A Linear-Time Routing Algorithm for Convex Grids

A Linear-Time Routing Algorithm for Convex Grids

TAKAO NISHIZEKI, MEMBER, IEEE, NOBUJI SAITO, MEMBER, IEEE, AND KIMINOBU SUZUKI

Abstract—**In this paper, we consider the channel routing problem involving two-terminal nets on rectilinear grids. An efficient algorithm is described which necessarily finds a routing in a given grid whenever it exists. The algorithm is not a heuristic but an exact one, and works for a rather large class of grids, called convex grids, including the grids of rectangular, T-, L-, or X-shape boundaries. Both the running time and required space are linear in the number of vertices of a grid.**

I. INTRODUCTION

"CHANNEL ROUTING" is a significant wiring problem encountered in the layout of two-dimensional circuits (typically, but not restrictively, VLSI circuits). It has been the subject of several investigations in the past [2]-[6], [8]-[10]. However, most of the known algorithms cannot always find all the connections even if they exist, or lead to long running time, due to their heuristic approach.

Traditionally, a rectangular paradigm is used in VLSI design. Modules have rectangular shapes, and the routing among them is achieved by partitioning the given space into rectangular channels. Each such channel is assigned nets with terminals along its boundary, and the nets are routed inside. Although there are a bewildering array of routing models, we assume Thompson's two-layer model [11] or Preparate and Lipski's three-layer model [9], in which the paths connecting terminals may cross each other and share turing points but cannot share segments (see Fig. 1). This is somewhat different from the model in [2], [6], [10] where turning points cannot be shared. If a channel routing problem involves only two-terminal nets in our model, then the problem can be effectively reduced to a problem for finding edge-disjoint paths, each of which connects two terminals specified on the boundary of a given *rectangular* grids [9], [11]. On the other hand Pinter proposed to partition a routing region into *nonrectangular* channels for the purpose of obtaining an area-efficient routing [8]. In this case the grid is not always rectangular, but the boundary is T-, L-, X-shape or the like. (See Fig. 2.) Although Frank [5] has given an algorithm for finding edge-disjoint paths in a grid, the grid is restricted to a rectangular one, and the naive implementation of his algorithm requires $O(n^2)$ time. Throughout this paper n is the number of vertices in a grid. Our algorithm for finding multicommodity flows in planar graphs yields an alternative algorithm for finding edge-disjoint paths for a certain class of planar graphs including the rectangular grids [7]. However, it also requires $O(n^2)$ time. (More precisely, the routing problem can be reduced to the "integral" multicommodity flow problem. The algorithm in [7] can fortunately find integral flows, so yield edge-disjoint paths for the class of graphs.)

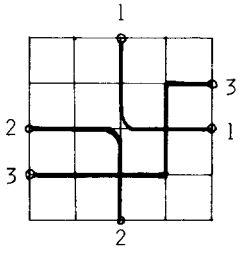


Fig. 1. Illustration for the routing model.

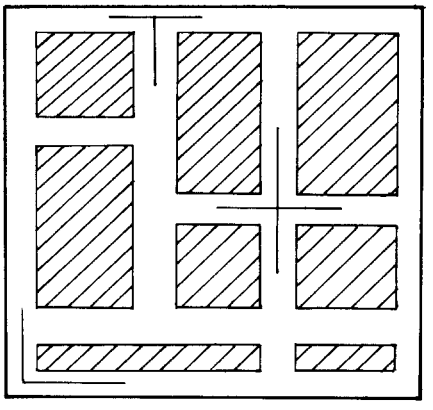


Fig. 2. Partition of a routing region into convex grids. (T-, L-, and X-shape regions.)

In this paper we give an efficient algorithm, which necessarily finds edge-disjoint paths in a given grid whenever they exist. Thus the algorithm is not heuristic unlike most of known routing algorithms. Futhermore the algorithm works for a rather large class of grids, called "convex grids," in which every two vertices are joined by a path with at most one bend. The grids of rectangular, T-, L-, or X-shaped boundaries are all convex. Both the running time and required space are linear in the number of vertices of a grid.

II. PRELIMINARIES

In this section we define some terms, and present a theorem. Let G^+ be the infinite graph whose vertex set consists of all points of the plane with integer coordinates and in which two vertices are connected if and only if the (Euclidean) distance between them is equal to 1. A *grid* is a finite subgraph of G^+. We say a grid G is *convex* if any two vertices of G are joined

Manuscript received October 5, 1983, revised August 9, 1984.

T. Nishizeki and N. Saito are with the Department of Electrical Communications, Faculty of Engineering, Tohoku University, Sendai 980, Japan.

K. Suzuki is with Toshiba Corporation, Semiconductor Division, Kawasaki 210, Japan.

0278/0070/85/0100-0068$01.00 © 1985 IEEE

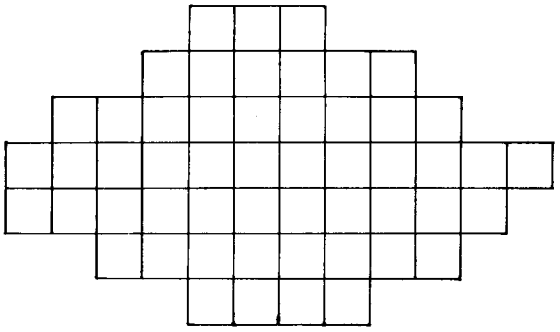


Fig. 3. A convex grid.

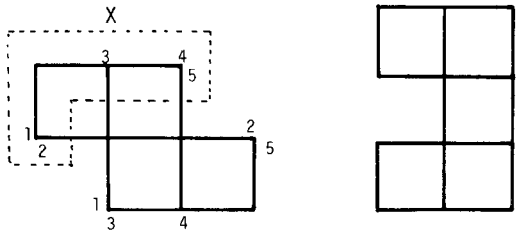


Fig. 4. Two examples of nonconvex grids. (Terminal numbers are attached to vertices in Fig. 4(a).)

by a path with at most one bend. The grids of rectangular, T-, L-, or X-shape boundaries are all convex. A typical example of a convex grid is depicted in Fig. 3, while two examples of nonconvex grids in Fig. 4.

By a *column* (*row*) of grid G we mean a region in G between two consecutive vertical (horizontal) lines. For a row r or G, let $x_l(r)$ (or $x_r(r)$) be the coordinate of the leftmost (rightmost) vertical edge that crosses row r. Then we immediately have an alternative characterization of convex grids.

Lemma 1: A grid G is convex if and only if G satisfies the following three conditions:

(a) each row of G crosses two vertical edges on the boundary;

(b) for any two rows r_a and r_b of G

$$x_l(r_a) \leqslant x_l(r_b) \text{ implies } x_r(r_a) \geqslant x_r(r_b);$$

(c) let $w_1, w_2, \cdots, w_k$ be the widths of rows, ordered from top to bottom, then

$$w_1 \leqslant w_2 \leqslant \cdots \leqslant w_{i-1} \leqslant w_i \geqslant w_{i+1} \geqslant \cdots \geqslant w_{k-1} \geqslant w_k$$

for some i, $1 \leqslant i \leqslant k$.

Let $G = (V, E)$ be a convex grid with vertex set V and edge set E. For $X \subset V$, we denote by $e(X)$ the number of edges with one end in X and the other in $V - X$. The *demand* of X, denoted by $d(X)$, is the number of terminal-pairs with one terminal in X and the other in $V - X$. The *margin* of X is $m(X) = e(X) - d(X)$.

For a column (or row) c of grid G we denote by $e(C)$ the number of edges that cross column (row) c. The *demand of column (row)* c, denoted by $d(c)$, is the number of terminal-pairs separated by c. The *margin* of c is $m(c) = e(c) - d(c)$. Grid G satisfies the *cut condition* if $m(c) \geqslant 0$ for every row or column c. This condition is necessary for the existence of the edge-disjoint paths, but is not sufficient.

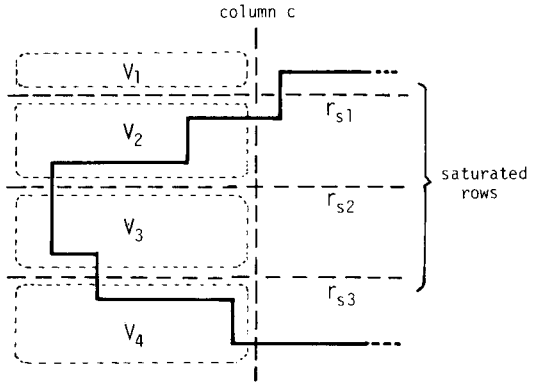


Fig. 5. Illustration of sets V_i.

Add to grid G a new edge joining each pair of terminals. The resulting graph is denoted by G^*. Define $e^*(X)$ for G^* in a way similar to $e(X)$ for G. A subset X of V is *odd* (or *even*) if $e^*(X)$ is odd (even). If X is odd, then at least one edge of G joining X and $V - X$ is used by none of the required edge-disjoint paths, an immediate consequence of a parity argument. The degree of vertex v in G^* is denoted by $\deg^*(v)$.

A row (or column) r is *saturated* if $m(r) = 0$. All the edges separated by a saturated r must be used by edge-disjoint paths. Assume that G has exactly t saturated rows $r_{s1}, r_{s2}, \cdots, r_{st}$, and let c be an arbitrary column. Then these t rows partition the set of vertices to the left of c into $t + 1$ subsets V_i, $i = 1, \cdots, t + 1$, as illustrated in Fig. 5. If V_i is odd, then at least one edge joining V_i and $V - V_i$ must be used by none of edge-disjoint paths; in particular the edge crosses column c since all the edges crossing saturated row $r_{s_{i-1}}$ or r_{s_i} must be occupied by edge-disjoint paths. Thus $e(c)$ must exceed $d(c)$ by at least the number of such odd V_i if there exist requried edge-disjoint paths. The number is called a *parity demand* of c, and denoted by $d_p(c)$. The *revised demand* of c is $d_r(c) = d(c) + d_p(c)$. Similarly define the revised demand $d_r(r)$ of row r. A grid G satisfies the *revised cut condition* if $d_r(c) \leqslant e(c)$ for every column or row c. The revised cut condition is necessary for the existence of edge-disjoint paths. When G^* is Eulerian in particular, the revised cut condition is equivalent merely with the cut condition, because then every $X \subset V$ is even. The "revised cut condition" was first introducted by Frank [5].

In this paper we assume that terminals are assigned only to vertices on the boundary of G; at most one terminal to a vertex of degree three, at most two to a vertex of degree two, and no terminal to the others. Thus $\deg^*(v) \leqslant 4$ for every $v \in V$.

We obtain the following theorem. The necessity is immediate as above, so omitted. In the succeeding sections we will give an algorithm for actually finding required paths and verify its correctness, which virtually establishes the sufficiency of the theorem.

Theorem 1: A convex grid G has edge-disjoint paths, each joining a pair of terminals specified on the boundary, if and only if G satisfies the revised cut condition, that is, every column or row c satisfies $d_r(c) \leqslant e(c)$.

Frank has obtained the same result only for rectangular grids [5]. Theorem 1 is critical in a sense: if a grid is not convex then the revised cut condition is no more sufficient for the

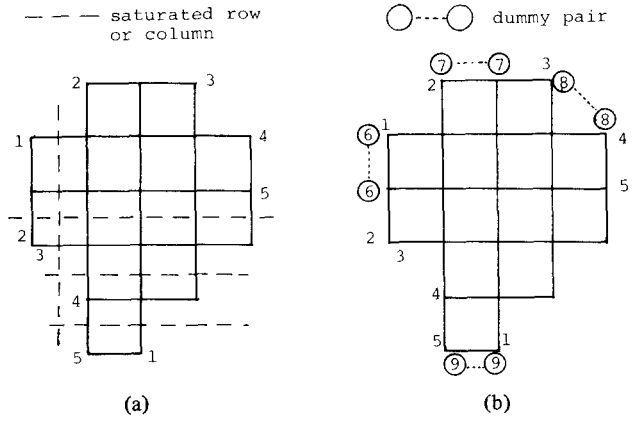


Fig. 6. Addition of dummy terminals. (a) A nongood grid not satisfying (iv). (b) A good grid with dummy terminals.

existence of edge-disjoint paths. For example the nonconvex grid G in Fig. 4(a) satisfies the cut condition, so the revised cut condition since G^* is Eulerian. However, G has no edge-disjoint paths between specified terminals; this follows immediately from the fact that $m(X) < 0$ for the vertex set X shown in Fig. 4(a).

III. Algorithm

In this section we give an algorithm PATHFINDER which finds edge-disjoint paths in a given convex grid G satisfying the revised cut condition. We need one more term. A grid G is *good* if G satisfies the following four conditions:

(i) G is convex;

(ii) G satisfies the cut condition;

(iii) terminals are assigned only to vertices on the boundary of G; at most one terminal to a vertex of degree three, at most two to a vertex of degree two, and no terminal to the others;

(iv) G^* is Eulerian.

Fig. 6 illustrates good and nongood grids.

The algorithm is outlined as follows. We first assign *dummy terminals* to vertices of the boundary so that new G becomes good. Applying procedure CUTOFF 1, 2, 3, or 4, we then decide which path uses each of the two edges incident to a corner v, and delete the two edges from G so that grid $G - v$ is also good. We then find all the remaining paths in $G - v$ by repeatedly applying CUTOFF's. Although the paths between dummy terminals are found, they are eventually discarded.

We now explain, in detail, how to assign dummy terminals to vertices of the boundary. First assign a new dummy terminal to each vertex $v \in V$ with $\deg^*(v) = 3$. Then pair off these terminals as follows. Consider the subsets of V partitioned by the saturated columns and rows. An even number of dummy terminals are assigned to vertices in each of such subsets (as shown in the proof of Lemma 2). For each subset, we pair off consecutive dummy terminals around the boundary in clockwise order, as illustrated in Figs. 6 and 7. Then we have the following lemma.

Lemma 2: Suppose that G is a convex grid satisfying the revised cut condition, and that dummy terminal-pairs are assigned to G as above. Then the resulting grid G' is good.

Proof: Since G satisfies the revised cut condition, the equation $d_p(c) = 0$ must hold for every saturated column (or row) c. Therefore, every subset of V partitioned by all the saturated rows and a saturated column is even. This fact immediately implies that each subset of V partitioned by all the saturated columns and rows is even and hence is attached an even number of dummy terminals. Thus the dummy terminals can be paired off in each of such subsets.

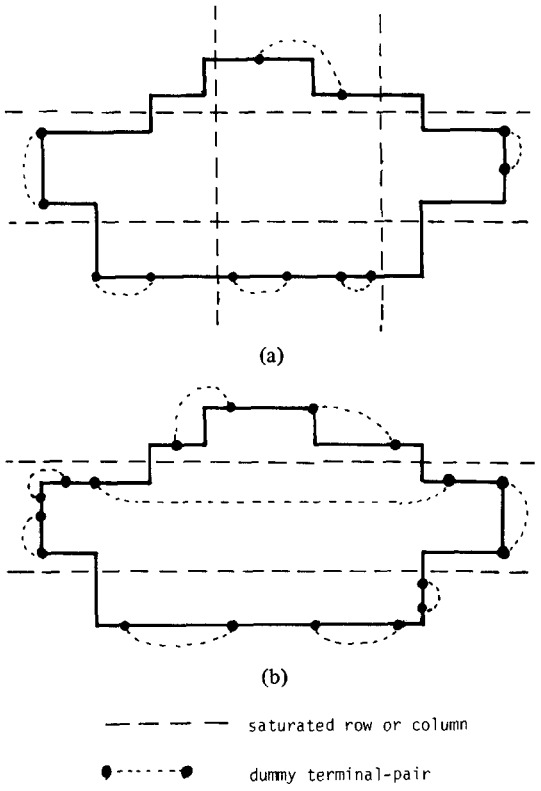


Fig. 7. Dummy terminal-pairs. (a) The case there exist a saturated column and a saturated row. (b) The case none of either columns or rows is saturated.

Since G' clearly satisfies Conditions (i), (iii), and (iv), it suffices to show that G' satisfies the cut condition. In what follows, a function defined with respect to G' is represented with a prime like e', d', and m'. Consider the following two cases.

Case 1: There exists a saturated column and a saturated row. Trivially $e'(c) = e(c)$ for every column (or row) c since graph G' is same as graph G. Clearly $d'(c) \leqslant d(c) + 2$ for every column (or row) c, and in particular $d'(c) = d(c)$ if c is saturated. (See Fig. 7(a).) Suppose that there exists a column (or row) c with $m'(c) < 0$. Then, since $m(c) = e(c) - d(c) \geqslant 0$, we have $m(c) = 1$ and $m'(c) = -1$. Since G'^* is Eulerian, $m'(c)$ must be even, a contradiction.

Case 2: None of either columns or rows is saturated. One may assume that none of columns is saturated. In this case it is not always true that $d'(c) \leqslant d(c) + 2$ for a column c. However the dummy terminals can be paired off so that $d'(c) \leqslant d(c) + d_r(c)$ (as illustrated in Fig. 7(b)). Therefore, the revised cut condition implies $m'(c) \geqslant 0$ for any column c. On the other hand one can easily verify $m'(r) \geqslant 0$ for every row r as in Case 1 above. Q.E.D.

We then apply procedure CUTOFF's to a corner of the shortest horizontal line segment of G, which must be either the uppermost or lowermost one by Lemma 1(c). We repeatedly

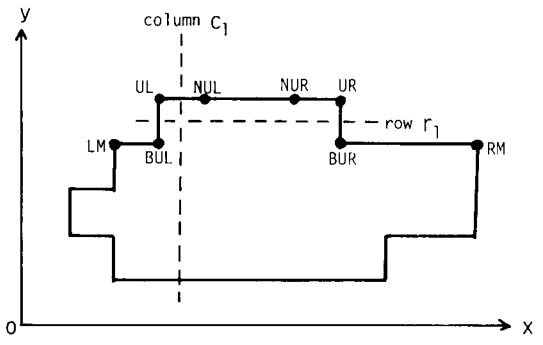


Fig. 8. Notation for vertices of G.

apply CUTOFF's to a corner of the segment a number of times equal to the number of the vertices there. Then the number of horizontal segments decreases by one in the resulting grid. Repeat this operation until G has exactly one horizontal segment, in case of which one can easily find the desired paths.

Throughout the execution of the algorithm we use UL, NUL, UR, NUR, BUL, BUR, LM, RM to generically denote eight vertices of grid G as specified in Fig. 8: UL (or UR) is the upper left (right) corner; NUL (NUR) is the next vertex to the right (left) of UL (UR); BUL (BUR) is the next vertex below UL (UR); LM (RM) is the leftmost (rightmost) vertex of the second uppermost line. The uppermost row is denoted by r_1, and the column separating UL and NUL by c_1. We are now ready to present algorithm PATHFINDER.

```
    procedure PATHFINDER:
    begin
1     add dummy terminals to given grid G so that the re-
      sulting grid becomes good;
      {the paths between dummy terminals will be discarded}
2     k := the number of the horizontal segments;
3     while k≥2 {there are two or more horizontal seg-
        ments} do
        begin
4         assume that the uppermost horizontal segment is
          the shortest; {otherwise rotate G by 180 degree}
5         while there remains a vertex on the segement do
            begin
6             if corner UL has a terminal
              then CUTOFF1
              else if corner UR has a terminal
                   then CUTOFF2
                   else if m(c1)>0 then CUTOFF3
                        else {m(c1)=0} then CUTOFF4;
7             delete every terminal-pair such that the
              two terminals are assigned to the same
              vertex {The desired path is trivial}
            end;
8         k := k-1 {the number of horizontal lines de-
            creased}
        end;
      {there remains exactly one horizontal line}
      decide, for each terminal pair, the horizontal line seg-
      ment between the two terminals as the required path
    end.
```

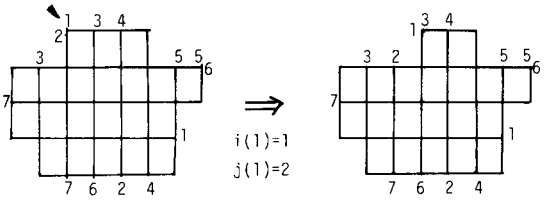


Fig. 9. CUTOFF1.

We define two sets of terminal-pairs separted by column c_1: $T_1(c_1)$ is the set of such pairs at least one terminal of which is located on segment LM-BUL-UL-UR-BUR-RM; and $T_2(c_1)$ is the set of pairs none of which is located on the segment. For two vertices u and v on the boundary, we write $u \leqslant v$ if vertices UL, u, and v appear clockwise on the boundary in this order. We denote by $x(v)$ the x-coordinate of vertex v, and by $y(v)$ the y-coordinate. A *u-v path* means a path between vertices u and v. We are now ready to present the details of CUTOFF's.

Case 1: Corner UL has a terminal. In this case we apply the following CUTOFF1. Note that UL has two terminals since G^* is Eulerian. Fig. 9 illustrates CUTOFF1.

```
procedure CUTOFF1:
begin
  let i(1) and j(1) be the two terminals on UL, and let i(2) and
  j(2) be the terminals paired with them;
  {assume that i(2)≤j(2)}
  if x(UR)-x(UL)≥2 {upper line has length ≥2} then
    begin
      reserve edge (UL, NUL) for the i(1)-i(2) path;
      reserve edge (UL, BUL) for the j(1)-j(2) path;
      i(1) := NUL;
      j(1) := BUL;
      G := G - UL {delete vertex UL from G}
      {the original i(1)-i(2) path in G consists of the
      i(1)-i(2) path in G - UL together with edge (UL,
      NUL), and the original j(1)-j(2) path in G consists
      of the j(1)-j(2) path in G-UL together with edge
      (UL, BUL).}
    end
  else {upper line has length one}
    begin
      if UR has no terminal then
        begin
          reserve segement UL-UR-BUR for the i(1)-i(2)
            path;
          reserve edge (UL, BUL) for the j(1)-j(2)
            path;
          i(1) := BUR;
          j(1) := BUL
        end
      else
        begin    {since G satisfies the cut condition,
                  UR has two terminals, one of which is
                  i(2)}
          let t(1) (≠i(2)) be the terminal of UR;
          let t(2) be the terminal paired with t(1);
          reserve edge (UL, UR) for the i(1)-i(2) path;
          reserve edge (UL, BUL) for the j(1)-j(2) path
```

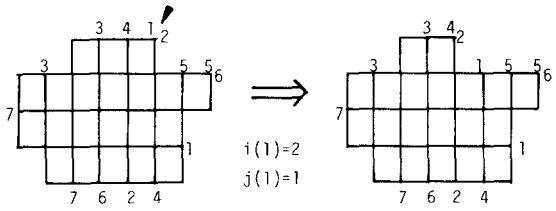


Fig. 10. CUTOFF2.

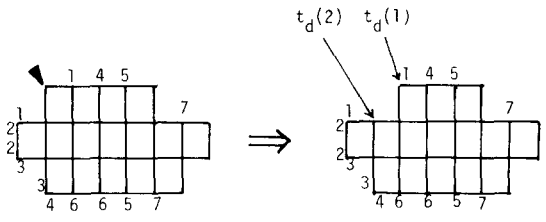


Fig. 11. CUTOFF3.

reserve edge (UR, BUR) for the $t(1)$-$t(2)$ path
delete pair $(i(1), i(2))$;
{the $i(1)$-$i(2)$ path has been found}
$j(1)$:= BUL;
$t(1)$:= BUR
end;
$G := G - \{UL, UR\}$
end
end;

Case 2: Corner UL has no terminal and corner UR has a terminal. In this case we apply CUTOFF2, which is the same as CUTOFF1 of the case G is turned right side left. CUTOFF2 is illustrated in Fig. 10.

Case 3: Both UL and UR have no terminal and c_1 is not saturated. In this case we apply the following CUTOFF3.

procedure CUTOFF3:
begin
create a new pair $(t_d(1), t_d(2))$ of dummy terminals;
if $x(UR)$-$x(UR) \geq 2$ then
begin
$t_d(1)$:= NUL;
$t_d(2)$:= BUL;
$G := G$ - UL
{the path between dummy terminals will be discarded. No path uses the two edges incident to UL.}
end
else
begin
$t_d(1)$:= BUR;
$t_d(2)$:= BUL;
$G := G - \{UL, UR\}$
end
end;

CUTOFF3 is illustrated in Fig. 11.

Case 4: Both UL and UR have no terminal and c_1 is saturated. In this case we apply the following CUTOFF4.

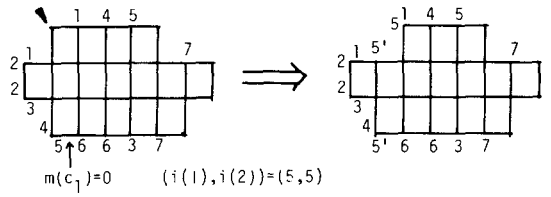


Fig. 12. CUTOFF4.

procedure CUTOFF4:
begin
if $T_1(c_1) \neq \phi$ then
choose an arbitrary pair $(i(1), i(2)) \in T_1(c_1)$
{assume $i(1) \leq i(2)$}
else {$T_1(c_1) = \phi$ and $T_2(c_1) \neq \phi$}
choose a pair $(i(1), i(2)) \in T_2(c_1)$ such that $i(1)$ is the nearest to vertex RM; {assume that $i(1) \leq i(2)$}
if $x(UR)$-$x(UL) \geq 2$ then
begin
reserve segment NUL-UL-BUL for the $i(1)$-$i(2)$ path;
create a new pair $(i'(1), i'(2))$;
$i'(1)$:= BUL;
$i'(2) := i(2)$;
$i(2)$:= NUL;
$G := G$ - UL
{the $i(1)$-$i(2)$ and $i'(1)$-$i'(2)$ paths will be found in G-UL, which, together with segment NUL-UL-BUL, constitute the original $i(1)$-$i(2)$ path in G}
end
else
begin
reserve segment BUR-UR-UL-BUL for the $i(1)$-$i(2)$ path;
create a new pair $(i'(1), i'(2))$;
$i'(1)$:= BUL;
$i'(2) := i(2)$;
$i(2)$:= BUR;
$G := G - \{UL, UR\}$
end
end;

CUTOFF4 is illustrated in Fig. 12.

IV. Correctness of Pathfinder

In this section we establish the correctness of algorithm PATHFINDER. It is trivial to verify the correctness for the case $k = 1$, i.e., the case G has exactly one horizontal line. At line 1 of PATHFINDER dummy terminals are initially added to the given grid so that the resulting grid becomes good. (See Lemma 2.) Therefore, it suffices to prove the following lemma, which leads to an easy proof, by induction on the number of vertices, for the correctness of PATHFINDER and also for the sufficiency of Theorem 1.

Lemma 3: Let $k \geq 2$. If CUTOFF is executed for a good grid G once, then the resulting grid G' obtained from G by deleting a corner is also good.

Proof: One can easily show that G' satisfies Conditions (i), (iii), and (iv) at the top of the preceding section. Therefore it suffices to verify Condition (ii), i.e., the cut condition. Assume that the uppermost horizontal line segment has length two or more; the proof for the other case is similar. We consider the following four cases depending on the executed CUTOFF.

Case 1: CUTOFF1. If $e(c)$ or $d(c)$ is changed by CUTOFF1, then c must be either the column c_1 or the row r_1. Therefore, we have $m'(c) \geqslant 0$ for every column or row c except c_1 and r_1.

First consider row r_1. Clearly $e'(r_1) = e(r_1) - 1$, $m(r_1) = e(r_1) - d(r_1) \geqslant 0$, and both $m(r_1)$ and $m'(r_1)$ are even. Furthermore, we have $d'(r_1) \leqslant d(r_1) + 1$ with equality for the case both terminals $i(2)$ and $j(2)$ lie on the horizontal segment UL-UR. Therefore, if $m'(r_1) < 0$ then $m(r_1) = 0$ and both $i(2)$ and $j(2)$ must lie on segment UL-UR. However, Condition (iii) implies that $m(r_1) > 0$ for such a case. Thus we have established $m'(r_1) \geqslant 0$.

Next consider column c_1. We easily have $m'(c_1) \geqslant -2$. Suppose that $m'(c_1) = -2$, then $m(c_1) = 0$ and both terminals $i(2)$ and $j(2)$ must be located to the left of column c_1. Furthermore one can easily know that c_1 is not the leftmost column. Let c_2 be the next column to c_1's left. Then an easy counting argument yields

$$d(c_2) \geqslant d(c_1) + 2 - (e(c_1) - e(c_2)).$$

Since c_1 is saturated, $e(c_1) = d(c_1)$. These two equations together imply $m(c_2) = e(c_2) - d(c_2) \leqslant -2$, contrary to the assumption. Thus we have established $m'(c_1) \geqslant 0$.

Case 2: CUTOFF2. Similar to Case 1.

Case 3: CUTOFF3. In this case column c_1 of G is not saturated. Furthermore row r_1 of G is not saturated since there is no terminal at corner UL or UR. Therefore, $m(c_1) \geqslant 2$ and $m(r_1) \geqslant 2$ since G^* is Eulerian. Clearly $e'(c_1) = e(c_1) - 1$, $d'(c_1) = d(c_1) + 1$, $e'(r_1) = e(r_1) - 1$, and $d'(r_1) = d(r_1) + 1$. Thus we have $m'(c_1) \geqslant 0$ and $m'(r_1) \geqslant 0$.

Case 4: CUTOFF4. Let $(i(1), i(2))$ be the terminal-pair chosen by CUTOFF4. Consider first the case $(i(1), i(2)) \in T_1(c_1)$. We easily have $m'(c) \geqslant 0$ for every column or row c except c_1 and r_1. Since $e'(c_1) = e(c_1) - 1$, $d'(c_1) = d(c_1) - 1$ and $m(c_1) \geqslant 0$, we have $m'(c_1) = e'(c_1) - d'(c_1) \geqslant 0$ as desired. For row r_1 we have $e'(r_1) = e(r_1) - 1$ and $d'(r_1) \leqslant d(r_1) + 1$. Since none of terminals of G is attached to corner UL or UR and G^* is Eulerian, we have $m(r_1) \geqslant 2$. These three equations imply $m'(r_1) \geqslant 0$.

Consider next the case $(i(1), i(2)) \in T_2(c_1)$. One can easily prove that $m(r_1) \geqslant 0$ and $m(c) \geqslant 0$ for every column c. If r is a row other than r_1, then $e'(r) = e(r)$ and $d'(r) \leqslant d(r) + 2$. Suppose that there exists such a row r with $m'(r) < 0$. Then $m(r) = 0$ and both terminals $i(1)$ and $i(2)$ must be located below the row r. Let X be the set of vertices above r, and Y to the left of c_1. Then an easy counting argument leads to

$$e(X \cap Y) + e(X \cup Y) = e(X) + e(Y) - 2e(X - Y; Y - X)$$

$$d(X \cap Y) + d(X \cup Y) = d(X) + d(Y) - 2d(X - Y; Y - X)$$

where $e(X - Y; Y - X)$ denotes the number of edges of G joining $X - Y$ and $Y - X$ and $d(X - Y; Y - X)$ denotes the number of terminal-pairs with one terminal in $X - Y$ and the other in $Y - X$. Subtracting the latter from the former, we have

$$m(X \cap Y) + m(X \cup Y) = m(X) + m(Y) - 2e(X - Y; Y - X) + 2d(X - Y; Y - X).$$

Since c_1 and r are saturated, $m(X) = m(Y) = 0$. Since G is a grid, $e(X - Y; Y - X) = 0$. Furthermore we have $d(X - Y; Y - X) = 0$ since $T_1(c_1) = \phi$ and every pair having one terminal on the boundary between UL and $i(1)$ is not separated by column c_1. Thus the right-hand side of the equation above must be zero. On the other hand, since G is a good grid, one can observe that each of the two terms in the left-hand side is nonnegative. Therefore, $m(X \cap Y) = m(X \cup Y) = 0$. However, since no terminal is attached to the corner UL, it can be easily observed that $m(X \cap Y) > 0$. This is a contradiction. Q.E.D.

V. Computation Time

In this section we show that PATHFINDER can be implemented so that it runs in $O(n)$ time. The problem is the time required for finding a terminal-pair $(i(1), i(2))$ in CUTOFF4.

V.1. Finding (i(1), i(2))

If $T_1(c_1) \neq \phi$, then we choose an arbitrary pair in $T_1(c_1)$. Otherwise, we need to find a pair $(i(1), i(2))$ in $T_2(c_1)$ such that $i(1)$ is the terminal nearest to vertex RM around the boundary (i.e., the smallest in the order $\leqslant$). A naive method finding such a pair requires $O(n)$ time. Hence the straightforward implementation of PATHFINDER would require $O(n^2)$ time since it repeats CUTOFF's $O(n)$ times. In what follows we show how to find $(i(1), i(2))$ efficiently.

Grid G is represented by n adjacency lists [1]. Moreover we employ two kinds of devices: a doubly linked list and two pointers TOP and NTOP.

Suppose that we are now starting to execute CUTOFF's for an uppermost horizontal line segment, that is, to execute the loop of lines 4–8 in procedure PATHFINDER. Find all the terminal-pairs in $T_1(c_1)$. Clearly it can be done in time proportional to the length of the segment LM-BUL-UL-UR-BUR-RM. The set $T_1(c_1)$ is represented by a doubly linked list containing them. Moreover we link each pair in $T_1(c_1)$ by a pointer with a vertex on the segment assigned its terminal so that the pair and the vertex can be directly accessed from each other. When the processing column c_1 proceeds to the next column to the right of current c_1 by CUTOFF1, 3, or 4, a pair may newly appear in $T_1(c_1)$ or disappear from $T_1(c_1)$. Using the pointers above, we can update the list $T_1(c_1)$ in $O(1)$ time per execution of each CUTOFF.

We next introduce two pointers TOP and NTOP. Let $(i(1), i(2))$ be a pair in $T_2(c_1)$ such that $i(1)$ is the terminal nearest to RM. Each pointer is directed to a vertex on the boundary of G so that the following properties hold:

(a) either $i(1) = \text{NTOP}$ or else $\text{TOP} \leqslant i(1)$ just before the while statement at line 5 is executed;

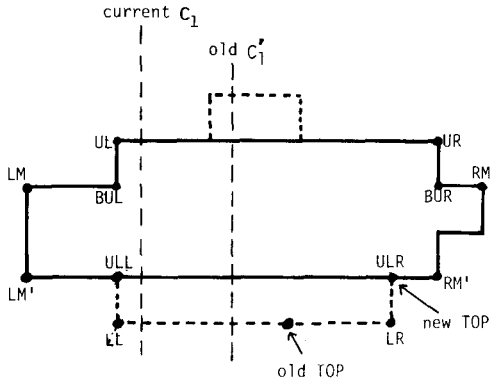


Fig. 13. Illustration for the proof of Lemma 4.

(b) if $T_1(c_1) = \phi$ and $T_2(c_1) \neq \phi$ then $i(1)$ = NTOP during the while statement at line 5 is executed.

If property (b) holds true, then it is quite easy to find $(i(1), i(2))$ when CUTOFF4 is executed: if $T_1(c_1) \neq \phi$ then simply choose an arbitrary pair in $T_1(c_1)$; otherwise, choose a pair in $T_2(c_1)$ a terminal of which is pointed by NTOP. We should show how to manage TOP and NTOP so that (a) and (b) hold.

Global Initialization of TOP and NTOP: Suppose that we are just now starting to execute the loop of lines 4-8 at the first time. Traversing the boundary of G clockwise from vertex RM, we find a pair $(i(1), i(2)) \in T_2(c_1)$ such that $i(1)$ is first encountered. Initialize pointers TOP and NTOP to $i(1)$. (If $T_2(c_1) = \phi$ then initialize TOP and NTOP to the lowest vertex of G on the vertical line $x = x$(UL).) Pointer NTOP must be altered when column c_1 proceeds right and a pair appear in $T_2(c_1)$ which has a terminal nearer to RM than NTOP. That is, if there exists a pair $(j(1), j(2)) \in T_2(c_1)$ such that $j(1) <$ NTOP and $j(2)$ is located at the lowest vertex on line $x = x$(UL), then we move NTOP to $j(1)$ for new c_1.

If the lowermost horizontal segment is shorter than the uppermost one, then we execute CUTOFF's to the former. In this case, after rotating G by 180 degrees, we execute CUTOFF's in the same way as the case the uppermost is shorter. We use counterparts of TOP and NTOP for the lowermost segment and manage them as well. We have the following lemma.

Lemma 4: Suppose that algorithm PATHFINDER now begins to execute the loop of lines 4-8 for the uppermost segment but not the first time and that $(i(1), i(2)) \in T_2(c_1)$ and $i(1)$ is the terminal nearest to RM.

(a) If the loop of lines 4-8 was executed for the uppermost segment also at the last time, then TOP $\leqslant i(1)$.

(b) If the loop of lines 4-8 was executed for the lowermost segment at the last time, then the following (i) or (ii) holds:

(i) TOP $\leqslant$ ULR implies TOP $\leqslant i(1)$, and ULR $<$ TOP implies ULR $\leqslant i(1)$;

(ii) $i(2)$ = ULL,

where, as shown in Fig. 13, LR (LL) is the rightmost (leftmost) vertex of the lowermost horizontal line processed at the last time, and ULR (ULL) is the next vertex above LR (LL).

Proof: (a) If $(i(1), i(2)) \in T_2(c_1')$ when the algorithm began CUTOFF's for the uppermost segment (drawn in a dotted line in Fig. 13) at the last time, then TOP $\leqslant i(1)$ follows immediately from the definitions of a convex grid and $T_2(c_1)$. Otherwise, $i(1)$ must be located to the left of the column c_1', and hence TOP $\leqslant i(1)$.

(b) The execution of CUTOFF's for the segment LR-LL possibly makes a terminal located on that line or a new dummy terminal move on the segment ULR-ULL. Therefore, if (i) is not true, then x(UL) = x(LL) and $i(2)$ must be the terminal moved to ULL by CUTOFF's done for the segment LR-LL (drawn in dotted line in Fig. 13). Q.E.D.

Local Initialization of TOP and NTOP: Suppose that we are now starting to execute the loop of lines 4-8 but not the first time. In this case we manage the pointers TOP and NTOP as follows. Find a terminal-pair $(i(1), i(2)) \in T_2(c_1)$ by traversing clockwise the boundary of G from TOP; traverse from vertex ULR if the vertex pointed by TOP disappeared from G (i.e., TOP was located on segment LR-LL). Furthermore move TOP to $i(1)$. If there is no such a pair, then move pointer TOP to the lowest vertex p of G on the line $x = x$(UL). If there exists a terminal $j(2)$ at p such that $(j(1), j(2)) \in T_2(c_1)$ and $j(1) <$ TOP, then set NTOP := $j(1)$. Otherwise, set NTOP := TOP. NTOP is altered when column c_1 moves right and a pair appears in $T_2(c_1)$ which has a terminal nearer to RM than current NTOP.

By Lemma 4 property (a) clearly holds true if pointers TOP and NTOP are managed as above. We should verify property (b). Consider an execution of the while statement on line 5 of PATHFINDER. If the loop of lines 6-7 is first executed, then the two equations $T_1(c_1) = \phi$ and $T_2(c_1) \neq \phi$ clearly imply $i(1)$ = NTOP if TOP and NTOP are managed as above. We shall show that this holds throughout the execution of the while statement.

Lemma 5: Suppose that we are executing CUTOFF's for the uppermost segment, and that CUTOFF4 chooses a terminal-pair $(i(1), i(2)) \in T_2(c_1)$ pointed by NTOP. Then the following holds during the execution of CUTOFF's for that segment: set $T_1(c_1)$ is not empty whenever the processing column c_1 is located to the left of $i(1)$.

Proof: The CUTOFF4 divides the terminal-pair $(i(1), i(2))$ into two pairs; terminal $i(2)$ is newly assigned to NUL (which becomes the upper left corner of G-UL). Thereafter the terminal $i(2)$ may move to another vertex by CUTOFF's, but is always located to c_1's left and in particular on segment LM-BUL-UL-UR. Hence $(i(1), i(2)) \in T_1(c_1)$ so $T_1(c_1) \neq \phi$ whenever c_1 is located to the left of $i(1)$. Q.E.D.

Lemma 6: Property (b) holds true, that is, if $T_1(c_1) = \phi$ and $T_2(c_1) \neq \phi$ then $i(1)$ = NTOP during the while statement on line 5 is executed.

Proof: The initializations of TOP and NTOP guarantee that NTOP points to the terminal in $T_2(c_1)$ nearest to RM just before the while statement on line 5 is executed. Thereafter NTOP is altered whenever a terminal appears in $T_2(c_1)$ which is nearer to RM than NTOP. However the terminal-pair $(i(1), i(2))$ pointed by NTOP disappears from $T_2(c_1)$ in the following two cases. We will show that the claim holds true in either case.

Case 1: c_1 moves to NTOP's right. In this case the pair pointed by NTOP disappears from $T_2(c_1)$, but then $T_2(c_1) = \phi$. If c_1 moves further right and a new pair appears in $T_2(c_1)$, then NTOP is necessarily moved to a terminal of the pair.

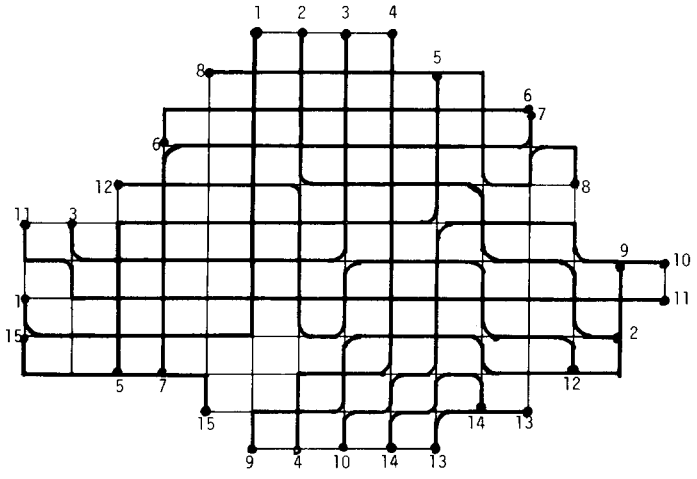


Fig. 14. A resulting routing.

Case 2: CUTOFF4 is executed for a pair $(i(1), i(2)) \in T_2(c_1)$ *pointed by NTOP.* In this case for the new c_1 we have $(i(1), i(2)) \notin T_2(c_1)$, and the terminal in $T_2(c_1)$ nearest to RM may not be pointed by NTOP. However Lemma 5 implies that $T_1(c_1) \neq \phi$ whenever column c_1 is located to $i(1)$'s left. When c_1 proceedes to $i(1)$'s right, either $T_2(c_1) = \phi$ or NTOP points to a terminal-pair newly appeared in $T_2(c_1)$ having a terminal nearest to RM. Q.E.D.

By Lemma 6 we can easily find a terminal-pair $(i(1), i(2))$.

V.2. Computation time

Clearly it can be done in linear time to check whether a given grid G satisfies the cut condition. Furthermore it can be done in linear time to add and pair off dummy terminals. The margin of all columns except c_1 remain unchanged by an execution of CUTOFF's. Furthermore $m(c_1)$ can be updated in $O(1)$ time per one execution of each CUTOFF. Therefore we may assume that $m(c)$ is known for all columns c during the execution of algorithm PATHFINDER.

In CUTOFF1 it is required to know whether $i(2) \leqslant j(2)$ for two given terminals $i(2)$ and $j(2)$ on the boundary. Making use of a geometrical feature of a convex grid, one can easily know it in $O(1)$ time. Thus one execution of CUTOFF1, 2, or 3 can be done in $O(1)$ time. Furthermore one execution of CUTOFF4 can be done in $O(1)$ time whenever set $T_1(c_1)$ together with pointers TOP and NTOP are managed correctly. Since CUTOFF1, 2, 3, and 4 are executed at most n times, they can be done in $O(n)$ time in total.

Lemma 7: Set $T_1(c_1)$ can be managed in $O(n)$ time in total.

Proof: Omitted since obvious.

Lemma 8: Pointers TOP and NTOP can be managed in $O(n)$ time in total.

Proof: Whenever executing line 5 for the uppermost segment, we find a pair $(i(1), i(2))$ by traversing clockwise the boundary from TOP (or from RM at the first time), and move TOP *clockwise* to the vertex $i(1)$. By the execution of line 5 for the lowermost segment pointer TOP is sometimes moved *counterclockwise* to a vertex (ULR in Fig. 13). Thus some edges are traversed more than once, but clearly all the edges are traversed at most twice. Pointer NTOP is often altered by CUTOFF's, but the alternation can be done in $O(1)$ time per execution of CUTOFF. Since CUTOFF is executed at most n times, NTOP can be managed in $O(n)$ time. Thus we have shown that the total time required for the management of TOP and NTOP is $O(n)$. Q.E.D.

Thus we have shown that algorithm PATHFINDER runs in $O(n)$ time. Clearly our date structure uses $O(n)$ space.

VI. Conclusion

We proposed an efficient algorithm for the channel routing problem. The algorithm necessarily finds edge-disjoint paths in a given grid whenever they exist. Thus the algorithm is not heuristic unlike most of known routing algorithms. Furthermore the algorithm works for convex grids, including the grids of rectangular, T-, L-, or X-shape boundaries. Both the running time and required space are linear in the number of vertices of a grid, so is optimal to within a constant factor. The algorithm has been coded in Pascal; it contains approximately 1700 lines. The program is implemented on FACOM 230/38s computer with 0.5 MIPs. Although a limited number of examples are experimented, the results are favorable. A grid of 125 vertices required less than one second. Fig. 14 shows the resulting routing, which contains 15 terminal pairs. Another grid of 486 vertices required less than three seconds.

Acknowledgment

The authors wish to thank Dr. S. Goto and Dr. K. Takamizawa of Nippon Electric Company for their useful comments, and also Y. Tanamura of Fujitsu Ltd. for his help on the computer experiments.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[2] D. J. Brown and R. L. Rivest, "New lower bounds for channel width," in *VLSI Systems and Computations*, H. T. Kung, B. Sproull, and G. Steel (eds.), pp. 178–185, 1981.

[3] D. Deutsh, "A dogleg channel router," in *Proc., 13th Design Automation Conf.*, pp. 425–433, 1976.

[4] D. Dolev, K. Karplus, A. Siegel, A. Strong, and J. D. Ullman, "Optimal wiring between rectangles," in *Proc. 13th Ann. ACM Symp. Theory of Computing*, pp. 312–317, 1981.

[5] A. Frank, "Disjoint paths in a rectilinear grid," in *Graph Theory Conf.*, (Logow, Poland), 1981, also to appear in *Combinatorica*.

[6] A. Hashimoto, and J. Stevens, "Wire routing by optimizing channel assignment," in *Proc. 8th Design Automation Conf.*, pp. 214–224, 1971.

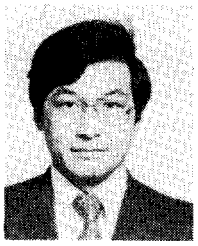
[7] K. Matsumoto, T. Nishizeki, and N. Saito, "An efficient algorithm for finding multicommodity flows in planar networks," *SIAM J. Comput.*, to be published.

[8] P. Y. Pinter, "Optimal routing in rectilinear channels," in *VLSI Systems and Computations*, H. T. Kung, B. Sproull, and G. Steel, (eds.), pp. 160–177, 1981.

[9] F. P. Preparate, and W. Lipski, Jr., "Three layers are enough," in *IEEE 23rd Symp. on Found. of Compt. Sci.*, (Chicago, IL), pp. 350–357, 1982.

[10] R. L. Rivest, A. Baratz, and G. Miller, "Probably good channel routing algorithms," in *VLSI Systems and Computations*, H. T. Kung, B. Sproull, and G. Steel, (eds.) pp. 153–159, 1981.

[11] C. D. Thompson, "A complexity theory for VLSI," Tech. Rep. CMU-CS-80-140, Ph.D. dissertation, Carnegie-Mellon University, Pittsburgh, PA, Aug. 1980.

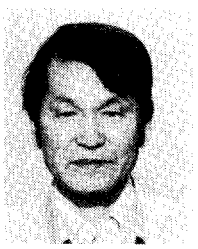


Takao Nishizeki (S'71–M'74) was born in Sukagawa, Japan, on February 11, 1947. He received the B.E., M.E., and Dr.Eng. degrees from Tohoku University, Sendai, Japan, in 1969, 1971, and 1974, respectively, all in electrical communication engineering.

He joined Tohoku University in 1974, and is currently Associate Professor of Electrical Communications. From 1977 to 1978 he was on leave at Mathematics Department of Carnegie-Mellon University, Pittsburgh, PA. His areas of research interest are algorithms, graph theory, network theory, and computational complexity.

Dr. Nishizeki is a member of the Institute of Electronics and Communication Engineers of Japan, the Information Processing Society of Japan, and the American Mathematical Society.

*



Nobuji Saito (M'62) was born in Oita, Japan, in 1928. He received the B.E. and D.E. degrees from Tohoku University, Sendai, Japan, in 1951 and 1962, respectively.

He has been with Tohoku University, since 1951, where he is at present a Professor of the Department of Electrical Communications in charge of research and teaching on the circuit and graph theories. He stayed at the Polytechnic Institute of Brooklyn as a Research Fellow, during 1964 academic year. He has written *Microwave Filters and Circuits* (ed. (Matsumoto) Academic Press, 1970, as one of coauthors, and "Graph Theory and Algorithms" Springer-Verlag, 1980, as one of editors and coauthors.

Dr. Saito is a member of the Institute of Electronics and Communication Engineers of Japan.

*

Kiminobu Suzuki was born on April 23, 1958. He received the B.E. degree in electrical communication in 1981, and the M.E. degree in information science in 1983, both from Tohoku University, Japan.

He joined Toshiba Corp. in 1983, and is now a member of Semiconductor Division. He has been engaged in the research and design of dynamic RAM's.

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

A Pattern Recognition Based Method for IC Failure Analysis

ANDRZEJ J. STROJWAS, MEMBER, IEEE, AND STEPHEN W. DIRECTOR, FELLOW, IEEE

Abstract—**Random fluctuations which are inherent in the IC manufacturing process cause production yields to be significantly less than 100 percent. Yield drop is caused by two types of faults, catastrophic and parametric. This paper deals with the diagnosis of parametric faults which occur during the manufacturing of IC's and cause the yield to drop below some acceptable level.**

A statistical pattern recognition approach to IC failure analysis is developed in this paper. Identification of faults is based on the analysis of the joint probability density function of circuit performances. A number of alternative approaches to IC failure analysis are developed and their efficiency is discussed. Also a complete statistical pattern recognition system with learning capability is proposed and all the stages of its development are discussed. An efficient method for fault simulation which is based upon statistical process simulation is proposed. The performance of the algorithms proposed in this paper has been successfully verified for data collected from commerical fabrication processes and from computer simulation.

Manuscript received January 31, 1984; revised May 18, 1984. This research was supported in part by the National Science Foundation under Grant ECS8203744 and by Harris Semiconductor Company.

The authors are with Carnegie-Mellon University, Pittsburgh, PA 15213.

I. INTRODUCTION

IT IS A well-known fact that *random fluctuations* which are inherent in the integrated circuit (IC) manufacturing process often cause the *production yield*, i.e., the ratio of the number of correctly functioning IC chips to the total number of chips fabricated, to be significantly less than 100 percent. Fluctuations which occur in the IC manufacturing process are due to

- variations in the material parameters,
- variations in the process control parameters,
- variations in the quality of processing equipment,
- nonuniform conditions of the fabrication process for different IC chips.

The fluctuations which are responsible for this decrease in yield can be classified as either *catastrophic* or *parametric faults*. Catastrophic faults are random defects which cause *hard failures* of an IC component. Such hard failures result in the failure of the whole chip. We say that a hard failure has occurred when the values of circuit performances are several orders of magnitude different from the tolerance limits. Examples of cat-

0278-0070/85/0100-0076$01.00