AN EFFICIENT VLSI SWITCH-BOX ROUTER

JER MIN JOU
JAU YIEN LEE
YACHYANG SUN
JHING FA WANG
National Cheng Kung University
Taiwan, ROC

Engineers at the the National Cheng Kung University have developed a tool for switch-box routing that can route regions with cyclic constraints and with terminals on three or four sides. Using a divide-and-conquer algorithm, the authors explore the greedy channel routing idea, applying techniques such as routing area partitioning, dynamic routing strategies, and the sweeping of concurrent bidirectional columns. The routing area is decomposed into three parts by two special parallel cut lines, which makes routing easier. The algorithm completely routes Burstein's switch-box problem and with an extension also routes the famous Deutsch channel example in 19 tracks.

Many digital systems are now built on a single chip, with sets of signal nets that connect circuit modules on the chip. The actual wired interconnection, or routing, of these nets is one of the most challenging problems in VLSI layout and design.[1] Traditionally, most chip routing is done using channel-router algorithms.[2-6] However, more difficult routing problems require the use of other routing approaches.

A channel is a rectangular routing region with terminals placed on two opposite, top and bottom sides. The list of terminals that must be connected electrically is called a signal set, and the connection of the set is called a net. Nets may exit on the two ends of the routing region not occupied by terminals, but the terminal locations on the left and right sides are not fixed. Custom VLSI chips, on the other hand, often have rectangular routing regions with fixed terminal locations on all four sides. Thus, routing these chips with a channel router is not always reliable.

The routing regions with fixed terminals on four sides are called switch boxes.[7-13] Switch boxes cause problems for *L*-shaped routing areas, as shown in Figure 1a; *Z*-shaped areas, as shown in Figure 1b; and channels with cyclic constraints, as shown in Figure 1c. In Figure 1c, we must route CH1 before CH2, CH2 before CH3, CH3 before CH4, and completing the cycle, CH4 before CH1.[14] We cannot use a channel router to complete this routing, so we need a special router, called a switch-box router. In addition, several researchers have proved that general channel routing is an NP-complete problem,[15-17] and switch-box routing is even more difficult than channel routing.[1] In short, there is a great need for efficient heuristic techniques that will solve this problem.

Several algorithms have been developed to attack switch-box routing. The Lee[18,19] and Hightower[20] algorithms work only a net at a time and so are time consuming, Hsu's search-path algorithm[14] is also sequential, although it tries to avoid blocking nets that have not yet been routed. The most popular method is based on the greedy style of algorithm[2] in which routing is done column by column (or row by row) using as much routing space per column (or row) as possible. Although these greedy routers are fast and simple,[8,9,11,12] they are often ineffective for complex routing. They may fail because the sweeping direction is wrong, for example. Sweeping refers to routing column by column or row by row. The algorithm may not always start in the appropriate direction. Another reason greedy algorithms often fail is that their routing strategy is always the same, even when the subregions and conditions are different. Finally, these algorithms may not connect too many nets, which can result in interconnection conflicts in the left and right sides.

In this article, we propose another greedy switch-box router that avoids these problems. Our router contains heuristics for routing area partitions and incorporates dynamic routing strategies and concurrent bidirectional column sweeping. A divide-and-conquer approach decomposes the routing area into three parts by means of a pair of parallel cut lines. The original switch-box problem is then transformed into two three-sided channels and a small (perhaps simpler) switch-box problem. In this way, we simplify the routing of the original problem. Experimental results to date are quite encouraging. Our algorithm completely routes Burstein's difficult switch-box example.[4] We have also extended it to route Deutsch's famous difficult channel example[5] in only 19 tracks,[13] which is the optimum number. It is one of the first, if not the first, router that can completely route the two difficult problems.

THE ROUTING PROBLEM

A routing region is an $(n+1,m+1)$ matrix of points within a rectangle that has two layers available for routing. This region or matrix is defined by

$$D = \{0,1,...,m\} * \{0,1,...,n\}$$

where m and n are positive integers such that each point in the matrix is defined by a pair of coordinates $\{x,y\}$ or two-tuple, where

$$x \in \{0,1,...m\}$$

and

$$y \in \{0,1,...n\}$$

Each two-tuple is called a gridpoint in the matrix. If we ignore the boundaries of this region, then we would have $(n-1) \times (m-1)$ gridpoints. Alternately, we could say that there are $(n-1)$ rows and $(m-1)$ columns available for routing in that region.

A set of gridpoints with the same y-coordinate is called a row or a track and is denoted by row(y), where row(y) = $\{(x,y) \mid x \in \{0,1,...,m\}\}$, for some fixed value of y. Thus, there are $(n+1)$ rows, one for each $y \in \{0,1,...,n\}$. Similarly, the set of gridpoints with the same x-coordinate is called a column denoted by col(x). Row(0) denoted by s_1 and row(n) denoted by s_3 form one of two pairs of opposite sides of the routing region and are represented as $P_1=\{s_1,s_3\}$. Similarly, col(0) denoted by s_2 and col(m) denoted by s_4 are the other pair of opposite sides of the routing region and are represented as $P_2=\{s_2,s_4\}$. Figures 2a through 2c illustrate these definitions.

Figure 2d gives an illustration of terminal lists. Let $L=(l_1,...,l_{n-1})$ be the left side terminal list of the routing region. Then l_i is a positive integer identifying the net that enters the routing region from the left end of row(i). If l_i=0, no net has to be connected to the terminal and it is called a null terminal. Similarly, let $R=(r_1,...,r_{n-1})$ be the right side terminal list of the routing region. Then r_i is a positive integer identifying the net that enters the routing region from the right end of row(i). If r_i=0, no net has to be connected to the terminal and it is called a null terminal. Similarly, let $T=(t_1,...,t_{m-1})$ be the top terminal list and $B=(b_1,...,b_{m-1})$ be the bottom terminal list. Then t_j is a positive integer identifying the net that enters the routing region at the top of col(j), while b_j is a positive integer identifying the net that enters the routing region at the bottom of col(j). If t_j=0 (bj=0), no net has to be connected to the terminal and again we have a null terminal.

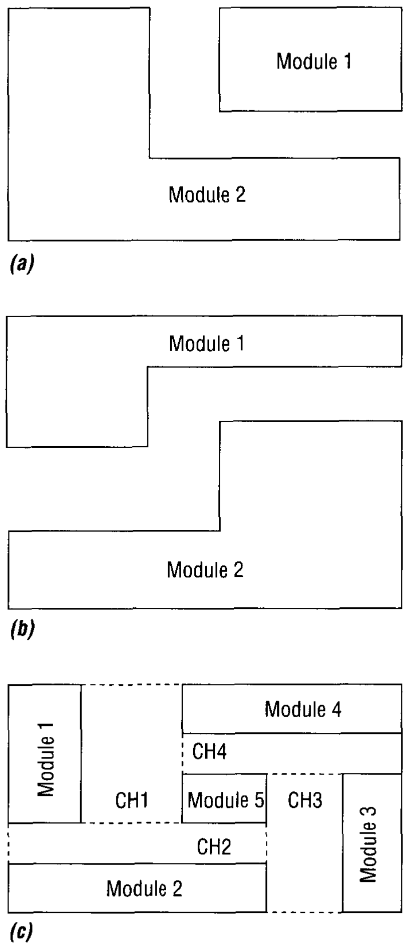


Figure 1. *Some cases that require a switch-box router.*

We allow two wires to cross one grid point and locate contacts or vias at such intersections to connect the two layers electrically.

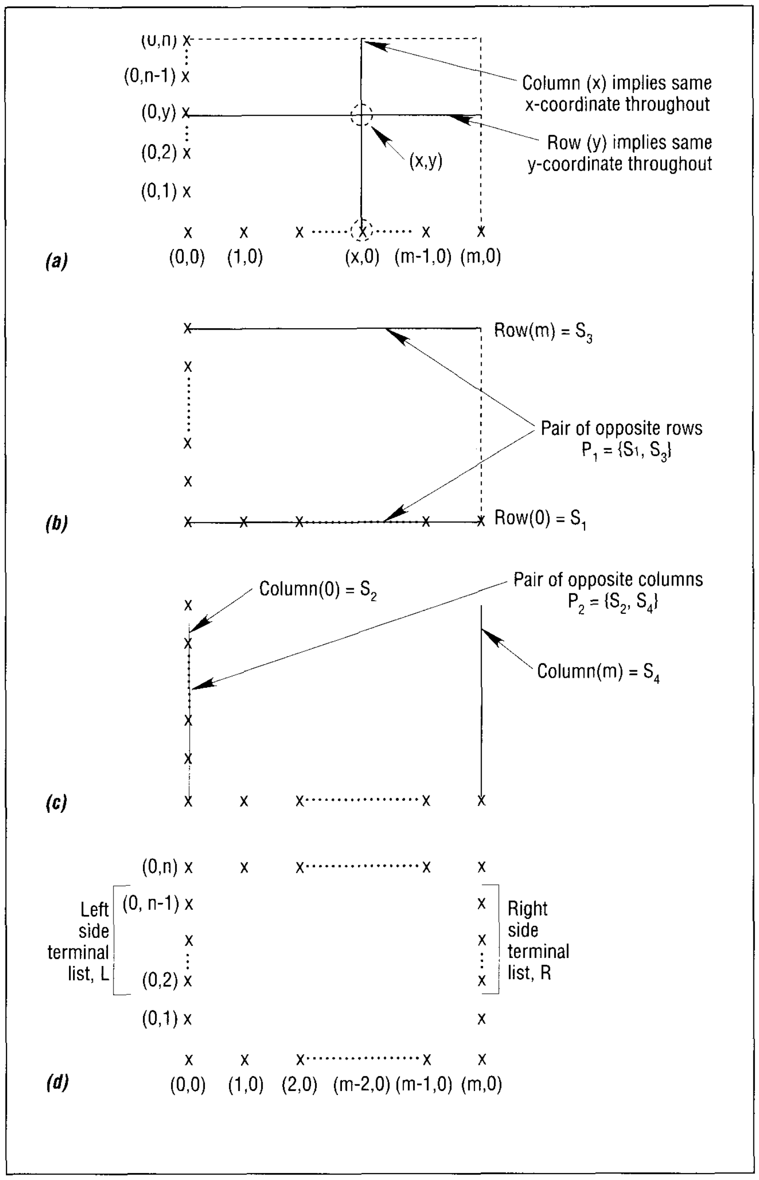

(0,n)
(0,n-1)
(0,y)
(0,2)
(0,1)
(x,y)
Column (x) implies same x-coordinate throughout
Row (y) implies same y-coordinate throughout
(0,0) (1,0) (x,0) (m-1,0) (m,0)
(a)
Row(m) = S3
Pair of opposite rows P1 = {S1, S3}
Row(0) = S1
(b)
Column(0) = S2
Pair of opposite columns P2 = {S2, S4}
Column(m) = S4
(c)
Left side terminal list, L
Right side terminal list, R
(0,n) (0, n-1) (0,2) (0,1)
(0,0) (1,0) (2,0) (m-2,0) (m-1,0) (m,0)
(d)

Figure 2. *Parts of a routing region: a row and column* ***(a)****; opposite row pair* P_1 ***(b)****; opposite column pair* P_2 ***(c)****; and terminal lists* ***(d)****.*

A routing wire, or simply a wire, is a vertical line on one layer, which is polysilicon, or a horizontal line on another layer, which is metal, along the columns and rows. (There are exceptions to this definition for a contact minimizer.) We allow two wires to cross one grid point and locate contacts or vias at such intersections to connect the two layers electrically.

We can state the switch-box routing problem in the following way: Given a routing region *D*, find the wires to connect all the terminals that belong to the same net within *D* such that we satisfy the connection requirements specified by *R*, *L*, *T* and *B* without definition violations.

Figure 3 shows a switch-box routing problem and its solution. To fully appreciate this problem, we need to understand a number of terms, including routing density, congestion, processing orientation, target column, start column, sparse region, and velocity ratio and sweeping velocity.

A divide-and-conquer approach decomposes the routing area into three parts by means of a pair of parallel cut lines.

Routing density. In a routing region, the two grid points of the same abscissa or ordinate, say j, on the opposite sides of P_i (i=1,2) determine a cut line denoted by L_{ij}. The routing density d_{ij} of each row(j) or col(j) (or cut line L_{ij}) is defined as the cardinality, or number of elements, of the set of nets whose horizontal or vertical wires must intersect the cut line L_{ij}.

Congestion. The congestion C_{ij} of each row(j) or col(j) (or cut line L_{ij}) is a measure of how wiring space is used across the cut line. It is therefore a ratio of wires intersecting at a cut line to the maximum number of wires that can cross that cut line. Thus,

$$C_{ij} = \frac{d_{ij}}{|s_{3-i}| - 2}, \quad \text{for } i = 1,2$$

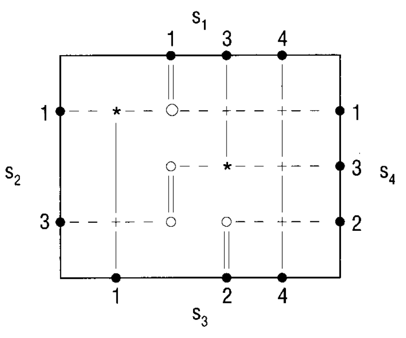
where i = 1,2 and $|s|$ is the cardinality of set s of gridpoints on some side; $|s_{3-i}| - 2$ is the maximum number of available tracks that cross cut line L_{ij}; and d_{ij} is the minimum number of available tracks that must cross cut line L_{ij}. Thus, C_{ij} represents the congestion of row(j) or col(j). When C_{ij} exceeds 1, the problem cannot be routed using our routing style.

Processing orientation. Determining processing orientation is the same as assigning the left side and right sides to either side pair P_1 or P_2 of the routing region.[8] The routing result depends on processing orientation. We propose a heuristic method to determine the processing orientation according to two criteria. First, the processing orientation must be perpendicular to the cut line with the least congestion to simplify the difficult left-right sweeping interconnections. Second, to obtain the benefit of greedy heuristics, we must choose the processing orientation with the longer length. We define it as

$$O_i = \frac{w_1}{\min\limits_{0<j<m \text{ or } n} C_{ij}} + \frac{w_2}{|s_{3-i}| - 2}, \quad \text{for } i = 1,2$$

where i= 1,2 and w_1 and w_2 are weights. The typical value for both w_1 and w_2 is 1. From this definition, we see clearly that the smaller the value of the least congestion and/or the longer the path along the processing orientation, the larger the value of O_i. Thus, we select the direction with the larger O_i value as the processing orientation.

Target column. The target column is column col(T) with minimum congestion, where left-right net interconnections will occur. We choose the target column using the processing orientation selected earlier. If more than one column satisfies this minimum-congestion, the column with the maximum absolute value of $T - (|s_i|/2)$ becomes the target column. We want to select the column farthest from the central routing region so that we can include the other columns with the least congestion in the sparse region defined later.



- - or || - metal layer,
| - polysilicon layer,
* - contact,
○ - turning point on same layer

Figure 3. *A switch-box routing problem and its solution. The routing region, D, is {0,1,2,3,4,5} * {0,1,2,3,4}. The right part, R, is (1,3,2); the left part, L, is (1,0,3); the top part, T, is (0,1,3,4), and the bottom part, B, is (1,0,2,4).*

After we decompose the routing region, we must define the different sweeping velocities in each subregion

Start column. The start column is one of two cut lines that are applied to decompose the routing region into three subregions. "Start" means that we sweep the sparse region starting from that column and going to the target column. If the target column is col(T), the start column is col(S) where S is defined as

$$S = T - k, \text{ if } T > (|s_i|/2) \text{ or } T = (|s_i|/2) \text{ and } C_{i\,(T-k)} < C_{i\,(T+k)}$$

or

$$S = T + k, \text{ if } T < (|s_i|/2) \text{ or } T = (|s_i|/2) \text{ and } C_{i\,(T-k)} \geq C_{i\,(T+k)}$$

where the typical value of k is either $|s_i|/3$ or 8 ... 11.

Sparse region. As we just mentioned, the whole routing region is decomposed into three parts. The routing subregion between the target column and the start column is called the sparse region. Beside the sparse region are two outer regions, called subregion I and subregion III. Figure 4 shows these areas.

Velocity ratio and sweeping velocity. After we decompose the routing region, we must define the different sweeping velocities in each subregion so we can route the two outer subregions concurrently and bidirectionally. Suppose the distance between the sparse region and the left side of the routing region is column X and the distance between the sparse region and the right side of the routing region is column Y, as shown in Figure 4. Then the velocity ratio, vr, is defined as

$$vr = \mathrm{Int}(X/Y + 0.5), \text{ if } X \geq Y$$

and

$$vr = \mathrm{Int}(Y/X + 0.5), \text{ if } X < Y$$

where Int(X) represents the integer part of X.

The velocity ratio is the ratio of the sweeping subregion I, V_L, to that of subregion III, V_R, along the processing orientation. V_L and V_R are defined as

$$V_L = vr, \text{ if } X \geq Y$$

or

$$V_L = 1, \text{ if } X < Y$$

and

$$V_R = 1, \text{ if } X \geq Y$$

or

$$V_R = vr, \text{ if } X < Y$$

Because the router alternately sweeps left and right columns in the two outer subregions as it goes toward the sparse region, it will reach opposite sides of the sparse region almost simultaneously.

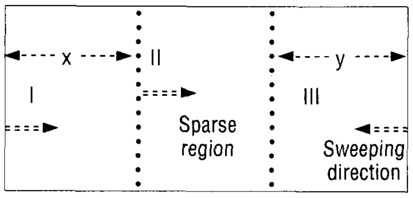


Figure 4. *A partitioned routing region, in which region II is the sparse region, S is the start column, and T is the target column.*

A GOOD SWITCH-BOX ROUTER

If there are only a few fixed terminals on the left and right sides of a switch box, the routing problem is usually easy. In general, however there are many terminals on the two sides, and the crowded part of the routing region is often in two outer subregions. Complicating matters are the many left-right terminal interconnections that are necessary to perfectly align the routing in the last column with terminals on the adjacent side.

To solve these problems, a switch-box router must be able to make doglegs[3,5] anywhere in the routing region, like a maze router.[18,19] Rivest and Fiduccia's GCRA greedy channel-routing algorithm[2] permits nets to change tracks at any time and thus satisfies this basic requirement, although the problem is still difficult. However, the GCRA does have the potential to solve the switch-box routing problem if we can extend it properly.

Another requirement for a switch-box router is to select the correct sweeping direction. Figure 5 shows what happens when the incorrect direction is chosen. As we will see later, selecting the correct direction is crucial to the routing quality. Using different rules in different routing regions is also desirable. Finally, we want to minimize the number of polysilicon interconnections and the number of vias to ensure the chip's yield and performance.

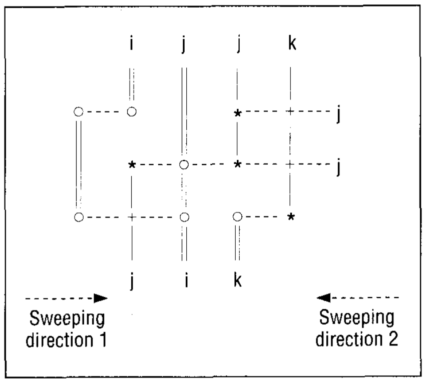


Figure 5. *The result produced by direction 2 sweeping is impossible routed by direction 1 sweeping.*

OUR ALGORITHM

In selecting the basic strategy of our router, we took into account all the concerns just described. First, we chose a good processing orientation and then applied a divide-and-conquer approach to solve the difficulty of crowded subregions. To do this, we decomposed the routing region into three subregions—I, II, and III—according to the positions of the target column and the start column. As shown in Figure 4, subregion II, also called the sparse region, is between these two columns.

We then applied a version of GCRA that we extended with concurrent, bidirectional column sweeping. The algorithm routes the two crowded subregions before routing the sparse region. To use routing area efficiently, the algorithm uses dynamically adjustable strategies to route the nets. It also eliminates unnecessary via holes and minimizes the use of polysilicon.

Finally, we routed any nets not routed by repeating the application of the extended greedy channel-routing algorithm.

To summarize, our program analyzes the data about the region to be routed, then sweeps bidirectionally to first complete routing in the dense regions, I and III. It then routes the sparse region, II. Finally, the program joins split nets—nets that occupy more than one track in the current column—across subregions. The actual routing is done by a procedure, called Sweep(*x*), which forms the inner loop of the total routing program. Figure 6 is an example of this procedure.It is used iteratively and routes wires in all three subregions. Figure 7 describes the routing program in C-like pseudocode.

Sweep(*x*) routes one column at a time, as Figure 6 shows. While operating on a given subregion, it can join wires through the join procedure, jog wires through the jog procedure, or minimize the use of vias through the via-min procedure. If it fails to achieve its goals in routing, it can increase the area of the subregion and try again to route a given set of wires. It begins by setting the sweep direction according to the value of a parameter, X. When *X* = ll, it sweeps columns from the left side of the routing region toward the sparse region. When *X* = rr, it sweeps columns from the right side toward the sparse region. When *X* = ss, it sweeps columns in the sparse region.

```
Step 1: Set the sweeping direction according to
   parameter x.
Step 2: if (available tracks exist) {
      bring top and bottom terminals into the
         nearest available tracks: rt, rb;
      join(rt,rb);
      jog(rt,rb);
      /* minimize on-line polysilicon and vias*/
      via-min();
   }
   else {
      increase number of rows;
      update routing data;
      goto step 2;
   }
Step 3: if ( next column == start_column
      || next column == target_column ) {
   if ( x == ll )
      completel = Yes;
   else completer = Yes;
   }
```

Figure 6. *The Sweep(X) procedure.*

```
Data input: specification of a switch-box routing
problem and the minimum jogging length.
  /* main procedure, written in C-like form */

Step 1: Analyze the input data, partition the routing
     region, and build the necessary database.
  /* concurrent bidirectional column sweeping in
subregion I and III */
Step 2: completel = completer = No;
   while ( !completel || !completer ) {
     /* route from one direction */
     for ($i$ = 1; $i \leq V_L$; i++)
        if (!completel) sweep(ll);
     /* routing switches to another direction */
     for ($i$ = 1; $i \leq V_R$; $i$++)
        if (!completer) sweep(rr);
   }

        /* routing in sparse region(subregion II) */
Step 3: if ( start_column ≤ target_column ) {
     for ($i$ = start_column; $i \leq$ target_column ; $i$++)
          sweep(ss);
   } else {
     for ($i$ = start_column; $i \geq$ target_column ; $i$--)
          sweep(ss);
Step 4: while (split nets exist) {
     increase one column;
     join as many split nets as possible;
   }
```

Figure 7. *Our routing algorithm.*

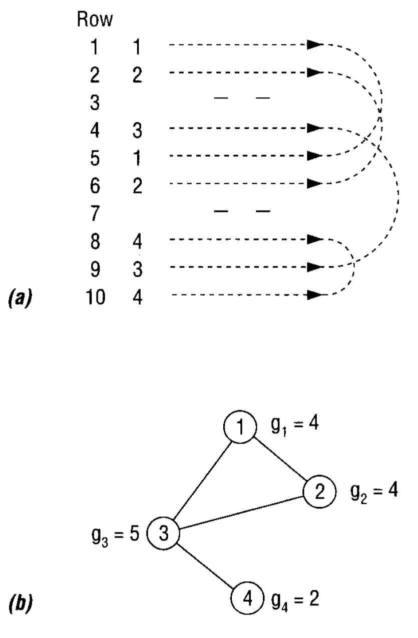


Figure 8. *Jog situation for split nets **(a)** and intersection graph G for jogging nets **(b)**.*

THE JOIN PROCEDURE

Join(high,low) joins as many split nets as it can between high and low tracks in the column. The procedure uses the following graph-theoretic approach to analyze the nature of the joining problem: For split nets on the current sweeping column, we construct a graph G. An arc exists between the split net i and net j vertices if and only if the vertical join wires of the two nets overlap.

Figure 8 gives an example. Figure 8a depicts the split nets, while Figure 8b shows the graph associated with them. Each vertex i has weight g_i, which indicates the length of the join wire. In Figure 8a, the corresponding weights are 4, 4, 5, and 2 for vertices 1, 2, 3, and 4 in Figure 8b. In general, the weight associated with vertex i is the weighted sum of the join length of net i and the number of tracks occupied by net i in the current sweeping column.

The objective of the join procedure is to find an independent set of vertices associated with the split nets that gives a maximum number of vertices. Also, if the number of vertices in one set equals the number of vertices in the other, then the set with the maximum sum of weights is chosen. A special case is when the weight associated with each vertex is exactly one. In this situation, the earlier problem is simply the following maximum independent set problem, where V is the set of vertices and E is the set of edges:[17]

Given a graph $G=(V,E)$ and a positive integer $K \leq |V|$, does G contain an independent set of size K or more?

An independent set is a set of vertices in a graph in which no two vertices are adjacent. In our problem, it means that the connection between one set of split nets would not intersect with a connection between another set of split nets. This problem is NP-complete.[17] We have developed a modified heuristic approach for practical implementations based on Deo's approach[21] that avoids exponential worst-case running times.

THE JOGGING PROCEDURE

The jogging procedure exploits the available vertical-layer, or polysilicon-layer, routing space so that the routing of the next columns is easier. The router moves each net in the current column towards its target row to a more favorable position. The parameters that dictate a more favorable location are

- type of routing subregion
- vertical constraints[3,5] between nets
- the next terminal or track occupied by the same net
- the available jogging length, or AJL (defined later)

The exact choice of jogging strategy is a complex process, as we will see later.

THE VIA-MIN PROCEDURE

This procedure minimizes the polysilicon interconnections and vias. After one column is routed, we can replace the polysilicon vertical wires with metal vertical wires on line instead of during postprocessing. This replacement procedure eliminates unnecessary vias, minimizes the use of polysilicon and keeps the contacts in adjacent tracks from meeting.

Unfortunately, via minimization only marginally decreases the number of vias in dense switch boxes. Since almost all tracks are occupied in all places, most vertical polysilicon wires cross horizontal metal tracks and so cannot be converted to metal. Nonetheless, in sparser switch boxes, such conversion can be quite dramatic and significantly decrease the number of vias.

JOGGING STRATEGIES

The jogging operation is very important because it affects the routing situation in subsequent columns. For example, we cannot solve the cyclic conflict[3,5] of nets h, i, j, and k in Figure 9 without performing the appropriate jogging operations. Rivest and Fidducia describe several jogging rules,[2] which we have extended so that the router can resolve the complex conflicts between nets associated with routing a switch box.

The jogging operation is very important because it affects the routing situation in subsequent columns.

The first step in selecting a jogging strategy is to define the target row. This row is the destination of each net in the current column that we want to move with the jogging operation. The distance between the row occupied by net i and its target row is called the jogging length and is denoted by JL_i. The target row for each net is not always free, since it may be occupied by another net. Consequently, we have another distance to denote: the distance between the row occupied by net i and the free row nearest to the target row. This distance is the available jogging length, or AJL_i.

The jogging ratio, JR_i for each net i becomes

$$JR_i = \frac{AJL_i}{JL_i}$$

The jogging criterion, JC_i, for each net i is

$$JC_i = w_3 * AJL_i + w_4 * JL_i$$

where w_3 and w_4 are weights. Typical values for w_3 and w_4 are 1.2 and 1.0. Using these definitions, we can describe different jogging strategies for each routing region. For our discussion, we assume that the routing region is divided into three subregions: I, II, and III (see Figure 4).

HORIZONTAL STRATEGIES

These strategies are grouped into sweeping in subregion I and II and sweeping in subregion II toward the target column. They are sufficent unless some net has more than one terminal in the target column, a case we address later.

Sweeping in subregion I or III—Strategy 1. For an unfinished net with unconnected terminals, the row of the next terminal is the target row of the net if there is no steady net condition.[2] If there is, then the target row is the net's current row.

Sweeping in subregion I or III—Strategy 2. For an unfinished net with all its terminals connected, the minimum-congestion row is the target row.

Sweeping in subregion II toward the target column—Strategy 3. When one side has terminals, the row of the next terminal is the target row of the net.

Sweeping in subregion II toward the target column—Strategy 4. When two sides have terminals, we can have two cases. In case one, the two sides are the top and bottom of the routing region. Here, the target

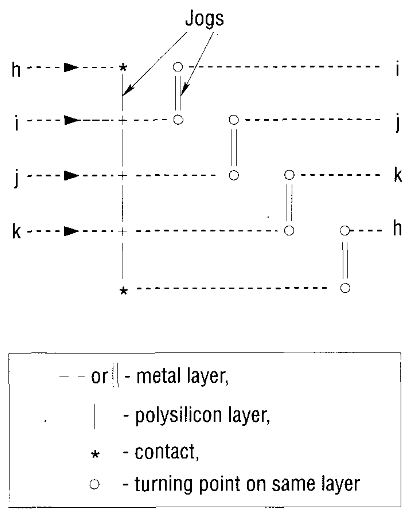


Figure 9. *Jogs and cyclic conflicts between nets.*

Vertical constraints are crucial to routability. The jogs for nets from a vertical constraint graph have priority over other jogs.

row is the net's current row. In the second case, one of the two sides is the target column. Here, the row position of the net dictates the target row. If the net's row is between the row positions of terminals, the target row is the net's current row (Figure 10a). Otherwise, the target row is the row of the terminal in the target column (Figure 10b).

Sweeping in region II toward the target column—Strategy 5. When three sides have terminals, the target row is the row of the terminal in the target column.

VERTICAL CONSTRAINTS

A horizontal wire of a net connected to the top termnal at a given column, must be placed above the wire of another net connected to the bottom terminal of the same column. This relationship can be represented by a directed vertical constraint graph, or VCG.[3] In the VCG, each node corresponds to a net, and a directed edge from node i to node j means that the horizontal wire of net i must be placed above that of net j. Figure 11 shows this representation..

Vertical constraints are crucial to routability. The jogs for nets arising from a VCG have priority over other jogs. The VCGs are built dynamically by our router along the sweeping direction, according to the following rules:

1. In the next three columns, termnate the current VCG, if not terminals of the leaf (or root) net of the VCG, which are located on the top (or corresponding bottom) side of the routing region exist. Build the next VCG from the next column, as shown in Figure 11a.

2. In the next three columns, terminate the current VCG if two terminals of the leaf (or root) net of the VCG, which are located on both the top and bottom sides of the same column exist. Build the next VCG from the next column as shown in Figure 11b.

CONFLICTS

Often there is a conflict in the set of potential jogs for the current column. Choosing a consistent set that is likely to simplify the subsequent routing is a critical problem. In this case, we prioritize jogs in the following order:

- VCG jog
- jog with $JR_i = 1$ (recall that the jogging ratio is available jogging length over the jogging length) if sweeping is in the sparse region
- jog of the net in the top or bottom track

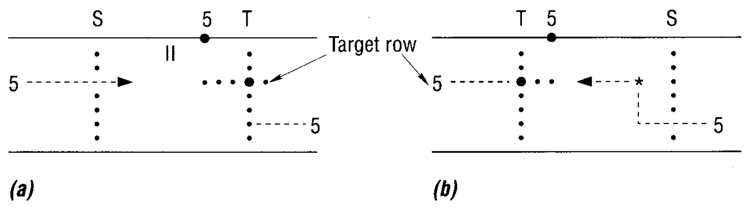


Figure 10. *Two conditions for two sides with terminals for net 5.*

- jog of the net with the largest value of JC_i (the jogging criterion discussed earlier)

The jogging operation is not always profitable for two reasons. The first is that each jog is a vertical wire segment that may block wirings of the other nets. The second reason is that jogging may generate additional contacts, which decreases the performance of the chip. For these reasons, we advise against making any jogs when the available jogging length of the net is shorter than the minimum jogging length, or MJL. The exceptions to this rule are the VCG jog and the jog when $JR_i = 1$.

We can adjust MJL dynamically according to the current column's routing congestion, C_{ia} and the following two columns' routing congestion, C_{ib} and C_{ic}, in the sweeping direction. This *dynamic rule* is defined as

$$\mathrm{MJL} = J + w_5 * (|s_{3-i}| - 2 - (|s_{3-i}| - 2) * C_{i,a} - w_6) * \mathrm{id}_1 * \mathrm{id}_2 * \mathrm{id}_3$$

$\mathrm{id}_1 = 1$, if $(|s_{3-i}| - 2 - (|s_{3-i}| - 2) * C_{ia} - w_6) > 0$
$\mathrm{id}_1 = 0$, otherwise

$\mathrm{id}_2 = 1$, if $C_{ib} < C_{ia}$
$\mathrm{id}_2 = 0$, otherwise

$\mathrm{id}_3 = 1$, if $C_{ic} < C_{ib}$
$\mathrm{id}_3 = 0$, otherwise

Jogging is not always profitable because each jog is a vertical wire segment that may block the wirings of other nets. Also, jogging may generate additional contacts

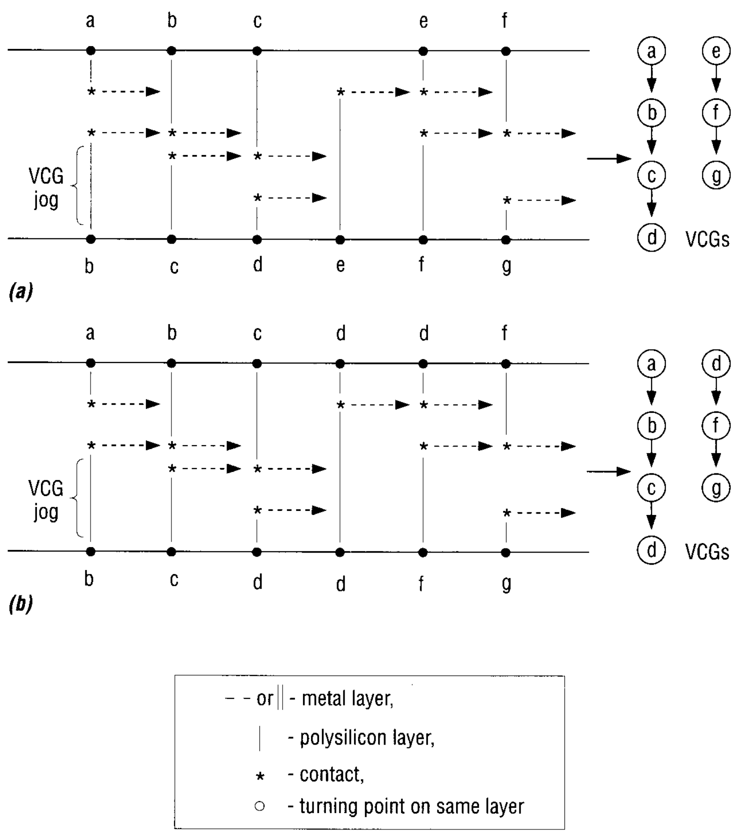


Figure 11. *Conditions A and B for vertical constraint jogs and graphs.*

The most important idea in our approach is the dynamic partitioning of the routing region.

where J is the initial value of minimun jogging length set by the user and w_5 and w_6 are the weights. Typical values for w_5 and w_6 are 1.5 and 3.0.

A SPECIAL CASE

Suppose we have the condition in Figure 12, in which a target column contains split nets. How are the left-right interconnections of the split nets satisfied? That is, how do they fan out to the multiple targets? Figure 13 shows a simple, yet effective algorithm to solve this problem.

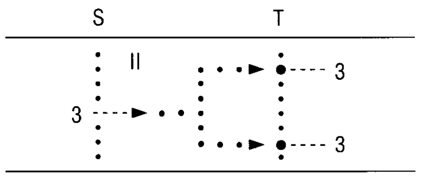


Figure 12. *Split-net fanout for net 3.*

```
while (split nets on the target_column exist) {
    shift sparse region one column toward the
        start_column to construct a new
        sparse region and new target_column;
    bring top and bottom terminals into the
        nearest available tracks;
    join as many split nets on the old
        target_column as possible;
    execute jogging operation on the old
        target_column;
}
```

Figure 13. *An algorithm to solve routing when split nets are on the target column.*

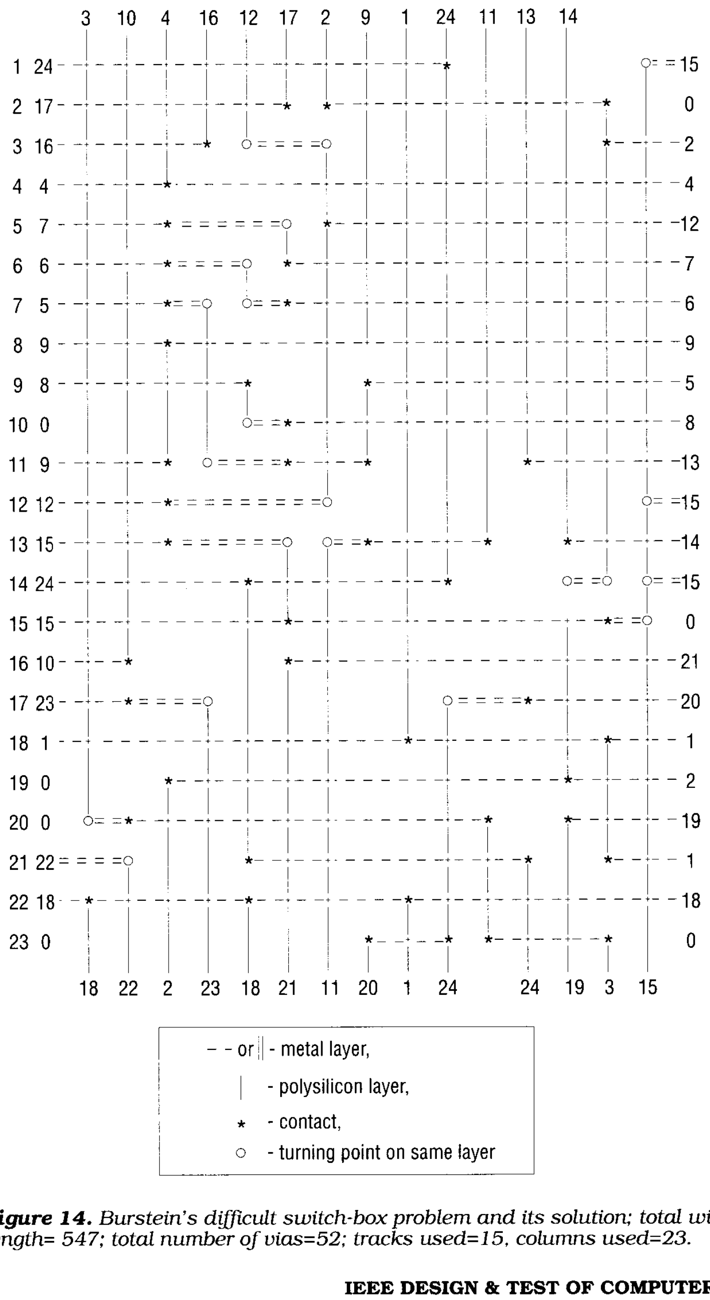


Figure 14. *Burstein's difficult switch-box problem and its solution; total wire length= 547; total number of vias=52; tracks used=15, columns used=23.*

EXPERIMENTAL RESULTS

The algorithm is in C code and runs on a VAX 11/780 under the VMS operating system. We have run it on a number of examples and the program gives us good routing solutions. Burstein and Pelavin[4] provide a difficult problem for switch-box routing, in which they attempt to wire the switch box manually. The result has three unconnected nets, while their hierarchical router left one net undone (net 24 was blocked). Figure 14 shows this complicated problem, and the solution we got. The CPU time was 0.98 second. Table 1 shows a more detailed analysis of the routing of this example.[8-11,22] Although our result for this example is worse than that of Weaver[22], Weaver's execution time of 1,390 seconds was quite a bit more than ours.

Smith et al.[7] provide a more difficult test. Their LRS router uses an area 49 tracks square, while Stenstrom's router[12] uses 42x43 tracks for the same example. Our router uses only 38x40 tracks, as Figure 15 shows. We essentially reduced the area by 37% over LRS and by 16% over Stenstrom's router. Moreover, the CPU time for this example was 6.74 seconds.

We also extended our router to route Deutsch's difficult channel example,[5] in which the maximum routing density is 19. We produced a 19-track solution as did Burstein's hierarchical router. Our routing solution, which took 10.67 seconds of CPU time, is given in Figure 16.[13] Table 2 shows a comparison of routers for this example.[2-4,6,9]

Table 1. *Router comparison for Burstein's difficult switch-box example.*

Router	Rows	Vias	Net Length
Our router	15	52	547
Weaver	15	41	531
Luk	16	58	577
M. Sadowska	15	59	561
Hamachi	15	67	564
Hsieh	15	63	567

Table 2. *Router comparison for Deutsch's difficult example.*

Router	Rows	Vias	Net Length
Our router	19	376	5,058
YACR2	19	287	5,020
Hamachi	20	412	5,302
Burstein	19	354	5,023
Yoshimura	20	308	5,075
Rivest	20	403	5,381

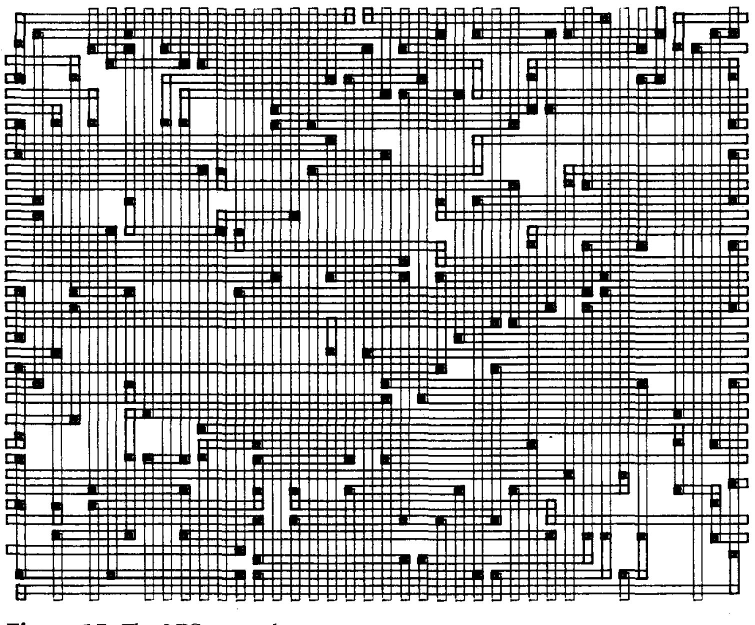


Figure 15. *The LRS example.*

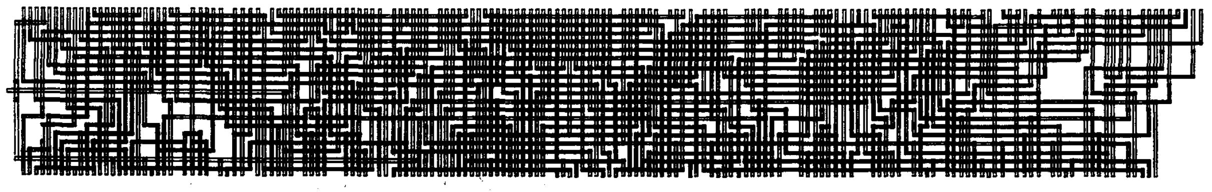


Figure 16. *Deutsch's difficult channel example.*

We have presented a new heuristic algorithm and implementation of a switch-box router for VLSI layout, which has produced good results. Several concepts are important to our approach, including congestion, VCG jog, and dynamic rules, but the most important idea is the dynamic partitioning of the routing region. Interconnecting wires of the same nets routed from left and right concurrently greatly increases routability. We are in the process of extending this router to multilevel metal interconnections[23] and irregular routing shapes.

ACKNOWLEDGMENTS

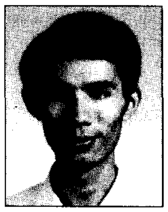
We thank the reviewers, in particular reviewer B, for their painstaking effort in reviewing the manuscript and providing many useful comments and helpful suggestions.

This research was supported in part by the National Science Council, Republic of China, under grant NSC-74-0201-E006-04.

REFERENCES

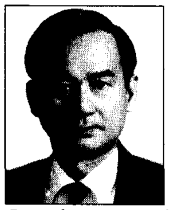
1. J. Soukup, "Circuit Layout," *Proc. IEEE,* Vol. 69, No. 10, 1981, pp. 1282-1304.
2. R. Rivest and C. Fiduccia, "A 'Greedy' Channel Router," *Computer-Aided Design,* Vol. 15, No. 3, 1983, pp. 135-140.
3. T. Yoshimura and E. Kuh, "Efficient Algorithms for Channel Routing," *IEEE Trans. on CAD,* Vol. CAD-1, No. 1, Jan. 1982, pp. 25-35.
4. M. Burstein and R. Pelavin, "Hierarchical Wire Routing," *IEEE Trans. Computer-Aided Design,* Vol. CAD-2, No. 4, Oct. 1983, pp. 223-234.
5. D. Deutsch, "A Dogleg Channel Router," *Proc. Design Automation Conf.,* 1976, pp. 425-433.
6. J. Reed, A. Sangiovanni-Vincentelli, and M. Santomauro, "A New Symbolic Channel Router: YACR2," *IEEE Trans. Computer-Aided Design,* Vol. CAD-4, No. 3, July 1985, pp. 208-219.
7. L.R. Smith et al., "A New Area Router, the LRS Algorithm," *Proc. Int'l. Conf. on Circuits and Computers,* 1982, pp. 256-259.
8. W.K. Luk, *A Greedy Switch-Box Router,* VLSI doc. V158, CS Dept., Carnegie Mellon Univ., Pittsburgh, Penn., 1984.
9. G. Hamachi and J. Ousterhout, "A Switch-Box Router with Obstacle Avoidance," *Proc. Design Automation Conf.,* 1984, pp. 173-179.
10. M. Marek-Sadowska, "Two-Dimensional Router for Double Layer Layout," *Proc. Design Automation Conf.,* 1985, pp. 117-123.
11. Y. Hsieh and C. Chang, "A Modified Detour Router," *Proc. Int'l. Conf. on Computer-Aided Design,* 1985, pp. 301-303.
12. J. Stenstrom and R. Mattheyses, "Switch-Box Routing the Greedy Way," *Proc. Int'l. Conf. on Computer-Aided Design,* 1985, pp. 307-309.
13. J. Jou et al., "A New Dynamic Adaptive Generalized Router: DAPRT," *Proc. Int'l. Symp. Circuits and Systems,* 1986, pp. 303-306.
14. C.P. Hsu, "A New Two-Dimensional Routing Algorithm," *Proc. Design Automation Conf.,* 1982, pp. 46-50.
15. T. Szymanski, "Dogleg Channel Routing Is NP-Complete," *IEEE Trans. Computer-Aided Design,* Vol. CAD-4, No. 1, Jan. 1985, pp. 31-41.

16. A. LaPaugh, *Algorithm for Integrated Circuit Layout: An Analytic Approach*, rpt. TR-248, MIT Lab. for Computer Science, Mass. Inst. of Tech., Cambridge, Mass., 1980.
17. M. Garey and D. Johnson, *Computers and Intractability:A Guide to The Theory of NP-Completeness*, Freeman, San Francisco, 1979.
18. C. Lee, "An Algorithm for Path Connections and Its Applications," *IRE Trans. Electronic Computers*, Vol. EC-10, Sept. 1961, pp. 346-365.
19. F. Rubin, "The Lee Connection Algorithm," *IEEE Trans. Computers*, Vol. C-23, 1974, pp. 907-914.
20. D. Hightower, "A Solution to the Line Routing Problem on the Continuous Plane," *Proc. Design Automation Workshop*, 1969, pp. 1-24.
21. N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, Prentice Hall, Englewood Cliffs, N.J., 1974.
22. R. Joobbani and D. Siewiorek, "Weaver: A Knowledge-Based Routing Expert," *Proc. Design Automation Conf.*, 1985, pp. 266-271.
23. J. Jou et al., "An Efficient 2-Or-3 Layer Channel Router for VLSI Layout," *Proc. Int'l. Symp. Circuits and Systems*, 1987, pp. 47-50.



Jer Min Jou is an instructor in the Department of Electrical Engineering at National Cheng Kung University, Tainan, Taiwan, ROC, where his research interests are in VLSI CAD, computer architecture, computer algorithms, and graph theory. He hols a BS in engineering science, an MS in electrical engineering, and a PhD from the National Cheng Kung University. He is a member of the IEEE Computer Society.

Jhing Fa Wang is an associate professor of electrical engineering at National Cheng Kung University, where his research interests are CAD tools for VLSI, graph theory, circuit layout and extraction systems, coding theory, and VLSI implementations. He holds a PhD in electrical engineering and computer science from the Stevens Institute of Technology in Hoboken, New Jersey.



Jau Yien Lee is chairman of the Department of Electrical Engineering at National Cheng Kung University, where his research interests are CAD/VLSI, signal processing, and thin-film circuitry. Previously, he did post doctorate work at San Jose State University in California and taught radio electronics in the ROC Army Signal School. He holds a BS, an MS, and a PhD in electrical engineering from National Cheng Kung University. He is a member of the AMSE, ICEE, and Phi Tau Phi.

Send questions or comments on this article to J. Jou, National Cheng Kung University, Dept. of Electrical Engineering, 1 University Rd., Tainan, Taiwan, ROC.