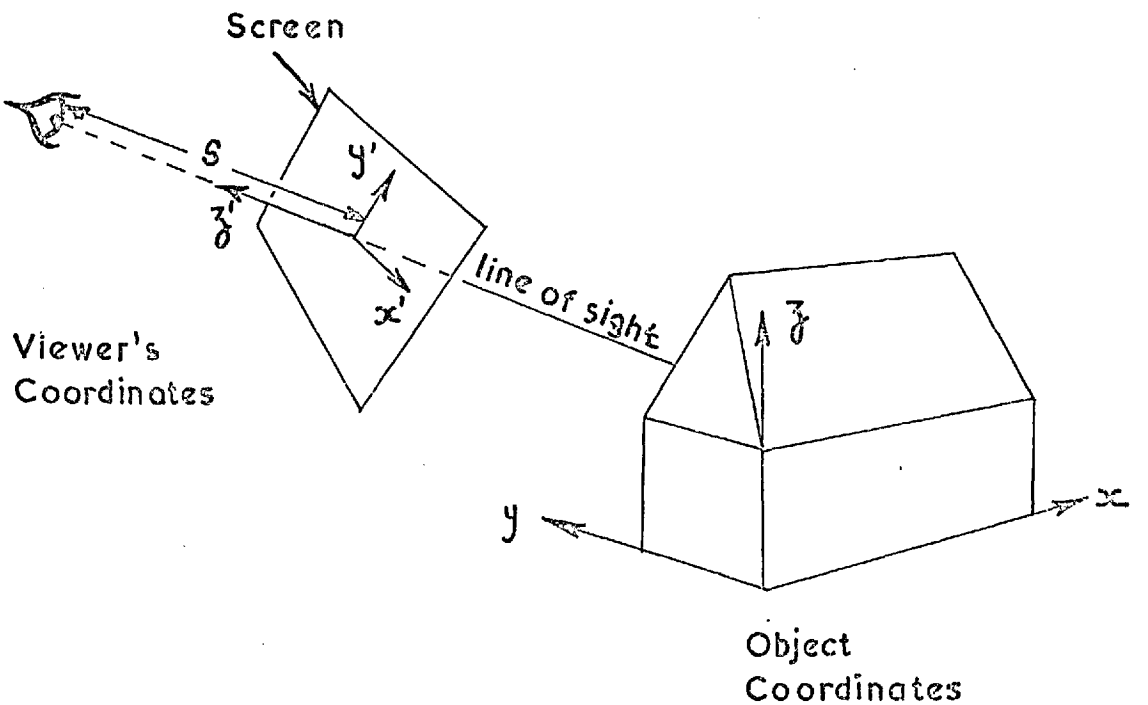


FIG. 4-3: COORDINATE SYSTEMS OF OBJECT AND VIEWER

'n' points and the i'th point has coordinates x y z , then using the trapezoidal rule for finding areas we obtain

$$a = \sum_{i=1}^{n} (y_j + y_i)(z_j - z_i)/2 \qquad \begin{matrix} j = i+1, & i<n \\ j = 1, & i=n \end{matrix}$$

$$b = \sum_{i=1}^{n} (z_j + z_i)(x_j - x_i)/2$$

$$c = \sum_{i=1}^{n} (x_j + x_i)(y_j - y_i)/2$$

and $d = -(ax_m + by_m + cz_m)$

The point $x_m\ y_m\ z_m$ is any point on the plane, this could be the first boundary point. In practice it is computed to be the mean of all the vertices of the face, this gives a mean value of 'd' in cases where the polygon is slightly twisted, as can arise in the case of approximated curved surfaces. A further point is that the factor of 2 can be omitted from the calculation.

4.5 Object Transformation.

The process of computing a projected image of an object onto a screen involves transforming the object from its original coordinate system onto the 2-dimensional coordinate system of the screen. The central part of this hidden-line algorithm assumes that the object to be drawn is viewed from a distance 's' along the z axis with the screen in the x,y plane. However, in general an object is defined in its own coordinate system and so arbitrary views must be handled by first transforming the object into the viewer's coordinate system as in figure 4-3. This transformation involves all point coordinates and all plane equations.

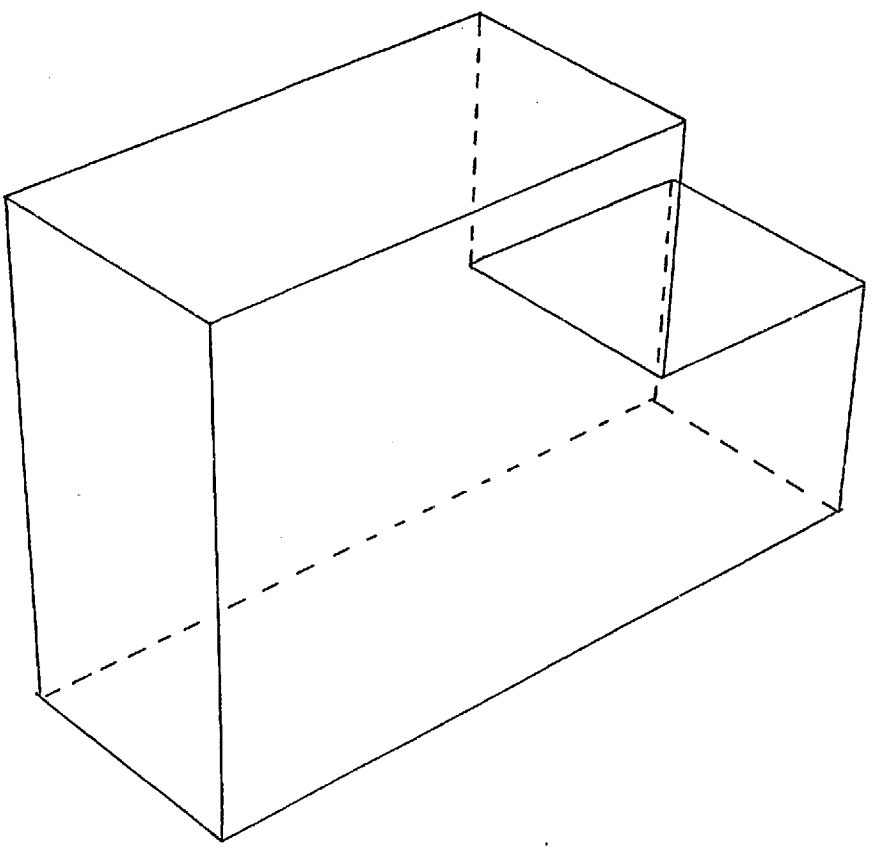


FIG. 4-4: REJECTION OF BACK FACES AND EDGES
(back edges are shown dotted)

Using the homogeneous notation for a point in 3-space (i.e. the point (x,y,z) is considered to be the point (x,y,z,1) in 4-space), the initial transformation of points into viewer's space can be described as

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

this transformation is an affine transformation, being composed of arbitrary rotations, scales, shifts and shears. Using a vector notation, the transformation of points can be written

$$p' = Ap$$

[A] is the 4x4 matrix used to transform points, but we also require to transform face equations into the viewer's coordinate system. Of course, this could be achieved by computing projected areas on the viewer's coordinate planes as in section 4.4. However it is more efficent to transform the face equations as follows. A face equation can be written as a product of two 4-vectors:

$$(a,b,c,d) \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

or $\vec{u}^T p = 0$

the transformed vector of coefficients, u', is defined by

$$u' = k(A^{-1})^T u$$

where k is a scalar chosen so as to normalise u'

proof: $(u')^T . p' = k((A^{-1})^T . u)^T . (Ap)$

$= k(u^T A^{-1}) . (Ap)$

$= ku^T (A^{-1}A) p$

$= k(u^T . p) = 0$

the matrix of cofactors (adjoint matrix) can be used instead of (A^{-1}), and this also handles the case where A is singular.

It should be noted that in the process of transforming points and face equations, no information is lost. Hence there is no need to hold both the object coordinate values and the viewer's coordinate values since this is merely duplicating information. Thus after being transformed, each point and face equation can be stored in their original locations in store. However, in cases where a sequence of views is required, then the actual matrix used to generate each successive view needs modification as follows:

Let A_n and A_{n+1} be the matrices needed to transform points from the object coordinate system into the viewer's coordinate system for views n and n+1. Then the matrix actually used to transform points from the viewer's coordinates of view 'n' into the viewer's coordinates of view 'n+1' is $B = A_{n+1}A_n^{-1}$ and $(B^{-1})^T$ is used to transform face equations.

Screen coordinates can simply be obtained from viewer's coordinates as follows:

$$x'' = x'/(1-fz')$$

$$y'' = y'/(1-fz')$$

'f' is known as the perspective factor and it equals 1/S. If a non-perspective projection is required, then 'f' is effectively zero and hence

$$x'' = x'$$

$$y'' = y'$$

this being a parallel projection of the object in viewer space onto the screen.

In either case, the computation needed to determine screen coordinates is cheap and hence there is no need to store screen values, since they can be computed when needed.

4.6 Rejection of Back Faces and Edges.

In many cases, a scene consists of a set of closed polyhedra, however it is clearly not possible to see all the faces of a closed polyhedron from a point outside that polyhedron. If we can detect all those faces in the scene that face away from the eye, then the object complexity can be significantly reduced. Figure 4-4 shows the back faces and edges of a simple non-convex polyhedron. This is where we can use the outward normal already mentioned. The equation of a plane in viewer's space is

$$a'x' + b'y' + c'z' + d' = 0$$

the face is a back face, and hence can be rejected if

$$d' + fd' < 0$$

if $f = 0$ (i.e. no perspective) then this test becomes even

more trivial.

The edges of a polyhedron can be classified into one of 3 types; those belonging to two back faces, those belonging to one back face and one front face and those belonging to two front faces. If an edge belongs to one or two front faces, then it is not necessarily obscured by its own polyhedron, but back edges, i.e. those edges belonging to two back faces will never be visible and hence can be rejected. Back edges are shown dotted in figure 4-4. In addition, a re-entrant edge belonging to one front face and one back face can also be rejected, however this is not so easy to detect because the faces belonging to each edge are not explicitly held by the program.

The method used to detect back edges is as follows. A temporary flag is set to zero for each edge. The program loops through all faces, and if the face is a front face then all boundary edges of the face have their flags set to 1. At the end of the process, all back edges will have zero flags.

4.7 The Half Space Clip.

A fundamental problem arises when computing perspective projections since a singularity occurs at the plane through the eye. Although, the projections of parts of the object behind the eye are mathematically defined, these must not be displayed. Simple ways around this problem are to disallow parts of the object behind the eye or alternatively to reject any edge or face completely if any part of it lies behind the eye. The latter solution is adequate in many cases. In the case of 'wireframe'

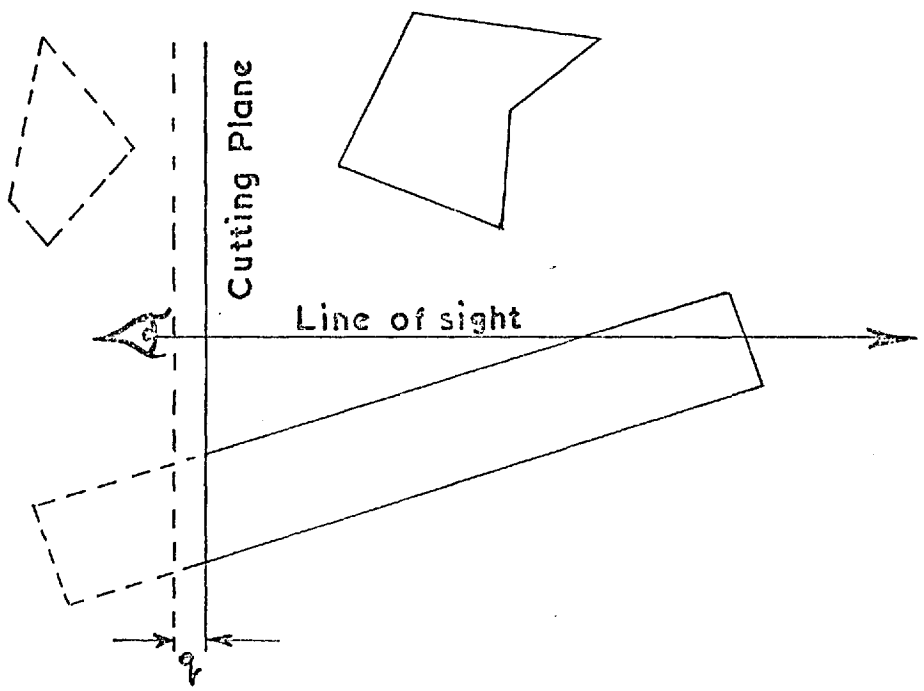


FIG. 4-5: ILLUSTRATION OF THE HALF-SPACE CLIP

drawings, a 'pyramid' clip can be used. All edges in the scene are clipped to a pyramid whose apex is at the eye. pyramid clipping has the advantage of clipping the image to the rectangular screen area. However, when solving the hidden-line problem, both edges and faces must be clipped, but a pyramid clip applied to faces is rather complicated and for this reason, we here adopt a half-space clip to remove those parts of the object behind the eye. The half space clip is illustrated in figure 4-5. It is accomplished by placing a cutting plane just in front of the eye at z = S-q where q is a small positive value. Any edge or face that lies wholly infront of the cutting plane is accepted, but any face or edge that lies wholly behind the cutting plane is rejected. Any edge or face that intersects the cutting plane is clipped in order to remove the portion behind the eye. Clipping is accomplished by simple linear interpolation and the resulting clipped face and its clipped edges are added to the object definition.

4.8 Further Rejection Tests and Preliminminary Calculations.

Before entering the edge processor, described later, the minimum and maximum values of x and y are calculated and stored for all faces in the scene that have not been rejected . When this is done, a further quick rejection test may lead to savings in computation. If a face is wholly to the left/right or wholly above/below the viewing area then it need not be considered. This rejection can be accomplished by means of a small number of simple inequality tests using the stored minima and maxima. These same stored values will also be used for quick gross overlap tests between edges and faces in the edge processor

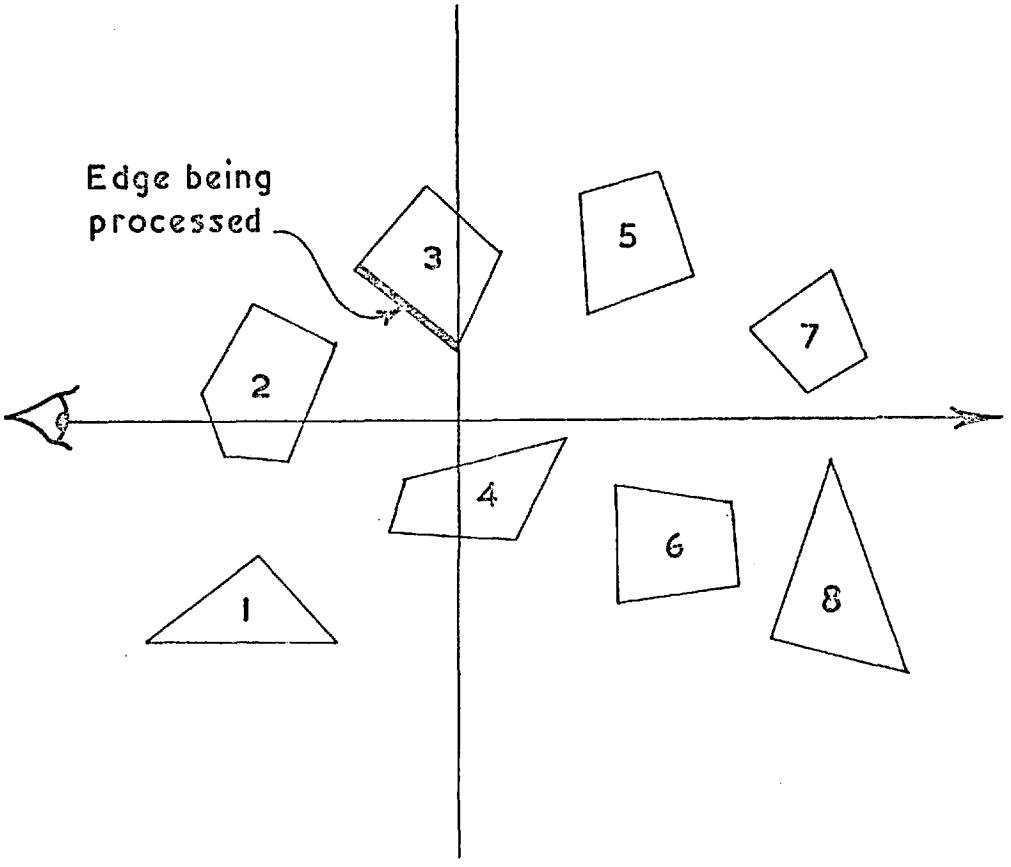


FIG. 4-6: PRIORITY ORDERING OF FACES IN EDGE PROCESSOR

below.

The final calculation needed before entering the edge processor is to compute and store the maximum z values of all faces and a priority list is initialised to contain references to all faces in order of decreasing maximum z values. The order of this list is shown in figure 4-6. The maximum z value of a face corresponds to the vertex nearest to the cutting plane through the eye. Having completed all preliminary rejection tests and calculations, the program takes each edge in turn, handing it to the edge processor for determination of the visible portions of the edge.

4.9 The Edge Processor.

The edge processor receives an object definition that in many cases is reduced and simplified by the procedures already described. The resulting object definition has the following characteristics:

(a) all faces face towards the eye

(b) all faces almost certainly lie within the viewing area

(c) all faces lie wholly in front of the eye

(d) no back edge is included

(e) all edges lie wholly infront of the eye

The function of the edge processor is to generate from each edge, a list of visible edge segments that lie wholly inside the viewing area. This involves clipping each line to the screen bounds as well as solving the hidden-line problem. It is inevitable that, at some stage in computing the visible line segments in an image, intersections with face boundary edges must be computed. Attempts to increase the efficiency of such hidden-line

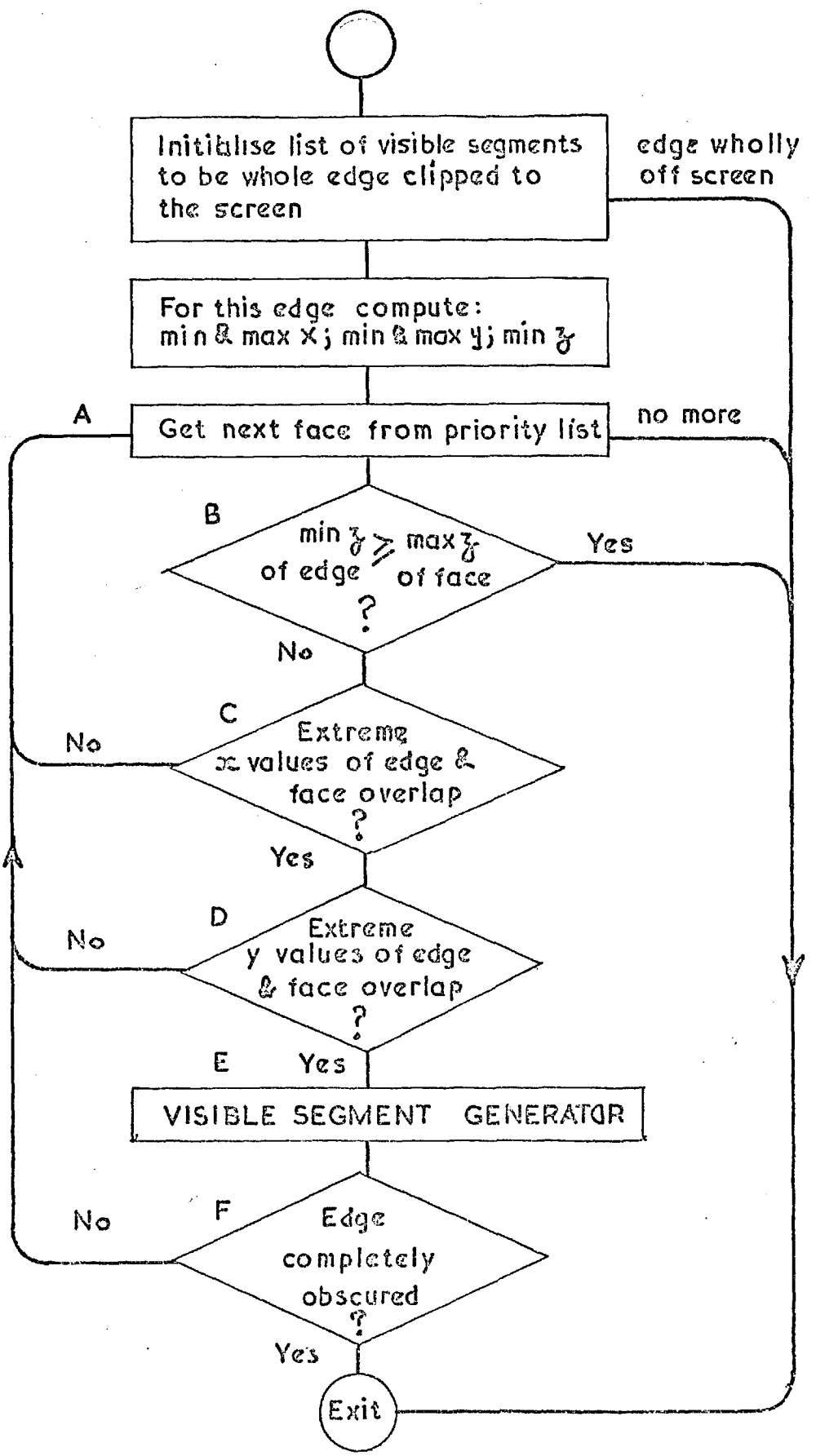


FIG. 4–7: FLOW CHART OF EDGE PROCESSOR

methods are usually based on reducing the number of computations that must be made. As mentioned before, if all edges are tested against all faces for visibility, then the computation time increases as the square of the number of items in the scene.

The flow chart in figure 4-7 gives a broad outline of the method. This flowchart describes the sequence of quick gross tests that are made in order to avoid computing edge intersections, unless they are really necessary. The edge to be processed is clipped by finding its intersections with the screen boundary and the list of visible edge segments is initialised to be the result of this clip. The list of visible edge segments is modified by each face in turn until one of the following three conditions is satified;

(a) the edge becomes wholly obscured
(b) a face is found that lies wholly beyond the edge
(c) the list of faces becomes exhausted

the resulting edge segments, if any, are then displayed.

A detailed analysis of the performance of the edge processor appears later, but we will here outline the functions of the various stages. If 'e' is the number of edges and 'f' the number of faces remaining in the scene after the general rejection tests described above, then consider each stage in turn as follows;

A: this box controls the main loop through all the faces and is executed a number of times, less than or equal to

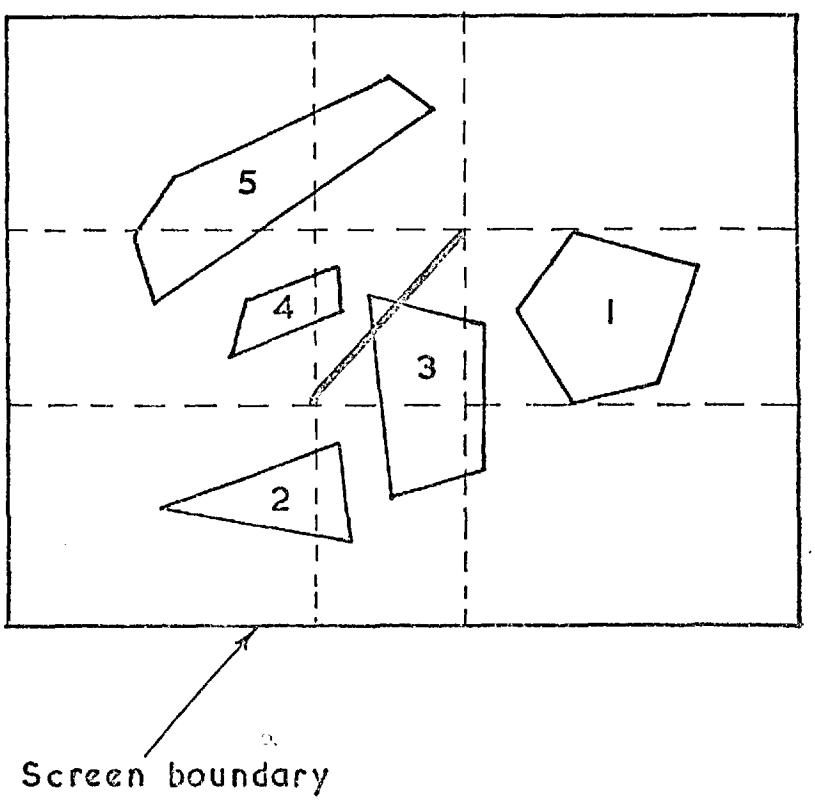


FIG. 4-8: 'X' AND 'Y' OVERLAP TESTS

e,f, depending on the consequences of boxes B and F.

B; the purpose of B is to restrict the computation to all faces whose nearest point (maximum z) lies beyond the furthest point (minimum z) of the edge being considered. Since the faces are considered in order of decreasing maximum z value, the procedure terminates for the first face found satisfying the test. B is executed the same number of times as A. For example in figure 4-6, only faces 1,2,3 and 4 are considered since 5 is the first face found that lies wholly beyond the edge.

C,D; these two tests further reduce the number of entries to E. These overlap tests are illustrated in figure 4-8. The face denoted '1' is immediately skipped by the x-overlap test C; face 2 is skipped by the y-overlap test D; but faces 3,4 and 5 are processed by E.

E; the major complication of the method is contained within E. As already seen, the routine is executed a number of times far fewer than the maximum e,f times because of the filtering effect of B,C,D. section 4.10 Describes E in much greater detail.

F; clearly, if the edge is totally obscured after processing by E then there is no point in considering it further.

Section 4.11 Below gives a much fuller analysis of the times taken in each stage of the procedure.

4.10 The Visible Segment Generator.

The visible segment generator is stage E of the edge

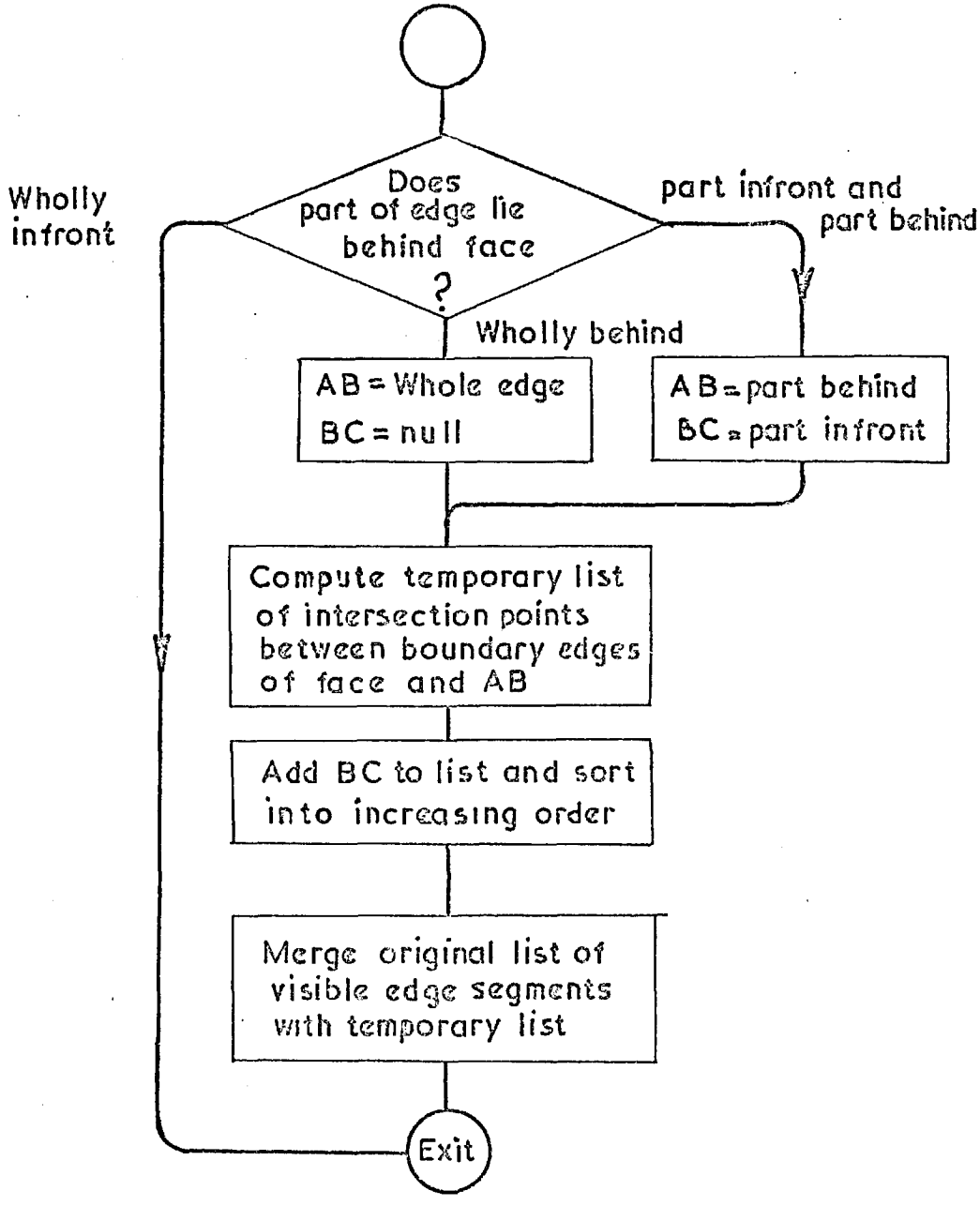


FIG. 4-9 : FLOW CHART OF VISIBLE SEGMENT GENERATOR

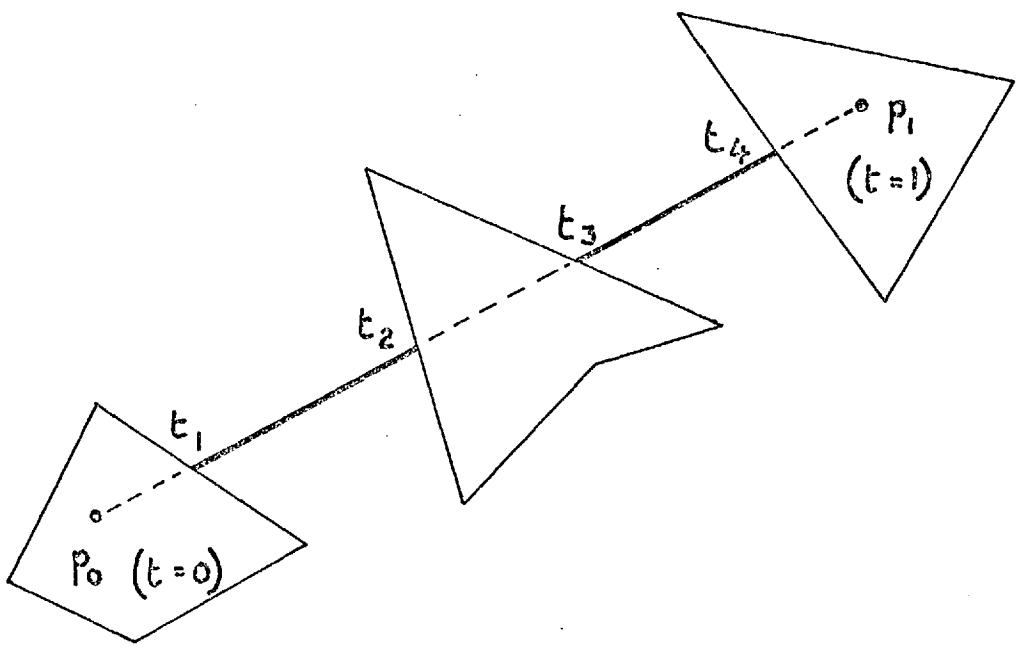


FIG. 4-10: PARAMETRIC REPRESENTATION OF VISIBLE EDGE SEGMENTS

processor. It is here that hidden-line removal actually takes place. The flow chart in figure 4-9 outlines the functions of the visible segment generator.

For computational purposes, an edge is represented as a parametric equation. If p_0 and p_1 are the two end points of the line, then any point on the line is defined as

$$p = (1 - t)\, p_0 + t p_1$$

as the parameter t varies from 0 to 1, p moves from p_0 to p_1. A list of visible edge segments is held as an even number of parameter values,

$$t_1, t_2, \ldots, t_n$$

visible segments being defined between alternate pairs of parameter values (see figure 4-10)

$$(t_1, t_2), (t_3, t_4), \ldots, (t_{n-1}, t_n) \qquad 0 \leq t_1 < t_2 < \ldots < t_n \leq 1$$

initially, the list of visible edge segments contains 2 parameter values

$$(t_1, t_2) = (0, 1)$$

The function of the visible segment generator is to compute the effect of each face in turn on the list of visible segments. The procedure involves some tests in three dimensions in viewer space followed by some two dimensional intersection tests in the plane of the screen. The result of these tests is a temporary list of segments not obscured by the face. This temporary list is merged with the original list to form the new list of visible edge segments ready for handing on to the next face.

The procedure starts by checking that at least part of the edge being processed lies further from the eye than the plane of the face. Using the earlier notation, u' represents the coefficients of the plane equation in viewer

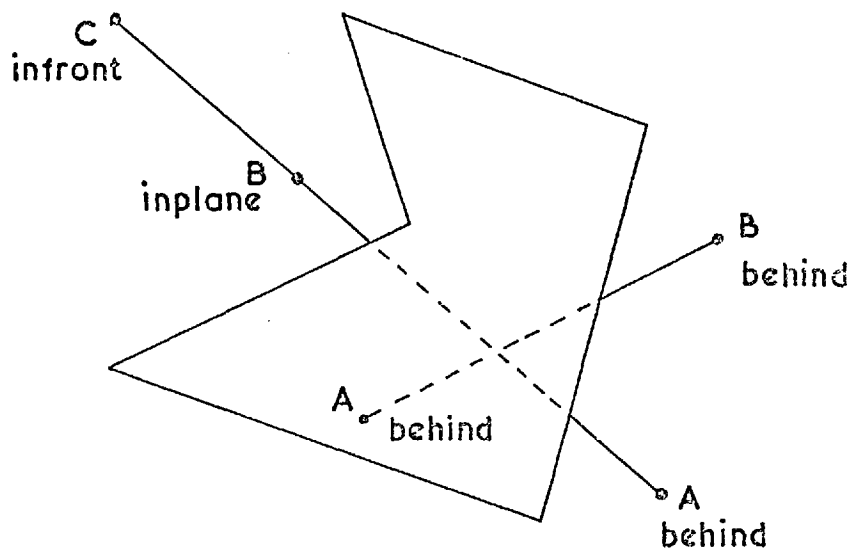


FIG. 4-11: NOTATION USED IN DESCRIPTION OF VISIBLE SEGMENT GENERATOR

space and p' is the homogeneous representation of a point, then

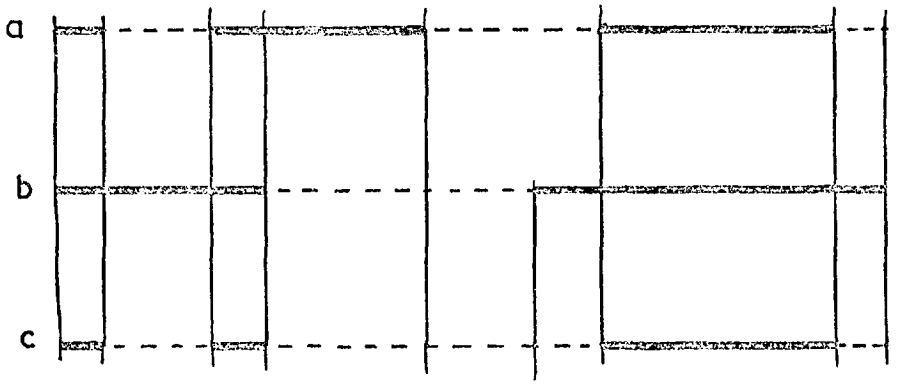
$u'.p' = 0$ means the point lies in the plane of the face

$u'.p' > 0$ means the point lies on the viewer's side of the face

$u'.p' < 0$ means the point lies beyond the face

Should both end points be in front of the face, then no further processing is required, since the edge must be wholly infront of the face, and hence cannot be obscured by it. If both end points lie behind the face, then it is permitted to proceed immediately with the 2D intersection tests. However, in cases where one end lies in front and one beyond, then it is necessary to clip the edge at B where it intersects the plane of the face and then to proceed with the 2D tests for the portion, AB behind the face as in figure 4-11.

The 2D tests deal with finding the intersections between an edge and a face boundary, both of which are projected onto the screen. It is known that the portion AB of the edge considered lies wholly beyond the face. An intersection point is only admitted if it lies between the end points of the boundary edge, although it may be outside the range AB for the edge being processed. This results in an unordered list of parameter values, each corresponding to an intersection of the edge (produced if necessary) with the boundary of the face. At this stage, the list is sorted and intersections beyond the range AB of the edge are removed; this may involve inserting parameter values for the end points A,B, in order to maintain visible portions between alternate pairs of values. Also, 2 values corresponding to the end points of the edge segment 'BC'



a : Original list of visible edge segments

b : Temporary " " " " "

c : Merged " " " " "

FIG. 4-12 : MERGE PROCEDURE USED IN VISIBLE SEGMENT GENERATOR

infront of the face should be added to the list.

This temporary list of visible edge segments corresponds to the solution of the hidden-line problem if the scene contains just this one face and one edge. The next stage in the procedure is to find the sections of edge that are visible in both the temporary list and the original list. This can be considered as finding the logical intersection of the two sets of visible line segments. It is accomplished by a simple merging procedure, the result of which could be an empty list. This is illustrated in figure 4-12. If the list is not empty, then the list of visible edge segments is handed over to the next face for further processing.

4.11 Performance of the Method.

At first sight, this hidden line method might appear similar to others which take each edge and exhaustively test it against all occluding faces in the scene. However, the sequence of simple rejection tests in the edge processor turn the algorithm from an inefficient proposition into an algorithm that can handle complex scenes with a relatively small amount of computation, and this is accomplished without an excessive increase in storage requirements. Also, the facility for handling a scene that is partly behind the eye is essential in such applications as viewing a chemical plant.

We are primarily concerned, in this section, with the storage requirements of the program and the speed of the algorithm. Firstly the storage requirements. These consist of the data associated with each point, edge and face as

follows:

(a) points - 3 real words are used for each point to hold the x,y and z coordinates.

(b) edges - 2 integer words are held to record the end points and 1 integer word is held for the flags used in the rejection of back edges

(c) faces - 4 real words for the plane equations

4 integer words for minimum and maximum x and y

1 integer word for the ordered list

1 real word for the maximum z value

1 integer word for a pointer into the list of boundary edges

(d) face boundaries - each face is held as a list of edges and an average of 4 edges per face is allowed by the program, requiring 4 integer words. However, all polygons do not have to comprise exactly 4 edges.

Therefore the number of words required are:

(a) reals - 3p+5f

(b) integers - 3e+10f

The total program logic, including input/output routines and graphics routines is about 10000 words on ATLAS II.

The time taken by the algorithm is clearly a function of the scene complexity, and broadly speaking it is

possible to quantify complexity as follows. We here refer back to figure 4-7 to try and examine the effect of scene parameters on the consequences of tests B,C and D.

The important paraneter to consider is the ratio of the average 'width/height/depth' of the edges and faces to the 'width/height/depth' of the scene. (Width/height/depth corresponds with x,y,z). It is assumed that this ratio is in fact the same in the three directions and it is also assumed that edges and faces are roughly the same size. For all the test objects considered below, these assumptions are correct within one order of magnitude.

Let us consider the number of times each of the tests B,C and D are executed for all edges considered. If 'r' is the depth ratio mentioned above, and as previously 'e' and 'f' are the number of edges and faces considered then:

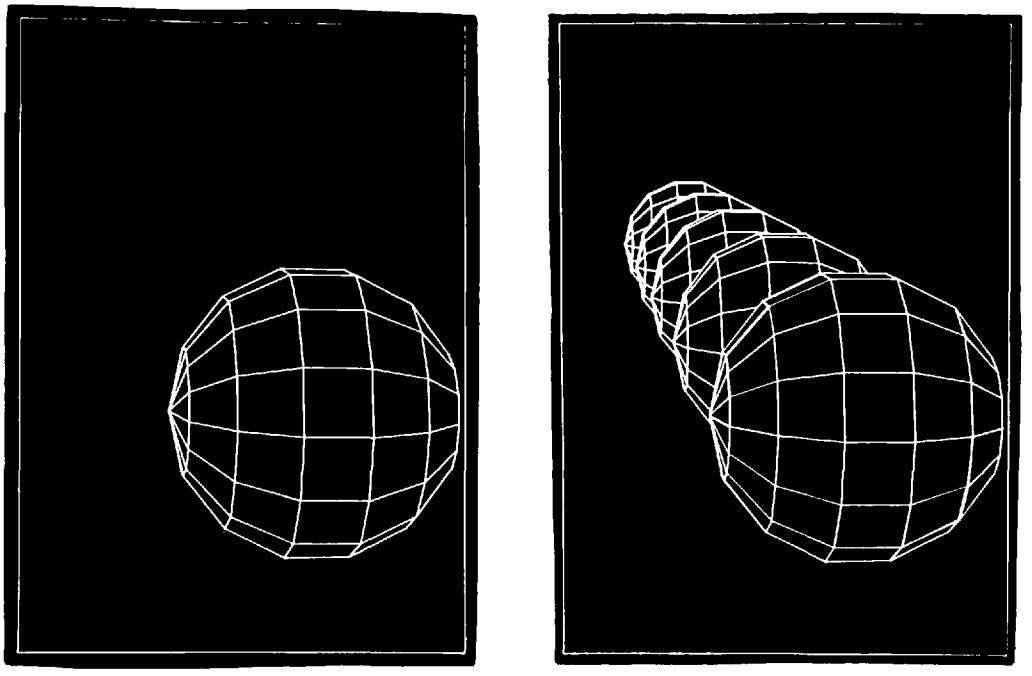
no. of times B fails (i.e. no. of entries to B and C) $e.f/(2-r)$

no. of times C fails (i.e. no. of entries to D) $e.f.r/(2-r)$

no. of times D fails (i.e. no. of entries to E) $e.f.r^2/(2-r)$

In the test objects, r lies somewhere between 0,06 in the case of the small boxes to 0,2 for the long boxes. Thus the square law term e.f only applies to B and C, both of which invoke trivial computation.

Table 4/1 gives a breakdown of a series of test objects together with the time taken for each object. Most of the test objects have been designed so that they are easy to regenerate by anyone wishing to compare their own



Case 1 Case 5

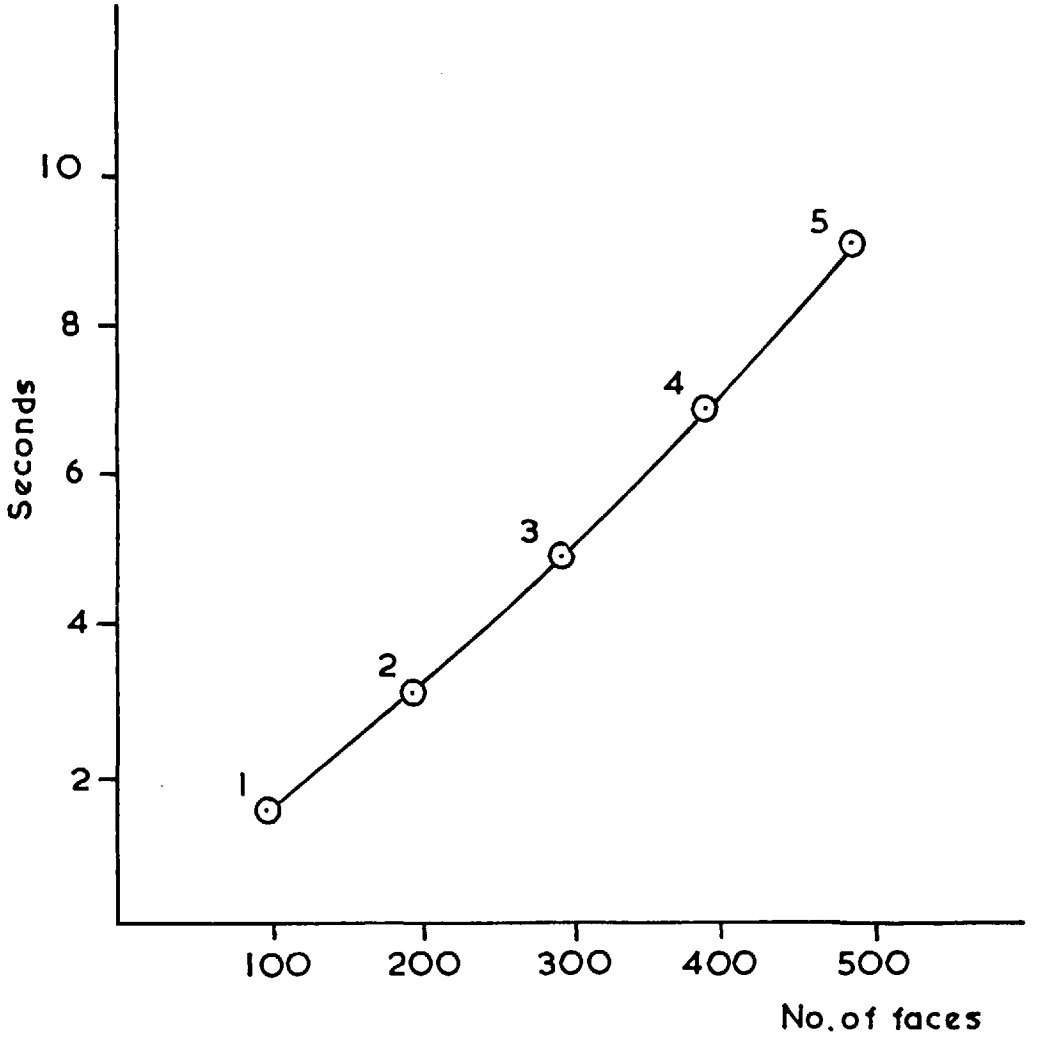
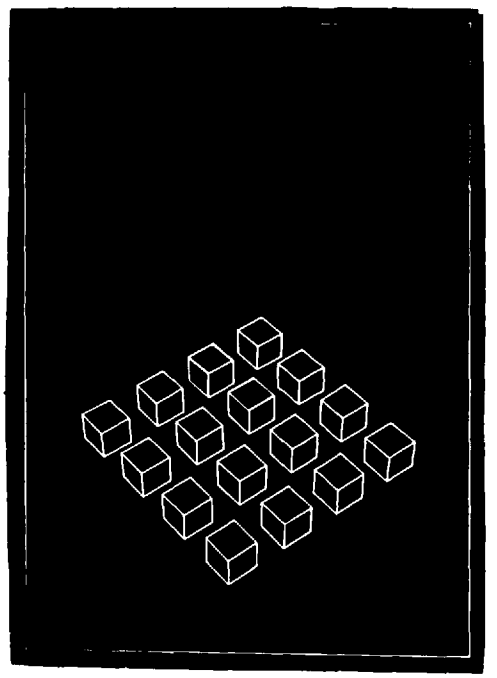
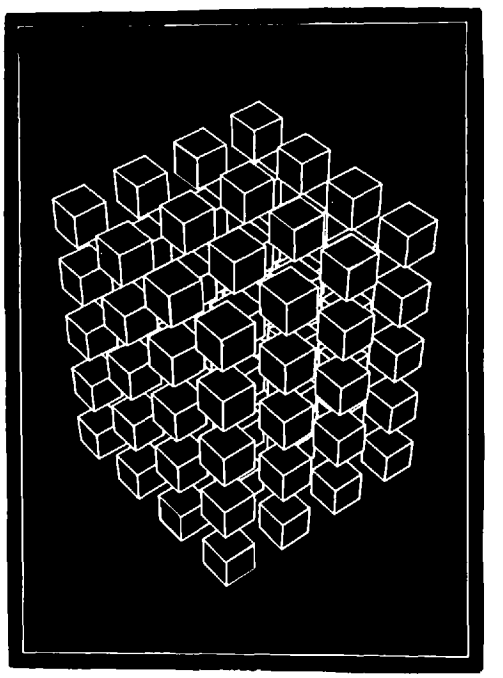


FIG. 4-13: TEST RESULTS OF GROUP 1



Case 6



Case 10

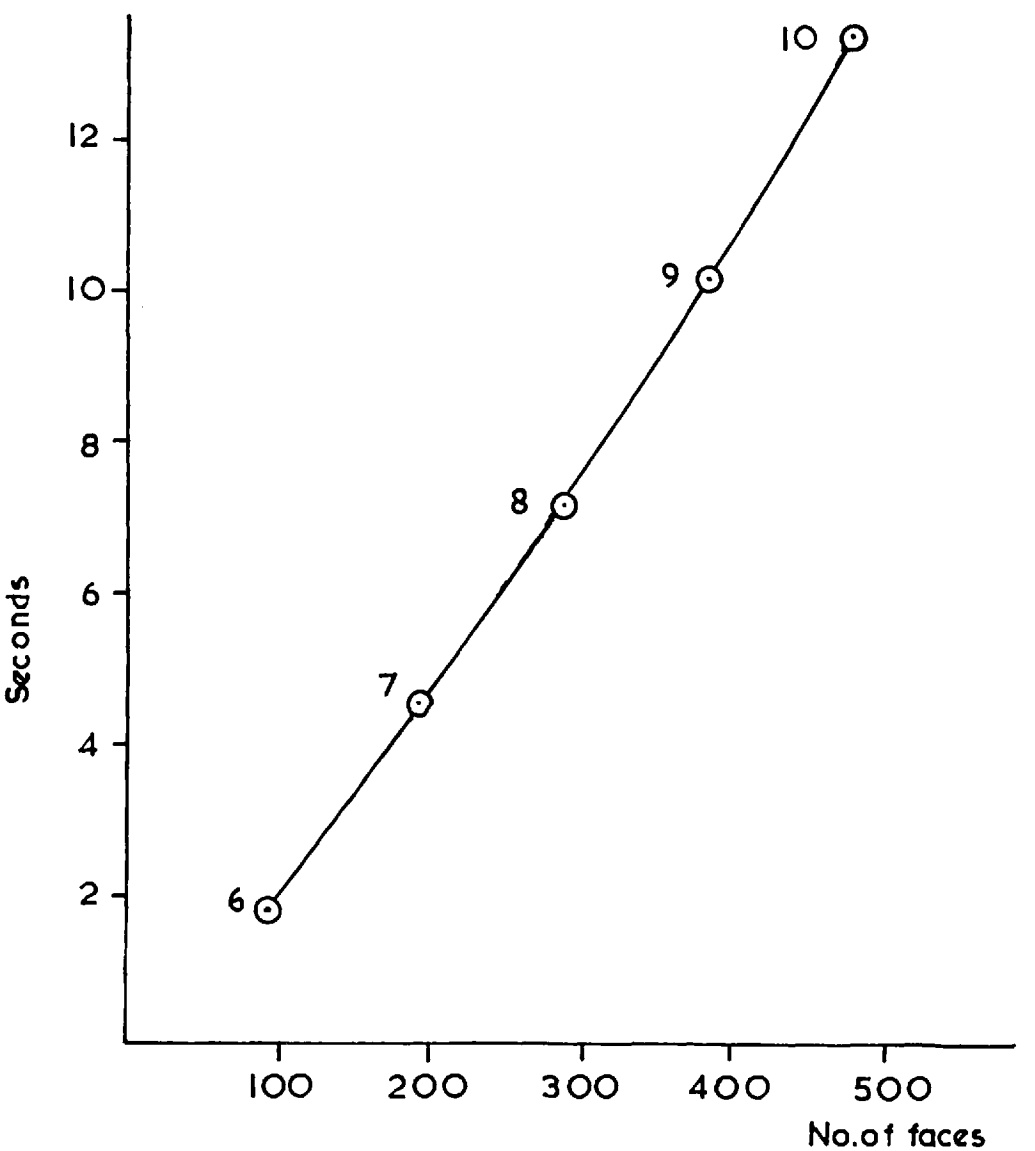
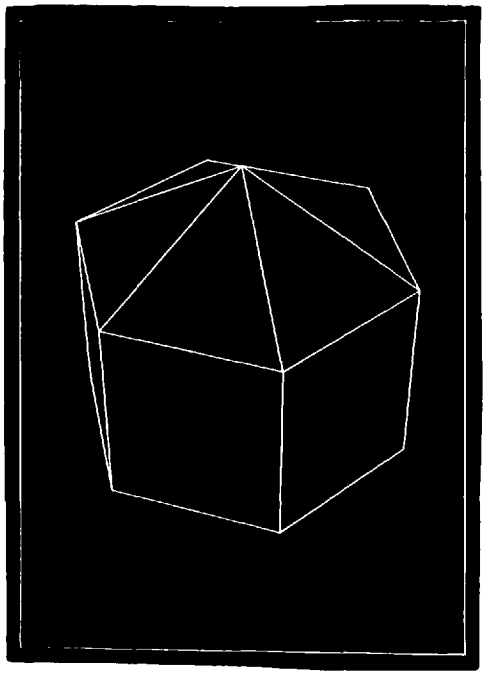
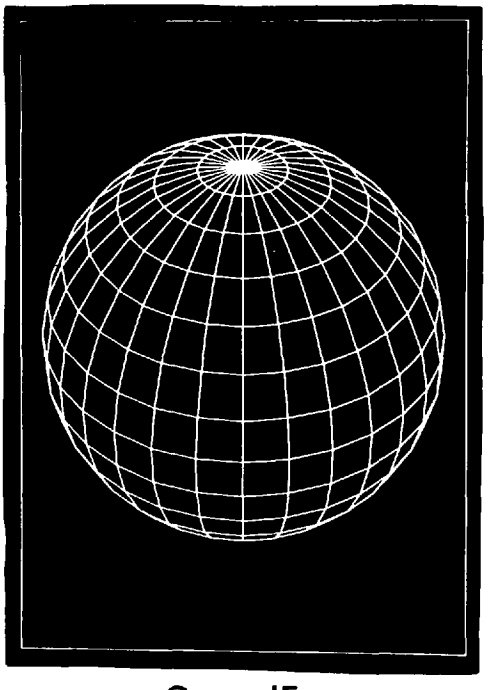


FIG.4-14: TEST RESULT ON GROUP 2



Case 11



Case 15

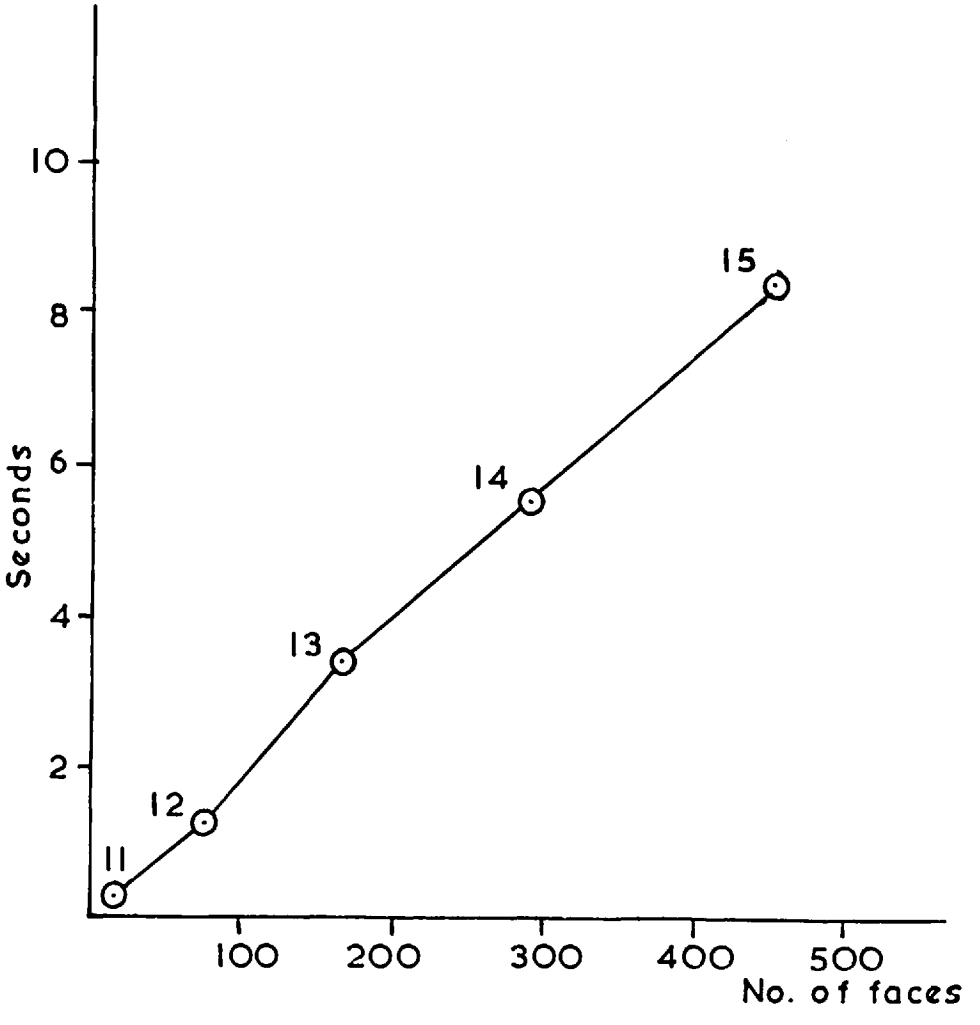
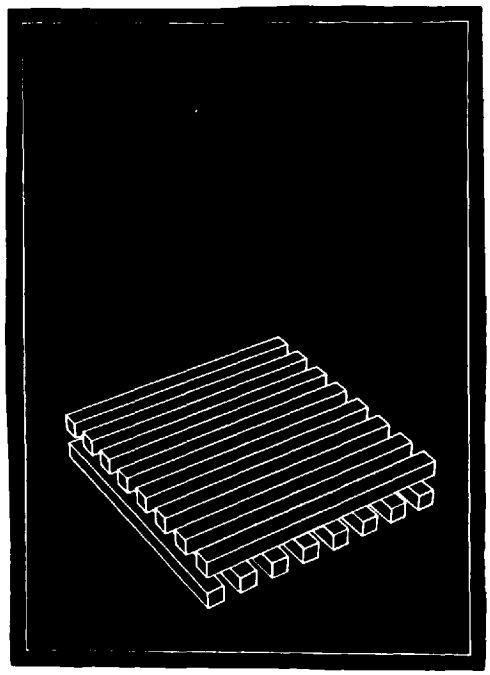
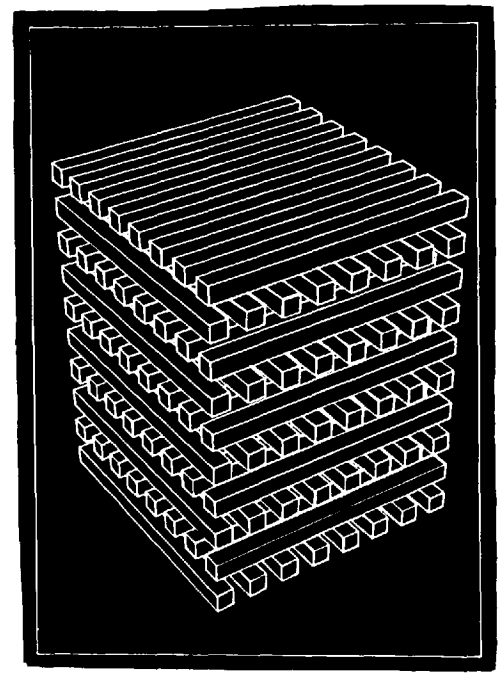


FIG. 4-15 : TEST RESULTS OF GROUP 3



Case 16



Case 20

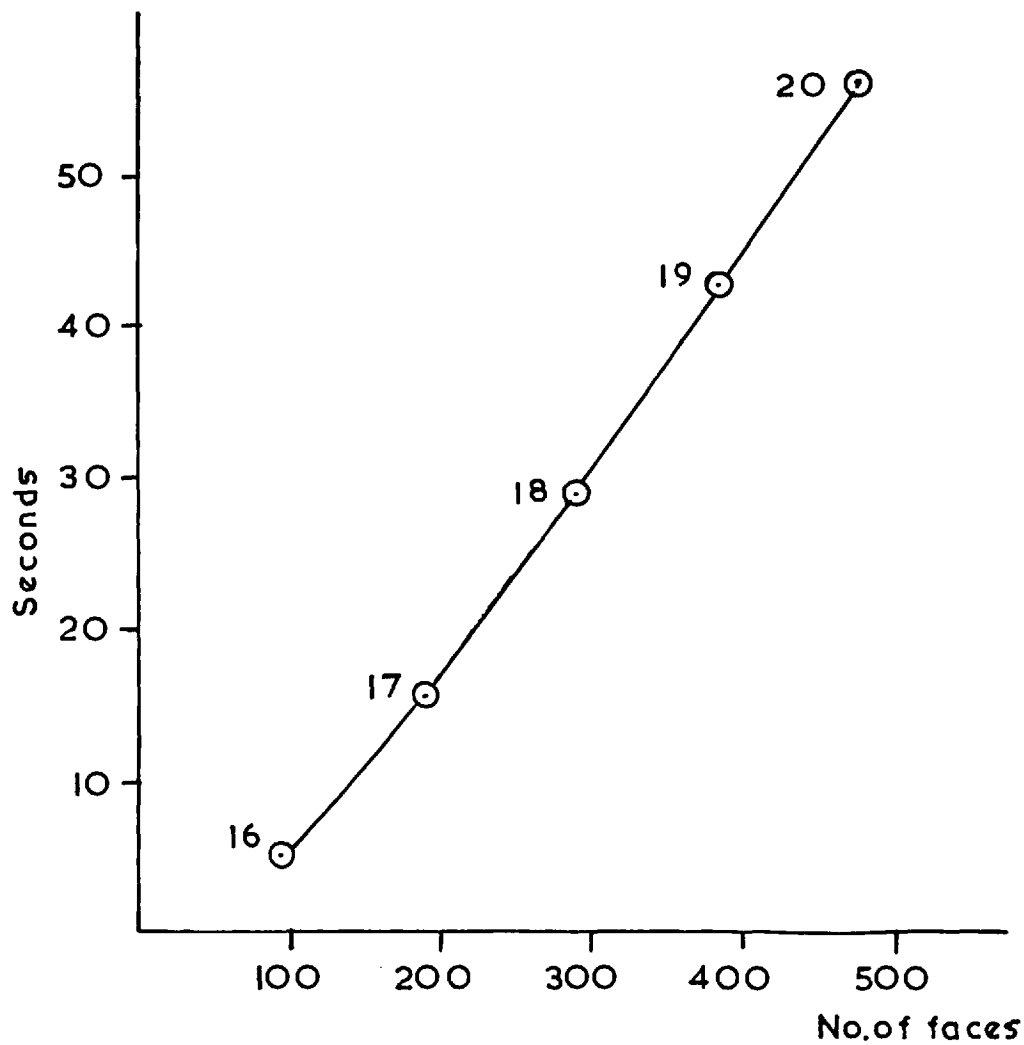
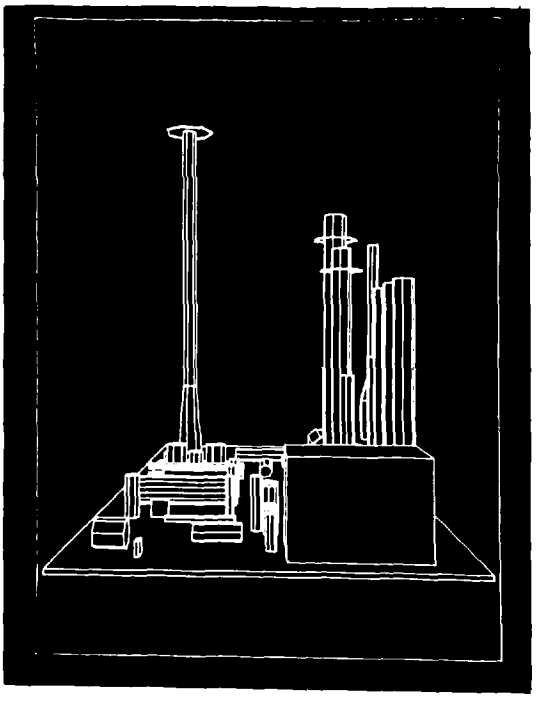
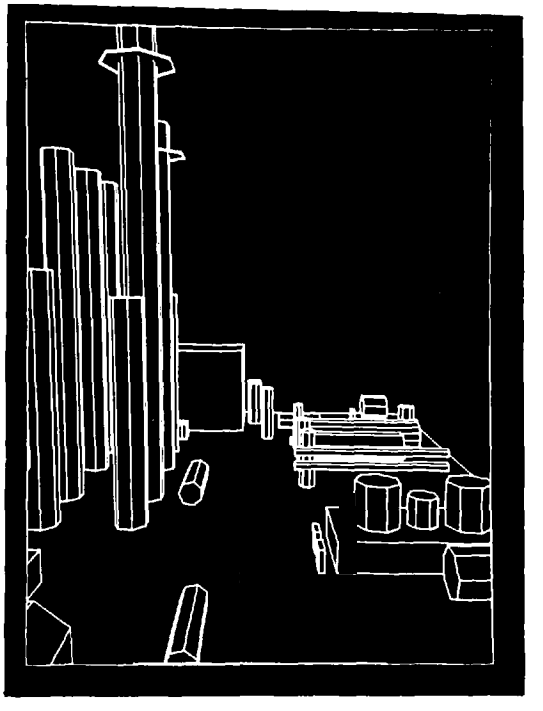


FIG. 4–16: TEST RESULTS OF GROUP 4





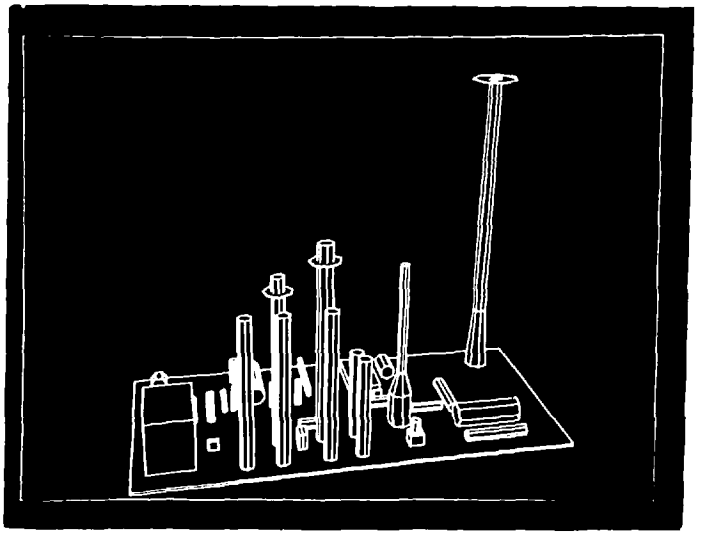


FIG. 4-17 : THREE VIEWS OF A TOPSA NAPTHA REFORMER

SCENE	COMPLEXITY							TIMES (SECONDS)	
	TOTAL POINTS	TOTAL EDGES	TOTAL FACES	EDGES CONSIDERED	FACES CONSIDERED	EDGE/FACE PAIRS	ENTRIES TO VSG	SETUP TIME	EDGE PROCESSOR
1	112	210	98	96	47	4512	289	0.3	1.3
2	224	420	196	197	93	17763	595	0.5	2.6
3	336	630	294	286	139	39754	972	0.7	4.2
4	448	840	392	380	184	69920	1384	0.9	6.0
5	560	1050	490	470	226	106220	1695	1.2	7.8
6	128	192	96	144	48	6912	295	0.2	1.6
7	256	384	192	288	96	27648	793	0.5	4.0
8	384	576	288	432	144	62208	1316	0.8	6.4
9	512	768	384	576	192	110592	1900	1.0	9.1
10	640	960	480	720	240	172800	2470	1.2	12.2
11	24	42	18	18	8	144	46	0.1	0.2
12	84	156	72	76	36	2736	230	0.2	1.0
13	180	342	162	158	76	12008	482	0.4	2.3
14	312	600	288	271	132	35772	845	0.7	4.8
15	480	930	450	426	208	88608	1412	1.1	7.4
16	128	192	96	144	48	6912	2059	0.3	4.6
17	256	384	192	288	96	27648	7154	0.5	15.1
18	384	576	288	432	144	62208	13504	0.8	27.9
19	512	768	384	576	192	110592	19961	1.0	41.7
20	640	960	480	720	240	172800	25747	1.2	54.7

TABLE 4/1: RESULTS OF HIDDEN-LINE ALGORITHM PERFORMANCE TESTS.

algorithm with the present method. These test cases have been devised so as to provide a variety of object types. The twenty test cases are made up of 4 groups comprising 5 cases in each group. The results obtained for each group are shown in figures 4-13 to 4-16. The first member of each group is the simplest and the fifth is the most complex. The four groups are as follows:

Group 1: (test cases 1 to 5)

Case 1 consists of one sphere, approximated by 98 faces (=7x14). Case 5 comprises 5 such spheres and the separation between the spheres is equal to the diameter.

Group 2: (test cases 6 to 10)

This group comprises from one to five layers of sixteen cubes each. The separation between the cubes being equal to the side length of each cube.

Group 3: (test cases 11 to 15)

Each member of group 3 consists of a single approximation to a sphere. The number of faces used in the approximation in the five cases being (3x6;6x12;9x18;12x24;15x30).

Group 4: (test cases 16 to 20)

This group comprises from two to ten layers of eight boxes each. The dimensions of each box are 1x1x15 with unit spacing between the boxes.

The depth ratio 'r' for groups 1,2 and 3 is roughly the same and this corresponds to similar computation times

for cases with similar numbers of edges and faces. However, the group 4 tests need about five times as much computation, corresponding to the much larger depth ratio. In all cases, the preliminary rejection tests reduce the number of edge/face pairs to be considered by a factor of around 3 or 4 but the real savings occur in the simple edge processor tests. For example in test case 10 (5 layers of cubes) the number of pairs is reduced from 172800 to 2470 , a factor of saving of about 70. Even in the case of the most difficult example, case 20 (10 layers of long boxes) there is a factor of saving of about 7. A further point of relevance is that within each group, the computation time increases a little more than linearly with the number of edges and faces in the scene.

The final test example shows a block model of a chemical plant comprising nearly 400 faces and just over 1000 edges. The computation time for these scenes is about 20 seconds on ATLAS II.

4.12 Comparison with Other Methods.

It is difficult to make a reliable assessment of the performance of the present method relative to other methods, since the information published with other methods is often scant or even non-existent. This is surprising, since the motivation for publishing a new method is usually to introduce some technique which is faster than existing techniques, yet the evidence to support this is always missing.

We will here consider a brief comparison with three other published methods, each of which embody fundamentally

different ideas. The three methods are those of Roberts, Loutrel, and Warnock.

Roberts' method was one of the earliest published methods. This method performs as the square of the number of objects in the scene and hence does not compete with the present method. The method attempts to formalise many of the techniques for detecting edge and volume intersections by using some of the ideas from linear programming. The method tends to deal in terms of volumes as the basic unit, rather than edges or faces and hence the processing needed for each 'unit' operation is large. For example to test whether a point is inside a volume is much more time consuming than to test whether a point is behind a face.

Loutrel published an algorithm that used the 'principle of quantitative invisibility' invented by Appel (18). The method assumes the scene is comprised of a set of closed, non intersecting polyhedra. The method attempts to cash in on the fact that visible edge segments can only be affected by the contour edges of each polyhedron (i.e. edges on the silhouette of each polyhedron). Thus if the scene comprises polyhedra where only a small percentage of the total edges are contour edges, then the method may have advantages. However, even in the case of a cube viewed on a corner, of the 9 visible edges, 6 of these are contour edges. Thus the factor of saving is not very large. Loutrel includes only one test example comprising a building of 74 faces and 220 edges. This took 5 seconds to compute on a CDC6600 - the CDC6600 being a much faster machine than ATLAS II. It is doubtful whether the object would have taken as long as 5 seconds on ATLAS, using the present

method.

Warnock's method has been widely recognised as the first successful attempt to produce an algorithm that performs nearly linearly with complexity. There is little information to indicate the speed of Warnock's method when used to solve the hidden-line problem, however, his algorithm applied to the hidden surface problem is slow. Unlike the first two methods mentioned, Warnock's method does not deal in terms of volumes or edges , but in terms of areas of the screen. The procedure recursively subdivides the screen into smaller and smaller areas until the remaining part of the scene within the area is simple enough to solve. Ironically, it appears that the complexity of the tests needed to check simplicity and the actual process of subdividing in fact take up a lot of computation.

CHAPTER 5

CONCLUSIONS.

5.1 Introduction.

In order to provide a complete computer-based design system for plant layout and pipe routing, then the basic algorithms described in chapters 2 to 4 need augmenting in a number of ways; first to overcome inadequacies in the algorithms themselves and second to provide solutions to the remaining problems which have not been tackled. In this chapter methods of overcoming these problems are suggested, and the outline of an integrated layout,routing and detailing system is given.

5.2 Further Ideas on the Vessel Layout Problem.

The vessel layout algorithms are the least successful on two main counts. First, the assumptions and simplifications made in order to tackle the problem are rather too severe. There are many more factors affecting plant layout than the cost of piping, and although some of these constraints can be included, one is still left with the problem of calculating a meaningful objective function. The approximations used to calculate piping costs are crude, particularly in the case of branched pipes, so even when an optimum is found, one cannot be sure that this is a real optimum in the corresponding real world situation. Of equal importance is the poor performance of the layout algorithms. It seems that little headway can be made without a radically different approach.

Current manual methods frequently employ simple models of the main plant items, the design proceeds by physically moving the elements of the model, while making a mental note of any violated engineering constraints, simple

hand calculations are used to estimate piping costs for each layout. This approach is certainly a proven one and hence it might be profitable to try and emulate it.

The most effective way of simulating a model is by means of an interactive graphics display. There are two main advantages that could be gained by such an approach. Firstly, the machine can very quickly calculate accurate estimates of piping costs for each layout generated by the designer at the display. Secondly, the designer could invoke the algorithms of chapter 2 to operate on small subsets of plant items, while maintaining overall control over the design. This hybrid approach of combining the automatic procedures with manual methods at a display would seem to be the most fruitfull proposition. Also, using the hidden-line techniques of chapter 4 would mean that realistic images of the computer based model could be obtained at any stage, although one would envisage working with wireframe drawings or even simple plans and elevations for any highly interactive work. The disadvantages of such an approach might be the limited screen size of most displays, the dependence on responsive computer resources which may be remote from the graphics terminal and also the difficulties inherent in several people working simultaneously at a display as opposed to standing around a model.

5.3 Further Ideas on the Pipe Routing and Detailing Problem.

In contrast to the vessel layout algorithms, the pipe routing algorithms proposed are much more successful, often

being able to produce realistic optimal routes conforming to the designer's wishes. However, further development work needs to be done, particularly with regard to the detailed design of pipe centre-lines, so as to ensure that all pipe clashes are resolved. The most profitable area to explore is the use of interactive graphics; however, whereas vessel layout problems are, to a large extent a 2D problem, the pipe routing problem is very much three dimensional. The synthesis of three dimensional shapes at a 2D screen is not very straight forward, but here we can provide a head start by routing all the pipes automatically using the algorithms of chapter 3.

The routing stage of the design process would start with an automatic routing system. This could just as well be offline as interactively driven. The designer would prepare simple instructions indicating the positions of pipe tracks, favourable and unfavourable routing areas. He would then request that some or all of the pipes be routed automatically according to these constraints. The pipe centre-lines so produced could be interrogated at the display screen, where it would be possible to perform edits on them. The problems of communicating 3 dimensional information via the screen are largely circumvented by the framework of routes already in existence when the interactive work starts. For example, it may be desired to move a section of pipe laterally on a pipe track. This can easily be done by identifying the pipe section and then requesting a move; the system would ensure that the pipe section remains connected at its end points.

The orthogonal nature of centre-lines is very helpful

for interactive techniques. One could display an isometric view (or indeed any axonometric view) of the plant and then any movement of a tracking cross can easily be identified as a movement in x,y or z. The actual manual routing of some pipes at the display screen is certainly a viable proposition. If a route is constructed on the screen, starting and ending at two known points, then provided the directions of all centre-line elements can be identified as belonging to one of the three standard directions of the isometric then the route is wholly determined in three dimensions. The construction of oblique sections of pipe is more complex, but still not insoluble. The fact that such sections of pipe are uncommon means that it does not matter if they are slightly more difficult to create at the display screen.

The great advantage of a route centre-line design system such as this is that the designer has to provide only a minimum amount of geometric information in order to construct the routes. Further, the system can always be instructed to search for pipe clashes and indicate these to the designer for him to resolve at the screen. This virtually guarantees error free routes and also avoids the problem of trying to devise an infallible pipe clash resolving algorithm.

A major problem is the detailing of fittings on the pipe centre-lines. It is possible to position some fittings automatically, particularly elbows and 'T' junctions. However, a development of the interactive routing system described above would solve the detailing problem particularly well. One would envisage displaying the pipe

centre-line to be detailed, and then fittings could be positioned by merely pointing at the centre-line. As the system knows all about the 3 dimensional geometry of the centre-line then it is easy to determine the 3 dimensional positions of fittings precisely.

5.4 Future Trends in Visualisation.

At any stage of the design process so far outlined, it would be convenient to provide an authentic looking visualisation of all or part of the design as in chapter 4. There are two restrictions to the hidden-line algorithm presented. The first is the inability to handle curved surfaces. However, good renderings can be obtained by means of polyhedral approximations. Perhaps a serious restriction is the number of objects that can be handled by the program. The implemented version can handle 500 faces, 750 points and 1200 edges. However, if one wished to represent adequately a plant of 20 vessels and 150 pipes, one might need 10000 faces. Such a scene would take minutes (but not hours) to compute and this would be tolerable. The main problem is how to handle the large amount of data involved. The object definition could only be held on backing store and one approach would then be to segment the object into areas of the screen, such that each area considered contains less than 500 faces. This segmentation could be done by means of a very simple procedure not requiring much computation or alternatively a segmentation procedure similar to Warnock's method of subdivision might be used. Future developments in the application of visualisation techniques could well include the use of half-tone pictures. In general, these are easier and quicker to

produce than hidden line pictures, but they can suffer from poor resolution in areas of fine detail. More important is the lack of generally available hardware to display such pictures. Hidden line pictures have the advantage that they can be displayed on any graphics device.

Current trends in 3-dimensional graphics hardware point the way to much more sophisticated visualisation in the future. There are already displays on the market that provide real time motion for not only wireframe drawings but also half-tone pictures. In the case of half tone systems, the two main drawbacks are the limited size of object handled and also the very high cost. It will be some time before we can look forward to a cheap simulated walk through a chemical plant.

5.5 Thoughts Towards an Integrated Layout, Routing and Detailing System.

In sections 5.2,5.3 and 5.4 we have examined the elements of an system which is hybrid in its approach combining the use of automatic algorithms with manual design methods at an interactive graphics display. Perhaps the main advantage that will be gained over the current wholly manual methods is the integration of the flow of information throughout the design process. At all stages, the transference of data from one design task to another can be automatic. It has already been mentioned that great improvements can be made in the economics of using current detailing and isometric systems. Also, similar interfaces can be provided to existing pipe stressing programs. At any stage of the design, intermediate materials listings can be

obtained at minimal cost and design documentation such as general arrangement plans can be produced automatically.

In order to provide an integrated system, it is necessary to provide a centralised data base which holds all information about the design. It is necessary that this data base is segmented such that different designers working on different areas of the plant can access the data base at the same time. Once the database is in existence, the design and implementation of both offline and interactive design programs can proceed.

It is essential that the design system is readily accessible to the designer, and in the case of the interactive programs the system must be responsive. The former requirement can be met by any of a number of commercially accessible multi-access systems, but the second requirement can only be met by means of a dedicated processor. Although processor costs are falling rapidly, the price of a large minicomputer is not yet low enough to be cost effective. Until facilities are available that combine the cheapness of multiaccess with the performance of a dedicated system, this is likely to remain a major barrier to the acceptance of the ideas presented here.

On the question of displays, the choice between the storage tube type display or a refreshed display is not clear. For many interactive tasks, a storage tube is adequate, provided the host machine is sufficiently responsive and the line bandwidth is sufficient to support frequent redrawing of the picture. However, for much of the 3D work, a refresh display is to be preferred since much of

the detailing of pipes and fittings frequently use light pen hits on existing picture parts. Also, real time rotation can be genuinely useful for appreciating a complex 3-dimensional situation. This facility is impossible to provide on a storage tube display, but adequate continuous rotation is easily provided by software techniques using a refreshed display.

The techniques exist now to produce an advanced layout, routing and detailing system whose performance would compete very favourably with current manual design methods. The increased speed of design incorporating automatic checking procedures should ensure better and cheaper designs in the future if such a system were implemented.

REFERENCES.

1 Vergin,R.C., and Rogers, J.D., "An Algorithm and computational Procedure for Locating Economic Facilities", Man. Sci., 13(6),(1967)

2 Hamelak, J.S., "The Placement of Computer Modules", J.ACM, 13(14),(1966)

3 Hillier, F.S. and Connors, M.M., "Quadratic Assignment Problem Algorithms and the Location of Indivisible Facilities", Man. Sci, 13(1),(1966)

4 Vollmann, T.E. and Buffa, E.S., "The Facilities Layout Problem in Perspective", Man. Sci., 12(10),(1966)

5 Pomentale, T., "An Algorithm for Minimising Backboard Wiring Functions", C.ACM, 8(11),(1965)

6 Steinberg, L., "The Backboard Wiring Problem: A Placement Algorithm", SIAM Review, 3(1),(1961)

7 Munkres, J., "Algorithms for the Assignment and Transportation Problems", J.SIAM, 5(1),(1957)

8 Lee, C.Y., "An Algorithm for Path Connections and its applications", IRE Trans. EC10, (pp346-353),(1961)

9 Nicholson, T.A.J., "Finding the Shortest Distance between two points in a Network", Comp. J., 9(3),(1966)

10 Farbey, B.A., Land, A.H., and Murchland, J.D., "The Cascade Algorithm for Finding All Shortest Distances in a Directed Graph", Man. Sci., 14(1),(1967)

11 Berge, C., "The Theory of Graphs and its

Applications", Methuen, London, (1962)

12 Loberman, H. and Weinberger, A., "Formal procedures for Connecting Terminals with a Minimal Total Wire Length", J.ACM, 4(4),(1957)

13 Galimberti, R. and Montanari, U., "An Algorithm for Hidden-line Elimination", C.ACM, 12(4),(1969)

14 Loutrel, P.P., "A Solution to the Hidden-line Problem for computer Drawn polyhedra", IEEE Transactions on Electronic Computers, EC19, 3 (1970)

15 Loutrel, P.P., "A Solution to the Hidden-line Problem for Computer Drawn Polyhedra", Dept. of Elec. Eng., New York Univ., Bronx, N.Y., Tech. Rep. 400-167,(1967)

16 Roberts, L.G., "Machine Perception of Three-Dimensional Solids", MIT Lincoln Laboratory, TR315, (1963)

17 Warnock, J.E., "A Hidden-Surface Algorithm for Computer-generated Halftone Pictures", Comp. Sci. Dept., Univ. of Utah, TR 4-15, (1969)

18 Appel, A., "The Notion of Quantitative Invisibility and the Machine Rendering of Solids", Proc. 22nd Nat. Conf. ACM, Thompson Books, Washington D.C. (1967)