EFFICIENT ALGORITHMS FOR CHANNEL ROUTING

by

T. Yoshimura and E. S. Kuh

Memorandum No. UCB/ERL M80/43

11 August 1980

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

EFFICIENT ALGORITHMS FOR CHANNEL ROUTING*

T. Yoshimura** and E. S. Kuh

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, California 94720

ABSTRACT

In the layout design of LSI chips, channel routing is one of the key problems. The problem is to route a specified net list between two rows of terminals across a two-layer channel. Nets are routed with horizontal segments on one layer and vertical segments on the other. Connections between two layers are made through via holes.

Two new algorithms are proposed. These algorithms merge nets instead of assigning horizontal tracks to individual nets.

The algorithms were coded in FORTRAN and implemented on a VAX 11/780 computer. Experimental results are quite encouraging. Both programs generated optimal solutions in 6 out of 8 cases, using examples in previously published papers. The computation times of the algorithms for a typical channel (300 terminals, 70 nets) are 1.0 second and 2.1 seconds, respectively.

---

*Research supported by the National Science Foundation Grant ENG-78-24425 and the Alexander von Humboldt Foundation.

**On leave from Nippon Electric Company, Japan.

1. Introduction

The routing problem in LSI layout is to realize a specified interconnection among modules in as small an area as possible. Several routing strategies are available. Among them, channel routing is the most important one; because (1) it is efficient and simple, (2) it guarantees 100% completion if constraints are noncyclic and channel height is adjustable, and (3) it is used in the layout design of custom chips as well as uniform structures such as gate arrays or polycells. Left edge algorithm [1,2,3] is commonly used for this problem. However, here we propose two algorithms which produce better solutions in all cases tested.

2. Description of the Problem

Two rows of terminals are placed on both sides of a channel. A number between 0 and N is assigned to each terminal. Terminals with the same number i ($1 \leq i \leq N$) must be connected by net i, while terminals with number 0 are left unconnected.

Two layers are available for routing. We assume horizontal tracks on one layer and vertical tracks on the other. Nets are laid on tracks. Horizontal tracks are isolated from vertical tracks, and connections between them are made through via holes. The problem is expressed by a net list as shown in Fig. 1. Arrows indicate whether nets are to be connected to terminals in the upper or lower sides of the channel. Fig. 2 shows a solution of this example.

Consider the example in Fig. 3(a). Because the vertical segment of one net cannot overlap that of another on the same track, a constraint has been introduced on the horizontal segments of net 1 and net 2. If we pay attention to the leftmost column, the horizontal segment of net 1 must be placed above that of net 2. By the same reasoning, net 2 must be above net 1 if the rightmost column is considered. In other words, this routing

requirement cannot be realized without dividing the horizontal segment of some net, as shown in Fig. 3(b). However, this kind of situation can be often avoided by rearranging the placement. In this paper, we assume that the routing requirement is always satisfied, i.e., there exists no cyclic conflict in the net list.

One objective function of the problem is to minimize the number of horizontal tracks used to realize the routing requirement.

Consider the example in Fig. 4(a). This gives an optimal solution if splitting of horizontal segments is not allowed. However, the same example can be realized with two tracks by horizontal splitting as shown in Fig. 4(b).

The splitting of horizontal segments of nets is called "doglegging." This is not only used to avoid the vertical conflict as mentioned above, but also used to minimize the number of horizontal tracks. The latter is more important than the former. In the case of doglegging, we assume that the horizontal splitting of a net is allowed at the terminal positions only, which implies that no additional vertical track is allowed.

3. Definitions

3.1 Vertical constraint graph

As mentioned earlier, any two nets must not overlap at a vertical column. If we assume that there is only one horizontal segment per net, then it is clear that the horizontal segment of a net connected to the upper terminal in a given column must be placed above the horizontal terminal in that column.

This relation can be represented by a directed graph G_v, where each node corresponds to a net and a directed edge from net a to net b means that net a must be placed above net b (Fig. 5). Therefore, if there is a cycle, the routing requirement cannot be realized without dividing some

nets. (Cycle $a \rightarrow b \rightarrow c \rightarrow a$ means that net a must be placed above itself.) However, it should be noted here that if the vertical constraint graph is acylic the routing requirement is always realizable.

3.2 Ancester and descendent

Node i is said to be ancester of node j (node j is descendent of node i), if there is a directed path from i to j in the vertical constraint graph.

3.3 Zone representation of horizontal segments

The horizontal segment of a net is determined by its leftmost and rightmost terminal connections. Let S(i) be the set of nets such that their horizontal segments intersect column i. Since horizontal segments of distinct nets must not overlap, the horizontal segments of any two nets in S(i) must not be placed on the same horizontal track. This condition must be satisfied at every column. However, it is easy to see that we only have to consider those S(i) which are not subsets of another set. Therefore, we assign the sequential number to the columns at which S(i) are maximal and these columns define zone 1, zone 2, etc. as shown in Table 1 for the example in Fig. 1. The number of elements in S(i) is called local density, and the maximum among them is called maximum density.

Clearly, we need not consider S(1) or S(2) because the horizontal constraints related to these sets are included in that of S(3). Zone representation is expressed as shown in Fig. 6(a).

Zones can be defined more clearly by using an interval graph based on a horizontal segment of nets. A graph $G(\overline{V},E)$ is an interval graph corresponding to a set of nets, where $v_i \in \overline{V}$ represents net n_i and $(v_i, v_j) \in E$ iff $n_i \cap n_j \neq 0$ [4,5]. The interval graph of the net list in Fig. 1 is shown in Fig. 6(b). In terms of an interval graph, a zone

is defined by a maximal clique and the clique number is the density.

Table 1

Column	S(i)	Zone
1	2	
2	1 2 3	
3	1 2 3 4 5	1
4	1 2 3 4 5	
5	1 2 4 5	
6	2 4 6	2
7	4 6 7	3
8	4 7 8	
9	4 7 8 9	4
10	7 8 9	
11	7 9 10	5
12	9 10	

It should be emphasized that the channel routing problem is completely described by the vertical constraint graph and the zone representation. Our problem is to determine an optimum ordering of nets such that (1) the vertical constraints as expressed by G_V are satisfied, and (2) the number of tracks needed for realization inherent in the zone representation is minimized. A lower bound is of course the maximum density.

3.4 Dogleg

So far, the vertical constraint graph and the zone representation are defined for the no-dogleg problem where the number of horizontal segments per net is limited to one. However, these two representations can be extended for the dogleg problem, where the horizontal segment of a net can be divided at its terminal positions. In this case, subnets whose horizontal segments are parts of a net between two consecutive terminals are introduced.

Fig. 7 gives a four-net example indicating subnets and cuts corresponding to maximum cliques. Fig. 8 shows the vertical constraint graph and the zone representation of the net list in which dogleg is not allowed. In the dogleg problem, subnets 1-a,1-b,...,4-b are considered instead of nets and the vertical constraint graph and the zone representation are shown in Fig. 9.

4. Simple Algorithm

To find an optimum realization of the channel routing problem may be very difficult. It may take an enormous amount of computation time for problems of practical size. Therefore, we propose heuristic algorithms which generate optimum or near optimum solutions with a reasonable amount of computation time.

4.1 Merging of nets

Let net i and net j be the nets such that there is no horizontal overlap in the zone representation, and no directed path between net i and net j in the vertical constraint graph, i.e., net i and net j can be placed on the same horizontal track. Then, the merging operation replaces net i and net j by a net i·j as illustrated in Fig. 10 and Fig. 11. Here net 6 and net 9 are candidates for merging. The operation "merging of nets" modifies the net list asshown in Fig. 11, where net 6 and net 9 are replaced by net 6·9. By merging, we mean that net 6 and net 9 are placed on the same horizontal track, although the position of the track is not yet decided. Consequently, the zone representation and the vertical constraint graph are updated as shown in Fig. 12.

4.2 Algorithm #1

Since it is assumed that there is no cycle in the vertical constraint graph, the updated vertical constraint graphs do not have cycles either. Hence, we can repeat this operation. The following algorithm merges nets

systematically according to the zone representation.

Algorithm 1

```
     procedure Algorithm 1 (i,j)
     begin ;
a1:    L = { } ;
a2:    for z = i to j do ;
       begin ;
a3:      L = L + {nets which terminate at zone z} ;
a4:      R = {nets which begin at zone z + 1} ;
a5:      merge L and R so as to minimize the increase
         of the longest path legnth in the vertical
         constraint graph ;
a6:      L = L - n1, n2,..., where nj is a net merged
         at step a5 ;
       end ;
     end ;
```

If there is a path $n_1 - n_2 - n_3 - \ldots, - n_k$ in the vertical con-constraint graph, then no two nets among $n_1, n_2, \ldots, n_k$ can be placed on the same track. Therefore, if the longest path length in terms of the number of nodes on the path is k, at least k horizontal tracks are necessary to realize the interconnections. Thus, nets are merged so as to minimize the longest path length as much as possible.

Fig. 13 illustrates how the vertical constraint graph is updated by the algorithm. First, net 5 and net 6 are merged, then net 1 and net 7, ..., and at the fourth iteration net 10 and net 4 are merged. Finally, we have the graph in Fig. 13(e). At this stage, the problem has almost been solved. The remaining task is to assign the horizontal tracks to each node of the graph. This is almost trivial. For example, we can assign track 1 to net 10·4, track 2 to net 1·7, track 3 to net 5·6·9, track 4 to net 2 (or net 3·8) and track 5 to net 3·8 (or net 2). Fig. 14 shows the solution corresponding to the graph in Fig. 13(e).

4.3 Merging algorithm

The key part of the algorithm is step a5 where two sets of nets are merged. In the following, this process is explained. To make the situation precise, let us introduce several definitions.

(1) $P = \{n1, n2, \ldots, np\}$ and $Q = \{m1, m2, \ldots, mq\}$ $(p \geq q)$ are the two sets of nets to be merged. Obviously, elements of P are on separate vertical paths from that of Q.

(2) A modified vertical constraint graph is defined as shown in Fig. 15, where two fictitious nodes s and t are added, corresponding to a source and a sink, respectively.

(3) $u(n)$, $n \in P \cup Q$: the length of the longest path from s to n

(4) $d(n)$, $n \in P \cup Q$: the length of the longest path from n to t

Example:

$Q = \{6,7\}$

$P = \{1,3,4\}$

$u(1) = 1, \quad u(3) = 3, \quad u(6) = 2, \ldots$

$d(1) = 4, \quad d(3) = 2, \quad d(6) = 2, \ldots$

The purpose here is to minimize the length of the longest path after merger. A heuristic merging algorithm will be given. First, let us introduce some basic intuitive ideas. First, a node $m \in Q$ is chosen, which lies on the longest path before merger; furthermore, it is farthest away from either s or t. Next, a node $n \in P$ is chosen such that the increase of the longest path after merger is minimum. In case of tie, we will choose n such that the condition $\frac{u(m)}{d(m)} = \frac{u(n)}{d(n)}$ is satisfied or nearly satisfied. These can be implemented by introducing the following:

(1) for $m \in Q$

$$f(m) = C_\infty * \{u(m) + d(m)\} + \max\{u(m), d(m)\}, \quad C_\infty >> 1$$

(2) for $n \in P$, $m \in Q$

$$g(n,m) = C_\infty * h(n,m) - \{\sqrt{u(m)*u(n)} + \sqrt{d(m)*d(n)},\}$$

where

$$h(n,m) = \max\{u(n), u(m)\} + \max(d(n), d(m)\}$$
$$- \max\{u(n) + d(n), u(m), + d(m)\}$$

- - the increase of the longest path length passing through n or m, by merging of n and m

<u>Merging algorithm</u>

```
given P, Q    ;
    begin    ;
       while Q is not empty do    ;
       begin    ;
          among Q, find m* which maximizes f(m)    ;
          among P, find n* which minimizes g(n,m*), and
          which is neither ancestor nor descendent of m*    ;
          merge n* and m*    ;
          remove n* and m* from P and Q respectively    ;
    end    ;
```

For the example given above, let us pick $C_\infty = 100$, we obtain the following table:

	P			Q	
	1	3	4	6	7
u( )	1	3	4	2	3
d( )	4	2	1	2	1
f(m)				402	403

Table 2a

Since $f(7) > f(6)$, node 7 is chosen first from Q. Next, we evaluate $g(n,7)$ using $C_\infty = 100$ and obtain the following:

n	1	3	4
h(n,7)	2	0	0
g(n,7)	196.27	-4.41	-4.46

Table 2b

Since g(4,7) is the smallest, we merge node 4 with node 7.

5. Improved Algorithm

5.1 Overview of the algorithm

In Algorithm #1, nets are merged when they are processed, and the merging may block the subsequent ones. Fig. 16 shows an example. Let us assume that, at zone 1, Algorithm #1 merges net a and net d, net b and net e, respectively (if we follow the merger algorithm of the last section, these mergings will not occur, but they are assumed for explanation). The vertical constraint graph and the zone representation are modified as shown in Fig. 17. The merged vertical constraint graph indicates that net f cannot be merged with either net c or net g because a cycle would be created. On the contrary, if net a and net d, net c and net e are merged, respectively, net f can be merged with net b.

To avoid this type of problem as much as possible, or to make the algorithm more flexible, we introduce another algorithm. In Algorithm #2 we construct a bipartite graph G_n, where a node represents a net and an edge between net a and net b means that net a and net b can be merged. A merging is expressed by a matching on the graph, and it can be updated dynamically. We will explain this idea by using the previous example.

At zone 1, we can see that net d (as well as net e) can be merged with any of three nets a, b, or c. So, the algorithm constructs the bipartite graph G_h in Fig. 18, and a temporary merging is indicated by a matching on the graph. The algorithm checks if the merging is feasible, but neither the vertical constraint graph nor the zone representation is

Step 2: for each net n_r which begins at zone z_{i+1}, add node n_r to the right side of graph G_h, and add edges between n_r and a node on the left side, if they can be merged ;

Step 3: check if the merging based on the current matching satisfies the vertical constraints. If not, modify the matching and graph G_h ;

Step 4: for each net n_l terminating at zone z_{i+1}, merge n_l with the net n_x specified by the matching on graph G_h. Then put the merged net $n_l \cdot n_x$ to the left side of G_h ;

end ;

end ;

5.2.1. Step 1

This step is for initialization. The algorithm simply adds nodes corresponding to the nets terminating at zone z_s. There is no edge in graph G_h at this stage.

5.2.2. Step 2

A node corresponding to a net originating at zone z_{i+1} is added to the right side of graph G_h. If the net can be merged with the net corresponding to a node on the left, the edge between them is added.

To reduce the cpu time and memory requirement, the number of edges per node is limited by a parameter. (Its value is fixed at 3 in the program.) Edges are selected according to the same intuitive ideas as in the Merging Algorithm of the last section.

The matching is updated by using a maximal flow algorithm.

5.2.3 Step 3

Here, we have to solve the following two problems:

updated at this stage. Next, we move to zone 2, where net g terminates at this zone and net f begins at the next zone. So, we add node g to the left side and node f to the right side of the graph G_n as shown in Fig. 19(a). Since the graph G_v in Fig. 16(b) indicates that net f can be merged with either net a, net b or net c, three edges are added and matching is also updated as shown by the heavier lines in Fig. 19(a). Of course there is no guarantee that the merging which corresponds to the updated matching satisfies the vertical constraint (horizontal constraints are satisfied automatically), so the algorithm checks the constraints and modifies the matching as shown in Fig. 19(b). This process will be explained later.

At zone 3, net d and net f terminate. This means that, in processing zone 3, node d and node f should be moved to the left side in graph G_n and merged with their partner nets a and b, respectively, as shown in Fig. 20(a). Net c and net e have not been merged yet, since e has not terminated. The vertical constraint graph is also updated as shown in Fig. 20(b). A matching is next sought for the updated G_h. The procedure will continue until all zones have been processed.

5.2 <u>Algorithm #2</u>

The general flow of Algorithm #2 is as follows.

```
proc. Algorithm #2 (z_s, z_t)   ;
begin   ;
Step 1:  for each net n_1 terminating at zone z_s, add node
         n_1 to the left side of graph G_h   ;
         for z_i = z_s to z_t do   ;
         begin   ;
```

(1) Vertical conflict check for the current matching.

(2) Update matching (when conflict is found).

It is obvious that the solution corresponding to any matching on graph G_h does not have horizontal overlaps because G_h is constructed based on the horizontal constraints. However, vertical constraints are not totally considered, so there may be vertical conflict. Fig. 21 shows an example, where the realization corresponding to the matching in Fig. 21(c) is not feasible.

The following gives a procedure to consider the contraints imposed by G_y. By using Algorithm A below, we derive conditions under which any matching in G_h will lead to feasible solutions.

<u>Algorithm A</u>

Given graph $G_h = (N, E_h)$ and graph $G_v = (n, E_v)$;

begin

c1: $E_x = 0$;

while N is not empty do ;

begin ;

c2: let N_o be the set of nodes which do not have ancestors in G_v ;

c3: remove a set of edges E_o from graph G_h, where $E_o = \{(i,j) \mid i,j \in N_o, (i,j) \in E_h\}$;

c4: if there is a node whose degree is equal to zero in graph G_h

c5: then let it be v,

c6: otherwise, choose one of the nodes $v \in N_o$, which has the smallest number of edges incident to it and let $E_x = E_x + E_y$, where E_y is the set of edges connecting to node v ;

c7: remove node v and connecting edges from graph G_h and graph G_v ;

end ;

end ;

Fig. 22 shows how the algorithm works. In the first iteration, $N_0 = \{a,c,d\}$ and two edges (a,c) and (a,e) are removed from G_h. Then node a is removed from G_h and G_v because the degree of node a in G_h is equal to zero etc. ....

Theorem

The merging corresponding to any matching on graph G_h is feasible if and only if E_x is empty.

Proof

We first prove that if E_x is empty, any matching of G_h is feasible. Let N_0 be the set of nodes with no ancestors in G_v. Obviously, those edges in G_h which connect nodes in N_0 and which are removed in c3 in the first go around are feasible edges for matching. Let $N_0^* \subset N_0$ be the set of isolated nodes in G_h after the removal of the edges above. These nodes are connected in G_h only to nodes in N_0 and are obvious candidates for merger; therefore, once deleted, they can be forgotten as far as G_v or subsequent merging is concerned. Let N_1 be the second set of nodes with no ancestors. This set consists of new nodes in $\overline{N}_1 \subset N_1$, while the rest belong to $N_0 - N_0^*$. Clearly, nodes in $\overline{N}_1$ are descendents of N_0^* and they are not descendents of $N_0 - N_0^*$. The edges in G_h removed at this step are therefore similarly feasible for matching. We next delete $N_1^* \subset N_1$ which are the isolated nodes. Since $E_x = 0$, the process continues until all edges are removed.

We next prove that if E_x is not empty, there exists a matching which is not feasible. We start as in the previous paragraph, but assume that

there exists no isolated node after the removal of the initial set of edges. Let $n_o^{(1)} \in N_o$ be a node in N_o. Then there is at least one edge in G_h which connects node $n_o^{(1)}$ and a node, say $n_d^{(1)}$ which is not in N_o. If node $n_d^{(1)}$ is a descendent of node $n_o^{(1)}$, then merger between them is clearly unfeasible. Thus we assume $n_d^{(1)}$ is a descendent of node $n_o^{(2)}$ in $N_o - \{n_o^{(1)}\}$. Meantime, node $n_o^{(2)}$ is also not isolated after the initial removal of edges. So, it is connected to a node $n_d^{(2)}$ which, by a similar reasoning, is a descendent of node $n_o^{(3)}$ in $N_0 - \{n_o^{(2)}\}$. Thus, we can construct a sequence of nodes $n_o^{(1)}$, $n_d^{(1)}$, $n_o^{(2)}$, $n_d^{(2)}$, $n_o^{(3)}$, $n_d^{(3)}$, ... Since the number of nodes is finite, there must be a "cycle" of nodes in the sequence. Without loss of generality, we can assume that the cycle is $n_o^{(1)}$, $n_d^{(1)}$, $n_o^{(2)}$, $n_d^{(2)}$, ...,$n_o^{(k)}$, $n_d^{(k)}$, $n_d^{(1)}$, and that no two nodes among it are identical. However, the merger according to the set of edges $E_s = \{(n_o^{(i)}, n_d^{(i)}) \mid i = 1,\ldots,k\}$ results in a cycle in the merged G_v, which is unfeasible. This completes the proof of the theorem.

<u>Corollary</u>

The merging corresponding to any matching on graph $G_r = (N, E_h - E_x)$ is feasible.

The example shown in Figs. 23 and 24 illustrates the use of the corollary. Fig. 23(a) gives the graphs G_h and G_v where the initial nodes with no ancestors and the connecting edges are specially marked. With the removal of the heavier edges, node d is isolated and removed, together with edges (d,g). In Fig. 23(b), the new nodes and edges to be considered next are marked. After the removal of edge (a,i), there exists no isolated node. Let us delete node a and remove edge $(a,h) \in E_x$ as shown in Fig. 23(c). The process then continues with no further problem. In Fig. 24(a) we show the horizontal graph G_r with E_x removed. The heavier edges represent an arbitrary matching. The corresponding vertical graph

after merging is shown in Fig. 24(b). In a sense, the bad element (a,h) which could have caused a problem has been deleted before matching.

6. Additional Comments on the Dogleg Problem

So far, we have discussed essentially the problem without doglegs. Of course, the presented algorithms can solve the dogleg problems using the zone representation and the vertical constraint graph introduced in section 3.4. However, additional consideration may be necessary to reduce the number of doglegs and cpu time. Thus we introduce a process "merging of subnets" which merges subnets belonging to the same net to form a net or larger subnet. If we carry this process to the extreme, the problem is reduced to the no-dogleg problem. Hence we impose the following two restrictions on the subnet merging.

Subnet i and subnet j can be merged only if:

(1) merging of net i and net j will not increase the length of the longest paths which pass through node i or node j on the vertical constraint graph.

(2) $L(i) \leftarrow d_{max} - p_1$, $L(j) \leftarrow d_{max} - p_1$, where $L(k)$ is the length of the longest path which passes node k, d_{max} is the maximum density, and p_1 is a parameter.

According to the results of the preliminary experiments, this process significantly reduces the cpu time and improves the solutions in the sense that the number of horizontal tracks and the number of doglegs are both reduced. Parameter p_1 has little effect on the results when its value is $0 \sim d_{max}/2$. Hence, its value is fixed at 3.

7. Computational Results

Algorithm #1 and Algorithm #2 have been coded in FORTRAN and implemented on the DEC VAX 11/780 computer.

7.1 Programs

Both programs have a parameter, the "starting column." In the explanation of algorithms, zone processing is carried out from zone 1 to zone n where n is the number of zones. However, we need not process in this order. In fact, we obtained better solution when we processed the zone with the highest density first. The above parameter specifies the starting zone. Programs process zones from this zone toward zone n (or zone 1), then toward zone 1 (or zone n). Usually, the starting zone is chosen among the zones which have the maximum density.

7.2 No-dogleg problem

Here, no dogleg is allowed. Ex. 1 ~ Ex. 5 are the data taken from the existing paper [1], and the "difficult example" was provided by Deustch and Schweikert of Bell Labs. Table 3 compares the number of horizontal tracks of optimum solutions, the results of left edge algorithms, Algorithm #1 and Algorithm #2. The cpu time for Algorithm #1 and Algorithm #2 are also listed. Figs. 25-31 show the computer outputs of Algorithm #2.

Table 3: No Dogleg

Problem data(#nets)	opt	solution left e	alg. 1(cpu)	alg. 2(cpu)
ex. 1 (21)	12	14	12 (0.05)	12 (0.07)
ex. 2 (30)	15	18	15 (0.07)	15 (0.43)
ex. 3b (47)	17	20	17 (0.17)	17 (0.57)
ex. 3c (54)	18	19	18 (0.18)	18 (0.72)
ex. 4b (57)	17	23	17 (0.23)	17 (0.77)
ex. 5 (62)	20	22	20 (0.22)	20 (0.88)
dif. ex(72)	26*	39	30 (0.40)	28 (1.60)

*Obtained in [1] by means of the method of branch and bound after four hours of computation.

The result indicates that both Algorithm #1 and Algorithm #2 reach the optimum solutions for the data in Ex. 1 - Ex. 5, and obtain considerably better solutions than the left edge algorithms for all examples.

7.3 Dogleg problem

Algorithms were applied for the previous examples and, of course, produced optimum solutions for ex. 1 -- ex. 5. However, some solutions have several doglegs. Table 4 shows the results for the "difficult example." We can see that all three programs calculate near optimum solutions as far as the number of horizontal tracks is concerned. However, the solutions of Algorithm #1 and Algorithm #2 require only 19 and 18 doglegs, respectively, while the solution of LTX has more than 50. Fig. 32 shows the computer output of Algorithm #2.

Table 4: Dogleg

data (#nets)	density	LTX	alg. 1(cpu)	alg. 2(cpu)
dif. ex(72)	19	21	21. (1.0)	20 (2.1)

8. Conclusion

We proposed two new algorithms for the channel routing problem. Algorithm #1 is a simple one, which depends on a process of merging nets instead of assigning horizontal tracks to each net. The basic idea of Algorithm #2 is the same as Algorithm #1, but it uses a matching algorithm to improve the solution. According to the computational results, when doglegs were not allowed, both algorithms produced optimum solutions for five examples taken from a published paper, and better solutions than LTX for a "difficult example." As for the dogleg problem, we used the "difficult problem" to make comparison. Algorithm #1 produced a solu-

tion which has the same number of horizontal tracks but far fewer doglegs than the solution of LTX. Algorithm #2 generated better solution than Algorithm #1 and LTX in the sense that the number of horizontal tracks and the number of doglegs are both smaller,

References

[1] Kernighan, B. W., Schweikert, D. G. and Persky, G., "An Optimum Channel-Routing Algorithm for Polycell Layouts of Integrated Circuits," Proc. 10th Design Automation Workshop, pp. 50-59 (1973).

[2] Persky, G., Deustch, D. N. and Schweikert, D. G., "LTX - a System for the Directed Automation Design of LSI Circuits," Proc. 13th Design Automation Conference (1976).

[3] Persky, G., Deustch, D. N. and Schweikert, D. G., "LTX - a Minicomputer-Based System for Automatic LSI Layout," Jl. Design Automation and Fault-Tolerant Computing, pp. 217-255, (May 1977).

[4] Ohtsuki, T., Mori, H., Kuh, E. S., Kashiwabara, T. and Fujisawa, T., "One-Dimensional Logic Gate Assignment and Interval Graphs," IEEE Trans. on CAS, pp. 675-684 (Sept. 1979).

[5] Lekerkerker, C. G. and Boland, J. Ch., "Representation of a Finite Graph by a Set of Intervals on the Real Line," Fund. Math., 51, pp. 45-64 (1962).

Acknowledgement

The second author wishes to achnowledge the support of Professor Dr.-Ing. R. Saal at the Technical University of Munich.

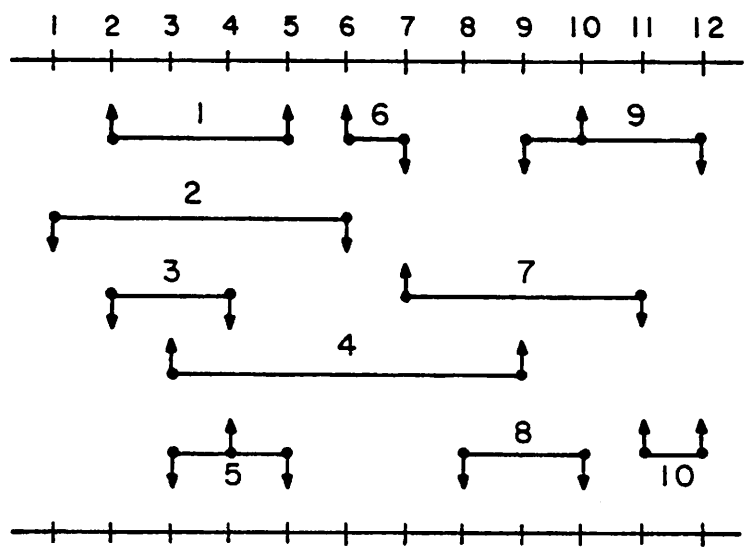


Figure 1. Netlist representation for routing requirement.

$$\begin{pmatrix} 0 & 1 & 4 & 5 & 1 & 6 & 7 & 0 & 4 & 9 & 10 & 10 \\ 2 & 3 & 5 & 3 & 5 & 2 & 6 & 8 & 9 & 8 & 7 & 9 \end{pmatrix}$$

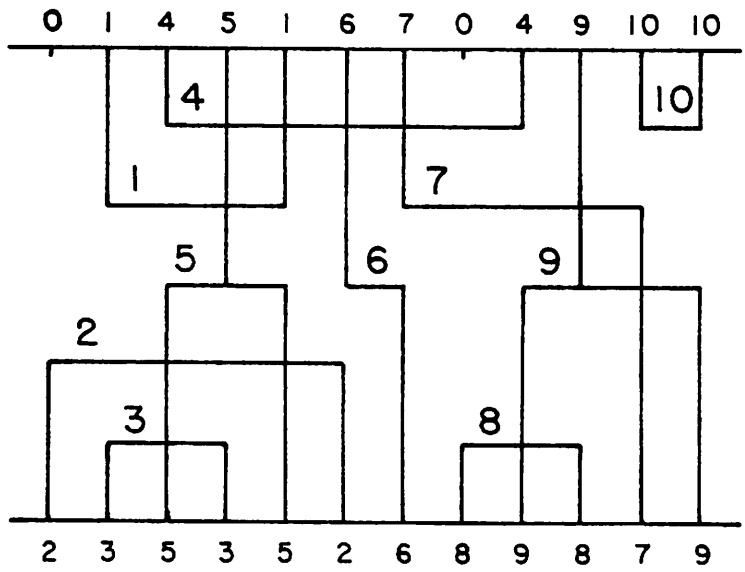


Figure 2. A realization for the requirement in Figure 1.

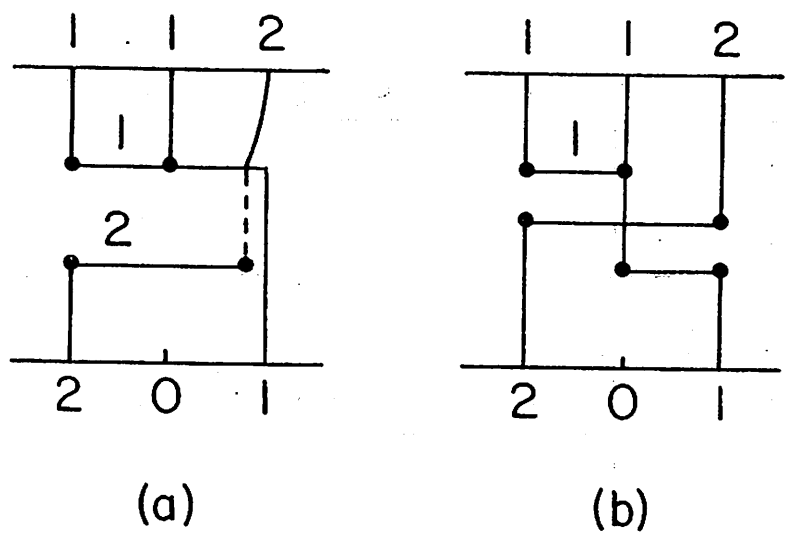


Figure 3. An example with cyclic conflict.

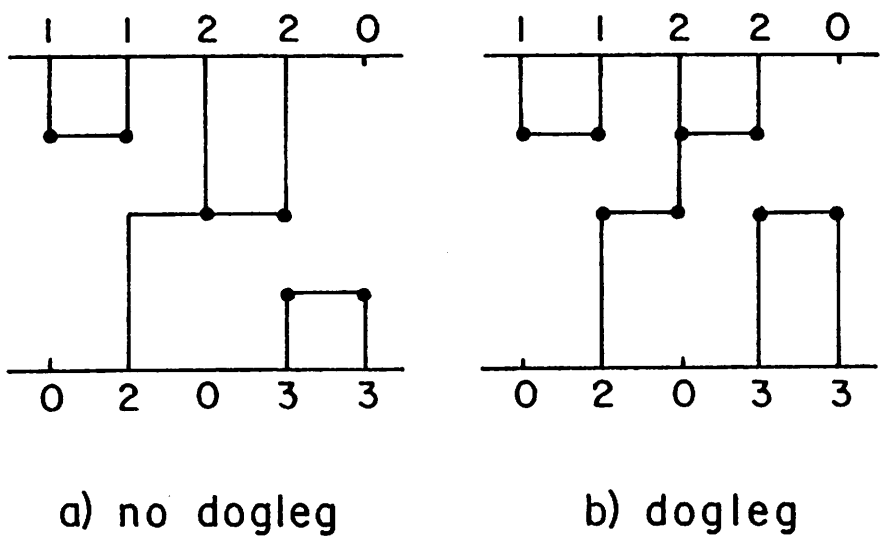


Figure 4. An example illustrating the advantage of using dogleg.

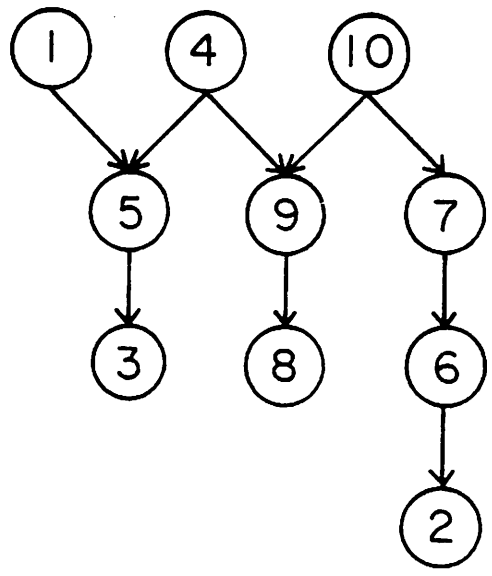


Figure 5. Vertical constraint graph G_v for the netlist in Figure 1.

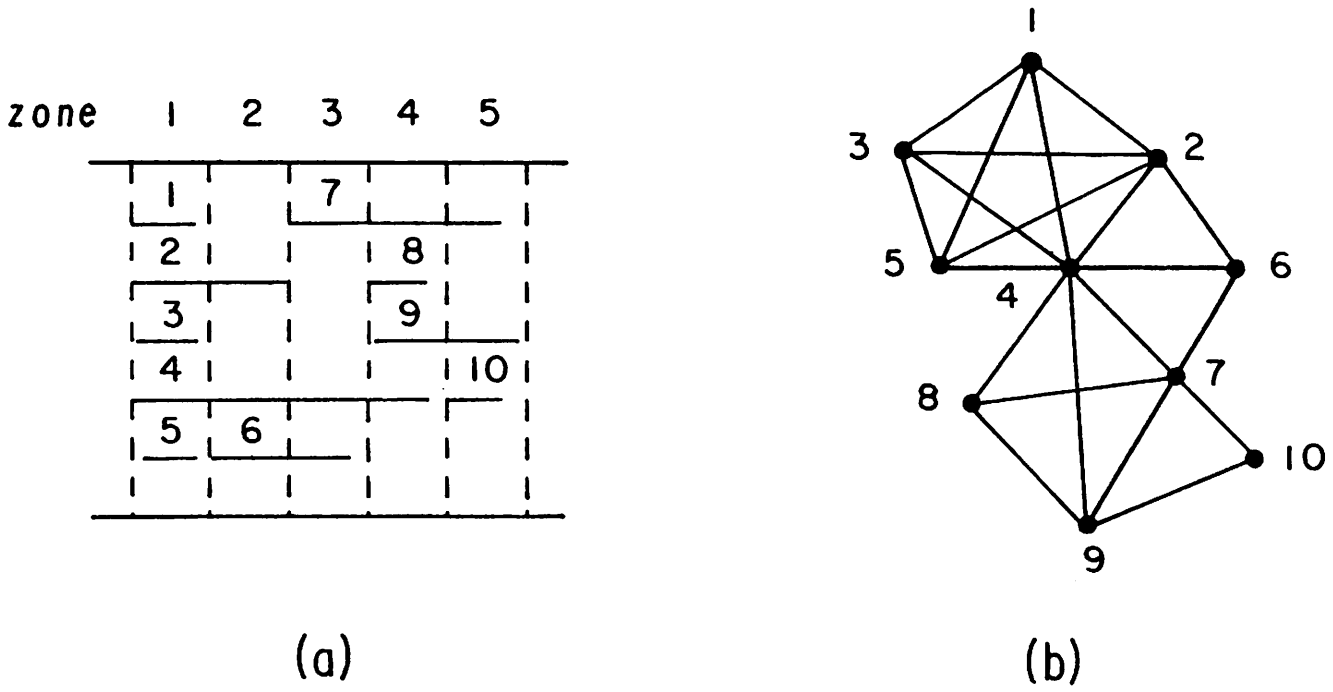


Figure 6. Zone representation and interval graph.

Maximal cliques are 12345, 246, 467, 4798 and 7910.

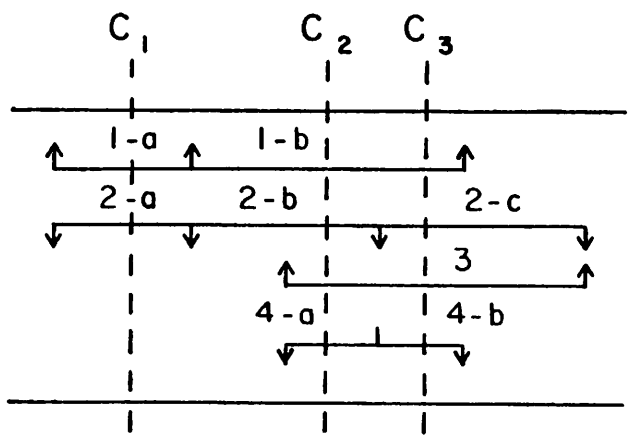


Figure 7. A four-net example illustrating subnets and cuts corresponding to maximal clique C_1, C_2 and C_3.

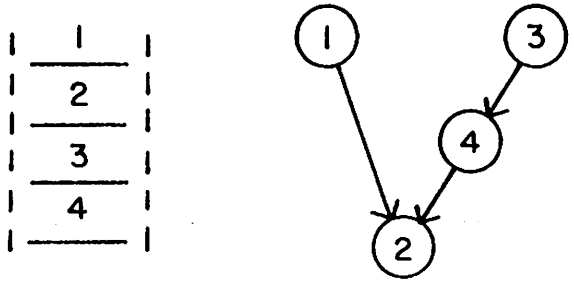


Figure 8. Zone representation and vertical constraint graph where no doglegging is allowed.

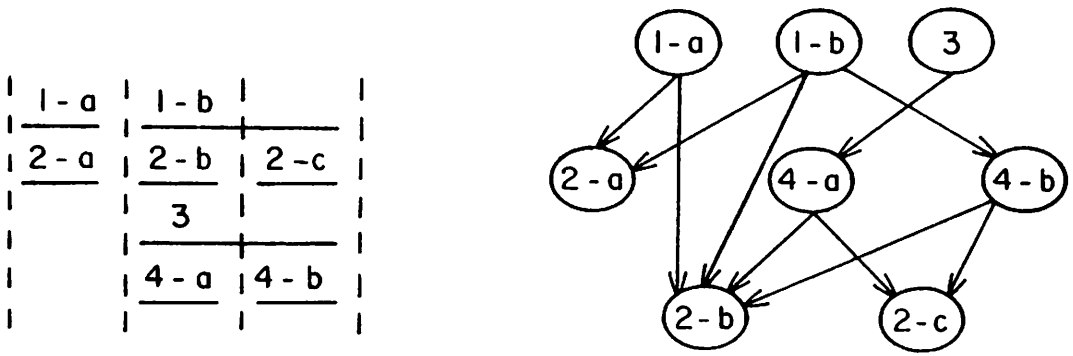
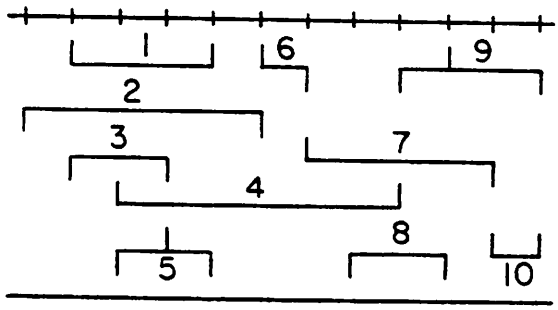
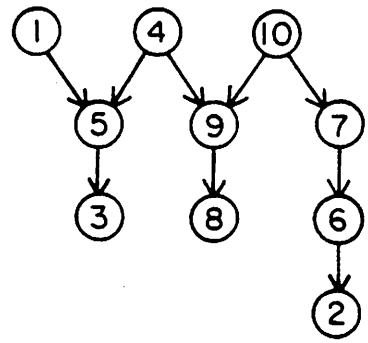


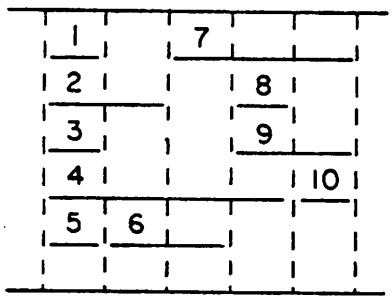
Figure 9. Zone representation and vertical constraint graph where dogleg is allowed.



(a) net list



(b) vertical constraint graph



(c) zone representation

Figure 10. Example of Figure 1.

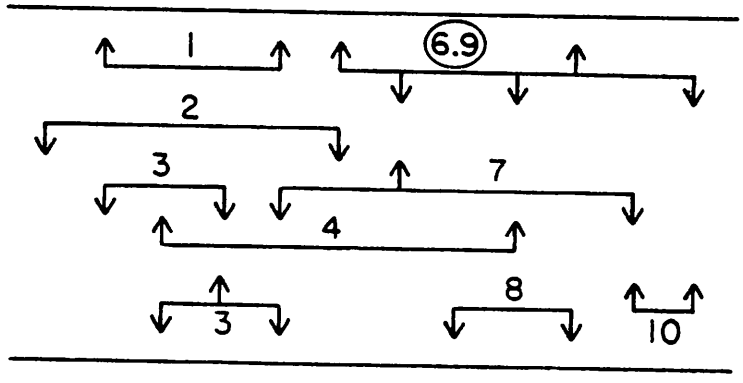
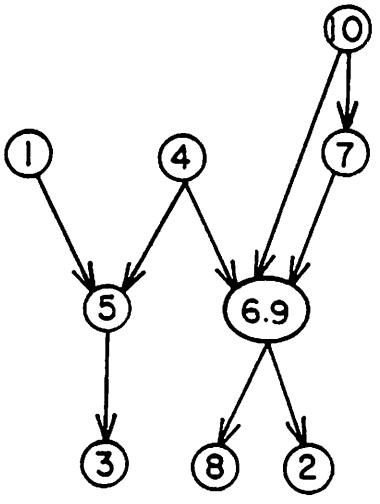
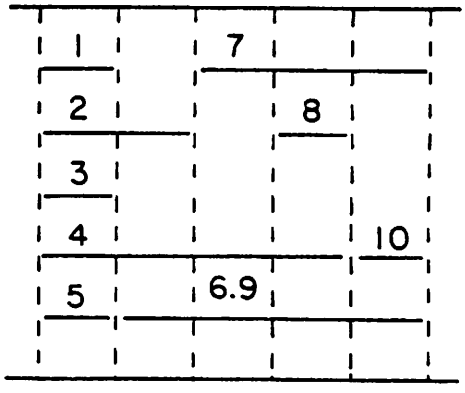


Figure 11. Netlist after merging of net 6 and net 9 to form net 6'9.



(a) updated vertical constraint graph



(b) updated zone representation

Figure 12. Merging of nets.

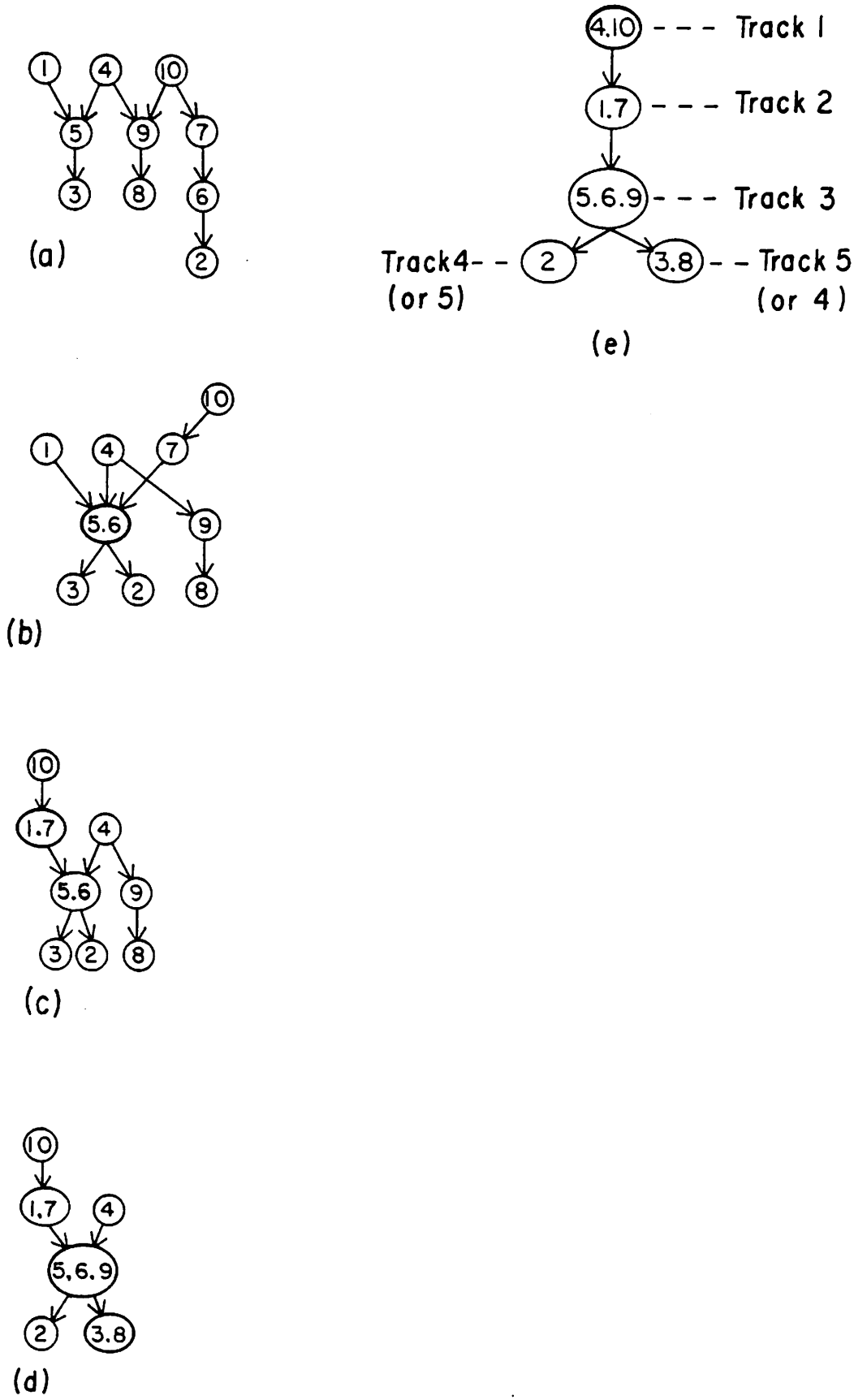


Figure 13. Illustration of Algorithm 1.

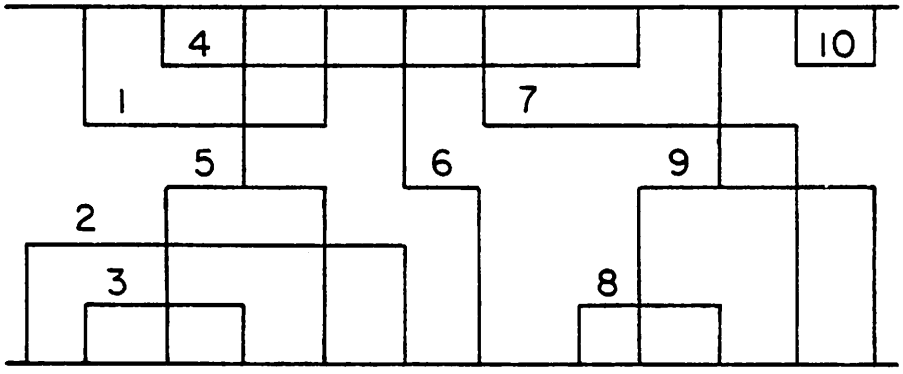


Figure 14. A solution corresponding to Figure 13(e).

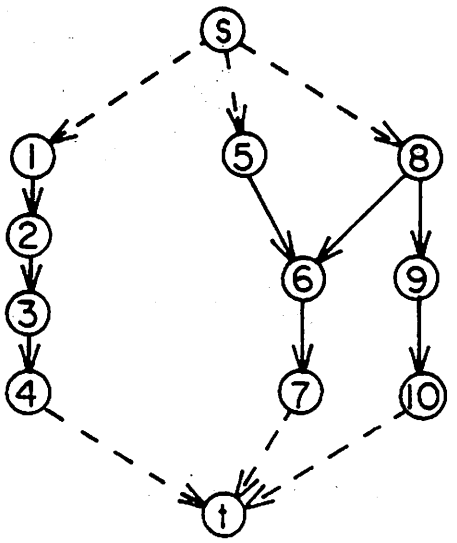


Figure 15. A modified vertical constraint graph.

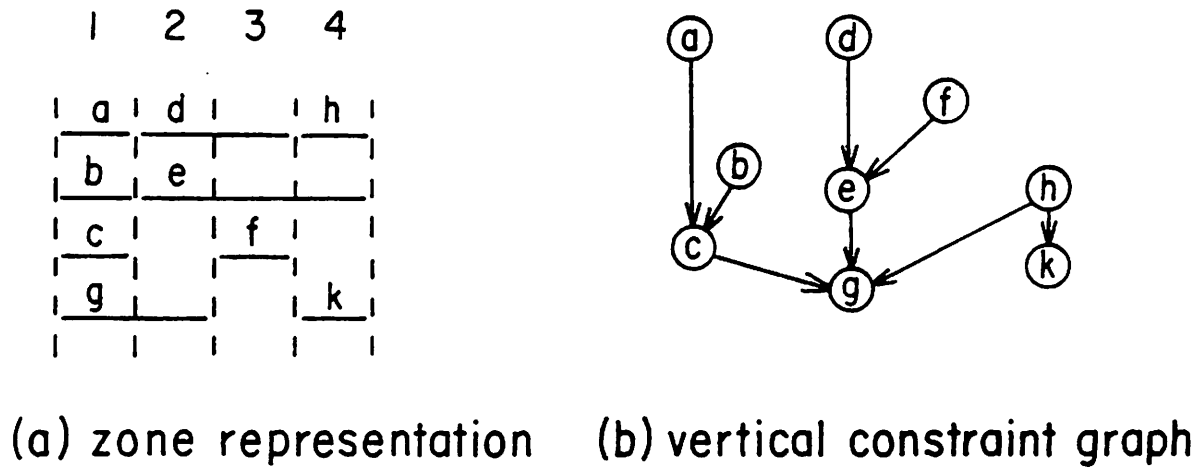


Figure 16. An example to show possible difficulties of Algorithm 1.

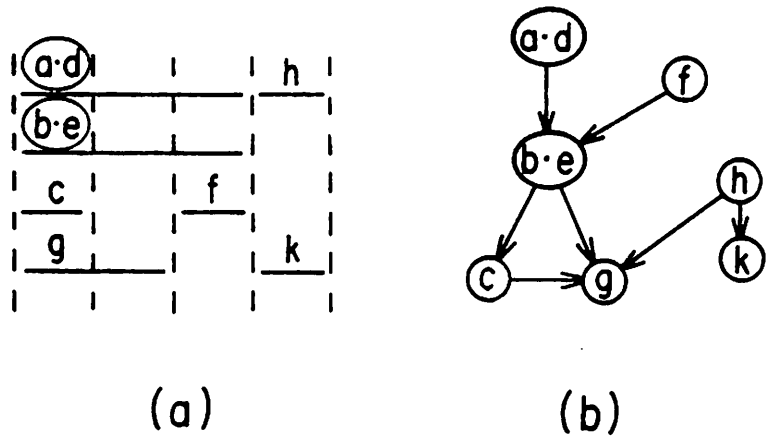


Figure 17. Updated zone representation and vertical constraint graph.

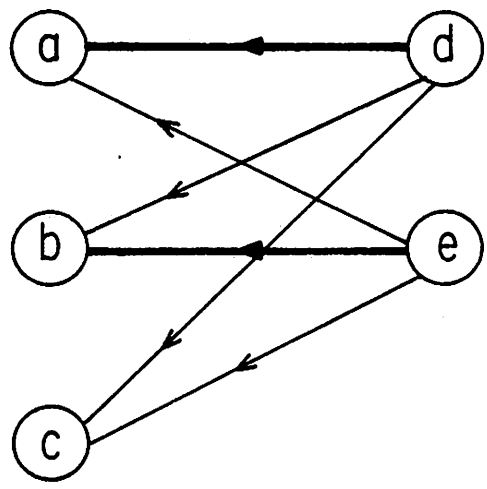


Figure 18. Graph G_h and a possible matching in processing zone 2.

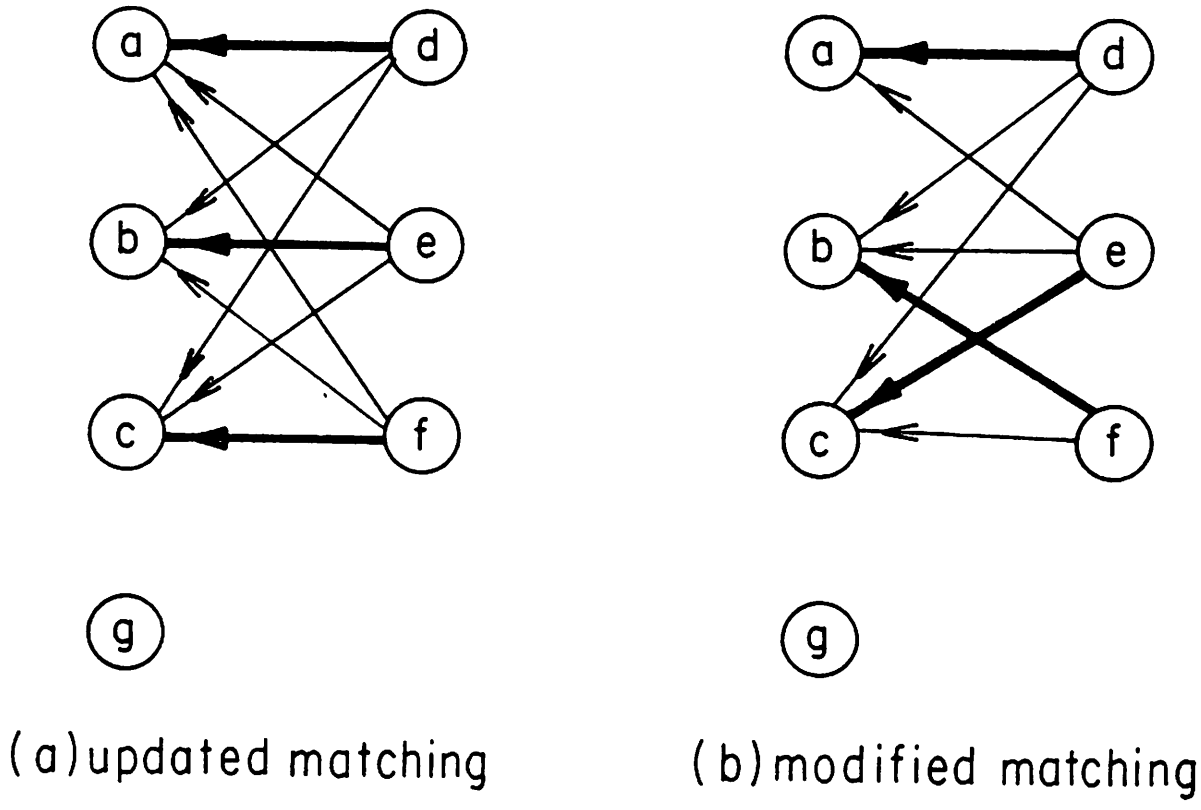


Figure 19. Updating graph G_h in processing zone 3.

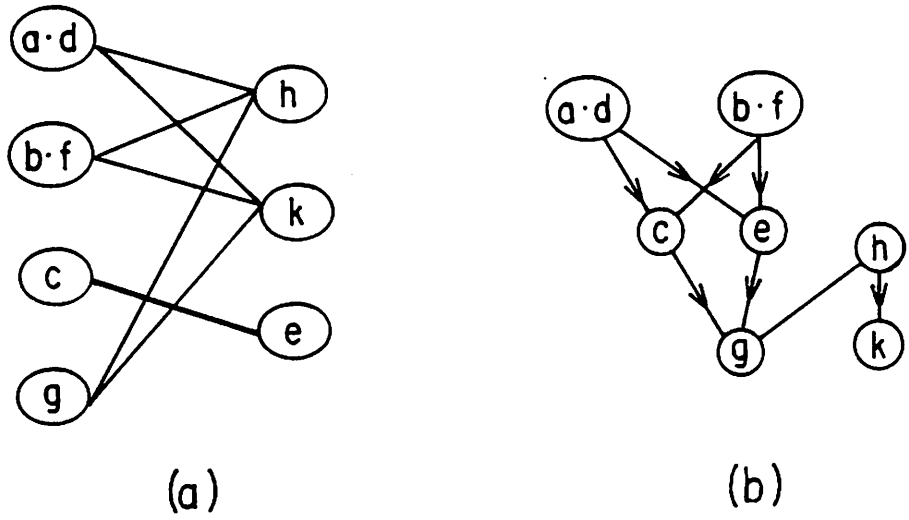


Figure 20. Updated graph G_h and G_r for the processing of zone 4.

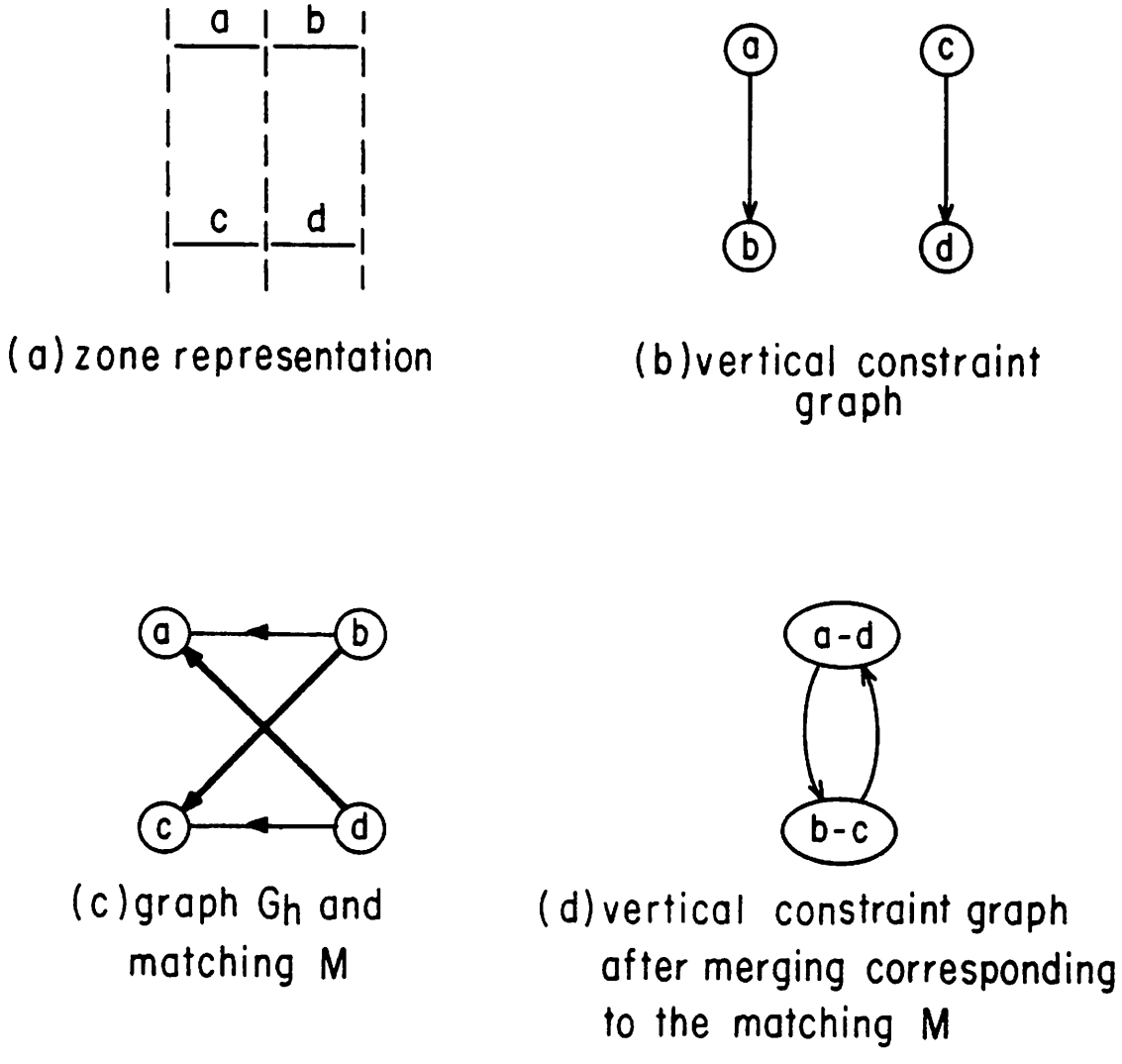


Figure 21. Example to illustrate a matching which violates the vertical constraint.

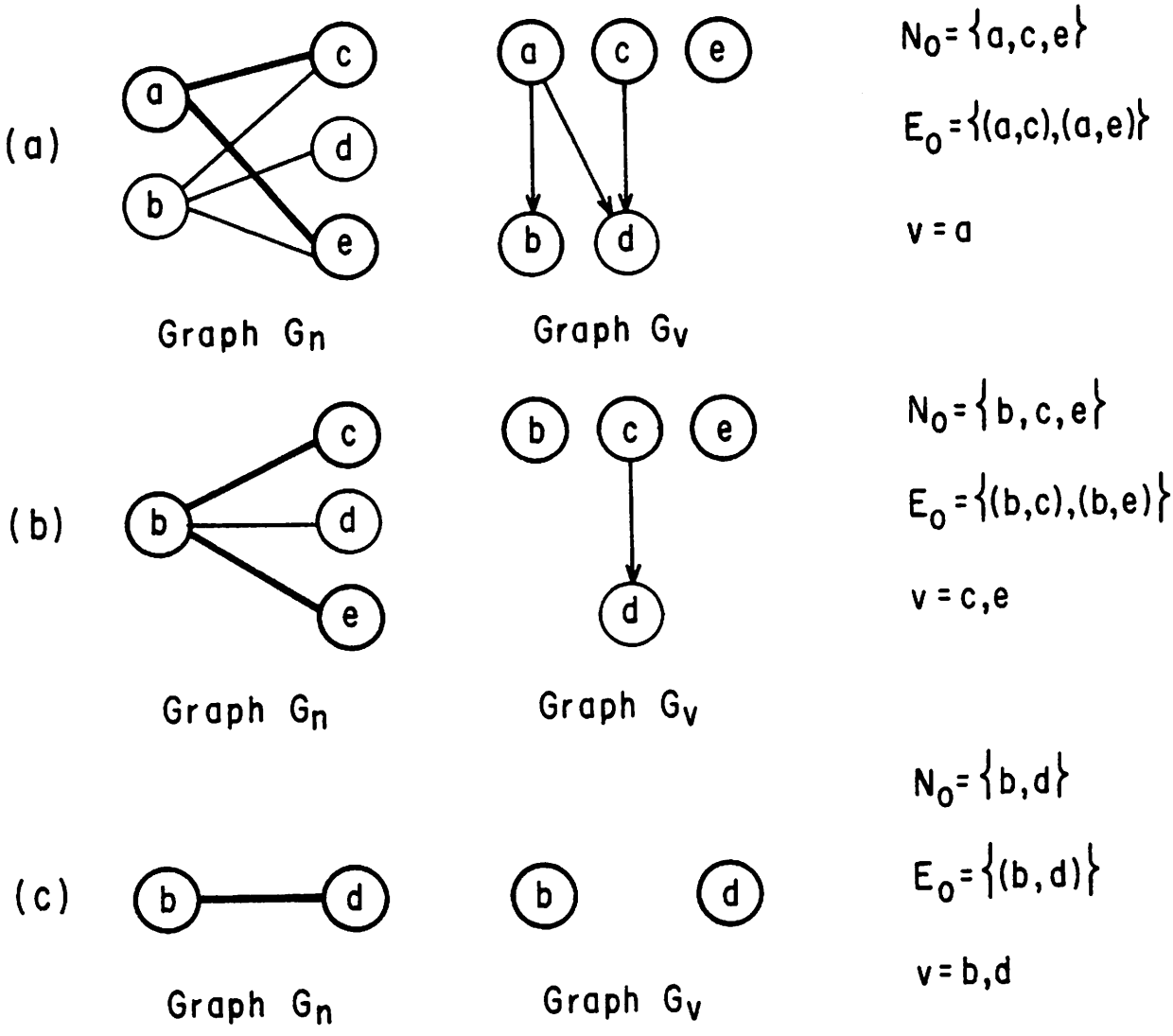


Figure 22. Illustration of Algorithm A.

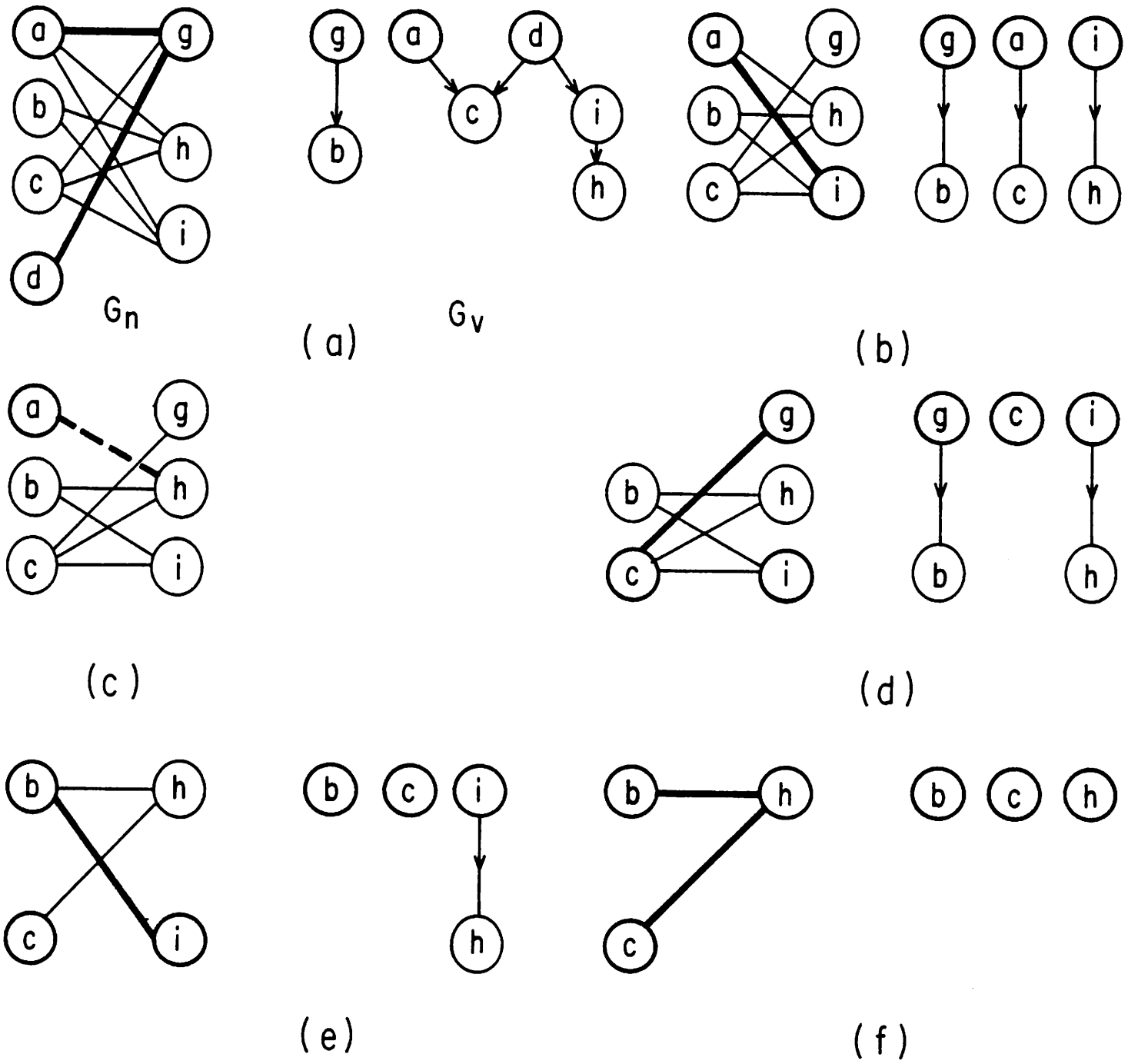


Figure 23. Example to illustrate the use of the corollary.

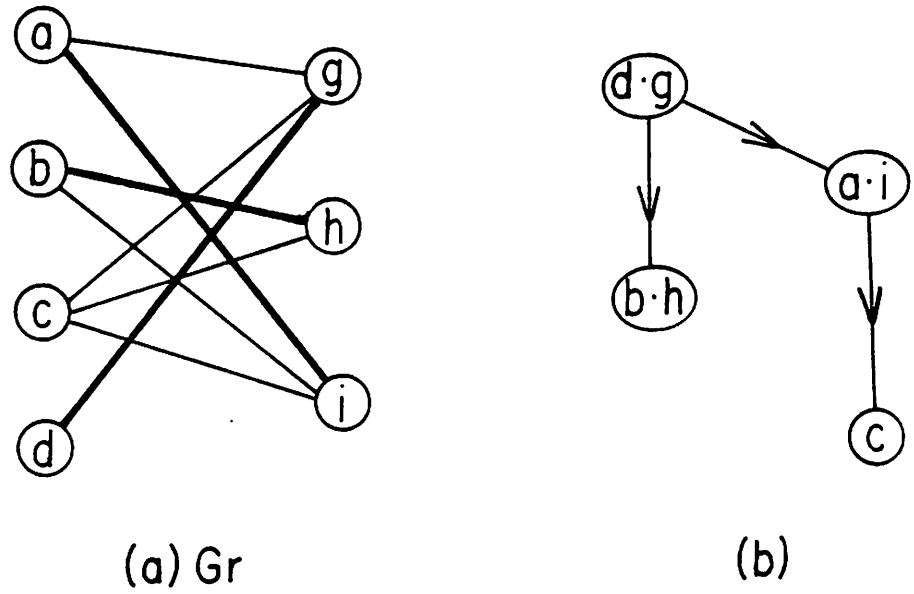
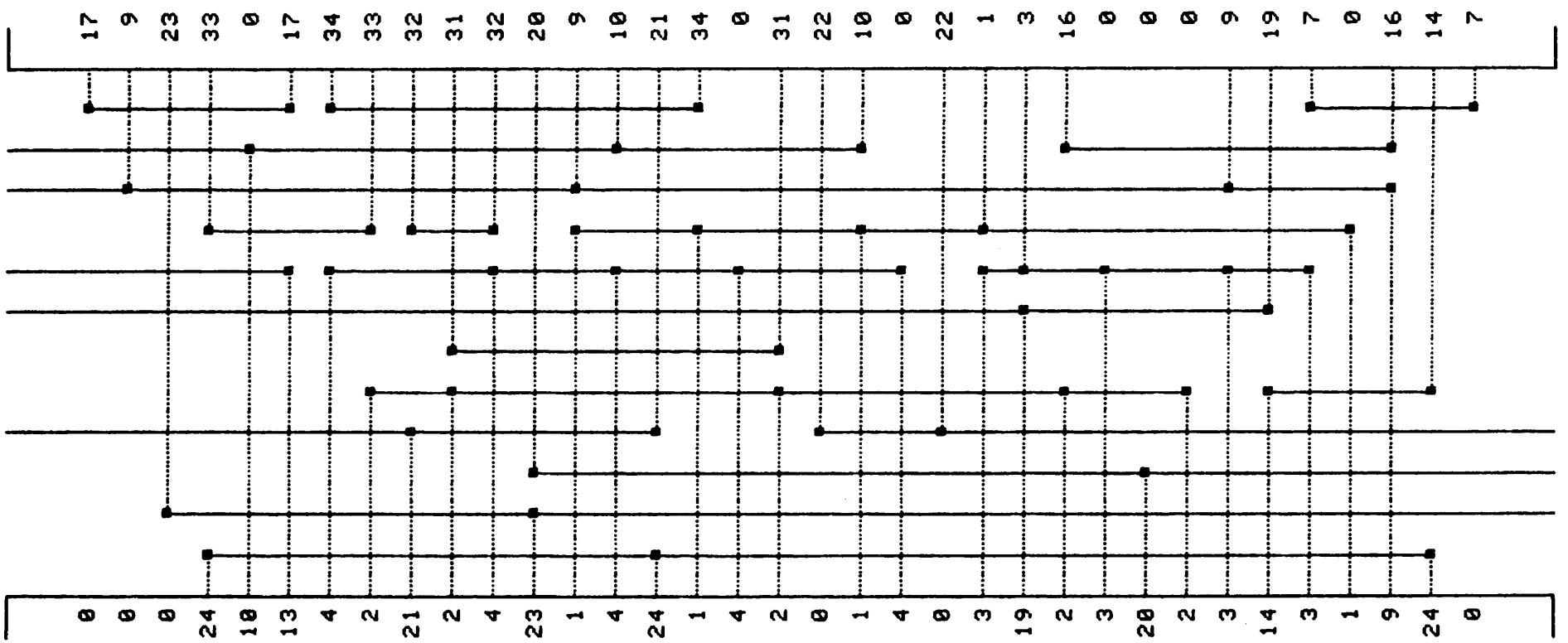


Figure 24. An arbitrary merging in G_r and the corresponding vertical constraint graph after merging.

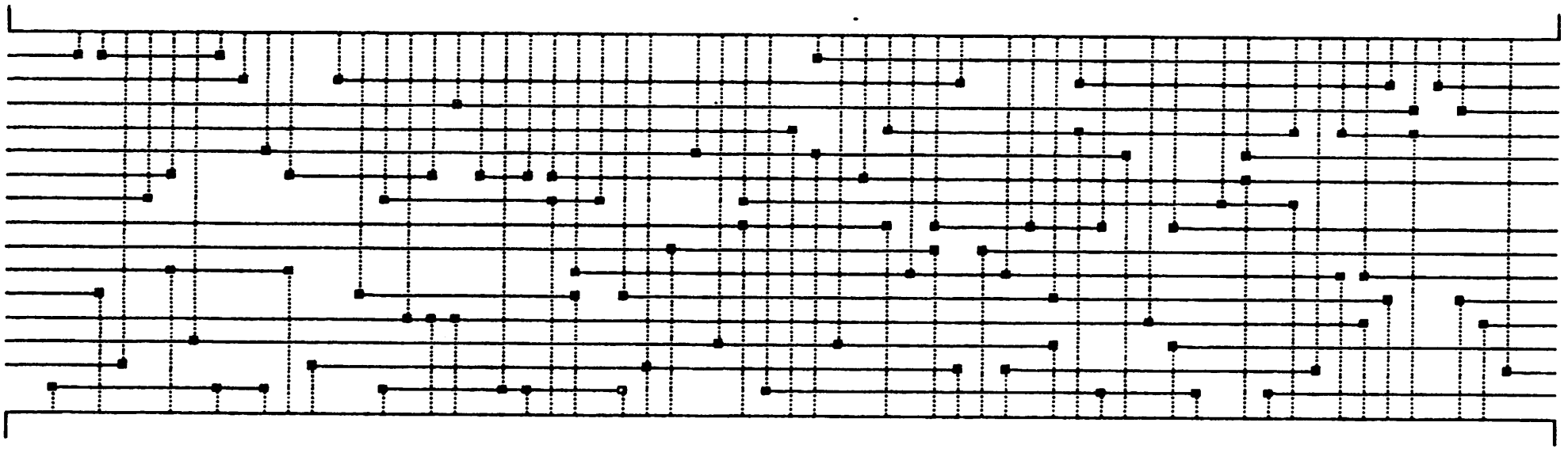
****** example.1 ******



number of tracks = 12
maximum density = 12

Figure 25. Example 1.

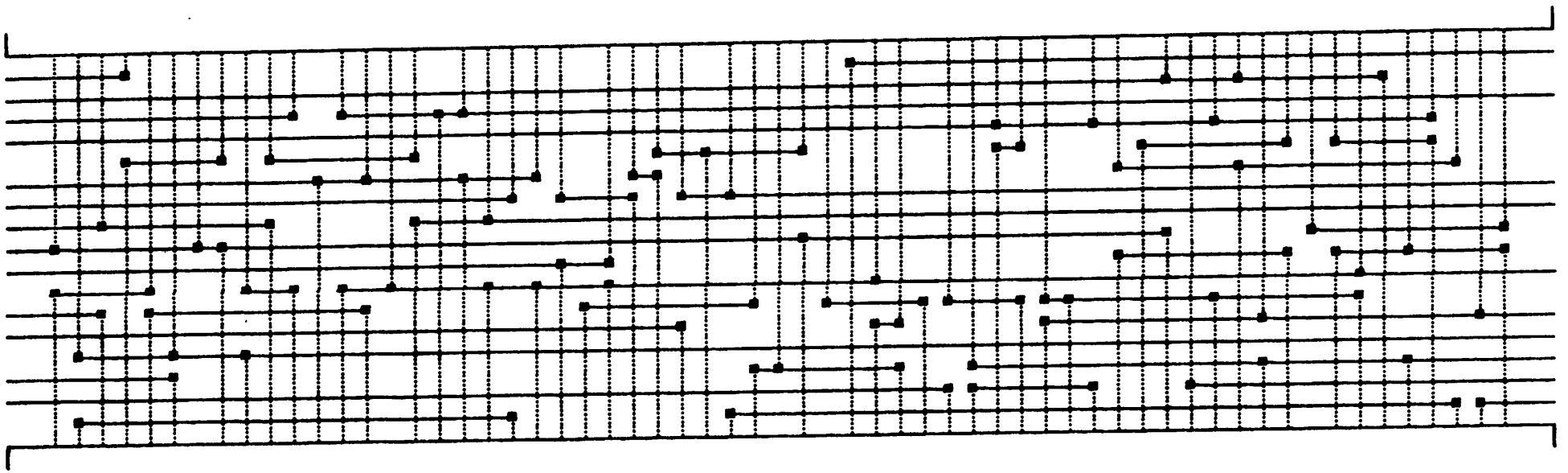
****** example.3a ******



number of tracks = 15
maximum density = 15

Figure 26. Example 3a.

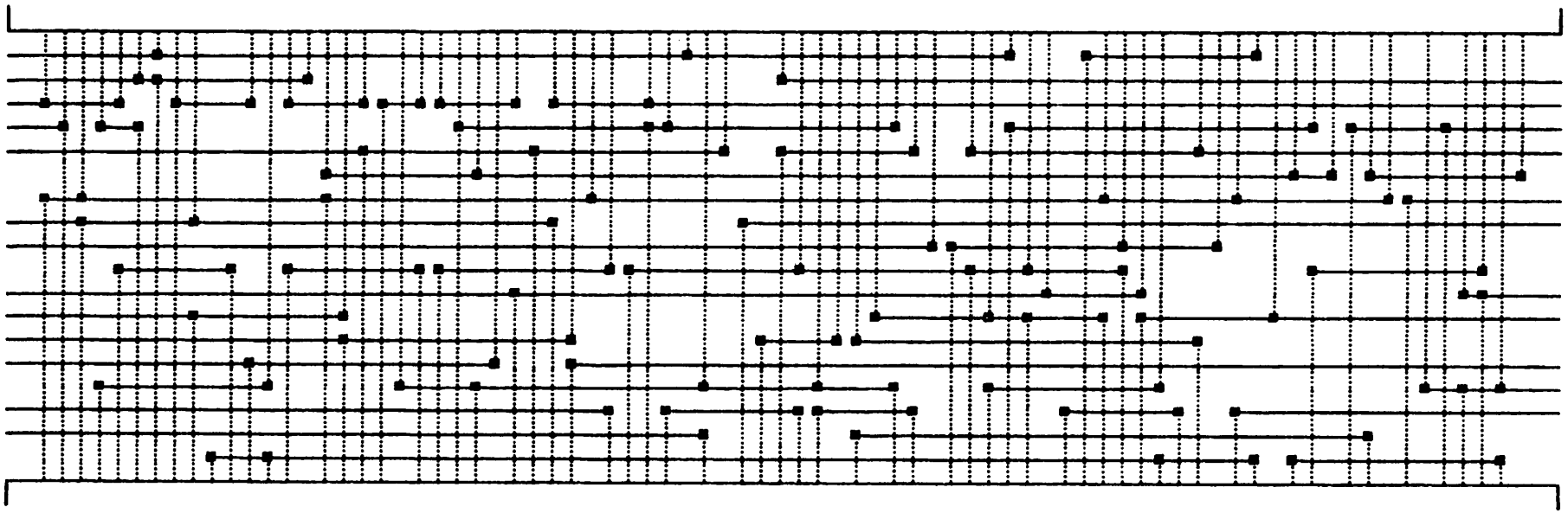
****** ex.3b ******



number of tracks = 17
maximum density = 17

Figure 27. Example 3b.

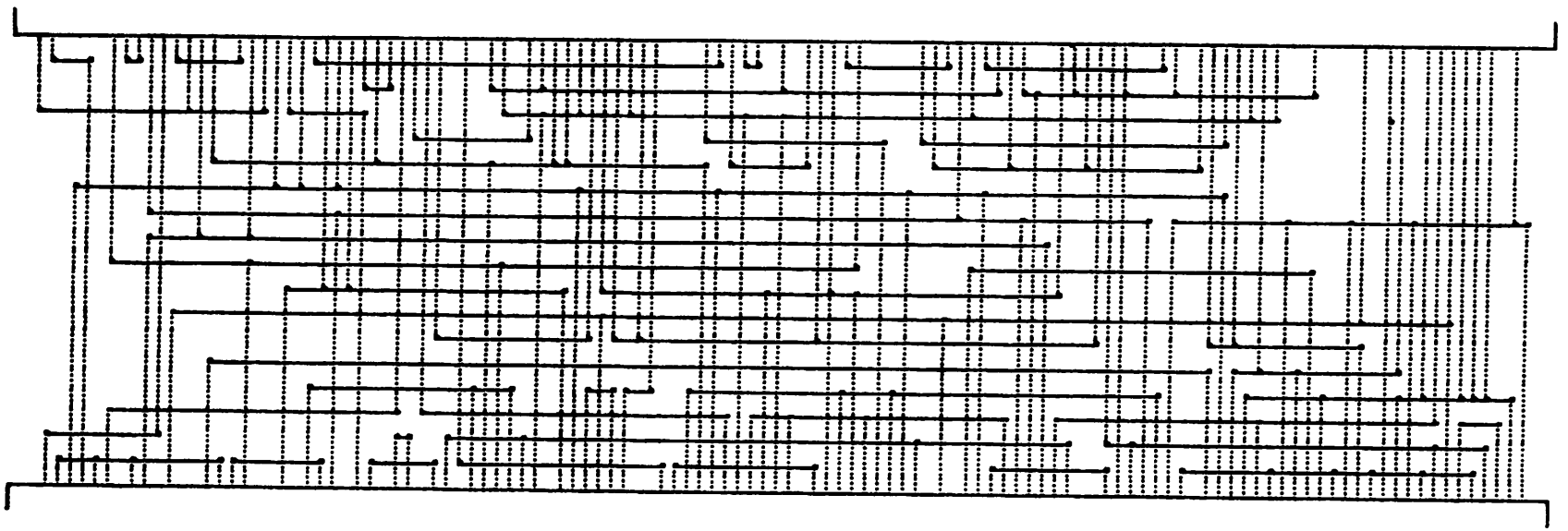
****** example 3c ******



number of tracks = 18
maximum density = 18

Figure 28. Example 3c.

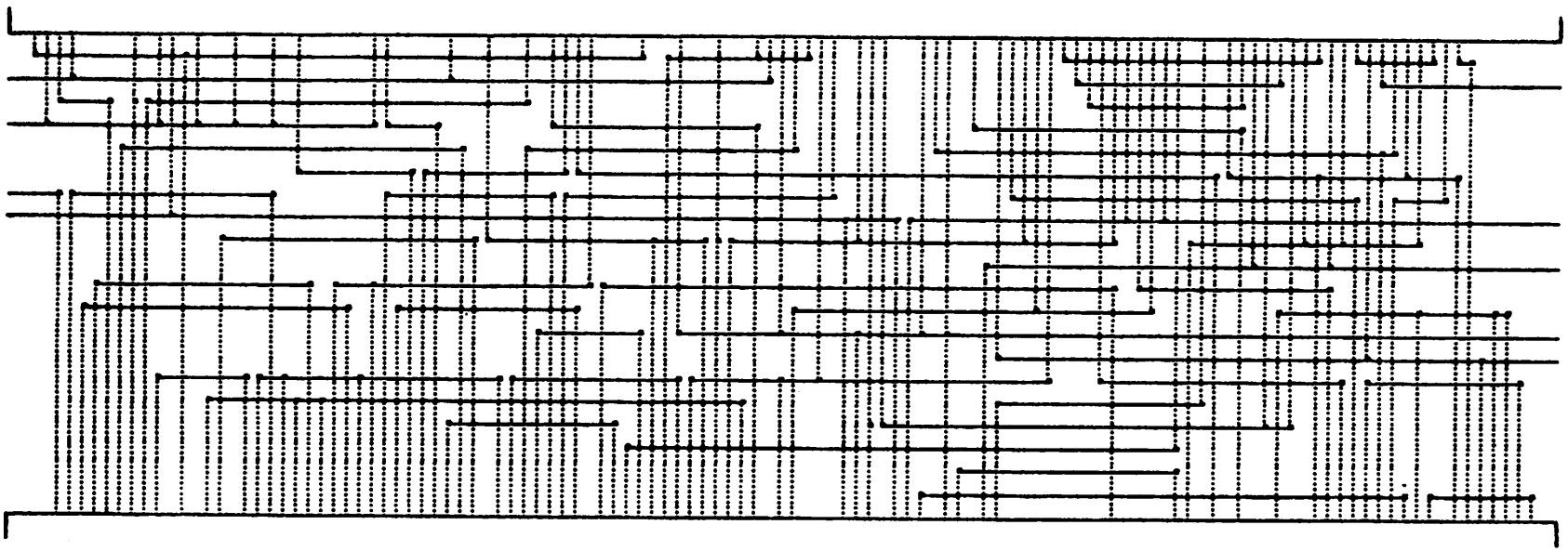
****** example 4b ******



number of tracks = 17
maximum density = 17

Figure 29. Example 4b.

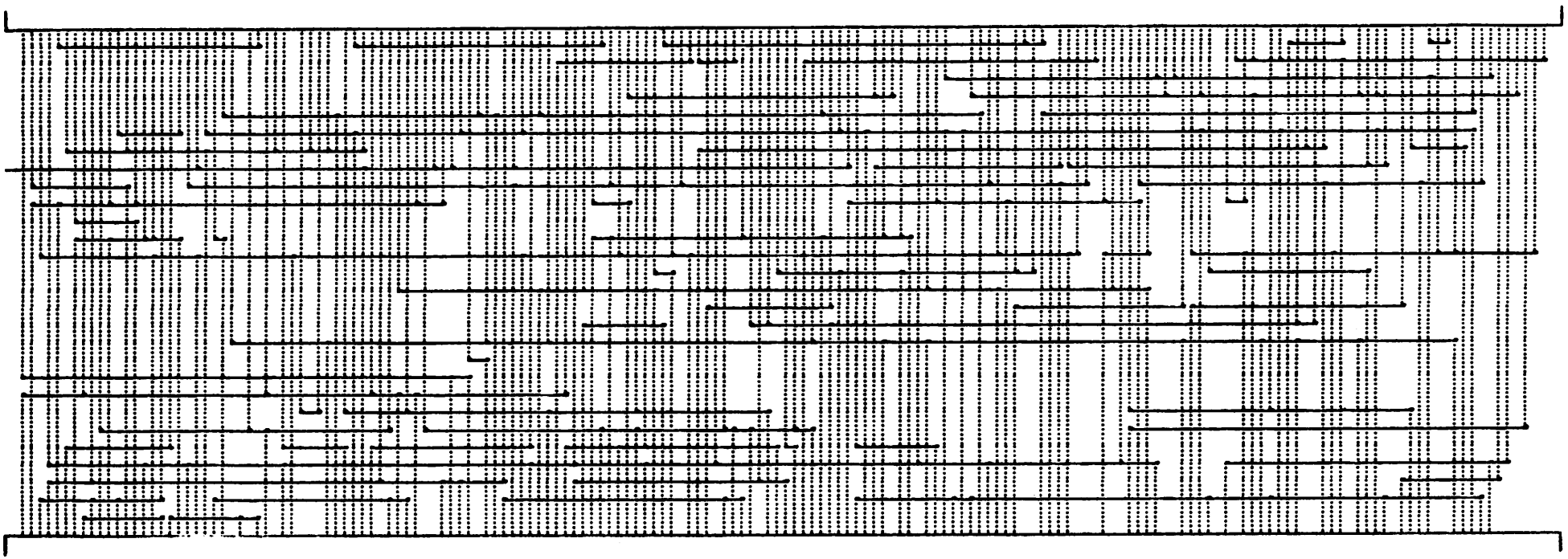
****** example 5 ******



number of tracks = 20
maximum density = 20

Figure 30. Example 5.

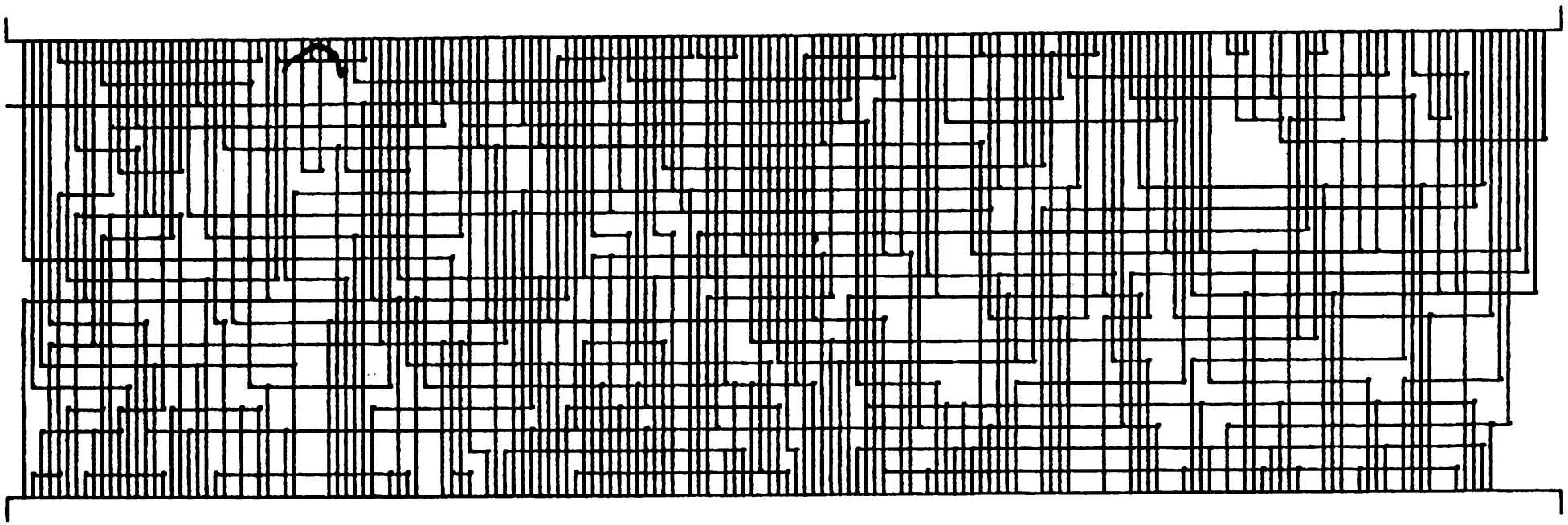
****** difficult example *****



number of tracks = 28
maximum density = 19

Figure 31. Difficult Example without dogleg

****** difficult example *****



number of tracks = 20
maximum density = 19

Figure 32. Difficult Example with dogleg.