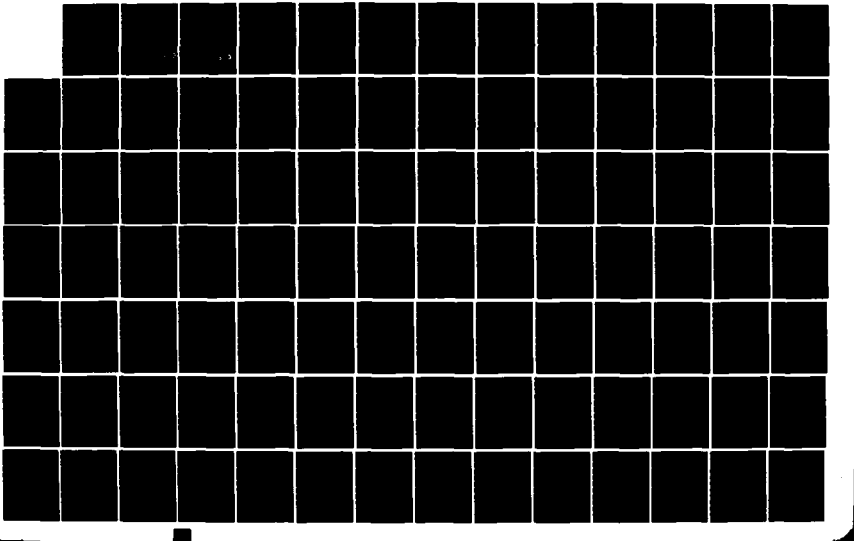
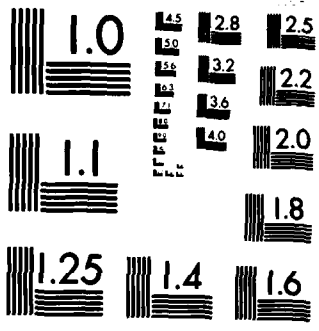
AD-A155 600 A COMPARISON AND ANALYSIS OF VINTR'S GLOBAL ROUTING ALGORITHM WITH THE LE..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. F T CHESLEY

1/2

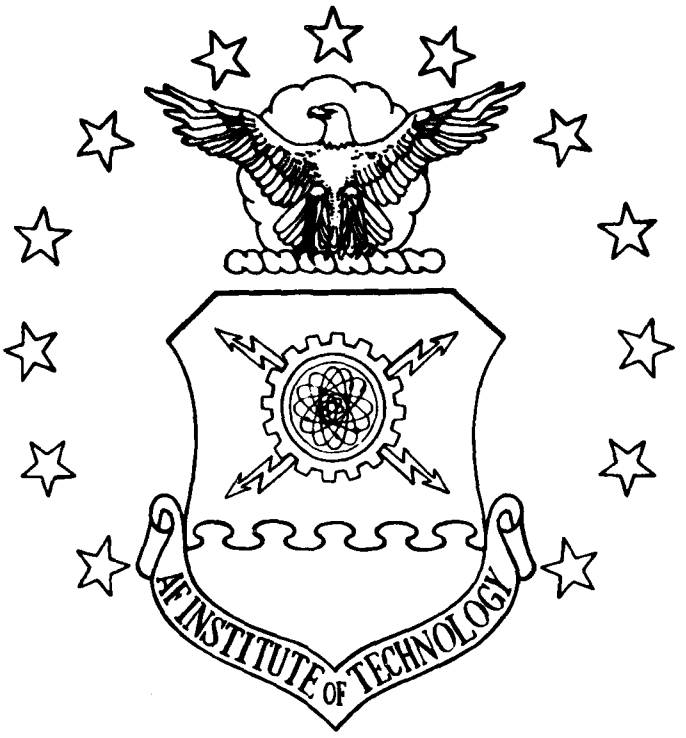
UNCLASSIFIED MAR 85 AFIT/GCS/ENG/85M-1 F/G 12/1 NL





B

(1)



A COMPARISON AND ANALYSIS OF VISTA'S GLOBAL ROUTING ALGORITHM WITH THE LEE ROUTING ALGORITHM IN TWO-LAYER PRINTED CIRCUIT BOARDS

D

Thesis

Fred T. Chesley
Capt USAF

AFIT/GCS/ENG/85M-1

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

A COMPARISON AND ANALYSIS OF VENTK'S GLOBAL ROUTING ALGORITHM WITH THE LEE ROUTING ALGORITHM IN TWO-LAYER PRINTED CIRCUIT BOARDS

Thesis

Fred T. Chesley
Capt USAF

AFIT/GCS/ENG/85M-1

AFIT/GCS/ENG/85M-1

A COMPARISON AND ANALYSIS OF VINTR'S GLOBAL ROUTING ALGORITHM WITH THE LEE ROUTING ALGORITHM IN TWO-LAYER PRINTED CIRCUIT BOARDS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Fred T. Chesley, B.S.

Capt USAF

Graduate Computer Systems

March 1985

Acknowledgments

Lt Col Harold Carter of the Air Force Institute of Technology, Department of Electrical Engineering proposed this project. I am deeply indebted to him for his insightful assistance and patience in the course of my research.

The Institute's Data Automation directorate also deserves many thanks, particularly to 2Lt Mark Strovink who was kind enough to prepare the graphs in this thesis and still manage to accomplish his normal duties. Appreciation is also expressed to the other staff members for their assistance in the draft and final preparation of this document.

Last, but certainly not least, I express my sincere gratitude to my wife, Marion, who provided support and encouragement along the way even during times of frustration.

Fred T. Chesley

Table of Contents

List of Figures

List of Tables

Abstract

Microcomputer software was designed and written to compare a standard routing technique (Lee) with an experimental, unpublished routing technique proposed by J. Vintr for two-layer printed circuit boards.

Vintr's algorithm, as studied in this thesis, uses a four-iteration approach to minimize unroutable nets and minimize route distance.

The unrouted nets and average route lengths were observed and analyzed for differing sizes of two-point nets.

Analysis revealed a reduction of unroutable connections across iterations, but congestion played a heavy role in the overall success of finding paths.

A recommendation is made that use of 8-bit microcomputers in design automation is impractical, and research in this area of technology can best be accomplished using larger computer systems.

A COMPARISON AND ANALYSIS OF VINTR'S GLOBAL ROUTING ALGORITHM WITH THE LEE ROUTING ALGORITHM IN TWO-LAYER PRINTED CIRCUIT BOARDS

I. Introduction

Background Information

The problem of printed circuit board (PCB) routing involves a series of connections between two specific points called terminals. An attempt is made to connect all terminals with no wires interfering with any other wires. Interference may take the form of wires physically intersecting with other wires, two wires too close to each other, or long wires on different layers of a PCB.(7)

These last two problems generally require that the wires maintain a minimum spacing at all points on the board and that wires on two-layer PCBs generally flow in directions perpendicular to each other (i.e. a horizontal flow on one layer and a vertical flow on the other layer).(5) A connection between PCB layers is called a via.(1)

Traditional routing of PCB's has been accomplished in two steps--loose routing and final routing.(7) The loose routing step is the planning stage for final routing. It determines which wires will run through specific pathways on the PCB. The primary consideration in this step is the reduction of congestion through these narrow pathways to avoid bottlenecks. This may necessitate rerouting some wires.

Final routing is the actual allocation of wires to tracks. There are three basic final routers: grid expansion, channel routing, and linear expansion.(7) The linear expansion router is efficient for simple tasks, but is slow, requires a large stack of data, and does not guarantee a connection even if one exists. The channel router always

makes all the connections even if it has to overflow the pathways and destroy the minimum spacing requirement between wires. The grid expansion router (Lee's router) requires a large amount of memory for large, dense PCB layouts, but guarantees a path connection if it exists with the path being the minimum wire distance.(4)

The basic Lee algorithm uses a matrix of grid points or cells to represent the surface or layer of a PCB with the distance between cells as the minimum wire separation. The algorithm (1) begins at a specific cell which is to be connected to another cell. The start cell is labeled as "1". Then all adjacent cells with a Manhattan distance of one are labeled as "2". All unlabeled cells with a Manhatten distance of one from these cells are labeled "3", and so on, until the target cell is reached. A retrace procedure then executed to find a minimum-distance path back to the start cell. Thus, if the target cell was reached on the ith expansion, the retrace begins at the target cell and finds a cell labeled (i-1). This cell is then used to find a cell labeled (i-2). In this manner, the shortest path between two cells is obtained.(6)

There are many variations of the basic Lee algorithm. Rubin (6) discusses variations to allow minimal-state cell coding, paths with minimum turns, and search size reduction. Hoel (3) incorporates an array of stacks rather than a single list to speed up searches. Hoel also implements a cost-encoding scheme which allows retracing codes to be assigned as soon as they are reached.

Algorithms which combine loose and final routing are called global routers.(7) One such global router is by Vintr.(8) This algorithm

initially routes a PCB using a standard router (e.g. Lee's router) and then uses an iterative approach to further minimize the total Manhattan distance of all paths. After the board is routed using the standard router, only a portion (say 10%) of the ends of each path are implemented. This is the end of the first iteration. The second iteration begins with the selection of a single connection, eliminates its tails and reroutes the entire path. Each connection is rerouted in this manner and then a larger portion (say 20%) of the ends of the path are implemented. This iterative process is continued until the implemented portion of each tail equals or exceeds 50%. Soukup states that although Vintr's "router is very efficient, it cannot guarantee all the connections". However, Soukup goes on to state that Vintr's router is the best global router today and, although it may not complete all the connections, it is a "cure for the most frequent cause of unroutable nets on PCB's: the blocking of pins". This research will explore Vintr's method of routing PCBs by looking at whether the algorithm minimizes Manhattan distance and whether the number of unrouted nets decreases.

The Problem

The Air Force Institute of Technology is an educational/research institution through which individuals can be introduced to the field of design automation and further state-of-the-art technology in the field of design automation. The tool to provide the means for this education and training is the Design Automation Hardware System (DAHS)(2) being developed by several AFIT faculty members. DAHS will be a single, dedicated computer system integrating all design automation software at

AFIT into a central data base. One aspect of DAHS will be the design and implementation of a user-oriented, two-layer printed circuit board routing program to be used by AFIT/ENG personnel in their research efforts. Preliminary research is required to develop and fully evaluate an efficient routing algorithm for the design automation of two-layer printed circuit board (PCB) routing. The software will be designed for use by the Department of Electrical and Computer Engineering at the Air Force Institute of Technology (AFIT) in support of DAHS.

Objective

The objective of this project is to provide the first step analysis of a relatively new routing algorithm by J. Vintr (8) using a microcomputer. The research conducted in this study will be the basis for later implementation of microcomputer-based design automation software supporting DAHS for faculty and students.

Standards

Several standards were deemed important for the successful completion of the project. First, the system should be written in a structured language to encourage easy modification and documentation. Second, the routing algorithm should use sub-algorithms designed to reduce run-time and minimize, as practical as possible, the length of connections and the number of unroutable nets. Third, the system should be designed to process virtually any size PCB. Fourth, the capability should be included to allow the user to enter data and to create data files. Fifth, the path for each net routed should be printable to allow later analysis.

Approach to Solution

With the above objectives in mind, the solution was approached in the following four major phases: system requirements, software design, software implementation, and analysis. In the initial phase, the system requirements were necessary to determine constraints the hardware would place on software execution, data storage, and data retrieval. The overall software design phase integrated the hardware and software requirements into a high-level model from which the software could be implemented. In addition, a language had to be selected to fulfill the requirements of modularity and structured design. The software implementation phase used the software design to flowchart the software prior to actual writing of code. The implementation phase also included testing and debugging. The final phase, analysis, was the systematic execution of the program and analysis of the results.

II. System Requirements

Objective

The system was evaluated with the following three requirements in mind: the minimization of unroutable nets, path distance, and processing time. Several parameters were considered in light of these system requirements and constraints.

Board Parameters

First, the effects of the system requirements on the PCB were evaluated. A typical PCB divided into a regular structure of uniform cells applicable to the algorithms discussed throughout this report is shown in Figure 2.1.

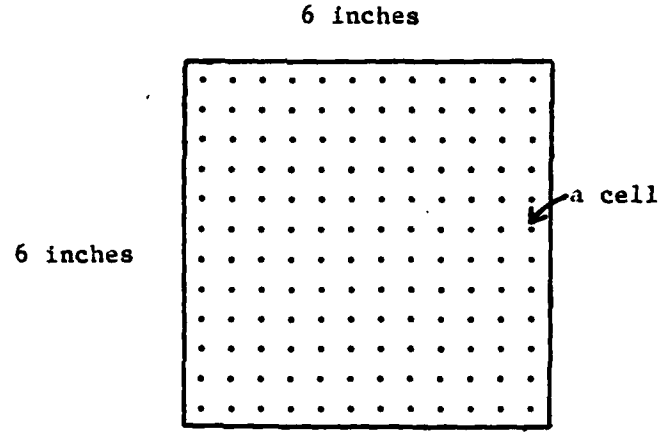
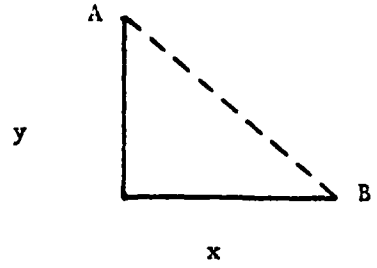


Fig 2.1. A PCB represented by a uniform grid structure

Most printed circuit boards for microcomputer based systems have dimensions of eight by ten inches or less and have two layers upon which the circuits are etched. The wire widths average 50 mils wide with 50

mil spacing between wires.(1) Consequently, the center-to-center distance of parallel wires is 100 mils. A grid structure can then be superimposed on the PCB. The grid points or cells maintain the 100 mil spacing necessary and represent the physical points on the PCB where pins reside or the paths where wire routes pass through. Two cells are adjacent if the Manhattan distance between them is one. A path can be formed only between adjacent cells. Each cell may either be an obstacle, that is, unusable due to physical blockages or it may be available for the placement of a pin or circuit path. This representation greatly simplifies the task of finding the minimum path for a wire. No curves exist and the Euclidean distance does not apply. Rather, distances are calculated on the Manhattan distance measure and the calculation is generally an order of magnitude faster than calculations based on the Euclidean measure.(1)

Figure 2.2 shows the representative difference in calculating minimum distance by Manhattan and Euclidean based measures.



Euclidean: $(x^2 + y^2)**0.5$
Manhattan: x + y

Fig 2.2. Euclidean/Manhattan distance from point A to B

To determine the dimensions of a grid used to represent an eight by ten inch PCB, it is easiest to calculate the number of cells in a one-inch distance and then multiply this value times the board dimensions in inches. Since cells are placed at 100 mil intervals, there are 10 cells per inch. Therefore, the dimensions of the PCB in question are 80 by 100 cells. The total number of cells in a single layer of the board is 8000 and with two layers, we have 16,000 cells. It is clear that the number of required cells would easily overflow a microcomputer's memory if one takes into account that the program, operating system, and system support modules or programs are also needed in core memory at the same time. Later testing showed an unacceptable length of time required to connect random points on this size board. Consequently, to preserve the original nature of the research, the 6 inch by 6 inch board was divided into four equal sized segments as shown in Figure 2.3.

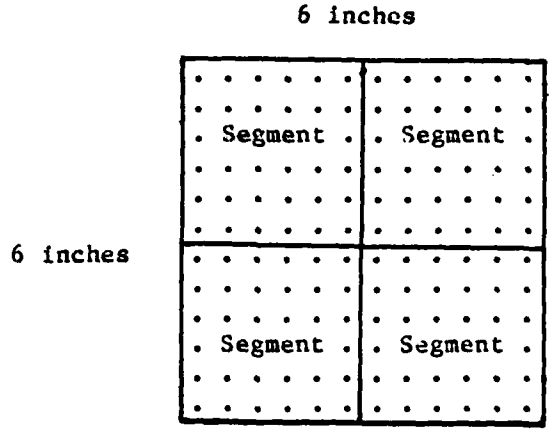


Fig 2.3. A PCB divided into two-by-two segments

Cell Parameters

The second factor influenced by the system requirements is that of cell coding. Hoel (3) and Rubin (6) both discuss strategies for cell-state encoding. Each cell requires a designation of whether or not the cell is free for use. In addition, a designation for the direction of the retrace path is required. The initial rationale for each cell's state allowed for three possible conditions. The first condition is that the cell is unavailable for use. The second condition is that the cell has been reached through previous cell expansion. The third condition is that a cell has not been previously reached and is available for expansion. These three values are simply flag conditions and can be represented by boolean values. It was later discovered that the third condition flag could be eliminated. Consequently, cell conditions could be represented by two bits. If these values are defined during software implementation as a packed array, only one word of memory is required for their storage. The retrace direction was a little more involved. Initially, three different directions were considered, each direction occupying one word of memory. A zero, plus one, or negative one value could then be stored in any of the three locations. However, it would have taken three words just to store the retrace directions. The other extreme would have been to define one word of storage for any and all of the possible retrace directions and decode each value when a path was being retraced. These two extremes were eliminated either due to excessive core memory consumption or excessive processing time devoted to decoding. A compromise was made in this area which occupied only one word of memory. The retrace

directions of horizontal, vertical, and layer change could each be allowed two bits of storage, and if all three were defined in software as a packed array, only one word of memory would be used. The allowable two bit values for each retrace direction were "0" for no change of direction, "1" for a plus one change of this direction, and "2" for a minus one change for this direction. The last value would then be the only value decoded. As a result of this cell analysis, 14,400 words of memory are required for a 6 inch by 6 inch grid. Apple UCSD Pascal allows only 18,000 words of memory available for code and data in any procedure.(9) This constraint allows only a single segment no larger than 3 inches by 3 inches (30 by 30 cells) to reside in central memory at one time. Thus, each segment shown in Figure 2.3 is further divided into cells as shown in Figure 2.4 below.

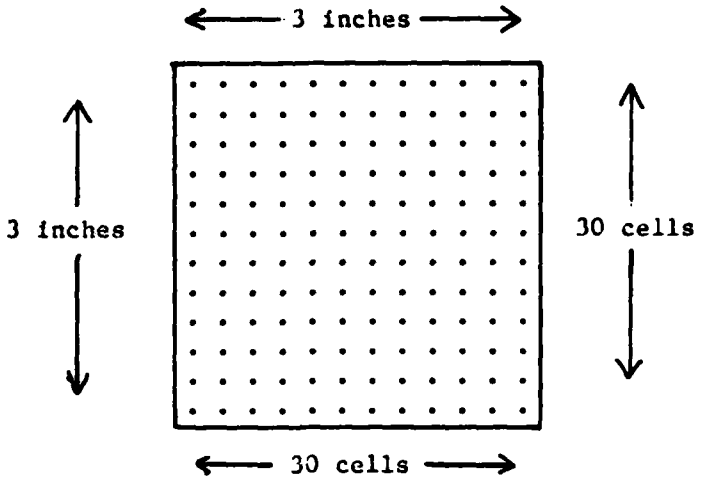


Fig. 2.4. Dimensions of a segment

Secondary Storage

The third factor influenced by the system requirements was that of secondary storage. The Apple UCSD Pascal system formats diskette storage in blocks. Each block stores 256 bytes of data.(9) Thus, to store one 30X30X2 grid segment at two words per cell, fifteen blocks are necessary. For a 6 inch by 6 inch PCB (four segments), sufficient memory exists to store all segments, the retrace file, the input file, and any intermediate files for processing. A user could potentially store 10 segments, the retrace file, an input file, and several intermediate files with the 128K RAM disk. Due to the nature of Apple Pascal, files must be stored on disk in contiguous blocks; therefore, when a file is opened for writing, the largest amount of contiguous storage is reserved for the file. Thus, all remaining storage could be reserved for the writing of one file. This problem is especially difficult if one is writing to multiple files. The solution was to specify an approximate amount of storage for the file. The system would reserve the space and use only the amount it needed when storing the file and release the remainder.

Processing Parameters

The fourth factor influencing the system is the general processing requirements. First, the cost function is the Manhattan distance between adjacent cells of the net. Thus, Lee's algorithm will produce a minimum distance/cost path. Second, Lee expansion is terminated when the net end point is reached then the retrace procedure is performed. Third, expansion is in all directions to ensure finding a path if indeed one exists. Fourth, to conserve memory and still have sufficient room

to store all cells reached during expansion, the largest potential number of cells reached during expansion must be known so the size of the list of cells reached can be defined. For a board of 60 by 60 cells, the approximate center is at cell (30,30). With no obstacles, a total of 30 expansions will result in the maximum number of cells reached, 120 cells per layer or 240 cells overall.

Expansion Directions

There is a general difficulty with any search algorithm as to the most appropriate direction to take. With a predetermined expansion algorithm, the first expansion direction may be opposite to the direction of the end cell. A requirement to always expand a cell in the direction of the end cell was considered but was disregarded since expanding the cell closest to the end cell would largely eliminate the effects of misdirection. The closest cell would expand in no more than three directions before a connection was made while the use of a sophisticated direction prioritization algorithm would add nothing to achieve shorter paths or lower processing time.

Overcoming Segment Processing Problems

As a cell is reached during the expansion phase, its coordinates are saved in a list. The expansion phase uses the entries of this list to reach additional cells, and it is unlikely a connection would be made when expanding the very first entry of the list. That is, the actual cell needed to make a connection may be far down the list and many entries might be needlessly expanded before the cell leading to a connection is expanded. In addition, cells from different segments will be entered into the list as segment boundaries are crossed. The result

of expanding this list will be an additional processing delay due to increased swapping of segments.

To eliminate or reduce the needless expansion of cells and increased segment swapping, an algorithm can be designed to ensure that the cell closest to the end cell is processed first followed by all cells of the same segment. This algorithm should sort the list of reached cells first in ascending Manhattan distance to the end cell and then by segment. The effect of the sorting will reduce both the number of cell expansions and the number of segment swaps.

Data Manipulation Parameters

The fifth factor influencing the system requirements is the need for a general purpose program to allow data entry and to generate a hardcopy listing of each routed net's path and length. The decision was made to provide a separate program for data input, segment initialization and obstacle cell specification. The main routing program would print the route file and path lengths since this file must be used as input to Vintr's algorithm and recreated from one iteration to the next.

Summary

Although the original board parameters were 8X10 inches, it was determined in preliminary testing that an unacceptable length of time was required to route nets. Thus, to preserve the original objectives, the board dimensions were reduced to 6X6 inches. The cell structure was designed such that only the most important information (retrace direction, cell availability condition, and cell reached condition) would be stored in the least amount of memory. Thus, each cell only requires two words of memory. Testing showed the 128K RAM disk provided

adequate space to store all data input, board segments, retrace paths, and intermediate files as long as sufficient space was defined in software. To speed up the connection process and eliminate the number of I/Os, two algorithms had to be designed. The first was to sort the list of cells reached during expansion so the closest to the target would be expanded first. The second algorithm uses the results of the first to build a new list by taking the cell closest to the end cell and all other reached cells on the same grid, then the next closest cell remaining in the reached cell list and all others on the same grid, and so on until no cells remain in the reached list. The last design factor was of data input and output. A separate program was written to input data and build the board segments. Since the retrace file is used as input to Vintr's algorithm, the net paths would be printed with each path's length before the route file is passed to Vintr's algorithm.

III. System Design

Introduction

To meet the requirements previously defined, four major software routines had to be developed. These four routines and how they interact with each other are shown in Figure 3.1, the overall system design. The second, third, and fourth routines are embedded within the box titled "ROUTE DATA FILES".

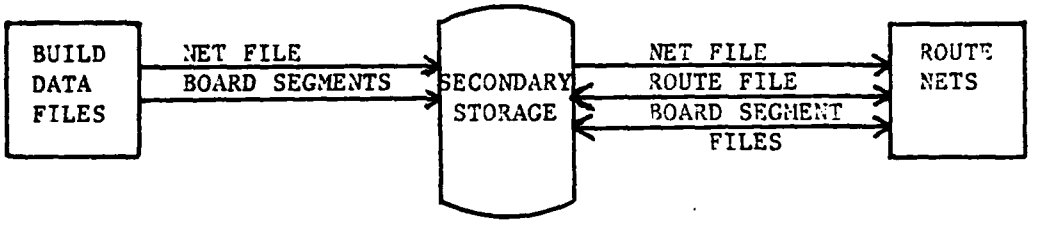


Fig 3.1. System Design

First of all, a data input routine was required to input the cell coordinates for each net added to the net file (the input data file) and the coordinates of obstacle cells. The second and third routines developed were the routing and grid management routines, respectively. These routines are highly dependent on each other for fast, efficient execution. In the fourth routine, a method was needed to provide for the primary direction of expansion. This requires some type of sorting scheme to expand cells closest to the target cell first. As seen from Figure 3.1, there are two separate programs to handle data entry and the routing/printing process.

Input Design

Figure 3.2 shows the overall input design. The user has the option to build the initial data file and/or specify obstacle cells. To build

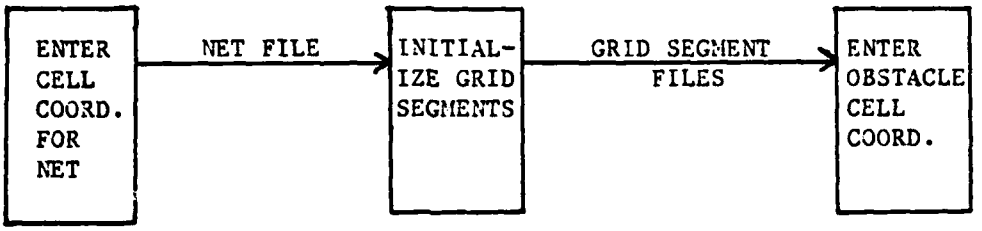


Fig 3.2. Input Design

the data files, the user is prompted for the start cell coordinates and then the end cell coordinates for each net. The coordinates of each net is displayed so the net may be rejected if any of the coordinate values have been entered incorrectly. When all nets have been entered, the entire board is initialized. Individual obstacle cells may then be designated on the board with the capability to ignore an obstacle cell if erroneous coordinates have been entered. All board segments are then saved on secondary storage. This initialization step ensures that all retrace directions and condition flags are set to zero values except for those coordinates which represent occupied cells (a start, end, or obstacle cell). Each start/end cell has its obstacle flag set to "1" on each layer of the grid to represent the physical placement of a chip's pin which typically extends through to the opposite side of the PCB. Obstacle cells are designated by having their obstacle flags set to "1" also. However, differing from start/end cells, obstacle cells must be

specified for each side of the PCB. This method of designating obstacle cells allows special paths to be constructed which are wider than normal, paths such as the power and ground circuits and edge connectors which generally occupy only one side of the PCB.

In summary, the input design allows input of two-value cell coordinates for the start and end points of each net and three-value cell coordinates for obstacle cells. The end points of a net may be rejected if an incorrect value is entered. Once the list of nets and obstacle cells are entered, they are permanently stored on secondary storage and are subsequently used to initialize all board segments.

Output Design

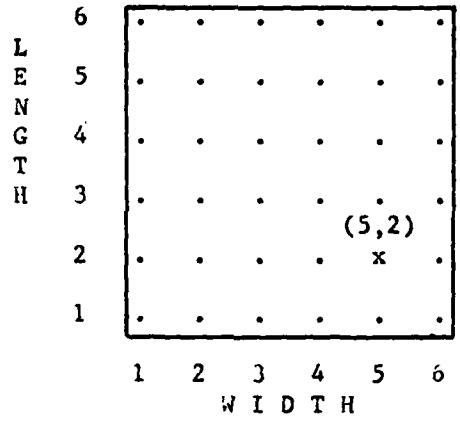
The output portion of the design provides the capability to obtain a printed copy of all net paths once the Lee or Vintr router has finished. There are three methods to do this. The first method merely prints the cell coordinates during the retracing of the path following a connection and does not save the path coordinates for later use in Vintr's algorithm. The second method stores retrace paths in primary memory as the path is retraced. However, this method uses valuable primary memory. The third and favored method stores the path on secondary storage for later output to a printer. The advantages of this method allows output of the paths as in method one, although not as fast as in method two, but does not use valuable primary memory to store the path. Another significant advantage is that the paths are permanently recorded for later use in Vintr's algorithm. As a path is retraced, the three coordinate values for each cell along the path are saved to pin-point the precise location of the path on the grid. To avoid the tedious

manual method of determining path distance, the output routine calculates the length of a particular path as it is printed.

In summary, three-valued coordinates for each retrace cell are saved on secondary storage to minimize system turnaround time and allow for their use in Vintr's algorithm. As paths are printed, an internal counter sums the number of cells reached during the retrace process so the total path distance can be printed.

Grid Design - Referencing Cells

The strategy of how one locates a cell in a large array when only a sub-portion of the large array is immediately available has a profound effect on the efficiency of Lee's algorithm. This dilemma suggests two representations of a specific cell location. The first representation is of a cell's absolute location and is shown in Figure 3.3.

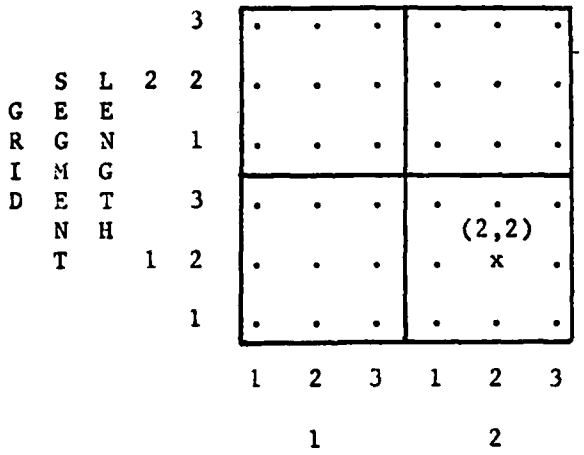


The absolute location of cell x is (5,2).

Fig 3.3. Absolute cell location

(It should be noted in reference to cell coordinates that the first coordinate denotes the width parameter and the second coordinate denotes the length parameter.) For example, if the board dimensions are 6X6 cells, a cell can be located simply by specifying its coordinates.

The second representation requires the coordinates for the cell's location within a segment and the coordinates for the segment's location within the board. For example, if only a portion of the entire grid is available at one time, say 3X3 cells, a cell's location must be specified within the segment, and the segment must be referenced with respect to its location on the overall board. Figure 3.3 references the same cell as in Figure 3.4 using segment coordinates and relative cell coordinates.



The relative location of cell x is (2,2) in segment (2,1).

Fig 3.4. Relative cell location

As cells are expanded and others are reached during the actual routing process, segment boundaries are crossed. Consequently, if both representations are kept in primary memory, the routing process could take place using absolute coordinates while a second algorithm could keep track of the segment's coordinates and of the cells within the segment. Whenever a cell other than the current cell is referenced due to expansion or retrace, a check is made to determine if the segment has changed. If so, the grid management routine (to be described later) is invoked. If the segment has not changed, new relative cell coordinates are calculated so the reached flag and the retrace direction are specified for the proper cell. The use of absolute coordinates for expansion and retrace and the use of relative coordinates for access to specific cell information is especially advantageous during retrace. The coordinates of the retrace path are saved as absolute coordinates, and retrace direction data from a cell (referenced by its relative coordinates) is used to calculate the absolute cell coordinates of the next cell in the retrace path.

Lee Router Design - Algorithm

The design of the Lee algorithm used in this project is basically a process of reading the end cells of a net, initializing various data parameters, expanding cells until a connection is made or is impossible, and retracing the path for a connected net. This same process is repeated for each net. Figure 3.5 shows the overall design of the Lee algorithm.

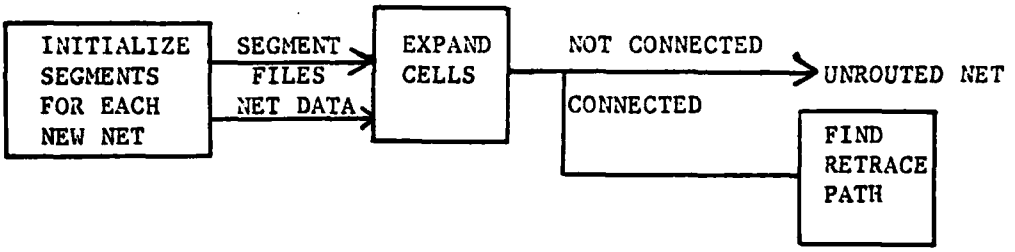


Fig 3.5. Lee algorithm design

Lee Router Design - Initialization

Upon entry into the Lee algorithm, the cells of all segments must be initialized. The initialization process resets all retrace directions to zero and the reached flag to "not reached" for all non-obstacle cells. The obstacle cells, at this point, represent both physical obstacles and the wire paths between cells. Once all segments have been initialized, the data lists for storing cells to be expanded (ELIST) and cells that are reached during expansion (RLIST) are also initialized so that neither list has any entries.

When a net to be routed is read from the net file, its start cell is immediately stored in ELIST and the same cell on side 2 of the PCB is stored in RLIST. Once these processes have taken place, the expansion process continues until a net is connected or it is determined that a connection is impossible.

Lee Router Design - Expansion

The design of the expansion process the heart of the Lee algorithm. It is the method of taking a cell with specified coordinates and determining if a path exists to each adjacent cell until a connection is made or no paths exist. For any one cell there is a maximum of five

adjacent cells which may be reached on a two-layer PCB. These five cells' coordinates can be easily calculated from a single cell's coordinates. A single cell has three coordinate values - width, length, and side. A fourth coordinate value can be calculated by adding "+1" to the cell's width coordinate, a fifth coordinate value by adding "-1" to the width coordinate. Similarly, the sixth and seventh coordinate values can be calculated from the cell's length coordinate. The side coordinate is trivial since it always has a value of "1" or "2". Since these adjacent coordinates are always calculated in the same manner, they are easily defined. As an example, if a cell with location (x,y,z) is to be expanded, the five adjacent cells are defined below.

CELL LOCATION:	(x,y,z)
ADJACENT WIDTH CELLS:	(x-1,y,z) and (x+1,y,z)
ADJACENT LENGTH CELLS:	(x,y-1,z) and (x,y+1,z)
ADJACENT SIDE CELL:	(x,y,1) or (x,y,2)

If the four width and length coordinate values are saved, they may be easily used to reference adjacent cells. Adjacent width cells are defined by using one of two newly calculated width values to replace the current cell's width coordinate. The adjacent length-wise cells can be referenced in a similar manner. By sequentially replacing the original values by the new values, all four directions may be considered without ever permanently changing the original cell coordinates.

Initially, no cells have been reached and the only condition flag set is the obstacle flag for the start/end cells and other obstacle

cells. As the start cell is expanded (an ELIST entry), each adjacent cell that is reached has its three coordinates saved in RLIST. The first expansion has at least one cell that may be reached unless the connection is impossible. If more than one cell is reached, each cell is saved. When the start cell has been fully expanded, the reached cells are moved into ELIST and expanded one at a time. To avoid repetitious expansion into the same cell, a reached cell has its reached flag set. Only available and unreached cells are eligible to be reached in the expansion process. If a connection has not been made by the time the last ELIST entry is expanded, the list of reached cells (RLIST) is moved into ELIST and expanded. This process is repeated until a connection is made or no further cells are available for expansion. Once a connection is made, the retrace procedure is invoked. If all ELIST entries have been expanded and no RLIST entries exist, the net is considered impossible and is saved in a list (IMPOSLIST).

Lee Router Design - Retrace

Once a connection has been made, the path should be retraced. Since a connection is made with the end cell, the retrace begins at the end cell. The next cell of the path is calculated from the current cell's absolute location and the retrace direction information stored in the current cell. When the coordinates of the next cell are the same as the start cell's width and length coordinates, the retrace path is complete. During this process, each new absolute cell location of the path is saved in the route file and the obstacle flag for each cell is set.

Of all the data items within the program, the items which may use more primary memory than any others are the retrace directions. If

negative values are allowed, each direction requires an entire "word" of memory. With three directions for each cell and hundreds of cells, it is easy to see the problem. To minimize storage for the grid entries, the retrace directions are coded as "0", "1", or "2". Then if data packing is used in the software implementation, only two bits are required for each of the three retrace directions. However, coding a "2" (a "10" in binary) for a negative value requires a check during retrace. The check will ensure a value of minus one is actually added to the coordinate in question to obtain the next cell in the retrace path. The other two values, "0" and "1", can be added directly to the proper absolute coordinates to determine the next cell location in the retrace path. For this technique to function properly, the retrace direction values must be stored properly. As a cell is expanded in one direction, the cell reached must have the value of the opposite direction (from which it was entered) stored in the proper retrace direction. Perhaps the easiest way to do this and the most foolproof way of ensuring the proper values have been stored is to "hard-code" this technique into the program.

Vintr Router Design

Vintr's router uses the Lee routing algorithm to do its work but requires some additional algorithms to function properly. Figure 3.6 shows the overall design of Vintr's algorithm and how it uses the Lee algorithm.

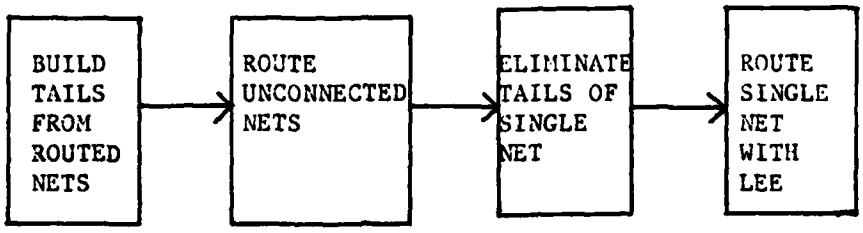


Fig 3.6. Vintr algorithm design

Once all nets have been expanded, the route paths previously saved to disk can be used for the first step of the Vintr router. The first algorithm is to build a file of path "tails". The tail lengths are calculated from the length of the routed net, the percentage of each path to implement (10% in this study), and the iteration. The end points of the net are added to the list of tail cells. Then the Manhattan distance between each intervening cell along the retrace path and the start or end cell is compared with the length of the tail to determine if the intervening cell should be included in the tail list. Figure 3.7 shows an example of a board after the Lee algorithm. Figure 3.8 shows the same board after the tails are built.

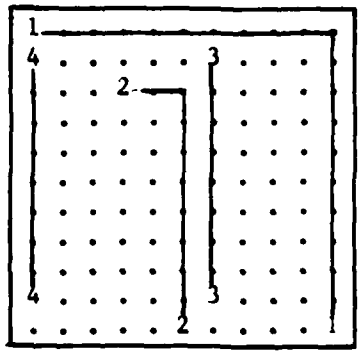


Fig. 3.7. Example of four routed nets

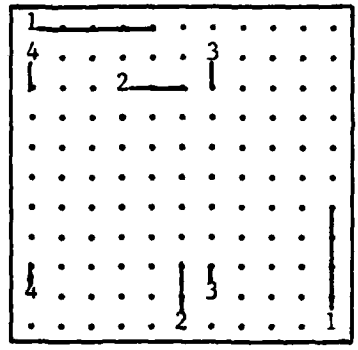


Fig. 3.8. Example of the tails for four routed nets

Intervening cells included in the tail list are flagged as "unavailable" and those cells not included are flagged as "available". This process is continued until tails are calculated for all routed nets.

When all the tails have been built, a second algorithm is executed. This algorithm attempts to find a path for previously unrouted nets. The unrouted nets are expanded by the Lee algorithm in an effort to increase the net completion rate. If a net can not be connected, it is again saved in the unrouted net list (IMPOSLIST); otherwise, the net and all cells along its retrace path are saved in the route file (ROUTFILE).

At this point, a third algorithm selects all of the tail list entries for the end points of a single net and eliminates the tails by turning the obstacle flag off, making the cell "available". The end points of the net are then passed to the Lee algorithm for routing. This process continues until all tails are routed. The algorithm then begins the next iteration.

In summary, for both the Lee and Vintr algorithm, separate sets of coordinates are maintained to identify a cell's location on the grid.

The first set of coordinates is used for the absolute location of a cell. The second set of coordinates are used to determine the relative cell location based on segment location and the cell location within the segment. Relative values are used when storing the condition flag and retrace direction for a specific cell. In the basic Lee algorithm an initialization procedure is performed, all possible directions of expansion are attempted, and a list of reached cells is constructed. The list is then used for expansion after all current cells have been expanded. Once a connection is made, the retrace procedure allows simple path retracing and storage of the path. Vintr's router requires several additional algorithms to build the tails of routed nets, to expand previously unrouted nets, and to expand the tails one at a time.

Grid Management Design

Two methods of grid management can be considered each with its own advantages and disadvantages. The first method is called the grid I/O method and the second method is called the grid vector method. The method of grid management is the single most significant factor in turnaround time. The first method, that of grid I/O, functions as follows. Anytime a new cell is referenced, a check is made to determine if the cell is located in an adjoining segment. If it is not, no extra work is done except for the calculation of its relative coordinates. However, several actions are initiated if the cell is in an adjoining segment. First, the old segment is saved to secondary storage. Second, the new segment is loaded into memory. Third, the cell's relative coordinates in the new segment are calculated. The difficulty with this algorithm arises when multiple cells along a segment border

expand across segment boundaries. This requires a number of segment swaps resulting in a tremendous amount of processing time doing I/O. The advantage in this method (which will become clearer from the discussion of the vector method) is in the fact that the algorithm allows a cell which could possibly make a connection when it first crosses a segment to make the connection.

The grid vector method expands all cells within a segment before swapping segments. This method requires that a "vector" or a list of all cells on a segment boundary be saved prior to segment swapping. Distances also need to be saved so that once a new segment is loaded, processing of the vector entries would begin with those with the shorter distance. The entries with the shortest distances would be processed one at a time until each path's distance was equal to the longest distance saved in the boundary vector. This ensures that a longer path did not make a connection before a shorter path was given an opportunity to complete the connection. The significant advantage of this method lies in the fact that a tremendous number of segment swaps are eliminated which reduces turnaround time. However, there are significant disadvantages also. The first disadvantage is that additional memory is required to record sixty (two layers of thirty entries each) entries per vector and the possibility of four vectors. In addition, if two cells are relatively close to each other but in different segments, much processing time would be spent in expanding all other cells of a segment before the connection could be made.

In summary, the two grid management methods will determine overall processing time. Each has its own advantages and disadvantages. One

method (I/O) will make a connection with a fewer total number of cell expansions but with a price of high I/O. The other method (vector) requires a larger number of cell expansions and larger amount of primary memory but refrains from any I/O until a segment swap is absolutely necessary.

Expansion List Sort Design

The primary reason for these algorithms is to expand cells closest to the target cell first and to expand all cells that have been reached in a specific segment before segments are swapped. This method provides some assurance that the target cell is reached as soon as possible and with a reduction in the number of segment swaps. The first algorithm sorts the cells reached during a particular expansion into ascending Manhattan distance sequence. The Manhattan distance is then calculated from the cell's current position to the end or target cell's position. The advantage here is that for long paths, there may be 100+ cells reached, and the cell closest to the target cell may be at the end of the list. Clearly, it would be a waste of time and resources to process all others first before making the connection. A second algorithm uses this sorted list to reduce the number of segment swaps. The second algorithm uses the first entry of the sorted list and calculates which segment it is located in, then searches the remainder of the list for other entries in the same segment. When all entries have been checked, the next closest entry is selected and its segment is calculated. This process is repeated until no "reached" cells remain. The resulting expansion list (ELIST) has entries sorted by segment which is based on the cell closest to the target cell. This method will reduce some of the segment swaps during expansion.

Summary

The design of several routines had to be considered before any program implementation could begin. A routine had to be devised to handle cell location within a segment for a multi-segmented grid. Routines had to be designed to input and output data files and to initialize the segments for obstacle cells and the end points for all nets. A routine was required to expand cells and mark their retrace directions and, once a connection was made, a separate routine had to find the path back to the start cell. During the expansion process, routines had to be developed to reduce the number of disk I/O and to first expand those cells closest to the end cell.

IV. Results and Analysis

Procedure

Four data sets were created with 20, 40, 60, and 80 randomly created two-point nets. Each net was created independently. Some characteristics of these data sets are shown in Table A. No obstacle cells were entered in any of the board configurations prior to routing.

TABLE A

Characteristics of the data sets used in analysis

	Number of 2-Point Nets			
	20	40	60	80
Minimum Net Length (cells)	9	6	5	6
Maximum Net Length (cells)	78	107	85	69
Average Net Length (cells)	38.2	40.2	39.4	35.5

The nets in each data set were sorted and stored by ascending and descending Manhattan distance. Thus, eight data sets were actually created and used for the analysis presented here. All data sets were saved to floppy disk. Each data set required no more than one block of storage, while each segment required 15 blocks of storage. Temporary storage files and intermediate route files were created and purged during program execution and, thus, their size could not be easily determined.

To execute the main program, all files associated with a specific data set were loaded into the RAM card. The program was then started and the clock time noted. At the completion of the program, the time

was noted and all files in the RAM card were saved to floppy diskette. These results are shown in Table C and discussed in the next section.

At the end of the Lee router and each iteration of Vintr's router (where each iteration of Vintr's router performs the Lee algorithm with an increase in tail length of 10%, see Table B), the path for each successful connection and its Manhattan distance was printed.

TABLE B

Percent of route path used to calculate tails

STEP	TAIL LENGTH
LEE	0%
VINTR #1	10%
VINTR #2	20%
VINTR #3	30%
VINTR #4	40%

Each Manhattan distance was subsequently entered into a calculator to obtain the mean and range of routed distances for each data set. This data is shown in Tables D through K and discussed in the following sections.

Output File Size and Processing Time

Table C shows the results of output file size and processing time for all eight data sets by net-size and the routing of short vs. long nets. Short nets are the data sets sorted in ascending Manhattan distance, and long nets are the data sets sorted in descending Manhattan distance. End-of-program output file sizes for the various data sets

Table C

Results for output file size and processing time

	Size of 2-Point Nets							
	Short First				Long First			
	20	40	60	80	20	40	60	30
Output File Size (Blks)	7	14	26	28	7	15	24	29
Processing Time (Hrs)	71.8	124	131.2	155	77	136.5	150.4	138.3

were variable and tended to level off in the 24-29 block range for the four largest data sets. This is due to the substantially higher number of unrouted nets in the 80 net data sets. Figure 4.1 shows the increase in output file size over data set size.

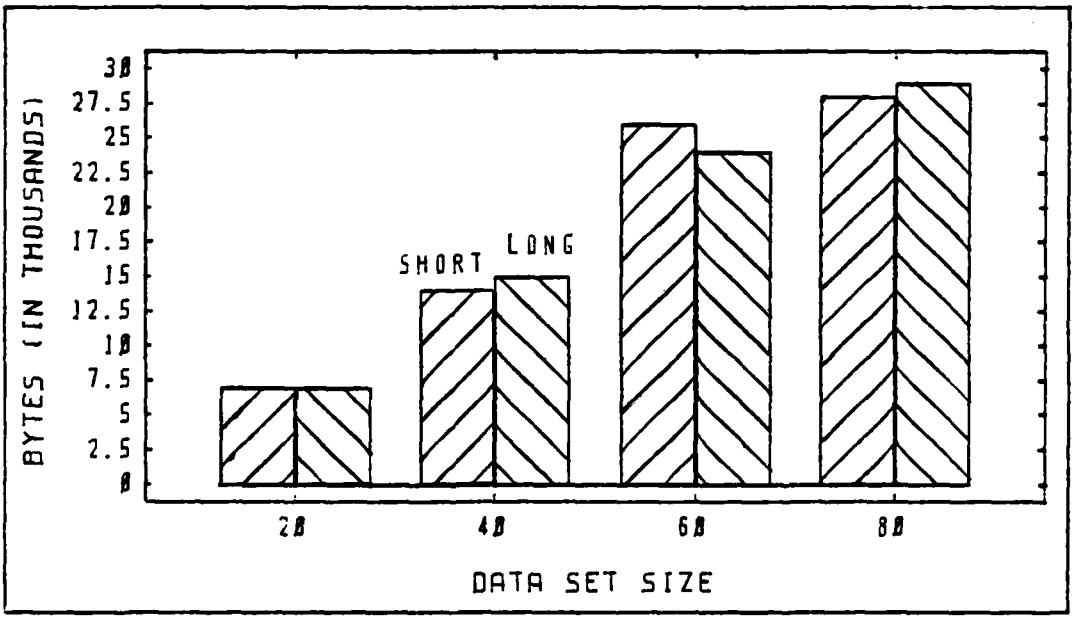


Fig. 4.1. Output File Size Results

As can be seen from Figure 4.2, the processing time was substantial, ranging from 71.8 to 155 hours per run of a data set. The increase is due largely to the increase in the number of nets to be routed and the resulting increase in routing congestion. The time nearly doubled going from 20 to 40 nets as would be expected on a relatively uncongested board. With congestion though, the time began to peak in the 135-155 hour range. This is explained by the fact that as more obstacles are encountered in the expansion process, fewer cells are reached. It is interesting to note for all data sets, except the two largest data sets, less time was required to route the shortest connections first. In processing the longer nets first of the largest data set, there were about 36% more unrouted nets for each iteration than when the shorter

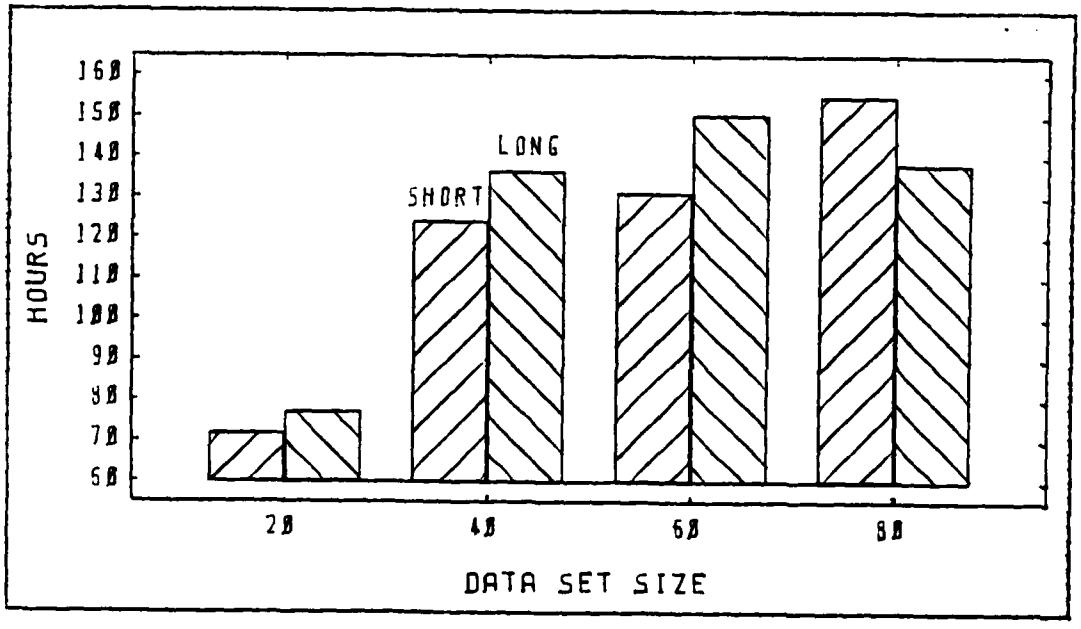


Figure 4.2. Processing Time Results

nets were processed first. These additional, unrouted nets were probably flagged as impossible early in the routing process to achieve such a low processing time.

Output Results and Analysis - 20 Nets

Tables D and E show the results for 20 nets by iteration for routing shortest and longest nets first, respectively. For each type of processing (shortest vs. longest), all nets were routed using the Lee algorithm. Consequently, no improvement was observed using Vintr's algorithm. It is interesting that even though the average route length was slightly longer when processing the longest nets first, the maximum path distance was several units less. Apparently, longer nets were closer to their minimum Manhattan distances and shorter nets were longer due to the obstructions generated by the longer nets. When the shorter nets were processed first, they achieved their minimum distance with longer nets requiring longer paths to make a connection.

Table D

20 Nets (shortest routed first)

	Iteration				
	Lee	1	2	3	4
Minimum Net Length (cells)	9	9	9	9	9
Maximum Net Length (cells)	82	82	82	82	82
Average Net Length (cells)	39.2	39.2	39.2	39.2	39.2
Number of Unrouted Nets	0	0	0	0	0
% Completed	100.0	100.0	100.0	100.0	100.0

Table E

20 Nets (longest routed first)

	Iteration				
	Lee	1	2	3	4
Minimum Net Length (cells)	9	9	9	9	9
Maximum Net Length (cells)	79	79	79	79	79
Average Net Length (cells)	39.6	39.6	39.6	39.6	39.6
Number of Unrouted Nets	0	0	0	0	0
% Completed	100.0	100.0	100.0	100.0	100.0

However, the minimum, average, and maximum route lengths in both methods of processing very nearly equalled their starting statistics. This condition indicates the net end points were sufficiently distributed, and the board was large enough that congestion was not a problem.

Output Results and Analysis - 40 Nets

The results for the 40 net data sets are shown in Table F for shortest nets routed first and in Table G for longest nets routed first. Again we see a 100% completion rate. It is also interesting to note that the processing of longest nets first resulted in a shorter maximum

Table F

40 Nets (shortest routed first)

	Iteration				
	Lee	1	2	3	4
Minimum Net Length (cells)	6	6	6	6	6
Maximum Net Length (cells)	109	109	109	109	109
Average Net Length (cells)	42.0	42.0	42.0	42.0	42.0
Number of Unrouted Nets	0	0	0	0	0
% Completed	100.0	100.0	100.0	100.0	100.0

Table G

40 Nets (longest routed first)

	Iteration				
	Lee	1	2	3	4
Minimum Net Length (cells)	11	11	11	11	11
Maximum Net Length (cells)	107	107	107	107	107
Average Net Length (cells)	44.5	44.5	44.5	44.5	44.5
Number of Unrouted Nets	0	0	0	0	0
% Completed	100.0	100.0	100.0	100.0	100.0

route length than did the processing of the shortest nets. Also, the number of nets to be routed does not congest the board enough to really affect the path lengths. The difference between the minimum possible length and the average route length of a net when shortest nets were routed first was only an increase of 1.8 cells. This value increased to 4.3 cells when routing the longest nets first. Thus, the cost for routing the longest nets first is an increase in the path length of the shorter nets, and path lengths increase to the extent that the average route length increases significantly, especially as board congestion increases.

Output Results and Analysis - 60 Nets

Tables H and I show the results of applying Vintr's algorithm on the 60 net data sets for routing shortest and longest nets first, respectively. These data sets were the first to show the usefulness of Vintr's algorithm. In routing the shortest nets first, only one net

Table H

60 Nets (shortest routed first)

	Iteration				
	Lee	1	2	3	4
Minimum Net Length (cells)	5	5	5	5	5
Maximum Net Length (cells)	125	90	116	177	155
Average Net Length (cells)	44.6	39.3	48.8	51.1	53.4
Number of Unrouted Nets	1	8	5	4	0
% Completed	98.3	86.7	91.7	93.3	100.0

Table I

60 Nets (longest routed first)

	Lee	Iteration 1	2	3	4
Minimum Net Length (cells)	5	7	5	5	5
Maximum Net Length (cells)	180	136	127	148	123
Average Net Length (cells)	55.7	55.7	49.5	50.7	49.8
Number of Unrouted Nets	7	9	5	1	1
% Completed	88.3	85.0	91.7	98.3	98.3

could not be routed. Only after an increase in unrouted nets, as shown in Figure 4.3, was the Vintr algorithm able to complete all connections. Surprisingly, the early iterations of Vintr's algorithm showed a

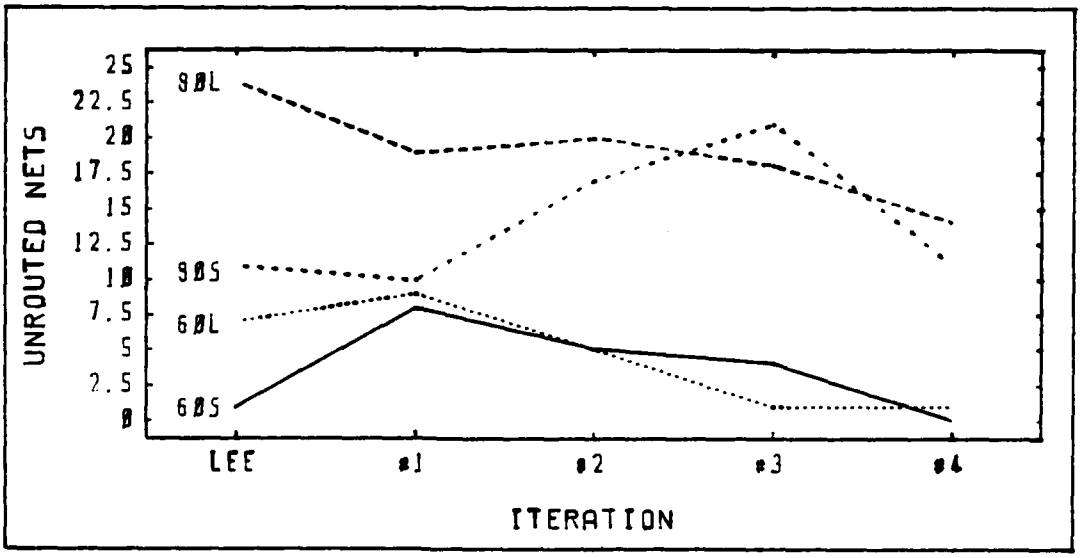


Figure 4.3. Unrouted Nets for 60/90 Net Data Sets

significant non-completion rate (see Table H). This "hump" effect (see Figure 4.3, a chart of the data in Table G) is apparently due to the following sequence of actions. At the start of each iteration in Vintr's algorithm, unrouted nets are routed first. In this way, the newly routed nets force other nets to take a more circuitous path, and only when less congestion is encountered are the previously routed nets more likely to find a connection. The construction of tails increases a net's chance to find a path, and until a net finds a path outside the congestion, additional nets may not be routed.

The average route lengths increased dramatically from the starting lengths as shown in Figure 4.4. Obviously, congestion played a major

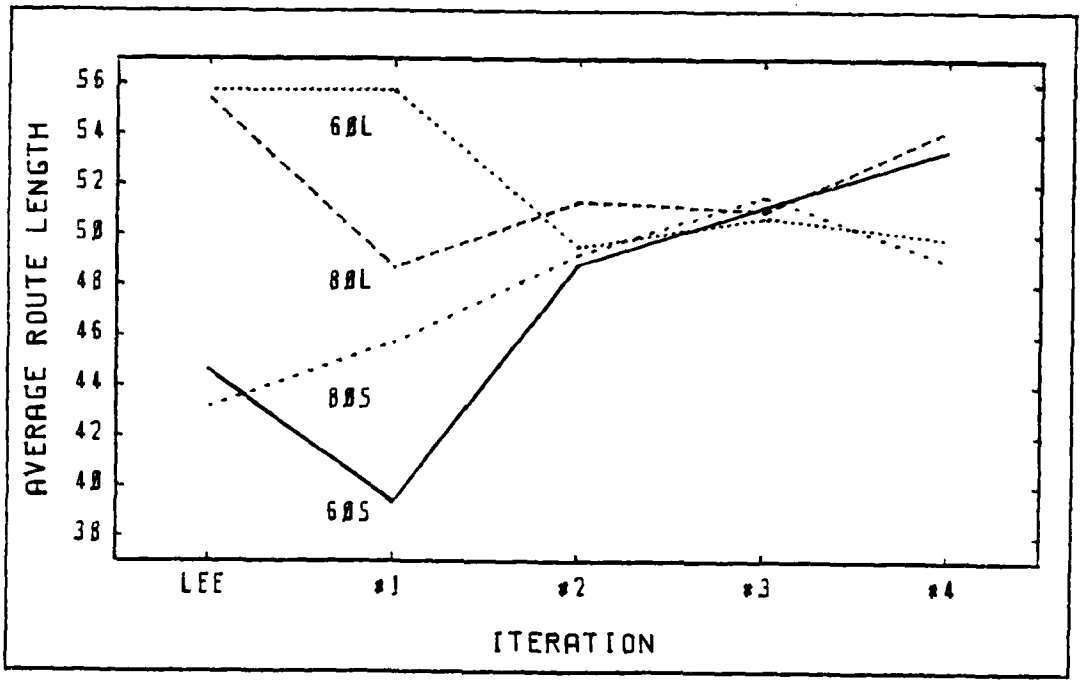


Figure 4.4. Average Route Length for 60/80 Net Data Sets

part in the processing of this data. One interesting observation when routing longer nets first is there appeared to be an improvement in both the maximum and average route lengths for the final two iterations. This apparent improvement was due to the fact that the unrouted net in iteration three was different than the one in iteration four. The unrouted net in the last iteration caused less congestion than the preceding net.

Output Results and Analysis - 80 Nets

Table J shows the results for 80 nets with shortest nets routed first versus Table K which shows the results of the same data set ordered by longest nets first. It is clear from this data that the board was heavily congested. For routing shortest nets first, there was no improvement in unrouted nets by Vintr's algorithm. In fact, the improvement seen was in the decrease in maximum net length.

Table J

80 Nets (shortest routed first)

	Iteration				
	Lee	1	2	3	4
Minimum Net Length (cells)	6	6	6	6	6
Maximum Net Length (cells)	201	138	159	233	162
Average Net Length (cells)	43.1	45.7	49.2	51.5	48.9
Number of Unrouted Nets	11	10	17	21	11
% Completed	86.3	87.5	78.8	73.8	86.3

Table K

80 Nets (longest routed first)

	Iteration				
	Lee	1	2	3	4
Minimum Net Length (cells)	6	8	10	8	6
Maximum Net Length (cells)	155	169	181	220	242
Average Net Length (cells)	55.4	48.7	51.3	50.9	54.2
Number of Unrouted Nets	24	19	20	18	14
% Completed	70.0	76.3	75.0	77.5	82.5

The "hump" effect can be seen when routing the shortest nets first (as seen in Figure 4.3), and the result was an 86.3% completion rate. In routing longest nets first, the number of unrouted nets began high and slowly improved with a completion rate of 82.5%. For those iterations where the least number of unrouted nets occurred, only about 21% of all possible cells on the entire board were used. This suggests perhaps the "random" generation of data points actually resulted in data being somewhat "grouped" on the board. Nets having an end point near the center of the "group" would have the greatest difficulty finding a path.

It is clear from Figure 4.4 that the average route length for these two data sets is difficult to analyze since there is a great amount of congestion. When routing short nets first, a comparison between the results of Lee's algorithm and the last iteration of Vintr's algorithm shows a significant decrease in the maximum net length and the average net length. On the other hand, when routing longer nets first, the average route length only slightly decreased, but the maximum net length increased significantly as a much better completion rate was achieved.

Summary

The Lee algorithm worked very well with uncongested boards. When congestion increased to the point where even Vintr's router could not reduce the number of unrouted nets to zero, Vintr's algorithm was able to improved the completion rate. It appears the maximum number of cells to be connected was around 65-70 using a random generation method of producing input data. For data sets of 60 and 80 nets, the better completion rate occurred with the processing of the shorter connections first.

In all cases, Vintr's algorithm gave results that were as good or better than Lee's algorithm. In most cases, Vintr's algorithm gave significant improvements in routing completion and average net length, it appears that the number of long nets in a data set will significantly impair the ability of Vintr's router to successfully route all nets.

Improvement, even though feasible, will come at a significant processing cost. In general, Vintr's algorithm required I*L processing time where I is the number of iterations, and L is the length of time to route nets using the Lee algorithm.

V. Conclusions and Recommendations

Conclusions

The most significant observation in this work is the fact that design automation using 8-bit microcomputers is unrealistic. The 6502 processor is very slow running about 1 MHz. With the constraint of 64K primary memory only small sections of a printed circuit board can be available at a time, in this case only a 3X3 inch segment. With the processing of larger boards I/O becomes of increasing importance in the execution speed of the algorithms. In addition, due to the segment swapping, algorithms were required to detect board segment changes and to convert absolute coordinates to relative coordinates on a continual basis. Even with faster processing speeds and either RAM disk or hard disk for secondary storage, the amount of I/O and the amount of time to consider all possible routes results in a substantially slower application. The size of the board in this case was only 6X6 inches. Initial testing of a board 8X12 inches with 30 two-point nets was taking well beyond a week. This situation is entirely unacceptable in an environment where students require faster turnaround.

The Lee algorithm produced the minimum length path in all cases unless congestion impeded the expansion process. The Vintr algorithm worked as advertised but there seemed to be a fairly narrow range of data set size that really allowed the algorithm to do its work. On one extreme, all nets were connected on the first execution of Lee's algorithm and subsequent execution of Vintr's algorithm did not reduce the average route length. On the other extreme, the board became so congested that it was difficult to predict, let alone expect, the number

of unrouted nets to decrease. One fact is for certain though, Vintr's algorithm will result in a fewer number of unrouted nets at the expense of processing time.

A reduction of the overall processing time should be expected if a 16-bit microcomputer is employed to accomplish the expansion process. A faster processor should significantly speed up processing and the increased range of memory addressing would allow much larger sections or an entire printed circuit board to reside in memory at one time. This would eliminate the algorithms devoted to segment swapping and for the conversion of absolute coordinates. Potentially, a 75% reduction in processing time might be observed using this program with different hardware configurations. Vintr's algorithm could be a viable approach if hardware specifically tailored for routing algorithms or design automation (similar to a database machine) were used.

In addition, if the pin data for the layout of actual chips was used, the net's end points should be more uniformly distributed on the board. Consequently, some of the congestion should be eliminated.

Varying the method of processing might also bring unexpected results. In the data sets where all connection were made, Vintr's algorithm did not affect the routes, but if the routing method was alternated from shortest processed first to longest processed first, different results might be observed.

Recommendations

No further research is recommended unless a more powerful microcomputer can be used to study the effects of Vintr's algorithm. At best this research could provide an introduction to the routing process

for students and the algorithms could be studied by students and improved upon. However, for true research purposes a larger-scale computer system is required to lower processing times to a tolerable level. The results of varying the length of tails and the number of iterations during Vintr's algorithm might prove interesting or significant.

BIBLIOGRAPHY

1. Akers, Sheldon B. "Routing," Design Autmation of Digital Systems. Vol. 1, Melvin A. Breuer (ed.), Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1972.

2. Carter, Harold. "A Plan for Digital Systems Design Automation," unpublished plan for the Air Force Institute of Technology, 1982.

3. Hoel, Jeffrey H. "Some Variations of Lee's Algorithm," IEEE Transactions on Computers, C-25: 19-24 (January 1976).

4. Lee, C. Y. "An Algorithm for Path Connections and Its Applications," IRE Transactions on Electronic Computers, EC-10: 346-353 (September 1961).

5. Newton, Richard A. "Computer-Aided Design of VLSI Circuits," Proceedings of the IEEE, 69: 1189-1199 (1981).

6. Rubin, Frank. "The Lee Path Connection Algorithm," IEEE Transactions on Computers, C-23: 907-914 (September 1974).

7. Soukup, Jiri. "Circuit Layout," Proceedings of the IEEE, 69: 1281-1304 (1981).

8. Vintr's router as reported by Soukup. J. Vintr is the inventor of the algorithm that was reported by R. Dutta at the 1980 CANDE workshop on hardware for CAD, Univ. Michigan, Ann Arbor. There were no proceedings published.

9. Apple Pascal Operating System Reference Manual and Apple Pascal Language Reference Manual, Apple Computer, Inc., Cupertino, California (1980). Apple is a registered trademark of Apple Computer, Inc. UCSD Pascal is a registered trademark of the Regents of the University of California.

APPENDIX A: SADT Design Diagrams

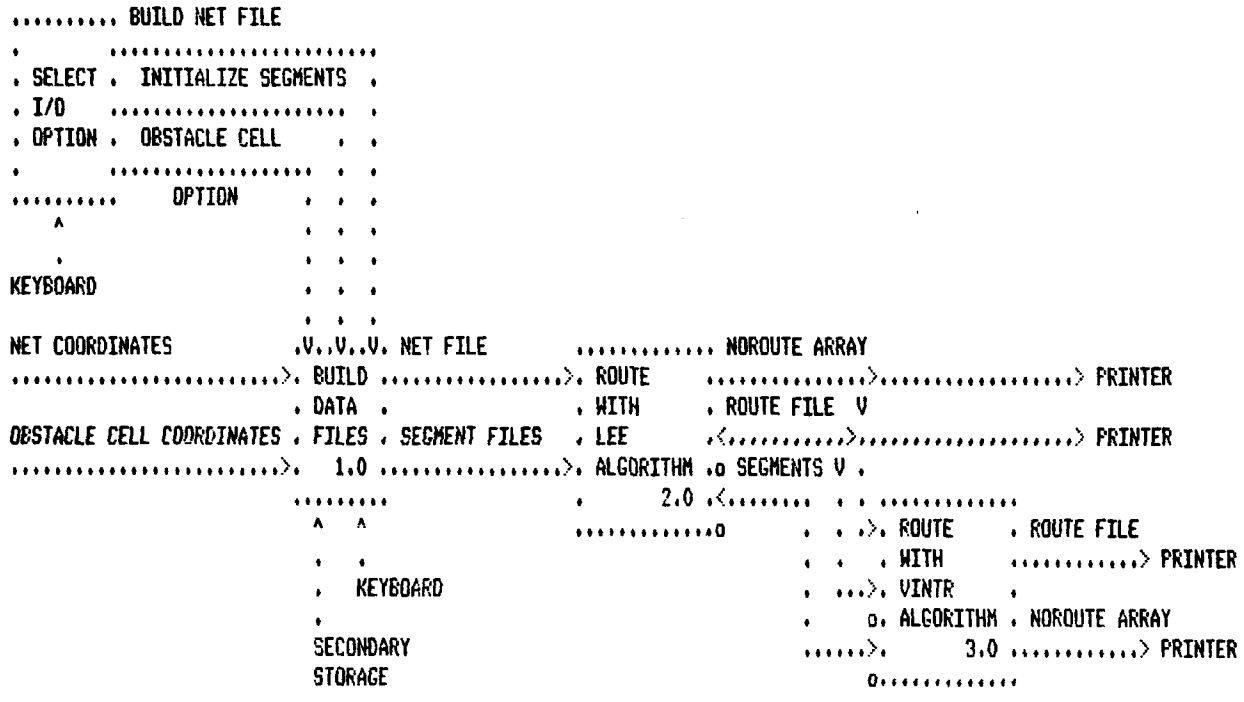


FIGURE 1. Overall System Design - Level 0.

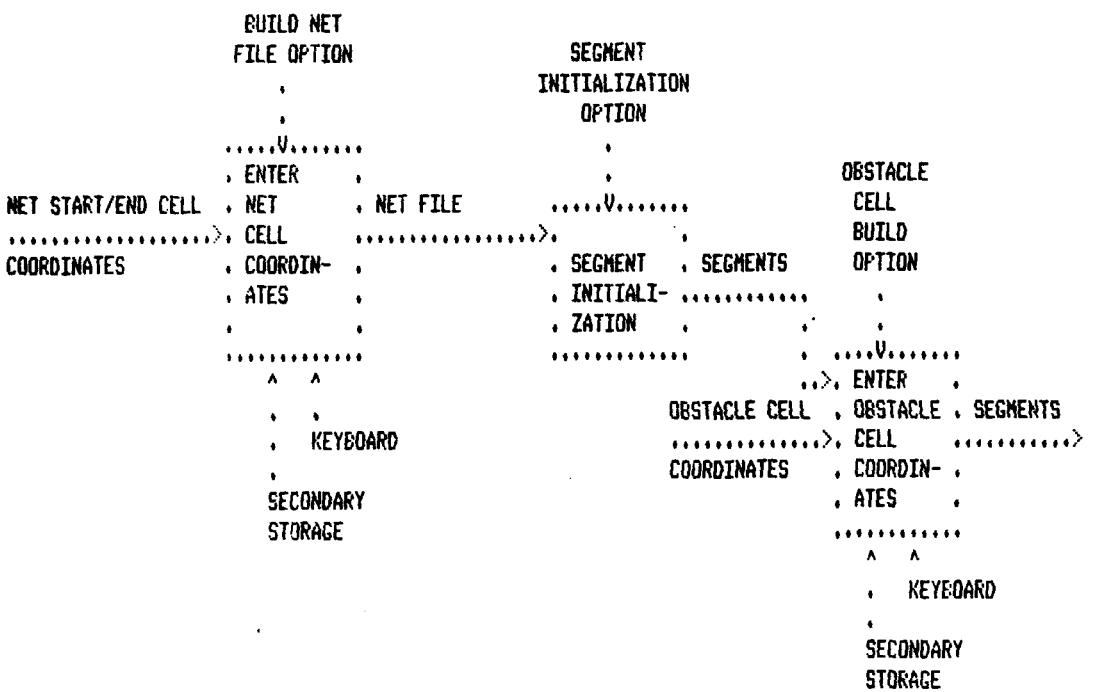


FIGURE 2. Build Data File - Level 1.0

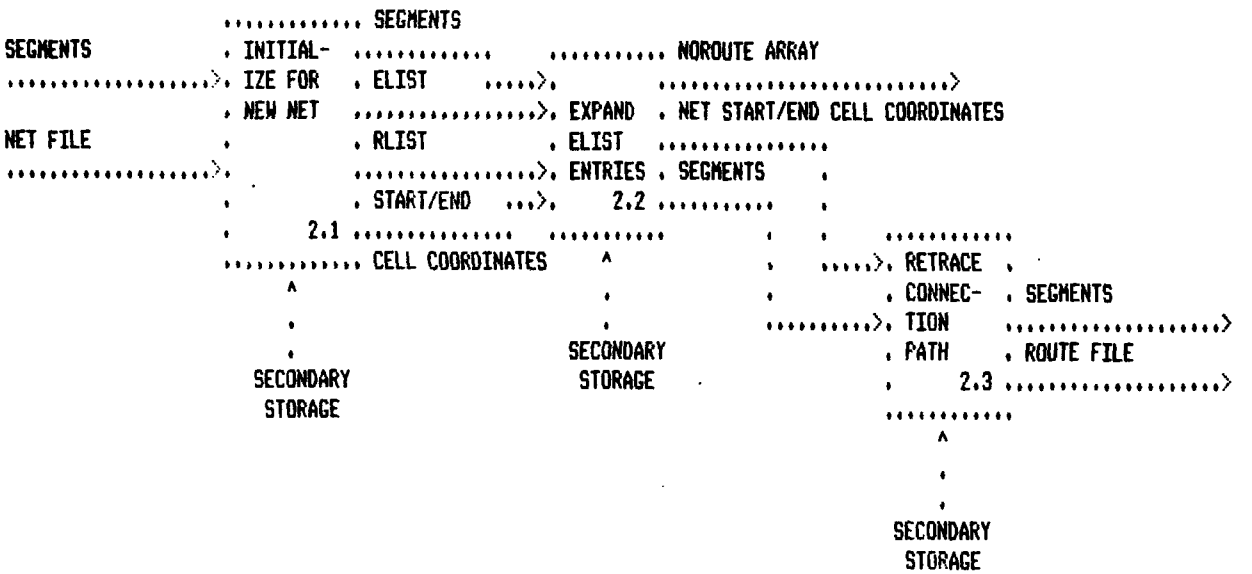


FIGURE 3. Route with Lee Algorithm - Level 2.0.

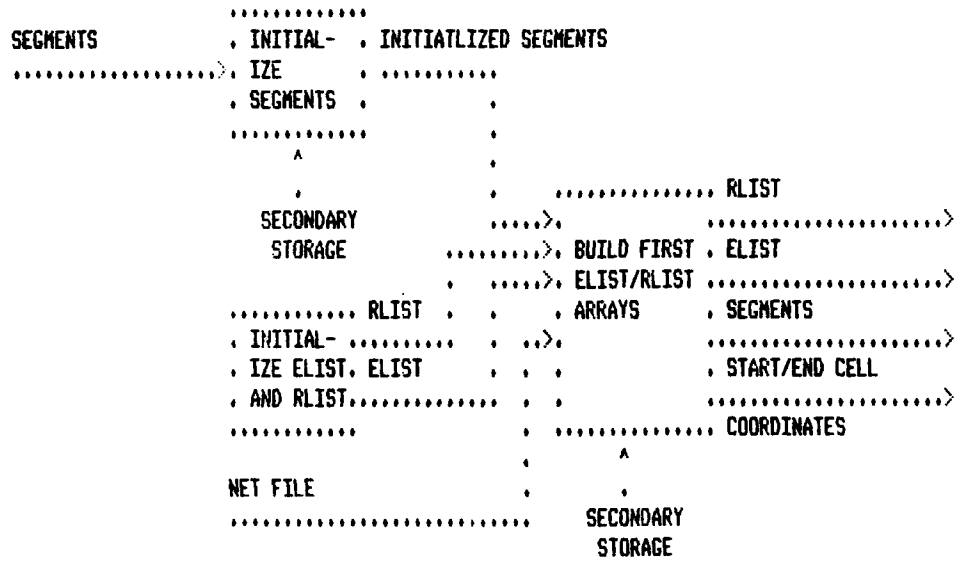


FIGURE 4. Initialization for New Interconnection Entries - Level 2.1.

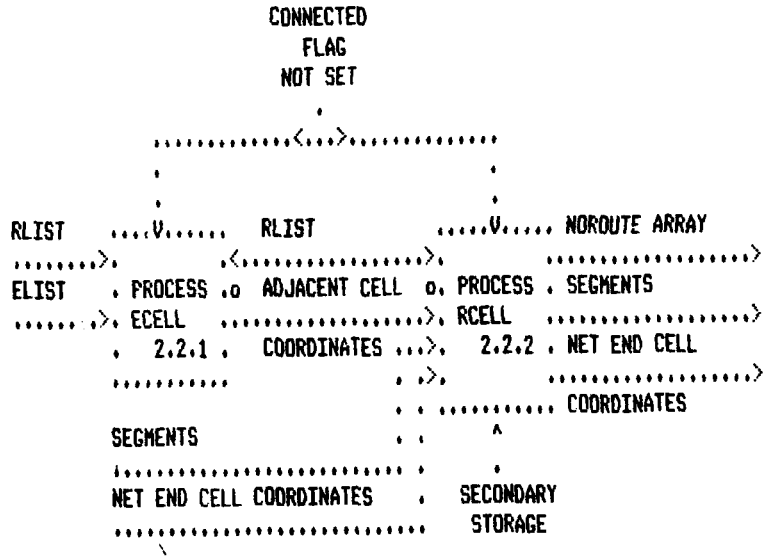


FIGURE 5. Expand ELIST Entries - Level 2.2.

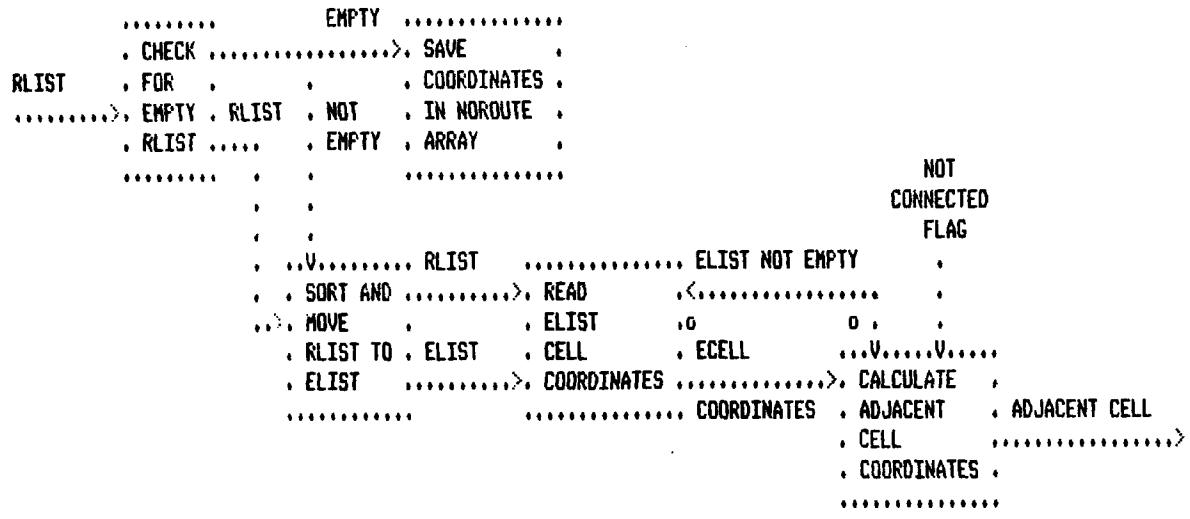


FIGURE 6a. Process ECELL - Level 2.2.1.

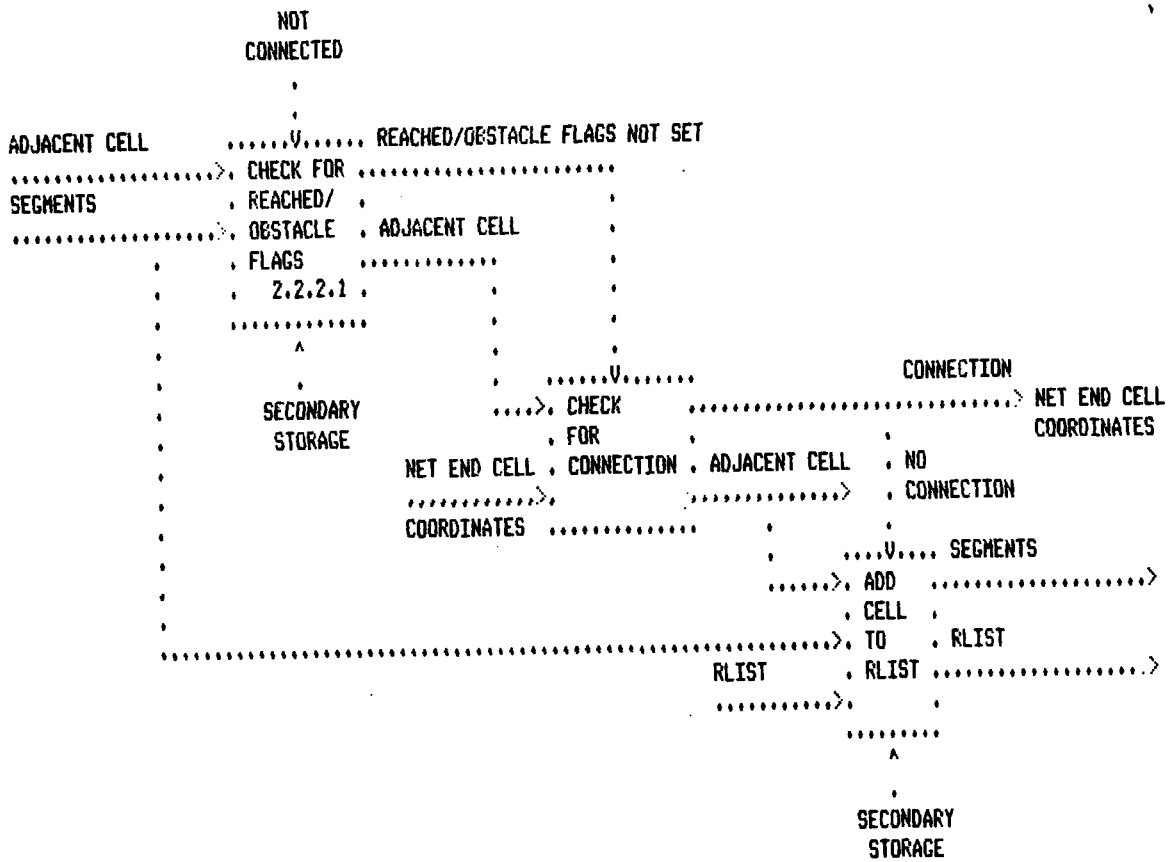


FIGURE 6b. Process RCELL - Level 2.2.2.

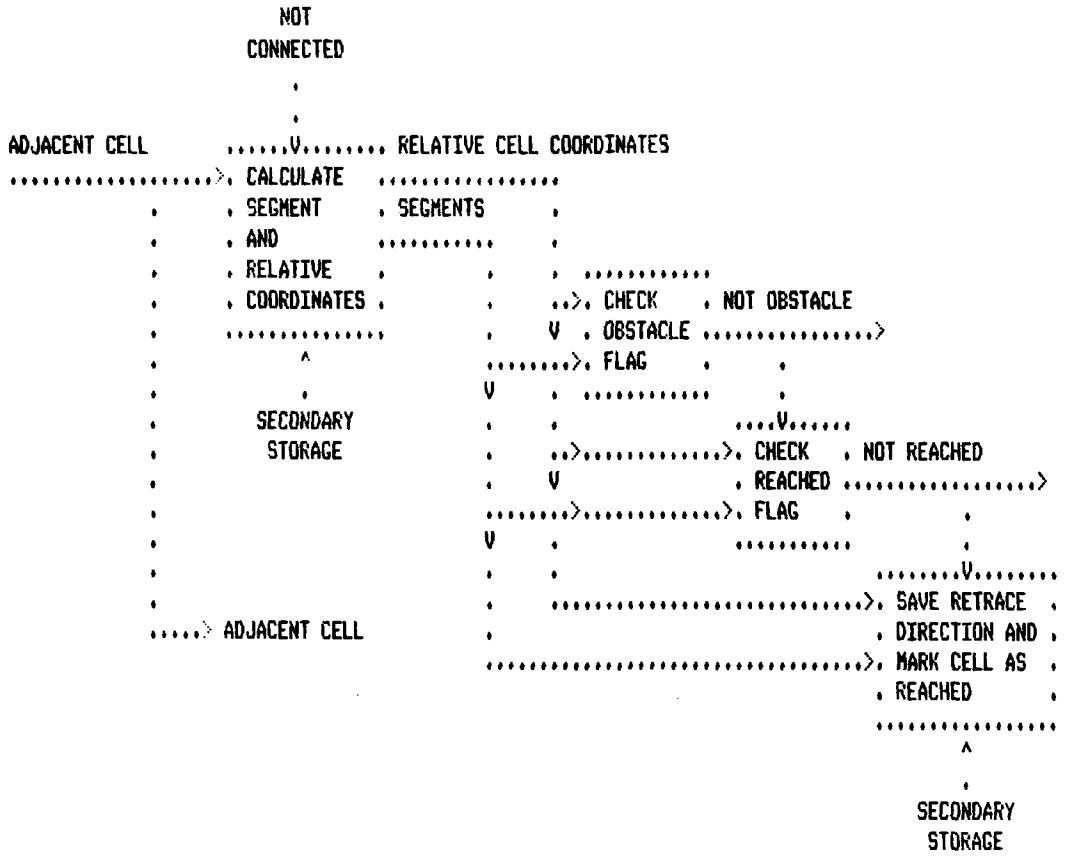


FIGURE 7. Check for Reached, Obstacle Cells - Level 2.2.2.1.

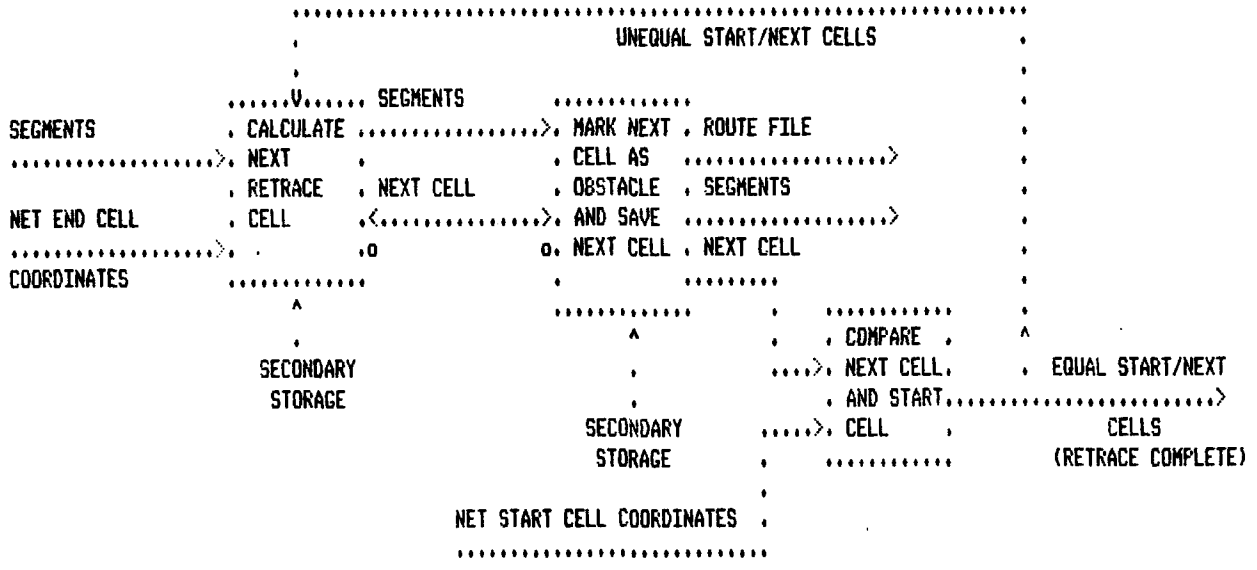


FIGURE 8. Retrace Connected Path - Level 2.3.

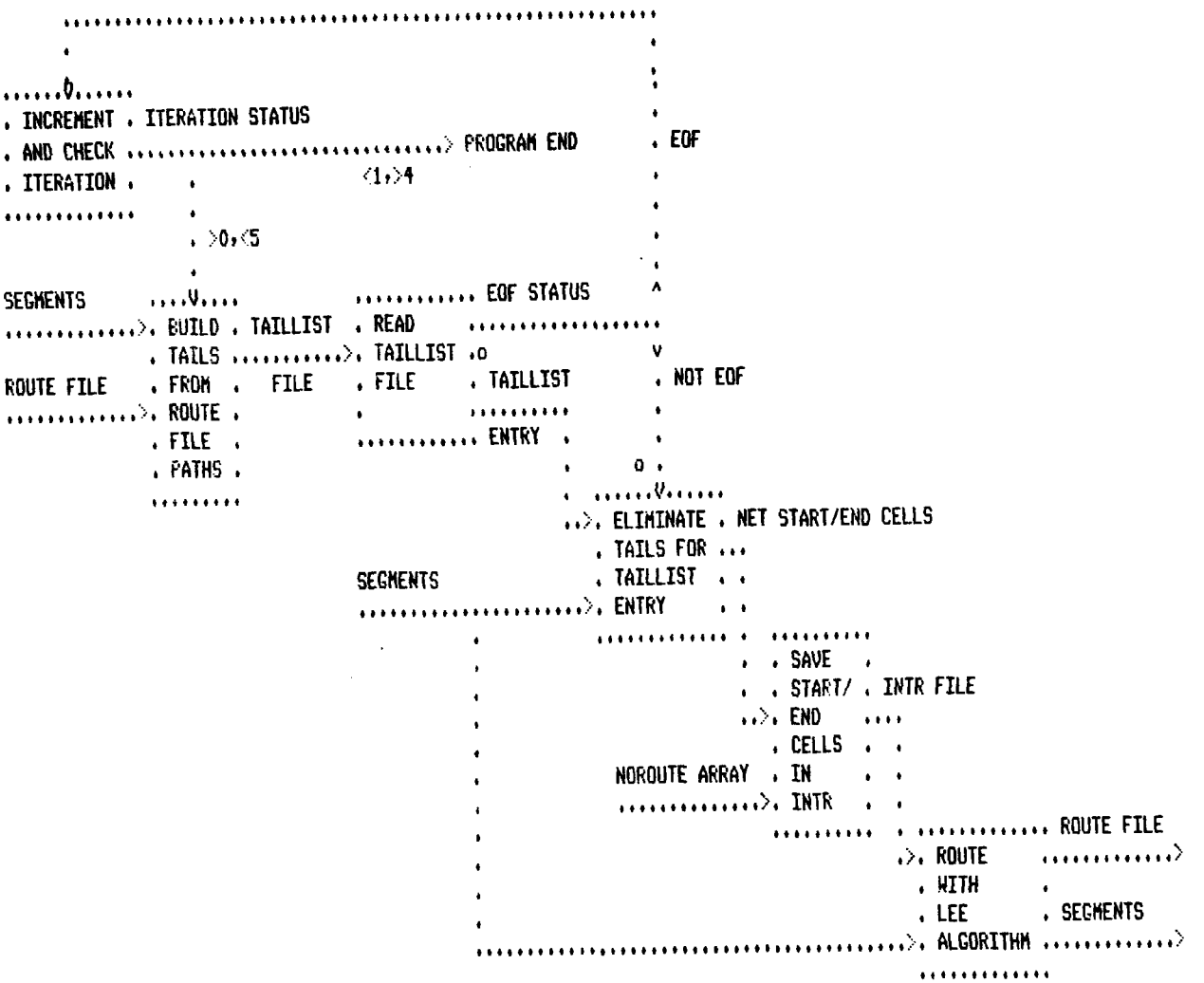
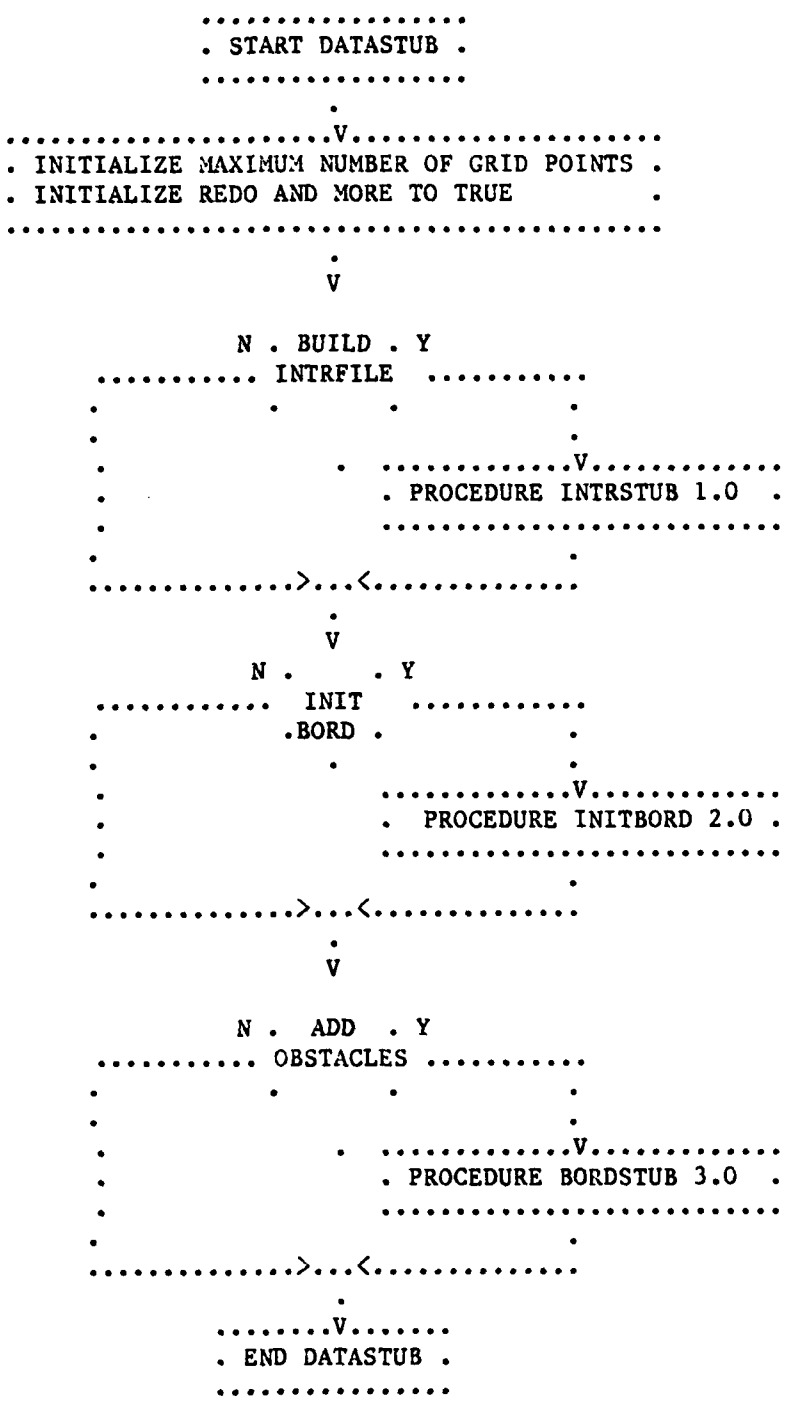
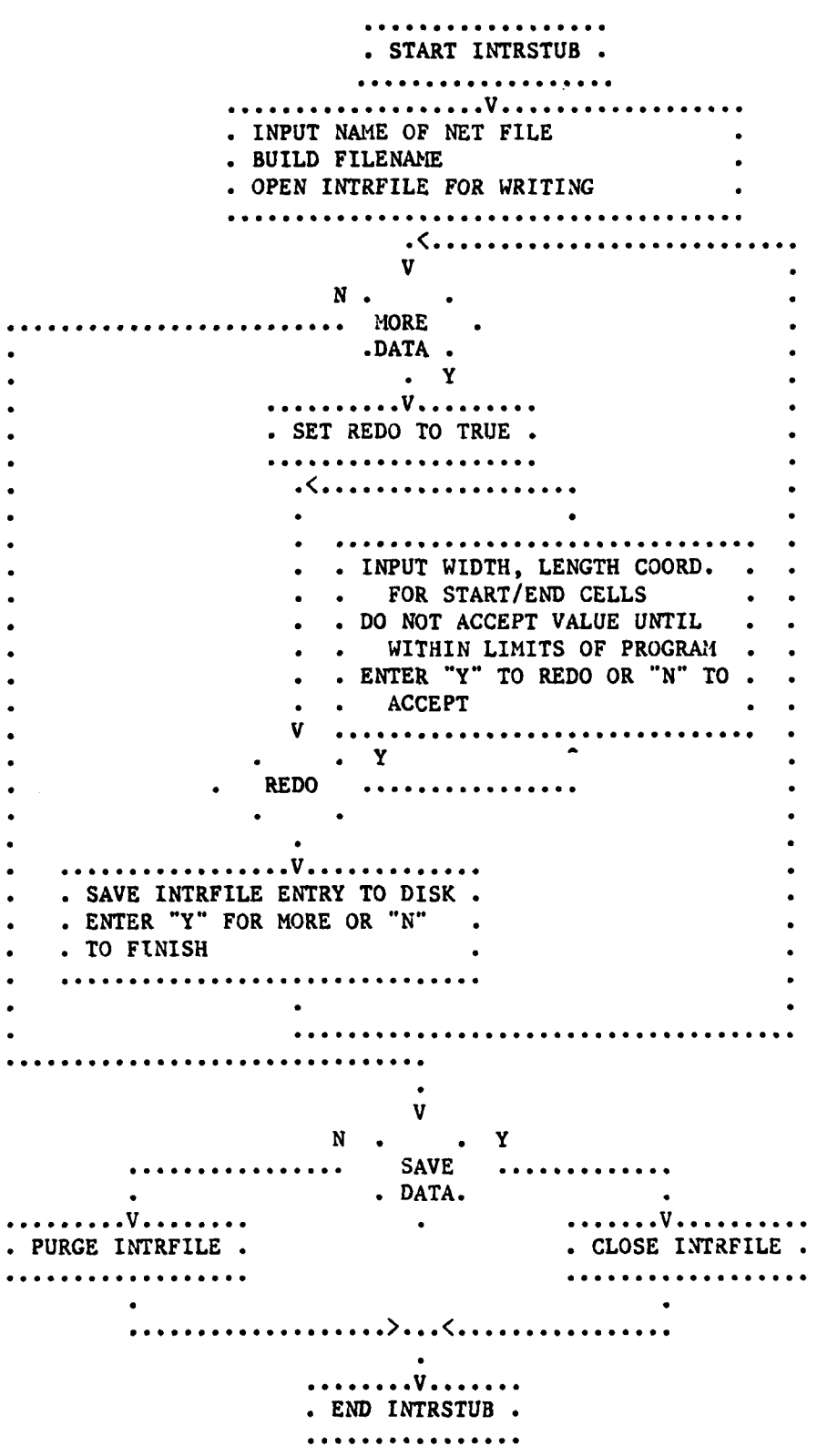


FIGURE 9. Route with Vintr Algorithm - Level 3.0.

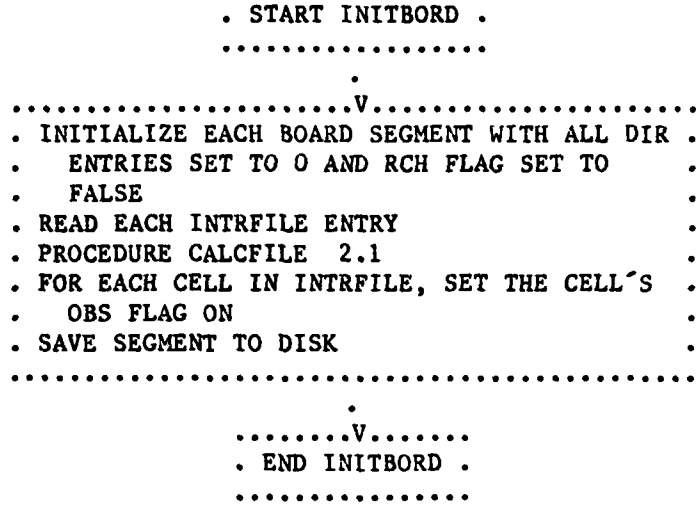
APPENDIX B: Software Implementation - Program DATASTUB



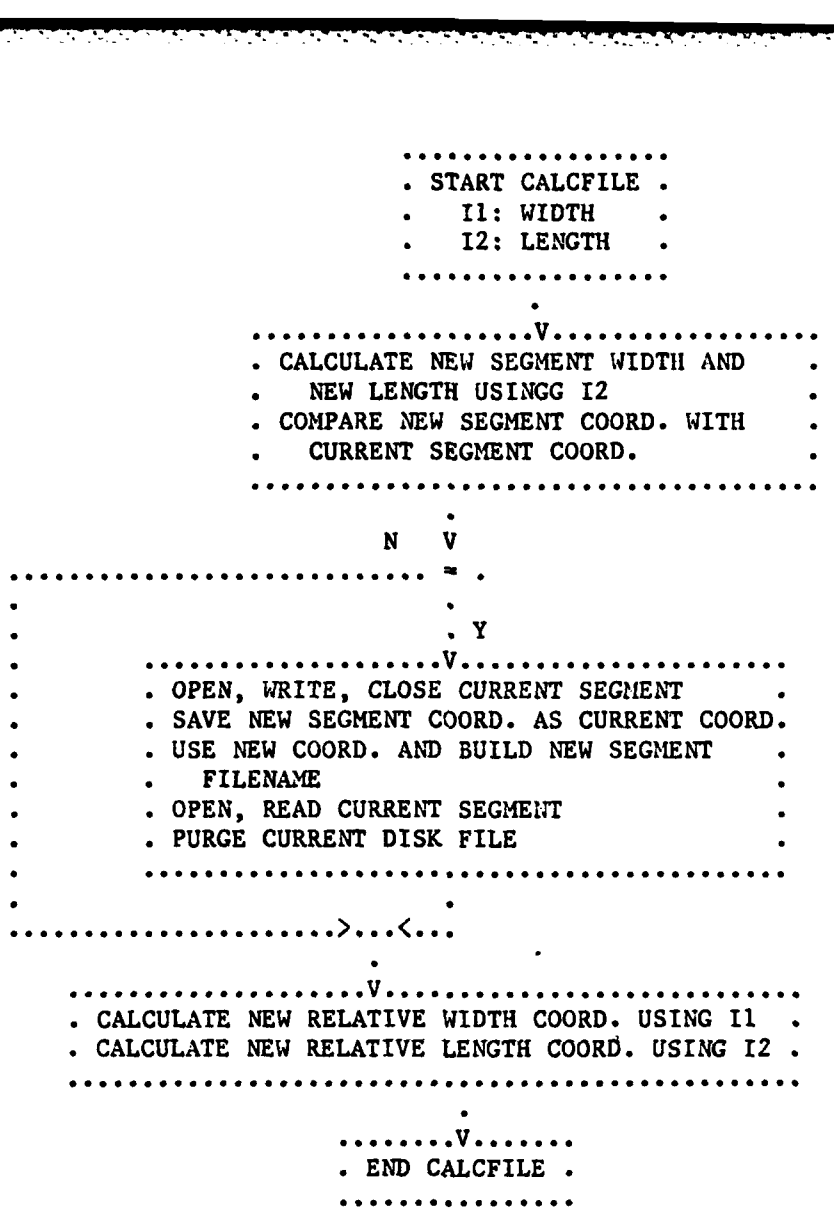
Program DATASTUB.



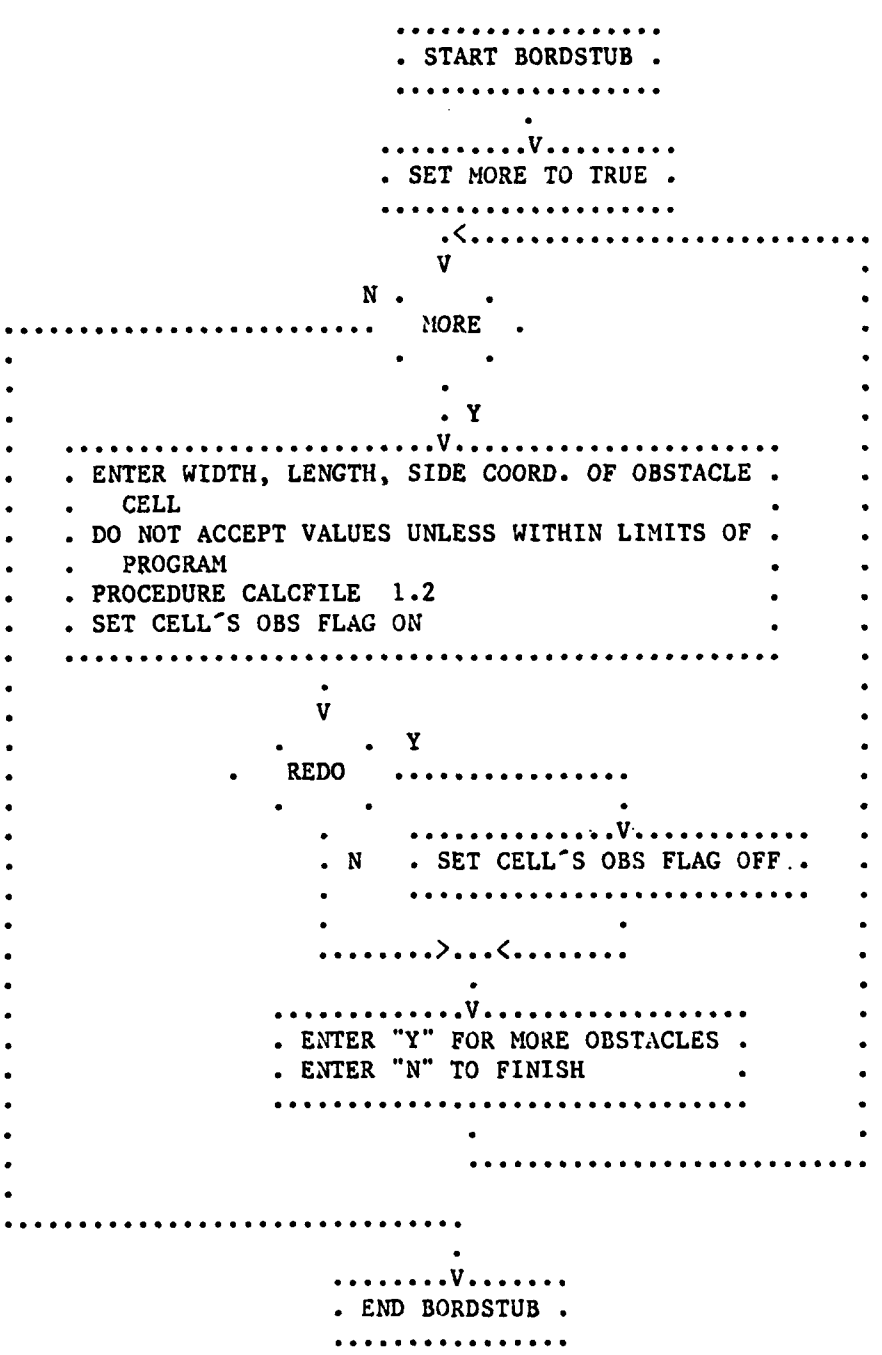
Procedure 1.0 - INTRSTUB



Procedure 2.0 - INITBORD

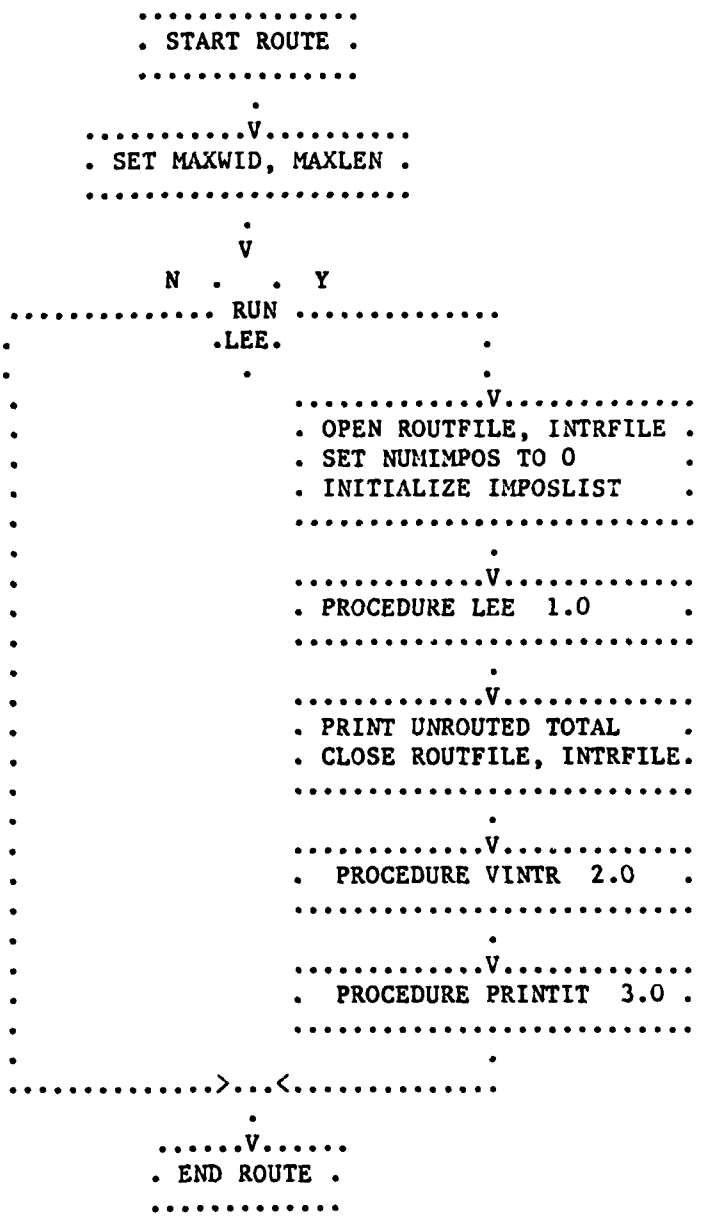


Procedure 2.1 - CALCFILE

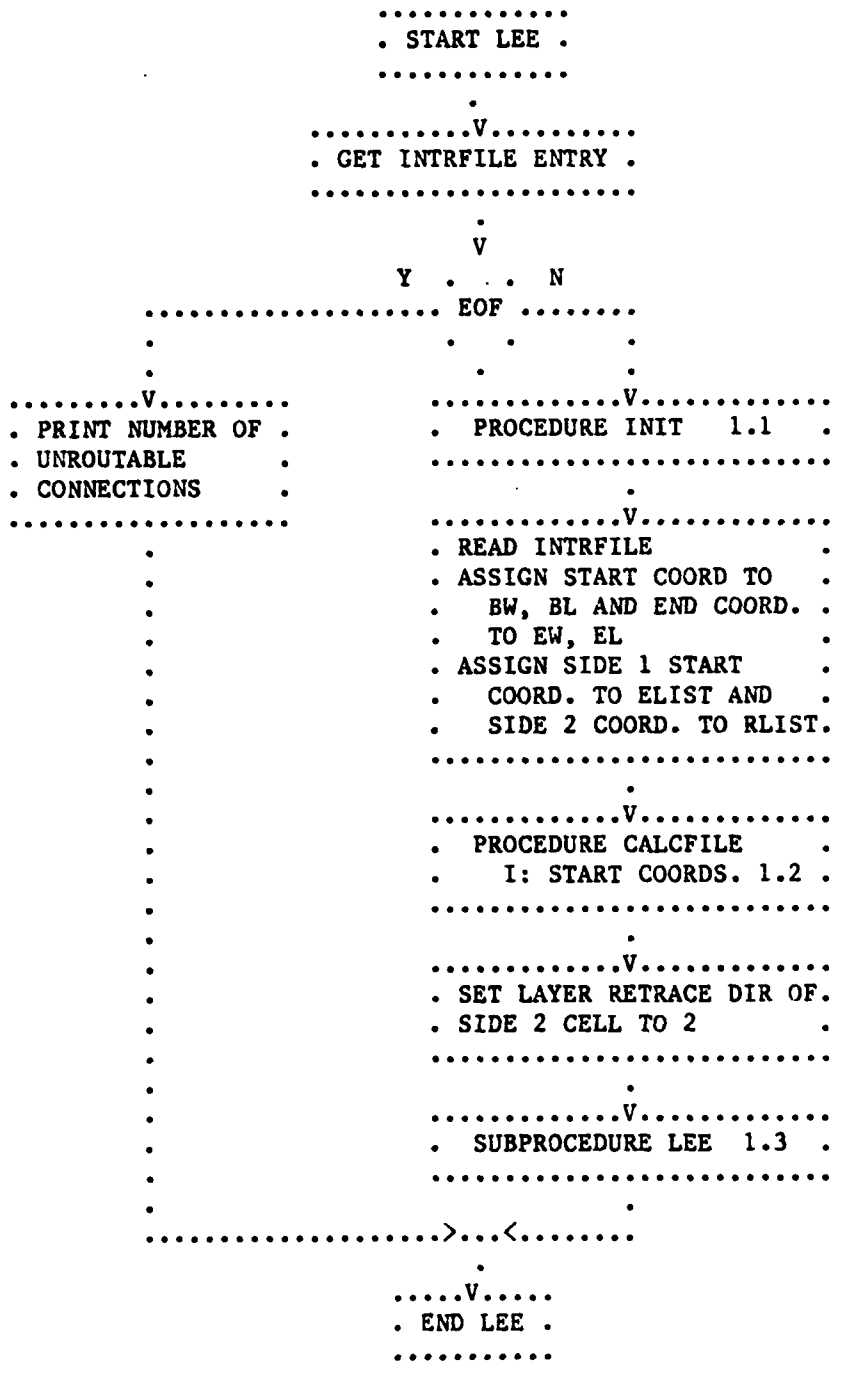


Procedure 3.0 - BORDSTUB

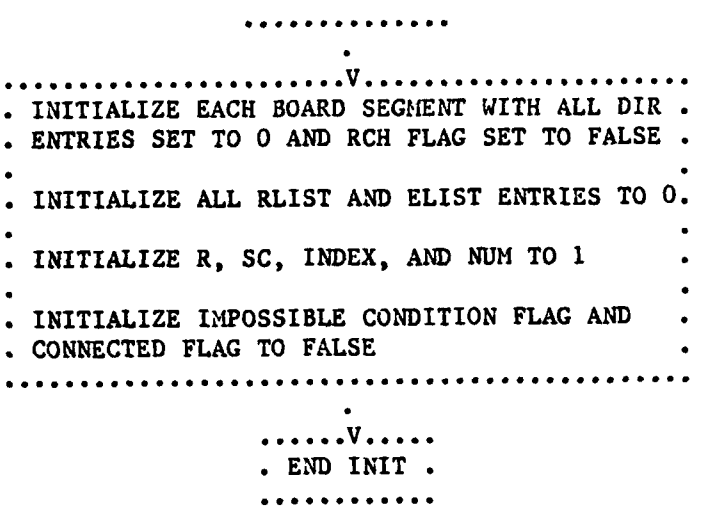
APPENDIX C: Software Implementation - Program ROUTE



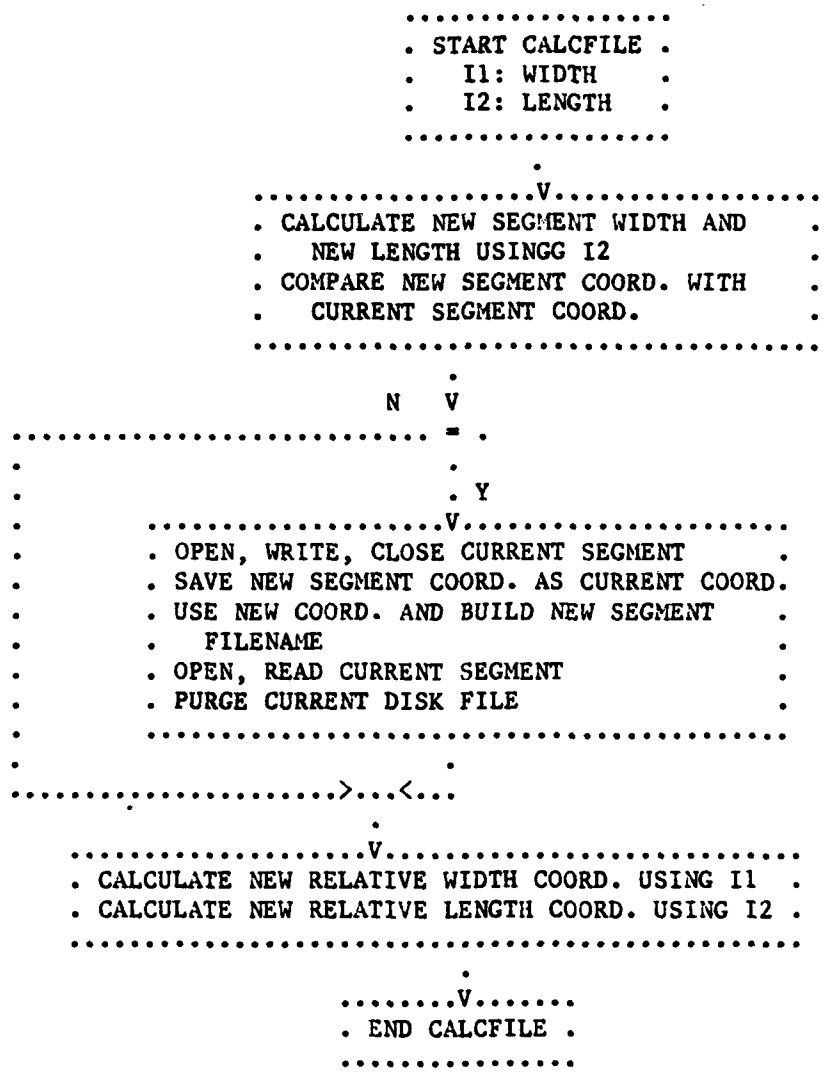
Program Route.



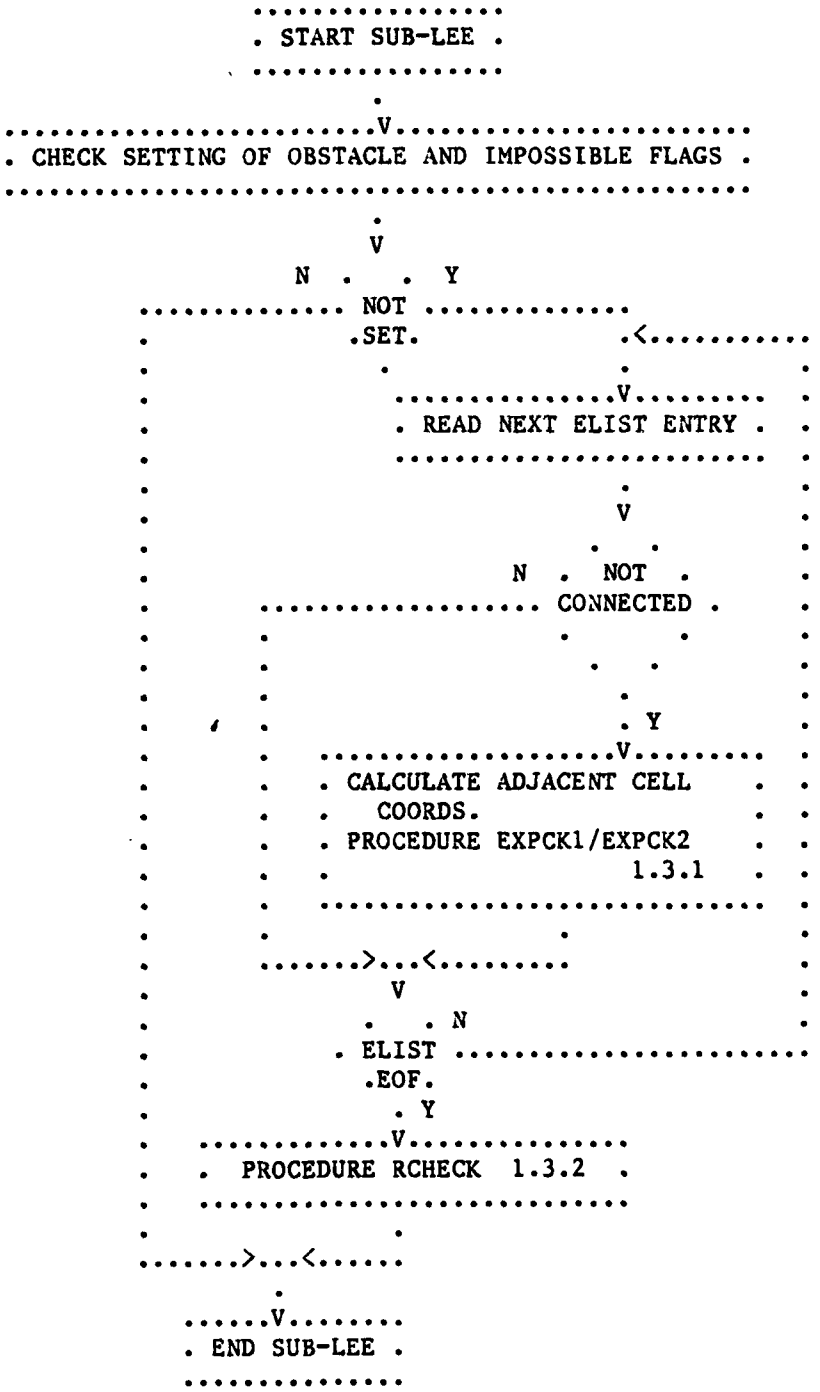
Procedure 1.0 - LEE



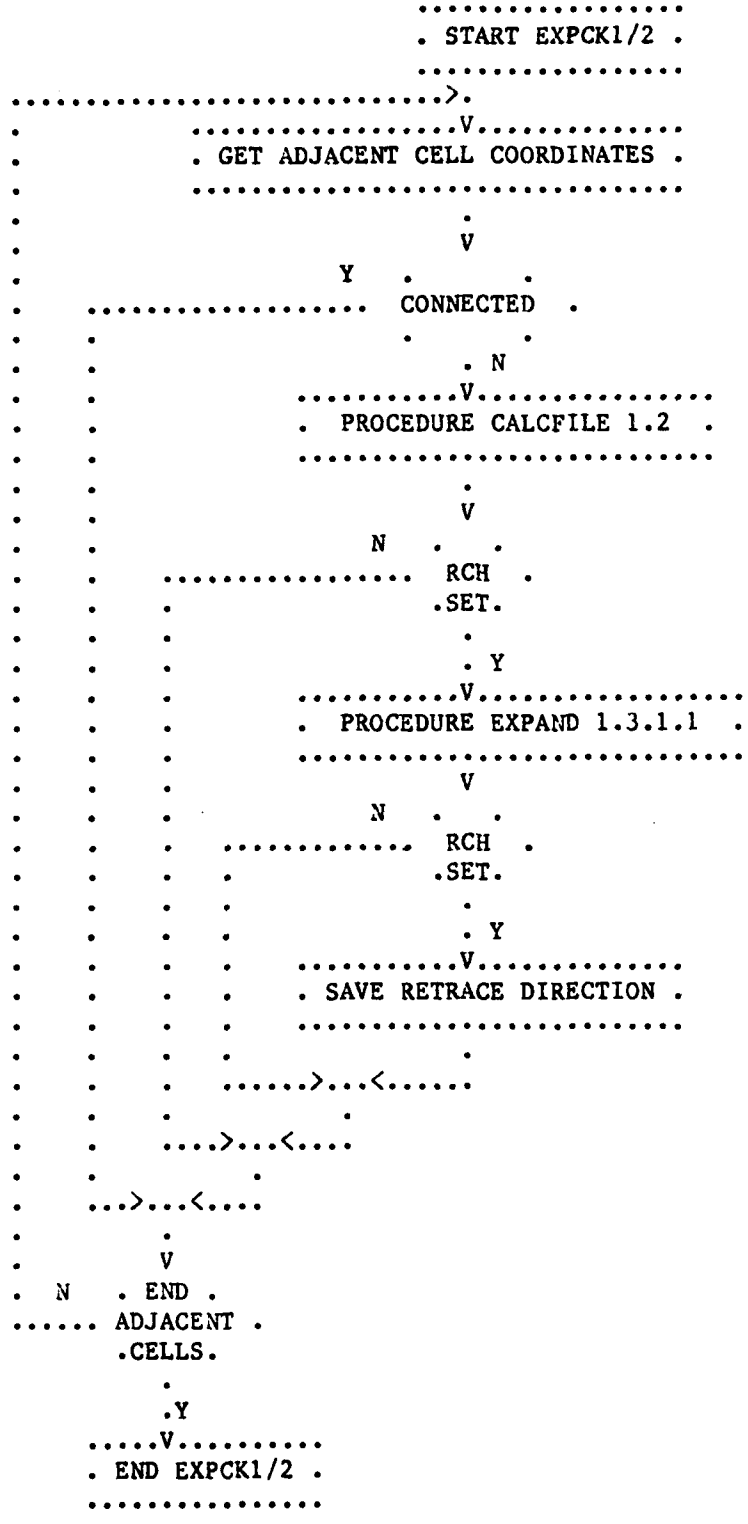
Procedure 1.1 - INIT



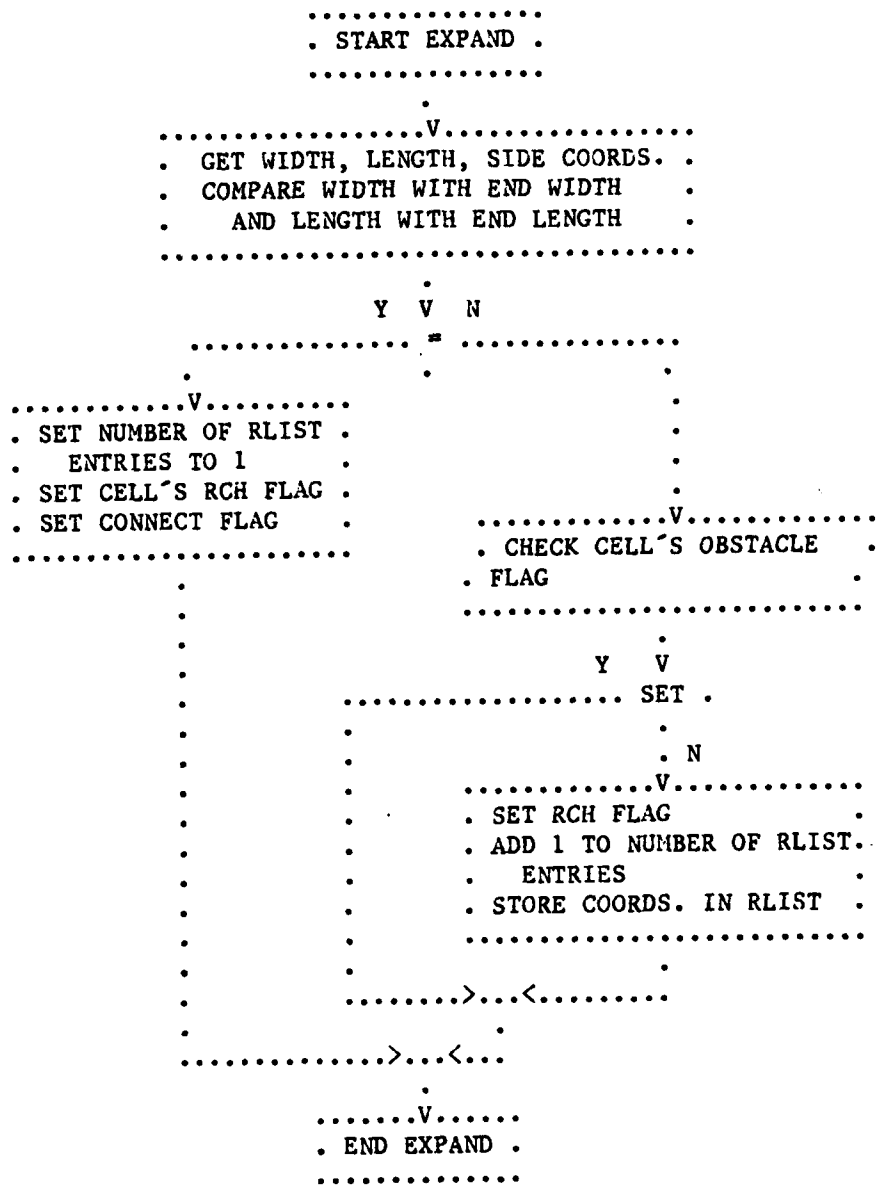
Procedure 1.2 - CALCFILE



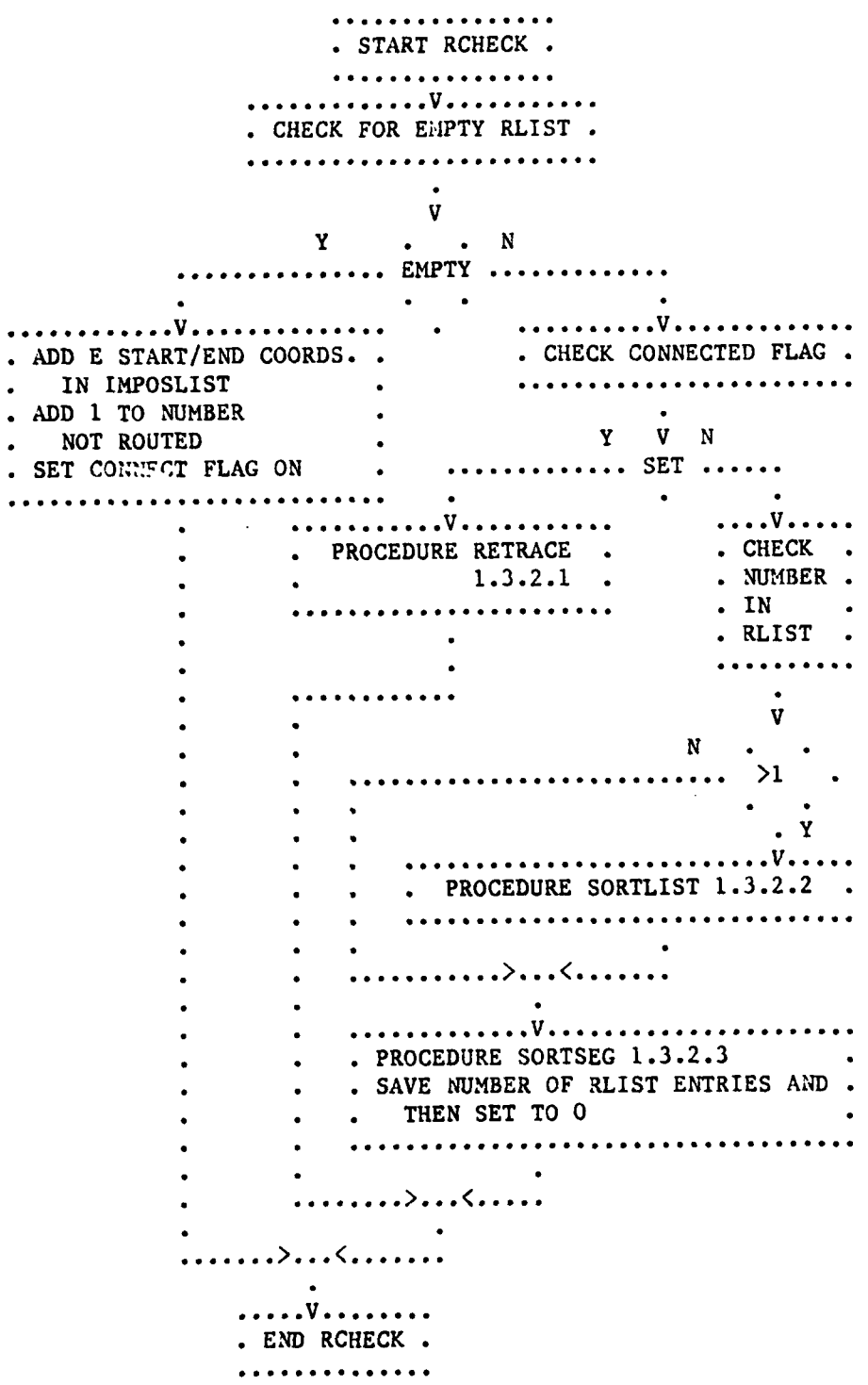
Sub-Procedure 1.3 - LEE



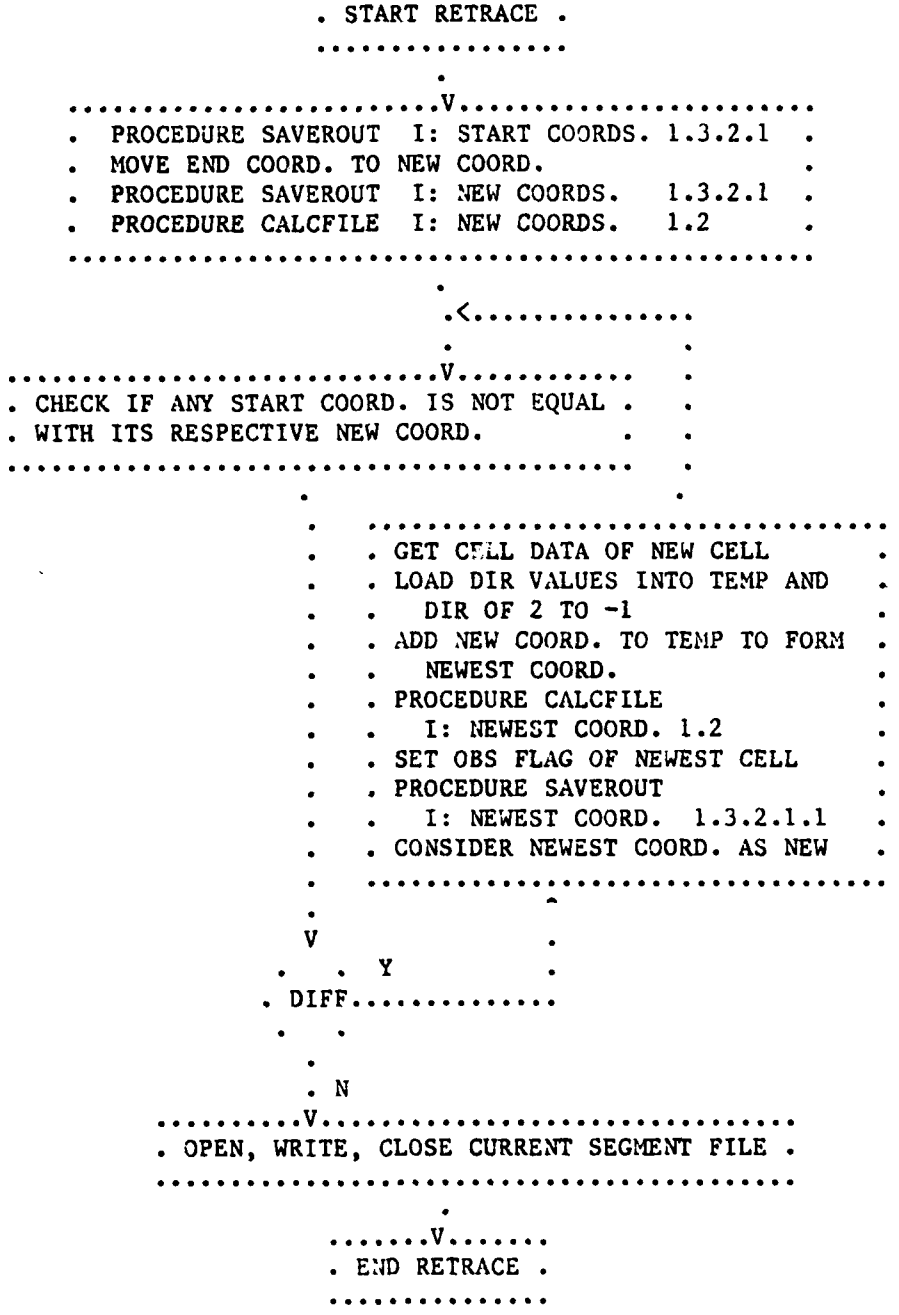
Procedure 1.3.1 - EXPCK1/EXPCK2



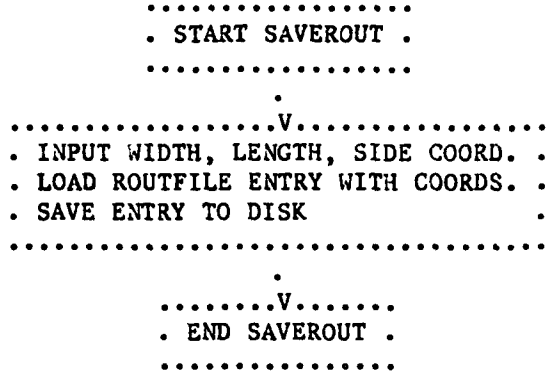
Procedure 1.3.1.1 - EXPAND



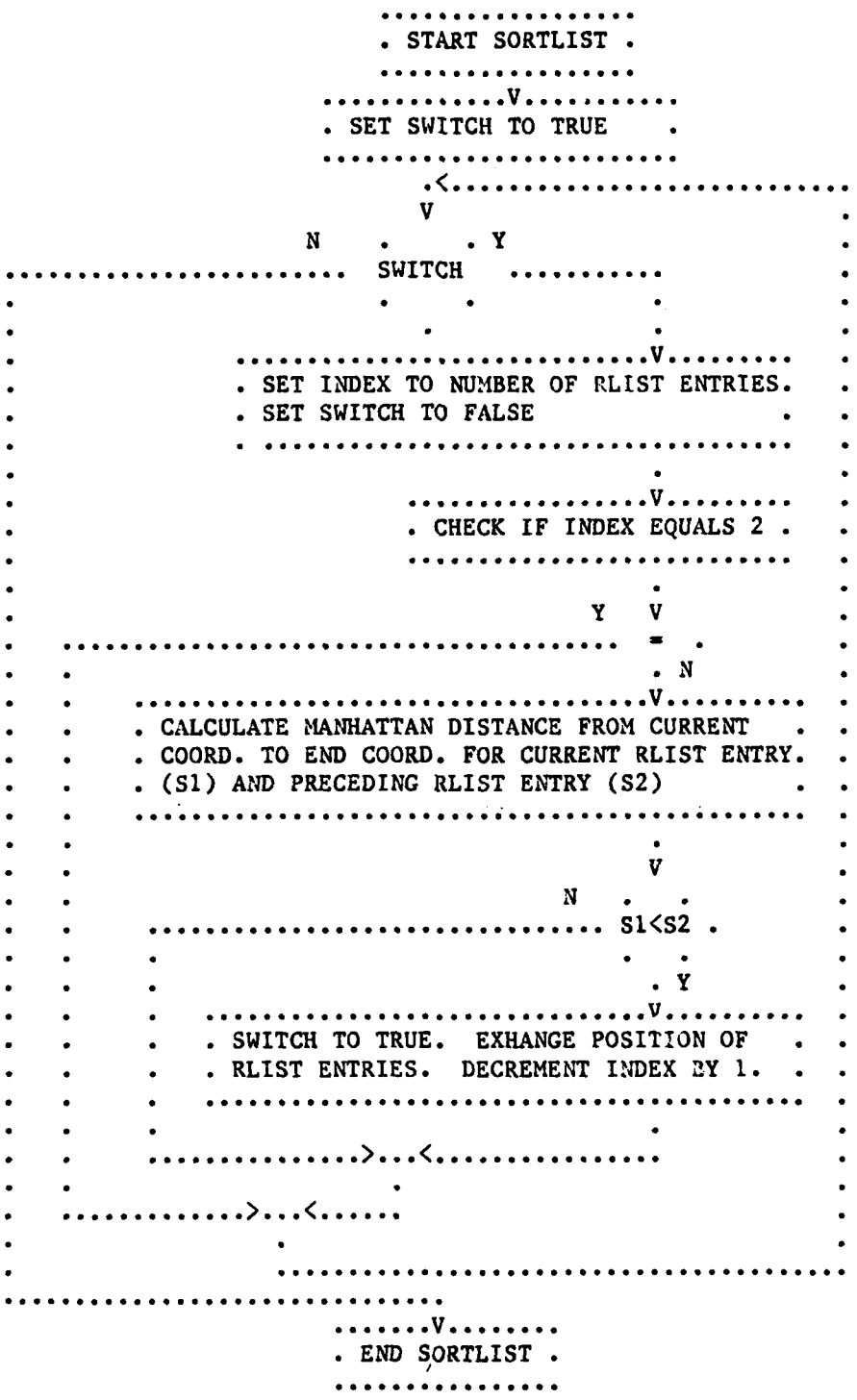
Procedure 1.3.2 - RCHECK



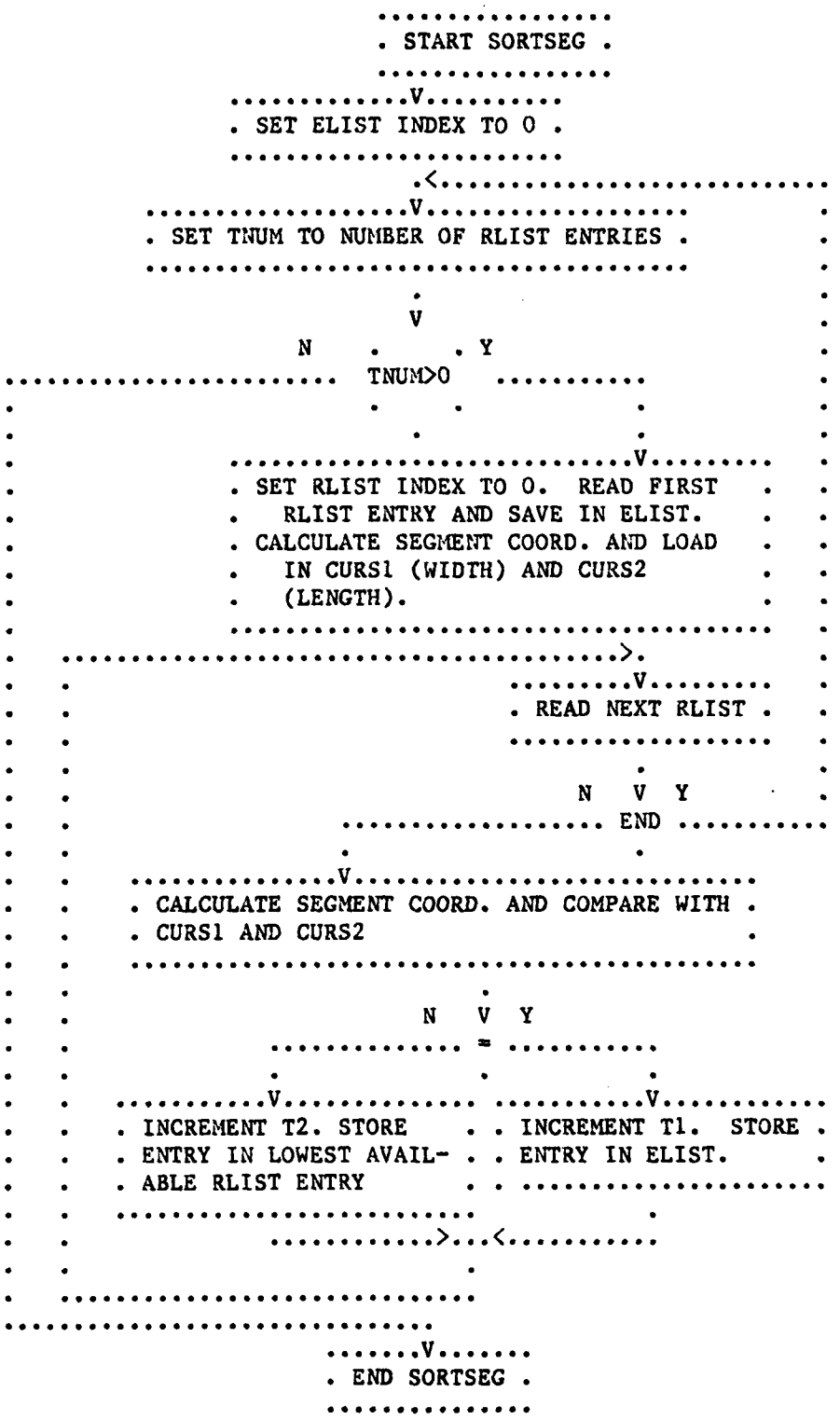
Procedure 1.3.2.1 - RETRACE



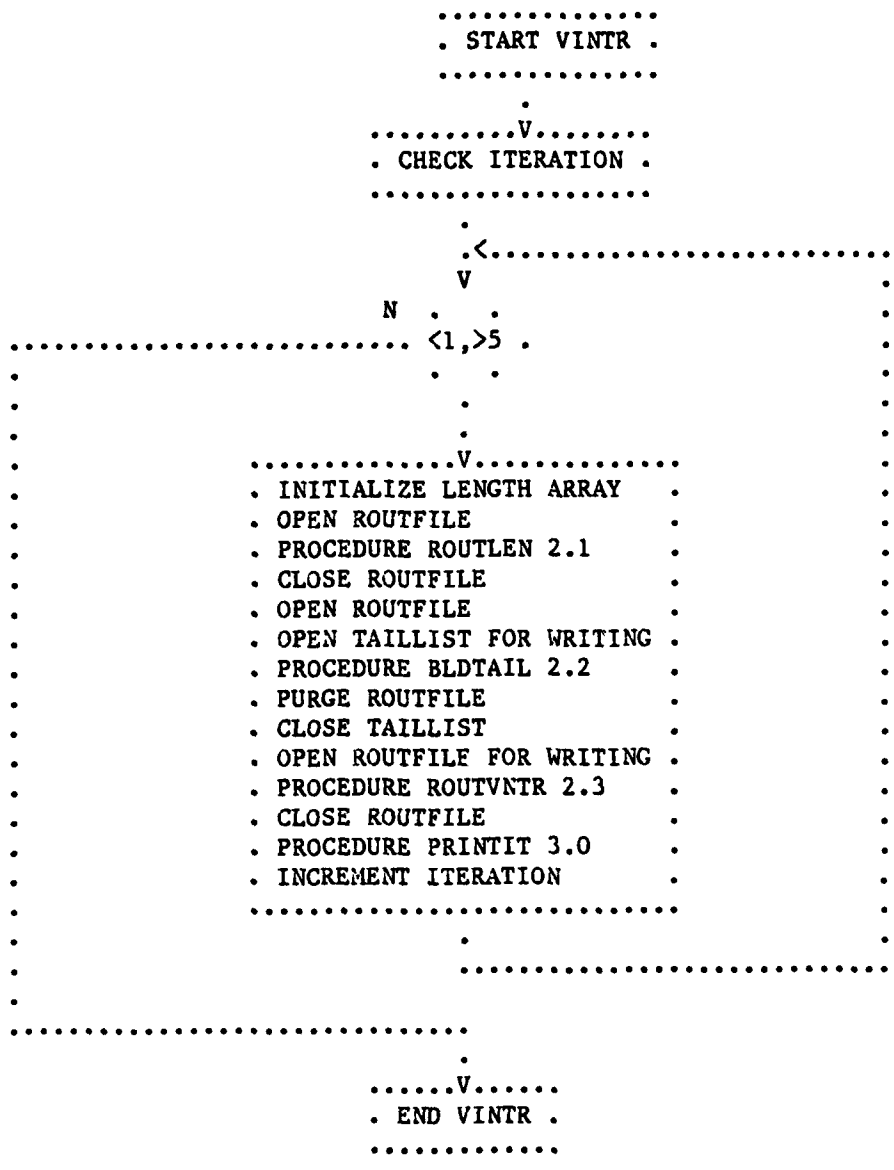
Procedure 1.3.2.1.1 - SAVEROUT



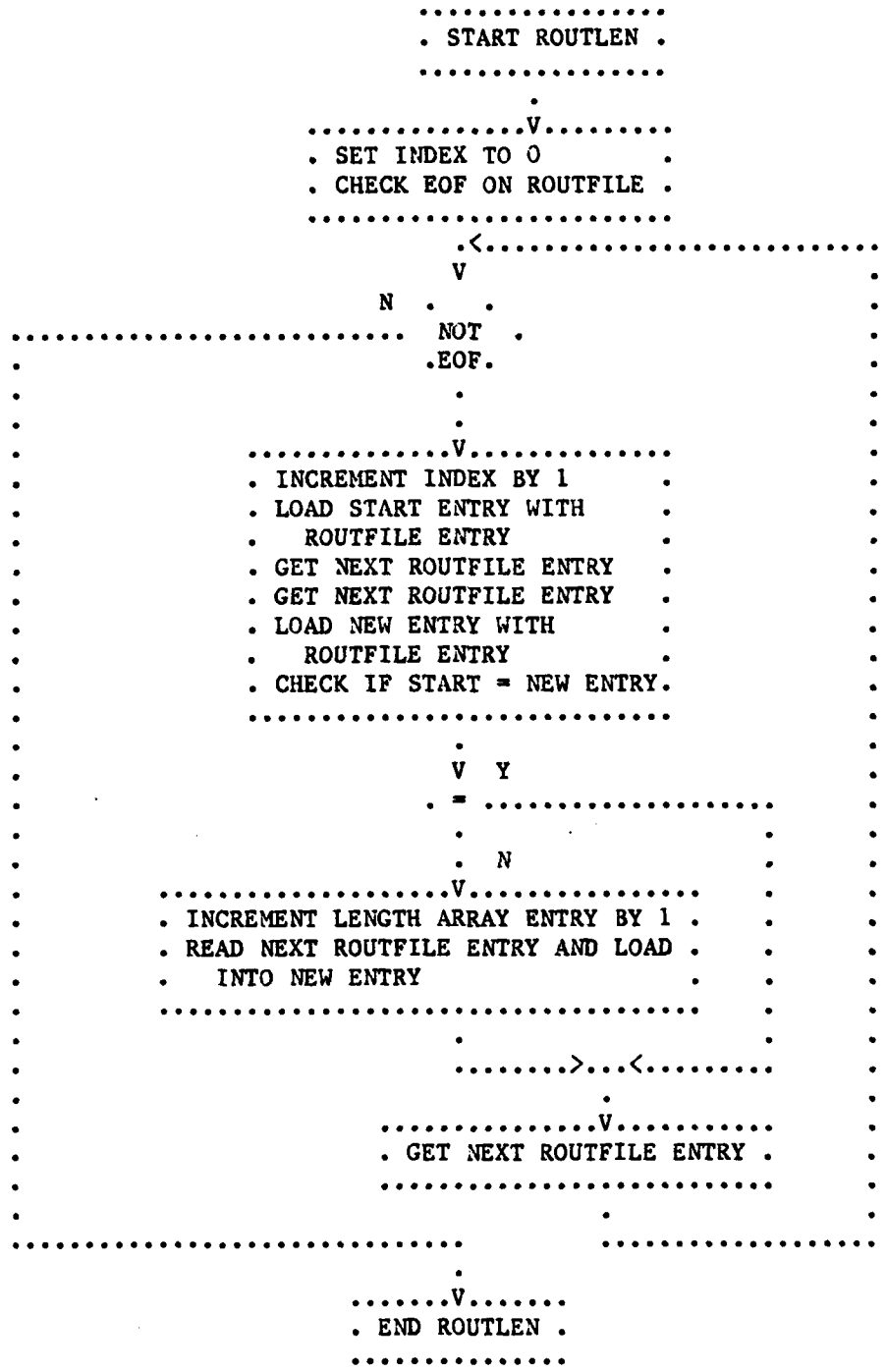
Procedure 1.3.2.2 - SORTLIST



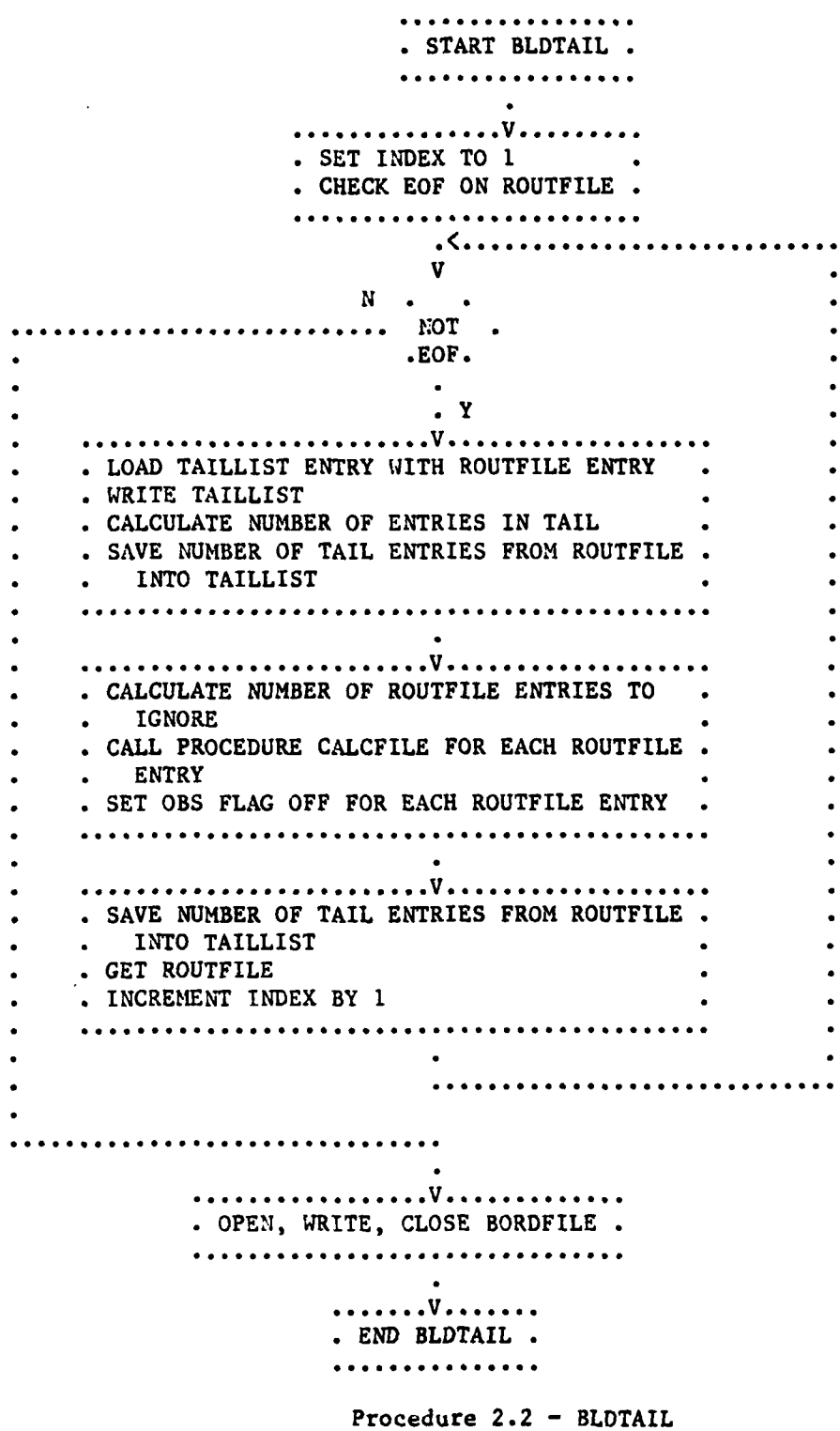
Procedure 1.3.2.3 - SORTSEG



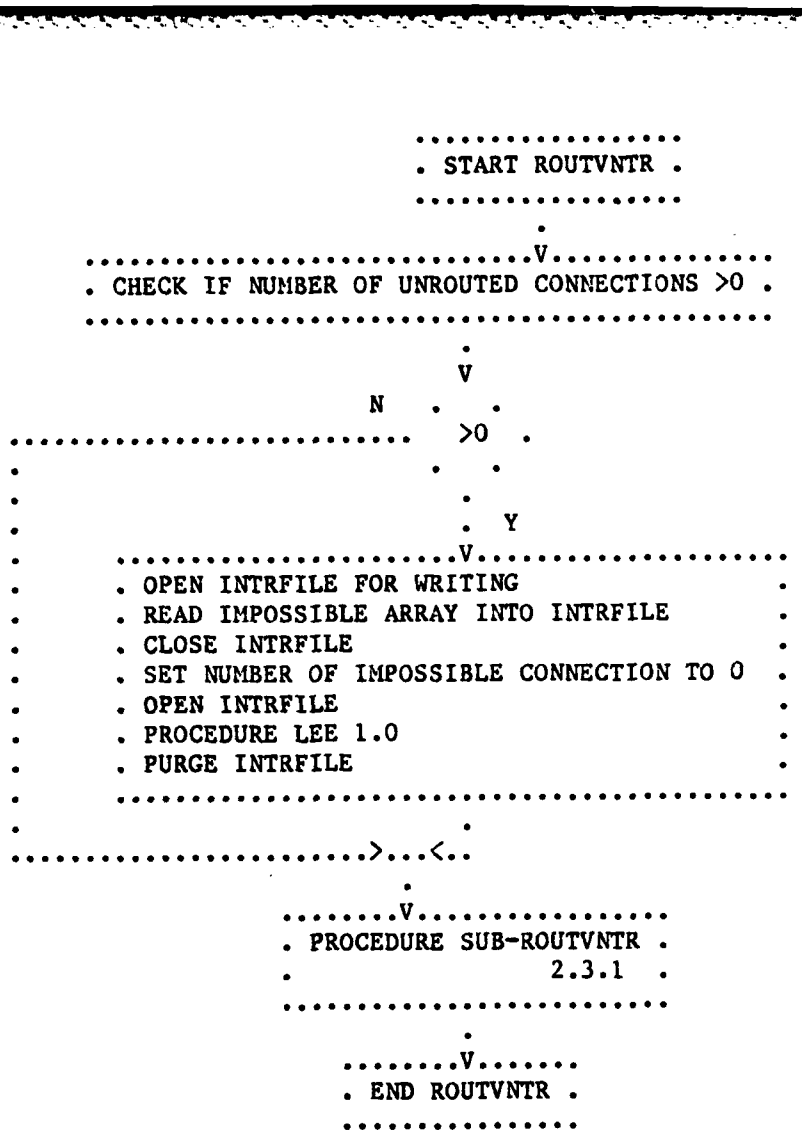
Procedure 2.0 - VINTR



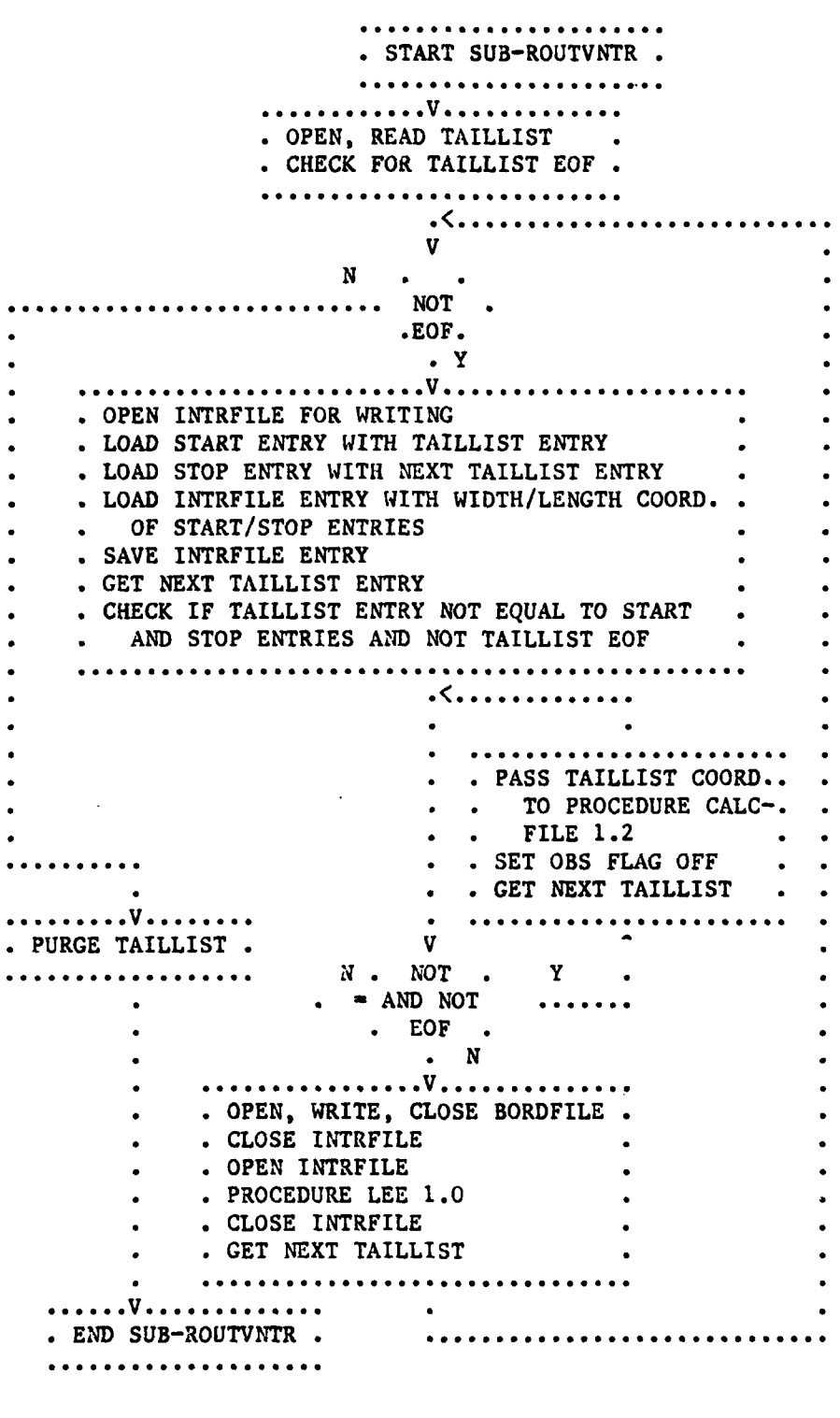
Procedure 2.1 - ROUTLEN



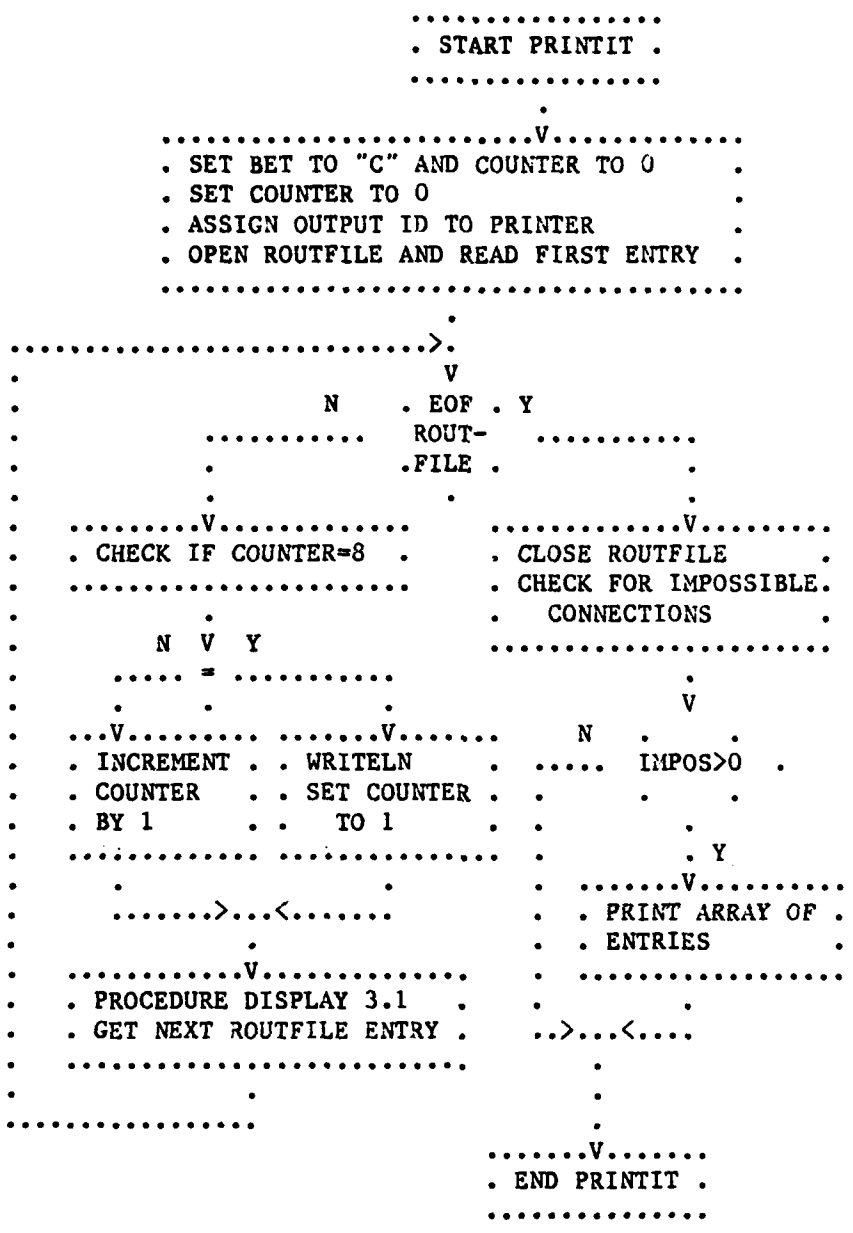
Procedure 2.2 - BLDTAIL



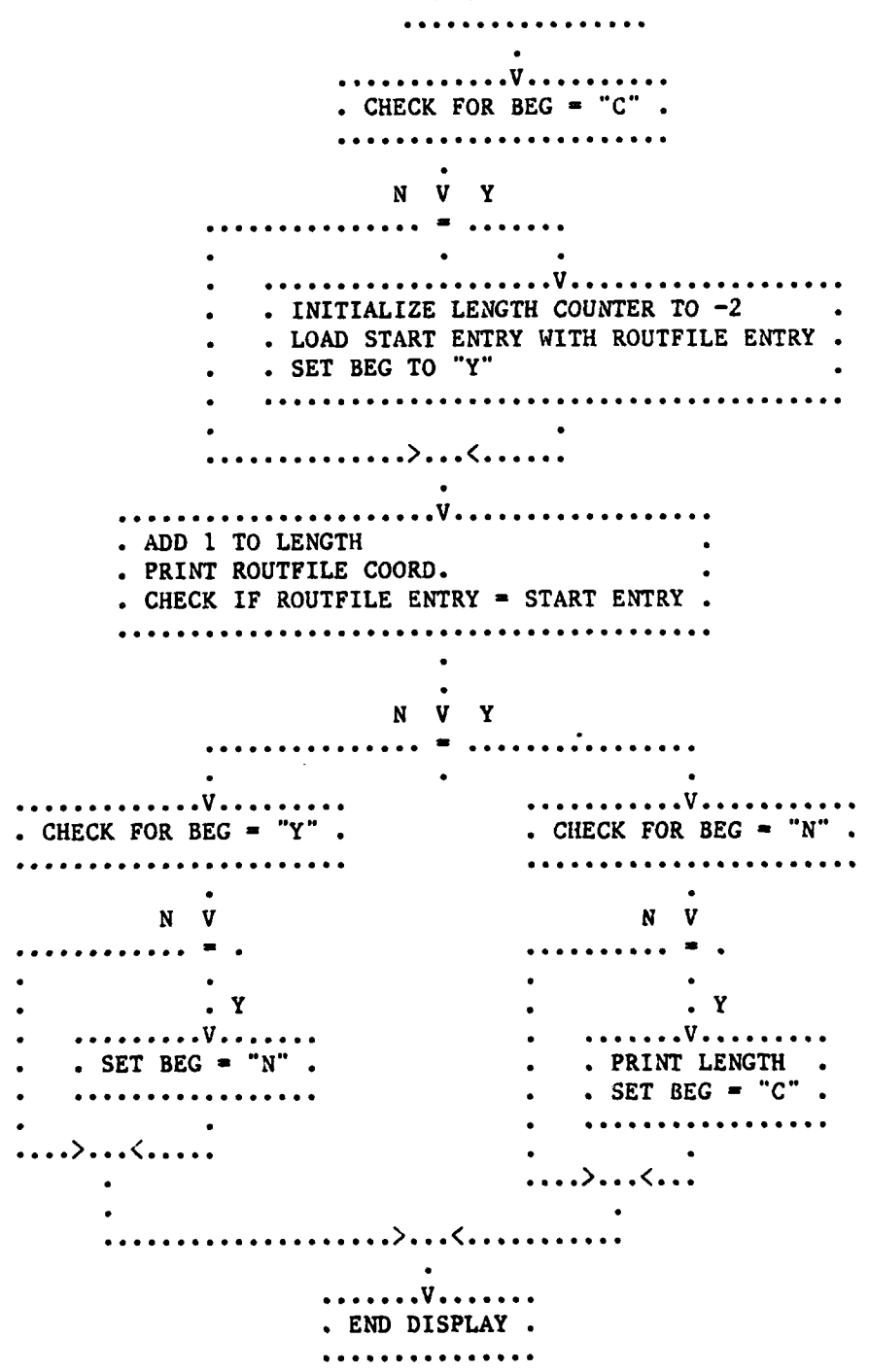
Procedure 2.3 - ROUTVNTR



Procedure 2.3.1 - SUB-ROUTVNTR



Procedure 3.0 - PRINTIT



Procedure 3.1 - DISPLAY

APPENDIX D: Program DATASTUB

Program

Program DATASTUB allows input of cell coordinates and definition of additional obstacle cells. When the user is finished with entering data, it can be saved to floppy diskette and the board initialized.

Inputs - (1) an interconnection file (INTRFILE) with start and end width/length coordinates.
(2) individual board segments (GRIDDATA) as required.

Outputs - (1) updated individual board segments (GRIDDATA).

Constants

MAXSEGWD, MAXSEGLN - these values define the number of segments along the width and length of the board, respectively. Together, they define the overall board dimensions. To increase the board dimensions, merely increase these values.

Array Types

INTRDATA - a 4X1 packed array storing integer values of 0..60. The values represent coordinate values. Element 1 is the start width, element 2 is the start length, element 3 is the stop width, and element 4 is the stop length.

GRIDDATA - a 30X30X2 packed array of RECDATA. Each specific element of the array corresponds to a specific position in a single board segment.

Record Type

RECDATA - each cell is represented by a single record. Each record has two flags and a 3X1 packed array storing integer values of 0..2. The first flag is OBS and designates whether or not the cell is an obstacle. The second flag is RCH and designates whether or not the cell has been previously reached during the current cell expansion phase. The 3X1 array (DIR) represents the integer values required for retrace once the end cell has been reached. The first element of DIR is the width retrace direction, the second element of DIR is the length retrace direction, and the third element of DIR is the side retrace direction. A value of 0 signifies no change, 1 signifies a +1 change, and 2 signifies a -1 change in the present coordinate position. These values are added to the current coordinates to achieve the next retrace cell. Note that only one direction parameter should ever be non-zero.

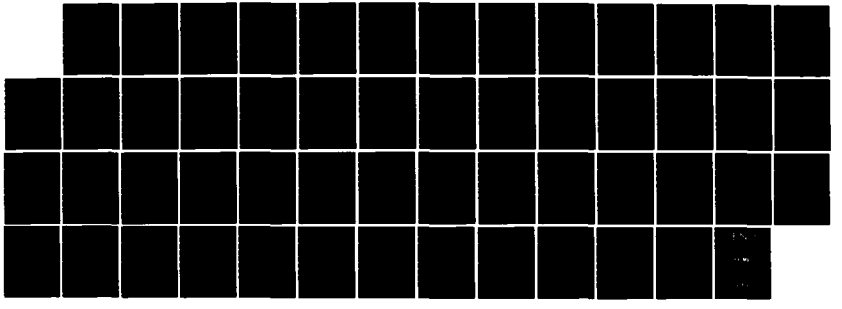
Variables

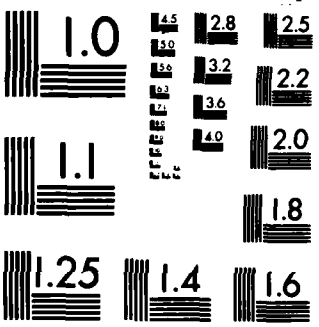
INTRFILE - a secondary storage file of INTRDATA. Entries are the width/length coordinates of the net's end cells to be connected.
INTR - the coordinates of a net's end cells.
BORDFILE - a secondary storage file of GRIDDATA. Each file is a 30X30X2 array of cells and is termed a board segment.
GRID - an individual entry in GRIDDATA.
SEGWD, SEGLN, CURSEGWD, CURSEGLN - these are integer values that keep track of the current segment loaded into memory (CURSEGWD and CURSEGLN) and the new segment (SEGWD and SEGLN) containing the cell currently under consideration. A suffix of WD designates a width coordinate. A suffix of LN designates a length coordinate.
NEWWD, NEWLN - these integer values are the relative coordinates of the current cell within a board segment.
NUM - a general purpose integer variable to store keyboard numerical input in INTRSTUB.
MAXWID, MAXLEN - these are integer values representing the maximum board dimensions.
ANSWER - a character value read from the keyboard in response to a question.
SEGCHR1, SEGCHR2 - the string equivalent of SEGWD and SEGLN, respectively.
CURSEGFILE - the name of the current secondary storage file loaded into memory. Other literal characters are concatenated with SEGCHR1 and SEGCHR2 to uniquely define the individual board segments.
I, J, K - integer value input representing the width, length, and side coordinates of cells in BORDSTUB and INITBORD and as an index in INITBORD.
NAME, FILENAME - NAME is the unique name a user can provide if building multiple INTRFILEs. FILENAME concatenates additional information to NAME and uses FILENAME to refer to the file.
REDO, MORE - boolean flags to indicate whether a value requires changing (REDO) or if there is more input data (MORE).

PROGRAM DATASTUB;
Function - to allow the user to build datafiles, specify obstacle cells, and initialize the segments.
Global parameters modified - MAXWID, MAXLEN, REDO, MORE.
Local parameters modified - none.
Calling procedures - none.
Called procedures - RESPONSE, INTRSTUB, INITBORD, BORDSTUB.
Superior procedure - program DATASTUB.
Sub-procedures - INTRSTUB, CALCFILE, INITBORD, BORDSTUB.

AD-A155 600 A COMPARISON AND ANALYSIS OF VINTR'S GLOBAL ROUTING ALGORITHM WITH THE LE.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. F T CHESLEY 2/2

UNCLASSIFIED MAR 85 AFIT/GCS/ENG/85M-1 F/G 12/1 NL





Special processing - if the user wishes to build an interconnection file, he will input the width/length coordinates for both start and end cells. The program will prompt for each specific coordinate and all four coordinates have to be entered. The nets can then be accepted or rejected. If there are more nets, additional data can be entered. If there are no more entries, the data can now be saved. Not saving the data will destroy the entire interconnection file. The program will then ask to initialize the board. To do this, the program will create and initialize the segments and mark those cell found in the interconnection (net) file as unavailable on the board. The last input routine is for obstacle cell coordinates. An entry can be rejected and multiple obstacle cells can be entered.

```
CONST
  MAXSEGWD=2;
  MAXSEGLN=2;
TYPE
  INTRDATA=PACKED ARRAY[1..4] OF 1..60;
  RECDATA=PACKED RECORD
          OBS:BOOLEAN;
          RCH:BOOLEAN;
          DIR:PACKED ARRAY[1..3] OF 0..2
         END;
  GRIDDATA=PACKED ARRAY[1..30,1..30,1..2] OF RECDATA;
VAR
  INTRFILE:FILE OF INTRDATA;
  INTR:INTRDATA;
  BORDFILE:FILE OF GRIDDATA;
  GRID:GRIDDATA;
  I,J,K:INTEGER;
  SEGWD,SEGLN,NEWWD,NEWLN:INTEGER;
  CURSEGWD,CURSEGLN:INTEGER;
  NUM,MAXWID,MAXLEN:INTEGER;
  ANSWER:CHAR;
  SEGCHR1,SEGCHR2,CURSEGFILE,NAME,FILENAME:STRING;
  REDO,MORE:BOOLEAN

BEGIN
  MAXWID:=MAXSEGWD*30;
  MAXLEN:=MAXSEGLN*30;
  REDO:=TRUE;
  MORE:=TRUE;
  WRITE ('DO YOU WISH TO BUILD AN INTERCONNECTION ');
  WRITE ('LIST?(Y/N) ');
  RESPONSE;
  IF (ANSWER='Y') OR (ANSWER='y') THEN
    INTRSTUB;
  WRITE ('DO YOU WISH TO INITIALIZE THE BOARD AT ');
  WRITE ('THIS TIME?(Y/N) ');
  RESPONSE;
```

```
  IF (ANSWER='Y') OR (ANSWER='y') THEN
    BEGIN (*THEN*)
      WRITELN ('INITALIZING BOARD, PLEASE WAIT.');
      INITBORD
    END; (*THEN*)
  WRITELN ('DO YOU WISH TO SPECIFY ADDITIONAL ');
  WRITE ('OBSTACLE CELL DATA?(Y/N) ');
  RESPONSE;
  IF (ANSWER='Y') OR (ANSWER='y') THEN
    BORDSTUB;
  WRITELN ('***END OF DATa STUB***')
END. (*END DATASTUB*)
```

```
PROCEDURE RESPONSE;
  Function - reads the keyboard for a user response to a question.
  Global parameters modified - ANSWER.
  Local parameters modified - none.
  Calling procedures - program DATASTUB, INTRSTUB, BORDSTUB.
  Called procedures - none.
  Superior procedure - program DATASTUB.
  Sub-procedures - none.
  Special processing - checks to ensure a yes or no response is given.

BEGIN
  REPEAT
    READ (KEYBOARD,ANSWER)
  UNTIL ANSWER IN ['Y','N','y','n'];
  WRITELN (ANSWER)
END; (*PROCEDURE RESPONSE*)
```

```
PROCEDURE NUMCHK;
  Function - validates that input data is within the proper numerical
    range of values.
  Global parameters modified - NUM.
  Local parameters modified - none.
  Calling procedures - INTRSTUB.
  Called procedures - none.
  Superior procedure - program DATASTUB.
  Sub-procedures - none.
  Special processing - ensure coordinate values range from 1 to 60.

BEGIN
  REPEAT
    READ (KEYBOARD,NUM)
  UNTIL NUM IN [1..60];
  WRITELN (NUM)
END; (*PROCEDURE NUMCHK*)
```

```
PROCEDURE INTRSTUB;
  Function - input interconnection cell coordinates.
  Global procedures modified - NAME, FILENAME, REDO, INTR, INTRFILE,
    MORE.
  Local procedures modified - none.
  Calling procedures - program DATASTUB.
  Called procedures - NUMCHK, RESPONSE.
  Superior procedure - program DATASTUB.
  Sub-procedures - none.
  Special processing - coordinates are input one at a time.  The entry
    can be redone, eliminated, or accepted.  If there is no more input
    data, the interconnection file can be saved or destroyed.

BEGIN
  WRITE ('WHAT IS THE NAME OF YOUR INTERCONNECTION FILE? ');
  READ (KEYBOARD,NAME);
  WRITELN (NAME);
  IF NAME='' THEN
    NAME:='INTRFILE';
  FILENAME:=CONCAT ('PSEUDO:',NAME,'.DATA');
  REWRITE (INTRFILE,FILENAME);
  WHILE MORE DO
    BEGIN (*WHILE1*)
      REDO:=TRUE;
      WHILE REDO DO
        BEGIN (*WHILE2*)
          WRITE ('INPUT START WIDTH. ');
          NUMCHK;
          INTR[1]:=NUM;
          WRITE ('INPUT START LENGTH. ');
          NUMCHK;
          INTR[2]:=NUM;
          WRITE ('INPUT END WIDTH. ');
          NUMCHK;
          INTR[3]:=NUM;
          WRITE ('INPUT END LENGTH. ');
          NUMCHK;
          INTR[4]:=NUM;
          WRITE ('ROUTE ',INTR[1],',',INTR[2],' TO ');
          WRITELN (INTR[3],',',INTR[4]);
          WRITE ('DO YOU WISH TO REDO THIS ENTRY?(Y/N) ');
          RESPONSE;
          IF (ANSWER='N') OR (ANSWER='n') THEN
            REDO:=FALSE
        END; (*WHILE2*)
      INTRFILE^:=INTR;
      PUT (INTRFILE);
      WRITE ('MORE CONNECTIONS?(Y/N) ');
      RESPONSE;
```

```
      IF (ANSWER='N') OR (ANSWER='n') THEN
        MORE:=FALSE
    END; (*WHILE1*)
  WRITE ('SAVE DATA?(Y/N) ');
  RESPONSE;
  IF (ANSWER='Y') OR (ANSWER='y') THEN
    CLOSE (INTRFILE,LOCK)
  ELSE
    CLOSE (INTRFILE)
END; (*PROCEDURE INTRSTUB*)
```

```
PROCEDURE CALCFILE (I1,I2:INTEGER);
  Function - the absolute width coordinate (I1) and absolute length
    coordinate (I2) are passed to this procedure to detect when a cell
    is in a different segment and to calculate the cells's relative
    coordinates.
  Global parameters modified - SEGWD, SEGLN.
  Local parameters modified - I1, I2.
  Calling procedures - INITBORD, BORDSTUB.
  Called procedures - CALCSEG, EXCHFILE, NEWCOOR.
  Superior procedure - program DATASTUB.
  Sub-procedures - CALCSEG, EXCHFILE, NEWCOOR.
  Special processing - if the segment has changed, segments are
  exchanged.  Relative coordinates are then calculated for the cell.

BEGIN
  SEGWD:=CALCSEG(I1);
  SEGLN:=CALCSEG(I2);
  IF (SEGWD<>CURSEGWD) OR (SEGLN<>CURSEGLN) THEN
    BEGIN (*THEN*)
      EXCHFILE; (*PROCEDURE*)
      NEWCOOR (*PROCEDURE*)
    END (*THEN*)
  ELSE
    NEWCOOR (*PROCEDURE*)
END;  (*PROCEDURE CALCFILE*)
```

```
FUNCTION CALCSEG (VAL:INTEGER):INTEGER;
  Function - this function returns an integer value.  It accepts an
    absolute integer coordinate (VAL) and computes the relative
    coordinate within a grid segment.
  Global parameters modified - none.
  Local parameters modified - VAL, CALCSEG.
  Calling procedures - CALCFILE.
  Called procedures - none.
  Superior procedure - CALCFILE.
  Sub-procedures - none.
  Special processing - this function is called twice in CALCFILE, once
    to calculate the relative width and once to calculate the relative
    length coordinates.

BEGIN
  CALCSEG:=1+((VAL-1) DIV 30)
END; (*PROCEDURE CALCSEG*)
```

```
PROCEDURE EXCHFILE;
  Function - writes the old segment to secondary storage and reads
    the new  segment into primary memory.
  Global parameters modified - CURSEGWD, CURSEGLN, SEGCHR1, SEGCHR2,
    CURSEGFILE, BORDFILE.
  Local parameters modified - none.
  Calling procedures - CALCFILE.
  Called procedures - none.
  Superior procedrue - CALCFILE.
  Sub-procedures - none.
  Special processing - as each new file is read into memory, its entry
    is purged from secondary storage.  Also, the width and length
    integer values are converted to string values and concatenated with
    other string values to form the current file name.  The [16]
    appended to the end of the filename ensures sufficient and necessary
    storage is reserved for the file.

BEGIN
  REWRITE (BORDFILE,CURSEGFILE);
  BORDFILE^:=GRID;
  PUT (BORDFILE);
  CLOSE (BORDFILE,LOCK);
  STR (SEGWD,SEGCHR1);
  STR (SEGLN,SEGCHR2);
  CURSEGWD:=SEGWD;
  CURSEGLN:=SEGLN;
  CURSEGFILE:=CONCAT('PSEUDO:SEG',SEGCHR1,SEGCHR2,
                     '.DATA[16]');
  RESET (BORDFILE,CURSEGFILE);
  GRID:=BORDFILE^;
  CLOSE (BORDFILE,PURGE)
END; (*PROCEDURE EXCHFILE*)
```

```
PROCEDURE NEWCOOR;
  Function - uses the absolute width (SEGWD) and length (SEGLN) integer
    values passed to CALCFILE to calculate the relative cell position
    within the segment.
  Global parameters modified - NEWWD, NEWLN.
  Local parameters modified - none.
  Calling procedures - CALCFILE.
  Called procedures - none.
  Superior procedure - CALCFILE.
  Sub-procedures - none.
  Special processing - none.

BEGIN
  NEWWD:=I1-((SEGWD-1)*30);
  NEWLN:=I2-((SEGLN-1)*30)
END; (*PROCEDURE NEWCOOR*)
```

```
PROCEDURE INITBORD;
  Function - initialize all segment parameters for the entire board.
  Global parameters modified - SEGWD, SEGLN, CURSEGWD, CURSEGLN,
    SEGCHR1, SEGCHR2, CURSEGFILE, BORDFILE, I, J, K, OBS, RCH, DIR,
    INTR.
  Local parameters modified - none.
  Calling procedures - INITBORD, BORDSTUB.
  Called procedures - CALCFILE.
  Superior procedure - program DATASTUB.
  Sub-procedures - none.
  Special processing - first, mark OBS and RCH flags as false and all
    DIR as 0, then use the interconnection entry coordinates to flag
    cells as obstacles.  Saves the segments to RAM disk.

BEGIN
  FOR SEGWD:=1 TO MAXSEGWD DO
    FOR SEGLN:=1 TO MAXSEGLN DO
      BEGIN (*FOR*)
        CURSEGWD:=SEGWD;
        CURSEGLN:=SEGLN;
        STR (SEGWD,SEGCHR1);
        STR (SEGLN,SEGCHR2);
        CURSEGFILE:=CONCAT('PSEUDO:SEG',SEGCHR1,SEGCHR2,
                           '.DATA[16]');
        WRITELN (CURSEGFILE);
        REWRITE (BORDFILE,CURSEGFILE);
        FOR I:=1 TO 30 DO
          FOR J:=1 TO 30 DO
            FOR K:=1 TO 2 DO
              WITH GRID[I,J,K] DO
                BEGIN (*WITH*)
                  OBS:=FALSE;
                  RCH:=FALSE;
                  DIR[1]:=0;
                  DIR[2]:=0;
                  DIR[3]:=0
                END; (*WITH*)
        BORDFILE^:=GRID;
        PUT (BORDFILE);
        CLOSE (BORDFILE,LOCK)
      END; (*FOR*)
  RESET (INTRFILE,FILENAME);
  WHILE (NOT EOF(INTRFILE)) DO
    BEGIN (*WHILE*)
      INTR:=INTRFILE^;
      I:=INTR[1];
      J:=INTR[2];
      CALCFILE (I,J);
      GRID[NEWWD,NEWLN,1].OBS:=TRUE;
      GRID[NEWWD,NEWLN,2].OBS:=TRUE;
      I:=INTR[3];
```

```
        J:=INTR[4];
        CALCFILE (I,J);
        GRID[NEWWD,NEWLN,1].OBS:=TRUE;
        GRID[NEWWD,NEWLN,2].OBS:=TRUE;
        GET (INTRFILE)
      END; (*WHILE*)
    CLOSE (INTRFILE);
    REWRITE (BORDFILE,CURSEGFILE);
    BORDFILE^:=GRID;
    PUT (BORDFILE);
    CLOSE (BORDFILE,LOCK)
  END; (*PROCEDURE INITBORD*)
```

```
PROCEDURE BORDSTUB;
  Function - to enter obstacle cells on the board and mark as
    unavailable.
  Global parameters modified - MORE, I, J, K, OBS.
  Local parameters modified - none.
  Calling procedures - program DATASTUB.
  Called procedures - CALCFILE.
  Superior procedure - program DATASTUB.
  Sub-procedures - none.
  Special processing - ensure each coordinate is valid and for all
    entries to be accepted, rejected, or redone.  Allow multiple cell
    inputs.

BEGIN
  MORE:=TRUE;
  WHILE MORE DO
    BEGIN (*WHILE*)
      WRITE ('ENTER WIDTH COORDINATE. ');
      REPEAT
        READ (KEYBOARD,I)
      UNTIL I IN [1..60];
      WRITELN (I);
      WRITE ('ENTER LENGTH COORDINATE. ');
      REPEAT
        READ (KEYBOARD,J)
      UNTIL J IN [1..60];
      WRITELN (J);
      WRITE ('ENTER SIDE COORDINATE. ');
      REPEAT
        READ (KEYBOARD,K)
      UNTIL K IN [1..60];
      WRITELN (K);
      CALCFILE (I,J);
      GRID[NEWD,NEWLN,K].OBS:=TRUE;
      WRITELN ('DO YOU WISH TO REDO THIS ');
      WRITE ('OBSTACLE CELL?(Y/N) ');
      RESPONSE;
      IF (ANSWER='N') OR (ANSWER='n') THEN
        REDO:=FALSE
      ELSE
        GRID[NEWWD,NEWLN,K].OBS:=FALSE;
      WRITE ('MORE?(Y/N) ');
      RESPONSE;
      IF (ANSWER='N') OR (ANSWER='n') THEN
        MORE:=FALSE
    END (*WHILE*)
END; (*BORDSTUB*)
```

APPENDIX E: Program ROUTE

Program

Program ROUTE is a Pascal program using a heuristic algorithm (LEE) to route wires on a two-layer printed circuit board (GRIDDATA). Following execution of the initial route (LEE), Vintr's algorithm is executed (VINTR). Although the program can only store 30X30X2 cells (GRIDDATA) in memory at one time, extra board segments are kept on secondary storage (BORDFILE) and retrieved as necessary.

Inputs - (1) an interconnection file (INTRFILE) with net width/length coordinates.
(2) individual board segments (GRIDDATA) as required.

Outputs - (1) a route file (ROUTFILE) with start and end cells and all intermediate cells forming the connection path.
(2) updated individual board segments (GRIDDATA).

Constants

MAXIMPOS - defines the maximum allowable unroutable nets.

MAXSEGWD, MAXSEGLN - these values define the number of segments along the width and length of the board, respectively. Together, they define the overall board dimensions. To increase the board dimensions, merely increase these values.

Array Types

INTRDATA - a 4X1 packed array storing integer values of 0..60. The values represent board coordinate values. Element 1 is the start width, element 2 is the start length, element 3 is the stop width, and element 4 is the stop length.

ROUTDATA - a 3X1 packed array storing integer values of 1..60. The values represent the coordinate values of cell elements. Element 1 is the width, element 2 is length, and element 3 is the side.

GRIDDATA - a 30X30X2 packed array of RECDATA. Each specific element of the array corresponds to a specific position in a single board segment.

Record Type

RECDATA - each cell is represented by a single record. Each record has two flags and a 3X1 packed array storing integer values of 0..2. The first flag is OBS and designates whether or not the cell is an obstacle. The second flag is RCH and designates whether or not the cell has been previously reached during the current cell expansion phase. The 3X1 array (DIR) represents the integer values required for retrace once the end cell has been reached. The first element of DIR is the width retrace direction, the second element of DIR is the length retrace direction, and the third element of DIR is the side retrace direction. A value of 0 signifies no change, 1 signifies a +1 change, and 2 signifies a -1 change in the present coordinate position. These values are added to the current coordinates to achieve the next retrace cell. Note that only one direction parameter should ever be non-zero.

Variables

INTRFILE - a secondary storage file of INTRDATA. Entries are the width/length coordinates of the net's end cells to be connected.

IMPOS - the width/length coordinates of an unrouted net.

INTR - the width/length coordinates of a net's end cells.

ROUTFILE - a secondary storage file of ROUTDATA. A single entry represents the coordinates of a cell. A connection is stored as a series of entries in the following form: start cell, end cell, intermediate cells, start cell. This format aids the calculations necessary in procedures CHKDIST and BLDTAIL.

ST, RT - these integer values hold the cell coordinates of a path and are used to partially control the printing process

ROUT - an individual entry in ROUTFILE.

BORDFILE - a secondary storage file of GRIDDATA. Each file is a 30X30X2 array of cells and is termed a board segment.

GRID - an individual entry in GRIDDATA.

RLIST, ELIST - 360X3 packed arrays with integer values of 0..120. Each is a sequential listing (up to 360 entries) of cell coordinates (width, length, side). RLIST stores the coordinates of cells that are reached during the current expansion phase. ELIST is the list of cells available for expansion in the next cell expansion phase.

WC, LC, SC - the current cell of width, length, and side, respectively.

WCA1, WCS1, LCA1, LCS1 - the width coordinate changes for possible expansion (WCA1 is a +1 change and WCS1 is a -1 change) and the length coordinate changes for possible expansion (LCA1 is a +1 change and LCS1 is a -1 change.

BW, BL, EW, EL, ES - coordinates for the start width, start length, end width, end length, and end side, respectively.

NW, NL, NS - coordinates for the new width, length, and side which are calculated from the current cell coordinates and the addition of the current cell's DIR elements.

SEGWD, SEGLN, CURSEGWD, CURSEGLN - these are integer values that keep track of the current segment loaded into memory (CURSEGWD and CURSEGLN) and the segment (SEGWD and SEGLN) containing the cell currently under consideration. A suffix of WD designates a width coordinate. A suffix of LN designates a length coordinate.

NEWWD, NEWLN - these integer values are the relative coordinates of the current cell within a board segment.

CONNECTED, IMPOSSIBLE - these are boolean flags. CONNECTED indicates a connection is successful for the net. IMPOSSIBLE indicates a connection is not successful for the net.

E, R, NUM - E is the index for ELIST, R is the index for RLIST, and NUM is the total number of entries in RLIST for the current expansion.

NUMIMPOS - an integer value of the total number of unrouted nets.

MAXWID, MAXLEN - these are integer values representing the maximum board dimensions.

ANSWER - a character value read from the keyboard in response to a question.
SEGCHR1, SEGCHR2 - the string equivalent of SEGWD and SEGLN, respectively.
CURSEGFILE - the name of the current secondary storage file loaded into memory. Other literal characters are concatenated with SEGCHR1 and SEGCHR2 to uniquely define the individual board segments.
IMPOSLIST - a list of cell coordinates for unrouted nets.
ITERATION - an integer value for the number of the current Vintr interation.
LENGTH - an integer value summing the Manhattan distance of a path.
X, Y, Z, CHAR1, CHAR2, CHAR3 - individual coordinate values of a path (width, length, side) are stored in X, Y, Z respectively. Then they are converted to string values and stored in CHAR1, CHAR2, and CHAR3 respectively. Used in the printing process.
OUTPUT - string variable storing the output media ("PRINTER").
BEG - a character value variable denoting various conditions during the printing process. A "C" indicates a change to a new path, "Y" indicates the start of a new path, and "N" indicates the end of the path.
INDEX - a general index.
I - a counter to sum the number of path cells printed across the paper.
FID - the variable name given to interactive use of the printer for the printing process.

```
PROGRAM ROUTE;
  Function - executes the LEE algorithm first and then VINTR.
  Global parameters modified - MAXWID, MAXLEN, ITERATION, ROUTFILE,
    NUMIMPOS, IMPOS, IMPOSLIST, INDEX.
  Local parameters modified - none.
  Calling procedures - none.
  Called procedures - YESNO, LEE, PRINTIT, VINTR.
  Superior procedure - none.
  Sub-procedures - YESNO, PRINTIT, CALCFILE, INIT, RETRACE, SORTLIST,
    SORTSEG, RCHECK, EXPAND, EXPCK1, EXPCK2, LEE, VINTR.
  Special processing - open the data input file (INTRFILE) and the data
    output file (ROUTFILE).  Initialize the impossible connection list.
    then run LEE, print the results and run VINTR.

CONST
  MAXIMPOS=40;
  MAXSEGWD=2;
  MAXSEGLN=2;
TYPE
  INTRDATA=PACKED ARRAY[1..4] OF 0..60;
  ROUTDATA=PACKED ARRAY[1..3] OF 1..60;
  RECDATA=PACKED RECORD
          OBS:BOOLEAN;
          RCH:BOOLEAN;
          DIR:PACKED ARRAY[1..3] OF 0..2
         END;
  GRIDDATA=PACKED ARRAY[1..30,1..30,1..2] OF RECDATA;
VAR
  INTRFILE:FILE OF INTRDATA;
  IMPOS,INTR:INTRDATA;
  IMPOSLIST:PACKED ARRAY[1..MAXIMPOS] OF INTRDATA;
  ROUTFILE:FILE OF ROUTDATA;
  ST,RT,ROUT:ROUTDATA;
  BORDFILE:FILE OF GRIDDATA;
  GRID:GRIDDATA;
  RLIST,ELIST:PACKED ARRAY[1..360,1..3] OF 0..60;
  INDEX,WC,LC,SC,WCS1,WCA1,LCS1,LCA1:INTEGER;
  I,SEGWD,SEGLN,NEWWD,NEWLN,ITERATION,LENGTH:INTEGER;
  NUMIMPOS,CURSEGWD,CURSEGLN,X,Y,Z:INTEGER;
  E,R,NUM,BW,BL,EW,EL,ES,NW,NL,NS,MAXWID,MAXLEN:INTEGER;
  ANSWER,BEG:CHAR;
  SEGCHR1,SEGCHR2,CURSEGFILE,CHAR1,CHAR2,CHAR3:STRING;
  OUTPUT:STRING[8];
  CONNECTED,IMPOSSIBLE:BOOLEAN;
  FID:INTERACTIVE;

BEGIN
  ITERATION:=0;
  MAXWID:=MAXSEGWD*30;
  MAXLEN:=MAXSEGLN*30;
  WRITE ('DO YOU WISH TO RUN THE LEE ROUTER?(Y/N)  ');
```

```
    YESNO;
    IF (ANSWER='Y') THEN
      BEGIN (*THEN*)
        REWRITE (ROUTFILE,'PSEUDO:ROUTFILE.DATA[30]');
        RESET (INTRFILE,'PSEUDO:INTRFILE.DATA');
        NUMIMPOS:=0;
        FOR INDEX:=1 TO 4 DO
          IMPOS[INDEX]:=0;
        FOR INDEX:=1 TO MAXIMPOS DO
          IMPOSLIST[INDEX]:=IMPOS;
        LEE; (*PROCEDURE*)
        WRITELN ('TOTAL UNROUTABLE CONNECTIONS=',NUMIMPOS);
        CLOSE (ROUTFILE,LOCK);
        CLOSE (INTRFILE);
        PRINTIT; (*PROCEDURE*)
        VINTR (*PROCEDURE*)
      END; (*THEN*)
    WRITE ('END ROUTE')
  END.  (*PROGRAM ROUTE*)
```

```
PROCEDURE YESNO;
  Function - reads the keyboard response to a question.
  Global parameters modified - ANSWER.
  Local parameters modified - none.
  Calling procedures - program ROUTE.
  Called procedures - none.
  Superior procedure - program ROUTE.
  Sub-procedures - none.
  Special processing - checks to ensure a yes or no answer is given.

BEGIN
  REPEAT
    READ (KEYBOARD,ANSWER)
  UNTIL ANSWER IN ['Y','N','y','n'];
  WRITELN (ANSWER)
END; (*PROCEDURE YESNO*)
```

```
PROCEDURE PRINTIT;
  Function - output ROUTFILE data and then the impossible connections
    to a printer.
  Global parameters modified - BEG, I, OUTPUT, CHAR1, CHAR1, INDEX,
    IMPOS.
  Local parameters modified - none.
  Calling procedures - program ROUTE, VINTR.
  Called procedures - DISPLAY.
  Superior procedure - program ROUTE.
  Sub-procedures - DISPLAY.
  Special processing - controls the number of retrace cells printed
    across the page and prints the unconnected nets.

BEGIN
  BEG:='c';
  I:=0;
  OUTPUT:='PRINTER:';
  RESET (FID,OUTPUT);
  WRITELN (FID,'***ROUTFILE PRINT***');
  RESET (ROUTFILE,'PSEUDO:ROUTFILE.DATA');
  STR (ITERATION,CHAR1);
  IF ITERATION>0 THEN
    WRITELN (FID,'ITERATION=',CHAR1);
  WHILE NOT EOF (ROUTFILE) DO
    BEGIN (*WHILE*)
      IF I=8 THEN
        BEGIN (*THEN*)
          WRITELN (FID);
          DISPLAY; (*PROCEDURE*)
          GET (ROUTFILE);
          I:=1
        END (*THEN*)
      ELSE
        BEGIN (*ELSE*)
          DISPLAY; (*PROCEDURE*)
          GET (ROUTFILE);
          I:=I+1
        END (*ELSE*)
    END; (*WHILE*)
  CLOSE (ROUTFILE);
  WRITELN (FID,'END OF ROUTFILE.');
  WRITELN (FID);
  WRITELN (FID);
```

```
  IF NUMIMPOS>0 THEN
    BEGIN (*THEN*)
      WRITELN (FID,'***NOROUTE PRINT***');
      FOR INDEX:=1 TO NUMIMPOS DO
        BEGIN (*FOR*)
          IMPOS:=IMPOSLIST[INDEX];
          WRITELN (FID,IMPOS[1],',',IMPOS[2],':',IMPOS[3],
                       ',',IMPOS[4])
        END; (*FOR*)
      WRITELN (FID,'END OF NOROUTE.')
    END; (*THEN*)
  WRITELN (FID);
  WRITELN (FID);
  CLOSE (FID)
END; (*PROCEDURE PRINTIT*)
```

```
PROCEDURE DISPLAY;
  Function - control the printing process of when paths change.
  Global parameters modified - RT, LENGTH, ST, BEG, X, Y, Z, CHAR1,
    CHAR2, CHAR3.
  Local parameters modified - none.
  Calling procedures - PRINTIT.
  Called procedures - none.
  Superior procedure - PRINTIT.
  Sub-procedures - none.
  Special processing - the coordinates of the path are read and con-
    verted to string values for printing.  The length is simultaneously
    cummed as cells are processed after subtracting 2 from the initial
    start length retrace path due to some redundant data for each cell
    in ROUTFILE.  The length is also printed when coordinates for a new
    path are encountered.

BEGIN
  RT:=ROUTFILE^;
  IF BEG='C' THEN
    BEGIN (*THEN*)
      LENGTH:=-2;
      ST:=RT;
      BEG:='Y'
    END; (*THEN*)
  X:=RT[1];
  Y:=RT[2];
  Z:=RT[3];
  STR (X,CHAR1);
  STR (Y,CHAR2);
  STR (Z,CHAR3);
  LENGTH:=LENGTH+1;
  WRITE (FID,CHAR1,',',CHAR2,',',CHAR3,'  ');
  IF (ST=RT) THEN
    BEGIN (*THEN1*)
      IF BEG='N' THEN
        BEGIN (*THEN2*)
          WRITELN (FID);
          WRITELN (FID,'LENGTH= ',LENGTH);
          WRITELN (FID);
          WRITELN (FID);
          BEG:='C'
        END; (*THEN2*)
      IF BEG='Y' THEN
        BEG:='N'
    END (*THEN1*)
END; (*PROCEDURE DISPLAY*)
```

```
PROCEDURE CALCFILE (I1,I2:INTEGER);
  Function - the absolute width coordinate (I1) and absolute length
    coordinate (I2) are passed to this procedure to detect when a cell
    is in a different segment and to calculate relative cell
    coordinates.
  Global parameters modified - SEGWD, SEGLN.
  Local parameters modified - I1, I2.
  Calling procedures - RETRACE, EXPCK1, EXPCK2, LEE, CHKDIST, BLDTAIL,
    ROUTVNTR.
  Called procedures - CALCSEG, EXCHFILE, NEWCOOR.
  Superior procedure - program ROUTE.
  Sub-procedures - CALCSEG, EXCHFILE, NEWCOOR.
  Special processing - if the segment has changed, segments are
    exchanged.  Relative cell coordinates are then calculated for the
    point.

BEGIN
  SEGWD:=CALCSEG(I1);
  SEGLN:=CALCSEG(I2);
  IF (SEGWD<>CURSEGWD) OR (SEGLN<>CURSEGLN) THEN
    BEGIN (*THEN*)
      EXCHFILE; (*PROCEDURE*)
      NEWCOOR (*PROCEDURE*)
    END (*THEN*)
  ELSE
    NEWCOOR (*PROCEDURE*)
END;  (*PROCEDURE CALCFILE*)
```

```
FUNCTION CALCSEG (VAL:INTEGER):INTEGER;
  Function - this function returns an integer value.  It accepts an
    absolute integer coordinate (VAL) and computes the relative cell
    coordinates within a segment.
  Global parameters modified - none.
  Local parameters modified - VAL, CALCSEG.
  Calling procedures - CALCFILE.
  Called procedures - none.
  Superior procedure - CALCFILE.
  Sub-procedures - none.
  Special processing - this function is called twice in CALCFILE, once
    to calculate the relative width and once to calculate the relative
    length cell coordinates.

BEGIN
  CALCSEG:=1+((VAL-1) DIV 30)
END; (*PROCEDURE CALCSEG*)
```

```
PROCEDURE EXCHFILE;
  Function - writes the old segment to secondary storage and reads
    the new  segment into primary memory.
  Global parameters modified - CURSEGWD, CURSEGLN, SEGCHR1, SEGCHR2,
    CURSEGFILE, BORDFILE, GRID.
  Local parameters modified - none.
  Calling procedures - CALCFILE.
  Called procedures - none.
  Superior procedrue - CALCFILE.
  Sub-procedures - none.
  Special processing - as each new file is read into memory, its entry
    is purged from secondary storage.  Also, the width and length
    integer values are converted to string values and concatenated with
    other string  values to form the current file name.  The [16]
    appended to the end of the filename ensures sufficient and necessary
    storage is reserved for the file.

BEGIN
  REWRITE (BORDFILE,CURSEGFILE);
  BORDFILE^:=GRID;
  PUT (BORDFILE);
  CLOSE (BORDFILE,LOCK);
  CURSEGWD:=SEGWD;
  CURSEGLN:=SEGLN;
  STR (SEGWD,SEGCHR1);
  STR (SEGLN,SEGCHR2);
  CURSEGFILE:=CONCAT('PSEUDO:SEG',SEGCHR1,SEGCHR2,
                     '.DATA[16]');
  RESET (BORDFILE,CURSEGFILE);
  GRID:=BORDFILE^;
  CLOSE (BORDFILE,PURGE)
END; (*PROCEDURE BORDFILE*)
```

```
PROCEDURE NEWCOOR;
  Function - uses the absolute width (SEGWD) and length (SEGLN) integer
    values passed to CALCFILE to calculate the relative cell position
    within the segment.
  Global parameters modified - NEWWD, NEWLN.
  Local parameters modified - none.
  Calling procedures - CALCFILE.
  Called procedures - none.
  Superior procedure - CALCFILE.
  Sub-procedures - none.
  Special processing - none.

BEGIN
  NEWWD:=I1-((SEGWD-1)*30);
  NEWLN:=I2-((SEGLN-1)*30)
END; (*PROCEDURE NEWCOOR*)
```

```
PROCEDURE INIT;
  Function - with the start of each new net, all DIR and RCH
    entries for the cells of all segments are initialized.  In
    addition, all RLIST and ELIST entries are initialized.
  Global parameters modified - CURSEGWD, CURSEGLN, SEGCHR1, SEGCHR2,
    CURSEGFILE, DIR, RCH, RLIST, ELIST, R, SC, IMPOSSIBLE, CONNECTED,
    NUM, BORFILE, GRID.
  Local parameters modified - none.
  Calling procedures - LEE.
  Called procedures - none.
  Superior procedure - program ROUTE.
  Sub-procedures - none.
  Special processing - none.

BEGIN
  WRITELN ('INITIALIZING BOARD SEGMENT.');
  FOR SEGWD:=1 TO MAXSEGWD DO
    FOR SEGLN:=1 TO MAXSEGLN DO
      BEGIN (*FOR*)
        CURSEGWD:=SEGWD;
        CURSEGLN:=SEGLN;
        STR (SEGWD,SEGCHR1);
        STR (SEGLN,SEGCHR2);
        CURSEGFILE:=CONCAT('PSEUDO:SEG',SEGCHR1,SEGCHR2,
                           '.DATA[16]');
        RESET (BORDFILE,CURSEGFILE);
        GRID:=BORDFILE^;
        CLOSE (BORDFILE,PURGE);
        FOR WC:=1 TO 30 DO
          FOR LC:=1 TO 30 DO
            FOR SC:=1 TO 2 DO
              WITH GRID[WD,LC,SC] DO
                BEGIN (*WITH*)
                  DIR[1]:=0;
                  DIR[2]:=0;
                  DIR[3]:=0;
                  RCH:=FALSE
                END; (*WITH*)
        REWRITE (BORDFILE,CURSEGFILE);
        BORDFILE^:=GRID;
        PUT (BORDFILE);
        CLOSE (BORDFILE,LOCK);
        WRITELN (CURSEGFILE,' INITIALIZED.')
      END; (*FOR*)
  RESET (BORDFILE,CURSEGFILE);
  GRID:=BORDFILE^;
  CLOSE (BORDFILE,LOCK);
```

```
    FOR WC:=1 TO 360 DO
      FOR LC:=1 TO 3 DO
        BEGIN (*FOR*)
          RLIST[WC,LC]:=0;
          ELIST[WC,LC]:=0
        END; (*FOR*)
    R:=1;
    SC:=1;
    IMPOSSIBLE:=FALSE;
    CONNECTED:=FALSE;
    INDEX:=1;
    NUM:=1
  END; (*PROCEDURE INIT*)
```

```
PROCEDURE RETRACE;
  Function - the retrace path is found.
  Global parameters modified - NW, NL, NS, OBS, BORDFILE.
  Local parameters modified - ADDW, ADDL, ADDS.
  Calling procedures - RCHECK.
  Called procedures - SAVEROUT, CALCFILE.
  Superior procedure - program ROUTE.
  Sub-procedures - SAVEROUT.
  Special processing - processing begins at the end cell.  While the
    current coordinates do not equal the start cell coordinates, each
    DIR value is retrieved from the current GRID entry, stored in an
    intermediate variable (ADDW, ADDL, and ADDS).  DIR values of "2" are
    stored as -1.  ADDW, ADDL, and ADDS are then added to the current
    coordinate position (NW, NL, or NS) except when the DIR value equals
    2.  In this case, -1 is added instead of 2.  Each successive set of
    cell coordinates are passed to SAVEROUT.

VAR
  ADDW,ADDL,ADDS:INTEGER;

BEGIN
  WRITELN (' BEGIN RETRACE ');
  SAVEROUT (BW,BL,1);
  NW:=EW;
  NL:=EL;
  NS:=ES;
  SAVEROUT (NW,NL,NS);
  WRITE (NW,',',NL,',',NS);
  CALCFILE (NW,NL);
  WITH GRID[NEWWD,NEWLN,NS] DO
    WRITELN (' ',DIR[1],' ',DIR[2],' ',DIR[3]);
  WHILE ((BW<>NW) OR (BL<>NL) OR (NS>1)) DO
    BEGIN (*WHILE*)
      WITH GRID[NEWWD,NEWLN,NS] DO
        BEGIN (*WITH*)
          IF DIR[1]=2 THEN
            ADDW:=-1
          ELSE
            ADDW:=DIR[1];
          IF DIR[2]=2 THEN
            ADDL:=-1
          ELSE
            ADDL:=DIR[2];
          IF DIR[3]=2 THEN
            ADDS:=-1
          ELSE
            ADDS:=DIR[3];
```

```
            NS:=NS+ADDS;
            NL:=NL+ADDL;
            NW:=NW+ADDW
          END; (*WITH*)
        CALCFILE (NW,NL);
        GRID[NEWWD,NEWLN,NS].OBS:=TRUE;
        SAVEROUT (NW,NL,NS);
        WRITE (NW,',',NL,',',NS);
        WITH GRID[NEWWD,NEWLN,NS] DO
          WRITELN (' ',DIR[1],' ',DIR[2],' ',DIR[3])
      END; (*WHILE*)
    REWRITE (BORDFILE,CURSEGFILE);
    BORDFILE^:=GRID;
    PUT (BORDFILE);
    CLOSE (BORDFILE,LOCK)
  END; (*PROCEDURE RETRACE*)
```

```
PROCEDURE SAVEROUT(I1,I2,I3:INTEGER);
  Function - the three coordinate values (I1, I2, I3) are passed to this
      procedure and saved on secondary storage.
  Global parameters modified - ROUT, ROUTFILE.
  Local parameters modified - I1, I2, I3.
  Calling procedures - RETRACE.
  Called procedures - none.
  Superior procedure - RETRACE.
  Sub-procedures - none.
  Special processing - none.

BEGIN
  ROUT[1]:=I1;
  ROUT[2]:=I2;
  ROUT[3]:=I3;
  ROUTFILE^:=ROUT;
  PUT (ROUTFILE)
END; (*PROCEDURE SAVEROUT*)
```

```
PROCEDURE SORTLIST;
  Function - entries are sorted according to the least Manhattan
    distance from RLIST's width and length entries to the end cell's
    coordinates.  The Manhattan distance of each two successive entries
    (S1 and S2) are compared.  If the higher indexed entry's distance is
    less than the other, then the two entries exchange positions in
    RLIST.  When all RLIST entries have been compared, the process
    begins again until no exchanges are made.
  Global parameters modified - RLIST.
  Local parameters modified - TEMP, SWITCH, S1, S2.
  Calling procedures - RCHECK.
  Called procedures - none.
  Superior procedure - program ROUTE.
  Sub-procedures - none.
  Special processing - a bubble-sort technique is used to sort RLIST in
    ascending distance using TEMP as a temporary array.  SWITCH is used
    to designate when entries are changed.  When SWITCH is true the sort
    starts from the beginning and processes RLIST until no exchanges are
    made (SWITCH=FALSE).

VAR
  TEMP:PACKED ARRAY[1..3] OF 1..60;
  SWITCH:BOOLEAN;
  S1,S2:INTEGER;

BEGIN
  SWITCH:=TRUE;
  WHILE SWITCH DO
    BEGIN (*WHILE*)
      SWITCH:=FALSE;
      FOR E:=R DOWNTO 2 DO
        BEGIN (*FOR*)
          S1:=ABS(RLIST[E,1]-EW)+ABS(RLIST[E,2]-EL);
          S2:=ABS(RLIST[E-1,1]-EW)+ABS(RLIST[E-1,2]-EL);
          IF S1<S2 THEN
            BEGIN (*THEN*)
              SWITCH:=TRUE;
              TEMP[1]:=RLIST[E-1,1]
              TEMP[2]:=RLIST[E-1,2]
              TEMP[3]:=RLIST[E-1,3]
              RLIST[E-1,1]:=RLIST[E,1];
              RLIST[E-1,2]:=RLIST[E,2];
              RLIST[E-1,3]:=RLIST[E,3];
              RLIST[E,1]:=TEMP[1];
              RLIST[E,2]:=TEMP[2];
              RLIST[E,3]:=TEMP[3]
            END (*THEN*)
        END (*FOR*)
    END (*WHILE*)
END; (*PROCEDURE SORTLIST*)
```

```
PROCEDURE SORTSEG;
  Function - to sort the RLIST entries into groups based upon the
    segment each belongs to in an effort to reduce the number of I/O
    operations.
  Global parameters modified - ELIST, RLIST.
  Local parameters modified - T1, T2, T3, TNUM, CURS1, CURS2, S1, S2.
  Calling procedures - RCHECK.
  Called procedures - none.
  Superior procedure - program ROUTE.
  Sub-procedures - none.
  Special processing - the first entry of RLIST is selected and the
    segment it belongs to is calculated.  That entry and all others
    belonging to the same  segment are written to ELIST and removed
    from RLIST.  Then the first entry is again retrieved from RLIST and
    the same process is continued until no RLIST entries remain.  T3 is
    the index for RLIST with TNUM entries in RLIST.  T2 is the total
    number of remaining entries in RLIST.  T1 is the total number of
    entries written to ELIST.  The segment coordinates of the first
    entry in RLIST (the closest cell to the end cell) are calculated and
    stored in CURS1 and CURS2 and the RLIST entry is stored in ELIST.
    Subsequent segment coordinates are stored in S1 and S2.  If the grid
    coordinates are the same, the new cell is stored in ELIST; other-
    wise, it is stored in the first available entry in RLIST.  When all
    cells of a specific segment are in ELIST, the first entry of
    RLIST is selected and the segment coordinates calculated.

VAR
  TNUM,T1,T2,T3,CURS1,CURS2,S1,S2:INTEGER;

BEGIN
  T1:=0;
  TNUM:=R;
  WHILE TNUM<>0 DO
    BEGIN (*WHILE*)
      T2:=0;
      FOR T3:=1 TO TNUM DO
        BEGIN (*FOR*)
          S1:=1+((RLIST[T3,1]-1) DIV 30);
          S2:=1+((RLIST[T3,2]-1) DIV 30);
          IF T3=1 THEN
            BEGIN (*THEN*)
              CURS1:=S1;
              CURS2:=S2
            END; (*THEN*)
```

```
            IF (S1=CURS1) AND (S2=CURS2) THEN
              BEGIN (*THEN*)
                T1:=T1+1;
                ELIST[T1,1]:=RLIST[T3,1];
                ELIST[T1,2]:=RLIST[T3,2];
                ELIST[T1,3]:=RLIST[T3,3]
              END (*THEN*)
            ELSE
              BEGIN (*ELSE*)
                T2:=T2+1;
                RLIST[T2,1]:=RLIST[T3,1];
                RLIST[T2,2]:=RLIST[T3,2];
                RLIST[T2,3]:=RLIST[T3,3]
              END (*ELSE*)
          END; (*FOR*)
        TNUM:=T2
      END (*WHILE*)
  END; (*PROCEDURE SORTSEG*)
```

```
PROCEDURE RCHECK;
  Function - saves unroutable nets, retraces path for routed nets,
    and prepares ELIST for the next series of expansions if no
    connection has been made.
  Global parameters modified - R, INTR, NUMIMPOS, IMPOSSIBLE, NUM,
    BORDFILE.
  Local parameters modified - none.
  Calling procedure - LEE.
  Called procedures - RETRACE, SORTLIST, SORTSEG.
  Superior procedure - program ROUTE.
  Sub-procedures - none.
  Special processing - stores unroutable nets in an array
    (IMPOSLIST).  If a connection has been made then RETRACE is called;
    otherwise, the RLIST entries are sorted and ELIST is built.

BEGIN
  IF R=0 THEN
    BEGIN (*THEN*)
      WRITELN (' ');
      WRITELN ('BW,',',BL,'->',EW,',',EL,' IMPOSSIBLE!');
      INTR[1]:=BW;
      INTR[2]:=BL;
      INTR[3]:=EW;
      INTR[4]:=EL;
      NUMIMPOS:=NUMIMPOS+1;
      IMPOSLIST[NUMIMPOS]:=INTR;
      REWRITE (BORDFILE,CURSEGFILE);
      BORDFILE^:=GRID;
      PUT (BORDFILE);
      CLOSE (BORDFILE,LOCK);
      IMPOSSIBLE:=TRUE
    END (*THEN*)
  ELSE
    IF CONNECTED THEN
      RETRACE
    ELSE
      BEGIN (*ELSE*)
        IF R>1 THEN
          SORTLIST;
        SORTSEG;
        NUM:=R;
        R:=0
      END (*ELSE*)
END; (*PROCEDURE RCHECK*)
```

```
PROCEDURE EXPAND (W,L,S:INTEGER);
  Function - to check the status of a cell to determine if it can be
    reached.
  Global parameters modified - R, RCH, ES, CONNECTED, RLIST.
  Local parameters modified - W, L, S.
  Calling procedures - EXPCK1, EXPCK2.
  Called procedures - none.
  Superior procedure - program ROUTE.
  Sub-procedures - none.
  Special processing - the cell coordinates (W, L, S) are passed and
    they are checked to determine if a connection has been made.  If so,
    the condition (CONNECTED) is set; otherwise, if the cell is not an
    obstacle, it is added to RLIST.

BEGIN
  IF (W=EW) AND (L=EL) THEN
    BEGIN (*THEN1*)
      R:=1;
      GRID[NEWWD,NEWLN,S].RCH:=TRUE;
      IF S=2 THEN
        BEGIN (*THEN2*)
          GRID[NEWWD,NEWLN,1].DIR[3]:=1;
          S:=1
        END; (*THEN2*)
      ES:=S;
      WRITELN ('CONNECTION IS MADE.');
      CONNECTED:=TRUE
    END (*THEN1*)
  ELSE
    IF NOT GRID[NEWWD,NEWLN,S].OBS THEN
      BEGIN (*THEN*)
        GRID[NEWWD,NEWLN,S].RCH:=TRUE;
        R:=R+1;
        RLIST[R,1]:=W;
        RLIST[R,2]:=L;
        RLIST[R,3]:=S
      END (*THEN*)
END; (*PROCEDURE EXPAND*)
```

```
PROCEDURE EXPCK1;
  Function - to check all adjacent cells for possible expansion
    and to record the retrace direction if a point is reached.
  Global parameters modified - DIR.
  Local parameters modified - none.
  Calling procedures - LEE.
  Called procedures - CALCFILE, EXPAND.
  Superior procedure - program ROUTE.
  Sub-procedures - none.
  Special processing - if the cell has not been connected, for each
    expansion direction attempted, a check is made to see if the cell
    has been previously reached.  If the cell has been reached, it is
    skipped and the next direction is attempted.  For a cell not
    reached, EXPAND is called and if the cell is then reached, the
    specific retrace direction is stored in the appropriate DIR location
    and the next direction is attempted.  This process continues until
    all four adjacent cells have been checked.

BEGIN
  IF (WCS1>=1) AND (NOT CONNECTED) THEN
    BEGIN (*THEN1*)
      CALCFILE (WCS1,LC);
      IF NOT GRID[NEWWD,NEWLN,SC].RCH THEN
        BEGIN (*THEN2*)
          EXPAND (WCS1,LC,SC);
          IF GRID[NEWWD,NEWLN,SC].RCH THEN
            GRID[NEWWD,NEWLN,SC].DIR[1]:=1;
      END (*THEN2*)
    END; (*THEN1*)
  IF (WCA1<=MAXWID) AND (NOT CONNECTED) THEN
    BEGIN (*THEN1*)
      CALCFILE (WCA1,LC);
      IF NOT GRID[NEWWD,NEWLN,SC].RCH THEN
        BEGIN (*THEN2*)
          EXPAND (WCA1,LC,SC);
          IF GRID[NEWWD,NEWLN,SC].RCH THEN
            GRID[NEWWD,NEWLN,SC].DIR[1]:=2;
      END (*THEN2*)
    END; (*THEN1*)
  IF (LCS1>=1) AND (NOT CONNECTED) THEN
    BEGIN (*THEN1*)
      CALCFILE (WC,LCS1);
      IF NOT GRID[NEWWD,NEWLN,SC].RCH THEN
        BEGIN (*THEN2*)
          EXPAND (WC,LCS1,SC);
          IF GRID[NEWWD,NEWLN,SC].RCH THEN
            GRID[NEWWD,NEWLN,SC].DIR[2]:=1;
      END (*THEN2*)
    END; (*THEN1*)
```

```
    IF (LCA1<=MAXLEN) AND (NOT CONNECTED) THEN
      BEGIN (*THEN1*)
        CALCFILE (WC,LCA1);
        IF NOT GRID[NEWWD,NEWLN,SC].RCH THEN
          BEGIN (*THEN2*)
            EXPAND (WC,LCA1,SC);
            IF GRID[NEWWD,NEWLN,SC].RCH THEN
              GRID[NEWWD,NEWLN,SC].DIR[2]:=2;
        END (*THEN2*)
      END; (*THEN1*)
  END; (*PROCEDURE EXPCK1*)
```

```
PROCEDURE EXPCK2
  Function - to check the adjacent cell on the opposite layer for
    possible expansion and record the retrace direction if the cell is
    reached.
  Global parameters modified - DIR.
  Local parameters modified - none.
  Calling procedures - LEE.
  Called procedures - CALCFILE, EXPAND.
  Superior procedure - program ROUTE.
  Sub-procedures - none.
  Special processing - if the cell has not been connected, the opposite
    direction expansion is attempted, a check is made to see if the
    cell has been previously reached.  If the cell has been reached,
    it is  skipped.  For a cell not reached, EXPAND is called and if
    the cell is then reached, the specific retrace direction is stored
    in the appropriate DIR location.

BEGIN
  IF (SC=1) AND (NOT CONNECTED) THEN
    BEGIN (*THEN1*)
      CALCFILE (WC,LC);
      IF NOT GRID[NEWWD,NEWLN,2].RCH THEN
        BEGIN (*THEN2*)
          EXPAND (WC,LC,2);
          IF GRID[NEWWD,NEWLN,2].RCH THEN
            GRID[NEWWD,NEWLN,2].DIR[3]:=2;
      END (*THEN2*)
    END; (*THEN1*)
  IF (SC=2) AND (NOT CONNECTED) THEN
    BEGIN (*THEN1*)
      CALCFILE (WC,LC);
      IF NOT GRID[NEWWD,NEWLN,1].RCH THEN
        BEGIN (*THEN2*)
          EXPAND (WC,LC,1);
          IF GRID[NEWWD,NEWLN,1].RCH THEN
            GRID[NEWWD,NEWLN,1].DIR[3]:=1;
      END (*THEN2*)
    END; (*THEN1*)
END; (*PROCEDURE EXPCK2*)
```

```
PROCEDURE LEE;
  Function - to drive the basic LEE algorithm.  It calculates the
    adjacent width and length coordinate values to be used during the
    actual expansion of a point.  LEE continues to execute until no
    ELIST entries are left to expand.
  Global parameters modified - BW, BL, EW, EL, ELIST, RLIST, WC, LC, SC,
    WCA1, WCS1, LCA1, LCS1, DIR.
  Local parameters modified - none.
  Calling procedures - ROUTVNTR.
  Called procedures - INIT, CALCFILE, EXPCK1, EXPCK2, RCHECK.
  Superior procedure - program ROUTE.
  Sub-procedures - none.
  Special processing - while there are still nets to route, LEE will
    retrieve each from INTRFILE and store the layer 1 beginning
    coordinates in ELIST and the layer 2 beginning coordinates in RLIST.
    Each entry in ELIST is expanded until the connection is made or it
    is determined that the connection is impossible.

BEGIN
  WHILE NOT EOF (INTRFILE) DO
    BEGIN (*WHILE1*)
      WRITELN ('***NEW CONNECTION***');
      INIT;
      INTR:=INTRFILE^;
      BW:=INTR[1];
      BL:=INTR[2];
      EW:=INTR[3];
      EL:=INTR[4];
      ELIST[1,1]:=BW;
      ELIST[1,2]:=BL;
      ELIST[1,3]:=1;
      RLIST[R,1]:=BW;
      RLIST[R,2]:=BL;
      RLIST[R,3]:=2;
      CALCFILE (BW,BL);
      GRID[NEWWD,NEWLN,2].DIR[3]:=2;
      WHILE (NOT CONNECTED) AND (NOT IMPOSSIBLE) DO
        BEGIN (*WHILE2*)
          FOR E:=1 TO NUM DO
            IF (NOT CONNECTED) THEN
              BEGIN (*THEN*)
                WC:=ELIST[E,1];
                LC:=ELIST[E,2];
                SC:=ELIST[E,3];
                WCS1:=WC-1;
                WCA1:=WC+1;
                LCS1:=LC-1;
                LCA1:=LC+1;
```

```
                IF (E=1) THEN
                  WRITELN (BW,',',BL,'->',EW,',',EL,'.',
                           WC,',',LC,',',SC);
                EXPCK1;
                EXPCK2
              END; (*THEN*)
            RCHECK
          END; (*WHILE2*)
        GET (INTRFILE)
      END (*WHILE1*)
    END; (*PROCEDURE LEE*)
```

```
PROCEDURE VINTR;
  Function - control the iteration and preparation of input data for
    processing by Vintr's algorithm and the printing of results.
  Global parameters modified - ITERATION, INDEX, LEN, ROUTFILE.
  Local parameters modified - TAILLIST.
  Calling procedures - program ROUTE.
  Called procedures - ROUTLEN, BLDTAIL, ROUTVNTR, PRINTIT.
  Superior procedure - program ROUTE.
  Sub-procedures - ROUTLEN, BLDTAIL, ROUTVNTR.
  Special processing - for each iteration the LEN array is initialized
    then processing control is sequentially passed to procedures for
    calculating the path lengths, building tails from the paths, routing
    the cell pairs, and printing results.

VAR
  TAILLIST:FILE OF ROUTDATA;
  NEW,START,STOP:ROUTDATA;
  REMOVE,COUNT,LENTAIL:INTEGER;
  LEN:ARRAY [1..80] OF INTEGER;

BEGIN
  FOR ITERATION:=1 TO 4 DO
    BEGIN (*FOR*)
      WRITELN ('ITERATION=#',ITERATION);
      FOR INDEX:=1 TO 80 DO
        LEN[INDEX]:=0;
      RESET (ROUTFILE,'PSEUDO:ROUTFILE.DATA');
      ROUTLEN;
      CLOSE (ROUTFILE,LOCK);
      RESET (ROUTFILE,'PSEUDO:ROUTFILE.DATA');
      REWRITE (TAILLIST,'PSEUDO:TAILLIST.DATA[30]');
      BLDTAIL;
      CLOSE (ROUTFILE,PURGE);
      CLOSE (TAILLIST,LOCK);
      REWRITE (ROUTFILE,'PSEUDO:ROUTFILE.DATA[30]');
      ROUTVNTR;
      CLOSE (ROUTFILE,LOCK);
      WRITELN ('TOTAL UNROUTABLE CONNECTIONS FOR);
      WRITELN ('ITERATION ',ITERATION,' IS ',NUMIMPOS);
      WRITELN;
      PRINTIT
    END (*FOR*)
END; (*PROCEDURE VNTR*)
```

```
PROCEDURE ROUTLEN;
  Function - calculates the path lengths.
  ROUTE Global parameters modified - INDEX.
  VINTR Global parameters modified - START, NEW, LEN.
  Local parameters modified - none.
  Calling procedures - VINTR.
  Called procedures - none.
  Superior procedure - VINTR.
  Sub-procedures - none.
  Special processing - lengths are calculated and stored in an array
    (LEN) corresponding to each entry in ROUTFILE.

BEGIN
  INDEX:=0;
  WHILE NOT EOF (ROUTFILE) DO
    BEGIN (*WHILE1*)
      INDEX:=INDEX+1;
      START:=ROUTFILE^;
      GET (ROUTFILE);
      GET (ROUTFILE);
      NEW:=ROUTFILE^;
      LEN[INDEX]:=LEN[INDEX]+1;
      WHILE (NEW<>START) DO
        BEGIN (*WHILE2*)
          LEN[INDEX]:=LEN[INDEX]+1;
          GET (ROUTFILE);
          NEW:=ROUTFILE^
        END; (*WHILE2*)
       GET (ROUTFILE)
    END (*WHILE1*)
END; (*PROCEDURE ROUTLEN*)
```

```
PROCEDURE BLDTAIL;
  Function - the length of a path is used to calculate the length of
    tail on each end of the path.  The intermediate cells are marked as
    available.  All cells along the tails are stored in TAILLIST.
  ROUTE Global parameters modified - BW, BL, INDEX, ROUT, NW, NL, NS,
    OBS, BORDFILE.
  VINTR Global parameters modified - TAILLIST, COUNT, REMOVE, LENTAIL,
    TAIL.
  Local parameters modified - none.
  Calling procedures - VINTR.
  Called procedures - CALCFILE.
  Superior procedure - VINTR.
  Sub-procedures - none.
  Special processing - ROUTFILE entries are read sequentially and cells
    that are part of a tail are saved in TAILLIST.  The length of a tail
    is calculated from the path length (LEN), the ITERATION, and a
    constant of 0.1.  Entries up to the length of the tail are saved,
    the intermediate cells have OBS in GRID set to available, and
    finally the second tail is saved.  This process repeats for each
    path in ROUTFILE.

BEGIN
  INDEX:=1;
  WHILE NOT EOF (ROUTFILE) DO
    BEGIN (*WHILE*)
      TAILLIST^:=ROUTFILE^;
      PUT (TAILLIST);
      LENTAIL:=TRUNC(LEN[INDEX]*ITERATION*0.1);
      FOR COUNT:=1 TO (LENTAIL+1) DO
        BEGIN (*FOR*)
          GET (ROUTFILE);
          TAILLIST^:=ROUTFILE^;
          PUT (TAILLIST)
        END; (*FOR*)
      REMOVE:=LEN[INDEX]-2*LENTAIL-1;
      FOR COUNT:=1 TO REMOVE DO
        BEGIN (*FOR*)
          GET (ROUTFILE);
          ROUT:=ROUTFILE^;
          NW:=ROUT[1];
          NL:=ROUT[2];
          NS:=ROUT[3];
          CALCFILE (NW,NL);
          GRID[NEWWD,NEWLN,NS].OBS:=FALSE
        END; (*FOR*)
```

```
      FOR COUNT:=1 TO (LENTAIL+1) DO
        BEGIN (*FOR*)
          GET (ROUTFILE);
          TAILLIST^:=ROUTFILE^;
          PUT (TAILLIST)
        END; (*FOR*)
      GET (ROUTFILE);
      INDEX:=INDEX+1
    END; (*WHILE*)
  REWRITE (BORDFILE,CURSEGFILE);
  BORDFILE^:=GRID;
  PUT (BORDFILE);
  CLOSE (BORDFILE,LOCK);
  WRITELN ('TAILLIST IS BUILT.')
END; (*PROCEDURE BLDTAIL*)
```

```
PROCEDURE ROUTVNTR;
  Function - process the unroutable nets, read TAILLIST entries
    and eliminate all obstacle cells in the path except for start/end
    cells, and passes the results to LEE for routing.
  ROUTE Global parameters modified - INTRFILE, INTR, ROUT, NW, NL, NS,
    OBS, BORDFILE.
  VINTR Global parameters modified - START, STOP.
  Local parameters modified - VNTRFILE, VNTR.
  Calling procedures - VINTR.
  Called procedures - CALCFILE, LEE.
  Superior procedure - VINTR.
  Sub-procedures - DONOROUTS.
  Special processing - DONOROUTS is called first if any unroutable nets
    exist.  The first entry in TAILLIST is read, its tails
    eliminated except for the start/end cells and this net is
    loaded into VNTRFILE and processed by LEE.  This process continues
    for each path in TAILLIST until an end of file (EOF) is encountered
    with TAILLIST.

VAR
  VNTRFILE:FILE OF INTRDATA;
  VNTR:INTRDATA;

BEGIN
  IF NUMIMPOS>0 THEN
    DONOROUTS;
  RESET (TAILLIST,'PSEUDO:TAILLIST.DATA');
  WHILE NOT EOF (TAILLIST) DO
    BEGIN (*WHILE1*)
      REWRITE (INTRFILE,'PSEUDO:VNTRFILE.DATA[30]');
      START:=TAILLIST^;
      GET (TAILLIST);
      STOP:=TAILLIST^;
      WRITELN ('BUILDING INTRFILE ENTRY.');
      INTR[1]:=START[1];
      INTR[2]:=START[2];
      INTR[3]:=STOP[1];
      INTR[4]:=STOP[2];
      WRITE ('ROUTE ',INTR[1],',',INTR[2],' TO ');
      WRITELN (INTR[3],',',INTR[4]);
      INTRFILE^:=INTR;
      PUT (INTRFILE);
      GET (TAILLIST);
      ROUT:=TAILLIST^;
      WHILE ((ROUT<>START) AND (ROUT<>STOP)
                           AND (NOT EOF(TAILLIST))) DO
        BEGIN (*WHILE2*)
          NW:=ROUT[1];
          NL:=ROUT[2];
          NS:=ROUT[3];
          CALCFILE (NW,NL);
```

```
            GRID[NEWWD,NEWLN,NS].OBS:=FALSE;
            GET (TAILLIST);
            ROUT:=TAILLIST^
          END; (*WHILE2*)
        REWRITE (BORDFILE,CURSEGFILE);
        BORDFILE^:=GRID;
        PUT (BORDFILE);
        CLOSE (BORDFILE,LOCK);
        WRITELN ('INTRFILE ENTRY COMPLETE.');
        CLOSE (INTRFILE,LOCK);
        RESET (INTRFILE,'PSEUDO:VNTRFILE.DATA');
        LEE;
        CLOSE (INTRFILE,LOCK);
        GET (TAILLIST)
      END; (*WHILE1*)
    CLOSE (TAILLIST,PURGE)
  END; (*PROCEDURE ROUTVNTR*)
```

```
PROCEDURE DONOROUTS;
  Function - route the unrouted nets.
  ROUTE Global parameters modified - INTRFILE, NUMIMPOS.
  VINTR Global parameters modified - none.
  Local parameters modified - none.
  Calling procedures - ROUTVNTR.
  Called procedures - LEE.
  Superior procedure - ROUTVNTR.
  Sub-procedures - none.
  Special processing - the entries from IMPOSLIST are loaded into
    VNTRFILE and then LEE is called to route the net.

BEGIN
  REWRITE (INTRFILE,'PSEUDO:VNTRFILE.DATA[30]');
  FOR INDEX:=1 TO NUMIMPOS DO
    BEGIN (*FOR*)
      INTRFILE^:=IMPOSLIST[INDEX];
      PUT (INTRFILE)
    END; (*FOR*)
  CLOSE (INTRFILE,LOCK);
  NUMIMPOS:=0;
  RESET (INTRFILE,'PSEUDO:VNTRFILE.DATA');
  LEE;
  CLOSE (INTRFILE,PURGE)
END; (*PROCEDURE DONOROUTS*)
```

VITA

Fred Thomas Chesley was born on 2 September 1950 in Washington, D.C. He graduated from high school in Sandpoint, Idaho in 1968 and attended Idaho State University, Pocatello, Idaho, from which he received the degree of Bachelor of Science in Secondary Education with a major in mathematics in August 1972. Upon graduation he was employed by the Department of Health and Welfare, State of Idaho, as a social caseworker. He entered the Air Force in April of 1977 and obtained a commission through Officer Training School in May 1978. He completed Computer Systems Development Officer school in September 1978 and was assigned as a systems analyst to Headquarters, Strategic Air Command, Offutt Air Force Base, Nebraska. He entered the School of Engineering, Air Force Institute of Technology, in June 1980.

Permanent address: 1822 Cal Young Rd., Apt. 147
Eugene, Oregon 97401

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/ENG/85M-1	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) See Box 19	
12. PERSONAL AUTHOR(S) Fred T. Chesley, B.S., Capt, USAF	
13a. TYPE OF REPORT MS Thesis 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1985 March 15. PAGE COUNT 137
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	Design Automation, Routing, Printed Circuit Boards, Lee's Algorithm, Vintr's Algorithm
09	02		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: A Comparison and Analysis of Vintr's Global Routing Algorithm with the Lee Routing Algorithm in Two-Layer Printed Circuit Boards

Thesis Chairman: Harold W. Carter, Lt Col, USAF
Assistant Professor of Electrical Engineering

Approved for public release: IAW AFR 190-17.

Lynn E. Wolaver 1 May 85

LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AIC)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Lt Col Harold W. Carter	513-255-6193	AFIT/ENG

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

Microcomputer software was designed and written to compare a standard routing technique (Lee) with an experimental, unpublished routing technique proposed by J. Vintr for two-layer printed circuit boards. Vintr's algorithm, as studied here, uses a four-iteration approach to minimize unroutable nets and minimize route distance. The unrouted nets and average route lengths were observed and analyzed for differing sizes of two-point nets.

Analysis revealed a reduction of unroutable connections across iterations, but congestion played a large role in the overall success of finding paths. A recommendation is made that use of 8-bit microcomputers in design automation is impractical, and research in this area of technology can best be accomplished using larger computer systems.

END

FILMED

7-85

DTIC