Efficient Algorithms for Channel Routing

TAKESHI YOSHIMURA AND ERNEST S. KUH, FELLOW, IEEE

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 and 2.1 s, respectively.

I. 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-percent 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. A channel routing algorithm called "left edge" was first proposed by Hashimoto and Stevens [1]. A modified version of the "left edge" algorithm has been implemented in the Bell Labs' Polycell Layout System, LTX [2]-[5]. In this paper we propose two new algorithms based on graph theoretical considerations. We tested our methods using the same examples provided by previous authors. In all cases our algorithms yield better results.

II. Description of the Problem

Consider a rectangular channel with two rows of terminals along its top and bottom sides. A number between 0 and N is assigned to each terminal. Terminals with the same number i ($1 \leqslant i \leqslant N$) must be connected by net i, while those with number 0 designate unconnected terminals.

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 on the upper or lower sides of the channel. Fig. 2 shows a solution of this example.

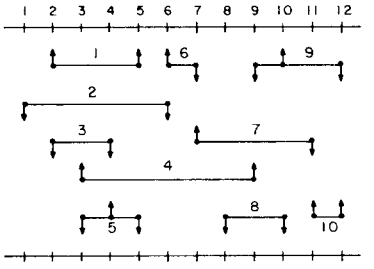


Fig. 1. Netlist representation for routing requirement.

0 1 4 5 1 6 7 0 4 9 10 10
2 3 5 3 5 2 6 8 9 8 7 9

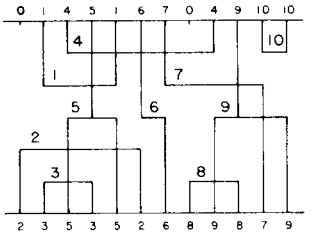


Fig. 2. A realization for the requirement in Fig. 1.

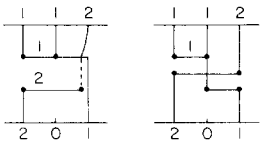


Fig. 3. An example with cyclic conflict.

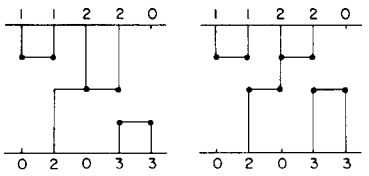
Consider the example in Fig. 3(a). Because the vertical segment of net 1 cannot overlap that of net 2 (shown dotted line) on the same vertical track, a constraint relation has been introduced on the horizontal segments of net 1 and net 2. If we pay attention to the leftmost column, the horizontal segment of net 1 must be placed above that of net 2. By the same reasoning, net 2 must be placed above net 1 if the rightmost column is considered. In other words, this routing specification cannot be realized without splitting some net into more than one horizontal segments as shown in Fig. 3(b). However, this kind of conflicting situation can often be avoided by rearranging the placement. In this paper, we assume that routing

Manuscript received November 13, 1980; revised July 20, 1981. This work was supported by the National Science Foundation under Grant ENG-78-24425 and the Alexander von Humboldt Foundation.

T. Yoshimura is with the Department of Electrical Engineering and Computer Sciences, and the Electronics Research Laboratory, University of California, Berkeley, CA 94720, on leave from Nippon Electric Company, Ltd., Japan.

E. S. Kuh is with the Department of Electrical Engineering and Computer Sciences, and the Electronics Research Laboratory, University of California, Berkeley, CA 94720.

0028-0070/82/0100-0025$00.75



(a) (b)

Fig. 4. An example illustrating the advantage of using dogleg. (a) No dogleg. (b) Dogleg.

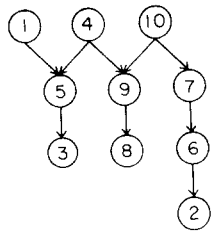
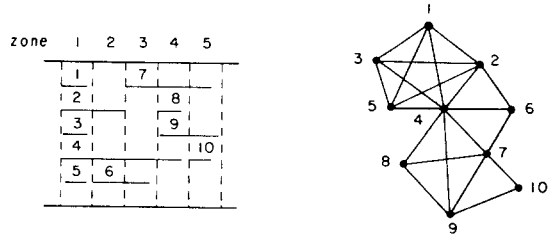


Fig. 5. Vertical constraint graph G_v for the netlist in Fig. 1.



(a) (b)

Fig. 6. Zone representation and interval graph. Maximal cliques are 12345, 246, 467, 4798, and 7910.

specification is always realizable in the sense that there exists no cyclic conflict in the net list.

The major objective of the problem is to minimize the number of horizontal tracks used to realize the routing requirement.

Consider the example in Fig. 4(a) where an optimal solution is given if splitting of horizontal segments is not allowed. However, the same example can be realized with only 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, but also used to minimize the number of horizontal tracks. The latter is perhaps 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.

III. Definitions

A. 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 at a given column must be placed above the horizontal segment of another net connected to the lower terminal at 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. (For example, a 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 acyclic the routing specification is always realizable.

B. 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.

C. 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 whose 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.

TABLE I

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	

Therefore, we assign zones the sequential number to the columns at which $S(i)$ are maximal. These columns define zone 1, zone 2, etc., as shown in Table I, 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 $S(3)$. The zone representation for this example is shown in Fig. 6(a).

Zones can be defined more clearly by using an interval graph defined by the horizontal segments of nets. A graph $G(\bar{V}, E)$ is an interval graph corresponding to a set of nets, where a node $v_i \in \bar{V}$ represents net n_i and an edge $(v_i, v_j) \in E$ iff $n_i \cap n_j \neq 0$ (i.e., net n_i and net n_j have horizontal overlap) [6], [7]. 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.

It should be emphasized that the channel routing problem is completely characterized 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

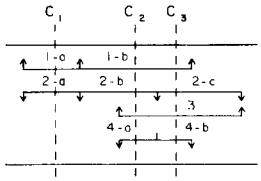


Fig. 7. A four-net example illustrating subnets and cuts corresponding to maximal clique C_1, C_2, and C_3.

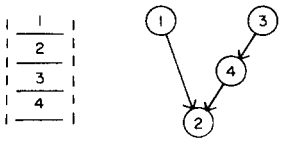


Fig. 8. Zone representation and vertical constraint graph where no doglegging is allowed.

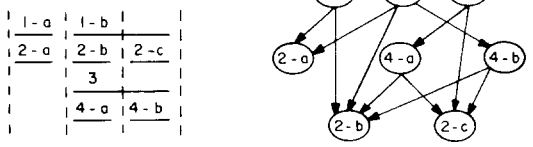


Fig. 9. Zone representation and vertical constraint graph where dogleg is allowed.

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 of the number of tracks is of course the maximum density.

D. Dogleg

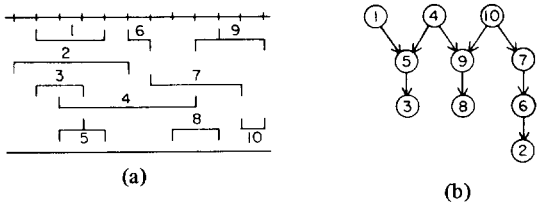
Up to now the vertical constraint graph and the zone representation are defined for the problem where the number of horizontal segments per net is limited to one, i.e., the realization does not allow doglegs. However, these two representations can be extended to the dogleg problem, where the horizontal 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 maximal 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 $1-a, 1-b, \cdots, 4-b$ are considered instead of nets 1, 2, 3, and 4, and the vertical constraint graph and the zone representation are shown in Fig. 9.

IV. A Simple Method Based on Merging of Nets

To find an optimum realization of the channel routing problem is very difficult. The problem has been shown to be NP complete [8]. Therefore, we propose heuristic algorithms which generate optimum or near optimum solutions with a reasonable amount of computation time.

A. "Merging of Nets"

Before describing the algorithm, we will define the following operation.



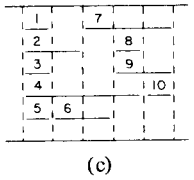


Fig. 10. Examples of Fig. 1. (a) Netlist. (b) Vertical constraint graph. (c) Zone representation.

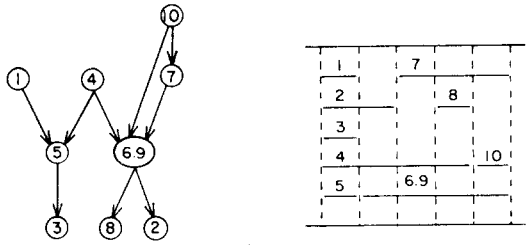


Fig. 11. Merging of nets. (a) Updated vertical constraint graph. (b) Updated zone representation.

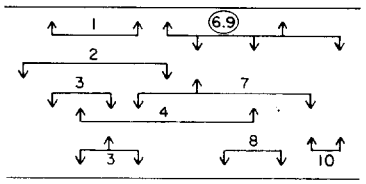


Fig. 12. Netlist after merging of net 6 and net 9 to form net $6 \cdot 9$.

Definition: Let i and j be the nets for which

a) there exists no horizontal overlap in the zone representation, and
b) there is no directed path between node i and node j in the vertical constraint graph

(i.e., net i and net j can be placed on the same horizontal track).

Then the operation "merging of net i and net j"

a) modifies the vertical constraint graph by shrinking node i and node j into node $i \cdot j$, and
b) updates the zone representation by replacing net i and net j by net $i \cdot j$ which occupies the consecutive zones including those of net i and net j.

Example: In the example of Fig. 10, net 6 and net 9 are candidates for merging. The operation "merging of net 6 and net 9" modifies the vertical constraint graph and the zone representation, as shown in Fig. 11. The updated vertical constraint graph and the zone representation correspond to the net list in Fig. 12, 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.

The following lemma is obvious.

Lemma: If the original vertical constraint graph contains no cycle, the updated vertical constraint graph does not have cycles either.

B. Algorithm Description

Since it is assumed that there is no cycle in the (initial) vertical constraint graph, we can repeat the operation "merging of nets" without generating any cycle in the graph. The following algorithm merges nets systematically according to the zone representation.

Algorithm #1:

```
        procedure Algorithm #1 (z_s, z_t)
        begin;
a1:     L = { };
a2:     for z = z_s to z_t 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 length in the vertical
            constraint graph;
a6:         L = L - {n_1, n_2, ···}, where n_j is a net
            merged at step a5;
        end;
    end;
```

If there is a path $n_1 - n_2 - n_3 - \cdots, -n_k$ in the vertical constraint graph, then no two nets among $n_1, n_2, \cdots, n_k$ 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 that the longest path length after merging is minimized 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, $\cdots$, and at the fourth iteration net 10 and net 4 are merged. Finally, we have the graph in Fig. 13(e). Then we assign the horizontal tracks to each node of the graph. 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).

C. Minimizing the Longest Path

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) $P = \{n1, n2, \cdots, np\}$ and $Q = \{m1, m2, \cdots, mq\}$ $(p \geq q)$ 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

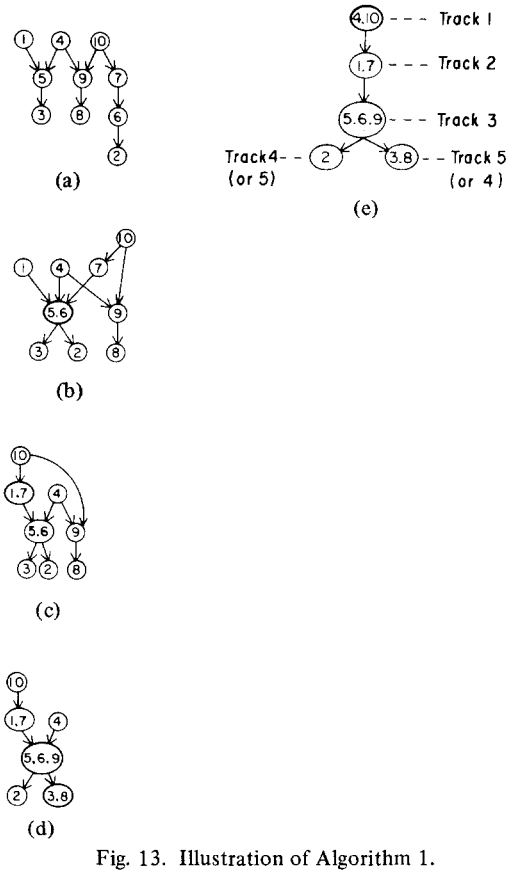


Fig. 13. Illustration of Algorithm 1.

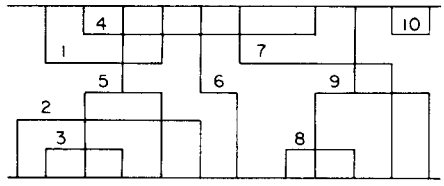


Fig. 14. A solution corresponding to Fig. 13(e).

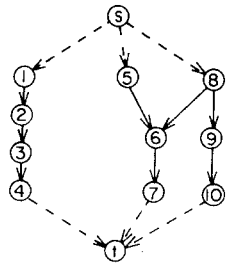


Fig. 15. A modified vertical constraint graph.

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

Example:

$$Q = \{6, 7\}$$

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

$$u(1) = 1, u(3) = 3, u(6) = 2, \cdots$$

$$d(1) = 4, d(3) = 2, d(6) = 2, \cdots.$$

The purpose here is to minimize the length of the longest path after merger. However, it will be too time consuming to find an exact minimum merger, hence a heuristic merging algorithm will be given. 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. If there are two or more nodes which will result in a minimum increase we choose n such that $u(n) + d(n)$ is maximum or nearly maximum and that the condition $u(m)/d(m) = u(n)/d(n)$ is satisfied or nearly satisfied. These can be implemented by introducing the following:

(1) for $m \in Q$

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

(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.

Merging Algorithm

```
given P, Q;
    begin;
a1:     while Q is not empty do;
        begin;
a2:         among Q, find m* which maximizes f(m);
a3:         among P, find n* which minimizes g(n, m*),
            and which is neither ancestor nor descendent
            of m*;
a4:         merge n* and m*;
a5:         remove n* and m* from P and Q,
            respectively;
        end;
    end;
```

For the example given above, let us pick $C_\infty = 100$, we obtain Table II(a). Since $f(7) > f(6)$, node 7 is chosen first from Q. Next, we evaluate $g(n, 7)$ using $C_\infty = 100$ and obtain Table II(b). Since $g(4, 7)$ is the smallest, we merge node 4 with node 7.

V. An Improved Algorithm Based on Matching

A. Overview of the Algorithm

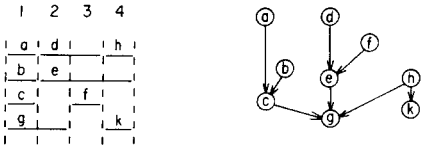
In Algorithm #1, nets are merged when they are processed, and it should be noted that a merging may block subsequent mergings. Fig. 16 shows an example. Let us assume that, at zone 1, Algorithm #1 merges net a and net d, net b and net e, respectively, (if we follow the Merging Algorithm of the last section, these mergings will not occur, but they are assumed for illustration only). 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.

TABLE II(a)

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

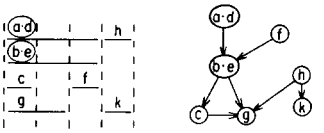
TABLE II(b)

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



(a) (b)

Fig. 16. An example to show possible difficulties of Algorithm 1. (a) Zone representation. (b) Vertical constraint graph.



(a) (b)

Fig. 17. Updated zone representation and vertical constraint graph.

To avoid this type of situation as much as possible, or in order to make the algorithm more flexible, we introduce another algorithm. In Algorithm #2 we construct a bipartite graph G_h, where a node represents a net and an edge between net a and net b signifies 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.

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

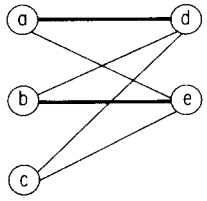


Fig. 18. Graph G_h and a possible matching in processing zone 2.

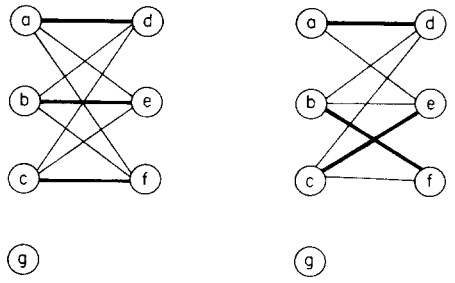


Fig. 19. Updating graph G_h in processing zone 3. (a) Updated matching. (b) Modified matching.

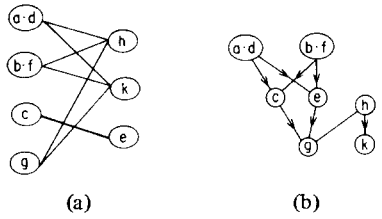


Fig. 20. Updated graph G_h and G_v for the processing of zone 4.

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_h 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.

B. Algorithm #2

The general flow of Algorithm #2 is as follows.

procedure 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;

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

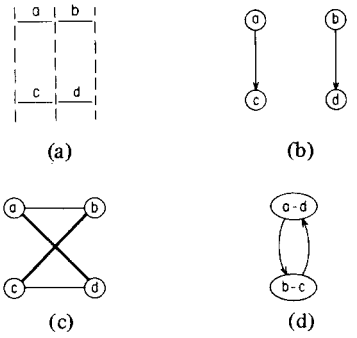


Fig. 21. Example to illustrate a matching which violates the vertical constraint. (a) Zone representation. (b) Vertical constraint graph. (c) Graph G_h and matching M. (d) Vertical constraint graph after merging corresponding to the matching M.

edges between n_r and a node on the left side, if they can be merged; then, find a maximum matching;

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_1 terminating at zone z_{i+1}, merge n_1 with the net n_x specified by the matching on graph G_h. Then put the merged net $n_1 \cdot n_x$ to the left side of G_h;

end;
end;

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.

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. Then, a maximum matching on graph G_h is obtained.

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.

3) Step 3: 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. Hence, it is necessary to 1) check the vertical conflict for the given matching, and 2) modify the matching when the conflict is found. In order to do these two operations, we use the following algorithm which derives the conditions under which any matching in G_h will lead to feasible solutions.

Algorithm A: Given graph $G_h = (N, E_h)$ and graph $G_v = (N, E_v)$;

begin
a1: $E_x = 0$;
while N is not empty do;
begin;

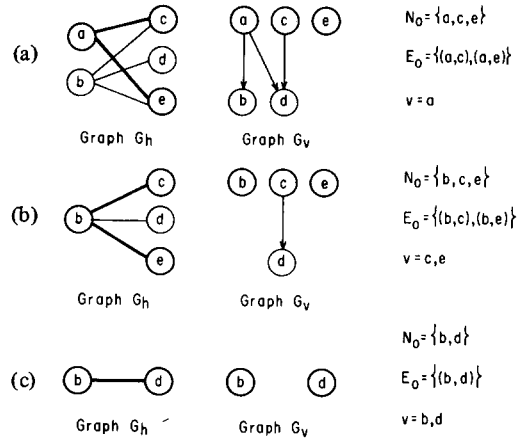


Fig. 22. Illustration of Algorithm A.

a2: let N_0 be the set of nodes which do not have ancestors in G_v;

a3: remove the set of edges E_0 from graph G_h, where $E_0 = \{(i,j) | i, j \in N_0, (i,j) \in E_h\}$;

a4: if there is a node whose degree is equal to zero in graph G_h, then let it be v, go to a6;

a5: otherwise, (a) choose one of the nodes $v \in N_0$, which has the smallest number of edges incident to it and (b) let $E_x = E_x + E_y$, where E_y is the set of edges connecting to node v;

a6: 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, $N_0 = \{a, c, e\}$ 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., $\cdots$.

Theorem: The mergings corresponding to any matching on graph G_h is feasible if and only if E_x is empty at the termination of Algorithm A.

Proof:

if: 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 a3 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, that is, the mergings corresponding to any matching on graph G_h will not create cycles containing nodes of N_0 in graph G_v. Therefore, once these nodes are 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 $\bar{N}_1 \subset N_1$, while the rest belong to $N_0 - N_0^*$. Clearly, nodes in $\bar{N}_1$ 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.

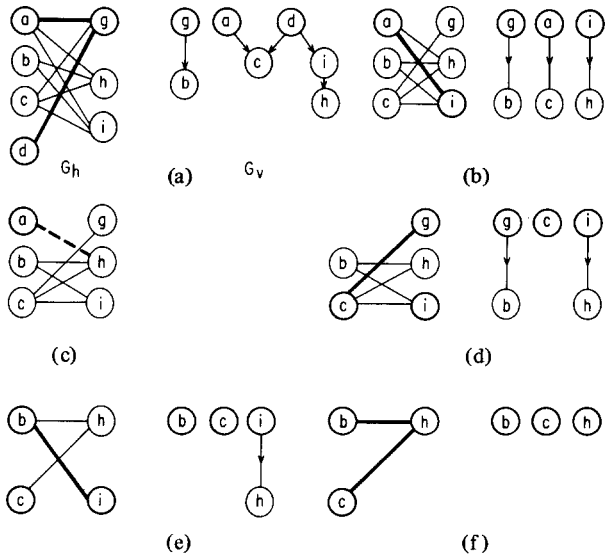


Fig. 23. Example to illustrate the use of the corollary.

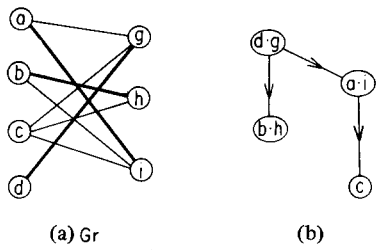


Fig. 24. An arbitrary merging in G_r and the corresponding vertical constraint graph after merging.

only if: 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 $n_0^{(1)} \in N_0$ be a node in N_0. Then there is at least one edge in G_h which connects node $n_0^{(1)}$ and a node, say $n_d^{(1)}$ which is not in N_0. If node $n_d^{(1)}$ is a descendent of node $n_0^{(1)}$, then merger between them is clearly unfeasible. Thus we assume $n_d^{(1)}$ is a descendent of node $n_0^{(2)}$ in $N_0 - \{n_0^{(1)}\}$. In the meantime, node $n_0^{(2)}$ 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 $n_0^{(3)}$ in $N_0 - \{n_0^{(2)}\}$. Since the number of nodes is finite, there must be a "cycle" of nodes in the sequence $n_0^{(1)}, n_d^{(1)}, n_0^{(2)}, n_d^{(2)}, n_0^{(3)}, n_d^{(3)}, \cdots$. Without loss of generality, we can assume that the cycle is $n_0^{(1)}, n_d^{(1)}, n_0^{(2)}, n_d^{(2)}, \cdots, n_0^{(k)}, n_d^{(k)}, n_0^{(1)}$. Consider the merger according to the set of edges $E_s = \{(n_0^{(i)}, n_d^{(i)}) | i = 1, \cdots, k\}$, and let $n_{0d}^{(i)}$ be the node produced by the merger of $n_0^{(i)}$ and $n_d^{(i)}$. Since $n_d^{(i)}$ is a descendent of $n_0^{(i+1)}$, $n_{0d}^{(i)}$ is a descendent of $n_{0d}^{(i+1)}$, for $i = 1, 2, 3, \cdots, k-1$. By the same reasoning, node $n_{0d}^{(k)}$ is a descendent of node $n_{0d}^{(1)}$. Hence, the merger creates a cycle in the merged G_v. This completes the proof of the theorem.

Corollary: 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

TABLE III
NO DOGLEG

Problem data(#nets)	opt	Solution left edge	Alg.1(cpu in sec.)	Alg.2(cpu in sec.)
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)	28*	39	30(0.40)	28 (1.60)

*Obtained in [5] by means of the method of branch and bound after four hours of computation.

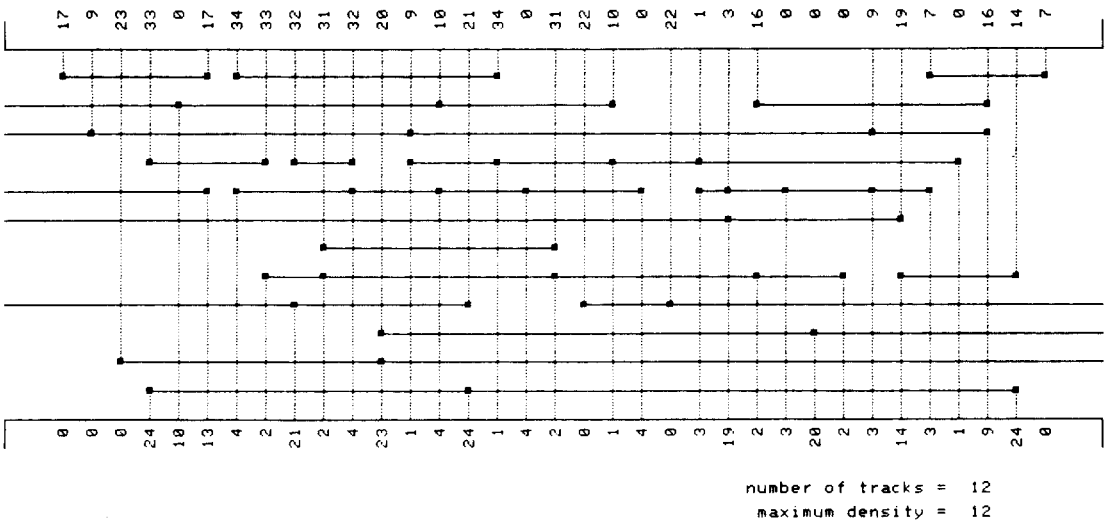


Fig. 25. Example 1.

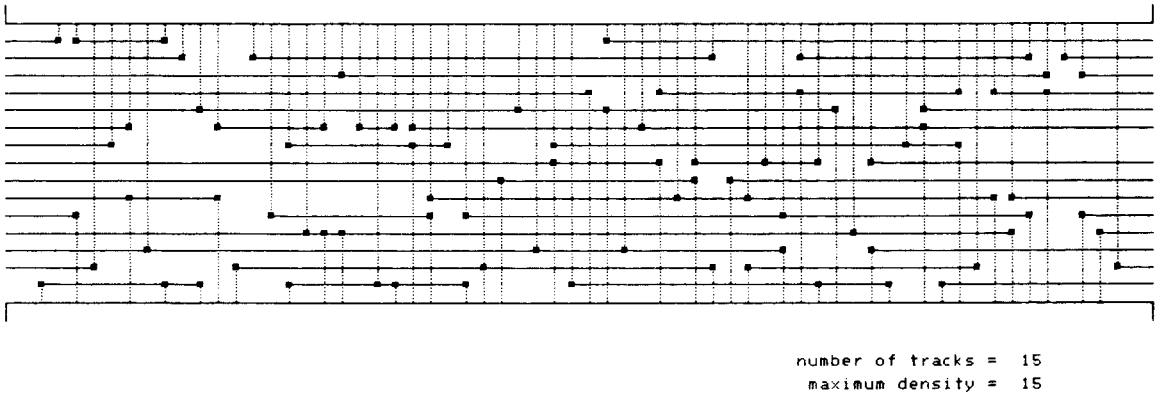


Fig. 26. Example 3a.

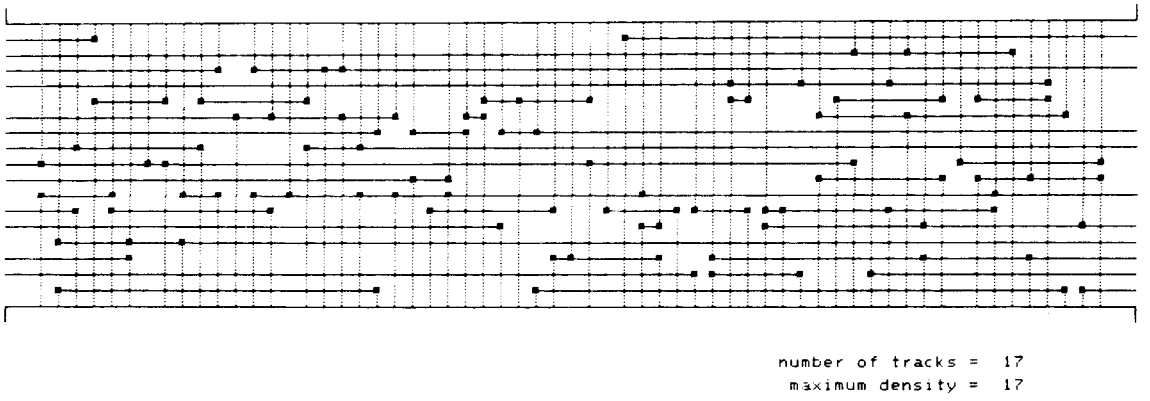


Fig. 27. Example 3b.

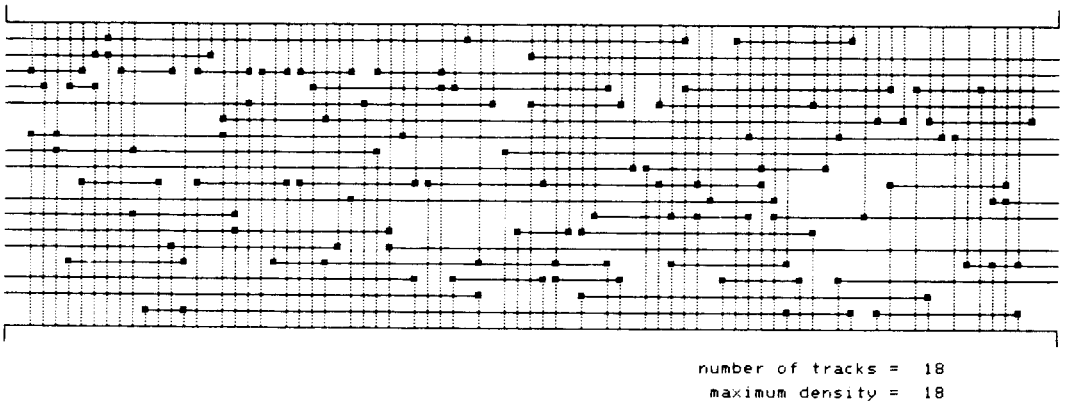


Fig. 28. Example 3c.

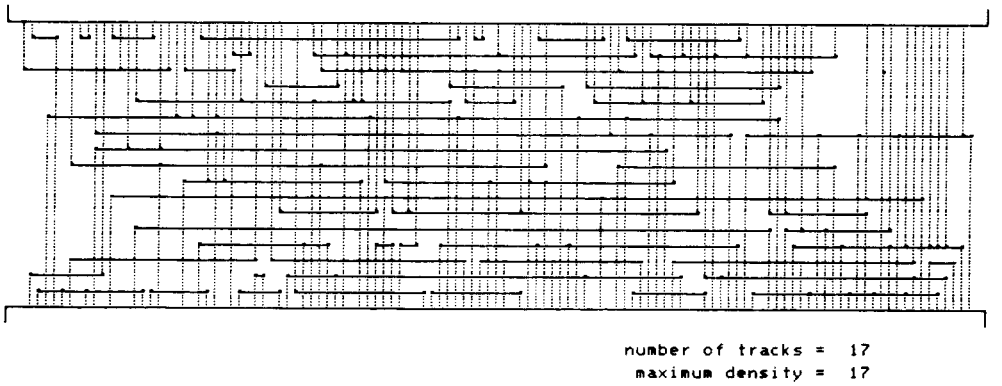


Fig. 29. Example 4b.

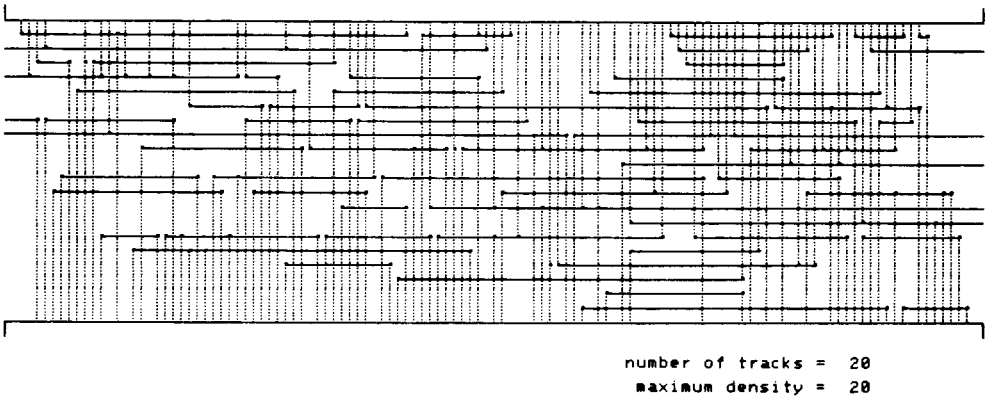


Fig. 30. Example 5.

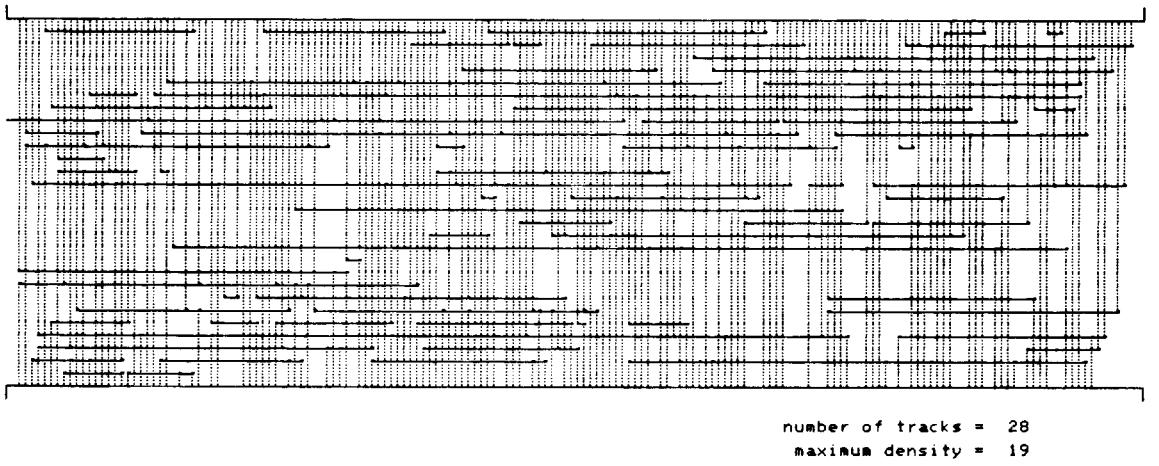


Fig. 31. Difficult example without dogleg.

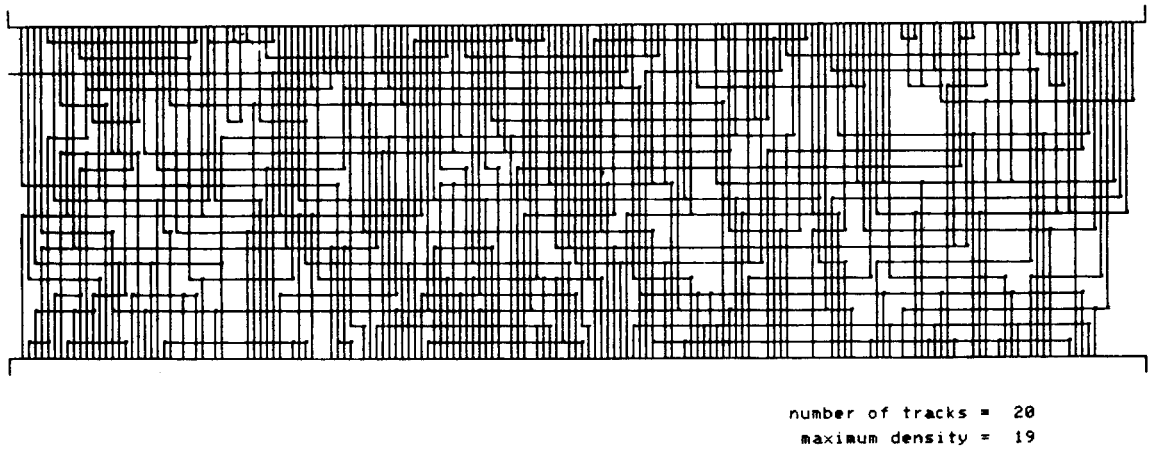


Fig. 32. Difficult example with dogleg.

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) and (a, 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.

In the Step 3 of the Algorithm, we check the vertical conflict by applying the Algorithm A to graph $G(N, M)$, where M is the set of matching edges. It is easy to see that the mergings corresponding to the matching are feasible if and only if E_x is empty. If the mergings are not feasible, then we apply Algorithm A to graph $G(N, E_h)$ and recalculate the maximum matching on graph $G_r = (N, E_h - E_x)$. The new matching is feasible because of the corollary.

VI. 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 III-D. 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 restriction on the subnet merging.

Subnet i and subnet j can be merged only if 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.

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.

TABLE IV
Dogleg

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

VII. Computational Results

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

A. 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.

B. No-Dogleg Problem

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

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

C. Dogleg Problem

Algorithms were applied for the previous examples and, of course, produced optimum solutions for Examples 1-5. However, some solutions have several doglegs. Table IV 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 Algorithms #1 and 2 require only 19 and 18 doglegs, respectively, while the solution of the LTX router of Bell Laboratories [2] has more than 50. According to the paper [4], the LTX router requires nearly 5 s to solve the problem of this size on HP-2100 minicomputer. Fig. 32 shows the computer output of Algorithm #2.

VIII. 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 previously published paper, and better solutions than LTX for the "difficult example." As for the dogleg problem, we used the "difficult problem" to make comparison. Algorithm #1 produced a solution 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.

Acknowledgment

The authors are grateful to Dr. S. Tsukiyama of Osaka University for his useful comments. The second author also wishes to acknowledge the support of Prof. Dr.-Ing. R. Saal at the Technical University of Munich.

References

[1] A. Hashimoto and S. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *Proc. 8th Design Automation Workshop*, pp. 155–169, 1971.

[2] D. N. Deutsch, "A dogleg channel router," in *Proc. 13th Design Automation Conf.*, 1976.

[3] G. Persky, D. N. Deustch, and D. G. Schweikert, "LTX–A minicomputer-based system for automatic LSI layout," *J. Design Automation and Fault-Tolerant Computing*, pp. 217–255, May 1977.

[4] —, "LTX–A system for the directed automation design of LSI circuits," in *Proc. 13th Design Automation Conf.*, 1976.

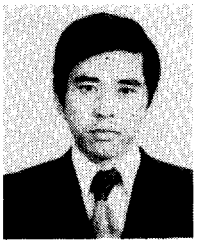
[5] B. W. Kernighan, D./G. Schweikert, and G. Persky, "An optimum channel-routing algorithm for polycell layouts of integrated circuits," in *Proc. 10th Design Automation Workshop*, pp. 50–59, 1973.

[6] T. Ohtsuki, H. Mori, E. S. Kuh, T. Kashiwabara, and T. Fujisawa, "One-dimensional logic gate assignment and interval graphs," *IEEE Trans. Circuits Syst.*, vol. CAS-26, pp. 675–684, Sept. 1979.

[7] C. G. Lekerkerker and J. Ch. Boland, "Representation of a finite graph by a set of intervals on the real line," *Fund. Math.*, vol. 51, pp. 45–64, 1962.

[8] W. Donath, private communication.

*



Takeshi Yoshimura was born on December 19, 1948. He received the B.E. and M.E. degrees in electronics from Osaka University, Japan, in 1972 and 1974, respectively.

He joined Nippon Electric Company, Ltd., in 1974, and is now a Research Member of Application System Research Laboratory, C & C System Research Laboratories. He has been engaged in the research and development of computer application system for communication network design and LSI Layout design. During 1979–1980 he was on leave at Electronics Research Laboratory, University of California, Berkeley, where he worked on LSI Layout design.

Mr. Yoshimura is a member of the Institute of Electronics and Communication Engineers of Japan.

*

Ernest S. Kuh (S'49–A'52–M'57–F'65) was born in Peking, China, on October 2, 1928, and attended Shanghai Jiao-tong University from 1945 to 1947. He received the B.S. degree from the University of Michigan, Ann Arbor, in 1949, S. M. degree from the Massachusetts Institute of Technology, Cambridge, in 1950, and the Ph.D. degree from Stanford University, Stanford, CA, in 1952.

From 1952 to 1956, he was a member of the Technical Staff at the Bell Telephone Laboratories, Murray Hill, NJ. He joined the Electrical Engineering Department faculty at the University of California, Berkeley, in 1956. From 1968 to 1972 he served as Chairman of the Department of Electrical Engineering and Computer Sciences. From 1973 to 1980 he served as Dean of the College of Engineering, University of California, Berkeley. He is coauthor of three books: *Principles of Circuit Synthesis*, 1959 with D. O. Pederson; *Theory of Linear Active Networks*, 1967 with R. A. Rohrer; and *Basic Circuit Theory*, 1969 with C. A. Dosoer.

Dr. Kuh is a member of the National Academy of Engineering and a member of the Academia Sinica. He is the recipient of the 1981 ASEE Lamme Medal and the 1981 IEEE Education Medal.