A NEW CLUSTERING ALGORITHM FOR THE TOPOLOGICAL DESIGN OF MULTILEVEL DATA NETWORKS

by

Raymond A. Vilis

Submitted to The School of Graduate Studies
in partial fulfillment of the
requirements for the degree of
Master of Applied Science

Department of Electrical Engineering
Faculty of Science and Engineering
University of Ottawa
Ottawa Canada
May 1977

TABLE OF CONTENTS

LIST OF FIGURE AND TABLES

LIST OF EQUATIONS

ABSTRACT

This thesis presents IMULEC, a new clustering algorithm for the interactive design of multilevel data networks. The algorithm is capable of handling different types of concentrators and provides the line-layout of the data network, under various given constraints, such as: line capacities, maximum traffic per concentrator, maximum number of input ports to a concentrator etc.

A comparison is made, for the single concentrator level case, with the algorithm NEWCLUST [3]. The results show that IMULEC reduces both network design cost and total execution times (with special savings in the clustering process).

ACKNOWLEDGEMENTS

The author wishes to express his sincere gratitude to Dr. N.D. Georganas for his guidance during the course of the work on this thesis.

Appreciation is also expressed to Mr. R. LeHenaff, and the members of the Electrical Engineering Department's support staff for their assistance.

The author would also like to thank the National Research Council of Canada for its financial support, in the form of a Postgraduate Scholarship.

CHAPTER 1

INTRODUCTION

Today's world is steadily becomimg one vast wired city, and as this occurs larger and more efficient data communication networks must be developed. Now, more than ever, the proper topological design of the network is critical.

Originally, when the number of users was relatively small, direct links between data centers and users were possible. But as the user population increased and became more remote from the data centers, these low speed/capacity lines became inadequate. To increase network speed, reliability and reduce overall construction costs, the network designers began to concentrate traffic, from clusters of users, at a point and then, using higher speed, more efficient lines, transmit the traffic to the data center. The development of these concentration points (given the name concentrators) created a totally new class of problems. Now, we not only needed to look at how users were to be connected to the transmisson system (direct or tree layout), but also where and how many of the concentrators should be used in the system, for maximum effectivness.

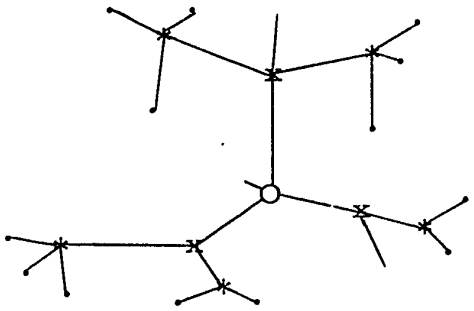
The first step was the use of a single level of concentrator facilities, feeding directly from the clusters to the data centers. But, due to the introduction of larger and larger systems, the use of several layers of concentrators may become more cost efficient.

In this study, we address ourselves to the question of the location and number of concentrators in each layer, of a multilevel concentrator network. The question of the optimal number of such layers is left open.

Some advantages of this multilevel type design include, greater network buffering (changes at one point tend not to ripple through as easily), good reliability, and maximum concentrator usage. Yet another, is the ability to use different transmission facilities at each level (e.g. telephone line, microwave links and then satellites). However, in the end, it is the fact that the multilevel design can be less expensive to build that really sells it.

Multilevel designs can take in a variety of organizations. Some, as in Figure 1.1, can simply be different capacities (from now on called levels) of concentration, each lower level

(smaller concentrators) feeding the next level up. The user, in this case, is only allowed to connect in at the lowest level.



Where: O - RDB Level 1
x - Level 2 concentrators
* - Level 3 concentrators
. - Level 4 Users

fig. 1.1

A small modification of this method, will allow users to connect to any level of concentration directly, dependent only on availibility and cost effectiveness (see Figure 1.2).

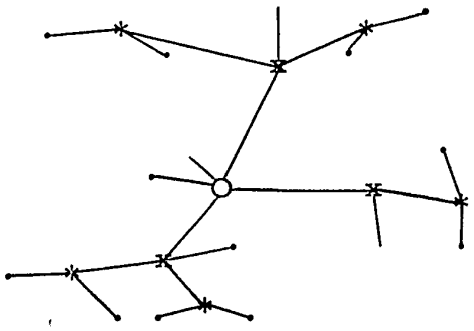


fig. 1.2

As we can see, both previous organizations represent a pyramid structure with no interconnections between units of the same level. If however, the need arose we could in fact have a fully or partial connected level of concentrators. This would be

useful, if high reliability or low delay times were needed (see Figure 1.3).

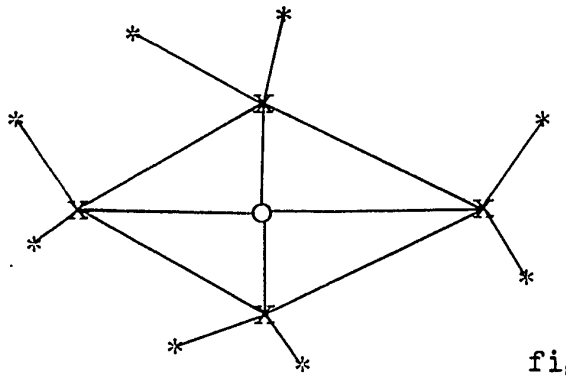


fig. 1.3

A small step futher down the road, would see microprocessor based processing units located at the different concentrator levels, processing minor requests by themselves, and in this way , reducing the flow of data requests upwards in the network. The list of modified versions could go on but the basic structure is always present.

In this thesis we are presenting a new algorithm for the interactive design of a multilevel data network. In our case we are considering a data retrival system, where a large body of users is connected to a Regional Data Base (RDB) via a hierarchically organized set of concentrator facilities. The concentrators are arranged in the organization shown in Figure 1.2. The exact number of levels of concentration, as stated

before, is considered given prior to the initialization of the algorithm.

To allow for greater flexibility in the execution of the algorithm, we have designed it to be interactive in nature. Ideally the algorithm would be used in conjunction with an interactive graphics terminal, using a backup processing unit. By using this type of system we can involve the network designer more closely with the actual solution producing process. This point is discussed further in Chapter 3.

There are three main sections to our algorithm. The first is the concentrator location problem and this is dealt with in Chapters 3 & 4. The second, the interconnection of users, is given in Chapters 3 & 5. Finally, the finalization of concentrator location process is outlined in Chapters 3 & 6.

The final result is an interactive, heuristic design of a multilevel network. Since our results are heuristic in nature, no elaborate mathematical proofs are given. But to provide a basis of comparison with existing work, Chapter 7 gives numerical comparisons between our algorithm and NEWCLUST [3], in the three level case (one RDB level, one layer of concentrators, and the users). In turn NEWCLUST has been related (by its

author) to works by Martin [14], Doll [15], and Bahl & Tang [17]. Our final conclusions and extensions are given in Chapter 8 of the thesis and in the Appendix a brief discussion of graphic terminals with a simple example on their use is included.

CHAPTER 2

TOPICS IN NETWORK TOPOLOGICAL DESIGN

To help introduce some of the problems encountered in the topological design problem, we have included the following short survey of some existing literature we studied on the subject.

2.1 The Clustering Problem

2.11 Introduction

One of the main problems we faced in our work was that of locating an initial set of possible concentrator sites to work with. To do this we needed an efficient clustering technique capable of identifying groupings in our network. It also had to be able to locate lower level units around the larger concentrator units. Some good clustering processes were found and are outlined below, but all have the drawback of not being able to easily select concentrator sites around existing units. Since, we had stated this as a prerequiste none could be used directly.

2.12 Center of Mass Clustering

Here, we outline a paper by McGregor & Shen [1], which shows one possible way of solving the clustering/concentrator location problem.

In McGregor & Shen's paper, clusters are formed in a "snowball" or "balancing" manner. The system selects the closest pair of nodes in the network and creates a new node at their center of mass (COM), with a traffic equal to the sum of the traffics of the absorbed nodes, and a weight equal to the number of nodes that the COM absorbed. The COM node is in turn combined with its closest neighbour in the same fashion. This process continues until the cluster of nodes represented by the COM node cannot be added to without violating one of the design constraints. Then the next two closest free nodes are grouped in the same manner. This continues until no further absorptions can take place without violation of design constraints or no two nodes are within a preset distance of each other.

Once this has been done, we have a set of COM's as shown in Figure 2.1, these COM's in turn are the representations of the clusters in Figure 2.2. The

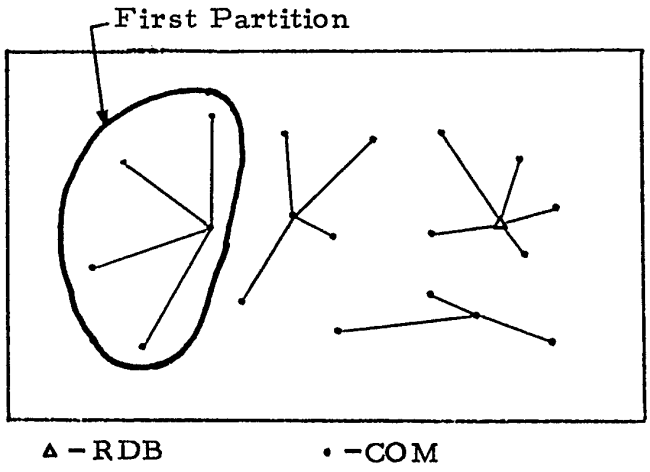


fig. 2.1

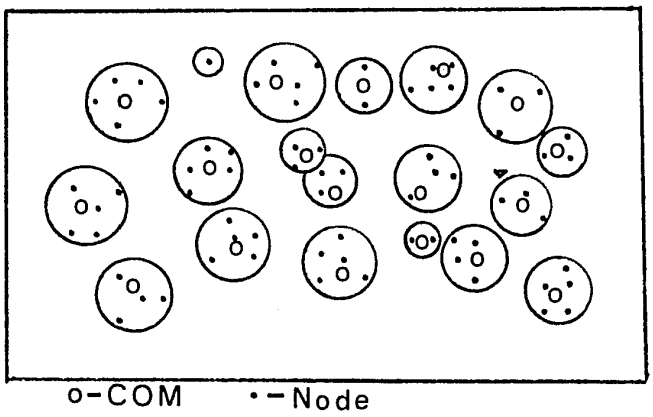


fig. 2.2

$$R_i = \sum_{j \in B_i} S_{i,j} * W_j - d(i) * \beta \qquad \text{eq. 2.1}$$

Where: R_i = estimate of relative benefit of a concentrator at node i.

B_i = set of nodes associated with concentrator at node i.

$S_{i,j}$ = savings of connecting node j to concentrator at i instead of to RDB.

W_j = weight of node j.

$d(i)$ = cost of connecting concentrator at i to RDB.

β = parameter used to empasize proximity of concentrator to RDB.

authors then calculated, using equation 2.1 and a modified "ADD" approach, the cost benefit of placing a concentrator at each of the COM sites in turn. The COM site with the largest cost benefit was selected and it and all other COM nodes that found it more economical to be connected to it, rather than directly to the CPU, were partitioned from the main network. The remaining COMs were scanned for the next best location and so on until all the network was partitioned as in Figure 2.1.

Although the authors claim significant computer time saving by this method, their execution time is still proportional to N^2 and this for large networks would tend to become too large. Another drawback of the procedure is the fact that the final result is dependent on the initial starting point of the process. Possible improvements to the algorithm would be the more careful selection of the starting point and allowing a number of clusters (COMs) to be formed simultaneously, instead of one at a time.

2.13 Nearest Neighbour Approach

An excellent example of this method is in the NEWCLUST algorithm [3], written by H. Dysart. In his work

Dysart starts his topological design by locating an initial candidate concentrator list, using the nearest neighbour approach. As we used the NEWCLUST algorithm for comparision purposes, we will include a detailed description of the nearest neighbour system used by it. The performance study of this clustering process is discussed in Chapter 7.

Step 1: For each of the N terminals in the network list it and its K nns (N.N.); K being a design parameter.

Step 2: For each terminal count the number of times it occurs in the K-N.N. lists.

Step 3: Each terminal now has a number that represents its frequency of occurance, f_i , in the K-N.N. lists. If F is the maximum frequency over all terminals then each terminal will have one of the frequencies, $f_i = j$, $j=1,2,...F$, associated with it. All terminals with the same $f_i = j$ are stored in a list S_l. The number of terminals in a list S_l is x_l and $\sum_{l=1}^{F} X_l = N$

Step 4: The weighted mean, (W.M.), of this discrete distribution and its integral part + 1(KM) are calculated.

Step 5: The candidate concentrator set C is initialized by the terminals of S_F, which have a frequency of $f_i = F$. Subsequent terminals are added to C by the addition of terminals from the set which has the next lower frequency (i.e. S_{F-1}, S_{F-2},...S_1) until either the addition of next set of terminals brings the number of concentrator locations above N/2 or the sets frequency is less than KM. If either of these conditions is met the algorithm terminates.

The advantages of this type of clustering over the previously outlined ones is its speed.

A more mathematical description of the K nearest neighbour approach can be found in a paper by Fukunaga and Hostetler [4].

2.14 Additional References

One paper that deals with the concentrator location problem using a very detailed and elaborate approach was written by Greenberg [2]. The paper presents an algorithm which contains the following features.

1) Preselection of candidate concentrator sites is unnecessary.

2) All sites in the network are candidates for concentrators.

3) Dynamic readjustment of concentrator locations occurs throughout the duration of the algorithm.

4) Useful conditions are given for determining optimality.

The algorithm, however, becomes too lengthy (in terms of execution time) to be used effectively for the design of large networks.

Some other papers that deal with the clustering problem, only in a more statistical fashion, have been written by Walton & Frisher [5], Koontz [6], and Gitman & Levine [7]. These papers deal basicaly with clustering in

the image processing field, where the shape of a cluster and not a concentrator location in the cluster, is critical. The interesting point of the papers, however, is their ability to process large sets of data points in relatively short computer times and generate results that show the distibution of the users (points) in the network (picture).

2.2 The Linedrop Problem

2.21 Introduction

Once we have located a set of possible concentrator sites, there arises the question of how the users must be connected to them. The most reliable method would be to create a "star" type system where each user has its own line to a concentrator (see Figure 2.3a).

a b

fig. 2.3

However, in most cases this becomes uneconomical and

is therefore not used. We would then go to the next possible connection type, that of a multidrop line layout (see Figure 2.3b). Many good algorithms are already present that can create this configuration and some are listed below.

2.22 Minimum Spanning Tree

The simplest of all the procedures starts from a star connected network, then forms the multidrop layout by breaking direct links to the RDB and routing the nodes traffic to the RDB through a neighbour. The basic algorithm [22] does not, however, take into account design constraints. There does exist updated versions of this process (e.g CMST [34], RCMST [9]) that do take constraints into account, but in general they represent one of the less efficient ways to create the line-drop layout.

2.23 Essau & Williams

This algorithm, developed by Essau & Williams [10], has become the basis for many multidrop line layout processes. Like the MST it starts from a "star" topology, only now it selects the order that the multidrop network is formed by the use of a cost benefit equation (Equation

2.2).

$$C_{i'j} = D(i,N) - D(i,j)$$

Where: $C_{i'j}$ = cost benefit of connecting node i to node j. eq. 2.2

$D(i,N)$ = line cost of a line directly to RDB from node i.

$D(i,j)$ = cost of routing node i through node j to RDB.

The joining with the highest cost benefit, that does not violate design constraints, is formed first and then the tables are updated and the next best joining is selected and so on.

The drawbacks, of this method, are the large memory requirements ($\approx N^2$) and an exponentially increasing execution time. It does, however, claim to result in solutions to within 5% of the optimal.

2.24 Modified Essau & Williams Algorithm

One of the small, but important, modifications made to the Essau & Williams method was done by Karnaugh [11]. Instead of computing the cost benefit of connecting each point to every other point in the network, Karnaugh finds

a nearest neighbour for each node and calculates the cost benefit only between a node and its nearest neighbour. This system not only cuts down on the computations done by the algorithm, but also on the amount of updating needed after each joining. The result is a saving in execution time with the same quality of answer. However, some of the time savings are lost due to the requirement for pre-processing of the distance matrix. Even with its improved time, it still has an exponentially increasing execution time.

2.25 Additional Algorithms

We now go to a paper by Whitney [12], which compares several line-drop processes including another modified Essau & Williams called SAHC [13]and the previously mentioned CMST and RCMST algorithms. Figure 2.4 shows the time graph for the various methods. As can be seen all have execution times that increase exponentially with the number of nodes.

Figure 2.5 shows how the algorithms relate, in terms of cost performance for a 200 node network. Note that the slower, more involved, algorithms result in the best cost saving.

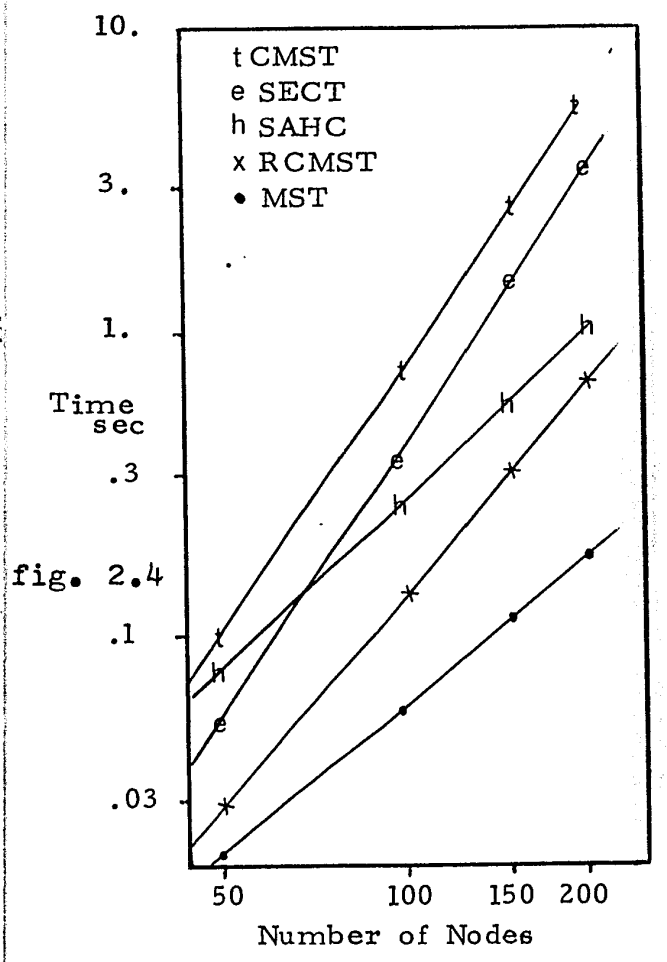


fig. 2.4

Algorithm Execution Times
(Logarithmic Scales)

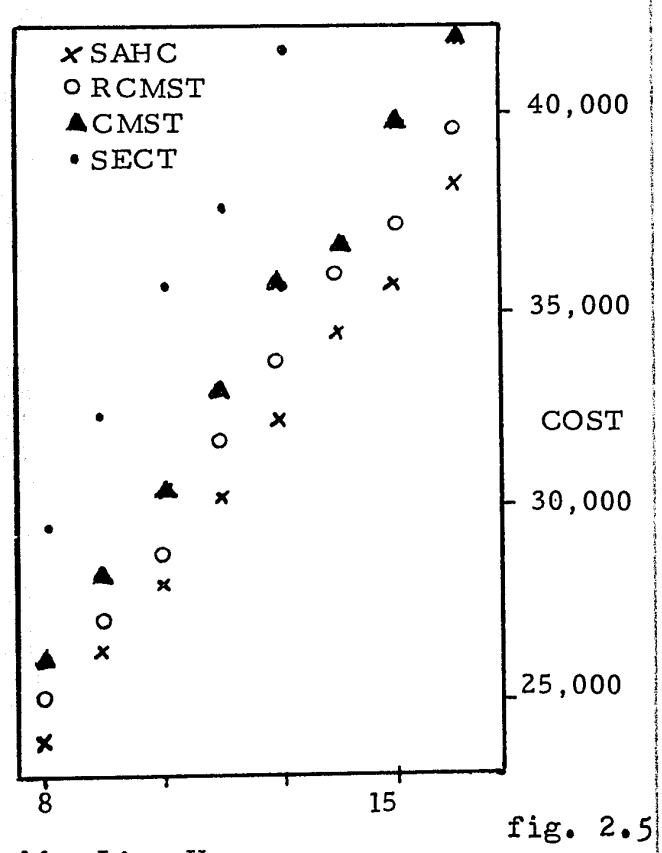


fig. 2.5

Max Line Usage
Cost-Performance Graph

2.3 Finalization of Concentrator Locations and Topology

2.31 Introduction

Until now we have seen processes that deal with only a discrete part of the total network design problem. We now will look at several algorithms that present a total package, from initial concentrator location to the final topology. They represent some of the more frequently used approaches.

2.32 The "ADD" Approach

By the "ADD" approach we refer to a system where concentrators are continually added to the network until no further cost benefit is incured by the addition of more concentrator units. This method is used by Martin [14], Doll [15], and Woo & Tang [16], in their layout processes.

Some of the variations introducted by the above papers include: first, considering only a subset of the nodes in the network as possible concentrator sites instead of every node and second, the manner in which concentrators and their associated nodes are removed from further consideration, during the process.

2.33 The "DROP" Approach

In this approach we select a set of possible concentrator locations from the network, using one of the clustering/concentrator location processes in section 2.1. We then proceed to drop concentrators from this list until we reach the point where no further cost reduction is realized by the removal of another concentrator.

Two examples of this type of algorithm are Bahl & Tang [17]and NEWCLUST [3]. Again, as we will be doing some comparisions with NEWCLUST later on, we include a detailed outline of the proceedure. It begins by creating an initial concentrator list as described in section 2.13 and then proceeds through the following steps to the final topology.

Step 1: Calculate the concentrator costs, F_i and the terminal to concentrator costs $C_{i,j}$,i = 1,2,...N; j= 1,2,...M.

Step 2: Drop any concentrator that does not have any terminals assigned to it.

Step 3: Initialize the concentrator subnetworks as follows: a) Assign each terminal to its closest

concentrator. b) Modify those concentrator subnetworks that have their capacity constraints violated. c) Reassign all terminals that have been dropped in b).

Step 4: a) Apply the multipoint optimization algorithm to each concentrator subnet. b) Partition the terminal of each subnet into supernodes and calculate their parameters. Calculate the variables for each concentrator.

Step 5: Determine each supernode's variables and gain of each concentrator.

Step 6: Determine which concentrator, j^*, has the largest gain G_j^*. If $G_j \leq 0$ GO TO Step 10; else GO TO Step 7.

Step 7: If j^* is the last concentrator that was checked for feasiblity, then GO TO Step 8; else check for the feasibility of j^*'s supernodes' reassignments. If any modifications occur GO TO Step 8; else GO TO Step 6.

Step 8: Close concentrator j^* and decrease the network cost by G_j^*.

Step 9: Assign j*'s supernodes to their respective closest feasible neighbours and update the parameters of the neighbours and their associated concentrators. Recalculate the necessary variables and concentrator gains. GO TO Step 6.

Step 10: Record the resultant topology and cost.

Step 11: Initialize the network cost to zero and repeat Steps 2-4 (excluding the formation of supernodes) and record the new topology and cost.

Step 12: Choose that topology which has the lowest cost as the final topology and terminate.

A study of the performance of the NEWCLUST algorithm will be done in Chapter 7.

2.34 Linear Regression Approach

This is a recently developed algorithm by Dirilten & Donaldson [18]. It considers the concentrator location problem and the optimization of their choice in a single step, with the use of regression lines.

The algorithm begins by finding the regression line that is the best mean-square fit to the node distribution

in the network. Next, the network is divided into two groups by means of a line passing through the clusters (the total network represents the first cluster) mean and perpendicular to the regression line. These subgroups are then themselves divided by the same method and the process continues until no subgroup has a capacity requirement greater then C_k(where C_k is the minimual size concentrator available). The process then regroups the subgroups in such a way as to make maximum use of concentrator capactiy, and at the same time maintaining the mimimum cost, given the design constraints of the problem.

The process results in a saving in both time and design cost when compared to an "ADD" [19]algorithm.

2.4 Summary

We believe that the papers refered to in this chapter represent a good cross-section of some of the best works in the topological design area. Detailed outlines of the papers were not included.

Although none of the papers dealt with the multilevel design problem directly, some did mention it

as a possible extension. There has in fact been very little work done in in trying to solve some of the unique problems that come with the multilevel design and none in the interactive approach to the problem. One example of such a problem is the arrangement of the different levels of concentration, and the fact that lower level units must be placed in such a way as to take into account the higher level units (see Blocking, section 4.26). The next chapters give our answer to the problem.

CHAPTER 3

OUTLINE OF THE NEW ALGORITHM

3.1 The IMULEC Algorithm

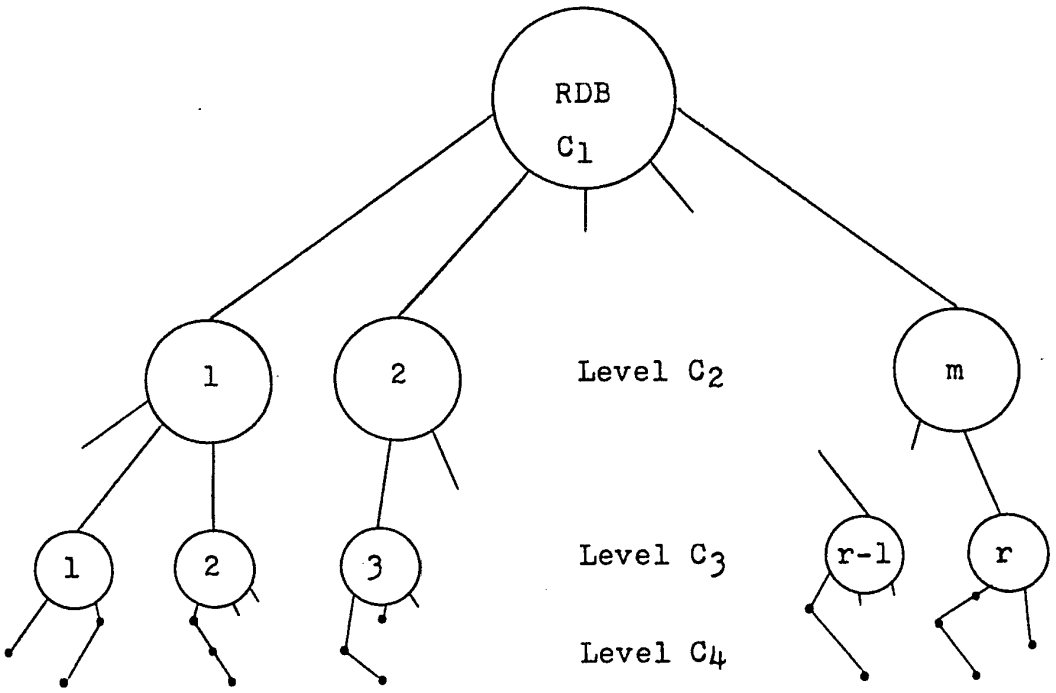
After we reviewed some of the existing algorithms for topological network design, one fact became quite clear, that is that most present algorithms allow for little or no designer interaction with the process. Interactive processing systems are already present in fields such as image processing. An excellent example of such a system is outlined in a report by Goldberg and Shlien [23]. Here the system does all the processing while still allowing the operator to use his expertise to achieve a final result superior to that of a simple input/output routine. The operator can easily select variations to be fed into the system and in this way take into account far more variables than normally would be possible. This same sort of system could be built using IMULEC. The operator would then be able to design around constraints that are not always time invarient and mathematically definable. Political, social and natural influences on network design are some examples. They are

fluid constraints and can best be handled by this type of interactive system. It is with this in mind that IMULEC (Interactive Multilevel Clustering) was formulated.

The problem can be stated as such; " Develop an algorithm capable of designing a multilevel hierarchical network (see Figure 3.1), and designed in such a way as to allow it to be used either interactively with a designer or to be simply an input/output routine".

To this end we first defined a line-drop procedure that was capable of distinguishing between levels of concentration. Since we would judge the goodness of the network design by its cost, we chose to use the cost optimization procedure outlined by Essau-Williams [10], as the core of the routine. We then incorporated the closest neighbour approach of Karangh and our own supergroups and "feasible joining area" concepts. We also included constraint variables into the design process, so as to simulate "real" design conditions as close as possible.

The next step, once we had the line-drop routine, was a clustering procedure to give us our initial set of



Level 1: Regional Data Base(s)

Level 2: Concentrators of capacity T_o, where $T_i \leq T_o \leq T_j$

Level 3: Concentrators of capacity T_q, where $T_k \leq T_q < T_i$

Level 4: User terminals

Figure 3.1

Network Organization

possible concentrator sites. After reviewing what had been done until now by people such as McGregor & Shen [1], Dysart [3] and others (as outlined in section 2.3), we decided that the best way to proceed, was by attempting to somehow convert the problem from a single point by point study to a group approach. We also looked for a system that would use as much of the initial information available about the network as possible.

After experimenting with a variety of statistical analysis, histograms and nearest neighbour approaches, we found that none gave us the speed or versatility needed . This led us to the idea of forming a grid over the network and then using the grid squares (from this point called sectors), to set up a frequency table and in this way giving us a mapping of the distribution of the network. After some experimenting, we developed the self-adjusting grid. From there we found that by scanning the grid with a "window" we could locate potential concentrator sites. The first scan of the network is done with a large window to identify areas in the plane where high user (or usage) concentration exists. We then entered each of these areas to select an exact concentrator candidate site (see Figure 3.2). Next, as we

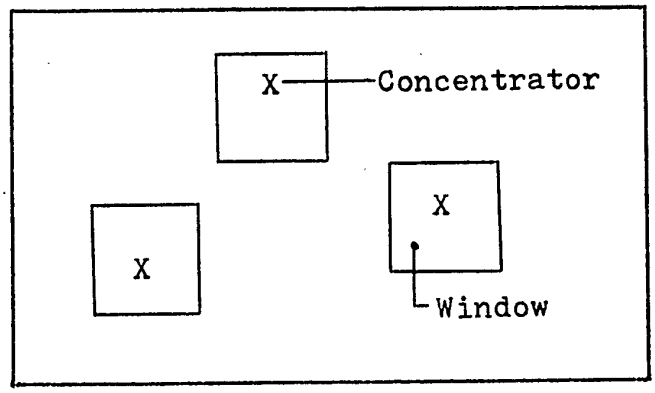


fig. 3.2

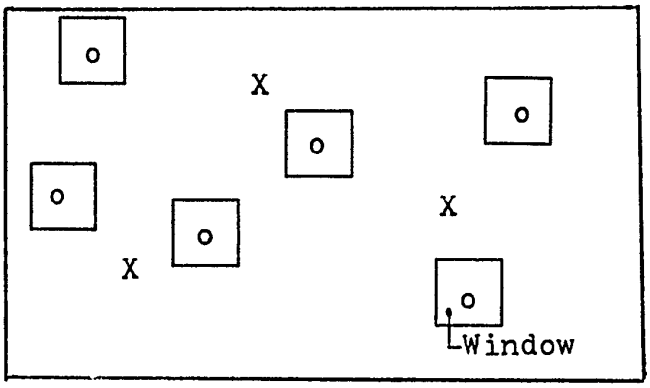


fig. 3.3

Example of smoothing effect of large window on distribution

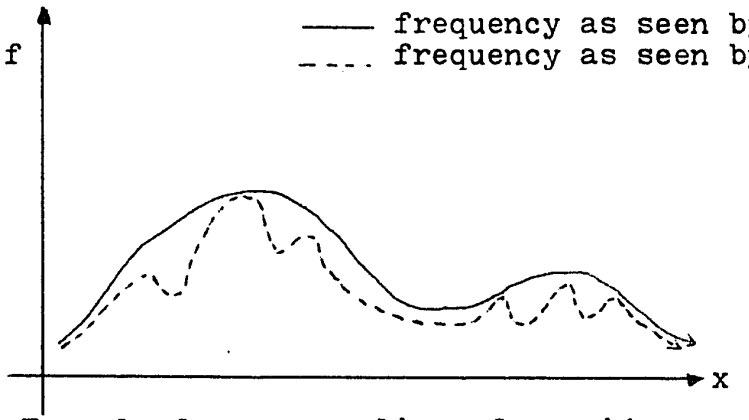


fig. 3.4

Example frequency slice of a grid

want to locate smaller facilities, we decrease the window size and are then able to pick out the smaller population peaks around the existing higher level devices (see Figures 3.3 and 3.4). This process could in fact select any number of concentrator levels by simply adjusting the variables properly. Again testing took place to fine tune the method to meet our other requirement of allowing an operator to interface with the routine. The detailed description of the process is done in Chapter 4.

The final step of the process was to develop a technique to select a final set of concentrators from the initial set found by the clustering part. Here two possibilities were looked at, one was to use a modified NEWCLUST approach, the second and the one finally chosen was to use a feature already built into the line-drop process. This procedure was originally designed to correct for to many input lines to a concentrator (see Chapter 5). This was the final step, and the package was ready for testing. To this end we had difficulty in selecting a method of comparision with a known algorithm, as no algorithm could be found that specifically designed four, or higher, level networks. We finally decided to test IMULEC against NEWCLUST at the three level stage.

This required modifications to IMULEC which affected its performance slightly. Another problem was that IMULEC was designed to be used interactively with an operator and as such sacrificed some processing speed for flexibility.

Finally after solving all the problems, we ended up with an algorithm that answered our originally stated problem. The next four chapters give a detailed outline of the routines used and in the fifth chapter we discuss results and final conclusions.

3.2 IMULEC's Flow Chart

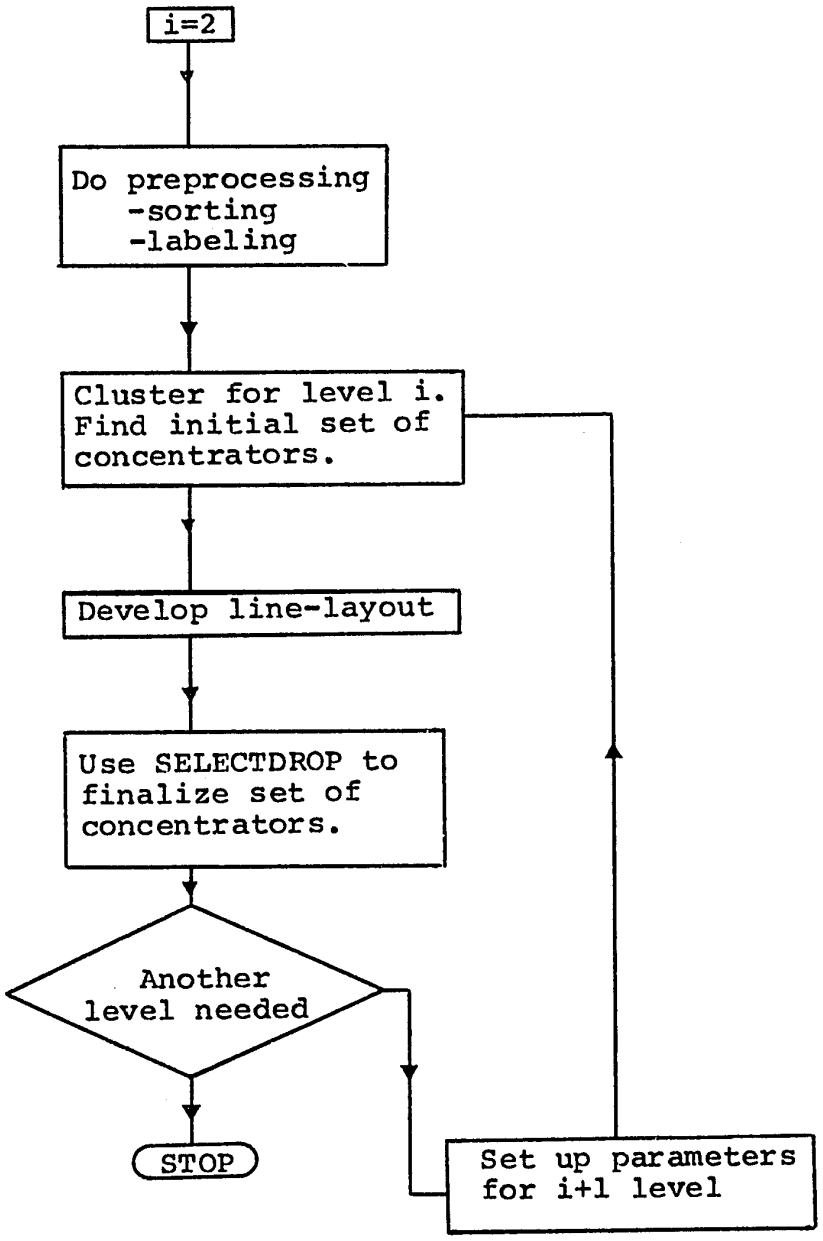


fig. 3.5

CHAPTER 4

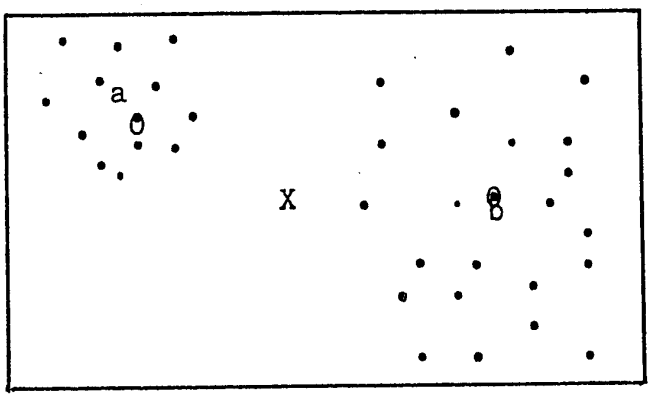
CLUSTERING PROCESS

4.1 Introduction

The goal of our clustering technique is to take a large number of user locations and identify a candidate set of possible concentrator locations in the network. These concentrators can be located in two different types of regions.

a) As shown by the region 'a' in Figure 4.1, we need to identify natural clusters in a network. Once these are located we must identify a satisfactory point in them to be our concentrator location.

b) The next group we must be able to identify is the one indicated by group 'b' in Figure 4.1. Here there is no real natural cluster, but there is sufficient traffic being generated in the zone to warrant the use of a concentrator device.



X- RDB

fig 4.1

X - RDB ⊕ - Concentrator

Another criterion that the clustering technique must meet is the ability to choose different levels of concentration from the network. This is shown in Figure 4.2.

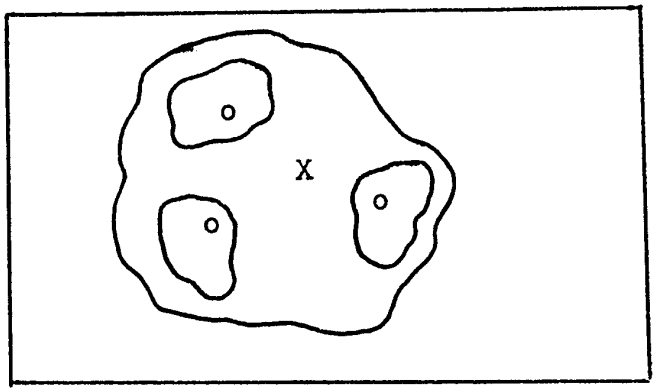


fig. 4.2

X - RDB o - Concentrator

Here we have a large cluster 'a',with its concentrator located at 'X', that has been identified by the first pass through the process. On a second pass the clustering technique will have to reduce its window (the area of a network that it is looking at, at a given

time), so as to locate smaller concentrators at locations 'o'. The ability to work inside a cluster could be viewed as a hybrid application of its two types of location abilities (as in Figure 4.1).This is a very important capability of the process, and it allows us to break the network down into as large a number of groups as we wish.

Realizing these problems, we have designed a fast and effective clustering process that allows us to use the same method to locate any number of different types of concentrators. There is no claim of optimality, or that this is necessarily the best process to use in all cases.

4.2 The Process

4.21 Sorting

The first step of the clustering process is to locate an artificial point and call it '0.0'. This point is located at an extreme edge of the network, such that no other user is located "below" or to the "left" (given the network is on a plane) of this point. All co-ordinates of the network would be in reference to this point, thus co-ordinates of a point are such that $X_i, Y_i \geq 0$.

We then create a list that contains the identifying number of each node in order of their proximity to the '0.0' point. In other words the closest node in the X direction would be located in the first position of the list, and the last position of the list contains the node furthest from the origin in the X direction. Figure 4.3 shows how this would appear.

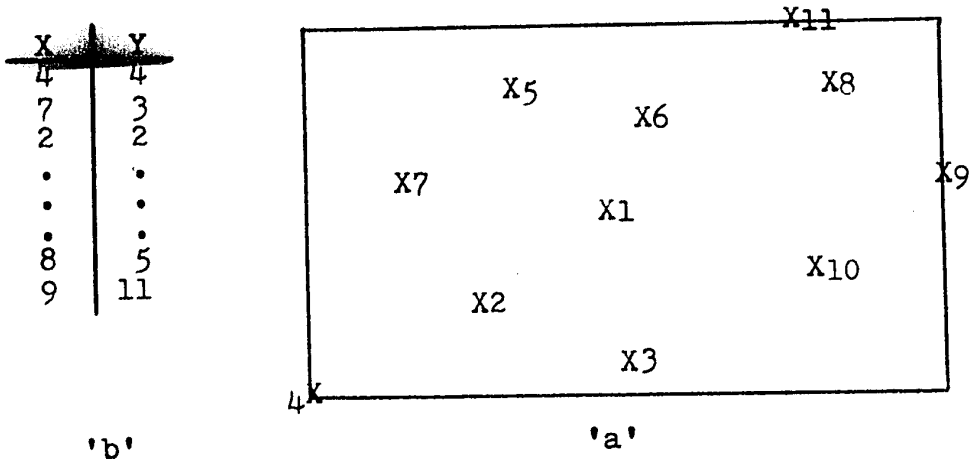


fig. 4.3

The way in which these lists are created (Figure 4.3b) depends on the programmer's ability and the computing machine used. One method found to be useful is a "bubble" sort technique. (see Appendix C,bubble sort program)

We must remember though that this sorting is needed only once for each network and does not in the overall picture effect the speed of the process greatly (see Section 4.41, Execution Time).

4.22 Grid

We now set up a grid (Figure 4.4) to section the network into rectangular shaped sectors. There are many ways in which to choose the exact shape and style of the grid, some are:

1) Using a straight set width for each X and Y bar.

2) Setting the bar width by setting the minimal number of nodes to be contained in each bar.

3) Setting the bar width by setting the minimal amount of traffic each bar must contain.

4) Assign a semi-fixed width for the bars by stating that the bars must be at least a certain width and that they must all stop on a node.

5) A combination of 2 & 3; or any other combination of steps.

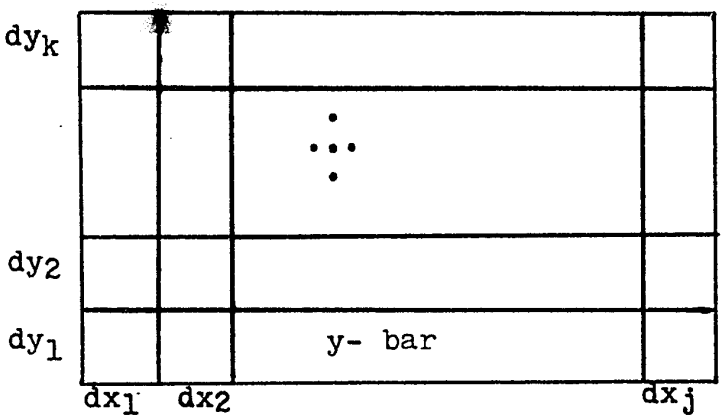


fig. 4.4

We have found that for now the fourth option is satisfactory. The other four options are all valid and each has its specific use, but the fourth is the one found to be most general. We therefore set the width of the X(dx) and Y(dy) bars to be:

$$dx_j \geq G\% \text{ of } X \text{ max}$$

$$dy_k \geq H\% \text{ of } Y \text{ max}$$

eq. 4.1

For now we will assume that G=H. Thus we are assuming a square grid. In other cases, say a network spanning only the southern part of Canada, we may use a rectangle as in Figure 4.5.

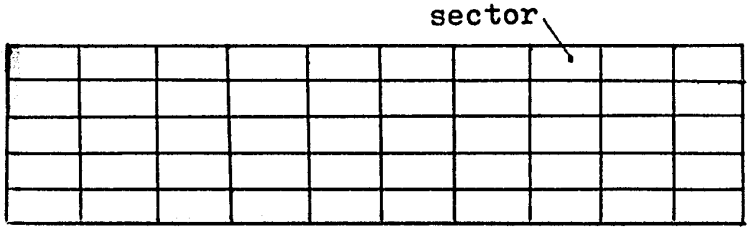


fig. 4.5

But no matter what shape the grid takes, we always attempt to keep the sectors as close as possible to squares. The ideal shape for a sector would be multi-sided cell or circle but to keep processing simple we approximate to a square.

The number of sectors in a grid is set by a design parameter K.

$$K = \frac{\text{number of nodes in the network}}{\text{number of sectors}}$$

eq. 4.2

We have found that $K \approx 1$ is a satisfactory value to design around (however, a mimimum grid of 121 sectors is needed, for smaller networks use an exact clustering procedure).

We now come to the reason for having the grid lines start and stop on a node. By forcing this to occur we allow the grid to adjust to the number of nodes in a network. If the network is sparsely populated then the bars will be larger, as the distance to an end node is larger. As the number of nodes increases, the probability that there is a node at exactly dx is greater. Thus, in

the sparse case we find wider bars being formed and even in some cases the dropping of the last one or two bars, if necessary. This process of flexible bar widths tends to make the grid more dependent on the network it is applied too.

4.23 Frequency Table

The frequency table indicates the number of nodes in each sector. The dimensions of the table are equal to the number of X,Y bars, in the grid (X-columns, Y-rows). The first entry at position (1,1) indicates the number of nodes in sector (1,1), as shown in Figure 4.6.

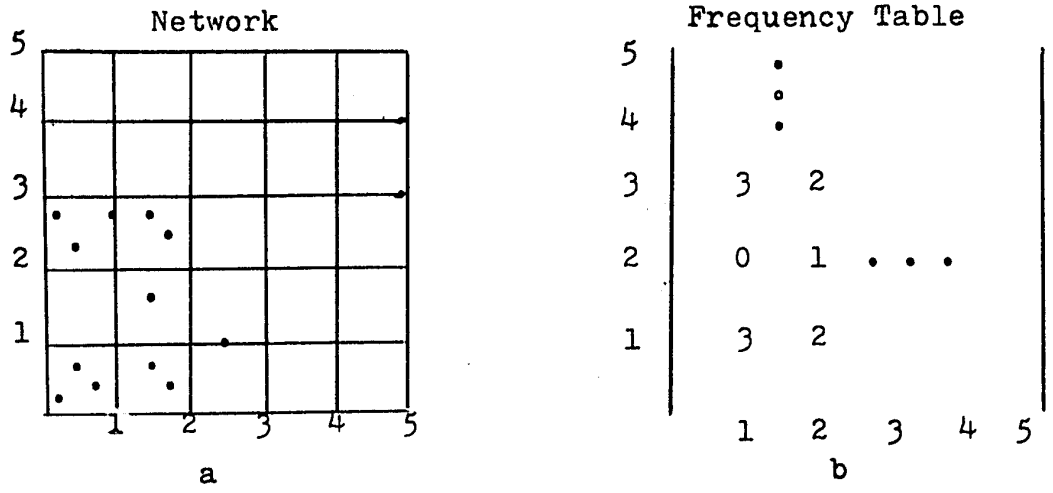


fig.4.6

The purpose of this table is to allow us to find the number of nodes in a given area quickly. It could be

modified to show not only number of nodes, but also indicate traffic in the sector.

4.24 Window

We now come to the choice of our "window" size. This "window" is defined as the area of a given number of sectors, in which a concentrator facility will be located. It is effectively a rectangular hole in a large mask that moves across a network scanning to see when the window contains the highest number of nodes.

Figure 4.7 shows some possible positions for the window and the number of nodes at these positions. The number of such pictures would depend on the number of sectors and the window size. In this example picture 'a' has the largest number of nodes thus it would be our first choice to locate a concentrator in, and then window 'd' and so on till we have enough possible sites.

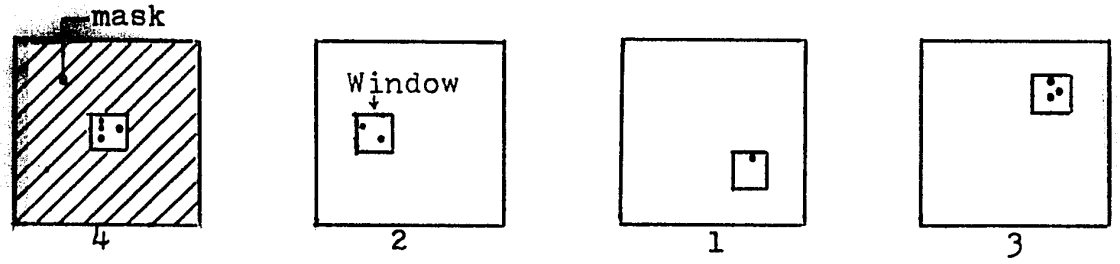


fig. 4.7

Thus we can easily see that the first problem to consider, when fixing the window, is the maximumn number of concentrators needed at a level and thus the maximumn number of windows required for that level.

The maximum number of candidate concentrator locations at any level is given by the equation 4.3.

$$N_{C_i} = \frac{\text{total traffic}}{C_{i_{MIN}}} - \sum_{J=1}^{i-1} N_{C_J} \qquad \text{eq. 4.3}$$

Where: N_{C_i} = number of concentrators of level C_i to be found.

$C_{i\,min}$ = mimimum capacity of concentrator of level i.

The reason for the summation term is to take into account that a user can be connected to an upper level unit directly. A concentrator already located should not have another level concentrator candidate located too close by (see sections 4.25 and 4.26, Buffering and Blocking).

We now proceed to find the dimensions of our window. For now we have assumed, with no loss in generality, a square window with sides Z sectors across.

Let $V = \frac{\text{total number of nodes in network}}{\text{number of sectors with at least one node}}$

U = lesser of;

1) Maximum number of nodes a concentrator can handle.

2) The number of nodes a concentrator need handle. e.g. $\frac{\text{total number of nodes}}{\text{total number of concentrators}}$ eq. 4.4

Therefore: $Z * Z * V = U$

$$Z = \left\{ \sqrt{\frac{U}{V}} \right\}$$

Where Z is the closest integer given the break point at .75 instead of the normal .5.

It should be noted that 'V' here indicates the average density of a sector. There are, however, other ways in which 'V' could be defined. It could, for example, be dependent on a threshold level greater than one node, or it could also include traffic density

information. All are valid approaches , the latter obviously resulting in a more intelligent window, but not necessarily a better solution.

4.25 Buffer

One problem that is encountered during the clustering process is that there may not always be enough room in the grid to locate all the required windows without any overlap. This problem is shown in Figure 4.8. Here we need to locate three windows of four by four sectors each. There are enough sectors to fit the windows in, if they were placed with only that in mind, but we pick window locations by their densities. Thus, by the time we get to locating the third and final window, we cannot do so without overlapping with one of the two windows already placed. This is where the buffer comes into play.

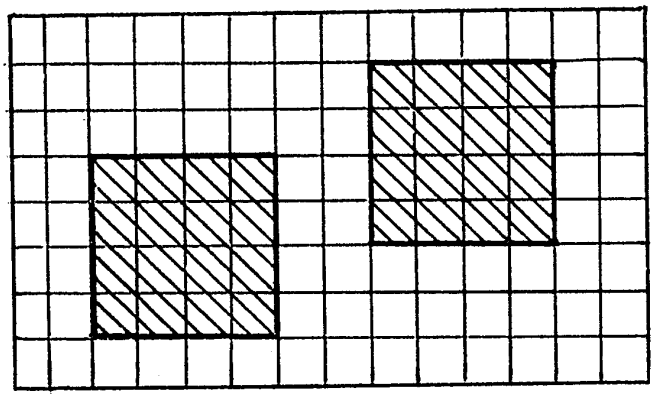


fig. 4.8

Normally the buffer is set for no overlap, but if we find as in this case that overlap is necessary the buffer is incremented by one to allow the overlap of any one row or column of sectors between windows. This is shown in Figure 4.9.

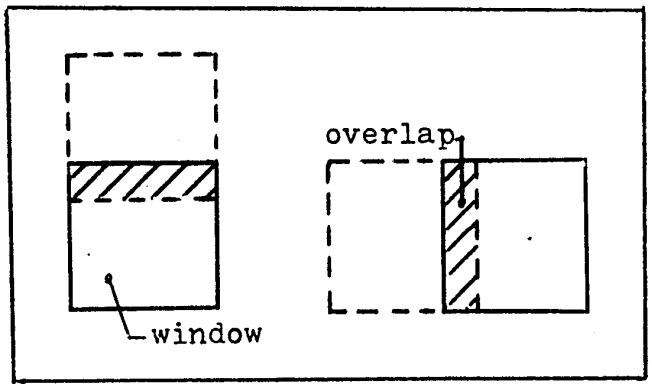


fig. 4.9

The routine then returns to the scanning section and once again scans the network, but this time allowing an overlap as shown in Figure 4.9. If after this run, still

not enough windows are found the buffer is again incremented and this process continues until we have all the necessary concentrator windows.

Two points should be noted here:

1) That there is a limit, set by the operator, to the maximum amount of overlap allowed. If this limit is reached the routine prints out an error message and we must start over again only this time changing either or both the number of sectors or the window size.

2) As the buffer changes the same zones may not be picked each time. This is due to the fact that by allowing overlap we may have created a more favourable , previously not allowable zone.

4.26 Blocking

Blocking is done to insure that two possible concentrator candidate sites are not located too close together. It also insures that a possible concentrator site is not located too close to an already established higher level unit.

The size of the blocking area is equal to the window size that is currently being used. Figure 4.10 shows such blocked areas.

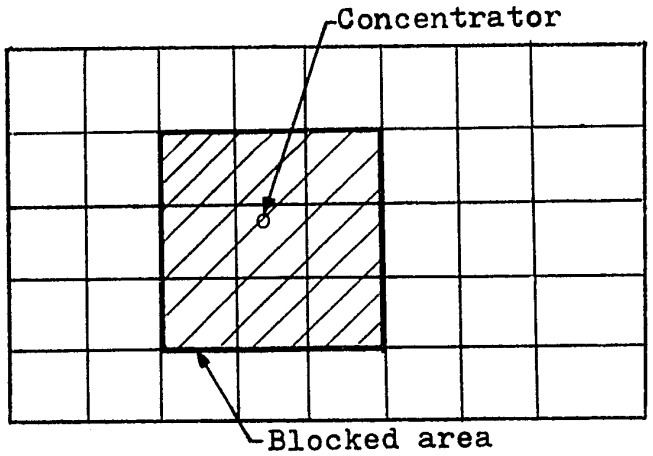


fig. 4.10

There does however exist the possibility that the concentrator location and the window size (a two by two) are such that the concentrator is at the edge of the blocked area. Figure 4.11 shows the two possible cases that occur. The first Figure 4.11b has the concentrator at the bottom of its sector thus this sector would be the top right of the blocked area. Figure 4.11a shows the concentrator in the top part of its sector. In this case the sector forms the bottom left corner of the window. If the window is larger than a two by two, but still has an even number of sectors along its sides, then the same process is followed except the two by two, formed as stated above, forms the core of the window (Figure 4.11c).

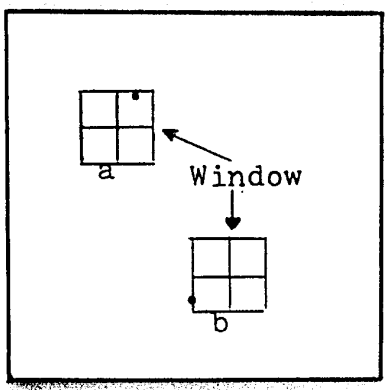


fig. 4.11a,b

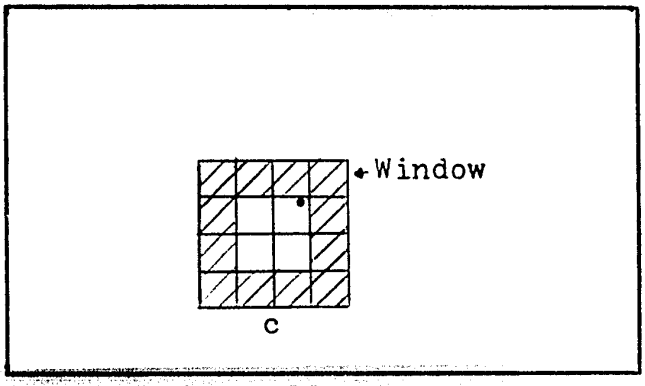


fig. 4.11c

fig. 4.11

Blocking need not be exact, it need only stop a candidate from being set too close to another unit, thus this approach is more than satisfactory.

4.27 Scanning

Once the selection of grid, window size, initial buffer and blocking has been done for a level, we proceed to run the window over the entire network picking off the zones that have the highest population of nodes (see Figure 4.7, window size). As one concentrator zone is selected (highest populated window of entire scan), the window which was chosen is blocked and all windows overlapping it by more than the allowable buffer size are disallowed as possible candidate sites. The scan stops when the specified number of concentrator sites are found, but before the process goes to the next step, it checks if any other windows exist that are equal in

quality to the last site picked. If so, the additional site is recorded along with the others (leaving it to the optimization process to decide if it should be a site). The population (P_K) of a window is given by equation 4.5.

$$P_K = \sum_{i=Y_t}^{Y_t+Z} \sum_{j=X_t}^{X_t+Z} F(i,j) \qquad \text{eq. 4.5}$$

Where: X_t ,Y_t are the identifying bar numbers of the window, see Figure 4.12.

Z is the number of sectors wide the window is (for now we have assumed square windows).

F(i,j) is the density of sector i,j.

Effectively equation 4.5 is the sum of all the nodes in the sectors contained in the window. This is summed using the frequency table set up before.

The location, of the "window" at any time in the grid, is given by the bar numbers of its bottom left sector, as shown in Figure 4.12

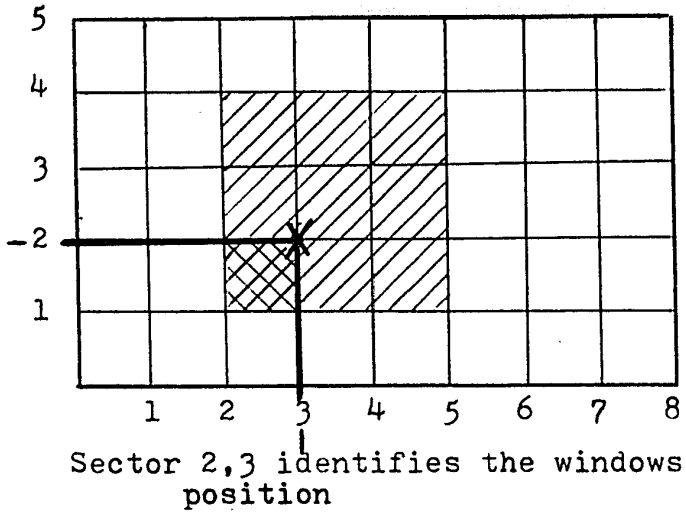


fig. 4.12

4.28 Concentrator Location in a Window

Once we have picked a set of windows in which we wish to locate candidate concentrator sites, we then must find the exact position of the candidate concentrator, in the window. This can be done by using a straight center of mass approach in the X,Y directions. Once the center X_C , Y_C is located we then have one of two possible choices for a site (the present algorithm uses the second).

1) The exact location X_C, Y_C.

2) The closest user terminal.

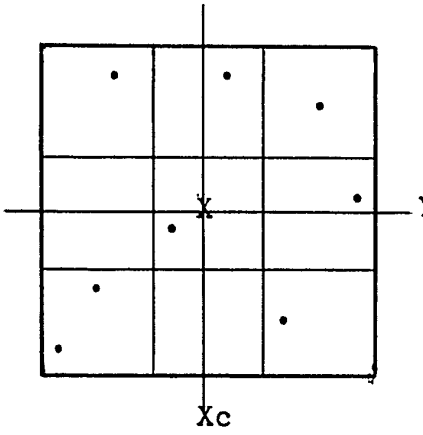
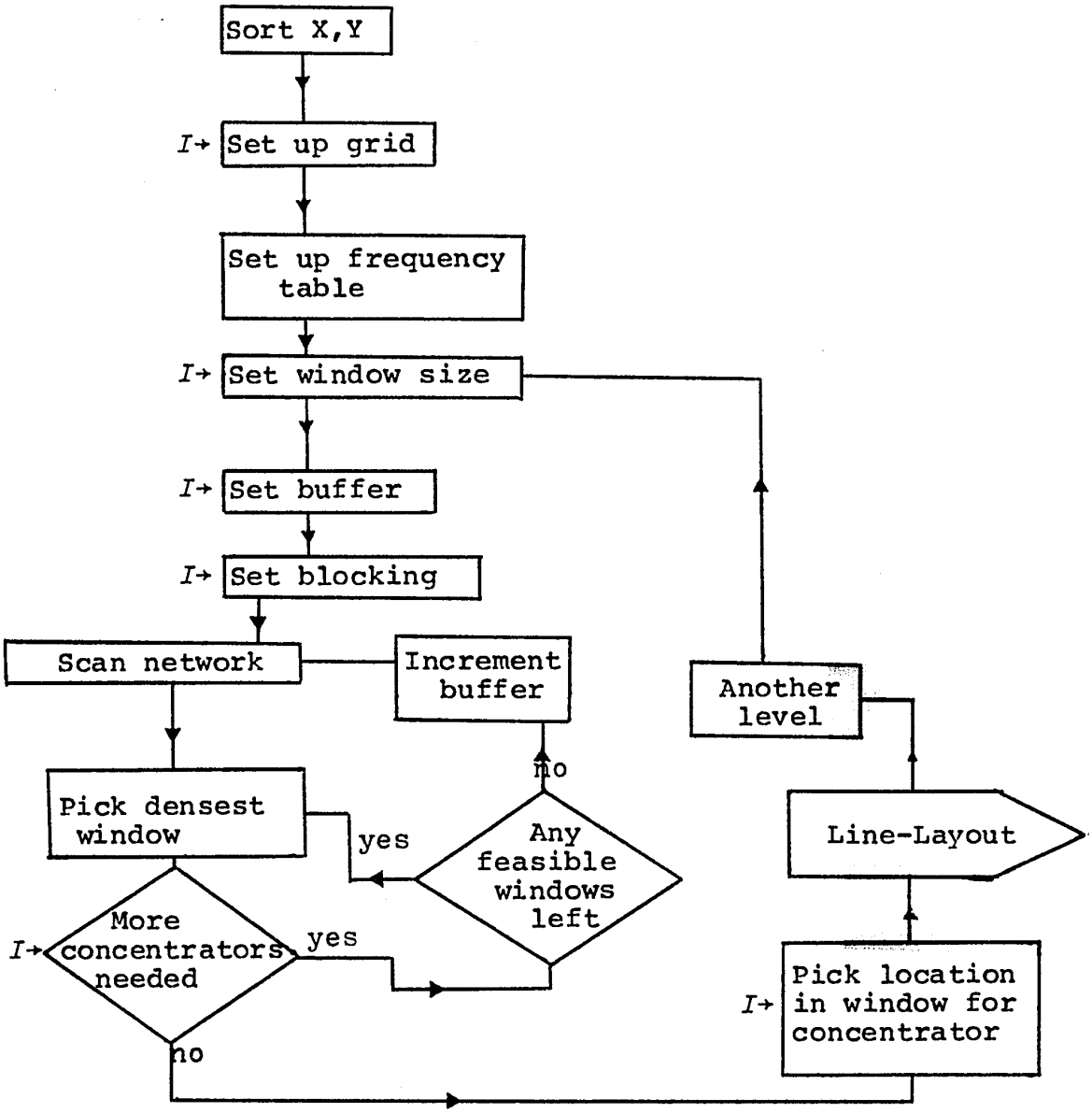


fig. 4.13

Another possible method that may yield slightly better results, but at the cost of an increase in execution time for the process, is to evaluate the cost benefit of having the concentrator located at each node in the window in turn, with the use of equation 2.1, as given in section 2.12.

The savings here in terms of a better layout are questionable in view of the increased computation time needed. We therefore chose the first method.

4.3 Flow Chart



The letter "I" indicates points at which a designer could interact with the process.

fig. 4.14

Step 1: Sort the nodes into two lists, showing their relative X and Y positions.

Step 2: Set up the grid lines as described in section 4.22.

Step 3: Count the number of nodes in each sector and record this information in the frequncy table.

Step 4: Set 'window' size for this pass.

Step 5: Set initial 'buffer' size (no overlap).

Step 6: Block off areas around any higher level units already in position.

Step 7: Scan network with the set 'window' and 'buffer'. Record the density of each 'window' position.

Step 8: Pick the 'window'position with the highest density.

Step 9: If no more candidate sites are needed proceed to Step 12, if more sites are needed continue.

Step 10: If another feasible candidate exists goto Step 8 if not goto Step 11.

Step 11: Increment buffer to allow an overlap.If buffer limit is at any time exceeded print warning and STOP, if not goto Step 7.

Step 12: Pick exact sites of concentrators in the windows.

Step 13: Proceed to next process of algorithm.

Step 14: If another level of concentrators are needed, update necessary tables and proceed to step 4.

4.4 Performance

4.41 Memory Requirements

The clustering process requires a minimal amount of memory as no distance matrix is needed. The total requirement is given by equation 4.6.

$$M = 10N + 2B + B^2 \qquad \text{eq. } 4.6$$

Where B= number of bars

N= number of nodes.

M= number of bytes needed. On an IBM 360/65 the word length is four bytes, therefore it would need 4M bytes of core.

Since the number of bars would rarely, if ever, go over fifty, the B squared term would not cause any problem. Also the B terms are semi-independent of the number of nodes in the network (see grid section).

Further memory savings came from the use of half words on the computer for most of the bookkeeping lists.

Also many of the lists can be destroyed after each pass through the process. Thus, we would use existing list room from another part of the algorithm (e.g. REW) to store these intermediate results.

4.42 Execution Time

The clustering processing time depends on four factors:

1) number of sectors in the network.

2) number of concentrators to be located.

3) size of window.

4) type of sort routine used.

The first three factors are linearly dependent on the number of nodes in the network in terms of processing time. This makes the process very efficient, in terms of execution time, when dealing with large networks (see Figure 4.15).

The fourth factor is the most time consuming of the process. Typically 70 to 80% of the first clustering pass is taken by the sort routine (e.g. for 320 nodes sorting

took 4.5 seconds, locating 5 concentrator sites took about 1.2 seconds). However, there are some important points to remember here:

1) Once we have sorted a network once, it need never be done again. Only the listing of the results need be stored for future use.

2) The sorting, even if run each time is only needed in the first pass. Successive applications of the clustering process to locate other levels need not sort again.

3) A good programmer could shorten the execution time of the sort appreciably.

Figure 4.15 shows the execution time vs number of nodes, behaviour of the process.

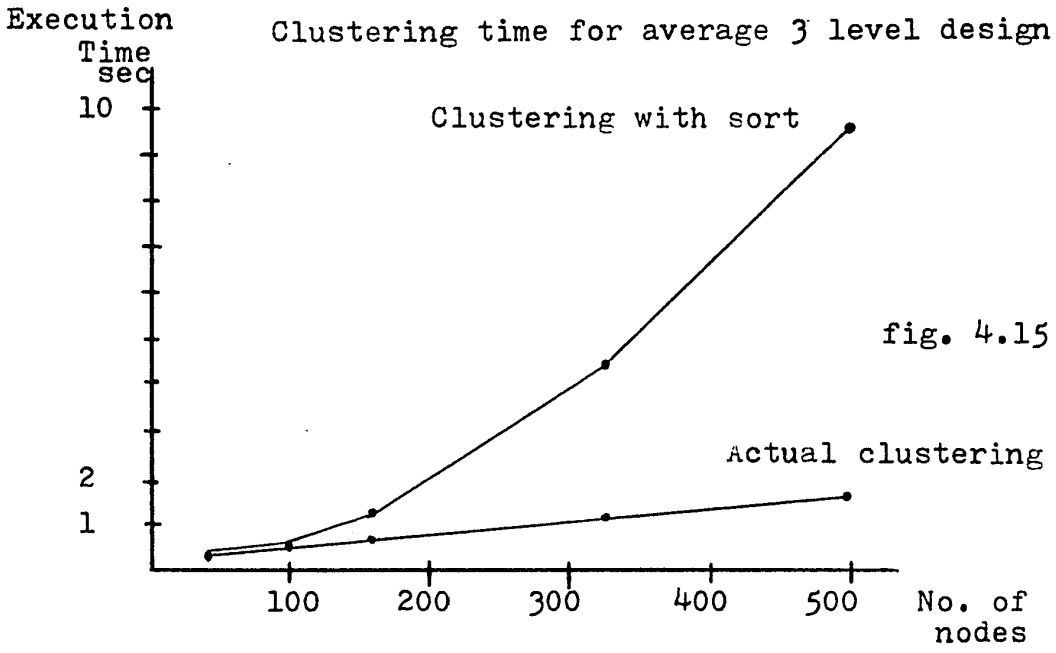


fig. 4.15

4.5 Example

Consider a network of 500 randomly positioned nodes, each with a randomly assigned traffic t_k. The total traffic is $(\sum t_k)$ 1253 units. The concentrators to be located have a minimum traffic capacity of 90 units and a maximum of 120. The following outlines the steps taken to locate the concentrators.

1) Determine the maximum size of grid to be used; since design parameter K should be ≈ 1 (see section 4.22), the grid is set to have 484

sectors (22 by 22 grid).

2) Next the process sets up the grid as described in section 4.22. The resulting grid is found to have 294 sectors with at least one node , as shown by the frequency table in Figure 4.16. From the figure we can also see that the grid is actually 22 by 21 sectors, instead of 22 square. This is due to the grid adjusting itself for the particular network as described in section 4.22. The number of concentrators is also found at this point and is equal to 13 (see equation 4.3). We also set V to be equal to 500/294 or 1.7.

3) We now calculate the window size to be:

$$U = \frac{500}{14} = 35.7$$

Thus:
$$X = \left\{\sqrt{\frac{U}{V}}\right\} = \frac{35.7}{1.7}$$
$$= \{4.58\}$$
$$X = 4$$

Thus the window will be four by four sectors.

4) After blocking out the region surrounding the

Frequency Table

1	1	1	0	0	1	0	1	0	0	0	2	1	0	1	2	0	2	0	0	0
2	0	2	4	3	1	0	3	4	2	0	2	2	1	1	0	1	2	3	1	1
4	2	0	2	1	3	0	3	1	2	1	4	1	2	0	1	3	1	1	2	0
2	0	1	3	4	1	0	0	1	1	2	2	1	0	0	4	1	2	0	4	1
1	3	1	1	1	2	0	1	0	1	1	3	1	0	1	0	1	1	1	0	0
2	0	2	0	3	2	0	0	4	0	4	3	1	3	3	0	2	2	1	0	0
0	2	3	0	0	3	1	0	1	0	1	2	0	1	0	0	3	1	1	2	0
1	0	3	2	0	1	3	2	1	3	3	2	2	2	0	1	3	2	4	0	2
0	1	0	2	6	3	0	2	1	1	0	0	1	1	1	3	1	0	1	1	0
0	1	1	2	0	1	0	1	0	1	0	1	0	1	2	2	0	1	0	3	1
1	3	1	2	0	2	1	3	0	0	3	1	0	2	0	1	2	0	1	2	1
2	2	1	0	2	4	1	0	1	0	0	1	1	3	1	0	0	0	1	1	0
0	0	0	1	0	3	1	1	1	0	2	0	0	0	0	0	0	2	1	0	1
0	0	1	0	0	1	3	0	0	3	1	0	0	0	0	1	0	1	1	1	2
0	0	1	0	0	3	0	2	0	1	0	0	0	0	2	0	1	1	1	1	1
0	1	1	1	1	1	2	1	1	1	0	1	2	1	1	3	1	1	3	2	2
0	1	3	0	0	1	0	0	2	0	0	2	2	0	0	3	1	1	1	3	1
2	1	2	0	1	0	0	1	1	0	0	0	1	2	1	0	1	0	3	0	1
1	0	1	2	1	1	2	2	2	1	2	1	0	2	2	2	2	1	0	1	1
0	1	3	1	1	0	0	1	1	2	1	0	0	3	0	0	2	1	2	1	1
0	0	1	2	2	0	0	0	1	2	2	1	0	3	0	0	1	0	0	0	0
2	0	0	1	3	0	0	1	3	1	0	0	1	2	3	0	0	1	0	0	0

□-RDB Blocked Area

fig 4.16

Column	Row	Density	Actual Node
9	17	30	224
3	18	30	482
3	14	30	142
16	14	25	113
17	18	24	199
18	6	23	164
14	4	22	293
8	1	22	61
6	9	22	58
3	1	19	402
13	19	17	484
11	13	17	174
7	13	17	156
1	10	17	262

- Node's Label

fig 4.17

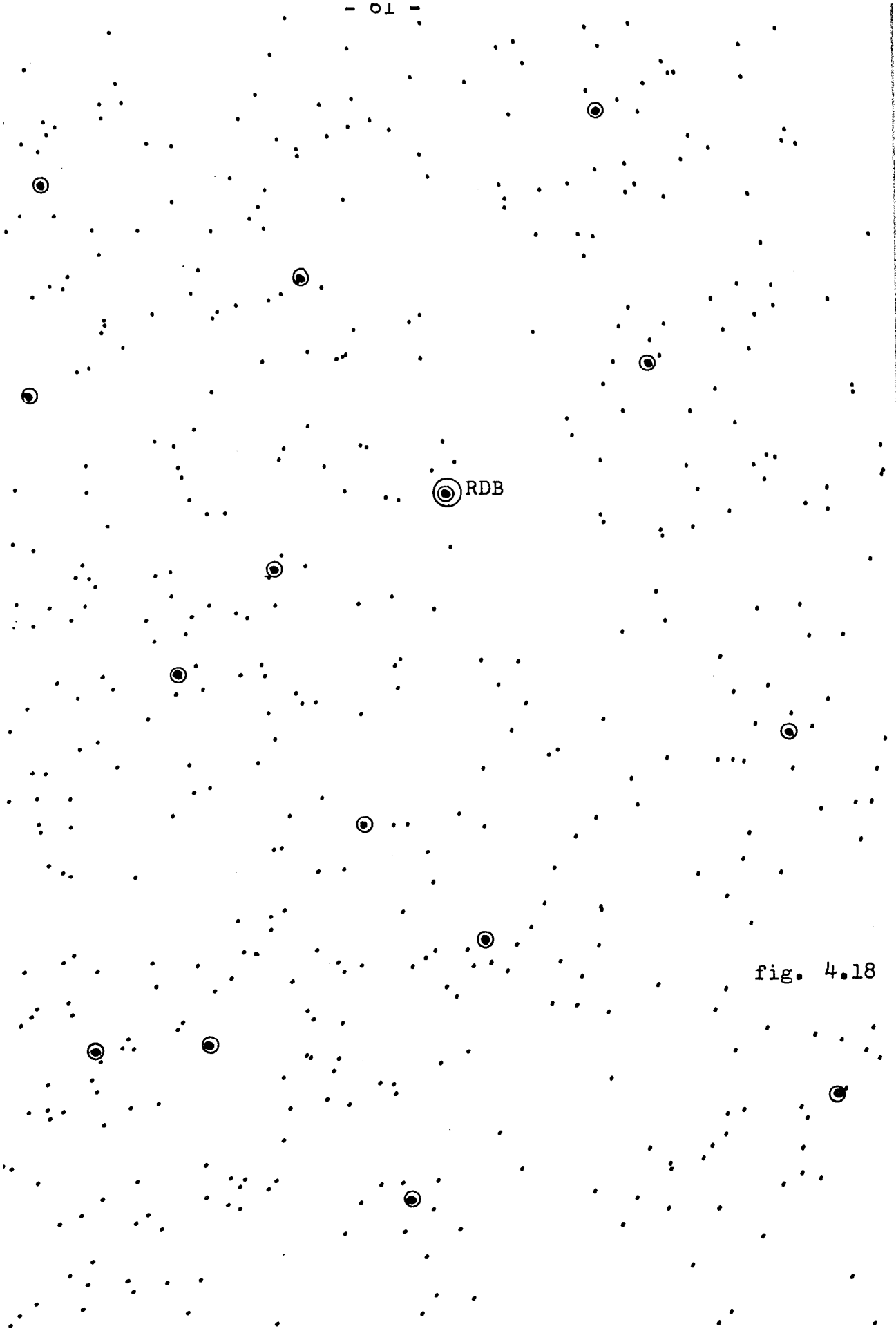


fig. 4.18

Initial Concentrator Locations

RDB, we scan the frequency table with the window to find the the areas that should hold an concentrator.

5) Figure 4.17 shows the windows chosen and their densities.

6) We now find the best terminals in each window, according to section 4.28 and thus construct our initial candidate concentrator list. It should be noted that 14 concentrators were identified by the process (RDB is not considered a member of this list). The reason for this is explained in section 4.27.

7) Figure 4.18, shows the concentrator locations in the actual network.

4.6 Summary

We now have a process by which we are able to locate a set of possible candidate concentrator locations. From this set of concentrator locations we must now choose a final set of concentrator locations. This is done by the next two processes REW and the 'SELECTDROP' sections. We

make no claims of optimality of the process in either results or time. Only that it represents a good engineering compromise between exactness and speed of execution.

CHAPTER 5

MULTIDROP LINE-LAYOUT PROCEDURE

5.1 Introduction

In this chapter we outline our multidrop line assignment technique. It is based on an algorithm first proposed by Essau-Williams in [10]. There have been, however, many key modifications done to the algorithm, to reduce the overall time and memory requirements.

One of these modifications is the use of the nearest neighbour concept proposed by Kanaugh [11]. Other equally important modifications introduced include a listing procedure, that cuts our bookkeeping requirements, and the introduction of the feasible joining area concept to the network design.

The process was designed to be used in large networks, with a set of concentrator devices. This last point is important for, as will be shown later, the algorithm performs more efficiently as the number of concentrator devices increase(see section 5.41).

All these various parts have been put together to give us a very fast and efficient algorithm for creating the multidrop line-layout between users and concentrators.

5.2 The Process

5.21 Supergroups

After running the first pass of the clustering process, we end up with a set of possible concentrator locations. The first step of the multidrop algorithm is to take this set and then assign each node in turn to the concentrator unit "closest" to it (unit that has the minimum direct link cost between itself and the node: thus we form supergroups). Next we check the "distance" (not geographic distance but a function of: determines cost of that link) between each node and the other members of the network. If the "distance" between two nodes is greater than the "distance" of one of the nodes to a concentrator, we set the "distance" between them to -1. This in effect disallows any further consideration of joining these two nodes, and thus creates our feasible joining areas. The feasible joining areas are defined as circles, centered at a node i, and with radius equal to the "distance" to i's closest concentrator. Nodes outside the the area are not considered for joining with node i. Figure 5.1 shows how these two groupings would appear in the network.

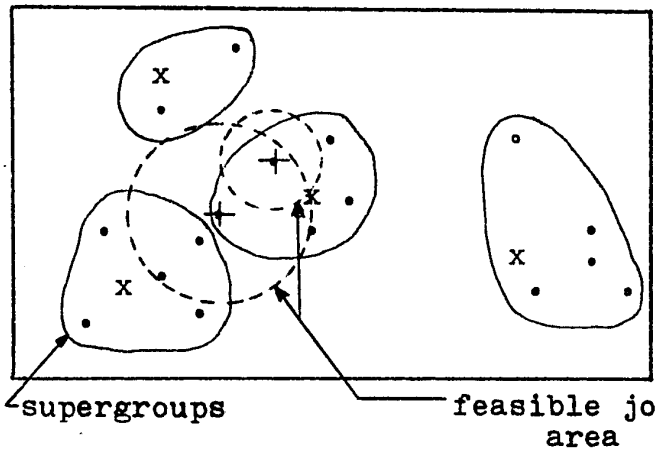


fig. 5.1

It is important to remember that these are not rigid groups and that a node can "jump" from group to group as the algorithm progresses, provided that a more economic design would be achieved by the move.

5.22 Nearest Neighbours

As shown in Figure 5.1, we now have defined a 'feasible joining area' for each node. It is from this area that a node's nearest neighbour will now be selected. The following relationships between a node i, and its nearest neighbour must B(i), always hold:

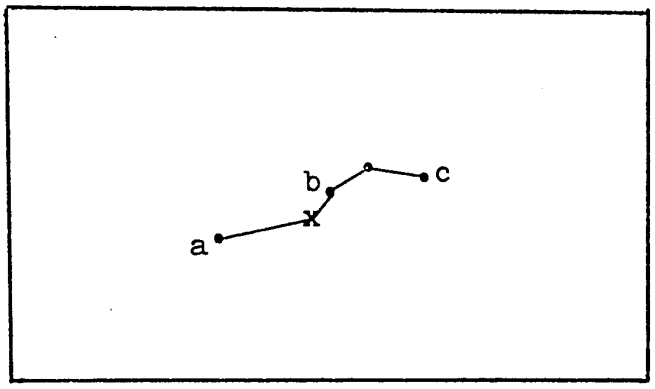
1) A node's nearest neighbour must always be the "closest" feasible node.

2) The joining of a node and its nearest neighbour should not result in the formation of a closed

loop.

3) As shown in Figure 5.2, the "distance" of a node to its concentrator depends on whether or not it is the end of a line or just a member of a line.

Definition of node's effective 'distance' to a Conc.



C(a,x)=C(a,x)
but
C(c,x)=C(b,x)

fig. 5.2

Therefore the joining of a node to its nearest neighbour must always result in its effective "distance" to a concentrator remaining the same or being reduced.

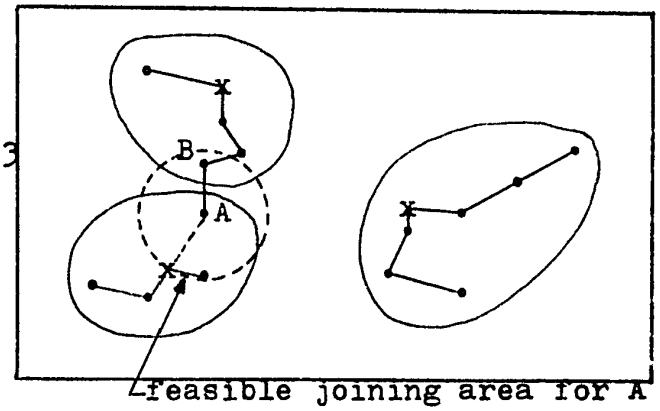
4) A node may not have a nearest neighbour such that the joining of the two would violate any of the constraints outlined in section 5.24.

By following these relationships we reduce the number of possible connections and therefore our design time. We also approach the problem in such a way that parallel processing of the network is possible. By

parallel processing we refer to the fact that the entire network is being worked on at any time. This is unlike the serial approach where once the network is subdivided, each subgroup is designed in turn with no further regard for the rest of the network.

Although the latter approach saves on memory and some execution time, problems arise in the fringe area between groups as shown in Figure 5.3.

fig. 5.3



Here node A and B belong to separate groups. Given the typical serial approach the link A-B would never be formed. However, by the use of the nearest neighbour concept and the 'feasible joining area', we can allow for this sort of fringe interaction to occur. The penalty is the requirement of more memory (see sections 5.42 and 5.43). The benefit is the possibility of a more economic design.

5.23 Joining

The sequence of joining is determined by equation 5.1. This equation is applied to all nodes not yet joined to its nearest neighbour.

$$R_A = C(A,Z) - C(A,B(A))$$

Where: $C(A,Z)$ is the "distance" (cost) of connecting A to its closest concentrator Z,

eq. 5.1

$C(A,B(A))$ is the "distance" from A to its nearest neighbour.

The largest R_A is chosen and the corresponding joining is made. It is checked to insure that it is a valid joining and then, if it is, all tables are updated (see section 5.3). The algorithm next checks, and corrects if necessary, any R values that have been affected by the joining (A to B(A)). It then selects the largest R from the modified table and so on.

5.24 Constraints

The joining of a node to its nearest neighbour is subject to the following constraints imposed by the designer:

1) Maximum traffic of low speed lines.

2) Maximum traffic of high speed lines. (there are in fact two or more different types of high speed lines in our case)

3) Maximum number of nodes that a concentrator can service. This can be set for each concentrator unit separately, or as a group.

5) Maximum number of input ports to a concentrator (except RDBs).

In the first four cases these constraints are checked as a network is connected and the network is not allowed to grow in such a way as to violate them. This is not so in the fifth case. We correct for this error, only after the initial assignments have been made. The actual correcting process is outlined in section 5.35 in the 'CORECT' subroutine.

5.3 Linedrop Implementation

5.31 Matrices in Implementation

The following is a list of the variables and matrices in the line-drop implementation and their uses.

N: the number of nodes in the network (including concentrators and RDBs).

NR: total low speed line traffic capacity.

NOM: size of the supernode set S, which contains all RDBs and concentrators of all types.Concentrators of different levels will require a small amount of preprocessing to assign the lower level units to the higher ones. This processing is handled in step 5 of the algorithm.

$MT(S_I)$: is the maximun traffic that S_I ($S_I \in S$) is allowed to carry. Where S_I represents a specific concentrator.

$MNL(S_I)$: is the maxiumn number of nodes that concentrator S_I is allowed to handle.

$MIL(S_I,1)$: the maximum value of input lines to the concentrator S_I.

MIL(I,J): contains information concerning the multidrop line that node I is connected to. Column one stores

the traffic in the line at node I. Column two stores the number of nodes already connected to the line. And column three to J store the labels of the nodes connected to the line when it joins node I.

PV(I): contains the number of the last node of the line that node I belongs to.

CV(I): indicates the number of nodes connected to node I.

M(I): identifies nodes that are members of S, and M(1) is the RDB.

B(I): identifies the next nearest neighbour of node I.

XA,YA,TV(I): identify the X,Y co-ordinates of node I and its generated traffic TV(I).

C(I,J): "distance" (cost) matrix for the network.

JS(I,2): connection table for the final result; connect node I to node JS(I,1).

CVC,CONN(I): holding matices used in the subroutine 'CORECT'.

5.32 Main Program

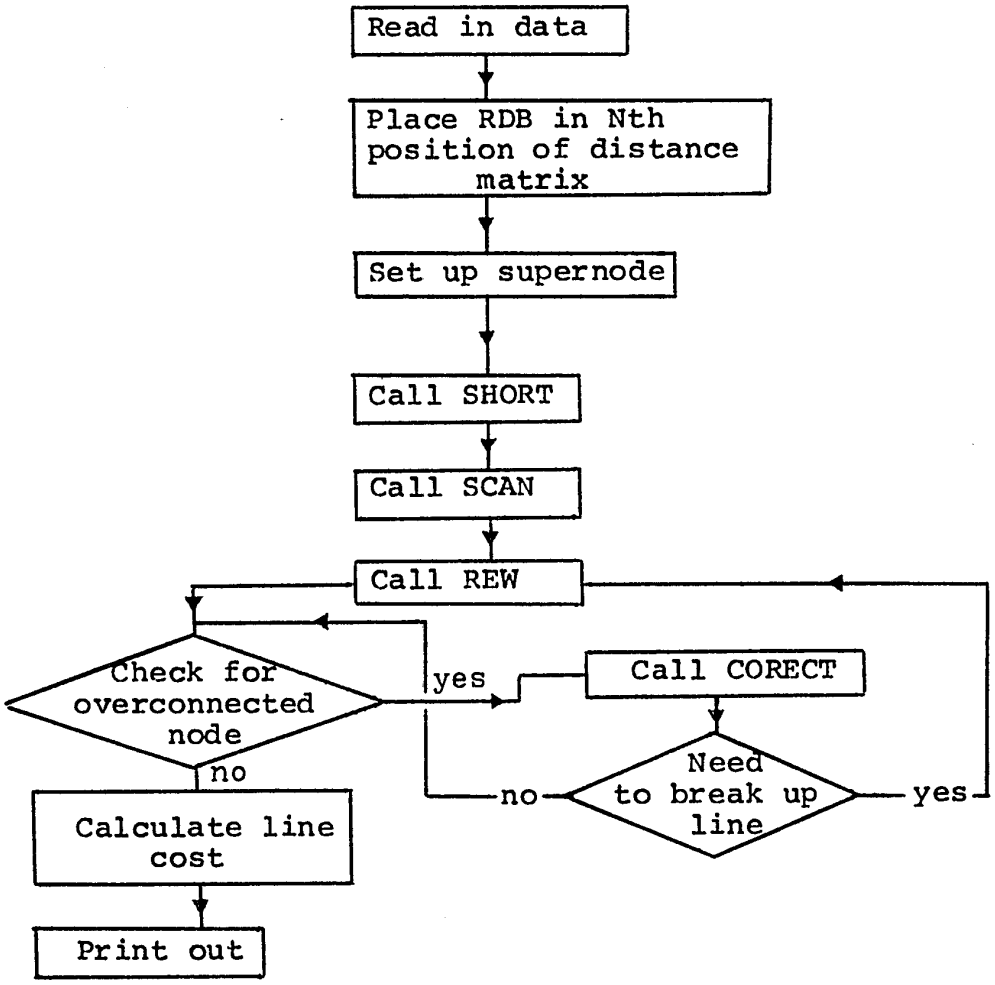


fig. 5.4

Step 1: Place RDB in the Nth position of XA,YA, and TV matrices.

Step 2: Find the "distance" between all pairs of points by calling subroutine 'DIST'.

Step 3: Set up supernode set S, by setting the "distance" between all concentrators and RDB equal to 0.0. When considering multilevel networks this step would also assign the concentrator of one level to a unit above it.

Step 4: For each node find its closest member of the set S and call that the RDB of that node.

Step 5: Call routine 'SHORT' so as to make the "distance" from node A to node B equal to -1 if A is closer to a member of S then it is to B.

Step 6: Call 'SCAN' and find the nearest neighbour of each node.

Step 7: Call 'REW'.

Step 8: Check if any member of S (other than RDB), has too many lines coming into it. If it does not goto Step 11.

Step 9: Check if 'CORECT' can keep line as a whole; if so goto Step 8.

Step 10: If the line has to be broken up reset variables and call 'REW'. Then goto Step 8.

Step 11: Calculate cost of connecting network, taking into account the different costs of the high and low speed lines.

Step 12: Print out results or have them sent to a display unit (see Appendix A).

5.33 Subroutine REW

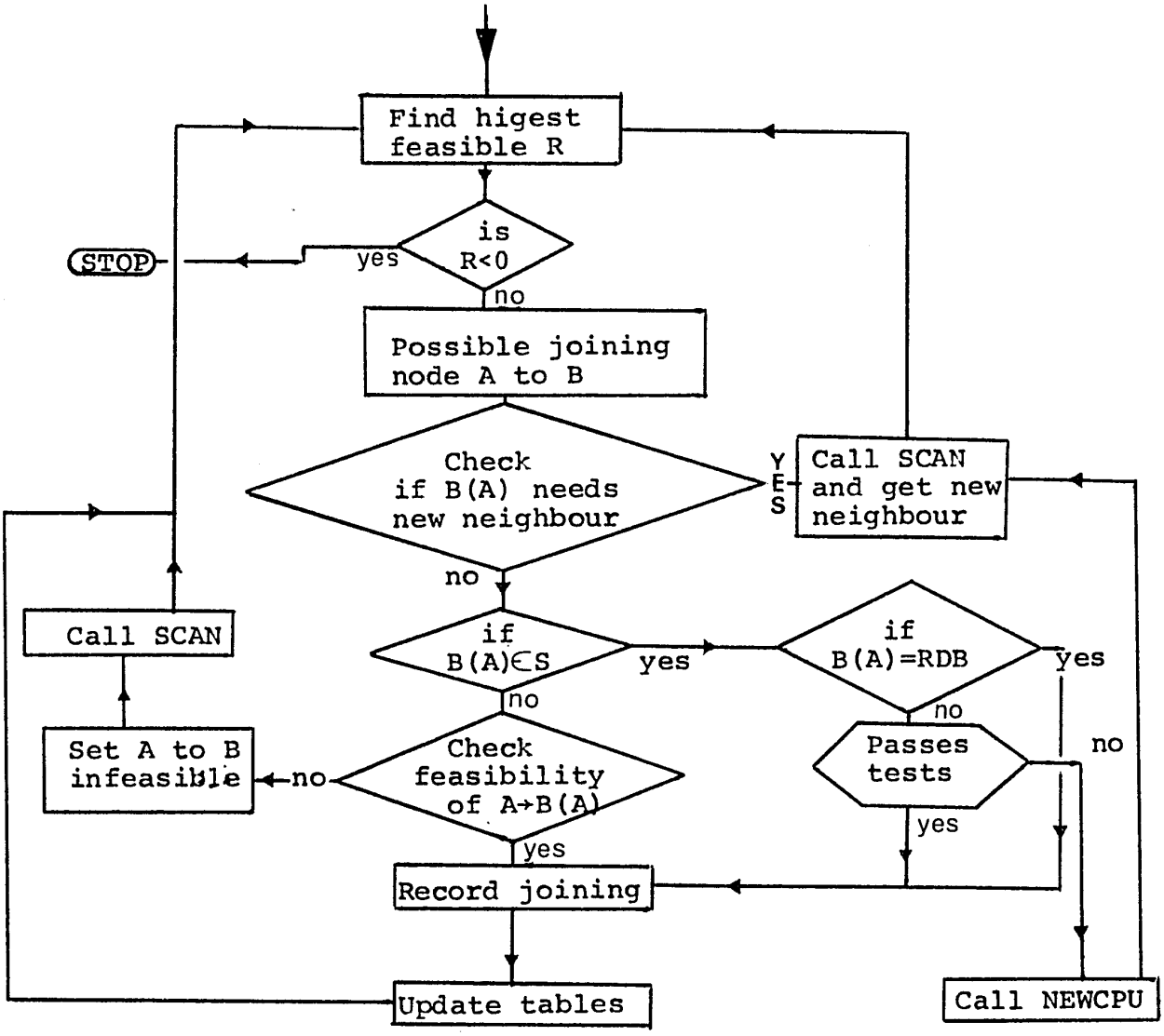


fig. 5.5

Step 1: Pick the best joining by finding the pair of nodes with higest value for R, where R= C(A,N)-C(A,B(A)) and B(A) is the closest neighbour of A. We also check to see that by joining A and B(A) we do not violate the following tests.

i) The joining cannot form a loop in the network.

ii) The "distance" between the two joined points is not equal to -1, since this would indicate that the joining has already been disallowed for some reason somewhere else in REW.

Step 2: Check if R is greater than 0.0, if not STOP.

Step 3: We now have a possible joining of the node A to B(A). It must be remembered that node A could in fact be the end point of a multidrop line, and thus represent a set of nodes. But as far as the routine goes, it is one node with traffic and other values equal to the sum of the individual values of all members of that line. Likewise B(A) could also be a single or group point.

Step 4: Check CV and B matrices to see if a new closest neighbour is reqiured for B(A). If a neighbour is needed call 'SCAN'. An example of this occuring would

be if we wished to connect I to J, but J was not only I's closest neighbour but also I is J's closest neighbour. Thus we would require a new neighbour for J once I and J were joined.

Step 5: Check if B(A) is a member of S. If not goto Step 9.

Step 6: Check if B(A) is the RDB. If it is goto Step 11.

Step 7: Check if connecting the line ending at A to S member B(A) does not violate:

i) the number of nodes S(B(A)) can serve.
ii) the total traffic S(B(A)) can handle.
If tests passed goto Step 11.

Step 8: Call subroutine 'NEWCPU' to get a new member of S assigned for A. Also have a new closest neighbour assigned to A by going to Step 13.

Step 9: Check if connecting A to B(A) does not overload low speed line capacity. If it does not goto Step 11.

Step 10: Set connection A to B(A) infeasible(done by setting C(A,B(A))=-1) and goto Step 14.

Step 11: Record joining of A to B(A).

Step 12: Update all tables TV,PV,CV,MIL and record that all nodes that were connected to node A are now also connected to B(A).

Step 13: Return to Step 1.

Step 14: Call 'SCAN' to find new nearest neighbour for A and then goto Step 1.

5.34 Secondary Subroutines

SHORT: If the "distance" between node 'h' and node 'k' is greater than the "distance" from node 'h' to its "nearest" member of S (RDB and concentrators), then the "distance" C(h,k) is set to infinity (-1). This in effect disallows any further consideration of joining 'h' to 'k', but not of joining 'k' to 'h'. The latter is considered as a completely separate question.

SCAN: This routine takes a given node i and scans the "distance" matrix C for its closest feasible neighbour. The result is stored in B(i).

DIST: The subroutine calculates the "distance" between all points in the network.

SELECTD: Calculates the "distance" between a point I and all other points in the network.

NEWCPU: If an assigned member of S to a node is no longer valid, then NEWCPU calculates the next closest feasible S member and assigns the node to it.

CORECT: In cases where the total number of input lines to a member of the set S exceeds its maximum value, CORECT reassigns some of the input lines to adjust the network to this constraint.

5.35 Subroutine CORECT

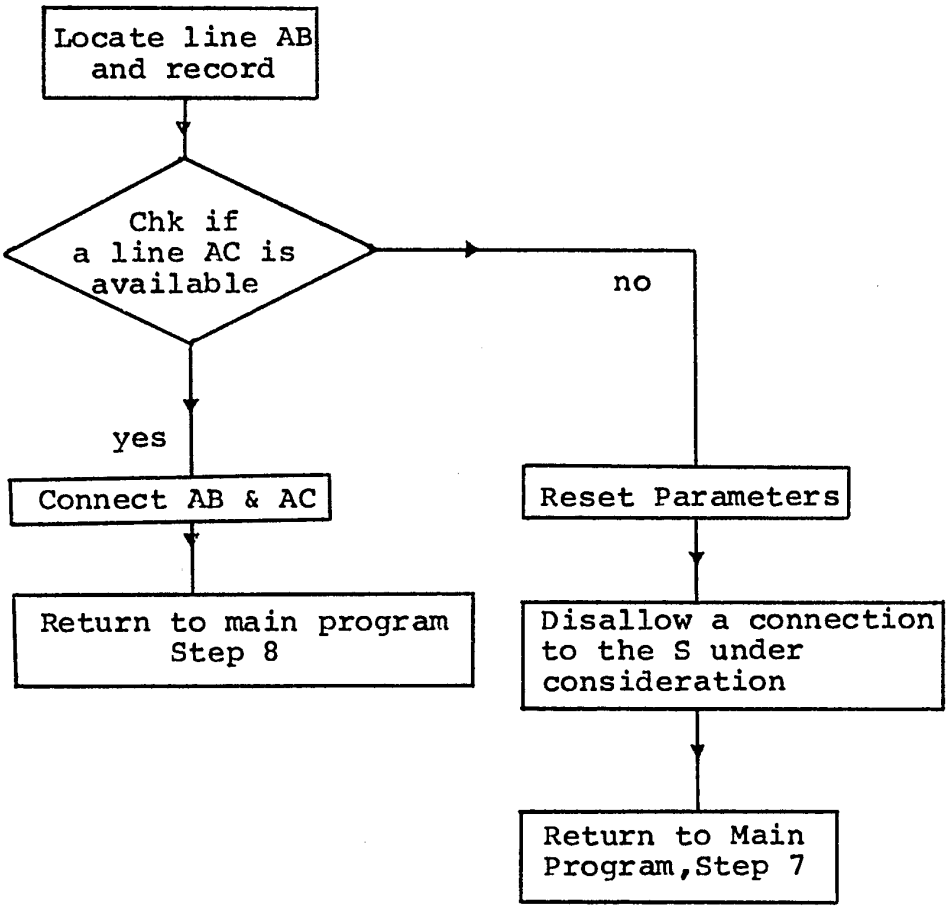


fig. 5.6

Step 1: Find the line coming into the overloaded node, with the least total traffic and call it AB.

Step 2: Check if another line (call it AC) already connected to the overloaded member of S can take the load of AB. If there are more than one AC type lines choose the one closest to AB's end point. If no AC found goto Step 4.

Step 3: Connect AB's end point to the endpoint of AC. Goto Step 7.

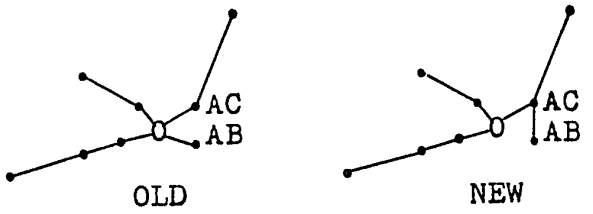


fig. 5.7

Step 4: IF AB cannot be connected to another line, reset all parameters CV,TV,C,B,etc for the nodes in the line ending at AB.

Step 5: Disallow a connection from a former member of AB to the member of S presently being considered. Also disallow any new connection that would cause a member of S to become overconnected.

Step 6: Reapply REW to the network, and then return to main program Step 7.

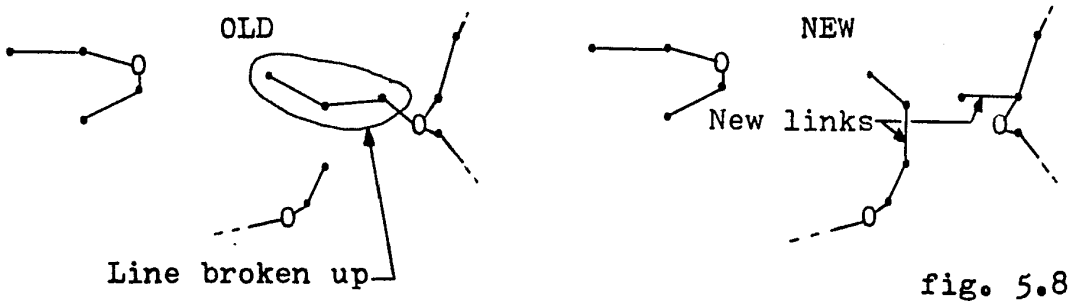


fig. 5.8

Step 7: STOP: Return to main program and check for any other overconnections.

5.4 Performance

5.41 Execution Time

REW has been found to be most effective when dealing with networks with a set S greater than 1.(e.g. a 60 node network with one RDB took 7.3 seconds of execution time while the same network with two additional concentrators took only 5 seconds, see Figure 5.9). This characteristic of REW is due to several reasons:

i) With a large set of S, the length of the low speed multidrop lines decreases and thus we encounter fewer nonfeasible joinings due to traffic overload.

ii) Joinings are only considered between nearest feasible neighbours. We also limit the choice of a node's nearest neigbour to a circle, centered at the node and radius equal to the "distance" from the node to its closest member of S. This boundary quickly disallows many possible false joinings from being considered. Also, as S increases in size, these circles decrease in size and thus we get even a better run. There is however a limit to the saving of computer time caused by a larger S set, as shown in Figure 5.9.

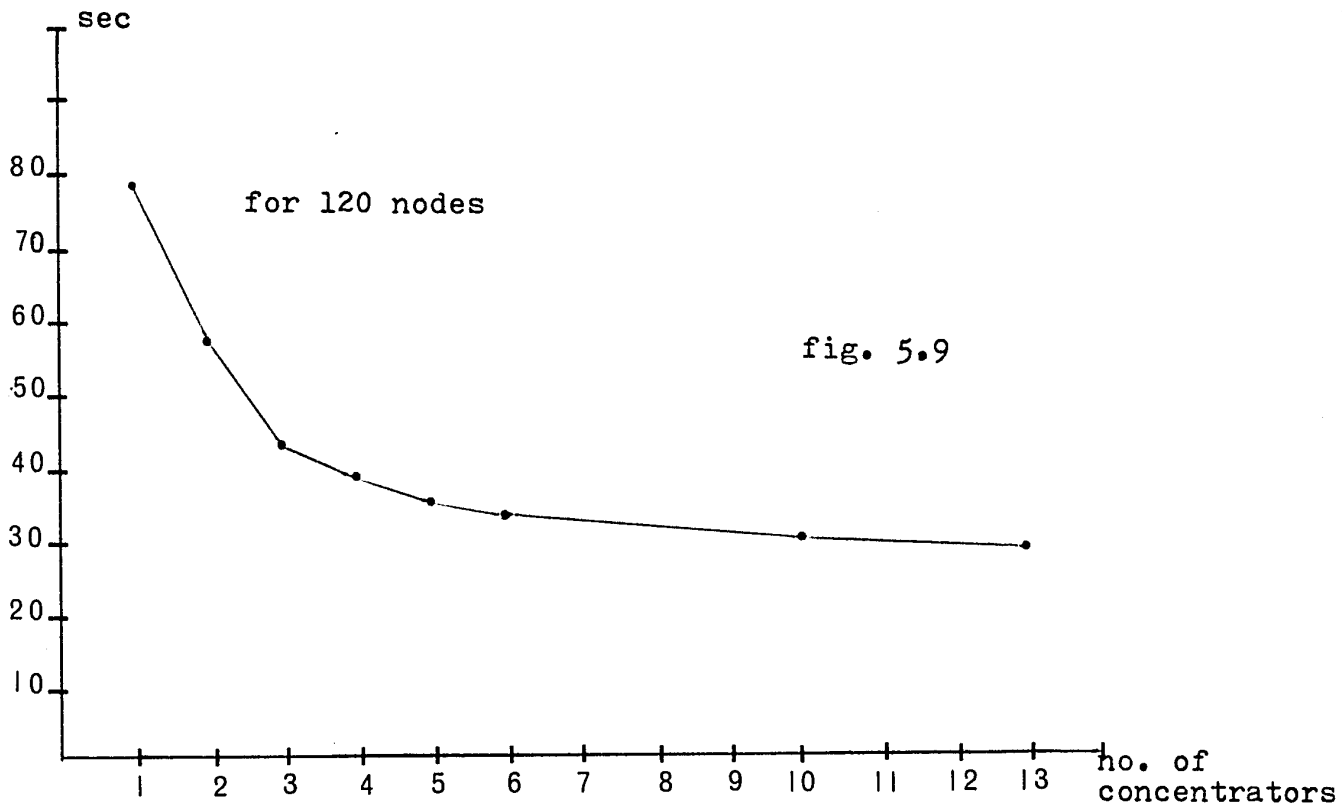


fig. 5.9

Typical execution time curve for line-drop process.

This localizing approach does not however restrict any feasible cost saving connection from occuring. It therefore lends itself ideally to our purpose.

Other time saving factors include minimum preprocessing, all nodes except the RDB retain original titles, thus we do away with the time and memory consuming "book-keeping" problem. Also no post-processing is normally required.

5.42 Present Memory Requirements

The memory requirements are approximated by the formula $N^2 + 13N + 5S$, where N is the number of nodes in the network (including RDBs and concentrators) and S is equal to the sum of all RDBs and concentrators. It should be noted that as N increases the formula can be approximated further to simply N^2 (N^2 term represents 70% of total memory storage requirements for a 40 node network,76% of a 60 node network, and 86% of a 120 node network. See Figure 5.10). An obvious area of further study would be to reduce this factor N^2, but at the same time not introducing a much larger execution time. But for now we are satisfied with saving execution time in lieu of

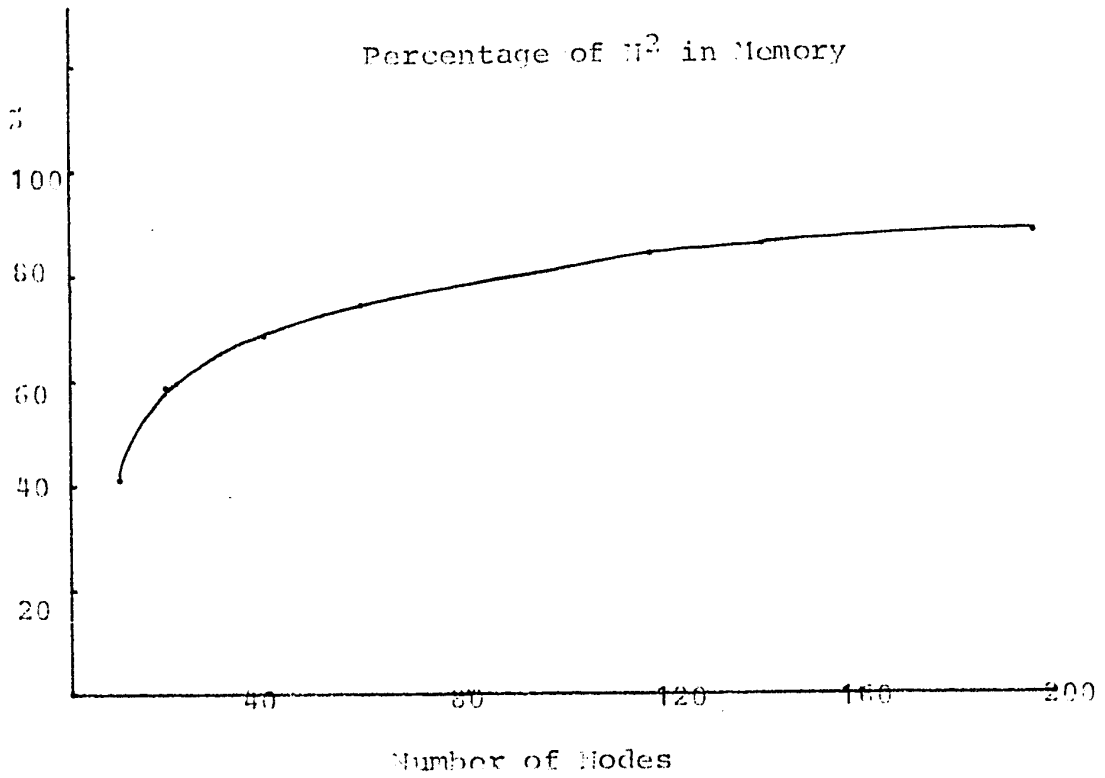

Percentage of H^2 in Memory
%
100
80
60
40
20
40
80
120
160
200
Number of Nodes

fig. 5.10

memory. One storage saving method that was employed was the use of half integer words, thus reducing most of the integer storage requirements by half.

5.43 Future Memory Requirements

Although not employed in the actual testing of the algorithm, we did develop and do some testing on a memory saving listing procedure near the end of the project. This new listing system took into account that we do not require all point to point "distances". Because we are using our feasible joining area, each node in fact could only be connected to a small subset of the nodes in the network.

Using this fact, we set up a three by Q matrix instead of a N×N (N being the number of nodes) to store the "distances" required. The first two columns are integer half words indicating the labels of the nodes under consideration. The third column holds the actual "distance" between the node in column one and the node in column two (see Figure 5.11).

NODES		DIST	
1	2	45.6	
1	3	17.8	Block 1
1	4	6.9	
	⋮		
2	1	45.6	
2	3	37.7	Block 2
2	4	4.2	
	⋮		⋮

Q = Block lenght * no. of nodes

Memory requirements = 2*Q

fig. 5.11

The listing is broken into blocks, the blocks being identified by the node label in column one. Therefore, if we needed all point to point "distances" this method would need 2(N-1)/N bytes of memory. But each node in fact needs the "distance" to only a subset of N. The size of this subset can be approximated by the formula N/S (where S is the number of all types of concentrators). This is not to say each block is equal in length only that N/S is the average length.

Now calculating the total memory requirement we find it to be (2(N-1)/S)N, or, if you take N to be large, $2N^2$ /S. As can be seen, as soon as we have more than two

concentrators in the network we save memory. In effect all we have done is remove the redundant information from our N*N "distance" matrix.

To give an idea of the amount of memory saved consider the case of a 500 node network. Using the present listing we would require one million bytes of memory (on an IBM 360/65). But if we used the proposed system, and assumed that on the average a 500 node network has 20 concentrators (see example Chapter 4), we find that our needs are cut to one tenth of our previous case (100K).

5.5 Example

Consider the case of a small ten node network, containing one RDB and one concentrator at node 6. The multidrop line-layout is found as follows.

1) Create a list indicating a node's closest concentrator (the RDB is taken as a concentrator). See Figure 5.12

2) Set up the "distance" matrix. If the "distance" from node i to node j is greater then the

distance" of i to its nearest concentrator, set the "distance" i-j to -1. The resulting table generates the feasible joining ares shown in Figure 5.12.

Node	Conc.
1	1
2	1
3	6
4	1
5	6
6	1
7	6
8	6
9	1
10	6

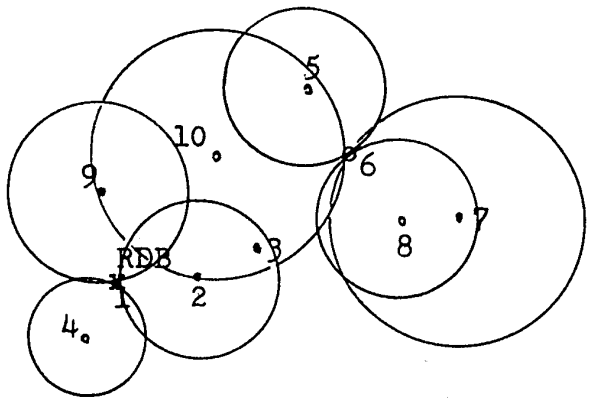


fig. 5.12

3) From the distance matrix find the initial nearest neighbour for each node, see Figure 5.13.

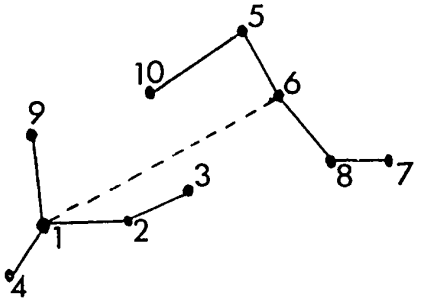
4) Proceed to create the line-layout under the given design constraints, and using the equation outlined in section 5.1.

5) Final topology is shown in Figure 5.14, note that node 3 was in supergroup one originally, but now is in group two. This cost effective group change

could occur because of the use of the feasible joining areas, which allow nodes the freedom to find the best connection. The final tables are given by Figures 5.15 and 5.16.

Node	N.N.
1	1
2	1
3	2
4	1
5	6
6	1
7	8
8	6
9	1
10	5

fig.5.13



--- High Speed Lines

—— Low Speed Lines

fig. 5.14

Node	Conc.
1	1
2	1
3	1
4	1
5	6
6	1
7	6
8	6
9	1
10	6

fig. 5.15

Node	N.N.
1	1
2	1
3	2
4	1
5	6
6	1
7	8
8	6
9	1
10	5

fig. 5.16

5.6 Summary

This procedure allows us to design multilevel networks using a multidrop line layout. It does not represent the fastest method of doing the multidrop assignment (the reader should refer back to the discussion of the serial vs parallel approach to processing in section 5.22), but it does allow us the degree of flexibility of design we desired, without a prohibative amount of memory or time requirements.

CHAPTER 6

'SELECTDROP' ROUTINE

6.1 Introduction

After the line-drop routine connects up the network we still do not have the most economical design that could be found by the algorithm. This is due to the fact that we chose only a candidate set of concentrators, not the final set. To check for possible redundant units, we first select from the concentrator set the ones that have a usage level below a preset level. Then taking each concentrator in turn we redesign that section of the network connected to it, to see if by dropping the unit a more economical design would be achieved. If a designer were hooked up interactively, we could allow him to choose the order in which concentrators would be considered, instead of doing it automatically. This would result in a saving in computer processing time by avoiding the consideration of concentrators that the designer deems necessary.

6.2 Methods

6.21 NEWCLUST

Developed by Dysart [3] this method takes multidrop lines and compresses them to single points and then moves these points from concentrator to concentrator, calculating the gain of each new joining as it is made (see section 2.43). As concentrators that are not cost effective are found, they are dropped, and this continues until no improvment in design cost can be found by further removals. This method, however, could not be made as interactive as we wished and therefore was not used. It would, however, be excellent if the automatic version of the algorithm was used.

6.22 SELECTDROP

This procedure was chosen as it could use facilities already built into the line-drop portion of IMULEC. It makes use of the algorithm's ability to correct for the number of input lines to a concentrator. Basically it takes the concentrator to be considered and declares it as being a concentrator with no input ports (effectively dropping it from the concentrator set). The routine then redesigns that part of the network that had been

connected to the concentrator. The new result is checked with the old one, and if it shows an improved cost it becomes the final design. If not, the network is reset to its original setup. As in NEWCLUST we used a scaled down version of Martin's [21] costing procedure, as outlined in his book.

Although this may seem time consuming, it should be remembered that in most cases no more then 30 nodes are connected to any one concentrator and at this size the line-drop process has a minimal execution time.

6.3 Summary

From our testing we have found that on the average the SELECTDROP procedure took approximately three seconds per concentrator considered. This figure takes into account the variations due to the level of the concentrator being studied and its 'usage' (percent of maximum capacity). We also found that dropping concentrators with a 'usage' level of approximately 45% or better did not result in a smaller design cost.

From our study of the SELECTDROP process we found that it also satisfied our desire for an interactive

system. The processing time needed to check a concentrator is sufficiently small as to allow it to be done on line with the designer. This point is critical since no designer would use the system if he had to waste large amounts of time, waiting for results.

CHAPTER 7

NUMERICAL RESULTS

7.1 Introduction

To give a basis of comparison with existing works, we ran the IMULEC algorithm for the three level case and compared the results with those of NEWCLUST. The NEWCLUST algorithm has been in turn compared, by its author, with two other known methods, those of Martin/Doll and Bahl & Tang (see Figure 7.6: reprinted with permisson of author).

To overcome the difficulty of coding NEWCLUST properly, we used the original program as written by Dysart. Also to correct for differences in programming ablility we used an optimizing compiler that was available to us.

We also include in this chapter, results on the network cost and execution time for the four level design of our test networks.

All the example networks used were randomly

generated, as also was the traffic at each node. We used four such examples for each network size, i.e., 100, 160, 320 and 500 nodes. The results reported are the averages over the four examples of each size.

7.2 Three Level Case

In this section we compare IMULEC and NEWCLUST in the three level case. That is to say, one RDB level, one level of concentrators and the user level. Figure 7.1, gives the time vs the number of nodes, for the clustering process, while Figure 7.2 gives the same for the line-layout process. Table 7.4 gives the relative costs of the different designs.

An example network of 500 nodes, designed using the three level IMULEC algorithm is given in Figure 7.3. As we have stated before the costing was a scaled down version of the costing outlined by Martin in his book [21].

7.3 Four Level Case

Here we show the performance of IMULEC in the four level case (one additional concentrator level). The networks used are the same as those used in the three

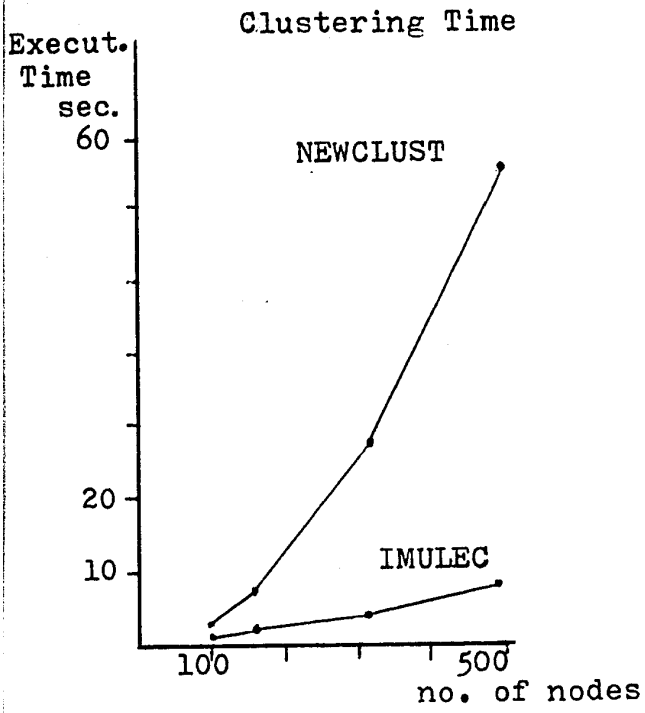

Clustering Time
Execut.
Time
sec.
60
20
10
NEWCLUST
IMULEC
100
500
no. of nodes

fig 7.1

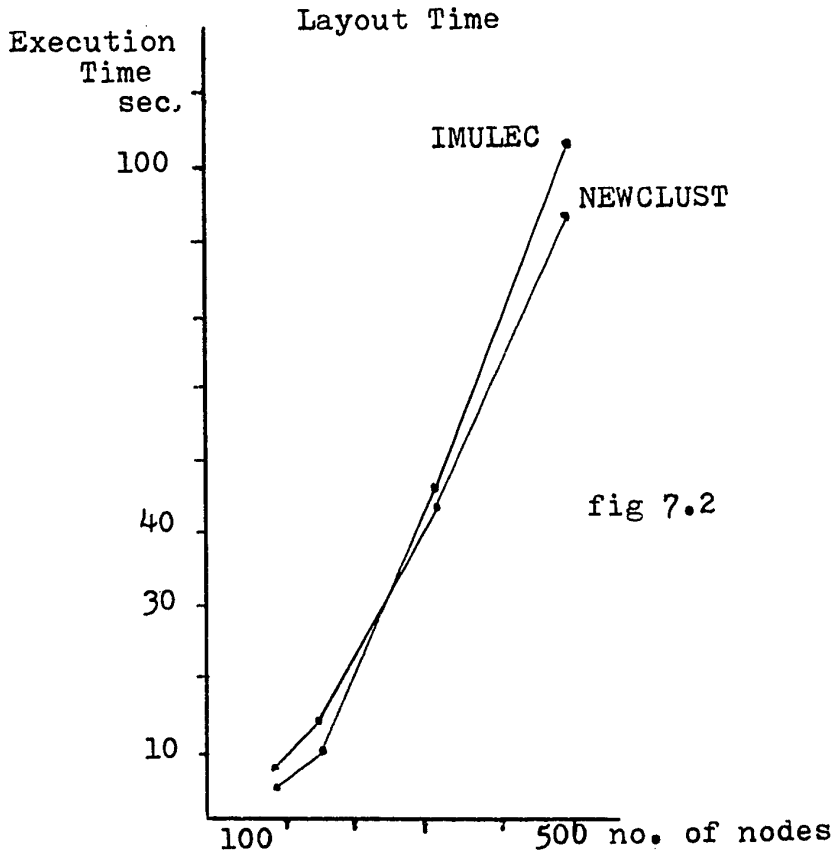

Layout Time
Execution
Time
sec.
100
40
30
10
IMULEC
NEWCLUST
100
500
no. of nodes

fig 7.2

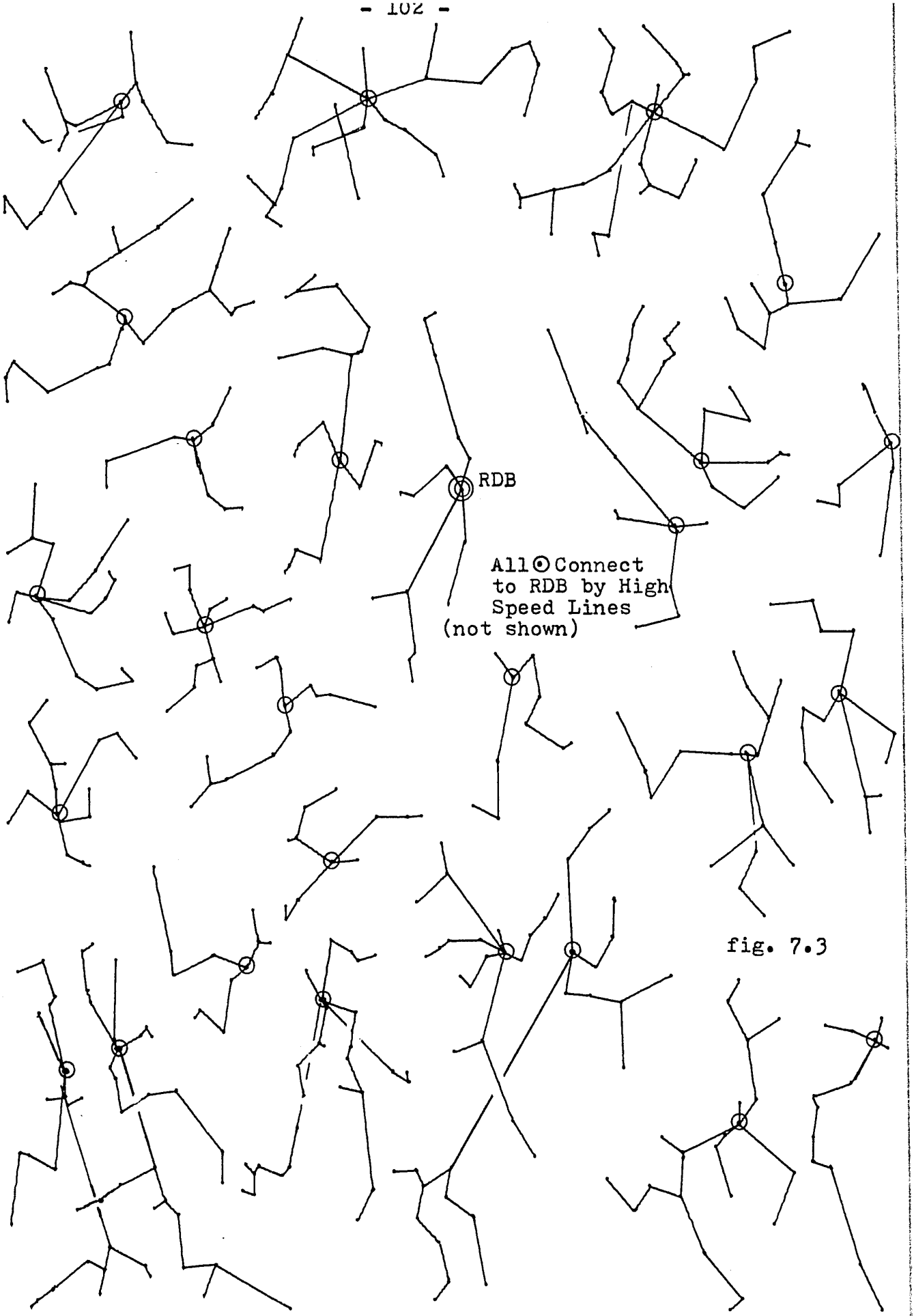


fig. 7.3

500 Node Network 3-Level Design

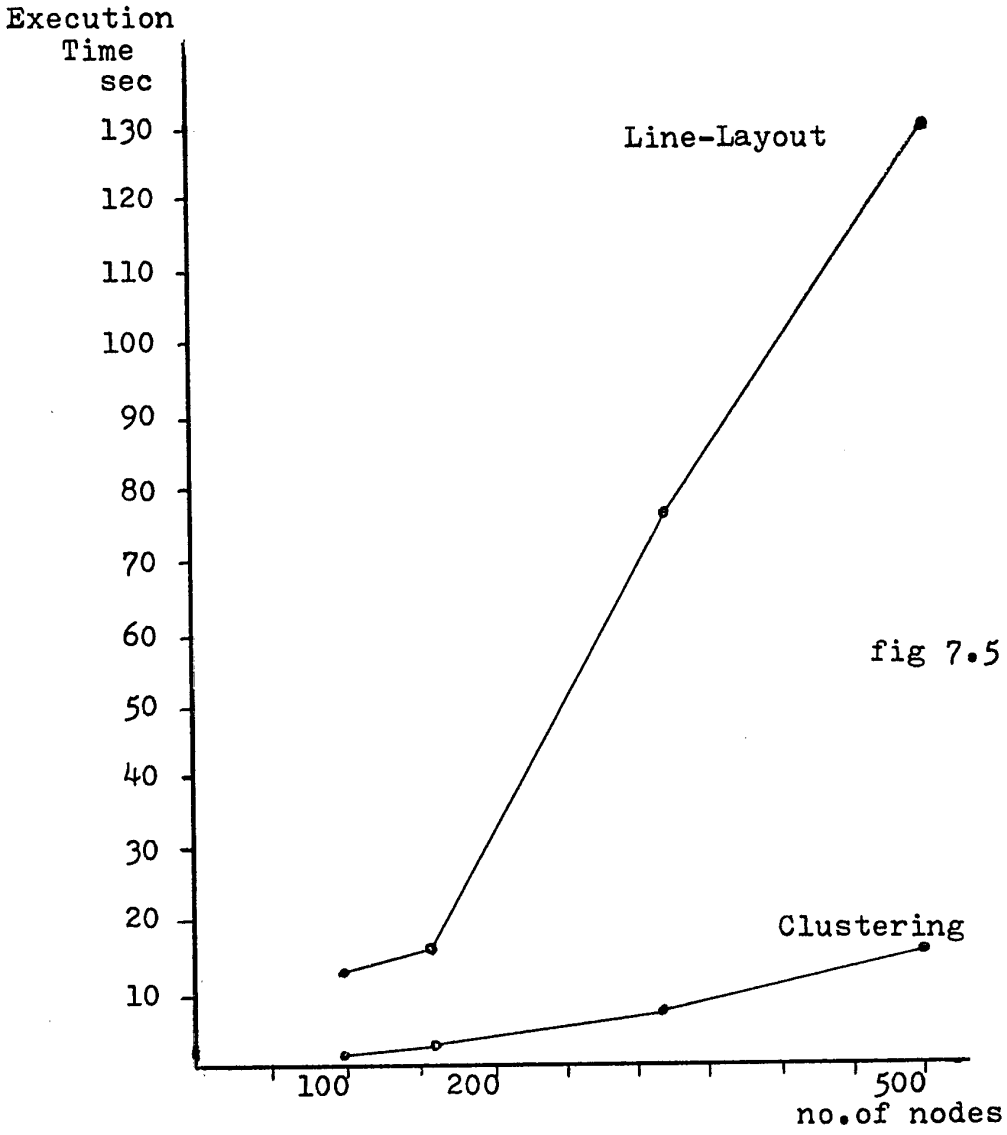


fig 7.5

Design Times for 4-Level Networks

level case.

Figure 7.5 shows the design time for the networks, while the costs are given in table 7.4. Figure 7.7 shows the same 500 node network as in Figure 7.3, only now we show the four level design.

Network Size	NEWCLUST 3-Level	IMULEC 3-Level		IMULEC 4-Level	
	Cost	Cost	R_1	Cost	R_2
100	2337.7	2310.4	1.17	2181.4	5.58
160	3265.6	3159.5	3.26	3057.5	3.23
320	5599.7	5532.1	1.21	5459.8	1.31
500	7642.2	7491.2	1.98	7270.7	2.94

Where: R_1 is the percent cost saving of IMULEC's design as compared to NEWCLUST.

R_2 is the percent cost saving of IMULEC (4-Level) over IMULEC (3-Level).

fig 7.4

Cost Comparison Table

7.4 Summary

All the test runs were done on an IBM 360/65 using an optimization compiler. One problem inherent to this

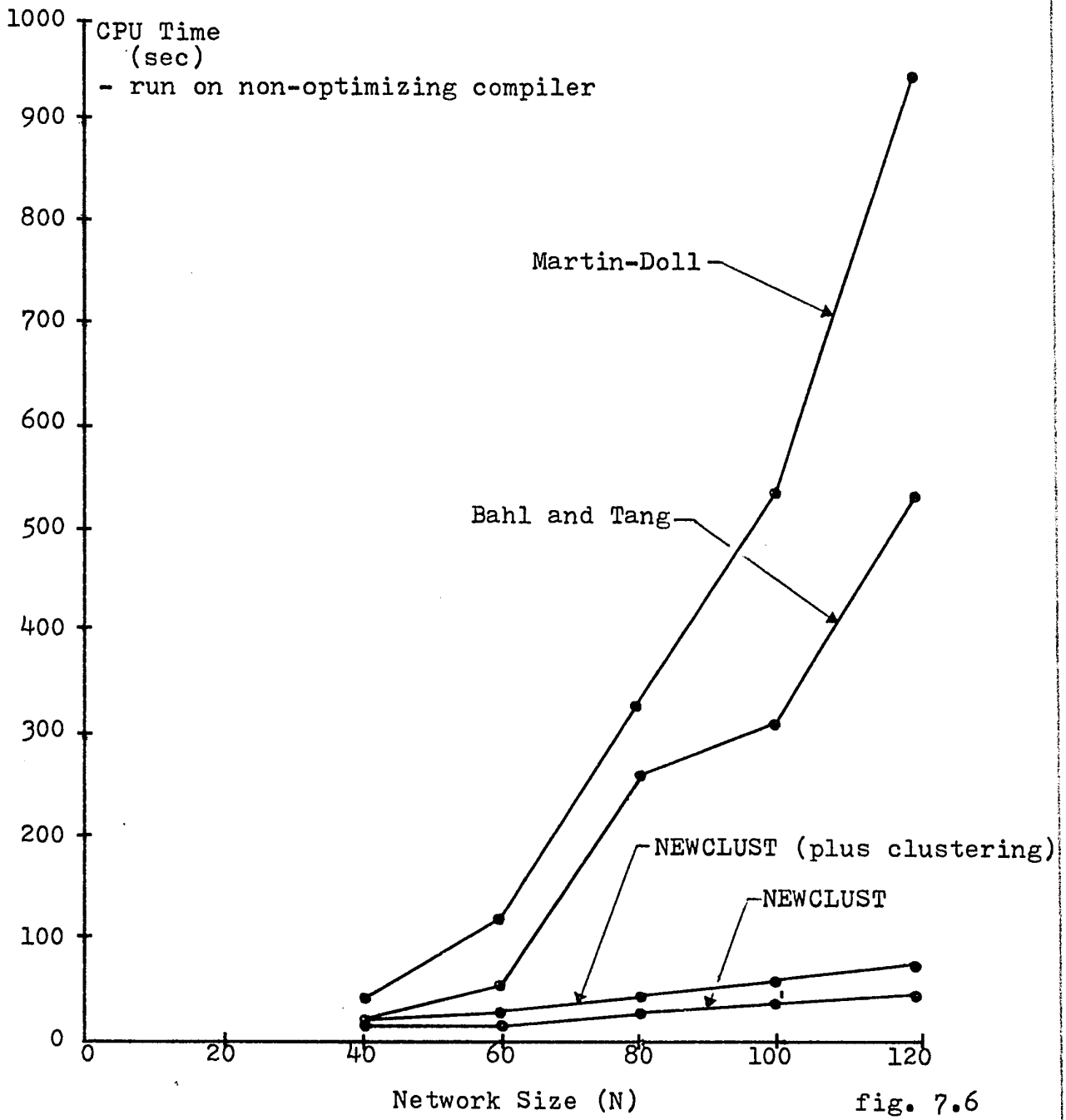


fig. 7.6

Algorithm Times

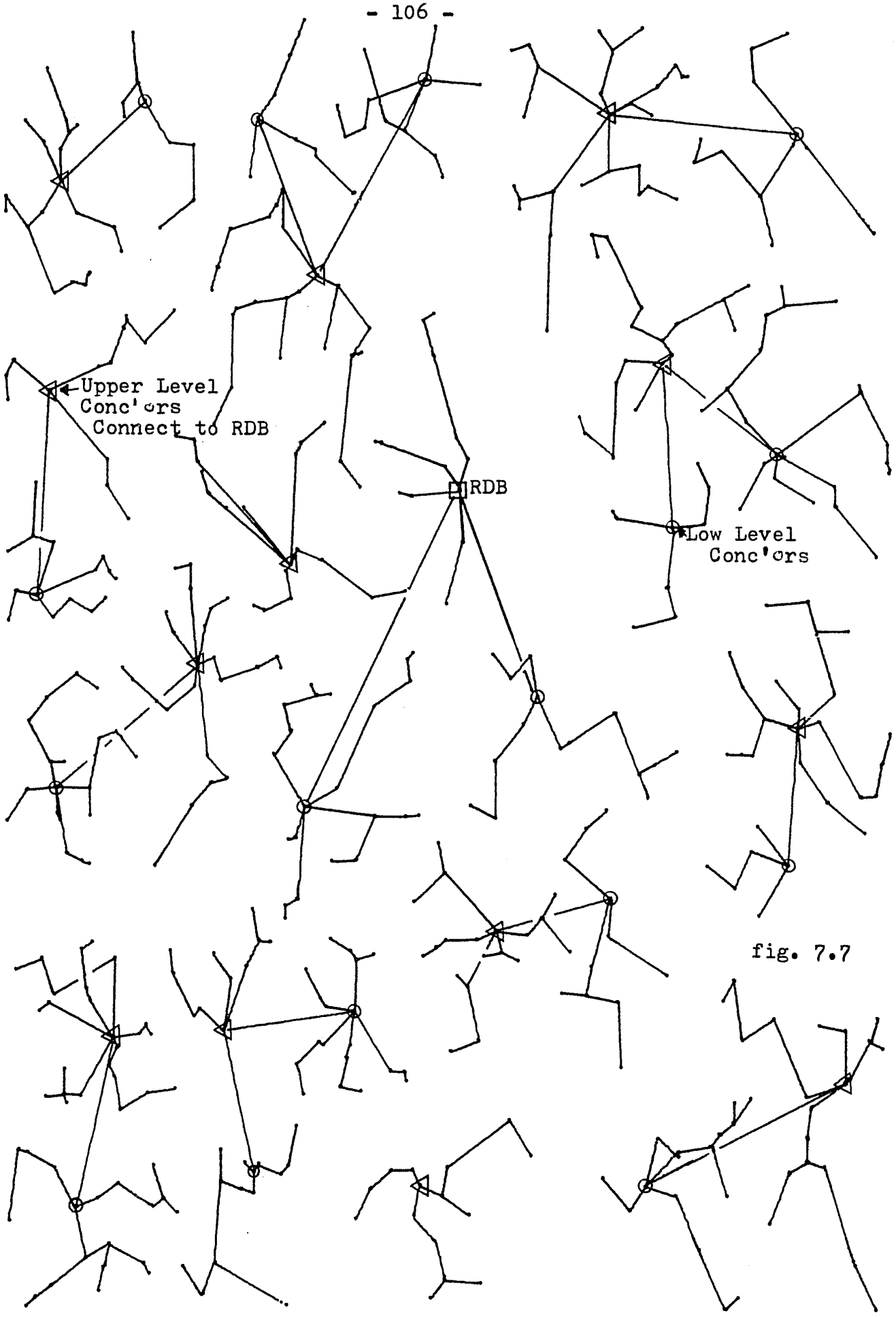


fig. 7.7

class of machines is the possibility of up to 10% variation in execution time for the same program, with the same data. This may cause some results to be slightly off by this amount.

We did not run networks with less then 100 nodes as the algorithtm was meant for the larger type systems. For every network size (e.g. 160 nodes) we ran, four different networks were designed and an average taken to give the final times and costs. Although this may not enough for a true statistical analysis, it does indicate how the algorithm is preforming.

CHAPTER 8

CONCLUSIONS, COMMENTS AND EXTENSIONS

8.1 Comments

8.11 Clustering

The clustering procedure of IMULEC, is basically designed for large (more than 120 nodes) networks. It is a flexible, easily manipulated process for locating clusters and possible concentrator sites.

As can be seen in chapter 7 (Figure 7.4), it also results in a large saving in computer time over the K-nearest neighbour approach of NEWCLUST. As pointed out in chapter 4, it has, except for the sorting procedure, a linear time-number of nodes relationship.

The clustering process could be improved, however, by the use of an operator, entering design parameters into the process from a graphics terminal, as the solution is being generated. It could also be modified to do a more extensive search for the exact concentrator locations in a window. But, as shown by Martin [21], the

exact location of a concentrator in a closed area has minimal effect on the final solution. Other improvements to the clustering could include a redefined frequency table recording not just node density, but also traffic and biasing parameters. This would allow the introduction of 'priority' areas in the network.

8.12 Line-Layout

IMULEC's comparison with NEWCLUST (Chapter 7), shows a definite improvement in design cost of the network, with little difference in layout execution time, but an appreciable overall execution time saving (clustering plus layout).

This is a significant result, since one must remember that using the serial approach of NEWCLUST the line-layout execution time is approximated by an equation of the form of equation 8.1, while for a parallel processing approach (see section 5.22), it is normally of the form of equation 8.2.

$$T_s = (A^{\gamma} + B^{\gamma} + C^{\gamma}) \qquad \text{eq. 8.1}$$

$$T_p = (A + B + C)^{\gamma} \qquad \text{eq. 8.2}$$

Where: A,B,C are the number of nodes in subnetworks A,B,C.

γ-design time factor.

But by the use of the feasible joining areas in IMULEC, we have been able to reduce the layout time to the point where it is almost equal to that of NEWCLUST.

IMULEC can effectively deal with concentrators of any capacity or any arrangement of capacities. It also can handle the restriction of the number of input ports to a concentrator. This last point is important, as it represents an improvement in the Essau-Williams class layout routines, that normally cannot handle this constraint.

The SELECTDROP 'optimization' procedure is, we believe, the ideal method to use given our desire to make it interactive. It allows for the selection of

concentrators to be considered for dropping in a discrete step approach. And in this way it can be used on-line with an operator.

8.13 Comments On IMULEC's Interactive Nature

In the formulation of the IMULEC algorithm, we attempted to allow for as much designer-algorithm interaction as possible. As we have shown in the flow charts of the various main sections (e.g. section 4.3), the operator can easily bias the solution at a number of points throughout the algorithm.

Due to the increased use and complexity of minicomputers, the use of intelligent graphics terminals in network design is becoming more and more a viable alternative to the old large scientific computer run algorithms. The IMULEC algorithm could be considered as a beginning in the development of interactive algorithms for this type of design work.

8.2 Extensions

The IMULEC algorithm is designed to handle connections between concentrators of different levels only. One area of further study would be to extend this to cover fully or partially connected levels of concentrators, as shown in Figure 1.3. By going to this type of arrangement, and by the proper use of a routing scheme, we could increase network reliability and most probably decrease its average delay.

Another modification to the algorithm would be to allow for minor processing at different levels of concentrators, thus instead of C_j as a concentrator's net output traffic (Equation 8.3), we would get the lesser value C_p. This effectively would reduce the upward flow of traffic in the network and thus cut down on the amount of higher level facilities needed.

$$C_j = \sum_{B_i \in S_k} t_{B_i}$$

Where: B_i is a node attached to conc. S_k

t-is the traffic of the node B_i

$$C_p = \beta C_j$$

$$0 \leq \beta \leq 1$$

eq. 8.3

8.3 Conclusions

We have just presented the new clustering algorithm IMULEC. It has been shown to give improved design costs when compared to NEWCLUST (which in turn is compared to other algorithms). The author believes that IMULEC represents one of the first network design algorithms developed to be used both interactively and/or automatically, and in all cases produce superior results, for large networks.

We also believe that further work could be done in the actual interactive testing of the algorithm and in the development of other interactive design schemes. By the use of this type of approach, the author believes, that we could design networks, that would take into account more 'variables' than any non-interactively designed system, and still produce 'better' results.

More work is needed, however, before the question of interactively designed multilevel data networks is solved. Several key areas include the choice of the optimal number of concentrator levels, the manner in which concentrators are connected internally and

externally in the level, and finally the question of intelligent concentrators. All require further research.

REFERENCES

[1] P.V. McGregor and D. Shen, "An Algorithm for the Access Facility Location Problem", IEEE Trans. Communications, Vol. 25, pp. 61, Jan. 1977.

[2] D.A. Greenberg, "A New Approach for the Optimal Placement of Concentrators in a Remote Terminal Communications Network", Proc. NTC, Vol. 1, pp. 37D1-7,1972.

[3] H. Dysart and N.D. Georganas, "Optimizing Concentrator Positioning in the Design of Two-Level Teleprocessing Networks", Proc. of 1976 IEEE Canadian Conference on Comm. and Power, Montreal, Oct. 1976.

[4] K. Fukunaga and L.D. Hostetler, "Optimization of k-Nearest-Neighbor Density Estimates", IEEE Trans. Information Theory, Vol.5, pp. 320, May 1973.

[5] R.L. Walton and P.D. Fisher, "A Picture Segmentation Algorithm and its Application to Clustering", Milwaukee Symposium on Automatic Control, 1974.

[6] W. Koontz, P. Narendra and K. Fukunaga, "A Graph-Theoretic Approach to Nonparametric Cluster Analysis, IEEE Trans. Computers, Vol. C-25, No. 9, pp. 936, Sept. 1976.

[7] I. Gitman and D. Levine, "An Algorithm for Detecting Unimodel Fuzzy Sets and its Application as a Clustering Technique", IEEE Trans. Computers, Vol. C-19, No. 7, pp. 583, July 1970.

[8] L. Kleinrock, Communication Nets; Stochastic Message Flow and Delay, McGraw-Hill, New York, 1964.

[9] R.L. Sharma and M.T. El-Bardai, "Suboptimal Communications Network Synthesis", Proc. International Conference on Comm., pp. 19.11-19.17, June 1970.

[10] L.R. Essau and K.C. Williams, "On Teleprocessing System Design", IBM Systems Journal, Vol. 5, pp. 142-147, 1966.

[11] M.Karnaugh, "A New Class of Algorithms for Multipoint Network Optimization",IEEE Trans. Comm., Vol. COM-24,

No.5, pp. 500-505, 1976; also Proc. ICC, Vol 3, pp. 337-11, 1975.

[12] K.M. Whitney, "Comparison of Network Topology Optimization Algorithms", Proc.ICCC, pp. 332-337, 1972.

[13] H. Frank, I.T. Frisch, and W. Chou, "Topological Considerations in the Design of the ARPA Computer Network", Proc. SJCC, pp.581-582, 1970.

[14] J. Martin, System Analysis for Design of Real Time Computer Systems, Prentice Hall Inc. Englewood Cliffs, New York, 1967.

[15] D. Doll, "Topology and Transmission Rate Considerations in the Design of Centralized Computer Communication Networks", IEEE TRANS. Comm. Tech., Vol. COM-19, pp. 339-344, 1971.

[16] L.S. Woo and D.T. Tang, "Optimization OF Teleprocessing Networks With Concentrators", Proc. NTC, Vol.1, pp.37C1-5. 1972.

[17] L.R. Bahl and D.T. Tang, "Optimization of Concentrator locations in Teleprocessing Networks:, in Computer Communications Networks and Teletraffic, J. Fox (ed.), Polytechnic Press, New York, N.Y., pp.355-362, 1972.

[18] H. Dirilten and R.W. Donaldson, "Topological Design of Teleprocessing Networks Using Linear Regression Clustering", IEEE Trans. Comm., Vol. COM-24, No. 10, Oct. 1976.

[19] A.A. Kuehn and M.J. Hamburger, "A Heuristic Program for Locating Warehouses", Management Science, Vol. 12, pp. 670-684, May 1966.

[20] F.T. Boesch (ed.), Large Scale Networks: Theory and Design, IEEE Press, New York, 1975.

[21] J. Martin, System Analysis for Data Transmission, Prentice-Hall Inc., Englewood Cliffs, N.Y., 1972.

[22] R.L. Page, "A Minimal Spanning Tree Clustering Method:, Comm. of the ACM, Vol. 17, No. 6, June 1974.

[23] M. Goldberg and S. Shlien, "Computer Implementation of a Four-Dimensional Clustering Algorithm", Energy, Mines and Resources Canada, Research Report 76-2, March 1976.

[24] P.K. Verma (ed.), Proceedings of ICCC Conference, Toronto, August 1976.

[25] H. Frank, I.T. Frisch, R. VanSlyke and W.S. Chou, "Optimal Design of Centralized Computer Networks", Networks, Vol. 1, pp.43-57, 1970.

[26] H.E. Livings and U.W. Pooch, "A Heuristic Teleprocessing Network Design Technique", Symposium on the Simulation of Computer Systems IV, August 1976.

[27] K.M. Chandy and R.A. Russell, "The DESIGN of Multipoint linkages in a Teleprocessing Tree Network", IEEE Trans. Computers, Vol. C-21, No. 10, Oct. 1972.

[28] A. Kershenbaum and W. Chou, "A Unified Algorithm for Designing Multidrop Teleprocessing Networks", IEEE trans. Comm., Vol. COM-22, No. 11, Nov. 1974.

[29] A.A.B. Pritjker and P.M. Ghare, "Locating New Facilities With Respect to Existing Facilities", AHE Trans., Vol. 2, No. 4, Dec. 1970.

[30] C.T. Zahn, "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters", IEEE Trans. Computers, Vol. C-20, No. 1, pp. 68, Jan. 1971.

[31] H. Frank, M. Gerla and W. Chou, "Issues in the Design of Large Distributed Computer Communication Networks", Proc. NTC, 1972.

[32] R.R. Boorstyn and H. Frank, "Large-Scale Network Topological Optimization", IEEE Trans. Comm., Vol. COM-25, No. 1, pp. 29-47, Jan. 1977.

[33] M. Gerla and L. Kleinrock, "On the Topological Design of Distributed Computer Networks", IEEE Trans. Comm., Vol. COM-25, No. 1, pp. 48-60, Jan. 1977.

[34] M. Milgram, B. Duboisson and B. Vacon, "A Computationally Efficient Clustering Algorithm", IEEE Trans. Systems, Man and Cybernetics, Vol. SMC-7, No.2, pp. 99-103, Feb. 1977.

[35] S.K. Chang, "A Model for Distributed Computer System Design", IEEE Trans. Systems, Man and Cybernetics, Vol. SMC-6, No. 5, pp. 344-359, May 1976.

[36] S. Lin, "Heuristic Programming as an aid to Network Design", Networks, Vol. 5, pp. 33-43, 1975.

[37] V.K.N. Whitney, "A Study of Optimal File Assignment and Communication Network Configuration", Ph.D. Thesis, University of Michigan, 1970.

APPENDICES

APPENDIX A

USE OF GRAPHIC TERMINALS IN NETWORK DESIGN

A.1 Introduction

One problem encountered during network topological design is viewing the final line layout quickly. Very often errors in the program's logic can be picked out easily given a visual read out of the result. The enclosed program is an example of a system which can be used to quickly recover the layout results in picture form. The display software is written in a manner as to be functionaly independent of the layout program it is attached to. Its only requirements are a x/y co-ordinate table and an interconnection table(e.g. connect pt a to pt d). An additional feature which can easily be implemented is a character generator to distinguish terminals , concentrators, RDBs etc.

A.2 Implementation

The program uses a Tektronix 4013 graphic display unit coupled to a IBM 360/65 computer via a hardwired line. To interface the two units and also provide some additional data manipulation features, there exists the

set of software packages listed in section A.6. These software packages are simple call up routines and if a user wishes he could implement his own set with minor difficulty.

The following outlines the basic steps of the program:

1) Run the routing algorithm and generate the topology to be displayed.

2) Set up the x/y and interconnection tables to be used by the display program, remembering to generate the routing table in the same order that it was found by the routing algorithm.

3) Pick scaling of picture(size) using the cursers on the machine, and also select type of display mode to be used. In this program we can pick from "display initial network" to "display final network" and any points between.

4) Display network.

5) If zoom-in or section display is desired block off area with cursers as in step 3.

6) Photograph or make "hardcopy" of the display.

A.3 Results

The Figures A1, A2, A3 and A4 show "hardcopy" copies of partially completed and completed multidrop assignment problems, designed using the Essau-Williams algorithm. This feature, of being able to show the network at any stage,can not only be used when explaining how the algorithm functions, but it also can be used by the designer to trace down errors in the network to where they were formed. The operator need only call back information a step at a time until the error is reached and by observing the error get an idea of where his program error lies.

Figure A5 shows the same network but this time the storage media is a photograph. Similiarly Figure A6 shows how symbols can be used to identify different types of units in the network(e.g. terminals-⧖ ,RDB-△).

Another important note to remember is that the cost of this type of system in terms of computer usage is minimal. This makes the picture form of read out very

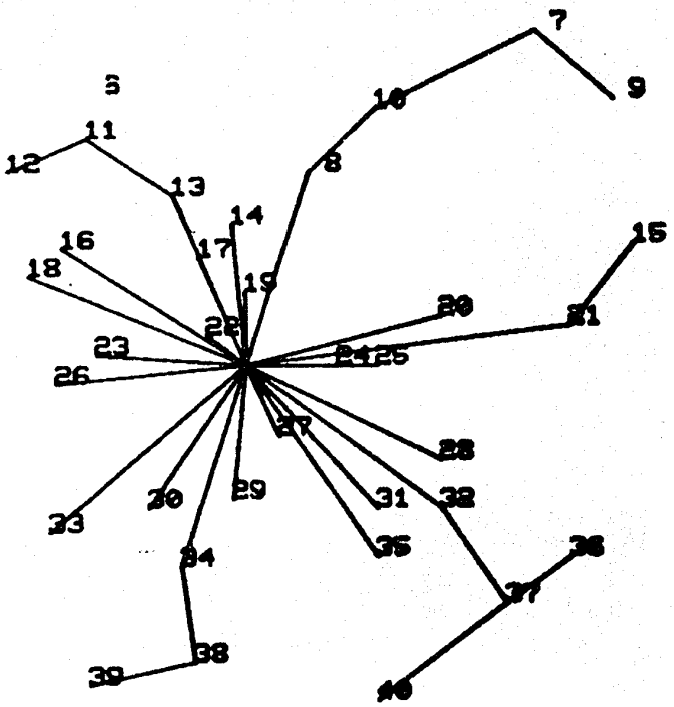
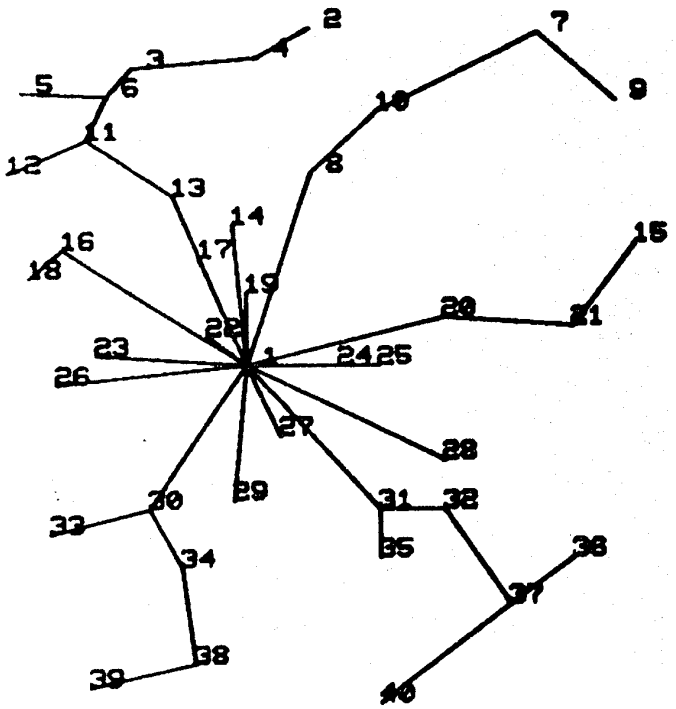

7
9
10
8
5
11
12
13
14
16
17
18
19
15
20
21
23
22
24
25
26
27
28
33
30
29
31
32
35
36
34
37
39
38
40

fig. A.1

A.2


2
7
3
4
9
5
6
10
11
12
8
13
14
16
17
15
18
19
20
21
23
22
1
24
25
26
27
28
30
29
31
32
33
35
36
34
37
39
38
40

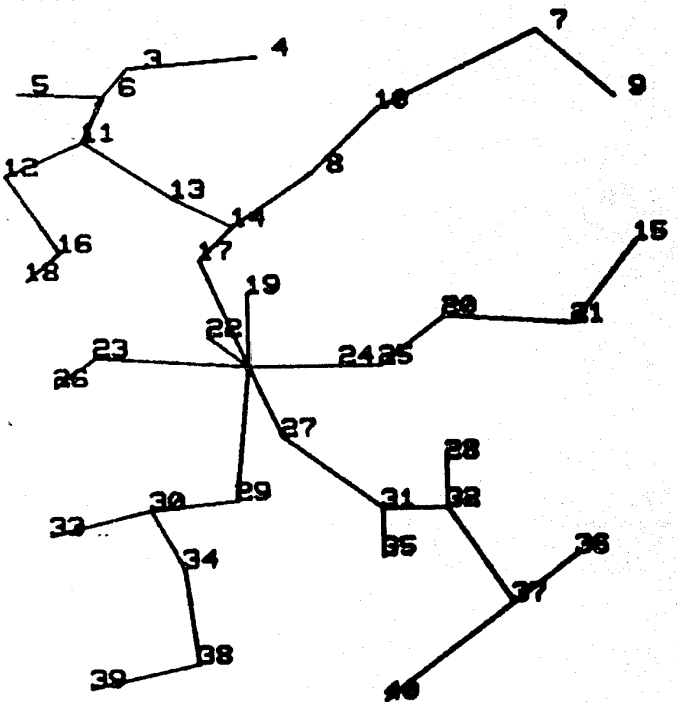
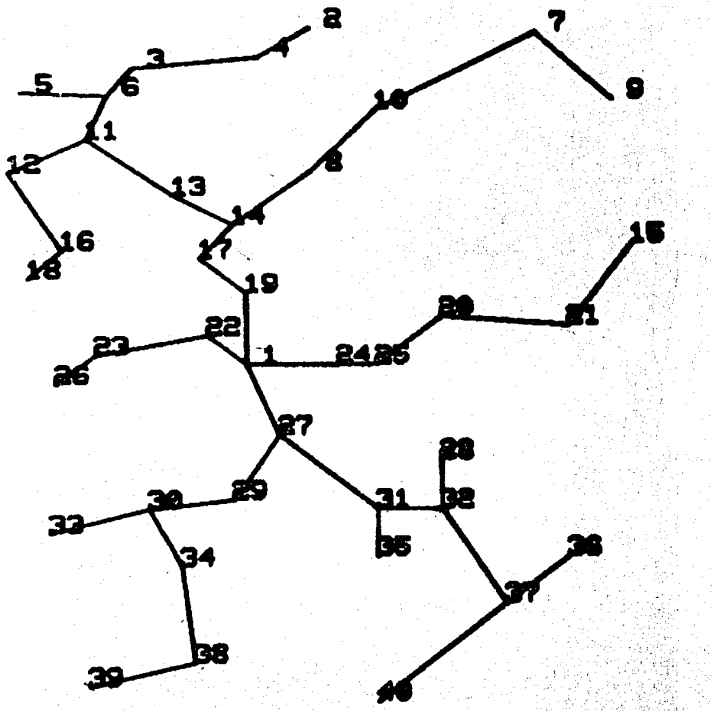
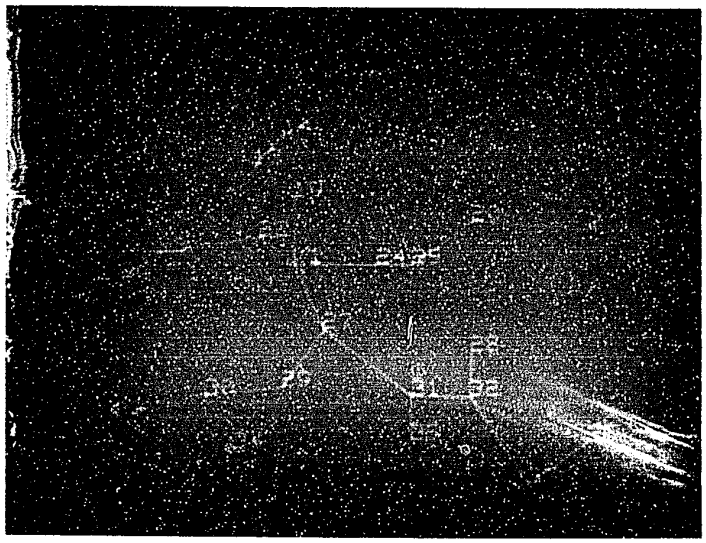

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40

fig. A.3

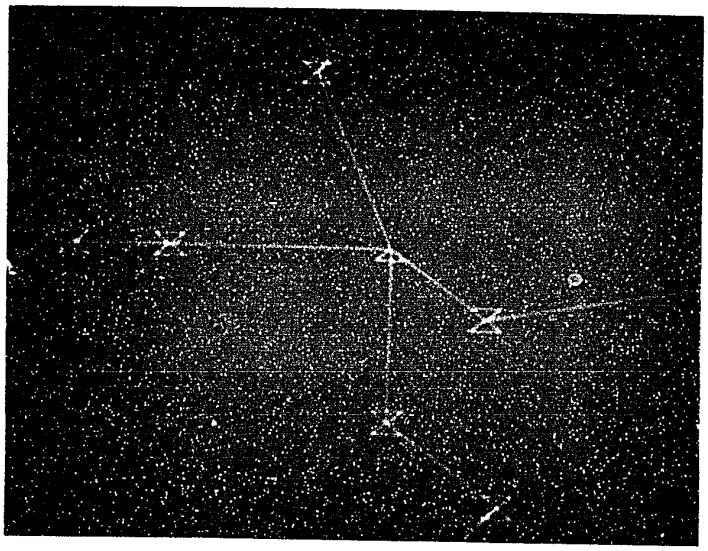
fig. A.4


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40



A.5

A.5



attractive as it saves the valuable time of the designer, who is now able to interpret his results quicker then if they were in only digital form.

A.4 Explanation of Program

1) Lines 122 to 142, set up the initial picture size and translates routing and x/y tables from the routing program to tables the design program understands.

2) Line 143, calls up a subroutine to scan the screen for inputs from the operator. The operater may enter one of the following codes: S,T,R,E,D and C; any other input is rejected.

3) Lines 144 to 151 , set a new picture size using the upper and lower contol inputs 'S' and 'T'.

4) Lines 152 to 164, when input on screen is a 'R' this subroutine is called to reset the picture size to its original size and then display the final circuit.

5) Lines 165 to 167, if the input is 'E' this section stops the display routine and goes to step 7.

6) Lines 168 to 184, under the input 'D' this section displays the network a step at a time showing how it was built. The 'C' input is used here to reset a counter if you wish to view the growth again.

7) Lines 185 to 191, print out results in digital form.

A.5 Conclusions

The system outlined in this section is one primarily setup to save a network designer time in interpreting results. The same sort of system, however, could be used when we run IMULEC interactively. The graphics terminal is a flexible, expandable type approach to solving network design problems and one that will receive wider attention as microprocessor backed display systems come into more general use.

L O G I T E K

1. Procédure

```
  //     EXEC   LOGITEK{,ASM=IEUASM}{,FORT=IEYFORT}
{ //ASM. SYSIN              DO      * }
  Programme(s) assembler 360       Assembler source program(s)
{ //FORT. SYSIN             DO      * }
  Programme(s) FORTRAN             FORTRAN source program(s)
{ //LKED. SYSIN             DO      * }
  Programme(s) objet               Object deck program(s)
{ //TEKTRO. SYSIN           DO      * }
  Données              (READ (5...) Data
```

1. Procedure

2. Codes d'erreur

2. Error codes

Pas d'erreur	0	OK
erreur du programmeur	4	programmer error
"break"	8	"break"
erreur de l'opérateur	12	operator error
erreur système (E/S)	16	system error (I/O)
erreur de donnée	20	data error
erreur E/S	24	I/O error
ligne coupée	28	line drop

3. Codes de terminaison

3. User abend codes

E/S apres ligne coupee	008	I/O following a line drop
erreur systeme (E/S)	004	system error (I/O)
autre	012	other

4. Messages

4. Messages

(GINIT)	TKX00	LIGNE OUVERTE	Line open (GINIT)
(GFIN)	TKX04	LIGNE FERMEE	Line closed (GFIN)
	TKX08	LIGNE COUPEE	Line drop

Spécifications		5. Specifications
re la ligne	GINIT	Opens the line
me la ligne	GFIN	Closes the line
le des données	GMODON(N)	Data mode
*4, E*2, R*8, E*4	N=0,1,2,3	R*4, I*2, R*8, I*4
aleur initiale	N=0	default value
gonale du viseur dans l'écran	GCADRE(X1,Y1,X2,Y2)	Viewpoint diagonal specificatio
leurs initiales	X1=Y1=0, X2=Y2=1024	default values
gonale du cadre	GECHEL(X1,Y1,X2,Y2)	Window diagonal specification
leurs initiales	X1=Y1=0, X2=Y2=1024	default values
oc commun,	COMMON/GTEK/	Common bloc,
de d'erreur, total erreur E/S	IERR, ITOT	error code, total I/O errors

Tracés		6. Plots
ace TEXTE en EBCDIC	GTTEXT(TEXTE,N)	Plots TEXTE in EBCDIC with
nb de caractères		N characters
Y vecteurs de points	GTDROI(x,Y,N)	Plots N-1 lines with
1 nombre de droites		X,Y ending points

Lecture		7. Input
XTE en EBCDIC, N: nb de	GLTEXT(TEXTE,N)	Reads a maximum of N characters
ractères à lire; au retour		in TEXT; on return N=# of cars.
nb de car. lus CHAR:	GLPCG(CHAR,X,Y)	typed CHAR: character typed;
ractère lu; X,Y position		X,Y cursor position
rseur		

Controle		8. Control
nne N fois	GSONNE(N)	Rings the bell N times
face l'écran	GENLEV	Erases the screen

. Transformations		9. Transformations
ncatène la translation X,Y	GTRANS(X,Y)	Concatenates the translation X
ec la transformation en cours		to the current transformation
em avec rotation X,Y,ANGLE	GROT(X,Y,ANGLE)	id. with rotation X,Y,ANGLE
estore la dernière trans-	GREST	Restores previous transformati
rmation		

APPENDIX B

CLUSTERING AND LINE-LAYOUT PROGRAMS

CLUSTERING PROGRAM

```
C
      INTEGER*2 TV
      INTEGER AVG
      INTEGER*2 NAVG,NSORT,TABLEL
      INTEGER*2 BUFFER,BUFERB,NCONB,NQUAD,NEXACT
      INTEGER*2 BARNO
      INTEGER XBAR,YBAR,TVTOT,SUMA,SUMB
C
      COMMON TABLEL(30,30),NCONC(50,3),NAVG(500,2),AVG(500),NEXACT(50)
      COMMON BARNO(600,2)
      COMMON XBAR(30,5),YBAR(30,5)
C*************************************************************
      COMMON XA(500),YA(500),TV(500),NSX(500),NSY(500)
 5400 FORMAT(' ',40I5)
 5401 FORMAT(' ',40I5)
 5404 FORMAT('1','YBARS TABLE ')
 5403 FORMAT('1','XBARS TABLE')
 5405 FORMAT('1',5X,'X',6X,'Y',6X,'TV',9X,'NSX',8X,'NSY',5X,'XBAR',5X
     *YBAR',5X,'NODE')
 5402 FORMAT(' ',2F8.3,3X,6I10)
 1003 FORMAT(2F10.2,I2)
 5416 FORMAT(' ',2I10,F12.2)
 1000 FORMAT(2I5,F10.2)
 1001 FORMAT(2I5)
 5407 FORMAT(' ',30I5)
 5412 FORMAT(' ','EXAMPLE',5X,'NO OF NODES',3X,'GRID AJUST FACTOR')
C
C********  CONSTANTS****
      NRAB=3
      DO 9000 NRAN=1,NRAB
      READ(5,1001)NQUAD,NCON
C
      BUFFER=1
      NB=30
      TVTOT=0
      SUMA=0
      SUMB=0
      NHD=0
      NHDB=0
      IC=0
      ID=0
C
C***
C
      READ(5,1000)NEXAM,N,PER
      READ(5,7777)LEVEL2,(NEXACT(I),I=1,LEVEL2)
 7777 FORMAT(10I5)
```

```
      I=0
      AAA=PER
      WRITE(6,5412)
      WRITE(6,5416)NEXAM,N,PER
      DO 5202 IA=1,N
      READ(5,1003)XA(IA),YA(IA),TV(IA)
      NSX(IA)=IA
      NSY(IA)=IA
      TVTOT=TVTOT+TV(IA)
 5202 CONTINUE
      DO 5206 KL=1,NB
      DO 5206 KLB=1,NB
 5206 TABLEL(KL,KLB)=0
C
C******  SORT
 5700 I=I+1
      CONSX=XA(NSX(I))
      LJ=I
      DO 5200 J=I,N
      IF(CONSX.LT.XA(NSX(J)))GOTO 5200
      LJ=J
      CONSX=XA(NSX(J))
 5200 CONTINUE
      J=LJ
      K=NSX(I)
      NSX(I)=NSX(J)
      NSX(J)=K
      IF(I.LT.N)GOTO 5700
      I=0
 5701 I=I+1
      CONSY=YA(NSY(I))
      LKA=I
      DO 5201 KA=I,N
      IF(CONSY.LT.YA(NSY(KA)))GOTO 5201
      LKA=KA
      CONSY=YA(NSY(KA))
 5201 CONTINUE
      KA=LKA
      K=NSY(I)
      NSY(I)=NSY(KA)
      NSY(KA)=K
      IF(I.LT.N)GOTO 5701
C
C******  BARS
C
      XBAR(1,2)=NSX(1)
      YBAR(1,2)=NSY(1)
      DO 5203 IB=1,N
      SUMA=SUMA+TV(NSX(IB))
      BARNO(NSX(IB),1)=IC+1
```

```
      NHD=NHD+1
      IR=IC+1
     IF(ABS(XA(XBAR(IR,2))-XA(NSX(IB))).LT.(XA(NSX(N))/AAA))GOTO 5500
      NTA=IB+1
      IF(NTA.GT.N)GOTO 5595
      IF(XA(NSX(IB)).EQ.XA(NSX(NTA)))GOTO 5500
5595  CONTINUE
      IC=IC+1
      XBAR(IC,1)=SUMA
      XBAR(IC+1,2)=NSX(IB)
      XBAR(IC,3)=NSX(IB)
      XBAR(IC,4)=NHD
      SUMA=0
      NHD=0
5500  CONTINUE
      SUMB=SUMB+TV(NSY(IB))
      BARNO(NSY(IB),2)=ID+1
      NHDB=NHDB+1
      IV=ID+1
     IF(ABS(YA(YBAR(IV,2))-YA(NSY(IB))).LT.(YA(NSY(N))/AAA))GOTO 5501
      NTB=IB+1
      IF(NTB.GT.N)GOTO 5596
      IF(YA(NSY(IB)).EQ.YA(NSY(NTB)))GOTO 5501
5596  CONTINUE
      ID=ID+1
      YBAR(ID,1)=SUMB
      YBAR(ID+1,2)=NSY(IB)
      YBAR(ID,3)=NSY(IB)
      YBAR(ID,4)=NHDB
      SUMB=0
      NHDB=0
5501  CONTINUE
5203  CONTINUE
      YBAR(ID+1,1)=SUMB
      YBAR(ID+1,3)=NSY(N)
      YBAR(ID+1,4)=NHDB
      XBAR(IC+1,1)=SUMA
      XBAR(IC+1,3)=NSX(N)
      XBAR(IC+1,4)=NHD
      IC=IC+1
      ID=ID+1
C
  SET UP TABLEL MATRIX     ******************************************
C
      DO 5207 KN=1,N
      JRX=BARNO(KN,1)
      JRY=BARNO(KN,2)
      TABLEL(JRY,JRX)=TABLEL(JRY,JRX)+1
 5207 CONTINUE
C
```

```
C***  PRINT
       WRITE(6,5405)
       DO 5211 KLL=1,N
 5211  WRITE(6,5402)XA(NSX(KLL)),YA(NSY(KLL)),TV(NSX(KLL)),NSX(KLL),NSY(
     *KLL),BARNO(KLL,1),BARNO(KLL,2),KLL
       WRITE(6,5403)
       DO 5204 IE=1,IC
 5204 WRITE(6,5400)(XBAR(IE,IG),IG=1,4)
        WRITE(6,5404)
       DO 5205 IH=1,ID
 5205 WRITE(6,5401)(YBAR(IH,II),II=1,4)
C**   PRINT OUT MATRIX TABLES    ************************************
       IDA=ID+1
       DO 5213 KY=1,ID
       WRITE(6,5407)(TABLEL(IDA-KY,KX),KX=1,IC)
 5213  CONTINUE
       CALL SCAVG(ID,IC,NQUADB,NQUAD,BUFFER,BUFERB,NCONB,NCON,IDA,N,LG,L
     *EVEL2)
 9000  CONTINUE
       STOP
       END
      SUBROUTINE SCAVG(ID,IC,NQUADB,NQUAD,BUFFER,BUFERB,NCONB,NCON,IDA,N
     *,LG,LEVEL2)
       INTEGER*2 TV
       INTEGER AVG
       INTEGER*2 NAVG,NSORT,TABLEL
       INTEGER*2 BUFFER,BUFERB,NCONB,NQUAD,NEXACT
       INTEGER*2 BARNO
       INTEGER RDB
       COMMON TABLEL(30,30),NCONC(50,3),NAVG(500,2),AVG(500),NEXACT(50)
       COMMON BARNO(600,2)
       COMMON XBAR(30,5),YBAR(30,5)
       COMMON XA(500),YA(500),TV(500),NSX(500),NSY(500)
 5409  FORMAT(' ',3I14)
 5408  FORMAT('1',9X,'COLUMN',8X,'ROW',11X,'VALUE',5X,'OF CONSECTORS')
 5410  FORMAT(' ',4I10)
 5411  FORMAT('1','ERROR IN PART B OF SCAVG, LIMIT EXCEX')
 5413  FORMAT(' ',3I14)
 5414  FORMAT('1','LIST ALL AVGS SCANNED')
 5415  FORMAT(' ',7X,'NCON',6X,'LIMIT',6X,'BUFFER',2X,'WINDOW')
 5416  FORMAT('-','PASS',3X,I4)
 5417  FORMAT('-','CON AT',I4)
 5419  FORMAT('1','RDB IS AT NODE',4X,I4)
C****    PART A    ******************************
C
       BUFERB=1
       NQUADB=2
       NCONB=10
       RDB=1
       NCONC(1,1)=BARNO(RDB,1)
```

```
      NCONC(1,2)=BARNO(RDB,2)
      NCONC(1,3)=0
      NEXACT(1)=RDB
      IL=LEVEL2+1
      NPASS=1
 5711 LGB=1
      LG=LEVEL2
      LAX=0
      LAY=0
      LEX=0
 5703 LBX=LAX+NQUAD
      IF(LBX.GT.IC)GOTO 5502
 5702 CONTINUE
      NSUM=0
      DO 5214 LC=1,NQUAD
      DO 5214 LD=1,NQUAD
      NSUM=NSUM+TABLEL(LAY+LD,LAX+LC)
 5214 CONTINUE
      LEX=LEX+1
      NAVG(LEX,1)=LAX+1
      NAVG(LEX,2)=LAY+1
      AVG(LEX)=NSUM
      LAY=LAY+1
      LBY=LAY+NQUAD
     IF(LBY.LE.ID)GOTO 5702
      LAX=LAX+1
       LAY=0
      GOTO 5703
 5502 CONTINUE
C
 WE NOW HAVE THE AVG OVER THE NET WITH WINDOW =NQUAD
C
C****     PART B          ******************
 LIMIT IS THE NO. OF DENSITIES WE WANT TO ORDER GOR THE PROBLEM AT
THIS LEVEL,IT IS DEPENDENT ON NCON WHICH IS THE N/. OF CONC. NEEDED
      WRITE(6,5408)
      LIMIT=NCON*2
WATCH THAT NO PARSEIS TO LARGE OR NCON MORE THAN 3X NO& OF QUADRENTS
 5712  NRTC=0
      IF(LGB.GT.LG)GOTO 5704
      NRTC=1
      NX=BARNO(NEXACT(LGB),1)
      NY=BARNO(NEXACT(LGB),2)
      NAZZ=NQUAD/2
      NX=NX-NAZZ
      NY=NY-NAZZ
      IF(NX.LT.1)NX=1
      IF(NY.LT.1)NY=1
      DO 5299 IZ=1,LEX
      IF(NAVG(IZ,1).NE.NX)GOTO 5299
```

```
      IF(NAVG(IZ,2).NE.NY)GOTO 5299
      LFB=IZ
      GOTO 5503
 5299 CONTINUE
 5704 CONST=-1.0
      DO 5215 LF=1,LEX
      IF(AVG(LF).LE.0)GOTO 5215
      IF(CONST.GT.AVG(LF)) GOTO 5215
      LFB=LF
      CONST=AVG(LF)
 5215 CONTINUE
      IF(CONST.LT.0.0)GOTO 5504
 5503 NCONC(LGB,1)=NAVG(LFB,1)
      NCONC(LGB,2)=NAVG(LFB,2)
      NCONC(LGB,3)=AVG(LFB)
WE NOW GO TO PICKING OUT OF THIS LIST THE BEST SECTORS
C****    PART   ****************
 5557 LQUAD=NQUAD-BUFFER
      LQUAD=NQUAD-BUFFER
      LOWX=NAVG(LFB,1)-LQUAD
      IHIGHX=NAVG(LFB,1)+LQUAD
      NIA=IC-(NQUAD-1)
      NIB=ID-(NQUAD-1)
      IF(IHIGHX.GT.NIA)IHIGHX=NIA
      LOWY=NAVG(LFB,2)-LQUAD
      IHIGHY=NAVG(LFB,2)+LQUAD
      IF(IHIGHY.GT.NIB)IHIGHY=NIB
      WRITE(6,5409)NCONC(LGB,1),NCONC(LGB,2),NCONC(LGB,3)
      LVC=1
      LVA=LVC
 5710 IF(NAVG(LVA,1).GE.LOWX)GOTO 5521
      LVA=LVA+1
      GOTO 5710
 5521 DO5223 LVB=LVA,LEX
      IF(NAVG(LVB,2).LT.LOWY)GOTO 5223
      IF(NAVG(LVB,2).GT.IHIGHY)GOTO 5522
      IF(AVG(LVB).LE.0)GOTO 5555
      AVG(LVB)=-1*AVG(LVB)
 5555 IF(NAVG(LVB,2).EQ.IHIGHY)GOTO 5522
 5223 CONTINUE
 5522 CONTINUE
      LOWX=LOWX+1
      LVA=LVB
      IF(LOWX.LE.IHIGHX)GOTO 5710
      IF(LGB.EQ.NCON)GOTO 5510
 5755 LGB=LGB+1
      IF(NRTC.NE.1)GOTO 5558
      GOTO 5712
 5558 CONTINUE
      GOTO 5704
```

```
 5510  CONTINUE
       DO 5266 MM=1,LEX
       IF(AVG(MM).NE.NCONC(LGB,3))GOTO 5266
       NCON=NCON+1
       GOTO 5755
 5266  CONTINUE
WE NOW JAVE THIS LIST OF WINDOWS YHAT WILL EACH HAVE A CONC.
 ********  PRINT OUT
       GOTO 5506
 5504  WRITE(6,5411)
       LGB=LGB-1
 5506  WRITE(6,5415)
       WRITE(6,5410)NCON,LIMIT,BUFFER,NQUAD
       CALL POSIT(IL,NQUAD,BUFFER,N,LGB)
       LG=IL
       NPX=1
       IF(NPASS.EQ.NPX)GOTO 5556
       NPASS=NPASS+1
       BUFFER=BUFERB
       NQUAD=NQUADB
       NCON=NCON+NCONB
       GOTO 5711
 5556  CONTINUE
TO MAKE MORE THAN ONE PASS REPLACE PPX WITH THE NO. OF RUNS NEEDED
  NCON IS EQUAL TO SUM OF ALL CONS AT ANY TIME
       WRITE(6,5416)NPASS
       WRITE(6,5419)RDB
       WRITE(6,5417)(NEXACT(LMI),LMI=1,LG)
       RETURN
       END
       SUBROUTINE POSIT(IL,NQUAD,BUFFER,N,LGB)
       INTEGER*2 TV
       INTEGER AVG
       INTEGER*2 NAVG,NSORT,TABLEL
       INTEGER*2 BUFFER,BUFERB,NCONB,NQUAD,NEXACT
       INTEGER*2 BARNO
       INTEGER XBAR,YBAR,TVTOT,SUMA,SUMB
       REAL MIDX,MIDXA,MIDXB,MIDY,MIDYA,MIDYB
       COMMON TABLEL(30,30),NCONC(50,3),NAVG(500,2),AVG(500),NEXACT(50)
       COMMON BARNO(600,2)
       COMMON XBAR(30,5),YBAR(30,5)
       COMMON XA(500),YA(500),TV(500),NSX(500),NSY(500)
C
C*******LOOP FORMED ******************
C
       ILB=IL
C
C****** BOUNDARY OF PARSE*********
       DO 5234 IIK=ILB,LGB
       AB=NCONC(IIK,1)
```

```
        AC=NCONC(IIK,2)
        AD=AC+(NQUAD - BUFFER)
        AE=AB + (NQUAD-BUFFER)
        XS=XA(XBAR(AB,2))
        XF=XA(XBAR(AE,3))
        YS=YA(YBAR(AC,2))
        YF=YA(YBAR(AD,3))
C
C********    FIND MIDX
C
        PARTS=NCONC(IIK,3)
        PARTS=ABS(PARTS)
        IJK=PARTS/2.
        REL=PARTS/2.
        ABC=IJK
        IF(ABC.EQ.REL)GOTO 5570
        NEOC=1
        GOTO 5571
 5570   NEOC=0
 5571   CONTINUE
        PARTS=PARTS+NEOC
        NMID=PARTS/2.
        NCOUT=0
        DO 5230 IAE=1,N
        IF(XA(NSX(IAE)).GT.XS.AND.XA(NSX(IAE)).LE.XF.AND.YA(NSX(IAE)).GT.
     *YS.AND.YA(NSX(IAE)).LE.YF)GOTO 5591
        GOTO 5230
 5591   CONTINUE
        NCOUT=NCOUT+1
        IF(NEOC.NE.1)GOTO 5573
        IF(NCOUT.GE.NMID)GOTO 5574
        GOTO 5230
 5573   IF(NCOUT.LT.(NMID+1))GOTO 5230
        MIDXA=XA(NSX((IAE-1)))
        MIDXB=XA(NSX(IAE))
        MIDX=MIDXA+ABS((MIDXA-MIDXB)/2.)
        GOTO 5575
 5230   CONTINUE
 5574   MIDX=XA(NSX(IAE))
 5575   CONTINUE
C
 FOR MIDY POINT
C
        NCOUTB=0
        DO 5231 IAR=1,N
        IF(YA(NSY(IAR)).GT.YS.AND.YA(NSY(IAR)).LE.YF.AND.XA(NSY(IAR)).GT.
     *XS.AND.XA(NSY(IAR)).LE.XF)GOTO5592
        GOTO 5231
 5592   CONTINUE
        NCOUTB=NCOUTB+1
```

```
      IF(NEOC.NE.1)GOTO 5576
      IF(NCOUTB.GE.NMID)GOTO 5577
      GOTO 5231
 5576 IF(NCOUTB.LT.(NMID+1))GOTO 5231
      MIDYA=YA(NSY((IAR-1)))
      MIDYB=YA(NSY(IAR))
      MIDY=MIDYA+ABS((MIDYA-MIDYB)/2.)
      GOTO 5578
 5231 CONTINUE
 5577 MIDY=YA(NSY(IAR))
 5578 CONTINUE
C** WE NOW HAVE EXACT PT FOT CONCENTRATOR MIDX,MIDY
 NOW PICK CLOSEST PT YO IT.
 HERE EE WOULD ALSO MAKE IT CLOSEST PT TO IT AND RDB BUT LINE COST
 IS SAALL THUS NOT IMPORTANT
      CDIST=10000.
      NXX=IAE-2
      IF(NXX.LE.0)NXX=1
      NXA=IAE+2
      IF(NXA.GT.N)NXA=N
      DO 5233 IAX=NXX,NXA
      BDIST=SQRT((MIDX-XA(NSX(IAX)))**2+(MIDY-YA(NSX(IAX)))**2)
      IF(BDIST.GT.CDIST)GOTO 5233
      CDIST=BDIST
      NEXACT(IL)=NSX(IAX)
 5233 CONTINUE
       IL=IL+1
 5234 CONTINUE
      IL=IL-1
      RETURN
      END
```

LINE-LAYOUT LISTING

```
      INTEGER*2 TV,M,MT,MNL,CPU,JS,CV,EXAM,MIL
      INTEGER*2 B,PV
      INTEGER*2 CVC,AA,AB,AC,AD,AF
      INTEGER*2 TVC
     INTEGER*2 JSB,PVB,CVB,TVB,MILB,NEWC,CONNB,CVCB
     COMMON TV(500),M(500),MT(500),MNL(500),CV(500),JS(500,2),CPU(500)
      COMMON C(320,320),B(320),XA(320),YA(320),PV(320),MIL(320,15)
      DIMENSION CVC(500),CONN(500)
     DIMENSION JSB(500),PVB(500),CVB(500),TVB(500),MILB(500,20),NEWC(99
    *)
      DIMENSION TVC(500),CONNB(500),CVCB(500)
     DIMENSION COSTM(500),HDIST(5),HHFIX(5),HFAC(5),DISTL(5),FIX(5),FAC
    *(5)
 800   FORMAT(6F10.3)
      DO 6225 IP=1,5
     READ(5,800)HDIST(IP),HHFIX(IP),HFAC(IP),DISTL(IP),FIX(IP),FAC(IP)
 6225  CONTINUE
C                  DATA VARIABLES
 801   FORMAT(5I5)
       NTRQ=1
       DO 9000 JAAB=1,NTRQ
       READ(5,801)EXAM,N,NOM,LOW3,LEVEL3
       NOFLEV=2
       N1=64
       N2=24
       NW=600
       N3=10
       NPAST=1
       NR=14
       ALPHA=3
       PERCE=0.35
       JTZ=0
       FIXC=2.0
C*****************************************************************
C            FORMAT STATEMENTS
C*****************************************************************
 600   FORMAT(2F10.2,I2)
C****CHANGE FORMAT POSITION COL'S 19,27 AND35***********
 400   FORMAT('1','CPU',I2)
 401   FORMAT(' ','CONC=',I3,'MAX TRAFFIC=',I2,'MAX NO NODES=',I2,'MAX I
    *NPUT LINES=',I2)
 402   FORMAT('CPU IS SET TO NODE 40 AND NODE 40 IS SET TO CPU')
 403   FORMAT(' ','CPU OF',I3,'IS=',I3)
 405   FORMAT(' ','COST',F8.2,7I8)
 406   FORMAT('1','EXAMPLE',I3)
 407   FORMAT('0','FINAL COST=',F8.2)
```

```
 409    FORMAT('1')
 412    FORMAT(' ','FINAL TOTAL TRAFFIC= ',I8,'NO OF NODES ',I8)
 430    FORMAT(' ','DROPPED CONCENTRATOR',I3)
 431    FORMAT(' ',' % USAGE OF CONCENTRATOR',I3,'IS',F5.2)
 432    FORMAT(' ','SAVED CONCENTRATOR',I3)
C*****************************************************************
C                        READ STATEMENTS
C*****************************************************************
        DO 4200 IA=1,N
        READ(5,600)XA(IA),YA(IA),TV(IA)
        MIL(IA,1)=TV(IA)
        MIL(IA,2)=0
        JS(IA,2)=0
        JSB(IA)=0
        PV(IA)=IA
 4200   CV(IA)=1
 804    FORMAT(10I5)
        READ(5,804)(M(IB),IB=1,NOM)
        NOMB=NOM-LEVEL3
        DO 9201 IBQ=1,NOMB
        MT(M(IBQ))=N1
        MNL(M(IBQ))=N2
        MIL(M(IBQ),1)=N3
 9201   CONTINUE
C*****************************************************************
        B(N)=N+1
C
C                          C/NSTANTS
        CPU(N)=N
        PV(N)=N
        COST=0.0
        NA=N-1
        KA=0
        MIL(N,2)=0
C*****************************************************************
C                 CHECK PRINT
C
        WRITE(6,400)M(1)
        IF(NOM.EQ.1)GOTO 4555
        DO 4203 IC=2,NOM
         TVC(M(IC))=TV(M(IC))
        MIL(M(IC),2)=0
 4203   WRITE(6,401)M(IC),MT(M(IC)),MNL(M(IC)),MIL(M(IC),1)
 4555   CONTINUE
C*****************************************************************
C              PLACE CPU IN N TH POSITION
C
        XB=XA(N)
        YB=YA(N)
        XA(N)=XA(M(1))
```

```
        YA(N)=YA(M(1))
        XA(M(1))=XB
        YA(M(1))=YB
        TC=TV(N)
        TV(N)=TV(M(1))
        TV(M(1))=TC
         MILA=MIL(N,1)
        MIL(N,1)=MIL(M(1),1)
        MIL(M(1),1)=MILA
        M(1)=N
        WRITE(6,402)
C*****************************************************************
        TVC(1)=TV(N)
        MT(N)=9999
        DO 9001 KPZ=2,NOM
 9001   IF(M(KPZ).EQ.N)M(KPZ)=1
C         FIND C MATRIX
C
        CALL DIST(NA,N)
        C(N,N)=0.0
C
C      SET  UP  SUPER  NODE
C
        NOM=NOM-LEVEL3
        IF(NOM.EQ.1)GOTO 4500
        DO 4206 IG=2,NOM
 4206   C(M(IG),N)=0.0
 4500   CONTINUE
C*****************************************************************R
C                 FIND CLOSEST CPU
C
        NOM=NOM+LEVEL3
        IF(NOM.EQ.1)GOTO 9500
        DO 8296   IKS=LOW3,NOM
        DO 8296   IKZ=LOW3,NOM
        C(M(IKS),M(IKZ))=-1.
 8296   CONTINUE
 9500   CONTINUE
        DO 4207 IH=1,N
        CLOS=100000.
        DO 4208 II=1,NOM
        IF(C(IH,M(II)).LT.0.0)GOTO 4208
        IF(CLOS.LT.C(IH,M(II)))GOTO 4208
        CLOS=C(IH,M(II))
        CPU(IH)=M(II)
 4208    CONTINUE
        WRITE(6,403)IH,CPU(IH)
        C(IH,N)=CLOS
 4207   CONTINUE
        CALL SHORT(NA,N)
```

```
      IF(NOFLEV.EQ.2)GOTO 6599
      IF(LOW3.GT.NOM)GOTO 6599
       DO 6297 IJN=LOW3,NOM
      IJ=M(IJN)
      C(IJ,N)=.1
 6297 CONTINUE
      DO 6296 IKM=LOW3,NOM
      DO 6295 IKA=LOW3,NOM
 6295 C(M(IKM),M(IKA))=-1
 6296 CONTINUE
 6599 CONTINUE
C
C         SCAN FOR BEST NEIGHBOUR
C
      DO 4209 IJ=1,NA
      CALL SCAN(IJ,N)
 4209 CONTINUE
C*****************************************************************
      CALL REW(N,NA,NR,NOM,KA,LEVEL3)
C*****************************************************************
C
C END OF FIRST RUN NOW CHK FOR TOO MANY INPUTS TO CONC
C
C
C  REMOVE NEXT CARD FOR INPUT PORT CHECKING
      GOTO 6577
C
      IF(NOM.EQ.1)GOTO 4527
 4728 DO 4237 IL=2,NOM
      AA=M(IL)
      MB=MIL(AA,2)
      IF(MIL(AA,1).LT.MB)GOTO 4526
 4237 CONTINUE
       GOTO 6577
 4526 CALL CORECT(AA,NR,N,NOM,NA,KA)
      GOTO 4728
 6577 CONTINUE
C
C FOUND FINAL DESIGN, NOW CHK FOR DROPPING OF CONCENTATORS
C SET UP LIST OF CONC. WITH LESS THAN PERCE USAGE
      JTZ=0
      DO 6201 MP=LOW3,NOM
      IF(M(MP).EQ.NW)GOTO 6201
      USED=(TV(M(MP))*100.)/(MT(M(MP))*1.0)
      WRITE(6,431)M(MP),USED
      USAGE=MT(M(MP))*PERCE
      NUSAGE=USAGE
      IF(NUSAGE.LE.TV(M(MP)))GOTO 6201
        JTZ=JTZ+1
      CONNB(JTZ)=M(MP)
```

```
 6201  CONTINUE
C      SORT THE CONC TO BE TESTED
       IF(JTZ.EQ.0)GOTO 6700
       DO 6214 MK=1,JTZ
       NLAR=-1
       DO 6215 NKB=MK,JTZ
       IF(NLAR.GE.TV(CONNB(NKB)))GOTO 6215
       NOLD=CONNB(NKB)
       CONNB(NKB)=CONNB(MK)
       CONNB(MK)=NOLD
       NLAR=TV(CONNB(NKB))
 6215  CONTINUE
 6214  CONTINUE
 4527  CONTINUE
 6700 CONTINUE
C                    PRINT OUT
C
       COST=0.0
       CV(N)=0
       TV(N)=0
       WRITE(6,406)EXAM
       DO 4215 IZ=1,NA
       IF(JS(IZ,2).EQ.N)GOTO 6571
       IF(JS(IZ,2).NE.JSB(IZ))GOTO 6571
C   LINK FORMED
       COST=COST+COSTM(IZ)
       L=JS(IZ,2)
       GOTO 6579
 6571  CONTINUE
C  LINJ NEVER FORMED BEFORE
       IF(JS(IZ,2).NE.N)GOTO 6535
       L=CPU(IZ)
       IF(L.NE.N)GOTO 2511
       TV(N)=TV(N)+TV(IZ)
       CV(N)=CV(N)+CV(IZ)
       GOTO 6534
 6535  L=JS(IZ,2)
 6534  CONTINUE
C NOW CHECK IF IT IS A CONC TO CONC
 2511  DO 6222 IXZ=1,NOM
 6222  IF(IZ.EQ.M(IXZ))GOTO 6572
C  IT IS A NODE TO CONC
                  D=SQRT((XA(IZ)-XA(L))**2+(YA(IZ)-YA(L))**2)
       IF(D.EQ.0.0)GOTO 6573
       DO 6223 IWA=1,5
       IF(D.LE.DISTL(IWA))GOTO 6223
       CLD=FIX(IWA)+(D-DISTL(IWA))*FAC(IWA)
       GOTO 6574
 6223  CONTINUE
 6573  CLD=0.0
```

```
 6574   CONTINUE
        COST=COST+CLD
        COSTM(IZ)=CLD
        GOTO 6579
 6572   CONTINUE
C   IT IS A CONC TO CONC CONNECTION
                  D=SQRT((XA(IZ)-XA(L))**2+(YA(IZ)-YA(L))**2)
        IF(D.EQ.0.0)GOTO 6576
        DO 6224 IWB=1,5
        IF(D.LE.HDIST(IWB))GOTO 6224
        CHD=HHFIX(IWB)+(D-HDIST(IWB))*HFAC(IWB)+FIXC
        GOTO 6578
 6224   CONTINUE
 6576   CHD=FIXC
 6578   CONTINUE
        COST=COST+CHD
        COSTM(IZ)=CHD
 6579   CONTINUE
 4215   WRITE(6,405)COST,IZ,L,TV(IZ),CV(IZ),B(IZ),PV(IZ),JS(IZ,1)
        WRITE(6,407)COST
        WRITE(6,412)TV(N),CV(N)
        WRITE(6,409)
        IF(NPAST.EQ.1)GOTO 6501
C NOW CHK IF COST SAVING WAS ACHEIVED BY THE DROPPING OF  CONC NW
C  SET TO KICK OUT IF A CONC IS NOT DROPPED, TON ALLOW TO CONTINUE SEND TO
        IF(COSTB.GT.COST)GOTO 6502
        WRITE(6,432)NW
        GOTO 6500
C THE NEW DESIGN IS NORE EXPENSIVE RESET TO OLD NETWORK
        DO 6202 MX=1,NA
        JS(MX,2)=JSB(MX)
        PV(MX)=PVB(MX)
        CV(MX)=CVB(MX)
        TV(MX)=TVB(MX)
        B(MX)=JSB(MX)
 6202   CONTINUE
        CV(N)=NCV
        TV(N)=NTV
        COST=COSTB
        DO 6203 MY=1,N
        NNQ=MILB(MY,2)+2
        DO 6204 MZ=1,NNQ
 6204   MIL(MY,MZ)=MILB(MY,MZ)
 6203   CONTINUE
        NOM=NOM+1
        GOTO 6501
 6502   CONTINUE
C IF SAVING DROP FROM M LIST THE CONC NW
        WRITE(6,430)NW
        DO 6205 MV=1,NOM
```

```
 6205   M(MV)=NEWC(MV)
        MT(NW)=999
 6501   CONTINUE
C NOW SEE IF ANY UNDERUSED CONC LEFT ,PICK AND START
        IF(JTZ.LT .1)GOTO 6500
        NPAST=0
        NW=CONNB(JTZ)
       JTZ=JTZ-1
C    STORE PRIMARY TABLES
        DO 6207 NMA=1,NA
        JSB(NMA)=JS(NMA,2)
        PVB(NMA)=PV(NMA)
        CVB(NMA)=CV(NMA)
        TVB(NMA)=TV(NMA)
 6207   CONTINUE
        NCV=CV(N)
        NTV=TV(N)
        COSTB=COST
        DO 6208 NMB=1,N
        NNP=MIL(NMB,2)+2
        DO 6209 NMC=1,NNP
 6209   MILB(NMB,NMC)=MIL(NMB,NMC)
 6208   CONTINUE
C FING ALL NODES ATTACHED TO THE CONC NW
        NNR=MIL(NW,2)
        DO 6210 NMD=1,NNR
        CVCB(NMD)=MIL(NW,(NMD+2))
        NTR=MIL(NW,(NMD+2))
        JS(NTR,2)=NTR
        PV(NTR)=NTR
        CV(NTR)=1
        TV(NTR)=MIL(NTR,1)
 6210   CONTINUE
        CVCB(NMD)=NW
C RESET END TOTALS
        CV(N)=CV(N)-CV(NW)
        TV(N)=TV(N)-TV(NW)
        TV(NW)=TVC(NW)
        CV(NW)=1
        JS(NW,2)=NW
        PV(NW)=NW
        MIL(NW,1)=TVC(NW)
        MIL(NW,2)=0
        IJ=NW
        CALL SELECD(IJ,N)
        C(NW,N)=C(NW,CPU(NW))
C    FIND NEW LIST OF CONC
        NME=0
        DO 6211 NMF=1,NOM
        IF(M(NMF).EQ.NW)GOTO 6211
```

```
      NME=NME+1
      NEWC(NME)=M(NMF)
 6211 CONTINUE
      DO 6266 LKQ=1,NA
      NQP=PV(LKQ)
 6266 C(LKQ,N)=SQRT((XA(NQP)-XA(CPU(NQP)))**2+(YA(NQP)-YA(CPU(NQP)))**2)
      NEI=0
      NOM=NME
      DO 6212 MI=1,NMD
      IJX=CVCB(MI)
      MIL(IJX,2)=0
      DO 6299 MAB=1,NA
      IF(PV(MAB).NE.IJX)GOTO 6299
      IJ=MAB
      IT=IJ
      JS(IJ,2)=IJ
      CV(IJ)=1
      TV(IJ)=MIL(IJ,1)
      PV(IJ)=IJ
      MIL(IJ,2)=0
C CALL NEW DISTANCE CALCULATION
      CALL SELECD(IJ,N)
C FIND NEW CLOSEST RDB
      CLOBB=10000.
      DO 6213 NMG=1,NOM
      IF(CLOBB.LT.C(IJ,NEWC(NMG)))GOTO 6213
      CLOBB=C(IJ,NEWC(NMG))
      CPU(IJ)=NEWC(NMG)
 6213 CONTINUE
      C(IJ,N)=CLOBB
      NEI=NEI+1
      CVC(NEI)=IJ
 6299 CONTINUE
 6212 CONTINUE
C SCAN FOR NEW NEAREST NEIGHBOUR
      DO 6267 IGG=1,NEI
      IJ=CVC(IGG)
C DO A SHORT ROUTINE
      DO 6284 IGF=1,NA
      IF(C(IJ,N).GE.C(IJ,IGF))GOTO 6284
      C(IJ,IGF)=-1
 6284 CONTINUE
      CALL SCAN(IJ,N)
 6267 CONTINUE
C RESET COMPLETE NOW REBUILD NETWORK
      CALL REW(N,NA,NR,NOM,KA,LEVEL3)
C  CORRECT FOR TOO MANY INOUT LINES
      IF(NOM.LE.1)GOTO 6527
      IL=0
      MIL(NW,1)=MIL(NW,2)
```

```
 6728   DO 6237 IL=2,NOM
        AA=NEWC(IL)
        MB=MIL(AA,2)
        IF(MIL(AA,1).LT.MB)GOTO 6526
 6237   CONTINUE
        GOTO 6527
6526   CALL CORECT(AA,NR,N,NOM,NA,KA)
        GOTO 6728
 6527   CONTINUE
        GOTO 6700
 6500   CONTINUE
 9000   CONTINUE
        STOP
        END
C*******************************************************************
        SUBROUTINE REW(N,NA,NR,NOM,KA,LEVEL3)
C*******************************************************************
        INTEGER*2 TV,M,MT,MNL,CPU,JS,CV,EXAM,MIL
        INTEGER*2 B,PV
       COMMON TV(500),M(500),MT(500),MNL(500),CV(500),JS(500,2),CPU(500)
        COMMON C(320,320),B(320),XA(320),YA(320),PV(320),MIL(320,15)
        GOTO 4508
C
C         PICK BEST JOINING
C
 4700   CALL SCAN(IJ,N)
 4710   CONTINUE
 4508   CLOSC=-1.0
        DO 4212  IO=1,NA
        IF(JS(IO,2).EQ.N)GOTO 4212
        IF(PV(IO).NE.IO)GOTO 4212
        IF(IO.EQ.PV(B(IO)))GOTO 2522
        IF(C(IO,B(IO)).LT.0.0)GOTO 4212
        R=C(IO,N)-C(IO,B(IO))
        IF(CLOSC.GT.R)GOTO 4212
        CLOSC=R
        IT=IO
 4212   CONTINUE
        IF(CLOSC.LT.0.0)GOTO 2503
C
C         KICK OUT
C
        IV=B(IT)
C
C*******************************************************************
C   FIND IF NEW B IS NEEDED
C
        IF(CV(IV).EQ.1)GOTO 4515
        IJ=IT
        CALL SCAN(IJ,N)
```

```
      IF(B(IJ).EQ.IV)GOTO 4515
      GOTO 4710
 4515 CONTINUE
C
C
C                     TESTS
C
      IW=PV(IV)
      IF(IW.NE.N)GOTO 2504
      NAT=CPU(IT)
      MNLB=CV(NAT)+CV(IT)
      TVB=TV(NAT)+TV(IT)
      IF(NAT.EQ.N)GOTO 2517
C          CHECK LINE CAPACITY
      IF(TVB.GT.MT(NAT))GOTO 2506
C           CHECH  NO OF LINES
      IF(MNLB.GT.MNL(NAT))GOTO 2506
      IF(PV(NAT).EQ.N)TV(N)=TV(N)+TV(IT)
C
C               IF TESTS PASSED JOIN
C
 2517 CONTINUE
      TV(NAT)=TVB
      CV(NAT)=MNLB
      IF(CPU(IT).EQ.N)GOTO 6566
      MIL(NAT,2)=MIL(NAT,2)+1
      NML=MIL(NAT,2)+2
      MIL(NAT,NML)=IT
 6566 IF(IV.EQ.IW)GOTO 2507
      GOTO 2507
 2506 CONTINUE
C
C IF NEED BE REASIGN TO NEW CPU
C
      CALL SELECD(IT,N)
      NOM=NOM-LEVEL3
      CALL NEWCPU(N,IT,NAT,NOM,IJ)
      NOM=NOM+LEVEL3
      GOTO 4700
 2711 TV(IW)=TVA
      CV(IW)=CV(IW)+CV(IT)
      MIL(IV,2)=MIL(IV,2)+1
      MIL(IV,(MIL(IV,2)+2))=IT
      IF(IV.EQ.IW)GOTO 2507
      TV(IV)=TV(IV)+TV(IT)
      CV(IV)=CV(IV)+CV(IT)
 2507 KA=KA+1
      JS(IT,1)=KA
      JS(IT,2)=IV
      IF(IW.EQ.N)GOTO 4595
```

```
      DO 4231 LL=1,N
      IF(PV(LL).NE.IT)GOTO 4231
      PV(LL)=IW
 4231 CONTINUE
 4595 CONTINUE
      C(IT,N)=C(IV,N)
      C(IT,IV)=-1
       GOTO 4710
C
C           TESTS FOR NONE CONC SITES
C
 2504   TVA=TV(IT)+TV(IW)
      IF(TVA.LE.NR)GOTO 2711
 2514 C(IT,IV)=-1.0
      IJ=IT
      GOTO 4700
 2522 C(IO,B(IO))=-1
      IJ=IO
      GOTO 4700
 2503 CONTINUE
      RETURN
      END
C*********************************************************************
      SUBROUTINE SCAN(IJ,N)
      INTEGER*2 TV,M,MT,MNL,CPU,JS,CV,EXAM,MIL
      INTEGER*2 B,PV
     COMMON TV(500),M(500),MT(500),MNL(500),CV(500),JS(500,2),CPU(500)
      COMMON C(320,320),B(320),XA(320),YA(320),PV(320),MIL(320,15)
      CLOSB=100000.
      DO 4210 IK=1,N
      IF(C(IK,N).LT.0.0)GOTO 4210
      IF(C(IJ,N).LT.C(IK,N))GOTO 4210
      IF(C(IJ,IK).LT.0.0)GOTO 4210
      IF(CLOSB.LT.C(IJ,IK))GOTO 4210
      CLOSB=C(IJ,IK)
      B(IJ)=IK
 4210 CONTINUE
       RETURN
      END
C*********************************************************************
      SUBROUTINE DIST(NA,N)
      INTEGER*2 TV,M,MT,MNL,CPU,JS,CV,EXAM,MIL
      INTEGER*2 B,PV
     COMMON TV(500),M(500),MT(500),MNL(500),CV(500),JS(500,2),CPU(500)
      COMMON C(320,320),B(320),XA(320),YA(320),PV(320),MIL(320,15)
      DO 4204 ID=1,NA
      DO 4205 IE=ID,N
      C(ID,IE)=SQRT((XA(ID)-XA(IE))**2+(YA(ID)-YA(IE))**2)
 4205 C(IE,ID)=C(ID,IE)
      C(ID,ID)=-1
```

```
 4204   CONTINUE
        RETURN
        END
C*****************************************************************
        SUBROUTINE SHORT(NA,N)
        INTEGER*2 B,PV
        INTEGER*2 TV,M,MT,MNL,CPU,JS,CV,EXAM,MIL
       COMMON TV(500),M(500),MT(500),MNL(500),CV(500),JS(500,2),CPU(500)
        COMMON C(320,320),B(320),XA(320),YA(320),PV(320),MIL(320,15)
        DO 4230 LC=1,NA
        CLOSR=C(LC,N)
        DO 4232 LD=1,NA
        IF(CLOSR.GT.C(LC,LD))GOTO 4232
 4233   C(LC,LD)=-1.0
 4232   CONTINUE
 4230   CONTINUE
        RETURN
        END
C*****************************************************************
        SUBROUTINE NEWCPU(N,IT,NAT,NOM,IJ)
        INTEGER*2 TV,M,MT,MNL,CPU,JS,CV,EXAM,MIL
        INTEGER*2 B,PV
       COMMON TV(500),M(500),MT(500),MNL(500),CV(500),JS(500,2),CPU(500)
        COMMON C(320,320),B(320),XA(320),YA(320),PV(320),MIL(320,15)
 404    FORMAT('0','CHANGE CPU OF',I3,'TO',I3)
        CPU(IT)=N
        C(IT,NAT)=-1
        CLOSD=100000.
        DO 4214 IR=1,NOM
        NTOV=TV(IT)+TV(M(IR))
        IF(NTOV.GT.MT(M(IR)))GOTO 4214
        MTOV=CV(IT)+CV(M(IR))
        IF(MTOV.GT.MNL(M(IR)))GOTO 4214
        IF(C(IT,M(IR)).LT.0.0)GOTO 4214
        IF(CLOSD.LT.C(IT,M(IR)))GOTO 4214
        CLOSD=C(IT,M(IR))
        CPU(IT)=M(IR)
 4214   CONTINUE
        WRITE(6,404)IT,CPU(IT)
        C(IT,N)=C(IT,CPU(IT))
        IJ=IT
        RETURN
        END
C*****************************************************************
        SUBROUTINE CORECT(AA,NR,N,NOM,NA,KA)
        INTEGER*2 TV,M,MT,MNL,CPU,JS,CV,EXAM,MIL
        INTEGER*2 B,PV
        INTEGER*2 CVC,AA,AB,AC,AD,AF
       COMMON TV(500),M(500),MT(500),MNL(500),CV(500),JS(500,2),CPU(500)
        COMMON C(320,320),B(320),XA(320),YA(320),PV(320),MIL(320,15)
```

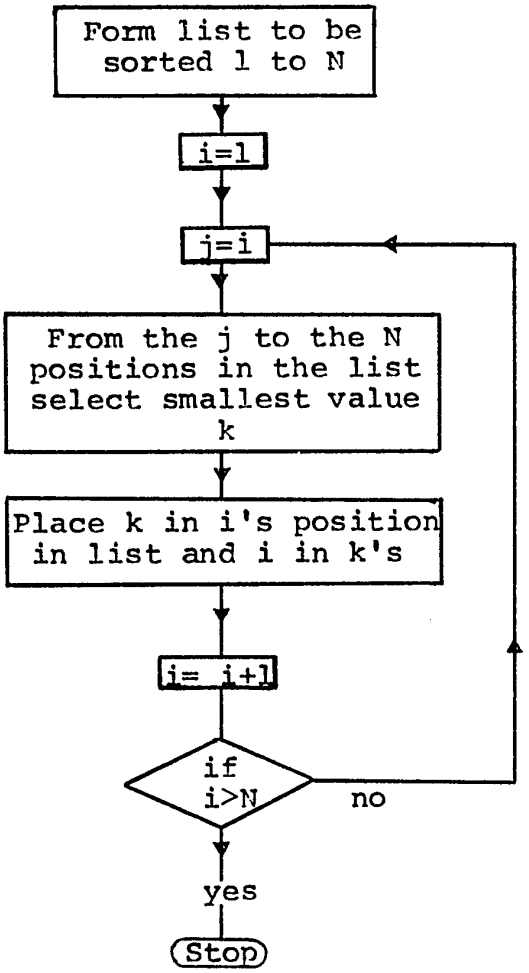
```
      DIMENSION CVC(500),CONN(500)
  410 FORMAT(' ','CONC ',I3,' HAD PT ',I3,' ENTER IT BUT NOW IT GOES TO
     * ',I3)
  411 FORMAT('1','CORRECTING FOR OVERCONNECTION OF CONC ',I3)
      WRITE(6,411)AA
      DO 4288 IU=1,N
 4288 CVC(IU)=0
C
C       FIND SMALLEST TV OF THOSE ENTERING OVERLOADED CONC AA
C    IF TWO LINES EQUAL THE LAST IS USED
C
      CLOST=1000.
      MA=MIL(AA,2)+2
      DO 4260 JAA=3,MA
      IF(CLOST.LT.TV(MIL(AA,JAA)))GOTO 4260
      AB=MIL(AA,JAA)
      CLOST=TV(AB)
      JAC=JAA
 4260 CONTINUE
C
C   THE SMALLEST LINE CALLED AB,WITH TV=CLOST
C  FIND IF ANOTHER LINE CAN TAKE OVER LOAD OF AB
C
      I=0
       DO 4261 JAB=3,MA
      IF(JAB.EQ.JAC)GOTO 4261
      IF((TV(MIL(AA,JAB))+TV(AB)).GT.NR)GOTO 4261
      I=I+1
      CONN(I)=MIL(AA,JAB)
 4261 CONTINUE
      IF(I.EQ.0)GOTO 4570
C
C IF ANOTHER LINE CAN,FIND BEST LINKS TO JOIN ,WITH MORE THAN 1 LINE TO
C   TO JOIN POSSIBLE
C FOR NOW CONNECT TO THE NEARSEST END OF LINE
C
      CLOSQ=10000.
      AD=CONN(1)
      IF(I.EQ.1)GOTO 4576
      DO 4262 JAE=1,I
      IF(C(AB,CONN(JAE)).LT.0.0)GOTO 4262
      IF(CLOSQ.LT.C(AB,CONN(JAE)))GOTO 4262
      CLOSQ=C(AB,CONN(JAE))
      AD=CONN(JAE)
 4262 CONTINUE
 4576 CONTINUE
C*************************************************************
C  THUS BEST CONNECTION IS AB TO AD
C
C   NOW UPDATE ALL TABLES
```

```
      CV(AD)=CV(AD)+CV(AB)
      TV(AD)=TV(AD)+TV(AB)
      WRITE(6,410)AA,AB,AD
      JS(AB,2)=AD
      MIL(AD,2)=MIL(AD,2)+1
      MIL(AD,(MIL(AD,2)+2))=AB
      GOTO 4590
 4570 CONTINUE
C
C THERE IS NO LINE THAT CAN TAKE WHOLE O LINE AB,SO BREAG UP LINE AB INTO
C  NODES AND REAPPLY REW
C  SET LIST OF NODES IN LINE AB
C
      JTB=1
      JT=1
      AF=AB
      CVC(1)=AB
 4712 IF(MIL(AF,2).EQ.0)GOTO 4571
      NJK=MIL(AF,2)
      DO 4265 IJC=1,NJK
      JT=JT+1
      CVC(JT)=MIL(AF,(IJC+2))
 4265 CONTINUE
 4571 JTB=JTB+1
      IF(CVC(JTB).EQ.0.0)GOTO 4572
      AF=CVC(JTB)
      GOTO 4712
 4572 CONTINUE
C RESET CV,TV AND REST OF VARIABLES
      NAT=AA
      TV(NAT)=TV(NAT)-TV(AB)
      CV(NAT)=CV(NAT)-CV(AB)
      TV(N)=TV(N)-TV(AB)
      CV(N)=CV(N)-CV(AB)
      DO 4267 JJ=1,JT
      IJ=CVC(JJ)
      CALL SELECD(IJ,N)
C  MAKE SURE YOU DON'T CONNECT TO CONC THATS FULLY CONNECTED
      DO 4268 III=1,NOM
      IF(MIL(M(III),2).LT.MIL(M(III),1))GOTO 4268
      C(IJ,M(III))=-1
 4268 CONTINUE
      IT=IJ
      CALL NEWCPU(IT,NAT,NOM,IJ)
       CV(IJ)=1
      TV(IJ)=MIL(IJ,1)
      MIL(IJ,2)=0
      PV(IJ)=IJ
 4267 CONTINUE
C
```

```
C WE NOW HAVE ALL NEW PROBLEM CALL REW
      CALL REW(N,NA,NR,NOM,KA)
 4590 CONTINUE
      NNA=MIL(AA,2)+1
      IF(JAC.EQ.(NNA+1))GOTO 4581
      DO 4280   IKL=JAC,NNA
      MIL(AA,IKL)=MIL(AA,(IKL+1))
 4280 CONTINUE
 4581  MIL(AA,2)=MIL(AA,2)-1
      RETURN
      END
C*****************************************************************
      SUBROUTINE SELECD(IJ,N)
      INTEGER*2 TV,M,MT,MNL,CPU,JS,CV,EXAM,MIL
      INTEGER*2 B,PV
     COMMON TV(500),M(500),MT(500),MNL(500),CV(500),JS(500,2),CPU(500)
      COMMON C(320,320),B(320),XA(320),YA(320),PV(320),MIL(320,15)
      DO 4264 IJB=1,N
     C(IJ,IJB)=SQRT((XA(IJ)-XA(IJB))**2+(YA(IJ)-YA(IJB))**2)
 4264 CONTINUE
      C(IJ,IJ)=-1
      RETURN
      END
//GO.SYSIN  DD  *
```

APPENDIX C

BUBBLE SORT



See 'SORT' section of clustering listing for detailed steps.

VITAE

NAME: Raymond A. Vilis

DATE AND PLACE OF BIRTH: Montreal, Quebec, Canada, May 1953

EDUCATION: B.A.Sc. (Electrical Engineering), May 1975,
University of Ottawa, Ottawa, Ontario