A STUDY
OF ROUTING ALGORITHMS FOR
PRINTED CIRCUIT BOARDS AND VLSI

by

XUAN KONG

A thesis

presented to the University of Manitoba

in partial fulfillment of the requirements

of the degree of

MASTER OF SCIENCE

in the Department of Electrical Engineering

Winnipeg, Manitoba, 1986

ISBN 0-315-33950-0

A STUDY OF ROUTING ALGORITHMS FOR PRINTED CIRCUIT BOARDS AND VLSI

BY

XUAN KONG

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

ABSTRACT

The routing problem for both printed circuit boards and VLSI are addressed in the thesis. Different routing algorithms applicable to printed circuit boards are discussed and evaluated. Using a printed circuit board computer-aided design software package, Optimate™, the relationship between the routing history and the routing performance is investigated by experiments. Results show that a certain type of routing history can generate better routing results. Empirical formulae are developed for predicting routability based on the information given by a placement configuration. Pertinent examples show that the formulae can predict routability well.

A new channel router is developed particularly suitable for VLSI layout. The new router can generate optimum results for one class of problems whose lower bound of track number needed is determined by the maximum ordering number rather than the maximum density number. It is shown that other channel routing algorithms usually cannot generate optimum results for the class considered. This thesis documents routing results from thirteen examples, ten of which produce optimum results. The algorithm is coded in PASCAL. Both the PCB and VLSI results were obtained on an Apollo DN660.

ACKNOWLEDGEMENTS

I wish to express my sincere thanks to my advisor, Dr. Witold Kinsner, for his excellent guidance, endurable motivation, consistent and generous support throughout the research work, for his major contribution to the writing of this thesis, and for suggesting the topic of this work.

The valuable help from the Industrial Applications of Microelectronics Centre Inc., Winnipeg, Canada is acknowledged. Special thanks are given to Mohinder Kauldher, Scott Hanford, and Joe Silva for their help in the experiments with the computer-aided engineering of printed circuit boards.

I also wish to gratefully acknowledge the partial finacial support from Sichuan University of the People's Republic of China, the National Sciences and Engineering Research Council (NSERC) of Canada and the Manitoba Research Council through the Strategic Research Support Program.

Finally, I would like to thank my parents for their love and understanding; nothing can come true without their encouragement and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND DEFINITIONS

The definitions provided here are based on the work done in [2].

Blind Via	A blind via is a via penetrating at least two layers, including one and (only one) outer layer. (See Layer, Outer Layer and Via.)
Buried Via	A buried via is a via penetrating at least two inner layers and no outer layers. Buried vias are not accessible from the outer layers on a PCB or on an SMB, thus it is very difficult to test them. (See Inner Layer, Layer, Outer Layer and Via.)
CAD	Computer Aided Design.
CAE	Computer Aided Engineering.
Component Pad	A component pad is the <u>physical</u> representation of a terminal on a net. For a PCB, the component pad is located on the solder layer, and for an SMB, the component pad is located on the component layer. (See Net and Terminal.)
Component Layer	A component layer is the outer layer where components are placed. A PCB has one component layer. An SMB may have either one or two component layers. (See Layer and Outer Layer.)
Connection	A connection is what makes different terminals on a net electrically common. (See Net and Terminal.)

DNR graph — Directed Net Relation graph.

EVMR — Efficient Variable-cost Maze Router.

Feed-through — A feed-through is a hole passing through all the layers of a PCB. Feed-throughs enable the components to be inserted into them and fixed on the board. Every feed-through is attached to a component pad located on only one layer of the board. This is in contrast to a via which has traces attached to it on at least two layers. If the component pad is not attached to a trace located on the solder layer, then the feed-through must also be attached to that trace located on another layer. In this case, the feed-through also plays the role of a via. (See Trace and Via.)

Feed-through Void

A feed-through void is the area on a PCB or an SMB where feed-through and vias are forbidden. Traces, however, may be allowed there. (Also see Routing Void.)

GA — Gate Array.

Gate Array Design

Gate array design methodology is based on a regular size of transistor cluster (cell) containing logic gates or components that are predefined up to the final stages of wafer processing. Gate array includes digital, analog and mixed digital/analog circuits.

Horizontal and Vertical Direction on the Layer

For PCB routing, a common approach is to give each layer a preferable direction parallel to the board sides. The layer with horizontal (vertical) preferable direction is called the horizontal (vertical) layer and the majority of connections on that layer are horizontally (vertically) routed. (See Layer.)

IC	Integrated Circuit.
Inner Layer	An inner layer is the layer enclosed by two outer layers. (See Layer and Outer Layer.)
Layer	A layer is a plane on which connections can be routed to make different terminals electrically equivalent and to form wires (defined as sets of component pads, traces, and vias). Traces on different layers are connected through vias. (Also see Outer Layer, Inner layer, Component Layer, and Solder Layer.)
Lead	A lead is usually referred to as the point where a wire is attached to a passive component such as a resistor or capacitor, or an IC with a lead package. (Also see Pin.)
LSI	Large Scale Integration.
Net	A net is a set of terminals and the connections which make those terminals electrically common.

	According to the function of the net, we call them the power net, signal net, etc. A net is realized as a wire through the process of routing. (Also see Wire.) Net={Terminals, Connections}.
Outer Layer	An outer layer of a PCB or an SMB is the layer accessible from outside of the board. (Also see Inner Layer.)
Overflow	An overflow is a connection which can not be routed under the given constraints.
PCB	Printed Circuit Board, also called Printed Wiring Board (PWB).
Pin	A pin is usually referred to as the point where a wire is attached to an IC chip. (Also see Lead.)
Placement	Placement is the process of arranging all the components within a two-dimensional area such that the configuration of the placement will facilitate the routing process.
Rats nest	A rats nest is a set of lines (abstract connections) and the terminals connected by those lines. A rats nest is converted into a set of wires by the process of routing. (Also see Net.)
RI	Routability Indicator.
Routing	Routing is the process of converting the intended connections (usually represented by rats nest) into wires within a two-dimensional multilayer region, provided that certain mechanical and electrical

constraints are satisfied.

Routing Void — A routing void is the area on a PCB or an SMB where the traces are not permitted to be placed.

SC — Standard Cell.

SMB — Surface Mount Board.

Solder Layer — A solder layer is the outer layer where pins of components are soldered to the component pads. For a PCB, the solder layer is located opposite to the component layer, and for an SMB, the solder layer is on the same side as the component layer.

Standard Cell Design

Standard cell design methodology is based on a library of predesigned (in shape and size) functional cells, called standard cells, that may be equivalent to standard SSI and MSI logic families (from a single inverter to an ALU and more). A full mask set is required to manufacture chips based on the standard cell methodology. Stardaed cell includes digital, analog and mixed digital/analog circuits.

Via — A via is a physical hole passing through a PCB or an SMB or an IC. Vias make connections between traces on at least two different layers, thus contributing to the creation of wires. Since vias do not carry any mechanical loads, they may be smaller than the feed-throughs. (Also see Blind Via and Burried Via.)

Terminal	A terminal is the end point of a connection. On a <u>physical</u> board, it is represented by a component pad. (Also see Component Pad.)
Trace	A trace is the <u>physical</u> representation of a connection which makes different points in a circuit electrically common on a single layer. Traces, defined on layers, together with vias and component pads constitute wires. (Also see Connection.)
VLSI	Very Large Scale Integration.
Wire	A wire is the <u>physical</u> realization of a net which makes different points in a circuit electrically common. A wire includes at least one trace and two component pads. If the traces are located on different layers, the wire also includes at least one via. For a PCB, the component pad is attached to a feed-through. (Also see Net.) Wire={Component Pads, Traces, Vias}

CHAPTER I

INTRODUCTION

The design of any electronic circuit involves two major phases: namely, electronic design and physical layout. In the electronic design phase, the designer develops the circuit which is able to complete the desired functions by using abstract functional blocks (such as NAND and/or NOR gates, registers, and latches). The first phase usually results in a schematic diagram. The physical layout phase can then be accomplished either manually or with the help of a computer. In the computer-aided layout method, the first step requires a process called schematic capture to convert the schematic diagram into a computer readable form. The schematic capture stage is followed by the conversion of the electrical representation of the circuit into the physical representation of the circuit. The conversion may consider physical constraints such as ringing, crosstalk [1] and heat dissipation. The physical representation may have different forms, including the printed circuit board (PCB), surface-mount board (SMB), gate-array (GA), standard cell (SC), and full custom [2].

In circuit design, there is the continuous challenge of increasing circuit density. The increasing density of circuits makes the circuit layout almost impossible to be done by a human without the aid of a computer. The problem is compounded by the requirement of very-high quality circuit layout which cannot be reached by human designers, either. Thus, a large amount of effort has been devoted to providing computer aids to assist the human designer with the layout problem [3]. Such aids fall into the categories of computer-aided design (CAD) and computer-aided engineering (CAE).

1.1 Partitioning of Physical Circuit Design

The layout problem can be described as follows: Many modules (components) are to be placed in a given area so that they do not overlap (placement), while at the same time, certain points (called terminals) must be connected by mutually noninterfering wires (routing). More precisely, for a given circuit and physical size of all modules, the placement process arranges all the modules within a given two-dimensional area in such a way that the locations of the placed modules facilitate the subsequent routing process. For a given circuit and placement configuration, the routing process converts the intended electrical connections (usually represented by a rats nest) into physical connections (such as the horizontal and vertical traces in PCBs) within a two-dimensional multilayer region, provided that certain constraints are satisfied. Since the module placement significantly affects the performance of routing, ideally the placement and routing processes should be done simultaneously in order to achieve optimum results. But due to the complexity of each step, it is almost impossible to do so at the present. Thus, the two processes are commonly treated iteratively. That is, if it is found that it is very difficult to achieve reasonable completeness of routing, it would be necessary to go back to repeat the placement process, and then do the routing again, based on the new placement of modules.

1.2 Classification of Routing Algorithms

Many routing algorithms have been developed in the past three decades. The first recognized algorithm was developed by Lee [4] in 1961. The algorithm can find an optimum path connecting two different points on a plane, according to a certain cost function. Since then, many other algorithms based on the Lee algorithm have been developed in order to achieve better results in terms of memory requirement, time, or routing patterns. All of those algorithms are classified as <u>maze routing algorithms</u>. In this thesis, maze routing is addressed in detail because of its applicability to printed circuit board design.

Another class of routing algorithms is the <u>line routing algorithm</u> developed originally by Hightower [5]. The line routing algorithm takes less time than the Lee algorithm but it may not find a solution which actually exists. Other similar algorithms are described in [6] and [7]. It should be noted that since both the maze and line routing algorithms trace the connecting paths one at a time, they are called <u>sequential</u>.

In recent years, another class of routing algorithms has been developed. These algorithms are classified as <u>channel routing algorithms</u>, and are widely used, especially in LSI and VLSI circuit design [8]. This kind of router requires two steps to complete its operation. In the first step, the router divides the available routing area into many rectangular sub-routing areas called channels. Then it decides which channels should be used for routing each net. Since this is a global routing procedure, the routing density can be evenly distributed to increase the routability. The second step is to route locally within the channel to determine the detailed position of each connection within a channel. This may result in the reduction of both the memory needed and the time consumed for routing. For these reasons, channel routing algorithms constitute the second major topic of this thesis.

Still another class of algorithms is emerging [9-11]. These algorithms use concepts from artificial (computational) intelligence, such as the expert system concept, which consider the history of the process itself or the histories of many processes from the past, and heuristics based on the process history. The algorithms with the heuristics are designed to produce layouts better than those without them. The heuristics can be applied to both the sequential (maze and line) and global (channel) routers. For this reason, they can be called routing algorithms with history.

Although many routing algorithms were first developed for printed circuit board design, it has been shown that these algorithms can also be used successfully in LSI and VLSI circuit design using gate-array and standard-cell circuit design methodologies. The Lee algorithm is still the most common algorithm implemented in CAD layout tools because it can always find a connection path, if such a path exists. This is especially true for printed circuit board design. According to a survey of printed circuit board CAD systems [12], among 26 systems from 22 vendors, 21 systems employ Lee's routing algorithm or its variations.

With the emergence of VLSI, researchers started designing and building special purpose chips. A hardware router described in [13] and [14] serves as an example of such a kind of chip. Most of those hardware routers are based on the Lee algorithm. To find a connection, <u>the maze hardware router</u> requires time O(d), where d is the length of the connection, while the software implementation of the maze router usually takes time $O(d^2)$,

1.3 Objectives of the Thesis

The objectives of this research work are:

a) To survey and evaluate the existing maze and channel routing algorithms;

b) To find a relationship between the routing histories and routing results: obtained from a commercially available PCB CAE software;

c) To find a relationship between the placement and routing processes through a study of how routability is influenced by different placement configurations and to find a possible routability indicator; and

d) To develop a new channel routing algorithm in order to achieve optimum results for at least one class of circuit layout problems.

1.4 Structure of the Thesis

The first part of the thesis deals with the routing problem of printed circuit boards and the second part deals with the problem of the LSI and VLSI layout. In the first part, Chapter II describes and evaluates the existing routing algorithms for printed circuit boards. For an improved version of Lee's routing algorithm, the effects of different routing histories on routing results are discussed in Chapter III. Then, in Chapter IV the routability problem is addressed and a method of predicting routability based on the placement results is introduced.

The second part of the thesis addresses the channel routing algorithm which is the most common routing strategy for LSI and VLSI layout. A new algorithm for channel routing is then presented in Chapter V. Chapter VI presents the experimental results on the new track assignment algorithm as well as the discussion on the results. In Chapter VII, its performance on a certain class of problems is compared with the performance of other channel routing algorithms and the reasons for its superior results over others are discussed. Conclusions for both the printed circuit board routing problem and the LSI and VLSI routing problem are drawn and suggestions for further studies are given in the last chapter of the thesis.

CHAPTER II

LEE'S ROUTING ALGORITHM AND ITS VARIATIONS

2.1 Lee's Path Connection Algorithm

As stated in Chapter I, Lee's algorithm [4] is the most common routing algorithm for printed circuit boards (PCBs). The algorithm is designed to solve the following problem: How to find an optimum path, connecting two different points on a plane defined by the PCB boundary, according to a certain cost function. The cost function includes the length of a path, crossovers with existing paths (if permitted), distance to other paths, and the number of corners.

Lee's algorithm has the following properties: (i) It always finds a path if one exists, and (ii) The path it finds always has the minimum possible cost.

Lee's algorithm can work with any monotonic path cost function (a monotonic function is a function whose first derivative does not change its sign). Such a monotonic function can also be a set of monotonic functions, F_i, represented by a vector $\mathbf{F}=[F_1,F_2......F_n]$.

Assume that a routing area is divided into a finite number, N, of subareas called cells labelled C_i. Further assume that the cells where routing is forbidden are tagged to prevent them from being used in the routing process. The remaining cells can be used to build up a path to connect two particular cells. The size of the cells is determined by computational and manufacturing considerations. An example of the routable cells and routing voids is illustrated in Fig. 1.

Each cell has an associated cost. The cost of a cell is a number representing either a single quantity or a combination of quantities such as the distance between the cell and its adjacent cells and the directions in which the cell is entered and exited. The cost of a cell C_i is denoted by $f(C_i)$. The cost function of a path is denoted by F(p), where p is the path which consists of the starting cell S and a set of cells C_1, C_2, ..., and C_n which are adjacent to one another. The cost

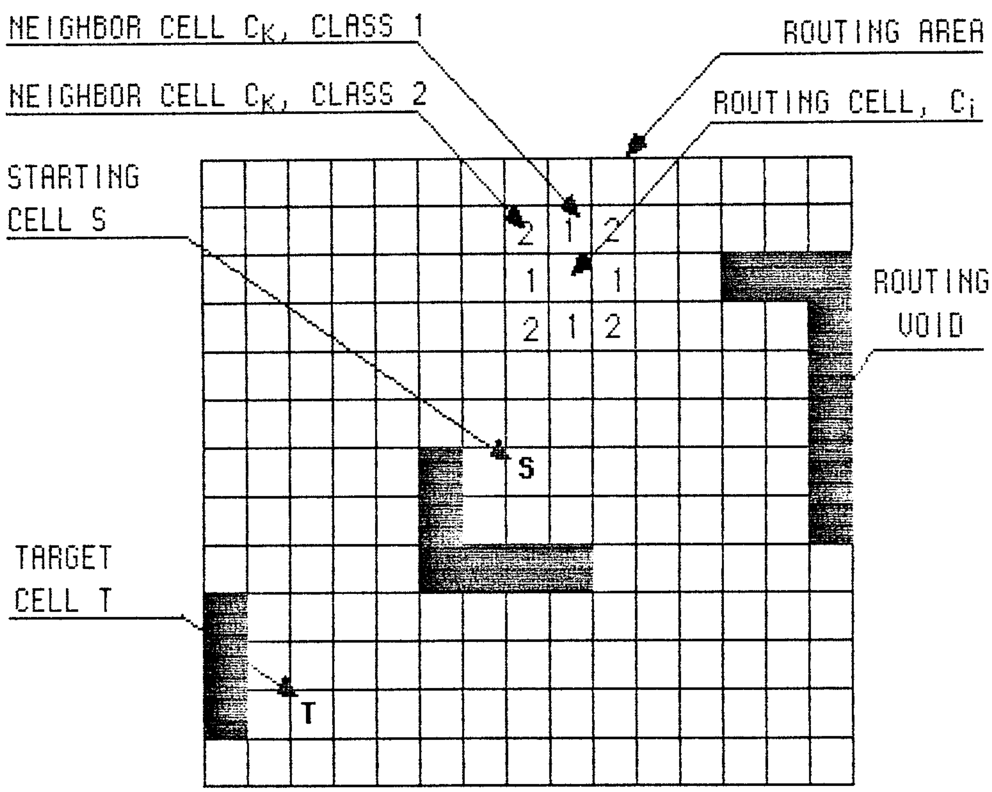


Fig. 1. A routing example with cell S to be connected to cell T

function is defined as the sum of the cost of each cell on the path, and can be expressed as

$$F(p)=\sum_{i=1}^{n} f(C_i) \tag{2.1}$$

for a path defined by the following set of (n+1) cells

$$p=\{S, C_1, C_2, ..., C_n\} \tag{2.2}$$

If two particular cells are to be connected, it is desirable to know whether any possible path exists to connect them. If it exists, can the path which has the minimum cost be found? The Lee algorithm can answer the two questions.

Let us first introduce several definitions. One of the cells to be connected is selected as the <u>starting</u> cell, and the other as the <u>target</u> cell. The process of finding the path between the starting and target cells requires labelling cells into non-visited and visited ones. A routable cell is labelled as non-visited if it has not been used in generating the minimum cost path connecting the starting and target cells. A routable cell is labelled as visited if it has already been used in generating the minimum cost path connecting the starting and target cells.

Furthermore, any routable cell C_i has at least one neighbor cell C_k, defined as the cell adjacent to the cell C_i. For example, Fig. 1 shows two classes of adjacent cells: (i) the nearest neighbors along the horizontal and vertical directions, marked 1, and (ii) the neighbors along the diagonal directions, marked 2. If the class 1 neighbors are considered in the algorithm, the resulting paths have only 90-degree corners along the horizontal and vertical directions, without any 45-degree corners. This definition of cell neighborhood emphasizes the <u>local</u> (cell-wise) nature of the Lee procedure rather than a <u>global</u> algorithm.

Since the process of finding a minimum cost path occurs in stages, one cell is added to the path at a time. The most recent cell found along the minimum cost path is called the <u>frontier cell</u>, and it remains the frontier cell until all of its neighbors are labelled as visited.

Finally, let us define two cell lists: L and L_1. The lists are variable in length from one stage of the path-finding procedure to another. The cell list L contains frontier cell(s) located along the minimum cost path. The path finding process begins with L containing the starting cell only. The cell list L_1 contains the cells which are adjacent to the cells in the cell list L and are labelled non-visited. When the path is restricted to the horizontal and vertical directions with 90-degree corners, the adjacent cells belong to class 1; when the path is allowed to have 45-degree corners, the adjacent cells include the cells of both class 1 and class 2.

Based on the above definitions, the principal steps of the algorithm can be stated as follows:

[Lee Algorithm]:

Step 1) Initialization: L={starting cell}; $L_1=\emptyset$; label all cells non-visited to indicate that they can be used in generating the minimum cost path.

Step 2) For each cell C_i in the list L, do the following:

Add each non-visited neighbor C_k of the cell C_i into the list L_1. Calculate the cost function of the path containing this cell C_k.

Step 3) Among all the new path cost functions, find the one (or ones) which has (have) the minimum value. Then, the cell(s) in list L_1 which result(s) in this function value can be identified. Add those cells to the list L, label them visited, and record the direction in which they are entered. If any of those cells in L is the target cell, the path finding process is completed.

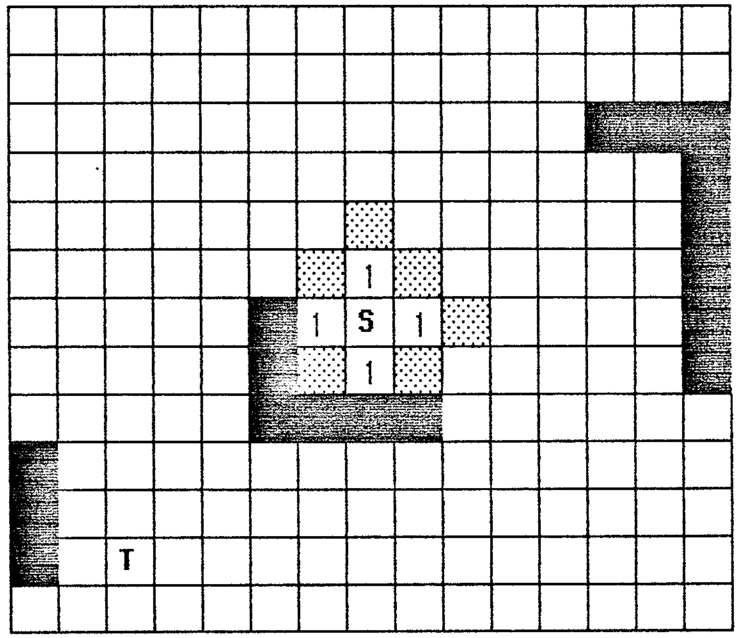
Step 4) Delete from L any cells whose neighbors have all been visited, and clear L_1.

Step 5) If L is empty, no path exists. Otherwise repeat from step 2.

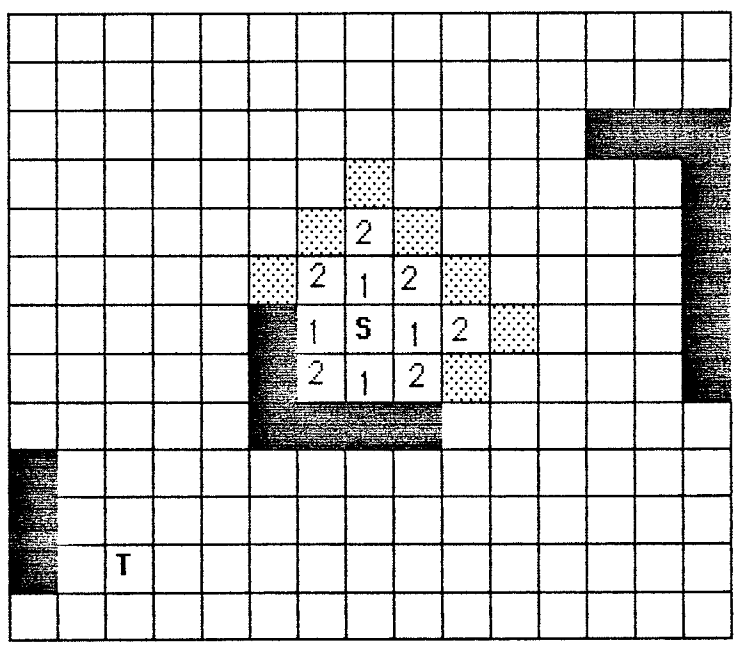
□

If the iteration exits at Step 3, then a path with the minimum cost has been found. Tracing back the path can be done by starting from the target cell and following the directions recorded for each cell to the starting cell.

As shown in Fig. 1, a connection between S and T needs to be found. For simplicity, consider the path length as the cost of the path. That is, each cell has a unit cost (except for the cells in routing voids which have infinite costs). According to the algorithm, the four neighbor cells (marked 1 in Fig. 2-a) of the starting cell are expanded and labelled visited during the first expansion stage, and they are the cells in the cell list L after the first expansion stage. Furthermore, the dotted cells in Fig. 2-a are the cells in the cell list L_1 during the second expansion stage. Similarly, the cells marked 2 in Fig. 2-b are the cells in the cell list L after the second expansion stage, and the dotted cells in Fig. 2-b are the cells in the cell list L_1 during the third expansion stage. At the end of the path-finding process, the cells are labelled as shown in Fig. 2-c, where the marked cells now represent the three equivalent minimum length connection paths. If one chooses the minimum number of corners on the path, then the path shown in

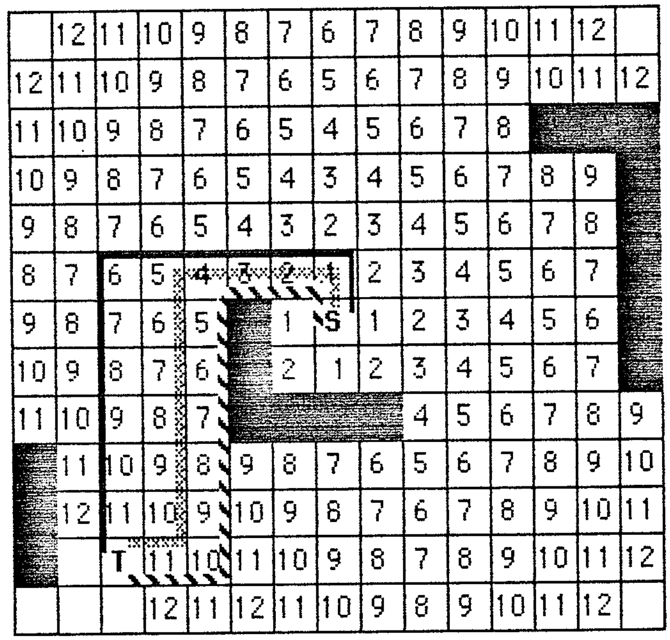


(a)

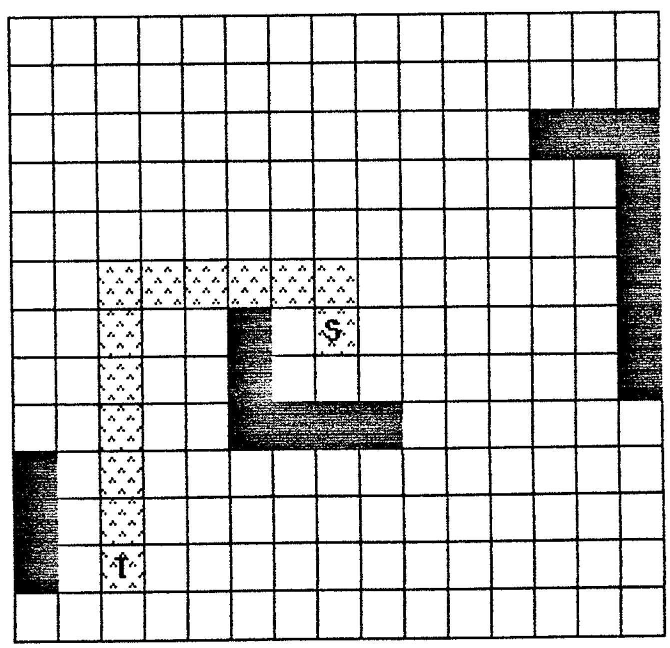


(b)

Fig. 2. Routing results of the Lee algorithm.
(a) Expansion result after the first stage
(b) Expansion result after the second stage



(c)



(d)

Fig. 2. Routing results of Lee algorithm (continued).
(c) Final expansion result
(d) Routing result

Fig. 2-d is the optimum connection path.

The order of element cost functions in the function vector **F** determines the priorities of the corresponding criteria. So, the criterion represented by F_1 in vector **F** has the highest priority and F_2 has the next highest priority, and so on.

Lee's algorithm uses a rather modest amount of storage. At any given time only the cells on the list L and L_1 are stored, along with the information concerning whether a cell has been visited and what its minimum-cost predecessor had been--the direction from which the cell is visited (this is called a cell map).

Lee's algorithm has several limitations. The very important condition is that the vector **F** must be monotonic. Fortunately, the layout problems considered can always satisfy this condition. The algorithm is very time-consuming because it can only route one path at a time. Also, once a path is routed, it becomes an obstacle for paths to be routed later. Thus the routing order seems to be very important.

Since publication of Lee's algorithm, many modifications of this algorithm have been made. Some of the modifications are discussed in the following sections.

2.2 Improvement 1: Reducing the Cell Map Storage

When Lee's algorithm is applied to a circuit board, the number of cells may be extremely large, depending on the smallest recognizable feature on the board, such as the trace width. Therefore, it is desirable to minimize the amount of storage needed for each cell. The minimum information in the cell map must include: (i) a means of distinguishing between obstacle cells (which cannot be used to generate a new path) and routable cells (which can be used to generate a new path), and (ii) a means of retracing the connection path from the target cell back to the starting cell.

Akers [15] presented a method to encode the cells by using 2 bits for each cell:

00=routable

01=reached, with trace bit 1

10=reached, with trace bit 2

11=obstacle

If the shortest path between two points S and T is desired (see Fig. 1), with the cost of the path being equal to the number of cells in that path, the following procedure can be used (see Fig. 3). A "1" is entered in each routable cell which is the neighbor of the starting cell (S), a second "1" is now entered in each routable cell which is the neighbor of the cell containing one of these "1". Next, a "2" is entered into each routable neighbor cell which has the second "1", and so on until the target cell is reached (the path length can be counted by a single counter). For back-tracing the path, if the target cell T is reached by a 2 preceeded by another 2 (as the example shown in Fig. 3), then it is traced back by following the sequence 2(T), 2, 1, 1, 2, 2, 1, ..., S (starting cell). If, in the back-tracing process, there are several cells eligible to be the next cell to be added to the path, then the one which can keep the direction of the path unchanged should be selected.

Akers' method cannot, however, be used if the cost

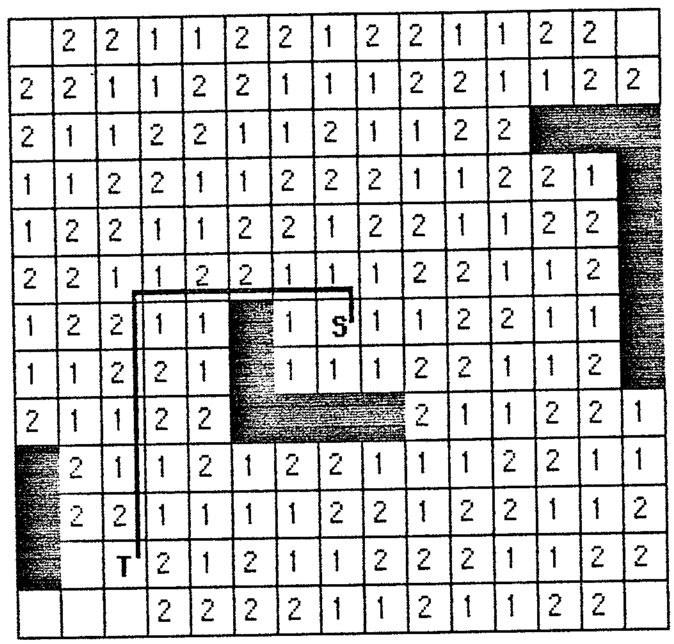


Fig. 3. Routing result of 2-bit coding method

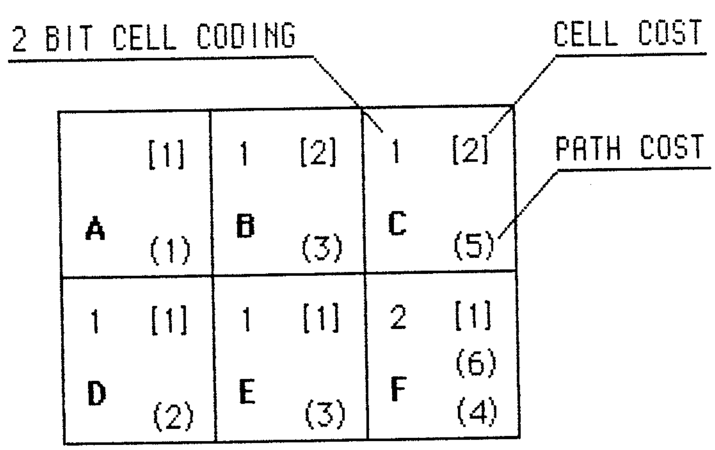


Fig. 4. An example of Akers' method failing to find path

cell and F is the target cell. It is now impossible to tell whether C or E is the prodecessor of F (it should be E). This example shows that Akers' method is limited to the case where the cost function is the length of the path. Since the routing length is a very important criterion in the routing problem, it is still a very powerful strategy for implementing Lee's algorithm. For a general cost function, a 3-bit cell coding method has been introduced by Rubin [16].

2.3 Improvement 2: Reducing the Area Expanded

Since Lee's algorithm (and its variations) examines many cells which are not on the way to the target, its speed is limited. All cells reachable with a cost less than the cost of reaching the target cell have been expanded, including those directly away from the target cell.

In [16], Rubin presented three methods which aim at reducing the number of cells expanded. It can be shown that for Lee's original algorithm with the Manhattan distance of d between the starting and target cells, the number of expanded cells is about $2d^2$ before the target cell is reached by expanding from the starting cell. (Manhattan distance between two points (x_1, y_1) and (x_2, y_2) is measured by the sum of the absolute values of (x_1-x_2) and (y_1-y_2).)

Pohl [16] developed a two-ended procedure for reducing the number of cells expanded. The main strategy of this method is to expand both the starting cell and the target cell such that these two expansions can be terminated when they meet in mid way. The expanded cell number will be reduced to $2*(2*(d/2)^2)=d^2$. That is half of what the original Lee algorithm needs. There is a drawback of this procedure. It is necessary in this procedure to distinguish between the cells expanded from the starting cell and those expanded from the target cell. Then, when a cell which has been previously visited is encountered, it is possible to determine whether this is a path doubling back upon itself or the completion of the search. Thus, storage for each cell will be increased by one bit. Figure 5 gives the expanding result of this searching strategy.

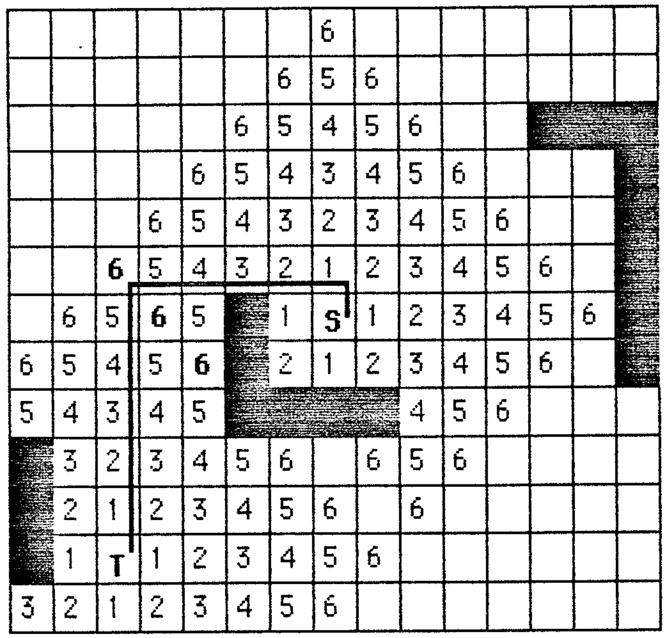


Fig. 5. Routing result of two-ended expansion method

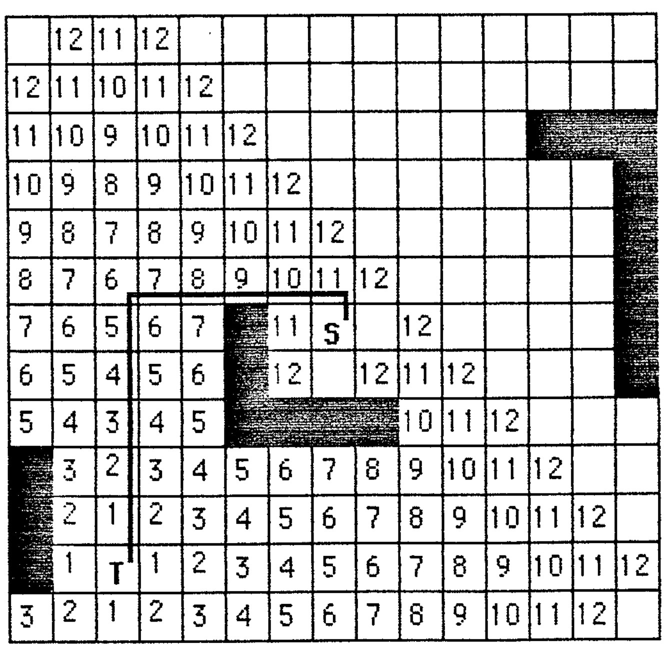


Fig. 6. Routing result of corner-end expansion method

Another method of speeding up the original Lee algorithm to reduce number of cells expanded is to choose one endpoint, which is nearer to one of the four corners of the available routing area, as the starting cell to be expanded [2]. Then, potentially fewer cells directly away from the target cell will be expanded because of the blockage of the natural edges of the routing region. The expanding result of this method is shown in Fig. 6.

Since the Manhattan distance gives the exact measure of how far apart two given cells may be in a rectangular grid, Rubin [16] introduced a predicted path cost function **H**, which could guide the expansion direction.

Let m_{ij} represent the Manhattan distance from cell C_i to cell C_j, and d_{ij} the distance from cell C_i to C_j along the minimum cost path. Clearly, condition $d_{ij} \geq m_{ij}$ always holds true. Assume that the path cost function $\mathbf{F}=[F_1(p),\ F_2(p),\ \ldots,\ F_r(p)]$ has components of the form

$$F_k(p) = g_k(p)+a_k*d_{si}, \qquad a_k \geq 0 \qquad (2.3)$$

where $g_k(p)$ is a monotonic path function, d_{si} is the distance between the starting cell C_s and the cell C_i which is added to the path most recently, a_k is a nonnegative constant, and not all a_k are zero. Since in routing, the path length is usually considered as an important factor, the path cost function **F** can always satisfy the above conditions. The predicted path cost function is constructed as

$$H_k(p)=F_k(p)+b_k*m_{it} \qquad b_k\le a_k \qquad (2.4)$$

where m_{it} is the Manhattan distance between the current cell C_i and the target cell C_t, and b_k is a nonnegative constant. It can be shown [16] that the algorithm with the predicted path cost function can always find a path whose **F** cost is minimum whenever such a path exists. It has also been shown [16] that the set of cells expanded with a predictor is a subset of the cells expanded without the predictor. Figure 7 shows the expanded area for the example shown in Fig. 1.

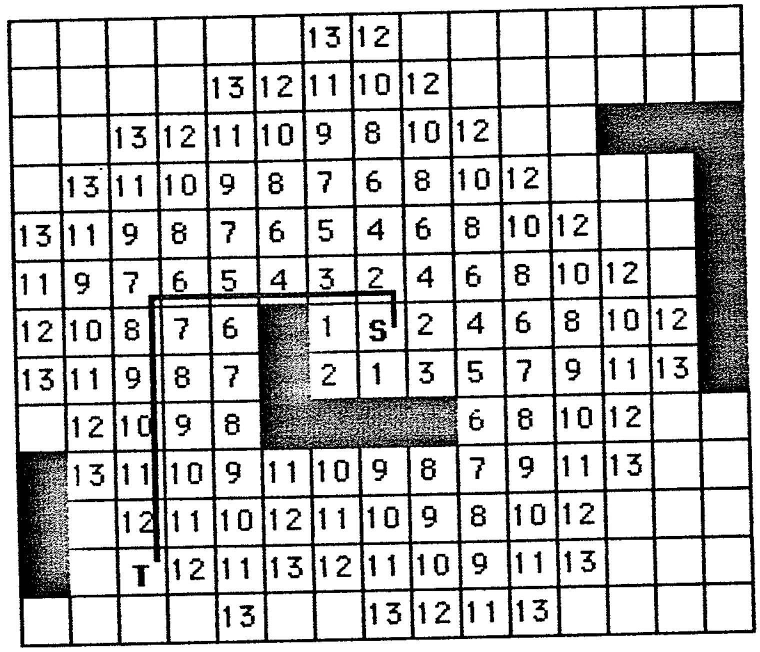


Fig. 7. Routing result of Rubin's method

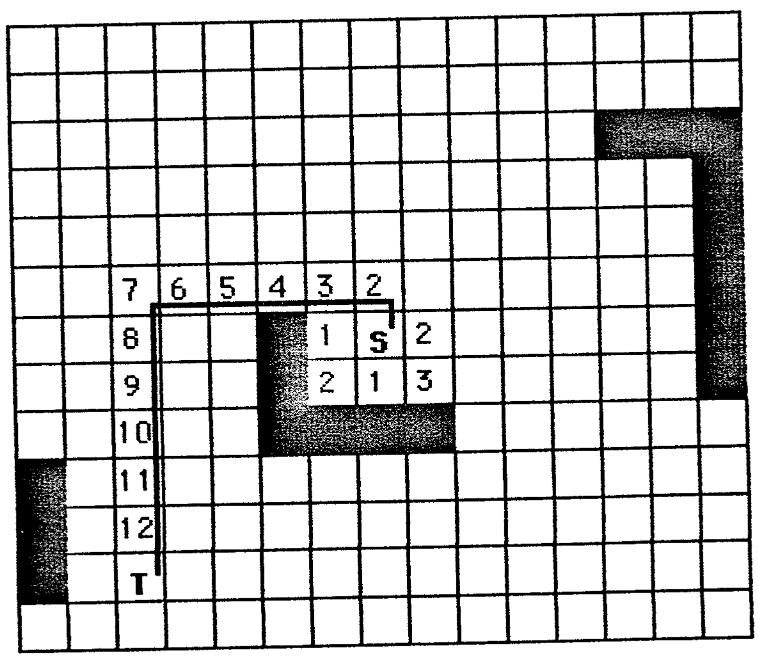


Fig. 8. Routing result of depth-first function guided expansion

Since there may be many synonymous cells (synonyms) of equal cost of the target cell in each path routing process, and with the use of the predictor function, the number of synonyms can increase considerably. Thus, some kind of order for the expansion of synonyms can be expected to reduce the number of cells expanded considerably.

Suppose that after expanding the starting cell, one of its neighbors nearer to the target is chosen. This defines a direction towards the target. If this cell is expanded next, then its neighbors can be generated in an order so that the next cell to be considered is the neighbor in the same direction towards the target. This procedure can be continued until the opposite boundary of the primary rectangle (primary rectangle is the rectangle whose one pair of opposite corner are the starting and target cells respectively) is reached. If the order for choosing the next cell to be expanded requires that all neighbors of the last-expanded cell are checked first, then at this point the single neighbor which is nearer to the target cell will be chosen, which establishes the preferred direction for the remaining cells. It was shown in [16] that this so called depth-first search will find minimum-cost paths.

By using the depth-first search strategy, the following algorithm can be formulated:

[Depth-first Searching Algorithm]:

Step 1) Place the starting cell(s) on the cell list. Set the direction entered to 0 and the cost threshold to 0.

Step 2) Find the **last** cell C in the cell list such that the cost of the path which includes the new cell C equals the threshold.

Step 3) If none, set the threshold to the least cost of the path which contains one of the cells on the cell list, and repeat Step 2.

Step 4) If C is the target cell, go to Step 11.

Step 5) If C is labelled as visited, go to Step 8.

Step 6) Otherwise, let d be the direction from which the cell was entered (visited), and consider its neighbors in directions d+1, d+2, d+3, d+4 (taken mod(4)).

a) If the neighbor was previously visited, or it is an obstacle, skip it.

b) Otherwise record its predicted cost and direction entered at the **end** of the cell list.

Step 7) Record the direction of cell C in which it was visited, and label it as visited.

Step 8) Delete C from the cell list.

Step 9) If any other cells exist in the list, repeat from Step 2.

Step 10) No path exists. Stop.

Step 11) Trace back along the path to its starting cell. Exit.

□

It should be noticed that if the neighbors of cell C have the same cost, then the neighbor entered by the same direction as C was entered will be expanded. This is because the last admissible neighbor of cell C has the direction d+4 which is the same as C's direction d due to the modulo 4. Therefore, the algorithm always tries to expand the least cost cell which can preserve the direction of the expansion, if it is possible. The expansion result is shown in Fig. 8.

Thus the major algorithms proposed in order to reduce the number of cells expanded have been presented. Table I presents a comparison of the number of cells expanded by the above-described

TABLE I
Comparison of number of cells expanded by different methods

Expanding Methods	Number of cells expanded
Lee's original algorithm	171
Two-ended expansion algorithm	101
Corner-ended expansion algorithm	116
Rubin's expansion algorithm	121
Depth-first function guided expansion algorithm	18

methods for the routing problem shown in Fig. 1. It is seen that the depth-first function guided expansion method produces the fewest number of cells expanded.

It has been shown in [17] that by using Rubin's strategy, if there is a small blockage near the target cell, the whole routing area could be filled (see Fig. 9, where S and T are the starting and target cells, respectively). So in [17], Korn introduced another method to get an even smaller number of cells expanded by sacrificing the guarantee of the path having the minimal cost. Being different from Rubin's method, the Efficient Variable-cost Maze Router (EVMR) constructs the prediced path cost function by weighting the distance from the target cell **more heavily** than the distance from the starting cell. That is, in a predicted path cost function:

$$H_k(p)=F_k(p)+b'_k*m_{it} \qquad b'_k>a_k \quad (\text{in Rubin's method } b_k\le a_k) \qquad (2.5)$$

where m_{it} is the same as in Eq. 2.4. Figures 9 and 10 show the results of routing the same example by using Rubin's method and EVMR, respectively. The reduction in the number of cells expanded is obvious.

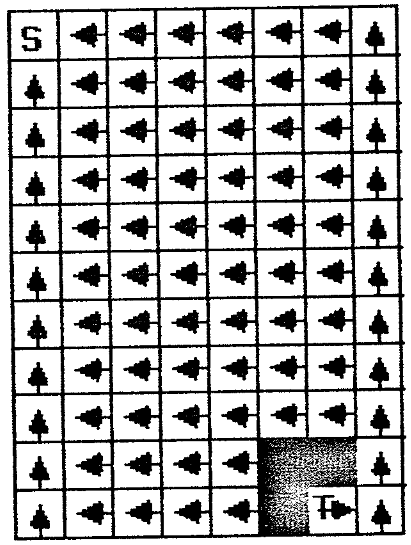


Fig. 9. A counter-example for Rubin's expansion method

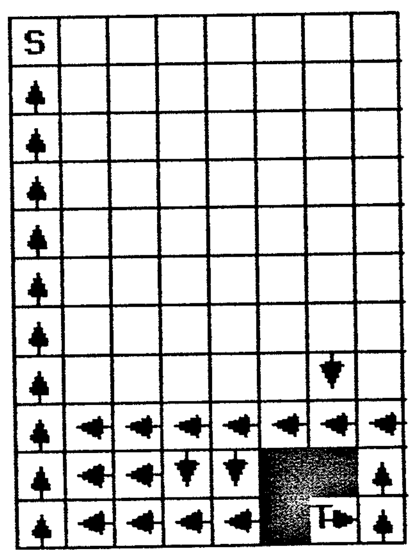


Fig. 10. Routing result of EVMR method

It is shown in [17] that the actual path cost is at or near the minimum despite the overpull (Korn calls $b'_k > a_k$ as overpull). The path cost is just one direction-change cost more than the minimal cost in the example.

Another strategy for reducing the expanding area is due to Hoel [18]. In his paper, a stack is employed to store the cells in the cell list. (The same cell list as in depth-first searching algorithm.) The property of last-in-first-out of the stack has a similar effect to taking modulo 4 of the direction in the depth-first searching algorithm.

2.4 Routing with Variable Searching Space

One key factor that makes Lee's routing algorithm slower than other routers is the large searching area for finding the potential path. Even though the seaching area can be greatly reduced by using the strategies introduced in the last section, it is still desirable to reduce the searching area further. The following three methods can be adopted to serve this purpose.

2.4.1 Single Rectangular Frame Technique[2]

Though it is possible for a path connecting two points on the board to lie anywhere on the board outside the rectangle defined by the two points to be connected, the rectangle is the most likely region where the path can be found [2]. The rectangle is defined by the two points either exactly (these two points serve as either the upper left and lower right corners or upper right and lower left corners of the rectangle) or it is slightly larger than the smallest rectangle. For example, assume the two points to be connected have the Manhattan distance (x+y), then the rectangle area for searching the path can be defined as the one shown in Fig. 11 with Δx, Δy being defined as:

$$\Delta x = \max\{x/a,\ y/b,\ \partial\} \tag{2.6a}$$

$$\Delta y = \max\{x/c,\ y/d,\ \beta\} \tag{2.6b}$$

where a, b, c, d, ∂, and β are pre-defined positive integers with the unit of grid length in the x and y directions. The positive integers ∂ and

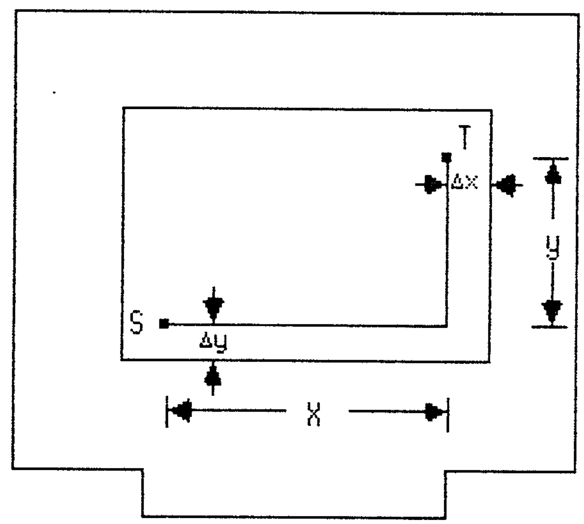


Fig. 11. Single rectangular frame

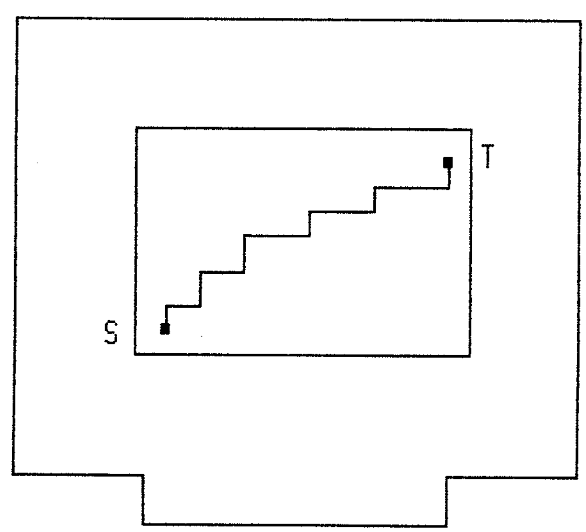


Fig. 12. Undesirable path pattern

β represent the minimum distances between the edges of the rectangular frame and the edges of the smallest rectangle passing through the endpoints S and T. Thus, the maze router is constrained to search the path connecting S and T within the rectangle defined above. If the router fails to find out the path, the rectangle can be enlarged by changing ∂ and β. Within the larger area, the router tries to search for the connection path again. This <u>interactive</u> strategy yields new CAE design options.

2.4.2 <u>Double Rectangular Frame Technique</u>[2]

The above method is simple and effective, but it may result in finding a very undesirable path as shown in Fig. 12. Fortunately, one improved version of the rectangular frame technique can be used to avoid such a zigzag trace. It is called the double rectangular frame technique [2]. This technique provides not only an outer-rectangle to prevent the router from searching the area outside it but also an inner-rectangle to prevent the router from searching the area inside it (Fig. 13). Obviously, this strategy not only avoids the undesirable zigzag path but reduces the searching area as well. Of course, if the

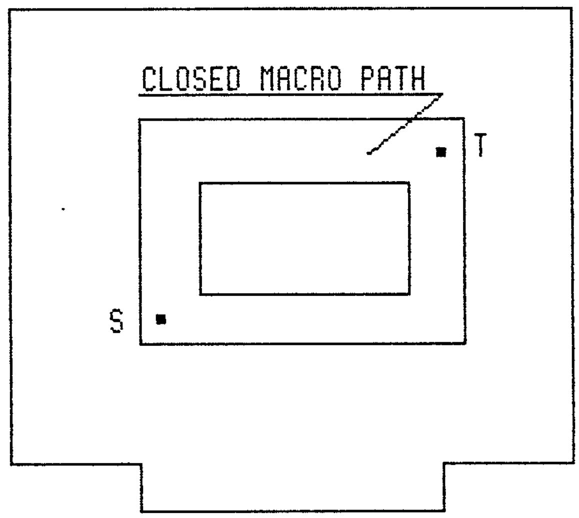


Fig. 13. Double rectangular frame

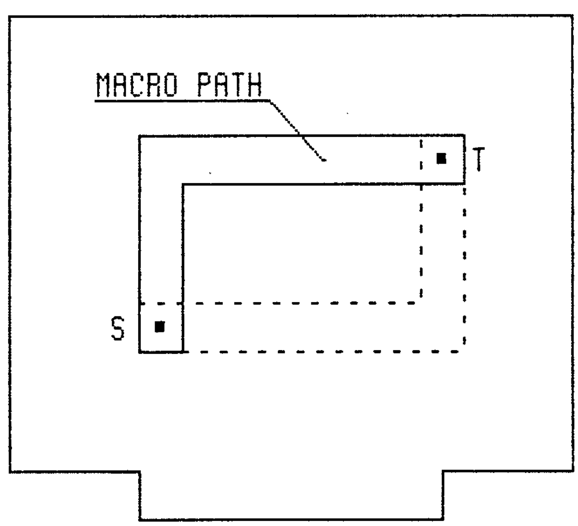


Fig. 14. L-shape routing area

router fails to find the path within the area defined, a larger area can be defined and the router can try to find the path within this area.

2.4.3 Variable Searching Area Restriction Technique

Since the path most likely lies in one L-shape of the double rectangular area, it is possible to reduce the searching area further. In [19], Tada et al. introduced a fast maze router with iterative use of a variable searching area restriction. The algorithm has the following strategy:

(i) The establishment of macro path: To restrict the searching area, a "macro path" is established. It is denoted by an L-shaped frame surrounded with solid or dotted lines as shown in Fig. 14. The choice of solid or dotted lines is made by considering previously routed line density within each frame.

(ii) Iterative use of router with varying macro path width.

The algorithm realizing the above strategy can be described as follows:

[Variable Searching Area Algorithm]:

Step 1) During the first trial, all pairs of points are tried with a narrow macro path width, so that a simple path may be successfully established with short machine time. Complex paths which will require wider path width may fail. This fact is an important feature to reduce computation time and achieve a high completion ratio because of simple paths taking less space.

Step 2) During the second trial, all connections that failed in the above process are attempted again with a wider macro path width.

step 3) In the same manner as mentioned above, successively expanding the macro path width, the router is used for r iterations, where r can be controlled outside of the program.

□

Experimental results of this method together with the single rectangular frame technique method are given in [19]. It has been seen that the routing completeness increases from 83.5% (for the single rectangular frame technique) to 97.4%, and at the same time the computation time of this method is only one quarter of the time required by the single rectangular frame technique. The test board is a printed circuit board with a routing area of 7.9x7.1 inches (236x213 cells), 80 integrated circuits (ICs), 4 discrete components and 686 required connections. The board has two layers for routing.

2.5 Other Routing Algorithms

Apart from the algorithms introduced above, many other strategies have also been developed for the printed circuit board routing. Among them are the Efficient Shortest Path Finding algorithm [20], Parametric Pattern Router [21], Saturated Zone Router [22], and Topographic Router [23]. The Efficient Shortest Path Finding router is based on Lee's algorithm for finding interconnections between points on different layers of the board. The Parametric Pattern Router aims at overcoming the problem of undesirable path shapes produced by Lee's router by using pre-defined path patterns in certain order until a path is found to connnect the starting and target points. The Saturated Zone Router initially partitions the board into regions called saturated zones within which all connections are completed. These saturated zones are subsequently merged into larger and larger saturated zones until the final combination yields a routing for the entire board. The merging of zones is done by routing the disconnections between zones. (A disconnection is where a path is needed to connect points which should be electrically equivalent.) The Topographic Router uses a topograph simulation strategy; that is, the router first assigns the obstacle cells a very high cost and a lower cost to the routable cells near to them and so on. Thus, the minimum cost path found by this router must be away from other paths routed. This can reduce the cross-talk between wires as well as balance the distribution of the routing density.

Still other algorithms based on the Lee algorithm can be found in [24] and [25]. To reduce the number of vias used during routing process, some via reduction techniques were also developed [26, 27]. (A via is a plated-through hole connecting traces on different layers. Also see ABBREVIATIONS and DEFINITIONS.)

As discussed in this chapter, the techniques based on Lee's algorithm have two fundamental limitations: (i) just one path can be routed at a time and (ii) it is difficult to route multi-terminal nets. To overcome these limitations, another strategy has been developed for routing multilayer printed circuit boards [26-40]. The first step of this method is the via assignment. Adding vias on board makes those terminals to be connected become possible to be connected by vertical and/or horizontal paths on different layers. Thus, the objective of via assignment is to convert the original connection problem into simpler connection problems, that is, the problems of connecting points lying on the single row by paths on a single layer. Since an increase in the number of vias on the board results in a larger, more expensive, and less reliable board, the optimum via assignment is the one which uses the fewest vias. This via assignment is followed by a single row routing which is a procedure to route the paths to connnect terminals and/or vias which lie on a single line. The objective of single row routing is to use the smallest area possible on both sides (or one side) of the row to route all paths required for connections.

2.6 Summary

In this chapter, Lee-type routing algorithms have been investigated extensively. Lee's original algorithm can guarantee the optimum solution provided the cost function vector is monotonic. On the other hand, Lee's algorithm becomes less practical as the size of the problem increases. So, other algorithms aiming at reducing the storage requirement and computation time were discussed. Strategies which could avoid the generation of undesired path patterns were also presented. Most of the algorithms presented are based on Lee's algorithm.

CHAPTER III

EFFECTS OF ROUTING HISTORY ON MAZE ROUTING

The previous chapter presented different routing algorithms and their routing performance. The routing results can be affected by many factors, even though we use the same routing algorithm and the same component placement. Therefore, we have investigated the relationship between different sets of parameters on the process of routing, by using a working OPTIMATE™ software package. Since the routing process is seldom completed with just one set of routing parameters, we define the <u>sequence</u> of using different sets of routing parameters for routing a board as the <u>routing history</u> for that board.

OPTIMATE™ is a printed circuit board schematic capture and layout software system developed by Secmai (France) and distributed by Optima Technology Inc. (USA) [2, 41]. It provides a graphical interactive software package for the entire printed circuit board physical design process, with interfaces to standard manufacturing services. The software provides multi-layer routing with a uniform distribution of interconnections over the layers.

The software uses many routing control parameters, including costs for horizontal and vertical routing paths on different layers, costs for 45-degree routing, costs for using vias, maximum number of vias to be used for one connection, maximum connection length, window definition, etc.. Some details of window definition are as follows: we can control routing area for each connection by using window definition, we can define single rectangular window or double rectangular frame and we can also specify the size of each rectangular frame.

3.1 Experimental Results for Small Routing Rectangle

Four boards have been used in the test. One of them (example four) is a surface-mount board and the rest are printed circuit boards. The sizes of the printed circuit boards are in the range of 11.0x4.2 inch to 11.8x6.7 inch and the size of the surface-mount board is 6.8x5.3 inch. The ratio between the board size and the number of ICs on the board is 1.24 $inch^2$/chip to 2.25 $inch^2$/chip for the printed circuit boards and 1.14 inch^2/chip for the surface-mount board. The total numbers of connections to be routed are 354, 342, 425, and 432 for example one, two, three, and four, respectively. All of the boards were designed for practical purposes at the Microelectronics Centre.

To avoid cross-effects of other parameters in our tests, only two major parameters are allowed to change and all other parameters are fixed. The two parameters allowed to change are the maximum number of vias allowed for each connection and the size of the rectangular routing area.

For convenience of description, let the set of routing parameters

$$\{(V_1, S_1), (V_2, S_2), \ldots, (V_n, S_n)\} \qquad (3.1)$$

denote the routing history, where V_n represents the maximum number of vias allowed for one connection, and S_n is the size of the single rectangular routing window. We should keep in mind that the set of routing parameters (V_m, S_m) is applied immediately before the set

(V_{m+1}, S_{m+1}). We should also keep in mind that the number indicating the size of the rectangle is not the actual size but is just equivalent to the Δx and Δy of Fig. 11 (Δx=Δy=S_m grid step, one grid step is equivalent to 25 mils). So the actual rectangular routing area is the rectangle whose four sides are S_m grid steps away from the sides of the inner rectangle. The outermost two points of the net to be connected serve as either the upper left and lower right corners or upper right and lower left corners of the inner rectangle.

For each board tested, we apply the following sets of parameters to route the board (all other parameters are kept constant):

$$H_1=\{(2,6), (2,12), (2,20), (2,30)\} \quad (3.2a)$$

$$H_2=\{(3,6), (3,12), (3,20), (3,30)\} \quad (3.2b)$$

$$H_3=\{(4,6), (4,12), (4,20), (4,30)\} \quad (3.2c)$$

$$H_4=\{(5,6), (5,12), (5,20), (5,30)\} \quad (3.2d)$$

$$H_5=\{(6,6), (6,12), (6,20), (6,30)\} \quad (3.2e)$$

and

$$H_6=\{(2,30), (3,30), (4,30), (5,30), (6,30)\} \qquad (3.3)$$

The routing results of routing history H_6 are then compared with those of routing history of H_1 to H_5. The comparison is based on the following results: (i) the number of successful connections, (ii) the average number of vias used for each successful connection, and (iii) the average routing length of each successful connection (for H_5 and H_6 only).

The selection of routing history H_6 as a reference for those four test boards has the following advantages over using other routing histories: (i) the number of successful connections is increased by up to 3 percent, (ii) the number of vias for each successful connection used is reduced by 3 to 8 percent. The routing history of H_6 results in a slight increment (one percent at the most) of the average routing length for each successful connection, which is undesirable, compared with using the routing history H_5. The routing results for the four test boards are shown in Table II. For each example in Table II, the first

TABLE II. Effects of routing history

			Fix # of vias at..., change size up to 30			Fix size at 30, change # of vias to...			Comparison		
			Number of comp. conn.	No. of vias per conn.	Ave. leng. per conn.	Number of comp. conn.	No. of vias per conn.	Ave. leng. per conn.	Completed conn. Incr.	Vias Incr. per conn.	Leng. Incr. per conn.
Example One	Number of vias	2	296	1.236	1975.14	289	1.239		-1%	0.2%	
		3	311	1.392	2073.16	314	1.331		+1%	-4%	
		4	317	1.448	2098.94	322	1.339		+2%	-8%	
		5	317	1.473	2136.19	325	1.415		+3%	-4%	
		6	317	1.494	2156.42	325	1.415	2182.86	+3%	-5%	+1%
Example Two	Number of vias	2	317	1.006	884.74	312	0.990		-2%	-2%	
		3	337	1.178	1007.93	339	1.130		+1%	-4%	
		4	339	1.218	1024.29	340	1.138		0.3%	-7%	
		5	339	1.224	1024.29	341	1.150		+1%	-6%	
		6	340	1.250	1035.40	341	1.150	1036.19	0.3%	-8%	0.1%

* The unit for length is 0.001 inch.

TABLE II. Effects of rouitng history (continued)

			Fix # of vias at..., change size up to 30			Fix size at 30, change # of vias to...			Comparison		
			Number of comp. conn.	No. of vias per conn.	Ave. leng. per conn.	Number of comp. conn.	No. of vias per conn.	Ave. leng. per conn.	Completed conn. Incr.	Vias Incr. per conn.	Leng. Incr. per conn.
Example Three	Number of vias	2	359	0.791		347	0.764		-3%	-3%	
		3	397	1.023		393	0.990		-1%	-3%	
		4	423	1.239		420	1.162		-1%	-6%	
		5	423	1.239		423	1.187		0%	-4%	
		6	424	1.281	1890.05	424	1.198	1901.36	+1%	-6%	0.6%
Example Four	Number of vias	2	334	1.048		314	1.022		-6%	-2%	
		3	347	1.248		344	1.154		-1%	-7%	
		4	348	1.282		350	1.186		+1%	-7%	
		5	347	1.291		350	1.186		+1%	-8%	
		6	347	1.291	1214.44	350	1.186	1184.55	+1%	-8%	-2%

* The unit for length is 0.001 inch.

three columns are the results of routing histories from H_1 to H_5, the second three columns are the results of routing history H_6, and the last three columns are comparisons between routing results of different routing histories. For the first three columns, each row corresponds to a routing history from H_1 to H_5. The results in each row are the number of completed connections, the average number of vias used for each successful connection and the average routing length for each successful connection. For the second three columns, the last row indicates the final routing results of routing history H_6.

The last three columns are the increment of the number of completed connections, the increment of the average number of vias used for each successful connection, and the increment of the average routing length for each successful connection, respectively. The formulae for calculating them are as follows:

$$\text{Comp. conn. Incr.} = \frac{(\#\text{ of comp. conn. using H6}) - (\#\text{ of comp. conn. using Hi})}{\#\text{ of comp. conn. using Hi}} \quad (3.4)$$

$$\text{Vias Incr.} = \frac{(\#\text{ of vias/conn. using H6}) - (\#\text{ of vias/ conn. using Hi})}{\#\text{ of vias/conn. using Hi}} \quad (3.5)$$

$$\text{Leng. Incr.} = \frac{(\text{Aver. leng./conn. using H6}) - (\text{Aver. leng/conn. using Hi})}{\text{Aver. leng./conn. using Hi}} \quad (3.6)$$

It is seen from Table II that the successful connections (wires) are distributed more evenly on the board if the routing rectangle is allowed to be reasonably large from the beginning. Furthermore, the router will select the solutions with fewer vias if fewer vias are allowed to be used for complete connections. Consequently, the routing of connections becomes easier, yielding a higher routing completion with fewer vias. Of course, the routing length may increase because of the larger routing rectangle and fewer allowed vias, but the results show that this increment is not significant.

For the last surface-mount board (SMB), the routing history of H_6 produces results that are better in almost every aspect as compared with the other routing histories of H_1 to H_5.

3.2 Experimental Results for Large Routing Rectangle

The following routing histories were applied to a PCB and an SMB:

$$H_1' = \{(2,6), (2,12), (2,20), (2,30), (2,40)\} \quad (3.7a)$$

$$H_2' = \{(3,6), (3,12), (3,20), (3,30), (3,40)\} \quad (3.7b)$$

$$H_3' = \{(4,6), (4,12), (4,20), (4,30), (4,40)\} \quad (3.7c)$$

$$H_4' = \{(5,6), (5,12), (5,20), (5,30), (5,40)\} \quad (3.7d)$$

$$H_5' = \{(6,6), (6,12), (6,20), (6,30), (6,40)\} \quad (3.7e)$$

and

$$H_6' = \{(2,40), (3,40), (4,40), (5,40), (6,40)\} \quad (3.8)$$

Notice that Eqs. (3.2) and (3.7) are similar except for the new parameter set (n, 40) appearing in Eq. (3.7). The size of the routing rectangle constitutes the difference between Eqs. (3.3) and (3.8).

The results of these tests are shown in Table III. They are consistent with those of Table II. It is seen that the time needed to

TABLE III. Effects of routing history

			Fix # of vias at..., change size up to 40			Fix size at 40, change # of vias to...			Comparison		
			Number of comp. conn.	No. of vias per conn.	Ave. leng. per conn.	Number of comp. conn.	No. of vias per conn.	Ave. leng. per conn.	Completed conn. Incr.	Vias Incr. per conn.	Leng. Incr. per conn.
Example Three	Number of vias	2	360	0.794		341	0.733		-5%	-8%	
		3	399	1.033		391	0.972		-2%	-6%	
		4	424	1.245		418	1.139		-1%	-9%	
		5	424	1.250		418	1.139		-1%	-9%	
		6	424	1.281	1890.05	420	1.162	1896.05	-1%	-9%	0.0%
Example Four	Number of vias	2	338	1.056		314	1.022		-7%	-3%	
		3	348	1.250		346	1.162		-1%	-7%	
		4	351	1.291		352	1.190		0.3%	-8%	
		5	351	1.299		353	1.201		0.5%	-8%	
		6	351	1.299	1214.44	353	1.201	1212.78	0.5%	-8%	-0.1%

* The unit for length is 0.001 inch.

find solutions for connections increases with the increasing size of the routing area, as expected. However, since the results of Table III are not much better than those of Table II, the large routing rectangle size is not recommended.

3.3 Summary

As we have seen from the experimental results, factors other than routing algorithm can still affect the performance of routing to a large extent. Different routing histories lead to different routing results even though the final sets of routing parameters for different routing histories are identical. The proper selection of routing history can improve the final routing results. Of course, more extensive study of the effects of other routing histories should also be done to improve the routing results as much as possible.

CHAPTER IV

EFFECTS OF PLACEMENT ON MAZE ROUTABILITY

As we stated in the previous chapters, the processes of placement and routing are usually done separately in succession, although they are heavily dependent on each other. This approach often causes the following problem: Since we do not have any objective evaluation of the placement until routing is done, we may not realize that it is either impossible or very difficult to route a board based on a given placement configuration. In other words, only after having spent a relatively long time and considerable effort on routing, we can realize that the placement needs to be improved in order to achieve reasonably good routing results. Therefore, it is desirable to find an indicator capable of predicting the routability based on a given placement result. It is also desirable to find some criteria which would make it possible to compare different placement configurations for the same circuit in terms of routability. This approach is usually called a pre-routing analysis.

Foster [42] presented one strategy for the pre-routing analysis. In his paper, he claimed that the reason for the pre-routing analysis, as a figure of merit for placement, may in the long run be the most important. Here, we present a very simple methodology which can evaluate different placement configurations for the same circuit and predict the routability of the given board. Though Foster's method may be a good indicator of the pre-routing analysis, the method introduced here is simpler, and we will show that it is reasonably reliable by performing experiments with the test boards. By using this method, we are able to predict the routability based on the given placement configurations.

4.1 A Pre-Routing Analysis of Connection Densities

Component placement within a printed circuit board or surface-mount board produces a rats nest. The rats nest contains all the information needed to complete routing of all the connections. Foster [42] suggested a method to gather the raw data for the pre-routing analysis as follows: First, subdivide the board into m vertical strips; then for every connection, a counter for a vertical strip is incremented whenever the connection crosses the vertical strip, signifying that a horizontal track will be needed somewhere in that vertical strip. The final value of the counter is called the connection density of that corresponding strip. It indicates the minimum horizontal tracks required to complete the routing in that vertical strip. Similarly, the board is also subdivided into n horizontal strips, and a counter for each horizontal strip is incremented whenever a connection crosses the horizontal strip, thus signifying that a vertical track will appear in that horizontal strip. The final value of the counter is the connection density of that corresponding horizontal strip. The number of vertical strips m and the number of horizontal strips n can be chosen according to the board size and the computational complexity. With larger m and n, more detailed data can be gathered.

We employ the same method to get the densities of the connections in horizontal and vertical layers. The board is first divided into vertical strips with the width resolution of 0.25 inch per strip. We then count the number of connections which cross the individual strip, and the resulting number indicates the minimum number of horizontal tracks needed to route those connections within that strip. Similarly, the board is divided into horizontal strips with the same resolution, and the number of connections which cross each horizontal strip is obtained.

Next, we normalize the densities by calculating the density of strips per unit length (inch). That is, if the density for a vertical strip is Ω and the length of the vertical strip is d, then the normalized density is Ω/d. Based on the above data, we can calculate the mean value and variance of all the normalized horizontal strip densities and vertical strip densities as follows:

$$M_h=(1/N_h)*\sum_{i=1}^{N_h} \Omega_{hi} \tag{4.1}$$

$$V_h^2=(1/(N_h-1))*\sum_{i=1}^{N_h}(\Omega_{hi}-M_h)^2 \tag{4.2}$$

$$M_v=(1/N_v)*\sum_{i=1}^{N_v} \Omega_{vi} \tag{4.3}$$

$$V_v^2=(1/(N_v-1))*\sum_{i=1}^{N_v}(\Omega_{vi}-M_v)^2 \tag{4.4}$$

where M_h and M_v are the mean values of the horizontal and vertical normalized densities, V_h^2 and V_v^2 are the variances of the horizontal and vertical normalized densities, N_h and N_v are the numbers of the horizontal strips and vertical strips, and Ω_{hi} and Ω_{vi} are the normalized densities for the horizontal and vertical strip i, respectively.

From a statistical point of view, we know that the mean value is the best representation of all the samples, and the variance is the measurement of the validity of the mean value as the best representation of the samples. Thus, we should expect that the larger the mean value of the connection density, the denser the board, and the more difficult the routing. Also, for the same mean density, but a larger variance, we can conclude that the connections are distributed more unevenly on the board, and that the routing of the board will be more difficult than the routing of a board with a smaller variance.

Notice that the symmetry of bus-oriented circuits may not comply with the above principles. Such circuits are routed in zones.

4.2 A Pre-Routing Analysis of Connection Length

Another critical parameter we should consider is the length of a connection. Here connection refers to the connection between any two pins in the rats nest. It is well known that to route a shorter connection is easier than to route a longer one. The size of available routing space also affects the routability of the circuit. Therefore, we shall use the average connection length per unit area of a board to represent the effects of the connection length and the routing space. It is expected that for the same density distribution, routing on a board with a shorter average length per unit area is easier than routing with a longer average length per unit area.

It should be noted that although the length of a connection in the rats nest is the Euclidean distance between the two corresponding terminals (Euclidean distance between two points (x_1, y_1) and (x_2, y_2)

is measured by the square root of the sum of $(x_1-x_2)^2$ and $(y_1-y_2)^2$), the minimum distance of a connection between the terminals routed by an orthogonal router on the board is the Manhattan distance. We prefer to use Manhattan distance between the two points to represent the length of the rats nest.

4.3 Routability Indicator

Four test boards were investigated in regard to the routability prediction based on the given placement configuration. We gathered the raw data of the connection distributions of these four boards. Based on the discussion of sections 4.1 and 4.2, we developed the following expressions to calculate the routability indicator (RI):

$$RI_1 = \frac{K_1}{\log((Mh*Vh+Mv*Vv)*(Total\_length/Size\_of\_board))} \qquad (4.5)$$

$$RI_2 = \frac{K_2}{\log((Mh+Mv)*(Vh+Vv)*(Total\_length/Size\_of\_board))} \qquad (4.6)$$

where K_1 and K_2 are constants, M and V are given by Eqs. 4.1 to 4.4, Total_length is the total length of the rats nest, and Size_of_board represents the routing area. These two formulae reflect the relationship between the routability and the factors affecting the routability which were discussed in sections 4.1 and 4.2, the selection of the functions in this formulae was made by curve fitting. The selection of the constants K_1 and K_2 depends on a subset of optimum routing parameters used. For example, the constants K_1 and K_2 are 11.77 and 12.46, respectively, for the Manhattan distance representing the Total_length, the routing history H_6, as given by Eq. 3.3, and the fixed subset of routing parameters as described in [72]. Due to time limitation, the constants were computed only once from a test board that had 100% completion of routing (Example 2). For the same Example 2 and the Eclidean distance, $K_1=11.58$ and $K_2=12.27$.

It should also be noticed that the size of the board may not be identical to the available routing area for that board, but the ratio of the available routing area and the size of the board is a constant for a given schematic diagram regardless of the placement configuration, provided that the size of the elements on the board has not been changed. The size of the board can still represent fairly well the available routing area, while the measurement of the size of the board is much easier than the measurement of the available routing area.

Tables IV-A and IV-B present the routability indicators calculated from four boards with placed components, using the Manhattan (with $K_1=11.77$ and $K_2=12.46$) and Euclidean (with $K_1=11.58$ and $K_2=12.27$) distance, respectively. Since Example 2 has 100% completion of routing, it serves as the basis for the calculation of K_1 and K_2. These constants were then used to calculate RI_1 and RI_2 for the other three boards. It can be seen that the routability indicators predict the actual routing completion well. Further work should improve the prediction accuracy.

4.4 Experimental Results for Routability

We routed the same four test boards as described in Chapter III, using OPTIMATE™ software with the same routing history. The routing completion percentage and the calculating results of the above formulae for these boards are summarized in Table IV.

TABLE IV-A. Routability Indicator using Manhattan distance

	Density				Total Length (mil)	Size of Board ($inch^2$)	Routability Indicator		Actual Routing Completion
	Horizontal		Vertical						
	Mean	Variance	Mean	Variance			RI1	RI2	
Example One	8.474	16.228	4.824	3.811	753175	48.0	87.60%	88.77%	91.81%
Example Two	4.133	3.126	3.756	2.736	366743	38.4	100%	100%	100%
Example Three	5.754	2.455	3.233	3.156	744994	60.5	97.18%	97.18%	99.76%
Example Four	11.998	40.208	8.344	16.267	506178	29.7	81.49%	82.54%	81.02%

TABLE IV-B. Routability Indicator using Euclidian distance

	Density				Total Length (mil)	Size of Board ($inch^2$)	Routability Indicator		Actual Routing Completion
	Horizontal		Vertical						
	Mean	Variance	Mean	Variance			RI1	RI2	
Example One	8.474	16.228	4.824	3.811	653679	48.0	87.12%	88.32%	91.81%
Example Two	4.133	3.126	3.756	2.736	303851	38.4	100%	100%	100%
Example Three	5.754	2.455	3.233	3.156	674899	60.5	96.41%	96.46%	99.76%
Example Four	11.998	40.208	8.344	16.267	417219	29.7	81.28%	82.35%	81.02%

From Table IV, we can see that the routability indicator, RI, is not identical to the actual routing completion percentage. This is partially due to the fact that the second-order statistics do not contain all the information in the rats nest. Another reason is that we use the board size to represent the available routing area because of the reason stated before, but the ratios between them are obviously different for different boards, and this may introduce some errors in our experiments.

Though the most valid method of evaluating a placement configuration is to actually route it, a good indicator may save time and effort of doing routing by just doing some simple calculations. Though Table IV compares the results between four different examples, we could compare results for the same board with different placement configurations. The later one is more attractive and important. We may also be able to find out the threshold value of the RI in order to reach a certain percentage of routing completion for a particular routing software and the routing parameters. Also, based on the values of RI for different placement configurations, we could select the best placement configuration to continue the routing process.

4.5 Summary

In this chapter, we related the placement and routing by using information contained in the placement configuration to predict the routability of the board. A method to gather the information from a placement configuration was introduced. The relationship between data obtained from a given placement configuration and the corresponding routing was analyzed. Two empirical formulae for calculating routability were developed and their validity was demonstrated by examples. The advantages for pre-routing analysis were analyzed, and we concluded that the most valuable feature of the routability indicator is in its ability to give a fairly good indication of how good a placement is for the routing to be done on it. Thus allowing us to select the placement configuration which can be routed with a higher routability percentage.

CHAPTER V

A NEW CHANNEL ROUTER

5.1 Introduction

The fundamental methods presently used to route printed circuit boards (PCBs) and surface-mount boards (SMBs) are the Maze routing, Line routing and Channel routing methods. Of these three classes of algorithms only one has suitable characterstics for LSI and VLSI integrated circuit (IC) design using gate array and standard cell methodologies. The method of channel routing is suitable for high-density chip layout because it utilizes a simultaneous modularized scheme. By splitting the routing process into two steps of topological routing and track assignment, the channel router allows iterative optimization of the routing at a lower overall cost than the other two methods.

In chapter II, we discussed many routing algorithms of Lee's type. Though they are different in many aspects, one thing remains the same. That is, they all can route one path at a time. Once a connection path is routed, it becomes an obstacle to other connections to be routed. Theoretically, those connections blocking other connections can be removed and be re-routed later, but there is no guarantee that they can be successfully routed at a later time because there are more connections existing on the board. Also, removing some connections and re-routing them is very time-consuming. The discussions on re-routing can be found in [43] and [44].

5.2 Ordering in Maze Routing Algorithms

It seems to be very important to decide on the order in which the nets should be routed. Unfortunately, there is no simple criterion according to which we should make such a decision. In practice, two ordering methods are used: (i) in the ascending order of the length of the net and (ii) in the descending order of the length of the net. The argument for the first method is that routing of a short-length connection causes fewer obstacles which prevent the completeness of the later routing. Also it is easier to route a longer length connection around a shorter one than vice versa. The argument for the other method is that since routing of a long-length connection is more difficult than routing a short-length connection, it is more desirable to route the longer one first in order to increase the completeness of the routing. But according to Abel [45], the performance of a maze router is independent of the order in which connections are attempted if the minimum length of connections successfully completed is used as the reasonable norm.

Even though there is not much difference between the two ordering methods based on the length of the net in the sense stated above, it is clear that one method may be more desirable than the other under certain circumstances. For example, if the propogation time delay on the wires is critical to the operation of the circuit, the length of the wire should be as short as possible, and the method of routing the longest connections first may be preferable. The reason is that the router can trace the long-length path connection along a more direct line because fewer obstacles (previously routed paths) exist on the board; and the short-length path connection may have slightly longer traces because of the other obstacles already existing on the board. Both factors will make contribution to the balance of the length of the routed path.

Kulkarni and Jayakumar [46] presented another ordering strategy in which the order of nets to be routed is based on a criterion other than the length of the net. This method was shown [46] to perform better than the methods of ordering based on the the length of the net for thirteen of the sixteen test printed circuit boards.

Lee's (maze) routing algorithm has another problem: that is, in order to trace the shortest possible wires, the router may make many connections through a congested area. This tendency may result in (i) an uneven density of wires on the board (wire clusters), and (ii) the need for any blocking connections to be routed at a later time.

5.3 The Channel Routing Strategy

To overcome the above problems, one solution is to use a channel routing strategy [47-49]. In this strategy, the routing process consists of the following two steps: (i) global routing (or loose routing) and (ii) track assignment (or channel routing).

5.3.1 Global Routing

The global router [50-54] first breaks the routing area into rectangular regions, which are called channels, between the components or modules to be interconnected. Then the global router decides through which channels individual connections will run. Since the global routing is rather simple, it may iterate several times in order to diminish or eliminate overflows (An overflow condition occurs when the number of tracks within a channel exceeds the maximum number of tracks that the channel can have.) The iterative process may achieve higher routability.

5.3.2 Local Routing

When the global routing is completed, the detailed connection paths of nets are determined by using a local routing approach. This local routing is called track assignment (or channel routing) [55-67], which we will discuss extensively in the following two chapters. Since the track assignment considers only a portion of the routing area, and the track assignment for different channels can be done independently, not only the memory needed to perform routing can be reduced significantly, but also the total time needed may be reduced.

5.4 Channel Routing Applicability

Due to the regular shape of modules in LSI and VLSI ICs, using gate array or standard cell methodologies, the module placement leaves rectangular spaces between the modules to be interconnected. Since such rectangular spaces are exactly what the channel router needs to operate on, channel routing is very suitable for gate-array and standard-cell LSI and VLSI layout.

The main goal of channel routing is to use the smallest number of tracks in order to route all the connections. That is, the optimum realization of a channel routing problem is the realization of the problem with the minimum number of tracks. The following sections are dedicated to a new algorithm for channel routing. The algorithm is simple but is shown to be very efficient through examples. Before we introduce the algorithm, the pertinent concepts and definitions are introduced in the next section.

5.5 Definitions

We shall describe (i) how a channel routing problem can be represented using graphical and matrix forms, (ii) how a directed net relation (DNR) graph can be constructed to show which pairs of nets should be exchanged to avoid any potential overlap of traces in the vertical tracks, (iii) how to construct a distance matrix used to resolve any potential overlap of traces in the horizontal tracks, and (iv) how to find the lower and upper bounds on the number of tracks required to realize the channel routing problem.

5.5.1 The Channel Routing Problem and its Representation

As shown in Fig. 15, a channel routing problem is defined on a rectangular area called a channel. Each channel has two rows of terminals along its top and bottom sides. Tracing of all the connections within the channel is done on two layers, which are electrically isolated from each other. We assume that the horizontal tracks are located on one layer, and the vertical tracks on the other. Connections are routed within tracks. Since horizontal tracks are

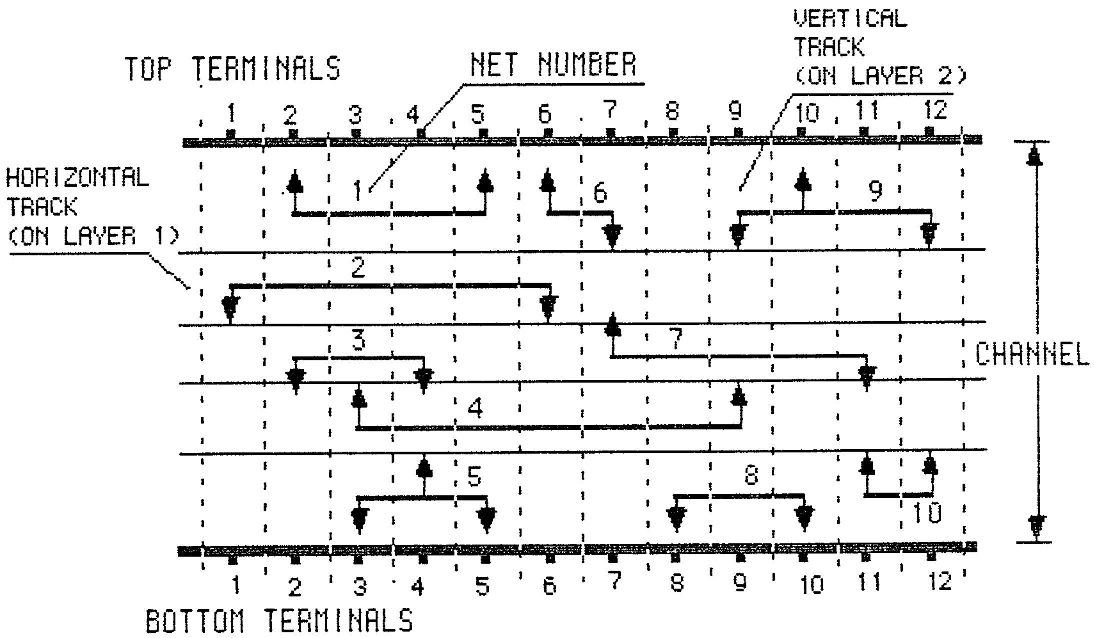


Fig. 15. Net list of a channel routing problem

NET NUMBER	TERMINAL NUMBER 1	2	3	4	5	6	7	8	9	10	11	M 12
1	0	1	2	2	1	0	0	0	0	0	0	0
2	-1	2	2	2	2	-1	0	0	0	0	0	0
3	0	-1	2	-1	0	0	0	0	0	0	0	0
4	0	0	1	2	2	2	2	2	1	0	0	0
5	0	0	-1	1	-1	0	0	0	0	0	0	0
6	0	0	0	0	0	1	-1	0	0	0	0	0
7	0	0	0	0	0	0	1	2	2	2	-1	0
8	0	0	0	0	0	0	0	-1	2	-1	0	0
9	0	0	0	0	0	0	0	0	-1	1	2	-1
N 10	0	0	0	0	0	0	0	0	0	0	1	1

Fig. 16. Matrix representation of channel routing problem of Fig. 15

isolated from vertical tracks, the connections between them are made through vias.

The channel routing problem is expressed by a graphical net list as shown in Fig. 15. This representation is referred to as <u>graphical representation</u>. For computational purposes, the problem can also be represented by a matrix A with dimensions NxM

$$A = [a_{ij}] \tag{5.1}$$

where N is the number of nets in the problem and M is the number of terminals on one side of the channel. The matrix elements a_{ij} have the following form:

$a_{ij}=+1$ if net i is connected to the terminal j on the <u>top</u> side of the channel;

$a_{ij}=-1$ if net i is connected to the terminal j on the <u>bottom</u> side of the channel;

$a_{ij}=2$ if net i crosses the vertical track j, but is not connected to the terminals within the track; and

$a_{ij}=0$ otherwise.

Such a matrix representation of the problem is shown in Fig. 16. It is readily seen that each column of the matrix has only a single positive 1 (+1) and/or a single negtive 1 (-1). The number of 2s can be greater than one. The maximum number of nonzero elements in one column is the maximum net density of the channel routing problem. Notice that, although the graphical representation does not call for any net numbering, the matrix representation requires a net numbering for their placement in the matrix. The net numbering may be arbitrary, as shown in Fig. 15.

In the realization of the problem, such as that shown in Fig. 17, we assume that the horizontal segment of each net can occupy no more than one track (i.e., no so called doglegs are allowed). With this restriction, a problem can be guaranteed to be solvable if and only if the channel routing problem does not have any cyclic conflict, as shown in Fig. 18. Such a cyclic conflict cannot be resolved because

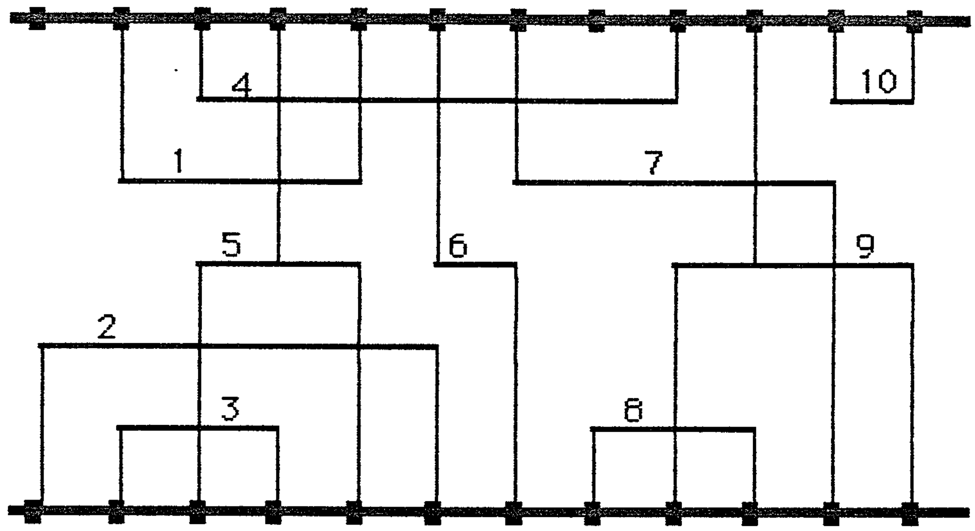


Fig. 17. A realization for the channel routing problem of Fig. 15.

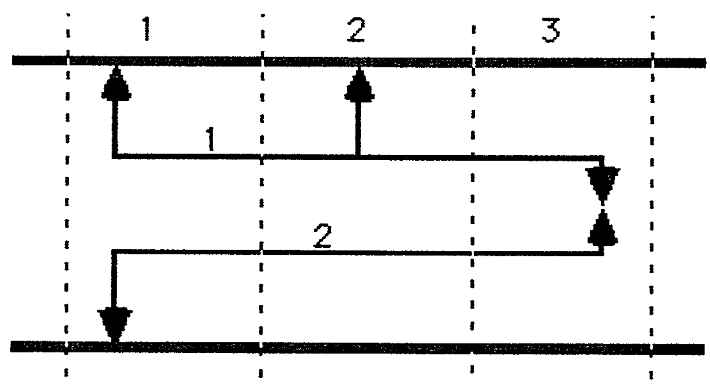


Fig. 18. An example with cyclic conflict

when the left vertical track 1 has no overlapping vertical segments of nets 1 and 2, the vertical segments of nets 1 and 2 must overlap in the vertical track 3. On the other hand, if we interchange the order of the nets (net 1 at the bottom, and net 2 at the top) then the conflict occurs in the left vertical track 1.

Notice that any non-cyclic trace conflict that may appear in the channel routing representation (either graphical or matrix) can be resolved by using proper procedures. For example, the vertical trace conflict between nets 6 and 7 shown in the vertical track 7 of Fig. 15 can be solved with the use of a directed net relation (DNR) graph (Section 5.5.2), and the horizontal trace conflict between nets 4 and 7 can be resolved by the use of a distance matrix (Section 5.5.3).

5.5.2 Directed Net Relation (DNR) Graph

Recall that a solvable problem can be realized when no overlap between two nets occur within a vertical track. Furthermore, no dogleg (a jump of the horizontal net segment between the horizontal tracks) is allowed, resulting in only one horizontal segment per connection. With these assumptions, it is clear that the horizontal segment of a net i connecting to the top terminal j (in the vertical track j) must be placed above the horizontal segment of another net k connecting to the bottom terminal j (in the same vertical track j). This condition results in an <u>ordered matrix representation</u> whose columns contain single pairs of +1 and −1, with the property that the +1 is in a row above the row with the −1. There are various algorithms used to re-order the matrix in order to resolve all the vertical trace conflicts.

We shall accomplish the re-ordering by using a <u>directed net relation (DNR) graph</u>. The DNR graph reflects the relationship between nets of a channel routing problem as follows: As shown in Fig. 19-a, each vertex i in DNR graph represents a net i in the channel routing problem. The graph contains a directed edge (i,k) from vertex i to vertex k if and only if net i is to be placed in a horizontal track above the track with the net k in order to avoid the vertical trace conflict. For a solvable problem, the complete graph should not have any directed cycles representing cyclic confilcts as shown in Fig. 19-b.

Next, we assign an <u>ordering number</u> to each vertex according to

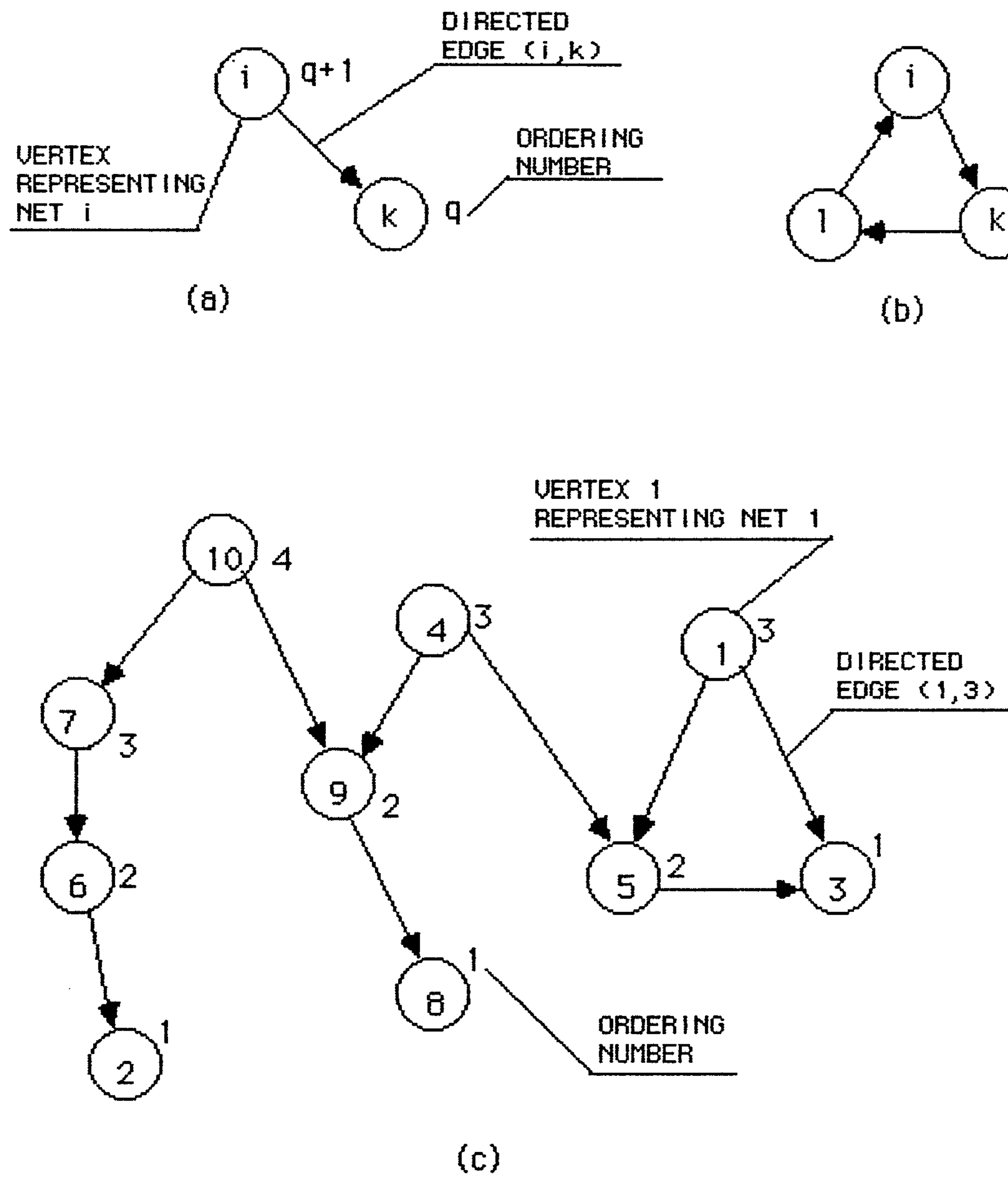


Fig. 19. The directed net relation (DNR) graph
(a) Nomenclature (b) Cyclic conflict
(c) Example corresponding to Fig. 15

its category. As shown in Fig. 19-c, the vertices fall into three categories: (1) the vertices with no edges emanating from them, representing the nets located close to the bottom of the channel, (2) the vertices with single edges emanating from them, representing pairs of nets to be exchanged, and (3) the vertices with more than one edge emanating from them, representing the nets to be exchanged more than once. The procedure for the ordering number assignment is as follows: First, any vertex of class 1 is assigned the ordering number 1. Then, each successive vertex k of class 2, with a directed edge from k to i, is assigned an ordering number q+1 if vertex i has been assigned the ordering number q. If the vertex k of class 3 has already been assigned an ordering number, we must compare the ordering number already assigned with the one to be assigned, and if the newly assigned number is larger than the old one, the old ordering number is replaced by the new one; otherwise it remains unchanged. This procedure continues until every vertex is assigned the proper ordering number.

Figure 19-c shows an example of the DNR graph, corresponding to the problem of Fig. 15. To construct the graph, we employ a scan of

all the nets in the channel from the left vertical track to the right. First, net 2 is considered, and a vertex is placed with the corresponding net number 2 in it. Since nets 1 and 3 appear in the vertical track 2, vertices 1 and 3 are placed. Since net 1 has to be placed in a track above the one for net 3, a directed edge (1, 3) is added between vertices 1 and 3. This procedure continues until the last vertical track is reached. Usually, the initial form of the graph does not resemble that of Fig. 19-c. Figure 19-c is obtained by a re-arrangement of the positions of its vertices, in order to resemble the final track assignment in the channel. Finally, we assign a corresponding ordering number to each vertex in the graph.

From the graph, we see that vertex 2 does not have any edges emanating from it, which means that net 2 does not have to be placed in a horizontal track above a track for any other nets. So, an ordering number 1 (shown outside the circle) is assigned to it. Since a directed edge from vertices 6 to 2 exists, an ordering number 2 (equal to 1+1) is assigned to vertex 6. This assignment continues up to vertex 10 as well as all the other vertices. Note that since there are directed edges (1, 5) and (5, 3) in the DNR graph, the ordering number of vertex 1 should be 3 instead of 2 which would otherwise result from the edge (1, 3).

5.5.3 The Distance Matrix

The directed net relation (DNR) graph reflects the constraint in the track assignment due to the overlaps in the vertical tracks. Clearly, it is not enough to just avoid the overlaps in the vertical tracks. Overlapping in the horizontal tracks must also be avoided. For example, we cannot place nets 2 and 3 in the same horizontal track even though there are no overlaps in any vertical tracks (see Fig. 15). Thus, a matrix called the distance matrix is created to reflect the constraint imposed by the overlaps in the horizontal tracks. The NxN symmetric matrix is denoted by

$$D = [d_{ik}] \tag{5.2}$$

where N is the number of the nets in a channel routing problem. Each element d_{ik} of the matrix indicates the horizontal distance between nets i and k. The distance between two nets is the number of vertical tracks left between them plus one if they are placed in the same horizontal track. If the two nets are overlapped when they are placed in the same horizontal track, the distance between them is equal to zero. For example, $d_{17}=d_{71}=2$, $d_{16}=d_{61}=1$, and $d_{15}=d_{51}=0$ are all elements of the distance matrix for the channel routing problem shown in Fig. 15.

5.5.4 Lower and Upper Bounds of the Track Number Required

Though effort has been devoted to finding the necessary and sufficient conditions for the number of tracks needed to realize a given channel problem [68, 69], no such conditions have been developed for a general channel routing problem. More precisely, only the number of horizontal tracks is of interest, because the number of vertical tracks is determined by the number of terminals in the problem. Without providing any detailed discussion on this topic here, it may be observed that the maximum net density is the obvious lower bound for the number of horizontal tracks required in a channel to realize a given problem, because every net crossing the vertical track where the maximum net density occurs requires one horizontal track. The maximum ordering number assigned to the vertices in DNR graph is also another lower bound for the number of horizontal tracks required, since the vertical constraint of the net implies that every net represented in the longest chain of the DNR graph has to be in different horizontal tracks. The number of the total nets in a given problem, on the other hand, is an upper bound of the number of tracks required to realize the problem.

5.6 The New Track Assignment Algorithm

The essence of the new track assignment algorithm is to assign proper nets, to the horizontal tracks so that the number of tracks required to realize the problem is minimized. The track assignment is done according to a priority number (defined later). The algorithm is as follows.

5.6.1 The New Track Assignment Algorithm

[New Track Assignment Algorithm]:

Step 0) Initialization.

ORDERINGNUMBER=highest ordering number in the DNR graph;

SET0={all nets to be routed}; //set of nets to be routed//

SET1=Ø; //set of nets routed//

TRACKNUMBER=0;

Step 1) TRACKNUMBER=TRACKNUMBER+1;

Step 2) Are there any nets in SET0 with ordering number equal to ORDERINGNUMBER? If yes, choose one of these nets as MOTHER NET A and goto Step 3; otherwise, ORDERINGNUMBER=ORDERINGNUMBER-1, goto Step 2.

Step 3) Create a set READYNETS with respect to MOTHER NET A. READYNETS includes the nets which have non-zero distance with net A and whose corresponding vertices in the current DNR graph have no edges directed to them. Notice that the selection rules for the nets in READYNETS imply that each net in READYNETS, together with net A, can be routed in one track, without violating any horizontal and vertical track constraints.

Step 4) Choose a proper subset S_0 of nets in READYNETS such that (i) all the nets in the subset, together with net A, can be assigned to one track, and (ii) the assignment of those nets in the current track may lead to the minimization of the total number of tracks required. Assign these nets and net A to track TRACKNUMBER.

Step 5) Delete those nets in S_0 and net A from SET0.

Add those nets and net A into SET1.

Considering the DNR graph G, delete those vertices (and their associated ordering number) which correspond to the nets assigned to track TRACKNUMBER. Also delete all the edges emanating from those vertices.

Step 6) Have all the nets been assigned ? (that is, if set SET0 equal to null set?) If yes, exit; otherwise goto Step 1.

□

5.6.2 Net Priority Function

In order to decide which net(s) in the set READYNETS should be routed in the same track with MOTHER NET A, first we establish a function referred to as priority function P(NET,A). This function should reflect the priority of a net NET to be selected for routing in the same track where net A is located. An example of a suitable function is given by:

$$\begin{aligned} P(NET,A) = \ & Klen*Length \\ & +Kdis*(Mdis-Distance) \\ & +Kord*Order\_number \end{aligned} \tag{5.3}$$

where Klen, Kdis, and Kord are weight factors for Length, Distance, and Ordering number, respectively; Mdis is the maximum distance among the distances between the net A and the nets in READYNETS; Length is the length of net NET, Order_number is the ordering number of net NET, Distance is the distance between net NET and A.

5.6.3 Selection of an Optimum Subset of Nets for Track Routing

The overall track assignment process occurs in stages, one stage for each track. Since some of the nets from the total set of nets eligible for routing are assigned to the current track, the DNR graph must be reduced after each stage.

The critical step in the new track assignment algorithm is Step 4. Since there may be some horizontal track conflicts between the nets in the set READYNETS, not all them can be assigned to one track simultaneously. Thus, the problem is how to properly select the net(s) from READYNETS so that not only the selected net(s) together with net A could be assigned to the same track, but also their assignment could result in an optimum realization.

Notice that at the end of each track assignment stage, the maximum ordering number in the current reduced DNR graph indicates the lower bound of the number of tracks required to complete the track assignment. Thus, a smaller maximum ordering number in the current reduced DNR graph indicates that fewer tracks are required to assign the rest of the nets. Consequently, the nets with a larger ordering number should be assigned first in order to obtain a reduced DNR graph with a smaller maximum ordering number. On the other hand, in order to minimize the number of tracks used in the channel, we should utilize the current routing track as much as possible. This can be achieved by assigning nets with the smallest distance between them (The distance is defined in Section 5.5.3.). For example, two long nets with short horizontal distance utilize the track better than three short nets with large distances.

It should also be noticed that the above discussion relates to a class of problems for which the maximum density number is not greater than the maximum ordering number. In other words, the lower bound on the number of the horizontal tracks required is determined by the maximum ordering number rather than the maximum density number.

For convenience, let us define <u>an eligible subset</u> S_i of the set READYNETS as a subset in which all the nets, together with net A, can be routed simultaneously in one track, or equivalently, as a subset with all the nets in the set READYNETS having non-zero distance between them. Notice that for all eligible subsets S_1, S_2, ..., S_l, we have:

$$S_1 \cup S_2 \cup \ldots\ldots \cup S_l = \text{READYNETS} \qquad (5.4)$$

($S_i \cap S_j$ is not necessarily equal to null for all i and j)

We can now define an optimum subset S_0 as follows: An optimum subset S_0 is an eligible subset of nets whose sum of the priority numbers of all the member nets in this subset is the highest among all eligible subsets. Thus, the nets in the optimum subset S_0 and the net A can be routed in the track TRACKNUMBER.

The last remaining question is how to find the optimum subset S_0. In general, the problem of generating the optimum subset S_0 is equivalent to finding a subset of vertices in a weighted graph G_R such that there are no edges between vertices in that subset, and the sum of the weights associated with the vertices in the subset is maximized. The weighted graph G_R consists of vertices and edges corresponding to the nets in READYNETS. Each vertex in G_R corresponds to a net in READYNETS, and the weight associated with each vertex is the priority number for the corresponding net. An edge between two vertices exists if and only if those two corresponding nets have <u>horizontal</u> overlapping and, consequently, can not be routed in the same track.

Since the "subset" problem is NP-complete, so it is very unlikely an efficient algorithm exists that could generate an optimum subset of vertices for a given weighted graph G_R. In our implementation of the routing algorithm, a sub-optimum alternative approach is used. A description of the implementation of the approach is given in Appendix A. The corresponding program listing in Pascal is included in Appendix B. The listing refers to the latest version of the program extensively tested on an Apollo DN660 workstation. The first version was developed and tested on Amdahl 580/5850 computer.

5.7 Efficiency of the New Algorithm

Two examples are selected to illustrate the efficiency of the new track assignment algorithm.

5.7.1 Example A

The net list of the first example is shown in Fig. 15. The corresponding DNR graph is shown in Fig. 20-a. The priority function P(NET, A) of Eq. 5.3 is evaluated using Klen=1, Kdis=1, and Kord=5.

We shall now demonstrate the procedure by executing all the steps of the algorithm of Section 5.6.1. The partial results are also demonstrated in Fig. 20.

Step 0) ORDERINGNUMBER=4.

SET0={1,2,3,4,5,6,7,8,9,10}.

SET1=Ø.

TRACKNUMBER=0.

Step 1) TRACKNUMBER=1.

Step 2) Net 10 is selected as net A; SET0={1,2,3,4,5,6,7,8,9}.

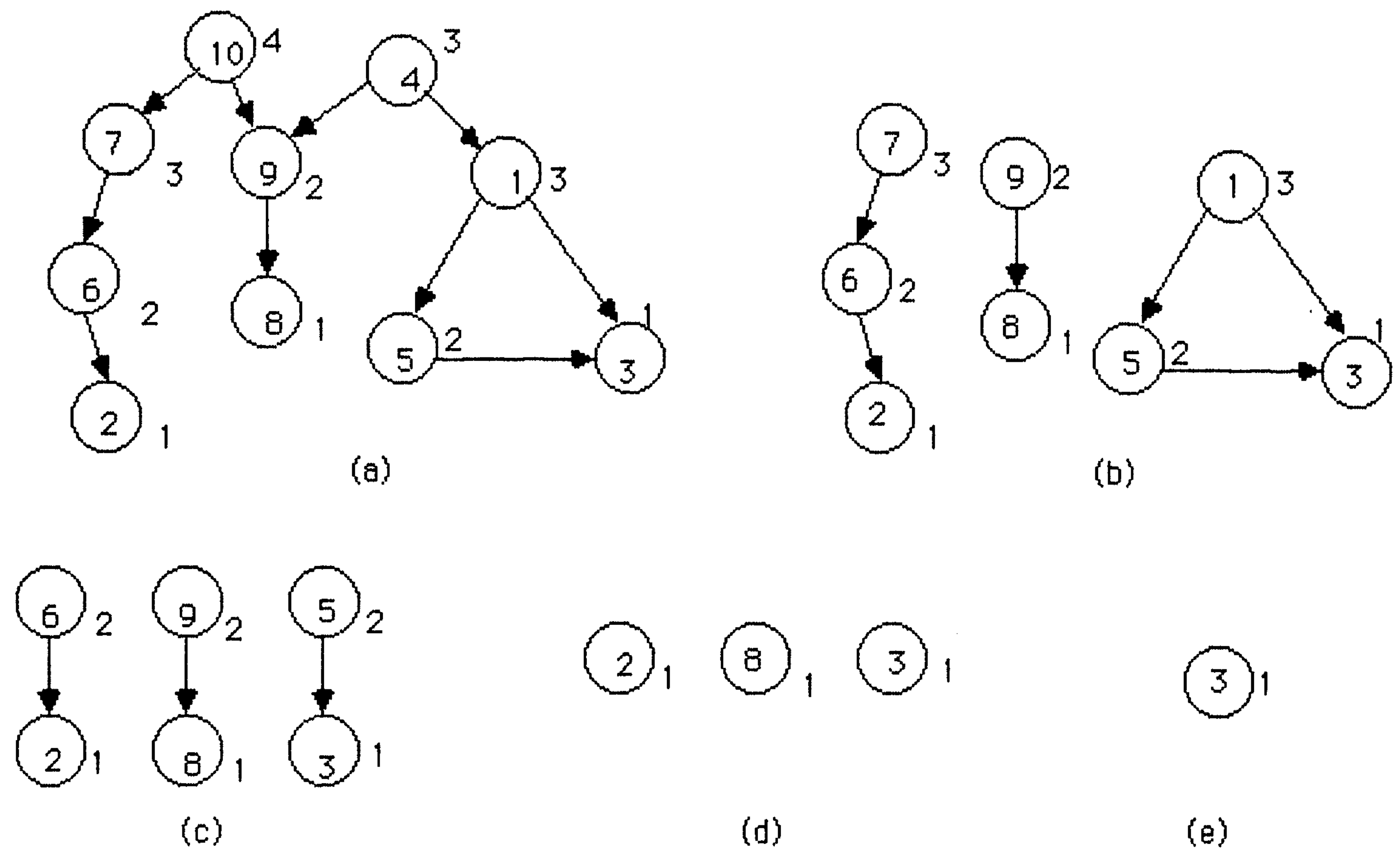


Fig. 20.

(a) Directed net relation (DNR) graph for Example A
(b)-(e). Reduced DNRG for Example A

Step 3) READYNETS={4, 1}.

Step 4) P(4, 10)=25; P(1, 10)=18.

S_0={4}.

Assign net 4 to track 1 and net 10 to track 1. For brevity, we use the following notation:

Net 4->track 1; net 10->track 1.

Step 5) SET0={1,2,3,5,6,7,8,9}.

Reduced DNR graph is shown in Fig. 20-b.

Step 6) Goto Step 1.

Step 1) TRACKNUMBER=2.

Step 2) ORDERINGNUMBER=3.

Step 2) Net 7 is chosen as net A; SET0={1,2,3,5,6,8,9}.

Step 3) READYNETS={7}.

Step 4) P(7, 1)=18.

S_0={7}.

Net 7->track 2; net 1->track 2.

Step 5) SET0={2,3,5,6,8,9}.

Reduced DNR graph is shown in Fig. 20-c.

Step 6) Goto Step 1.

Step 1) TRACKNUMBER=3.

Step 2) ORDERINGNUMBER=2.

Step 2) Net 9 is selected as net A; SET0={2,3,5,6,8}.

Step 3) READYNETS={5, 6}.

Step 4) P(5, 9)=12; P(6, 9)=13.

S_0={5,6}.

Net 5->track 3; net 6->track 3; net 9->track 3.

Step 5) SET0={2,3,8}.

Reduced DNR graph is shown in Fig. 20-d.

Step 6) Goto Step 1.

Step 1) TRACKNUMBER=4.

Step 2) ORDERINGNUMBER=1.

Step 2) Net 2 is selected as net A; SET0={2,3,8}.

Step 3) READYNETS={8}.

Step 4) P(8, 2)=7.

S_0={8}.

Net 8->track 4; net 2->track 4.

Step 5) SET0={3}.

Reduced DNR graph is shown in Fig. 20-e.

Step 6) Goto Step 1.

Step 1) TRACKNUMBER=5.

Step 2) Net 3 is selected as net A; SETO=$\emptyset$.

Step 3) READYNETS=$\emptyset$.

Step 4) $S_0=\emptyset$.

Net 3->track 5.

Step 5) SETO=$\emptyset$.

Reduced DNR graph is a null graph.

Step 6) Exit.

□

The final track assignment result is shown in Fig. 21 (similar to that of Fig. 17). The realization can be easily verified to be an optimum one because the number of horizontal tracks used is equal to the maximum density number.

5.7.2 Example B

Another example is the problem with the net list shown in Fig. 22. The corresponding DNR graph is shown in Fig. 23.

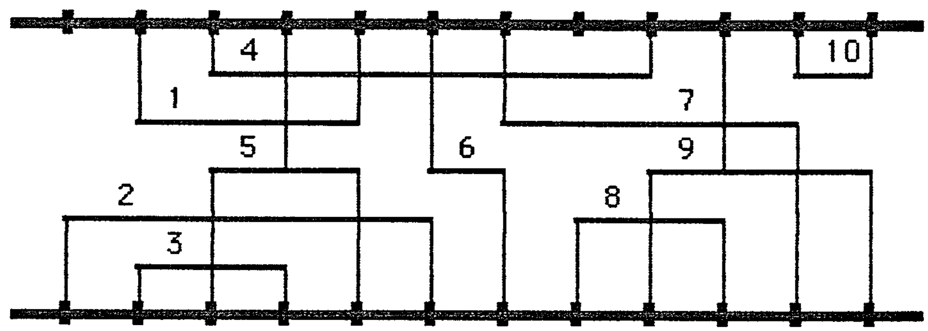


Fig. 21. The realization of Example A by using the new algorithm

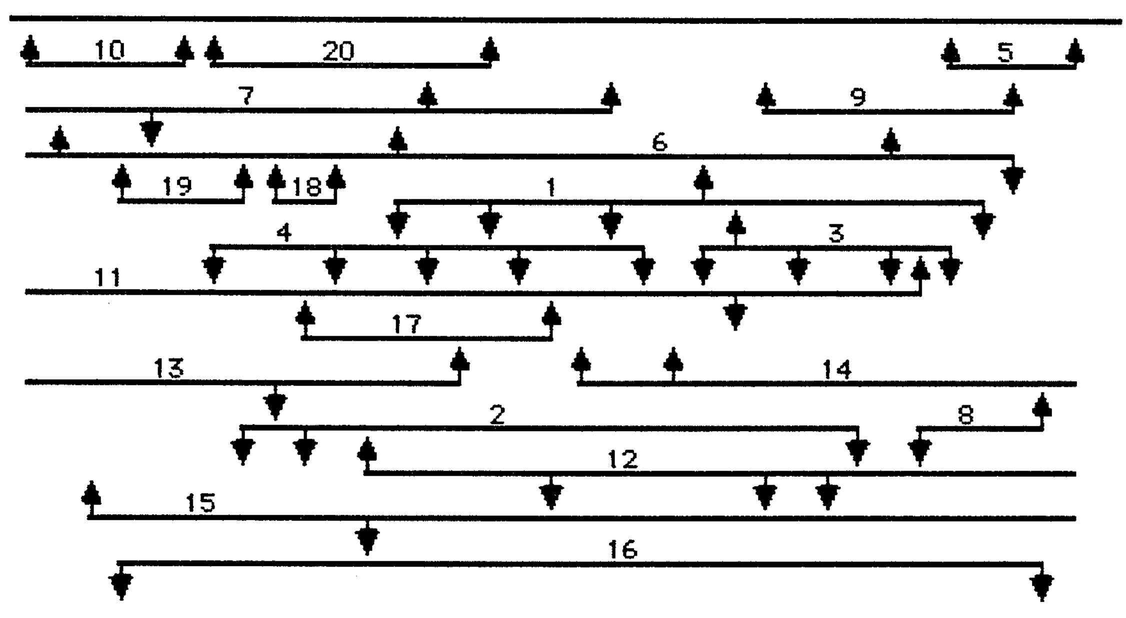


Fig. 22. The net list for Example B

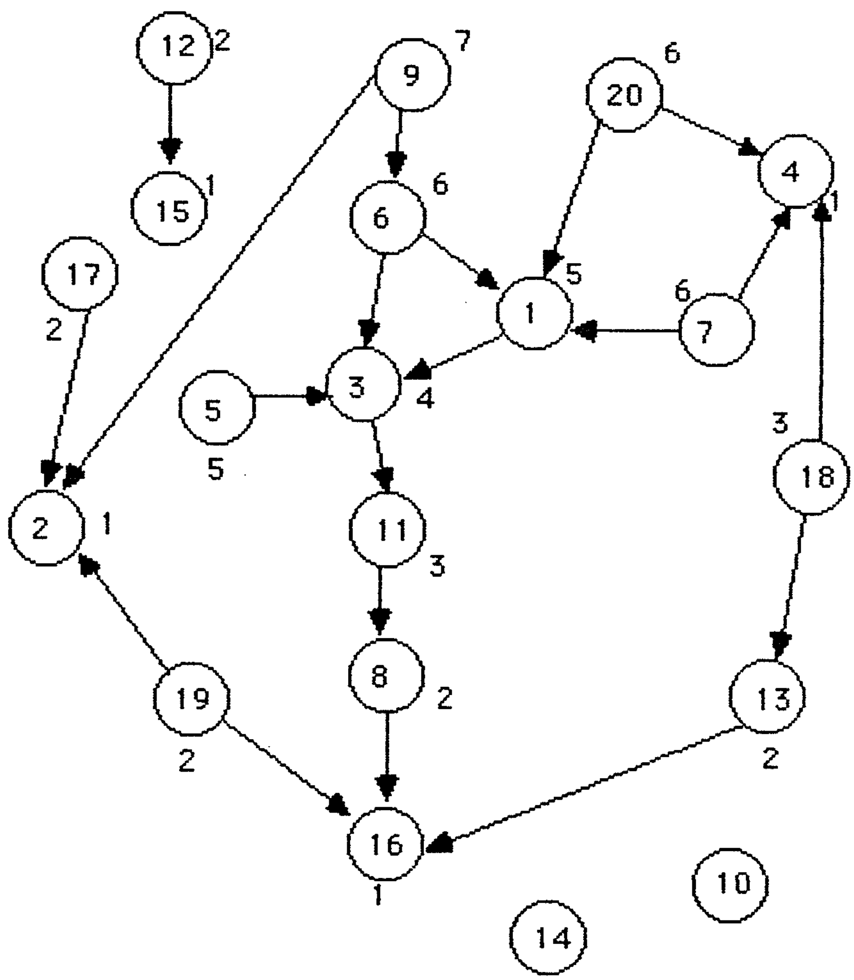


Fig. 23. The directed net relation (DNR) graph for Example B

The parameters in the priority function of Eq. 5.3 are the same as those for the first example.

First, we select net 9 as net MOTHER NET A, thus READYNETS ={7, 10, 17, 18, 19, 20}. The priority numbers of nets 7, 10, 17, 18, 19, and 20 are 63, 10, 35, 22, 16, and 49, respectively. The optimum subset S_0 is $S_0=\{7\}$. This process of selection is shown in the first row of Table V. Each bold net in each row of Table V is an element of the S_0 with respect to each MOTHER NET A shown in the left-most column. The realization of the example according to the results of this algorithm is shown in Fig. 24. Obviously, it is an optimum one because the number of the horizontal tracks used is equal to the maximum density number.

5.8 Summary

In this chapter, we discussed the channel routing problem, which appears to be the most important problem in LSI and VLSI layout. Since LSI and VLSI circuits are more complex than PCB

TABLE V. Selection of MOTHER NET and READYNETS for example B

net selected as MOTHER NET	net(s) in READYNETS (priority number of this net)					
9	**7(136)**	10(17)	17(40)	18(17)	19(12)	20(76)
20	5(14)	**10(66)**	**14(98)**			
6						
1	**18(17)**	**19(12)**				
5	4(43)	**13(46)**	17(40)			
3	**4(43)**	17(40)				
11						
17	**8(10)**					
12						
2						
16						
15						

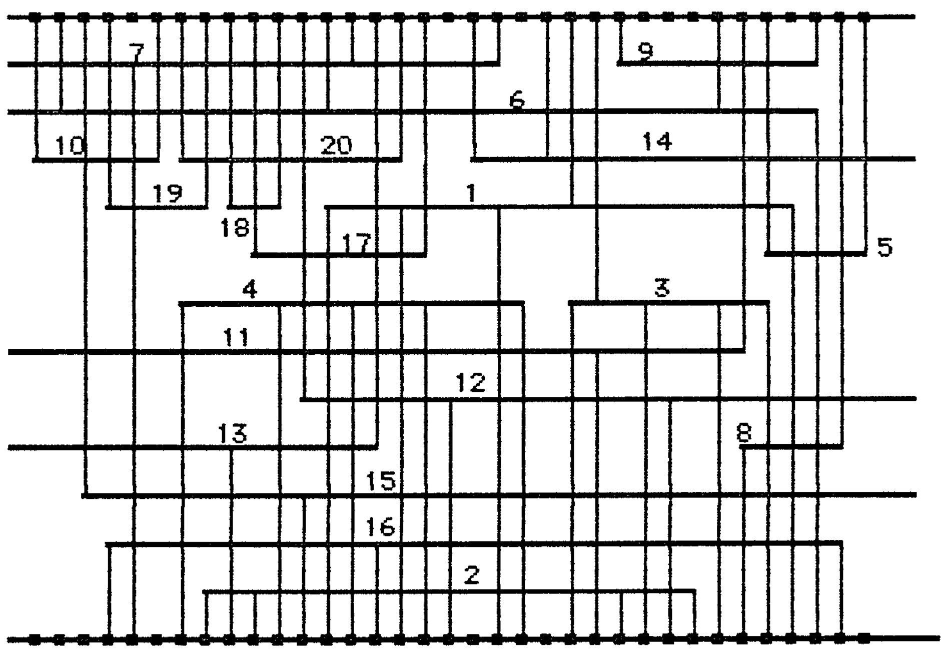


Fig. 24. The realization of Example B by using algorithm

circuits, and since the maze routing and line routing algorithms lack the ability to consider the routing globally, these algorithms are not suitable for the LSI and VLSI circuit layout. A routing strategy suitable for LSI and VLSI IC layout is channel routing. The objective of a channel router is to route all the nets within a channel by using the fewest horizontal tracks. A new track assignment algorithm has been developed to solve this problem, especially for the kind of problems whose lower bound of the solution is determined by the maximum ordering number in a DNR graph rather than the maximum density number. The efficiency of this algorithm was illustrated by two examples.

CHAPTER VI

EXPERIMENTAL RESULTS AND DISCUSSION

6.1 Experiment with Restricted MOTHER NET Selection Strategy

The new track assignment algorithm is implemented in PASCAL code. A flow-chart and the explanation of the implementation can be found in Appedix A, the source program is in Appedix B. Thirteen examples are used to test the new channel routing algorithm. Seven examples (Examples 5, 6, 7, 9, 10, 11, and 12) are taken from the existing paper [71]. The experimental results of the thirteen examples are shown in column four of Table VI.

The priority function used in the program is slightly different from the one shown in Section 5.6:

$$\begin{aligned} P(NET,A) = \ & Klen*Length \\ & +Kdis*(Mdis-Distance) \\ & +Kord*Order\_number/Curr\_order \end{aligned} \qquad (6.1)$$

where Length is the length of the net NET, Mdis is the maximum distance among the distances between every net in READYNETS and the MOTHER NET A, Distance is the distance between net NET and A, Order_number is the ordering number of the net NET, Curr_order is the current maximum ordering number in the DNR graph, and Klen, Kdis, and Kord are the weight factors of Length, distance, and Ordering number, respectively and they are assigned the value of 1, 1, and 5 in experiments. To calculate the priorities of the nets with different ordering numbers, the last term uses the ratio between the ordering number of the net NET and the current maximum ordering number instead of the ordering number itself. This approach equalizes the contribution of the ordering numbers of the nets to the priorities of the nets throughout the track assignment procedure.

Section 5.6.3 describes the procedure for the selection of an optimum subset S_0 for problems with the maximum density number being not greater than the maximum ordering number. For problems with the maximum density number larger than the maximum ordering number in the DNR graph, we may miss the net which crosses the vertical track where the maximum density occurs. Therefore, we should modify the definition of the optimum subset S_0 as follows:

An optimum subset S_0 is an eligible subset of READYNETS such that (i) one of the nets in it should cross the vertical track where the maximum net density exists, if it is possible (this criterion is called in-density-check); and (ii) the sum of the priority numbers of all the nets in this subset should have the largest value among all subsets which satisfy condition (i).

The nets in the new optimum subset S_0, together with MOTHER NET A, are routed in one track. The results are the same as those shown in the fourth column of Table VI. That is, the in-density-checking does not improve the performance of the router. So, the priority number of each net fairly well reflects its priority in routing it in the current track.

TABLE VI. Track assignment results.

Examples	Maximum Ordering Number	Maximum Density Number	Track number Required by algorithms		Optimum Track Number
			Restrict Moth-net selection	Relaxed Moth-net selection	
1	4	5	5	5	5
2	2	4	4	4	4
3	3	3	3	4	3
4	5	5	5	6	5
5	6	18	19	18	18
6	23	19	31	30	28
7	3	20	20	20	20
8	7	12	12	12	12
9	13	17	18	18	17
10	7	12	12	12	12
11	4	15	16	15	15
12	9	17	18	18	17
13	5	5	5	5	5

6.2 Experiment with Relaxed MOTHER NET Selection Strategy

Next, we modify the algorithm by relaxing the condition for selecting MOTHER NET from

$$\text{currentorder-ordernum[Mother\_net]} < 1 \quad (6.2)$$

to

$$\text{currentorder-ordernum[Mother\_net]} < 2 \quad (6.3)$$

That is, we do not restrict the selection of the MOTHER NET to those nets whose vertices in the current DNR graph have the maximum ordering number. The results for the thirteen examples are shown in the fifth column of Table VI. We can see that two more examples (the examples 5 and 11) get the optimum realizations and for both of them the maximum density number is much larger than maximum ordering number.

For all the above thirteen examples, the best realizations produced by the new channel routing algorithm are shown in Fig. 25-37.

6.3 Discussion

In this section, we discuss the results shown in the fourth and fifth columns in details. When the maximum ordering number is larger than (or equal to) the maximum density number, the maximum ordering number becomes the lower bound of the number of tracks required for the realization. Thus, to guarantee that each track has one of the nets in the longest chain of the DNR graph is essential for the realization to be optimum (or the number of tracks used reaches the lower bound of the track number required). This is why we should select MOTHER NET only from those unassigned nets which have the highest ordering number in the DNR graph, if the maximum ordering number is larger (or equal to) the maximum density number of the problem. This was proved by the results of Example 3 and Example 4 from columns four and five. When we restrict the router to select the MOTHER NET only from those unassigned nets whose ordering number is the largest in the current DNR graph, it produced the optimum results for these two examples. Both of these two examples have the lower bound of the required number of tracks determined by the maximum ordering number. But if we relaxed the MOTHER NET selection condition from Eq. 6.2 to Eq. 6.3, the realizations produced are no longer the optimum ones.

On the other hand, when the maximum density number is larger than the maximum ordering number, the minimum number of tracks is determined by the maximum density number. It is more important to use the current track efficiently rather than to assign the nets with current highest ordering number. We still give higher priority to the nets having the higher ordering number, so we can minimize the possibility of the case that more tracks are required at the final stage of the track assignment because of the vertical constraint. Thus, we do not restrict ourselves to select MOTHER NET only from unassigned nets with current highest ordering number in this case. The improvement of realizations of Examples 5 and 11 shows this clearly. For both of them, the maximum density number is much larger than the maximum ordering number. The relaxed MOTHER NET selection condition results in more efficient usage of the tracks and therefore, the track number required is reduced and the optimum realizations are reached.

Example 6 (the so-called difficult example) shows that the minimum requirement of the number of the horizontal tracks is equal to neither the maximum density number nor the maximum ordering number for example 6. This may be the reason why the relaxed MOTHER NET selection condition produces better realization. It is also worthy of mention that the realization of Example 6 requires 29 tracks if the relaxed MOTHER NET selection condition is used and the weight factor for the ordering number is changed from 5 to 10. This indicates the necessity of finding the optimum set(s) of the weight factors for the priority function of Eqs 5.1 and 6.1.

6.4 Summary

Thirteen examples were tested, using the new channel routing algorithm. The experimental results show that the new algorithm has very good performance on these examples, especially for one class of problems whose lower bound of track number required are determined by the maximum ordering number. This is because of the nature of this algorithm. The algorithm considers the ordering number of each net and gives the higher priorities to those nets having higher ordering number in the procedure of assigning nets to tracks. To cope with the examples whose lower bounds of the number of tracks required are determined by the maximum density number, a relaxed MOTHER NET selection strategy was also introduced and tested. The improvement of the results for this kind of problems was demonstrated through two examples.

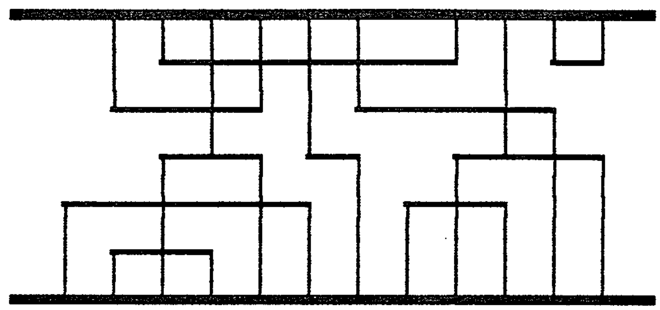


Fig. 25. Example 1

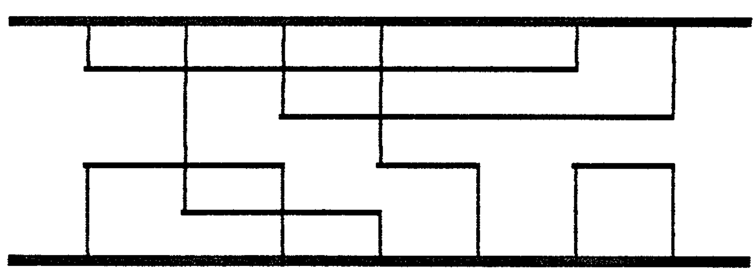


Fig. 26. Example 2

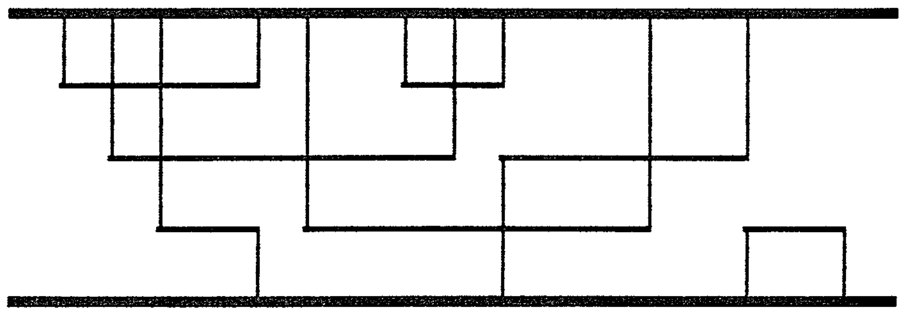


Fig. 27. Exampe 3

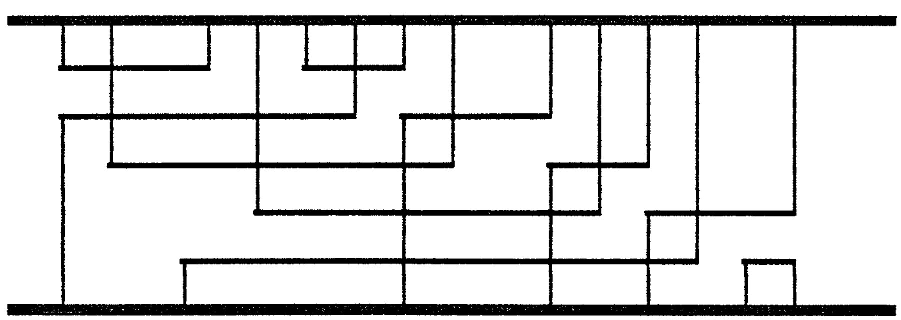


Fig. 28. Example 4

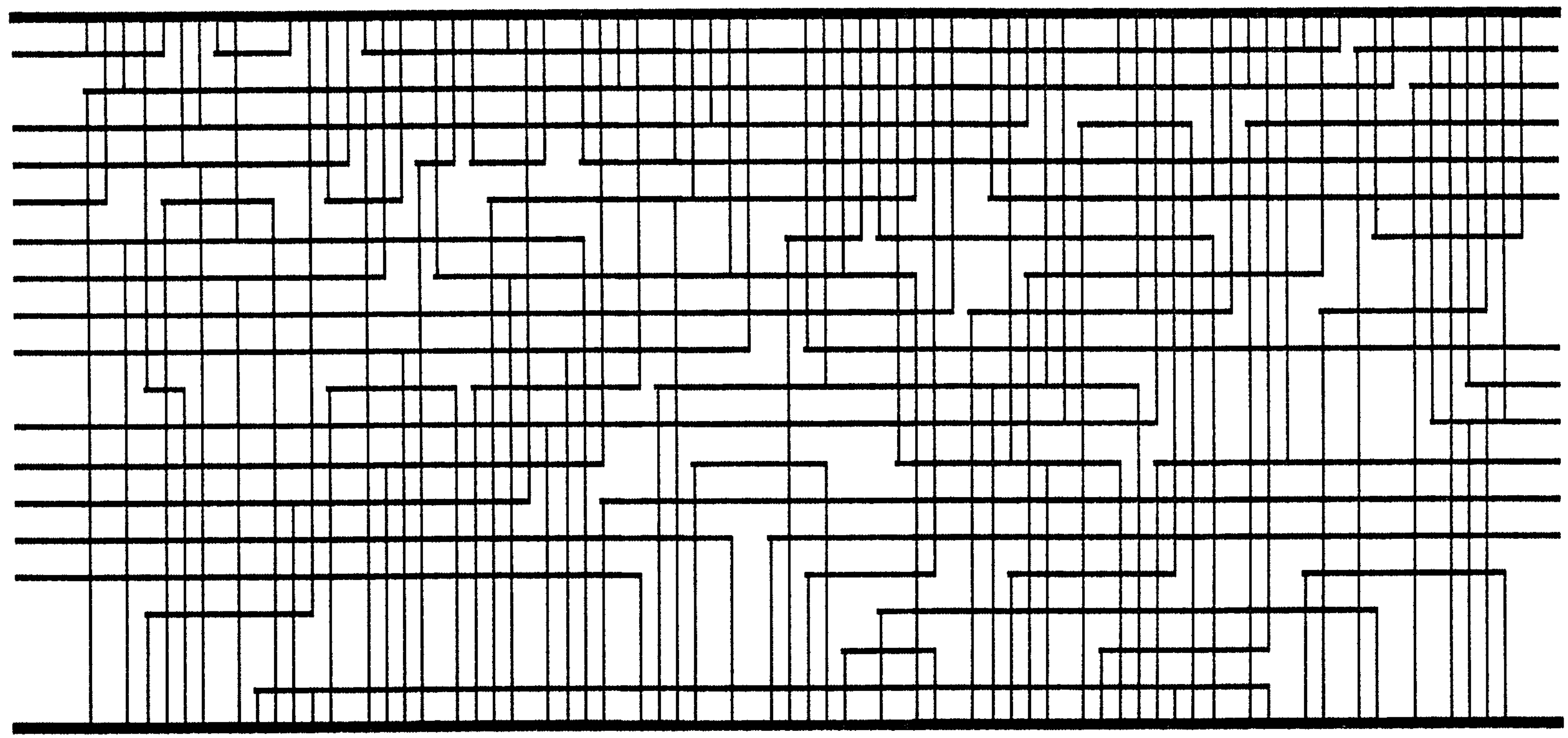


Fig. 29. Example 5

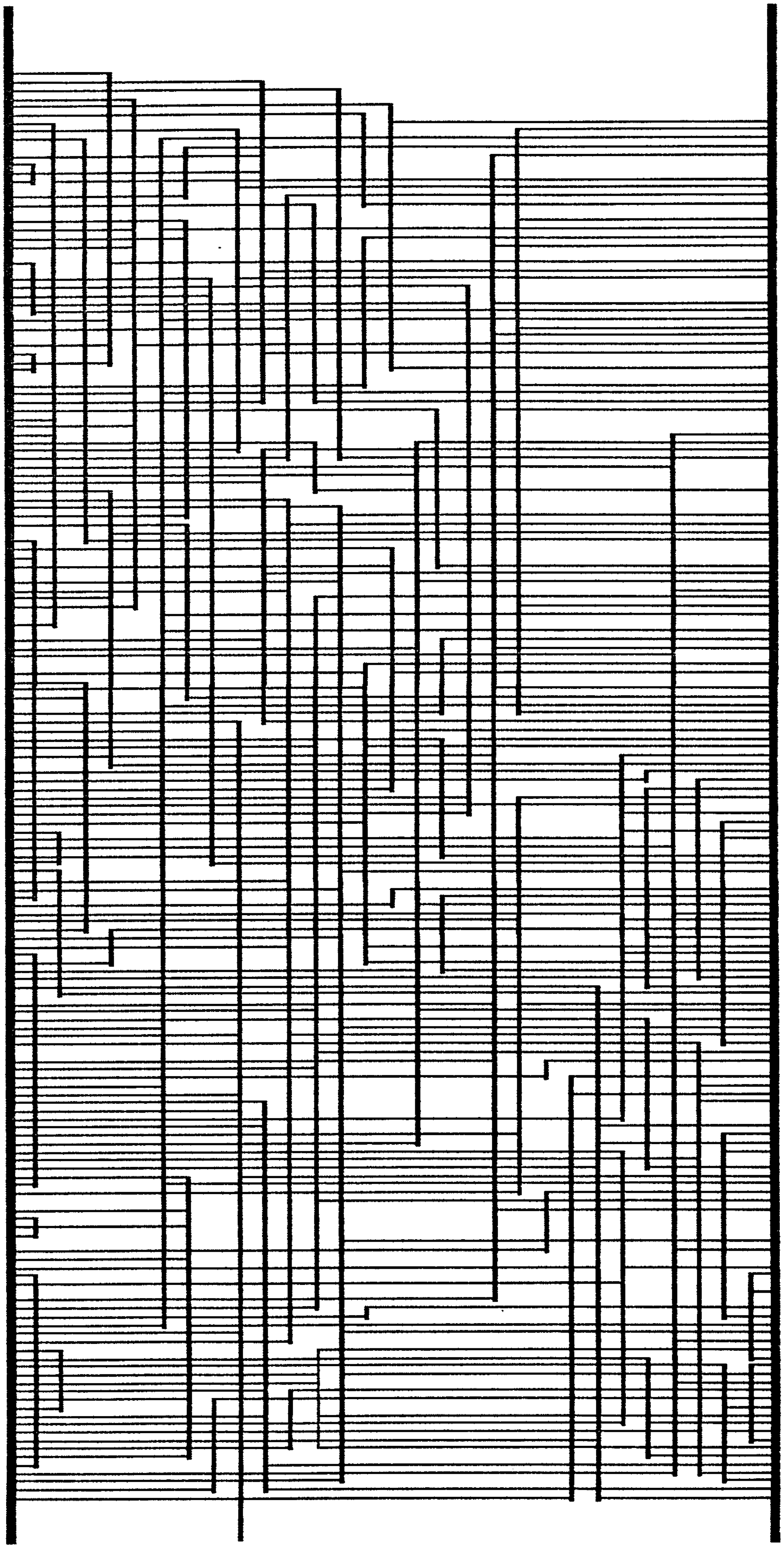


Fig. 30. Example 6

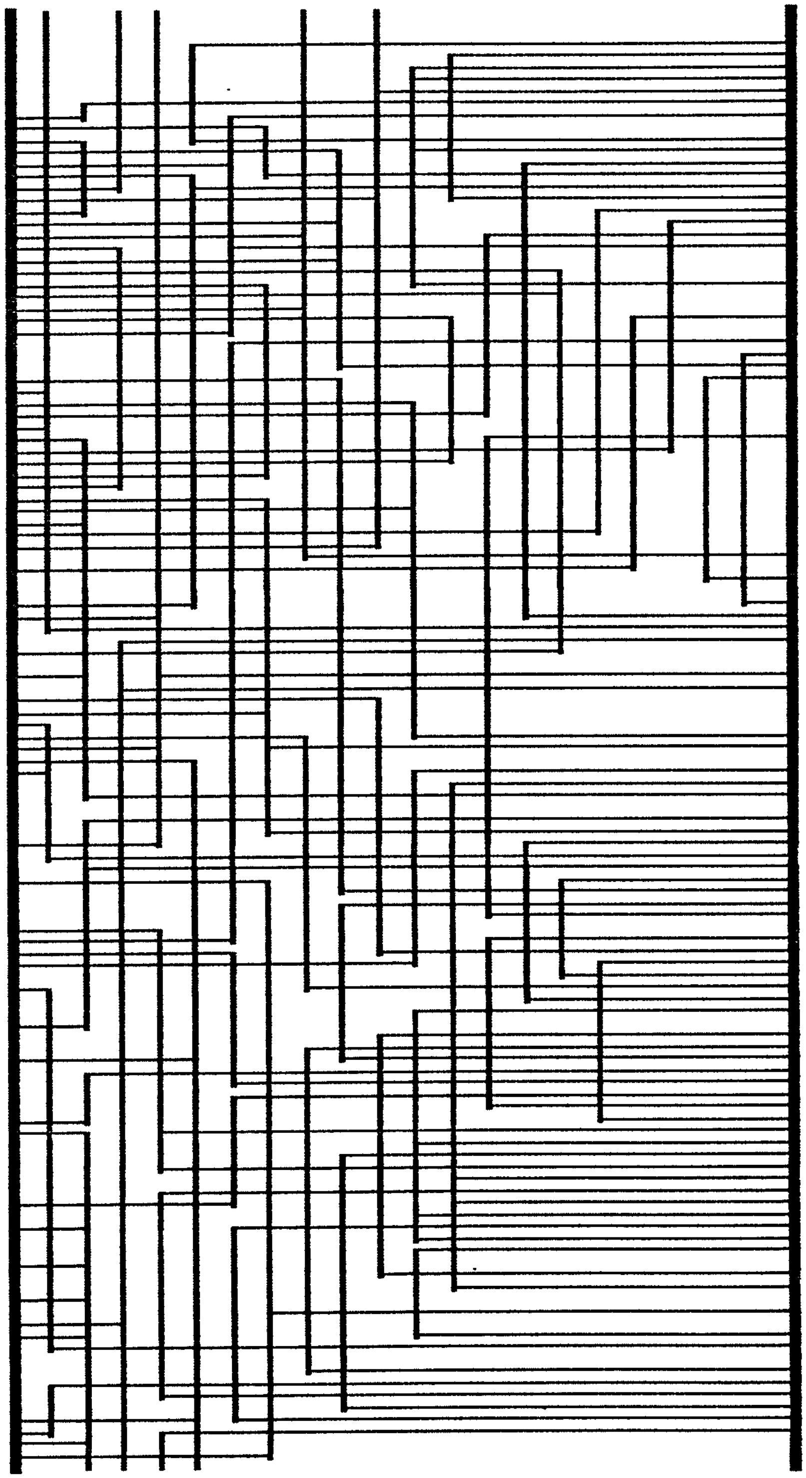


Fig. 31. Example 7

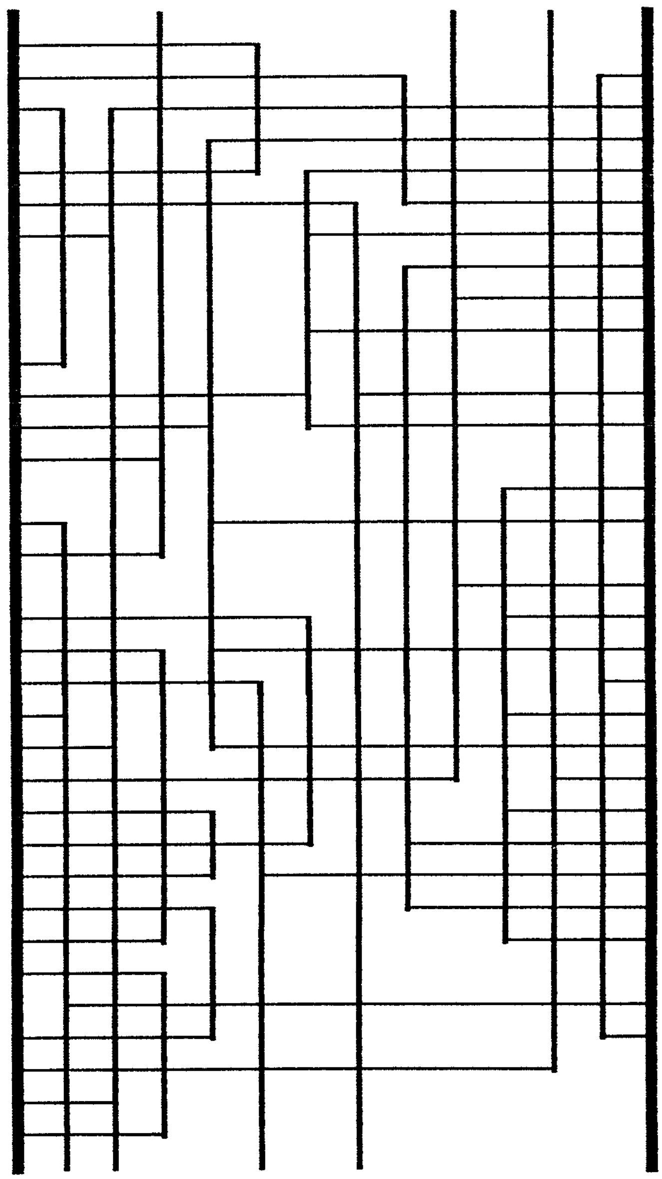


Fig. 32. Example 8

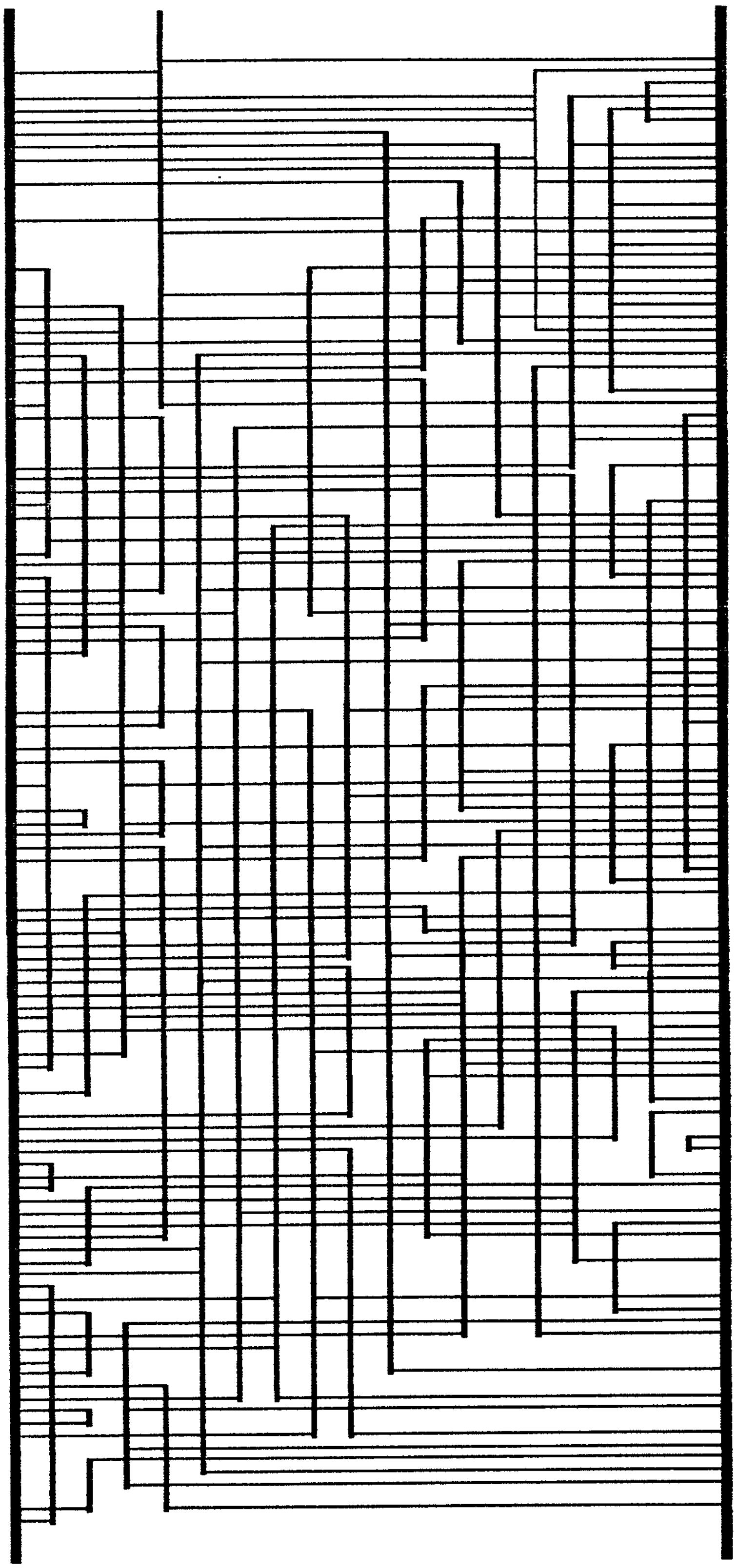


Fig. 33. Example 9

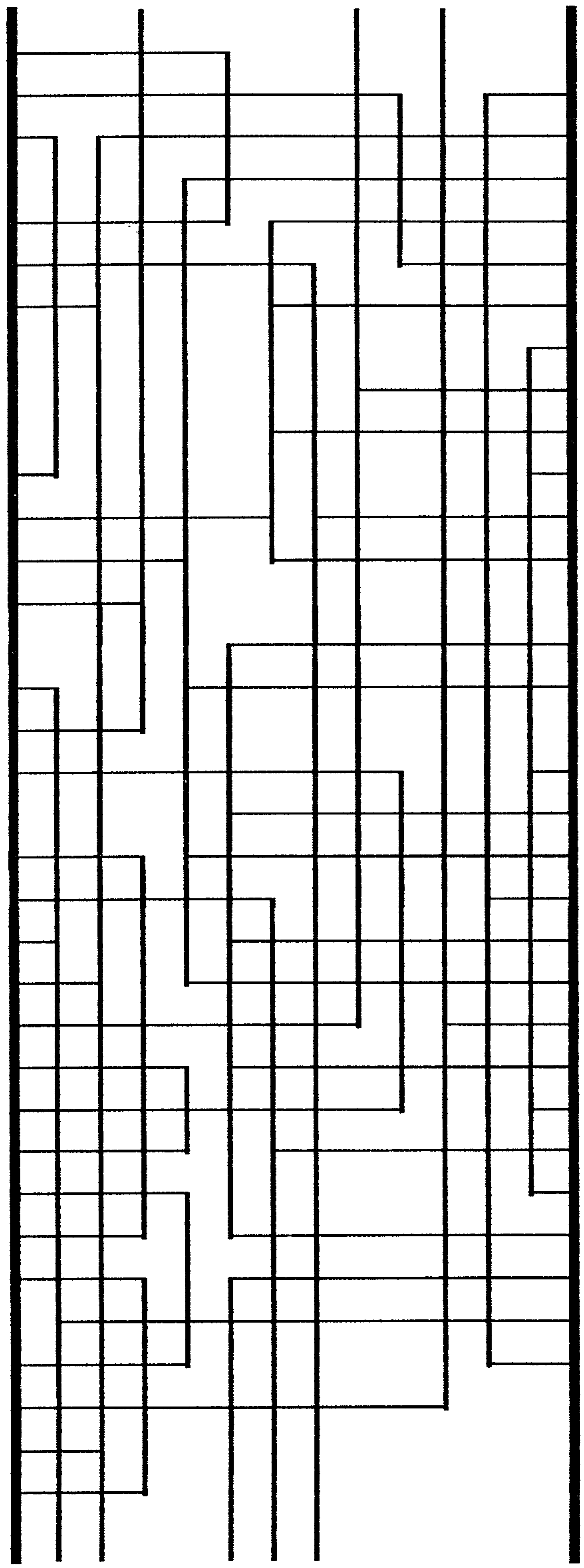


Fig. 34. Example 10

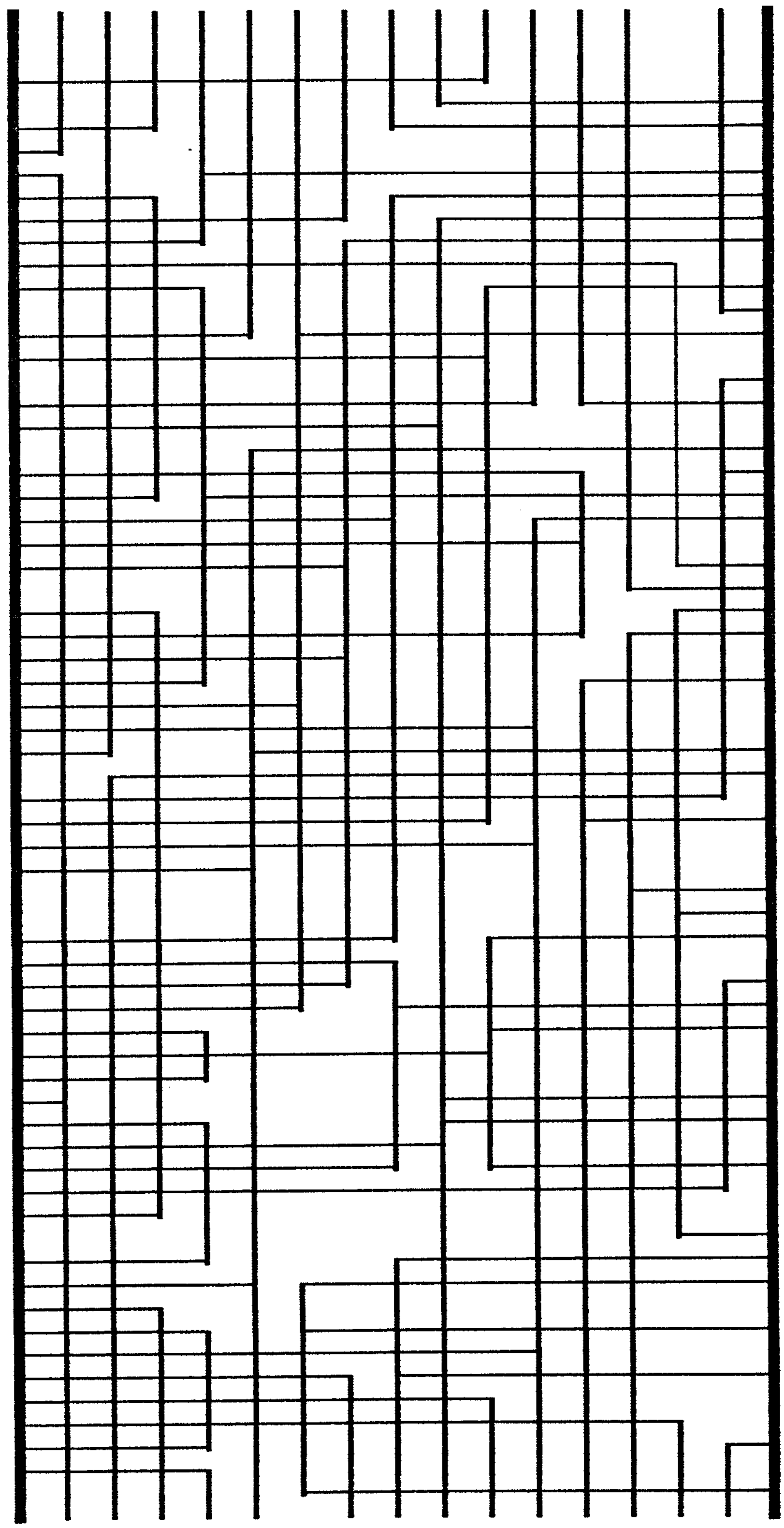


Fig. 35. Example 11

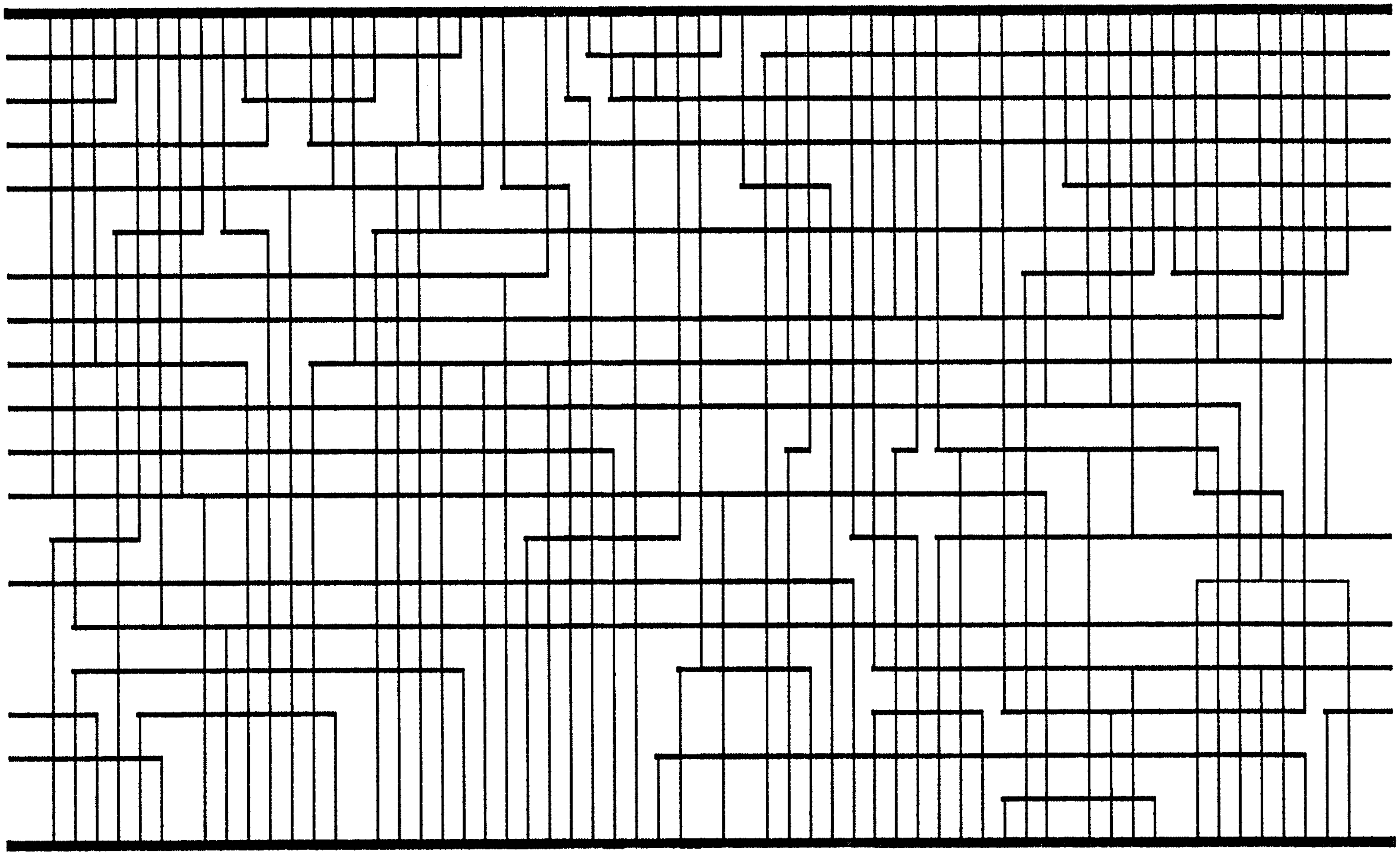


Fig. 36. Example 12

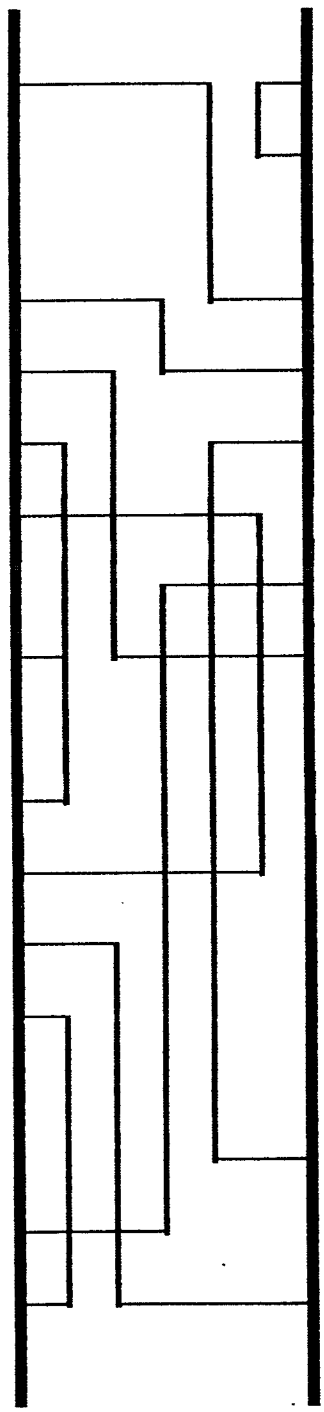


Fig. 37. Example 13

CHAPTER VII

COMPARISON WITH OTHER CHANNEL ROUTING ALGORITHMS

In Chapter V, we developed a new channel routing algorithm capable of finding optimum realizations for a class of channel routing problems. The routing problems of that class have the lower bound of the required track number determined by the maximum ordering number rather than the usual maximum density number. In order to demonstrate the advantages of the new algorithm, this chapter describes two other well-known channel routing algorithms: (i) the Left-Edge Greedy Algorithm, and (ii) the Zone-Based Algorithm. One example is used to illustrate their inability to generate optimum realizations for this class of problems. Finally, we address the reason why the new channel routing algorithm can produce results better than the other algorithms.

7.1 The Left-Edge Greedy Channel Routing Algorithm

The left-edge greedy channel router was first introduced by Hashimoto and Stevens [70]. We present this algorithm here to show that it cannot generate an optimum realization for an example belonging to the class discribed above.

Suppose that the nets to be assigned to tracks are contained in the set NETS. To simplify the description of the left-edge algorithm, we define the left and right edges of a net; that is the left edge of a net coincides with the leftmost vertical track spanned and the right edge of the net coincides with the rightmost vertical track. Clearly, for the routing to be realizable, no two nets must overlap in a single horizontal track. This can be assured by the requirement that the right edge of one net is located on the left side of the left edge of another net. A simple sorting by the left edge of all the nets in NETS resolves any horizontal trace conflicts, while minimizing the distance between adjacent nets in the same track. This greedy strategy may result in over-utilization of the first track, thus producing blockages for nets in the subsequent tracks.

The algorithm can then be stated as follows:

[Left-Edge Greedy Channel Router]

Step 1) Sort the nets in the set NETS by the left edge of each net. Thus, after sorting, the first net in the set NETS has a connection with the leftmost terminal.

Step 2) Assign the first net to the next track and delete this net from the set NETS.

Step 3) Find the first net in NETS so that its left edge is to the right of the right edge of the last net selected. Assign this net to the current track and delete this net from set NETS.

Step 4) Repeat Step 3 until no nets can be assigned to the track.

Step 5) If the set NETS is not empty, goto Step 2; otherwise exit.

□

7.1.1 Example C for the Left-Edge Greedy Algorithm

The example shown in Fig. 38 has ten nets to be assigned to tracks. Both the maximum ordering number in the corresponding DNR graph and the maximum density number are 5. So, the maximum

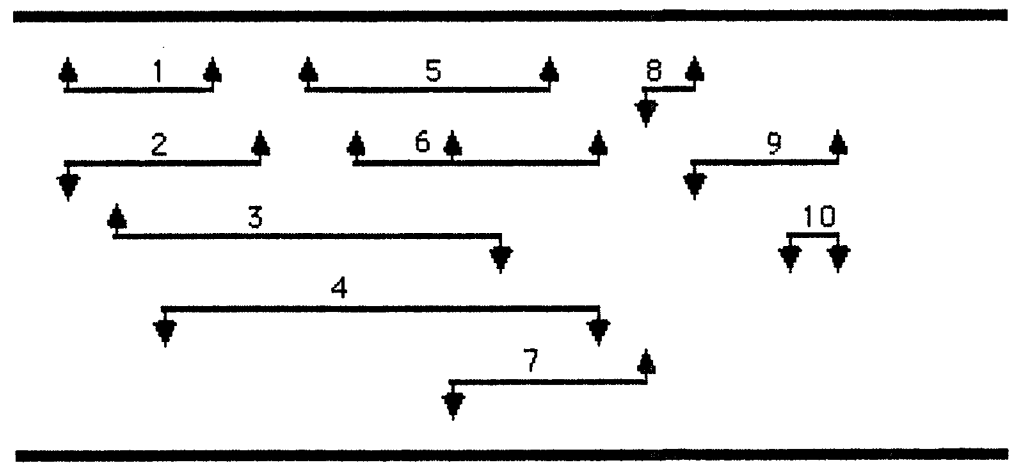


Fig. 38. Example C to illustrate the advantages of the new algorithm

ordering number determines the lower bound on the number of tracks required. The algorithm handles the example as follows:

Step 1) Sort the nets in NETS by their left edge. The sorting result is shown in Fig. 39-a.

Step 2) Assign net 1 to track 1.

Step 3) Assign net 5 to track 1.

Step 4) No other nets can be assigned to track 1 because of the vertical constraint.

Step 5) Goto Step 2.

Step 2) Assign net 2 to track 2.

Step 3) Assign net 6 to track 2.

Step 4) No other nets can be assigned to track 2 because of the vertical constraint.

Step 5) Goto Step 2.

Step 2) Assign net 3 to track 3.

Step 4) No other nets can be assigned to track 3 because of the vertical constraint.

Step 5) Goto Step 2.

Step 2) Assign net 4 to track 4.

Step 4) No other nets can be assigned to track 4 because of the

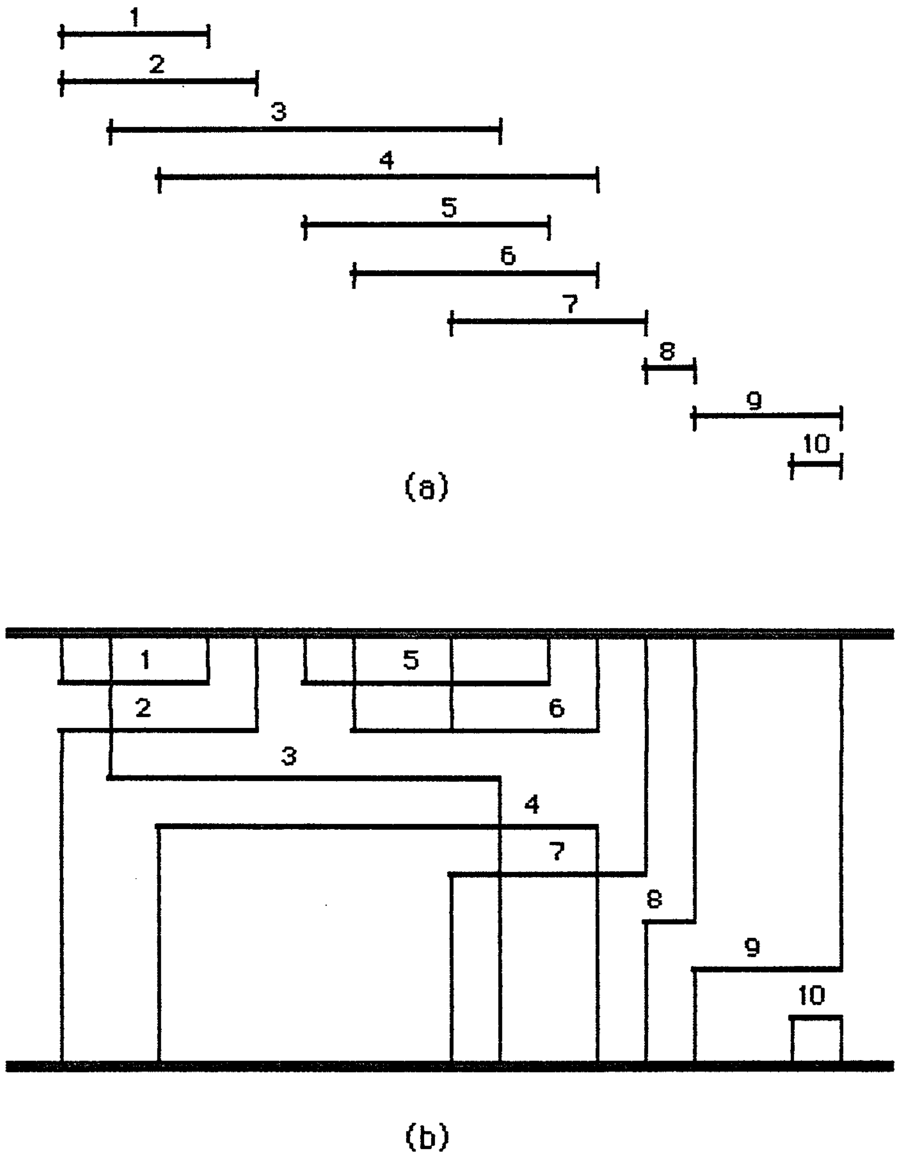


Fig. 39. Left-edge greedy channel routing algorithm

(a). Sorted result by left edge
(b). Realization using left-edge greedy algorithm

vertical constraint.

Step 5) Goto Step 2.

Step 2) Assign net 7 to track 5.

Step 4) No other nets can be assigned to track 5 because of the vertical constraint.

Step 5) Goto Step 2.

Step 2) Assign net 8 to track 6.

Step 4) No other nets can be assigned to track 6 because of the vertical constraint.

Step 5) Goto Step 2.

Step 2) Assign net 9 to track 7.

Step 4) No other nets can be assigned to track 7 because of the vertical constraint.

Step 5) Goto Step 2.

Step 2) Assign net 10 to track 8.

Step 5) All nets have been assigned. Exit.

□

Figure 39-b is the realization of Example C. It is seen that the ten nets are routed within 8 tracks, rather than the minimum of 5 tracks, resulting from the new channel routing algorithm (see Section 7.3).

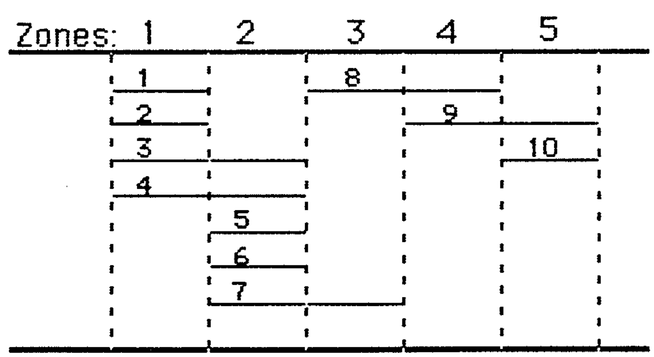
7.2 The Zone-Based Channel Routing Algorithm

Two efficient algorithms for channel routing were first introduced by Yoshimura and Kuh [71]. Algorithm #2 is an improved version of Algorithm #1. We present Algorithm #2 here to show that it cannot generate an optimum realization for the example shown in Fig. 38, either. The algorithm is also called the zone-based channel routing algorithm.

First, we introduce two definitions used in the zone-based algorithm: (i) the zone representation of horizontal segments, and (ii) the bipartite graph G_h. The zone representation of horizontal segments can be defined as follows: The concept of a vertical track originates from the local zone associated with a terminal in the channel, and therefore cannot carry information about groups of nets that must be placed in different horizontal tracks due to their inherent horizontal overlap. Yoshimura and Kuh introduced the concept of a zone containing such a group of nets that must be placed in separate horizontal tracks. Thus, zoning of a channel partitions the channel routing problem into smaller problems. A systematic method of zone creation may be based on the concept of local net density S(j) and a local maximum set $S_m(j)$. Let S(j) be the set of nets whose horizontal segments intersect a vertical track j associated with terminal j. A set S(j) is said to be the local maximum if it is not the proper subset of the adjacent set S(j-1) and S(j+1). The vertical track with the local maximum set, $S_m(j)$, together with all the vertical tracks associated with the proper subsets of $S_m(j)$ constitutes a zone. Figure 40 shows the zoning of a channel routing problem of Fig. 38. It is seen that zone 1 includes vertical tracks from 1 to 5. Note that the local maximum is spread over tracks 3 and 4. Also notice that the local maximum shows where the maximum net density occurs within the zone (The local net density is the number of element in the set S(j)). The Bipartite Graph $G_h(V, E)$ is a graph defined as follows: Each vertex i in G_h corresponds to a net, and an edge (i, k) between vertices i and k exists if and only if nets i and k can be assigned to the same track without having horizontal track overlapping.

Column (Vertical Track)	S(I)	Zone
1	1, 2	
2	1, 2, 3	
3	1, 2, 3, 4	1
4	1, 2, 3, 4	
5	2, 3, 4	
6	3, 4, 5	
7	3, 4, 5, 6	
8	3, 4, 5, 6	
9	3, 4, 5, 6, 7	2
10	3, 4, 5, 6, 7	
11	4, 5, 6, 7	
12	4, 6, 7	
13	7, 8	3
14	8, 9	4
15	9	
16	9, 10	5
17	9, 10	

(a)



(b)

Fig. 40. Efficient channel routing algorithm

(a). Zone generation for the example

(b). Zone representation for the example

The zone-based algorithm can then be stated as follows:

[Zone-Based Channel Router]

Step 0) Initialization: Set i=1.

Step 1) For each net n_l terminating at zone Z_i, add the corresponding vertex n_l to the left side of the bipartite graph G_h.

Step 2) For each net n_r which begins at zone Z_{i+1}, add the corresponding vertex n_r to the right side of the bipartite graph G_h, and add edges between the vertex n_r and a vertex on the left side, if they can be merged (it means no horizontal overlapping between the corresponding nets); then, find a maximum matching (maximum matching means that the number of pairs of vertices on the left and right sides of the bipartite graph which can be merged is maximized).

Step 3) Check if the merging based on the current matching satisfies the vertical constraints (Vertical constraints refer to the vertical trace overlapping). If not, modify the matching so that no vertical constraints are violated.

Step 4) For each net n_1 terminating at Z_{i+1}, merge the corresponding vertex n_1 in the bipartite graph with the vertex n_x specified by the matching on graph G_h. Then replace the vertex n_x with the merged vertex $n_1 \bullet n_x$ on the left side of G_h.

Step 5) Increment the zone count $i=i+1$. If Z_i is not the last zone, then repeat from Step 1.

Step 6) Based on the final bipartite graph G_h, assign nets to the proper tracks by taking the vertical constraints into consideration. The number of vertices is the number of tracks required for the realization of the problem. The vertices merged as a single vertex in the final bipartite graph G_h imply that their corresponding nets occupy one track in the realization of the problem.

□

7.2.1 Example C for the Zone-Based Algorithm

We now show that the zone-based channel routing algorithm can not generate an optimum realization for the example shown in Fig. 38, either. The algorithm progresses on the example as follows:

Step 1) Put vertices for nets 1 and 2 to the left side of the bipartite graph G_h.

Step 2) Add vertices for nets 5, 6, and 7 to the right side of graph G_h and add edges between vertices which can be merged, a maximum matching is 1-5 and 2-6 (The matching is shown by thicker lines in Fig. 41-a.).

Step 3) The matching satisfies the vertical constraints.

Step 4) Since nets 5 and 6 terminate at zone Z_2(shown in Fig. 41-a by thicker circles), merge their vertices with those for nets 1 and 2, respectively. Since nets 3 and 4 terminate at Z_2, add their vertices to the left side of the graph G_h (see Fig. 41-b). Notice that they merge with no other vertices.

Step 5) i=2. Goto Step 2.

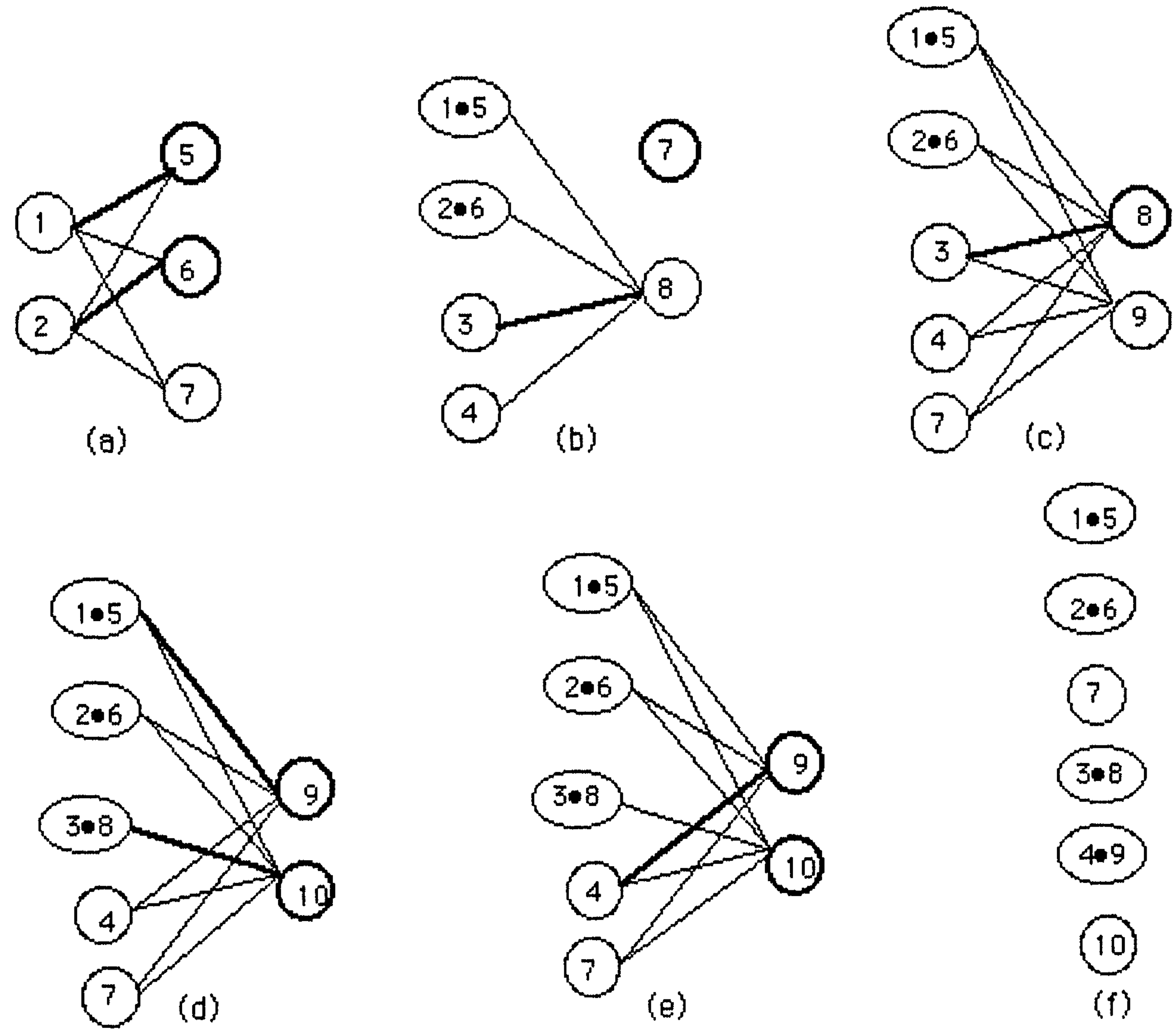


Fig. 41. Bipartite graph for the example at different stages
(a) i=1 (b) i=2 (c) i=3 (d) i=4 (e) i=4
(f) Merged G_h

Step 2) Add the vertex for net 8 to the right side of the graph G_h, and add the corresponding edges to the graph, too. The maximum matching is 3-8 (Fig. 41-b).

Step 3) The matching satisfies the vertical constraints.

Step 4) Since there is no match for the vertex of net 7, which terminates at zone Z_3, the vertex 7 is merged with no vertex on the left side of the bipartite graph G_h.

Step 5) i=3. Goto Step 2.

Step 2) Add the vertex for net 9 to the right side of the graph G_h, and add edges to the graph accordingly. A maximum matching is 3-8 and 4-9 (Fig. 41-c).

Step 3) The matching satisfies the vertical constraints.

Step 4) Net 8 terminates at zone Z_4, the vertices for net 8 and net 3 are merged.

Step 5) i=4. Goto Step 2.

Step 2) Add the vertex for net 10 to the right side of the graph G_h, and add corresponding edges to the graph G_h. Maximum matching could be (1•5)-9 and (3•8)-10 (Fig. 41-d).

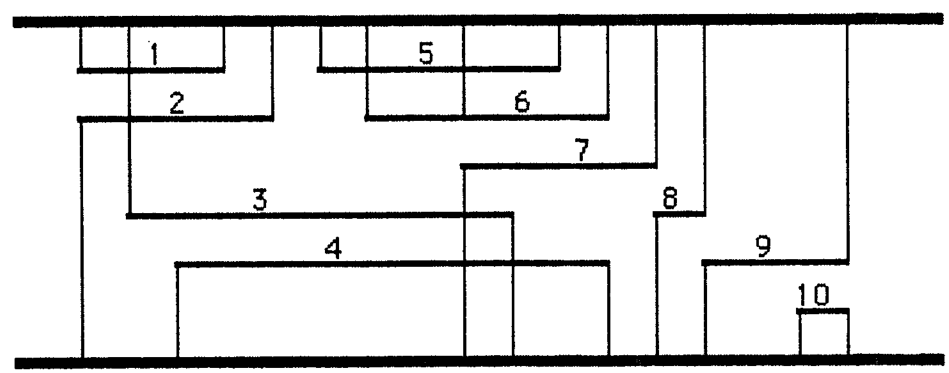


Fig. 42. Realization of the example using efficient routing algorithm

Step 3) The matching violates the vertical constraints. Only possible matching is 4-9 (Fig. 41-e).

Step 4) Merge vertices for net 4 and net 9 (Fig. 41-f).

Step 5) i=5. Z_i is the last zone.

Step 6) Based on Fig. 41-f, the realization of the problem is shown in Fig. 42.

□

Once again, the realization of Fig. 42 requires six tracks, and therefore is not an optimum one for Example C.

7.3 Example C for the New Track Assignment Algorithm

The DNR graph corresponding to the same example of Fig. 38 is shown in Fig. 43. Figure 44 shows the realization of the example obtained by the new track assignment routing algorithm. This optimum realization needs only 5 tracks.

Example C has a maximum density number 5 and a maximum ordering number 5. So, the lower bound of the track number required

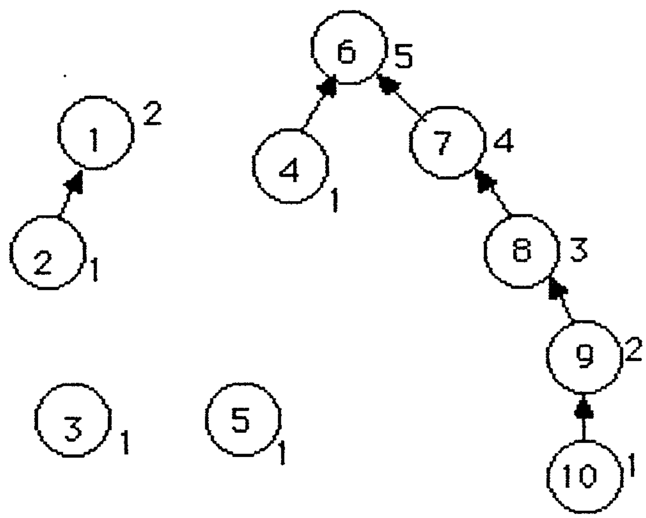


Fig. 43. Directed net relation graph for the example

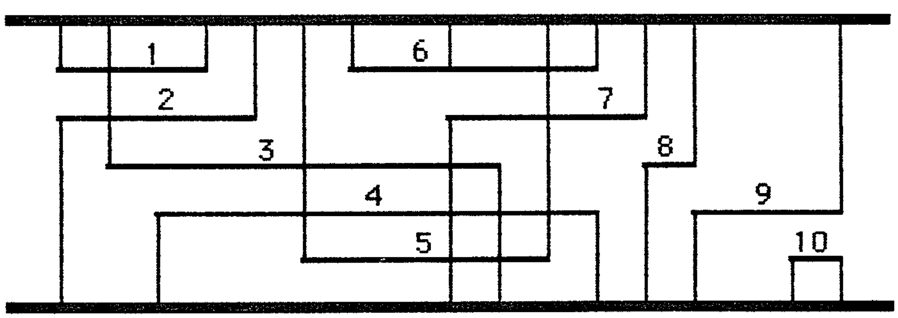


Fig. 44. Realization of the example using new channel routing algorithm

for realization of the problem is determined by the maximum ordering number. Thus, it is necessary to have one net in the longest chain in DNR graph (Fig. 43) in every track in order to get the optimum result. The left-edge greedy channel router does not consider the ordering number of each net at all. This results in the tendency to use the first track efficiently. However, this heavy usage of the first track may inhibit the efficient usage of subsequent tracks because of the vertical track constraint. Similarly, the zone-based channel router fails to generate optimum results because the router considers only the vertical constraints locally between adjacent zones.

In contrast, the new channel router does consider the fact of ordering number for each net, and gives a higher priority to the nets with a higher ordering number in the process of assigning them to the tracks. This feature yields optimum realizations for the problems of the class in which the lower bound of the number of tracks required for the realization of the problem is determined by the maximum ordering number.

7.4 Summary

This chapter described two well-known channel routing algorithms, the greedy algorithm and the zone-based algorithm, and compares their performance with the performance of the new track assignment algorithm. As illustrated by an example, the two algorithms cannot generate optimum realizations for one class of problems whose lower bound of the number of the horizontal tracks required is determined by the maximum ordering number in DNR graph rather than the maximum density number.

CHAPTER VIII

CONCLUSIONS AND RECOMMENDATIONS

Routing is a very challenging problem in circuit design because it produces the physical circuit layout which may greatly affect the performance and quality of the circuit. First of all, the quality of the physical layout depends on the quality of the component placement. For a given placement configuration, very different physical layouts can be achieved by using existing routing strategies.

Many routing algorithms have been developed for the circuit layout. Lee's type (or maze) algorithms have dominated the design of printed circuit boards. Their popularity can be attributed to their ability to find a connection whenever the connection exists. The maze algorithms however, are not only inefficient in terms of space and time required to complete the routing, but also may generate connections of undesirable shapes. Many improvements of Lee's algorithm have been developed to achieve better results.

Improved routing results may be obtained not only by the use of improved algorithms, but also by the application of optimum sets of routing parameters. Furthermore, for the same sets of routing parameters, different sequences of applying them may lead to very different routing results. This was demonstrated through experiments. Thus, to study how to apply the routing parameters is as important as to study the algorithms themselves.

In LSI and VLSI design using gate array and standard cell methodologies, channel routing algorithms are preferred, because the regular shapes of modules used in these circuits and their arrangement within the chips result in the rectangular areas between modules, where the channel router must produce all the interconnections. Although several algorithms have been developed for such a channel routing (or track assignment) problem, none of them can cope with the case when the maximum ordering number of the routing problem is larger than (or equal to) the maximum density number. Therefore, we developed a new channel routing (track assignment) algorithm capable of producing optimum realizations of problems of that class.

As demonstrated in the previous chapters, this thesis has contributed to general and technical knowledge by achieving the following results:

a) Investigated and evaluated the algorithms suitable for routing of printed circuit boards, especially those algorithms of the Lee type. The investigation concentrated on the issue of storage requirement and expansion area.

b) Established a relationship between the routing history and routing results. The relationship was investigated by observing the routing performance under selected routing histories, using the OPTIMATE™ computer-aided design software package, running on the Apollo DN660 computer.

c) Developed empirical formulae for predicting circuit routability based on the information given by a placement configuration. Though the formulae themselves cannot improve the placement configuration in terms of routability, they could be very helpful in the evaluation of the different placement configurations efficiently.

d) Developed a channel routing (track assignment) algorithm which can generate optimum realizations for a class of problems for which other channel routing algorithms often give only suboptimum realizations. The optimality of the new algorithm was demonstrated through experimentation on the Apollo DN660 computer.

Although the conclusions drawn are important to the theory and practice of CAE, and our understanding of the subject has increased considerably through the work done, many other important questions have been discovered. Therefore, the following research is recommended to improve the work done and presented in this thesis:

a) The relation between other sets of routing histories and the routing performance should be investigated in order to seach for better routing histories that could achieve better routing performance.

b) Programs for collecting all the data required for calculating the routability indicator can be developed in order to improve the speed and accuracy of the calculation. Further experiments with different routing parameters could lead to an improved routability indicator capable of predicting the level of circuit routing completion accurately, and therefore influencing circuit placement more accurately.

c) The new channel routing algorithm should be modified in order to cope with other classes of problems; that is, for the case when the maximum density number is larger than the maximum ordering number. This might be solved by modifying the criteria for selecting the MOTHER NET. Also, an optimum set of the weight factors in the priority function needs to be found in order to achieve the optimum results.

REFERENCES

[1] A. Wexler, "Getting a handle on impedance, cross-talk, time delay, and ringing," Printed Circuit Design, pp.14-17, December 1985.

[2] W. Kinsner, Computer-Aided Engineering of Printed Circuit Boards. Course notes, Microelectronics Centre and University of Manitoba, Winnipeg, Canada; July 1985, 300 pp.

[3] J. Soukup, "Circuit layout," Proceedings of IEEE, vol. 69, pp. 1281-1304, October 1981.

[4] C. Lee, "An algorithm for connections and its applications," IRE Transactions on Electronic Computer, vol. EC-10, pp. 346-365, September 1961.

[5] D. Hightower, "A solution to line routing problems on the continuous plane," Proceedings of 6th Design Automation Workshop, vol. 6, pp. 1-24, 1969.

[6] W. Heyns, "A line-expansion algorithm for the general routing problem with a guaranteed solution," Proceedings of 17th Design Automation Conference, vol. 17, pp. 243-249, 1980.

[7] G. Patterson and B. Phillips, "A proven operational CAD system for PWB design--based on a mini-computer and featuring fully automatic placement and routing," Proceedings of 13th Design Automation Conference, vol. 13, pp. 259-264, 1976.

[8] G. Rabbat, "VLSI routing," in Hardware and Software Concepts in VLSI, pp. 368-405, Van Norstrand Reinhold Company, 1983.

[9] R. Joseph, "An expert system approach to completing partially routing printed circuit boards," Proceedings of 22nd Design Automation Conference, vol. 22, pp. 523-528, 1985.

[10] R. Joobbani, "WEAVER: A knowledge-based routing expert," Proceedings of 22nd Design Automation Conference, vol. 22, pp.266-272, 1985.

[11] R. Joobbani, An Artificial Intelligence Approach to VLSI Routing, Kluwer Academic Publishers, 1986.

[12] VLSI System Design staff, "Survey of circuit board CAD systems," VLSI System Design, vol. VII, pp. 64-77, March 1986.

[13] M. Breuer and K. Shamsa, "A hardware router," Journal of Digital System, Vol. 4, No. 4, pp. 393-408, 1981.

[14] A. Iosupovici, "A class of array architectures for hardware grid routers," IEEE Transactions on Computer-Aided Design, vol. CAD-5, pp. 245-255, April 1986.

[15] S. Akers, "A modification of Lee's path connection algorithm," IEEE Transaction on Electronic Computers, vol. EC-16. pp. 97-98, January 1967.

[16] F. Rubin, "The Lee path connection algorithm," IEEE Transactions on Computers, vol. C-23, pp.907-914, September 1974.

[17] R. Korn, "An efficient variable-cost maze router," Proceedinds of 19th Design Automation Conference, vol. 19, pp. 425-431, 1982.

[18] J. Hoel, "Some variations of Lee's algorithm," IEEE Transactions on Computers, vol. C-25, pp. 19-24, January 1976.

[19] F. Tada, K. Yoshimura, T. Kagata, and T. Shirakawa, "A fast maze router with iterative use of variable search space restriction," Proceedings of 17th Design Automation Conference, vol. 17, pp. 250-254, 1980.

[20] J. Geyer, "Connection routing algorithm for Printed Circuit Boards," IEEE Transactions on Circuit Theory, vol. CT-18, pp. 95-100, January 1971.

[21] T. Asano, "Parametric Pattern Router," Proceedings of 19th Design Automation Conference, vol. 19, pp. 411-417, 1982.

[22] W. Wu and D. Schmidt, "A new routing algorithm for two-sided boards with floating vias," Proceedings of 13th Design Automation Conference, vol. 13, pp. 151-160, 1976.

[23] C. Fisk, D. Caskey, and L. West, "ACCEL: Automated Circuit Card Etching Layout," Proceedings of the IEEE, vol. 25, November 1967.

[24] P. Agrawal and M. A Breuer, "Experiments with a density router for PC cards," IEEE Transactions on Computers, vol. C-28, pp. 262-267, March 1979.

[25] S. Futagami, I. Shira-Kawa, and H. Ozaki "An automatic routing system for single-layer printed wiring boards," IEEE Transactions on Circuit and Systems, vol. CAS-29, pp. 46-51, January 1982.

[26] R. Chen, Y. Kajitani, and S. Chan "A graph-theoretic via minimization algorithm for two-layer printed circuit boards," IEEE Transactions on Circuit and Systems, vol. CAS-30, pp. 284-299, May 1983.

[27] M. Marek-Sadowska, "An unconstrained topological via minimization problem for two-layer routing," IEEE Transactions on Computer-Aided Design, vol. CAD-3, pp. 184-190, July 1984.

[28] B. S. Ting, E. S. Kuh, and I. Shirakawa, "The multilayer routing problem: algorithms and necessary and sufficient conditions for the single-row single-layer case," IEEE Transaction on Circuits and Systems, vol. CAS-23, pp.768-778, 1976.

[29] E. S. Kuh, T. Kashewabara, and T. Fujisawa, "On optimum single-row routing," IEEE Transactions on Circuits and Systems, vol. CAS-26, pp.361-368, 1979.

[30] L. Andersen, "On single-row routing," IEEE Transaction on Circuits and Systems, vol. CAS-27, pp1262-1263, 1980.

[31] T. T. Tarng, M. Marek-Sadowska, and E. S. Kuh, "An efficient single-row routing algorithm," IEEE Transactions on Computer-Aided Design, vol. CAD-3, pp.178-183, 1984.

[32] S. Tsukiyama, E. S. Kuh, and I. Shirakawa, "An algorithm for single-row routing with prescribed street congestions," IEEE Transaction on Circuits and Systems, vol. CAS-27, pp.765-772, 1980.

[33] R. Raghavan and S. Sahni, "Single row routing," IEEE Transactions on Computers, vol. C-32, pp.209-220, 1983.

[34] R. Raghavan and S. Sahni, "Optimal single row router," 19th Design Automation Conference Proceedings, pp.38-42, 1982.

[35] R. Tsui and R. Smith, "A high-density multilayer PCB router based on necessary and sufficient conditions for row routing," 18th Design Automation Conference Proceedings, pp.372-381, 1981.

[36] S. Han and S. Sahni, "Single-row routing in narrow streets," IEEE Transactions on Computer-Aided Design, vol. CAD-3, pp.235-241, 1984.

[37] B. S. Ting, E. S. Kuh and A. Sangiovanni-Vincentell, "Via assignment problem in multilayer printed circuit board," IEEE Transaction on Circuits and Systems, vol. CAS-26, pp.261-272, 1979.

[38] S. Tsukiyama, I. Shirakawa, and S. Asahara, "An algorithm for the via assignment problem in multilayer backboard wiring," IEEE Transaction on Circuits and Systems, vol. CAS-26, pp.369-377, 1979.

[39] T. Gonzalez, "An approximation problem for the multi-via assignment problem," IEEE Transactions on Computer-Aided Design, vol. CAD-3, pp257-264, 1984.

[40] S. Tsukiyama, E. S. Kuh, and I. Shirakawa, "On the layering problem of multilayer PWB wiring," IEEE Transactions on Computer-Aided Design, vol. CAD-2, pp.30-38, 1983. Also in 18th Design Automation Conference Proceedings, pp.738-745, 1981.

[41] OPTIMATE™ Manual. Billerica (MASS): Optima Technology Inc., 1986.

[42] J. Foster, "Prerouting analysis programs," Proceedings of 12th Design Automation Conference, vol. 12, pp. 306-310, 1975.

[43] I. Shirakawa and S Futagami, "A rerouting scheme for single-layer printed wiring boards," IEEE Transactions on Computer-Aided Design, vol. CAD-2, pp. 267-271, October 1983.

[44] D. Kinniment, "Performance comparison of conventional backtracking," IEE Proceedings, vol. 127, part G, pp. 309-312, December 1980.

[45] L. Abel, "On the ordering of connections for automatic routing," IEEE Transactons on Computer, vol. C-21, pp. 1227-1236, November 1972.

[46] K. Kulkarni and V. Jayakumar, "Ordering of connections for automated routing," IEEE Transactions on Computer, vol. C-28, pp. 791-794, October 1979.

[47] J. Avenier, "Digitizing, layout, rule checking--the everyday task of chip designers," IEEE Proceedings, vol. 71, pp. 49-56, January, 1983.

[48] G. Persky, D. Deutsch, and D. Schweikert, "LTX--a system for the directed automatic design of LSI circuit," Proceedings of 13th Design Automation Conference, vol. 13, pp. 399-407, 1976.

[49] D. Hightower, "A generalized channel routing," Proceedings of 17th Design Automation Conference, vol. 17, pp. 12-21, 1980.

[50] U. Gupta, D. Lee, and J. Leung, "An optimal solution for the channel-assignment problem," IEEE Transactions on Computers, vol. C-28, pp. 807-810, November 1978.

[51] J. Li and M. Marek-Sadowska, "Global routing for gate arrays," IEEE Transactions on Computer-Aided Design, vol. CAD-3, pp. 298-307, October 1984.

[52] G. Clow, "A global routing algorithm for general cells," Proceedings of 21st Design Automation Conference, vol. 21, pp. 45-49, 1984.

[53] M. Wiesel, "Loose routing for gate arrays," 1984 IEEE International Symposium on Circuit and Systems, pp. 444-448, 1984.

[54] D. Lee and J. Leung, "On the 2-dimensional channel assignment problem," IEEE Transactions on Computer, vol. c-33, pp. 2-6, January 1984.

[55] M. Marek-Sadowska, "Two-dimensional router for double layer layout," Proceedings of 22nd Design Automation Conference, vol. 22, pp. 117-123, 1985.

[56] D. Deutsch, "A 'dogleg' channel router," Proceedings of 13th Design Automation Conference, vol. 13, pp. 425-433, 1976.

[57] F. Preparata and W. Lipski, "Optimal three-layer channel routing," IEEE Transactions on Computers, vol. c-33,pp. 427-437, May 1984.

[58] H. Carter and M. Breuer, "Efficient single-layer routing long a line of points," IEEE Transactions on Computer-Aided Design, vol. CAD-2, pp. 259-266, October 1983.

[59] M. Marek-Sadowska and T. Tarng, "Single-layer routing for VLSI: analysis and algorithms," IEEE Transactions on Computer-Aided Design, vol. CAD-2, pp. 246-259, October 1983.

[60] Y. Chen and M. Liu, "Three-layer channel routing," IEEE Transactions on Computer-Aided Design, vol. CAD-3, pp.156-163, April 1984.

[61] S. Hambrusch, "Channel routing algorithms for overlap models," IEEE Transactions on Computer-Aided Design, vol. CAD-4, pp. 23-30, January 1985.

[62] T. Szymanski, "Dogleg channel routing is NP-complete," IEEE Transactions on Computer-Aided Design, vol. CAD-4, pp. 31-41, January 1985.

[63] T. Yoshimura, "An efficient channel router," Proceedings of 21st Design Automation Conference, vol. 21, pp. 38-44, 1984.

[64] M. Marek-Sadowska and E. Kuh, "General channel-routing algorithm," IEE Proceedings, vol. 130, Pt. G, No. 3, pp. 83-88, June 1983.

[65] A. LaPaugh and R. Pinter, "On minimizing channel density by lateral shifting," IEEE International Conference on Computer-aided Design, pp. 123-124, 1983.

[66] R. Rivest and C. Fiduccia, "A 'greedy' channel router," Proceedings of 21st Design Automation Conference, vol. 19, pp. 418-424, 1982.

[67] C. Hsu, Signal Routing in Integrated Circuit Layout. UMI Research Press, 1986.

[68] D. Richards, "Complexity of single-layer routing," IEEE Transactions on Computers, vol. c-33, pp. 286-288, March 1984.

[69] A. Gamal and Z. Syed, "A stochastic model for interconnections in custom integrated circuits," IEEE Transactions on Circuits and Systems, vol. CAS-28, pp. 888-894, September 1981.

[70] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," Proceedings of 8th Design Automation Workshop, vol. 8, pp. 155-169, 1971.

[71] T. Yoshimura and E. Kuh, "Efficient algorithms for channel routing," IEEE Transactions on Computer-Aided Design, vol. CAD-1, January 1982.

[72] W. Kinsner and X. Kong, "Schematic Capture, Placement and Routing of PCBs and SMBs: Examples" Technical Report, MC86-1, Microelectronics Center (IAMC), University of Manitoba, Winnipeg, Canada; August 21, 1986, 80 pp.

APPENDIX A

NEW CHANNEL ROUTER IMPLEMENTATION

The general flow-chart of the program which implements the new channel routing algorithm is shown as in Fig. 45. Except for the net-to-track assignment procedure (ASSIGNTRACK), the procedures of Fig. 45 are quite straightforward.

The net-to-track assignment procedure, ASSIGNTRACK, includes two procedures, FINDER and EXPERT, and one function FPRIO. The function FPRIO evaluates the priority function (as defined in section 6.1) of a net NET to be routed in the same track containing the MOTHER NET.

Since no efficient algorithm has been developed for the selection of optimum subset of the nets from the set of READYNETS, a sub-optimum approach is used here. The selection algorithm can be explained as follows: First, create eligible subsets S_i by finding m (a pre-determined number, e.g. m=4) nets which have the first m largest priority numbers among the nets in the set of READYNETS, and placing each of them into a different subset S_i (for i=1, 2, ..., m). Then, exand the eligible subset S_i by adding as many nets as possible from the set READYNETS to each S_i according to the decreasing order of the priority number, provided that they do not have horizontal overlapping. Finally, from the m eligible subsets S_i, select one suboptimum subset S_0 whose sum of the priority numbers is maximum.

The procedure FINDER will sort the nets in READYNETS by decreasing order of priority number of each net. The procedure EXPERT is to find the maximum possible subset of nets S_i (TEMP1 in program) from the set READYNETS such that no nets in it have horizontal overlapping. Note that we first assign one of the m nets into the subset TEMP1 in ASSIGNTRACK procedure.

The flow-chart of the procedure ASSIGNTRACK is shown in Fig. 46.

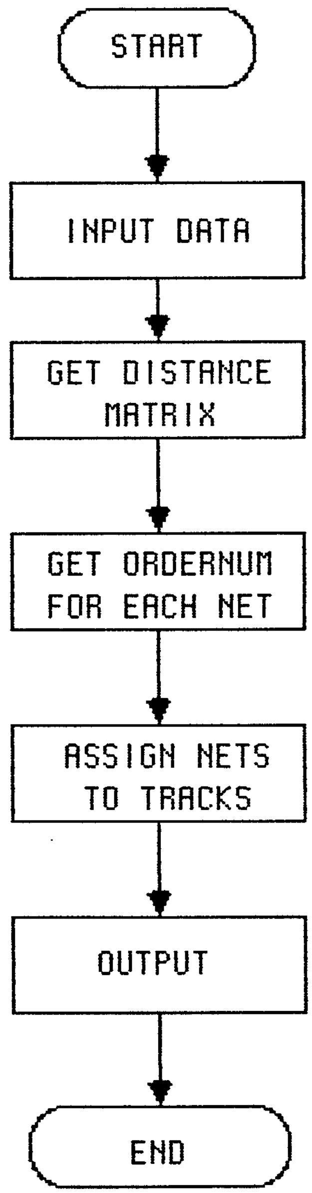


Fig. 45. General flowchart

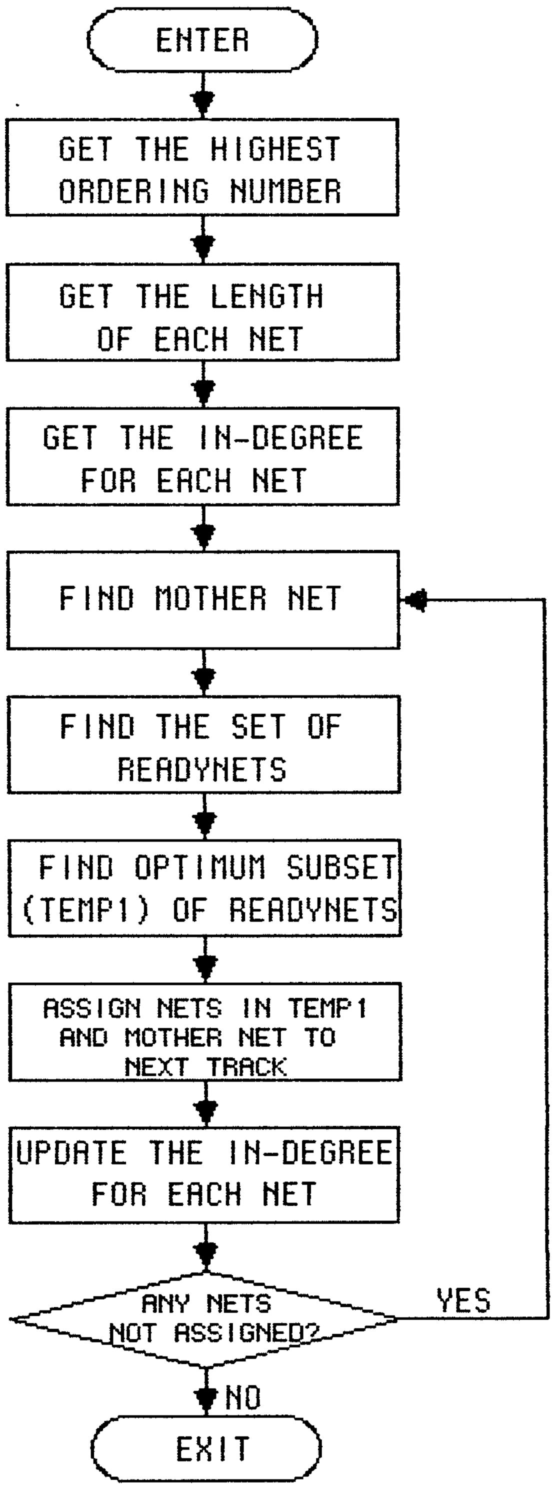


Fig. 46. Flowchart for procedure ASSIGNTRACK

APPENDIX B

NEW CHANNEL ROUTER PROGRAM LISTING

```
PROGRAM CHANNEL_ROUTER(INPUT,OUTPUT);
(**********************************************************************)
(****     MOTHER NET SELECTION:                                   ****)
(****       ORDER[MOTH]-CURRENTORD<1;                             ****)
(****       ORDER[MOTH]*NUMNETFOLL[MOTH] MAXIMUM;                 ****)
(****     PRIO:=LENG+(MAXDIS-DIS)+5*ORDER/CURRENTORD;             ****)
(****     NO DENSITY CHECK;                                       ****)
(**********************************************************************)
CONST
 NETLIST='DATAØØ';       (*NETLIST CONTAINS THE NAME OF THE INPUT DATA*)
 NETNUMØ=1Ø;             (*NETNUMØ IS THE NUMBER OF NETS TO BE ROUTED *)
 TERNUM=17;              (*TERNUM IS THE NUMBER OF TERMINALS ON EITHER
                                UPPER OR LOWER SIDE OF CHANNEL       *)
 CONSTB=1;               (*CONSTB IS FOR CURRNTORD-ORDER[MOTH]<CONSTB *)
TYPE
 TYPE_ONE=ARRAY[1..NETNUMØ] OF INTEGER;
 TYPE_TWO=ARRAY[1..NETNUMØ] OF BOOLEAN;
 TYPE_THREE=-1..2;
 TYPE_FOUR=ARRAY[1..TERNUM] OF TYPE_THREE;
 TYPE_FIVE=PACKED ARRAY[1..3] OF CHAR;
 TYPE_SIX=ARRAY[1..NETNUMØ] OF TYPE_FIVE;
 TYPE_SEVEN=ARRAY[1..NETNUMØ, 1..NETNUMØ] OF INTEGER;
 TYPE_EIGHT=ARRAY[1..NETNUMØ, 1..TERNUM] OF TYPE_THREE;
 TYPE_NINE=ARRAY[1..NETNUMØ, 1..NETNUMØ] OF BOOLEAN;
 ZZ=INTEGER;
 TY1=TYPE_ONE;
VAR
 CONSTA      :INTEGER;
 CDIS,CLEN   :INTEGER;
 MATRIXNAM   :TYPE_SIX;
 MATRIXIN    :TYPE_EIGHT;
 MATRIXDIS   :TYPE_SEVEN;
 ORDERNUM    :TYPE_ONE;
 MATRIXOUT   :TYPE_ONE;
 NUMNETFOLL  :TYPE_ONE;
 MATRIXREL   :TYPE_NINE;
 MAXORDERING:INTEGER;
 DENSITY     :INTEGER;
 CHANNELNUM  :INTEGER;
PROCEDURE INPUTDATA;
(**********************************************************************)
(****                                                             ****)
(****                        INPUT DATA                           ****)
(****                                                             ****)
(**********************************************************************)
VAR
 INDEX1,INDEX2,TEST :INTEGER;
IN1,IN2:INTEGER;
 FLAG1,FLAG2        :BOOLEAN;
 INPUT_DATA         :TEXT;
BEGIN
 OPEN(INPUT_DATA,NETLIST,'OLD');
 RESET(INPUT_DATA);
 READLN(INPUT_DATA,CONSTA);
 READLN(INPUT_DATA,CLEN);
 READLN(INPUT_DATA,CDIS);
```

```
FOR INDEX1:=1 TO NETNUMØ DO
BEGIN
 READ(INPUT_DATA,MATRIXNAM[INDEX1]);
 FOR INDEX2:=1 TO TERNUM DO
  READ(INPUT_DATA,MATRIXIN[INDEX1,INDEX2]);
 READLN(INPUT_DATA);
END;
CLOSE(INPUT_DATA);
(* WRITE('*******');
FOR INDEX1:=5Ø TO 8Ø DO
WRITE(INDEX1:3);
WRITELN;
FOR INDEX1:=1 TO NETNUMØ DO
BEGIN
 WRITE(INDEX1:4,'***');
 FOR INDEX2:=5Ø TO 8Ø DO
 WRITE(MATRIXIN[INDEX1,INDEX2]:3);
 WRITELN;
END;*)
FOR INDEX1:=1 TO TERNUM DO
BEGIN
 IN1:=Ø;
 IN2:=Ø;
 FOR INDEX2:=1 TO NETNUMØ DO
 BEGIN
  IF (MATRIXIN[INDEX2,INDEX1]=1) THEN IN1:=IN1+1;
  IF (MATRIXIN[INDEX2,INDEX1]=-1) THEN IN2:=IN2+1;
 END;
 IF (IN1>1) OR (IN2>1) THEN WRITELN('INPUT IS WRONG AT',INDEX1)
END;
END;
PROCEDURE DISTANCE;
(********************************************************************)
(****                                                            ****)
(****           CALCULATE THE DISTANCES BETWEEN NETS             ****)
(****                                                            ****)
(********************************************************************)
VAR
 INDEX1,INDEX2        :INTEGER;
 TEMP,TEMPB,TEMPE     :INTEGER;
 FUNCTION GETDIS(NET1:INTEGER;NET2:INTEGER):INTEGER;
 (******************************************************************)
 (****     CALCULATE THE DISTANCE BETWEEN NET1 AND NET2        ****)
 (******************************************************************)
 VAR
  INDEX,FLAG11        :INTEGER;
  NET,FLAG12,FLAG21   :INTEGER;
  BFLAG,FINISH        :BOOLEAN;
  BFLAG1,BFLAG2       :BOOLEAN;
 BEGIN                      (* BEGINNING OF THE PROCEDURE DISTANCE *)
  INDEX:=Ø;
  FLAG11:=Ø;
  FLAG12:=Ø;
  FLAG21:=Ø;
  BFLAG:=TRUE;
  FINISH:=FALSE;
```

```
(********FIND NET1 AND THE BEGINNING OF NET1************************)
WHILE (INDEX<TERNUM) AND BFLAG DO
BEGIN
 INDEX:=INDEX+1;
 BFLAG1:=(MATRIXIN[NET1,INDEX]<>Ø);
 BFLAG2:=(MATRIXIN[NET2,INDEX]<>Ø);
 IF (BFLAG1 OR BFLAG2) THEN
 BEGIN
  IF BFLAG1 AND BFLAG2
  THEN
  BEGIN
   BFLAG:=FALSE;
   FINISH:=TRUE;
   GETDIS:=Ø
  END
  ELSE
   IF BFLAG1 THEN
   BEGIN
    FLAG11:=INDEX;
    BFLAG:=FALSE
   END
   ELSE
   BEGIN
    NET:=NET2;
    NET2:=NET1;
    NET1:=NET;
    BFLAG:=FALSE;
    FLAG11:=INDEX
   END
 END
END;
(*********END OF FINGING NET1 AND THE BEGINNING OF NET1************)
(**************FIND DISTANCE BETWEEN NET1 AND NET2****************)
IF NOT FINISH THEN                               (**FIND THE END OF NET1**)
BEGIN
 BFLAG:=TRUE;
 WHILE (INDEX<TERNUM) AND BFLAG DO
 BEGIN
  INDEX:=INDEX+1;
  IF MATRIXIN[NET1,INDEX]=Ø THEN
  BEGIN
   BFLAG:=FALSE;
   FLAG12:=INDEX-1
  END
 END;
 IF BFLAG THEN FLAG12:=TERNUM;   (**END OF FINDING THE END OF NET1**)
 INDEX:=Ø;                          (**FIND THE BEGINNING OF NET2**)
 BFLAG:=TRUE;
 WHILE (INDEX<TERNUM) AND BFLAG DO
 BEGIN
  INDEX:=INDEX+1;
  IF MATRIXIN[NET2,INDEX]<>Ø THEN
  BEGIN
   BFLAG:=FALSE;
   FLAG21:=INDEX
  END
 END;                         (**END OF FINDING THE BEGINNING OF NET2**)
 IF FLAG21>FLAG12 THEN GETDIS:=FLAG21-FLAG12 ELSE GETDIS:=Ø
```

```
   END
   (********END OF FINDING DISTANCE BETWEEN NET1 AND NET2*************)
  END;
 BEGIN                                 (*BEGINNING OF PROCEDURE DISTANCE*)
  TEMPE:=NETNUMØ-1;
  FOR INDEX1:=1 TO TEMPE DO
  BEGIN
   MATRIXDIS[INDEX1,INDEX1]:=Ø;
   TEMPB:=INDEX1+1;
   FOR INDEX2:=TEMPB TO NETNUMØ DO
   BEGIN
    TEMP:=GETDIS(INDEX1,INDEX2);;
    MATRIXDIS[INDEX1,INDEX2]:=TEMP;
    MATRIXDIS[INDEX2,INDEX1]:=TEMP
   END
  END;
  MATRIXDIS[NETNUMØ,NETNUMØ]:=Ø
 END;                                        (*END OF PROCEDURE DISTANCE*)
 PROCEDURE GETRESULTS;
 (*********************************************************************)
 (****                                                             ****)
 (****       CALCULATE THE ORDERING NUMBER FOR EACH NET.           ****)
 (****                                                             ****)
 (*********************************************************************)
 VAR
  NUMNETPREC               :TYPE_ONE;
  TEMP,TEMPFOLLNET         :TYPE_ONE;
  FLAGØ                    :BOOLEAN;
  NUM_OF_ORDERONE,INDEX1   :INTEGER;
  PRECNUM                  :INTEGER;
  ORDERING,NETNAME         :INTEGER;
  PROCEDURE GETRELATION;
  VAR
   INDEX1,INDEX2,COUNT     :INTEGER;
   UPPERNET,LOWERNET       :INTEGER;
   FLAG1                   :BOOLEAN;
  BEGIN                       (*BEGINNING OF PROCEDURE GETRELATION*)
   FOR INDEX1:=1 TO NETNUMØ DO
   BEGIN
    FOR INDEX2:=1 TO NETNUMØ DO
     MATRIXREL[INDEX1,INDEX2]:=FALSE;
   END;
   FOR INDEX1:=1 TO TERNUM DO
   BEGIN
    INDEX2:=Ø;
    FLAG1:=FALSE;
    COUNT:=Ø;
    WHILE (INDEX2<NETNUMØ) AND (NOT FLAG1) DO
    BEGIN
     INDEX2:=INDEX2+1;
     IF MATRIXIN[INDEX2,INDEX1]=1 THEN
     BEGIN
      UPPERNET:=INDEX2;
      COUNT:=COUNT+1;
     END;
```

```
    IF MATRIXIN[INDEX2,INDEX1]=-1 THEN
    BEGIN
     LOWERNET:=INDEX2;
     COUNT:=COUNT+1;
    END;
    IF COUNT=2 THEN FLAG1:=TRUE
   END;
   IF FLAG1 THEN MATRIXREL[UPPERNET,LOWERNET]:=TRUE
  END;
  FOR INDEX1:=1 TO NETNUMØ DO
  BEGIN
   NUMNETFOLL[INDEX1]:=Ø;
   NUMNETPREC[INDEX1]:=Ø;
   FOR INDEX2:=1 TO NETNUMØ DO
   BEGIN
    IF MATRIXREL[INDEX2,INDEX1] THEN
      NUMNETPREC[INDEX1]:=NUMNETPREC[INDEX1]+1;
    IF MATRIXREL[INDEX1,INDEX2] THEN
      NUMNETFOLL[INDEX1]:=NUMNETFOLL[INDEX1]+1;
   END
  END
 END;                                  (*END OF PROCEDURE GETRELATION*)
 PROCEDURE ORDERZ(NAMEOFNET:INTEGER;ORDER:INTEGER;NUM_PREC_IT:INTEGER);
 VAR
  INDEX1     :INTEGER;
  PRECNUM    :INTEGER;
 BEGIN                           (*BEGINNING OF PROCEDURE GETORDERING*)
  ORDER:=ORDER+1;
  INDEX1:=Ø;
  WHILE (NUM_PREC_IT>Ø) DO
  BEGIN
   INDEX1:=INDEX1+1;
   WHILE (NOT MATRIXREL[INDEX1,NAMEOFNET]) DO
    INDEX1:=INDEX1+1;
   NUM_PREC_IT:=NUM_PREC_IT-1;
   IF ORDERNUM[INDEX1]<ORDER THEN
    ORDERNUM[INDEX1]:=ORDER;
   PRECNUM:=NUMNETPREC[INDEX1];
   IF PRECNUM>Ø THEN ORDERZ(INDEX1,ORDER,PRECNUM)
  END
 END;                              (*END OF THE PROCEDURE GETORDERING*)
BEGIN                              (*BEGINNING OF PROCEDURE GETRESULTS*)
 GETRELATION;
 NUM_OF_ORDERONE:=Ø;
 FOR INDEX1:=1 TO NETNUMØ DO
 BEGIN
  TEMP[INDEX1]:=Ø;
  ORDERNUM[INDEX1]:=1;
 END;
 FOR INDEX1:=1 TO NETNUMØ DO
 BEGIN
  IF NUMNETFOLL[INDEX1]=Ø THEN
  BEGIN
   NUM_OF_ORDERONE:=NUM_OF_ORDERONE+1;
   TEMP[NUM_OF_ORDERONE]:=INDEX1
  END
 END;
```

```
 FOR INDEX1:=1 TO NUM_OF_ORDERONE DO
 BEGIN
  ORDERING:=1;
  NETNAME:=TEMP[INDEX1];
  PRECNUM:=NUMNETPREC[NETNAME];
  IF (PRECNUM>Ø) THEN ORDERZ(NETNAME,ORDERING,PRECNUM)
 END;
 FOR INDEX1:=1 TO NETNUMØ DO
  IF (ORDERNUM[INDEX1]=1) AND (NUMNETPREC[INDEX1]=Ø)
   THEN ORDERNUM[INDEX1]:=(ORDERING+1) DIV 2;
END;                              (*END OF THE PROCEDURE GETTRESULTS*)
PROCEDURE ASSIGNTRACK;
(**********************************************************************)
(****                                                             ****)
(****                  ASSIGN NETS TO TRACKS.                     ****)
(****                                                             ****)
(**********************************************************************)
VAR
 NO1,NO2,SEED,NETNAMER,INDEX1      :INTEGER;
 INDEX,CURRENTORD,LENGTH           :INTEGER;
 INDEX2,INDEXZ,TRACKNUM            :INTEGER;
 MOTHNET,MAXLEN,MAXDIS,NUMSEED     :INTEGER;
 PRIONUM1,PRIONUM,ORDERØ6,M        :INTEGER;
 DENSITY_LOCATION                  :INTEGER;
 NUMB_READYNETS,PRIONUM4,INDEXØ    :INTEGER;
 PRIORITY,TEMP1,TEMP2              :TYPE_ONE;
 MATRIXLEN,READYNETS               :TYPE_ONE;
 RESULTNET,FANINDEGREE             :TYPE_ONE;
 ASSIGNED                          :TYPE_TWO;
 LOCATION                          :ARRAY [1..TERNUM] OF BOOLEAN;
 FLAG,FINISH,FLAG1,FLAG2           :BOOLEAN;
 NOTESTED,MVALID,AVALID1,AVALID2:BOOLEAN;
 CHECKER,INDENSITY,BVALID          :BOOLEAN;
 FUNCTION FPRIO(MO:INTEGER;SO:INTEGER;MAL:INTEGER;MAD:INTEGER):INTEGER;
 (********************************************************************)
 (****          CALCULATE THE PRIORITY NUMBER FOR NET 'SO'       ****)
 (****      TO BE ROUTED IN THE SAME TRACK WITH MO(THER) NET.    ****)
 (********************************************************************)
 VAR
  LENGTH,DISTANCE,ORDER       :INTEGER;
 BEGIN
  LENGTH:=MATRIXLEN[SO]*CLEN;
  ORDER:=ORDERNUM[SO]*CONSTA;
  DISTANCE:=MAD-MATRIXDIS[MO,SO];
  FPRIO:=LENGTH+DISTANCE*CDIS+((CONSTA*ORDER) DIV CURRENTORD)
 END;
 PROCEDURE EXPERT(VAR NO1:ZZ;VAR NO2:ZZ;VAR TEMP1:TY1;VAR TEMP2:TY1);
 (********************************************************************)
 (****                                                           ****)
 (**** FIND THE SUB-OPTIMUM SUBSET TEMP1 FROM THE SET READYNETS. ****)
 (****                                                           ****)
 (********************************************************************)
 VAR
  TEMP,INDEX1,INDEX2     :INTEGER;
  NET1,NET2,TEMPØ,T1     :INTEGER;
  TEST,ILLEGAL,FLAG      :BOOLEAN;
```

```
BEGIN
 INDEX2:=Ø;
 WHILE INDEX2<NO2 DO
 BEGIN
  INDEX2:=INDEX2+1;
  NET2:=TEMP2[INDEX2];
  INDEX1:=Ø;
  ILLEGAL:=FALSE;
  WHILE (INDEX1<NO1) AND (NOT ILLEGAL) DO
  BEGIN
   INDEX1:=INDEX1+1;
   NET1:=TEMP1[INDEX1];
   IF (MATRIXDIS[NET1,NET2]=Ø) AND (NOT (NET1=NET2)) THEN
    ILLEGAL:=TRUE;
  END;
  IF NOT ILLEGAL THEN
  BEGIN
   NO1:=NO1+1;
   TEMP1[NO1]:=NET2;
   PRIONUM:=PRIONUM+PRIORITY[NET2]
  END
 END
END;
PROCEDURE FINDER(VAR NETS:TY1;NETNUMB:INTEGER);
(**********************************************************************)
(****                                                             ****)
(****   SORT THE NETS IN (READY)NETS BY THE PRIORITY NUMBER.     ****)
(****                                                             ****)
(**********************************************************************)
VAR
 GAP,I,J,NET1,NET2 :INTEGER;
BEGIN
 GAP:=NETNUMB DIV 2;
 WHILE (GAP>Ø) DO
 BEGIN
  I:=GAP;
  WHILE (I<=NETNUMB) DO
  BEGIN
   J:=I-GAP;
   WHILE (J>Ø) DO
   BEGIN
    NET1:=NETS[J];
    NET2:=NETS[J+GAP];
    IF (PRIORITY[NET1]<PRIORITY[NET2]) THEN
    BEGIN
     NETS[J]:=NET2;
     NETS[J+GAP]:=NET1
    END;
    J:=J-GAP;
   END;
   I:=I+1;
  END;
  GAP:=GAP DIV 2;
 END;
END;
PROCEDURE MDENSITY;
(**********************************************************************)
(****              CALCULATE THE MAXIMUM DENSITY.                 ****)
(**********************************************************************)
```

```
VAR
 INDEX1,INDEX2      :INTEGER;
 DENSITY_TEMP       :INTEGER;
BEGIN                              (* DENSITY IS THE MAX-DENSITY  *)
 DENSITY:=Ø;
 FOR INDEX2:=1 TO TERNUM DO
 BEGIN
  DENSITY_TEMP:=Ø;
  FOR INDEX1:=1 TO NETNUMØ DO
   IF MATRIXIN[INDEX1,INDEX2]<>Ø THEN DENSITY_TEMP:=DENSITY_TEMP+1;
  IF DENSITY_TEMP>=DENSITY THEN
   DENSITY:=DENSITY_TEMP;
 END;
END;
BEGIN                        (*BEGIN OF THE PROCEDURE OF ASSIGNTRACK*)
 NO1:=Ø;
 NO2:=Ø;
 (************GET THE HIGHEST ORDERING NUMBER************************)
 CURRENTORD:=Ø;
 FOR INDEX1:=1 TO NETNUMØ DO
  IF ORDERNUM[INDEX1]>CURRENTORD THEN
   CURRENTORD:=ORDERNUM[INDEX1];
 MAXORDERING:=CURRENTORD;
 (************END OF GETTING THE HIGHEST ORDERING NUMBER*************)
 (*****************GET THE LENGTH OF EACH NET***********************)
 FOR INDEX:=1 TO NETNUMØ DO
 BEGIN
  INDEX1:=Ø;
  FLAG:=TRUE;
  WHILE (INDEX1<TERNUM) AND FLAG DO
  BEGIN
   INDEX1:=INDEX1+1;
   IF MATRIXIN[INDEX,INDEX1]<>Ø THEN FLAG:=FALSE
  END;
  INDEX2:=INDEX1;
  FLAG:=TRUE;
  WHILE (INDEX2<TERNUM) AND FLAG DO
  BEGIN
   INDEX2:=INDEX2+1;
   IF MATRIXIN[INDEX,INDEX2]=Ø THEN
   BEGIN
    INDEX2:=INDEX2-1;
    FLAG:=FALSE;
   END
  END;
  MATRIXLEN[INDEX]:=INDEX2-INDEX1
 END;
 (*************END OF GETTING THE LENGTH OF EACH NET****************)
 (***********************INITIALIZE FANINDEGREE*********************)
 FOR INDEX1:=1 TO NETNUMØ DO
 BEGIN
  FANINDEGREE[INDEX1]:=Ø;
  FOR INDEX2:=1 TO NETNUMØ DO
  IF MATRIXREL[INDEX2,INDEX1] THEN
   FANINDEGREE[INDEX1]:=FANINDEGREE[INDEX1]+1
 END;
 (*********************END OF INITIALIZATION************************)
 MDENSITY;                      (*PROCEDURE OF GET MAXIMUM DENSITY*)
```

```
FOR INDEX1:=1 TO NETNUMØ DO
 ASSIGNED[INDEX1]:=FALSE;
FINISH:=FALSE;
TRACKNUM:=Ø;
WHILE NOT FINISH DO
BEGIN
 TRACKNUM:=TRACKNUM+1;
 FLAG:=FALSE;
 (**********************************************************************)
 (**   FLAG INDICATE THAT THERE ARE SOME NETS HAVING THE SAME        **)
 (**     ORDERING NUMBER AS THE CURRENTORDORING(=CURRENTORD)         **)
 (**********************************************************************)
 (**********************************************************************)
 (****      THE FOLLOWING IS THE PROCEDURE OF SELECTING            ****)
 (****                 MOTHER NET--MOTHNET                         ****)
 (**********************************************************************)
 WHILE (NOT FLAG) DO
 BEGIN
  INDEX2:=-1ØØ;
  FOR INDEX1:=1 TO NETNUMØ DO
  BEGIN
   ORDERØ6:=ORDERNUM[INDEX1];
   AVALID1:=NOT ASSIGNED[INDEX1];
   AVALID2:=((CURRENTORD-ORDØ6)<CONSTB) AND (FANINDEGREE[INDEX1]=Ø);
   IF (AVALID1 AND AVALID2) THEN
   BEGIN
    FLAG:=TRUE;
    IF (MATRIXLEN[INDEX1]*ORDERØ6)>INDEX2 THEN
    BEGIN
     MOTHNET:=INDEX1;
     INDEX2:=MATRIXLEN[INDEX1]*ORDERØ6
    END
   END
  END;
  CURRENTORD:=CURRENTORD-1
 END;
 CURRENTORD:=CURRENTORD+1;
 WRITELN('MOTHER NET IS ......',MOTHNET:16);
 (**********************GET SET READYNETS****************************)
 NUMB_READYNETS:=Ø;
 FOR INDEX1:=1 TO NETNUMØ DO
 BEGIN
  FLAG1:=FALSE;
  FLAG2:=FALSE;
  FLAG:=FALSE;
  IF MATRIXDIS[MOTHNET,INDEX1]>Ø THEN FLAG1:=TRUE;
  IF (FANINDEGREE[INDEX1]=Ø) AND (NOT ASSIGNED[INDEX1])
  THEN FLAG2:=TRUE;
  FLAG:=FLAG1 AND FLAG2;
  IF FLAG THEN
  BEGIN
   NUMB_READYNETS:=NUMB_READYNETS+1;
   READYNETS[NUMB_READYNETS]:=INDEX1;
   WRITE('READY NET',NUMB_READYNETS:5);
   WRITELN('         IS   ',READYNETS[NUMB_READYNETS]);
  END
 END;
 (***************END OF GETTING SET READYNETS***********************)
```

```
MAXLEN:=Ø;
MAXDIS:=Ø;
FOR INDEXZ:=1 TO NUMB_READYNETS DO
BEGIN
 INDEX1:=READYNETS[INDEXZ];
 IF MATRIXLEN[INDEX1]>MAXLEN THEN MAXLEN:=MATRIXLEN[INDEX1];
 IF MATRIXDIS[MOTHNET,INDEX1]>MAXDIS THEN
  MAXDIS:=MATRIXDIS[MOTHNET,INDEX1]
END;
(*********GET THE PRIORITY NUMBER OF EACH NET IN READYNETS*********)
FOR INDEX1:=1 TO NUMB_READYNETS DO
BEGIN
 INDEXZ:=READYNETS[INDEX1];
 PRIORITY[INDEXZ]:=FPRIO(MOTHNET,INDEXZ,MAXLEN,MAXDIS)
END;
(*********END OF GETTING THE PRIORITY OF EACH NET IN READYNET******)
IF NUMB_READYNETS>Ø THEN
BEGIN                        (*SELECT OPTIMUM SUBSET FROM READYNETS*)
 FINDER(READYNETS,NUMB_READYNETS);
 INDEXØ:=1;
 M:=4;
 IF (NUMB_READYNETS<M) THEN M:=NUMB_READYNETS;
 PRIONUM1:=Ø;
 WHILE (M>Ø) DO
 BEGIN
  SEED:=READYNETS[INDEXØ];
  PRIONUM:=PRIORITY[SEED];
  NO1:=1;
  NO2:=NUMB_READYNETS-1;
  TEMP1[1]:=SEED;
  FOR INDEXZ:=2 TO NETNUMØ DO
   TEMP1[INDEXZ]:=Ø;
  INDEXZ:=1;
  WHILE READYNETS[INDEXZ]<>SEED DO
  BEGIN
   TEMP2[INDEXZ]:=READYNETS[INDEXZ];
   INDEXZ:=INDEXZ+1
  END;
  FOR INDEX:=INDEXZ TO NO2 DO
   TEMP2[INDEX]:=READYNETS[INDEX+1];
  (***************************************************************)
  (*                                                             *)
  (*  TEMP1: CONTAINS SEED NET                                   *)
  (*  TEMP2; CONTAINS CANDIDATE NETS FOR BEING ADDED INTO TEMP1  *)
  (***************************************************************)
  WRITELN;
  WRITE('SET TEMP1 CONTAINS:        ');
  FOR INDEX:=1 TO NO1 DO
   WRITE(TEMP1[INDEX]);
  WRITELN;
  WRITE('SET TEMP2 CONTAINS:        ');
  FOR INDEX:=1 TO NO2 DO
   WRITE(TEMP2[INDEX]);
  WRITELN;
  EXPERT(NO1,NO2,TEMP1,TEMP2);
  (***************************************************************)
  (*TEMP1: NETS SELECTED TO BE ROUTED IN THE SAME TRACK AS MOTHNET*)
  (*TEMP2: CANDIDATE NETS FOR BEING SELECTED FOR NEXT TEMP1      *)
  (***************************************************************)
  WRITE('RESULT SET TEMP1 CONTAINS:');
```

```
FOR INDEX:=1 TO NO1 DO
 WRITE(TEMP1[INDEX]);
WRITELN;
IF PRIONUM1<PRIONUM THEN
BEGIN
 PRIONUM1:=PRIONUM;
 FOR INDEX:=1 TO NO1 DO
  RESULTNET[INDEX]:=TEMP1[INDEX];
 FOR INDEX:=NO1+1 TO NETNUMØ DO
  RESULTNET[INDEX]:=Ø;
END;
M:=M-1;
INDEXØ:=INDEXØ+1
END;
WRITELN;
WRITE('SELECTED OPTIMUM SUBSET CONTAINS:');
FOR INDEX:=1 TO NUMB_READYNETS DO
 WRITE(RESULTNET[INDEX]:4);
WRITELN;
(* ASSIGN NETS TO TRACK **)
INDEX:=1;
WHILE (RESULTNET[INDEX]<>Ø) DO
BEGIN
 NETNAMER:=RESULTNET[INDEX];
 ASSIGNED[NETNAMER]:=TRUE;
 MATRIXOUT[NETNAMER]:=TRACKNUM;
 FOR INDEX1:=1 TO NETNUMØ DO
  IF MATRIXREL[NETNAMER,INDEX1] AND (NETNAMER<>INDEX1) THEN
   FANINDEGREE[INDEX1]:=FANINDEGREE[INDEX1]-1;
 INDEX:=INDEX+1
END
END;          (*END OF SELECTING OPTIMUM SUBSET FROM SET READYNETS*)
MATRIXOUT[MOTHNET]:=TRACKNUM;
ASSIGNED[MOTHNET]:=TRUE;
FOR INDEX1:=1 TO NETNUMØ DO
 IF MATRIXREL[MOTHNET,INDEX1] AND (MOTHNET<>INDEX1) THEN
  FANINDEGREE[INDEX1]:=FANINDEGREE[INDEX1]-1;
(****************CHECK IF ALL NETS ASSIGNED*************************)
FINISH:=TRUE;
FOR INDEX:=1 TO NETNUMØ DO
BEGIN
 WRITELN(INDEX:5,ASSIGNED[INDEX]:2Ø);
 IF NOT ASSIGNED[INDEX] THEN FINISH:=FALSE;
END;
MVALID:=FALSE;
FOR INDEX1:=1 TO NETNUMØ DO
 IF (NOT ASSIGNED[INDEX1]) AND (ORDERNUM[INDEX1]=CURRENTORD)
 THEN MVALID:=TRUE;
IF (NOT MVALID) THEN CURRENTORD:=CURRENTORD-1;
IF FINISH THEN CHANNELNUM:=TRACKNUM
END
END;                             (*END OF THE PROCEDURE ASSIGNTRACK*)
PROCEDURE DONE;
(********************************************************************)
(****                                                            ****)
(****                    OUTPUT RESULTS                          ****)
(****                                                            ****)
(********************************************************************)
```

```
VAR
 OUTPUT_DATA     :TEXT;
 INDEX,INDEXX    :INTEGER;
 ROUTING_RESULTS:PACKED ARRAY[1..2Ø] OF CHAR;
 ANSWER$         :STRING;
BEGIN
 WRITELN('*************************************************************');
 WRITELN('*****************');
 WRITELN('*********                    PARAMETER SET                 ');
 WRITELN('          *********');
 WRITELN('********* WEIGHT FACTOR FOR ORDERING OF NET IS......');
 WRITELN(CONSTA:3,'      *********');
 WRITELN('********* WEIGHT FACTOR FOR LENGTH OF EACH NET IS...');
 WRITELN(CLEN:3,'       *********');
 WRITELN('********* WEIGHT FACTOR FOR DISTANCE BETWEEN NETS...');
 WRITELN(CDIS:3,'       *********');
 WRITELN('*************************************************************');
 WRITELN('*****************');
 WRITELN;
 WRITELN;
 WRITELN;
 WRITELN(' ':5,'MAXIMUM ORDERING NUMBER IS:');
 WRITELN('*':8,MAXORDERING:5,'*':8);
 WRITELN;
 WRITELN(' ':5,'MAXIMUM DENSITY IS:');
 WRITELN('*':8,DENSITY:5,'*':8);
 WRITELN;
 WRITELN(' ':5,'THE CHANNEL WIDTH IS:');
 WRITELN('*':8,CHANNELNUM:5,'*':8);
 WRITELN;
 FOR INDEX:=1 TO CHANNELNUM DO
 BEGIN
  WRITE('****TRACK',INDEX:5,'    *****');
  FOR INDEXX:=1 TO NETNUMØ DO
  IF MATRIXOUT[INDEXX]=INDEX THEN
   WRITE(MATRIXNAM[INDEXX]:5);
  WRITELN;
 END;
 FOR INDEX:=1 TO 4 DO
 BEGIN
  FOR INDEXX:=1 TO 2Ø DO
  WRITE('*');
  WRITE('FINISH');
  FOR INDEXX:=1 TO 2Ø DO
  WRITE('*');
  WRITELN
 END
END;
```

```
(********************************************************************)
(********************************************************************)
(******                                                        ******)
(******                                                        ******)
(******                     MAIN PROGRAM                       ******)
(******                                                        ******)
(******                                                        ******)
(********************************************************************)
(********************************************************************)
BEGIN
 INPUTDATA;
 DISTANCE;
 GETRESULTS;
 ASSIGNTRACK;
 DONE
END.
```