COMPUTER REPRESENTATION OF GRAPHICAL INFORMATION WITH APPLICATIONS

NEWCASTLE UNIVERSITY LIBRARY
087 07079 4
Thesis L1283

E. S. HARRISON B.Sc. (Eng) A.K.C.

This thesis has been submitted for the
Degree of Doctor of Philosophy in the
Faculty of Science of the University of
Newcastle upon Tyne.

FEBRUARY, 1971

BEST COPY

AVAILABLE

Poor text in the original thesis.
Some text bound close to the spine.
Some images distorted

ACKNOWLEDGEMENTS

I should like to thank Dr. H.I. Scoins, Senior Lecturer in the Computing Laboratory for his help and guidance during my period of research at Newcastle.

I should also like to thank Mrs. Harvey for typing the thesis and for her help in collating it.

Finally, I am indebted to I.B.M. United Kingdom Laboratories for their help in the copying of this thesis.

ABSTRACT

The research work contained in this thesis lies mainly in the field of computer graphics.

The initial chapters are concerned with methods of representing three dimensional solids in two dimensions. Chapter 2 describes a method by which points in three dimensions can be projected onto a two dimensional plane of projection. This is an essential requirement in the representation of three dimensional solids.

Chapter 3 describes a method by which convex polyhedra can be represented by computer. Both the hidden and visible faces of the polyhedron can be located by the method described. Having tackled this problem, the rather more difficult problem of representing the non convex polyhedron has been attempted and the results of this work are presented in Chapter 4.

Line drawings of the various polyhedra, produced on a graph plotter, are given as examples at the end of Chapters 2, 3 and 4.

The problem of how to connect a given line drawing such that the distance travelled by the pen of some computer display is kept to a minimum is discussed in Chapter 5 and various definitions of the concepts involved are given.

Theory associated with this 'Pen-Up Problem' has been developed and is explained in detail in the early part of Chapter 6. A method of obtaining an optimal solution to the problem is presented in the latter part

of this chapter in addition to various enumerative schemes which have been developed to obtain good feasible solutions to the pen up problem under various conditions.

Extensive C.P.U. timing experiments have been carried out in Chapter 7 on the various enumerative schemes introduced in Chapter 6 and it has been possible to reach conclusions on the applicability of the various methods.

Several topics of interest which have arisen during the main research work are presented as appendices. The programs which have been coded during the period of research are also included as appendices.

E.S. Harrison.

CONTENTS

CONTENTS - Cont'd

CONTENTS - Cont'd

CONTENTS - Cont'd

PAGE

CONTENTS - Cont'd

LIST OF DIAGRAMS

LIST OF DIAGRAMS Cont'd

LIST OF DIAGRAMS Cont'd.

LIST OF DIAGRAMS Cont'd.

FIGURE LIST

LIST OF APPENDICES

BLANK PAGE
IN
ORIGINAL

CHAPTER 1

INTRODUCTION.

A general introduction to the Thesis with a comprehensive analysis of procedure line.

1.1. General Introduction.

The research work which has been carried out by the author during the last three years can be said to lie in the computer graphics field. The initial work concerned methods by which a straight line could be drawn by the pen of some computer display when the pen is restricted to move in one of a number of different directions. This work was particularly applicable to the display equipment available at the University at that time which was a Benson Lehner graph plotter in which the pen is restricted to move in any one of eight directions. The work was essentially an extension of research which had been initiated by Ian Leitch at the Laboratory in collaboration with Dr. Scoins, a Senior Lecturer in the Laboratory.

The extension concerned a condition which was known as a 'Knights Move'. Since the research presented in this thesis is almost entirely concerned with the drawing of lines on a computer display (in one way or another) this initial research work has been included in this introductory chapter for completeness. A mathematical consideration of the problem is presented and examples of the Knights Move condition (produced on the graph plotter) are included at the end of the chapter.

One of the main fields in computer graphics is the computer representation of three dimensional solids in two dimensions. For the last six years or so many researchers have tackled the problem.

The simplest method and naturally the first to be attempted, was the wire frame drawing in which the three dimensional solid is represented by a number of lines in space. These lines are then projected (by some convenient method) onto a "plane of projection" which has been mathematically defined. The lines on the projection plane, corresponding to the lines in space, are then connected, there being no attempt made to ascertain those lines which are hidden to the viewpoint.

Puckett[18] in his work for N.A.S.A. (National Aeronautics and Space Administration) and Johnson[12] at M.I.T. (Massachusetts Institute of Technology) were perhaps the first researchers to utilise this technique, which has become widely used in the design of motor cars and the like.

However, before any work can proceed in this type of research, it is necessary to have some means of projecting points in three dimensions onto a two dimensional plane of projection. Methods of projection and the algorithm which has been used to project a number of spatial vertices onto a projection plane are discussed in Chapter 2. The main type of projection which has been utilised is perspective.

Methods of projection, related to computer displays, are being considered by some researchers (such as Sutherland[27]) but it has not been possible to devote much time to this and the author has contented himself with a method which allows a solid to be represented by both orthographic and perspective projections. The projection algorithm has essentially been developed to allow more fruitful avenues of research to be investigated.

The solids which are represented by computer in this thesis are polyhedra. Various other forms of solids can be represented by computer of course and one of the first researchers to produce a method of representing quadratic surfaces was Ruth Weiss.[5]

Probably the first researcher to consider the polyhedron in a computer display was Larry Roberts[2] at M.I.T. He produced an algorithm in 1963 which considered polyhedra and a method by which the hidden lines could be eliminated. From this initial work various people attempted methods of representing the convex polyhedron (such as Loutrel and Cole). The convex polyhedron was represented initially since this is a particularly simple case in that any face of the polyhedron is either completely hidden or completely visible to the viewpoint. From these first steps it was possible to consider the more difficult exercise of representing the general polyhedron which can have faces which are only partially hidden to the viewpoint.

The method adopted by Cole[8] could not be extended to the general polyhedron (as could Loutrel's[9]) and the method as presented had several rather grave disadvantages, although the general approach seemed to possess many advantages.

This general approach to the problem of representing the convex polyhedron has been used in Chapter 3 and many of the disadvantages of Cole's initial method have been overcome. In addition, new techniques have been introduced to produce an algorithm which appears to have some powerful features.

It is possible to choose any viewpoint in space from which to view the convex polyhedron with the restriction that the viewpoint lies outside the polyhedron. The first main step in the algorithm is to locate the visible plane faces and the associated visible edges. It is then possible, if required, to locate the hidden faces by a slight change in the logic of the program.

Various concepts and definitions are introduced during the description of the method and two hand-worked examples are developed. Several computer displays (produced on the graph plotter) are presented at the end of the chapter. Some of these show the hidden edges with dotted lines and some have the hidden edges completely removed.

The rather more difficult problem is to produce a computer display to represent the general polyhedron and a completely new approach has been taken in this case. The type of approach adopted is similar to that of Loutrel[10] and Appel.[14]

It is convenient in the case of the general polyhedron, which can have an edge with portions both visible and hidden to the viewpoint, to calculate, for a given edge, the number of plane faces of the polyhedron which hides the edge from the viewpoint. Since this number, which has been defined as the 'depth count', can vary along a given edge it is convenient to divide an edge into a number of portions, each of which has a given depth count. In this way it is possible to determine the portions of any given edge which are visible (or hidden) to the viewpoint.

In this respect the line segments on the projection plane which correspond to the edges of the polyhedron have been divided into a number of 'partial line segments' (a formal definition for which is given) which are associated with a constant depth count. Those partial line segments which have zero depth count have no faces of the polyhedron which hide them from the viewpoint and are thus visible. Non-zero depth counts (which will be positive) signify partial line segments which are hidden to the viewpoint.

Rather than have to calculate the depth count of every partial line segment it has been possible to calculate the change in depth count along a given line segment. In this way it is only necessary to calculate the depth count of one of the partial line segments (of a given line segment) and from this value the depth counts of the other partial line segments (belonging to the given line segment) can be obtained.

One of the most important computational aspects of the algorithm is the determination of the intersection points occurring between the line segments on the plane of projection and because of this the method used has been explained in some detail.

The results of this research work are presented in Chapter 4 and a number of computer displays of non convex polyhedra are given at the end of the chapter.

It should perhaps be pointed out at this stage that there are basically two different types of approach to computer displays of three dimensional solids, depending on the type of hardware being used. The 'calligraphic' display (described in a later section) is concerned with the drawing of lines on the display 'screen' by an oscilloscope type of action. The graph plotter belongs to this category in which edges of the solids being considered are depicted by line segments on the screen of the display.

In the 'raster' type of display it is possible to shade the surfaces of the solid and produce different tones on the screen, so that a given solid can be represented simply by the various shades produced on the screen. By these methods it is possible to reproduce photographs by computer and a new field of research comes within reach.

Unfortunately, it was not possible to produce any worthwhile research in this field since, to produce shading using an off-line graph plotter necessitates the production of an enormous amount of paper tape. Although work was initiated in this direction by the author, it was quickly realised that the facilities available precluded any hope of productive research.

The final results of the work which has been described so far produces a line drawing on the plane of projection. Initially these line drawings were drawn on the graph plotter simply by building into the problem programs commands which would produce instructions to the graph plotter to draw the various line segments as the program proceeded. Later in the work various simple methods were incorporated so that the distance moved by the pen of the graph plotter tended to be reduced. As these various methods increased in number it became obvious that it would be beneficial to produce an algorithm which would accept the line segments of a line drawing (in addition to the co-ordinates of each point) and utilise some method of connecting these line segments such that the total distance travelled by the pen was kept to a respectable minimum. Since the line drawings produced could be described as connected graphs the problem was defined as being that of developing a method of drawing a connected graph such that the distance travelled by the pen, in excess of the distance of the line segments, was minimised.

The excess distance travelled by the pen corresponds to the distance travelled with the pen in a raised position. So that it is possible to connect the line drawing by a continuous sequence of pen movement it was found necessary to add to the original line drawing a number of pen-up lines (line segments drawn with the pen in a raised position). It was thus necessary to choose these pen up line segments such that their total distance tended to be minimal.

The various concepts which have been developed and a discussion of the Pen Up Problem (as it has been called) is given in Chapter 5. It is necessary, in fact, to choose the pen up segments such that the addition of these to the original line drawing converts the line drawing to one which has all of its points associated with an even number of line segments. Graphs of this kind are known as Eulerian Cycles and their importance lies in the fact that it is possible to connect them by a continuous line without going over any line more than once.

This of course is the requirement to connect a given line drawing by a continuous sequence of pen movement. The pen up problem is concerned with the number of odd degree points n in the line drawing for it is necessary to select $^{n}/_{2}$ line segments (which will be pen up lines) between these n points such that the odd points are each associated with just one of the pen up segments. As a simple example consider the line drawing shown below

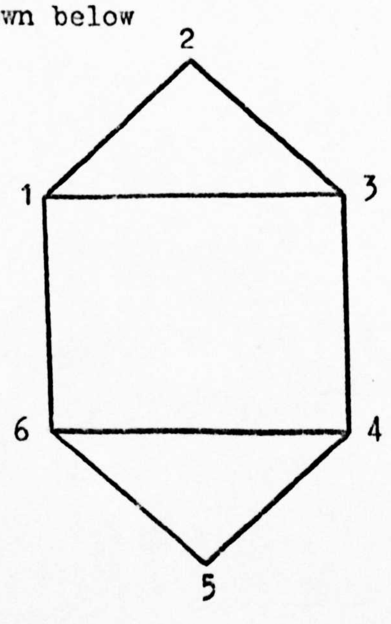


Diagram 1.1

Of the six points in the drawing four are of odd degree and it is thus necessary to select two pen up lines which span these four points. A valid choice could be 1-3 and 4-6 for example or 3-4 and 1-6.

It should be clear that given n odd degree points there are a number of ways of selecting the $^n/_2$ pen up lines. The total number of distinct pen up lines is given by n.(n-1)/2 so that when n = 4 the number of distinct pen up lines is 6. These are 1-3, 1-4, 1-6, 3-4, 3-6, 4-6 for the example given.

The number of ways of choosing the $^n/_2$ pen up lines is equivalent to the number of combinations of $^n/_2$ pairs of points from the n available. This number is given by

$$\frac{n!}{\left(\frac{n}{2}\right)!\,2^{n/2}}$$

In this case (for n = 4) the number of ways of choosing the pen up lines is 3. The 3 choices are as follows :

- 1-3, 4-6
- 1-4, 3-6
- 1-6, 3-4

So as to reduce the total pen up distance it would be best to select those two pen up lines such that the total distance was a minimum.

Suppose 1-3, 4-6 is an optimal choice in the example given. The segments would then be added to the line drawing as pen up lines. It is then necessary to assign a direction to each of the line segments so that the pen can connect the line drawing by a continuous sequence with the proviso that some of the segments are connected with the pen in a raised position. One solution would be as follows :-

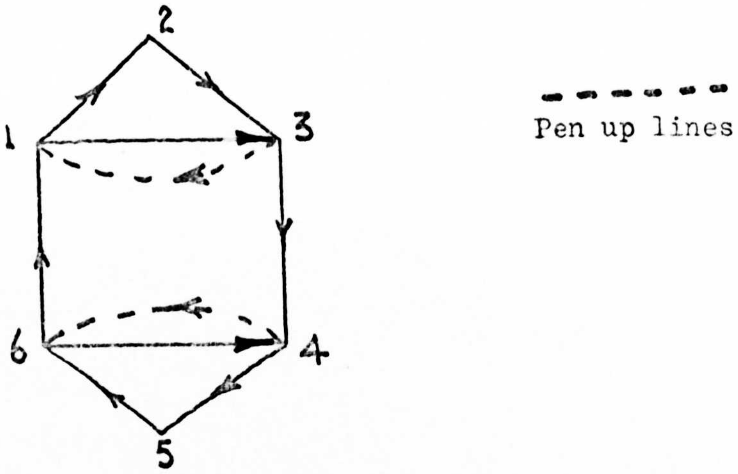


Diagram 1.2

with the pen starting and finishing at point 1. The direction of travel of the pen is given by

1-2, 2-3, 3-4, 4-5, 5-6, 6-4, 4-6, 6-1, 1-3, 3-1

It is obvious that it is desirable to obtain a quick choice of the pen up lines and in this respect a heuristic method of choosing the pen up lines has been developed. The method is described in Chapter 5.

For any given line drawing there are various ways in which the drawing can be connected depending on the choice of pen up lines. There exists, for any line drawing, a particular choice of pen up lines such that the total distance travelled by the pen of the display is a minimum. This has been called the Optimal Pen Up Problem and an implicit enumeration algorithm has been developed to obtain an optimal solution. This algorithm is described in Chapter 6 and the theory associated with the pen up problem is also elucidated in this chapter.

The implicit enumeration algorithm requires both a lower bound cost and an upper bound cost. The lower bound has been obtained by defining a problem which is related to the pen up problem and has been called the Image Pen Up Problem. A feasible solution to this problem furnishes a lower bound cost to the pen up problem.

The upper bound cost has been obtained by utilising the concept of the 'domino graph, a term first introduced by Scoins and Snow in their research at the University. The methods by which the upper bound cost and the lower bound cost are obtained for the pen up problem are explained in Chapter 6.

Methods to reduce the time taken to reach a good (or optimal) solution to the pen up problem (under various conditions) have been developed and are included at the end of Chapter 6.

Chapter 7 consists of timing experiments which have been carried out on the IBM 360/67 computer to compare the C.P.U. (central processing unit) times taken by the various forms of the general implicit enumeration algorithm. The C.P.U. times taken to obtain both the upper and lower bound costs have also been obtained.

The C.P.U. times have been compared and various conclusions have been made. The C.P.U. timer used in this chapter has been explained in one of the appendices.

Various related research topics which have arisen from the main research work are also included in the appendices.

Listings of the more important programs which have been developed are given at the end of this thesis. It has been the aim of the author to make these as clear as possible by the inclusion of numerous comments. Flow diagrams are also included in the main text of the thesis and usually follow the explanation of the method used.

The thesis includes a number of line drawings which have been output on the graph plotter. So that these drawings can be recognised at first glance, they have been referred to as 'figures' and a figure list is given in the contents. Line drawings which have been drawn by hand have been referred to as 'diagrams' and a diagram list is also given in the contents.

1.2. Facilities Available at Newcastle University Computing Laboratory.

The two computers in general use at Newcastle University are the I.C.L. KDF9 computer and the I.B.M. 360/67 which was the first multi-access computer produced by I.B.M.

The total storage of the I.C.L. computer is 16K 48 bit words and that of the I.B.M. computer is 192K 32 bit words with an additional 1,000K words of virtual storage.

A Benson Lehner incremental graph plotter, which is off-line, is available at the laboratory. The input data to the plotter consists of characters on paper tape produced as output from the KDF9 computer. The drawings shown in this thesis were produced on the Benson Lehner plotter. Since the 360/67 computer does not have an output paper tape punch it was necessary to use the KDF9 in producing the output paper tape which was subsequently input to the graph plotter.

There exist two Algol compilers associated with the KDF9 computer. Walgol (Whetstone Algol) is a load and go compiler used for programs requiring less than five minutes of C.P.U. time. Kalgol (Kidsgrove Algol) is the compiler used for programs requiring more than five minutes C.P.U. time and in this case the compiled programs are stored on magnetic tape and may be accessed by use of a suitable call tape.

C.T.P. Walgol was introduced by a research group in Computer Typesetting and is used where output paper tape from the KDF9 is to be produced as input data to the graph plotter. In this case procedures, which produce the characters to move the pen of the graph plotter, exist as library procedures. C.T.P. Walgol uses the on-line paper tape punch.

When large numbers of characters are required at the output punch it is not economical to use the on-line paper tape punch since elapsed computer time can become excessive. As a direct result of this a scheme known as Device 4 has been introduced which allows characters to be stored on audio magnetic tape instead of being output to the punch. In this way the elapsed time required by a given program (producing paper tape output) can be substantially reduced. Paper tape output can be produced from Device 4 by an associated off-line punch which simply copies the characters from the audio magnetic tape.

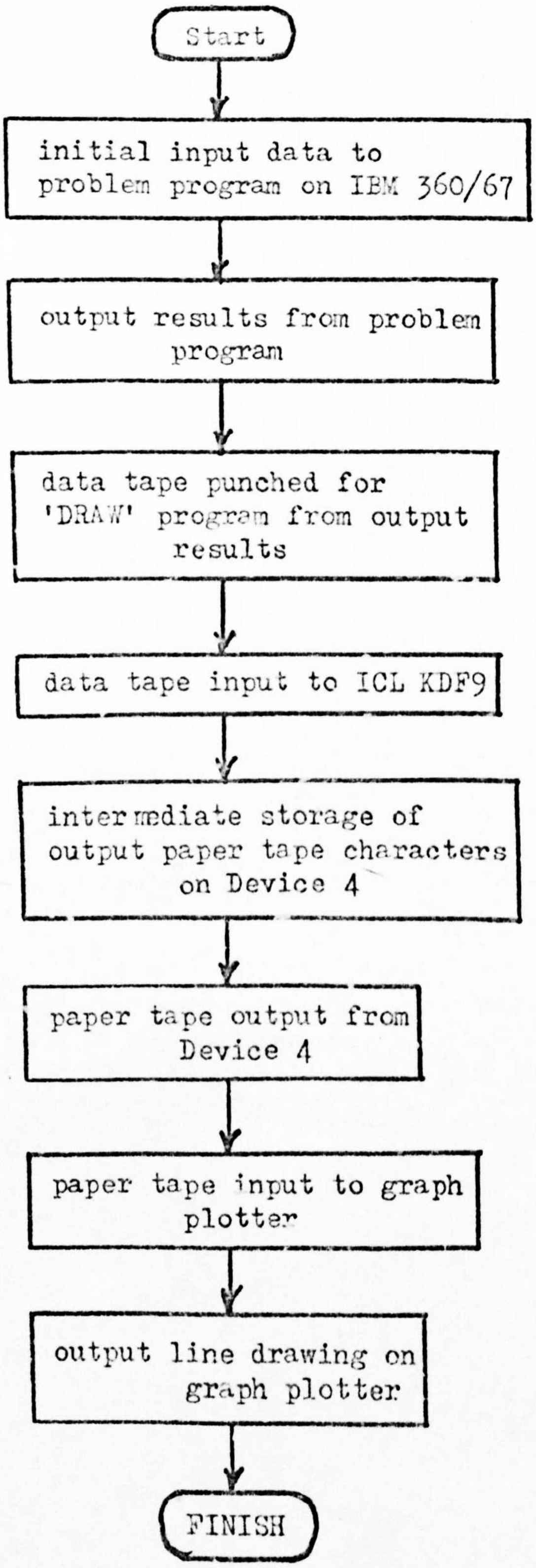
It is necessary when using Device 4 to have the compiled programs on magnetic tape and so it is necessary to use the Kalgol compiler. A short description of Device 4 is given in a later section. The I.B.M. 360/67 computer, on which most of the programming for this thesis was carried out, is a multi-access computer which facilitates up to 32 separate consoles at any given time. It is possible to store programs on disc and various editing and other facilities are available. The Michigan Terminal System (M.T.S.) has been implemented: thissystem was developed for a 360/67 computer at Michigan University in collaboration with I.B.M.

The programs for this thesis were written in

Algol 60 and the I.B.M. computer was used because it was easier to debug programs using a terminal system, with the extensive editing facilities available, than on the KDF9 computer.

Since it is not possible to produce paper tape output from the 360/67, there being no paper tape punch, it was found necessary to write a program to run on the KDF9 computer which accepted as data the line segments of a given line drawing. The output from this program was paper tape which was then fed as input to the graph plotter.

The drawings shown in this thesis have been produced with the use of this program. It is in fact the only program which uses the KDF9 computer, all other programs being run on the more powerful I.B.M. computer.


Start
initial input data to problem program on IBM 360/67
output results from problem program
data tape punched for 'DRAW' program from output results
data tape input to ICL KDF9
intermediate storage of output paper tape characters on Device 4
paper tape output from Device 4
paper tape input to graph plotter
output line drawing on graph plotter
FINISH

FLOW DIAGRAM OF SCHEME USED TO OBTAIN FINAL LINE DRAWING ON GRAPH PLOTTER UTILISING THE FACILITIES AVAILABLE.

Diagram 1.3

1.3. Computer Displays.

There are two basic types of computer displays in common use.

The first of these is the incremental plotter, the computer generated data for which consists of information to move the pen. The plotter can be either on-line or off-line. On-line plotters receive the data directly from the computer whereas off line plotters are fed with data in tape or card form previously output from the computer. The plotters are incremental in that the pen moves in small increments in any one of a specified number of directions. The plotter at Newcastle University Computing Laboratory is a Benson Lehner incremental plotter which is described in the following section. It is possible to obtain various colours of drawing simply by changing the colour of ink in the pen. The second and perhaps more widely used display is the cathode ray tube display which produces the drawing on a screen very similar to the oscilloscope in electrical engineering. The cathode ray displays can themselves be split into two broadly defined subsets.

The calligraphic display converts the digital information from the computer into analogue form at the electrodes so that the electron beam is suitably deflected across the screen. Calligraphic displays are therefore very similar indeed to the ordinary oscilloscope in that voltages are varied to produce electron beam deflection. The information associated with any drawing can therefore be suitably arranged within the computer before being output to the display. The second type of cathode ray tube display is the raster display which produces pictures in a similar way to that of a television set. The picture is produced in a fixed and unvarying sequence which is usually from left to right and from top to bottom. The raster display thus has the disadvantage that the information associated with any picture must be sorted in the same way each time, that is, from left to right and from top to bottom. It is mainly due to this restriction that raster displays are utilised for character displays more than anything else. Apart from this restriction, raster displays are useful in that the electronic equipment required, such as deflection amplifiers, is fairly cheap when compared to the more elaborate equipment required for the calligraphic display.

Raster displays are very useful if shading of the surfaces of solids is required and pictures of this sort have a decided advantage over those displayed by the incremental plotters and calligraphic displays, which are usually line drawings. Any attempt to shade surfaces using these displays has always met with very unsatisfactory results.

The resolution of the cathode ray tube display is dependent on the number of rows and columns of points which are present on the screen and which define the display co-ordinate system. This number can vary from say 1024 on a good display to even 4096 on the most modern displays at present in use. The resolution of the incremental plotters is governed by the distance of a single increment which is 0.1 mm on the Benson Lehner plotter and can be as good as 1/500 inch on a modern Calcomp plotter.

1.4. Graph Plotter.

The computer drawn figures in this thesis have been produced on the Benson Lehner incremental graph plotter, the input data for which was output from the KDF9 computer.

The graph plotter is off-line in that paper tape output from the computer acts as the input to the graph plotter.

The pen of the graph plotter can move in any one of eight numbered directions as shown below.

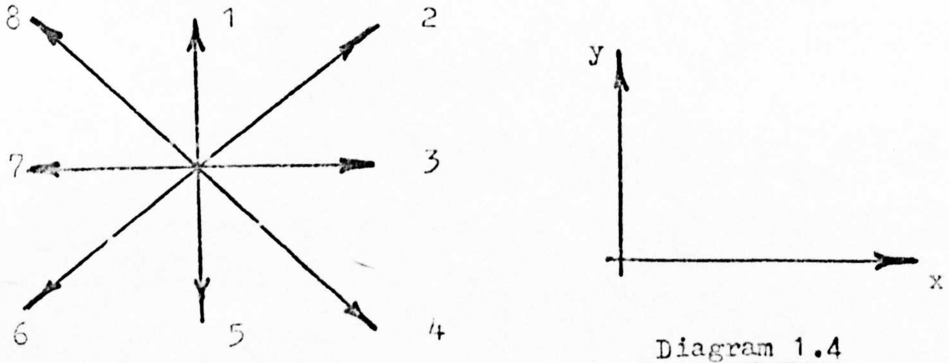


Diagram 1.4

The increment of the pen in each direction is 0.1mm measured along the cartesian axes so that a diagonal move in directions 2, 4, 6 and 8 is of length $\sqrt{2}$ x (0.1) mm.

The maximum speed of the pen is 2 cm/sec and two pens of nib widths 0.2mm and 0.4 mm are available for use.

The paper tape code conveys two plotting instructions for every row of paper tape output. The first four holes convey the first instruction and the second four holes convey the second instruction. The plotter recognises in all twelve separate instructions from the paper tape. Eight of these correspond to the eight different directions of motion shown above and the remaining four instructions will cause the pen to raise, lower, do nothing or stop reading paper tape.

A procedure plot (r,s) will output characters on paper tape corresponding to the numerical values assumed by r and s. The procedure outputs two instructions at once_ so that one row of paper tape is output for each call of plot.

Movement in any one of the eight directions is possible by assigning the corresponding numbers to r and s, which are of course both integer. Special combinations of the integers r and s are used to produce the pen up, pen down, do nothing or stop instructions. After each of the pen up, pen down and stop instructions, three do nothing instructions are inserted to allow for the finite time of mechanical motion in pen movement.

The system software has its own cartesian co-ordinate system and at the beginning of any plot the pen is assumed to be at the origin (0,0).

Restrictions exist for total movement in any one of the four cartesian directions in any given program as follows

$$4000 \geq x \geq -200$$

and $$3500 \geq y \geq -1500$$

The upper bound on x can be changed by a special plot instruction.

The most severe restriction on pen movement is in the negative x direction which corresponds to the paper unwinding into the plotter. It is in this direction that there is danger of the paper coming out of the sprocket holes which explains why only 2 cm of movement is allowed in this direction.

It should be clear from this that it is always desirable to begin a plot at or very close to the lowest x value of the line drawing. If the pen does in fact push up against the lateral extremities of the plotter the photo-electric paper tape reader will stop reading tape.

It is possible to move the pen by a manual control in order that the pen may be suitably positioned before any tape is read by the photo-electric reader.

The procedure plot is an Algol procedure with a Usercode body and is available as a library procedure in C.T.P. Walgol. Its function is to produce paper tape from the output punch of the I.C.L. KDF9 computer corresponding to the instructions specified by the integers r and s. The procedures open gp and close gp are analagous to the standard procedures open (dv) and close (dv). They respectively claim and deallocate the paper tape punch of the I.C.L. machine and are library procedures in C.T.P. Walgol. Open gp should be called before the first call of plot and close gp is called following the last call of plot.

Open gp claims buffer areas required by the procedure plot and is written in Usercode. Close gp before deallocating the punch causes any partially filled buffer area used by plot to be output. Close gp is also written in Usercode.

Gap gp (n) is a Usercode procedure which produces 2n 'do nothing' instructions and is a library procedure in C.T.P. Walgol, the formal parameter n being a positive integer.

1.5. Device 4.

When large amounts of paper tape are to be generated by computer it is not economical to use the on line paper tape punch since the total elapsed time required by the computer may become excessive.

In this respect a system was developed at Newcastle University which allowed the characters corresponding to the paper tape characters to be stored on audio magnetic tape. By this means the total elapsed time taken by a given program producing output paper tape can be reduced.

The system by which this was carried out is known as Device 4. When the given program has finished it is possible to retrieve the paper tape characters from the magnetic tape by initiating an off line paper tape punch attached to Device 4. This punch produces paper tape characters corresponding to those stored on the magnetic tape. It should also be clear from this discussion that an added advantage of Device 4 is that it is possible to retrieve punched tape should a roll be lost or damaged.

1.6. Algorithm to Compute a Coded String of Instructions to Move a Pen Between Two Points by Incremental Steps.

This thesis contains methods by which polyhedra can be drawn by computer. The representations consist of various line drawings of the polyhedra. The hidden lines can be shown dotted, full or can be omitted altogether.

Another important aspect of the thesis is that the organisation of any given line drawing and problems related to it are tackled.

It can thus be said that this thesis is concerned alomost entirely with the drawing of lines in one way or another.

Early research work by the author consisted of methods by which an incremental plotter, which could traverse in any one of eight directions, could be programmed to trace a line between two given points. The results of this work are presented in the following section.

The line traced out by the pen is the 'best' approximation to the actual straight line in that each move is at least as good as any other possible move under the existing conditions. The line can be considered to be drawn on a mesh of size h where h is the incremental length of the pen in the directions of the cartesian axes. The extreme co-ordinate positions will be assumed to lie on the grid.

Directions referred to are those numbered in the se c tion describing the Benson Lehner graph plotter.

TEXT
CUT OFF IN THE
ORIGINAL

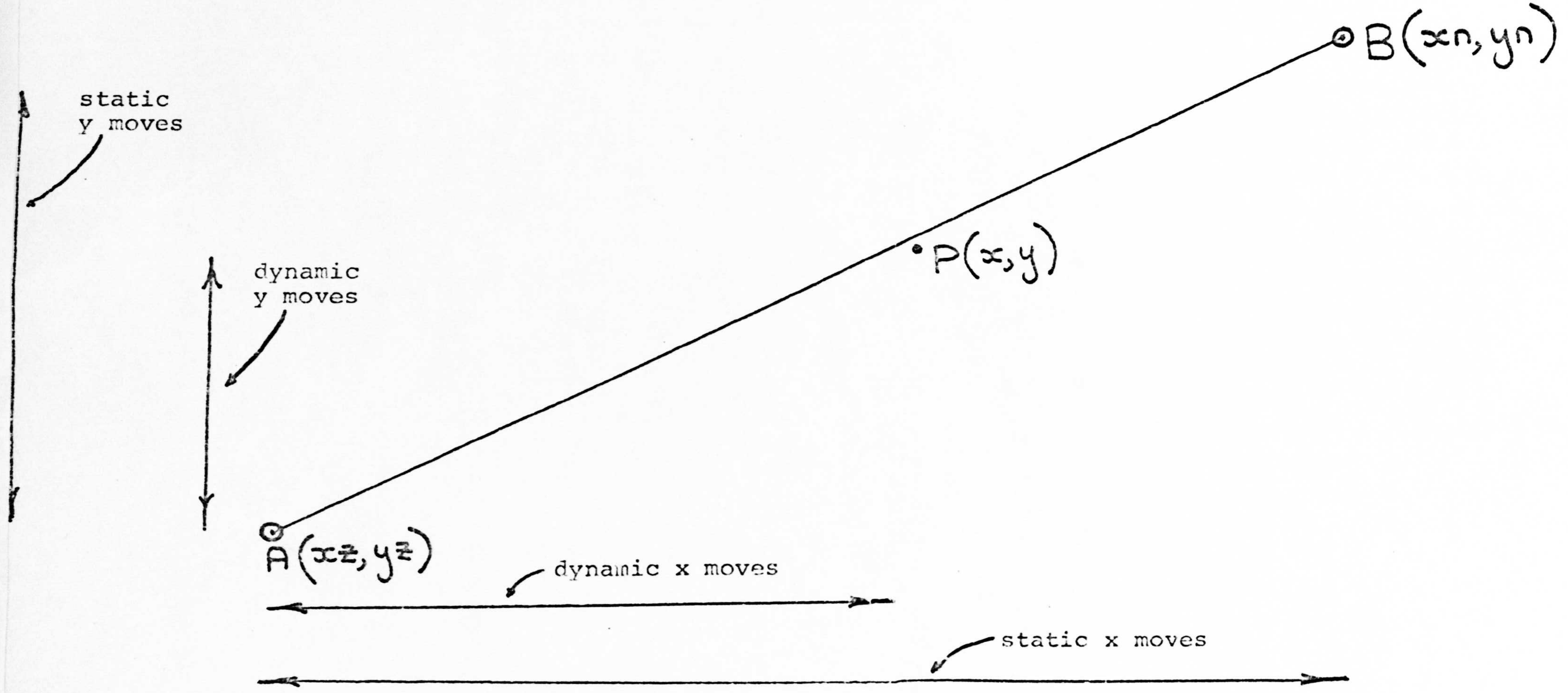

B(xn, yn)
P(x, y)
A(xz, yz)
static y moves
dynamic y moves
dynamic x moves
static x moves

Diagram 1.5

The problem is to compute a string of incremental steps from A to B such that for any increment the pen is as near the true line, shown in full, as any other candidate for that position.

No generality is lost by considering the line to be drawn in the first quadrant. (See Diagram 1.5).

For any given extreme co-ordinate positions A and B it is only necessary to utilise two directions of the pen. One of the moves will be a straight (s) move parallel to the cartesian axes and the other will be an adjacent diagonal move (d).

It is a fairly simple matter to determine the s and d moves for any given points A and B. Consider the following booleans

$$B = xn - xs > 0 \qquad C = yn - ys > 0$$

$$K = \text{abs}\ (xn-xs) - \text{abs}\ (yn-ys) > 0$$

If B is true the d move must be either 2 or 4 and the s move 1,3 or 5. If in addition C is true the d move must be 2 and if K is true the s move must be 3.

Thus, by inspection of three booleans, it is possible for any given extreme co-ordinate positions A and B, to determine the d and s moves as shown in tabular form below.

	d move		s move
$B\wedge C$	2	$K\wedge B$	3
$B\wedge C$	4	$K\wedge B$	7
$B\wedge\overline{C}$	8	$K\wedge C$	1
$B\wedge\overline{C}$	6	$K\wedge C$	5

Suppose the total number of moves in the x direction between A and B is a0, the static x moves. Similarly b0 will be the static y moves.

Consider that a string of instructions has been determined and that the pen is currently at a position P (x,y). The number of x moves made to P is say a, the dynamic x moves and b is the corresponding dynamic y moves made to P. As the string of instructions increases a and b increase until B is reached when a = a0 and b = b0.

Now

$$ha0 = xn - xs; \; hb0 = yn - ys; \qquad \text{(h is mesh size)}$$

$$ha = x - xs; \; hb = y - ys;$$

The perpendicular distance of P from the true line between A and B is given by

$$s = \frac{(y-ys).(xn-xs) - (yn-ys).(x-xs)}{sqrt\left\{(xn-xs)^2 + (yn-ys)^2\right\}}$$

$$= \frac{h.(ba0 - ab0)}{sqrt\ (a0^2+b0^2)}$$

a0, b0 and h are constant so that

$$s = kk\ (ba0 - ab0) = kk \begin{vmatrix} b & b0 \\ a & a0 \end{vmatrix}$$

where kk is a constant given by

$$kk = \frac{h}{sqrt\ (a0^2 + b0^2)}$$

Now suppose

w0 = static s moves

v0 = static d moves

w = dynamic s moves

v = dynamic d moves

It follows that

$$wO = abs\ (abs\ (aO) \sim abs\ (bO)\ )$$

$$vO = min\ (abs\ (aO),\ abs\ (bO)\ )$$

$$w = abs\ (abs\ (a) \sim abs\ (b)\ )$$

$$v = min\ (abs\ (a),\ abs\ (b)\ )$$

Substituting values for a, b, aO and bO in terms of w, wO, v, vO, the perpendicular distance s from the true line is given by

$$\underline{s = \pm kk\ (w.vO - v.wO)}$$

The sign of s depends on the values assumed by wO, vO, w and v according to the absolute values of a, b, aO and bO. No generality is lost by assuming

$$\underline{s = kk\ (w.vO - v.wO)}$$

since the sign of s determines on which side of the line P lies.

Consider the change in s, δs for both a s move (δss) and a d move (δsd)

$$\delta ss = vO \text{ since w increases by 1}$$

$$\text{and } \delta sd = -wO \text{ since v increases by 1}$$

If the value of s at P is sp then the new value of s after one increment will be ss after a s move and sd after a d move, given by

$$ss = sp + vO$$

$$sd = sp - wO$$

Depending on which is the smaller of ss and sd determines which of the moves should be made.

Thus

if $|\varepsilon s| < |\varepsilon d|$ a s move is made

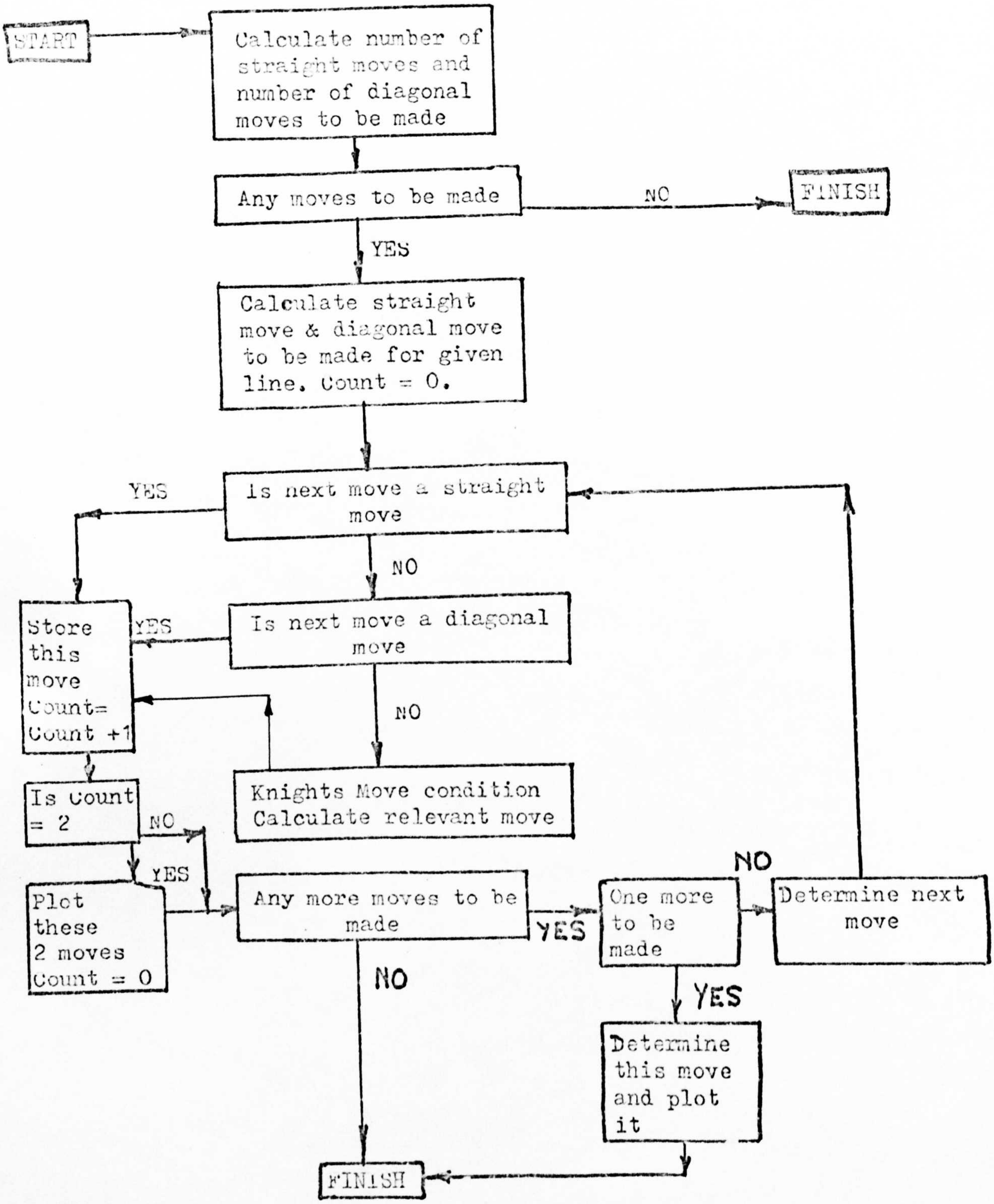
if $|\varepsilon s| > |\varepsilon d|$ a d move is made

The above determination is simple but a problem as to which move is selected exists if $|\varepsilon s| = |\varepsilon d|$ for then the s move is as 'good' as the d move.

Note that it is not sufficient to specify at each equality that a certain move should be made for if the string of moves is computed from B to A the two lines will differ slightly in this region.

It is necessary that different moves be made at this condition in the two different directions in which the line can be plotted. In this case the string of instructions from A to B is exactly the same as the reversed string of instructions from B to A. If the line is drawn in the two directions no divergence will occur which is the required situation.

A situation of the type explained above has been referred to as a Knights Move condition. This is discussed in the following section.


START
Calculate number of straight moves and number of diagonal moves to be made
Any moves to be made
NO
FINISH
YES
Calculate straight move & diagonal move to be made for given line. Count = 0.
Is next move a straight move
YES
NO
Store this move Count= Count +1
YES
Is next move a diagonal move
NO
Knights Move condition Calculate relevant move
Is Count = 2
NO
YES
Plot these 2 moves Count = 0
Any more moves to be made
YES
One more to be made
NO
Determine next move
YES
NO
Determine this move and plot it
FINISH

FLOW DIAGRAM OF PROCEDURE LINE

Diagram 1.6

This is based on the theory which has been explained earlier.

Two moves are plotted at once since two moves correspond to one row of paper tape characters. (See Section 1.4).

1.6.1. Knights Move Condition. In practice a Knights Move condition will lead to a thickening of the plotted line in the area in which the condition occurs. The extent of the thickening will depend on the resolution of the graph plotter. If the resolution is poor, there will be more thickening than if the resolution is good.

Possibly the simplest example of a line in which a Knights Move condition occurs is a line having terminal co-ordinates (0,0) and (2,1) as shown below.

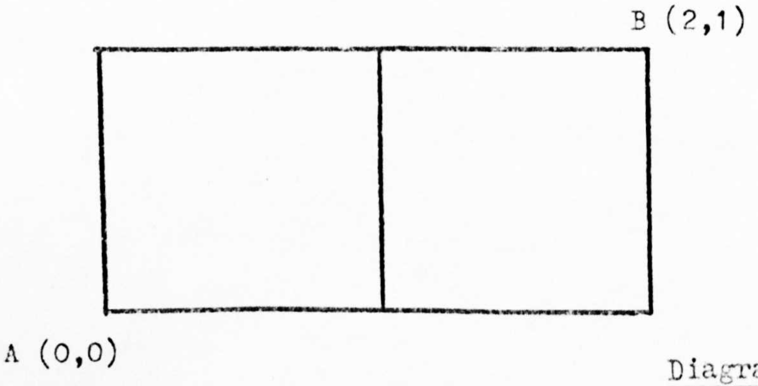


Diagram 1.7

In this case the s move is 3 and the d move 2. The constants defined earlier thus have the following values.

a0 = 2	b0 = 1	h is given by
w0 = 1	v0 = 1	1 in this case
εS = 1	εd = -1	

Initially ε is zero.

If a d move is selected the new ε value εd will be -1 whereas if a s move is selected the new ε value εs will be +1. Since these have the same magnitude a Knights Move condition exists.

Suppose at the Knights Move condition a d move is selected.

Hence in this case ꞩ = -1

The second (and final) move will be a s move since the final value of ꞩ will be zero (ꞩ = ꞩ + 1 = 0) in this case. (Note that ꞩ = -2 if a d move were to be selected).

In the reverse string of moves (from B to A) a diagonal move will be made at the Knights Move condition and then a straight move and thus the two lines will traverse as shown below.

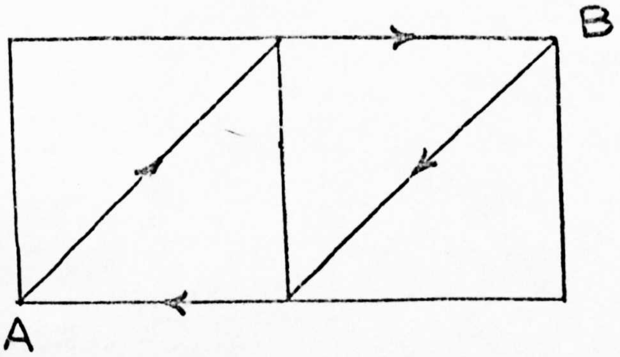
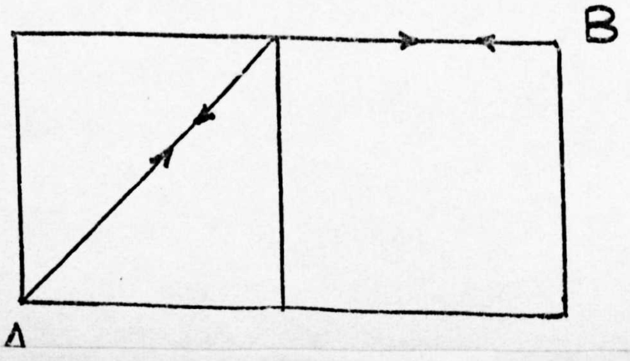


Diagram 1.8

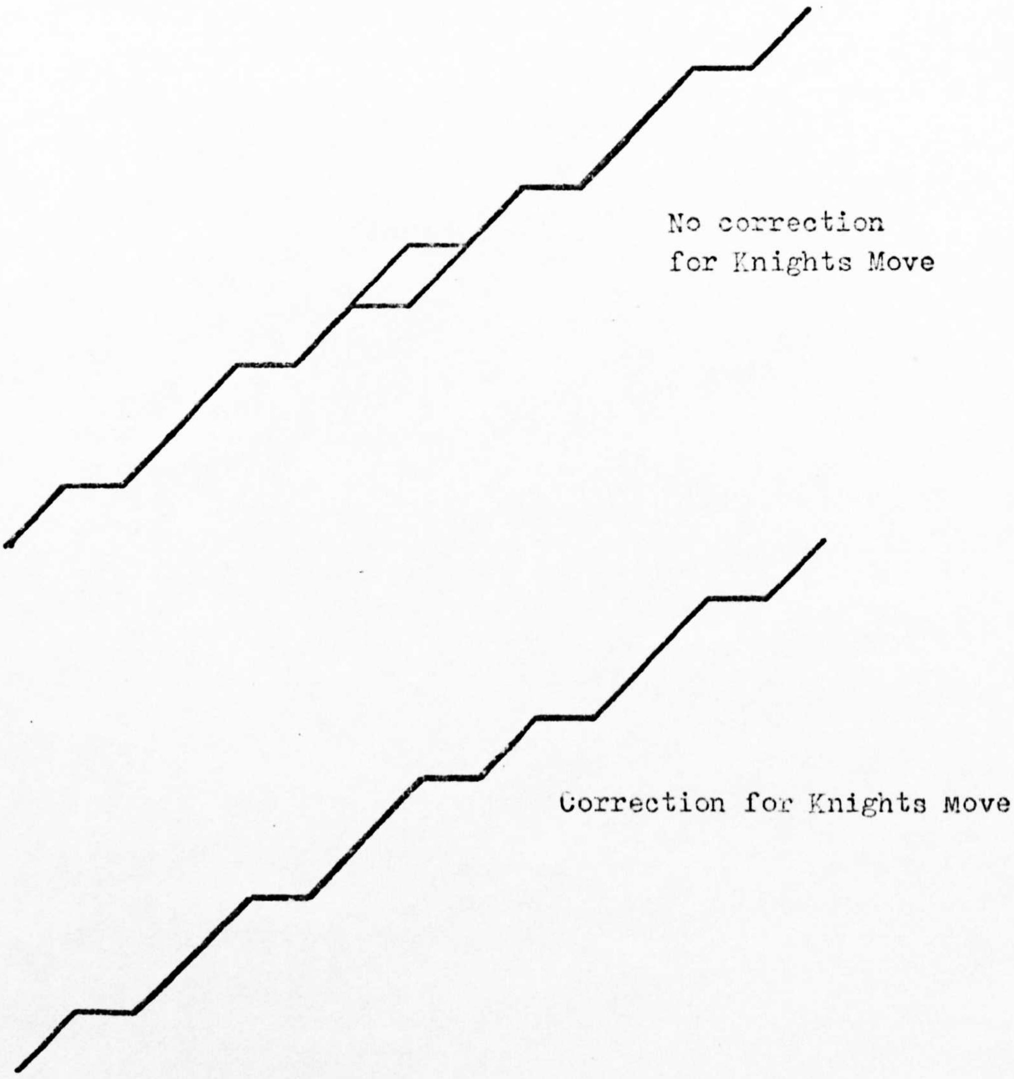
Now suppose that at the Knights Move condition different moves are to be made depending on the direction in which the line is drawn. In the direction A to B suppose the move made at the Knights Move condition is a diagonal (or straight) move and that in the opposite direction this move is a straight (or diagonal) move. In this case the two lines traverse as shown below



No divergence of the two lines occurs and thus no thickening of the line will be seen in practice. Thus it is possible to allow for the Knights Move condition by utilising the direction in which the line is drawn. In procedure line (given in Appendix 1) this is achieved by inspection of boolean B.

It should be noted that the Knights Move condition does not occur for all lines.

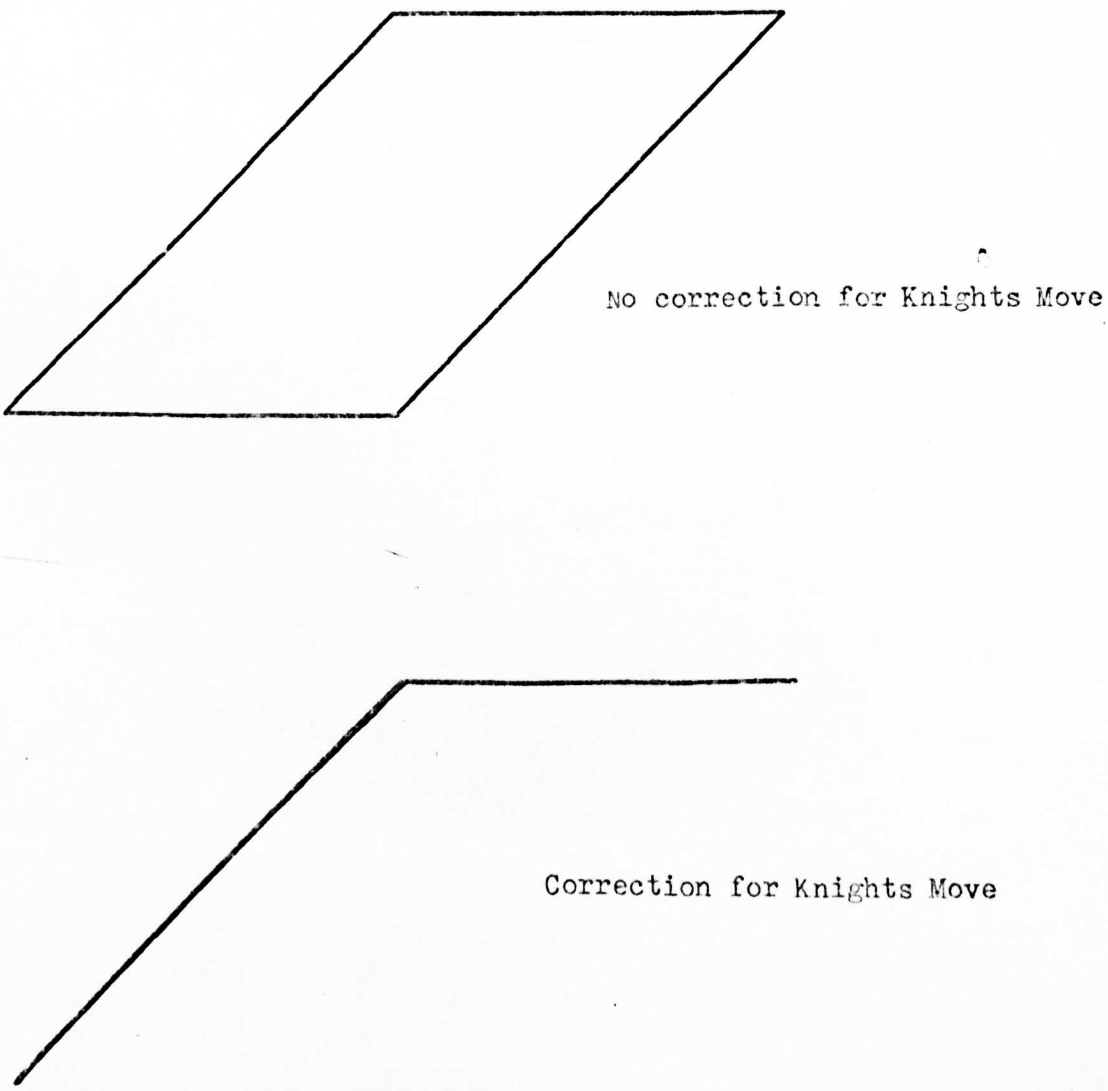
Two examples of lines in which a Knights Move condition occurs (produced on the graph plotter) are shown. In both cases the lines have been drawn in two directions so that any divergence can be clearly seen.



EXAMPLE OF KNIGHTS MOVE.

Line (0,0)to (14,9)

Figure 1.



No correction for Knights Move

Correction for Knights Move

EXAMPLE OF KNIGHTS MOVE.

Figure 2.

Line (0,0) to (2,1)

CHAPTER 2.

PLANE PROJECTIONS

Theory associated with projecting a three dimensional point onto a two dimensional plane of projection.

CHAPTER 2

PLANE PROJECTIONS

2.1. Introduction.

In the field of computer graphics one of the most important topics is a means by which objects in three dimensional space can be represented on a plane in two dimensions. In order to represent three dimensional objects it is first necessary to be able to project the vertices onto a viewing plane, called the plane of projection, which will correspond to the screen of a display or the paper of a graph plotter.

The method by which this is achieved is known as projection. The three dimensional objects to be considered will be polyhedra and various methods of projecting the vertices will be discussed.

This chapter describes a simple method by which vertices in three dimensions can be projected on to a two dimensional plane of projection.

2.2. Projection.

Consider some point V in space to be the viewpoint and consider the vertices of some polyhedron S to be joined to V. Any section of these lines by a plane, known as the plane of projection, is called a projection.

The projection of any vertex P of S in the plane of projection is the intersection of PV with the plane of projection.

This type of projection is sometimes known as axonometric projection and the subject dealing with projections of this kind is known as axonometry.

2.2.1. Perspective Projection.

If the viewpoint V is at a finite distance from S, lines from the vertices of S converge at V. This type of projection is known as PERSPECTIVE and depicts the object as it would appear to an observer at V. However, due to the convergence of the sight lines the true proportions of the object do not appear in the projection.

2.2.2. Orthographic Projection.

If the viewpoint is an infinite distance from S the lines of sight will be parallel and the projection is said to be ORTHOGRAPHIC if the plane of projection is at right angles to the cylinder of lines from S. If the plane of projection is at any other angle to the cylinder of lines the projection is said to be OBLIQUE.

One of the important things pertaining to projections is that lines in space are projected into lines on the projection plane. In projection the lengths and direction of lines may, however, vary.

Isometric orthographic projection is an orthographic projection such that the projection plane makes equal angles with the three cartesian co-ordinate axes. If the projection plane makes equal angles with two axes the orthographic projection is dimetric and if the angles made by the projection plane differ for all three axes the orthographic projection is said to be trimetric.

Note that to define the position of the projection plane for perspective it is necessary to give some line of sight and position the plane so that it is at right angles to this.

There are a number of invariants in any projection the important ones of which are listed below.

(a) Collinearity of lines is preserved

(b) Concurrency of lines is preserved.

DIAGRAM ILLUSTRATING THE PERSPECTIVE PROJECTION OF A CUBE

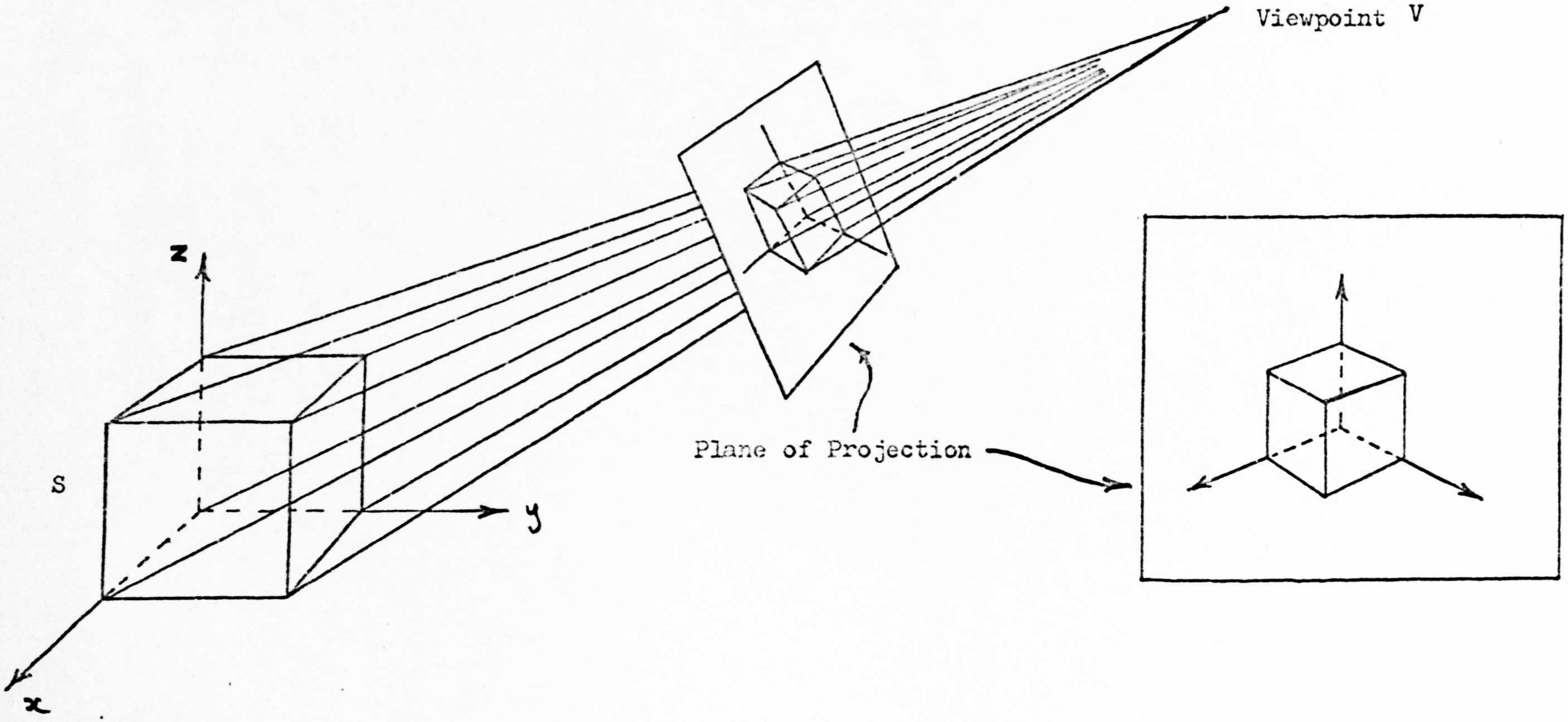


Diagram 2.1

2.3. Mathematical Principles of Perspective Projection.

The information required to project any given vertex P(x,y,z) in three dimensions onto a plane of projection in two dimensions consists of a number of variables.

The viewpoint V from which the object is seen is given as a point in cartesian co-ordinates (xv,yv,zv) and the line of sight is given as the angles made with the three cartesian axes and are given by α, B and γ. A seventh variable d fixes the distance of the plane of projection from the viewpoint.

Now suppose that the viewpoint is positioned at the origin. It will thus be necessary to translate the original vertex P (x,y,z) to a new position Pt (xt,yt,zt) given by

$$\begin{bmatrix} xt \\ yt \\ zt \end{bmatrix} = \begin{bmatrix} x - xv \\ y - yv \\ z - zv \end{bmatrix} \qquad \underline{\hspace{4cm}}(1)$$

It is now required to rotate the translated vertex Pt until the z axis coincides with the given line of sight. This can be achieved by rotating Pt by α about the x axis, B about the y axis and γ about the z axis.

The three rotation matrices to do this are as follows:

$$Rx = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \qquad \begin{array}{l} \text{rotation by } \alpha \\ \text{about x axis} \end{array} \underline{\hspace{2cm}}(2)$$

$$Ry = \begin{bmatrix} \cos B & 0 & \sin B \\ 0 & 1 & 0 \\ -\sin B & 0 & \cos B \end{bmatrix} \qquad \begin{array}{l} \text{rotation by } B \\ \text{about y axis} \end{array} \underline{\hspace{2cm}}(3)$$

$$Rz = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{array}{l} \text{rotation by } \gamma \\ \text{about z axis} \end{array} \underline{\hspace{2cm}}(4)$$

These three rotations can be combined to give one matrix of rotation R given by

$$R = Rx.Ry.Rz$$

so that

$$R = \begin{bmatrix} \cos\gamma\cos\beta & -\sin\gamma\cos\beta & \sin\beta \\ \sin\alpha\sin\beta\cos\gamma & \cos\gamma\cos\alpha & \\ +\cos\alpha\sin\gamma & -\sin\alpha\sin\beta\sin\gamma & -\sin\alpha\cos\beta \\ \sin\beta\cos\alpha\cos\gamma & \sin\alpha\cos\gamma & \\ +\sin\gamma\sin\alpha & -\cos\alpha\sin\beta\cos\gamma & \cos\alpha\cos\beta \end{bmatrix} \qquad \_\_\_\_\_\_\_\_(5)$$

The rotated vertex Pr (xr, yr, zr) is thus given by

$$\begin{bmatrix} xr \\ yr \\ zr \end{bmatrix} = R . \begin{bmatrix} xt \\ yt \\ zt \end{bmatrix} \qquad \_\_\_\_\_\_\_\_(6)$$

Now consider the plane of projection to be parallel to the xy plane at a distance d from the origin as shown in Diagram 2.2.

The determination of the projected point Pp (xp, yp, zp) is then easily calculated from the geometry of similar triangles. The projected point Pp (xp, yp, zp) has each of its co-ordinates in a constant ratio k to those of Pt where k is given by

$$k = \frac{d}{z_r} \qquad \_\_\_\_\_\_\_\_(7)$$

so that xp and yp are given by

$$xp = k.xr \qquad \_\_\_\_\_\_\_\_(8)$$

$$yp = k.yr \qquad \_\_\_\_\_\_\_\_(9)$$

The projected vertex Pp is thus

$$\begin{bmatrix} xp \\ yp \\ zp \end{bmatrix} = k. \begin{bmatrix} xr \\ yr \\ zr \end{bmatrix} \qquad \_\_\_\_\_\_\_\_(10)$$

The co-ordinate axes on the plane of projection are taken as the x y axes so that the projected point on the plane of projection, corresponding to the original spatial vertex P, is given by (xp, yp).

For each spatial vertex it is therefore possible by multiplying the rotated coordinates by a constant ratio which depends on the value of zr and d, to project the vertex so that the z coordinates are the same and therefore all the projected vertices lie in a given plane, the plane of projection.

It is fairly easy to scale the object on the plane of projection so that the value of d is superfluous in that the variation of it simply alters the scaling on the projection plane.

The algorithm for perspective projection is given in Appendix 2.

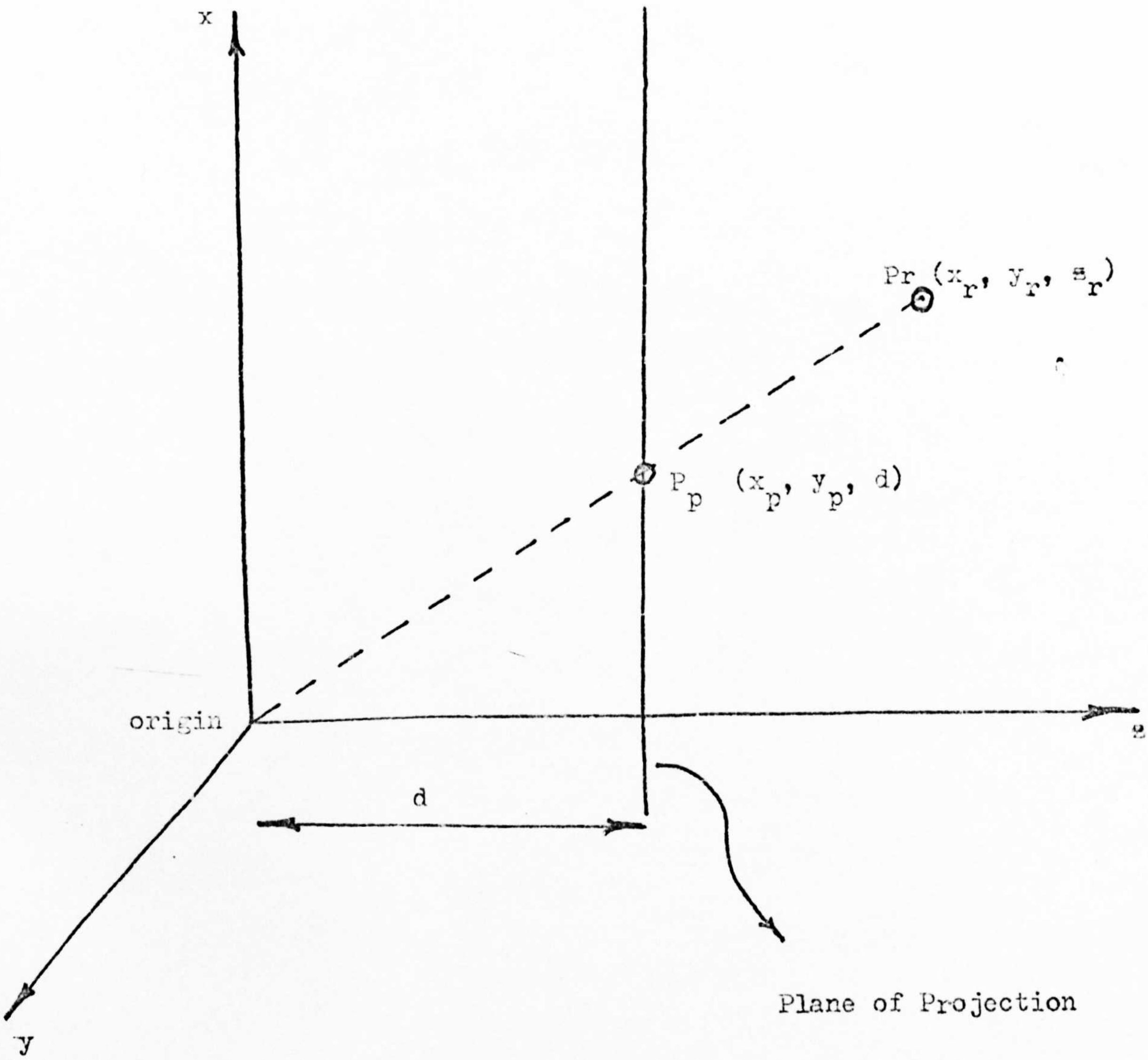


DIAGRAM SHOWING THE PROJECTION OF THE ROTATED VERTEX P_r ONTO THE PLANE OF PROJECTION.

Diagram 2.2

2.4. Wire Frame Drawings.

As has been seen in a preceding section, lines are projected into lines, and it is only because of this that it is possible to represent objects by the simple wire frame drawing.

Possibly the first researcher to realise this simple fact in the computer representation of objects was T. Johnson[12] at the Lincoln Laboratories in Massachusetts. About a year or so later in 1964 Pucket[18] produced a fairly good paper on this development, for the National Aeronautics and Space Administration (N.A.S.A.)

The method is simple since it is only necessary to store data corresponding to the co-ordinates of the vertices of the object and information concerning the formation of the line segments. Each of the spatial vertices is then projected onto the projection plane and the corresponding extreme points of the line segments on the plane of projection are connected by straight lines.

Any object can be represented in some way by straight lines in space so that the wire frame drawing has a wide range of application.

An algorithm has been written to produce simple wire frame drawings and examples of the possibilities of the method are shown. One of the great advantages of the algorithm presented is that since the formation of the line segments for a given object does not change it is only necessary to organise the data associated with a line drawing once. Other views of the object can be quickly generated and all that is required is the new projection of the spatial vertices from the new viewpoint. The rest of the information remains unchanged and the line drawing can be connected as in previous cases.

The only disadvantage of this is that the pen up distances under certain conditions may become very large and no effort is made to check for the coincidence of points on the projection plane, which could lead to a reduction in the calculation time.

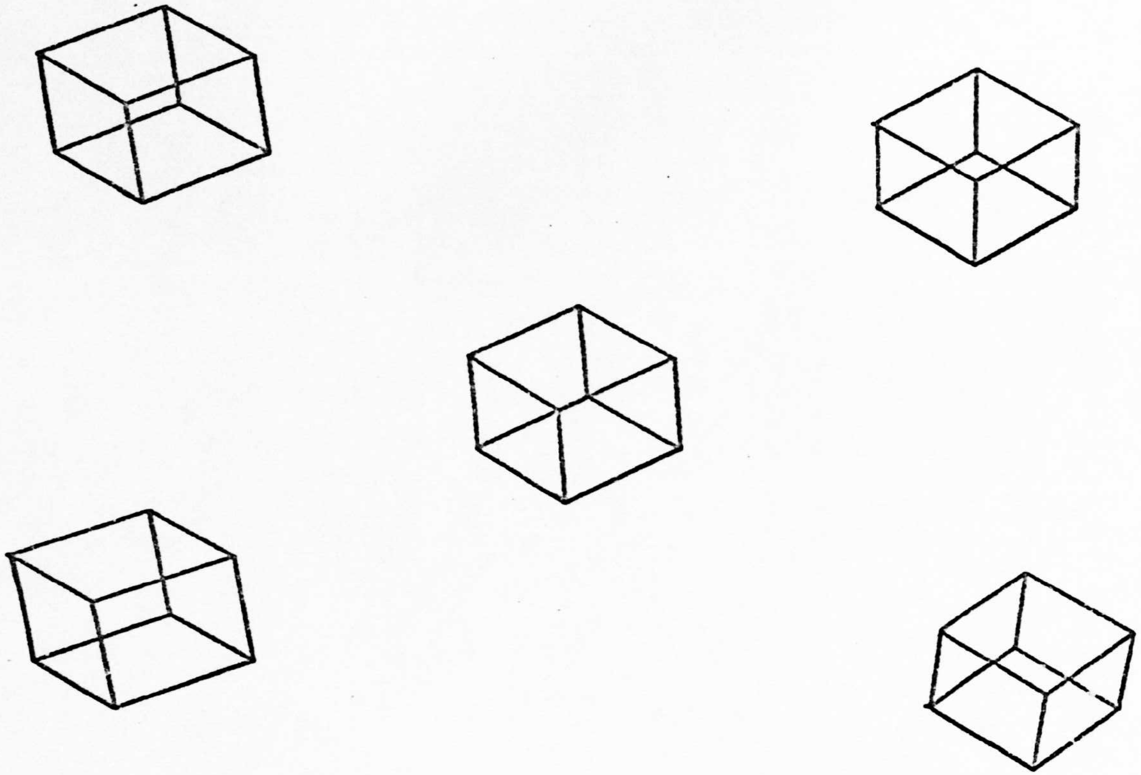
However, since the method is so easily programmed to deal with varying viewpoints it was considered that nothing much was lost since the advantage of wire frame drawings is that it affords a method by which any object can be viewed from a number of viewpoints.

The wire frame drawing is quickly and easily produced since no effort is made to locate which of the lines of the object are hidden to the viewpoint. In reality the hidden line problem, as it has been called is solved because it is not possible to see surfaces which have other ~~pen~~ opaque surfaces between it and the eye.

The hidden line problem has received much attention during the last few years. Methods of locating the visible and hidden edges of polyhedra have been developed by the author and are presented later in the thesis.

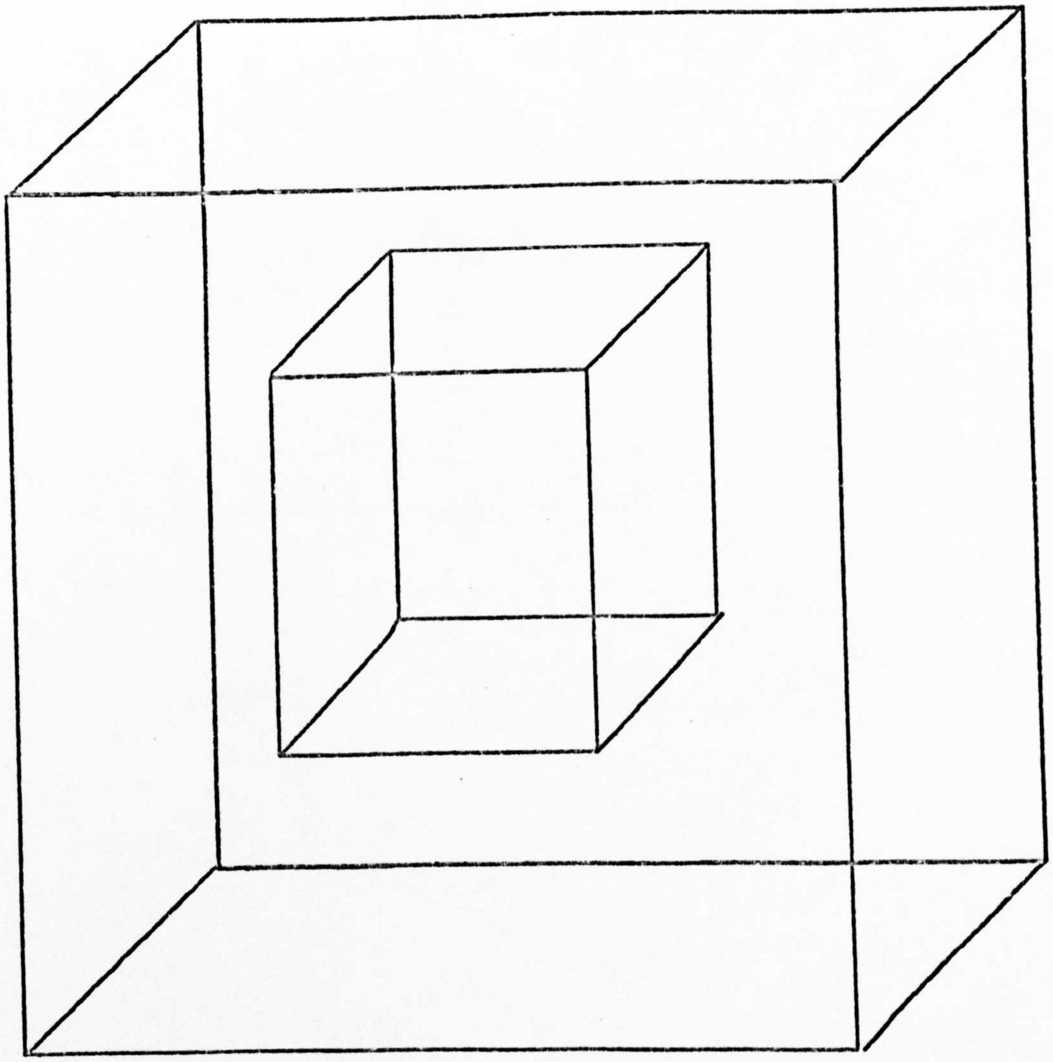
The following chapter is concerned with a method by which convex polyhedra can be represented by computer while Chapter 4 deals with the rather more difficult problem of the non-convex polyhedron.

In both cases practical examples produced on the graph plotter, are included.



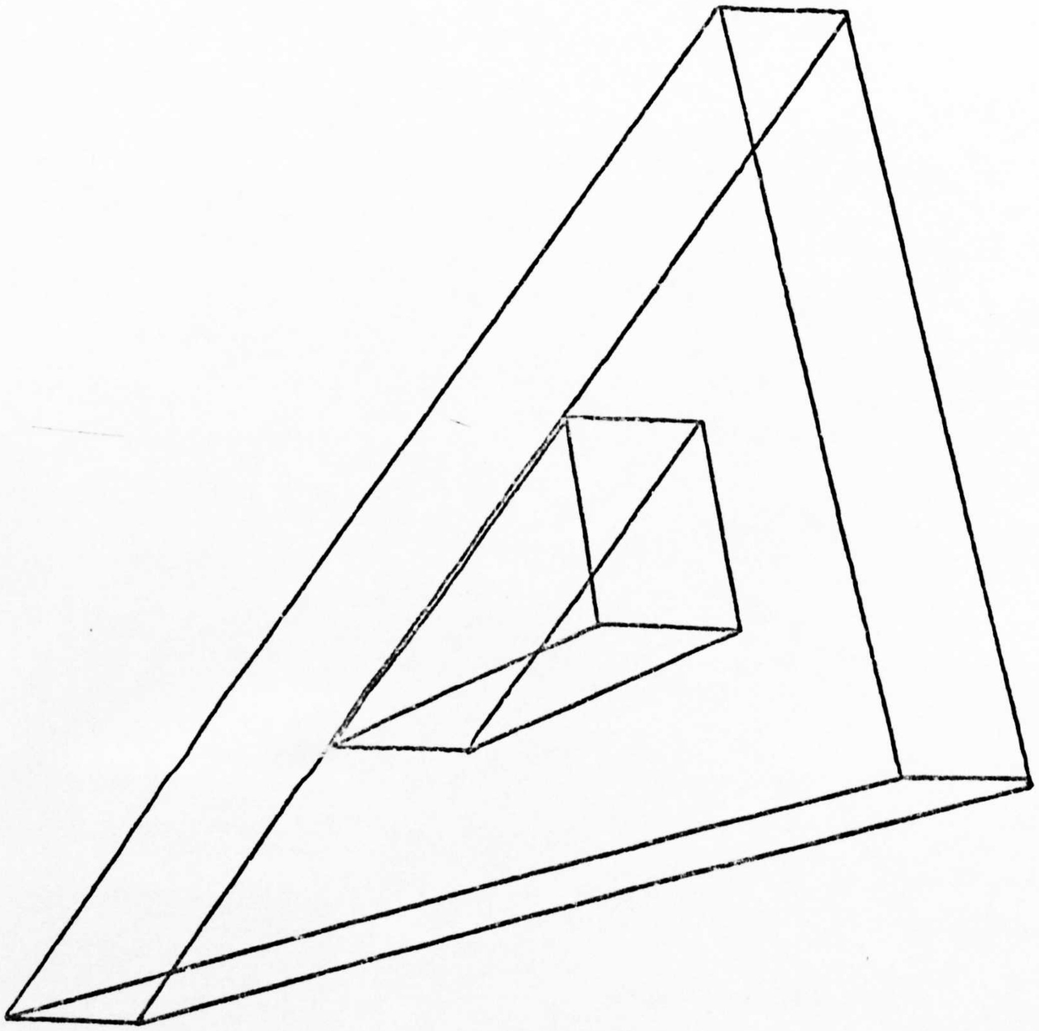
EXAMPLE OF A ROTATING CUBE.

Figure 3.



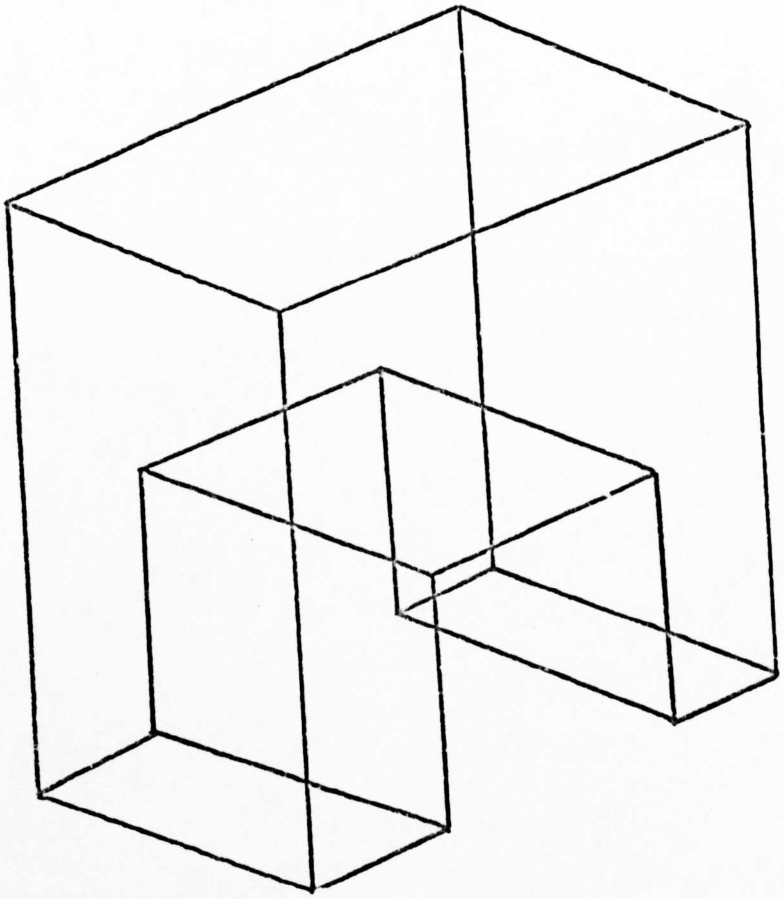
NON-CONVEX POLYHEDRON.

Figure 4.



WIRE-FRAME DRAWING

Figure 5.



WIRE-FRAME DRAWING

Figure 6.

EXAMPLE SHOWING THE EFFECT OF PERSPECTIVE.

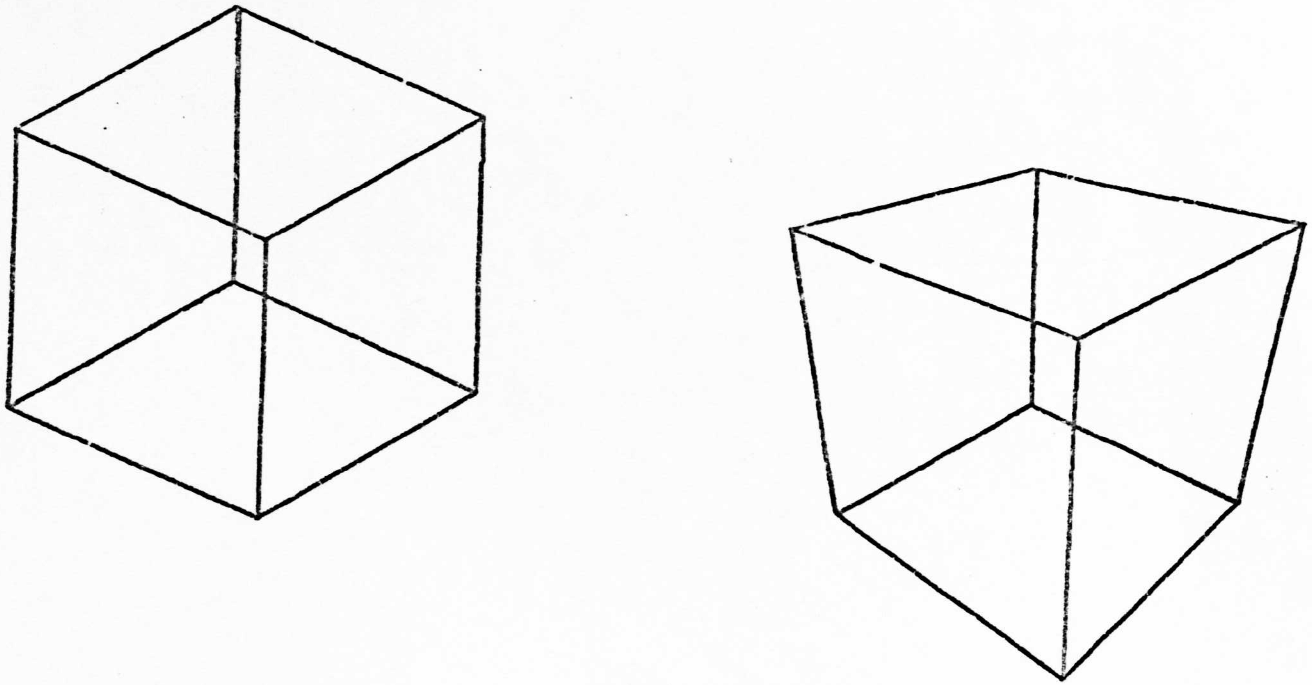


Figure 7.

CHAPTER 3

COMPUTER REPRESENTATION OF CONVEX POLYHEDRA

Theory associated with an algorithm to determine both the visible and hidden edges of any given convex polyhedron.

CHAPTER 3

Computer Representation of Convex Polyhedra.

3.1. Introduction.

In 1966, Cole published a short paper in the Computer Journal on the representation of convex polyhedra from minimal information.

The method used enabled both the hidden plane faces and the visible plane faces of the polyhedra to be found from input data consisting simply of the co-ordinates of the vertices of the polyhedra.

This algorithm has many serious disadvantages however. Each plane face of the polyhedron can be located more than once so that it was necessary, as each plane face was found, to check it against a list of already located faces to ascertain whether the face had been found previously. With polyhedra consisting of a large number of faces this can become a very serious disadvantage.

As each plane face was located, it could be rejected not only if it had already been found, but also if it divided the existing perimeter list of points into two disjoint pieces. These two cases of rejection necessitated a large amount of checking for each plane face of the polyhedron.

Although the algorithm is rather sketchy, the general idea has been utilised in this thesis to produce an algorithm which has many complementary features. By introducing the concept of 'valid third vertices' it has been possible to produce an algorithm which locates each plane face of the polyhedron once and once only. The current perimeter list of line segments defines the existing perimeter list in terms of segments rather than points (as Cole used). By this means a convenient method of locating each edge of the polyhedron without the need to check existing lists of already located edges has been developed. In addition, a method has been devised to alter the current perimeter list such that the list is allowed to split into any number of disjoint pieces.

A method of reducing the number of vertices which can lie on plane faces still to be located has been developed. This is particularly beneficial for polyhedra with a large number of vertices.

The only part of the algorithm to be described which is taken from Cole's algorithm is the general method of locating the initial perimeter list of line segments. All the other concepts which are introduced are the original work of the author.

It should be noted that the method to be described inherently utilises the fact that for a given convex polyhedron any given plane face is either completely hidden, or completely visible to the viewpoint. A given edge of the polyhedron is also either completely hidden or completely visible.

It is because of these considerations that the computer representation of convex polyhedra is simpler than that of the general polyhedron which may have both faces and edges only partially hidden to the viewpoint.

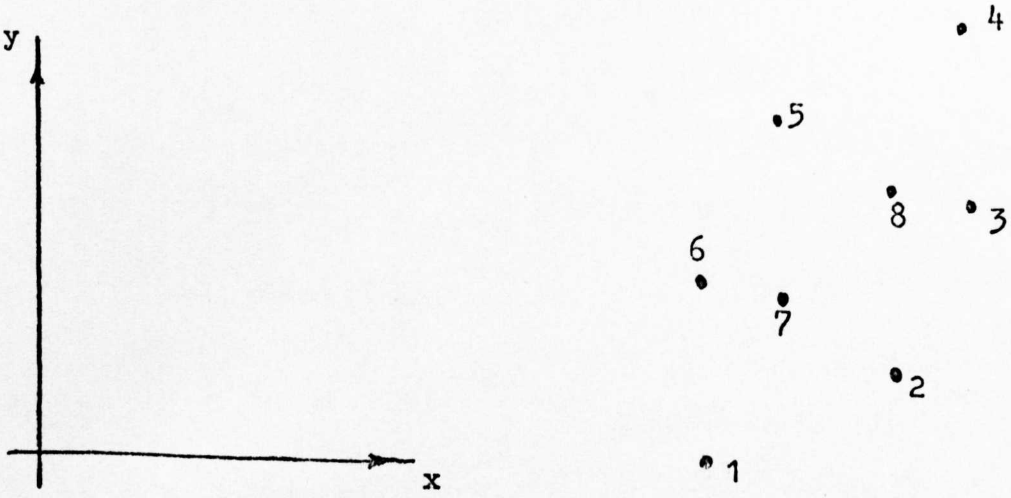
A completely new approach is needed when considering the general polyhedron. An algorithm has been developed which determines those portions of each edge which are visible to the viewpoint. This work is presented in the following chapter. (Chapter 4.)

3.2. General Method.

The input data which is associated with the convex polyhedron consists of the cartesian co-ordinates of the n spatial vertices of the polyhedron. Suppose that some viewpoint V (xv, yv, zv), which lies outside the polyhedron and a line of sight from V, defined by the angles it makes with the co-ordinate axes, are also supplied as data. The n spatial vertices of the polyhedron can now be projected onto a plane of projection at right angles to the line of sight.

Suppose, for clarity, that edges of the polyhedron are referred to as segments on the projection plane and that spatial vertices have corresponding points on the projection plane. In addition, plane faces of the polyhedron will have corresponding convex polygons on the projection plane.

As an example suppose that a cube is to be represented and suppose that the 8 spatial vertices have been projected onto the projection plane as shown below:



Cartesian Co-ordinate Axes on the Plane of Projection

Diagram 3.1

These points will correspond to a completed drawing as shown below in Diagram 3.2.

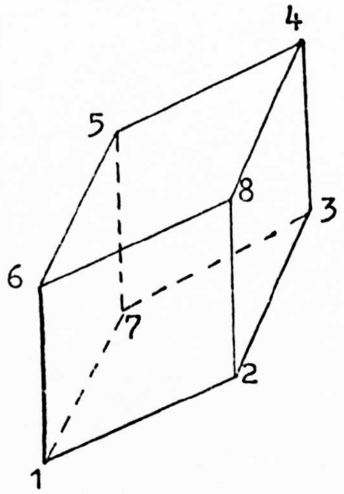
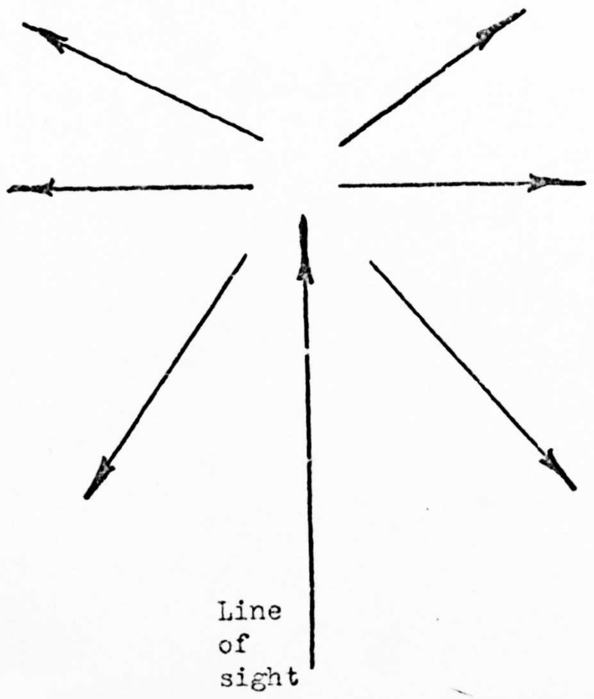


Diagram 3.2

An edge of the polyhedron will be called an initial perimeter edge if the two plane faces associated with it are a hidden face and a visible face as seen from the viewpoint. Thus, in Diagram 3.2, the initial perimeter edges will be 1-2, 2-3, 3-4, 4-5, 5-6 and 6-1.

The first step of the algorithm is to locate the initial perimeter edges of the convex polyhedron.

3.2.1. Location of Initial Perimeter Edges. If the vertices of each plane face, ordered in some direction (either clockwise or anticlockwise), were supplied as data, it would be possible to determine the outward normal vectors to each face. The angles made by each of these outward normals with the line of sight could then be calculated. If this angle were greater than 90° for a given face then the face would be visible from the viewpoint, otherwise the face would be hidden. The special case, when the angle is equal to 90° will be regarded as a 'hidden' case.



Angles made by outward normal vectors with line of sight less or equal to 90° - hidden faces.

Faces associated with these outward normal vectors are visible. Angle made with line of sight are greater than 90°.

Diagram 3.3

In Diagram 3.2 the outward normal vector to face (1,2,8,6) makes an angle of about 180° to the line of sight and is therefore a visible face. The face (1,7,5,6) has an outward normal vector which makes an angle of about 60° to the line of sight and is therefore a hidden face.

After calculating these angles it would be a fairly simple matter to locate those edges associated with both visible and hidden faces which would be the initial perimeter edges. However, the only information relating to the convex polyhedron to be assumed in this discussion is the cartesian co-ordinates of the vertices so that it is necessary to develop another method of locating these edges.

It should be noted at this stage that the spatial vertices of the convex polyhedron have been projected onto the plane of projection as shown in Diagram 3.1. No edges of the convex polyhedron have been supplied as data and reference will only be made to Diagram 3.2 to explain the algorithm more clearly.

Since the polyhedron is convex the initial perimeter edges will form an enclosing convex polygon on the projection plane which will enclose all the points of the polyhedron.

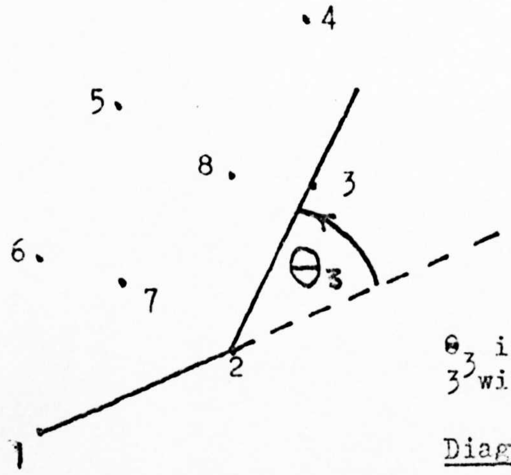
In the example shown in Diagram 3.2 the enclosing convex polygon is (1,2,3,4,5,6) so that it is required to locate these points from the information implicitly contained in Diagram 3.1. An initial point on the enclosing polygon can be found as follows :

Find the point on the plane of projection which has the least y co-ordinate. If there is more than one point satisfying this condition then choose that point among them which has the least x co-ordinate. If there is still more than one point satisfying these conditions then choose that point whose corresponding spatial vertex is nearest to the viewpoint. This point has its corresponding spatial vertex visible to the viewpoint. In Diag. 3.1 the initial point will be 1. The next point to be located on the enclosing convex polygon will be that point which when joined to the initial point, makes the least angle with the x axis on the projection plane. This will correspond to point 2 in Diagram 3.1.

The first two points located on the enclosing convex polygon are thus 1 and 2.

The next point to be located will be that point which when connected to the last point obtained, makes the least angle (measured in an anti-clockwise direction) with the previous line segment located (1-2). Thus from among the possible points 3,4,5,6,7 and 8, point 3 is selected.

The first three points to be located on the enclosing convex polygon are thus 1,2 and 3.



θ_3 is angle made by point 3 with line segment 1-2.

Diagram 3.4

The method of locating the next points on the enclosing convex polygon can be repeated with 2-3 now the previous line segment. The next point located in Diagram 3.1 will be 4. The terminating condition is when the initial point (1 in this case) makes a smaller angle with the previous line segment than any other point. Thus, in Diagram 3.1, 7 will be selected as making the least angle with segment 5-6 but point 1 makes a smaller angle. 6-1 is thus the final line segment on the enclosing convex polygon. The enclosing convex polygon is thus (1,2,3,4,5,6). In the procedure to locate the initial perimeter edges, given in appendix 3, the number of points to be tested is gradually reduced since as each point is located on the enclosing convex polygon it will obviously not occur again and so can be deleted from the available list of points.

3.2.2. Faces Parallel to the Line of Sight. A difficulty exists in locating the next point on the enclosing convex polygon if more than one point makes the least angle with the previously located segment. This corresponds to the case when a plane face of the polyhedron has an outward normal vector making an angle of 90° with the line of sight. The plane face is then parallel to the line of sight.

Consider the following example of a cube:

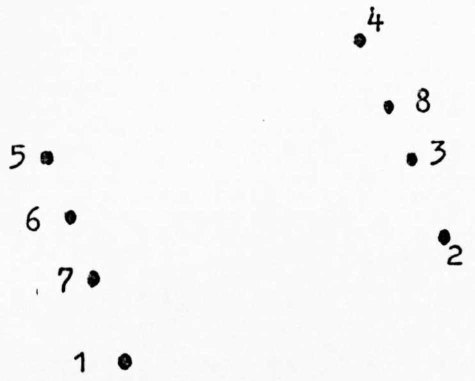


Diagram 3.5

and suppose that these points correspond to a final drawing as shown below:

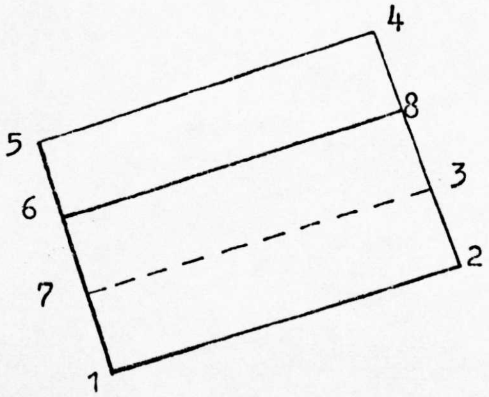


Diagram 3.6

The cube corresponds to that given in Diagrams3.1 and 3.2 rotated slightly so that faces (2,8,4,3) and (1,6,5,7) are parallel to the line of sight.

Points 1 and 2 will be the first two points located on the enclosing convex polygon.

Points 3,4 and 8 make equal least angles with the line segment 1-2. It is not sufficient to locate points 3, 4 and 8 and sort them according to their distances (on the projection plane) from the last point located. This would give the initial points of the enclosing convex polygon as (1,2,3,8,4). Segment 3-8 does not correspond to an edge of the polyhedron. Consecutive points on the enclosing convex polygon must correspond to initial perimeter edges of the polyhedron, although we only deduce these edges from the co-ordinatesof the spatial vertices and their projected images.

In the example given in Diagram 3.5 the points 3, 4 and 8 have been located. Select that point which lies furthest from the last point located. This will correspond to point 4 since this lies furthest from point 2 in the plane of projection. If there is more than one point satisfying this condition select that point from among them whose corresponding spatial vertex lies nearest to the viewpoint. This point has its corresponding vertex visible and will lie on the enclosing convex polygon.

Now it is possible that points which lie between points 2 and 4 on the plane of projection also lie on the enclosing convex polygon. The points must have their corresponding spatial vertices visible to the viewpoint.

Points will lie on the enclosing convex polygon if their corresponding vertices lie on the <u>viewpoint side</u> of the plane perpendicular to the line of sight which passes through the vertices 2 and 4. Thus,. in the example given, point 8 has its corresponding spatial vertex lying on the viewpoint side of this plane while 3 does not. Point 8 thus lies on the enclosing convex polygon and so the initial points on the enclosing convex polygon in Diag.3.5 are 1, 2, 8,4 and these correspond to the initial perimeter edges 1-2, 2-8, 8-4.

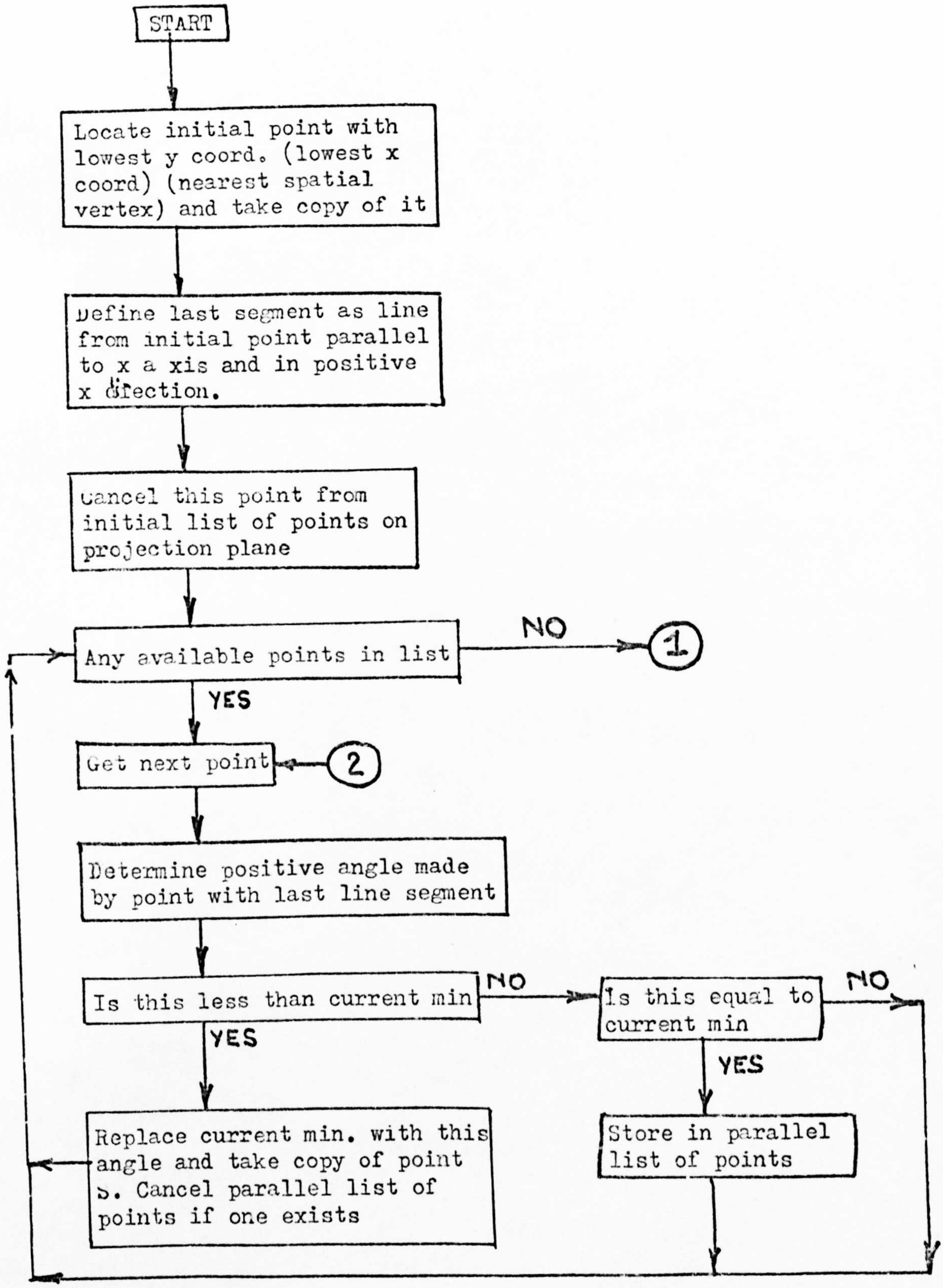
If there is more than one point between 2 and 4 which lies on the enclosing polygon, then these are sorted according to their distances (on the projection plane) from point 2.

Note from Diagram 3.6 that of the initial perimeter segments located, 2-8 and 8-4 are each associated with a visible face and a hidden face.

2-8 is associated with the hidden face (2,8,4,3) and the visible face (2,8,6,1).

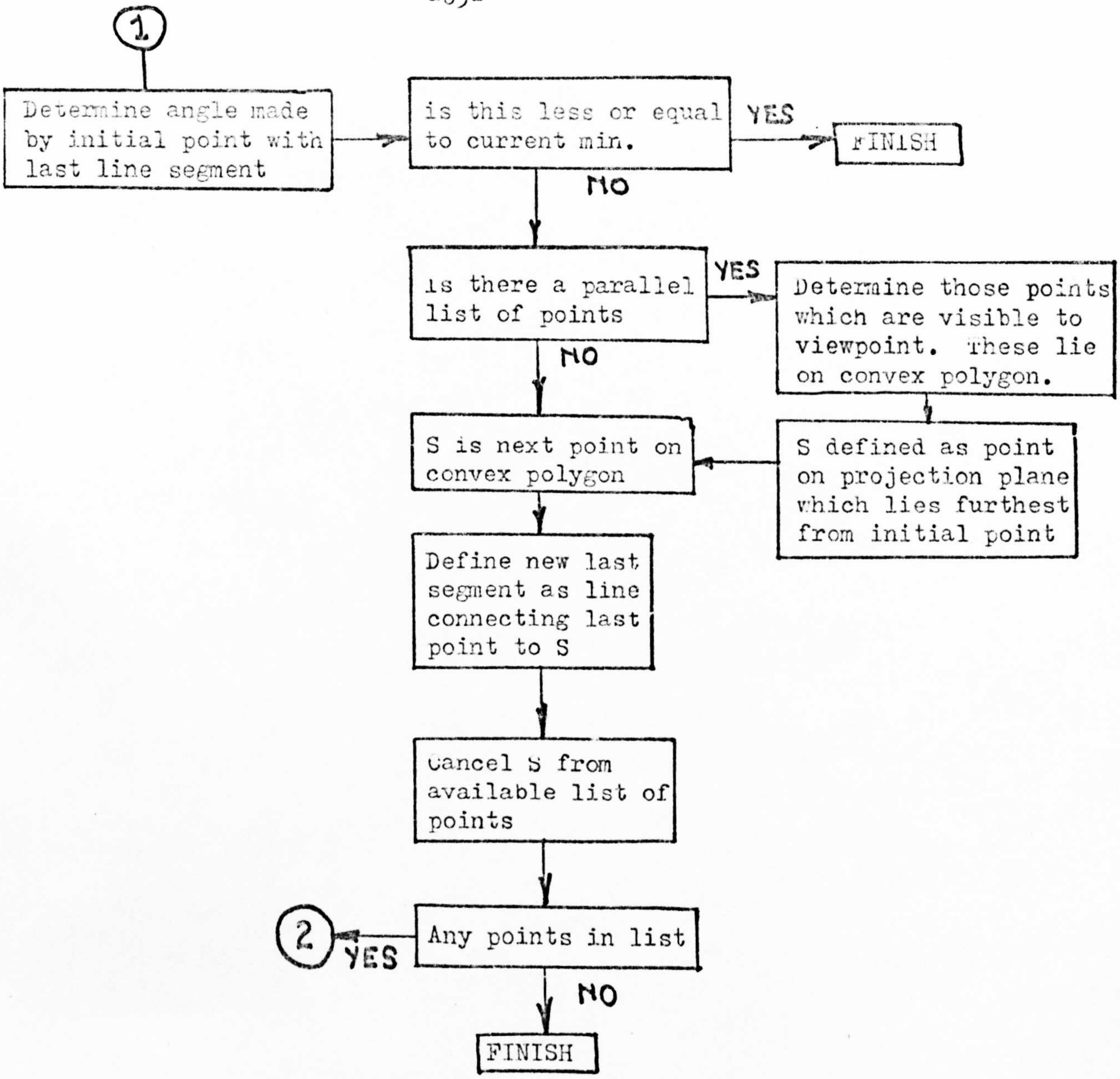
8-4 is associated with the hidden face (2,8,4,3) and the visible face (8,4,5,6).

Both 2-8 and 8-4 thus satisfy the definition of initial perimeter edges given earlier.


START
Locate initial point with lowest y coord. (lowest x coord) (nearest spatial vertex) and take copy of it
Define last segment as line from initial point parallel to x a xis and in positive x direction.
Cancel this point from initial list of points on projection plane
Any available points in list
NO
1
YES
Get next point
2
Determine positive angle made by point with last line segment
Is this less than current min
NO
Is this equal to current min
NO
YES
YES
Replace current min. with this angle and take copy of point s. Cancel parallel list of points if one exists
Store in parallel list of points

FLOW DIAGRAM OF METHOD USED TO LOCATE POINTS OF INITIAL CONVEX POLYGON

Diagram 3.7


1
Determine angle made by initial point with last line segment
is this less or equal to current min.
YES
FINISH
NO
Is there a parallel list of points
YES
Determine those points which are visible to viewpoint. These lie on convex polygon.
NO
S is next point on convex polygon
S defined as point on projection plane which lies furthest from initial point
Define new last segment as line connecting last point to S
Cancel S from available list of points
Any points in list
2
YES
NO
FINISH

FLOW DIAGRAM OF METHOD USED TO LOCATE POINTS OF INITIAL CONVEX POLYGON (Cont'd)

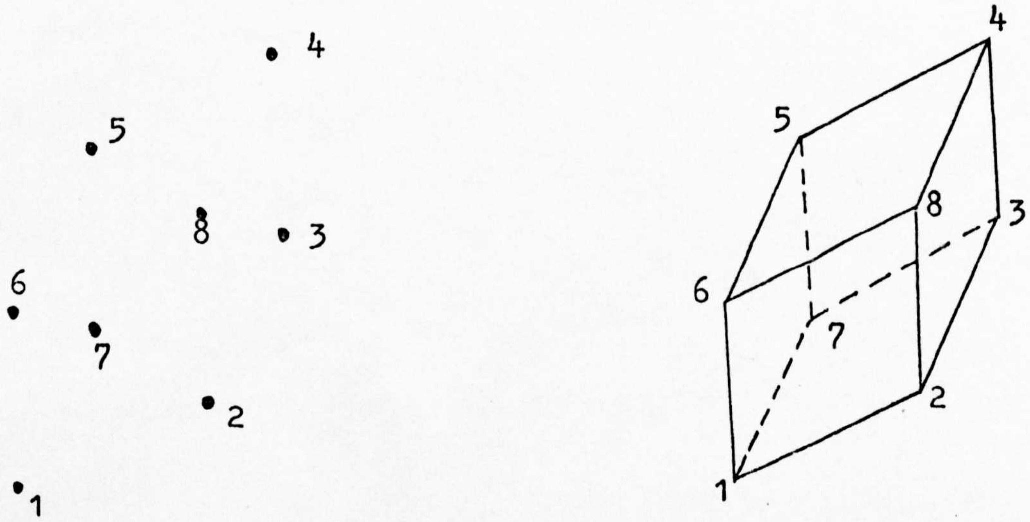
Diagram 3.7

3.2.3. Current Perimeter List of Line Segments. The segments corresponding to the initial perimeter edges will form an initial entry to a current perimeter list of line segments. The line segments contained in this list are to have corresponding edges which are associated with just one visible plane face yet to be located.

The plane polygons associated with each of these segments will be found and the current perimeter list will be altered accordingly so that the above definition of the line segments will always be true. Note that the definition holds true initially since each initial perimeter edge (segment) is associated with one visible plane face (polygon) yet to be located. A list of line segments to be subsequently drawn and called the draw list will be kept and as each plane face is located, visible edges associated with it will be added to this list. Since each of the initial perimeter edges is visible they will form the initial draw list.

Consider again the cube given in Diagrams 3.1 and. 3.2.

Diagram 3.8



The points on the left of the diagram correspond to a final line drawing shown on the right.

The initial perimeter edges are 1-2, 2-3, 3-4, 4-5, 5-6 and 6-1.

3.2.4. Location of Visible Plane Faces. It is required to locate the visible plane face (1, 2, 8, 6) which is associated with the first segment 1-2 in the current perimeter list. Suppose the plane defined by the two vertices of the first current perimeter line segment and another vertex 3, taken from a vertices list of valid third vertices (to be discussed in a later section) is given by

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

where the constants a,b,c and d can be determined from the determinant

$$\begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x & y & z & 1 \end{vmatrix} = 0$$

where (x_1, y_1, z_1) refers to the cartesian co-ordinates of vertex 1. The visible plane face associated with edge 1-2 will be such that none of the vertices of the polyhedron will lie on the viewpoint side of it. It is therefore required to obtain a third vertex, from the vertices list, such that the vertices 1, 2 and the third vertex define a plane which satisfies this condition. The vertices lying on this plane will define the visible plane face of the polyhedron. The corresponding points on the projection plane will correspond to a convex polygon and the edges of the face can thus be obtained in a similar manner as in locating the enclosing convex polygon.

Initially the vertices list will contain all the vertices of the convex polyhedron. Consider that a 'nearest' plane is already defined by the constants a, b, c and d and that another vertex L is a candidate for the third vertex.

In the projection of the vertices the viewpoint V was translated to the origin so that, with the viewpoint substituted in the equation of the plane, the equation has a value given by the value of d.

Now suppose the vertex L has its cartesian co-ordinates substituted in the equation of the plane. The value s of the equation of the plane is given by

$$s = ax_L + by_L + cz_L + d$$

where $(x_L\ y_L\ z_L)$ are the cartesian co-ordinates of the vertex L. If $s \times d < 0$ the vertex L lies on the opposite side of the plane to the viewpoint. If $s \times d > 0$ the vertex L lies on the same side of the plane as the viewpoint. In this case the currently defined plane must be replaced by the plane defined by the vertices 1, 2 and L which becomes the new nearest plane. If $s \times d = 0$ vertex L lies on the currently defined plane and may be stored as being a vertex lying on the nearest plane located so far. In practice this corresponds to an absolute value of s x d being less than some small value δ.

After every vertex in the vertices list has been tested in this way the nearest plane will define the visible face associated with the edge 1-2. The vertices which have been stored (corresponding to $s \times d = 0$ since the last nearest plane was replaced) will be the other vertices lying on this visible face.

As explained earlier the corresponding points can now be sorted in the same way as in locating the initial perimeter edges. For example, the visible plane face associated with segment 1 - 2 is (1,2,8,6) which has a corresponding convex polygon on the plane of projection as shown below.

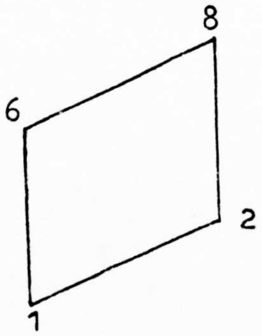


Diagram 3.9

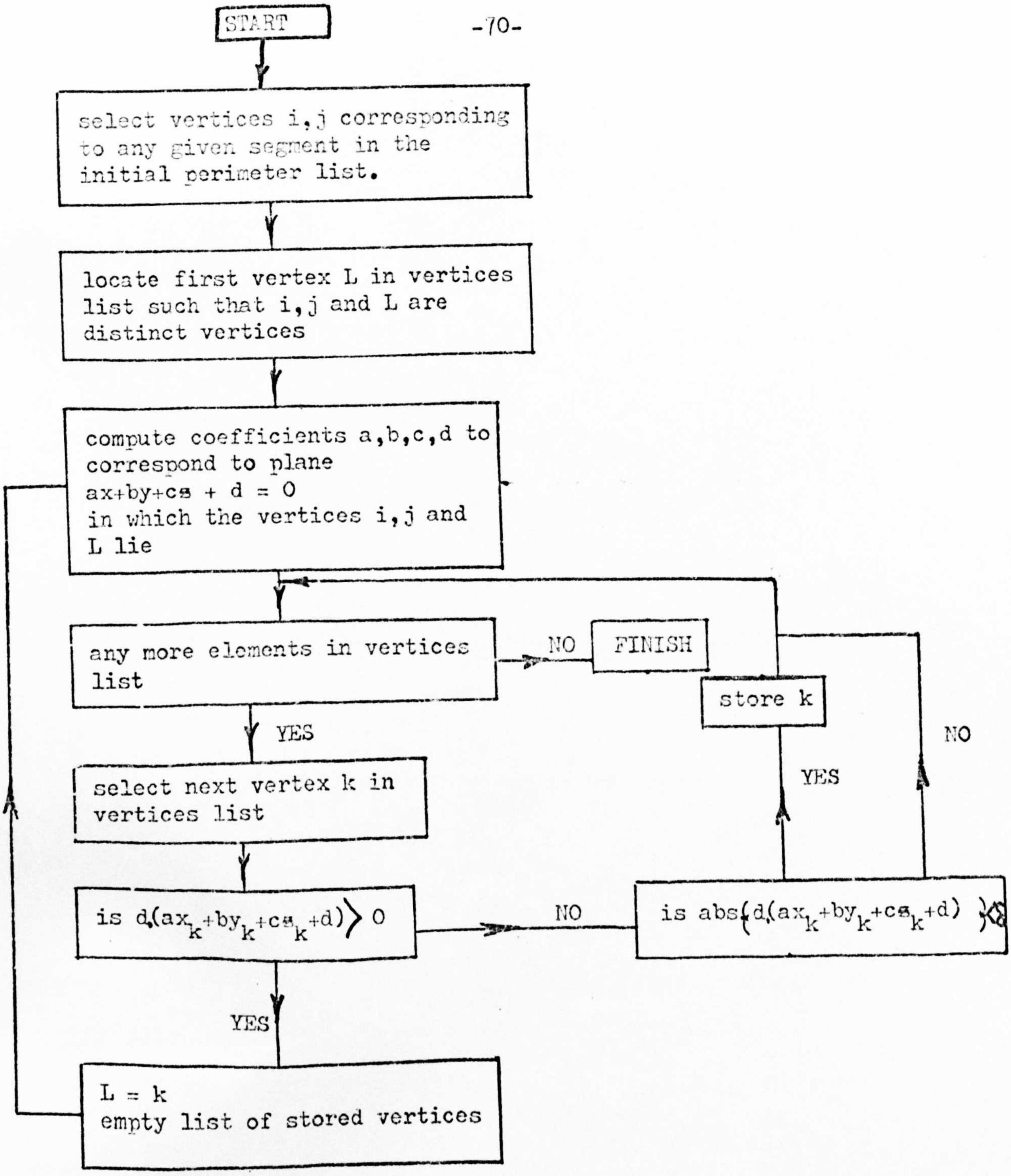
The points on this polygon will be sorted into the order (1,2,8,6). The corresponding edges of the plane face are thus 1-2, 2-8, 8-6 and 6-1.

The visible edges which lie on the visible face located are stored in a 'plane' list of line segments.

The plane list is thus (1-2, 2-8, 8-6, 6-1).

The next step in the algorithm is to alter the current perimeter list of line segments such that the segments which are present in the list after the alteration are associated with only one plane face still to be located.

Note that it is necessary at this stage to alter the existing current perimeter list (1-2, 2-3, 3-4, 4-5, 5-6, 6-1) since line segments 1-2 and 6-1 are not now associated with one face still to be located since the visible plane face on which they lie (1,2,8,6) has now been located.


START
select vertices i,j corresponding to any given segment in the initial perimeter list.
locate first vertex L in vertices list such that i,j and L are distinct vertices
compute coefficients a,b,c,d to correspond to plane $ax+by+cz+d=0$ in which the vertices i,j and L lie
any more elements in vertices list
NO
FINISH
store k
YES
NO
YES
select next vertex k in vertices list
is $d(ax_k+by_k+cz_k+d) > 0$
NO
is $\mathrm{abs}\{d(ax_k+by_k+cz_k+d)\} < \delta$
YES
L = k
empty list of stored vertices

FLOW DIAGRAM ILLUSTRATING THE METHOD USED TO LOCATE THE NEAREST PLANE.

Diagram 3.10

At the finish the nearest plane is defined as being that on which the vertices a, b and L lie. The stored list of vertices contains the remaining vertices on this plane.

3.2.5. Formation of New Perimeter List of Line Segments. In the example given earlier the visible edges of the plane face associated with edge 1-2 have been found and are stored in a plane list (1-2, 2-8, 8-6, 6-1). The current perimeter list of line segments is 1-2, 2-3, 3-4, 4-5, 5-6, 6-1. Now since each segment in this current perimeter list is, by definition, associated with just one face of the polyhedron to be located, it is clear that if a segment of this list occurs in the plane list it can be cancelled from the current perimeter list. Suppose also that these segments are deleted from the plane list of segments.

In the example given, segments 1-2 and 6-1 will be cancelled in both lists so that the remaining lists are as follows :

current perimeter list: 2-3, 3-4, 4-5, 5-6

plane list : 6-8, 8-2

Since each visible edge, apart from those in the initial perimeter list, is associated with two visible plane faces, the remaining segments in the plane list will now be associated with just one plane face to be located. These edges therefore satisfy the conditions of current perimeter edges and may therefore be added to the current perimeter list which is now as follows :

2-3, 3-4, 4-5, 5-6, 6-8, 8-2

The segments remaining in the plane list are added to the draw list of visible line segments since this is the first occurrence of these segments. Note that before the plane face (1-2, 2-8, 8-6, 6-1) was located these line segments were not associated with any plane faces already located.

A convenient method of cancelling the segments in the current perimeter and plane lists is by having a direction associated with each of the segments. Suppose this direction is anti-clockwise for the current perimeter list and clockwise for the plane list.

The line segments of the current perimeter list have directions as shown below:

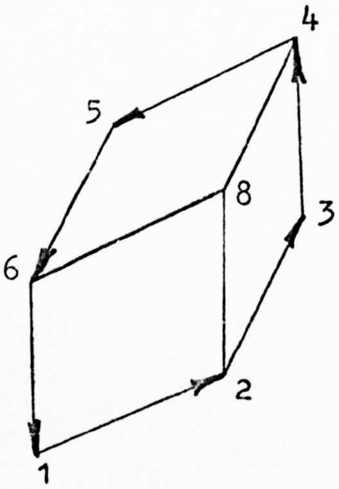


Diagram 3.11

The line segments of the plane list have directions as follows :

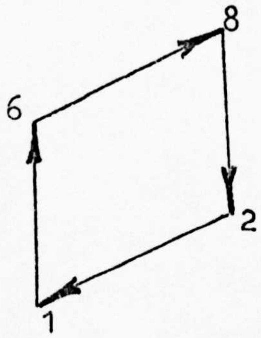


Diagram 3.12

The line segments which are cancelled are those which have opposite directions associated with them and are thus 1-2 and 6-1.

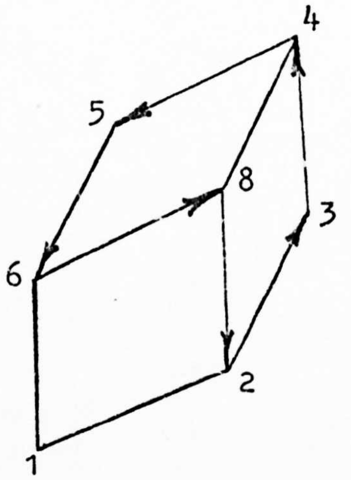


Diagram 3.13

The new current perimeter list is thus

2-3, 3-4, 4-5, 5-6, 6-8, 8-2

The first segment of the new current perimeter list (in the example given this is 23) is taken and the visible face associated with the corresponding edge located. The current perimeter list of line segments is again altered and the method is repeated until the plane lists and current perimeter lists completely cancel. At this stage the current perimeter list will be empty so that there are no visible edges of the polyhedron associated with plane faces still to be located. Moreover the draw list of edges will contain all visible edges of the convex polyhedron.

3.2.6. Location of Hidden Faces. The method can now be repeated to find the hidden faces of the polyhedron. The initial entry in the current perimeter list of line segments consists as before of the initial perimeter edges. In this case however these edges are not added to the draw list of edges which will now consist of the hidden edges of the convex polyhedron.

The plane face located in this case will be that plane face such that none of the vertices of the polyhedron lie on the opposite side of the plane face to the viewpoint. If the constants a,b,c,d define the current plane face and if L is a vertex being tested, L will replace the third vertex if

$$d \times (ax_L + by_L + cz_L + d) < 0$$

where (x_L, y_L, z_L) are the cartesian co-ordinates of vertex L for then L lies on the opposite side of the plane face to the viewpoint.

This is the only alteration necessary in the general method described earlier.

3.3. Valid Third Vertices.

The algorithm locates each plane face of the polyhedron once and once only so that it is not necessary, after locating a plane face, to check against a list of plane faces to ascertain whether the face has been located before. It is not sufficient, however, to select any vertex (occurring in the vertices list) as a third vertex to define a plane with the two vertices of the current perimeter list. In this case, as will be shown later, it is possible to find a plane face a second time.

The faces of the convex polyhedron have associated convex polygons on the plane of projection and as explained earlier the current perimeter list of line segments has anticlockwise directions associated with each of the line segments.

Given an initial segment from the current perimeter list it is required to locate the polygon associated with it. Now the points lying on this plane polygon will all lie on one side of the segment since the polygon is convex.

Consider the following example:

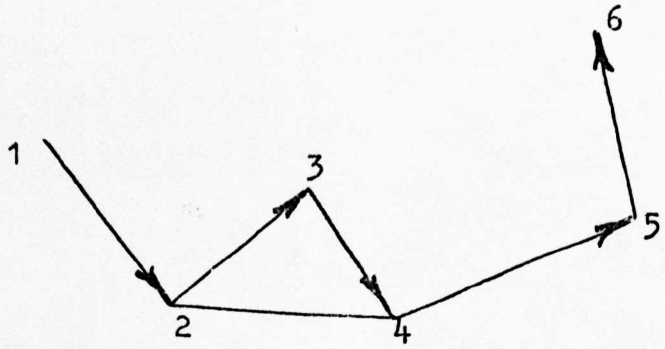


Diagram 3.14

The current perimeter list of line segments at this stage is 1-2, 2-3, 3-4, 4-5 and 5-6 along with other segments which need not be considered in this example. The plane face corresponding to the polygon (2,4,3) has been found.

Now suppose 2-3 is selected as the edge for which the visible plane face of the polyhedron is to be located. The corresponding points on the plane of projection will all lie on one side of segment 2-3. By the method used to find the new current perimeter at each stage, this side of the segment will be the left hand side (in the direction 2-3). Therefore all points lying on the convex polygon associated with segment 2-3 will lie to the left of it. Thus any points on the plane of projection which lie to the right of the segment under review must be invalid as third vertices. This will prohibit point 4 from being used as a third point for if it was used then the already located polygon (2,3,4) would be located again. Suppose the visible face associated with edge 2-3 is (2,3,1). Vertex 1 is a valid third vertex since the corresponding point lies to the left of the directed segment 2-3.

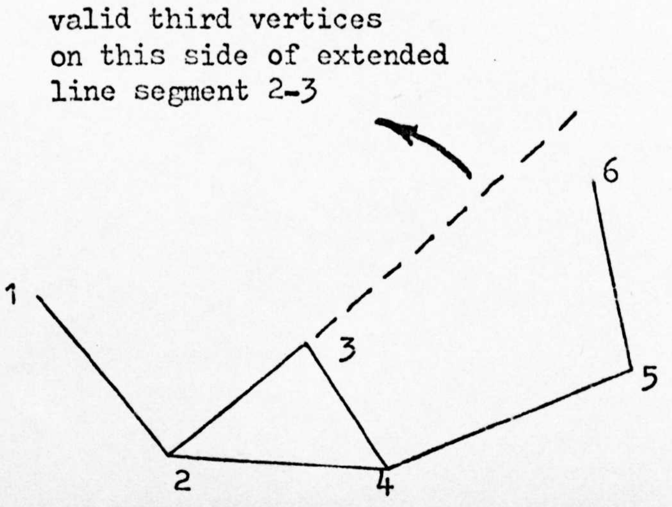


Diagram 3.15

Thus, by placing this restriction on the third vertices used, no face of the convex polyhedron will be located more than once. A simple test can be used to determine which side of a directed line a given point lies.

Consider the directed line segment 2-3 and a third point L. The area of the triangle D defined by the points 2,3 and L is given by

$$D= \begin{vmatrix} x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_L & y_L & 1 \end{vmatrix}$$

where (x_2, y_2) are the cartesian co-ordinates of point 2. If the determinant D is positive the point (x_L, y_L) lies to the left of the directed line 2-3.

Thus, for each of the vertices in the vertices list, it is possible by determining the sign of the area of a triangle, to determine whether a vertex is a valid vertex for the line segment under consideration.

By this means plane faces of the polyhedron will be located once and once only.

3.4. Reduction in the Number of Vertices in Vertices List.

As explained in the last section each vertex of the vertex list must be tested in order to ascertain whether it can be used as a valid third vertex. It is also possible to reduce the number of vertices in the vertices list as the algorithm proceeds.

It is fairly obvious that vertices should only be present in the vertices list if they lie on plane faces yet to be located. If it can be shown that it is not possible for a given vertex to be on any more plane faces, then the vertex can be erased from the vertex list. The reduction in the number of vertices in the vertices list will be particularly beneficial if there exist a large number of spatial vertices for the given convex polyhedron.

Suppose a situation exists where two adjacent segments of the current perimeter list both lie on the plane face located.

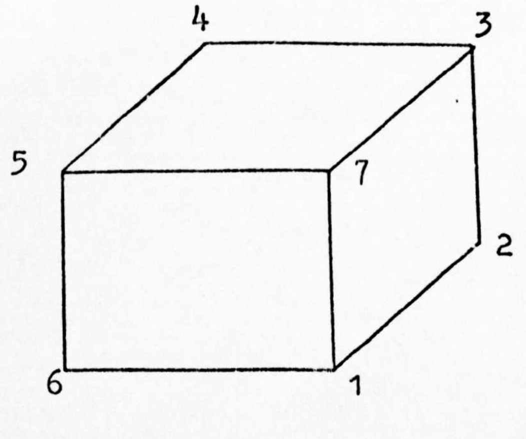


Diagram 3.16

In the example above suppose the current perimeter list is 1-2, 2-3, 3-4, 4-5, 5-6, 6-1.

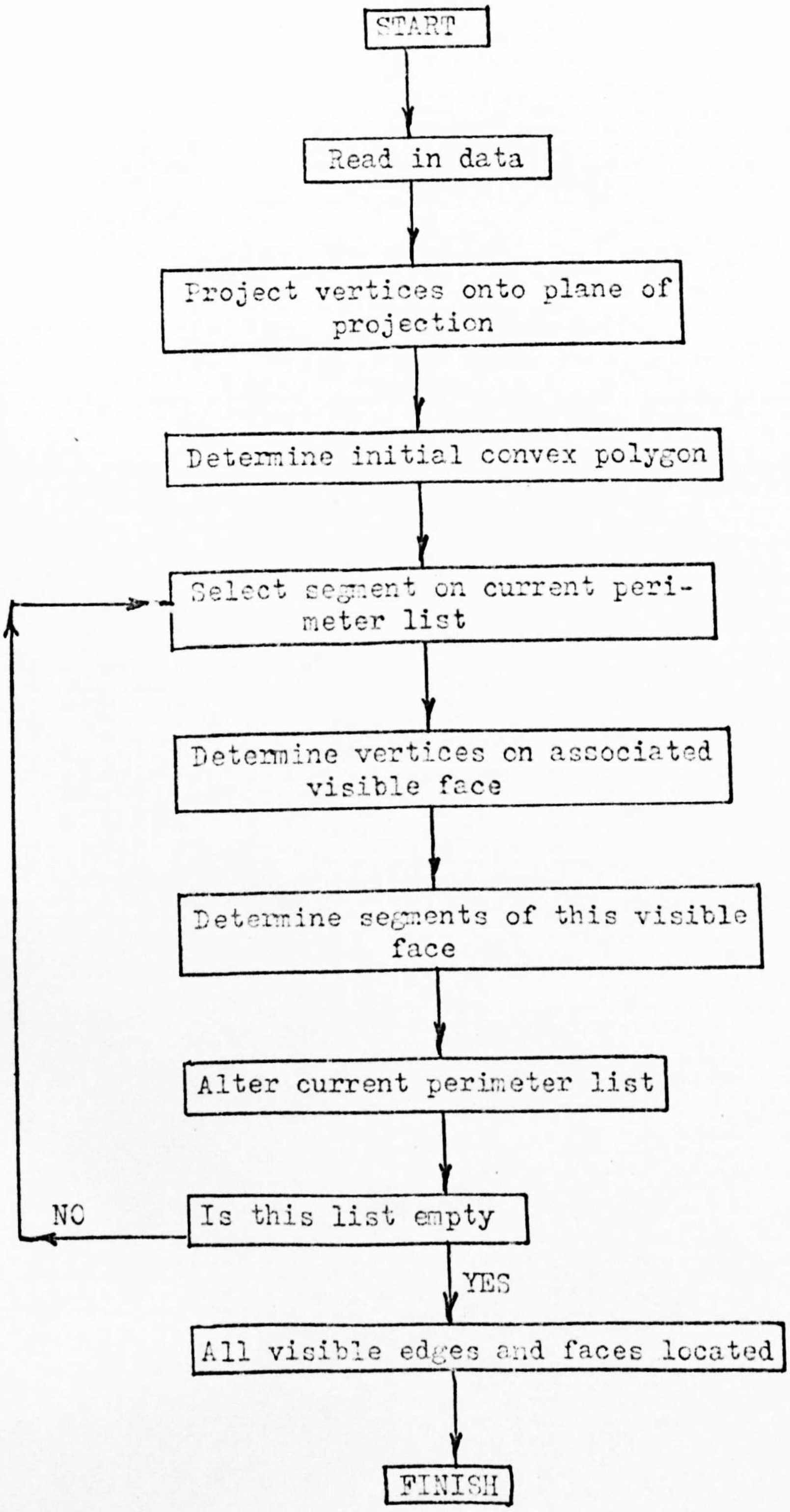
The visible plane face located is say 1-7,7-3,3-2, 2-1.

The segments 1-2 and 2-3 will be cancelled in both lists so that the new current perimeter list will be 3-4, 4-5, 5-6, 6-1, 1-7, 7-3.

Now since segments 1-2 and 2-3 were on the current perimeter list there can be no other non located visible edges of the polyhedron from the vertex 2.

In the example above both these edges (1-2 and 2-3) have been located. There can now be no more visible edges from the vertex 2 and thüs 2 can be deleted from the vertices list, since no more visible plane faces are associated with it. Thus if, at any time, two adjacent current perimeter segments lie on the plane face located the vertex which is common to the two corresponding edges can be deleted from the vertices list since there are no more edges associated with this vertex to be located.

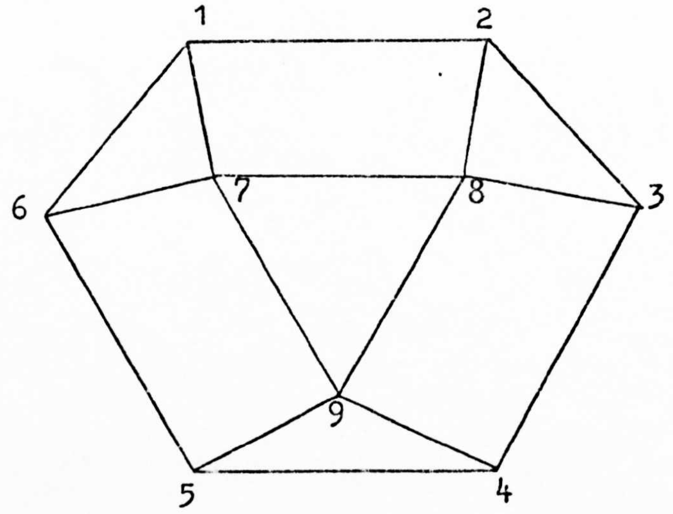
These deleted vertices need only be re-introduced when locating the hidden faces of the polyhedron if they are associated with edges on the initial perimeter list. It is only in this case that they lie on plane faces which are hidden to the viewpoint.


START
Read in data
Project vertices onto plane of projection
Determine initial convex polygon
Select segment on current perimeter list
Determine vertices on associated visible face
Determine segments of this visible face
Alter current perimeter list
Is this list empty
NO
YES
All visible edges and faces located
FINISH

Flow Diagram of Method used to Locate Visible Edges of a given Convex Polyhedron.

Diagram 3.17

3.5. General Example



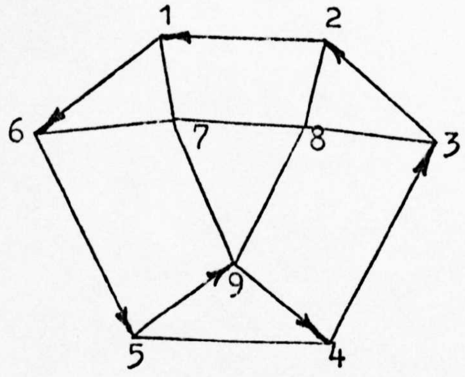
Consider the convex polyhedron the visible edges of which are shown above. As a general example of the method each plane face of the polyhedron will be located.

The initial perimeter edges will be 5-4, 4-3, 3-2, 2-1, 1-6, 6-5.

Thus the current perimeter list will be 5-4, 4-3, 3-2, 2-1, 1-6, 6-5.

The visible plane face associated with edge 5-4 has edges 5-9, 9-4, 4-5 so that the new current perimeter is

4-3
3-2
2-1
1-6
6-5
5-9
9-4



initial entry in <u>draw</u> list

5-4
4-3
3-2
2-1
1-6
6-5

Edges added to draw list from first plane face 5-9, 9-4.

next visible plane face

4-9 edges added to draw list 9-8, 8-3.

9-8

8-3

3-4 vertices rejected - none

new current perimeter

3-2

2-1

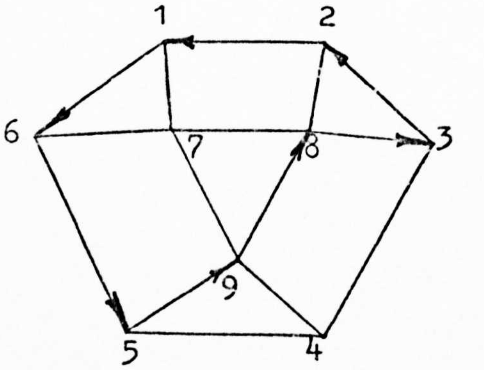
1-6

6-5

5-9

9-8

8-3



next visible plane face

3-8 edges added to draw list 8-2

8-2

2-3 vertices rejected - vertex 3

new current perimeter

2-1

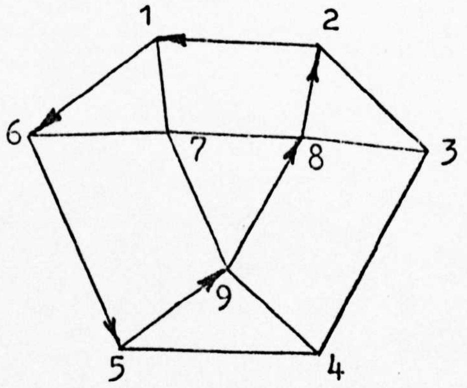
1-6

6-5

5-9

9-8

8-2



next visible plane face

2-8 edges added to draw list 8-7, 7-1

8-7

7-1

1-2 vertices rejected - vertex 2

new current perimeter

1-6

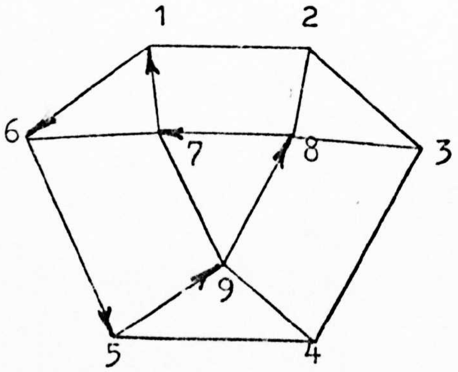
6-5

5-9

9-8

8-7

7-1



next visible plane face

1-7 edges added to draw list 7-6

7-6

6-1 vertices rejected - vertex 1

new current perimeter

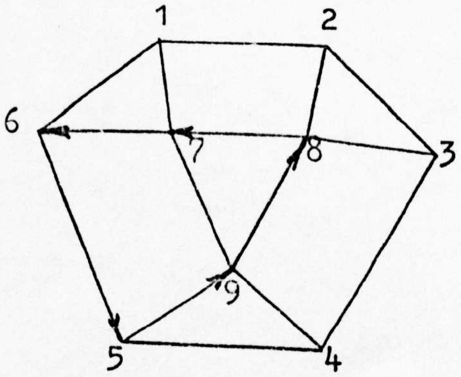
6-5

5-9

9-8

8-7

7-6



next visible plane face

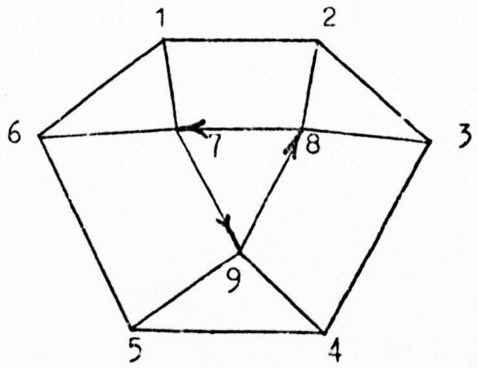
5-6	edges added to draw list 7-9
6-7	
7-9	
9-5	vertices rejected - vertex 5, vertex 6.

new current perimeter

9-8

8-7

7-9



next visible plane face

9-7	edges added to draw list
7-8	
8-9	vertices rejected - vertex 9, vertex 8, vertex 7

new current perimeter

——

At this point the current perimeter list and plane lists have completely cancelled.

Final draw list

5-4
4-3
3-2
2-1
1-6
6-5
9-8
8-3
8-2
8-7
7-1
7-6
7-9
5-9
9-4

It can be seen that the final draw list contains all the visible edges of the convex polyhedron. As the method proceeded a total of 8 vertices were rejected.

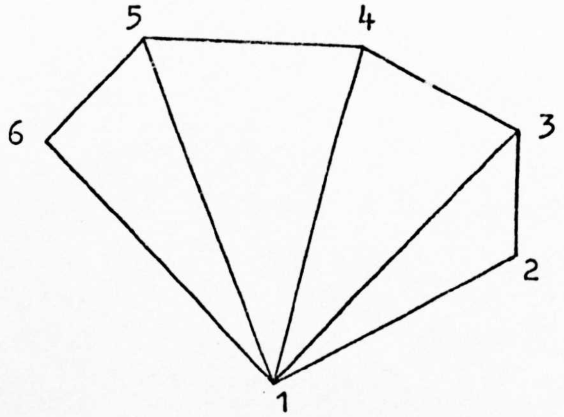
As a guide to the formation of the new current perimeter after each location of a visible plane face the directed current perimeter line segments are shown on the convex polyhedron in each case.

N.B. The draw list is not meant necessarily to be the order in which the lines are drawn. The drawing of a given list of line segments is discussed in detail in Chapter 5.

3.6. Example of Current Perimeter Segments Splitting into Two Disjoint Pieces.

In the general method explained there is no need to select the first segment of the current perimeter list. In the example given below the first segment is never selected to show that it has no effect on the general method. In addition, the current perimeter list is split into two disjoint pieces.

Consider the following example of a convex polyhedron



The initial perimeter list of edges will be 1-2, 2-3, 3-4, 4-5, 5-6, 6-1. Suppose segment 4-5 is selected from the initial current perimeter list (which initially is the same as the initial perimeter list).

next visible plane

1-5

5-4

4-1

initial entry in draw list

1-2, 2-3, 3-4, 4-5, 5-6, 6-1

new current perimeter

1-2

2-3

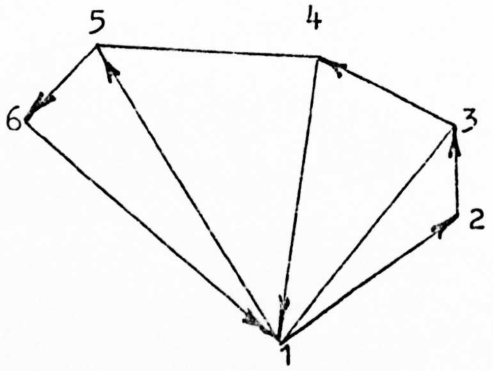
3-4

4-1

1-5

5-6

6-1



edges added to draw list

4-1, 1-5

At this stage the current perimeter consists of two convex polygons.

segment 2-3 selected

next visible plane face

2-1

1-3

3-2

new current perimeter

3-4

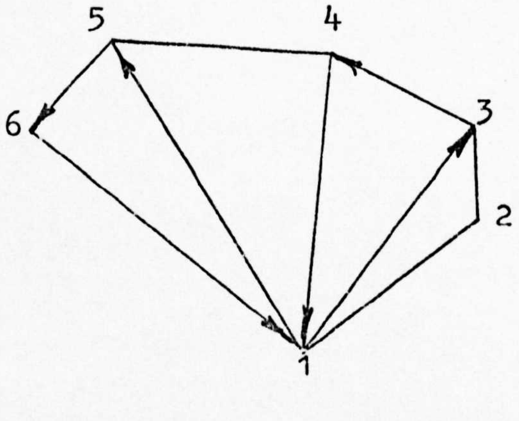
4-1

1-5

5-6

6-1

1-3



edges added to draw list 1-3

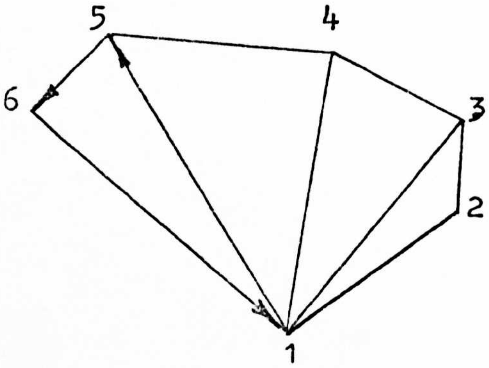
segment 4-1 selected

next visible plane face

3-1

1-4

4-3



new current perimeter

1-5

5-6

6-1

no edges added to draw list.

segment 5-6 selected

next visible plane face

1-6

6-5

5-1

new current perimeter

---

final draw list of visible segments

1-2

2-3

3-4

4-5

5-6

6-1

4-1

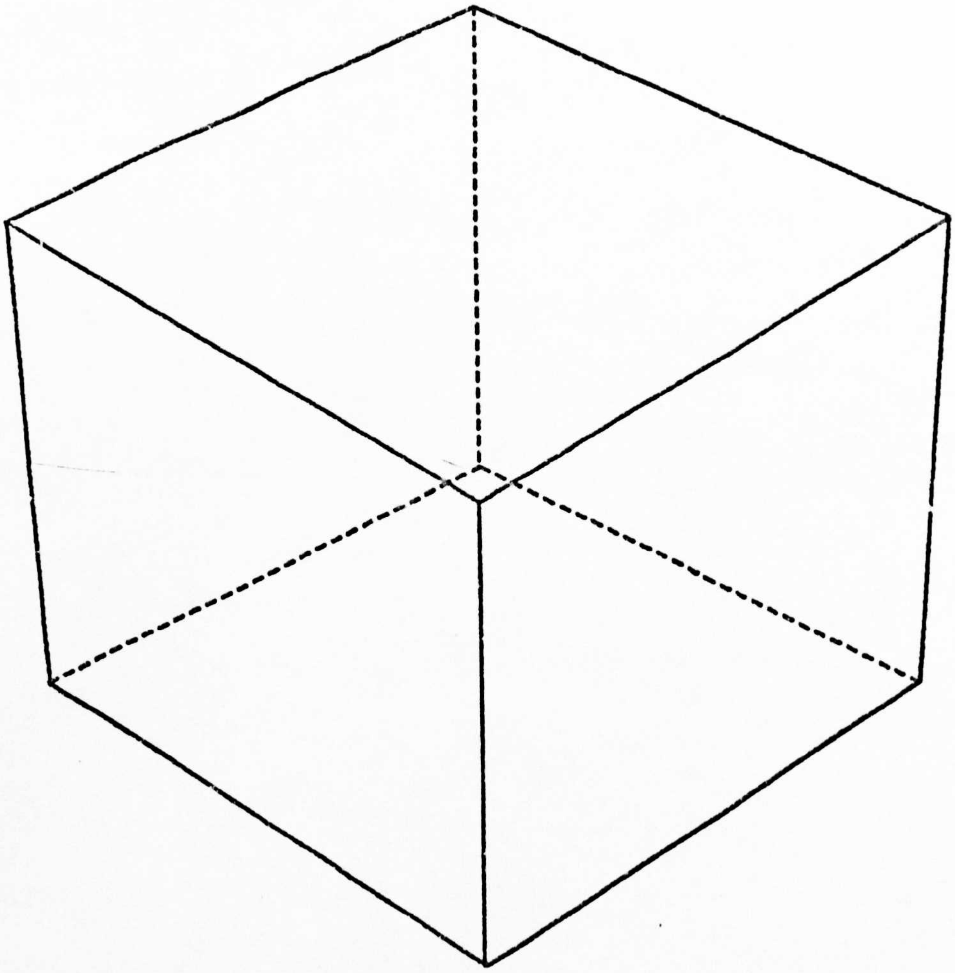
1-5

1-3

The draw list thus contains all the visible edges of the convex polyhedron.

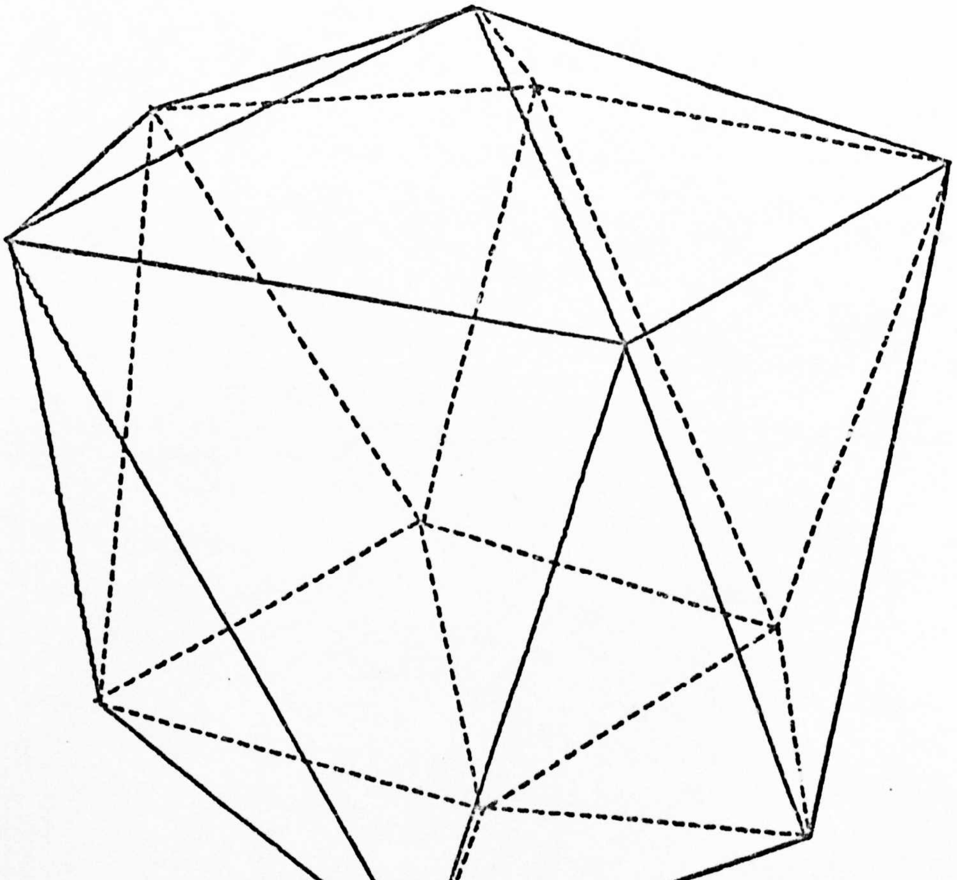
A number of examples of convex polyhedra, produced by the algorithm described, are presented. These have been produced using the general scheme (described in the introduction) for obtaining final line drawings on the graph plotter. (See Diagram 1.3).

The Algol programs relating to the work described in this chapter, are presented in Appendix 3.



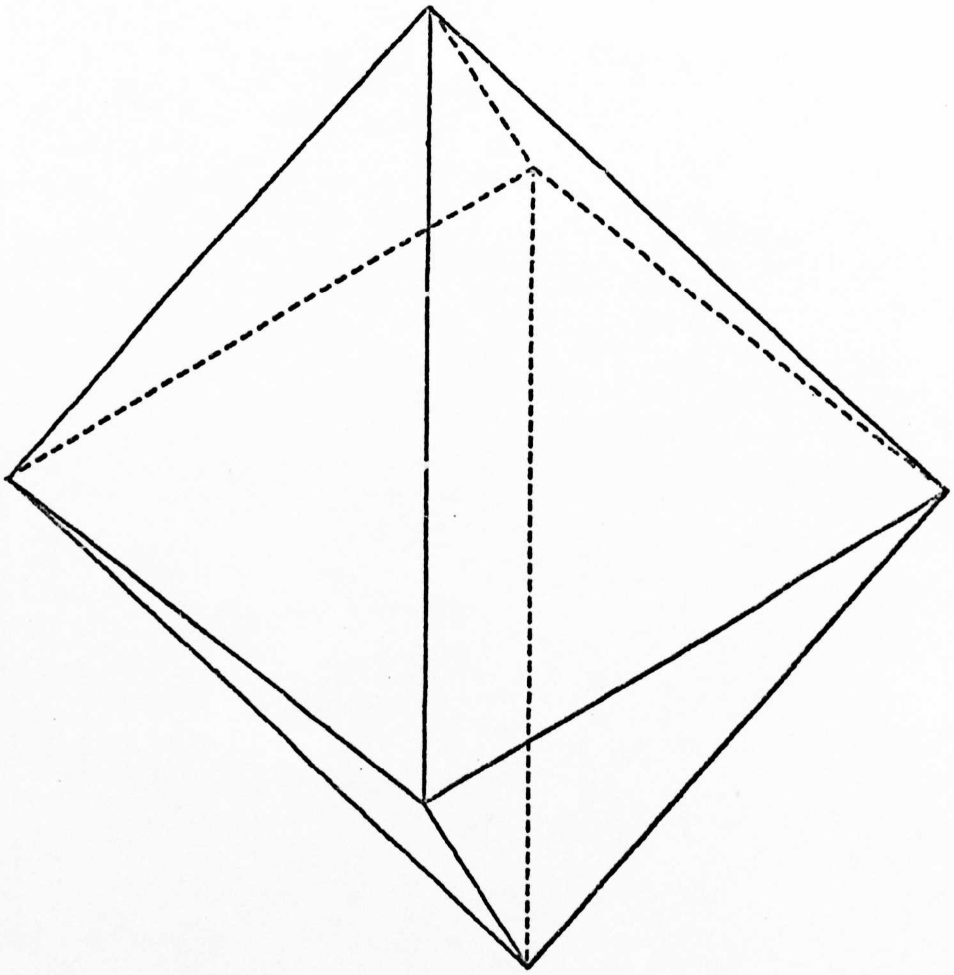
EXAMPLE OF A CUBE.

Figure 8.



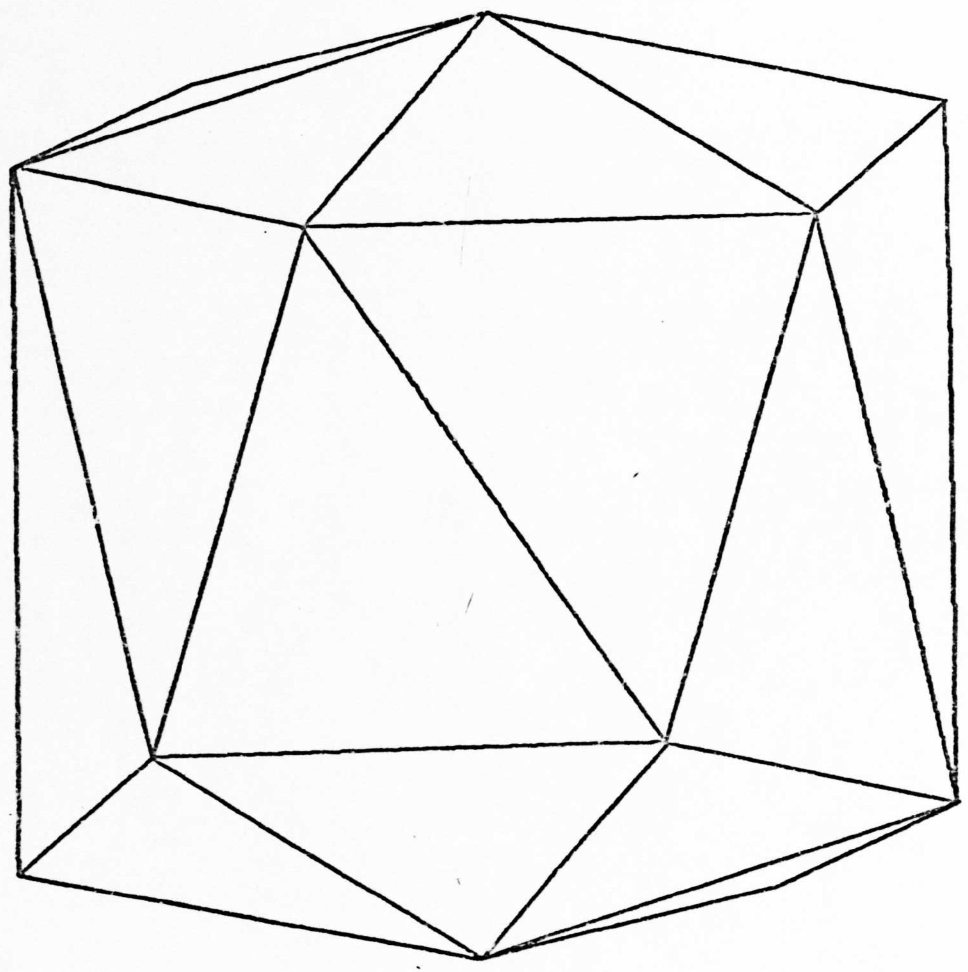
PERSPECTIVE VIEW OF ICOSAHEDRON.

Figure 9



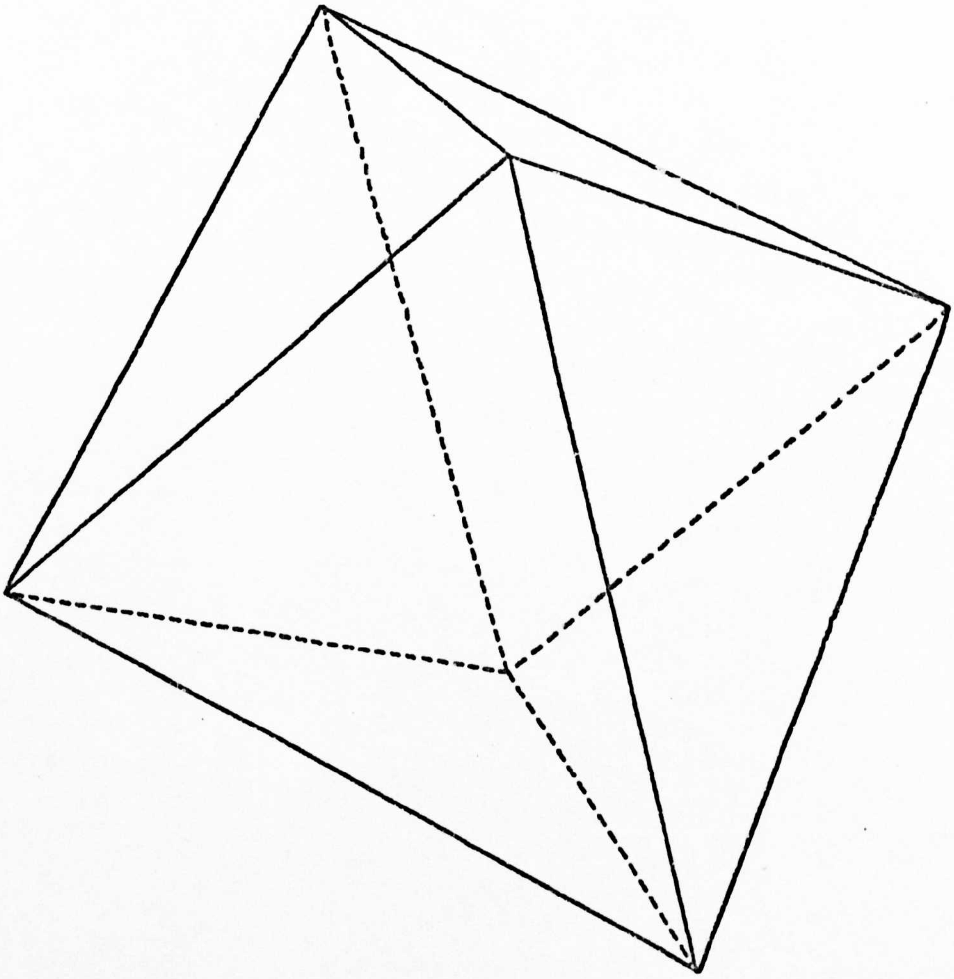
PERSPECTIVE VIEW OF OCTAHEDRON.

Figure 10.



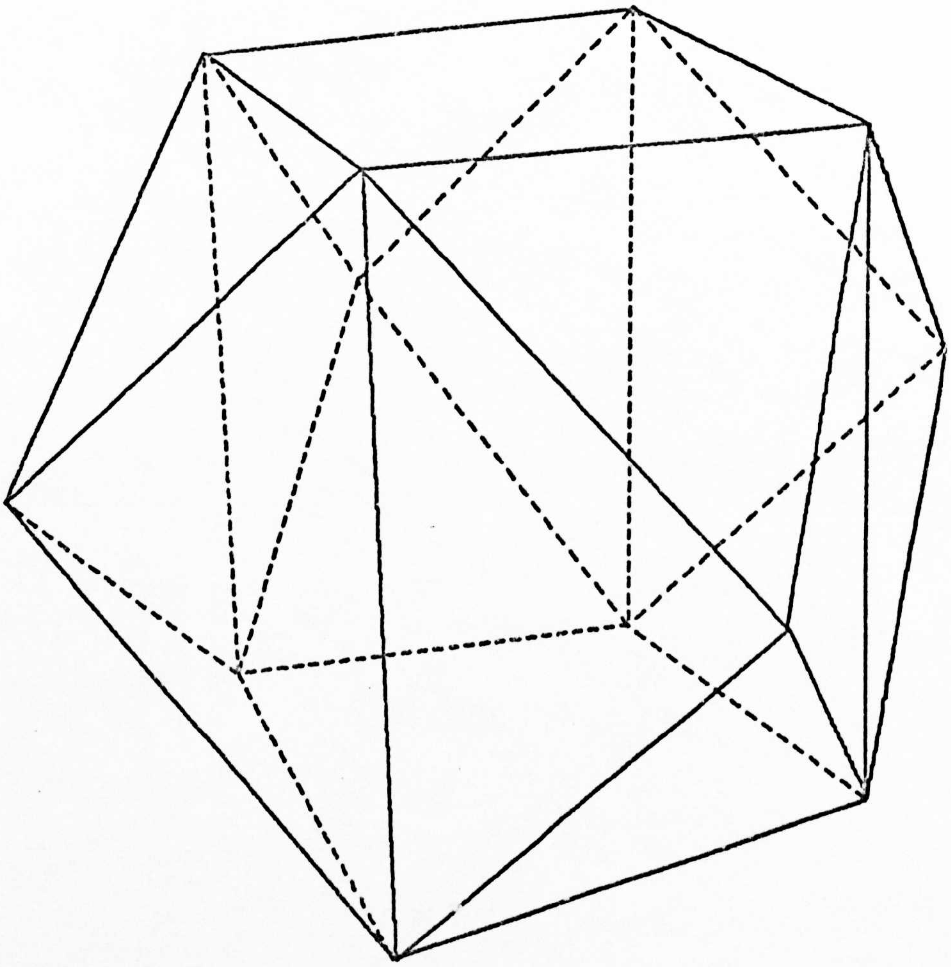
PERSPECTIVE VIEW OF ICOSAHEDRON. HIDDEN LINES DELETED.

Figure 11.



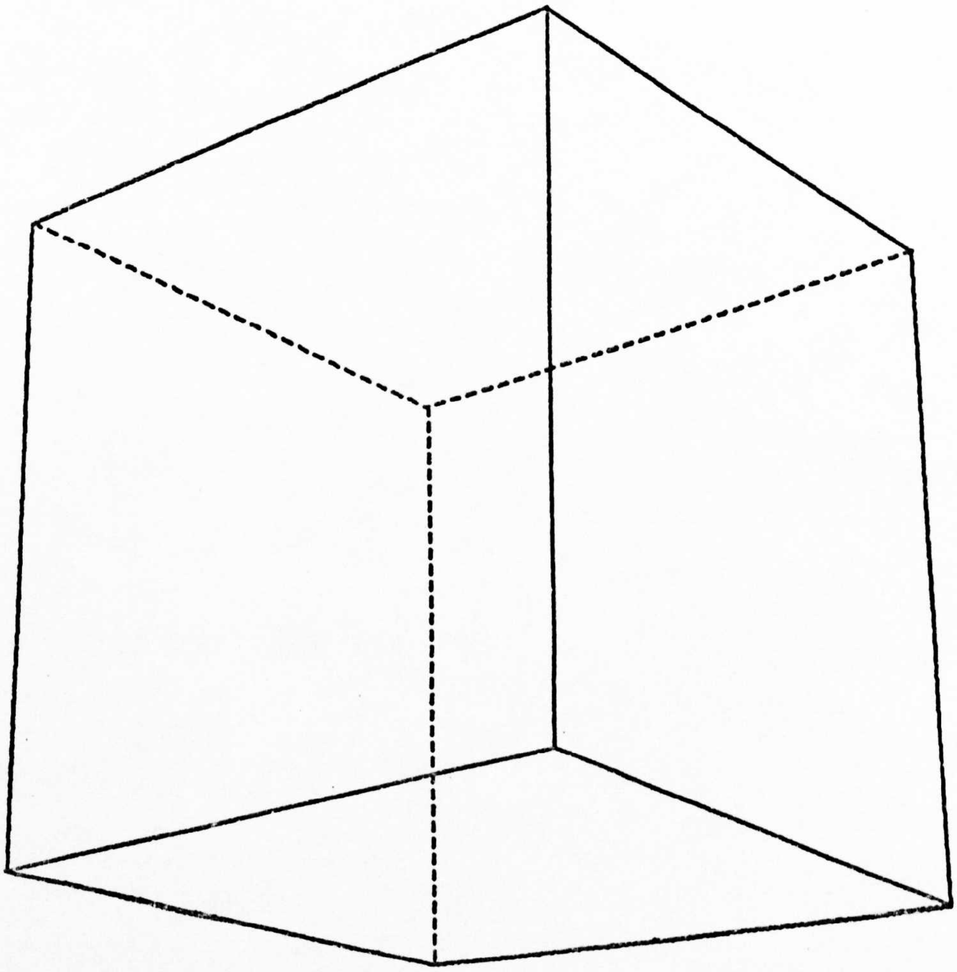
PERSPECTIVE VIEW OF OCTAHEDRON.

Figure 12.



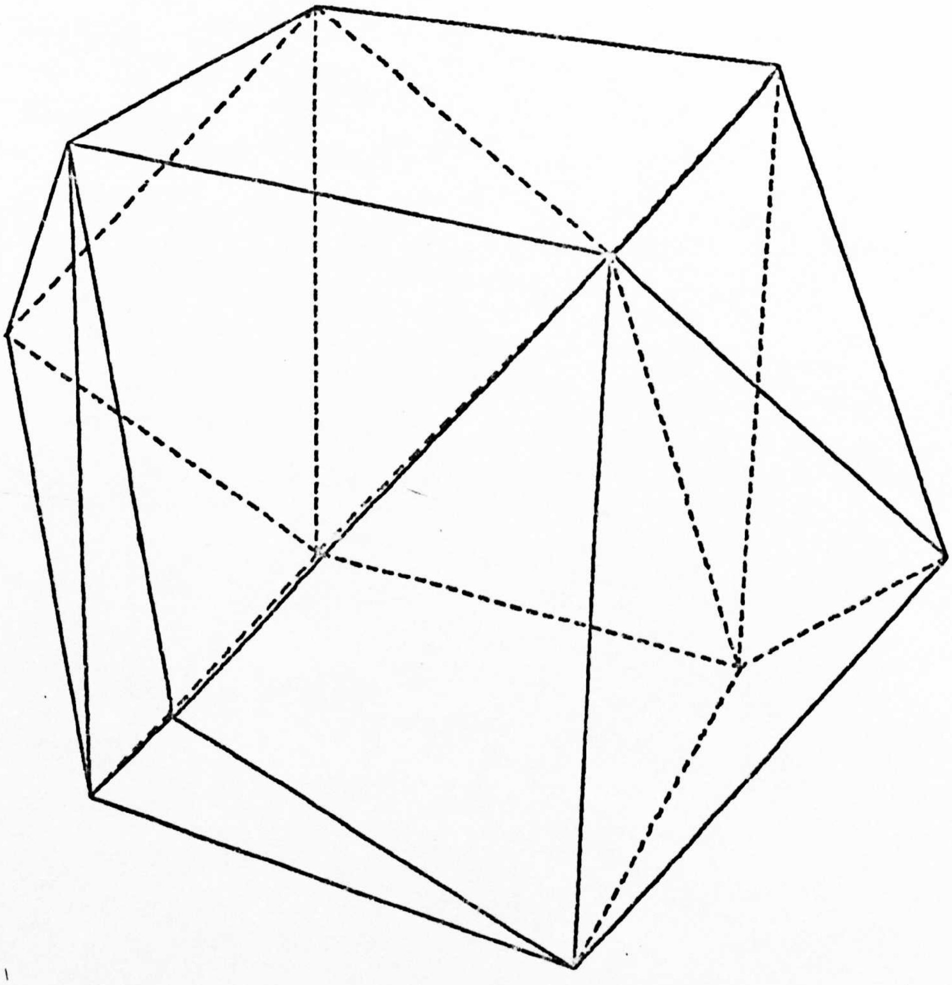
PERSPECTIVE VIEW OF CONVEX POLYHEDRON WITH 18 FACES.

Figure 13



EXAMPLE OF A CUBE.

Figure 14



AN EXAMPLE OF A CONVEX POLYHEDRON.

Figure 15

EXAMPLE OF A ROTATING CUBE.

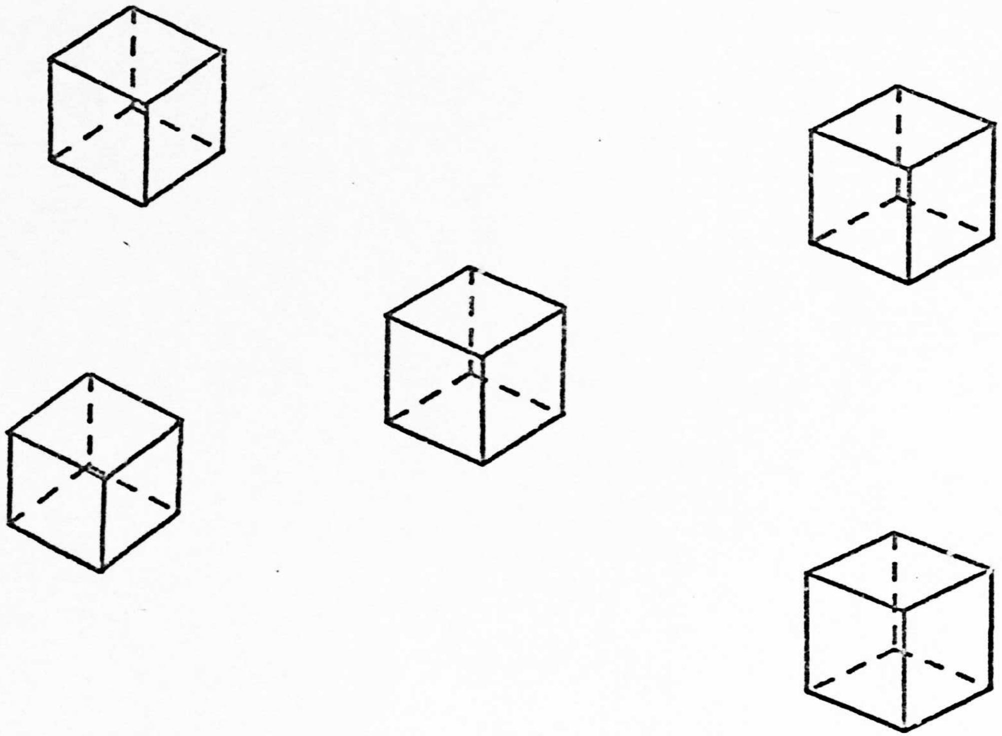
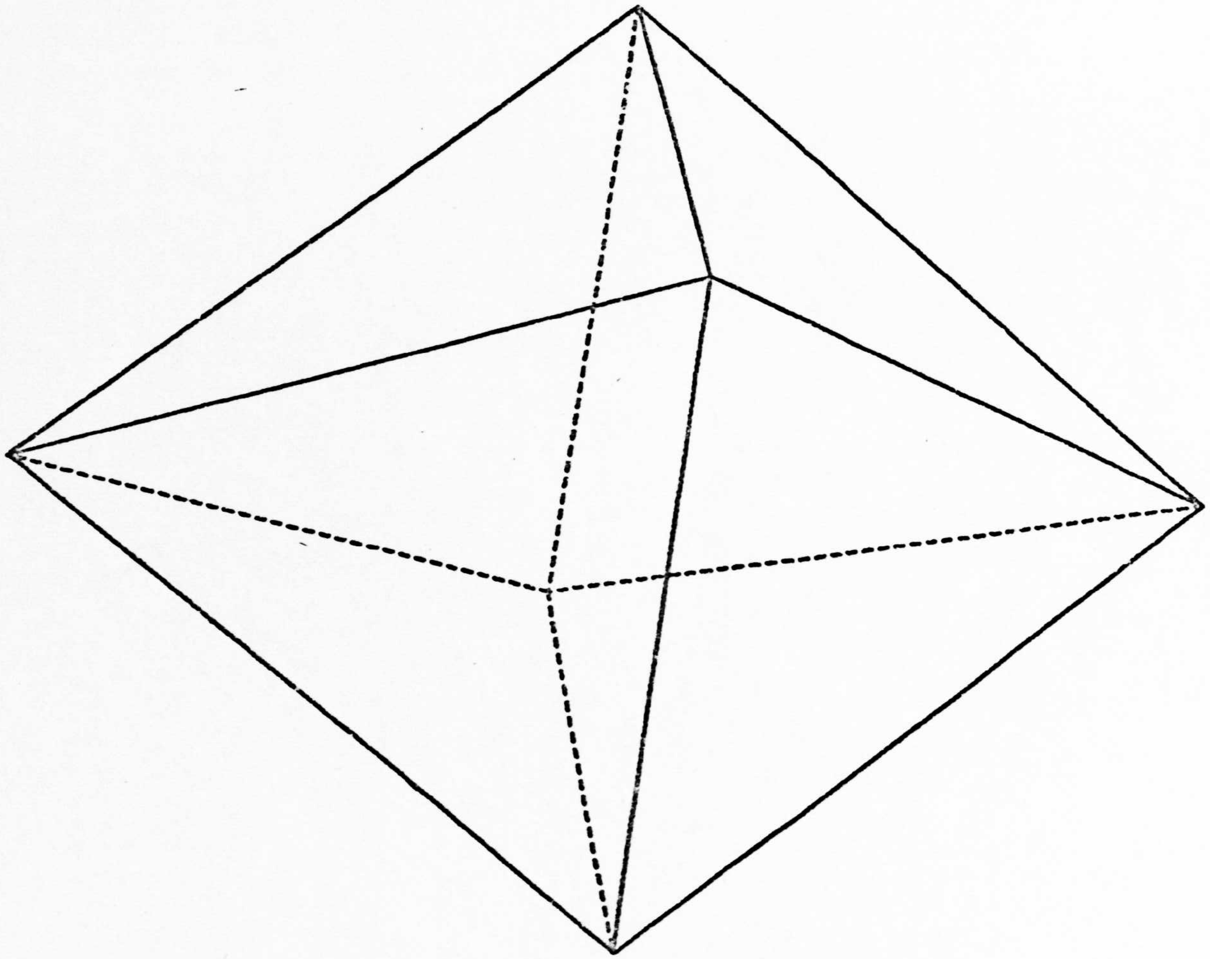


Figure 16.



EXAMPLE OF AN OCTAHEDRON.

Figure 17.

CHAPTER 4.

COMPUTER REPRESENTATION OF GENERAL POLYHEDRA

Description of a simple method to determine the visible and hidden edges of any non-convex polyhedron.

CHAPTER 4

COMPUTER REPRESENTATION OF GENERAL POLYHEDRA.

4.1. Introduction.

The problem of deciding if a given edge of a polyhedron is visible or hidden to the viewpoint is much easier when a convex polyhedron is being considered since in this case any plane face is either completely hidden or completely visible to the viewpoint. A method of representing convex polyhedra has been explained in the previous chapter.

With non-convex polyhedra, a plane face can have portions which are visible to the viewpoint and portions which are hidden to the viewpoint. Similarly, edges of non convex polyhedra can have portions which are both visible and hidden to the viewpoint. An example of a non convex polyhedron having an edge which contains both visible and hidden portions is shown in Diagram 4.1.

The edge 1-5 and the face (1,5,4,3) have both visible and hidden portions.

The problem of deciding if an edge is visible or hidden to the viewpoint has gained wide recognition in the field of computer graphics being known in general as the 'Hidden Line Problem'.

An algorithm is presented in this chapter which divides the line drawing corresponding to the non convex polyhedron into a number of partial line segments (to be defined later) for which a depth count, which is a measure of the number of plane faces masking the corresponding edge from the viewpoint, is obtained. If the depth count for a particular partial line segment is greater than zero then the partial line segment is hidden to the viewpoint, otherwise it is visible to the viewpoint.

A convenient method of arranging the information associated with any polyhedron is also presented. The least amount of input data required to define a given polyhedron consists only of the cartesian co-ordinates of the n spatial vertices in addition to an ordered list of vertices corresponding to each plane face of the polyhedron.

From this input data it is possible to determine additional information associated with the non-convex polyhedron which is required by the algorithm.

As in the previous chapter faces of the polyhedron will be referred to as polygons on the plane of projection but in this case the polygons may be non convex. Edges of the polyhedron will be referred to as segments on the projection plane and vertices will have corresponding points on the projection plane.

4.2. Computer Representation of Information Associated with a Given Polyhedron.

In the computer representation of polyhedra (both convex and non convex) it is necessary and important to have some convenient method of arranging the information associated with the polyhedron without using an excessive amount of storage. It is also important to reduce to a minimum the amount of imput data associated with a given polyhedron.

The least amount of information required to completely define any given polyhedron consists of a list of vertices of each of the plane faces of the polyhedron, ordered in some direction as seen from outside (or inside) the polyhedron (the direction can either be clockwise or anti-clockwise). In addition to this the cartesian co-ordinates of each of the spatial vertices of the polyhedron are also required.

Suppose the list of vertices of each of the plane faces corresponds to a one dimensional array faces to which a pointer, which is a one dimensional array pointer, indicates the position in the faces list of the beginning of the ith plane face. Thus 'faces (pointer (i))' is the first vertex of the ith plane face. The final vertex of the ith plane face will thus be the element '(pointer (i + 1) -1)' in the faces list which corresponds to 'faces (pointer (i + 1) -1)'. The vertices of each plane face can thus be considered to be stored as sub-lists in the faces list.

As a simple example suppose the faces (1,12,2,3) and (1,3,4,5,) of diagram 4.1 are to be stored in the faces list. The relevant faces list and pointer list in this case will be as follows.

Face List	Pointer List
1	1
12	6
2	11
3	
1	
1	
3	
4	
5	
1	

Note that the final vertex corresponds to the initial vertex in each case. (In the faces list).

It is important that the vertices in the faces list are ordered in some direction (clockwise or anti-clockwise). From the two lists above, it can be seen that the sub-list containing the vertices of the second plane face (in this case (1,3,4,5,1))lies between elements 'pointer (2)' (which is 6) and 'pointer (3) - 1' (which is 10). The vertices of face 2 are thus (1,3,4,5,1).

From the input data supplied it is possible to determine additional information associated with the polyhedron. The edges of the polyhedron can be determined and may be stored in a two dimensional array edges. Edges (i,1) and edges (i,2) will thus be the terminal vertices of the ith edge of the polyhedron.

Consider now a convenient method by which the edges of the polyhedron can be determined .and stored in the array edges.

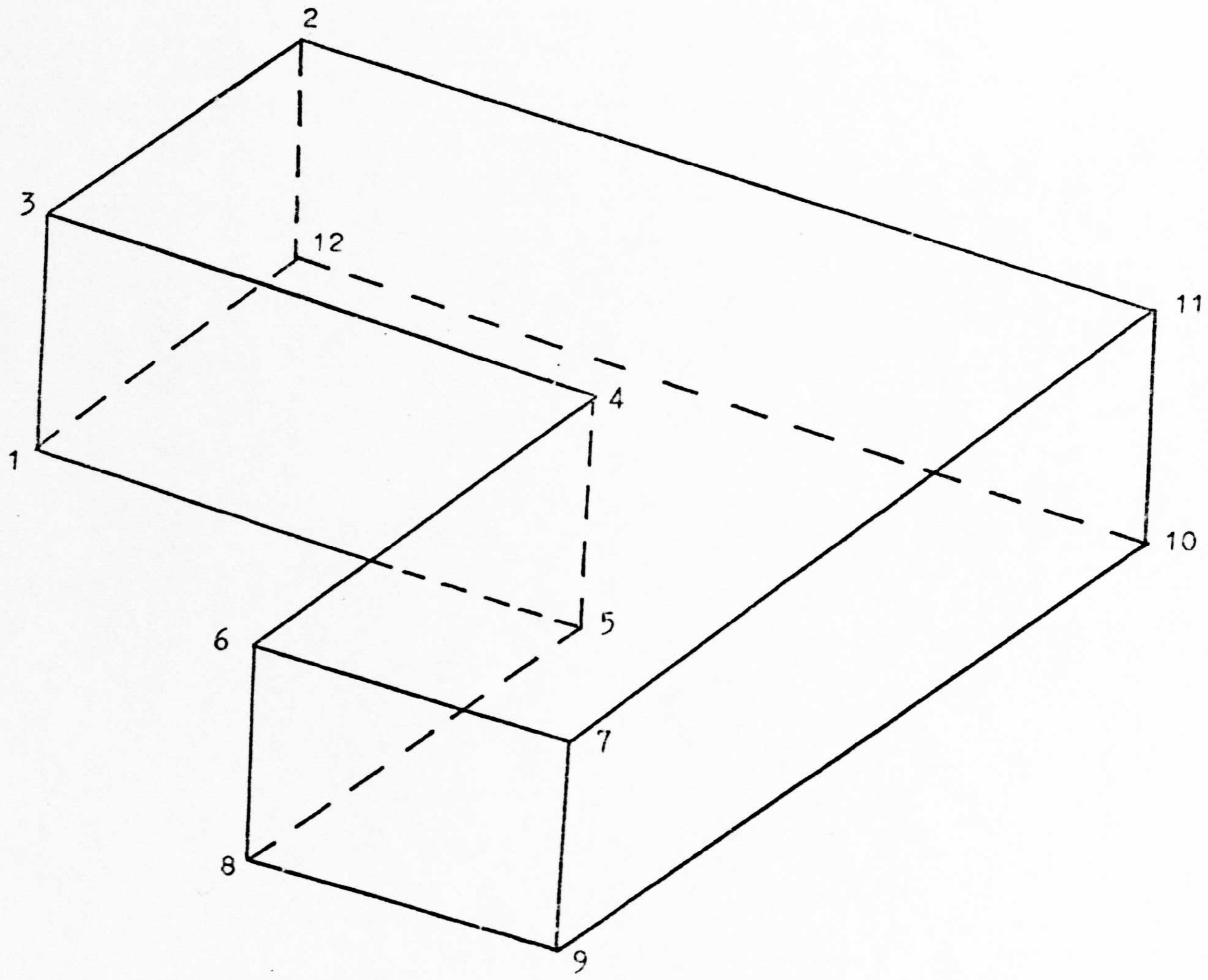


Diagram 4.1

EXAMPLE OF NON-CONVEX POLYHEDRON

4.2.1. Determination of Edges of a Given Polyhedron. Any two adjacent vertices in any sub-list of the faces list defines an edge of the polyhedron. Since each edge of the polyhedron is associated with two plane faces it is not sufficient, nor economical, to simply define adjacent vertices of the faces list as corresponding to edges of the polyhedron since each edge would then be located twice.

Suppose an edge i, corresponding to two adjacent vertices in the faces list, is only transferred to the edges list if

$$\text{faces } (i) > \text{faces } (i + 1)$$

Since each of the sub-lists of vertices (corresponding to each of the plane faces) has elements stored in a given direction, each edge of the polyhedron will be transferred to the edges list only once. It is therefore possible, from the input data, to form an edges list containing each edge of the polyhedron once and once only.

At this stage a faces list of the vertices of each plane face, an edges list of the terminal vertices of each edge and a list of the cartesian co-ordinates of each of the vertices of the polyhedron exists.

4.2.2. Determination of Plane Faces Associated With a Given Edge. In the general method to be described it is necessary to have easy access to the two plane faces associated with a given edge. For each of the edges in the edges list it is therefore necessary to search through the faces list until the two corresponding vertices of the given edge are located as adjacent vertices in the faces list. By this means it will be possible to form a two dimensional array *polygons* such that for any edge i the two plane faces associated with it are given by 'polygons (i,1)' and 'polygons (i,2)' which will act as pointers to the faces list. (The example to follow will clarify these details).

It is important to notice at this stage that since the two dimensional array *edges* has elements 'edges (i,1)' and 'edges (i,2)' as the terminal vertices of the ith edge, the edges array can similarly be used to define the ith line segment on the projection plane. This will of course have terminal points 'edges (i,1)' and 'edges (i,2)'.

In a similar way the faces list, which has elements corresponding to the vertices of each plane face can be used to define the points associated with each of the corresponding polygons on the projection plane.

4.2.3. *Example*. Consider as an example of the method the various arrays which would be set up for the polyhedron shown in Diagram 4.1.

The faces list, the associated pointer list and the edges list are shown below:

Faces List

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

Pointer List

1
6
11
16
21
28
35
40
45

Edges List

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

These lists refer to those set up for the polyhedron given in Diagram 4.1.

Note that there 8 faces (p), 12 vertices (n) and 18 edges (e) so that Euler's equality of p + n = e + 2 is satisfied.

It is now possible to set up the polygons list whose ith entry gives the two plane faces associated with the ith edge. The polygons list is as follows :

1	7
1	2
2	5
3	5
2	3
4	5
3	4
1	5
6	8
4	6
3	6
2	6
1	6
5	7
6	7
7	8
5	8
4	8

Thus from the above list the face 6 lies between the elements' pointer (6)' and 'pointer (7)-1' of the faces list.

The 6th plane face of the polyhedron is thus (2,11,7,6,4,3,2)

Edge number 6 (9-8) has plane faces 4 and 5 associated with it. These plane faces are thus (6,7,9,8,6) and (1,5,8,9,10,12,1) respectively.

4.3. General Method.

Consider a given line segment i on the plane of projection. It is probable that there are a number of other line segments which intersect segment i along its length. Line segments which intersect i at its terminal points are not considered to be intersections. It is convenient to define the two types of intersections which are considered to occur along a given line segment.

4.3.1. Real Intersection. A real intersection between a primary line segment i and a secondary line segment j is said to occur if the corresponding point of intersection of edge j lies on the viewpoint side of the corresponding point of intersection of i.

4.3.2. Virtual Intersection. The intersection will be termed virtual if the corresponding point of intersection of edge i lies on the viewpoint side of the corresponding point of intersection of edge j.

Thus for a given primary line segment there may exist a number of both real and virtual intersections occurring with secondary line segments.

Corresponding to the real intersections occurring along a given primary line segment will exist a number of partial line segments defined as follows.

4.3.3. Partial Line Segments. If there are k real intersections along a given primary line segment then the segment will be divided into (k + 1) partial line segments.

Consider, as an example, line segment 1-5 of Diagram 4.1. A real intersection occurs with the segment 6-4 since the edge 6-4 lies on the viewpoint side of the edge 1-5. Thus there is just one real intersection occurring along the segment 1-5. Suppose this intersection point is labelled as point 13 on the plane of projection. The segment 1-5 can thus be represented as follows :

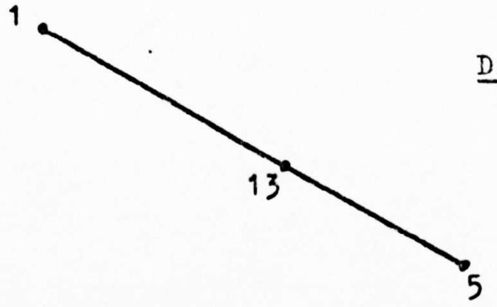


Diagram 4.2

In this case segment 1-5 consists of two partial line segments 1-13 and 13-5.

Note that segments 1-3, 1-12, 5-4 and 5-8 intersect segment 1-5 at its extreme points but these intersections are not to be considered. (As explained earlier).

Now suppose segment 6-4 is taken as the primary line segment. An intersection with segment 1-5 does of course occur but in this case the intersection is virtual. The line segment 6-4 thus has no real intersections occurring along its length and so from the definition, 6-4 is divided into one partial line segment which corresponds to the line segment 6-4.

4.3.4. Depth Count . The importance of partial line segments on the plane of projection is that the partial edge corresponding to it will have a constant number of plane faces of the polyhedron lying between it and the viewpoint. This number of plane faces will correspond to some integer value known as the depth count and formally defined as follows:

The <u>depth count</u> dc of a partial line segment is the number of plane faces of the polyhedron which mask the partial line segment from the viewpoint. It is thus the number of plane faces which lie between the corresponding partial edge and the viewpoint and whose corresponding polygons on the plane of projection enclose the given partial line segment.

The algorithm in this chapter determines the depth count for every partial line segment on the projection plane. If the depth count is greater than zero then the corresponding partial edge is hidden to the viewpoint. Partial line segments with depth counts of zero have their corresponding partial edges visible to the viewpoint for then there are no plane faces of the polyhedron which mask the partial edge from the viewpoint.

4.3.5. <u>Variation of Depth Count Along a Line Segment</u>. The depth count along a given line segment can only change at real intersection points since only then is it possible fdr the plane faces associated with the intersecting segment to mask the edge from the viewpoint. Thus, in determining the variation of depth count <u>along a line segment</u>, it is only necessary to determine the change occurring at real intersection points.

Consider any primary line segment 1-2 which has a real intersection occurring with a secondary line segment 3-4 at point 5

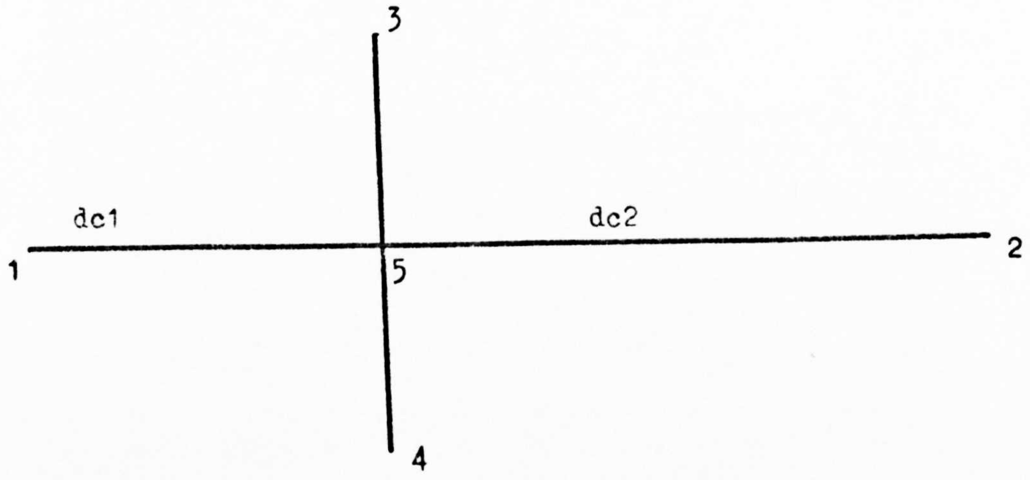


Diagram 4.3

Segment 1-2 is thus divided into two partial line segments 1-5 and 5-2.

Suppose dc1 is the depth count of the partial line segment 1-5 and that dc1 has some known value. Suppose that it is required to determine the depth count dc2 of the partial line segment 5-2 from the value of dc1 .

Consider the two polygons associated with the secondary line segment 3-4. These polygons can either (1).both lie to the left of segment 3-4, (2).both lie to the right of 3-4 or (3).the polygons can lie on either side of 3-4.

Case 1. The polygons both lie to the left of 3-4. The two plane faces corresponding to these polygons must thus mask the partial edge 1-5 from the viewpoint. The partial edge 5-2 cannot therefore be hidden by these two plane faces and so the depth count dc2 must be subsequently reduced by 2 so that

$$\underline{dc2 = dc1 - 2}$$

In this case the depth count along the line segment 1-2 is decreased.

Case 2. In this case the two polygons lie to the right of the secondary line segment 3-4 and they must therefore mask the partial edge 5-2 from the viewpoint. Since the partial edge 1-5 cannot be hidden by these two faces the depth count dc2 must be increased by 2 and is given by

$$\underline{dc2 = dc1 + 2}$$

The depth count is thus increased in this case.

Case 3. The two polygons lie on either side of the secondary line segment so that each of the corresponding plane faces masks just one of the partial edges and so in this case there is no change in depth count and dc2 is thus given by

$$\underline{dc2 = dc1}$$

Thus at a real intersection point the change, δ dc, in the value of the depth count is completely dependent on the relative position of the two polygons associated with the secondary line segment. The change in depth count is either 0, +2 or -2.

For any primary line segment on the plane of projection it is thus only necessary to determine the depth count of one of the partial line segments. From this value it is possible to determine the depth counts of the remaining partial line segments associated with the primary partial line segment.

4.3.6. Determination of Initial Depth Count Along a Line Segment. It is necessary to determine the equations of the planes corresponding to the plane faces of the polyhedron. This can be achieved fairly easily by choosing three vertices associated with each of the plane faces of the polyhedron and determining the value of the constants a,b,c and d, such that the plane is given by

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

The constants a,b,c and d can be obtained from the determinant

$$\begin{vmatrix} x1 & y1 & z1 & 1 \\ x2 & y2 & z2 & 1 \\ x3 & y3 & z3 & 1 \\ x & y & z & 1 \end{vmatrix} = 0$$

where vertices 1 (x1, y1, z1), 2 (x2, y2, z2) and 3 (x3, y3, z3) are three vertices lying on the given plane face.

Now suppose it is required to determine the depth count of a given partial line segment. Suppose the spatial point M (xm, ym, zm) corresponds to the mid-point of the given partial line segment on the plane of projection. For each of the plane faces of the polyhedron, determine those which lie on the viewpoint side of M. Only these faces can have any hiding effect on the partial edge under consideration. If the polygons on the plane of projection, corresponding to these faces, enclose the mid point of the partial line segment, then the plane face masks M (xm, ym, zm) and the partial edge from the viewpoint. In this case the depth count is increased by one. (The initial value of depth count is zero).

Thus to determine the depth count of any given partial line segment it is necessary to locate those plane polygons of the polyhedron which lie on the viewpoint side of the corresponding partial edge. The next step is to locate those plane faces from among these whose corresponding polygons on the plane of projection enclose the given partial line segment. The number of plane faces satisfying these two conditions is thus equal to the value of the depth count for this particular partial line segment.

Note that it is a fairly easy matter to determine whether a given plane face lies on the viewpoint side of the mid-point of an edge in space. As was explained in Chapter 2 of this thesis, the viewpoint will always be translated to the origin so that from the constants a,b,c and d which define a given plane

if $dx(axm + bym + czm + d) < 0$

the midpoint M (xm, ym, zm) of the given edge lies on the opposite side of the plane to the viewpoint (origin). Thus if $dx (axm + bym + czm + d) > 0$

M lies on the same side of the plane as the viewpoint.

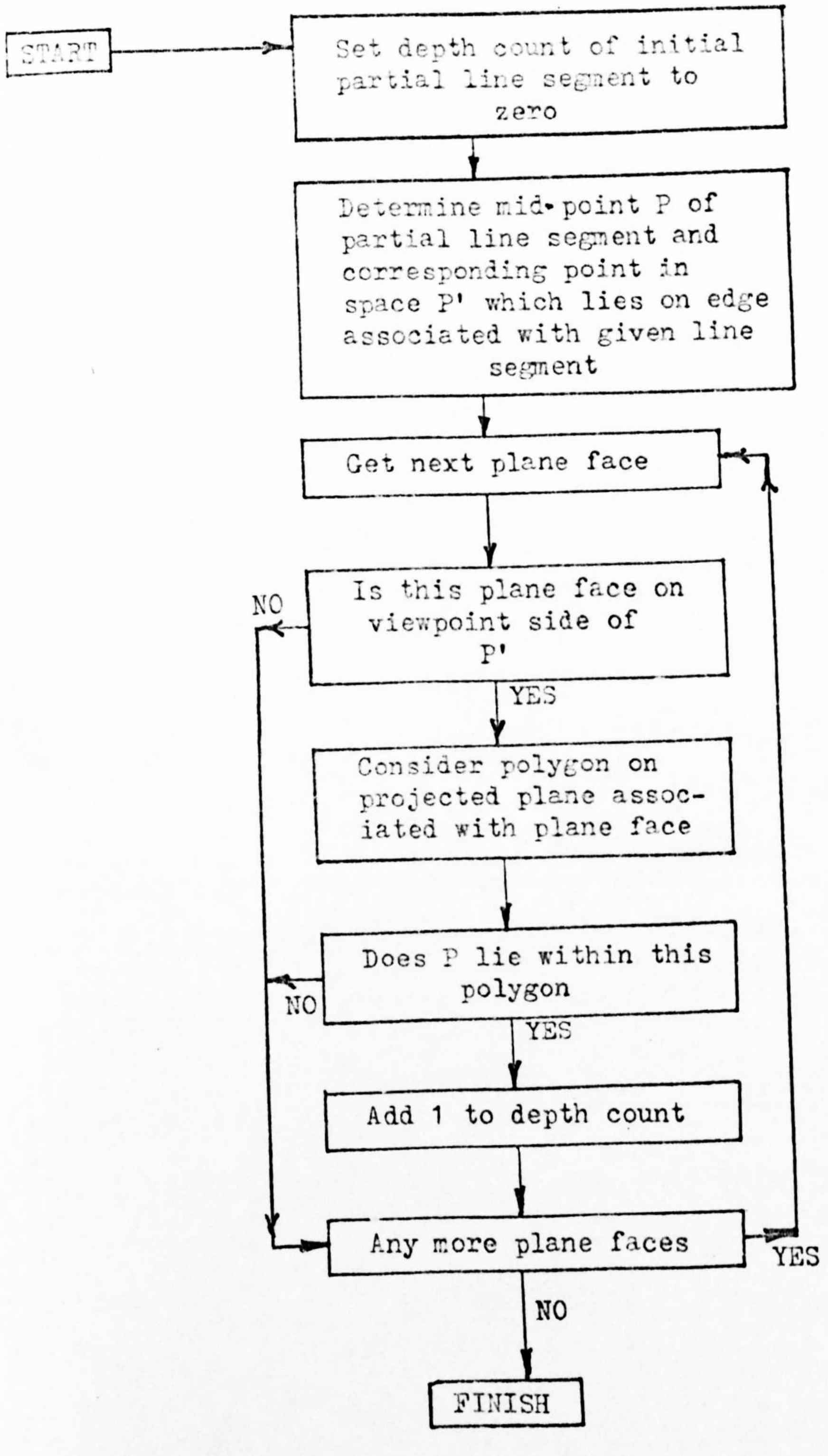
Note

In the actual method used to locate the faces lying on the viewpoint side of a given mid-point M the two faces on which the mid-point lies are not considered.

It is thus not possible for a given mid-point to actually lie on one of the planes considered so that

$$dx\ (axm + bym + czm + d) = 0$$

will never occur.


START
Set depth count of initial partial line segment to zero
Determine mid-point P of partial line segment and corresponding point in space P' which lies on edge associated with given line segment
Get next plane face
Is this plane face on viewpoint side of P'
NO
YES
Consider polygon on projected plane associated with plane face
Does P lie within this polygon
NO
YES
Add 1 to depth count
Any more plane faces
YES
NO
FINISH

FLOW DIAGRAM OF METHOD USED TO DETERMINE INITIAL DEPTH COUNT OF A PARTIAL LINE SEGMENT.

Diagram 1.4.

4.4. Intersection of Line Segments.

One of the most important computational aspects of the algorithm under discussion is the determination of each of the intersections occurring on the plane of projection. It is particularly important that the method used to locate the intersections is efficient since for each view of the polyhedron the intersections between the line segments will probably change and so will have to be determined for each viewpoint.

As explained earlier it is only necessary to consider the real intersections along any given line segment. It should be noted, however, that *every* intersection is a real intersection for one of the intersecting line segments (and a virtual intersection for the other line segment) so that it is necessary to locate all the intersections which occur between the line segments.

It is of course important that each intersection is located only once. Since the two dimensional array edges has elements 'edges (i,1)' and 'edges (i,2)' which are the terminal points of the ith line segment, it is possible to locate each intersection only once by examining those line segments j (with terminal points edges (j,1) and edges (j,2)) for which $j > i$. By this simple organisation each intersection will be found once and once only.

4.4.1. To determine if Two Given Line Segments Intersect.

Since there may be a large number of line segments in a given line drawing it is convenient to have a method which quickly determines if an intersection actually occurs within the lengths of the two give line segments. If an intersection is found to occur then the actual point of intersection can then easily be determined.

Consider the two intersecting line segments 1-2 and 3-4 as shown below

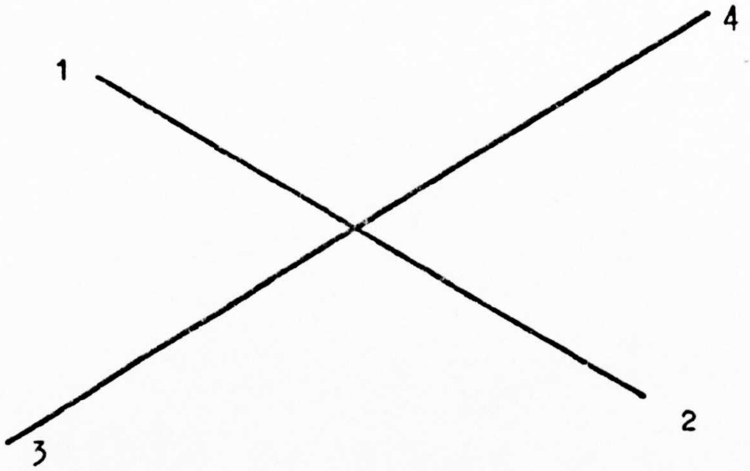


Diagram 4.5.

Suppose the signs of the following four triangles are given as follows (areas)

$$\begin{aligned} \Delta 134 &= a \\ \Delta 342 &= b \\ \Delta 132 &= c \\ \Delta 142 &= d \end{aligned}$$

where a,b,c and d are +1 if the associated area is positive and -1 if the area is negative.

Note that the sign of the area of any triangle with vertices (x1, y1), (x2, y2), x3, y3) is given by the sign of the determinant

$$\begin{vmatrix} x1 & y1 & 1 \\ x2 & y2 & 1 \\ x3 & y3 & 1 \end{vmatrix}$$

Now consider a boolean expression Q given by

$$Q = (a = b) \wedge (c = d)$$

If Q is true then the segments intersect within both their lengths. This is because if a = b, points 1 and 2 lie on either side of segment 3-4 and if c = d points 3 and 4 lie on either side of 1-2. When both conditions are true an intersection must occur.

Note that an intersection is only said to occur between two segments if the intersection point lies within both their lengths. Line segments which intersect at the extreme points, such as 1-12 and 12-10 in Diagram 4.1 are not considered as being valid intersections.

When two line segments have been found to intersect it is a fairly trivial exercise to determine the actual intersection point.

4.4.2. To Determine if the Intersection is Real or Virtual. The next step in the algorithm is to determine if the intersection is real or virtual for the primary line segment under consideration.

Consider the two edges of the polyhedron which correspond to the intersecting line segments under consideration on the projection plane. Suppose the spatial co-ordinates on each of these edges, corresponding to the intersection point, are determined. The edge which lies nearer to the viewpoint (this is always the origin as explained in chapter 2) is that whose corresponding spatial intersection point lies closer to the origin (viewpoint).

If this edge corresponds to the primary line segment under consideration, then the intersection must be virtual, otherwise the intersection is real.

4.4.3. Example. Consider the edges in space corresponding to two intersecting line segments on the projection plane as shown in Diagram 4.6.

Suppose the edges have labels i and j so that the corresponding line segments on the projection plane are also labelled i and j. Let segment i be the primary line segment under consideration.

Suppose it has already been determined that an intersection between line segments i and j occurs. Corresponding to the intersection point (marked k in the diagram) will correspond the points pi and pj in space which lie on the edges i and j respectively. The distances of pi and pj from the origin are calculated and it is found that pj lies nearer to the viewpoint than pi. Edge j therefore lies nearer to the viewpoint (origin) than edge i. The intersection on the projection plane is thus a real intersection since segment i is the primary line segment under consideration. Note that if the primary line segment had been segment j the intersection would be virtual.

Each intersection is located once and then the type of intersection is determined. If this is real then the primary line segment will be divided into partial line segments as explained earlier. If the intersection is virtual, however, then it is necessary to store information relating to the intersection since this will be real when considering the current secondary line segment as a primary line segment. The intersection will not be located again since only secondary line segments j for which j>i (i is primary line segment) are determined.

Thus after considering all valid secondary line segments (for a given primary line segment) it is necessary to check whether the current primary line segment has appeared earlier in a virtual intersection. If this is the case then the intersection will be real for the current primary line segment under consideration.

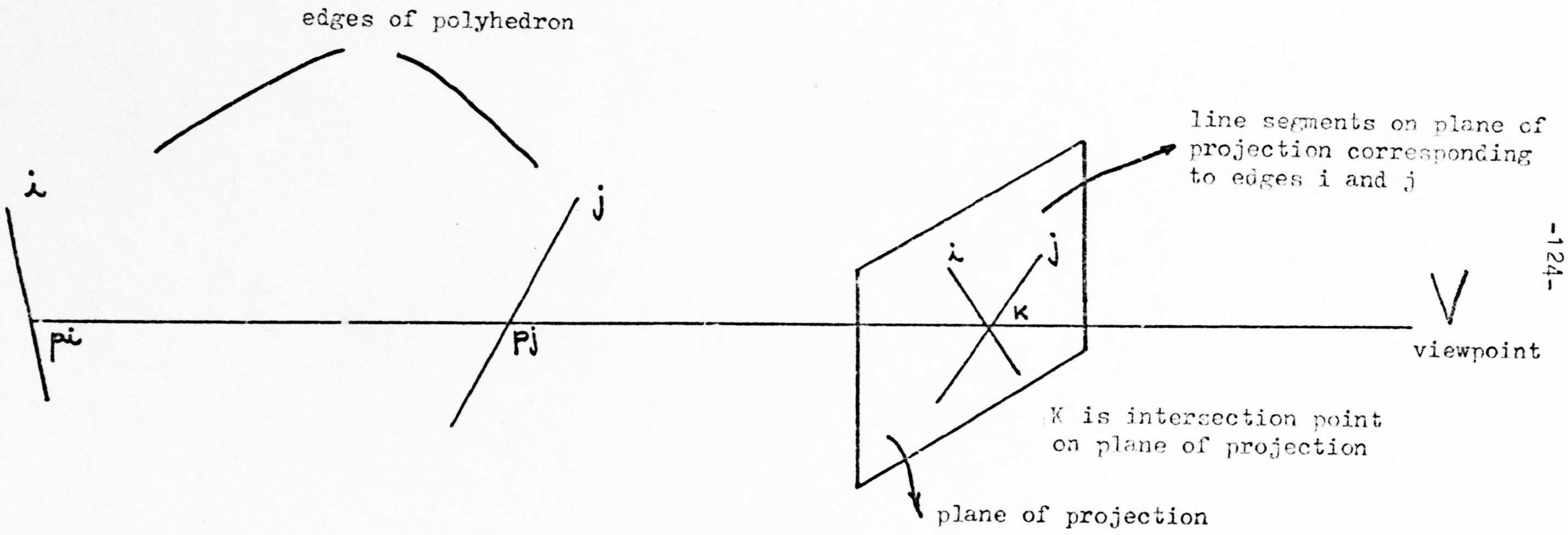
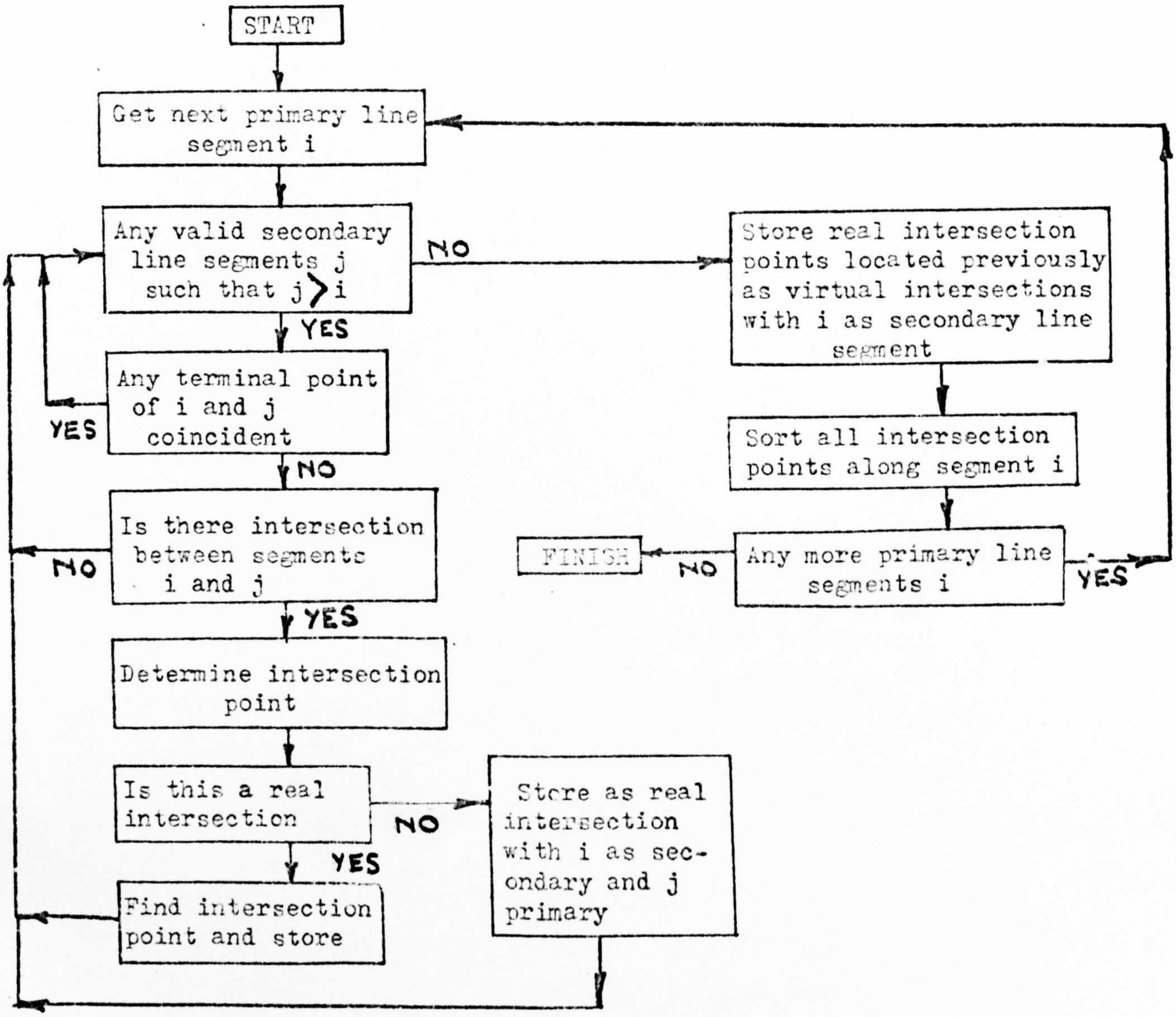

edges of polyhedron
i
j
pi
Pj
line segments on plane of projection corresponding to edges i and j
i
j
K
V
viewpoint
K is intersection point on plane of projection
plane of projection

Diagram 4.6


START
Get next primary line segment i
Any valid secondary line segments j such that j > i
NO
YES
Any terminal point of i and j coincident
YES
NO
Is there intersection between segments i and j
NO
YES
Determine intersection point
Is this a real intersection
NO
YES
Find intersection point and store
Store as real intersection with i as secondary and j primary
Store real intersection points located previously as virtual intersections with i as secondary line segment
Sort all intersection points along segment i
FINISH
NO
Any more primary line segments i
YES

FLOW DIAGRAM OF METHOD USED TO LOCATE REAL INTERSECTIONS AND ASSOCIATED PARTIAL LINE SEGMENTS ON PROJECTION PLANE

Diagram 4.7

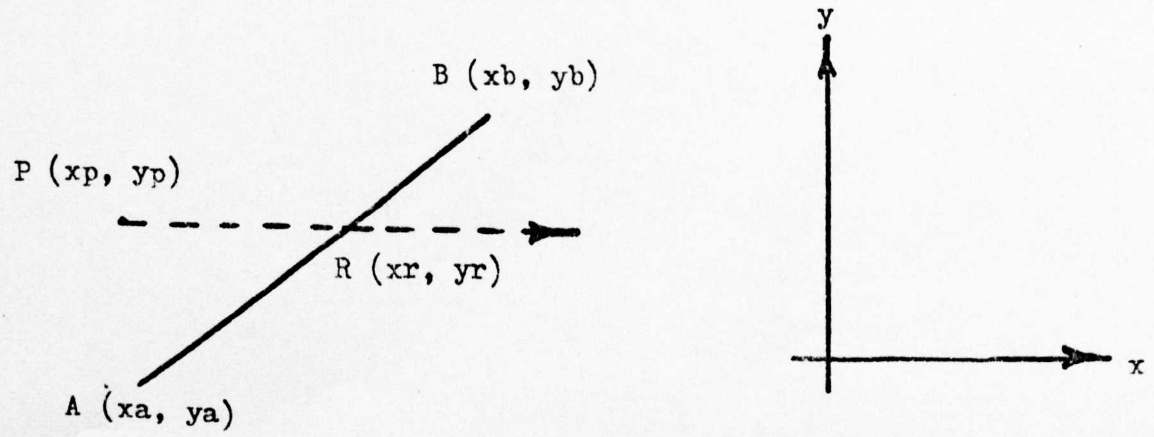
4.5. To Determine if a Point Lies Within a Given Polygon.

To determine the depth counts of the partial line segments associated with a given line segment it is necessary initially to determine the depth counts of one of the partial line segments. From this value of depth count it is then possible to determine the depth counts of adjacent partial line segments. The initial determination and the changes in the depth count necessitates a method of deciding if a point lies within a given polygon.

The method which has been utilised depends on the fact that an extended line drawn in any direction from the given point will intersect the segments of the polygon an odd number of times if the point lies within the polygon and an even number of times (or not at all) if the point lies outside the polygon.

Suppose the point under consideration is P (xp,yp) and consider the line to be drawn in the positive x direction on the plane of projection. It is necessary to calculate the number of times this line intersects the segments of the polygon under consideration.

Consider any line segment of the polygon with terminal points A (xa, ya) and B (xb, yb) as shown in Diagram 4.8,



The first step is to determine whether P lies within the y co-ordinates of the line segment. Consider the boolean expression E given by

$$E = yp \leqslant ya \quad \text{eqv} \quad yp > yb$$

where eqv is a mnemonic for the boolean 'equivalent'.

If E is true a second test must now be made to determine whether P lies to the left of the point R (xr,yr) which is the point of intersection of the extended line (drawn in the positive x direction) with the given line segment (note that yp = yr).

From Diagram 4.8 it is seen that xr is given by

$$(yr-ya)(xb-xa) = (yb-ya)(xr-xa)$$

so that

$$xr = (yr-ya)(xb-xa)/(yb-ya) + xa$$

If xr > xp then P lies to the left of the line segment and the line drawn in the positive x direction intersects the line segment.

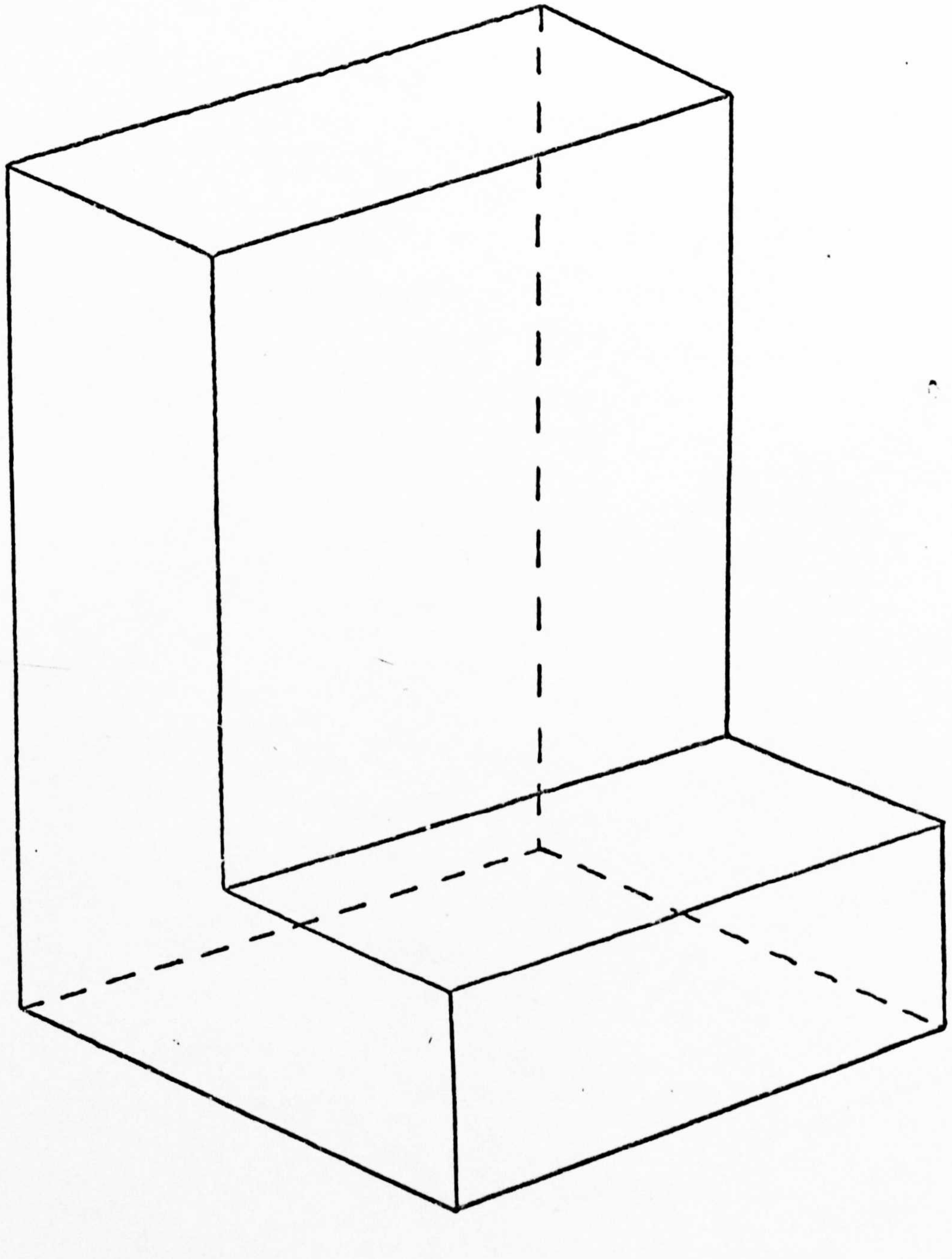
If xr < xp then P lies to the right of the line segment and no intersection takes place.

NOTE

The method which has been explained in this Chapter is a simple but convenient method of locating the visible and hidden edges of any given non-convex polyhedron.

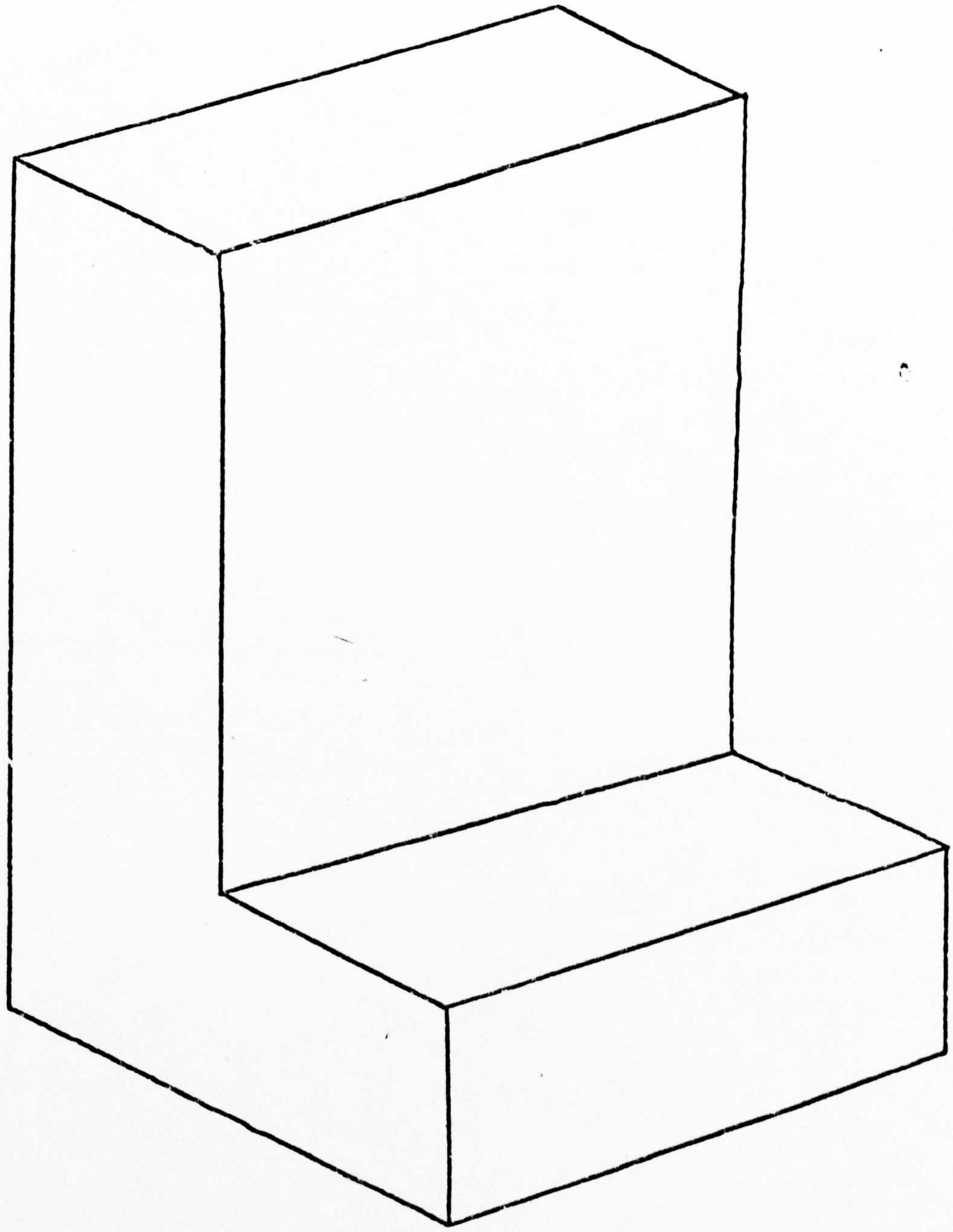
The method as presented is rather restricted however and does not overcome all the problems. It was felt that because of its simplicity, it possesses advantages over those methods which approach the problem in a more rigorous way.

A number of examples produced on the graph plotter are included in the text.



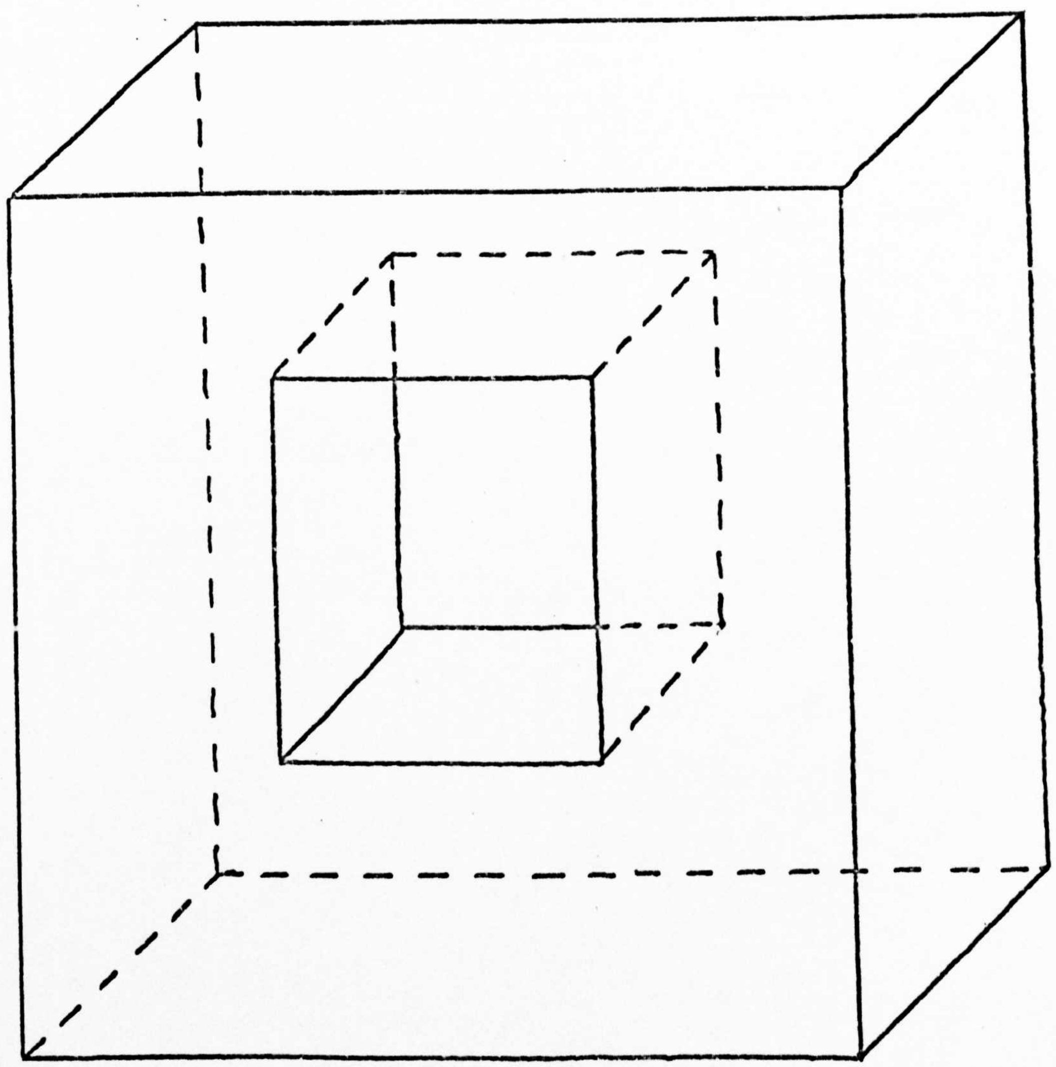
NON-CONVEX POLYHEDRON. HIDDEN LINES DOTTED.

Figure 18



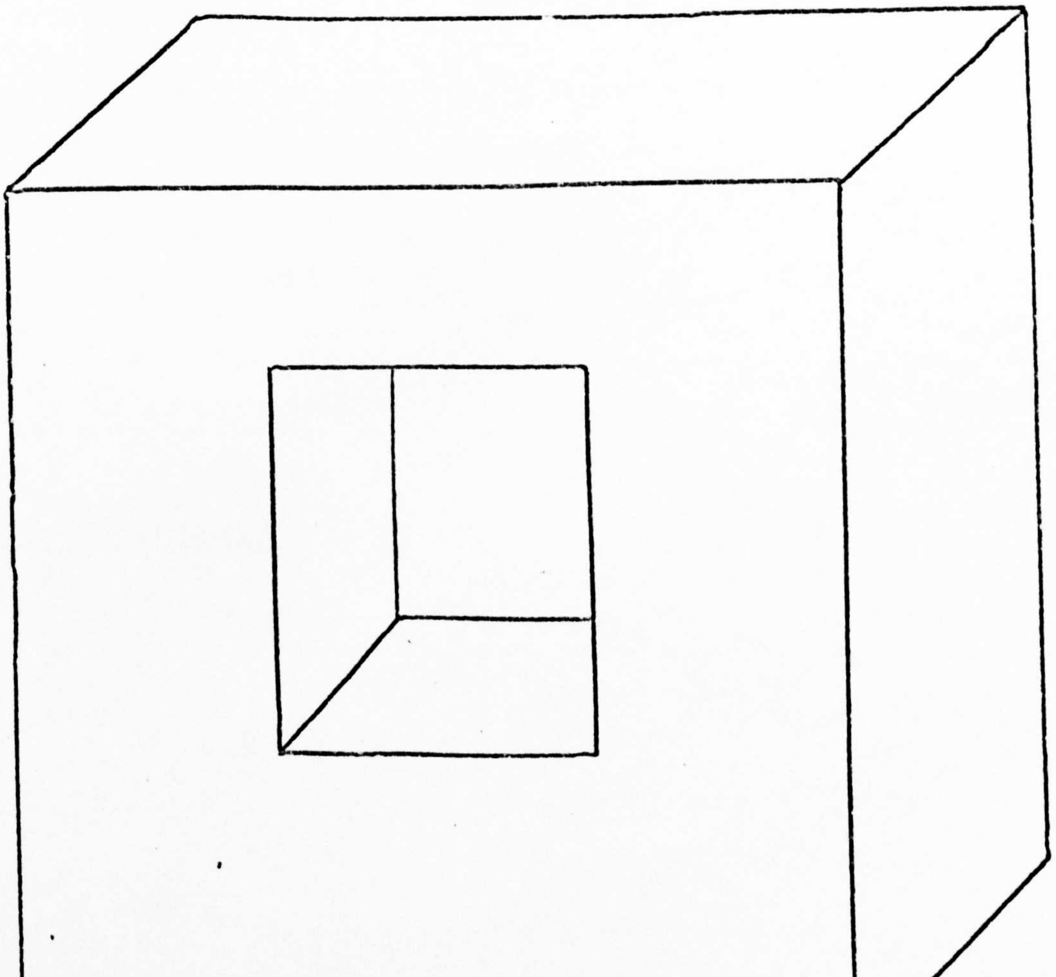
NON-CONVEX POLYHEDRON. HIDDEN LINES ERASED.

Figure 19



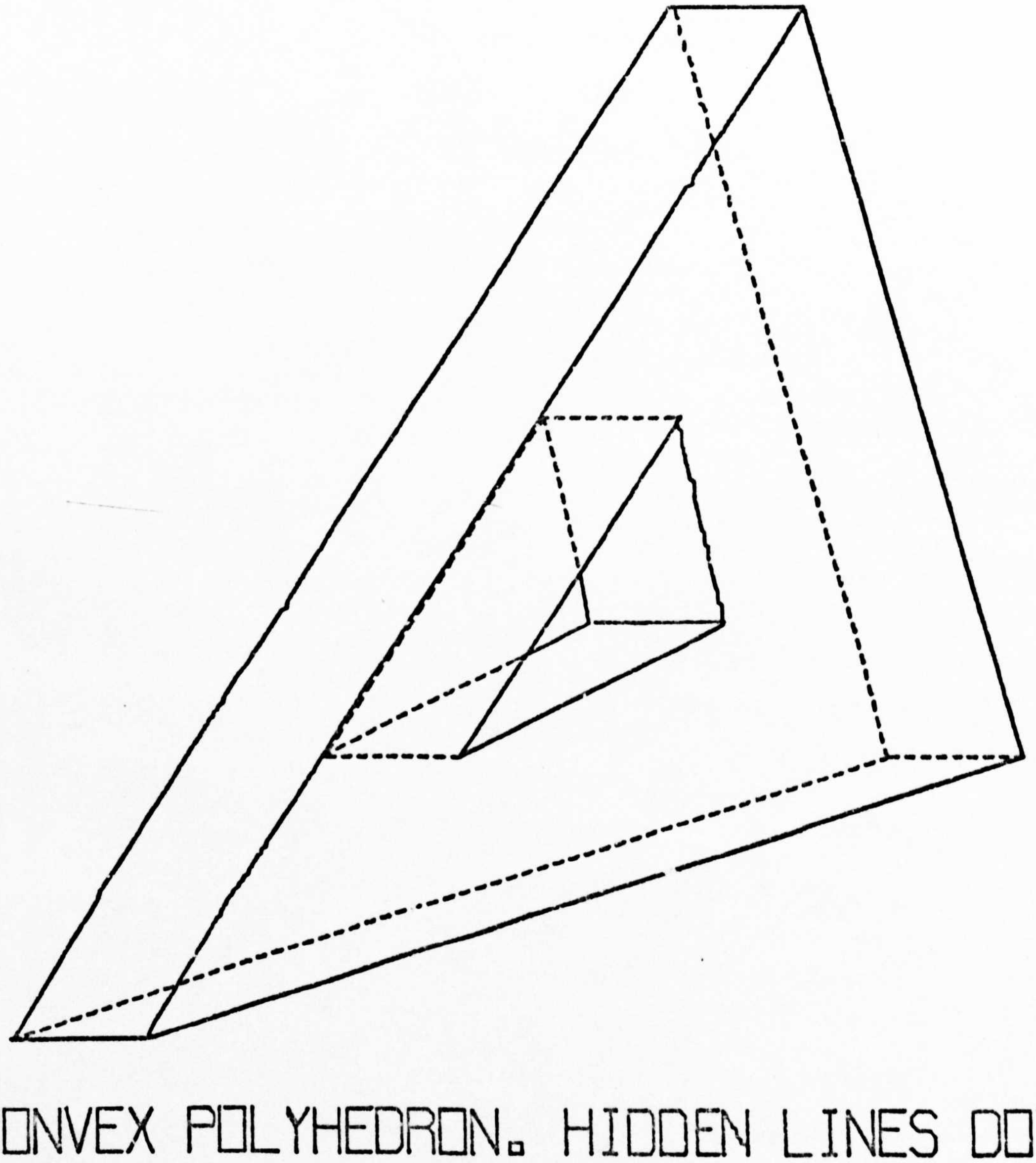
NON-CONVEX POLYHEDRON. HIDDEN LINES DOTTED.

Figure 20



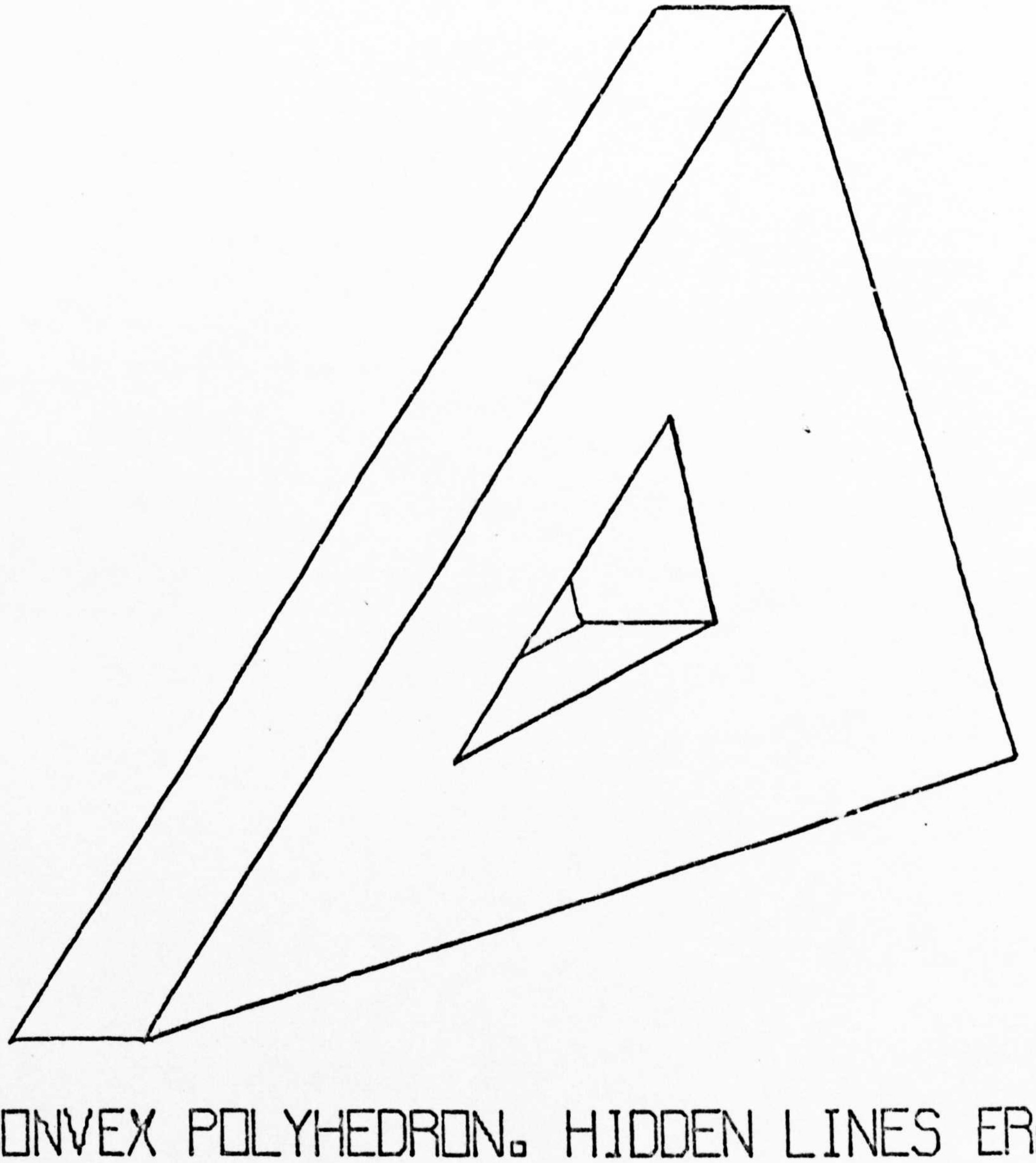
NON-CONVEX POLYHEDRON. HIDDEN LINES ERASED.

Figure 21



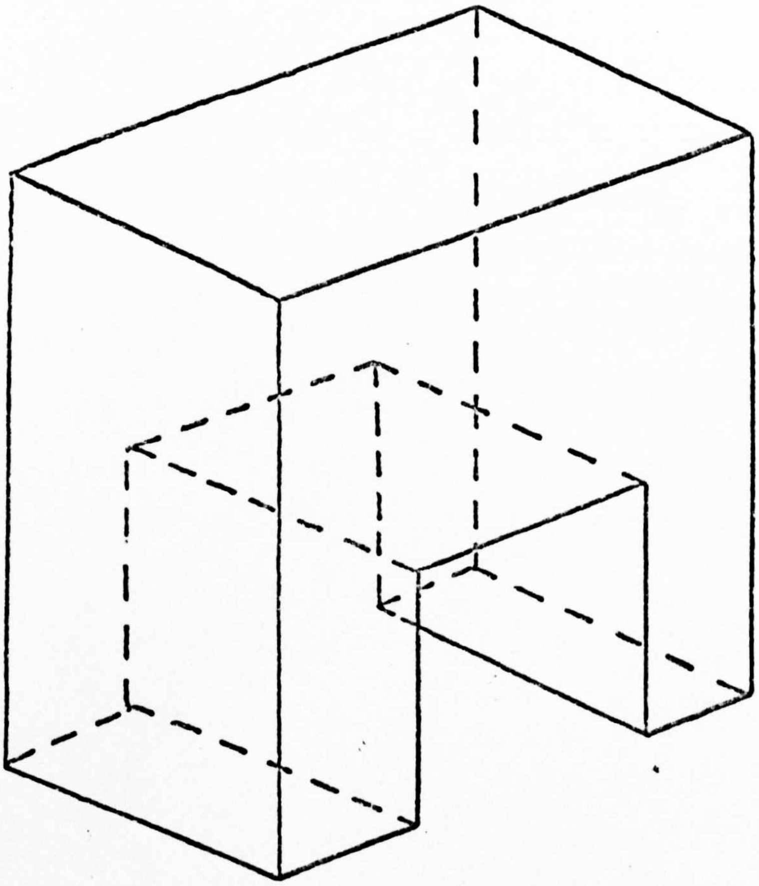
NON-CONVEX POLYHEDRON. HIDDEN LINES DOTTED.

Figure 22



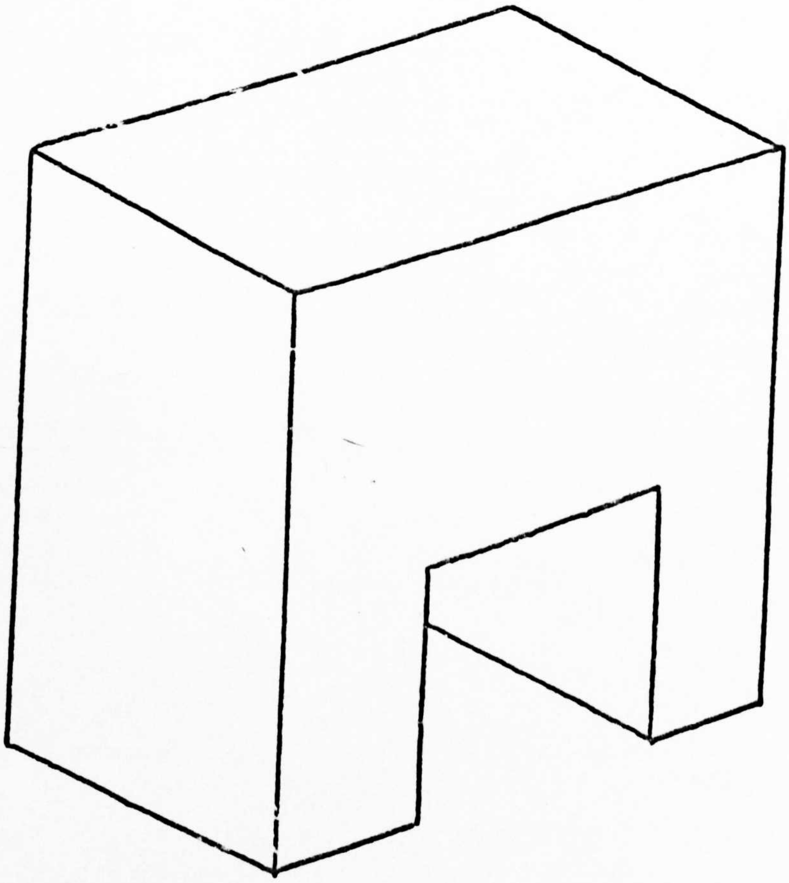
NON-CONVEX POLYHEDRON. HIDDEN LINES ERASED.

Figure 23



NON-CONVEX POLYHEDRON. HIDDEN LINES DOTTED.

Figure 24



NON-CONVEX POLYHEDRON. HIDDEN LINES ERASED.

Figure 25

CHAPTER 5

COMPUTER ORGANISATION OF LINE DRAWINGS

Introduction to the concepts involved in the drawing of any given connected graph.

CHAPTER 5

COMPUTER ORGANISATION OF LINE DRAWINGS.

5.1. Introduction.

The objective of most of the work in computer graphics is concerned with the production of a line drawing. This may be some two dimensional representation of a three dimensional solid, examples of which appear in earlier chapters, or indeed may simply be a reproduction of some two dimensional figure. There are, of course, many ways in which a three dimensional object may be represented in two dimensions, the most popular of which for draughtsmen is by constructing plan views and elevations of the object. These views can be quickly produced in computer aided design by using orthographic projections of the solids with the viewpoint directly above the object for a plan view and to the side of it for side elevations.

Perspective views of objects as illustrated in this thesis may be produced fairly easily and recent research on the hidden line problem has enabled the representation of objects in real time to be almost a reality. Whatever the representations may be, it is fair to say that the most popular method of displaying information concerning three dimensional objects and figures is by the line drawing.

It is, in fact, only in the last couple of years or so that efforts have been initiated to represent solid objects using the raster type of display. Probably the most significant contribution to date in this type of display is that of John Warnock of the University of Utah. The line drawing is particularly suited to the graph plotter and the calligraphic display which can both be programmed to join up points by straight lines. Indeed the more modern calligraphic displays have associated analogue circuits which produce 'ramp' wave forms and so considerably speed up the process of drawing lines.

In papers devoted to topics which concern the production of line drawings no mention ever seems to be made of the methods by which the drawings are drawn by the 'pen' of the display. Indeed the problem seems not to exist in the eyes of many researchers in the computer graphics field. It would seem at present that the methods of producing the line drawings are almost wholly dependent on the form of the output drawing and so this suggests that one must know what the drawing looks like before the method is decided. When representing a given object from a number of different viewpoints, for example, it is not possible to know the form of the output drawing.

It seems of importance, therefore, to investigate methods by which any given line drawing can be conveniently represented in computer storage and then be manipulated in such a way that it is in a form suitable to be drawn with the pen of the display. In addition, it is desirable, particularly with a graph plotter, to keep the amount of pen movement to an acceptable minimum. In this respect it has been necessary to define a drawing efficiency which relates the amount of movement with the pen in a raised position to the amount of pen down movement for any given line drawing. This allows a comparison of the efficiency of the various ways in which a given line drawing can be connected.

So that the problem can be tackled in a logical manner and for the sake of completeness, it was found necessary to define a line drawing as follows.

5.1.1. Line Drawing. A line drawing will be considered to be a connected graph which has co-ordinates associated with each of the nodes.

By considering a line drawing as being a graph it has been possible to utilise some of the related topics in graph theory. In particular, the theory asssociated with Eulerian chains and cycles has been useful. It should also be noted that no generality is lost by considering a line drawing to be a connected graph since any drawing can be considered as being a collection of connected graphs.

Since the following sections are of particular importance in the theory to be developed later in this thesis, the work has been treated in an exhaustive manner.

5.2. Computer Representation of Graphs.

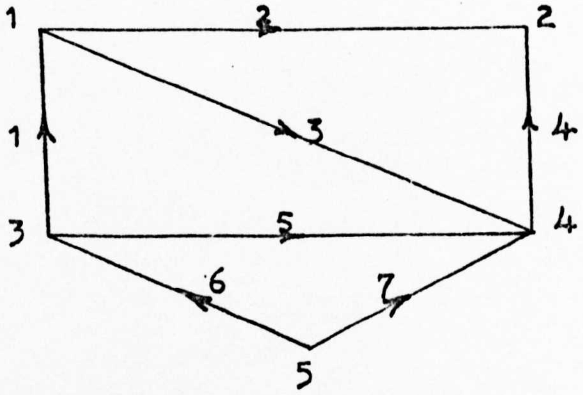
Since a line drawing is to be considered to be a graph then it is clear that line drawings can be represented in the computer in much the same way as graphs. As will become clearer later in this chapter a line drawing at different stages in its manipulation within the computer is considered as being both a directed graph and a non-directed graph. The final line drawing on the display is non-directed in that no particular direction need be assocated with any of the segments. However, when the drawing is being drawn by the pen of the display each of the segments necessarily has a direction associated with it which corresponds to the direction of the pen movement.

The more important methods of representing directed graphs within the computer will be elucidated for the sake of completeness.

5.2.1. Connection Matrix. Suppose the graph has n nodes and consider a n x n matrix m associated with the graph. Every element of the connection matrix m is either 0 or 1 depending on the conditions set out below

$$mij = \begin{cases} 1 \text{ if a segment connects node i to node j} \\ 0 \text{ otherwise} \end{cases}$$

As a simple example consider the connection matrix of the following graph



the connection matrix is given by

$$m = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \qquad \text{\_\_\_\_\_\_\_\_\_\_\_\_(1)}$$

Note that if a 1 is present along the principal diagonal at mii this indicates that there is a loop at the vertex i.

5.2.2. Incidence Matrix. The incidence matrix associated with any graph of n nodes and s segments is a nxs matrix p in which rows represent nodes and columns represent segments. Each element of p is either 1, -1 or 0 depending on the following conditions

$$pij = \begin{cases} 1 \text{ if ith node is initial node of jth segment} \\ -1 \text{ if ith node is final node of the jth segment} \\ 0 \text{ otherwise} \end{cases}$$

It is clear that the incidence matrix requires more storage than the associated matrix if $s > n$ as is likely for most line drawings.

The incidence matrix for the above graph is as follows

$$p = \begin{bmatrix} -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad \underline{\qquad\qquad\qquad}(2)$$

5.2.3. List of Line Segments. Perhaps the simplest way of representing a directed graph is by a list of ordered pairs of nodes. The ith pair of nodes correspond to the ith segment.

For the example given earlier the representation is

31
12
14
42
34
53
34

In addition to the information associated with the terminal nodes of the segments it is necessary for a given line drawing to supply the cartesian co-ordinates or some other co-ordinates for each of the nodes.

The representation which has been used in this thesis is the simple one of a list of line segments and the cartesian co-ordinate system has been used to determine the position of each of the nodes.

5.3. Essential Definitions.

It is convenient at this stage to define the more relevant terms in graph theory to be used.

5.3.1. Graph. A graph G (V,X) consists of a finite set of nodes V$\{vi\}$ and a set of segments X$\{xj\}$ of non-directed pairs of distinct nodes of V.

5.3.2. Finite Graph. A finite graph is a graph having a finite number of segments and nodes.

5.3.3. Chain and Cycle. Consider a graph G and let v_1 v_2 ....vn be a succession of nodes such that there exist segments xi joining Vi and V_{i+1}, i=1,2.....n-1. The succession of segments x_1 x_2......x_{n-1} is called a chain and if the terminal nodes coincide the succession of segments is called a cycle.

If there exists a chain between any two nodes then the nodes are said to be connected.

5.3.4. Connected Graph. A graph G is connected if every pair of nodes in G is connected.

5.3.5. *Directed Graph.* A directed graph is a graph which has a direction associated with each segment.

5.3.6. *Degree of Node.* The degree of some node v is the number of segments incident to it. A node of odd degree has an odd number of segments incident to it and similarly a node of even degree has an even number of segments incident to it.

5.3.7. *Subgraph.* If, in some Graph G, one or more nodes are omitted, together with the segments linking these nodes, the remaining portion of the graph is a subgraph of G.

5.3.8. *Matching.* In any graph G (V,X) a set of edges $M \subset X$ is called a *matching* in G if no two edges of M are adjacent.

A matching in G will be called a *maximal* matching if no matching has higher cardinality. A vertex is said to be *covered* or *exposed* depending on whether or not an edge of M meets it.

If every vertex is covered the matching is said to be *perfect.*

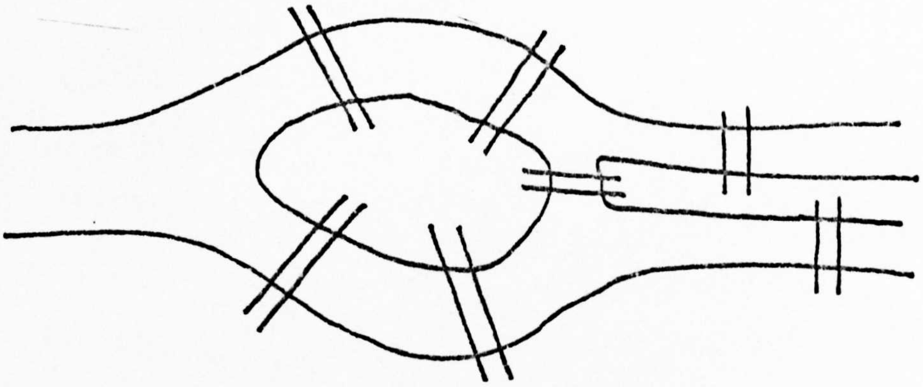
5.3.9. *Regular Graph.* A graph of n nodes is said to be regular if the degrees of each of its n nodes are the same.

5.3.10. *Planar Graph.* A graph G is called planar if it can be drawn inthe plane in such a manner that all intersections of edges are vertices of G.

5.4. Eulerian Cycles and Chains.

The importance of Eulerian cycles and chains lies in the fact that they can be drawn without it being necessary to raise the pen from the paper and without going over any line more than once.

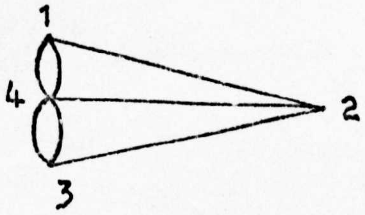
Cycles and chains of this kind were discovered by the celebrated mathematician Euler in his solution of the famous Königsberg Bridge Problem. The problem was to determine whether it was possible to cross each of the seven bridges over the river Kaliningrad (arranged as shown below) once and once only.



The answer was in fact, no, and Euler proved this to be so in 1736.

Euler's solution to this problem did, in fact, initiate the mathematical theory of graphs.

The above drawing can be represented by a connected graph as shown below.



The graph has 4 nodes and the problem is how does one recognise whether a given graph possesses an Eulerian chain or cycle.

The definitions which follow serve to answer this question.

5.4.1. Eulerian Cycle. If a graph G is finite and connected and if all the nodes are of even degree then G possesses an Eulerian cycle.

5.4.2. Eulerian Chain. If a graph G is finite and connected and if the number of nodes of odd degree is two then G possesses an Eulerian chain, the terminal nodes of which are the two odd nodes.

5.5. Essential Proofs.

In the scheme of organising the line drawings it is first necessary to digress on the definitions given for the Eulerian cycles and chains of the preceding section and to prove some of the topics which were introduced.

5.5.1. Lemma 1: Any graph possesses an even number of odd degree nodes.

Proof :

Suppose there are Nm nodes of degree m in the graph and further suppose there to be k nodes and s segments.

Then

$$\sum_{m=1}^{k} m.Nm = 2s \_\_\_\_\_\_\_\_ (3)$$

since the sum represents a count of the number of terminal nodes of the segments.

$$\therefore \sum_{m = 1,3,5, \dots\dots} m.\ Nm = 2s - \sum_{m = 2,4,6\dots\dots} m.\ Nm \qquad (4)$$

The right hand side of equation 4 must be even so that

$\sum_{m = 1,3,5.} m.\ Nm$ is even and the lemma is proved

5.5.2. Theorem 1. If a graph G is finite and connected and the degree of each node is even then there exists a cycle in G which goes through every segment once and once only.

Proof:

That G is connected is evidently necessary.

Consider a connected graph G and suppose that a chain is traced out from some initial node p to an arbitrary node q such that every time a segment is traversed it is deleted from the graph. If p does not coincide with q then q must be of odd degree so that there must always be an available segment along which to travel. The process can thus only come to an end by returning to p. It is therefore always possible to trace a cycle s from some node p.

The segments, if any, which have not been deleted from G will form k subgraphs b_1, b_2, b_3, ...b_k each of which will be connected and contain nodes of even degree. Since G was connected, s must contain at least one node n_i (i = 1,2.......k) from each of the k subgraphs.

It will thus be possible, at each n_i, to trace out a cycle ci having ni as the initial and final node. The segments belonging to this cycle are again deleted. If ci is not an Eulerian cycle the same process can be repeated and the original cycle s can thus be extended until all the original segments of G are contained in s which will then be the Eulerian cycle in G.

5.5.3. Theorem 2. If a graph G has 2n odd degree nodes there are n chains that together traverse all the segments of G once and once only. Each of the chains begins at an odd node and finishes at an odd node.

Proof:

Suppose the odd nodes are divided into n pairs since this is possible according to Lemma 1.

Now suppose a graph H is formed by adding to G n segments, each of which joins a pair of the odd nodes such that the odd nodes are covered by the n segments. H is thus a connected graph whose nodes are of even degree.

Theorem 1 (just proved) states that there must be a cycle c in H which traverses all segments once and once only. Now suppose the segments just added are now removed. The remaining segments must consist of n chains which together traverse the remaining segments. In addition, since the removed segments of H had terminal nodes of odd degree then these n chains must start and finish at odd nodes.

Since n of the nodes must be the end nodes of n chains it is thus not possible to have less than n chains to cover G.

5.6. General Method.

To draw any connected graph $G(X,Y)$ by a continuous sequence of pen movement it would be particularly convenient if it were possible to connect every segment of Y once and once only and also be able to draw G by continuous movement of the pen without it being necessary to raise the pen from the paper at any stage. If this were possible the cardinality of the set $Z = \{ z : z \in X, z$ of odd degree$\}$ must be either zero or two by Theorems 1 and 2.

If Z is null the graph contains an Eulerian cycle and all nodes of G are of even degree. It is also possible to choose any of the nodes of G as the initial and final nodes of the directed cycle since these coincide.

If $|Z|$, the cardinality of Z, is two, G is covered by an Eulerian chain and it is possible to connect G by a directed Eulerian chain such that the terminal nodes belong to Z. In both the above cases the total distance travelled by the pen is the least possible to draw the graph G and is thus equal to the sum of the distances associated with each of the segments $y \in Y$.

Now consider what can be done if Z exceeds two as is likely in most cases. It is possible to choose a set W of segments which link together the $(z /2)$ Eulerian chains of G (X,Y) to form an Eulerian cycle $G'(X,Y \cup W)$ if W is a perfect matching of the nodes $z \in Z$. If W is a perfect matching then all of the nodes in Z are covered so that the degree of the nodes in G' are all even since one has been added to each one. Note that it is possible to produce an

Eulerian chain from G if the cardinality of W is $Z_{/2}-1$ and the matching of W is not maximal. In this case all but two of the nodes in Z are covered so that the two exposed nodes will be the extreme nodes of the direct Eulerian path obtained from G'

It should be noted from Lemma 1 that it is always possible to produce a perfect matching W in Z since the cardinality of Z is always even.

Since the segments $w\epsilon W$ are added to G they will be connected with the pen in a raised position so that the distances associated with the pen up segments will be additional or 'excess' pen movement.

After the selection of the pen-up segments it is necessary to sort the segments belonging to $Y\cup W$ so that a directed Eulerian cycle is produced and the graph can be drawn by continuous movement of the pen, care being taken to raise the pen when traversing segments ϵW.

It is important to choose the pen-up segments such that the excess pen movement associated with them is minimal. If n is the cardinality of Z, the number of ways of choosing the n/2 segments of W from the nx(n-1)/2 feasible ones is given by

$$\frac{n!}{\left(\frac{n}{2}\right)!\,2^{\frac{n}{2}}}$$

The problem is obviously combinatorial in nature and will be referred to as the 'Pen-up Problem'. For each of the choices of pen up segments there will exist an amount of pen up movement which can be considered to be excess pen movement in that no line of the original graph is being drawn.

In this respect it is necessary to define a Drawing Efficiency Nd which relates the total pen up and pen down movement to draw a given line drawing which has co-ordinates associated with each of the nodes and therefore distances associated with any two nodes. The notion of drawing efficiency will be explained in the following section.

It is also necessary to devise some method by which the Pen Up Problem can be quickly solved to give a good feasible solution.

5.7. Drawing Efficiency of a Line Drawing.

Since it is desirable that the pen should connect the line segments in an ordered sequence such that each line segment is drawn once and once only, it is apparent that the line drawing should be made into an Eulerian cycle by the addition of a number of pen up lines. If there are n nodes of odd degree in the line drawing then it is necessary to select n/2 pen up lines which will cover the n odd points so that the n/2 chains can be linked together to form one continuous sequence of pen movement.

The selection of the pen up lines is, of course, combinatorial in nature, but it is desirable that the total distance associated with the pen up segments should be kept small.

If there are s segments in the given line drawing and di represents some measure of the length of the ith segment the total distance which has to be plotted, pd, is given by

$$pd = \sum_{i=1}^{s} di \quad \text{\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} (5)$$

If gi represents, in a similar manner, the length of the ith pen up segment then the total distance which the pen must travel in excess of pd is pu and is given by

$$pu = \sum_{i=1}^{n/2} gi \quad \text{\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} (6)$$

It should be clear that pu should be kept as small as possible for the connection of the line drawing to be efficient. Indeed, if the line drawing has no odd points then pu is of course zero and no excess pen movement is required so that in this case one can say that the connection is as efficient as possible.

It is clear that it would be very convenient to define some factor which gives a measure of the efficiency with which a line drawing has been connected. In this respect it is convenient to define a drawing efficiency Nd which relates pu to pd.

The drawing efficiency is defined as follows :

$$Nd = \frac{pd}{pu + pd} \quad \text{\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} (7)$$

If the line drawing has no points of odd degree then it is an Eulerian cycle and no pen up lines are required so that pu = 0 and the drawing efficiency is 100%.

An Eulerian cycle or chain can thus be defined as being a finite connected graph with a drawing efficiency of 100%.

For any given line drawing, pd is constant and as the pen up distance pu increases the drawing efficiency Nd decreases.

The variation of Nd with pu is shown in graphical form in Diagram 5.1. Since there is a choice of pen up line segments which gives a minimum value of pu, it is possible to relate this choice to Nd and aseociate with it an Optimal Drawing Efficiency defined as follows :

The Optimal Drawing Efficiency of a line drawing is the highest drawing efficiency possible for the given drawing and corresponds to pu being chosen so that it is minimal.

5.8. Organisation Scheme.

The information required to represent any line drawing consists of the number of nodes and the cartesian co-ordinates of each, together with the number of segments and a list of the terminal points of each segment. This is the representation which will be used to describe any line drawing. The information is similar to that required to represent any graph but in addition the co-ordinates of each point are required.

As explained earlier it is required to add pen up segments to the line drawing which will span the odd points of the graph so that each point will be of even degree and the line drawing will consist of an Eulerian cycle. So that the line drawing can be connected by an ordered sequence of pen movement it is necessary to order the segments so that the two dimensional array'list' of the terminal points of the s line segments are related as follows :

$$\text{list}\left[i,2\right] = \text{list}\left[i+1,1\right] \quad i = 1,2 \ldots.(s-1) \qquad (8)$$

and $$\text{list}\left[1,1\right] = \text{list}\left[s,2\right] \qquad (9)$$

since the initial and final nodes of the cycle are the same.

5.8.1. <u>Domino Graph</u>.. Consider for any line drawing the odd points. Suppose there are n in number, then it is required to add n/2 pen up segments such that each odd point is associated with one pen up segment. It is perhaps advisable to define the form of the graph consisting of the pen up lines and the odd degree nodes as a domino graph.

A <u>Domino graph</u> is a perfect matching on the complete graph of the n odd nodes in the original graph G and the nX(n-1)/2 possible joins between them.

For any given line drawing it is necessary to add a domino graph linking the odd points of it so that each of the points of the line drawing is even and so forms an Eulerian cycle.

From the input information for any line drawing it is an easy enough exercise to locate which points are of odd degree by the number of times each point is associated with a line segment. Having located the odd degree points it is now necessary to choose the spanning segments of the associated domino graph. The choice need not of course be optimal but a good quick choice of the pen up lines, which tends to minimise the total pen up distance, is required. Once the pen up lines have been selected it is necessary to sort the line segments into a directed Eulerian cycle to be drawn by the pen.

Note that it is of course possible to add more than n/2 pen up lines but since this would only increase pu it is obvious that the minimum number of pen up lines should be added since this tends to minimise pu.

5.9. Coincident Points.

Since the number of feasible solutions to the pen up problem rises rapidly as n increases, it is obviously worthwhile initially to see if n cannot be reduced. It is possible that two or more points of the line drawing are coincident and it is obviously necessary to investigate the consequences of coincident points.

Consider that c of the points are coincident in that the distance between any two of them is less than some small value δs. Since the resolution of the graph plotter is only 0.1 mm one can consider any points for which the scaled distance is less than 0.1 mm to be coincident, for then nothing will be lost in the plotting.

For each of the coincident points it will be necessary to change the labelling of the associated segments so that the coincident points are represented by one label only. This is a trivial exercise.

It is necessary for the general algorithm to find which of the points are of odd degree and it is thus a simple matter to ascertain the even points. The degree of each of the points is therefore known. Consider that the c coincident points are formed by a number e of even points and a number d of odd points. No matter what the value of e the addition of the even segments to the coincident point will be an even number of segments. Thus to ascertain the degree of the coincident point it is only necessary to consider the value of d.

If d is odd then the addition of the d odd degree nodes will produce an odd number of segments since the product of two odd numbers is odd. The degree of the coincident point will in this case be odd and n will be reduced by (d-1).

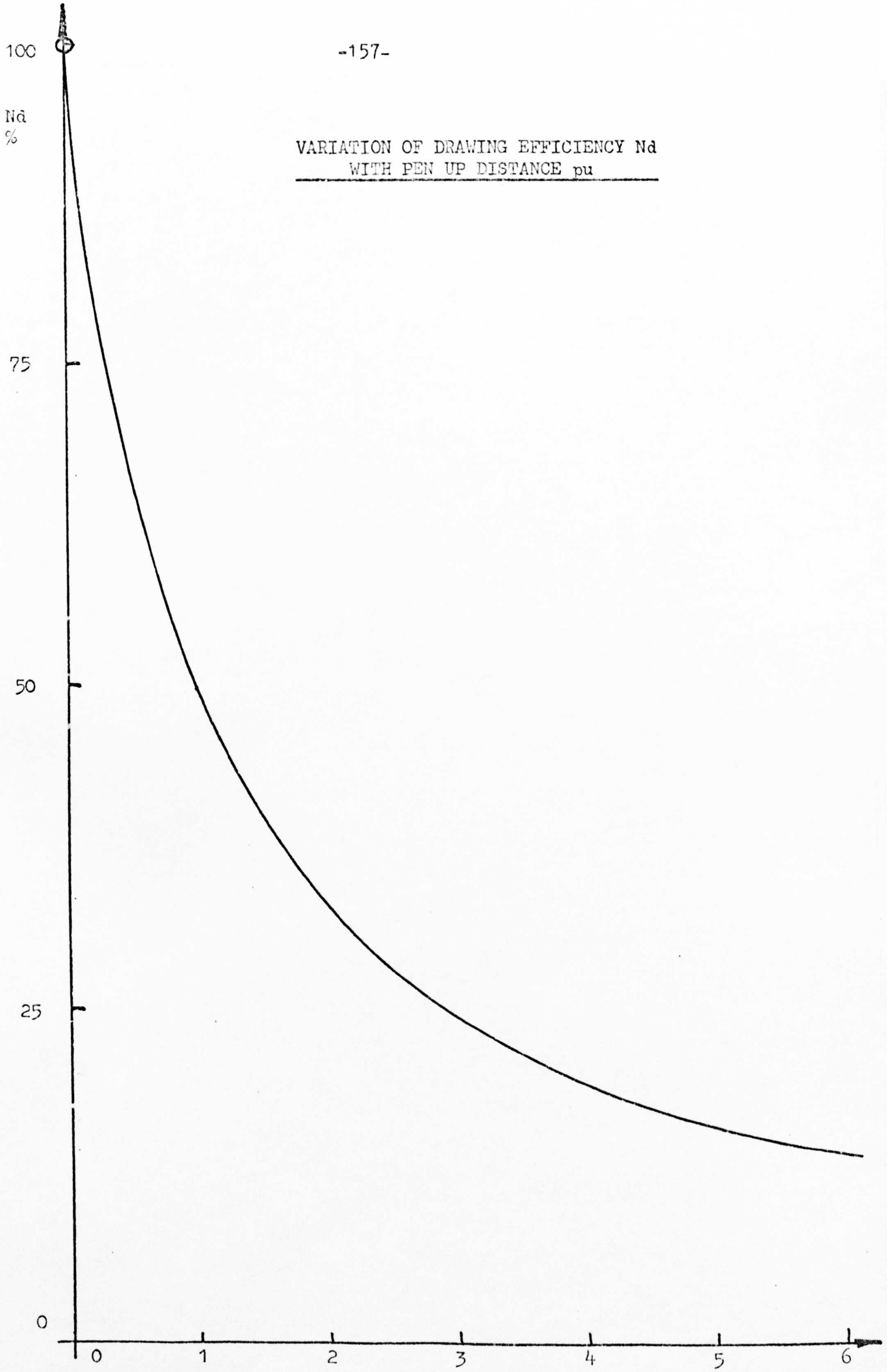
If d is even the degree of the coincident point will be even and so n will be reduced by d.

The above scheme is represented in tabular form below.

No. of odd degree nodes in Drg. d.	Degree of coincident point	No. of odd degree points saved
odd	odd	d-1
even	even	d

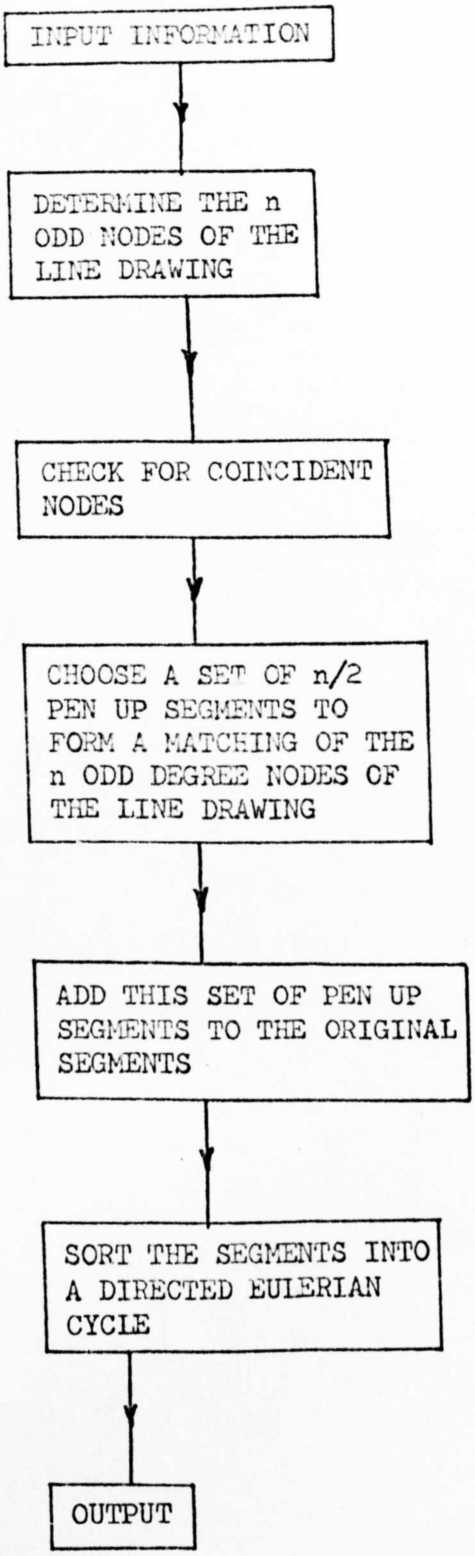
Note that the reduction in the number of odd degree points is always even so that the number of odd points in the graph remains even,

It is possible after locating the coincident points that the terminal points of a line segment are coincident. If this is the case then the segment can be deleted so that the number of segments is reduced by one.



units of pu are multiples of pd

Diagram 5.1


INPUT INFORMATION
DETERMINE THE n ODD NODES OF THE LINE DRAWING
CHECK FOR COINCIDENT NODES
CHOOSE A SET OF n/2 PEN UP SEGMENTS TO FORM A MATCHING OF THE n ODD DEGREE NODES OF THE LINE DRAWING
ADD THIS SET OF PEN UP SEGMENTS TO THE ORIGINAL SEGMENTS
SORT THE SEGMENTS INTO A DIRECTED EULERIAN CYCLE
OUTPUT

SCHEME OF ORGANISATION FOR ANY GIVEN LINE DRAWING.

Diagram 5.2

5.10. Choosing the Pen Up Segments.

Suppose the given graph has n nodes of odd degree. It is required to select n/2 segments which cover the odd nodes so that if ui is a measure of the length of the ith selected segment $\sum_{i=1}^{n/2} ui$ is minimised.

These segments will be the pen up lines to be added to the original segments of the line drawing. A good quick choice of the segments is required since the efficiency of the scheme will be lost if too much C.P.U. time of the computer is expended in the organisation of it.

The problem ~~o~~of finding the optimal choice of pen up lines such that pu is a minimum is discussed at some length later in this thesis.

A heuristic method has been developed to obtain the pen up segments and will now be explained.

5.10.1. Heuristic Method.

Consider a nxn cost matrix m such that the element $m\left[i,j\right]$ is some measure of the distance between the odd nodes i and j. Since interest lies only in the magnitudes of these distances m will be symmetric so that

$$m\left[i,j\right] = m\left[j,i\right] \qquad \text{\_\_\_\_\_\_\_\_\_\_}(10)$$

The main diagonal represents the distance of a node from itself so that

$$m\left[i,i\right] = 0 \quad i = 1,2 \ldots\ldots n \qquad \text{\_\_\_\_\_\_\_\_}(11)$$

Now consider setting up a n x 4 matrix b the elements of which will be as follows

b [i,1] = nearest point to i (excluding i)
b [i,2] = next nearest point to i (excluding i)
b [i,3] = distance between b [i,1] and i
b [i,4] = (distance between b[i,2] and i) - b[i,3]

High values of b [i,4] suggest that the nearest point to i is that much better a choice than the next nearest. Small values of b [i,4] similarly suggest that there is not much difference in choice between the nearest and next nearest points to i.

Suppose the matrix b is set up for every point i.

Now scan through b [i,4] i = 1,2......n and suppose point j has the highest value. This suggests that it would be advisable to link point j to b [j,1] since the next best choice is not very good.

Consider j to be linked to b [j,1] .

These points are therefore linked together and so cannot be the terminal points of any future pen up segments to be chosen.

The problem thus reduces to choosing $\frac{n}{2}$ -1 segments from the n-2 remaining .

It is also necessary to check through the columns b [i,1] and b [i,2] in case j and b [j,1] occur for these are no longer valid as the nearest points and next nearest points.

The method is repeated choosing elements j and b [j,1] corresponding to the highest value of b [j,4]. The obvious disadvantage of the method is that after each choice, columns $b[i,1]$ and $b[i,2]$ need to be checked to see if either of the elements just chosen occurs. If they do then it is necessary to adjust the associated row of b.

Note also that only n/2-1 choices need be made since no choice is required when only two points remain.

5.11. Sorting the Segments to form an Eulerian Cycle.

At this stage of the organisation the pen up segments which have been added to G are such that the new graph formed, G', contains an Eulerian cycle. It is necessary to sort the segments G' such that they form a continuous sequence which will be a directed Eulerian cycle. This will enable the pen of the display to follow every edge of G' once and once only so that the cycle can be drawn in one continuous movement of the pen.

Berge[24] describes an algorithm for tracing an Eulerian cycle without ever having to correct the route taken. The method obeys two rules as follows :-

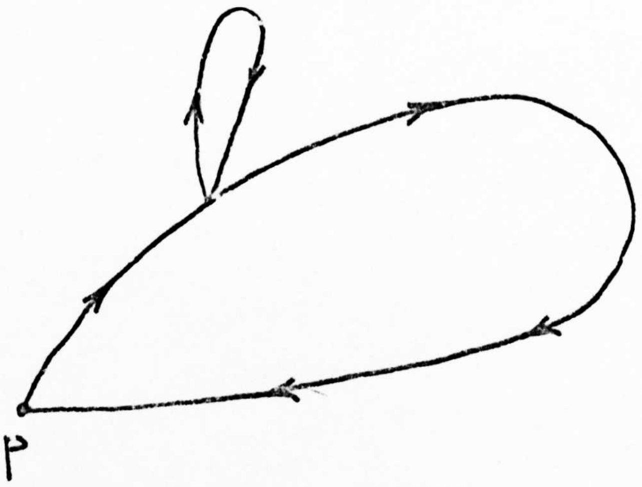
1. Start from any node p, each time an edge has been followed, erase it.
2. Never use a segment if, at that particular moment, the deletion of this segment would divide the graph G' into two connected components.

The second of these rules means that, before a segment is chosen a check must be made to ensure that the graph will not be divided into two connected components. This causes the algorithm to be inefficient for use on a computer.

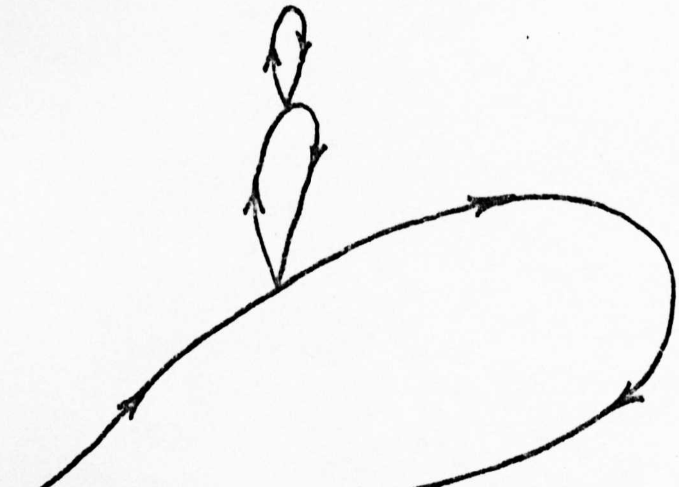
A method for forming a directed Eulerian cycle from a graph G' which has all nodes of even degree has been devised. The method follows directly from the proof given for Theorem 1.

Start from any node p and trace out a chain until rode p is reached. This will always be possible and the chain will correspond to an initial Eulerian cycle. Every time a segment is traversed it is deleted from the graph.

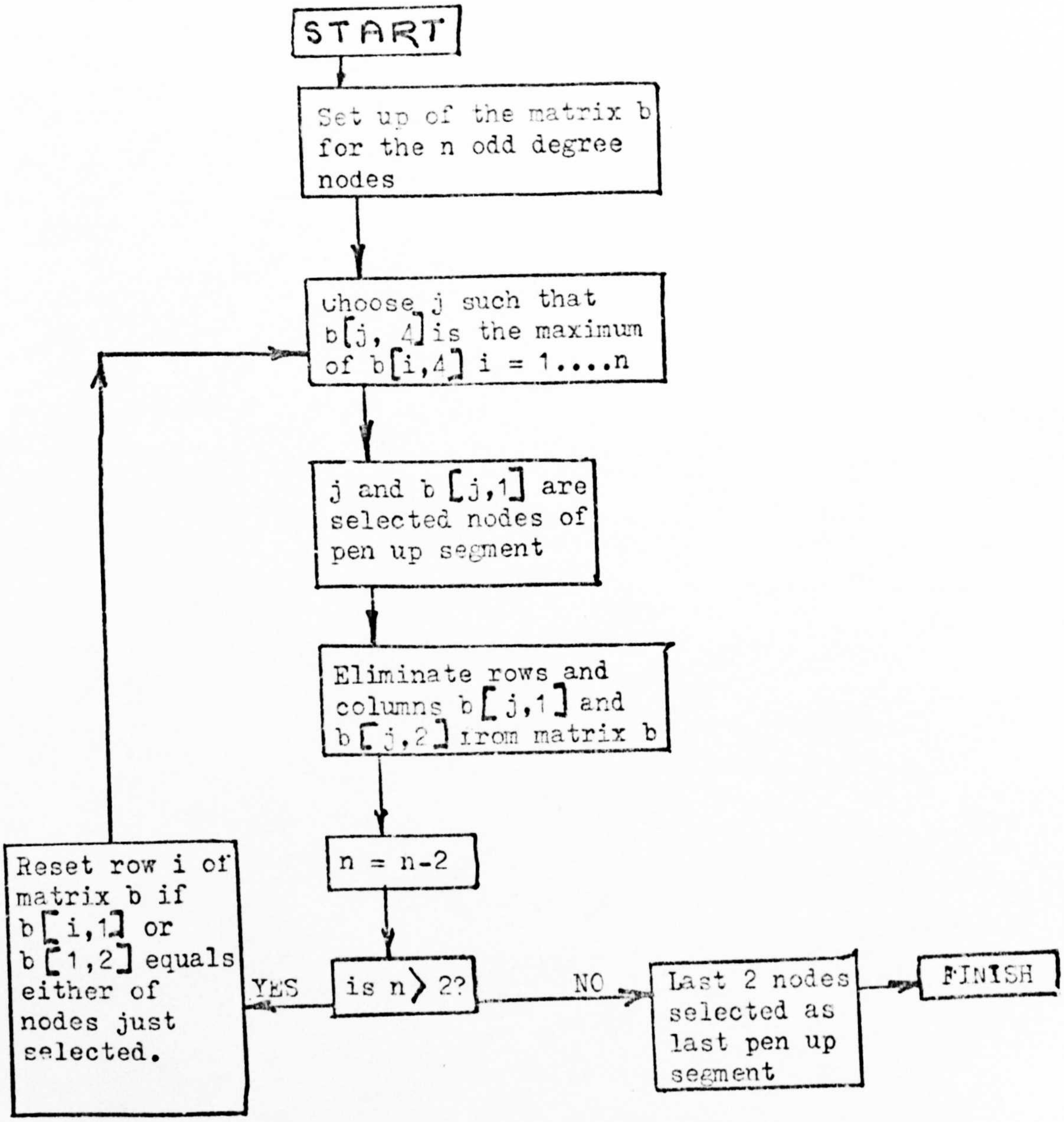
Now check every node of this initial cycle to determine whether any non deleted segments emanate from it. If a segment does emanate from a node then it will be possible to trace another Eulerian cycle starting from this node, which can be added to the original Eulerian cycle to form an extended cycle as illustrated below. The segments which are traversed are again deleted from G'.



The nodes of this second cycle can be checked to determine whether any non deleted segments branch from it. If so then the cycle can be extended again to include a third Eulerian cycle which can be formed from this initial node as shown below.



By this means it is possible to extend the initial Eulerian cycle until all the segments of G' have been deleted. The final cycle will be the directed Eulerian cycle contained in G'.


START
Set up of the matrix b for the n odd degree nodes
Choose j such that b[j, 4]is the maximum of b[i,4] i = 1....n
j and b [j,1] are selected nodes of pen up segment
Eliminate rows and columns b [j,1] and b [j,2] from matrix b
n = n-2
is n > 2?
YES
Reset row i of matrix b if b [i,1] or b [1,2] equals either of nodes just selected.
NO
Last 2 nodes selected as last pen up segment
FINISH

FLOW DIAGRAM FOR HEURISTIC METHOD OF CHOOSING THE PEN UP SEGMENTS

Diagram 5.3

CHAPTER 6

PEN UP PROBLEM

Theory Associated with the Pen Up Problem.

CHAPTER 6

PEN UP PROBLEM

6.1. Introduction.

It was explained in the previous chapter that the pen up problem consists of finding a set of segments in the given line drawing such that they form a perfect matching of the n odd degree nodes in the drawing.

If di is a measure of the length of the ith segment of this set, then the optimal pen up problem consists of finding a minimising choice of $\sum_{i=1}^{\frac{n}{2}}$ di. The number of feasible solutions to the pen up problem will equal the number of ways in which n/2 pairs of nodes can be chosen from the n odd nodes of the line drawing. The number of feasible solutions is thus $\dfrac{n!}{\left(\frac{n}{2}\right)!\; 2^{\frac{n}{2}}}$

The number of feasible solutions thus increases rapidly with n, there being 945 feasible solutions for an n value of 10. It is not practicable, even for small n values, to carry out complete enumeration. An implicit enumeration scheme has been developed in which it is only necessary to examine a small percentage of the total number of feasible solutions to be certain that an optimal solution to the pen up problem has been obtained. In addition, various enumeratiwe schemes have also been developed which enable good feasible solutions to be obtained rather more quickly than in the general implicit enumeration scheme.

In this chapter the theory associated with the pen up problem will be discussed and an associated problem, referred to as the image pen up problem, will be developed.

A criterion for optimality has been developed for the pen up problem and if conditions are satisfied for the criterion then the optimal solution has been obtained. In most cases, however, an optimal solution will not satisfy the criterion. An example of the pen up problem which satisfies the criterion for optimality and an example which does not, are presented in this chapter.

6.2. Number of Feasible Solutions Sn to Pen Up Problem.

The number of feasible solutions to the pen up problem for a line drawing containing n odd degree nodes is the number of ways in which $\frac{n}{2}$ pairs of nodes can be chosen from the n. If Sn is the number of feasible solutions for a given n then Sn is given by

$$Sn = \frac{n!}{\left(\frac{n}{2}\right)!\ 2^{\frac{n}{2}}} \quad \text{\_\_\_\_\_\_\_\_\_\_\_\_}(1)$$

so that $$Sn = \frac{n\ (n-1)\ (n-2)\ldots\ldots\left(\frac{n}{2}+1\right)}{2^{\frac{n}{2}}}$$

n	2	4	6	8	10
Sn	1	3	15	105	945
$\log_{10}$ Sn	0	0.477	1.176	2.02	2.98

Note that

$$Sn+2 = \frac{(n+2)\ (n+1)}{\left(\frac{n}{2}+1\right) .\ 2} \cdot Sn$$

so that

$$\underline{Sn+2 = (n+1)\ Sn}$$

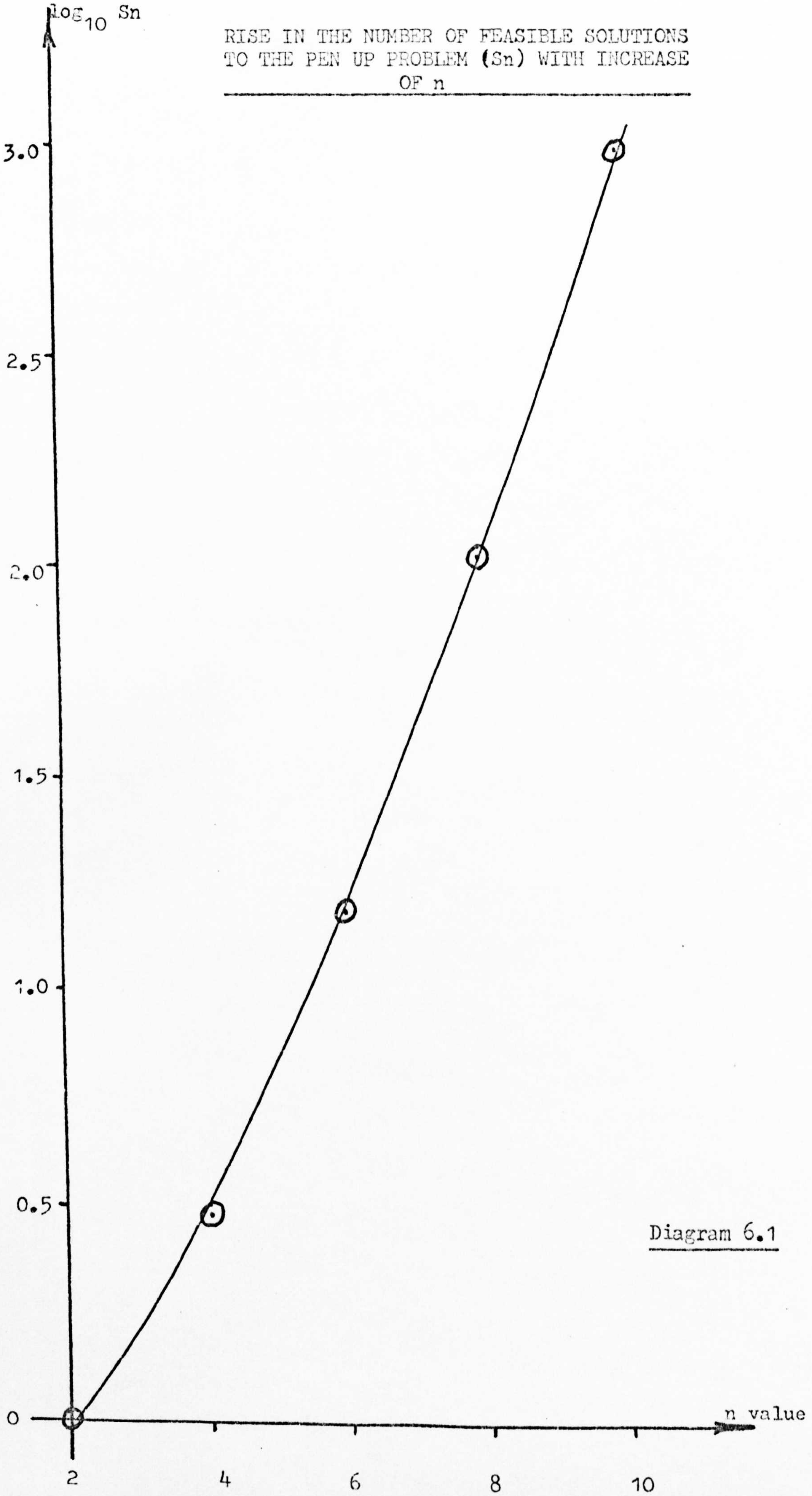

log₁₀ Sn
RISE IN THE NUMBER OF FEASIBLE SOLUTIONS TO THE PEN UP PROBLEM (Sn) WITH INCREASE OF n
3.0
2.5
2.0
1.5
1.0
0.5
0
n value
2
4
6
8
10

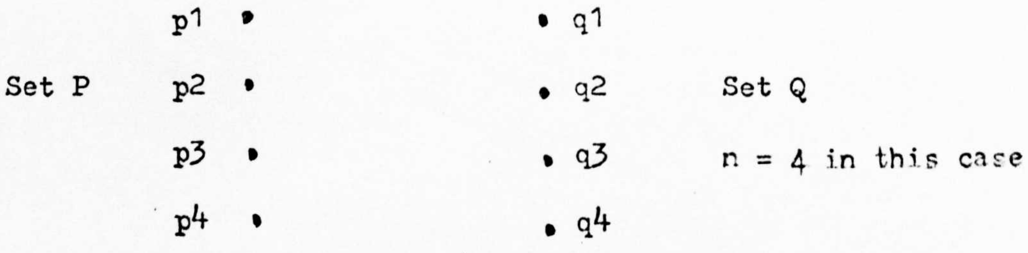
Diagram 6.1

6.3. Sub-Problems of the Pen up Problem.

Any feasible solution to the pen up problem consists of a set of segments which form a perfect matching on the n odd degree nodes. Consider a division of the set S of odd degree nodes into two disjoint sets P and Q such that

$$P \cup Q = S, \; P \cap Q = 0, |P| = |Q| = \frac{n}{2}$$

Corresponding to a given division of S there will exist a number of feasible solutions to the pen up problem. Each feasible solution will correspond to matching one of the elements of P with an element of Q. If $p \in P$ and $q \in Q$ are elements of P and Q respectively then each division will correspond to the situation shown below.



There are a number of feasible solutions to the pen up problem existing for each division of S. These solutions are also feasible solutions to the assignment problem of order $\frac{n}{2}$ with the two given sets P and Q.

Thus, corresponding to the pen up problem of order n there are a number, Gpq, of sub-problems, each of which is an assignment problem of order $\frac{n}{2}$. Gpq is equal to the number of ways in which the n elements of S can be divided into two sets P and Q. Gpq is thus given by

$$Gpq = \frac{n!}{\left(\frac{n}{2}\right)!\left(\frac{n}{2}\right)!^{2}} \qquad \text{\_\_\_\_\_\_\_\_(2)}$$

n	2	4	6	8	10	12
Gpq	1	3	10	35	126	462
$\log_{10}$Gpq	0	0.48	1	1.54	2.1	2.66

The number of feasible solutions Sn to the pen up problem is linked to the number Gpq of sub problems as follows

$$Gpq = \frac{Sn.2^{\left(\frac{n}{2}-1\right)}}{\frac{n}{2}} \qquad \text{\_\_\_\_\_\_\_\_(3)}$$

Note also that

$$Gpq_{n+2} = \frac{4\ (n+2)\ (n+1)}{(n+2)^{2}} \ \bullet\ Gpq_{n}$$

so that $$Gpq_{n+2} = \frac{4\ (n+1)}{(n+2)} \bullet Gpq_{n} \qquad \text{\_\_\_\_\_\_\_\_(4)}$$

Now each of the sub-problems will have an optimal solution associated with it, which will be the optimal solution of the $\frac{n}{2}$ assignment problem. It will be convenient to refer to these optimal solutions as local optimal solutions. The $\frac{n}{2} \times \frac{n}{2}$ cost matrix E corresponding to a sub-problem will have rows and columns which correspond to the elements of P and Q respectively.

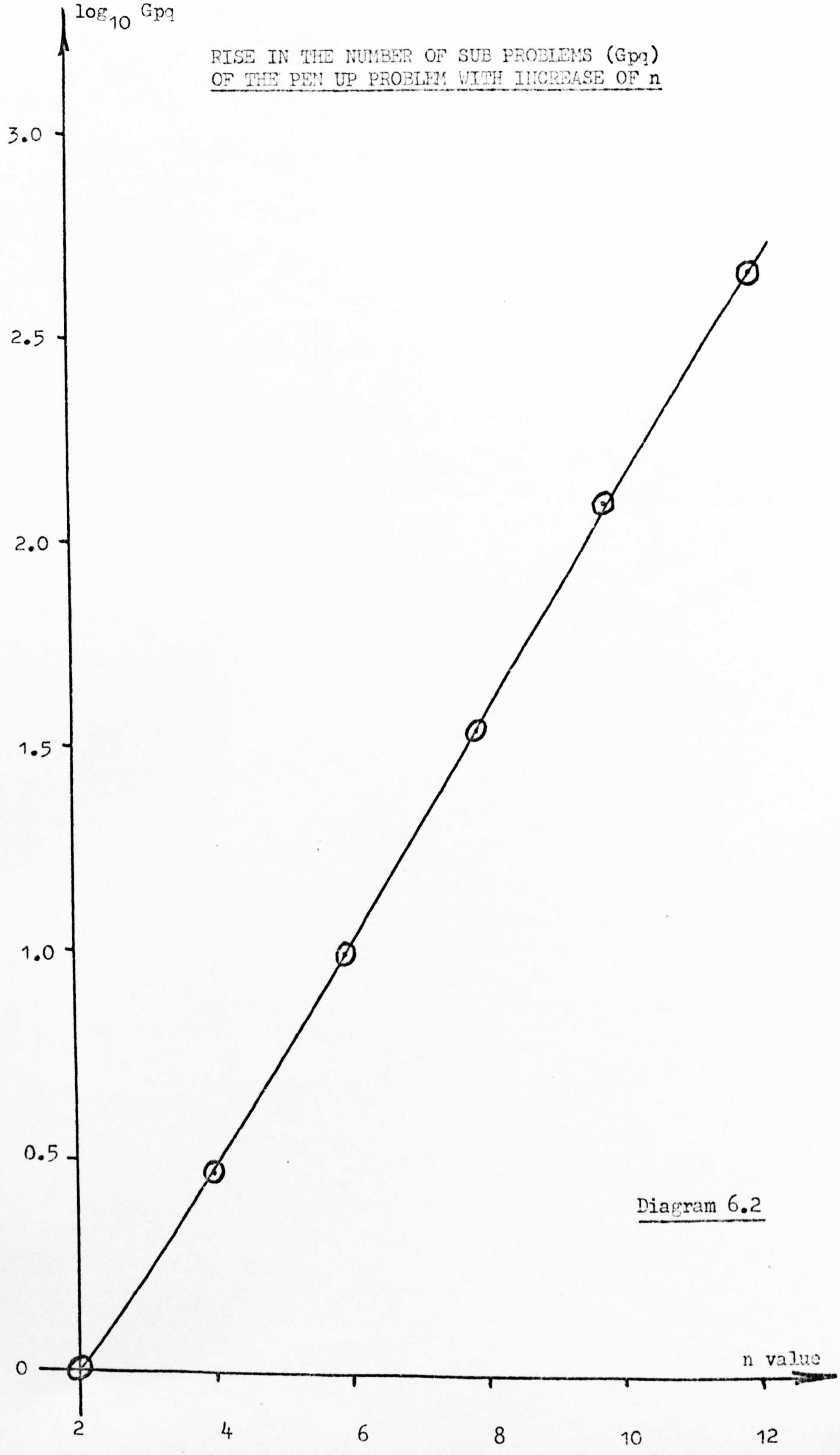
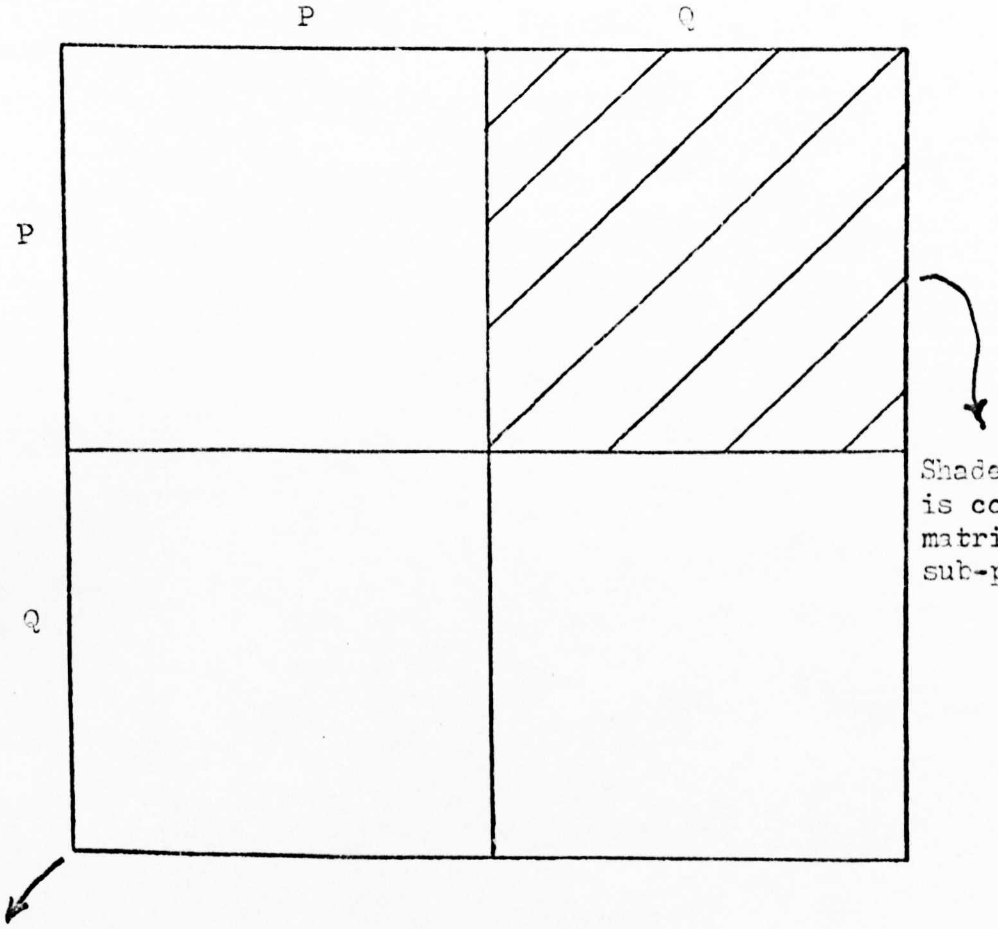

$\log_{10}$ Gpq
RISE IN THE NUMBER OF SUB PROBLEMS (Gpq) OF THE PEN UP PROBLEM WITH INCREASE OF n
3.0
2.5
2.0
1.5
1.0
0.5
0
n value
2
4
6
8
10
12

Diagram 6.2

If C is the n x n cost matrix of the pen up problem such that Cij is some measure of the distance between the odd nodes i and j then it will be possible, by permuting the rows and columns of C simultaneously, to arrange C such that E corresponds to one quarter of it.



Shaded area is cost matrix E of sub-problem

original cost matrix C of pen up problem

Every sub-problem thus has a cost matrix E which can be made to correspond to one quarter of the original cost matrix C.

6.4. Lower Bound to Given Sub-Problems.

Sinee each of the sub-problems is an $\frac{n}{2} \times \frac{n}{2}$ assignment problem it is possible to obtain a lower bound to each sub-problem by obtaining a feasible solution to the dual assignment problem.

Suppose implicit costs a_i $(i = 1,2....\frac{n}{2})$ and bj $(j = 1,2....\frac{n}{2})$ to be associated with the rows and colunms of E respectively. The dual assignment problem is

$$\text{maximise} \left(\sum_{i=1}^{\frac{n}{2}} ai + \sum_{j=1}^{\frac{n}{2}} bj \right) \qquad \text{\_\_\_\_\_\_\_\_\_\_(5)}$$

such that

$$Eij - ai - bj \geqslant 0 \qquad \begin{matrix} i = 1,....\frac{n}{2} \\ j = 1,....\frac{n}{2} \end{matrix}$$

Consider the following method of obtaining a feasible solution to the dual assignment problem with cost matrix E.

The initial values of the implicit costs a and b are zero.

Assign to the costs ai $(i = 1...\frac{n}{2})$ the value of the smallest element occuring in each of the $\frac{n}{2}$ rows. In the reduced cost matrix $\delta E = E - ai - bj$, a zero will therefore have been introduced into every tow. Now scan each of the columns of E which do not contain a zero and assign to the corresponding implicit cost bj the value of the smallest element occurring. Each of the rows and columns will thus contain a zero and the cost $\sum_{i=1}^{n/2} ai + \sum_{j=1}^{n/2} bj$

will form a lower bound to the assignment problem.

6.4.1. Example.

initial cost matrix E

$$\begin{array}{cccc|c}
2 & 2 & 1 & 3 & a1 \\
4 & 3 & 6 & 5 & a2 \\
3 & 2 & 1 & 8 & a3 \\
1 & 4 & 3 & 5 & a4 \\
\hline
b1 & b2 & b3 & b4 &
\end{array}$$

The set of implicit costs a will be assigned the values a1 = 1, a2 = 3, a3 = 1, a4 = 1 and the reduced cost matrix δE so far will be as follows.

$$\begin{array}{cccc|c}
 & & & & a \\
1 & 1 & 0 & 2 & 1 \\
1 & 0 & 3 & 2 & 3 \\
2 & 1 & 0 & 7 & 1 \\
0 & 3 & 2 & 4 & 1 \\
\hline
b1 & b2 & b3 & b4 &
\end{array}$$

Each element of the above matrix corresponds to Eij-ai-bj with all costs b being zero at present.

The only column which does not contain a zero is column 4 so that the implicit cost corresponding to this column, b4, is given the value of the smallest element of the column, which is 2. Thus b4 = 2. The implicit costs are thus

a1 = 1, a2 = 3, a3 = 1, a4 = 1

b1 = 0, b2 = 0, b3 = 0, b4 = 2

and the final reduced matrix is as follows

$$\begin{array}{c|cccc|c} & & & & & a \\ \hline & 1 & 1 & 0 & 0 & 1 \\ & 1 & 0 & 3 & 0 & 3 \\ & 2 & 1 & 0 & 5 & 1 \\ & 0 & 3 & 2 & 2 & 1 \\ \hline b & 0 & 0 & 0 & 2 & \end{array}$$

The solution to the dual assignment problem has a cost Cd given by

$$Cd = \sum_{i=1}^{\frac{n}{2}} ai + \sum_{j=1}^{\frac{n}{2}} bj$$

$$Cd = 8$$

It is therefore possible for each of the Gpq sub-problems to obtain a lower bound to each of the local optimal solutions.

6.5. Optimal Solution to Pen up Problem.

Consider an n x n matrix x to be a feasible solution to the pen up problem such that for any given sub-problem

$$\underset{\substack{i\in Q\\ j\in P}}{xij} = \underset{\substack{i\in p\\ j\in Q}}{xij} = 1 \text{ and } xij = 0 \text{ otherwise } \underline{\hspace{4em}}(6)$$

x is symmetric

If Cij$\in$C is the original cost matrix, then for a given sub-problem the local optimal solution is such that

$$\sum_{\substack{i\in p\\ j\in Q}} Cij.xij \text{ is minimised}$$

The optimal solution to the pen up problem will be such that the total cost $\sum_{\substack{i \in P \\ j \in Q}} C_{ij}.\, x_{ij}$ is minimised over all P and Q.

6.6 Lower Bound to Pen Up Problem.

Consider a new set of implicit costs U_k $(k=1,2....n)$ which are to be associated with the original cost matrix C of the pen up problem such that for each element

$$C_{ij} - U_i - U_j \geq 0 \quad \text{\_\_\_\_\_\_\_\_\_\_\_\_}(7)$$

It has been shown in an earlier section that the optimal pen up problem is to minimise the total cost σ given by

$$\sigma = \sum_{\substack{i \in P \\ j \in Q}} C_{ij}.\, x_{ij}$$

taken over all the sets P and Q.

Now suppose a reduced matrix $\delta C_{ij} \in \delta C$ has elements defined as follows

$$\delta C_{ij} = C_{ij} - U_i - U_j$$

then

$$\sigma = \sum_{\substack{i \in P \\ j \in Q}} (\delta C_{ij} + U_i + U_j).\, x_{ij}$$

$$= \sum_{\substack{i \in P \\ j \in Q}} \delta C_{ij}.\, x_{ij} + \sum_{\substack{i \in P \\ j \in Q}} (U_i + U_j).\, x_{ij}$$

$$= \sum_{\substack{i \in P \\ j \in Q}} \delta C_{ij}.\, x_{ij} + \sum_{i \in P} U_i + \sum_{j \in Q} U_j$$

now since $x_{ij} \geq 0$ and $\delta C_{ij} \geq 0$

then

$$\sigma \geq \sum_{i \in P} U_i + \sum_{j \in Q} U_j$$

$$\sigma \geq \sum_{k \in s} U_k \quad \text{\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_}(8)$$

Where S is the original set of n elements.

From (8) it can be seen that $\sum_{k \in S}$ Uk will give a lower bound to the pen up problem.

6.7 Criterion for Optimality.

It has been shown that an optimal solution has a total cost σ given by

$$\sigma = \sum_{\substack{i \in P \\ j \in Q}} \delta Cij.\, xij + \sum_{i \in P} Ui + \sum_{j \in Q} Uj$$

taken over all P and Q.

Now suppose x is given a feasible solution to the pen up problem consistant with equation (6) being satisfied.

Further suppose that the implicit costs U are chosen so that

$$\delta Cij = 0 \quad \underline{\text{if}} \quad xij > 0 \qquad \text{\_\_\_\_\_\_\_\_\_\_(9)}$$

$$\text{and} \qquad \delta Cij > 0 \quad \underline{\text{if}} \quad xij = 0$$

(note from the above two conditions $\delta Cij.\ xij = 0$)

If it is possible to choose the elements of U such that the above two conditions are true then x must correspond to an optimal solution since any alteration in the elements of δC and x will only result in an increase of $\delta Cij.\ xij$ from its zero value. It is not always possible to choose the elements of U such that the above conditions are true but if it is possible then x must correspond to an optimal solution.

The problem of maximising $\sum_{k \in S}$ Uk consistant with there being positive elements in the reduced cost matrix δC

will be referred to as the image pen up problem and in the two examples which follow feasible solutions to the image problem will be obtained prior to establishing an optimal solution to the pen up problem for the given two cost matrices.

6.8. Examples.

The criterion for optimality given earlier only holds for a few optimal solutions and an example for which it holds and an example for which it does not hold will be given. The examples both correspond to an n value of 6.

6.8.1. Criterion True. The original cost matrix C is given by

$$C = \begin{array}{|c|c|c|c|c|c|c} \multicolumn{6}{c}{} & Uk \\ \cline{1-6} \diagup & 6 & 3 & 5 & 6 & 7 & 0 \\ \cline{1-6} 6 & \diagup & 5 & 2 & 8 & 9 & 0 \\ \cline{1-6} 3 & 5 & \diagup & 5 & 6 & 7 & 0 \\ \cline{1-6} 5 & 2 & 5 & \diagup & 5 & 6 & 0 \\ \cline{1-6} 6 & 8 & 6 & 5 & \diagup & 2 & 0 \\ \cline{1-6} 7 & 9 & 7 & 6 & 2 & \diagup & 0 \\ \cline{1-6} \end{array}$$

Now consider increasing the implicit costs Uk (k=1,2....n) from their initial zero values to the values of the element in the corresponding rows which have the smallest value of δ Cij given by

$$\delta Cij = Cij - Ui - Uj$$

There are many orders in which the implicit costs can be given a value,but for simplicity suppose U_1 is given the value of the smallest element in row 1.

U_1 will thus be given a value of 3 and the reduced cost matrix will be

$$\delta C = \begin{array}{|c|c|c|c|c|c|c}
\multicolumn{6}{c}{} & U_k \\
\hline
\diagup & 3 & 0 & 2 & 3 & 4 & 3 \\
\hline
3 & \diagup & 5 & 2 & 8 & 9 & 0 \\
\hline
0 & 5 & \diagup & 5 & 6 & 7 & 0 \\
\hline
2 & 2 & 5 & \diagup & 5 & 6 & 0 \\
\hline
3 & 8 & 6 & 5 & \diagup & 2 & 0 \\
\hline
4 & 9 & 7 & 6 & 2 & \diagup & 0 \\
\hline
\end{array}$$

The elements of column 1 and row 1 have thus been reduced by 3.

Now suppose U_2 ic to be assigned a value, U_2 will be given the value cf the smallest element in row 2 which is 2.

The reduced cost matrix at this stage will thus be

$$\delta C = \begin{array}{|c|c|c|c|c|c|c}
\multicolumn{6}{c}{} & U_k \\
\hline
\diagup & 1 & 0 & 2 & 3 & 4 & 3 \\
\hline
1 & \diagup & 3 & 0 & 6 & 7 & 2 \\
\hline
0 & 3 & \diagup & 5 & 6 & 7 & 0 \\
\hline
2 & 0 & 5 & \diagup & 5 & 6 & 0 \\
\hline
3 & 6 & 6 & 5 & \diagup & 2 & 0 \\
\hline
4 & 7 & 7 & 6 & 2 & \diagup & 0 \\
\hline
\end{array}$$

Row 3 has already a zero present so that U_3 cannot be raised above zero without a negative element being introduced. Similarly with row 4 so that both U_3 and U_4 will retain their zero values.

The smallest element in row 5 is 2 so that U_5 will be assigned the value of 2 and the reduced cost matrix will now be as follows

$$\delta C = \begin{array}{|c|c|c|c|c|c|c} \multicolumn{6}{c}{} & Uk \\ \hline \diagup & 1 & 0 & 2 & 1 & 4 & 3 \\ \hline 1 & \diagup & 3 & 0 & 4 & 7 & 2 \\ \hline 0 & 3 & \diagup & 5 & 4 & 7 & 0 \\ \hline 2 & 0 & 5 & \diagup & 3 & 6 & 0 \\ \hline 1 & 4 & 4 & 3 & \diagup & 0 & 2 \\ \hline 4 & 7 & 7 & 6 & 0 & \diagup & 0 \\ \hline \end{array}$$

Row 6 already has a zero present so that U_6 must remain at zero consistent with there being no negative elements in the reduced cost matrix.

In this case the lower bound cost $\sum_{k \in s} Uk$ is given by 7 and the optimal solution x is given by

$$x = \begin{array}{|c|c|c|c|c|c|} \hline \diagup & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & \diagup & 0 & 1 & 0 & 0 \\ \hline 1 & 0 & \diagup & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & \diagup & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & \diagup & 1 \\ \hline 0 & 0 & 0 & 0 & 1 & \diagup \\ \hline \end{array}$$

Since both δC and x are symmetric it will only be necessary to inspect elements of x and δC corresponding to

$$xij \; \ldots\ldots \; i > j$$
$$\delta Cij \; \ldots\ldots \; i > j$$

This will correspond to the lower half of each matrix.

Now for xij = 1 there correspond the values

$$x_{31} = 1 \qquad x_{42} = 1 \qquad x_{65} = 1$$

and the corresponding elements in the reduced cost matrix δC are all zero.

$$\delta c_{31} = 0 \qquad \delta c_{42} = 0 \qquad \delta c_{65} = 0$$

The remaining values of x are zero and the remaining values of δC are non zero so that in this case the criterion for optimality holds and the optimal solution is given by the matrix x.

The optimal cost is given by $\sum$ Cij. xij = $C_{31} + C_{42} + C_{65} = 3 + 2 + 2 = 7$ which is equal to the lower bound $\sum\limits_{k \in s}$ Uk evaluated previously.

Note that the optimal cost evaluated from the reduced cost matrix has zero value.

6.8.2. <u>Criterion False.</u> Consider the original cost matrix as follows

$$C = \begin{array}{|c|c|c|c|c|c|}
\hline
\diagup & 6 & 4 & 3 & 7 & 2 \\
\hline
6 & \diagup & 2 & 3 & 4 & 1 \\
\hline
4 & 2 & \diagup & 7 & 6 & 5 \\
\hline
3 & 3 & 7 & \diagup & 4 & 3 \\
\hline
7 & 4 & 6 & 4 & \diagup & 2 \\
\hline
2 & 1 & 5 & 3 & 2 & \diagup \\
\hline
\end{array}$$

By a similar method as explained in detail for the previous example, the reduced matrix δC with corresponding implicit costs U will be as follows

$$\delta C= \begin{array}{|c|c|c|c|c|c|c}
 \multicolumn{6}{c}{} & U_k \\
\hline
\diagup & 3 & 1 & 0 & 3 & 0 & 2 \\
\hline
3 & \diagup & 0 & 1 & 1 & 0 & 1 \\
\hline
1 & 0 & \diagup & 5 & 3 & 4 & 1 \\
\hline
0 & 1 & 5 & \diagup & 1 & 2 & 1 \\
\hline
3 & 1 & 3 & 1 & \diagup & 0 & 2 \\
\hline
0 & 0 & 4 & 2 & 0 & \diagup & 0 \\
\hline
\end{array}$$

a lower bound cost is thus given by 7.

Now consider an optimal solution x given by

$$x= \begin{array}{|c|c|c|c|c|c|}
\hline
\diagup & 0 & 0 & 1 & 0 & 0 \\
\hline
0 & \diagup & 1 & 0 & 0 & 0 \\
\hline
0 & 1 & \diagup & 0 & 0 & 0 \\
\hline
1 & 0 & 0 & \diagup & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & \diagup & 1 \\
\hline
0 & 0 & 0 & 0 & 1 & \diagup \\
\hline
\end{array}$$

The elements of x which have unit values are

$$x_{32} = 1 \quad x_{41} = 1 \quad x_{56} = 1$$

and the corresponding elements of the reduced matrix δC have zero values.

However in this case there are several elements of δC, apart from the three listed above, which have zero values and so the criterion for optimality does not hold for this particular example.

Note however that the optimal cost

$$C_{32} + C_{41} + C_{56} = 7$$

is still equal to the value obtained for the lower bound.

6.9 Summary.

The pen up problem may be considered to be a number of sub-problems which are simple assignment problems. To each of these sub-problems it is possible to obtain a lower bound to the local optimal solution by obtaining a feasible solution to the dual assignment problem. The cost matrix E of each of these sub-problems may be considered to correspond to one quarter of the original cost matrix C. The implicit costs associated with the dual assignment problem were such that the reduced cost matrix δE given by

$$\delta E_{ij} = E_{ij} - a_i - b_j \quad \begin{matrix} i\varepsilon P \\ j\varepsilon Q \end{matrix}$$

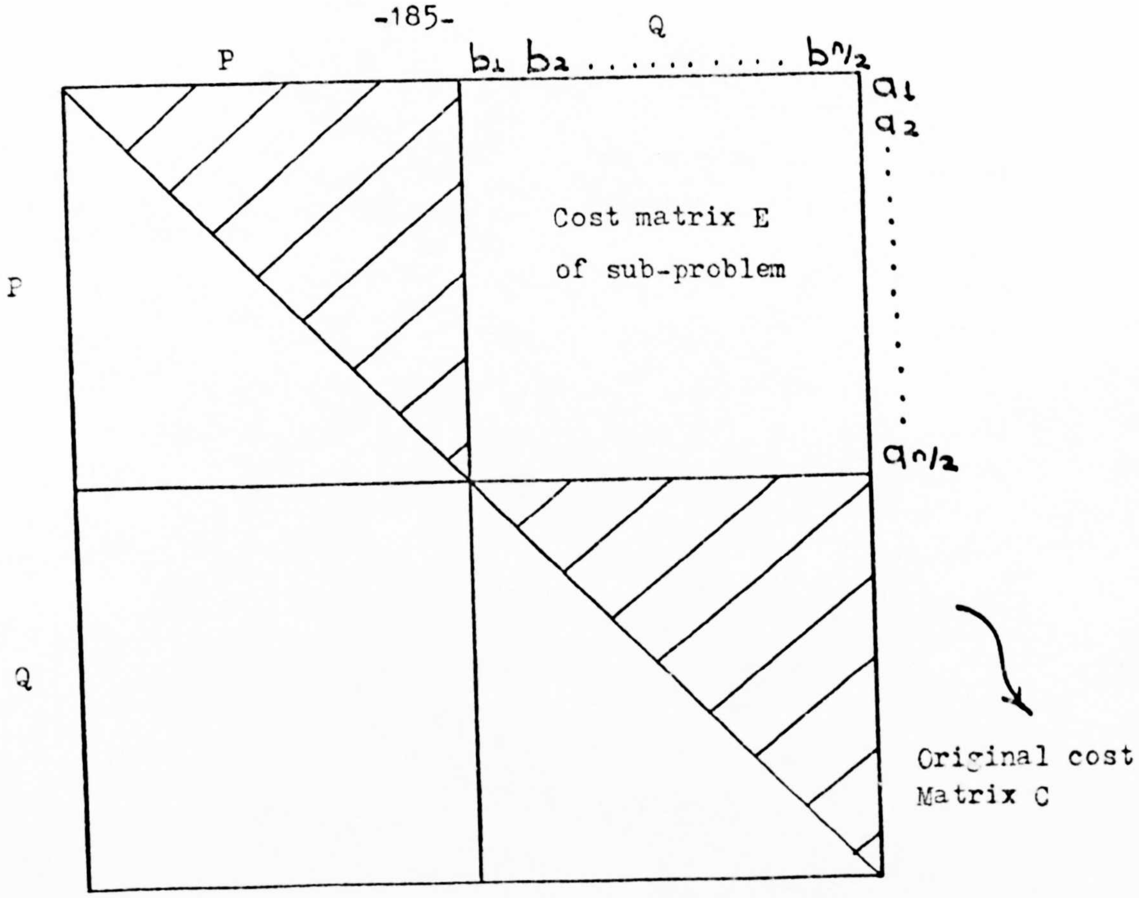
had non negative elements.

By introducing a new set of implicit costs Ui and by suitably reducing the original cost matrix C to a cost matrix δC given by

$$\delta C_{ij} = C_{ij} - U_i - U_j$$

it has been possible to form a new problem, called the image pen up problem, a feasible solution to which will give a lower bound to the pen up problem. A feasible solution to the image pen up problem may be considered as a method of obtaining a lower bound to the Gpq lower bounds of the sub-problems.

Consider a given sub-problem with sets P and Q and suppose the implicit costs a and b have been introduced,



Note that a lower bound $\sum_{i=1}^{n/2} ai+\sum_{j=1}^{n/2} bj$ to any of the sub-problems may not give a lower bound to the pen up problem since elements of the original cost matrix C, shown shaded, when reduced, may be negative. The constraints on the elements of U are such that the elements of the reduced cost matrix δC are non negative. It is these extra constraints on U which allow $\sum_{K\in S} U_K$ to give a lower bound to all the Gpq sub-problems, and therefore to the pen up problem..

6.10. The Image Pen Up Problem

The image pen up problem is as follows :

$$\text{maximise} \sum_{k \in s} Uk$$

such that the reduced matrix δC given by

$$\delta Cij = Cij - Ui - Uj$$

has elements which are non negative so that $\delta Cij \geqslant 0$. A feasible solution to the image problem will give a lower bound to the pen up problem. Methods which obtain a good feasible solution to the image problem have been developed and will be described in detail. Initially all elements of U will have zero value.

Consider the cost matrix C and suppose the smallest element fi which occurs in each row i is found, together with the number of times gi it occurs. It is required to maximise $\sum_{k \in s}$ Uk and two similar heuristic methods can be used to obtain a good feasible solution.

6.10.1. Method 1. Choose that row j for which gi (i = 1,....n) has the least value and assign to Uj the value of the smallest element in row j. Hence Uj = fj

Since C is symmetric there will be gj rows of the reduced matrix δC which will have a zero introduced.

For every row i of δC without a zero again find the smallest element fi, and the number of times it occurs gi and choose that row j which has the smallest value of gi. The corresponding element of

U, Uj, is assigned the value fj and the method is repeated until all rows of the reduced matrix δC have zeros present.

If there are cases of equal gi for two or more rows then the row with the greatest value of fi is chosen.

The method tends to minimise the number of zeros introduced at each step so that the number of elements of U which are non-zero is maximised and the cost $\sum_{k \in s}$ Uk tends towards a high value.

6.10.2. Example

$$\delta C = C = \begin{array}{|c|c|c|c|c|c|ccc} \hline & & & & & & U & f & g \\ \hline \diagup & 5 & 4 & 3 & 4 & 3 & 0 & 3 & 2 \\ \hline 5 & \diagup & 2 & 1 & 2 & 4 & 0 & 1 & 1 \\ \hline 4 & 2 & \diagup & 5 & 4 & 3 & 0 & 2 & 1 \\ \hline 3 & 1 & 5 & \diagup & 2 & 4 & 0 & 1 & 1 \\ \hline 4 & 3 & 4 & 2 & \diagup & 1 & 0 & 1 & 1 \\ \hline 3 & 4 & 3 & 4 & 1 & \diagup & 0 & 1 & 1 \\ \hline \end{array}$$

row 3 is chosen and $U_3 = 2$

$$\delta C = \begin{array}{|c|c|c|c|c|c|ccc} \hline & & & & & & U & f & g \\ \hline \diagup & 5 & 2 & 3 & 4 & 3 & 0 & 2 & 1 \\ \hline 5 & \diagup & 0 & 1 & 3 & 4 & 0 & & \\ \hline 2 & 0 & \diagup & 3 & 2 & 1 & 2 & & \\ \hline 3 & 1 & 3 & \diagup & 2 & 4 & 0 & 1 & 1 \\ \hline 4 & 3 & 2 & 2 & \diagup & 1 & 0 & 1 & 1 \\ \hline 3 & 4 & 1 & 4 & 1 & \diagup & 0 & 1 & 1 \\ \hline \end{array}$$

In this case rows 4, 5 and 6 have equal claim and in these cases the row with the lowest number is chosen.

So $U_4 = 1$

$$\delta C = \begin{array}{cccccc|ccc} & & & & & & U & f & g \\ \diagup & 5 & 2 & 2 & 4 & 3 & 0 & 2 & 2 \\ 5 & \diagup & 0 & 0 & 3 & 4 & 0 & & \\ 2 & 0 & \diagup & 2 & 2 & 1 & 2 & & \\ 2 & 0 & 2 & \diagup & 1 & 3 & 1 & & \\ 4 & 3 & 2 & 1 & \diagup & 1 & 0 & 1 & 2 \\ 3 & 4 & 1 & 3 & 1 & \diagup & 0 & & \end{array}$$

$U_1 = 2$

$$\delta C = \begin{array}{cccccc|ccc} & & & & & & U & f & g \\ \diagup & 3 & 0 & 0 & 2 & 1 & 2 & & \\ 3 & \diagup & 0 & 0 & 3 & 4 & 0 & & \\ 0 & 0 & \diagup & 2 & 2 & 1 & 2 & & \\ 0 & 0 & 2 & \diagup & 1 & 3 & 1 & & \\ 2 & 3 & 2 & 1 & \diagup & 1 & 0 & & \\ 1 & 4 & 1 & 3 & 1 & \diagup & 0 & & \end{array}$$

At this stage only row 5 has not a zero present so

$U_5 = 1$

$$\delta C = \begin{array}{cccccc|c} & & & & & & U \\ \diagup & 3 & 0 & 0 & 1 & 1 & 2 \\ 3 & \diagup & 0 & 0 & 2 & 4 & 0 \\ 0 & 0 & \diagup & 2 & 1 & 1 & 2 \\ 0 & 0 & 2 & \diagup & 0 & 3 & 1 \\ 1 & 2 & 1 & 0 & \diagup & 0 & 1 \\ 1 & 4 & 1 & 3 & 0 & \diagup & 0 \end{array}$$

The cost of the feasible solution is thus 6.

6.10.3. Method 2. In this method the row with the largest value of fi occurring is chosen and the corresponding element of U assigned this value. The same scheme as in Method 1 is followed until all rows of the reduced matrix δC have a zero present.

Note that Method 2 will give a particularly poor cost if the row chosen has the smallest element occurring a large number of times, for then this number of zeros will be introduced into the reduced matrix.

Method 1 is not particularly good if the values of fi for the rows chosen are small.

Consider as an example of Method 2 the initial cost matrix used for Method 1.

6.10.4. Example.

$$\delta C = C = \begin{array}{|c|c|c|c|c|c|cc}
\multicolumn{6}{c}{} & U & f \\ \hline
\diagup & 5 & 4 & 3 & 4 & 3 & 0 & 3 \\ \hline
5 & \diagup & 2 & 1 & 3 & 4 & 0 & 1 \\ \hline
4 & 2 & \diagup & 5 & 4 & 3 & 0 & 2 \\ \hline
3 & 1 & 5 & \diagup & 2 & 4 & 0 & 1 \\ \hline
4 & 3 & 4 & 2 & \diagup & 1 & 0 & 1 \\ \hline
3 & 4 & 3 & 4 & 1 & \diagup & 0 & 1 \\ \hline
\end{array}$$

Row 4 is chosen as opposed to row 3 for Method 1.

$U_1 = 3$

$$\delta C = \begin{array}{|c|c|c|c|c|c|cc} \multicolumn{6}{c}{} & U & f \\ \hline \diagup & 2 & 1 & 0 & 1 & 0 & 3 & \\ \hline 2 & \diagup & 2 & 1 & 3 & 4 & 0 & 1 \\ \hline 1 & 2 & \diagup & 3 & 4 & 3 & 0 & 1 \\ \hline 0 & 1 & 5 & \diagup & 2 & 4 & 0 & \\ \hline 1 & 3 & 4 & 2 & \diagup & 1 & 0 & 1 \\ \hline 0 & 4 & 3 & 4 & 1 & \diagup & 0 & \\ \hline \end{array}$$

Row 2 is chosen so that $U_2 = 1$

$$\delta C = \begin{array}{|c|c|c|c|c|c|cc} \multicolumn{6}{c}{} & U & f \\ \hline \diagup & 1 & 1 & 0 & 1 & 0 & 3 & \\ \hline 1 & \diagup & 1 & 0 & 2 & 3 & 2 & \\ \hline 1 & 1 & \diagup & 5 & 4 & 3 & 0 & 1 \\ \hline 0 & 0 & 5 & \diagup & 2 & 4 & 0 & \\ \hline 1 & 2 & 4 & 2 & \diagup & 1 & 0 & 1 \\ \hline 0 & 3 & 3 & 4 & 1 & \diagup & 0 & \\ \hline \end{array}$$

Row 3 is chosen $U_3 = 1$

$$\delta C = \begin{array}{|c|c|c|c|c|c|cc} \multicolumn{6}{c}{} & U & f \\ \hline \diagup & 1 & 0 & 0 & 1 & 0 & 3 & \\ \hline 1 & \diagup & 0 & 0 & 2 & 3 & 2 & \\ \hline 0 & 0 & \diagup & 4 & 3 & 2 & 1 & \\ \hline 0 & 0 & 4 & \diagup & 2 & 4 & 0 & \\ \hline 1 & 2 & 3 & 2 & \diagup & 1 & 0 & 1 \\ \hline 0 & 3 & 2 & 4 & 1 & \diagup & 0 & \\ \hline \end{array}$$

Row 5 is the only row in the reduced matrix which does not have a zero present so

$U_5 = 1$

$$\delta C = \begin{array}{|c|c|c|c|c|c|c}
 & & & & & & U \\
\hline
\diagup & 1 & 0 & 0 & 0 & 0 & 3 \\
\hline
1 & \diagup & 0 & 0 & 1 & 3 & 2 \\
\hline
0 & 0 & \diagup & 4 & 2 & 2 & 1 \\
\hline
0 & 0 & 4 & \diagup & 1 & 4 & 0 \\
\hline
0 & 1 & 2 & 1 & \diagup & 0 & 1 \\
\hline
0 & 3 & 2 & 4 & 0 & \diagup & 0 \\
\hline
\end{array}$$

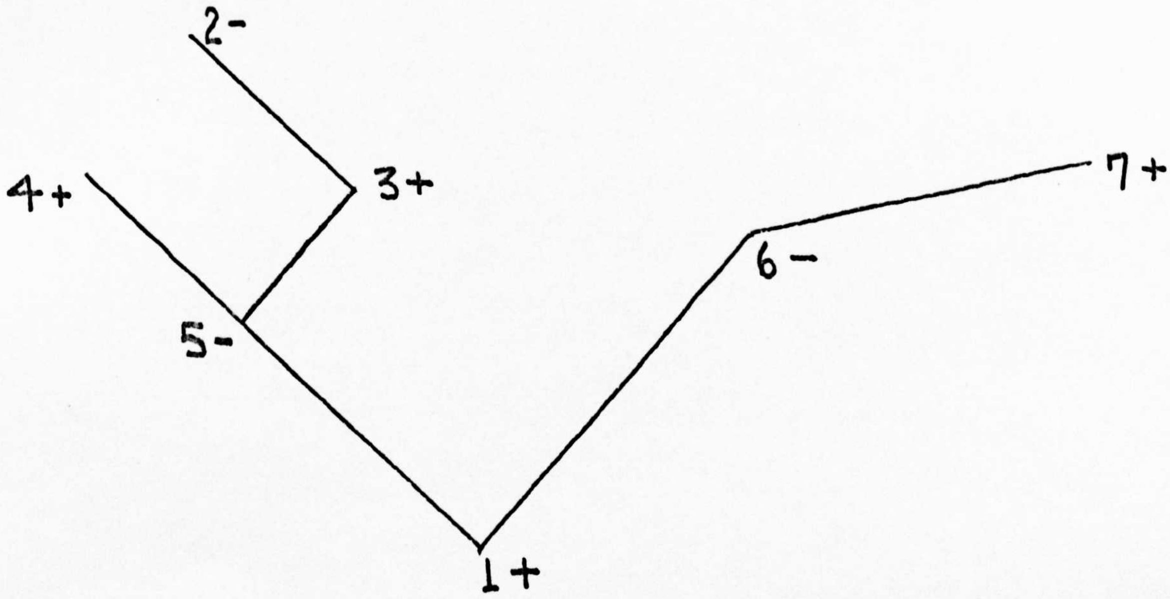
In this case the cost is 7.

Note that for Method 2 a total of 14 zeros have been introduced into the reduced cost matrix whereas only 12 were introduced for Method 1.

6.11. Improving the Initial Feasible Cost to the Image Problem.

It is possible, in some cases, to improve the cost of the feasible solution obtained by one of the heuristic methods described earlier. Suppose a graph called the image graph is associated with the reduced matrix δC such that the image graph has n nodes and a segment links node i to node j if $\delta Cij = 0$. Since it is assumed that the reduced matrix δC has been obtained by one of the heuristic methods described earlier there will be at least one segment associated with each of the nodes since the terminating condition for each of the heuristic methods was that a zero was present in each of the rows of the reduced matrix δC.

Suppose there is an isolated tree present in the image graph. Label each of the nodes of this tree by positive and negative signs such that no two adjacent nodes have the same sign. This is equivalent to labelling the nodes of consecutive levels of the tree with alternating positive and negative signs. An example is shown below.



A positive sign associated with any of the nodes i is to indicate that the corresponding implicit cost Ui is to be increased by some increment δ which is yet to be determined. Similarly, a negative sign is to indicate that the corresponding implicit cost is to be decreased by an amount δ.

Suppose the positively labelled nodes belong to a set A, the negatively labelled nodes to a set B and the remaining nodes in the image graph to a set C.

Now consider the maximum value δ can assume in any given case. For the nodes A the maximum value to which δ can be raised is $\delta+$ given by

$$\delta + = \frac{\min (\delta Cij)}{2} \qquad \text{where } \begin{matrix} i\in A \\ j\in A \end{matrix}$$

since any further rise in δ will cause elements $\delta Cij_{\substack{i\in A \\ j\in A}}$ to assume negative values.

Nodes belonging to the set C have no alteration in their implicit costs. Thus the maximum value δ can assume such that no negative elements $\delta Cij_{\substack{i\in A \\ j\in C}}$ are introduced is δr given by

$$\delta r = \min (\delta Cij) \qquad \begin{matrix} i\in A \\ j\in C \end{matrix}$$

Any further increase in δr will cause negative elements to be introduced into the reduced cost matrix.

The maximum value to which δ can be raised such that no negative elements are introduced into the reduced cost matrix is given by the minimum value of $\delta+$ and δr so that

$$\delta = \min (\delta +, \delta r)$$

When δ has been determined, the implicit costs U are altered so that for positively labelled nodes in the tree the costs are increased by δ and for negatively labelled nodes in the tree the implicit costs are reduced by δ.

If n_+ is the number of positively labelled nodes in the tree and n- is the number of negatively labelled nodes in the tree, the cost of the initial feasible solution will be increased if $n_+ > n_-$. Thus, the labelling of the nodes of the tree is such that, if possible, n_+ is always greater than n_-.

The amount dC by which the cost is increased is given by

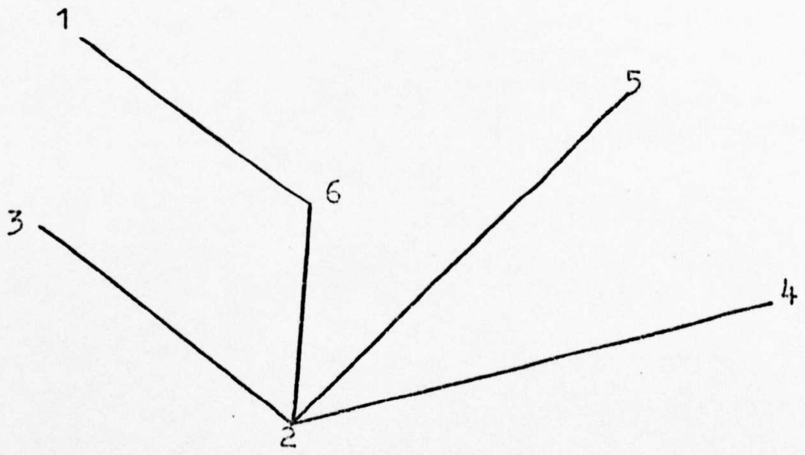
$$dC = (n_+ - n_-) \times \delta$$

Example

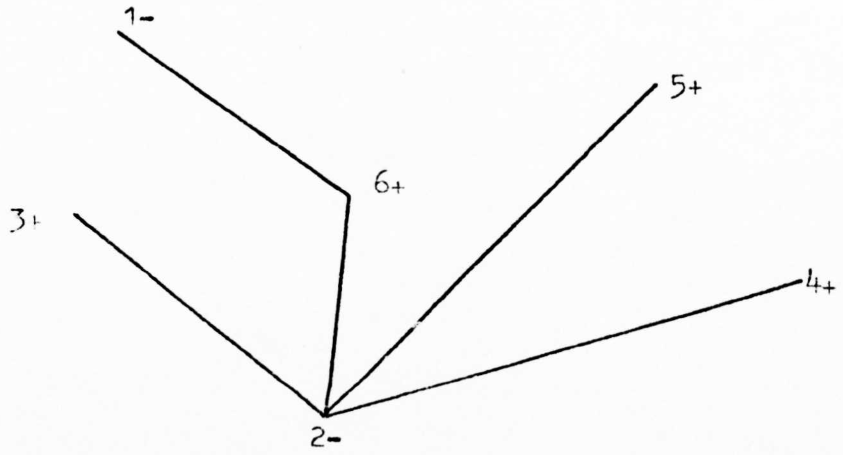
Suppose the reduced matrix δC is given by

$$\delta C = \begin{array}{|c|c|c|c|c|c|c}
\cline{1-6}
\diagup & 1 & 3 & 4 & 2 & 0 & 2 \\
\cline{1-6}
1 & \diagup & 0 & 0 & 0 & 0 & 1 \\
\cline{1-6}
3 & 0 & \diagup & 3 & 2 & 4 & 1 \\
\cline{1-6}
4 & 0 & 3 & \diagup & 1 & 3 & 2 \\
\cline{1-6}
2 & 0 & 2 & 1 & \diagup & 4 & 1 \\
\cline{1-6}
0 & 0 & 4 & 3 & 4 & \diagup & 0 \\
\cline{1-6}
\end{array}
\quad U$$

The image graph corresponding to this is given by



It can be seen in this case that the image graph is a tree. Suppose the nodes are labelled with positive and negative signs such that no two adjacent nodes have the same sign. This is illustrated below



Since all nodes of the image graph are contained in the tree, in this case δ is given by

$$\delta = \delta_{+} = \frac{1}{2}$$

The minimum element in the reduced matrix corresponding to two positively labelled nodes is $\delta C_{54} = 1$ and so the maximum value of δ equals one half this value.

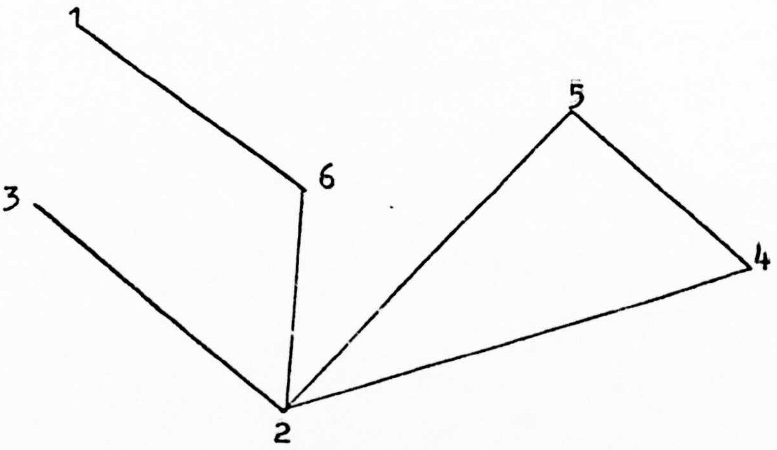
The increase in the cost dC of the initial feasible solution to the image problem is given by

$$dC = \delta.(n_{+} - n_{-})$$
$$= \frac{1}{2} . (4 - 2)$$
$$\therefore dC = 1$$

The implicit costs U_3, U_4, U_5 and U_6 are thus

$U_1 = 1\frac{1}{2}$ $U_2 = \frac{1}{2}$ $U_3 = 1\frac{1}{2}$ $U_4 = 2\frac{1}{2}$ $U_5 = 1\frac{1}{2}$ $U_6 = \frac{1}{2}$

The image graph is now as follows :



corresponding to a reduced matrix as follows

$$\delta C = \begin{array}{|c|c|c|c|c|c|c}
\hline
\diagup & 2 & 3 & 4 & 2 & 0 & 1\frac{1}{2} \\
\hline
2 & \diagup & 0 & 0 & 0 & 0 & \frac{1}{2} \\
\hline
3 & 0 & \diagup & 2 & 1 & 3 & 1\frac{1}{2} \\
\hline
4 & 0 & 2 & \diagup & 0 & 2 & 2\frac{1}{2} \\
\hline
2 & 0 & 1 & 0 & \diagup & 3 & 1\frac{1}{2} \\
\hline
0 & 0 & 3 & 2 & 3 & \diagup & \frac{1}{2} \\
\hline
\end{array} \quad U$$

There are now no isolated trees in the image graph since a loop has been introduced. It is not therefore possible, by the method explained, to increase the cost of the feasible solution any further.

6.12. Upper Bound to Pen Up Problem.

A method has been developed for obtaining an upper bound cost to the pen up problem.

Consider the graph associated with the reduced matrix after the feasible solution to the image problem has been obtained by the methods described earlier. This has been called the image graph and nodes i and j are linked by a segment if

$$\delta Cij = Cij - Ui - Uj = 0$$

If the image graph corresponds to a domino graph (defined in the previous chapter) then there exists a feasible solution to the pen up problem, the cost of which coincides with that of the image problem.

If the image graph is not domino then consider adding segments to the graph until it becomes a domino graph. Consider the segments corresponding to non zero elements in the rediced matrix δC. These segments will correspond to elements

$$\delta Cij > 0$$

These non zero elements can be sorted in ascending order of magnitude. Add the segment corresponding to the smallest element to the image graph and test to see if the image graph is domino. If it is domino then the upper bound cost ub will be given by

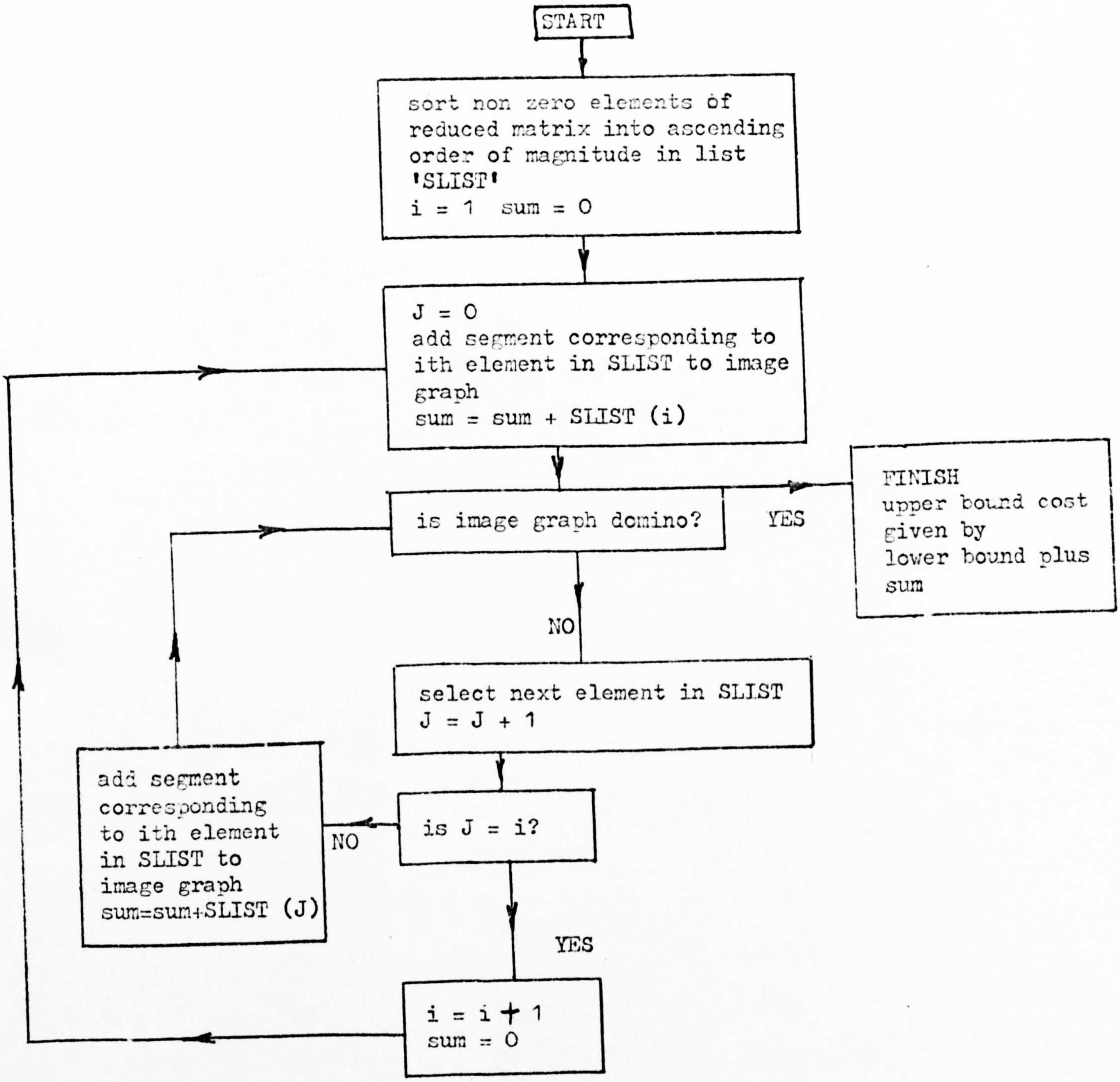
$$ub = lb + c$$

where lb is the lower bound cost, which equals the cost of the feasible solution to the image problem, and c is the cost of the added segment.

If the image graph is not domino then add to the initial graph the segment corresponding to the next smallest element in the reduced cost matrix. By this means it is possible to add segments to the image graph until a domino graph results. The difference in cost between the lower bound cost and upper bound cost will be the cost of the added segments.

It is obvious that this cost should be kept as small as possible for the upper bound to be a good one.

The method by which the segments are added to the image graph is such that the cost of the added segments at any stage tends to be small. The method by which this is achieved is shown in Diagram 6.3.


START
sort non zero elements of reduced matrix into ascending order of magnitude in list 'SLIST'
i = 1 sum = 0
J = 0
add segment corresponding to ith element in SLIST to image graph
sum = sum + SLIST (i)
is image graph domino?
YES
FINISH
upper bound cost given by lower bound plus sum
NO
select next element in SLIST
J = J + 1
is J = i?
NO
add segment corresponding to ith element in SLIST to image graph
sum=sum+SLIST (J)
YES
i = i + 1
sum = 0

SLIST (i) is the size of the ith element in SLIST. Sum is the differenc? in upper bound cost and lower bound cost at any stage.

FLOW DIAGRAM TO OBTAIN AN UPPER BOUND COST TO PEN UP PROBLEM FROM EXISTING LOWER BOUND COST.

Diagram 6.3

6.13. Implicit Enumeration Scheme.

As explained in an earlier section the pen up problem consists of a number of sub problems to which a lower bound to the local optimal feasible solution can be obtained. It is required to find some convenient method by which elements pi (i = 1, ...$\frac{n}{2}$) of P and qi (i=1,...$\frac{n}{2}$) of Q can be generated such that all Sn feasible solutions to the pen up problem can, if necessary, be obtained.

Suppose the elements of P are obtained for each of the sub problems such that

$$pi < pi + 1 \qquad i = 1,2.... \frac{n}{2} - 1$$

This can be achieved by using a recursive 'for' loop to obtain the elements of P for each of the divisions of the set S of n odd nodes into disjoint sets P and Q.

After the elements of P have been obtained the elements of Q are easily found since

$$Q = S-P$$

Now consider finding all feasible solutions for a given division such that an element from P is linked to an element of Q. Obviously it would be unwise to generate a feasible solution more than once.

Suppose the elements of Q are arranged such that

$$qi > pi \qquad i = 1,2......\frac{n}{2}$$

There may be many different ways, for a given sub-problem, in which the elements of Q can be so arranged. For each arrangement the feasible solution will be the $\frac{n}{2}$ pairs of rodes

$$pi-qi \qquad i = 1,2.....\frac{n}{2}$$

By this means it is possible to locate every feasible solution to the pen up problem once and once only. The pen up problem is split into sub-problems and each sub-problem will correspond to obtaining $\frac{n}{2}$ elements of P.

Consider the method set out above for obtaining the 15 feasible solutions corresponding to an n value of 6

Elements of P			Elements of Q		
1	2	3	4	5	6
			4	6	5
			5	4	6
			5	6	4
			6	4	5
			6	5	4
1	2	4	3	5	6
			3	6	5
			5	3	6
			6	3	5
1	2	5	3	4	6
			4	3	6
1	3	4	2	5	6
			2	6	5
1	3	5	2	4	6

For each division as set out above the corresponding elements of P and Q are linked together to form $\frac{n}{2}$ pairs.. For example, in the last division shown, the feasible solution is 12, 34, 56 corresponding to a set P(1,3,5). By this method every feasible solution of the pen up problem is obtained but it will be seen that only 5 sub-problems are found in this case. This is because of the constraints placed on the elements of P and Q. It will always be necessary to have 1 in set P since if it were in Q it would never be possible to have qi>pi for this particular element. If 6 was allowed in set p then it would be possible to have sets of P of

126, 136, 146, 156 and 145 which accounts for the other 5 sub-problems.

The general implicit enumeration scheme will obtain elements of P and from this elements of Q can be obtained. A feasible solution to the dual assignment problem of order $\frac{n}{2}$ for sets P and Q will then be obtained. If this cost is lower than the best solution so far then feasible solutions to the pen up problem linking elements of P and Q are obtained. As better feasible solutions to the pen up problem are obtained these will replace the best existing solution. Thus by using this implicit enumeration scheme, it is possible to be able to skip sub-problems and the feasible solution associated with them.

<u>NOTE</u>.

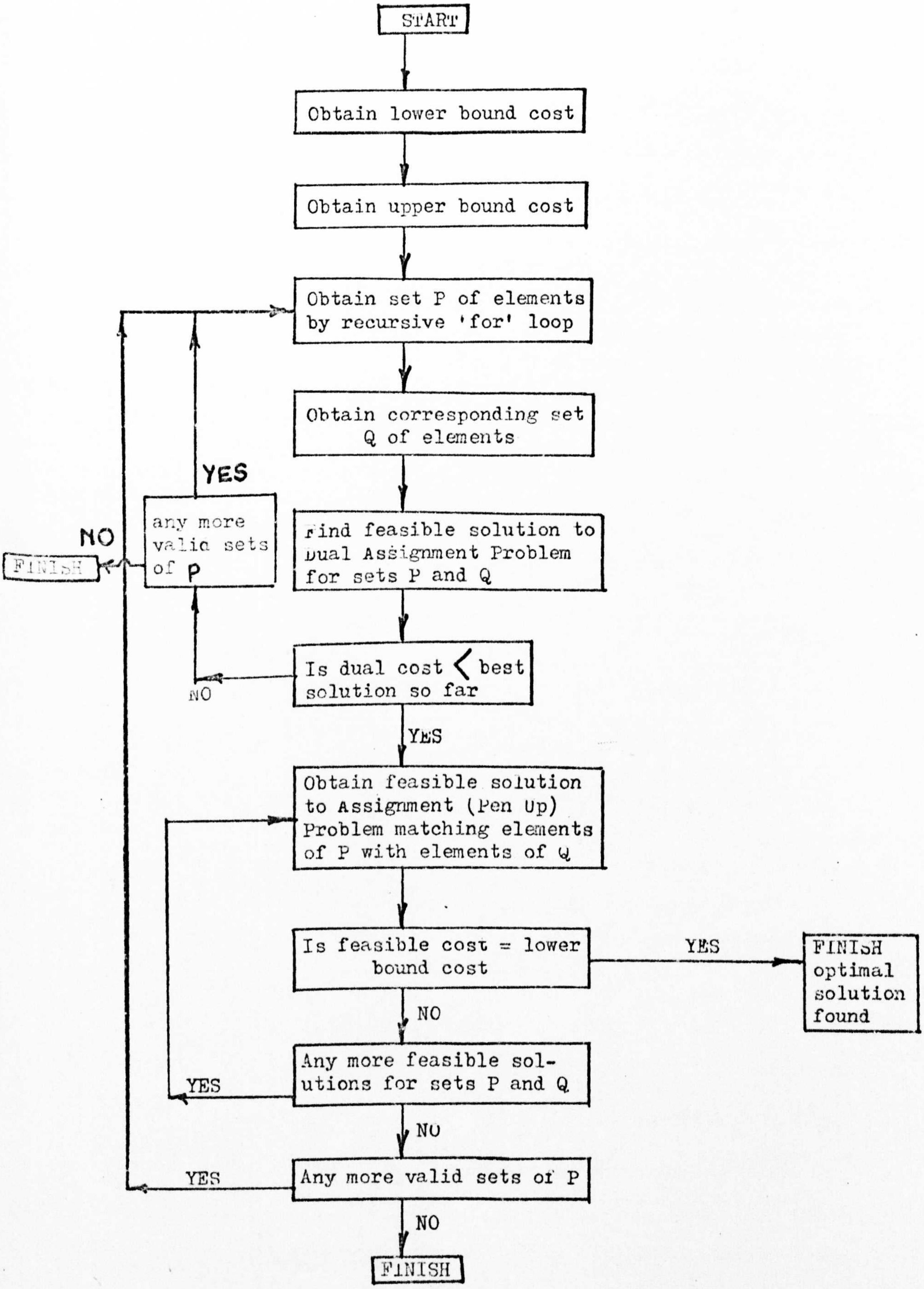
In the C.P.U. timing experiments, the results of which are given in the following chapter, the number of subsets which need to be examined is in each case determined and these have been compared with the total number of subsets for the given value of n.

The number of subsets in this case do not correspond to the values of Gpq but are equal to the number of sets of P such that $pi < pi + 1$.

In each case the total number of subsets could be obtained by incrementing a counter each time a set P of elements was obtained. If the feasible cost to the dual assignment problem with sets P and Q was less than the best existing solution to the pen up problem it was necessary to obtain feasible solutions to the pen up problem. Another count could then be kept of the number of subsets which needed to be examined more closely.

Thus in the results of Chapter 7 the number of subsets do not in fact refer to Gpq.

TEXT
CUT OFF IN THE
ORIGINAL


START
Obtain lower bound cost
Obtain upper bound cost
Obtain set P of elements by recursive 'for' loop
Obtain corresponding set Q of elements
Find feasible solution to Dual Assignment Problem for sets P and Q
Is dual cost < best solution so far
NO
any more valid sets of P
YES
NO
FINISH
YES
Obtain feasible solution to Assignment (Pen Up) Problem matching elements of P with elements of Q
Is feasible cost = lower bound cost
YES
FINISH optimal solution found
NO
Any more feasible solutions for sets P and Q
YES
NO
Any more valid sets of P
YES
NO
FINISH

FLOW DIAGRAM ILLUSTRATING THE GENERAL IMPLICIT ENUMERATION SCHEME

Diagram 6.4

6.14. Binary Chopping.

The general implicit enumeration algorithm will obtain an optimal solution to the pen up problem by looking for feasible solutions better than the existing best solution. By using this approach it is always certain to obtain an optimal solution but for large values of n, the order of the cost matrix, the C.P.U. time taken to do so could well become very large. Consider now a simple method by which a good feasible solution to the pen up problem can be obtained much more quickly.

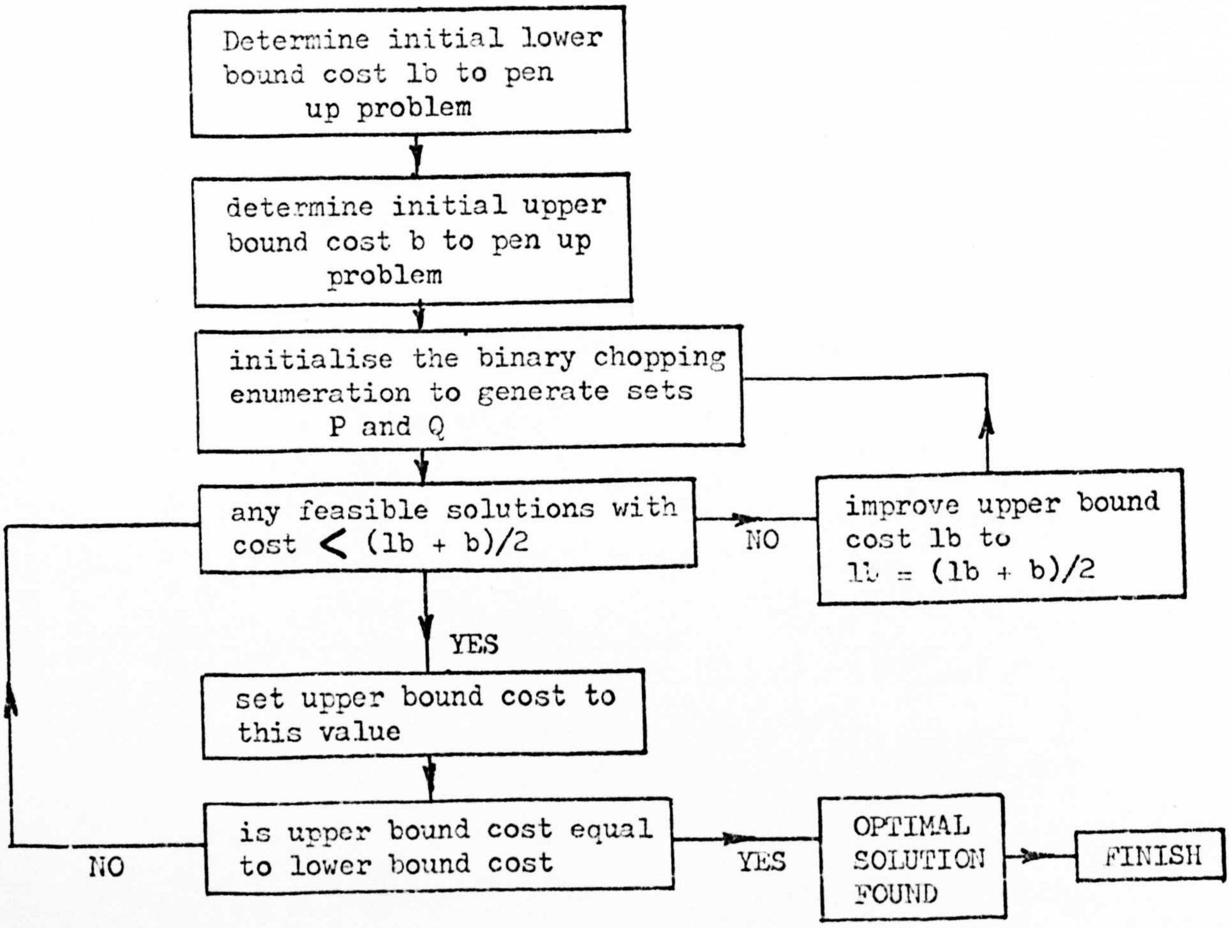
Suppose the cost of the existing best solution at any time is b and the lower bound cost to the pen up problem is lb. Instead of looking for feasible solutions better than b suppose feasible solutions having a cost better than (lb + b)/2 are looked for.

The advantage of this method is that there will be a much better chance of being able to skip sub-problems of the pen up problem. In addition, if the initial upper bound of the pen up problem was a poor one a better feasible solution can be obtained very quickly. It can happen, of course, that when the upper bound is particularly good no feasible solutions with a cost better than (lb + b)/2 exist especially if the initial lower bound cost was poor. In this case several different methods could then be tried. For example, the initial lower bound cost could be replaced by (lb + b)/2 and the method repeated. This binary chopping approach is particularly beneficial if only a given amount of C.P.U. time is to be allowed in obtaining a good feasible solution or where a good feasible solution has to be found quickly.

6.14.1. Optimal Solution by Backtracking. In the general scheme of binary chopping described earlier the exercise was not necessarily to reach an optimal solution but rather obtain a good feasible solution as quickly as possible. An optimal solution can still be obtained by implementing a backtracking technique which uses constantly improving upper and lower bound costs.

After each best feasible solution (with cost c) has been obtained by a binary chopping enumeration, the new upper bound cost can be replaced by c. The new lower bound cost lb can be replaced by (lb + b)/2 since no feasible solutions with a cost better than this exist. By this means it is possible to implement a number of binary chopping enumerations until it is certain that an optimal solution has been obtained. This will be so when the upper and lower bound costs are equal in value.

The method of obtaining an optimal solution is shown as a flow diagram.


Determine initial lower bound cost lb to pen up problem
determine initial upper bound cost b to pen up problem
initialise the binary chopping enumeration to generate sets P and Q
any feasible solutions with cost < (lb + b)/2
NO
improve upper bound cost lb to lb = (lb + b)/2
YES
set upper bound cost to this value
is upper bound cost equal to lower bound cost
NO
YES
OPTIMAL SOLUTION FOUND
FINISH

BACKTRACKING SCHEME TO OBTAIN AN OPTIMAL SOLUTION TO PEN UP PROBLEM BY CONSTANTLY IMPROVING THE UPPER AND LOWER BOUND COSTS.

Diagram 6.5

6.15. Reducing the Number of Pen up Distances.

The number of possible pen up distances associated with a line drawing consisting of n odd degree nodes is nx(n-1)/2. As n increases to high values the number of possible pen up distances becomes very large and it is not practicable to consider all the pen up distances. If the pen up distances vary greatly in size then it is even more unreasonable to consider all the pen up distances since a good feasible solution to the pen up problem is unlikely to contain many large pen up distances.

The reduction in the number of pen up distances can be simply achieved by only considering those pen up distances which are less than some small value f. This is equivalent to setting up a new pen up problem where an odd node is not now allowed to be paired with any of the other odd nodes. Nodes are only allowed to be paired if the distance between them is less than the value of f.

By eliminating some of the pen up distances each of the odd nodes i of the line drawing is associated with a number di of other odd nodes such that

$$di \leqslant n-1 \qquad i = 1,2.....n$$

In the general implicit enumeration scheme di=n-1 (i = 1,2....n) since each of the odd nodes was associated with every other odd node.

Note that with a reduction in the number of pen up distances an optimal solution to this pen up problem may not coincide with an optimal solution to the original problem with all pen up distances allowed. This is because it is possible, though unlikely, that an optimal solution will include large pen up distances ($\geqslant f$). The value of f is thus important in that it should be small enough to enable a fairly large number of pen up distances to be eliminated but of course it is also possible that a feasible solution to the reduced problem may not exist if f is too small.

A convenient value can be determined for f from the upper bound procedure discussed in Section 6.12. f can be assigned the value of the line segment with greatest cost which is added to the original image graph to make it into a domino graph.

By this means it is certain that both a feasible solution exists to the pen up problem and that f is reasonably small in value.

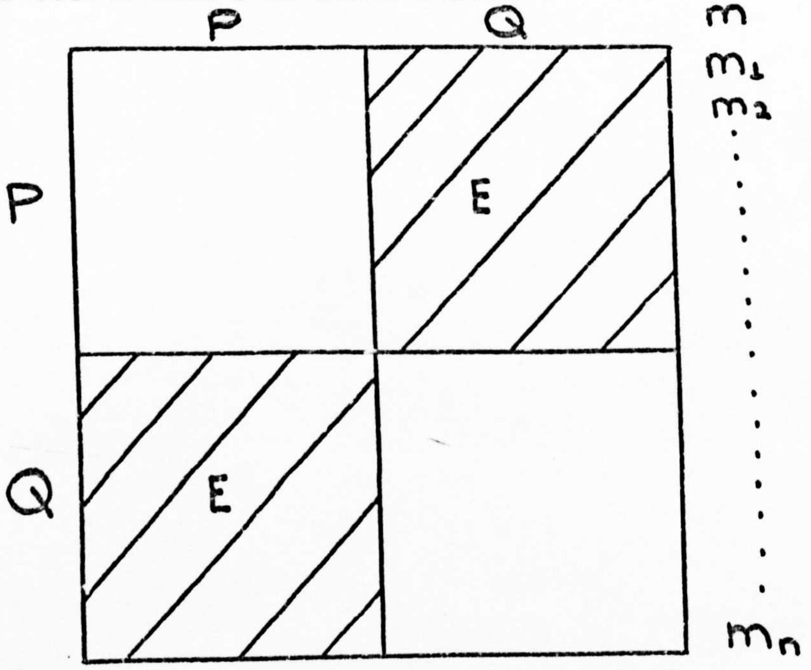
6.16. Sets of Sub Problems.

In the general implicit enumeration algorithm, the set of odd nodes S was divided into two disjoint sets P and Q and a lower bound to the local optimal solution was found by obtaining a feasible solution to the associated dual assignment problem. This was achieved by assigning values to two sets of implicit costs $ai(i=1,2.....\frac{n}{2})$ and bj $(j = 1,2.....\frac{n}{2})$ such that if E was the cost matrix corresponding to the given sub problem

$$Eij - ai - bj \geqslant 0 \qquad \begin{matrix} i \in P \\ j \in Q \end{matrix}$$

Now it is possible to obtain the lower bound cost by using just one set of implicit costs m_i $(i = 1,2.......n)$ which are associated with the rows of the original cost matrix C of the pen up problem.

Suppose the rows and columns of C are permuted simultaneously so that E corresponds to one quarter of the original cost matrix C as shown below



The implicit costs m are assigned values such that a zero is introduced into the shaded portions of each row of C, the reduced cost matrix given by

$$\delta C_{ij} = C_{ij} - m_i - m_j$$

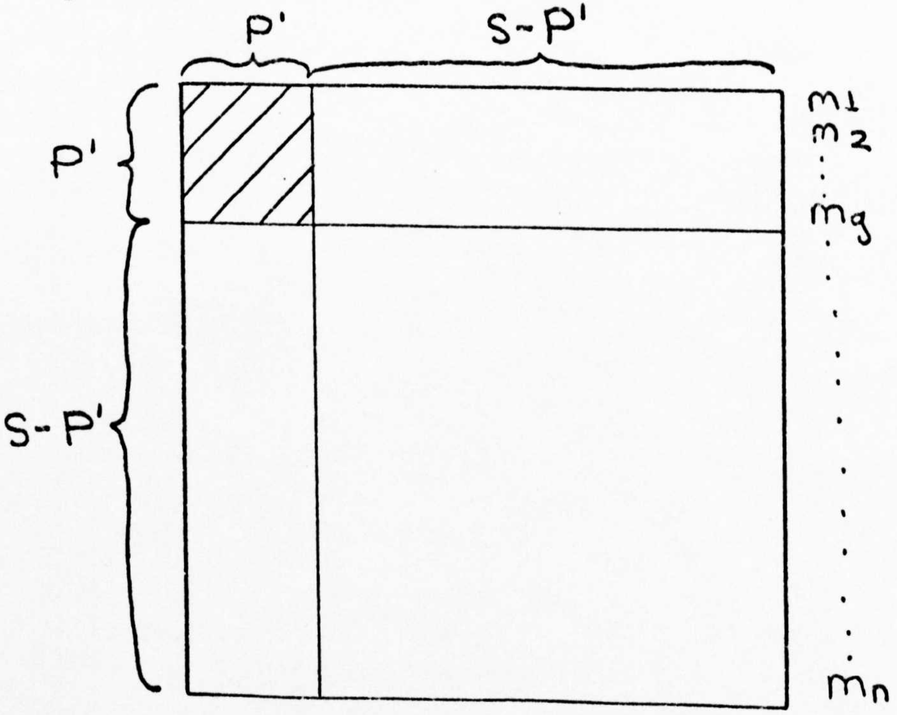
In this way the implicit costs a_i $(i = 1,2....\frac{n}{2})$ can be considered to correspond to the implicit costs m_i $(i = 1,2...\frac{n}{2})$ and the implicit costs b_j $(j = 1,2...\frac{n}{2})$ to m_i $(i = \frac{n}{2} + 1, \frac{n}{2} + 2....n)$. It is thus possible to obtain a feasible solution to the dual assignment problem by utilising one set of implicit costs m and using the original cost matrir C of the pen up problem.

The elements of P, pi(i=1,2....$\frac{n}{2}$) are obtained by the use of a recursive for loop such that

$$p_{i+1} > p_i \qquad i = 1,2......\frac{n}{2} - 1$$

Suppose that the number of elements obtained at any stage in set P is g where g $< \frac{n}{2}$. The importance of the disjoint sets P and Q was that elements from P were not allowed to be paired together and elements from Q were not allowed to be paired together.

Corresponding to the incomplete set P of cardinality g will correspond a number of different complete sets P of cardinality $\frac{n}{2}$. Suppose that it is required to obtain a lower bound to the local optimal solutions of the various sub problems corresponding to each of the completed sets P. Suppose the incomplete set P of cardinality g is P' and consider arranging the rows and columns of the original cost matrix C as follows



The implicit costs mi (i = 1,2....g) will be the smallest costs assigned for each of the completed sets P and will thus correspond to choosing the smallest elements in the unshaded portions of rows 1,2.....g. Thus a zero will be introduced into the unshaded portions of the rows 1,2....g of the reduced matrix C. This corresponds to the fact that elements of P will never occur in different sets of the associated sub problems whereas elements of S-P' will, and any one of these is thus allowed to be paired with an element from P'.

The elements of S-P' are allowed to be paired with each other and so no restriction exists for the implicit costs mi (i = g + 1, g + 2,......n).

The implicit costs must always be chosen so that the elements of the reduced cost matrix

$$\delta Cij = Cij - mi - mj \geqslant 0$$

The lower bound cost to the best local optimal solution existing for the various associated sub problems is thus

$$\sum_{i=1}^{n} mi$$

Note that the method simply corresponds to finding the least of each implicit cost mi (i = 1,2....n) for each of the possible sub-problems having one of the sets containing the elements of P'.

Thus, as the elements of P are obtained it is possible to determine lower bounds to the uncompleted set and so determine whether it is possible with the elements already contained in P to improve on the best existing feasible solution after completing the set P.

6.16.1. Subset Grouping Depth. A parameter sgd, called the subset grouping depth exists which determines the number of elements contained in P before a check on the lower bound is made. The elements of P are obtained as p1, then p2, then p3 until $p\frac{n}{2}$ has been found.

If sgd = $\frac{n}{2}$ the method will correspond to the original implicit enumeration scheme since $\frac{n}{2}$ elements are obtained. If sgd = 2, the first two elements of P are obtained and a lower bound is obtained to the best feasible solution which exists for the set of sub-problems such that p1 and p2 are not paired together. Every time two new elements are obtained for P the associated lower bound will be determined.

Thus, it is of considerable interest to vary the value of sgd for various n values in order to ascertain the best value of sgd for a given n. This would be expected to be about n/4 because the gain in skipping sets of sub problems would then be balanced by the number of lower bounds to be calculated. If the lower bound to the set of sub problems exceeds the best existing feasible solution to the pen up problem then the general method continues until $\frac{n}{2}$ elements of P have been obtained.

It is possible to obtain the lower bound to the set of sub problems for each element of P obtained after the element psgd has been found but this could lead to excessive calculation and so cancel out any advantage gained in being able to skip sets of sub problems.

CHAPTER 7

C.P.U. TIMING EXPERIMENTS ON IMPLICIT ENUMERATION ALGORITHM

Extensive C.P.U. Timing experiments on the various enumerative schemes used to solve Pen Up Problem.

CHAPTER 7

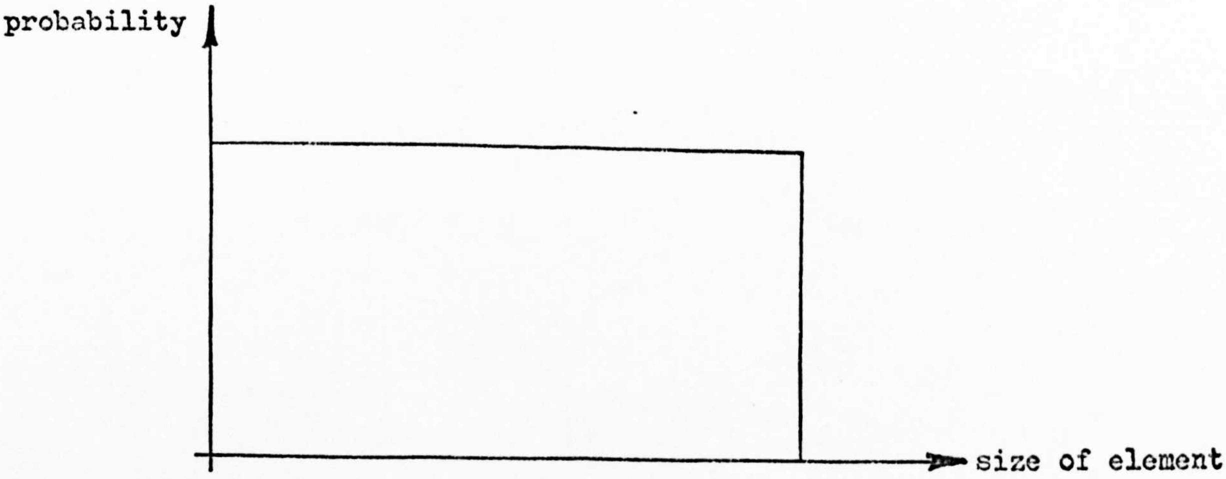
C.P.U. TIMING EXPERIMENTS ON IMPLICIT ENUMERATION ALGORITHM

7.1. Introduction.

The various methods by which the performance of the general implicit enumeration algorithm can be improved were explained in the previous chapter. In this chapter, C.P.U. timing experiments have been carried out on each of the methods, to ascertain the order of the improvement in C.P.U. times brought about to reach both good and optimal solutions. In addition, C.P.U. timing tests have been carried out on the general implicit enumeration algorithm, the heuristic method explained in Chapter 5 and various other algorithms where it is of interest to ascertain the rise in C.P.U. time as the order of the cost matrix n was increased.

In practice, the rows and columns of the cost matrix correspond to nodes of a line drawing and so the distance between the nodes, which correspond to the elements of the cost matrix, should satisfy the triangle inequality. In this chapter comparisons are being made between the various enumerative schemes put forward earlier and it is therefore not of vital importance that the cost matrices should satisfy the triangle inequality. The cost matrices which have been used have elements supplied by a random number generator and so the elements will not in general satisfy the triangle inequality. The random number generator used has a rectangular distribution and elements with integer values were generated between a lower limit 1 and an upper limit U. Unless otherwise stated the lower limit was 0 and the upper limit was 100.

The significance of the distribution being rectangular is that any element between limits U and 1 has an equal chance of occurring. The distribution is termed rectangular since the probability curve is rectanguler in shape as shown below



The program used for the random number generator was taken from the I.B.M. Scientific Subroutine Package and is written in Fortran.

The timing of the programs was achieved by using a C.P.U. timer, written in Assembler. The timer, CPUT(a) returned an integer value 'a' which was the number of time units used by the central processor unit since the last call of the timer. Thus it was possible to determine the C.P.U. time used by a particular section of any program. It was convenient to convert the C.P.U. time taken to seconds and this was in fact carried out each time.

In most of the cases, timing experiments were carried out for several values of n, the order of the cost matrix, so that the variation of C.P.U. time with increase in n could be established. It has been possible from the results obtained, to draw various conclusions on the applicability of each of the methods.

The algorithms have all been written in Algol 60 and the experiments have been carried out on the I.B.M. 360/67 computer. M.T.S. was used for each of the tests.

7.2. The Standard Implicit Enumeration Algorithm.

The standard implicit enumeration algorithm determines a lower bound to the pen up problem by the methods explained in the previous chapter. From this feasible solution to the image problem it was possible, by the addition of segments (with small cost) to the image graph, to obtain a feasible solution to the pen up problem and the corresponding cost gave a good upper bound to the pen up problem.

A recursive 'for' loop was used to generate the elements of one of the disjoint sets P. The elements of the other set Q could be quickly determined since Q = S - P where S is the set of n odd degree nodes. The sets P and Q formed a sub-problem of the pen up problem. A lower bound to this sub-problem was found by determining a feasible solution to the associated dual assignment problem. If this cost was greater than the best solution found so far, it was not possible, by obtaining feasible solutions to this sub problem, to improve on the best solution and so the sub-problem was skipped and another set of elements P was determined. If the cost was less than the best solution obtained, all feasible solutions of the sub-problem better than the existing best solution were located.

The number of sub problems skipped in each case was counted so as to form some guide as to the efficiency of the implicit enumeration algorithm.

Values of n from 4 to 18 were tested

before the C.P.U. times required became too large. The results of these tests are presented below.

7.2.1. Results

n	No. of Subsets Checked	Total No. of Subsets	C.P.U. Time Secs.	Lower Bound	Upper Bound	Optimal Solution
4	0	2	0.01	52	52	52
6	0	6	0.01	56	56	56
8	0	16	0.01	62	62	62
10	0	70	0.01	87	87	87
12	0	252	0.01	102	102	102
14	35	924	46.3	84	109	95
16	374	3432	371.	82	115	98
18	305	12870	1162	78	109	86

The results are presented graphically in Diagram 7.1.

7.2.2. Conclusions

The results show that the method by which the upper bound is obtained from the previously determined lower bound is particularly good at the lower values of n and it is not until n reaches a value of 14 that a difference in cost between the two bounds occurs.

Thus, for values of n from 4 to 12 there is no need for any enumeration at all. As n increases beyond the value of 12 however, the rise in the C.P.U. time is exponential in nature and with an n value of 18 the C.P.U. time required to reach an optimal solution is 1162 seconds. The rapid rise in C.P.U. time corresponds to the exponential rise in the number of associated sub-problems to be solved.

The number of subsets which need to be tested varies from between 2½% to 10% and in general it would be expected that the percentage number of subsets checked would fall as n increases. At an n value of 18 only 2½% of the total number of sub-problems need to be solved.

It is of particular interest to investigate the actual C.P.U. times required to reach the optimal solutions in each case so as to ascertain the amount of C.P.U. time expended after an optimal solution has been found. The results corresponding to the initial test were as follows:

n value	C.P.U. Time Required to Reach Optimal Secs	Total C.P.U. Time Secs	% C.P.U. Time Wasted in Looking for a Better Solution which does not exist
14	7	46	84
16	93	371	75
18	277	1162	76

It can be seen that the amount of C.P.U. time which is expended after an optimal solution has been found is in each case very large indeed, varying between 75%and 85% of the total C.P.U. time required for each enumeration. This shows the need for a good lower bound for if the lower bound cost actually corresponded to an optimal solution in the above cases, a large amount of C.P.U. time could have been saved.

RISE IN CPU TIME OF STANDARD IMPLICIT ENUMERATION ALGORITH WITH INCREASE OF n

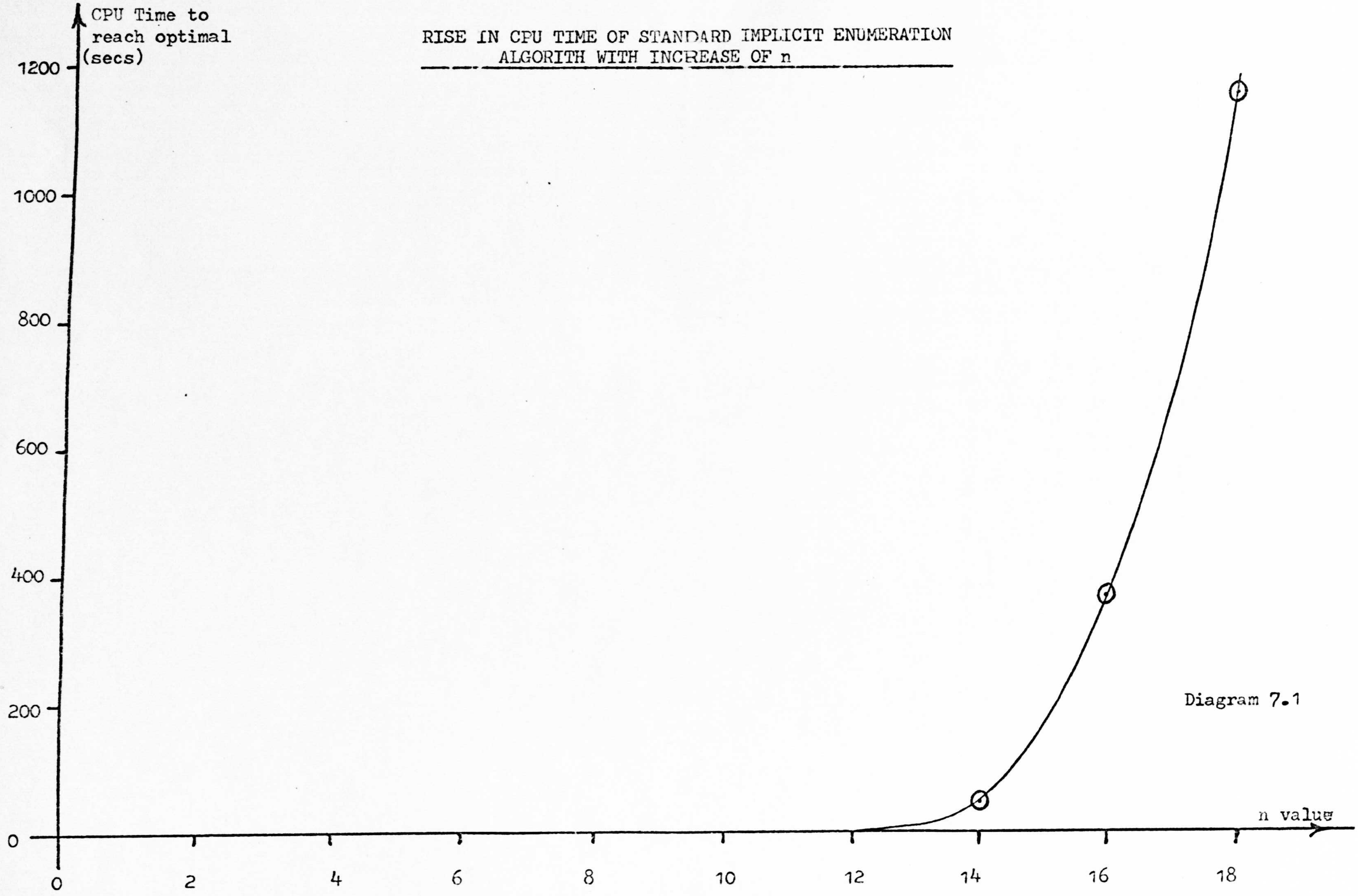


Diagram 7.1

7.3. The Heuristic Method.

The heuristic method of finding a good feasible solution to the pen up problem was described in Chapter 5 and C.P.U. timing tests have been carried out on the method ranging in n values from 4 to 400. The method developed is used to organise any given line drawing into an Eulerian cycle and it thus should be very efficient in choosing the pen up lines.

The solutions to the pen up problem are of course not necessarily optimal but it is of interest to compare the cost of the solutions obtained with the optimal solutions for a given n. Unfortunately, because of the C.P.U. time required to find an optimal solution using the implicit enumeration scheme, the heuristic costs can only be compared with the corresponding optimal costs for values of n up to and including 18.

The timing results are presented as two graphs, one for n values ranging from 4 to 40 and the other for n values ranging from 40 to 400. The timing results are as follows.

7.3.1. Results

n	4	6	8	10	12	.14	16	18	20	24	28	32	36	40
CPU Secs	.013	0.032	0.051	0.077	0.093	0.128	0.171	0.192	0.234	0.347	0.396	0.524	0.686	0.893

n	40	80	120	160	200	240	280	320	360	400
CPU Secs	0.89	2.90	6.56	11.57	17.49	23.83	32.33	43.93	56.82	66.58

The results for the smaller and larger values of n are presented in Diagrams 7.2 and 7.3 respectively.

RISE IN CPU TIME OF HEURISTIC METHOD WITH INCREASE OF n

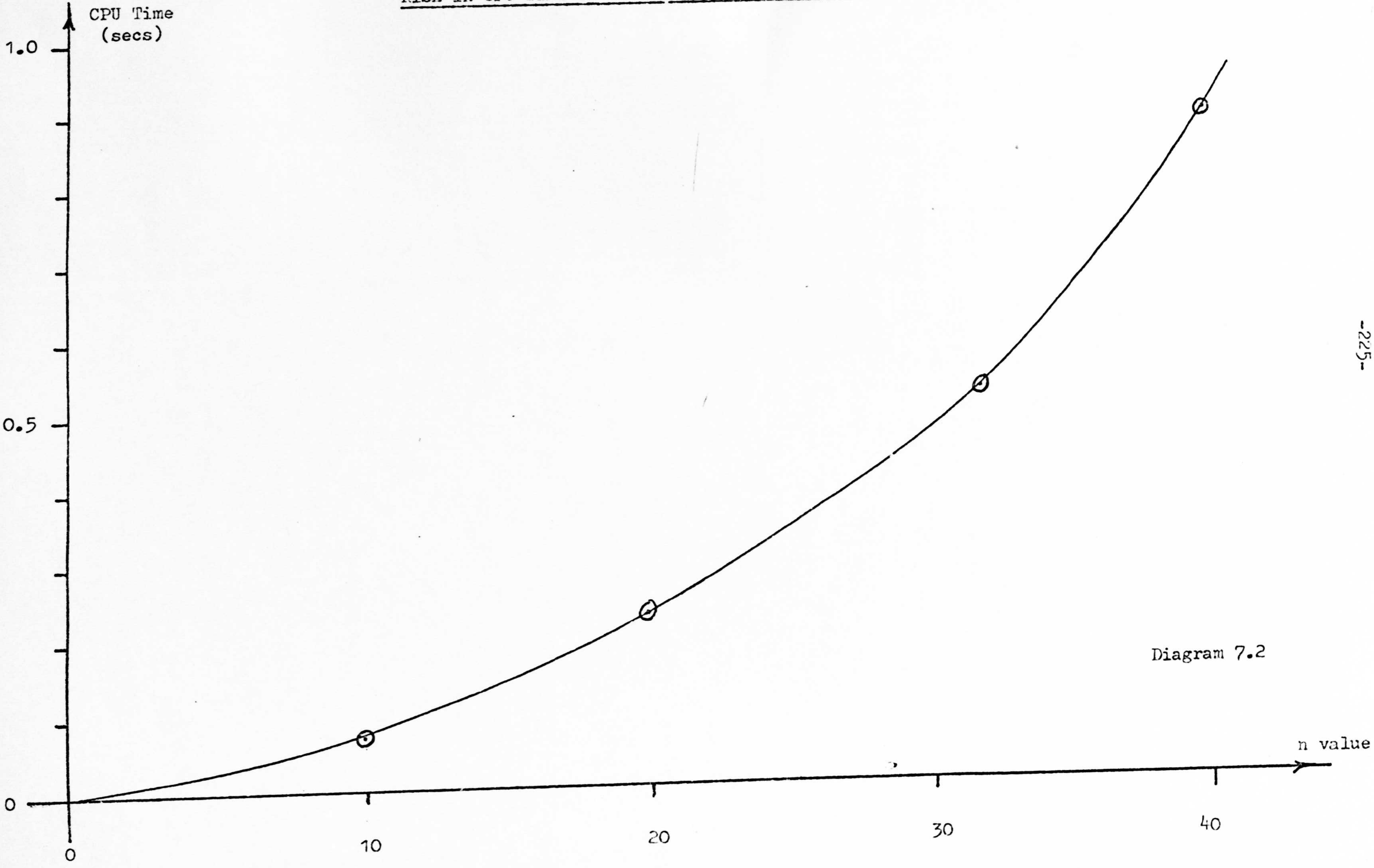


Diagram 7.2

RISE IN CPU TIME OF HEURISTIC METHOD WITH INCREASE OF n

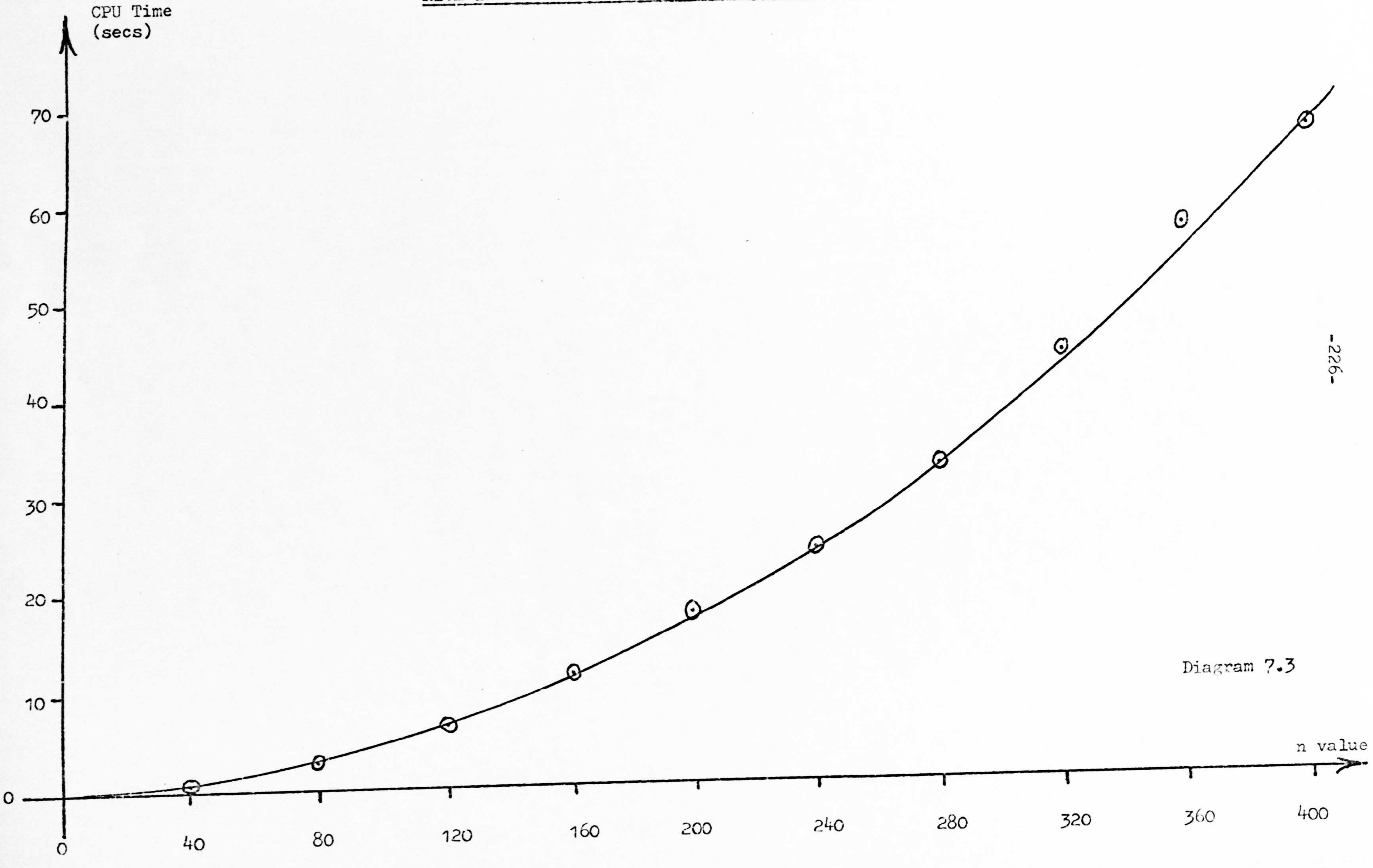


Diagram 7.3

7.3.2. Conclusions.

The results show that the heuristic method is indeed economical of C.P.U. time and although the times rise exponentially with n, the increase in C.P.U. time is fairly small as n increases. At an n value of 400 only just over 66 seconds are required to find the feasible solution. At the smaller values of n the rise in C.P.U. time with n is fairly close to being linear but of course as n increases the rise in C.P.U. time becomes more and more exponential in nature.

The C.P.U. times required to obtain good feasible solutions to the pen up problem are much less than those required to reach optimal solutions using the implicit enumeration algorithm.

It is of considerable interest to obtain the costs using the heuristic method and in this respect the heuristic costs for the various n values have been determined and are given in the table below.

n	4	6	8	10	12
heuristic cost	52	56	62	98	123
optimal cost	52	56	62	87	102

n	14	16	18
heuristic cost	123	141	152
optimal cost	95	98	86

It can be seen that for the lower values of n the heuristic costs are in fact equal to the optimal costs but as n increases to values greater than 8 the difference between the optimal and heuristic costs rises.

At an n value of 10 the difference in the costs is 11 corresponding to an optimal cost of 87 while at the highest value of n tested (18) the difference in the costs is 66 corresponding to an optimal cost of 86.

7.4. Variation in Size of Elements of the Cost Matrix.

The implicit enumeration scheme is only economical if it is possible to skip sets of solutions. The size of the elements or pen up distances should thus have a great effect on the C.P.U. time required by the algorithm to reach an optimal solution. It would be expected that if the elements of the cost matrix differed greatly in size, the algorithm would iterate to an optimal solution much more quickly than if the elements were similar in size. In this respect the total variation, or range in the size of the elements, was varied from 20 to 100 and the C.P.U. times required in each case to reach an optimal solution were determined. The tests were carried out for n values of 10, 12 and 14 and the results are presented below.

7.4.1. Results

n =10

Range of Elements	C.P.U. Time Secs.
20	4.21
40	3.77
60	3.43
80	2.05
100	2.0

n = 12

20	22.34
40	18.8
60	12.94
80	9.32
100	10.0

n = 14

20	212.9
40	218.2
60	144.2
80	96.2
100	73.2

In each case the cost of the smallest element was 20. The cost of the largest element in the cost matrix was thus 120.

The results are presented graphically in Diagrams 7.4, 7.5 and 7.6.

VARIATION OF CPU TIME TO REACH OPTIMAL WITH VARIATION IN RANGE OF ELEMENTS

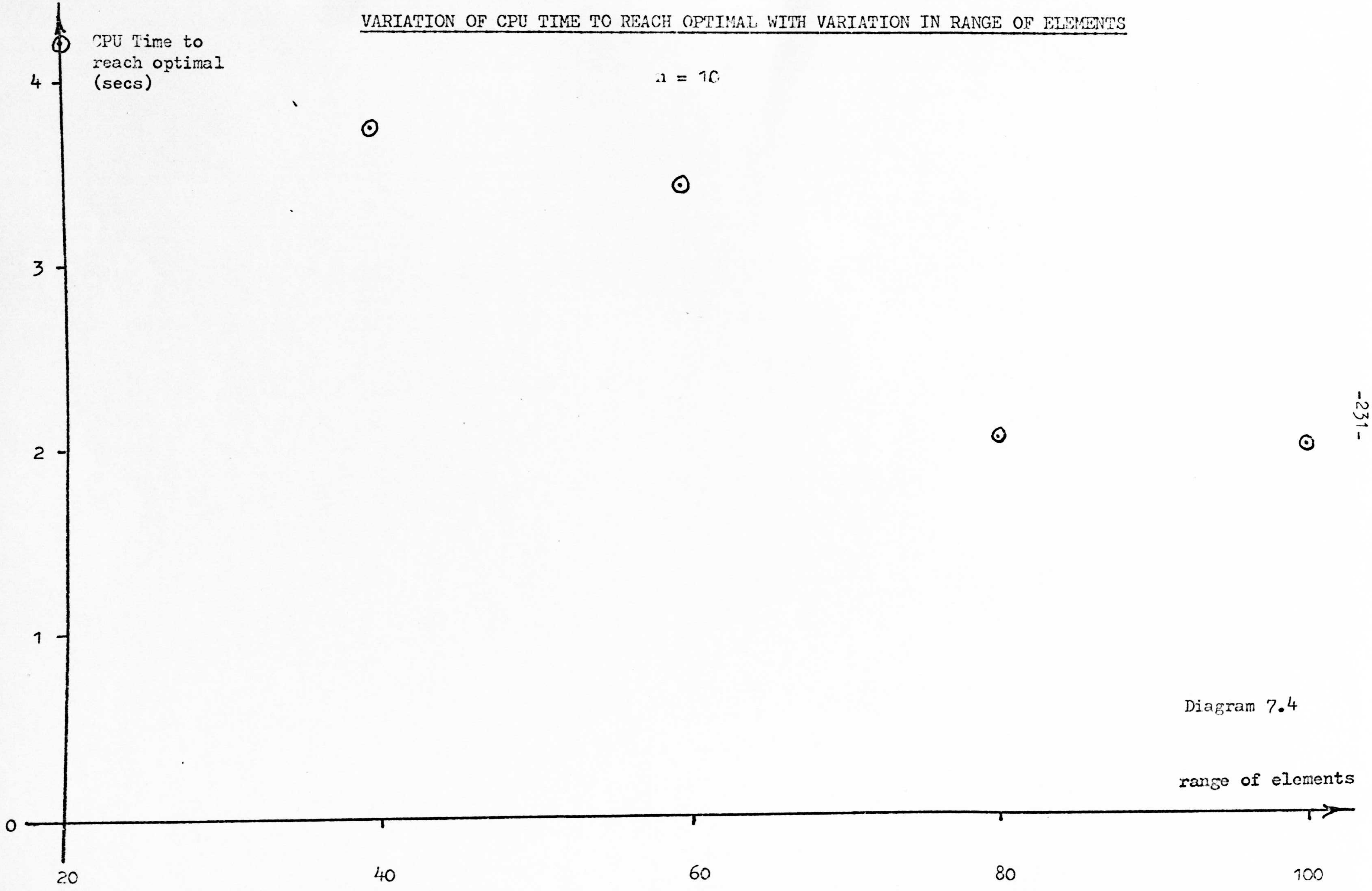


Diagram 7.4

VARIATION OF CPU TIME TO REACH OPTIMAL WITH VARIATION IN RANGE OF ELEMENTS

n = 12

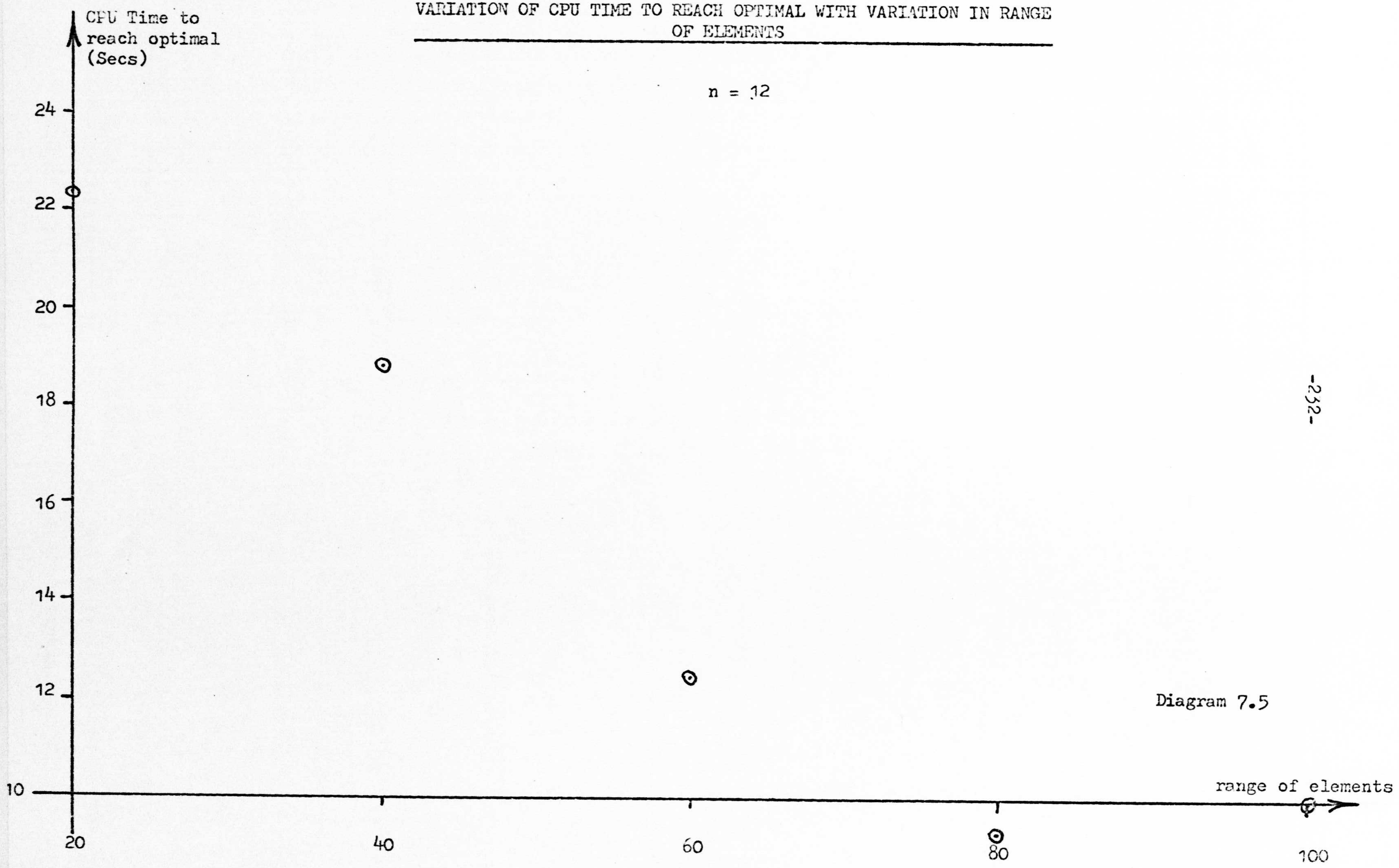


Diagram 7.5

TEXT
CUT OFF IN THE
ORIGINAL

VARIATION OF CPU TIME TO REACH OPTIMAL WITH VARIATION IN RANGE OF ELEMENTS

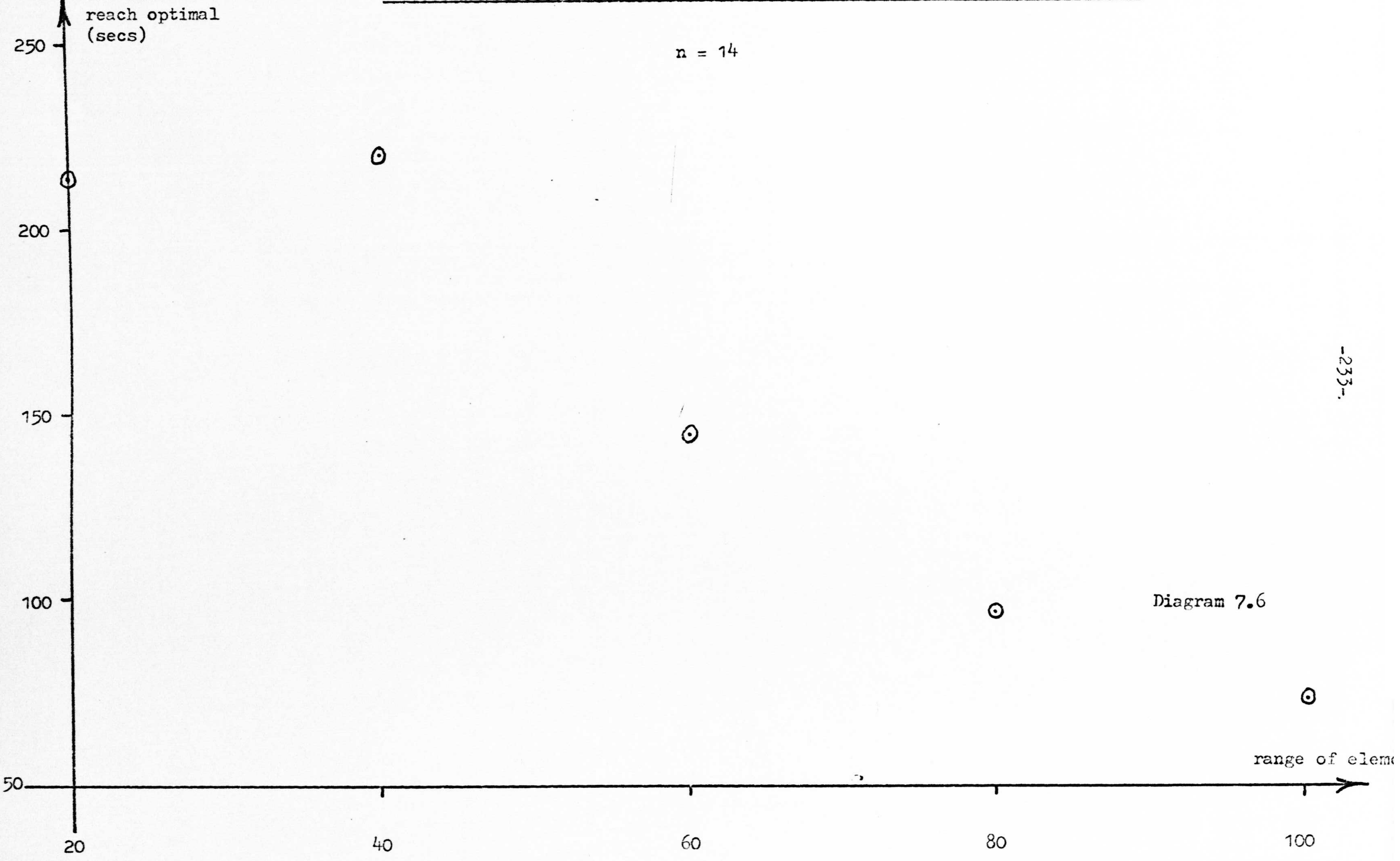


Diagram 7.6

7.4.2. Conclusions

The results show that as the range in the size of the elements in the cost matrix is decreased the C.P.U. times required to reach optimal solutions is increased. It can thus be said that the implicit enumeration scheme is most efficient when the range in the size of the elements is high.

Comparing, in each case, the extremes in the C.P.U. times, it can be seen that the time to reach an optimal solution with a range of 20 is more than double that with a range of 100. Indeed for n = 14 the C.P.U. time required for a range of 100 is just more than one third of the time required for a range of 20.

These results are to be expected since the pen up problem has been divided into a number of sub-problems and the implicit enumeration scheme is only economical of C.P.U. time if it is possible to skip some of these sub-problems, which will correspond to a number of feasible solutions. If the range in the size of the elements is large there is more chance that for a given division of the n nodes into two sets P and Q, which corresponds to a sub-problem, a lower bound to the local optimal solution will be large and so enable the sub-problem to be skipped. If the range is small the local optimal solutions to each sub-problem will tend to be fairly close together and so there will be less chance of skipping the sub-problems.

7.5. Direct Solution of Sub-Problems.

For each division of the n elements into two disjoint sets P and Q the standard implicit enumeration algorithm will obtain feasible solutions to the pen up problem by matching elements from P with elements from Q. It is therefore possible in each of the sub-problems to obtain a number of feasible solutions which will improve on the best solution found so far. More than one iteration can therefore take place in each of the sub-problems.

It is of considerable interest to obtain the local optimal solution to a given sub-problem by solving directly the associated assignment problem of order n/2.

The algorithm used to solve the assignment problems is that of Silver, which is given as algorithm 27 in C.A.C.M.

Wright[28] concluded that Silver's algorithm was more efficient than Kuhn's algorithm to solve the assignment problem and it was because of this, and also the easy availability of the algorithm which was the reason for this particular algorithm being chosen. Although the best method of solving an assignment problem is probably that of Ford-Fulkerson, the purpose of this section is to determine the advantages which exist in solving the associated assignment problems directly and this can be done by using any efficient algorithm to solve the assignment problem.

So that it was possible to compare the two methods for n values less than 14 the initial upper bounds were set fairly high in each case so that it was possible to have a number of iterations.

The n values for which the two methods were compared were 10, 12, 14, 16 and 18 and the results are given below.

7.5.1. Results.

n	10	12	14	16	18
CPU Secs Standard	0.91	2.77	55.3	380	1280
Direct Solution CPU Secs	0.59	1.18	37.4	195	752

COMPARISON OF STANDARD IMPLICIT ENUMERATION ALGORITHM WITH DIRECT SOLUTION OF SUB-PROBLEMS (ASSIGNMENT PROBLEMS)

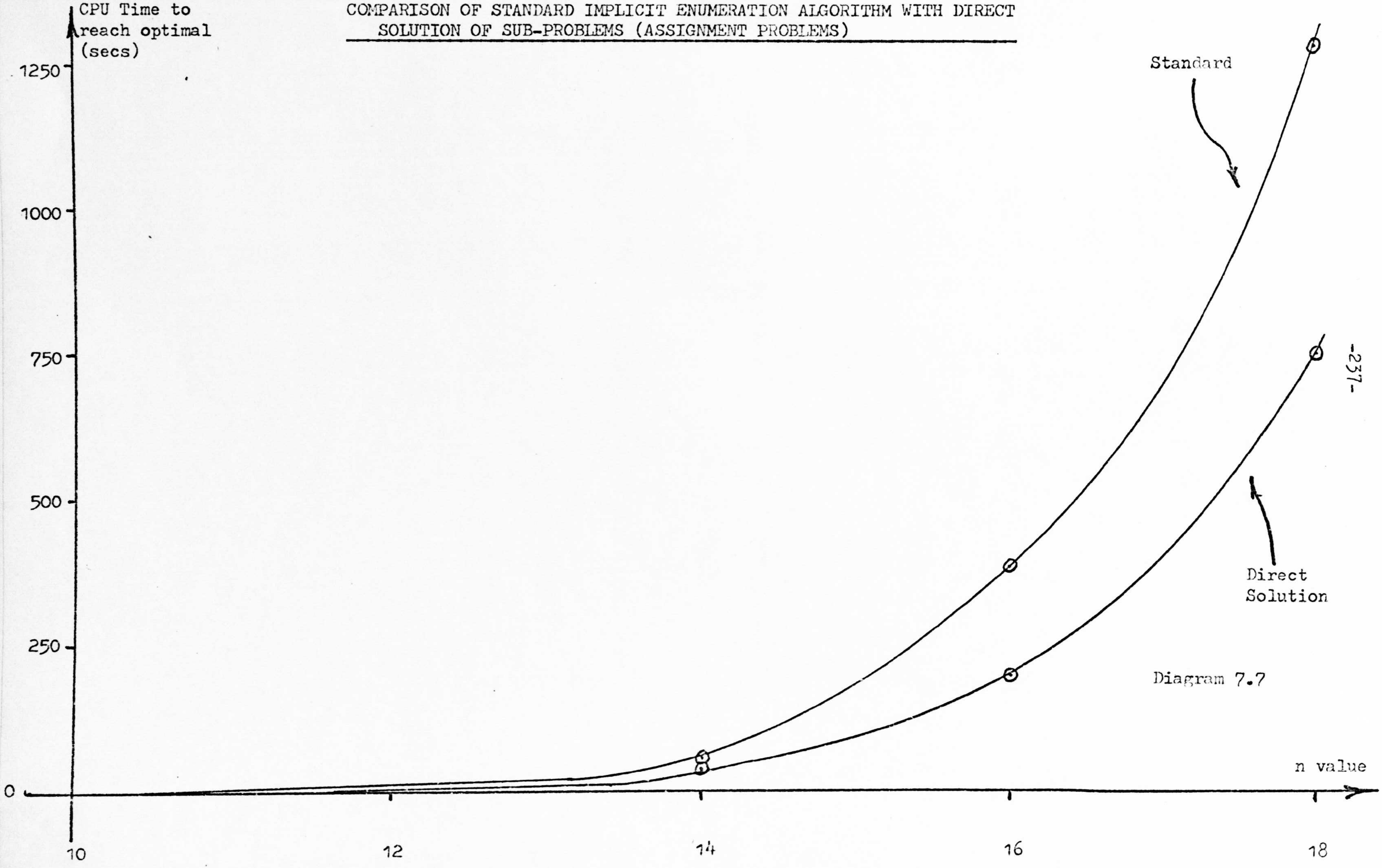


Diagram 7.7

7.5.2. Conclusions.

The results show clearly that obtaining direct solutions to the sub-problems is much more economical than by finding a number of feasible solutions for each division of the elements. In each of the cases tested the direct solution method was of the order of twice as fast as the standard one and as an example it took only 752 seconds to reach an optimal solution for n = 18 compared with 1280 for the standard algorithm.

By using the direct solution method there will always be a smaller number of iterations to the optimal solution of the pen up problem and so it follows that the method will always tend to be quicker than that of the standard method.

It is of interest to compare the number of iterations for each value of n for the direct solution and standard methods. The following number of iterations were made:

n	Number of iterations standard	Number of iterations direct
10	8	3
12	10	2
14	12	5
16	14	5
18	37	3

It can be seen that in each case the number of iterations made by the direct method was much less than that of the standard. The great advantage of the direct method is that many subsidiary iterations are excluded since for each division, only the best feasible solution is found and no iterations within the subsets takes place.

The difference in the number of iterations for the two methods seems to grow with increase of n and for an n value of 18 the number of iterations by the standard method is 37 compared with only 3 of the direct solution

7.6. Reduction of Upper Bound.

In any implicit enumeration scheme in which the mechanism is to iterate to better solutions there must always be an initial feasible solution, usually referred to as an upper bound of the problem. There are of course many initial feasible solutions which can be obtained for the pen up problem but the lower the upper bound the better the chance of skipping feasible solutions.

The method of obtaining the upper bound in the standard algorithm has been explained in detail earlier and consists of adding segments to the image graph associated with the feasible solution to the image problem until a domino graph results. By this means it has been possible to minimise the difference in cost between the upper and lower bounds.

It would not, of course, be unreasonable to obtain an upper bound from the heuristic method obtained earlier which appears to be very economical of C.P.U. time even at extremely large values of n. In this respect it has been necessary to carry out timing tests on the standard upper bound algorithm to establish the variation in the C.P.U. times with increase of n. The results of these experiments are presented in a later section.

This section presents results of experiments carried out to determine the variation in C.P.U. time to reach an optimal solution with variation of the initial upper bound. Separate tests have been carried out for n values of 10, 12, 14 and 16 and the results are presented in graphical form in Diagrams 7.8, 7.9, 7.10 and 7.11.

7.6.1. Results.

n = 10

Upper Bound	CPU Time to reach Optimal secs.
125	0.862
120	0.847
115	0.844
110	0.843
105	0.837
100	0.689
95	0.447
90	0.324

Optimal = 87

n = 12

Upper Bound	CPU Time to reach Optimal secs.
125	2.61
120	2.6
115	2.11
110	1.52
105	0.96

Optimal = 102

n = 14

Upper Bound	CPU Time to reach Optimal secs
125	53.6
120	53.3
115	52.5
110	48.5
105	44.4
100	44.3

Optimal = 95

n = 16

Upper Bound	CPU Time to reach Optimal secs.
125	391
120	382
115	371
110	340
105	317
100	315

Optimal = 98

TEXT
CUT OFF IN THE
ORIGINAL

VARIATION OF CPU TIME TO REACH OPTIMAL WITH VARIATION OF UPPER BOUND

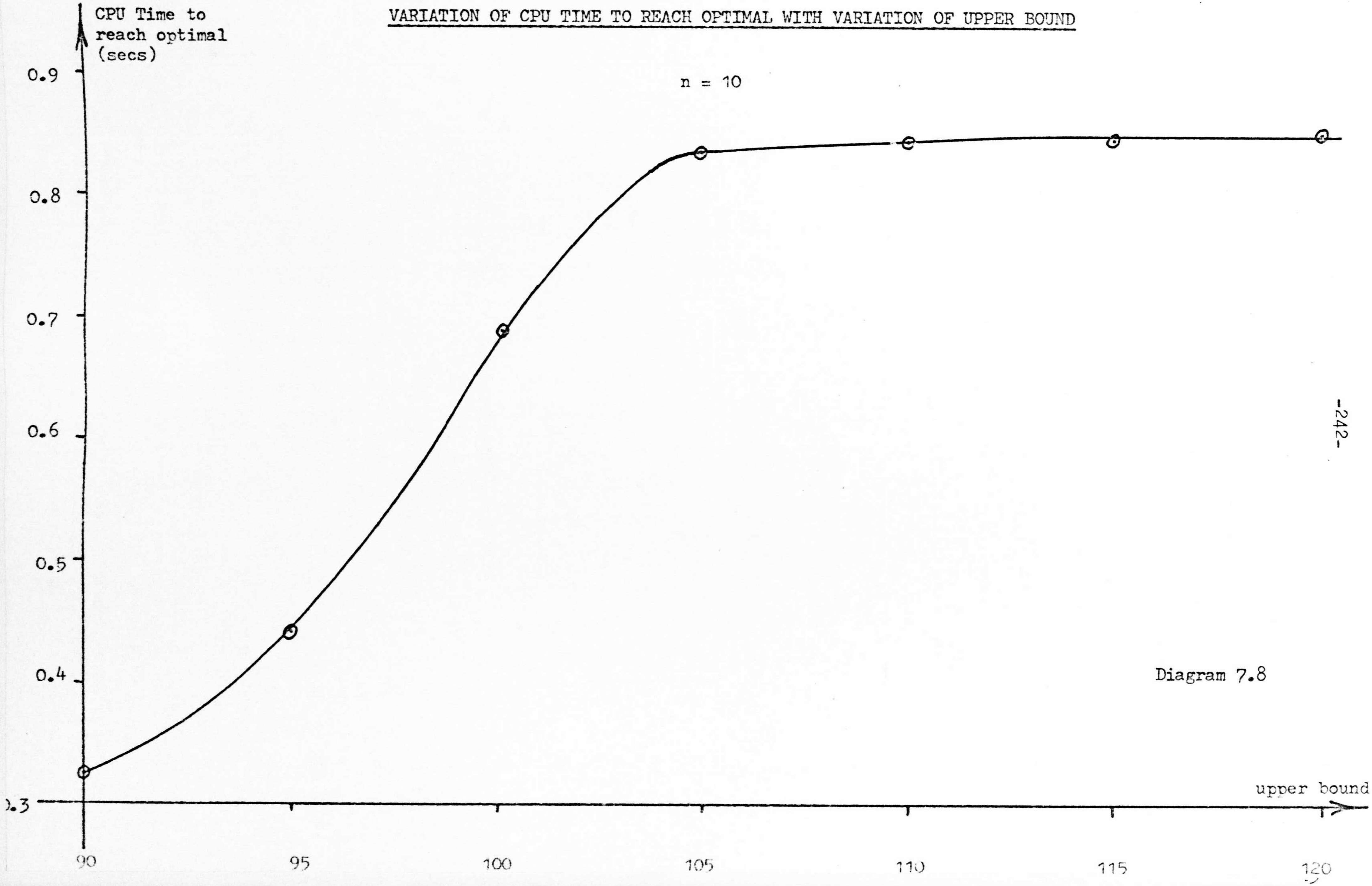


Diagram 7.8

VARIATION OF CPU TIME TO REACH OPTIMAL WITH VARIATION OF UPPER BOUND

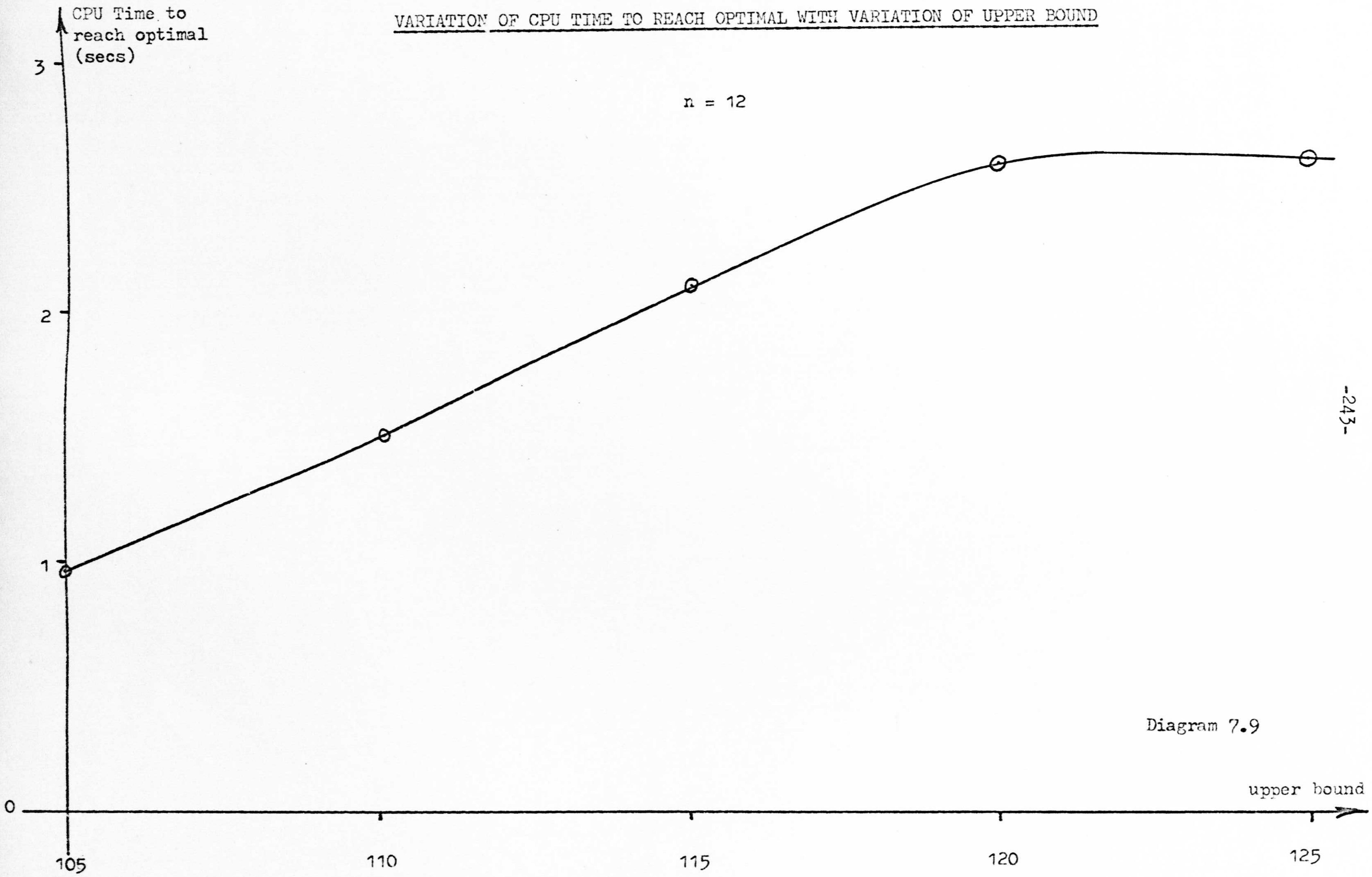


Diagram 7.9

VARIATION OF CPU TIME TO REACH OPTIMAL WITH VARIATION OF UPPER BOUND

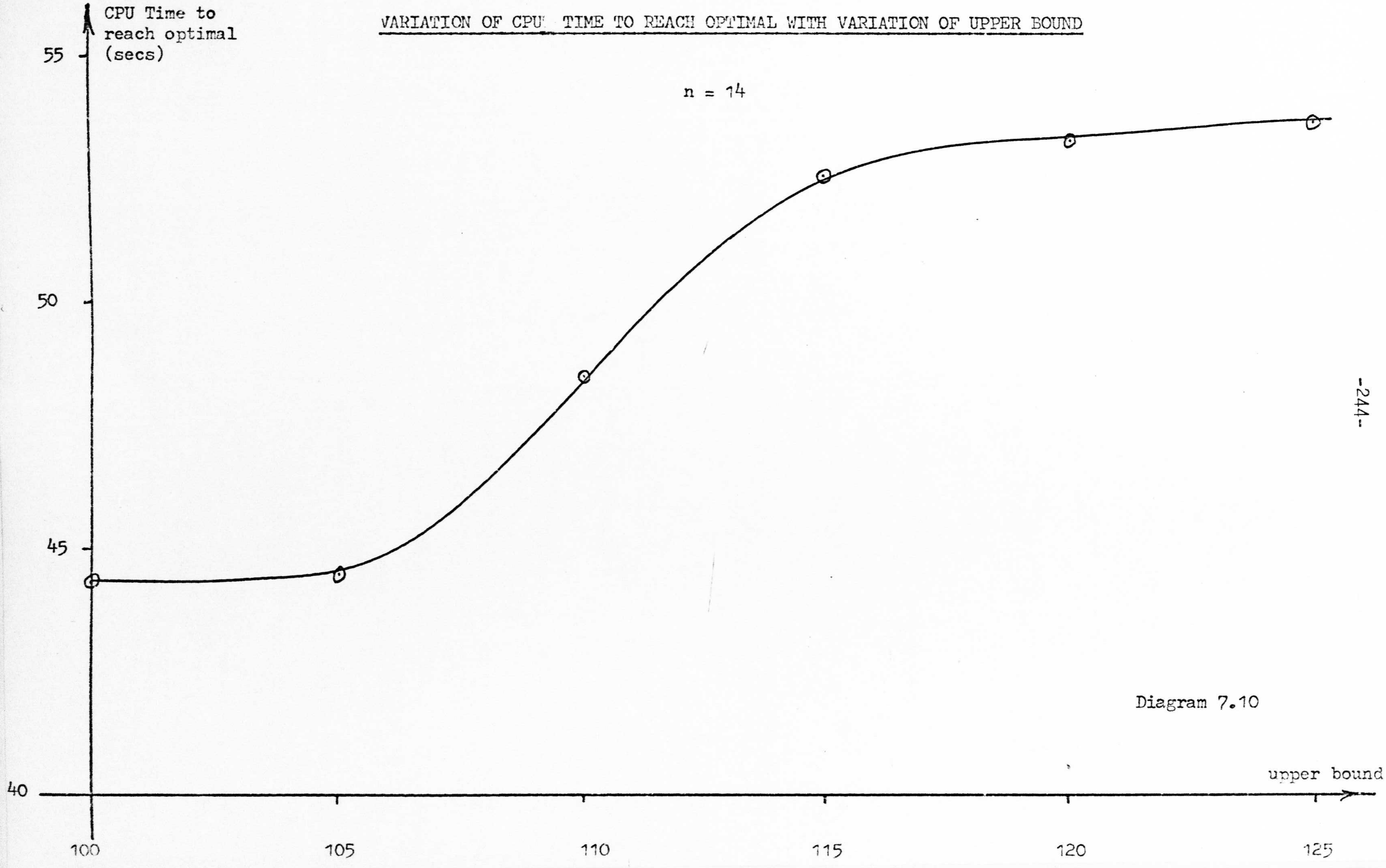


Diagram 7.10

TEXT
CUT OFF IN THE
ORIGINAL

VARIATION OF CPU TIME TO REACH OPTIMAL WITH VARIATION OF UPPER BOUND

n = 16

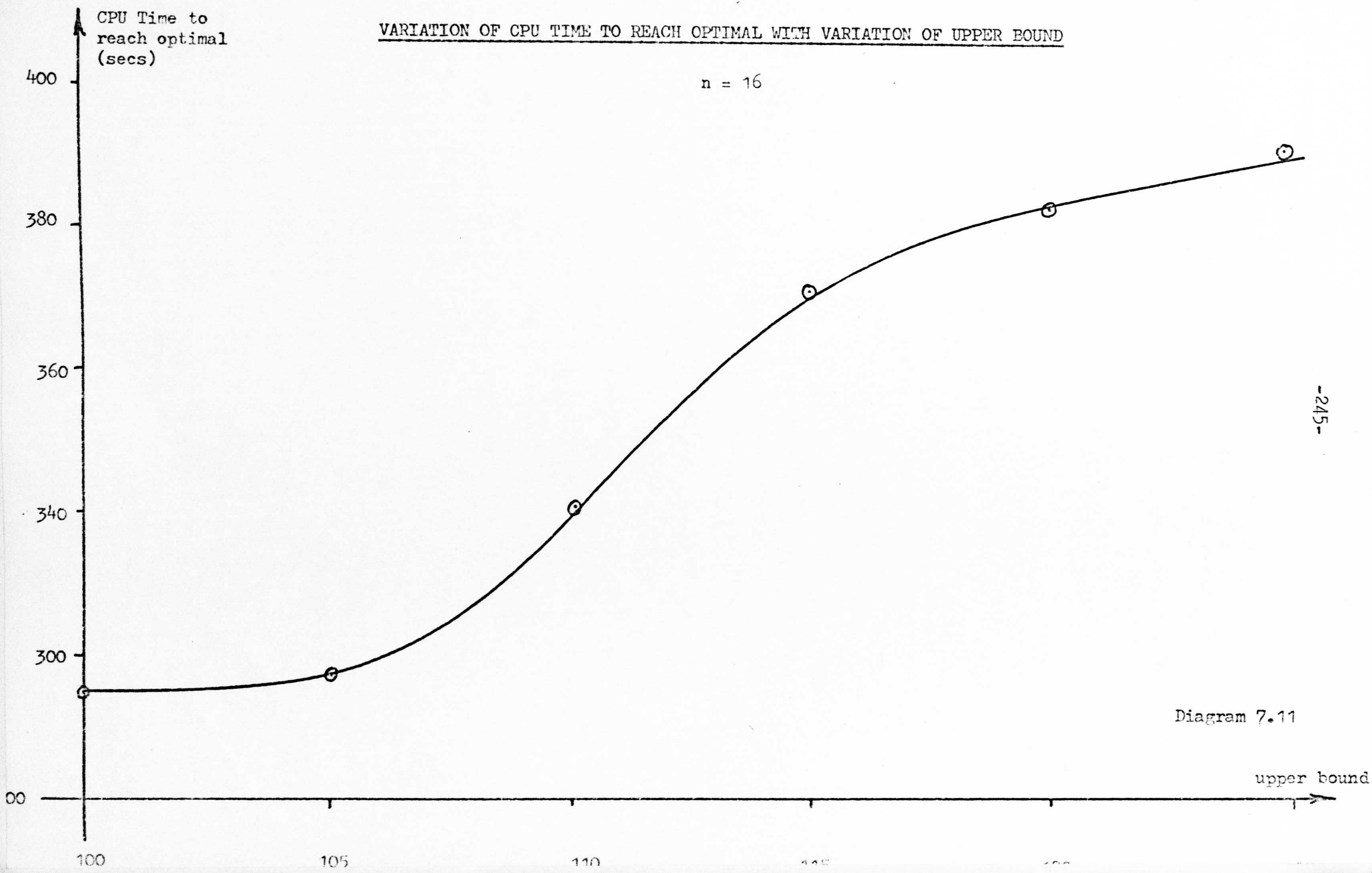


Diagram 7.11

7.6.2. Conclusions.

The results show clearly that as the initial upper bound is reduced the C.P.U. time required to reach an optimal solution is also reduced.

In each of the four cases tested the fall in C.P.U. time as the upper bound was reduced can be divided into three main phases. At the higher values of upper bound the fall in C.P.U. time was fairly small but as the upper bound was reduced further a rapid fall in the C.P.U. time was seen to take place. As the upper bound approached the optimal solution in each-case the fall in C.P.U. time again became fairly small.

It is thus important to have the initial upper bound as small as possible and it is noted that for n values of 14 and 16 the standard upper bound cost lies in the mid portion of each of the graphs, where the slope is greatest. It would of course be better if the standard upper bound cost were to lie on the flatter portion of the graphs where a greater reduction in C.P.U. time is gained.

The aim of any upper bound procedure must obviously be to get as close to the optimal solution as is possible and so maximise the reduction in C.P.U. time. It should be noted that in the standard upper bound algorithm the cost of the initial lower bound will have a great effect on how good the initial upper bound solution is.

It was also noted that the costs obtained by the heuristic method lay on the flatter portion of the graphs where there was little saving in C.P.U. time.

7.7. Standard Upper Bound Algorithms.

It was seen in the last section that it is important to have a good initial upper bound since it is then possible to reduce the C.P.U. time required to reach a good or optimal solution. Of course it is only worthwhile to reduce the upper bound cost consistent with the method itself not being too complex and taking a large amount of C.P.U. time.

It has been seen in an earlier section that the heuristic method of obtaining a feasible solution is very economical of C.P.U. time, taking only 66 seconds for an n value of 400. In this respect it is important to investigate the variation in C.P.U. time required for the standard upper bound procedure with increase in n. It is important also to ascertain the C.P.U. time required at the larger values of n as it would be unwise to spend a great deal of computer time to find an initial upper bound to the problem.

The C.P.U. time taken by the standard upper bound procedure was determined for n values from 4 to 30 and the results are presented graphically in Diagram 7.12.

7.7.1 Results

n	Lower Bound	Upper Bound	CPU time for Upper Bound (secs)
4	52	52	0.02
6	56	56	0.05
8	62	62	0.10
10	87	87	0.18
12	102	102	0.37
14	84	109	1.38
16	82	115	2.09
18	78	109	3.06
20	84	98	3.62
22	93	108	5.55
24	100	141	19
26	94	126	49
28	93	134	174
30	101	148	423

RISE IN CPU TIME OF STANDARD UPPER BOUND ALGORITHM WITH INCREASE OF n

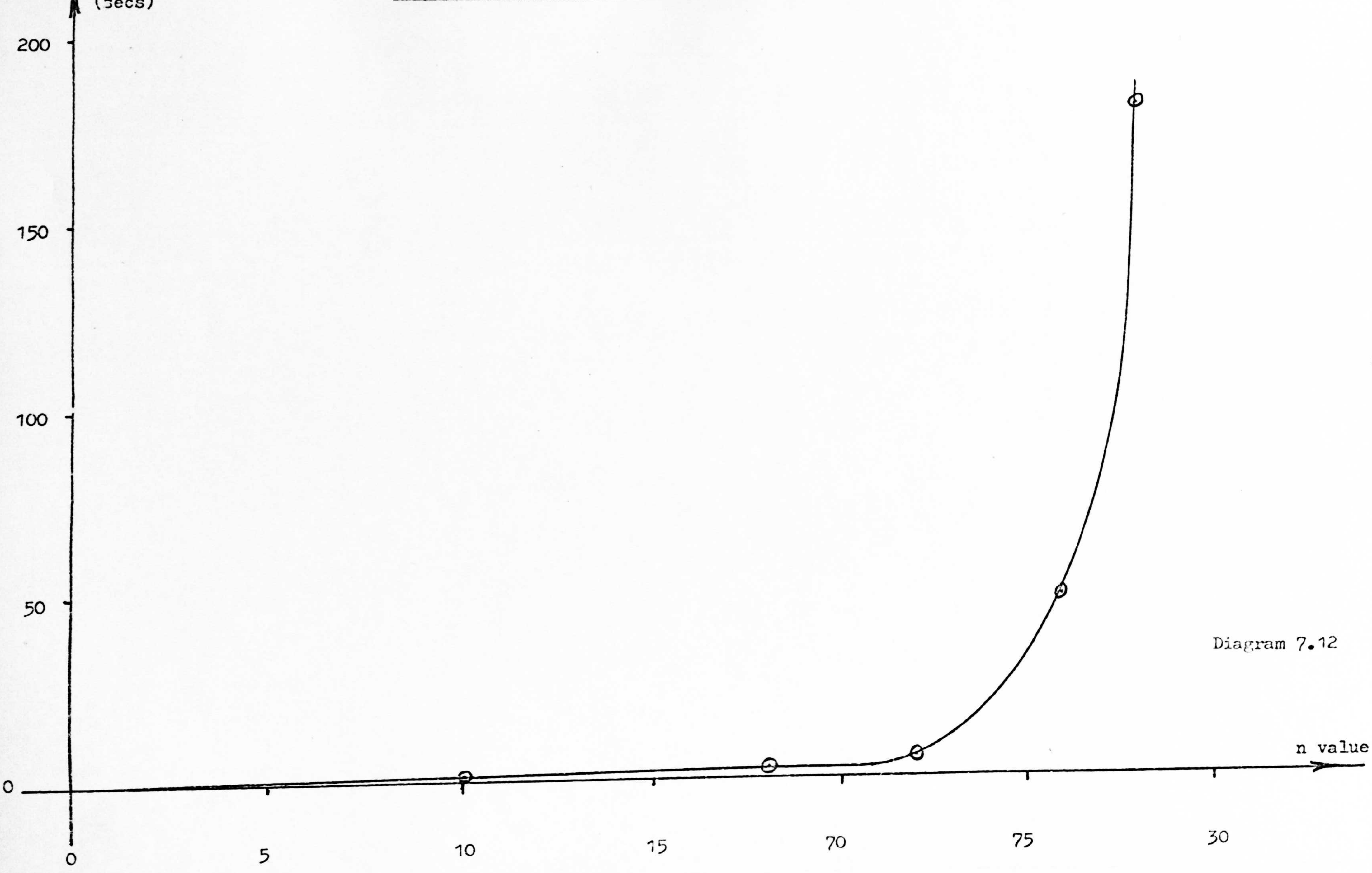


Diagram 7.12

7.7.2 Conclusions.

It can be seen in each of the cases tested the actual difference in cost between the lower and upper bounds is quite low but the cost in C.P.U. time becomes very large at the higher values of n, being 424 seconds at an n value of 30.

The variation in C.P.U. time with n is fairly linear up to an n value of about 22 after which the rise in the C.P.U. time becomes exponential in nature.

It would thus appear that the standard upper bound algorithm should only be used for n values up to about 22 and for higher values of n it would appear the heuristic method would be more beneficial in obtaining a good upper bound in a smaller amount of C.P.U. time.

Although the costs of the heuristic method are not as good as the standard upper bound algorithm their economy of C.P.U. time makes them much more favourable than the standard method for the higher values of n.

7.8. Reduction of Pen Up Distances.

As explained in the previous chapter it is possible to reduce the number of nodes associated with any other node so that the total number of pen up distances to be considered is reduced. If the distance between any two nodes was greater than a given value it was not possible to match these two nodes to give a pen up segment and so the number of feasible solutions to the pen up problem was subsequently reduced.

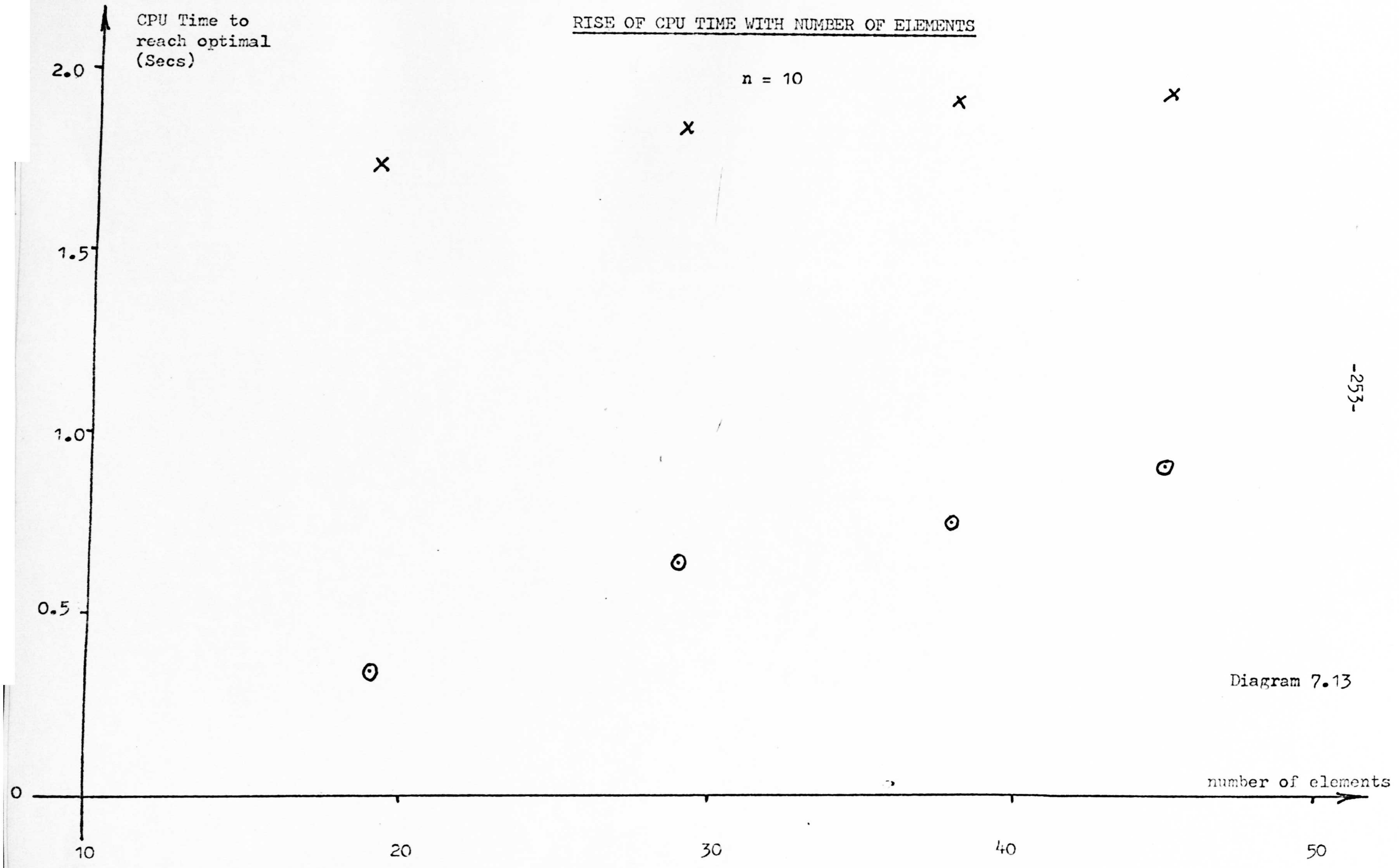
In order to ascertain the variation in C.P.U. time to reach an optimal solution as the number of 'feasible' pen up distances was reduced, timing experiments were carried out with varying numbers of feasible pen up distances for each of the n values tested.

The timing tests were carried out for n values of 10, 12, 14 and 16 for which the maximum number of feasible pen up distances were 45, 66, 91 and 120 respectively. The number of feasible pen up distances for each of the tests was varied. In addition, the pen up distances associated with each node were sorted in increasing order of magnitude. By this means good feasible solutions should be obtained more quickly since the smaller values of pen up distances associated with each of the nodes of the disjoint set P will be obtained first.

Similar timing tests were therefore carried out after sorting the pen up distances associated with each of the nodes.

7.8.1. Results

Number of Odd Nodes	Number of Pen Up Distances	C.P.U. Time $\frac{\text{Secs}}{\text{ns}}$	C.P.U. Time $\frac{\text{Secs}}{\text{s}}$
10	45	0.91	1.94
	38	0.76	1.92
	29	0.64	1.86
	19	0.36	1.71
12	66	2.77	7.88
	56	2.45	7.80
	39	1.60	7.57
	25	0.93	7.35
14	91	55.29	34.12
	74	51.31	34.09
	54	45.03	34.03
	34	37.25	33.98
16	120	380	163
	98	341	162
	71	274	160
	47	206	157



RISE OF CPU TIME WITH NUMBER OF ELEMENTS

Diagram 7.13

TEXT
CUT OFF IN THE
ORIGINAL

RISE OF CPU TIME WITH NUMBER OF ELEMENTS

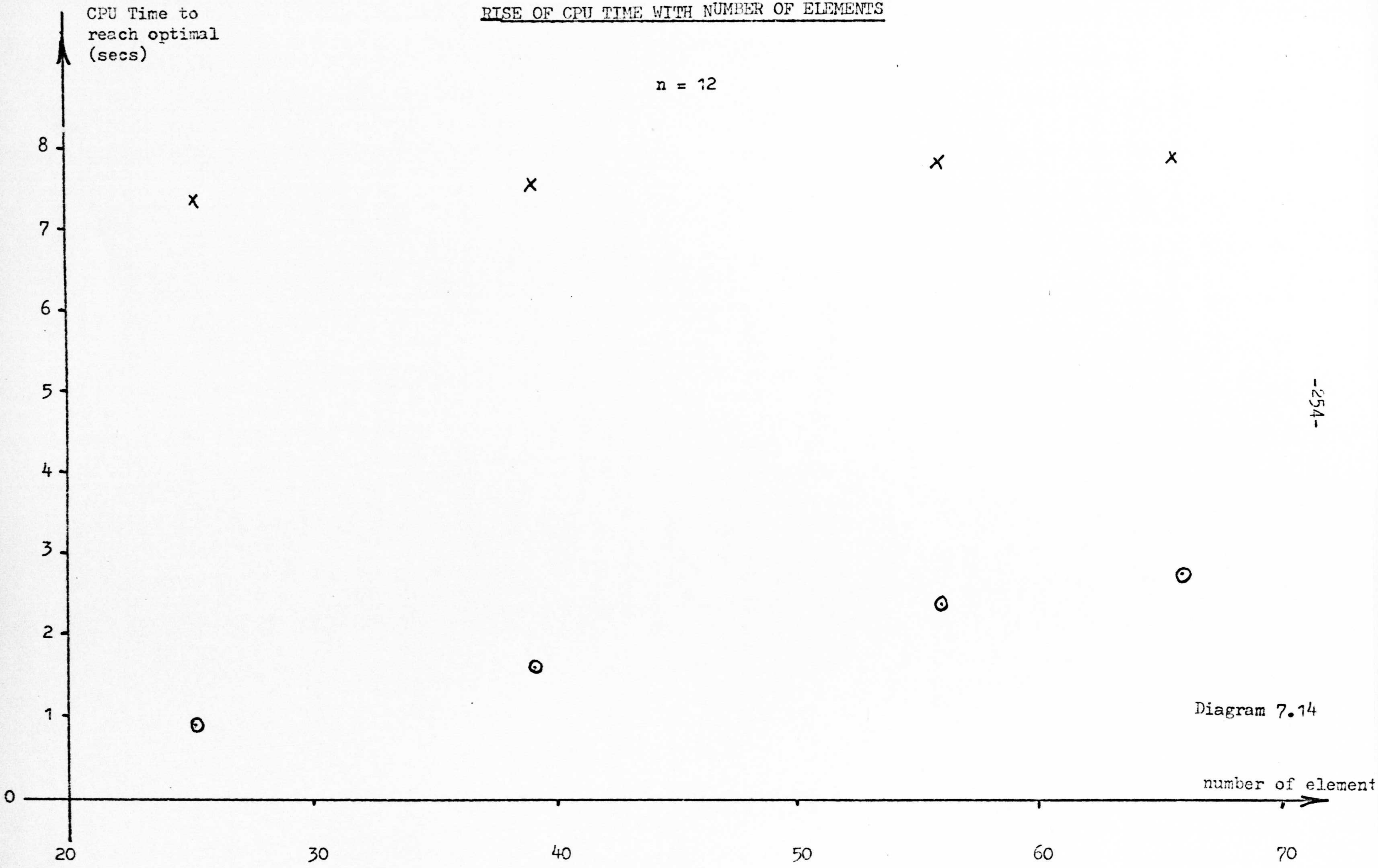


Diagram 7.14

RISE OF CPU TIME WITH NUMBER OF ELEMENTS

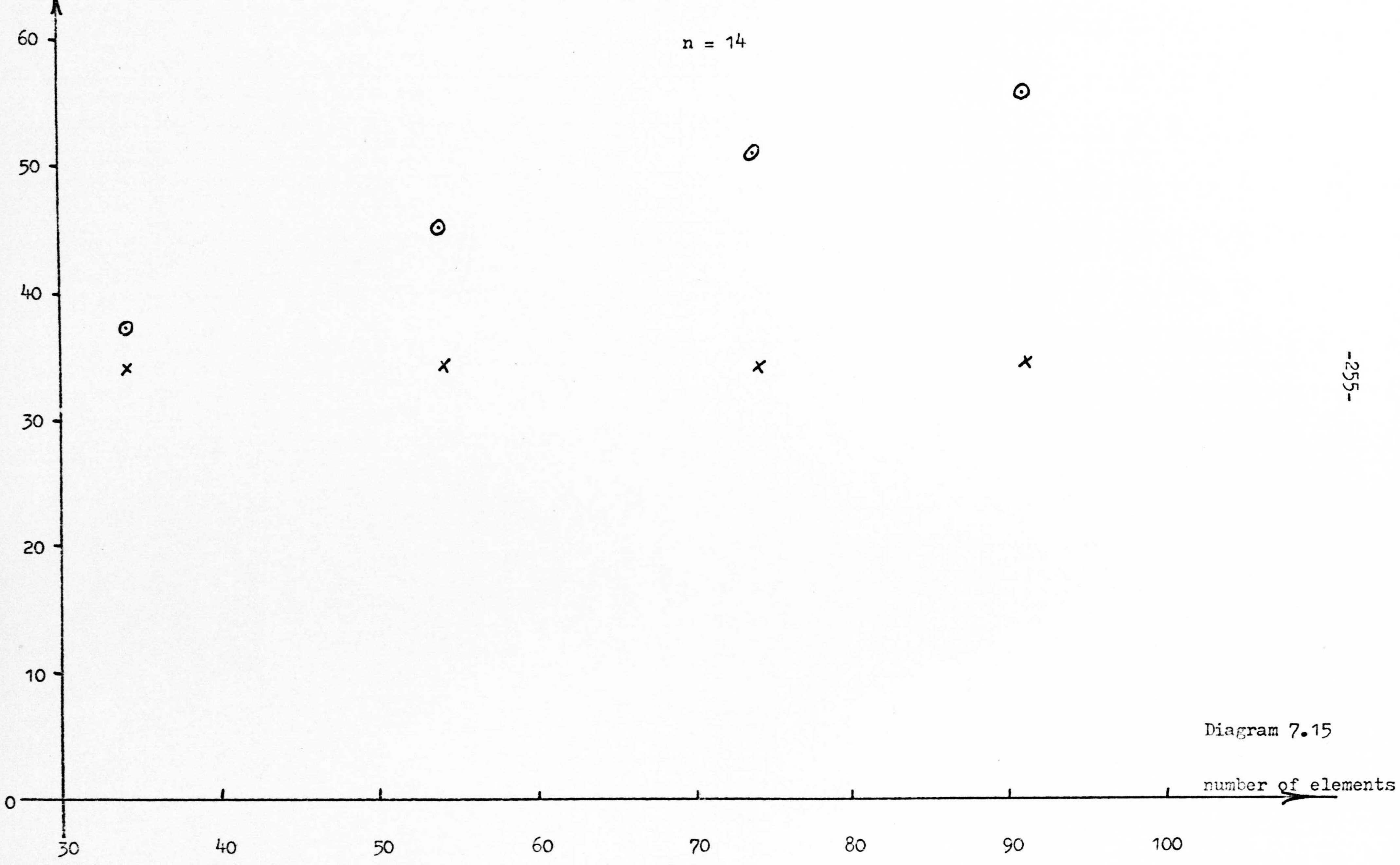


Diagram 7.15

RISE OF CPU TIME WITH NUMBER OF ELEMENTS

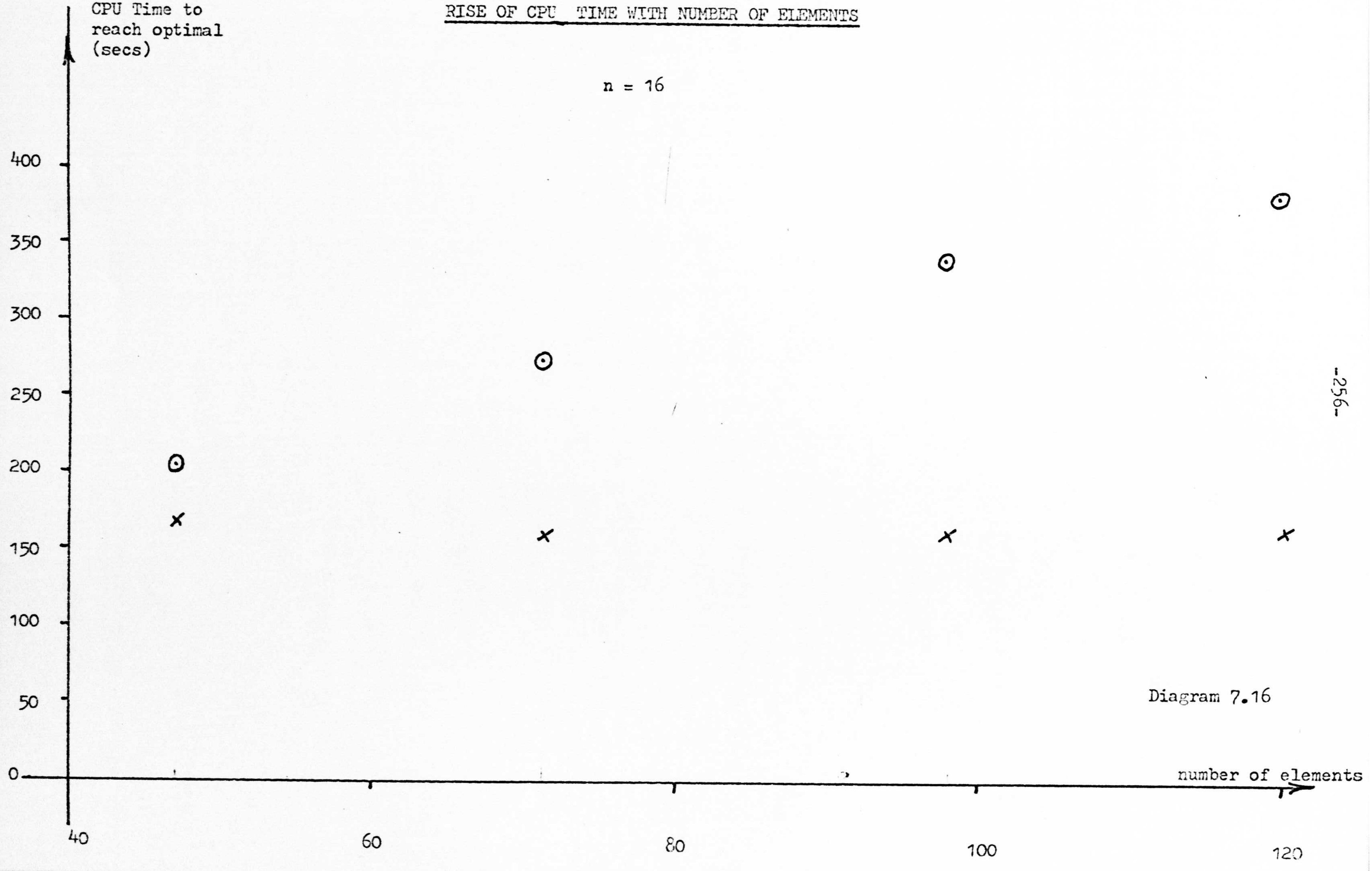


Diagram 7.16

7.8.2. Conclusions.

The reduction in the C.P.U. time required to reach an optimal solution in each case was directly proportional to the number of feasible pen up distances allowed. It is thus of importance to be able to reduce the number of pen up distances to a minimum consistent with there being a feasible solution obtainable.

At the two low values of n (10 and 12) the C.P.U. time required to reach the optimal solutions for the non sort method were better than those for the sort method but as n (and the number of associated pen up distances) was increased the sorting method appeared to be a definite advantage.

It was interesting to note that as the number of pen up distances was reduced for the sort method no great improvement in the C.P.U. times took place and it would thus appear that there is no real need to reduce the pen up distances at all if they are first sorted for each of the n nodes. This is because a good solution is obtained very quickly and there is therefore little advantage to be gained from there being fewer pen up distances present. When no sorting takes place however, good solutions do not tend to be found quickly and as the results show, a greater time saving is obtained with a reduction in the pen up distances.

From the results it would appear that most advantage is gained by simply sorting the pen up distances.

associated with the nodes since the time to reach an optimal solution for n = 16 with sorting and no reduction in the pen up distances is 163 seconds, whereas with a reduction in the pen up distances from 120 to 47 and no sorting the C.P.U. time to reach an optimal solution is 206 seconds. Sorting the pen up distances thus seems to have a greater effect than simply reducing the number of them. This criterion seems to hold for n values greater than 12. For n values lower than this no advantage seems to be gained from sorting the pen up distances and it can thus be said that at these low values of n the reduction in the number of pen up distances has a greater effect than the sorting of them.

It should be noted that a convenient upper bound cost to the pen up distances is provided by the standard upper bound procedure described in the previous chapter. The upper bound corresponds to the cost of the largest pen up segment which is added to make the image graph associated with the feasible solution to the image problem a domino graph.

7.9. Binary Chopping.

The great advantage of binary chopping as mentioned earlier is that good solutions tend to be found in a shorter time than those of the standard algorithm. It would also be expected that the binary chopping scheme would be most advantageous in the initial period of the enumeration.

In order to compare the performance of the binary chopping scheme with the standard scheme, timing tests for both n = 14 and n = 16 have been carried out. The time taken in each case to iterate to the next solution has been obtained and the results of these tests are presented in graphical form in Diagrams 7.17 and 7.18.

So that it was possible to have a number of iterations before reaching the best solutions the initial upper bounds were made fairly high.

7.9.1. Results

n = 14

Standard	
Solution Obtained	CPU Time secs.
349	0
229	0.06
178	0.16
167	0.29
162	0.44
151	0.61
143	0.74
141	4.36
130	4.73
130	4.73
126	4.84
113	6.74
96	16,91
95	17.91

Total CPU Time 58. secs

Binary Chopping	
Solution Obtained	CPU Time secs.
349	0
178	0.06
130	1.85
96	6.51

Total CPU time 44 secs

n = 16

Standard	
Solution Obtained	CPU Secs
357	0
337	0.08
324	0.21
231	0.32
230	0.49
226	0.54
179	0.79
146	0.94
127	1.25
122	1.97
118	8.18
115	8.34
114	22.53
105	87.18
98	100.46

Total CPU Time 371 secs.

Binary Chopping	
Solution Obtained	CPU Secs
357	0
179	0.2
127	0.46
105	28.83

Total CPU Time 256 secs.

COMPARISON OF BINARY CHOPPING SCHEME WITH STANDARD IMPLICIT ENUMERATION ALGORITHM (FIRST 20 SECONDS OF ENUMERATION

n = 14

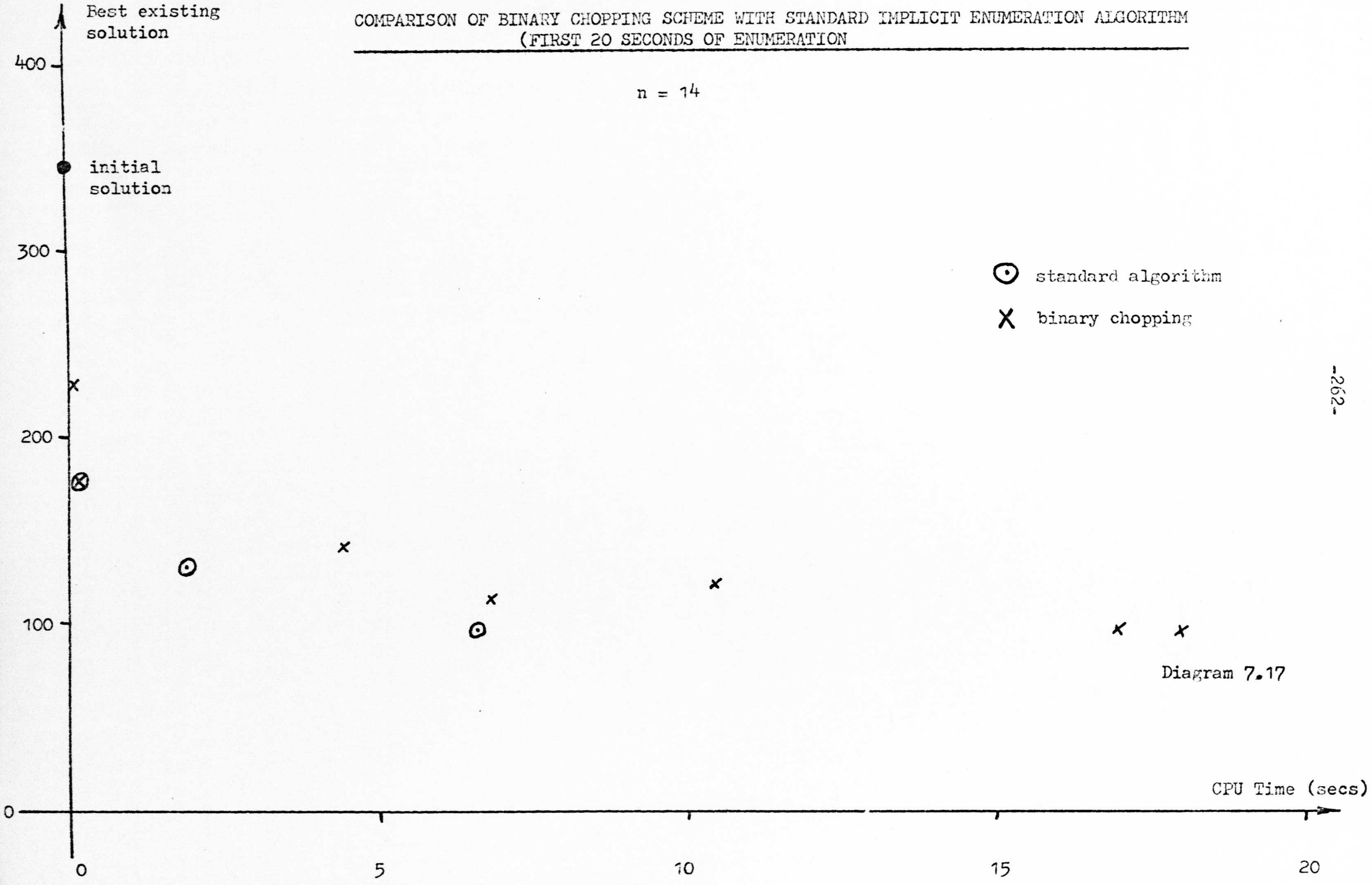


Diagram 7.17

COMPARISON OF BINARY CHOPPING SCHEME WITH STANDARD IMPLICIT ENUMERATION ALGORITHM (FIRST 30 SECONDS OF ENUMERATION).

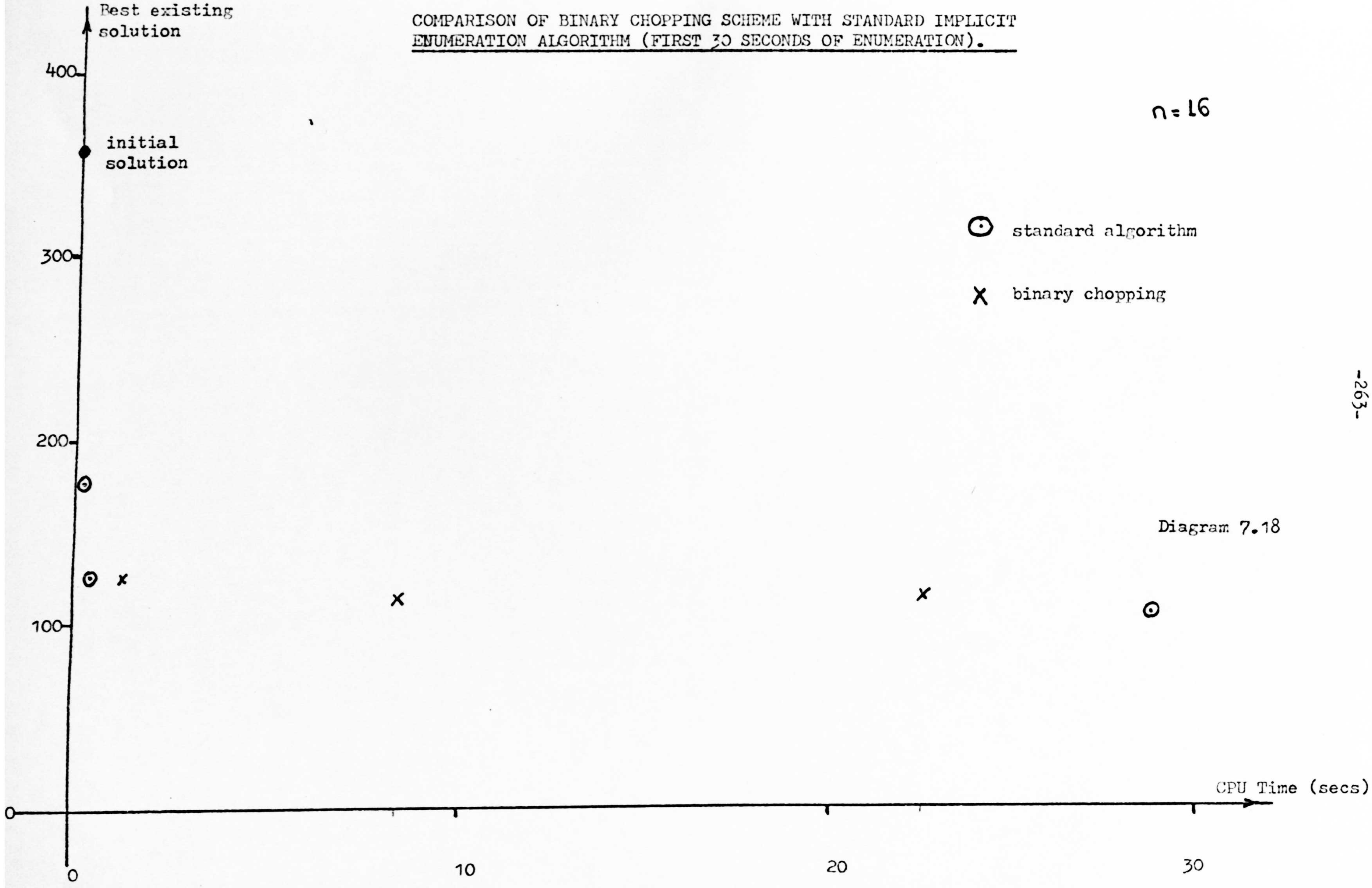


Diagram 7.18

7.9.2. Conclusions.

It can be clearly seen from the results that binary chopping obtains good solutions much more quickly than the standard algorithm. The total time taken in each case by the chopping algorithm is less than the standard, being 44 seconds for n = 14 and 256 seconds for n = 16. This compares with the performance of the standard algorithm of 58 seconds for n = 14 and 371 seconds for n = 16. The best solution found by binary chopping was only one unit greater than the optimal cost for an n value of 14 and this was found in 6.5 seconds compared with 16.9 seconds for the standard. For an n value of 16 the best solution found by binary chopping was 105 compared with the optimal of 98. This was found in 28.8 seconds compared with 87 seconds by the standard method.

It is of considerable interest to note the CPU times taken after the best solutions have been found. For n = 14 the standard takes over 40 seconds more in completing the search <u>after</u> the optimal solution has been found. This is o ver 60% of the total time taken. The binary chopping method takes another 38 seconds after finding its best solution which it obtains in just over 6 seconds. This is therefore about 80% of the total time taken.

For n = 16 the standard algorithm continues for another 270 seconds before terminating which accounts for over 70% of the total CPU time while the binary chopping scheme takes over 220 seconds after finding its best solution which it obtained in 28 seconds. This is well over 80% of the total time taken. It is only to be expected that binary chopping will tend to have a large CPU time waste at the end of the enumeration since it may well be looking for a solution which does not exist. It also of course tends to obtain the best solution fairly early in the unumeration. The CPU time expended in trying to make improvements when the best solutions have been found are nevertheless extremely high in both cases and show the importance of having good lower bounds.

The graphs show how the iteration proceeds for the two different methods. Note that each successive solution (marked by points) will be below and to the right of the solution immediately before it. As expected there are fewer iterations in the binary chopping scheme than the standard. The number of iterations for binary chopping in the two cases is only of the order of 25% of those for the standard scheme.

The results show that for n = 14 a feasible solution whose cost is 130 is obtained by binary chopping in 1.85 seconds while this takes 4.7 seconds to obtain by the standard method.

NOTE.

The method of obtaining an optimal solution to the pen up problem by the binary chopping backtrack technique explained in the previous chapter was tested to ascertain the time taken to obtain an optimal solution.

The CPU time taken was in every case found to be extremely high since for each backtracking iteration with the newly determined upper and lower bounds, the CPU times of the iterations decreased very slowly so that if ℓ iterations were necessary the total CPU time taken was of the order of ℓ times that of the first iteration.

This was extremely high and it was concluded that although an optimal solution was obtained in each case,the method could not be considered to be satisfactory.

7.10. Sets of Sub Problems.

It was explained in the previous chapter that it is possible to obtain a lower bound cost to the pen up problem under the restriction that a number of elements of the complete set S are not allowed to be paired together. This number of elements was referred to as the subset grouping depth, sgd.

When sgd elements of the set P had been obtained a lower bound cost was calculated as described in the previous chapter. If this cost was not less than the best existing solution to the pen up problem it was possible to skip a number of sub problems. These sub problems would all correspond to having the sgd elements contained in set P.

By this means it may be possible to improve the performance of the general implicit enumeration algorithm since a number of sub problems (taken together) could be skipped.

C.P.U. timing tests were carried out for n values of 12, 14 and 16 so as to ascertain the improvement (if any) in the C.P.U. time to reach an optimal solution to the pen up problem.

It was also of considerable interest to vary the value of sgd for each of the n values tested. The value of sgd was varied from 2 to $\frac{n}{2}$ for each of the n values tested. For low values of sgd, where only a small number of elements were not allowed to be paired together there was less chance of skipping a number of sub problems. If it was possible however, then the number of sub problems skipped would be large.

On the other hand, for large values of sgd, there would be a better chance of being able to skip a number of sub problems but in this case the number of sub-problems would be fairly small.

The variation in C.P.U. time to reach an optimal solution with variation in sgd is shown in Diagrams 7.19, 7.20 and 7.21, corresponding to n values of 12, 14 and 16 respectively.

7.10.1. Results.

n = 12

sgd	CPU Time to reach optimal. secs.
2	10.5
3	10.1
4	9.9
5	11.2
6	10.5

n = 14

sgd	CPU Time to reach optimal. secs.
2	60.3
3	55.8
4	53.5
5	53.5
6	61.9
7	59.8

n = 16

sgd	CPU Time to reach optimal. secs.
2	378
3	383
4	399
5	410
6	427
7	422
8	391

VARIATION IN CPU TIME TO REACH OPTIMAL WITH VARIATION IN THE SUB SET GROUPING DEPTH (sgd)

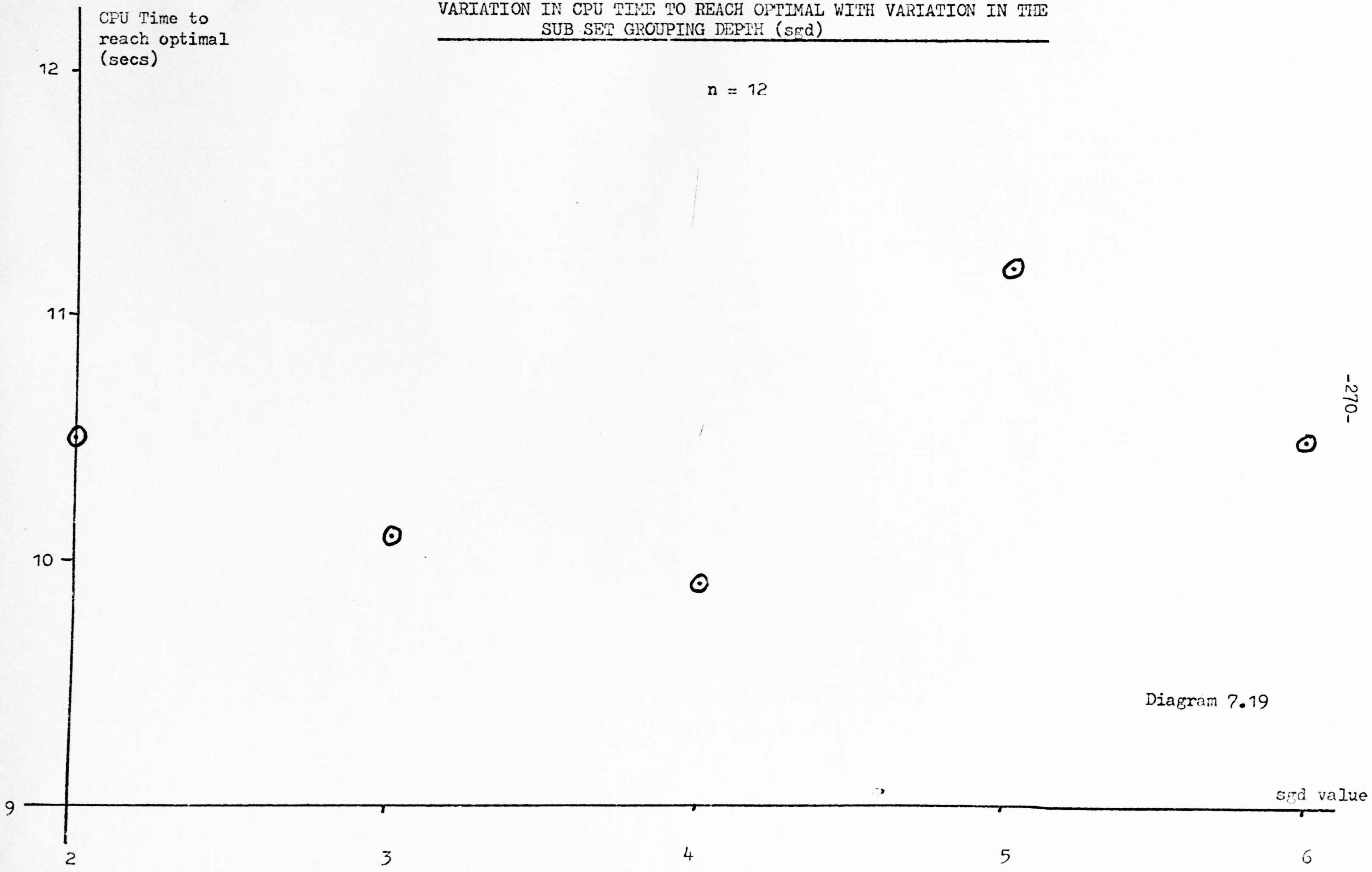
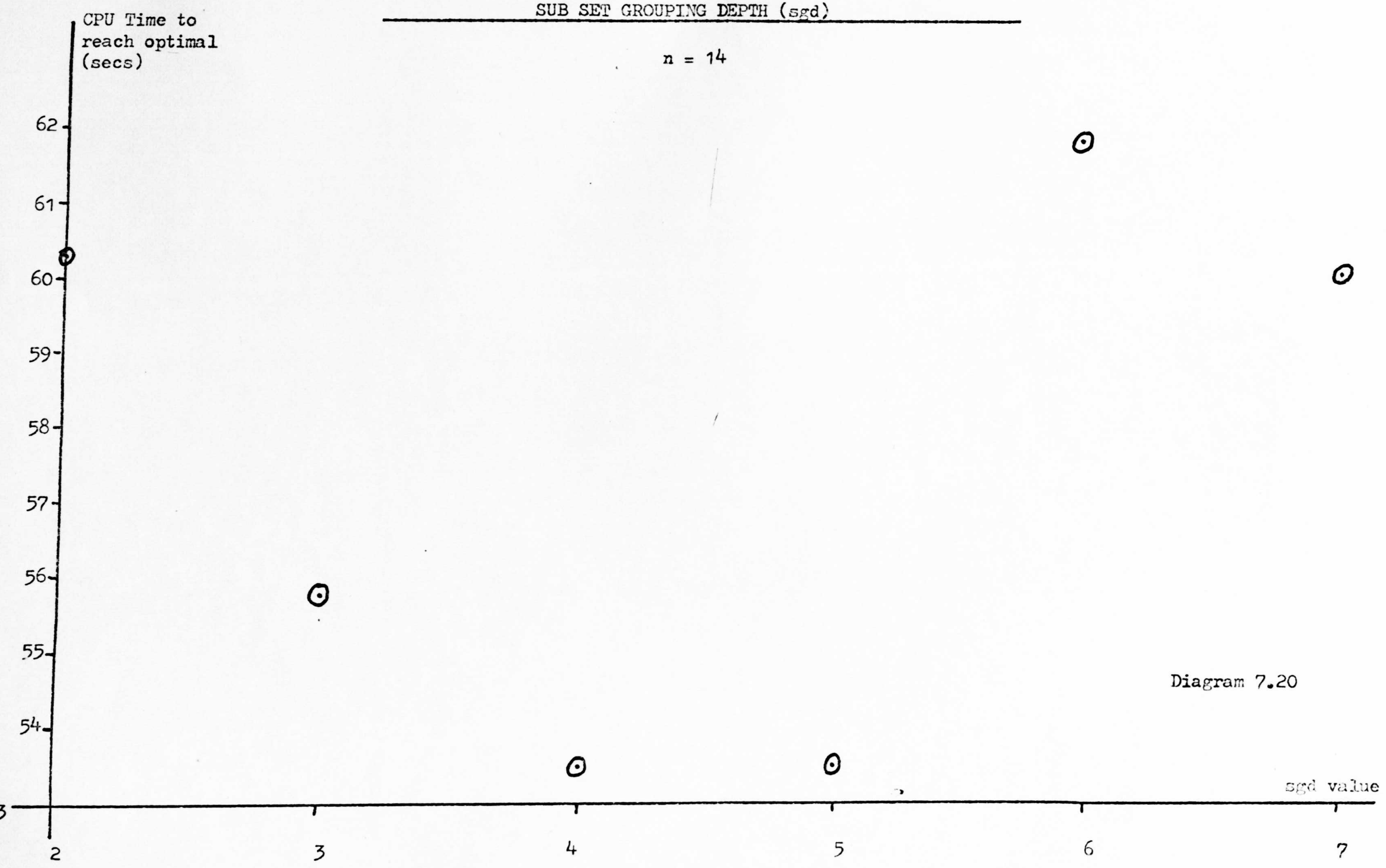


Diagram 7.19

VARIATION IN CPU TIME TO REACH OPTIMAL WITH VARIATION IN THE SUB SET GROUPING DEPTH (sgd)



n = 14

Diagram 7.20

VARIATION IN CPU TIME TO REACH OPTIMAL WITH VARIATION IN THE SUB SET GROUPING DEPTH (sgd)

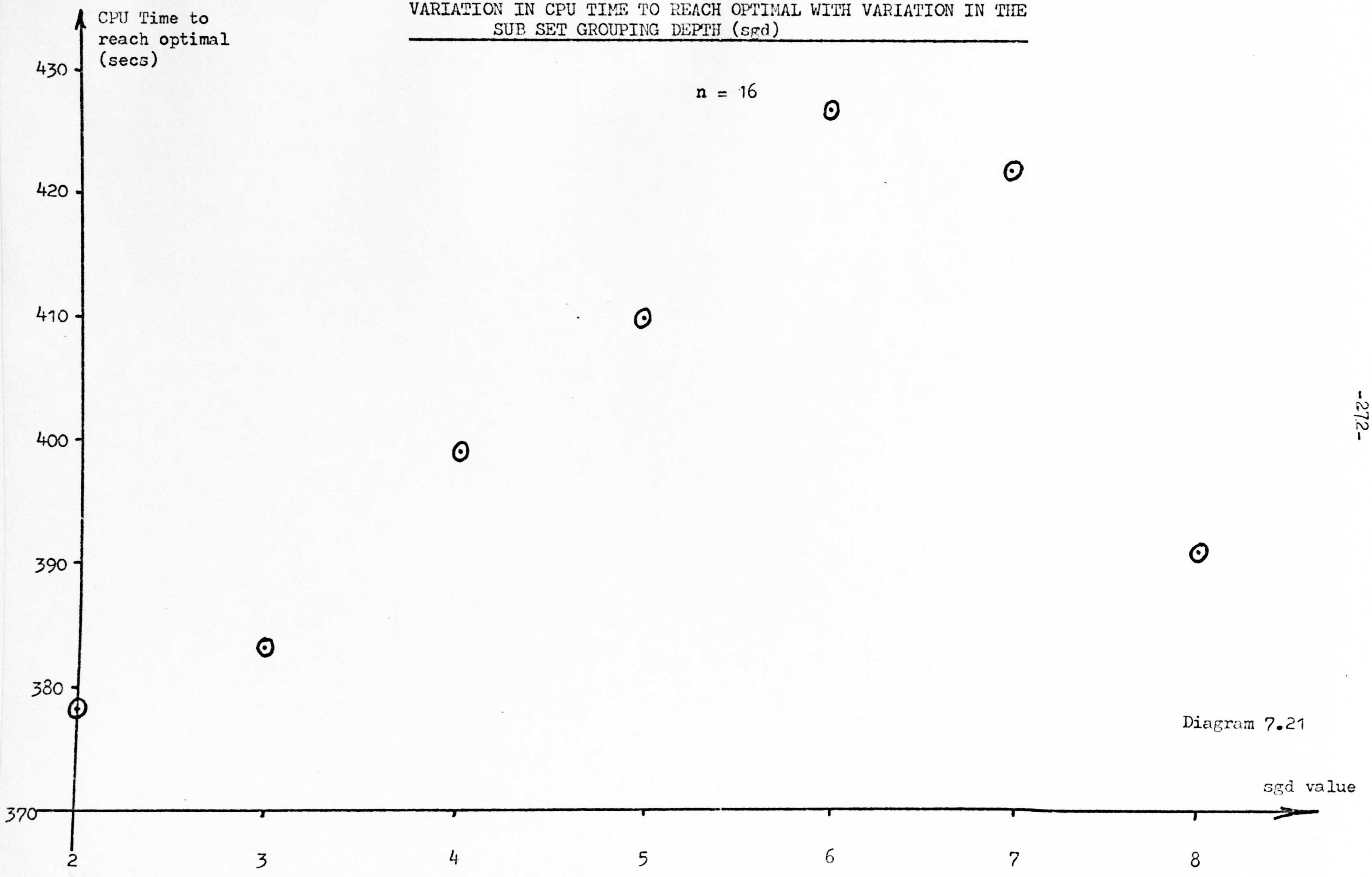


Diagram 7.21

7.10.2. Conclusions.

It can be seen that in each of the cases tested, a reduction in the C.P.U. time to reach an optimal solution was obtained, but only for certain values of the subset grouping depth.

The maximum reduction in C.P.U. time varied from 3% for an n value of 16 to 10% for an n value of 14.

One of the significant results arising from the tests was the large variation in C.P.U. time with variation in the value of the subset grouping depth. In general, as sgd was decreased from $\frac{n}{2}$, thh C.P.U. times increased slightly and then decreased with further decrease of sgd. In the cases of n=12 and n=14, however, the C.P.U. times were then seen to increase with further decrease in the sgd value. This was the expected trend (discussed in 6.16) since at sgd values of about n/4, maximum benefit would probably be gained between skipping a large number of sets of sub problems of low cardinality and a small number of sets of sub problems of high cardinality.

For n=12 the maximum reduction in C.P.U. time corresponded to an sgd value of 4 and for n=14 the maximum reduction in C.P.U. time corresponded to an sgd value of both 4 and 5.

For n=16 it was seen that the C.P.U. times continued to fall (after the initial rise which also occurred for n=12 and n=14) as the value of sgd was decreased. In this case the maximum decrease corresponded to an sgd value of 2.

7.11. Summary of Conclusions.

It is perhaps useful at this stage to give a short summary of the more important conclusions which have been reached for the C.P.U. timing experiments carried out.

The rise in the C.P.U. time taken to reach an optimal solution for the standard implicit enumeration algorithm was seen to be of exponential form as n was increased. At an n value of 18 the C.P.U. time required to reach an optimal solution was of the order of 1160 seconds compared with a corresponding value of about 45 seconds for an n value of 14.

The direct solution of each of the sub problems of the pen up problem (which are simply assignment problems of order $\frac{n}{2}$) was perhaps the method by which the standard implicit enumeration scheme was improved most. The C.P.U. time taken to reach an optimal solution in this case was of the order of half that taken by the standard algorithm in each of the cases tested.

The reduction in the number of feasible pen up distances to the pen up problem was seen to reduce the C.P.U. time taken to reach an optimal solution in an approximately linear manner. An added improvement occurred when the sizes of the distances associated with each of the nodes were sorted in ascending order of magnitude. In this case it was found that the effect of the reduction in the number of pen up distances was diminished and when sorting of the distances was carried out it was concluded that little extra improvement was gained by also reducing the number of pen up distances.

In this case the costs of the best solutions obtained did not always equal the optimal costs obtained by the standard implicit enumeration algorithm. These costs were however, very close to the optimal costs.

It was shown that the cost of the initial upper bound solution to the pen up problem has indeed a distinct effect on the CPU time taken to reach an optimal solution. The C.P.U. time was seen to decrease as the initial upper bound cost was reduced. The timing tests carried out on the standard upper bound algorithm showed that the C.P.U. time taken to calculate an upper bound cost increased exponentially with n and it was concluded that for n values greater than about 22 the heuristic method of obtaining an initial feasible solution (which could be used as the initial upper bound cost) was preferred.

The importance of obtaining a good lower bound cost was shown by noting that about 70% of the total C.P.U. time taken to reach an optimal solution using the standard algorithm was expended <u>after</u> the optimal cost had been reached. If the lower bound cost was in fact equal to the optimal cost (for a given n value) then it would have been possible to terminate the enumeration.

By grouping sets of sub problems it was seen that an improvement of between 3 and 10% (for the n values tested) on the standard algorithm was obtained. It was also noted that the C.P.U. times varied substantially with variation in the value of the sub set grouping depth defined in 7.11.

The C.P.U. timing experiments carried out on the heuristic algorithm showed that the C.P.U. time required for very large values of n was reasonably small and only 66 seconds of C.P.U. time was required to solve a pen up problem of order 400.

The binary chopping scheme was seen to obtain good feasible solutions to the pen up problem much more quickly than the standard algorithm although in this case the C.P.U. time taken *after* the best solution had been found tended to be very high for each of the cases tested. The best solution obtained in this case did not always equal the optimal cost obtained by the standard implicit enumeration scheme.

APPENDIX 1.

Program Listings.

The following are program listings relating to the work discussed in Chapter 1.

Procedure 'line' is the procedure used to produce the line drawings shown in this thesis.

Linedot is a similar procedure used to draw the dotted lines.

Also included in this appendix is a sort procedure which is the standard sort procedure used in the programs of Chapters 3, 4 and 7.

```
'PROCEDURE' LINE(XZ,YZ,XN,YN);
'VALUE' XZ,YZ,XN,YN; 'INTEGER' XZ,YZ,XN,YN;
'COMMENT' PROCEDURE PLOTS A LINE BY AN ORDERED SEQUENCE OF INCREMENTAL MOVES
SUCH THAT EACH MOVE IS AT LEAST AS GOOD AS ANY OTHER MOVE UNDER THE
EXISTING CONDITIONS;
'BEGIN' 'INTEGER' D,S,X,Y,W,V,G,B,DB,J; 'BOOLEAN' BB,C,K,E;
X:=XN-XZ; Y:=YN-YZ;
'COMMENT' CHECK TO SEE IF ANY MOVES TO BE MADE;
'IF' X=0 'AND' Y=0 'THEN' 'GOTO' FINISH;
BB:=X>0; C:=Y>0; K:=ABS(X)-ABS(Y)>0;
'COMMENT' NEXT 4 LINES OF CODE DETERMINE THE STRAIGHT AND DIAGONAL MOVES
TO BE MADE;
'IF' BB 'THEN' 'BEGIN' 'IF' C 'THEN' D:=2 'ELSE' D:=4 'END'
        'ELSE' 'IF' C 'THEN' D:=8 'ELSE' D:=6;
'IF' K 'THEN' 'BEGIN' 'IF' BB 'THEN' S:=3 'ELSE' S:=7 'END'
       'ELSE' 'IF' C 'THEN' S:=1 'ELSE' S:=5;
W:=ABS(ABS(X)-ABS(Y));
'IF' ABS(X)<ABS(Y) 'THEN' 'BEGIN' V:=ABS(X); G:=ABS(Y) 'END' 'ELSE'
                          'BEGIN' V:=ABS(Y); G:=ABS(X) 'END';
'COMMENT' G IS THE TOTAL NUMBER OF MOVES TO BE MADE, B IS PROPORTIONAL TO
THE PERPENDICULAR DISTANCE OF THE PLOTTED LINE FROM THE ACTUAL LINE AND DB
IS THE CHANGE IN B FOR A GIVEN MOVE;
B:=V-W; DB:=0;
E:='TRUE'; W:=-W-W; V:=V+V;
'COMMENT' DB=W FOR A DIAGONAL MOVE. DB=V FOR STRAIGHT MOVE;
R:  B:=B+DB; E:='NOT' E;
```

THIS IS THE REMAINDER OF THE LINE PROCEDURE.

```
'COMMENT' B>0 IS CONDITION FOR DIAGONAL MOVE. B<0 IS CONDITION FOR
STRAIGHT MOVE AND B=0 IS KNIGHTS MOVE CONDITION;
'IF' B>0 'THEN' 'BEGIN'  X: DB:=W;
                   'IF' E 'THEN' PLOT(J,D) 'ELSE' J:=D
                'END' 'ELSE'
'IF' B<0 'THEN' 'BEGIN'    Y:   DB:=V;
                   'IF' E 'THEN' PLOT(J,S) 'ELSE' J:=S
                'END' 'ELSE' 'IF' BB 'THEN' 'GOTO' X 'ELSE' 'GOTO' Y;
G:=G-1;
'COMMENT' REDUCE THE NUMBER OF MOVES TO BE MADE BY 1 AND TEST TO SEE IF
ANY MORE MOVES NEEDED;
'IF' G>0 'THEN' 'GOTO' R;
'COMMENT' CHECK IF A MOVE HAS BEEN STORED WITHOUT BEING PLOTTED.
IF SO PLOT THIS MOVE;
'IF' E 'THEN' 'ELSE' PLOT(J,0);
FINISH:
'END'LINE;
```

PROCEDURE USED TO PLOT A DOTTED LINE.

```
'PROCEDURE' LINEDOT(XZ,YZ,XN,YN,A);
'VALUE' XZ,YZ,XN,YN,A; 'INTEGER' XZ,YZ,XN,YN,A;
'COMMENT' PROCEDURE PLOTS A DOTTED LINE. THE LENGTH OF DOT IS 0.1*A WITH
RESTRICTION THAT 'A' IS EVEN. PEN STARTS AND FINISHES IN DOWN POSITION;
'BEGIN' 'INTEGER' D,S,COUNT,Y,W,V,P,G,B,DB,J; 'BOOLEAN' B,C,K,E;
X:=XN-XZ; Y:=YN-YZ;
'COMMENT' CHECK IF ANY MOVES TO BE MADE;
'IF' X=0 'AND' Y=0 'THEN' 'GOTO' FINISH;
B:=X>0;  C:=Y>0;  K:=ABS(X)-ABS(Y)>0;
'IF' B 'THEN' 'BEGIN' 'IF' C 'THEN' D:=2 'ELSE' D:=4 'END'
                'ELSE' 'IF' C 'THEN' D:=8 'ELSE' D:=6;
'IF' K 'THEN' 'BEGIN' 'IF' B 'THEN' S:=3 'ELSE' S:=7 'END'
                'ELSE' 'IF' C 'THEN' S:=1 'ELSE' S:=5;
W:=ABS(ABS(X)-ABS(Y));
P:=10; COUNT:=0;
'COMMENT' PLACE PEN DOWN AT START;
PLOT(0,10);
'IF' ABS(X)<ABS(Y) 'THEN' 'BEGIN' V:=ABS(X); G:=ABS(Y) 'END' 'ELSE'
                          'BEGIN' V:=ABS(Y); G:=ABS(X); 'END';
'COMMENT' SET UP INITIAL VARIABLE VALUES;
B:=V-W; DB:=0; E:='TRUE';
W:=-W-W; V:=V+V;
R:     B:=B+DB;  E:='NOT' E;
```

REMAINDER OF LINEDOT PROCEDURE

```
'IF' B>0 'THEN' 'BEGIN'  X:  DB:=W;
                   'IF' E 'THEN' 'BEGIN'     PLOT(J,D);
                                      COUNT:=COUNT+2
                              'END' 'ELSE' J:=D
                'END' 'ELSE'
'IF' B<0 'THEN' 'BEGIN'  Y:   DB:=V;
                   'IF' E 'THEN' 'BEGIN' PLOT(J,S);
                                      COUNT:=COUNT+2
                                 'END' 'ELSE' J:=S
                'END' 'ELSE' 'IF' B 'THEN' 'GOTO' X 'ELSE' 'GOTO' Y;
'COMMENT' REDUCE MOVES TO BE MADE BY 1;
G:=G-1;
'COMMENT' CHECK IF TOTAL MOVES MADE WITH PEN IN GIVEN STATE = A;
'IF' COUNT=A 'THEN' 'BEGIN' P:=19-P; COUNT:=0; PLOT(0,P) 'END';
'IF' G>0 'THEN' 'GOTO' R;
'IF' E 'THEN' 'ELSE' PLOT(J,0);
FINISH:
'COMMENT' FINISH WITH PEN DOWN;
PLOT(0,10);
'END'LINEDOT;
```

NOTE: PLOT(0,9) is pen up.

PLOT(0,10) is pen down.

THE FOLLOWING PROCEDURE 'SORT' IS THE STANDARD SORT PROCEDURE USED IN ALL OF THE PROBLEM PROGRAMS PRESENTED IN THIS THESIS.

```
'PROCEDURE' SORT(B,M,N);
'VALUE' N; 'INTEGER' N ; 'ARRAY' M; 'INTEGER' 'ARRAY' B;
'COMMENT' PROCEDURE SORTS ELEMENTS OF M INTO ASCENDING ORDER OF MAGNITUDE
BY A SIMPLE EXCHANGE METHOD.
'BEGIN' 'INTEGER' A,C,J,I; 'REAL' MIN,CC;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' B(/I/):=I;
'FOR' A:=1 'STEP' 1 'UNTIL' N-1 'DO'
'BEGIN' MIN:=M(/A/); J:=A;
    'FOR' I:=A+1 'STEP' 1 'UNTIL' N 'DO' 'IF' M(/I/)<MIN 'THEN'
    'BEGIN' MIN:=M(/I/); J:=I 'END';
'COMMENT' FROM ELEMENT A TO ELEMENT N FIND THE LEAST VALUE OF M AND
INSERT THIS INTO THE LOCATION M(/A/). ALSO PLACE THE CORRESPONDING ELEMENT
J INTO THE LOCATION B(/A/). AFTER N-1 PASSES,M WILL CONTAIN ENTRIES IN
ASCENDING ORDER OF MAGNITUDE AND B WILL CONTAIN THE CORRESPONDING ELEMENTS
THE FOLLOWING STATEMENTS SWAP ELEMENTS A AND J IN THE ARRAYS M AND B;
CC:=M(/J/); M(/J/):=M(/A/); M(/A/):=CC;
C:=B(/J/); B(/J/):=B(/A/); B(/A/):=C
'END'
'END'SORT;
```

APPENDIX 2

Program Listings.

Procedure 'project' is the procedure used to obtain the projected points on the plane of projection given the n spatial vertices and the angles of the line of sight. This procedure is used extensively in Chapter 3 and Chapter 4.

The wire frame drawings shown in Chapter 2 were obtained by simply projecting the vertices of the given figures onto the plane of projection. The segments corresponding to the edges in space were then connected up.

THIS IS A LISTING OF PROCEDURE PROJECT WHICH IS USED TO OBTAIN THE PROJECTED POINTS GIVEN THE SPATIAL VERTICES.

```
'PROCEDURE' PROJECT(P,Q,R,X,Y,N,D,A,B,G);
'VALUE' N,D,A,B,G; 'REAL' D,A,B,G; 'INTEGER' N;
'ARRAY' P,Q,R,X,Y;
'COMMENT'  P,Q,R ARE THE ARRAYS CONTAINING THE N TRANSLATED SPATIAL
VERTICES. A,B,G ARE ANGLES MADE BY LINE OF SIGHT WITH X,Y AND Z
AXES. (CARTESIAN COORDINATES) RESPECTIVELY. D IS DISTANCE OF
PROJECTION PLANE AND X,Y ARE THE ARRAYS TO CONTAIN THE PROJECTED POINTS;
'BEGIN' 'INTEGER' I; 'REAL' K,A11,A12,A21,A22,A23;
A11:=COS(B)*COS(G); A12:=-SIN(G)*COS(B); A13:=SIN(B);
A21:=SIN(A)*SIN(B)*COS(G)+SIN(G)*COS(A);
A22:=-SIN(A)*SIN(G)*SIN(B)+COS(A)*COS(G); A23:=-SIN(A)*COS(B);
'COMMENT' ABOVE VARIABLES ARE ELEMENTS OF ROTATION MATRIX;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'BEGIN'      K:=D/R(/I/);
             X(/I/):=(A11*P(/I/)+A12*Q(/I/)+A13*R(/I/))*K;
             Y(/I/):=(A21*P(/I/)+A22*Q(/I/)+A23*R(/I/))*K
'END'
'END'PROJECT;
```

APPENDIX 3.1

Convex Polyhedron Program Specification.

Abstract.

The program is written in Algol 60 and it locates the visible edges and faces of the convex polyhedron in addition to the hidden edges and faces.

Input.

- Number of vertices n of convex polyhedron.
- n spatial vertices of convex polyhedron.
  (cartesian co-ordinates).
- A viewpoint given in cartesian co-ordinates
  which must lie outside the convex polyhedron.
- Angles of the line of sight (from the viewpoint)
  in degrees.
- Distance of projection plane from the viewpoint.

Output.

- The projected vertices of the convex polyhedron.
- Each plane face of the polyhedron as it is located.
- The visible edges of the polyhedron.
- The hidden edges of the polyhedron.
- The C.P.U. time taken by the program.

APPENDIX 3.2

Program Listings.

The following listings are concerned with the coding necessary in the convex polyhedron program. The C.P.U. time is not determined in the actual listing given but simply consists of two calls of CPUT from which the C.P.U. time can be obtained.

THE FOLLOWING IS A LISTING OF THE CONVEX POLYHEDRON PROGRAM

```
'BEGIN'
'COMMENT' FOLLOWING ARE THE TWO CODED PROCEDURES REQUIRED BY PROGRAM;
'PROCEDURE' PRINT(D,S,N); 'VALUE' D,S,N; 'INTEGER' D; 'REAL' S,N; 'CODE';
'PROCEDURE' CPUT(A); 'INTEGER' A; 'CODE';
'INTEGER' N,CP,CCP,J,I,L,LL,DD,COUNT,COPY;
'REAL' A,B,C,D,XS,YS,ZS,DIST,ALPHA,BETA,GAMMA;
'BOOLEAN' TWICE;
OUTSTRING(1,'('COMPUTER DISPLAY OF CONVEX POLYHEDRA')'); SYSACT(1,2,1);
OUTSTRING(1,'('ENTER NUMBER OF VERTICES')'); SYSACT(1,2,1);
ININTEGER(0,N); SYSACT(1,2,1);
'BEGIN' 'ARRAY' XV,YV,ZV,XX,YY,ZZ,X,Y(/1:N/),DELTA(/1:N,1:N/);
        'INTEGER' 'ARRAY' PER,Q,TT,FOUND(/1:N/),RIM,BR,FON(/1:4*N,1:2/);
        'BOOLEAN' 'ARRAY' BOOL,ON(/1:N/);
         'PROCEDURE' PERIMETER;
         'PROCEDURE' FILL;
         'PROCEDURE' CONSTANTS;
         'REAL' 'PROCEDURE' DIS;
         'PROCEDURE' FINDPLANE;
         'PROCEDURE' PROJECT;
          'PROCEDURE' SORT;
OUTSTRING(1,'('ENTER SPATIAL VERTICES')'); SYSACT(1,2,1);
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'BEGIN'  INREAL(0,XX(/I/)); INREAL(0,YY(/I/)); INREAL(0,ZZ(/I/));
        'COMMENT' PRINT OUT THE SPATIAL VERTICES;
         PRINT(1,-2.0,XX(/I/)); PRINT(1,-2.0,YY(/I/)); PRINT(1,-2.0,ZZ(/I/));
         SYSACT(1,2,1)
'END';
OUTSTRING(1,'('ENTER VIEWPOINT AND DISTANCE OF PROJECTION PLANE')');
SYSACT(1,2,1);
INREAL(0,XS); INREAL(0,YS); INREAL(0,ZS); INREAL(0,DIST);
'COMMENT' PRINT OUT THE VIEWPOINT AND DIST;
PRINT(1,-2.0,XS); PRINT(1,-2.0,YS); PRINT(1,-2.0,ZS); PRINT(1,-2.0,DIST);
```

CONTINUATION OF CODING FOR CONVEX POLYHEDRON PROGRAM.

```
OUTSTRING(1,'('ENTER ANGLES OF LINE OF SIGHT')'); SYSACT(1,2,1);
INREAL(0,ALPHA); PRINT(1,-2.0,ALPHA);
INREAL(0,BETA); PRINT(1,-2.0,BETA);
INREAL(0,GAMMA); PRINT(1,-2.0,GAMMA); SYSACT(1,2,1);
'COMMENT' CHANGE TO RADIANS;
A:=3.14159/180;
ALPHA:=ALPHA*A; BETA:=BETA*A; GAMMA:=GAMME*A;
'COMMENT' TRANSLATE VIEWPOINT TO ORIGIN;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'BEGIN'  XV(/I/):=XX(/I/)-XS;
         YV(/I/):=YY(/I/)-YS;
         ZV(/I/):=ZZ(/I/)-ZS
'END';
PROJECT(XV,YV,ZV,X,Y,N,DIST,ALPHA,BETA,GAMMA);
'COMMENT' ARRAYS X AND Y CONTAIN THE N PROJECTED POINTS;
OUTSTRING(1,'('PROJECTED POINTS AS FOLLOWS')'); SYSACT(1,2,1);
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'BEGIN'  PRINT(1,-2.0,X(/I/)); PRINT(1,-2.0,Y(/I/));
         SYSACT(1,2,1)
'END';
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' Q(/I/):=I;
FILL;
PERIMETER(X,Y,N,PER,K,Q);
'COMMENT' INITIAL CONVEX PLOYGON CONTAINED IN ARRAY PER;
'FOR' I:=1 'STEP' 1 'UNTIL' K 'DO'BR(/I,1/):=RIM(/I,1/):=PER(/I/);
'FOR' I:=1 'STEP' 1 'UNTIL' K-1 'DO' BR(/I,2/):=RIM(/I,2/):=PER(/I+1/);
BR(/K,2/):=RIM(/K,2/):=PER(/1/);
COUNT:=KK:=K;
'COMMENT' SET ALL POINTS ORIGINALLY VALID AS THIRD VERTICES;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' BOOL(/I/):='TRUE';
```

CONTINUATION OF CODING FOR CONVEX POLYHEDRON PROGRAM.

```
BACK:
SYSACT(1,2,1); OUTSTRING(1,'('INITIAL PERIMETER')'); SYSACT(1,2,1);
'FOR' I:=1 'STEP' 1 'UNTIL' K 'DO'
'BEGIN'  PRINT(1,-2.0,BR(/I,1/)); PRINT(1,-2.0,BR(/I,2/));
         SYSACT(1,2,1)
'END';
AGAIN:
L:=RIM(/1,1/); LL:=RIM(/1,2/);
'FOR' I:=L,LL 'DO' BOOL(/I/):='FALSE';
'COMMENT' NOW LOCATE FIRST VALID THIRD VERTEX;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' BOOL(/I/) 'THEN'
'BEGIN'  'IF' ANTICLOCK(I) 'THEN'
         'BEGIN'  COPY:=DD:=I; BOOL(/I/):='FALSE;
              I:=N+1
         'END'
'END';
'COMMENT' DETERMINE CONSTANTS OF PLANE DEFINED BY L,LL AND DD . THEN
LOCATE DD SUCH THAT IT IS NEAREST PLANE;
CONSTANTS;
FINDPLANE;
SYSACT(1,2,1); OUTSTRING(1,'('NEW PERIMETER')'); SYSACT(1,2,1);
'FOR' I:=1 'STEP' 1 'UNTIL' K 'DO'
'BEGIN'  PRINT(1,-2.0,RIM(/I,1/)); PRINT(1,-2.0,RIM(/I,2/));
         SYSACT(1,2,1)
'END';
'COMMENT' POINTS STILL AVAILABLE ARE THOSE FOR WHICH BOOL(/I/) IS STILL TRUE;
'COMMENT' CHECK TO SEE IF PERIMETER LIST IS NULL. IF NOT THEN STILL PLANES
TO BE LOCATED;
'IF' K¬=0 'THEN' 'GOTO' AGAIN;
```

CONTINUATION OF CODING FOR CONVEX POLYHEDRON PROGRAM.

```
'IF' TWICE 'THEN'
'BEGIN'  OUTSTRING(1,'('EDGES VISIBLE TO VIEWPOINT')'); SYSACT(1,2,1);
         'FOR' I:=1 'STEP' 1 'UNTIL' COUNT 'DO'
         'BEGIN' PRINT(1,-2.0,BR(/I,1/)); PRINT(1,-2.0,BR(/I,2/));
                SYSACT(1,2,1)
         'END';
         'COMMENT' SET INITIAL PERIMETER TO CURRENT PERIMETER SO THAT
         HIDDEN PLANES CAN NOW BE LOCATED;
         'FOR' I:=1 'STEP' 1 'UNTIL' KK 'DO'
         'BEGIN'  RIM(/I,1/):=BR(/I,1/); RIM(/I,2/):=BR(/I,2/);
                 BOOL(/RIM(/I,1/)/):=BOOL(/RIM(/I,2/)/):='TRUE'
         'END';
        K:=KK; KK:=COUNT+1; TWICE:='FALSE';
         'GOTO' BACK
'END';
OUTSTRING(1,'('EDGES HIDDEN TO VIEWPOINT')'); SYSACT(1,2,1);
'FOR' I:=KK 'STEP' 1 'UNTIL' COUNT 'DO'
'BEGIN'  PRINT(1,-2.0,BR(/I,1/)); PRINT(1,-2.0,BR(/I,2/));
         SYSACT(1,2,1)
'END'
'END';
```

```
'PROCEDURE' PERIMETER(XT,YT,N,PER,C,Q);
'VALUE' N; 'INTEGER' N,C; 'ARRAY' XT,YT; 'INTEGER' 'ARRAY' PER,Q;
'COMMENT' PROCEDURE ACCEPTS N POINTS (XT,YT) ON PLANE OF PROJECTION AND
RETURNS IN PER C ORDERED POINTS WHICH DEFINE THE ENCLOSING CONVEX POLYGON
OF THE ORIGINAL N POINTS;
'BEGIN' 'INTEGER' I,J,GG,G,S,CC,LI; 'REAL' W,WW,A,TOT,INIT,PI;
'INTEGER' 'ARRAY' PLANE(/1:N/),NODE(/1:N,1:2/);
'ARRAY' STORE(/1:N/);
          'PROCEDURE' RETURN;
          'PROCEDURE' FIRS;
          'PROCEDURE' ERASE;
ROUND:='TRUE';
'COMMENT' SET UP LIST PROCESSING BI DIRECTIONAL ARRAYS;
NODE(/1,1/):=N; NODE(/1,2/):=2;
'FOR' I:=2 'STEP' 1 'UNTIL' N-1 'DO'
'BEGIN' NODE(/I,1/):=I-1; NODE(/I,2/):=I+1 'END';
NODE(/N,1/):=N-1; NODE(/N,2/):=1;
'COMMENT' SET POINTER LI TO FIRST ELEMENT IN LIST;
LI:=1;
PI:=3.1415927; INIT:=2*PI;
C:=0;
'COMMENT' SET INITIAL COUNT OF PERIMETER POINTS C TO ZERO. NOW LOCATE
INITIAL POINT ON CONVEX POLYGON;
FIRS(G,N,Q); S:=G;
'COMMENT' TAKE COPY IN S OF FIRST ELEMENT;
AGAIN:    J:=0;
'COMMENT' J IS FLAG FOR PARALLEL PLANES;
GG:=G; MIN:=INIT;
```

THE FOLLOWING IS THE REMAINDER OF THE PERIMETER PROCEDURE.

```
'COMMENT' ERASE ELEMENT FROM LIST AND PUT IN PER LIST;
ERASE(G); C:=C+1; PER(/C/):=G;
'FOR' I:=LI,NODE(/I,2/) 'WHILE' I¬=LI 'DO'
'BEGIN'  A:='IF' ROUND 'THEN' DELTA(/GG,I/) 'ELSE' DELTA(/I,GG/);
         'IF' A<MIN 'THEN' 'BEGIN' MIN:=A; G:=I; J:=0; 'END' 'ELSE'
         'IF' ABS(A-MIN)<'-6 'THEN'
             'BEGIN' J:=1; STORE(/J/):=WW:=DIS(X(/G/),Y(/G/),X(/GG/),Y(/GG/));
                 PLANE(/J/):=G
             'END';
             J:=J+1; STORE(/J/):=W:=DIS(X(/I/),Y(/I/),X(/GG/),Y(/GG/));
             PLANE(/J/):=I;
             'IF' W>WW 'THEN' 'BEGIN' WW:=W; G:=I 'END';
             'COMMENT' G IS POINT FURTHEST FROM INITIAL POINT GG;
         'END'
'END';
'COMMENT' G IS NEXT POINT CHOSEN. IF J IS NON ZERO A PARALLEL PLANE HAS
BEEN LOCATED;
'IF' J>0 'THEN' RETURN;
'IF' G=LI 'THEN' LI:=NODE(/G,2/);
'COMMENT' IF G INITIAL POINT FOR LIST PROCESSING ARRAY REPLACE WITH NEXT;
'IF' ROUND 'THEN'
'BEGIN' 'IF' MIN>=PI 'THEN' ROUND:='FALSE'; 'GOTO' AGAIN 'END';
'COMMENT' CHECK TERMINATING CONDITION;
'IF' DELTA(/S,G/)>MIN 'THEN' 'GOTO' AGAIN;
'END'PERIMETER;
```

```
'PROCEDURE' FIRS(G,N,Q);
'VALUE' N; 'INTEGER' G,N; 'INTEGER' 'ARRAY' Q;
'COMMENT' PROCEDURE LOCATES THE NODE WITH THE LEAST Y COORDINATE.
IF THERE IS MORE THAN ONE THEN THAT NODE WHICH HAS THE LEAST X COORDINATE
IS CHOSEN AND IF STILL MORE THAN ONE THEN THAT ONE WHOSE SPATIAL VERTEX IS
CLOSEST TO THE VIEWPOINT IS CHOSEN;
'BEGIN' 'INTEGER' J,I; 'REAL' MIN;
MIN:=Y(/Q(/1/)/); G:=Q(/1/);
'COMMENT' MIN IS THE CURRENT LEAST VALUE OF Y AND G IS THE ELEMENT WITH
THIS LEAST VALUE;
'FOR' I:=2 'STEP' 1 'UNTIL' N 'DO'
'BEGIN'     J:=Q(/I/);
  'IF' Y(/J/)<MIN 'THEN'
   'BEGIN' MIN:=Y(/J/);  G:=J 'END' 'ELSE'
   'IF' ABS(Y(/J/)-MIN)<'-6 'THEN'
   'BEGIN' 'IF' X(/J/)<X(/G/) 'THEN' G:=J 'ELSE'
    'IF' ABS(X(/J/)-X(/G/))>'-6 'THEN' 'ELSE'
    'IF' ZV(/J/)>ZV(/G/) 'THEN' G:=J
   'END'
'END'
'END'FIRS;
```

```
'PROCEDURE' FILL;
'COMMENT' PROCEDURE CALCULATES THE POSITIVE ANGLE BETWEEN EACH OF THE
NODES ON THE PLANE OF PROJECTION AND STORES THEM IN THE TWO DIMENSIONAL
ARRAY DELTA. THE POSITIVE ANGLE BETWEEN NODES I AND J IS GIVEN BY
DELTA(/I,J/).
THE GLOBAL PROCEDURE ANG IS USED AND IS CALLED N*(N-1)/2 TIMES;
'BEGIN' 'INTEGER' I,J; 'REAL' PI;
PI:=3.14159;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' I-1 'DO'
'BEGIN' DELTA(/I,J/):=ANG(X(/J/)-X(/I/),Y(/J/)-Y(/I/));
        DELTA(/J,I/):=DELTA(/I,J/)+('IF' DELTA(/I,J/)>PI 'THEN' -PI 'ELSE' PI)
'END'
'END'FILL;

'PROCEDURE' ERASE(S);
'VALUE' S; 'INTEGER' S;
'COMMENT' PROCEDURE ELIMINATES THE ELEMENT S FROM THE AVAILABLE LIST
OF ELEMENTS. THIS PROCEDURE IS INVOKED BY THE PERIMETER
PROCEDURE AND IS DECLARED WITHIN PERIMETER;
'BEGIN' NODE(/NODE(/S,1/),2/):=NODE(/S,2/);
        NODE(/NODE(/S,2/),1/):=NODE(/S,1/)
'END'ERASE;

'BOOLEAN' 'PROCEDURE' ANTICLOCK(I);
'VALUE' I; 'INTEGER' 2;
'COMMENT' PROCEDURE DETERMINES WHETHER A GIVEN NODE IS A VALID THIRD
VERTEX. IF SO ANTICLOCK IS SET TRUE OTHERWISE IT IS SET FALSE;
'IF' X(/LL/)*Y(/I/)-X(/I/)*Y(/LL/)-X(/L/)*Y(/I/)+X(/I/)*Y(/L/)+X(/L/)*Y(/LL/)-
X(/LL/)*Y(/L/)>0 'THEN' ANTICLOCK:='TRUE' 'ELSE' ANTICLOCK:='FALSE';
```

```
'PROCEDURE' RETURN;
'COMMENT' PROCEDURE ACCEPTS AN ARRAY 'PLANE' OF VERTICES, WHICH LIE ON
A FACE PARALLEL TO THE LINE OF SIGHT AND DETERMINES THOSE WHICH ARE
VISIBLE AND WHOSE CORRESPONDING POINTS WILL LIE ON THE ENCLOSING CONVEX
POLYGON;
'BEGIN' 'REAL' P,Q,R,M; 'INTEGER' S,I; 'INTEGER' 'ARRAY' B(/1:L/);
LL:=GG; DD:=G; L:=PLANE(/1/);
'IF' L=G 'THEN' L:=PLANE(/2/);
'COMMENT' VERTICES L,LL AND DD ARE DISTINCT VERTICES ON PARALLEL FACE;
CONSTANTS;
'COMMENT' GLOBAL VARIABLES A,B,C AND D DEFINE THE PARALLEL FACE;
M:=-1/(XV(/LL/)*(YV(/DD/)*C-B*ZV(/DD/)-YV(/LL/)*(XV(/DD/)*C-A*ZV(/DD/)
   +ZV(/LL/)*(XV(/DD/)*B-A*|=(/DD/)));
P:=M*(YV(/DD/)*C-B*ZV(/DD/)-YV(/LL/)*C+ZV(/LL/)*B);
Q:=M*(XV(/LL/)*C-XV(/DD/)*C+A*ZV(/DD/)-ZV(/LL/)*A);
R:=M*(-XV(/LL/)*B-YV(/LL/)*A+XV(/DD/)*B-A*YV(/DD/));
L:=0;
'COMMENT' P,Q,R DEFINE PLANE PERPENDICULAR TO PARALLEL FACE WHICH PASSES
THROUGH THE TWO EXTREME POINTS ON PROJECTION PLANE;
'FOR' I:=1 'STEP' 1 'UNTIL' J 'DO' 'IF' G¬=PLANE(/I/) 'THEN'
'BEGIN'    S:=PLANE(/I/);
           ERASE(S);
           'IF' S=LI 'THEN' LI:=NODE(/LI,2/);
           'IF' P*XV(/S/)+Q*YV(/S/)+R*ZV(/S/)>0
           'THEN' 'BEGIN'   L:=L+1; PLANE(/L/):=S;
                            STORE(/L/):=STORE(/S/)
                 'END'
'END';
SORT(B,STORE,L);
'FOR' I:=1 'STEP' 1 'UNTIL' L 'DO' PER(/C+I/):=PLANE(/B(/I/)/);
C:=C+L+1;
PER(/C/):=G;
'END'RETURN;
```

```
'PROCEDURE' FINDPLANE;
'COMMENT' PROCEDURE LOCATES NEXT PLANE;
'BEGIN' 'REAL' G; 'INTEGER' COUNT,I,D;
COUNT:=3;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' BOOL(/I/) 'THEN'
'BEGIN'  G:=D*(A*XV(/I/)+B*YV(/I/)+C*ZV(/I/)+D);
         'IF ABS(G)>'-6 'THEN'
         'BEGIN' 'IF' TWICE 'EQUIV' G>0 'THEN'
             'BEGIN' 'IF' ANTICLOCK(I) 'THEN'
                 'BEGIN' COUNT:=3; DD:=2;
                        CONSTANTS;
                      'COMMENT' NEW THIRD POINT LOCATED;
                 'END'
             'END'
         'END' 'ELSE'
         'BEGIN' COUNT:=COUNT+1;
             'COMMENT' STORE POINT AS VERTEX ON CURRENT PLANE;
             FOUND(/COUNT/):=I
         'END'
'END';
FOUND(/1/):=L; FOUND(/2/):=LL; FOUND(/3/):=DD;
PERIMETER(X,Y,COUNT,Q,D);
'COMMENT' STORE THE POINTS IN CLOCKWISE ORDER IN ARRAY FON;
S:=0;
'FOR' I:=D 'STEP' -1 'UNTIL' 2 'DO'
'BEGIN' S:=S+1;
         FON(/S,1/):=FOUND(/Q(/I/)/);
         FON(/S,2/):=FOUND(/Q(/I-1/)/)
'END';
FON(/D,1/):=FOUND(/Q(/1/)/); FON(/D,2/):=FOUND(/Q(/D/)/);
'FOR' I:=1 'STEP' 1 'UNTIL' D 'DO'
'BEGIN'  TT(/FOUND(/I/)/):=0; ON(/I/):='TRUE' 'END';
```

THIS IS THE REMAINDER OF PROCEDURE FINDPLANE;

```
'COMMENT' PRINT OUT RELEVENT INFORMATION;
OUTSTRING(1,'('NEXT PLANE LOCATED')'); SYSACT(1,2,1);
'FOR' I:=1 'STEP' 1 'UNTIL' D 'DO'
'BEGIN' PRINT(1,-2.0,FON(/I,1/)); PRINT(1,-2.0,FON(/I,2/));
         SYSACT(1,2,1);
'END';
'COMMENT' CHECK TO ASCERTAIN IF ANY SEGMENTS OF CURRENT PERIMETER
CAN BE ERASED. IF ANY CURRENT PLANE SEGMENTS CAN BE ERASED SET
CORRESPONDING ELEMENT IN BOOLEAN ARRAY ON FALSE;
'FOR' I:=1 'STEP' 1 'UNTIL' D 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' K 'DO'
'IF' FON(/I,2/)¬=RIM(/J,1/) 'THEN' 'ELSE'
'IF' FON(/I,1/)=RIM(/J,2/) 'THEN'
'BEGIN'   ERAD(J,K,RIM); ON(/I/):='FALSE; J:=K+1;
         'FOR' S:=1,2 'DO' TT(/FON(/I,S/)/):=TT(/FON(/I,S/)/)+1
'END';
'COMMENT' CHECK TO ASCERTAIN IF ANY POINTS CAN BE ELIMINATED;
'FOR' I:=1 'STEP' 1 'UNTIL' D 'DO'
'IF' TT(/FOUND(/I/)/)=2 'THEN' BOOL(/FOUND(/I/)/):='FALSE'
'FOR' I:=1 'STEP' 1 'UNTIL' D 'DO' 'IF' ON(/I/) 'THEN'
'BEGIN'   K:=K+1;  COUNT:=COUNT+1;
      'FOR' J:=1,2 'DO' RIM(/K,J/):=BR(/COUNT,J/):=FON(/I,J/)
'END';
'END'FINDPLANE;
```

```
'PROCEDURE' CONSTANTS;
'COMMENT' PROCEDURE ACCEPTS THREE VERTICES DEFINED BY THE INTEGERS L,LL AND
DD AND RETURNS VARIABLES A,B,C AND D SUCH THAT THEY DEFINE THE PLANE ON WHICH
THE THREE VERTICES LIE;
'BEGIN'
A:=YV(/L/)*(ZV(/LL/)-ZV(/DD/)-ZV(/L/)*(YV(/LL/)-YV(/DD/))+YV(/LL/)*ZV(/DD/)
   -YV(/DD/)*ZV(/LL/);
B:=-XV(/L/)*(ZV(/LL/)-ZV(/DD/)+ZV(/L/)*(XV(/LL/)-XV(/DD/))
   -XV(/LL/)*ZV(/DD/)+XV(/DD/)*ZV(/LL/);
C:=XV(/L/)*(YV(/LL/)-YV(/DD/)-YV(/L/)*(XV(/LL/)-XV(/DD/)+XV(/LL/)*YV(/DD/)
   -XV(/DD/)*YV(/LL/);
D:=-(XV(/L/)*(YV(/LL/)*ZV(/DD/)-YV(/DD/)*ZV(/LL/)-YV(/L/)*(XV(/LL/)*ZV(/DD/)
-ZV(/LL/)*XV(/DD/)+ZV(/L/)*(XV(/LL/)*YV(/DD/)-XV(/DD/)*YV(/LL/)));
'COMMENT' PLANE DEFINED BY AX+BY+CZ+D=0;
'END'CONSTANTS;


'PROCEDURE' ERAD(I,K,PER);
'INTEGER' I,K; 'INTEGER' 'ARRAY' PER;
'COMMENT' PROCEDURE ELIMINATES ELEMENT I FROM ARRAY PER BY SHIFTING
THE ELEMENTS DOWN ONE PLACE AND REDUCING NUMBER OF ELEMENTS K BY ONE;
'BEGIN' 'INTEGER' J;
K:=K-1;
'FOR' J:=I 'STEP' 1 'UNTIL' K 'DO'
'BEGIN'  PER(/J,1/):=PER(/J+1,1/);
         PER(/J,2/):=PER(/J+1,2/)
'END'
'END'ERAD;
```

APPENDIX 4.1

General Polyhedron Program Specification

Abstract.

The program is written in Algol 60 and it calculates those portions of the edges of the polyhedron which are hidden to the viewpoint and those which are visible.

Input.

- Number of vertices n of the polyhedron.
- Cartesian co-ordinates of the n spatial vertices of the polyhedron.
- Number of faces of the polyhedron and the vertices of each face ordered in a given direction.
- Viewpoint given in cartesian co-ordinates which must lie outside the polyhedron.
- Angle of the line of sight (from the viewpoint) in degrees.
- Distance of projection plane from the viewpoint.

Output.

- The projected vertices of the polyhedron.
- The edges of the polyhedron.
- The depth counts of each of the partial line segments defined for the polyhedron (definitions given in Chapter 4).
- The C.P.U. time taken by the program.

APPENDIX 4.2

Program Listings.

The following listings are related to the non-convex polyhedron program, the theory for which was explained in Chapter 4.

CODING OF MAIN BODY OF NON-CONVEX POLYHFDPON PROGRAM

```
'INTEGER' POL,N,CP,CPU,S,L,I,J,H,U,V,CC,COUNT,NUMBER,PP,JJ;
'REAL' A,B,C,D,AA,DEN,BR,CX,CY,CZ,GX,GY,ALPHA,BETA,GAMMA,QX,QY,QZ;
OUTSTRING(1,'('ENTER VALUES FOR PP,POLY,N AND L')'); SYSACT(1,2,1);
ININTEGER(0,PP); ININTEGER(0,POL); ININTEGER(0,N); ININTEGER(0,L);
'COMMENT' CAN NOW DECLARE SOME ARRAYS;
'BEGIN' 'ARRAY' P,Q,R,X,Y(/1:N/),E(/1:POL,1:4/),DEPTH(/1:7*L,1:7/);
'INTEGER' 'ARRAY' TERM,POLY(/1:L,1:2/),PTS(/1:2*L+POL/),
                  FIRST(/1:POL+1/),START(/1:L+1/);
A:=3.14159/180;
OUTSTRING(1,'('ENTER VIEWPOINT AND VALUES FOR ALPHA,BETA AND GAMMA')');
SYSACT(1,2,1);
I REAL(0,CX); INREAL(0,CY); INREAL(0,CZ);
INREAL(0,ALPHA); INREAL(0,BETA); INRFAL(0,GAMMA);
INREAL(0,D);
'COMMENT' CHANGE ANGLES TO RADIANS;
ALPHA:=ALPHA*A; BETA:=BETA*A; GAMMA:=GAMMA*A;
OUTSTRING(1,'('ENTER SPATIAL VERTICES')'); SYSACT(1,2,1);
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'BEGIN'  INREAL(0,P(/I/)); P(/I/):=P(/I/)-CX;
         INREAL(0,Q(/I/)); Q(/I/):=Q(/I/)-CY;
         INREAL(0,R(/I/)); R(/I/):=R(/I/)-CZ;
'END';
PROJECT(P,Q,R,X,Y,ALPHA,BETA,GAMMA,D);
'COMMENT' X AND Y CONTAIN TRANSFORMED POINTS;
OUTSTRING(1,'('ENTER NUMBER OF POINTS AND POINTS OF EACH FACE')');
SYSACT(1,2,1);
H:=0;
```

CODING OF MAIN BODY OF NON-CONVEX POLYHEDRON PROGRAM

```
'FOR' I:=1 'STEP' 1 'UNTIL' POL 'DO'
'BEGIN'    ININTEGER(0,S); FIRST(/I/):=H+1;
   'FOR' J:=1 'STEP' 1 'UNTIL' S 'DO'
    'BEGIN'   H:=H+1; ININTEGER(0,PTS(/H/)) 'END';
   H:=H+1; PTS(/H/):=PTS(/FIRST(/I/)/)
'END';
FIRST(/POL+1/):=H+1;
FINISH:  CPUT(CP);
'COMMENT' DETERMINE EQUATION OF EACH PLANE IN FORM AX+BY+CZ+D=0;
'FOR' I:=1 'STEP' 1 'UNTIL' POL 'DO'
'BEGIN'    J:=FIRST(/I/);
   H:=PTS(/J/); U:=PTS(/J+1/); V:=PTS(/J+2/);
    CONSTANTS(E(/I,1/),E(/I,2/),E(/I,3/),E(/I,4/))
'END';
'COMMENT' DETERMINE TERMINAL POINTS OF LINE SEGMENTS;
L:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' POL 'DO'
'BEGIN'    H:=FIRST(/I+1/)-2;
    'FOR' J:=FIRST(/I/) 'STEP' 1 'UNTIL' H 'DO' 'IF' PTS(/J/)<PTS(/J+1/)
     'THEN' 'BEGIN' L:=L+1; POLY(/L,1/):=I;
             TERM(/L,1/):=PTS(/J/); TERM(/L,2/):=PTS(/J+1/)
            'END'
'END';
'COMMENT' NOW DETERMINE TERMINAL POINTS OF LINE SEGMENTS;
'FOR' I:=1 'STEP' 1 'UNTIL' L 'DO'
'BEGIN' 'FOR' J:=1 'STEP' 1 'UNTIL' POL 'DO'
    'BEGIN'  H:=FIRST(/J+1/)-2;
    'FOR' J:=FIRST(/I/) 'STEP' 1 'UNTIL' H 'DO' 'IF' PTS(/J/)<PTS(/J+1/) 'THEN'
    'BEGIN'  L:=L+1;  POLY(/L,1/):=I;
          TERM(/L,1/):=PTS(/J/); TERM(/L,2/):=PTS(/J+1/)
     'END'
 'END';
```

CODING OF MAIN BODY OF NON-CONVEX POLYHEDRON PROGRAM

```
'COMMENT' DETERMINE SECOND POLYGON ASSOCIATED WITH EACH LINE SEGMENT;
 'FOR' I:=1 'STEP' 1 'UNTIL' L 'DO'
 'BEGIN' 'FOR' J:=1 'STEP' 1 'UNTIL' POL 'DO'
      'BEGIN'   H:=FIRST(/J+1/)-2;
      'FOR' U:=FIRST(/J/) 'STEP' 1 'UNTIL' H 'DO' 'IF' PTS(/U/)<PTS(/U+1/)
     'THEN' 'ELSE' 'IF' PTS(/U/)=TERM(/I,2/) 'AND' PTS(/U+1/)=TERM(/I,1/) 'THEN'
     'BEGIN' POLY(/I,2/):=J; 'GOTO' FIN 'END'
'END';
FIN:
'END';
OUTSTRING(1,'('SPATIAL LINE SEGMENTS')'); SYSACT(1,2,1);
'FOR' I:=1 'STEP' 1 'UNTIL' L 'DO'
'BEGIN' PRINT(1,3,I);
       'FOR' J:=1,2 'DO' PRINT(1,3,TERM(/I,J/)); SYSACT(1,2,1); 'END';
OUTSTRING(1,'('SPATIAL POLYGONS OF LINE SEGMENTS')'); SYSACT(1,2,1);
'FOR' I:=1 'STEP' 1 'UNTIL' L 'DO'
'BEGIN'   PRINT(1,5,I);
     'FOR' J:=1,2 'DO' PRINT(1,4,POLY(/I,J/)); SYSACT(1,2,1)
'END';
INTERLINES(X,Y,N,TERM,L,DEPTH,CC);
START(/L+1/):=CC+1;
FILLDEPTH;
CPUT(CPU);
'COMMENT' L1 IS PRIMARY AND L2 IS SECONDARY LINE SEGMENT.
(XF,YF),(XL,YL) ARE TERMINAL COORDS OF PARTIAL LINE SEGMENT;
OUTSTRING(1,'('   L1   L2   XF   YF   XL   YL DEPTH COUNT')'); SYSACT(1,2,1);
'FOR' I:=1 'STEP' 1 'UNTIL' CC 'DO'
'BEGIN'  'FOR' J:=1 'STEP' 1 'UNTIL' 7 'DO' PRINT(1,-3.0,DEPTH(/I,J/));
          SYSACT(1,2,1)
'END';
OUTSTRING(1,'('CPU TIME ')'); PRINT(6,4,625/48000000*(CPU-CP));
'END'
'END';
```

```
'PROCEDURE' INTERLINES(X,Y,N,E,RR,DEPTH,T);
'VALUE' N,RR; 'INTEGER' N,RR,T; 'ARRAY' X,Y,DEPTH; 'INTEGER' 'ARRAY' E;
'COMMENT' DETERMINES ALL REAL INTERSECTIONS OCCURRING ON PLANE OF PROJECTION
OF THE RR SEGMENTS CONTAINED IN TWO DIMENSIONAL ARRAY E. X AND Y ARE ARRAYS
CONTAINING THE N PROJECTED POINTS. RESULTS STORED IN ARRAY DEPTH, THERE BEING
C REAL INTERSECTIONS;
'BEGIN' 'ARRAY' XX,YY,M(/1:N/),L(/1:2*RR,1:4/);
'INTEGER' 'ARRAY' SB,NUM(/1:N/);
          'PROCEDURE' INTERPOINT;
          'PROCEDURE' SORT;
          'PROCEDURE' MASK;
          'PROCEDURE' INTERSECT;
'INTEGER' S,I,J,CC,A,B,C,D; 'REAL' F,G;
'COMMENT' SET NUMBER OF REAL INTERSECTIONS T AND NUMBER OF VIRTUAL
INTERSECTIONS CC TO INITIAL VALUES OF ZERO;
'FOR' I:=1 'STEP' 1 'UNTIL' RR 'DO'
'BEGIN' S:=0;
'COMMENT' S IS NUMBER OF REAL INTERSECTIONS FOR THE PRIMARY SEGMENT I;
A:=E(/I,1/); B:=E(/I,2/);
'COMMENT' A AND B ARE TERMINAL POINTS OF PRIMARY LINE SEGMENT I;
          'FOR' J:=I+1 'STEP' 1 'UNTIL' RR 'DO'
          'BEGIN' C:=E(/J,1/); D:=E(/J,2/);
          'COMMENT' C AND D ARE TERMINAL POINTS OF SECONDARY SEGMENT J;
             'IF' (A=C 'OR' A=D 'OR' B=C 'OR' B=D) 'THEN' 'ELSE'
             'IF' INTERSECT(X(/A/),Y(/A/),X(/B/),Y(/B/),X(/C/),Y(/C/),X(/D/),Y(/D/))
            'THEN' 'BEGIN'
                    INTERPOINT(X(/A/),Y(/A/),X(/B/),Y(/B/),X(/C/),Y(/C/)
                              X(/D/),Y(/D/),F,G);
                    'COMMENT' (F,G) IS POINT OF INTERSECTION. NOW TEST
                    FOR REAL INTERSECTION. IF VIRTUAL THEN STORE SINCE THIS
                    WILL BE REAL FOR PRIMARY J AND SECONDARY I;
```

CONTINUATION OF INTERLINES PROCEDURE.

```
                     'IF' MASK(I,J,F,G) 'THEN'
                     'BEGIN' S:=S+1; XX(/S/):=F; YY(/S/):=G;
                         M(/S/):=DIS(X(/A/),Y(/A/),F,G); NUM(/S/):=J
                     'END' 'ELSE'
                     'BEGIN'    CC:=CC+1;
                         L(/CC,1/):=I; L(/CC,2/):=J;
                         L(/CC,3/):=F; L(/CC,4/):=G
                     'END'
                   'END'
          'END';
          'COMMENT' TEST IF SEGMENT I HAS BEEN SECONDARY LINE SEGMENT FOR
          PREVIOUSLY LOCATED VIRTUAL INTERSECTION;
          'FOR' J:=1 'STEP' 1 'UNTIL' CC 'DO' 'IF' L(/J,2/)=I 'THEN'
          'BEGIN' S:=S+1;
              M(/S/):=DIS(X(/A/),Y(/A/),L(/J,3/),L(/J,4/));
              NUM(/S/):=L(/J,1/);
              XX(/S/):=L(/J,3/); YY(/S/):=L(/J,4/)
          'END';
          'COMMENT' SORT THE REAL INTERSECTIONS ALONG PRIMARY SEGMENT I;
          SORT(SB,M,S);
          DEPTH(/T+1,1/):=I; DEPTH(/T+1,2/):=0;
          DEPTH(/T+1,3/):=X(/A/); DEPTH(/T+1,4/):=Y(/A/);
          'COMMENT' SET POINTER TO START OF REAL INTERSECTIONS FOR PRIMARY
          LINE SEGMENT I. STORE THE S REAL INTERSECTIONS FOR I IN ARRAY DEPTH;
          'FOR' J:=1 'STEP' 1 'UNTIL' S 'DO'
          'BEGIN' T:=T+1; C:=SB(/J/);
             DEPTH(/T,5/):=DEPTH(/T+1,3/):=XX(/B/);
             DEPTH(/T,6/):=DEPTH(/T+1,4/):=YY(/C/);
             DEPTH(/T+1,1/):=I; DEPTH(/T+1,2/):=NUM(/C/)
          'END';
'END';
'COMMENT' SET INITIAL POINT FOR NEXT PARTIAL LINE SEGMENT;
T:=T+1; DEPTH(/T,5/):=X(/B/); DEPTH(/T,6/):=Y(/B/)
'END'
'END'INTERLINES;
```

```
'PROCEDURE' FILLDEPTH;
'COMMENT' CALCULATES THE DEPTH COUNT ASSOCIATED WITH EVERY PARTIAL LINE
SEGMENT BY AN INITIAL DETERMINATION OF DEPTH COUNT AND ASSOCIATED CALCULATIONS
TO ASCERTAIN DEPTH COUNTS OF ADJACENT PARTIAL LINE SEGMENTS;
'BEGIN' 'REAL' F,G; 'INTEGER' D,I,J,K,S; 'BOOLEAN' BOOL;
  'FOR' I:=1 'STEP' 1 'UNTIL' L 'DO'
  'BEGIN'  J:=START(/I/);
  'COMMENT' J IS POSITION IN DEPTH OF FIRST PARTIAL LINE SEGMENT;
          G:=(DEPTH(/J,3/)+DEPTH(/J,5/))/2;
          H:=(DEPTH(/J,4/)+DEPTH(/J,5/))/2;
          'COMMENT' DETERMINE DEPTH COUNT OF FIRST PARTIAL LINE SEGMENT
          FOR PRIMARY LINE SEGMENT I;
          FLANK(G,H,I,D);
          'COMMENT' FLANK RETURNS D AS DEPTH COUNT OF FIRST PARTIAL LINE SEG;
          DEPTH(/J,7/):=D;
          J:=START(/I+1/)-1;
          'COMMENT' J IS POSITION IN DEPTH OF LAST PARTIAL LINE SEGMENT.
          NOW SCAN REST OF PARTIAL LINE SEGMENTS IN PRIMARY SEGMENT I;
          'FOR' K:=START(/I/)+1 'STEP' 1 'UNTIL' J 'DO'
          'BEGIN'  G:=(DEPTH(/K,3/)+DEPTH(/K,5/))/2;
                H:=(DEPTH(/K,4/)+DEPTH(/K,6/))/2;
                S:=DEPTH(/K,2/);
                BOOL:=INSIDE(G,H,POLY(/S,1/));
                'IF' BOOL 'EQUIV' INSIDE(G,H,POLY(/S,2/)) 'THEN'
                'BEGIN' 'IF' BOOL 'THEN' D:=D+1 'ELSE' D:=D-1
                'END';
                DEPTH(/K,7/):=D
             'END'
  'END'
'END'FILLDEPTH;
```

```
'BOOLEAN' 'PROCEDURE' MASK(L1,L2,XX,YY);
'VALUE' L1,L2,XX,YY; 'INTEGER' L1,L2; 'REAL' XX,YY;
'COMMENT' PROCEDURE DETERMINES IF SEGMENT L2 LIES ON VIEWPOINT SIDE OF L1;
'BEGIN' 'INTEGER' A,B; 'REAL' RATIO,D,DD;
     A:=TERM(/L1,1/); B:=TERM(/L1,2/);
     'COMMENT' A AND B ARE TERMINAL POINTS OF SEGMENT L1;
     RATIO:=DIS(X(/A/),Y(/A/),XX,YY)/DIS(X(/A/),Y(/A/),X(/B/),Y(/B/));
     D:=P(/A/)+RATIO*(P(/B/)-P(/A/));
     A:=TERM(/L2,1/); B:=TERM(/L2,2/);
     'COMMENT' A AND B NOW TERMINAL POINTS OF SEGMENT L2;
     RATIO:=DIS(X(/A/),Y(/A/),XX,YY)/DIS(X(/A/),Y(/A/),X(/B/),Y(/B/));
     DD:=P(/A/)+RATIO*(P(/B/)-P(/A/));
     'IF' ABS(D)>ABS(DD) 'THEN' MASK:='TRUE' 'ELSE' MASK:='FALSE'
'END'MASK;


'PROCEDURE' INTER(XA,YA,XB,YB,XC,YC,XD,YD,X,Y);
'VALUE' XA,YA,XB,YB,XC,YC,XD,YD; 'REAL' XA,YA,XB,YB,XC,YC,XD,YD,X,Y;
'COMMENT' PROCEDURE DETERMINES INTERSECTION (ASSUMED TO EXIST) BETWEEN
SEGMENTS (XA,YA)-(XB,YB),(XC,YC)-(XD,YD) AND ASSIGNS INTERSECTION POINT
TO VARIABLES (X,Y);
'BEGIN' 'REAL' DX1,DY1,DX3,DY3,K,C;
     DX1:=XB-XA; DY1:=YB-YA;
     DX3:=XD-XC; DY3:=YD-YC;
     C:=DX3*DY1-DY3*DX1;
     'IF' ABS(C)<'-6 'THEN'
     'BEGIN' OUTSTRING(1,'('PARALLEL LINES')');
           SYSACT(1,2,1)
     'END' 'ELSE'
     'BEGIN' K:=(DX3*(YC-YA)-DY3*(XC-XA)/C;
           X:=XA+K*DX1; Y:=YA+K*DY1
     'END'
'END'INTER;
```

```
'PROCEDURE' FLANK(G,H,L1,D);
'VALUE' G,H,L1; 'INTEGER' D,L1; 'REAL' G,H;
'COMMENT' GIVEN A POINT (G,H) ON THE PROJECTION PLANE ON SEGMENT L1 PROCEDURE
DETERMINES CORRESPONDING SPATIAL POINT AND DETERMINES THE NUMBER OF PLANES
WHICH MASK THIS POINT FROM THE GIVEN VIEWPOINT;
'BEGIN' 'REAL' SX,SY,SZ,RATIO; 'INTEGER' I,J;
I:=TERM(/L1,1/); J:=TERM(/L1,2/);
'COMMENT' I AND J ARE TERMINAL POINTS OF SEGMENT L1;
RATIO:=DIS(G,H,X(/I/),Y(/I/))/DIS(X(/J/),Y(/J/),X(/I/),Y(/I/));
'COMMENT' RATIO GIVES RATIO ON PROJECTION PLANE OF (G,H) FROM POINT I
TO LENGTH FROM POINT I TO POINT J;
SX:=P(/I/)+RATIO*(P(/J/)-P(/I/));
SY:=Q(/I/)+RATIO*(Q(/J/)-Q(/I/));
SZ:=R(/I/)+RATIO*(R(/J/)-R(/I/));
'COMMENT' (SX,SY,SZ) IS POINT IN SPACE CORRESPONDING TO (G,H);
D:=0;
'COMMENT' SET INITIAL DEPTH COUNT TO ZERO;
'FOR' J:=1 'STEP' 1 'UNTIL' POL 'DO'
'BEGIN'   'IF' J=POLY(/L1,1/) 'OR' J=POLY(/L1,2/) 'THEN' 'ELSE'
          'IF' E(/J,4/)*(SX*E(/J,1/)+SY*E(/J,2/)+SZ*E(/J,3/)
          +E(/J,4/))>=0 'THEN' 'ELSE'
          'IF' INSIDE(G,H,J) 'THEN' D:=D+1;
          'COMMENT' ABOVE STATEMENT CHECKS TO ASCERTAIN IF PLANE J IS ASSOCIATED
          WITH EDGE L1. IF IT IS NOT THEN CHECK MADE TO DETERMINE IF
          (SX,SY,SZ) LIES ON OPPOSITE SIDE TO VIEWPOINT. IF IT DOES THEN
          DEPTH COUNT D IS 1NCREASED BY 1;
'END'
'END'FLANK;
```

```
'BOOLEAN' 'PROCEDURE' INSIDE(PX,PY,J);
'VALUE' PX,PY,J; 'INTEGER' J; 'REAL' PX,PY;
'BEGIN' 'REAL' S; 'INTEGER' I,A,B,LAST;
'BOOLEAN' ON;
'COMMENT' DETERMINES IF GIVEN POINT (PX,PY) LIES INSIDE POLYGON J.
          INSIDE SET TRUE IF POINT INSIDE;
           ON:='FALSE'; LAST:=FIRST(/J+1/)-2;
'COMMENT' LAST POINTS TO LAST ELEMENT IN PTS LIST OF POLYGON J;
'FOR' I:=FIRST(/J/) 'STEP' 1 'UNTIL' LAST 'DO'
'BEGIN'  A:=PTS(/I/); B:=PTS(/I+1/);
           'COMMENT' A AND B ARE ADJACENT ELEMENTS OF POLYGON J;
           'IF' PY≤Y(/A/) 'EQUIV' PY>Y(/B/) 'THEN'
           'BEGIN'   S:=PX-X(/A/)-(PY-Y(/A/))/(Y(/B/)-Y(/A/))*(X(/B/)-X(/A/));
                'IF ABS(S)<'-6 'THEN'
                'BEGIN'  ON:='TRUE'
                      'GOTO' QUIT
                'END' 'ELSE' 'IF' S<0 'THEN' ON:='NOT' ON
           'END'
'END';
QUIT:    INSIDE:=ON
'END'INSIDE;
```

APPENDIX 5.1

The General Draw Program.

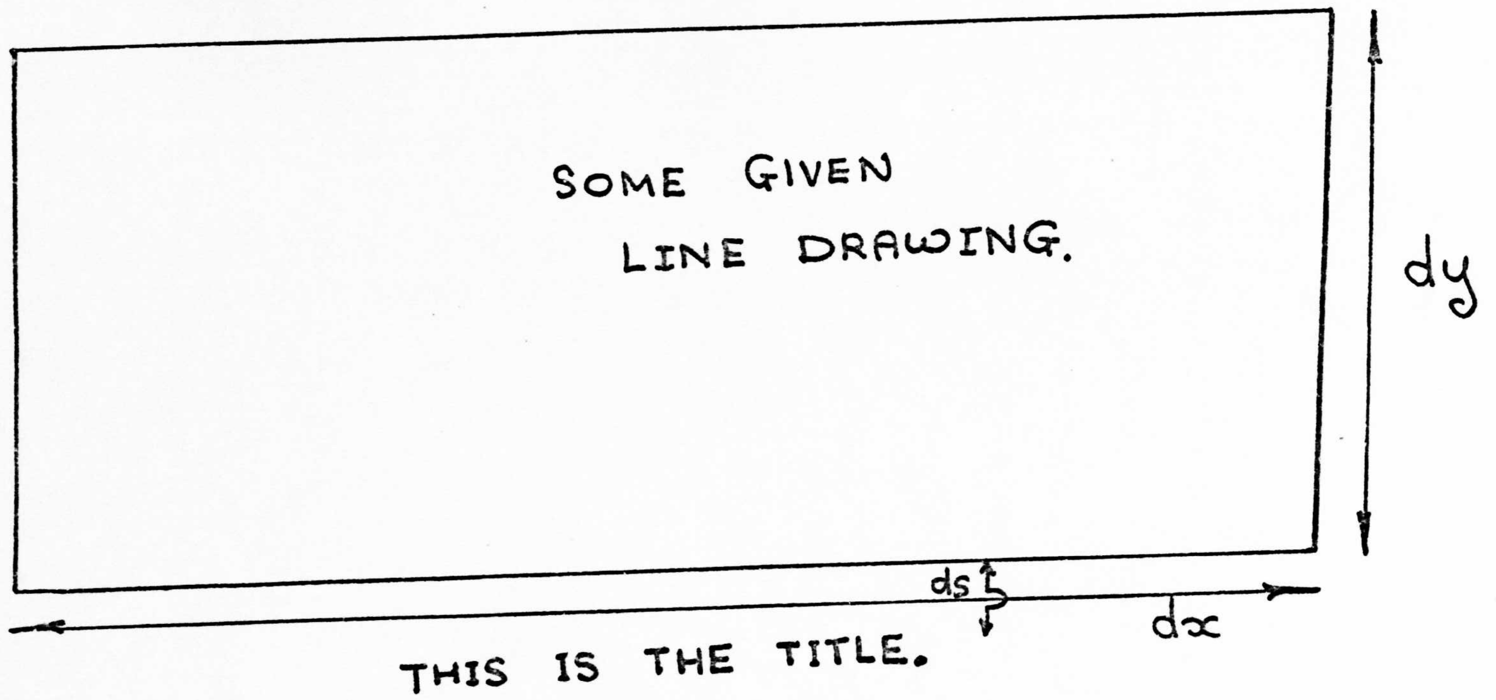
In Chapter 5 it was explained that it is possible to add pen up segments to any given line drawing to enable the line drawing to be drawn in a continuous sequence by the pen of the display.

A program has been written in Algol 60 which utilises the methods explained both to choose the pen up segments and sort the segments into an Eulerian cycle. Various facilities have been added to the program so that it is possible to scale the given line drawing to a given size in both the x and y directions of the cartesian co-ordinate system. It is possible to draw both full and dotted lines, the size of the dots being a variable of the program.

A procedure which was available to the author allowed given characters, read in from a data tape, to be drawn in a suitable form on the graph plotter. The characters which could be drawn on the plotter were KDF9 basic symbols. By this means it was possible to add a title to each line drawing, the title in each case being positioned so that it lay a given distance below the least y co-ordinate of the drawing and so that it was positioned central of the extreme x co-ordinate.

The size of the characters on the graph plotter could be varied in addition to the actual number of characters in the title.

The program is thus very general in that there are several variables which allow any given line drawing to be output on the graph plotter in any suitable form. The computer drawn figures presented in this thesis have been produced by use of this program. For some of the more complex drawings it was necessary to manually position the many parts of the drawing by using several stop instructions in the program which caused the paper tape being read by the photo-electric reader to stop so that the pen could be moved to a suitable position by use of the manual control.



n = number of characters
in title (18 in this case)
h = height of characters

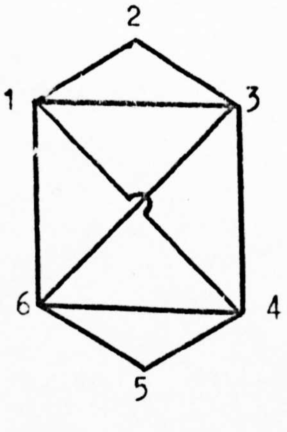
- n, h, ds, dx and dy are all variables of DRAW program
- title automatically positioned in centre of drawing

APPENDIX 5.2

Description of Procedure 'Loop'.

Given an Eulerian cycle the procedure loop traces a path through the cycle such that every segment is traversed once and once only. The directions associated with each of the segments in this path are such that the cycle is a directed Eulerian cycle.

It is convenient at this stage to give a simple example of the way in which the procedure loop, by list processing technique, forms a path through the given Eulerian cycle.



Consider the Eulerian cycle, shown above and for each node determine the adjacent nodes (neighbours) as shown below

Node	Neighbours	Degree of Node
1	2,3,4,6	4
2	1,3	2
3	1,2,4,6	4
4	1,3,5,6	4
5	4,6	2
6	1,3,4,5	4

Note that the degree of each node is even.

It is an arbitrary choice as to which node is taken as the start of the cycle.

Suppose node 1 is chosen as the initial node. The next node in the path will simply be selected as the first available neighbour. In this case 2 is the first available neighbour. The path so far is thus 1-2. After selecting a node is is necessary to cancel this node in both lists since node 2 is a neighbour of node 1 and node 1 a neighbour of node 2. Thus the first available neighbour for node 2 will now be node 3.

Continue building this path until node 1 (the initial node) is reached when the path will form an initial Eulerian cycle.

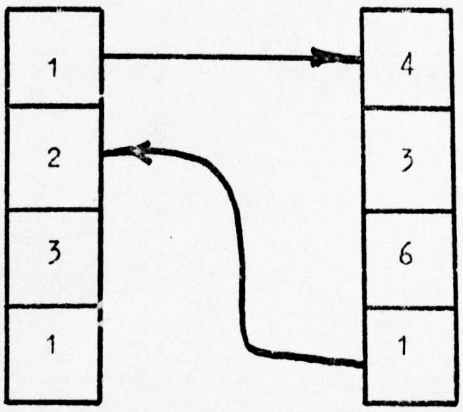
The initial cycle will thus be

1-2-3-1

Now scan every node of this cycle until a node is found to which some neighbours still exist. When a node has been found trace a secondary cycle from this node. In this particular case node 1 still has some neighbours so that a secondary cycle is traced from node 1 as follows:

1-4-3-6-1

The cycle at this point consists of two cycles joined together as a list as illustrated below



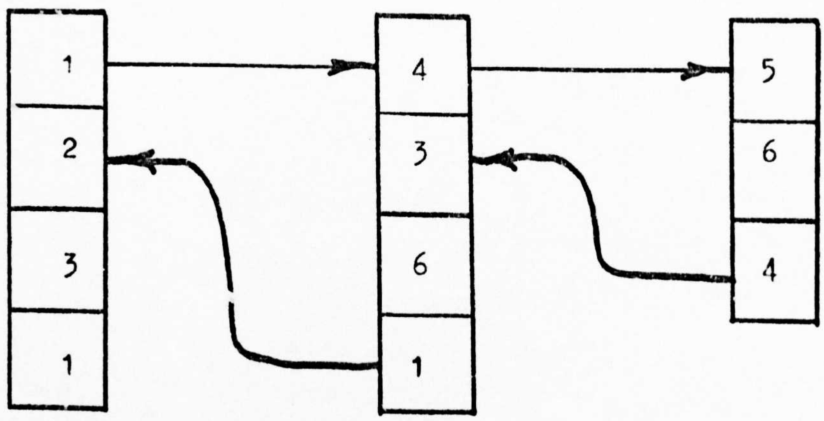
The cycle at present is thus

1-4-3-6-1-2-3-1

Now scan each node of the secondary cycle until a node is located which still has some neighbours associated with it. In this case node 4 still has some neighbours and so a further cycle is traced from this node as follows

4-5-6-4

The cycle at this point consists of three separate cycles joined together as a list as illustrated below

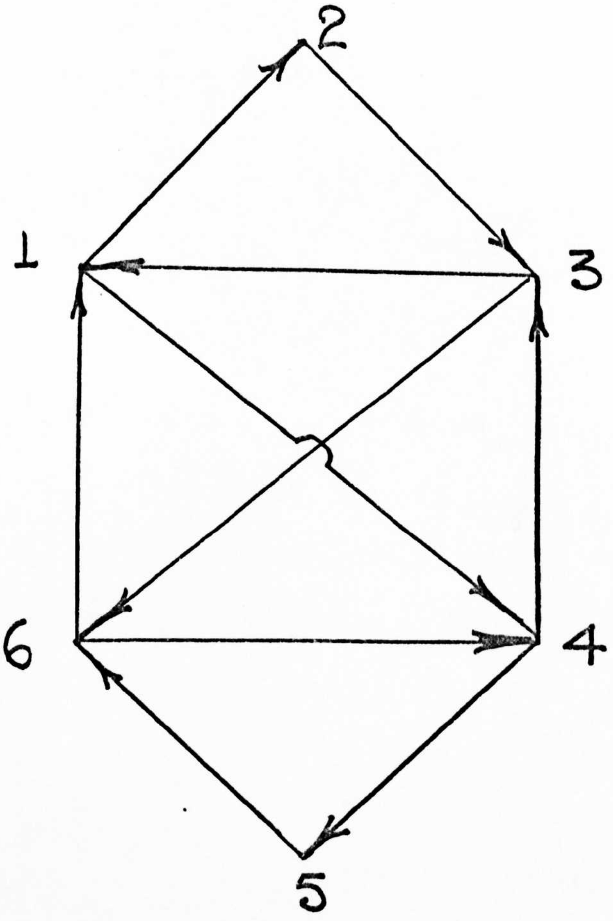


At this point there are no nodes which have neighbours still remaining and so the list is complete.

The Eulerian cycle is thus

1-4-5-6-4-3-6-1-2-3-1

and the directions associated with each of the segments are shown below


2
1
3
6
4
5

APPENDIX 5.3

Method of Determining whether a Given Graph is Connected.

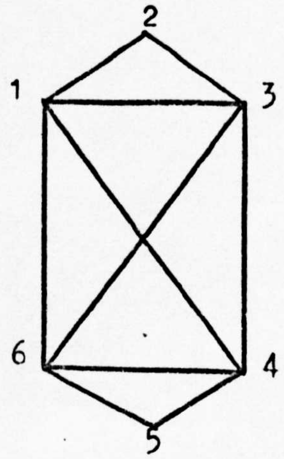
A convenient method of determining whether a given graph is connected has been derived from procedure loop, described in the previous Appendix.

The only difference existing between this procedure and loop is that the cycles traced out for each of the secondary cycles are replaced in this case by paths and the terminating condition for each path is when a selected node has no neighbours remaining.

As each node is encountered a flag is set and if the flags associated with all the nodes of the graph have been set then the graph is connected. If it is not possible to extend any of the secondary (or the initial) paths and all flags have not been set then the graph cannot be connected.

Example of Connected Graph.

Consider the graph given in the previous appendix since this will serve to outline the difference in the two algorithms.



Node	Neighbour
1	2,3,4,6
2	1,3
3	1,2,4,6
4	1,3,5,6
5	4,6
6	1,3,4,5

As before trace out an initial path but in this case continue until it is not possible to extend the path. Every time a node is located set an associated boolean from its initial false value to two. The initial path will be

1-2-3-1-4-3-6-1

There are no more nodes associated with node 1 and so the path terminates. At the end of this initial path only node 5 has not been traversed. Now attempt to extend the path by scanning each node to locate a node which has neighbours still associated with it. If one exists then form a path (in the same way as for the initial path) from it.

In this case node 4 still has some neighbours associated with it. The path from 4 will thus be

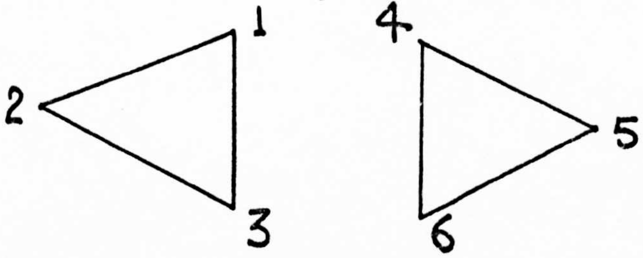
4-5-4-6-5

and the path terminates since node 5 has no neighbours.

Node 5 has now been traversed so that all nodes have been flagged and the graph is thus connected.

Example of Graph which is not Connected.

Consider the graph given below



Node	Neighbours
1	2,3
2	1,3
3	1,2
4	5,6
5	4,6
6	4,5

The initial path will be

1-2-3-1

and the path terminates since 1 has no neighbours left.

Now scane this path to ascertain which of the nodes still has neighbours associated with it. There are none and since only nodes 1,2 and 3 have been flagged the graph cannot be connected.

APPENDIX 5.4

Program Listings

These are the program listings concerned with the main DRAW program. Procedure 'pen up' uses the heuristic method to obtain the pen up segments.

THIS IS THE MAIN CODING REQUIRED IN THE 'DRAW' PROGRAM.
THE DECLARATIONS OF VARIABLES HAVE BEEN OMITTED.

```
OPEN(20); OPEN(30); OPENGP;
'FOR' I:=1 'STEP' 1 'UNTIL' A 'DO' 'FOR' J:=1,2,3 'DO' LIST(/I,J/):=READ(20);
'COMMENT' READ IN VALUES OF COORDINATES;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'BEGIN' X(/I/):=READ(20); Y(/I/):=READ(20) 'END';
'COMMENT' FIND EXTREME VALUES FOR X AND Y COORDINATES;
XMIN:=XMAX:=X(/1/); YMIN:=YMAX:=Y/1/);
'FOR' I:=2 'STEP' 1 'UNTIL' N 'DO'
'BEGIN'   'IF' X(/I/)<XMIN 'THEN' 'BEGIN' E:=I; XMIN:=X(/I/) 'END'
          'ELSE' 'IF' X(/I/)>XMAX 'THEN' XMAX:=X(/I/);
          'IF' Y(/I/)<YMIN 'THEN' YMIN:=Y(/I/) 'ELSE'
          'IF' Y(/I/)>YMAX 'THEN' YMAX:=Y(/I/)
'END';
'COMMENT' CALCULATE REQUIRED SCALING FACTORS SX AND SY;
SX:=SX/(XMAX-XMIN); SY:=SY/(YMAX-YMIN);
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'BEGIN' X(/I/):=X(/I/)*SX; Y(/I/):=Y(/I/)*S 'END';
S:=SX*(XMAX-XMIN); B:=0;
PLOT(0,9);
D:=(SC*L*5-S)/2;
'COMMENT' MOVE IN X AND Y DIRECTIONS TO AVOID EXCEEDING THE LIMITS;
'IF' D>0 'THEN' LINE(0,0,D,0);
```

REMAINDER OF BODY OF DRAW PROGRAM

```
'IF' (Y(/E/)-SY*YMIN)>0 'THEN' LINE(0,0,0,(Y(/E/)-SY*YMIN));
LINEDRAW(LIST,A,X,Y,E,N,PAIRS,UP,NEXT);
PLOT(0,10);
'COMMENT' PLOT LINES ACCORDING TO VALUE OF LIST(/I,3/) IN EACH CASE;
'FOR' I:=1,NEXT(/I/) 'WHILE' NEXT(/I/)>0 'DO'
'BEGIN' G:=PAIRS(/I/); K:=PAIRS(/NEXT(/I/)/); J:=UP(/NEXT(/I/)/);
     'IF' J>0 'THEN'
          'BEGIN' 'IF' J=1 'THEN' LINE((/G/),Y(/G/),X(/K/),Y(/K/)) 'ELSE'
                'BEGIN' PLOT(0,9); LINE(X(/G/),Y(/G/),X(/K/),Y(/K/));
                                         PLOT(0,10)
                'END'
          'END' 'ELSE' LINEDOT(X(/G/),Y(/G/),X(/K/),Y(/K/))
'END';
'COMMENT' MOVE TO SUITABLE POSITION AND THEN PRINT OUT TITLE;
PLOT(0,9); LINE(0,0,-B-D,-(Y(/E/)-SY*YMIN)-DOWN-14*SC);
GPBASICSYMBOL(-1,5);
J:=INBASICSYMBOL(20);
'FOR' J:=1 'STEP' 1 'UNTIL' L 'DO' GPBASICSYMBOL(INBASICSYMBOL(20),SC);
'COMMENT' INBASICSYMBOL READS IN A CHARACTER AS DATA AND GPBASICSYMBOL
REPRODUCES THIS CHARACTER AS PUNCHED PAPER TAPE CHARACTERS FOR INPUT
TO THE GRAPH PLOTTER;
CLOSE(20); close(30); CLOSEGP
'END;
```

THE FOLLOWING IS A LISTING OF THE LINEDRAW PROCEDURE WITH LOCAL PROCEDURES DEFINED BY THEIR HEADINGS ONLY

```
'PROCEDURE' LINEDRAW(STORE,A,X,Y,D,N,SEQ,UP,NEXT);
'VALUE' D; 'INTEGER' A,D,N; 'ARRAY' X,Y; 'INTEGER' 'ARRAY' STORE,SEQ,UP,NEXT;
'COMMENT' PROCEDURE ACCEPTS A NUMBER OF LINE SEGMENTS AND FORMS A DIRECTED
EULERIAN CYCLE BY THE ADDITION OF A NUMBER OF PEN UP SEGMENTS;
'BEGIN' 'INTEGER' ODDN,I,J; 'REAL' M,SIZE;
'PROCEDURE' COINC;
'PROCEDURE' ODDPOINTS;
'PROCEDURE' PENUP;
'PROCEDURE' LOOP;
COINC(STORE,X,Y,N,A,0.1);
ODDPOINTS(STORE,A,N,ODD,ODDN);
'IF' ODDN>0 'THEN' PENUP(ODD,ODDN,SIZE,A,STORE);
LOOP(D,A,N,STORE,SEQ,UP)
'END'LINEDRAW;
```

```
'PROCEDURE' PENUP(ODD,ODDN,S,A,STORE);
'VALUE' ODDN; 'INTEGER' ODDN,A; 'REAL' S; 'INTEGER' 'ARRAY' ODD,STORE;
'COMMENT' PROCEDURE SELECTS THE PAIRS OF ODD DEGREE POINTS FROM THE ODD
POINTS STORED IN THE ARRAY ODD. ODDN/2 PAIRS ARE SELECTED. THE PAIRS ARE
ADDED TO THE ORIGINAL PAIRS LIST IN THE ARRAY STORE. THE TOTAL COST OF THESE
ADDED SEGMENTS IS ASSIGNED TO THE VARIABLE S;
'BEGIN' 'ARRAY' MAT(/1:ODDN,1:ODDN/),INFORM(/1:ODDN,1:4/);
'PROCEDURE' SETUP;
'PROCEDURE' SELECT;
'PROCEDURE' RESET;
'REAL' 'PROCEDURE' DIS(I,J);
'VALUE' I,J; 'INTEGER' I,J;
'COMMENT' CALCULATES DISTANCS BETWEEN TWO POINTS I AND J;
DIS:=(X(/I/)-X(/J/))**2+(Y(/I/)-Y(/J/))**2;
'INTEGER' I,Q,P,J,START,C,K,POINTER;
'INTEGER' 'ARRAY' LIST(/1:ODDN,1:2/);
'COMMENT' SET UP THE COST MATRIX OF DISTANCES BETWEEN THE ODD DEGREE POINTS;
'FOR' I:=1 'STEP' 1 'UNTIL' ODDN-1 'DO' 'FOR' J:=U+1 'STEP' 1 'UNTIL' ODDN 'DO'
MAT(/I,J/):=MAT(/J,I/):=DIS(ODD(/I/),ODD(/J/));
'COMMENT' CHECK TO SEE IF ONLY TWO ODD POINTS;
'IF' ODDN=2 'THEN'
'BEGIN' A:=A+1; STORE(/A,1/):=ODD(/1/); STORE(/A,2/):=ODD(/2/);
       STORE(/A,3/):=2; S:=MAT(/1,2/);
       'GOTO' FIN
'END';
```

REMAINDER OF PEN UP PROCEDURE

```
'COMMENT' SET UP THE LIST PROCESSING ARRAY LIST FOR THE N ODD POINTS;
LIST(/1,1/):=ODDN; LIST(/1,2/):=2;
'FOR' I:=2 'STEP' 1 'UNTIL' ODDN-1 'DO'
'BEGIN' LIST(/I,1/):=I-1; LIST(/I,2/):=I+1 'END';
LIST(/ODDN,1/):=ODDN-1; LIST(/ODDN,2/):=1;
START:=1;
'COMMENT'SET UP THE INFORMATION ASSOCIATED WITH EACH POINT AND STORE IN THE
ARRAY INFORM;
'FOR' I:=1 'STEP' 1 'UNTIL' ODDN 'DO' SETUP(I);
C:=0; S:=0; K:=ODDN'/'2-1;
'COMMENT' SELECT PAIRS OF POINTS UNTIL THERE ARE ONLY TWO REMAINING.
THE PAIRS ARE STORED IN THE ARRAY STORE;
'FOR' I:=1,I+1 'WHILE' I¬=K 'DO'
'BEGIN' SELECT; S:=S+MAT(/P,POINTER/);
  A:=A+1; STORE(/A,1/):=ODD(/P/); STORE(/A,2/):=ODD(/POINTER/); STORE(/A,3/):=2;
  RESET
'END';
S:=S+MAT(/START,LIST(/START,2/)/);
A:=A+1; STORE(/A,1/):=ODD(/START/); STORE(/A,2/):=ODD(/LIST(/START,2/)/);
STORE(/A,3/):=2;
FIN:
'END'PENUP;
```

```
'PROCEDURE' RESET;
'COMMENT' PROCEDURE ERASES THE POINTS P AND POINTER (WHICH HAVE BEEN CHOSEN
BY THE PROCEDURE SELECT). IN ADDITION A CHECK IS CARRIED OUT TO DETERMINE
WHETHER IT IS NECESSARY TO SET UP THE NEAREST AND NEXT NEAREST POINTS FOR
ANY OF THE REMAINING POINTS;
'BEGIN' 'INTEGER' S,I,K;
'FOR' S:=P,POINTER 'DO'
'BEGIN' LIST(/LIST(/S,1/),2/):=LIST(/S,2/); LIST(/LIST(/S,2/),1/):=LIST(/S,1/);
   'IF' S=START 'THEN' START:=LIST(/S,2/)
'END';
'FOR' I:=START,LIST(/I,2/) 'WHILE' I¬=START 'DO' 'FOR' K:=3,4 'DO'
'IF' INFORM(/I,K/)=P 'THEN'
'BEGIN' SETUP(I); K:=5 'END' 'ELSE'
'IF' INFORM(/I,K/)=POINTER 'THEN'
'BEGIN' SETUP(I); K:=5 'END'
'END'RESET;
```

```
'PROCEDURE' SETUP(I);
'VALUE' I; 'INTEGER' I;
'COMMENT' PROCEDURE SETS UP FOR EACH POINT THE NEAREST AND NEXT NEAREST
POINTS AND THE DIFFERENCE IN DISTANCE. THE INFORMATION IS STORED IN THE
ARRAY INFORM;
'BEGIN' 'REAL' FIPST,NEXT;
FIRST:=NEXT:=1000000;
'FOR' J:=START,LIST(/J,2/) 'WHILE' J¬=START 'DO' 'IF' I¬=J 'THEN'
'BEGIN' 'IF' MAT(/I,J/)<FIPST 'THEN'
   'BEGIN' FIRST:=MAT(/I,J/); P:=J 'END' 'ELSE' 'IF' MAT(/I,J/)<NEXT 'THEN'
   'BEGIN' NEXT:=MAT(/I,J/); Q:=J 'END'
'END';
INFORM(/I,1/):=FIRST; INFORM(/I,2/):=NEXT-FIPST;
INFORM(/I,3/):=P; INFORM(/I,4/):=Q
'END'SETUP;
```

```
'PROCEDURE' SELECT;
'COMMENT' PROCEDURE SELECTS THAT POINT WHICH HAS THE MAXIMUM DISTANCE
BETWEEN THE NEAREST AND NEXT NEAREST POINTS;
'BEGIN' 'REAL' MAX;
MAX:=INFORM(/START,2/); POINTER:=START;
'FOR' I:=LIST(/START,2/),LIST(/I,2/) 'WHILE' I¬=START 'DO'
'IF' INFORM(/I,2/)>MAX 'THEN' 'BEGIN' MAX:=INFORM(/I,2/); POINTER:=I 'END';
P:=INFORM(/POINTER,3/); Q:=INFORM(/POINTER,4/)
'END'SELECT;
```

```
'PROCEDURE' ODDPOINTS(LIST,A,N,ODD,ODDN);
'VALUE' A,N; 'INTEGER' A,N,ODDN; 'INTEGER' 'ARRAY' LIST,ODD;
'COMMENT' PROCLDURE LOCATES THE ODD DECREE POINTS BY A COUNT OF THE NUMBER
OF TIMES A POINT OCCURS AS A TERMINAL POINT OF A LINE SEGMENT;
'BEGIN' 'INTEGER' I,J,JJ,C,D;
ODDN:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
   'BEGIN'  C:=0;
     'FOR' J:=1 'STEP' 1 'UNTIL' A 'DO' 'FOR' JJ:=1,2 'DO'
     'IF' LIST(/J,JJ/)=I 'THEN' C:=C+1;
     'IF' C'/'2*2¬=C 'THEN' 'BEGIN' ODDN:=ODDN+1; ODD(/ODDN/):=I 'END'
   'END'
'END'ODDPOINTS;
```

```
'PROCEDURE' LOOP(D,A,N,STORE,SEQ,UP);
'VALUE' D,A,N; 'INTEGER' D,A,N; 'INTEGER' 'ARRAY' STORE,SEQ,UP;
'COMMENT' PROCEDURE SORTS THE SEGMENTS CONTAINED IN THE ARRAY STORE INTO
AN EULERIAN CYCLE;
'BECIN' 'INTEGER' S,COUNT,G,B,I,SS,C;
'BOOLEAN' INITIAL;
'INTEGER' 'ARRAY' START,FINISH(/1:N+1/),TAB(/1:2*A,1:2/);


'PROCEDURE' CANCEL(J,K);
'VALUE' J,K; 'INTEGER' J,K;
'COMMENT' PROCEDURE CANCELS THE SEGMENT J-K IN THE TAB LIST BY ALTERATION
OF THE ASSOCIATED START AND FINISH POINTERS;
'BEGIN' 'INTEGER' I,II,L,D;
START(/J/):=START(/J/)+1; D:=FINISH(/K/);
'FOR' I:=START(/K/) 'STEP' 1 'UNTIL' D 'DO' 'IF' TAB(/I,1/)=J 'THEN'
'BEGIN'  FINISH(/K/):=D:=D-1;
     'FOR' II:=I 'STEP' 1 '1NTIL' D 'DO'
     'FOR' L:=1,2 'DO' TAB(/II,L/):=TAB(/II+1,L/);
     I:=D+1
'END'
'END'CANCEL;


'PROCEDURE' BUILD(K,J,I);
'VALUE' K,J,I; 'INTEGER' K,J,I;
'COMMENT' PROCEDURE STORES THE SEGMENTS IN THE TAB LIST;
'BEGIN' 'INTEGER' S,D,SS;
D:=START(/K+1/);
'FOR' S:=COUNT 'STEP' -1 'UNTIL' D 'DO'
'FOR' SS:=1,2 'DO' TAB(/S+1,SS/):=TAB(/S,SS/);
'FOR' S:=K+1 'STEP' 1 'UNTIL' N+1 'DO' START(/S/):=START(/S/)+1;
TAB(/D,1/):=J; TAB(/D,2/):=I;
COUNT:=COUNT+1
'END'BUILD;
```

PROCEDURE LOOP CONTINUED

```
'COMMENT' SET THE INITIAL VALUES FOR THE START POINTERS;
'FOR' I:=1 'STEP' 1 'UNTIL' N+1 'DO' START(/I/):=1;
COUNT:=0;
'COMMENT' BUILD UP THE TAB LIST;
'FOR' I:=1 'STEP' 1 'UNTIL' A 'DO'
'FOR' S:=1,2 'DO' BUILD(STORE(/I,S/),STORE(/I,3-S/),STORE(/I,3/));
'COMMENT' SET THE INITIAL FINISH POINTERS;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' FINISH(/I/):=START(/I+1/)-1;
'COMMENT' SET UP INITIAL VALUES FOR VARIABLES;
SEQ(/1/):=D; NEXT(/1/):=2; SS:=1; S:=D; C:=1; INITIAL:='TRUE';
'COMMENT' BEGIN TO FORM THE EULERIAN CYCLE;
BACK:
'FOR' B:=START(/S/),START(/I/) 'WHILE' S¬=: 'DO'
'BEGIN'  I:=TAB(/B,1/); CANCEL(S,I); S:=I;
         C:=C+1; SEQ(/C/):=I; UP(/C/):=TAB(/B,2/); NEXT(/C/):=C+1
'END';
'COMMENT' SET MARKER FOR FINISH OF INITIAL EULERIAN CYCLE;
'IF' INITIAL 'THEN'
'BEGIN' NEXT(/C/):=-2; INITIAL:='FALSE' 'END' 'ELSE' NEXT(/C/):=G;
'COMMENT' CHECK TO SEE IF ANY SEGMENTS NOT CONTAINED IN PRESENT CYCLE;
'FOR' J:=SS,NEXT(/J/) 'WHILE' J>0 'DO'
'BEGIN'   D:=SEQ(/J/);
     'IF' START(/D/)<=FINISH(/D/) 'THEN'
     'BEGIN' SS:=J; G:=NEXT(/SS/); NEXT(/SS/):=C+1;
          S:=D; 'GOTO' BACK
     'END'
'END'
'END'LOOP;
```

```
'PROCEDURE' COINC(LIST,X,Y,N,A,INC);
'VALUE' INC; 'INTEGER' A,N,INC; 'INTEGER' 'ARRAY' LIST; 'ARRAY' X,Y;
'COMMENT' PROCEDURE CHECKS FOR COINCIDENT POINTS .
A CHECK IS ALSO MADE TO ENSURE SEGMENTS DO NOT CONTAIN TWO COINCIDENT POINTS;
'BEGIN' 'INTEGER' I,J,S,SS,C,K;
'BOOLEAN' 'ARRAY' BOOL(/1:N/);
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' BOOL(/I/):='TRUE';
'COMMENT' CHECK FOR COINCIDENT POINTS;
C:=K:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' N-1 'DO' 'IF' BOOL(/I/) 'THEN'
'BEGIN' K:=K+1; X(/K/):=X(/I/); Y(/K/):=Y(/I/);
   'FOR' J:=I+1 'STEP' 1 'UNTIL' N 'DO'
   'IF' ABS(X(/I/)-X(/J/))>=INC 'THEN' 'ELSE' 'IF' ABS(Y(/I/)-Y(/J/))<INC 'THEN'
   'BEGIN' 'FOR' S:=1 'STEP' 1 'UNTIL' A 'DO' 'FOR' SS:=1,2 'DO'
     'IF' LIST(/S,SS/)=J 'THEN' LIST(/S,SS/):=K; BOOL(/J/):='FALSE'
   'END'
'END';
N:=K;
'COMMENT' CHECK FOR SEGMENTS CONSISTING OF 2 COINCIDENT POINTS AND
ELIMINATE THESE;
'FOR' I:=1 'STEP' 1 'UNTIL' A 'DO' 'IF' LIST(/I,1/)=LIST(/I,2/) 'THEN'
'BEGIN'  A:=A-1;
   'FOR' S:=I 'STEP' 1 'UNTIL' A 'DO'
   'FOR' SS:=1,2 'DO' LIST(/S,SS/):=LIST(/S+1,SS/)
'END'
'END'COINC;
```

APPENDIX 6.1

Proof that an Optimal Solution to the Pen Up Problem using the Reduced Cost Matrix is an Optimal Solution using the Original Cost Matrix.

The implicit enumeration algorithm, described in Section 6.13 uses the reduced cost matrix δC and it is therefore necessary, for the sake of completeness, to prove that an optimal solution using this matrix is also an optimal solution using the original cost matrix C.

It is also of interest to determine the relationship between the costs in the two cases.

Now $\delta Cij = Cij - Ui - Uj \qquad \begin{matrix} i \in P \\ j \in Q \end{matrix}$ ________(1)

where P and Q are disjoint sets and $P \cup Q = S$, the total set of elements.

The optimal pen up problem is to minimise the total cost σ given by

$$\sigma = \sum_{\substack{i \in P \\ j \in Q}} Cij.\ Xij$$

where $Xij \in X$ is a feasible solution to the pen up problem. Consider a feasible solution X using the original cost matrix C with corresponding cost σ'

$$\sigma' = \sum_{\substack{i \in P \\ j \in Q}} (\delta Cij + Ui + Uj).\ Xij$$

$$= \sum_{\substack{i \in P \\ j \in Q}} \delta Cij.\ Xij + \sum_{\substack{i \in P \\ j \in Q}} (Ui + Uj).\ Xij$$

Now suppose σ_r' is the total cost using the reduced cost matrix so that

$$\sigma r' = \sum_{\substack{i \in P \\ j \in Q}} \delta Cij.\ Xij \qquad \text{\_\_\_\_\_\_\_\_\_\_\_\_\_\_(2)}$$

Then $$\mathcal{C}' = \mathcal{C}r' + \sum_{\substack{i \in P \\ j \in Q}} (Ui + Uj).\ Xij$$

$$= \mathcal{C}r' + \underset{i \in P}{Ui} + \underset{j \in Q}{Uj}$$

Now $$\underset{i \in P}{Ui} + \underset{j \in Q}{Uj} = \underset{k \in S}{Uk}$$

where S = P∪Q, the total set of nodes.

$$\therefore \quad \mathcal{C}' = \mathcal{C}r' + \sum_{k \in s} Uk \qquad \_\_\_\_\_\_\_\_\_\_ (3)$$

Now for a given feasible solution $\sum_{k \in s} Uk$ is constant and simply the sum of the implicit costs

Thus it follows that if x is an optimal solution using the reduced cost matrix δC, it must also be optimal for the original matrix C since the total costs always differ by a constant value.

The relationship between the two costs is given by equation (3).

APPENDIX 6.2

Nested For Loop.

The nested for loop used to obtain elements of set P in the implicit enumeration algorithm described in Chapter 6 is a modification of a nested for loop developed by Dr. Scoins at the Laboratory.

The elements of set P are determined such that the following relationships are true

$$Pi + 1 > Pi \qquad i = 1,2,3.....s-1 \qquad (1)$$

$$Qi > Pi \qquad i = 1,2.......n \qquad (2)$$

where s = n/2 and Q is the disjoint set.

In the for loop actually used $Pi + 1 > Pi$ simply for convenience and ease of coding. The basic principles to be discussed remain the same. For the kth element determined ($k < s$) Pk must be such that (s-k) elements are available and greater in value than Pk so that a complete set P can be obtained while equation (1) is satisfied.

There must thus be an upper bound for each element Pk and this must be dependent on the value of k. Since the first element obtained for P is Ps the upper bound for this is 1. The upper bound for P_1 is n - 1 since n must be available for the corresponding element in the disjoint set Q and

$$Q_1 > P_1$$

Similarly the upper bound for P_2 must be n-2. Thus it is possible for each of the elements of P to obtain a corresponding upper bound. This corresponds to an element of a one dimensional array ub(i) $i = 1,2......s$

The nested for loop is thus as follows

```
for P(n)=P(n)+1 step 1 until ub(n) do
if n=1 then GETQ else fornest (n-1)
```

When n (in the nested for loop) reaches the value 1 a feasible set of s elements exists for P and procedure GETQ (which is also recursive) is invoked. GETQ simply locates sets of Q which satisfy equation (2) and whose pairings with P form a feasible solution to the pen up problem. If n is not 1 then the for nest procedure is invoked within itself (recursion).

Although this method furnishes a classical way of determining elements of P and Q the number of procedure calls will be quite large and the object program when executed will probably be inefficient because of this.

It would probably have been better if the elements of P had been obtained by an iterative procedure which the compiler would handle in a much more efficient manner.

Unfortunately, time did not allow the author to investigate this particularly interesting possibility.

The advantage of utilising a nested for loop lies in the ease of the coding without regard to compiler efficiency.

APPENDIX 6.3

Implicit Enumeration Program Specification.

Abstract.

The program is written in Algol 60 and it determines an optimal solution to the pen up problem; the elements of the cost matrix are obtained from a random number generator. There are various forms of the program depending on the enumerative scheme being used. (See Chapter 6).

Input.

- Number of nodes in odd degree graph.
- Various other items of data may be input for the various enumerative schemes such as the size of the largest element.

Output.

- Lower bound cost.
- Upper bound cost.
- Elements of the cost matrix.
- Each best feasible solution as it is obtained and the cost of this solution.
- Optimal cost.
- C.P.U. time taken.

NOTE

In most cases the total number of subsets and the number of subsets which need further examination are also output. Other items of information pertaining to the enumerative scheme being used may also be output.

APPENDIX 6.4

Program Listings

The following program listings are related to the general implicit enumeration algorithm explained in Chapter 6.

THE FOLLOWING IS A LISTING OF THE GENERAL IMPLICIT ENUMERATION ALGORITHM.

```
'BEGIN'
'INTEGER' N,UP,CP,CPP,I,J,C,OPT,S,T,K,SUM,PP,SS,DELTA,E,CC,BB;
'REAL' X,TI;
'COMMENT' THE FOLLOWING ARE THREE CODED PROCEDURES USED IN THE PROGRAM;
'PROCEDURE' RANDOM(X); 'REAL' X; 'CODE';
'PROCEDURE' CPUT(A); 'INTEGER' A; 'CODE';
'PROCEDURE' PRINT(D,F,N; 'INTEGER' D,F,N; 'CODE';
OUTSTRING(1,'('ENTER NUMBER OF NODES IN ODD DEGREE GRAPH')');
SYSACT(1,2,1);
ININTEGER(0,N);
S:=N'/'2;
'COMMENT' CAN NOW DECLARE SOME ARRAYS;
'BEGIN' 'INTEGER' 'ARRAY' II,LIST,TOTAL,R,CD,UB(/1:N/),EE(/1:N+1/),
          MAT(/1:N,1:N/),TABLE(/1:(N-1)*S/);
'BOOLEAN' 'ARRAY' BOOL(/0:N/);
'PROCEDURE' PRINTOUT;
'PROCEDURE' FORNEST;
'INTEGER' 'PROCEDURE' ASSIGN;
'INTEGER' 'PROCEDURE' LOWBOUND;
'PROCEDURE' GETQ;
'PROCEDURE' PRINTMAT;
'PROCEDURE' IMPROVE;
'COMMENT' DETERMINE ELEMENTS OF THE COST MATRIX MAT;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'FOR' J:=1 'STEP' 1 'UNTIL' I-1 'DO'
'BEGIN' RANDOM(X); MAT(/I,J/):=MAT(/J,I/):=100*X 'END';
'COMMENT' FIRST CALL OF THE C.P.U.  TIMER;
CPUT(CP);
'COMMENT' OBTAIN A LOW BOUND COST TO PEN UP PROBLEM;
K:=LOWBOUND(N,CD,MAT);
IMPROVE(MAT,CD,N,K);
OUTSTRING(1,'('LOWER BOUND')'); PRINT(1,-5.0,K);
SYSACT(1,2,1);
```

REMAINDER OF IMPLICIT ENUMERATION ALGORITHM.

```
UP:=UPPERBOUND(MAT,CD,N,SIZE)+K;
PRINT(1,-2.0,UP); SYSACT(1,2,1);
OUTSTRING(1,'('UPPER BOUND')');
'COMMENT' PRINT OUT THE COST MATRIX;
OUTSTRING(1,'('COST MATRIX')'); SYSACT(1,2,1);
PRINTMAT(MAT);
'COMMENT' SET UPPER BOUND VALUES FOR NESTED FOR LOOP;
C:=0; UB(/S/):=1; UB(/1/):=N-1;
'FOR' J:=2 'STEP' 1 'UNTIL' S-1 'DO' UB(/J/):=UB(/J-1/)-1;
'COMMENT' ARRANGE ELEMENTS IN TABLE FOR USE IN NESTED FOR LOOP;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'BEGIN' 'FOR' J:=I+1 'STEP' 1 'UNTIL' N 'DO'
      'BEGIN'  C:=C+1;
               TABLE(/C/):=J
      'END';
   EE(/I+1/):=C+1
'END';
EE(/1/):=1;
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' BOOL(/J/):='TRUE';
II(/S+1/):=TOTAL(/S+1/):=0;
'COMMENT' CALL NESTED FOR LOOP  'FORNEST';
OPT:=0;'IF' OPT+1>K 'THEN' FORNEST(S);
'COMMENT' SECOND CALL OF C.P.U. TIMER;
CPUT(CPP);
SYSACT(1,14,3);
OUTSTRING(1,'('OPTIMAL COST')'); PRINT(1,-5.0,OPT+1);
SYSACT(1,2,1);
OUTSTRING(1,'('CPU TIME')'); PRINT(1,-5.3,625/48000000*(CPP-CP));
OUTSTRING(1,'('   SECS')');
SYSACT(1,2,1);
'END'
'END';
```

THE FOLLOWING ARE 2 PROCEDURES USED TO PRINT OUT INFORMATION DURING THE EXECUTION OF THE IMPLICIT ENUMERATION ALGORITHM.

```
'PROCEDURE' PRINTOUT;
'COMMENT' PROCEDURE PRINTS OUT THE BEST FEASIBLE SOLUTION FOUND SO FAR
AND STORES THIS SOLUTION IN THE TWO DIMENSIONAL ARRAY LIST. THE FEASIBLE COST
IS PRINTED AND A CHECK IS MADE TO ASCERTAIN WHETHER AN OPTIMAL SOLUTION
HAS BEEN FOUND. IF SO THE ALGORITHM TERMINATES OTHERWISE THE BEST FEASIBLE COST,
BEST,IS SET ONE LESS THAN THE FEASIBLE COST;
'BEGIN' 'INTEGER' I;
'FOR' I:=S 'STEP' -1 'UNTIL' 1 'DO'
'BEGIN' LIST(/J,1/):=P(/J/); LIST(/J,2/):=Q(/J/);
  PRINT(1,-3.0,P(/J/)); PRINT(1,-2.0,Q(/J/))
'END';
SYSACT(1,2,1);
'IF' TOTAL(/1/)=OPTIMAL 'THEN'
'BEGIN'  BEST:=OPTIMAL-1;
  'FOR' J:=1 'STEP' 1 'UNTIL' S 'DO' P(/J/):=UB(/J/)+1
'END' 'ELSE' BEST:=TOTAL(/1/)-1
'END'PRINTOUT;

'PROCEDURE' PRINTMAT(MAT);
'INTEGER' 'ARRAY' MAT;
'COMMENT' PROCEDURE PRINTS OUT THE SYMMETRIC COST MATRIX MAT INSERTING AN
ASTERICK ALONG THE MAIN DIAGONAL;
'BEGIN' 'INTEGER' I,J;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'BEGIN' 'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
  'IF' I=J 'THEN' OUTSTRING(1,'('   *')') 'ELSE' PRINT(1,-2.0,MAT(/I,J/));
  SYSACT(1,2,1)
'END'
'END'PRINTMAT;
```

THIS PROCEDURE IS INVOKED BY 'GETQ' AND IS A RECURSIVE PROCEDURE. IT OBTAINS FEASIBLE MATCHINGS OF SET Q WITH SET P SUCH THAT qi IS GREATER THAN pi. (i=1,2.....s).

```
'PROCEDURE' RECUR(I,P,Q);
'VALUE' I; 'INTEGER' I; 'INTEGER' 'ARRAY' P,Q;
'COMMENT' RECURSIVE PROCEDURE WHICH FINDS FEASIBLE SOLUTIONS TO THE
PEN UP PROBLEM BY MATCHING ELEMENTS OF P WITH THOSE OF Q;
'BEGIN' 'INTEGER' Z,K,L,J;
L:=EE(/P(/I/)+1/)-1;
'FOR' K:=EE(/P(/I/)/) 'STEP' 1 'UNTIL' L 'DO'
'BEGIN' Z:=TABLE(/K/);
  'IF' BOOL(/Z/) 'THEN'
  'BEGIN' 'FOR' J:=S 'STEP' -1 'UNTIL' I+1 'DO'
    'IF' Z=Q(/J/) 'THEN' 'GOTO' QUIT;
    TOTAL(/I/):=TOTAL(/I+1/)+MAT(/P(/I/),Z/); Q(/I/):=Z;
    'IF' TOTAL(/I/)>BEST 'THEN' 'ELSE'
    'IF' I=1 'THEN' PRINTOUT 'ELSE' RECUR(I-1,P,Q)
'END';
QUIT:'END'
'END'RECUR;
```

FORNEST IS A NESTED 'FOR' LOOP WHICH OBTAINS A SET 'P' OF ELEMENTS.
GETQ IS THEN CALLED TO OBTAIN THE CORRESPONDING SET 'Q' OF ELEMENTS.

```
'PROCEDURE' GETQ;
'COMMENT' PROCEDURE' DETERMINES THE ELEMENTS OF SET Q AFTER THE SET P OF
ELEMENTS HAS BEEN DETERMINED BY THE RECURSIVE FOR LOOP. CONTROL IS THEN
PASSED TO PROCEDURE ASSIGN WHICH DETERMINES A LOWER BOUND TO THE PEN UP
PROBLEM FOR SETS P AND Q.
IF THIS BOUND IS LOWER THAN THE BEST EXISTING SOLUTION CONTROL IS PASSED TO
THE RECURSIVE PROCEDURE RECUR WHICH LOCATES FEASIBLE SOLUTIONS TO THE PEN
UP PROBLEM BY MATCHING ELEMENTS OF P WITH THOSE OF Q;
'BEGIN' 'INTEGER' I,J;
'FOR' I:=1 'STEP' 1 'UNTIL' S 'DO' BOOL(/P(/I/)/):='FALSE';
J:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' BOOL(/I/) 'THEN'
  'BEGIN' J:=J+1; Q(/J/):=I 'END';
'IF' ASSIGN(S,P,Q)<=BEST 'THEN' RECUR(S,P,Q);
'FOR' I:=1 'STEP' 1 'UNTIL' S 'DO' BOOL(/P(/I/)/):='TRUE'
'END'GETQ;

'PROCEDURE' FORNEST(N);
'VALUE' N; 'INTEGER' N;
'COMMENT' A RECURSIVE FOR LOOP WHICH DETERMINES THE ELEMENTS OF THE SET P;
'FOR' P(/N/):=P(/N+1/)+1 'STEP' 1 'UNTIL' UB(/N/) 'DO'
'IF' N=1 'THEN' GETQ 'ELSE' FORNEST(N-1);
```

```
'INTEGER' 'PROCEDURE' ASSIGN (S,P,Q);
'VALUE' S; 'INTEGER' S; 'INTEGER' 'ARRAY' P,Q;
'COMMENT' ASSIGN OBTAINS THE COST OF A FEASIBLE SOLUTION TO THE DUAL
ASSIGNMENT PROBLEM OF ORDER S AND WITH DISJOINT SETS P AND Q BY
INTRODUCING AT LEAST ONE ZERO IN EACH OF THE ROWS AND COLUMNS OF THE
REDUCED COST MATRIX;
'BEGIN' 'INTEGER' MIN,I,J,K,T,B;
'BOOLEAN' 'ARRAY' D(/1:S/); 'INTEGER' 'ARRAY' R(/1:S/);
'FOR' I:=1 'STEP' 1 'UNTIL' S 'DO' D(/I/):='TRUE';
K:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' S 'DO'
'BEGIN'  B:=P(/I/);  MIN:=MAT(/B,Q(/1/)/);
   'FOR'  J:=2 'STEP' 1 'UNTIL' S 'DO'
   'IF' MAT(/B,Q(/J/)/) <MIN 'THEN' MIN:=MAT(/B,Q(/J/)/);
   'FOR'  J:=1 'STEP' 1 'UNTIL' S 'DO' 'IF' MAT(/B,Q(/J/)/)=MIN
   'THEN'  D(/J/):='FALSE';
   R(/I/):=MIN; K:=K+MIN
'END';
'FOR'  I:=1 'STEP' 1 'UNTIL' S 'DO' 'IF' D(/I/) 'THEN'
'BEGIN'  B:=Q(/I/); MIN:=MAT(/P(/1/),B/)-R(/1/);
   'FOR'  J:=2 'STEP' 1 'UNTIL' S 'DO'
   'BEGIN'  T:=MAT(/P(/J/),B/)-R(/J/);
      'IF' T<MIN 'THEN' MIN:=T
   'END';
   K:=K+MIN
'END';
ASSIGN:=K
'END' ASSIGN;
```

```
'PROCEDURE' IMPROVE(MAT,COL,N,E);
'VALUE' N; 'INTEGER' N; 'REAL' E; 'INTEGER' 'ARRAY' MAT,COL;
'COMMENT' THIS PROCEDURE IMPROVES THE COST OF THE INITIAL FEASIBLE SOLUTION
TO THE IMAGE PEN UP PROBLEM (OBTAINED BY THE LOWBOUND PROCEDURE) BY
ADDING SEGMENTS OF SMALL COST TO THE IMAGE GRAPH UNTIL IT IS
OF DOMINO FORM;
'BEGIN' 'INTEGER' I,M,A,G,J 'REAL' K,L;
'INTEGER' 'ARRAY'  TREE(/1:N,1:3/);
'PROCEDURE' LEVEL (K,S);
'VALUE' K,S; 'INTEGER' K,S;
'BEGIN' 'INTEGER' J,C;
'COMMENT' RECURSIVE PROCEDURE WHICH LOCATES A TREE IN THE IMAGE GRAPH
IF ONE EXISTS;
A:=-A;
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
'IF' J=S 'THEN' 'ELSE'
'IF' MAT(/K,J/)-COL(/K/)-COL(/J/)=0 'THEN'
'BEGIN' 'IF' TREE(/J,1/)=1 'THEN' NEWM;
   TREE(/J,3/):=TREE(/J,1/):=1; TREE(/J,2/):=A;
   LEVEL(J,K);
   A:=-A
'END'
'END' LEVEL;


'REAL' 'PROCEDURE' DELTA (S);
'VALUE' S; 'INTEGER' S;
'COMMENT' DETERMINES THE IMPROVEMENT IN COST OF THE FEASIBLE SOLUTION
TO THE IMAGE PEN UP PROBLEM;
'BEGIN' 'INTEGER' C,MIN,I,B;
MIN:=E; B:=S+1;
'FOR'I:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' TREE(/I,2/)=1 'THEN'
'FOR'  J:=S*I+1 'STEP' 1 'UNTIL' N 'DO'
'IF' TREE(/J,B/)¬=S 'THEN' 'ELSE' 'IF' I¬=J 'THEN'
'BEGIN'  C:=MAT(/I,J/)-COL(/I/)-COL(/J/);
         'IF' C<MIN 'THEN' MIN:=C
'END';
DELTA:=MIN
'END' DELTA;
```

CONTINUATION OF PROCEDURE 'IMPROVE'.

```
'PROCEDURE' INCREMENT;
'BEGIN' 'INTEGER' I;
'COMMENT' RELABELS NODES SO THAT NUMBER OF POSITIVELY LABELLED NODES
IN TREE IS GREATER THAN NUMBER OF NEGATIVELY LABELLED. IF EQUAL IN NUMBER,
NEW TREE IS SET UP;
G:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' G:=G+TREE(/I,2/);
'IF' G<0 'THEN'
'BEGIN' 'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' TREE(/I,2/):=-TREE(/I,2/)
'END' 'ELSE' 'IF' G=0 'THEN' NEWM
'END' INCREMENT;

'PROCEDURE' NEWM;
'BEGIN' 'COMMENT' FINDS ROOT M OF NEW TREE WHICH IS TO BE SET UP.
IF NO NEW ROOT AVAILABLE PROCEDURE TERMINATES;
'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' J=M 'THEN' 'ELSE'
'IF' TREE(/J,3/)=0 'THEN'
'BEGIN' M:=J; J:=N+1; 'GOTO' AGAIN 'END';
   'GOTO' OUT
'END' NEWM;

AGAIN:   'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'FOR' J:=1,2 'DO' TREE(/I,J/):=0;
M:=A:=1;
'FOR' J:=1,2,3 'DO' TREE(/M,J/):=1;
LEVEL(M,M);
INCREMENT;
L:=DELTA(1)/2; K:=DELTA(0); 'IF' L<K 'THEN' K:=L;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
COL(/I/):=COL(/I/)+TREE(/I,2/)*K;
E:=E+G*K;
'FOR'  I:=1 'STEP' 1 'UNTIL' N 'DO' TREE (/I,3/):=0;
'GOTO' AGAIN;
OUT:
'END' IMPROVE;
```

THE FOLLOWING IS THE CODING REQUIRED IN THE GENERAL IMPLICIT ENUMERATION ALGORITHM TO REDUCE THE NUMBER OF PEN UP DISTANCES

```
EE(/1/):=1; C:=0;
'COMMENT' THE UPPER BOUND COST OF THE PEN UP DISTANCES IS GIVEN BY 'UP';
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'BEGIN'  SS:=0;
      'FOR' J:=I+1 'STEP' 1 'UNTIL' N 'DO'
      'IF' MAT(/I,J/)<UP 'THEN'
       'BEGIN'  C:VC+1; SS:=SS+1;
                TABLE(/C/):=VJ(/SS/):=J;
                COST(/SS/):=MAT(/I,J/)
      'END';
ASSOCIATED WITH EACH OF THE NODES IS REQUIRED;
SORT(DV,COST,SS);
'FOR' J:=1 'STEP' 1 'UNTIL' SS 'DO' TABLE(/EE(/I/)+J-1:=VJ(/DV(/J/)/);
EE(/I+1/):=C+1
'END';
'COMMENT' PRINT OUT THE NUMBER OF PEN UP DISANCES (ELEMENTS);
OUTSTRING(1,'('NUMBER OF ELEMENTS')');
SYSACT(1,2,1);
PRINT(1,-2.0,C); SYSACT(1,2,1);
```

```
'INTEGER' 'PROCEDURE' LOWBOUND (N,CD,MAT);
'VALUE' N; 'INTEGER' N; 'INTEGER' 'ARRAY' CD;MAT;
'COMMENT' LOWBOUND OBTAINS THE COST OF AN INITIAL FEASIBLE SOLUTION
TO THE IMAGE PEN UP PROBLEM;
'BEGIN' 'INTEGER' I,J,B,K,E,M,MIN,G,D;
'BOOLEAN' 'ARRAY' A(/1:N/);
'FOR' I:=1 'STEP' 1 'UNTIL' N'DO'
'BEGIN' A(/I/):= 'TRUE'; CD(/I/) :=0 'END';
B:=LL+SP+1; K:=E:=0;
BACK:  M:=N;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' A(/I/) 'THEN'
'BEGIN' MIN:=B;
   'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' I=J 'THEN' 'ELSE'
      'BEGIN'   G:=MAT(/I,J/)-CD(/I/)-CD(/J/);
        'IF'  G<MIN 'THEN' 'BEGIN' MIN:=G; D:=1 'END' 'ELSE'
        'IF'  G=MIN 'THEN' D:=D+1
      'END';
   'IF'  D<M 'THEN' 'BEGIN' M:=D; MM:=MIN; F:=I 'END' 'ELSE'
   'IF'  D=M 'THEN'
    'BEGIN' 'IF' MIN>MM 'THEN'
          'BEGIN'  MM:=MIN;  F:=I 'END'
    'END'
'END';
CD(/F/):=MM; E:=E+1; A(/F/):='FALSE';
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' I=F 'THEN' 'ELSE' 'IF' A(/I/) 'THEN'
'BEGIN'  G:=MAT(/F,I/)-CD(/F/)-CD(/I/);
    'IF' G=0 'THEN'
    'BEGIN'  A(/I/):='FALSE'; E:=E+1 'END'
'END';
K:=K+MM;
'IF' E<N 'THEN' 'GOTO' BACK;
LOWBOUND:=K
'END' LOWBOUND;
```

NOTE: this procedure uses method 'a'. procedure using method 'b' is very similar and is therefore not presented in the listings.

LISTING OF UPPERBOUND PROCEDURE

```
'INTEGER' 'PROCEDURE' UPPERBOUND(MAT,COST,N,DOM);
'VALUE' N; 'INTEGER' DOM,N;
'INTEGER' 'ARRAY' MAT,COST;
'BEGIN' 'INTEGER' I,J,S,DELTA,C,E,DD;
'INTEGER' 'ARRAY' COPYS,START(/1:N+1/),ORDER,TAB,COPYT(/1:N*(N-1)/),
SS(/1:N*(N-1)'/'2,1:2/);
'ARRAY' CC(/1:N*(N-1)'/'2/);
'BOOLEAN' 'PROCEDURE' DOMINO;
'PROCEDURE' ELEMENTS;
E:=0;
'FOR' I:=1 'STEP' 1 'UNTIL' N-1 'DO'
'FOR' J:=I+1 'STEP' 1 'UNTIL' N 'DO'
'IF' MAT(/I,J/)>COST(/I/)+COST(/J/) 'THEN'
'BEGIN' E:=E+1; SS(/E,1/):=I; SS(/E,2/):=J;
        CC(/E/):=MAT(/I,J/)-COST(/I/)-COST(/J/)
'END';
SORT(ORDER,CC,E); C:=0;
START(/1/):=1;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
'BEGIN' 'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' I=J 'THEN' 'ELSE'
          'IF' MAT(/I,J/)=COST(/I/)+COST(/J/) 'THEN'
         'BEGIN' C:=C+1; TAB(/C/):=J 'END';
     START(/I+1/):=C+1
'END';
DD:=C;
'FOR' I:=1 'STEP' 1 'UNTIL' C 'DO' COPYT(/I/):=TAB(/I/);
'FOR' I:=1 'STEP' 1 'UNTIL' N+1 'DO' COPYS(/I/):=START(/I/);
'IF' DOMINO 'THEN' DOM:=DELTA:=0 'ELSE' ELEMENTS(DOM,DELTA);
UPPERBOUND:=DELTA
'END'UPPERBOUND;
```

THIS IS A LISTING OF PROCEDURE ELEMENTS

```
'PROCEDURE' ELEMENTS(DOM,DELTA);
'INTEGER' DELTA,DOM;
'BEGIN' 'INTEGER' I,J;

'PROCEDURE' ADD(S); 'VALUE' S; 'INTEGER' S;
'BEGIN' 'INTEGER' K,EE,KK,J,M,I;
I:=ORDER(/S/); K:=SS(/I,1/); KK:=SS(/I,2/);
M:=N+1; DD:=DD+1;
EE:=START(/K/)+1;
'FOR' I:=DD 'STEP' -1 'UNTIL' EE 'DO' TAB(/I/):=TAB(/I-1/);
TAB(/START(/K/)/):=KK;
'FOR' J:=K+1 'STEP' 1 'UNTIL' M 'DO' START(/J/):=START(/J/)+1;
DD:=DD+1;
EE:=START(/KK/)+1;
'FOR' I:=DD 'STEP' -1 'UNTIL' EE 'DO' TAB(/I/):=TAB(/I-1/);
TAB(/START(/KK/)/):=K;
'FOR' J:=KK+1 'STEP' 1 'UNTIL' M 'DO' START(/J/):=START(/J/)+1;
DELTA:=DELTA+CC(/S/)
'END'ADD;
```

REMAINDER OF ELEMENTS PROCEDURE

```
'PROCEDURE' DELETE;
'BEGIN' 'INTEGER' I;
'FOR' I:=1 'STEP' 1 'UNTIL' C 'DO' TAB(/I/):=COPYT(/I/);
'FOR' I:=1 'STEP' 1 'UNTIL' N+1 'DO' START(/I/):=COPYS(/I/);
DELTA:=0; DD:=C;
'END'DELETE;
DELTA:=0;
'FOR' I:=1,I+1 'WHILE' I.=E 'DO'
'BEGIN'  ADD(I); J:=0;
FF: 'IF' DOMINO 'THEN'
'BEGIN' DOM:=CC(/I/)+COST(/SS(/E,1/)/)+COST(/SS(/E,2/)/);
         'GOTO' OUT
'END' 'ELSE' J:=J+1;
'IF' J=I 'THEN' DELETE 'ELSE'
'BEGIN' ADD(J); 'GOTO' FF 'END'
'END';
OUT:
'END'ELEMENTS;
```

```
'BOOLEAN' 'PROCEDURE' DOMINO;
'COMMENT' PROCEDURE CHECKS EXISTING IMAGE GRAPH TO DETERMINE WHETHER IT IS
OF DOMINO FORM;
'BEGIN' 'INTEGER' I,J,S,D,FIRST,LAST;
'INTEGER' 'ARRAY' LIST(/1:N'/'2,1:3/);
'BOOLEAN' 'ARRAY' BOOL(/1:N/);
S:=N'/'2; D:=0;
'COMMENT' D IS THE NUMBER OF SEGMENTS WHICH HAVE BEEN SELECTED AT ANY TIME.
BOOL(/I/) IS TRUE IF NODE I HAS NOT BEEN ALLOCATED TO A SEGMENT, OTHERWISE
IT IS FALSE;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' BOOL(/I/):='TRUE';
BACK:
'FOR' I:=1,I+1 'WHILE' 'NOT' BOOL(/I/) 'DO' ;
'COMMENT' AT THIS POINT NODE I IS SELECTED AS A NODE SINCE IT IS NOT
CONNECTED TO ANY OTHER SEGMENT;
BOOL(/I/):='FALSE'; D:=D+1; LIST(/D,1/):=I;
LAST:=START(/I+1/)-1; FIRST:=START(/I/);
'COMMENT' FIRST AND LAST ARE POINTERS TO THE SUB-LIST OF NODES CONNECTED TO I;
BB:
'COMMENT' FIND NODE TAB(/J/) WHICH IS NOT CONNECTED TO ANY OTHER NODE;
'FOR' J:=FIRST 'STEP' 1 'UNTIL' LAST 'DO' 'IF' BOOL(/TAB(/J/)/) 'THEN'
'BEGIN'BOOL(/TAB(/J/)/):='FALSE'; LIST(/D,2/):=TAB(/J/);
    LIST(/D,3/):=J; J:=LAST+1; LAST:=0
'END';
'COMMENT' CHECK TO SEE IF NODE HAS BEEN PAIRED;
'IF' LAST¬=0 'THEN'
'BEGIN'  AA:
    'IF' D=1 'THEN' 'BEGIN' DOMINO:='FALSE'; 'GOTO' OUT 'END';
    BOOL(/LIST(/D,1/)/):=BOOL(/LIST(/D-1,2/),2/):='TRUE'; D:=D-1;
    FIRST:=LIST(/D,3/)+1; LAST:=START(/LIST(/D,1/)+1/)-1;
    'IF' LAST<FIRST 'THEN' 'GOTO' AA 'ELSE' 'GOTO' BB
'END';
'COMMENT' CHECK TO SEE IF ALL NODES HAVE BEEN PAIPED;
'IF' D<S 'THEN' 'GOTO' BACK 'ELSE' DOMINO:='TRUE';
OUT:
'END'DOMINO;
```

```
'INTEGER' 'PROCEDURE' GROUP(P,D,N);
'VALUE' D,N; 'INTEGER' D,N; 'INTEGER' 'ARRAY' P;
'COMMENT' PROCEDURE OBTAINS A LOWER BOUND TO THE PEN UP PROBLEM UNDER THE
RESTRICTION THAT ELEMENTS P(/D/) TO P(/S/) (WHERE D<S AND S=N/2) ARE NOT
TO BE PAIRED;
'BEGIN' 'INTEGER' A,MIN,J,K,I,C;
'INTEGER' 'ARRAY' R(/1:N/),B(/1:S+D-1/);
'BOOLEAN' 'ARRAY' G(/1:N/);
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' G(/I/):='TRUE';
A:=C:=0;
'FOR' I:=S 'STEP' -1 'UNTIL' D 'DO' G(/P(/I/)/):='FALSE';
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' R(/I/):=0;
'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' G(/I/) 'THEN'
'BEGIN' C:=C+1; B(/C/):=I 'END';
'COMMENT' ARRAY B CONTAINS ELEMENTS WHICH CAN BE PAIRED;
'FOR' I:=S 'STEP' -1 'UNTIL' D 'DO'
'BEGIN'   K:=P(/I/);
  'FOR' J:=2 'STEP' 1 'UNTIL' C 'DO' 'IF' MAT(/K,B(/J/)/)-R(/K/)-R(/B(/J/)/)<MIN
  'THEN' MIN:=MAT(/K,B(/J/)/)-R(/K/)-R(/B(/J/)/);
  R(/K/):=MIN; A:=A+MIN
'END';
'FOR' I:=1 'STEP' 1 'UNTIL' C 'DO'
'BEGIN'   MIN:=1000000;
  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO' 'IF' B(/I/)¬=J 'THEN'
  'BEGIN'  K:=B(/I/);
    'IF' MAT(/K,J/)-R(/K/)-R(/J/)<MIN 'THEN' MIN:=MAT(/K,J/)-R(/K/)-R(/J/);
    R(/K/):=MIN;  A:=A+MIN
    'END'
  'END';
GROUP:=A
'END'GROUP;
```

APPENDIX 7.

The C.P.U. Timer.

It is perhaps necessary at this stage to give a short description of the program used extensively in Chapter 7 to determine the C.P.U. time expended by portions of the various programs tested.

The relevant procedure, called CPUT, has one formal parameter "a" which is an integer variable called by name. The procedure returns "a" as the number of time units expended by the C.P.U. for the problem program since the last call of the procedure. By successive calls of the procedure it is therefore possible to calculate the C.P.U. time used by the problem program between the two calls.

The program is written in the I.B.M. 360 Assembler Language. It is necessary in the initial coding to satisfy certain I.B.M. linkage conventions and also to save the contents of the general registers.

The essential part of the procedure consists of a call to the supervisor which necessitates an interrupt to occur so that the supervisor can perform the service required by the program. In this case the call is supervisor call 38 (called TTROUTER) which returns the number of C.P.U. time units used by the problem program (measured in units of 13 microseconds) since the last call. The number of time units is returned in register 0.

To determine the C.P.U. time taken by a given piece of program two calls of CPUT are therefore necessary. It is easy to alter this C.P.U. time to seconds and this was carried out in each of the cases tested in Chapter 7.

The general registers are restored to their initial values before control branches back to the instruction following the point of invocation of the procedure.

Procedure CPUT was compiled as a "coded" procedure in the author's private library of procedures. Coded procedures are available in M.T.S. so that users can invoke previously compiled procedures at linkage edit time during execution of the main program.

By this means it is possible to invoke procedures written in different source languages without any inconvenience to the programmer. In fact, the random number generator used to obtain the elements of the cost matrix of the pen up problem was written in Fortran and was present as a coded procedure in the author's private library.

ASSEMBLER LISTING OF C.P.U. TIMER.

```
CPUT      CSECT                                START OF CONTROL SECTION
* EQUATE REGISTERS
ADR       EQU    8
CDSA      EQU    10
PBT       EQU    11
FSA       EQU    13
STH       EQU    14
BRR       EQU    15
* EQUATE CONSTANTS
CAP1      EQU    X'0D4'                       DECIMAL 212
EPILOGP   EQU    X'0E8'                        DECIMAL 232
          USING PBTAB,PBT                      11 IS BASE FOR PBTAB
*
PBTAB     DS     F
          DC     'CPUT'                        TITLE OF CODED PROCEDURE
          DS     F
          DC     H'32'
          DC     X'04'
          DC     X'01'
ENTRY     DC     A(PBTAB,0,PARMD&F)
EDSAVE    DS     2F
PARMDEF   DC     XL2'C211'                     DEFINITION OF PARAMETER PASSED
          STM    0,15,SAVEAREA                 STORE CONTENTS OF REGISTERS
          SVC    38                            SUPERVISOR CALL
* C.P.U. TIME RETURNED IN REGISTER 0
          BAL    BRR,CAP1(FSA)
          DC     H'8'
          DS     H
          L      ADR,24(CDSA)                  LOAD ADDRESS IN WHICH TO PUT RESULT
          ST     0,0(ADR)                      STORE CONTENTS OF REG. 0 IN THIS ADDRESS
          LM     0,15,SAVEAREA                 RESTORE CONTENTS OF REGISTERS
          B      EPILOGP(FSA)                  BRANCH BACK TO ALGOL PROGRAM
*
SAVEAREA  DS     16F                           SAVEAREA FOR REGISTERS
          END    ENTRY
```

REFERENCES

1. Half-Tone Perspective Drawings by Computer
C. Wylie, G. Romney, D. Evans and A. Erdahl.
A.F.I.P.S. Conference Proceedings, Vol. 31.
1967.

2. Machine Perception of Three-Dimensional Solids.
L.G. Roberts
M.I.T. Lincoln Laboratory.
Technical Report 315.
Lexington, Mass.
1963.

3. VISTA - Computed Motion Pictures for Space Research.
G.A. Chapman and J.J. Quann.
A.F.I.P.S. Conference Proceedings Vol. 31.
1967.

4. Perspective Representation of Functions of Two Variables.
B. Kubert, J. Szabo and S. Guiberi.
J.A.C.M. Vol. 15 Number 2.
1968 (April).

5. BE VISION. Package of IBM 7090 Fortran programs to draw orthographic views of combinations of plane and quadric surfaces.
R. Weiss.
J.A.C.M.
1966 (April).

6. An Algorithm for Hidden Line Elimination.
R. Galimerti and U. Montanari.
C.A.C.M. Vol. 12.
1969 (April).

7. Real Time Display of computer generated half-tone perspective pictures.
G.W. Romney, G.S. Watkins and D.C. Evans
I.F.I.P. Congress.
1968 (August).

8. Plane and Stereographic projections of convex polyhedra from minimal information.
Cole.
Computer Journal.
1966 (~~July~~).
May

9. Determination of hidden edges in polyhedral figures: convex case.
P. Loutrel.
Technical Report 400-145
New York University.
1966 (September).

10. Ph.D. Thesis,
P. Loutrel.
New York University.
1967 (September).

11. Three-Dimensional Computer Display.
D. Ophir, B.J. Shepherd and R.J. Spinrad.
C.A.C.M. Volume 12.
1969 (June).

12. Sketchpad 3: A computer program for drawing in three-dimensions.
T.E. Johnson.
Proceedings A.F.I.P.S. Joint Computer Conference Vol. 23
1963 (Spring).

13. The Visibility Problem and Machine rendering of solids.
A. Appel.
IBM Research Report R.C. 1618.
1966 (May).

14. The Notion of Quantitative invisibility and the machine rendering of solids.
A. Appel.
Proceedings A.C.M. Conference.
1967.

15. Techniques for Shading Machine Rendering of Solids.
A. Appel.
A.F.I.P.S. Conference Proceedings Vol. 32.
1968.

16. On Calculating the Illusion of Reality.
A. Appel.
I.F.I.P. Congress.
1968 (August).

17. Computer Program to Plot an Isometric Projection of a Solution Space Surface.
C.R. Brauer.
University of Utah
Technical Report 4-9
1968 (August).

18. Computer Method for Perspective Drawing.
H.R. Puckett
Journal of Spacecraft and Rockets.
Vol. 1. Number 1.
1964.

19. Computer Graphics
F. Gruenberger (Editor)
Thompson Book Company
1967.

20. Computer Graphics in Communication.
W.A. Fetter.
McGraw Hill Book Company.
1964.

21. Computer Graphics in Space Flight.
F.A. Heacock
McGraw Hill Book Company.

22. Sketchpad : A Man Machine Graphical Communication System.
I.E. Sutherland.
Proceedings A.F.I.P.S. Joint Computer Conference.
1963.

23. Computer Graphics Techniques and Applications.
Parslow, Prowse and Green.
Plenum Press
1969.

24. The Theory of Graphs and its Applications.
Claude Berge
Wiley
1962.

25. Linear Programming
Saul Gass
McGraw Hill Book Company.
1964.

26. Linear Programming and Extensions
G.B. Dantzig
Princetown University Press
1963.

27. Computer Displays
I.E. Sutherland
Scientific American
June, 1970.

28. Comparison of Kuhns and Silvers Algorithms to Solve the Assignment Problem.

Wright

M.Sc. Thesis, Newcastle University.

1964.