DOUBLE-LAYER CHANNEL ROUTING WITH IRREGULAR BOUNDARIES

by

M. L. Liu and Y. K. Chen

Memorandum No. UCB/ERL M81/79

5 October 1981

DOUBLE-LAYER CHANNEL ROUTING WITH IRREGULAR BOUNDARIES

by

M. L. Liu and Y. K. Chen

Memorandum No. UCB/ERL M81/79

5 October 1981

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

DOUBLE-LAYER CHANNEL ROUTING WITH IRREGULAR BOUNDARIES*

by

M. L. Liu** and Y. K. Chen***

ABSTRACT

In the layout design of LSI chips, double-layer channel routing is used frequently. Usually the boundaries of a channel are parallel straight lines. But sometimes because of different sizes of cells, the boundaries have indentations. We hope some nets can be put into indentation areas to decrease the width of the channel.

Two kinds of problems will be discussed in this paper. For one kind, the height of indentations is fixed; for the other kind, the height of indentations is adjustable. We propose two different algorithms for them.

The algorithms were coded in PASCAL and implemented on VAX 11/780 computer. The computational results are satisfactory. Since some nets are laid inside indentations, the width of the channels decreases.

---

*Reaearch supported by the National Science Foundation Grant ENG-78-24425.
**On leave from Tianjin University, Tianjin, China.
***On leave from Tsinghua University, Beijing, China.

DOUBLE-LAYER CHANNEL ROUTING WITH IRREGULAR BOUNDARIES

M. L. Liu and Y. K. Chen

1. Introduction

Channel routing is one of the key problems in the LSI layout. Two rows of cells are placed on two sides of a channel. Along the channel, every terminal of cells has a certain number, and terminals with the same number must be connected by a net (Fig. 1).

So-called "double-layer routing" means that there are two layers for routing. One of them is used for horizontal wires, another for vertical wires.

Usually, boundaries of a channel are parallel straight lines. There are several efficient algorithms used for regular-shape channel-routing problems [1],[2]. But, because of different sizes of cells, the boundaries of a channel sometimes have indentations (Fig.2). If we still employ the usual algorithms for this kind of channel, the area of indentation (shaded in Fig.2) will be wasted space. Therefore, we hope some nets can be put into indentation areas to decrease the width of the channel.

In order to apply our new algorithms, we can divide indentation into some rectangular areas by using horizontal lines (dotted lines in Fig.2). Henceforth we will call such kind of area, "block"; and call the usual main area, "main block".

In this paper we will only describe algorithms for indentations. For the main block we will still employ the usual algorithms, for example the merging algorithm.

In general, blocks are not very wide. In order to put as many nets into blocks as possible, the dogleg can be used. In this way, the horizontal segment of a net can be divided into some subnets at its terminal position. Subnets are shorter and easier to be routed than original net. Henceforth, the so-called "net" will usually mean subnet.

Two kinds of problems will be discussed in this paper. For one kind, the height of blocks is fixed; for the other kind, the height of blocks is adjustable. We propose two different algorithms for them.

For algorithms to be presented in this paper, the vertical constraint graph (V.C.G) is crucial, as defined in reference [2]. Every net (i.e. subnet) has its corresponding node in the graph; and if there is a directed path from node A to node B (Fig.3), then in the channel, net A should be laid on a track above net B. We will call node A (or net A) the ancestor of node B (or net B), and node B the descendant of node A .

Bcause there is no essential difference between indentations along upper boundary of the channel and those along lower boundary, in the following sections we will pay more attention to the upper boundary. For the lower boundary, similar algorithms can be used.

In the beginning, for the sake of simplicity, we will discuss only one rectangular indentation along the upper boundary.

2. A block with adjustable height

If an indentation has adjustable height, it can contain all available nets.

The available nets which can be laid into the indentation must satisfy the following necessary and sufficient conditions:

A. Horizontal condition: Both the starting column and the ending column of the net must be inside the range of the indentation. For example, in Fig.4, net A satisfies this condition but nets B,C,D,E do not.

B. Vertical condition: Not only the net itself satisfies the horizontal condition but also all its ancestors do. Because in the channel the position of any net must be below all its ancestors, if one of its ancestors can not be laid inside the indentation, then the net itself can not . For instance, in Fig.5, net A does not satisfy the horizontal condition, although its "son", net B, does. Obviously net B can not then be inside the indentation.

According to these conditions, all available nets can be selected; then we can arrange them in the indentation by employing the merging algorithm.

The whole algorithm for a channel with a adjustable-height block includes four steps:

<u>Step 1</u>. Select available nets.

An array called "Qualifying" (abbreviation "Q") is adopted for expressing whether or not a net is available.

First, we check the horizontal condition for every net. If net i satisfies the condition, let Q[i]:=1; otherwise let Q[i]:=0.

Second, check the vertical condition for every net with Q=1. A recursive procedure is used for depth first searching. The searching starts from every node with Q=1, and goes upward along the opposite direction of arrows searching for ancestors of the node. Suppose the searching starts from node i. Along the path of searching there may be five different cases.

<u>Case 1</u>.
Encounter a node j of which Q[j]=0. Return to the starting node i and let Q[i]:=0. (Fig.6a and in Fig.7 A-B)

<u>Case 2</u>.
Encounter a node j of which Q[j]=1 and j does not have any ancestor. Let Q[j]:=2 and return to the starting node i; then search other ancestors of node i if it has any. Otherwise, if node i does not have other ancestors, let Q[i]:=2. (Fig.6b and in Fig.7 C-D,E-F)

<u>Case 3</u>.
Encounter a node j of which Q[j]=1 and j has some ancestors. Consider node j as a new starting node and begin a new searching.(Fig.6c and in Fig.7 G-H)

<u>Case 4</u>.
Encounter a node j of which Q[j] has already become 2. Return to starting node i and search other ancestors of node i if it has any. Otherwise, if node i does not have other ancestors, let Q[i]:=2. (Fig.6d and in Fig.7 I-D after Q[D] became 2)

<u>Case 5</u>.
The starting node i does not have any ancestors at all. Let Q[i]:=2. (Fig.6e and in Fig.7 J)

After the searching is finished, all nets of which Q=1 became either Q=0 or Q=2. The latter are available nets (Fig.8).

Now let us define a graph called sub-V.C.G. It contains all nodes corresponding to all available nets for the indentation, and a directed edge from net A to net B means that net A must be placed above net B.

Step 2. Delete all nodes with Q=2 from original V.C.G. and form the sub-V.C.G. defined above. A theorem is proposed as follows:

> Theorem There is a directed path between any two nodes in the sub-V.C.G. if and only if there was a directed path between these two nodes in the original V.C.G. and the path was through the same intermediate nodes.
>
> Proof: Because the V.C.G. expresses the vertical constraint relationship, it contains all nodes corresponding to all nets, and all edges between any two relative nodes. When the sub-V.C.G. is constructed no new nodes or new edges can be added. Hence any directed path existing in the sub-V.C.G. definitely also existed in original V.C.G.
>
> Conversely, if two nodes A and B are in both the original V.C.G. and the sub-V.C.G. and there is a directed path in the original V.C.G. Then there must be the same path in the sub-V.C.G. For example, in Fig. 9, there is a path A-X-Y-B in the original V.C.G. Since node B is also in the sub-V.C.G, Q[B]=2. Certainly Q[Y],Q[X] must equal 2. Otherwise if any of nodes Y or X can not be put in the block, their descendant node B can not be put in the block too, that means Q[B] does not equal 2, a contradiction. Therefore nodes Y and X must be in the sub-V.C.G. Moreover, from the original V.C.G. to the sub-V.C.G. the vertical constraint relationship is not changed. In other words, the edges between nodes A,X,Y,B will remain, and the path is the same. [Q.E.D]

According to this theorem, we do not need to construct the sub-V.C.G. seperately, and also we do not need to reconstruct the V.C.G. All that is needed is deletion of all edges between any two nodes such as i and j of which Q[i]=2 and Q[j]=0. After that, the sub-V.C.G. and remainder of the original V.C.G. can be used as they are.

Step 3. All available nets are laid into the indentation by using the merging algorithm as introduced in [2].

Step 4. All nets left are laid into the main block by using the merging algorithm.

3. A block with fixed height

If the height of block is fixed and specified by the input, sometimes it is impossible to lay all those nets satisfying horizontal and vertical conditions into the block. In that case, some "available nets" will be left to be put into main block.

For this kind of problem, we present a so-called " from top down to bottom" algorithm. According to this algorithm, the procedure is from the top track of the block down to the bottom track of the block along the vertical direction instead of the horizontal direction (for instance in merging or matching algorithms the procedures go forward zone by zone).

It is shown that once a net has been laid into the indentation, all its ancestors can no longer be put into the main block. For example, in Fig.10, if net B is inside the indentation, net A definitely must be inside too. Conversely, if net A is inside the indentation, net B can be either inside or outside. Hence, the processing of net A should be earlier than that of net B. Generally speaking, for the top track of the block, we should select those nets which are without any ancestors.

A linked list called "waiting list" is used. Only those nets which satisfy the horizontal condition as indicated above and are at the top level of the V.C.G. (in other words, do not have ancestors at all) can be selected for the waiting list. For example, in Fig.11, if any of nets A,B,C and D satisfy the horizontal condition, then they can be written into the waiting list.

Once some nets have been laid on a track of the block, we delete them from the waiting list and the V.C.G. Then, maybe some new nets will appear at the top level of the V.C.G. For example, in Fig.11, suppose nets B and C have been selected to lay on a track and have been deleted from the V.C.G, if any of nets E,F and G satisfy the horizontal condition also, they will be written into the waiting list.

For this algorithm, the sub-V.C.G. is not needed. After those nodes corresponding to nets which are laid inside the indentation have been deleted from the original V.C.G, the remainder of V.C.G. can be used as it is.

A hierarchical method is adopted. For every track of the block, nets are selected from the waiting list according to the idea, "The net which is more difficult to arrange and is of more influence on its descendants will be chosen first." For the first net of a track, the selection criteria are the following:

A. The length of the net --L[i],

L[i]:=(# of the ending column of the net)-(# of the starting column of the net).

B. The height of the node (the longest directed path from the node to the bottom of the V.C.G.) --H[i]. For example, in Fig.11, H[A]=5, H[B]=6, H[C]=3, H[D]=1.

C. The degree of the node (how many edges it connects in the V.C.G.) --D[i]. For example, in Fig.11, D[A]=D[B]=1, D[C]=2, D[D]=0.

The objective function used to select the first net of a track is defined as

rw[i]:=C1*L[i]+C2*H[i]+D[i],

where C1 and C2 are two constants expressing priority.

Usually, we choose C1:=200, C2:=5.

From the waiting list, we will choose the net of which the value of rw is the largest, as the first net of a track.

If the first net can not occupy the whole length of the track, we hope that as many nets as possible will be added on the track . For instance, in Fig.12, suppose net A is the first net of the track. From net A expanding to the right side of the track, nets B and C are assigned one by one if they can be; similarly, to the left side of the track, nets D and E are assigned.

For selecting these added nets, the objective function is defined as

f[i]:=C3*G[i]-L[i].

where G[i] is the length of the gap between net i and nets assigned already. C3 is a constant. Usually we choose C3:=10.

From the waiting list we always choose the net of which the value of f is the smallest, and add that net to the track.

The procedure goes from the top track to the bottom of the block, until the waiting list is used up or the block is full.

4. More complex shape of boundaries

In the two preceding sections we discussed only the simplest cases. There was only one rectangular indentation along the upper boundary of the channel. Now we will discuss more general and more complex cases.

If the block is along the lower boundary (for example, in Fig. 13), the main difference from the cases previously discussed is the exchanging of the concepts of ancestor and descendant. In Fig. 13, net B is now the ancestor of net A and the decendant of net C. Furthermore, for the fixed-height block cases, the algorithm is now from the bottom of the block up to the top of the block.

If there are more blocks than one along a boundary, the algorithm for every block is the same as indicated above. But we must pay attention to the processing sequence. In terms of relationship among blocks, there are three different situations:

A. Parallel (Fig. 14)

There are no common columns between any two parallel blocks. If a net can be put into one block, it can not be put into any other. Therefore, no block has priority of treatment over any other. We can process them in arbitrary sequence, for instance, A-D-C-B or D-B-A-C, etc. Usually, for convenience, we simply use the sequence "from left to right" : A-B-C-D.

B. Serial (Fig. 15)

If there are some common columns between blocks, we say these blocks " serial."

Since the starting and ending columns of the upper block are always inside the range of the lower block, all nets satisfying the horizontal condition of the upper block always satisfy that of the lower block. But conversely, that is not necessarily true. If a net satisfies the horizontal condition of the lower block, it does not always satisfy that of upper block.

Hence we should always process the upper block before the lower one. We use a so-called "priority graph" to represent the priority of blocks. In the priority graph, every node represents a corresponding block; if there is a directed path from node A to node B, block A should be processed before block B. Priority graphs for Fig. 14 and Fig. 15 are shown in Fig. 16a and Fig. 16b, respectively.

C. Serial and parallel

There are both serial and parallel relationships of blocks - for example, in Fig.17a and Fig.17b. Their priority graphs are shown in Fig.17c and Fig.17d, respectively.

If the indentations are along the lower boundary, there is a similar consideration but with opposite direction, "from lower to upper." For example, in Fig.18 the processing sequence is A-B-C-D-E.

In cases with fixed hieght of blocks, the shapes of boundaries are specified and entirely determined by input data. For dividing blocks and determining the priority of blocks, the necessary input data about the shapes of boundaries are only the position and height of every "stair" along the boundaries. For example, in Fig.19, they are the x coordinate (position--column number) and the y coodinate (height--how many tracks from the main block) of points a,b,c,d,e,f and g.

As the definition a stack is an ordered list in which all insertions (pushing in) and deletions (popping out) are made at its top [3]. A stack of records is used in this procedure. Every record contains the height and the number of column of a stair. We consider stairs of a boundary from left to right. Their records are pushed into the top of the stack one by one. But if at the top of the stack there is a record containing a larger height value than that of the new stair, the record at the top of stack must be popped out from the stack and a new block will be determined with its height and the range of columns. For example, in Fig.19, the records of stairs a,b and c are pushed into the stack in sequence. But, because the height of stairs c and b are larger than that of stair d, before the record of stair d is pushed into the stack, records c and b must be popped out and blocks 1 and 2 will be formed. Then, record e will be pushed into the stack; and then, before record f is pushed in, record e should be popped out and block 3 will be determined, etc.

It is obvious, that stack has the following properties:

A. The upper record always has a larger height value than that of lower records.

B. Once a record is popped out from the stack, a block will be determined.

C. Blocks are formed in required sequence.

For another kind of problem which deals with adjustable height of blocks, the input data tell the starting and the ending columns of every block. Hence the serial and parallel relationships of blocks can also be determined.

5. Conclusion

Two algorithms were proposed for either indentations with adjustable-height or indentations with fixed-height .

The algorithms were coded in PASCAL and implemented on VAX 11/780 computer.

Some computational results are shown in Fig. 20 to Fig. 22 (fixed height of blocks) and Fig. 23, Fig.24 (adjustabl height of blocks). The results are satisfactory. Since some nets are laid inside indentations, the number of tracks in the main block decreases .

Reference

[1] D. N. Deutch, C. Persky and D. G. Schwiekert. "LTX-A System for the Directed Automatic Design of LSI Circuits" Proc. 13th Design Auto. Conf. 1976, pp.399-407.

[2] T. Yoshimura and E. S. Kuh. "Efficient Algorithms for Channel Routing" UCB/ERL M80/43, Aug. 1980.

[3] E. Horowitz and S. Sahni. "Fundamentals of Data Structures" Computer Science Press, Inc. 1976.

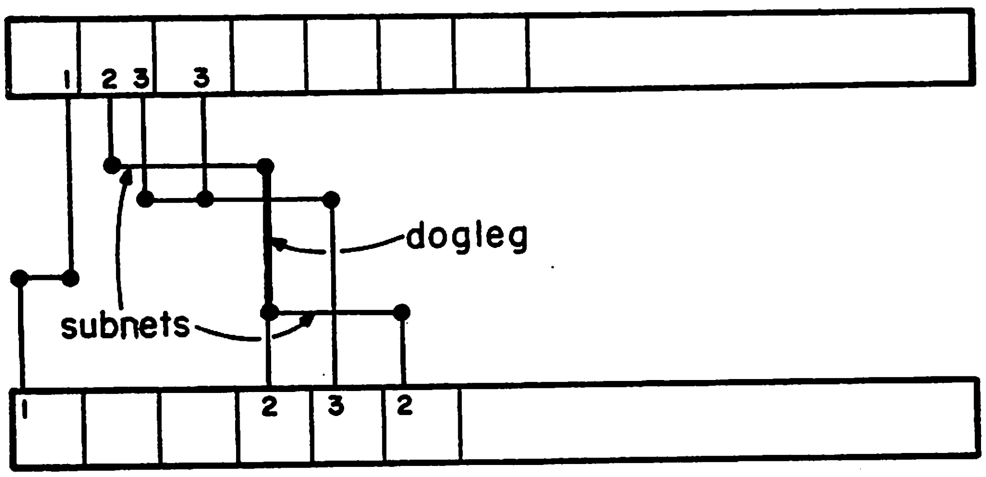


Fig. 1

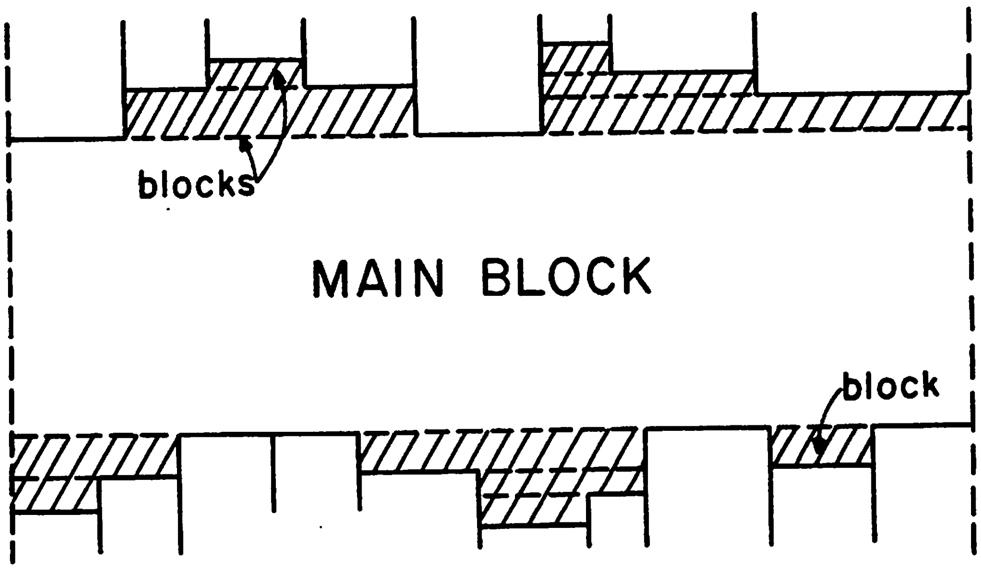


Fig. 2

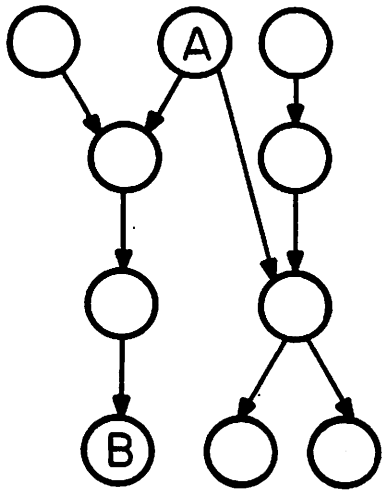


Fig. 3

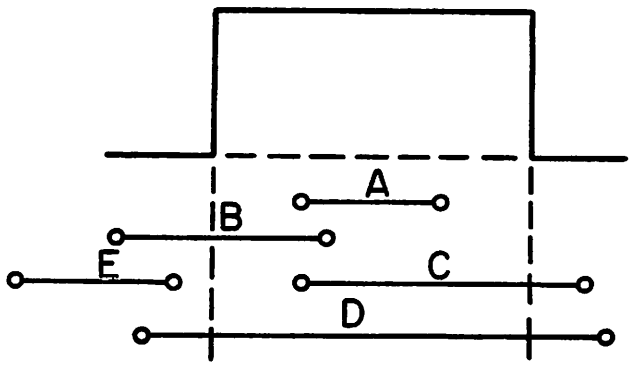


Fig. 4

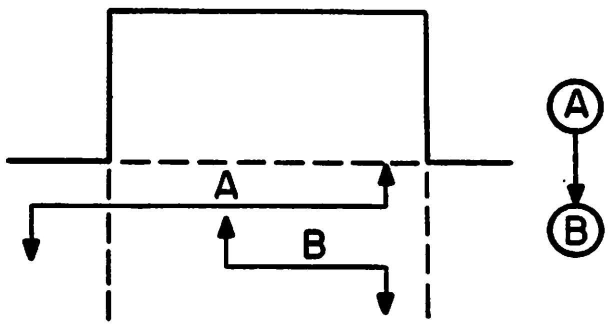


Fig. 5

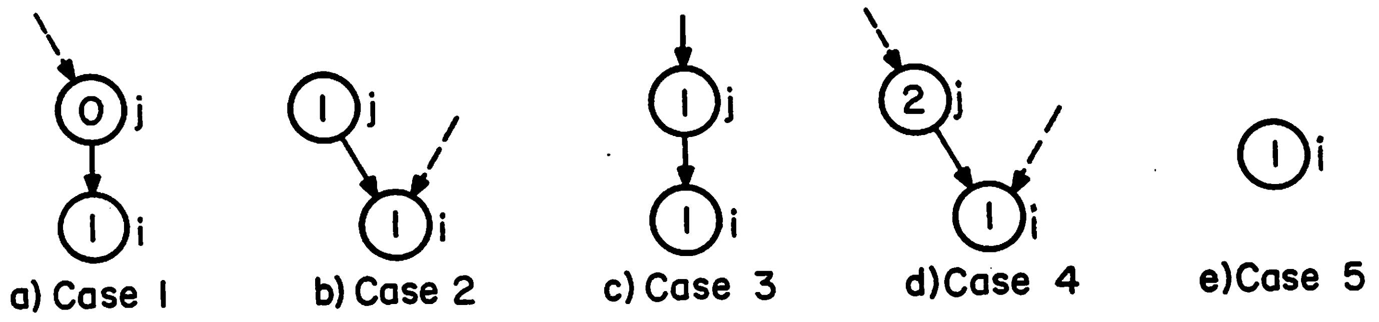


Fig. 6

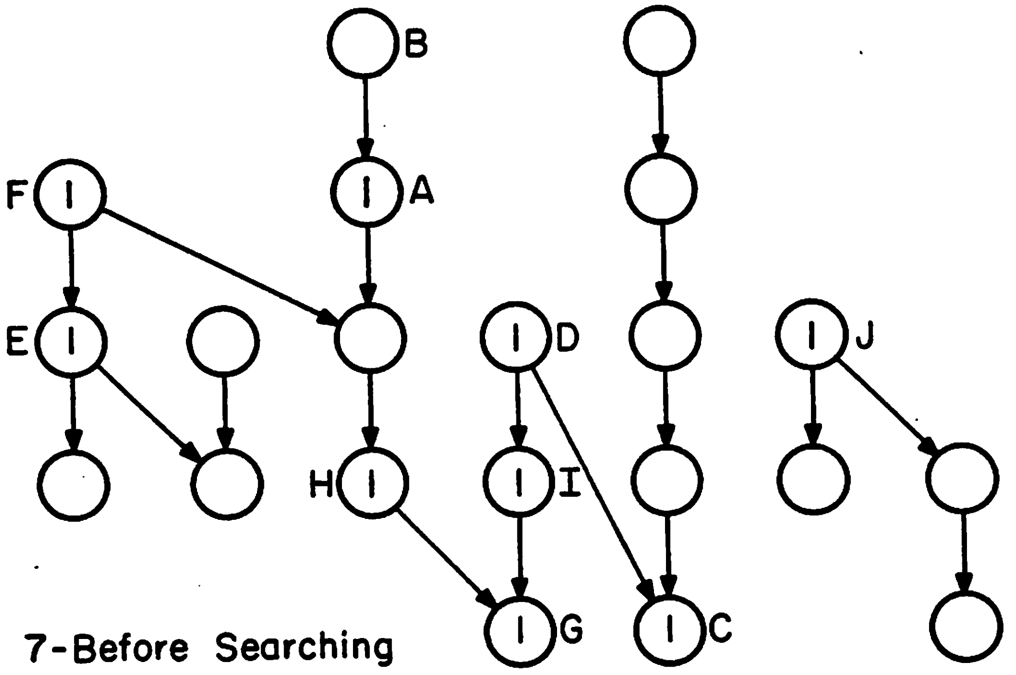


Fig. 7-Before Searching

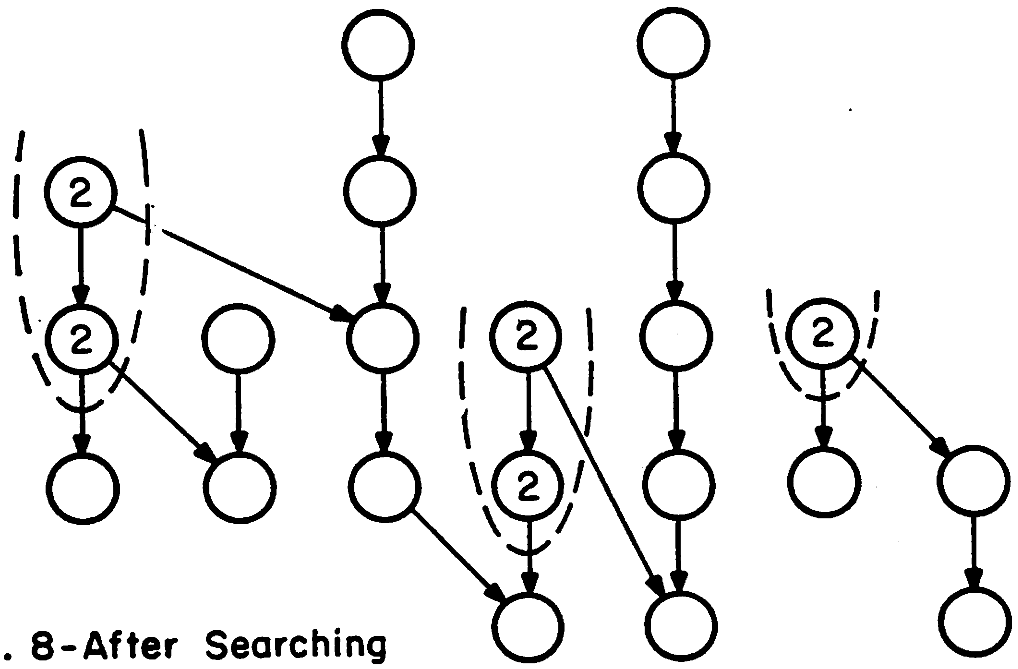


Fig. 8-After Searching

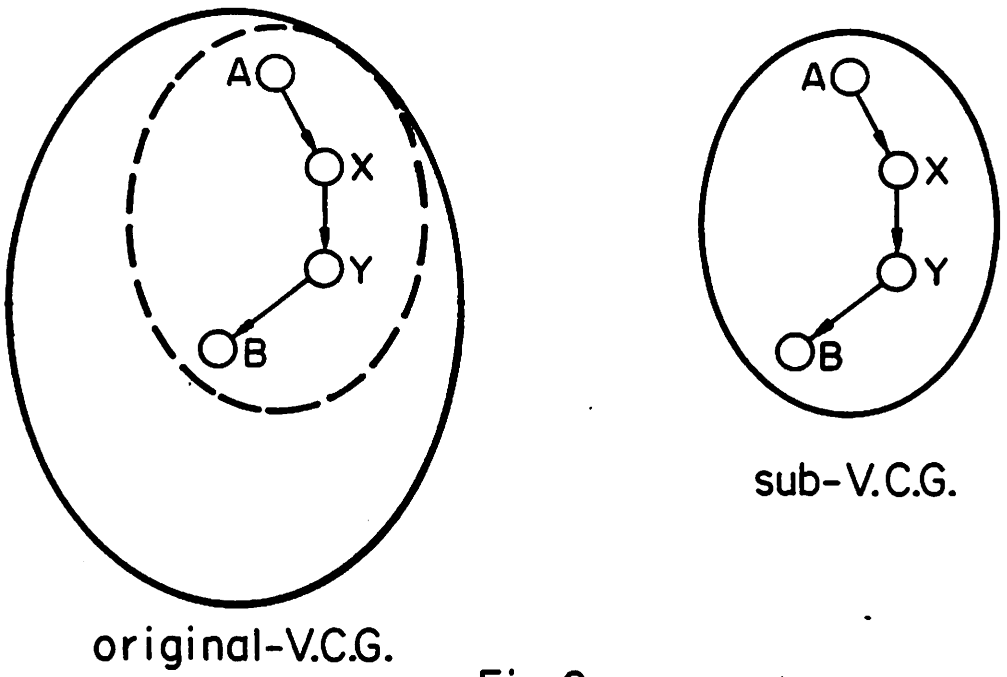


Fig. 9

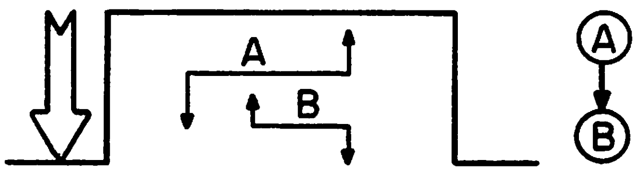


Fig. 10

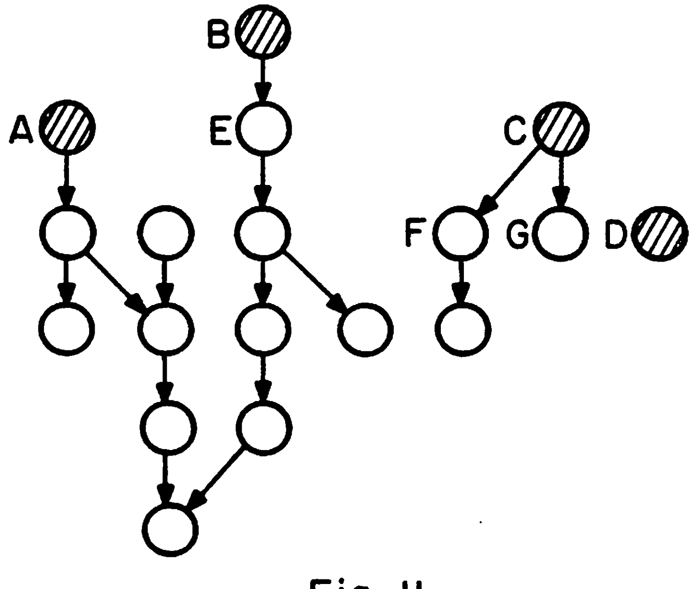


Fig. 11

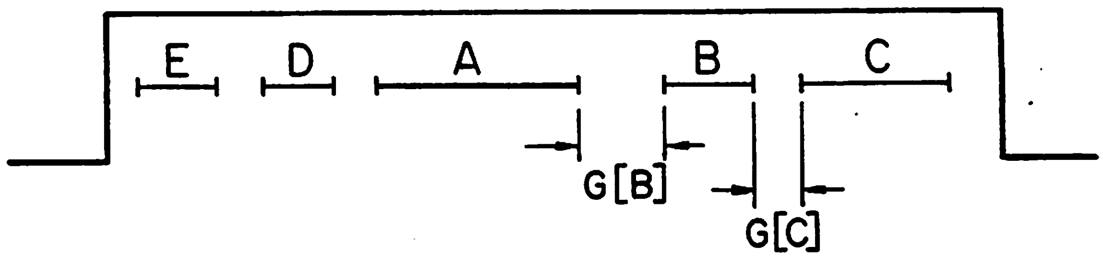


Fig. 12

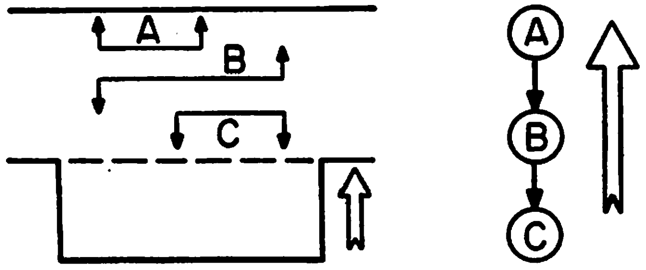


Fig. 13

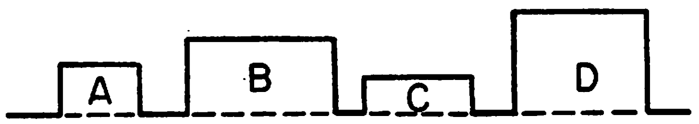


Fig. 14

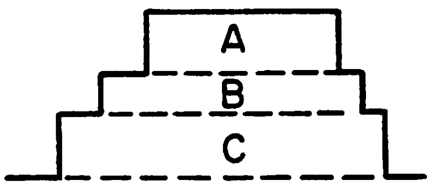
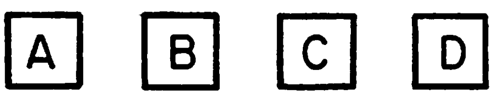
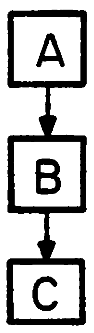


Fig. 15



(a)



(b)

Fig. 16

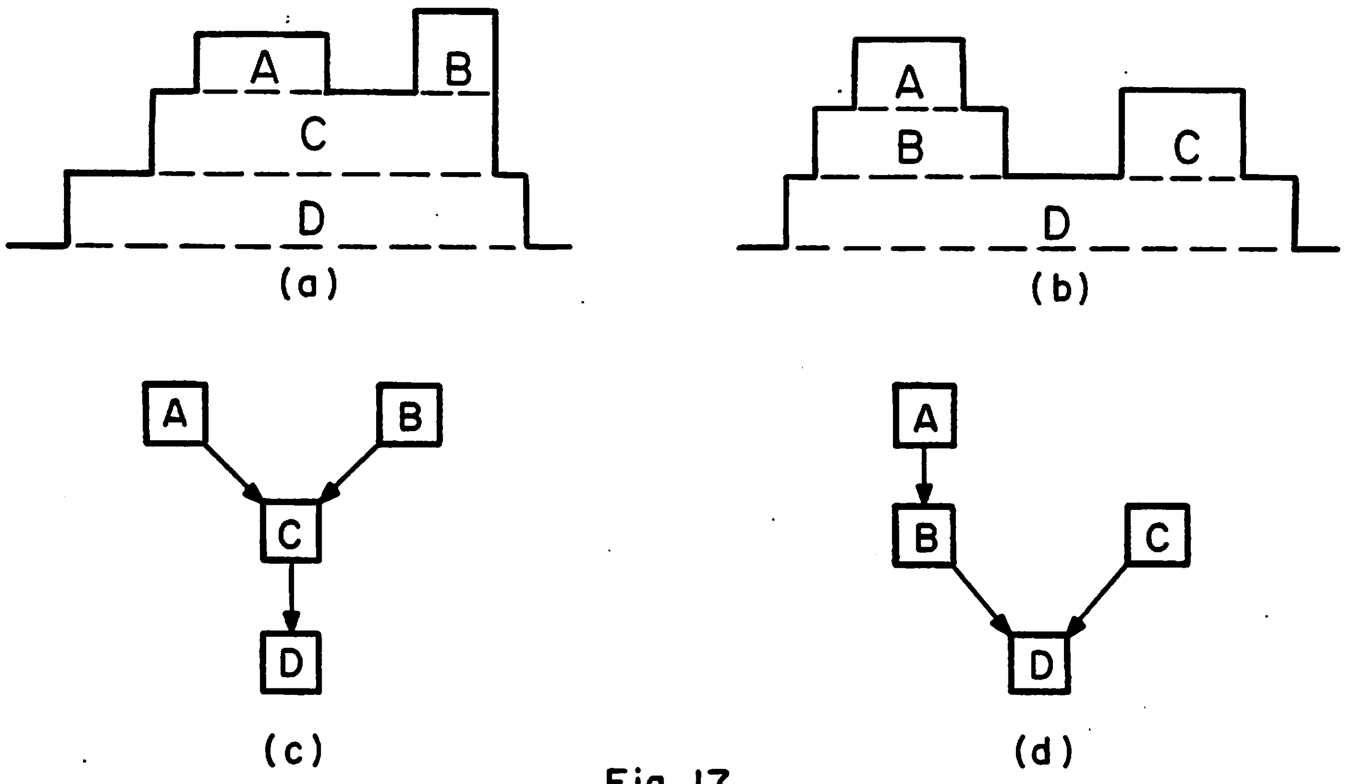


Fig. 17

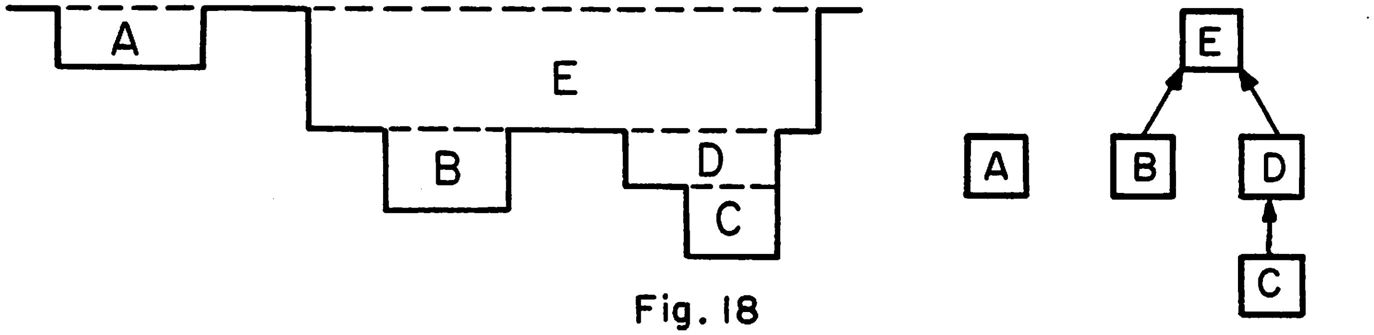


Fig. 18

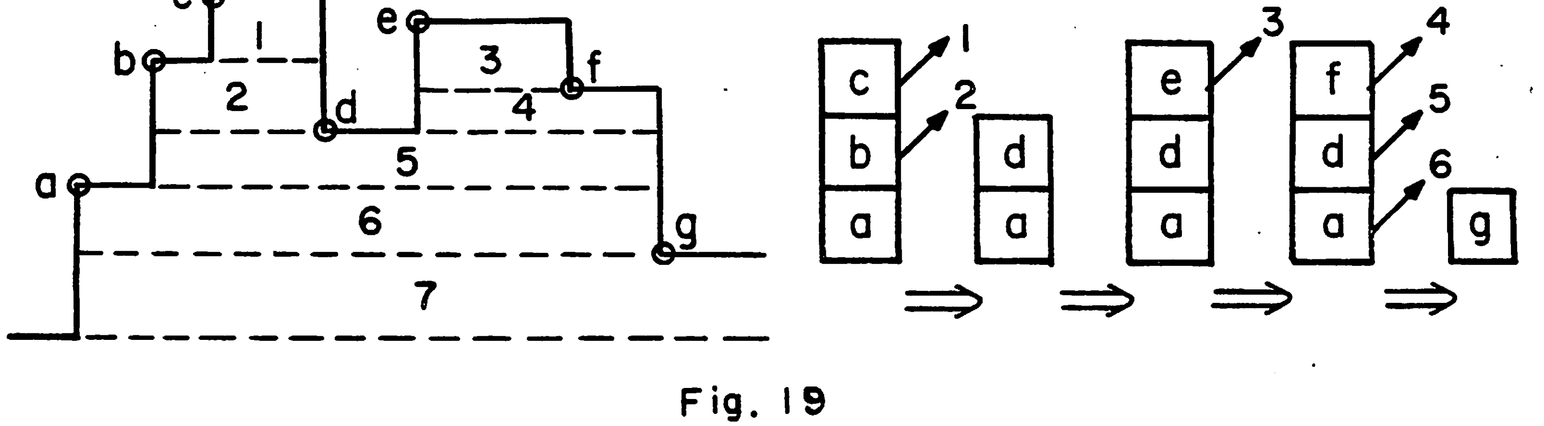
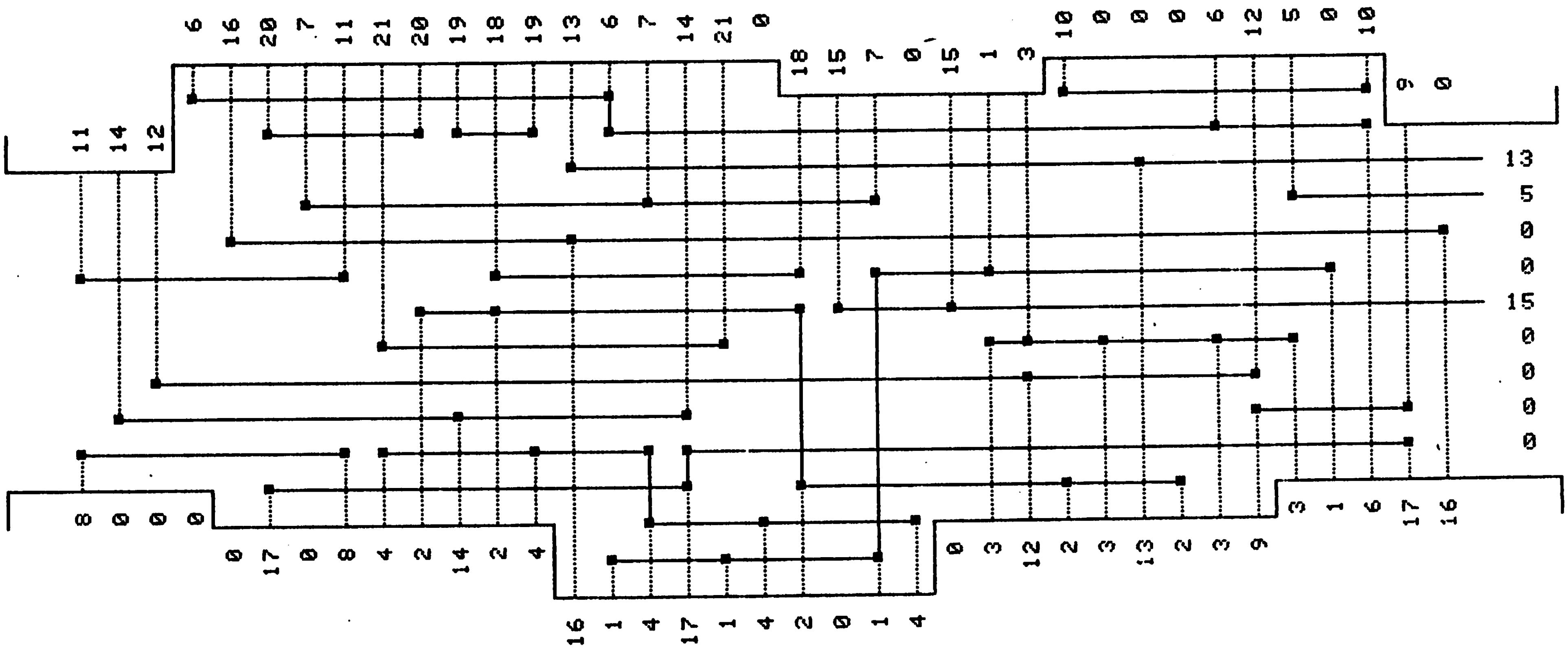


Fig. 19

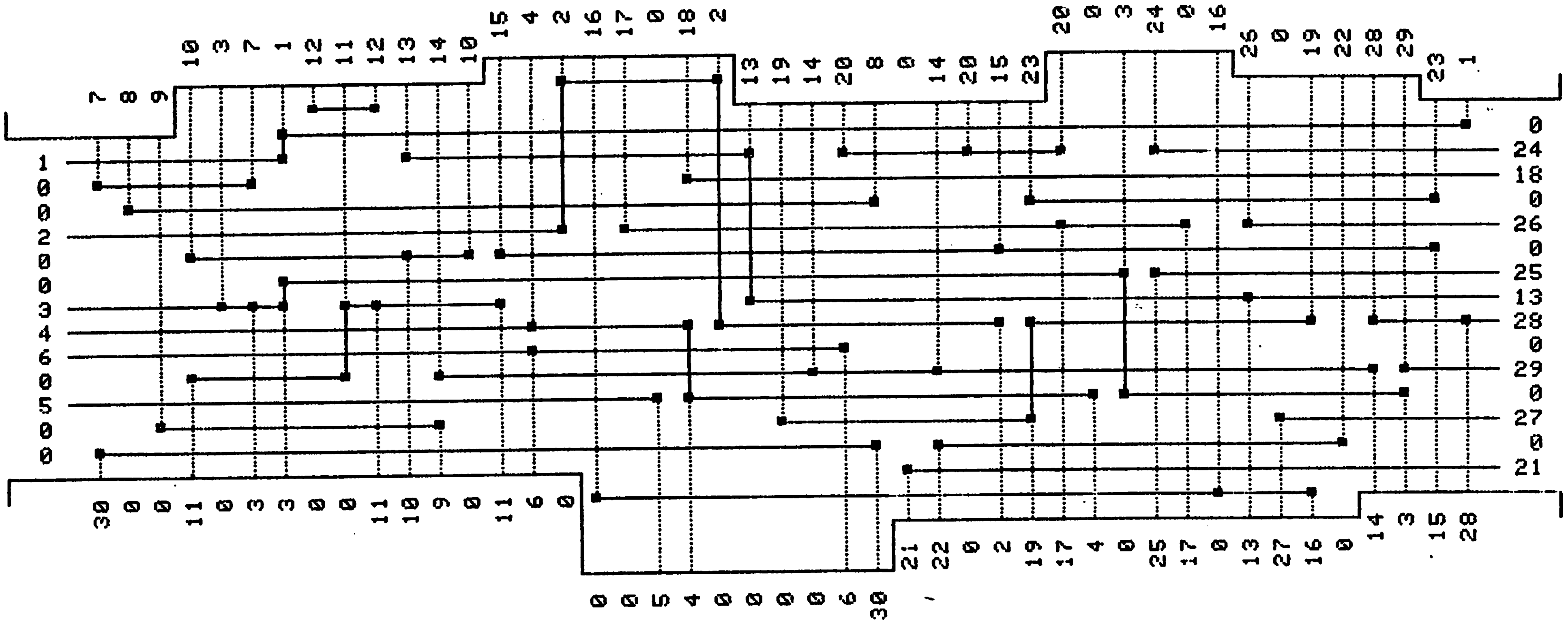
```
**************example 1*******************
           ( with dogleg )
```



```
# column= 40      # zone= 23    before merging vmax=  6    maximum density= 12
   # net= 21   start zone=  8    after merging vmax=  6   number of tracks=  8
```

Fig. 20

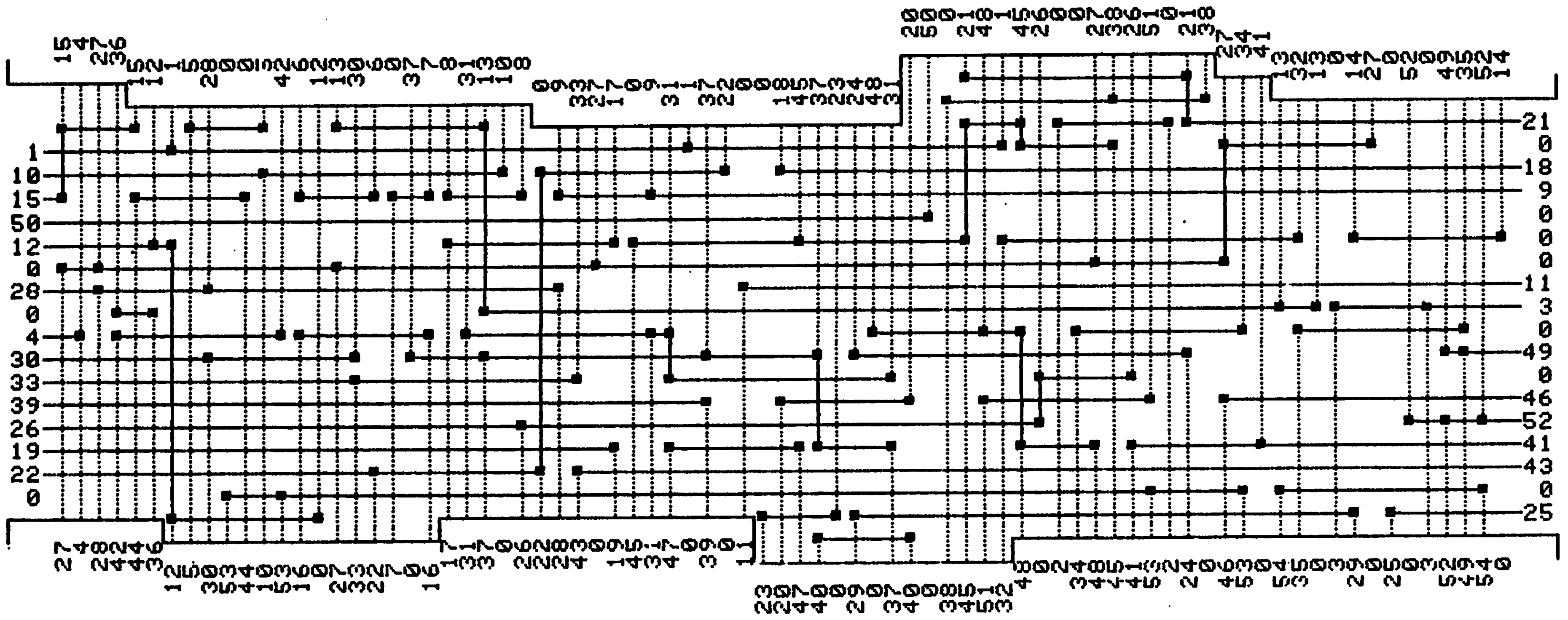
```
****** example 3a ******
     ( with dogleg )
```



```
# column= 60      # zone= 28     before merging vmax=  4    maximum density= 15
   # net= 30  start zone= 16      after merging vmax=  5   number of tracks= 13
```

Fig. 21

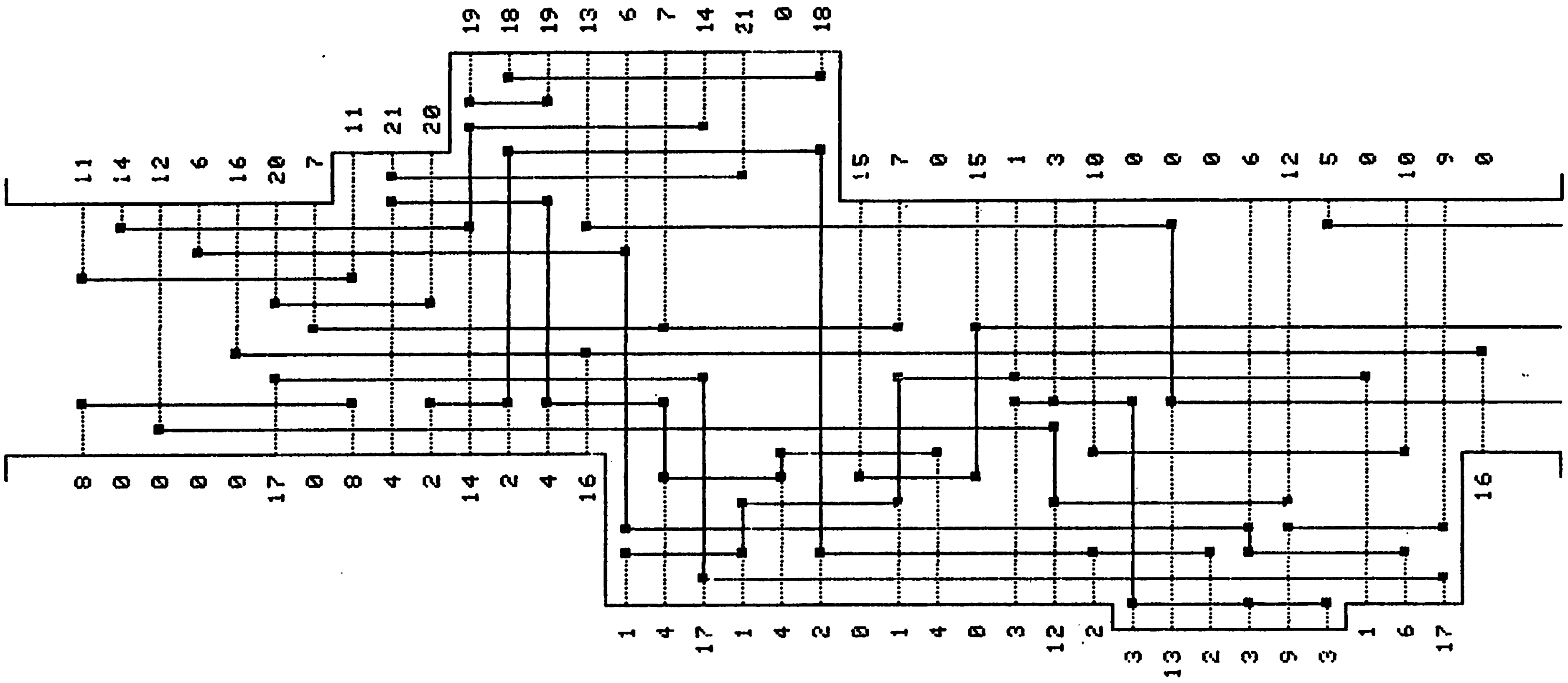
****** example 3c ******
(with dogleg)



```
# column=103          # zone= 50     before merging vmax=  6     maximum density= 18
    # net= 54    start zone= 23       after merging vmax=  7    number of tracks= 16
```

Fig. 22

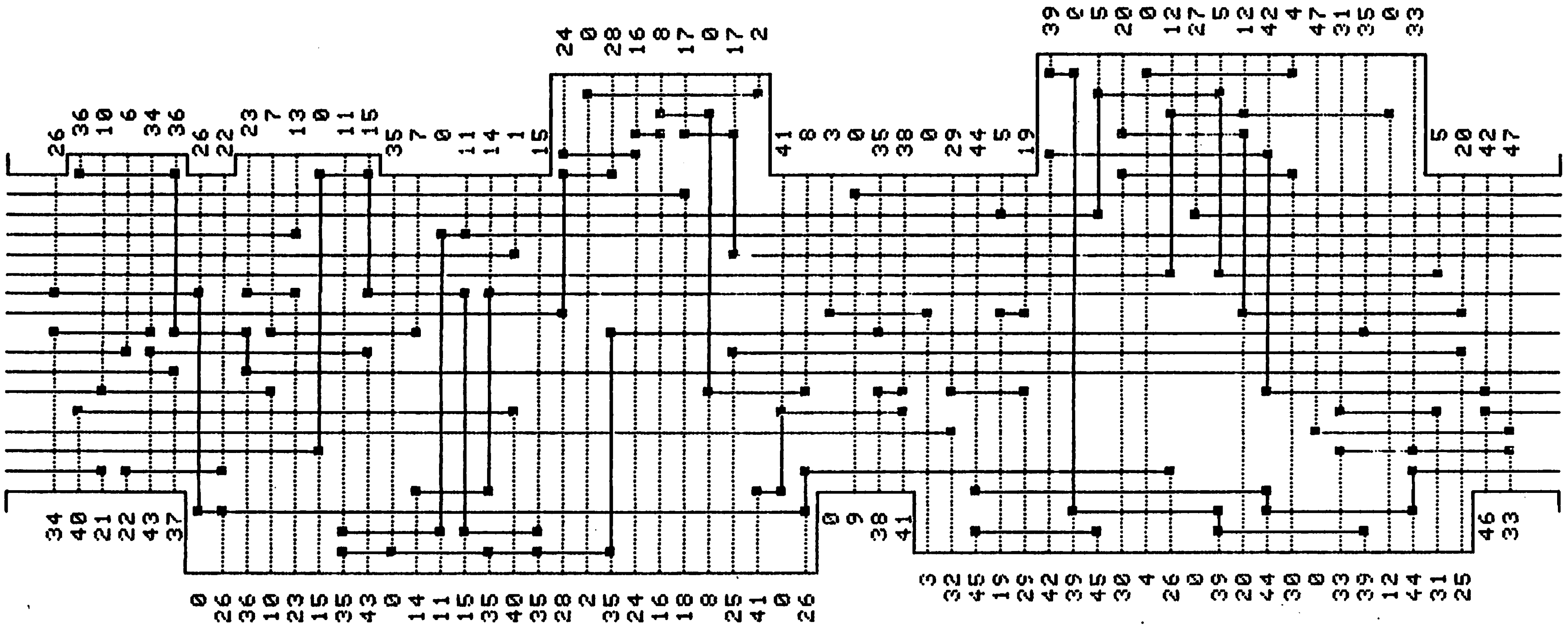
```
**************example 1*******************
              ( with dogleg )
```



```
# column= 40        # zone= 23    before merging vmax=  6    maximum density= 12
   # net= 21   start zone=  9      after merging vmax=  4   number of tracks=  9
```

Fig. 23

```
******* ex.3b *******
   ( with dogleg )
```



```
# column= 84        # zone= 42    before merging vmax=  7    maximum density= 17
   # net= 47    start zone=  5     after merging vmax=  7   number of tracks= 15
```

Fig. 24