CARNEGIE-MELLON UNIVERSITY

Computer Science Department

A Greedy Switch-box Router

W. K. LUK

May 1984

VLSI Document V158

Keywords and index categories: automatic routing, greedy algorithm, switch-box router, VLSI cad

Supported in part by the Defense Advanced Research Projects Agency, Department of Defense, ARPA Order 3597, monitored by the Air Force Avionics Laboratory under contract F33615-81-K-1539. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

A Greedy Switch-box Router[1]

W. K. LUK
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract: The greedy channel router of Rivest and Fiduccia is extended into an efficient switch-box router. The algorithm is based on two simple operations called join-split-nets and jog-to-right-target derived from the channel router. Terminals are on the boundary of a rectangular region, and the router uses two orthogonal layers of wires to generate the solution. The router always succeeds in finding a solution by inserting sufficient horizontal and vertical tracks in case of insufficient routing area. The result is generated through a single column-wise scan across the routing region. The expected running time is proportional to $M(N+N_{net})$, where M,N and N_{net} are respectively the number of columns, rows and nets in the region. The scan direction is crucial to the algorithm and we have proposed good heuristic which is based on the augmented channel density distribution in finding it. Results from a number of examples are evaluated. The implemented router is designed for assembling custom VLSI designs, it works in parallel with other tools such as a layout editor via a simple interface. The router output is in CIF.

Keywords: automatic routing, greedy algorithm, switch-box router, VLSI cad

1. Introduction

Placement and routing is an important part of design automation, both for printed circuit board and chip level layout [5]. The problem is: given a set of modules and interconnection information (net-list), how to place the modules and connect the terminals of the modules in an optimal way, namely minimum amount of layout area, shortest overall wiring length, etc. The process is generally divided into a number steps: placement of modules, creation and partition of routing region between the modules, global (or rough) assignment of the wiring paths for each net, detailed wiring of the individual routing regions. This report presents a specific tool for handling detailed routing: a *switch-box router.*

A channel router, switch-box router and river router form a set of routers that are sufficient to handle the detail routing in a placement and routing system [5]. Switch-box routing may not be necessary for gate-array routing, where the fixed terminals are only on two opposing sides of a rectangular routing region. But for custom VLSI layout, it is not uncommon to have routing regions where terminals are located on all four sides. As there are no known polynomial-time optimal channel and switch-box routing algorithms, and also no

[1]Supported in part by the Defense Advanced Research Projects Agency, Department of Defense, ARPA Order 3597, monitored by the Air Force Avionics Laboratory under contract F33615-81-K-1539. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

known algorithm to determine the routability for switch-box routing, a solution is usually based on a heuristic.

We present a fast heuristic to handle switch-box routing, based on an extension of the greedy heuristic for channel routing proposed by Rivest and Fiduccia [4]. The switch-box algorithm is as time-efficient as the greedy channel router which has been found to be useful in VLSI layout. The switch-box router has been implemented and can work integrally and in parallel with a VLSI layout tool such as CAESAR for generating the interconnection for cells, all the way down to the mask level description (CIF).

The report first presents the greedy switch-box algorithm, then discusses some features about its implementation and examples.

2. The greedy switch-box algorithm

We assume the placed modules are rectangular and their terminals are on the module boundaries, further the modules are aligned in such a way that they enclose a rectangular routing region.

2.1. Definition

A rectangular routing region is defined as $R=\{0,1,2,....,M\}\times\{0,1,2,....,N\}$, where M,N are positive integers. Each pair $(x,y)\in R$ is called a grid point. A set of grid points $col(x)=\{(x,y)|y\in\{0,1,2,....,N\}\}$ for $x=0,1,2,....,M$ is called a column. Similarly a set of grid points $row(y)=\{(x,y)|x\in\{0,1,2,....,M\}\}$ for $y=0,1,2,....,N$ is called a row. The two columns $col(0)$ and $col(M)$ forms the left ($LEFT$) and right ($RIGHT$) boundaries of the routing region, likewise the two rows $row(0)$ and $row(N)$ are the bottom ($BOTTOM$) and top (TOP) boundaries.

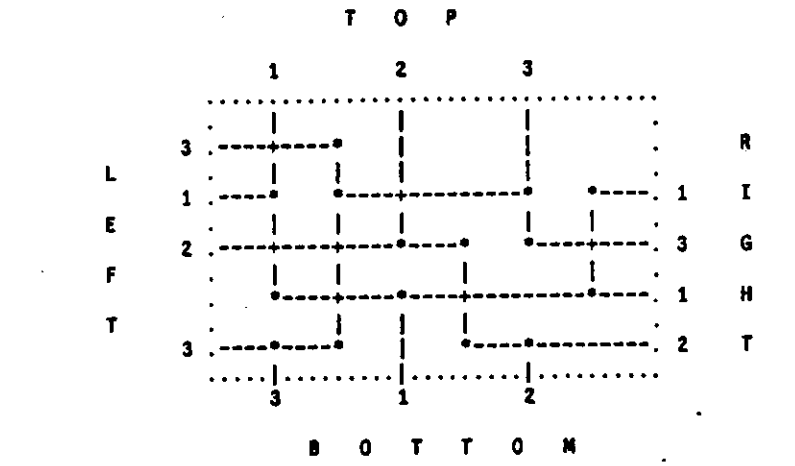
The terminals from the modules are located on the four boundaries of the routing region R. Each terminal is related to an integer n called the net. Each net n specifies the terminals on the four boundaries that are to be connected together, i.e. all the terminals that bear the same net number will eventually be connected by the router. The set of nets specifying the entire connectivity of the terminals is called a net-list. Without loss of generality, we may assume the net-list of the routing problem is a set of integers $\{1,2,....,N_{net}\}$.

The connectivity and location of each terminal is represented as $LEFT(i)=n$ or $RIGHT(i)=n$ or $TOP(i)=n$ or $BOTTOM(i)=n$, depending which edge the terminal is on, where i stands for the coordinate of the terminal along the edge and n stands for the net number.

The routing problem is to find a solution for connecting all the terminals that belong to the same net within the given routing region. Connection is defined by *wire* (or *track*) which is allowed to run either horizontally or vertically along the rows and columns (along the grids). Only a single wire is allowed to

occupy each row and column segment, and cross-over of two wires (say a horizontal and a vertical wire) at a grid point is allowed.

An example is shown in Figure 2-1.



$R = \{0,1,2,3,4,5,6,7\} \times \{0,1,2,3,4,5,6\}$

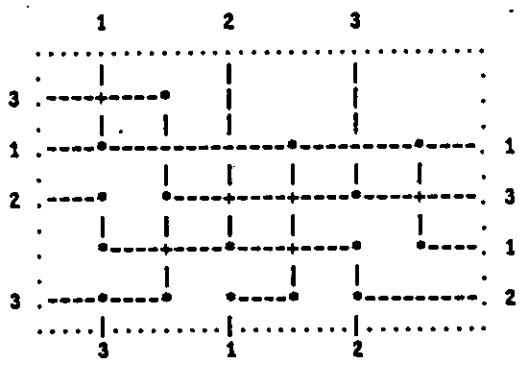
$TOP(1,2,3,4,5,6) = [1,0,2,0,3,0]$

$BOTTOM(1,2,3,4,5,6) = [3,0,1,0,2,0]$

$LEFT(1,2,3,4,5) = [3,0,2,1,3]$

$RIGHT(1,2,3,4,5) = [2,1,3,1,0]$

(a)



(b)

Figure 2-1: A switch-box routing problem:
(a) specification and a solution
(b) solution from this switch-box router

2.2. Greedy channel routing algorithm

Since optimal channel routing is NP-complete [2], and determining routability for switch-box has no known solution, we rely on finding efficient heuristics. The switch-box routing heuristic is an extension of the greedy channel routing algorithm [4].

2.2.1. Optimality

In channel routing, we are interested in finding a routing solution which uses as few horizontal tracks as possible. A common lower bound for this solution is the channel density[2]. Our heuristic is based on the following reasoning:

Optimality can be achieved if we can guarantee for each column, there is only *one* horizontal track for each net. The routing heuristic is to minimize the number of horizontal tracks per column per net. The method is to scan columns, say from left to right, and try to join the split horizontal tracks (assuming there are any) that belong to the same net as much as possible.

2.2.2. The greedy algorithm

Without loss of generality, we scan the routing region columns from left to right. The control structure of the greedy heuristic can be formulated as follows. Assume the routing region is $R=\{0,1,2,....,M\}\times\{0,1,2,....,N\}$.

```
(CH0)  calculate channel density;
       insert horizontal rows equal to the channel density
          into the initial routing channel;
loop for i from 1 to M-1 do
     (CH1)  if empty-track-exists than
               bring TOP(i) and BOTTOM(i) into empty rows;
     (CH2)  join split nets as much as possible;
     (CH3)  bring split nets closer by jogging;
     (CH4)  solve conflict by jogging to the next top/bottom terminals;
     (CH5)  if step (CH1) failed then
            increase number of rows;
            repeat (CH1); update columns 1 to i;
while split-net-exists do
     (CH6)  increase number of columns by 1;
            join split nets as much as possible;
```

For details about the greedy channel routing algorithm, see [4]. We review briefly the major control steps:

- Step CH1 (**bring-in-top-bottom**): Bring *TOP*(*i*) and *BOTTOM*(*i*) into empty rows to start the

[2] A lower bound on the number of horizontal tracks required for channel routing. Assume the columns are numbered from 0 to M and the terminals are lying along the *TOP* and *BOTTOM* edges at some of the integer points 1,2,....,$M-1$. Column 0 and column M are the left and right boundaries of the channel. Let d_i be the minimum number of horizontal tracks that pass through the column i, in order to maintain the connectivity between the terminals on the left, on the right, and the current column i. *Channel density* D is defined as $max\{d_1,d_2,....,d_{M-1}\}$. The tuple $(d_1,d_2,....,d_{M-1})$ is called the density distribution d and can be used as a measure for the sparseness of a channel.

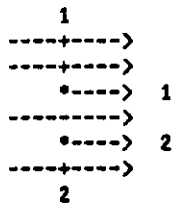


Figure 2-2: Net 1 enters from the top and net 2 from the bottom

routing for each column. See Figure 2-2.

- Step CH2 (**join-split-nets**): It is the key step for the heuristic which joins split nets as much as possible. There are priorities in choosing the nets to be joined, but such rules are by no means unique. Naturally, split nets are joined according to the following priority: a split net that when joined can free up the most number of tracks, a split net with tracks that are farther apart, ... See Figure 2-3.

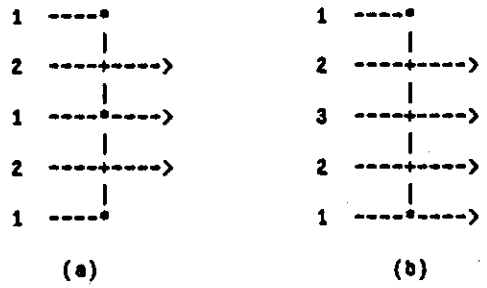


Figure 2-3: Joining split nets: (a) join net 1 to free more tracks,
(b) join net 1 because tracks farther apart

- Step CH3 (**jog-for-join**): Split nets are brought closer by jogging so that they may be more easily joined together. Because the farther apart the split net tracks are, the higher the chance of being blocked by rows in between. This is not a compulsory step for the heuristic, but making use of it may improve the routing result. See Figure 2-4.

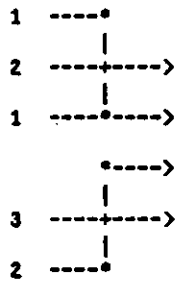


Figure 2-4: Bring split net closer by jogging

- Step CH4 (**jog-bypass-conflict**): This is a means of solving the cyclic conflict by jogging a track of a certain net to its next top or bottom terminals, see Figure 2-5c. This step may also be considered as an optional one, since cyclic conflict may also be handled solely by step CH2, see Figure 2-5b. But making use of it may improve the routing result.

- Step CH5 (**extend-row**): In case the routing channel does not have sufficient rows, the router

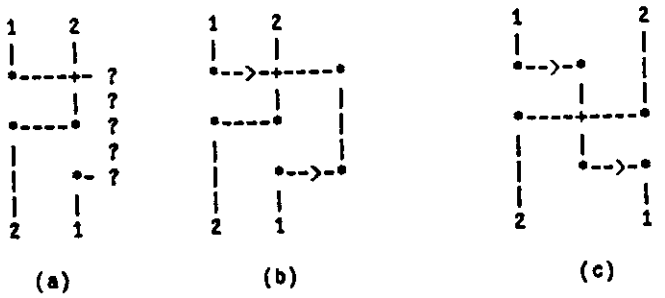


Figure 2-5: (a) Cyclic conflict between net 1 and 2,
(b) Solving by joining, (c) Solving by jogging

increases the tracks by 1 or 2 to allow the top and bottom terminals to enter and continue the routing.

- Step CH6 (**extend-column**): In case the routing reaches the far right edge and there are still some split nets, the router extends the routing region beyond the right-most column to join up all the remaining horizontal tracks.

The step join-split-nets (CH2) is the *key* for the greedy channel router to function, whereas jog-for-join (CH3) and jog-bypass-conflict (CH4) are mainly for getting better results. This statement may be illustrated from the following tests on the Deutsch channel routing problem. This problem has a channel density of 19.

test	nb. of tracks used
all CH2, CH3, CH4 used	20
without CH3	22
without CH4	27
without CH3 and CH4	28

2.3. Extension to switch-box routing

We extend the greedy heuristic to handle switch-box routing by relaxing some operations that are not vital to the functioning of the greedy algorithm and modifying them to overcome the additional constraints on switch-box routing. Assuming the scanning is from left to right, these additional constraints are:

- to match the terminals on the *LEFT* of the routing region
- to match the terminals on the *RIGHT* of the routing region

To overcome the constraint, we use the following heuristic:

- **Bring in left terminals:** The left edge terminals enter directly into the routing region (column 1) as horizontal tracks.
- **Jog to right target (jog_R):** Instead of jogging to the next top and bottom terminals as in Step CH4 of the greedy channel router which we call $jog_{T/B}$, the horizontal tracks are jogged to a *target row*, a row where a right edge terminal is located. We call this step jog_R. The following strategies are used:

- All nets that have right edge terminals are put into a *priority queue*. The choice of the jogging nets is based on the following priority: First choose a net whose target row on the right is empty and for which there also exists a vertical track from the net to a target row. Second jog a net whose target row on the right is empty with priority also based on how close to the empty target row it can be jogged. Third jog a net that can be brought closer to the target row. When a net reaches its target row, it is deleted from the queue. See Figure 2-6. In case there are more than one nets satisfying each of the above conditions, higher priority is given to the one whose initial position is farther away from the target row, and then the one with a closer final position from the target row.

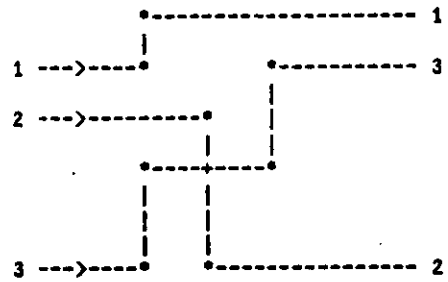


Figure 2-6: Net 1, 2 and 3 jog to the right side target terminals

- In order to avoid the *deadlock* condition when jogging to the right side targets (see Figure 2-7), a net is allowed to jog as close as possible or pass the target row so as to destroy the deadlock, and then such net is masked so that it will not be allowed to oscillate back to its original row. Other deadlocked nets can in turn be jogged to the target rows.

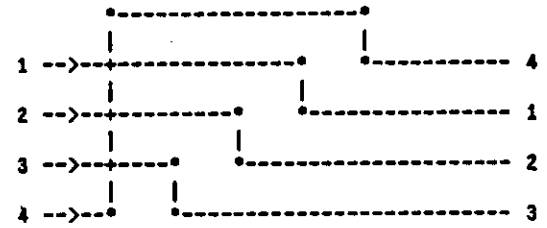


Figure 2-7: Cyclic condition when jogging to the right side target terminals, Net 4 breaks the deadlock

We have shown that the jog-to-right-target is the *basic* operation for (1) bringing nets to their final targets, (2) handling of cyclic conflict and deadlock between the left and right edge terminals. It is as vital as the join-split-nets operation in channel routing. The combination of both of them enables switch-box routing.

The optimal way for a net to arrive at its target row is by jogging only *once* as shown in nets 1 and 2 in Figure 2-6, and nets 1, 2 and 3 in Figure 2-7. Each of them uses only a single vertical track, whereas the other nets in the figures require two or more. Since each jogging wastes one vertical track, too many joggings may result in running out of tracks. A *distance dependent threshold scheme* is used to avoid excessive jogging. A net is only allowed to jog to its target row only if it can be brought to or beyond half way between the initial and the target positions.

- **Fanout to targets:** For nets that occupy more than one location on the right edge, when such net

becomes *unsplit* and is *close* to the right edge, fan out the horizontal track to the final terminal locations. We call this step fanout to target. See Figure 2-8. *Close* is a control parameter in the algorithm, it depends on (1) the free space (sparseness) of the vertical and horizontal tracks near the right edge, (2) the number multiple terminals (terminals that belong to the same net) on the right edge, and (3) the number of columns from the right edge. A typical default value is between 2 to 5 columns from the right edge. The augmented density distribution d' (Section 2.5.1) $(d_0,d_1,....d_M)$ may be used for measuring sparseness. Fanout operation is started when all the d_i's to the right of the scan column is below a certain threshold.

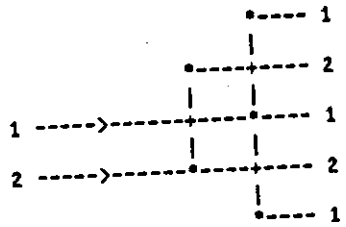


Figure 2-8: Net 1 and 2 fan out to the right side target terminals

Steps **bring-in-left-terminals** and **fanout-to-targets** are orthogonal to the greedy channel routing algorithm, they can be inserted directly into the channel routing algorithm. Step $\mathbf{jog_R}$ conflicts with the steps CH3 and CH4 of the channel router, we next show how to handle the problem and give the algorithm for switch-box routing.

2.3.1. Conflict between $\mathbf{jog_{T/B}}$ and $\mathbf{jog_R}$

As mentioned earlier, the switch-box router besides jogging to the top/bottom terminals (Step CH4), allows nets that have right edge terminals to jog to the right edge targets. We use two kinds of jogging in switch-box routing:

- $\mathbf{jog_{T/B}}$: jog to the next top or bottom terminals as in the channel routing algorithm
- $\mathbf{jog_R}$: jog to the right edge terminals

Figure 2-9 shows a simple example (Example 1) of how the two joggings conflict with each other and we show how the joggings are handled.

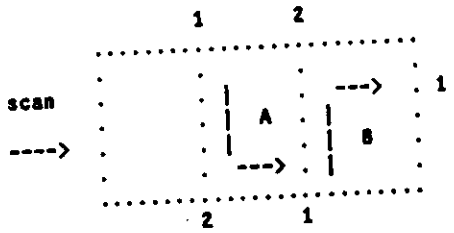


Figure 2-9: Example 1: The two joggings conflicting each other

We assume the router scans from left to right. In region A, net 1 has to jog downward to solve the conflict

between net 1 and 2 as in the channel router. In region B, net 1 has to jog upward to meet the right edge target. We illustrate a number of possible jogging combinations for the generation of the solution. They are

- **SWJOG1 (jog_R)**: For nets that have terminals on the *RIGHT*, perform jog_R until the net has occupied a track that matches with one of the right edge terminal positions. For nets that only have terminals on *TOP* and *BOTTOM*, perform $jog_{T/B}$ as in the greedy channel router. The result for Example 1 is shown in Figure 2-10.

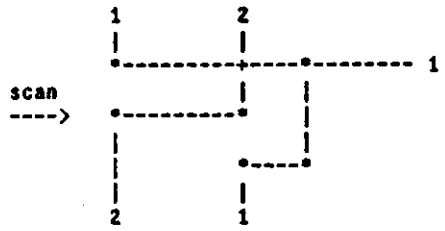


Figure 2-10: Routing Example 1 using jogging jog_R

- **SWJOG2 ($jog_{T/B}$; jog_R)**: Another jogging scheme is to first perform $jog_{T/B}$ for every net and then switch to perform jog_R at the column where the last top and/or bottom terminals appear. The result for Example 1 is shown in Figure 2-11.

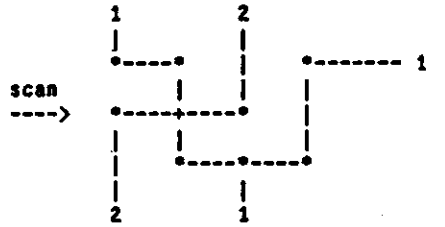


Figure 2-11: Routing Example 1 using jogging ($jog_{T/B}$; jog_R)

- **SWJOG3 ($jog_{T/B} \parallel jog_R$)**: Figure 2-12 shows another routing for Example 1. It applies both kind of joggings to the same net (net 1 in this case) in *parallel*.

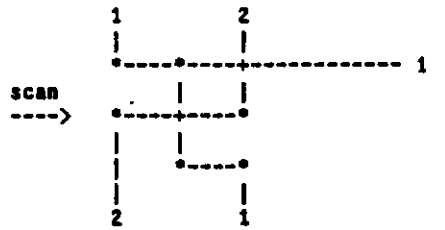


Figure 2-12: Routing Example 1 using parallel jogging ($jog_{T/B} \parallel jog_R$)

Net 1 in the Figure 2-12 tends to jog downward to solve the cyclic conflict (between net 1 and 2) but its right side terminal tends to make it stay upward towards the target. We should not apply both joggings $jog_{T/B}$ and jog_R to each column, otherwise it would lead to *fanout* of horizontal tracks and contradict the basic greedy heuristic – minimize the number of horizontal tracks per column. Consequently these fanout nets have to be joined and results in oscillation between the fanning out and joining operations. Parallel jogging is basically a different heuristic for switch-

box routing and cannot be incorporated into the original greedy heuristic.

Careful observation reveals that the parallel jogging which leads to a fanout net can be obtained by scanning the routing region from right to left using the scheme SWJOG1 (jog_R) (Figure 2-13). This shows that the single jogging rule jog_R can give a good routing if a suitable scanning direction has been chosen. So the routing result depends on the scan direction as is the case for the greedy channel router.

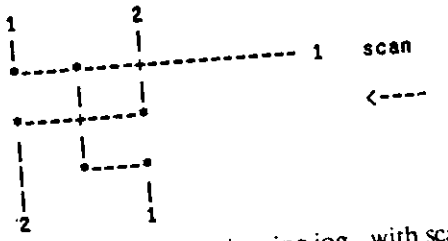


Figure 2-13: Routing Example 1 using jogging jog_R with scan reversed, same result as ($jog_{T/B}$ || jog_R)

2.3.2. The jogging strategy for switch-box routing SWJOG

SWJOG3 ($jog_{T/B}$ || jog_R) contradicts the original greedy idea of trying to minimize the number of horizontal tracks by joining split nets. One has to keep track of the fanning out (due to the parallel jogging) and joining operations. This complicates the algorithm though it is less sensitive to the scan direction. The same result can be obtained if the scanning direction is reversed. In other words, the choice of the scan direction is more *crucial*. We present in the sequel a heuristic to determine the scan direction.

SWJOG1 (jog_R) performs better than SWJOG2 ($jog_{T/B}$;jog_R) because jog_R handles the right side constraint of the switch-box directly and earlier. Though ($jog_{T/B}$;jog_R) may sometime use less horizontal tracks than jog_R since it always jogs to the top and bottom terminals to handle cyclic conflict, we think that the most important constraint for switch-box routing is the right side terminals which should not be handled too late in the scan which may result in running out of room to jog to the right side terminals.

To improve the use of $jog_{T/B}$ and jog_R for nets that have right side terminals, we may make the following compromise between SWJOG1 (jog_R) and SWJOG2 ($jog_{T/B}$;jog_R). We call such rules **SWJOG**:

- For nets that do not have right side terminals, always perform $jog_{T/B}$.
- For nets that have a right side terminal and whose right-most top/bottom terminal is on the right p-portion of the routing region, perform jog_R rather than $jog_{T/B}$ for that net (see Figure 2-14a).
- For nets that have a right side terminal and whose right-most top/bottom terminal is on the left p-portion of the routing region, perform ($jog_{T/B}$;jog_R), i.e. $jog_{T/B}$ before the last top/bottom terminal and jog_R at and after the last top/bottom terminal (see Figure 2-14b).
- p may vary between 0 and 1. If p=0, it performs jog_R. If p=0, it performs ($jog_{T/B}$;jog_R). A

typical value for p is 0.5.

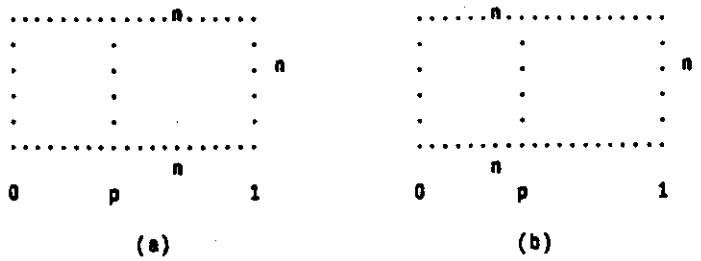


Figure 2-14: The modified jogging SWJOG for switch-box routing: (a) jog_R for net n, (b) ($jog_{T/B}$;jog_R) for net n

2.4. The switch-box routing algorithm

Based on the extended jogging strategies related to the switch-box routing, the general structure of the switch-box routing algorithm is as follows. Assume the routing region is $R=\{0,1,2,....,M\}\times\{0,1,2,....,N\}$.

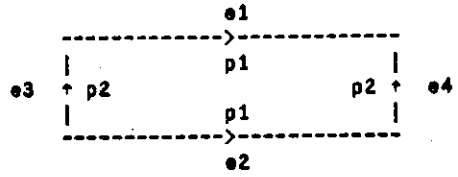
```
(SW0)  determine scan direction;
       bring in LEFT terminals into column 1;
loop for i from 1 to M-1 do
     (SW1)  if empty-track-exists then
               bring TOP(i) and BOTTOM(i) into empty rows;
     (SW2)  join split nets as much as possible;
     (SW3a) for net-with-no-right-terminals do
               bring split nets closer by jogging;
     (SW3b) for net-with-right-terminals do
               SWJOG;
     (SW4)  when close-to-right-edge do
               fanout to targets;
     (SW5)  if step (SW1) failed then
            increase number of rows;
            repeat (SW1); update columns 1 to i;
while split-net-exists do
     (SW6)  increase number of columns by 1;
            join split nets as much as possible;
```

The algorithm terminates in a single column-wise scan across the routing region and always succeeds in finding a solution by inserting enough horizontal and vertical tracks (steps SW5 and SW6). Next we estimate the expected running time for finding a solution, in terms of M,N,N_{net}, the total number of columns, rows and nets. The running time t_0 for step SW0 is proportional to $M+N+N_{net}$, since it is the time to determine the augmented density distribution d^+. At each column, step SW1 requires a time t_1 proportional to N, the column height. The complexity to carry out steps SW2, SW3a, SW3b and SW4 depends on the search for priority to join split nets, and priority to jog and fan out nets to the target rows. In general, it is a function of N_{net} and N, inefficient heuristic may lead to exponential searching time. In practice, the different combination of split nets and nets that can be jogged to targets are small, so the expected time $t_{2,3,4}$ is basically proportional

to $N+N_{net}$. The overall time for steps SW1 to SW4 to scan through the M columns is proportional to $M(t_1+t_{2,3,4})=M(N+N_{net})$. The time required to insert an extra horizontal track (step SW5) is proportional to MN. The time required to insert an extra column (step SW6) is proportional to $N+N_{net}$. In practice, the extra rows and columns needed to complete a routing is a small constant. So the overall expected running time is proportional to $M(N+N_{net})$.

2.5. Scan direction

Determination of the scan direction is equivalent to the assignment of the *LEFT* edge to one of the four edges e_1, e_2, e_3 and e_4 of the routing region. The scan direction is determined in two steps:



- Step 1: Locate the *TOP-BOTTOM* and *LEFT-RIGHT* pairs.
- Step 2: Then separate out the *LEFT* and *RIGHT* edges.

2.5.1. Augmented channel density

First we need the following measures in order to determine the scan direction.

Given a switch-box routing problem, we call the two pair of opposite edges respectively $\mathbf{p}_1$ and $\mathbf{p}_2$. Take

$$\mathbf{p}_1=\{\mathbf{e}_1,\mathbf{e}_2\} \text{ and } \mathbf{p}_2=\{\mathbf{e}_3,\mathbf{e}_4\}$$

Let the number of tracks for $\mathbf{p}_1$ and $\mathbf{p}_2$ be respectively t_1 and t_2, these two numbers are $M-1$ and $N-1$ according to the definition for a routing region R.

The *augmented channel density* D_1^+ of $\mathbf{p}_1$ is defined as the density D_1 which is the overall minimum number of tracks required to maintain the connectivity of the terminals on two opposite edges $\mathbf{p}_1$ ($\mathbf{e}_1$ and $\mathbf{e}_2$) as in a channel routing problem, *plus* the tracks required to connect the nets on the other two edges $\mathbf{p}_2$ ($\mathbf{e}_3$ and $\mathbf{e}_4$) with those nets on $\mathbf{p}_1$. Assuming $\mathbf{p}_1$ is the top and bottom pair. One can view it as the case of an infinitely long channel, with the nets belonging to the edges $\mathbf{e}_3$ ($\mathbf{e}_4$) span from the left (right) infinity into the finite channel through the left (right) edge. The augmented channel density D_2^+ of $\mathbf{p}_2$ is similarly defined.

Using the same notation as for the channel density, d_0 (respectively d_M) stands for the minimum number of tracks required to bring all the nets on $\mathbf{e}_3$ ($\mathbf{e}_4$) edge into the routing region through the left-most (right-most) column. The augmented density distribution $\boldsymbol{d}^+$ is

$$\boldsymbol{d}^+=(d_0,\boldsymbol{d},d_M)=(d_0,d_1,\ldots,d_{M-1},d_M)$$

The *track availability* or *tracks to augmented channel density ratio* s_i, $i=1,2$ for the edge pair $\mathbf{p}_1$ and $\mathbf{p}_2$ is defined as

$$s_1 = t_1 / D_2^+ \quad \text{and} \quad s_2 = t_2 / D_1^+$$

Example: Refer to the example in Section 3.4.1, let $\mathbf{p}_1$ be the edge pair $\mathbf{e}_1$ and $\mathbf{e}_2$, and $\mathbf{p}_2$ be the edge pair $\mathbf{e}_3$ and $\mathbf{e}_4$.

The augmented density distribution for $\mathbf{p}_1$ and $\mathbf{p}_2$ are

$$d_1^+ = (13,14,14,13,12,12,13,13,12,13,12,11,9,8,7,7,8,9,9,9,10,11,11,11,11)$$

$$d_2^+ = (16,16,17,17,16,16,14,15,15,16,16,16,16,16,17,17,16,16)$$

Each number represents the minimum number of tracks required at each column of the channel. The augmented channel density D^+ is the maximum of each set of numbers.

$$t_1 = 23$$

$$t_2 = 16$$

$$D_1^+ = \textit{augmented-density}(\mathbf{e}_1,\mathbf{e}_2) = 14$$

$$D_2^+ = \textit{augmented-density}(\mathbf{e}_3,\mathbf{e}_4) = 17$$

$$s_1 = t_1 / D_2^+ = 23/17 = 1.35$$

$$s_2 = t_2 / D_1^+ = 16/14 = 1.14$$

□

2.5.2. Finding the scan direction

We first describe how to locate the *TOP-BOTTOM* and *LEFT-RIGHT* pair. It is based on a property of the greedy heuristic which tends to minimize the number of tracks that are perpendicular to the scan direction. Given a switch-box routing problem, it is natural to assign the *LEFT-RIGHT* pair as one that has a smaller number of *track availability* or *tracks to augmented channel density ratio s*.

So the *LEFT-RIGHT* pair is chosen according to:

$$\textit{LEFT-RIGHT} = \begin{cases} \mathbf{p}_1 & \text{if } s_1 < s_2 \\ \mathbf{p}_2 & \text{if } s_2 \le s_1 \end{cases}$$

And the *TOP-BOTTOM* is chosen as the remaining pair of edges.

Example: In the above example, we should choose the *LEFT-RIGHT* pair as $\mathbf{p}_2 = (\mathbf{e}_3,\mathbf{e}_4)$, and the

TOP-BOTTOM pair as $p_1=(e_1,e_2)$. So the scan direction should be parallel to the edges e_1 and e_2. □

The starting *LEFT* edge for scanning is decided between the two remaining opposing *LEFT-RIGHT* edges based on the following rules (see Figure 2-15):

- Rule 1: The *LEFT* edge should be chosen from the edge which has more terminals and especially multiple terminals (terminals that belong to the same net and appear more than once on the same edge). This poses less of burden on the jog_R and fanout operations.
- Rule 2: The *RIGHT* edge should be close to the region which is potentially less populated with wires (tracks). This implies more free vertical tracks for the router to join the split nets and fan out to target terminals. The track sparseness can be measured by the augmented density distribution.

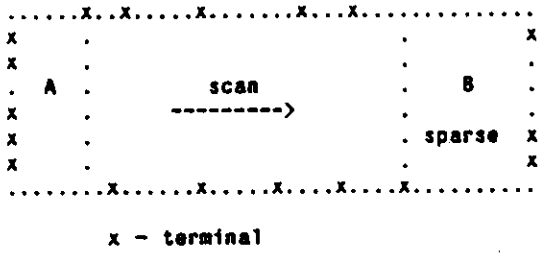


Figure 2-15: Choice of scan direction from A to B:
more terminals and multiple terminals on side A,
more free space on side B

The above rules can be formulated by the selection weighing function:

$$w(e_i)=w_s(n_{s,i})+w_m(n_{m,i})+w_d\left(\sum_{j\in J(i)} d'_j\lfloor (i+1)/2\rfloor\right) \text{ for } i=1,2,3,4$$

w_s, w_m and w_d are monotonic increasing functions for evaluating Rule 1 and Rule 2. $n_{s,i}$ stands for the number of single terminals on the edge e_i, $n_{m,i}$ stands for the number of additional multiple terminals on the edge e_i, and $\sum_{j\in J(i)} d'_j\lfloor (i+1)/2\rfloor$ measures the track density of the region close to the edge e_i. $J(i)$ is an index function, it represents a set of columns residing inside half of the routing region that is next to the edge e_i.

$$J(i)=\{N-\lfloor (N-1)/2\rfloor,\ldots,N-2,N-1\} \text{ for } i=1$$

$$J(i)=\{1,2,\ldots,\lfloor (N-1)/2\rfloor-1\} \text{ for } i=2$$

$$J(i)=\{1,2,\ldots,\lfloor (M-1)/2\rfloor-1\} \text{ for } i=3$$

$$J(i)=\{M-\lfloor (M-1)/2\rfloor,\ldots,M-2,M-1\} \text{ for } i=4$$

Further the contribution from the single terminals $n_{s,i}$ on the two opposite edges may have been absorbed in that of the augmented density distribution, so the selection function may be simplified to:

$$w(e_i) = w_m(n_{m,i}) + w_d\left(\sum_{j \in X(i)} d^+_{j,\lfloor (i+1)/2 \rfloor}\right)$$

The *LEFT* edge is chosen as the edge that has the higher $w(e_i)$, the *RIGHT* edge is taken as the remaining edge.

Example: Using the previous example, we remain to decide between e_3 and e_4. Assuming the weighing functions for w_m and w_d are linear and have same weight. Recall that

$$d^+_1 = (13,14,14,13,12,12,13,13,12,13,12,11,9,8,7,7,8,9,9,9,10,11,11,11,11)$$

$$w(e_3) = n_{m,3} + \sum_{j \in X(3)} d^+_{j,2}$$

$$= 0 + (14 + 14 + 13 + 12 + 12 + 13 + 13 + 12 + 13 + 12 + 11) = 139$$

$$w(e_4) = n_{m,4} + \sum_{j \in X(4)} d^+_{j,2}$$

$$= 3 + (8 + 7 + 7 + 8 + 9 + 9 + 9 + 10 + 11 + 11 + 11) = 100$$

So we should assign e_3 as the *LEFT* edge and e_4 as the *RIGHT* edge, i.e. scan from e_3 to e_4.

□

We have tested the same example using the four possible scan directions. Result agrees with the prediction that the best scan is from e_3 to e_4. The given routing region is 23 by 16, so only the predicted scan succeeds in finding a solution without the use of additional tracks.

scan direction	tracks used
e_1 ----> e_2	23 x 18
e_2 ----> e_1	23 x 20
e_3 ----> e_4	23 x 16
e_4 ----> e_3	24 x 16

3. Implementation

3.1. General features

The implemented router takes a mask description (CIF file) as input, and generates a routing output as an individual cell in the form of another CIF file. A user can specify the routing problem via a VLSI layout editor (e.g. CAESAR) or an input specification. The input specification includes a *layout* plus a *routing region* specified in the form of a rectangular window. The router will connect all the terminals that intersect the same label (character string) touching the above region. These labels will also appear as name extensions in the CIF file containing the routing results.

The resulting cell can then be inserted into the user's own layout hierarchy. We recommend running the router in parallel as background job when other CAESAR editing is going on, since it may take a while for the router to complete its job.

If the routing area initially specified is not enough, the router will increase its area in order to complete the routing. So the router always succeeds in finding a solution by inserting horizontal or vertical tracks, pushing into neighboring cells in this case.

The router works in two steps. A technology independent wiring is first generated, and then transformed into the final mask description. Two layers are used for connection. Currently, the transformation only supports NMOS based on the Mead-Conway design rules.

Metal and polysilicon layers are used to run in each of the vertical and horizontal directions. Usually metal runs along the longer span and polysilicon along the other. Contacts are added whenever layer changes. Polysilicon and metal wires for signal nets (not VDD and GND) are converted into nominal width inside the routing region, the nominal width is 2 lambdas for polysilicon and diffusion, and 4 lambdas for metal.

The interface between the input format of the router and the layout editor CAESAR is written in C (by Hank Walker) and the router and the CIF translator are written in Franz Lisp. All the programs are running on a VAX-11/780. The switch-box router is used as part of a set of routing tools: channel, river, switch-box routers for custom VLSI layout [3].

3.2. Guard region

There is a space called a *guard region* between the internal wiring and the terminals on the four edges of the routing region. It is used for matching the different layers (if any) between the inside wiring which is either polysilicon or metal and the terminals which may be polysilicon, diffusion or metal. Further, due to the mismatch of the terminal coordinates and the routing grid, the terminals may not be able to go straight into the inside routing region. The guard region allows them to jog a little bit to match the internal routing grid (see Figure 3-2). A typical guard has a size of about 8 to 10 lambdas on each side of the routing region.

3.3. Three sided routing

The router can route a region with terminals fixed on any three sides. The algorithm is simply based on the greedy channel router with the rows of the left-most column initialized to the *LEFT* terminals (see Figure 3-1).

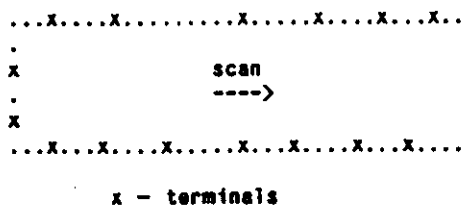


Figure 3-1: A 3-sided routing problem

3.4. Example

3.4.1. Example SW1

A switch-box routing example is taken from [1]. The specification is:

$$R=\{0,1,2,....,24\}\times\{0,1,2,....,17\}$$

$$NET=\{1,2,....,24\}$$

$$\mathbf{e}_1=[15,0,2,4,12,7,6,9,5,8,13,15,14,15,0,21,20,1,2,19,1,18,0]$$

$$\mathbf{e}_2=[24,17,16,4,7,6,5,9,8,0,9,12,15,24,15,10,23,1,0,0,22,18,0]$$

$$\mathbf{e}_3=[0,3,10,4,16,12,17,2,9,1,24,11,13,14,0,0]$$

$$\mathbf{e}_4=[0,18,22,2,23,18,21,11,20,18,20,0,24,19,3,15]$$

0 represents a location that is not occupied by a terminals. The routing result is shown in Figure 3-3. It requires a region of 23×16 tracks. We have not been able to come up with a better hand-routed solution. This result is compared with two lower bounds on switch-box routing based on the number of terminals and augmented density. They suggest that the grid must be greater than 20×14 (based on the two lower bounds). The result requires 2 to 3 tracks more in both directions.

- *terminal number*: The minimum grid size required to hold the terminals on the four edges. This bound is 20×14 in the example.
- *augmented density*: The augmented channel density (D_2^+, D_1^+) for the two edge pairs (Section 2.5.1). This bound is 17×14 in the example.

The cpu time on a VAX-11/780 required to generate the wiring is 1.1 *s* (which is about 0.2 *s* to read input data and 0.9 *s* to run the wiring) and the time to translate the wiring into CIF is 0.84 *s*.

□

The switch-box router was tested for the Deutsch difficult problem and the cpu time on a VAX-11/780 required to generate the wiring is about 4.5 *s*. It requires 20 tracks (the channel density is 19).

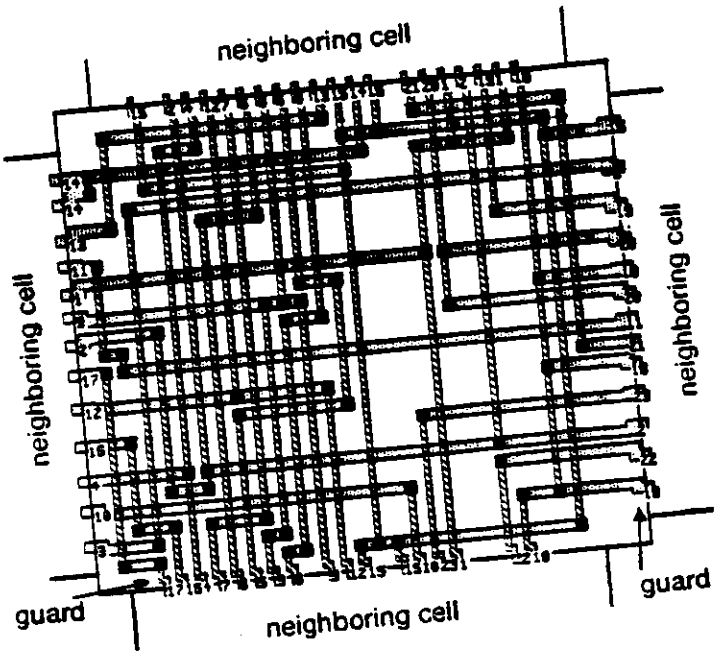


Figure 3-2: Guard region in switch-box routing

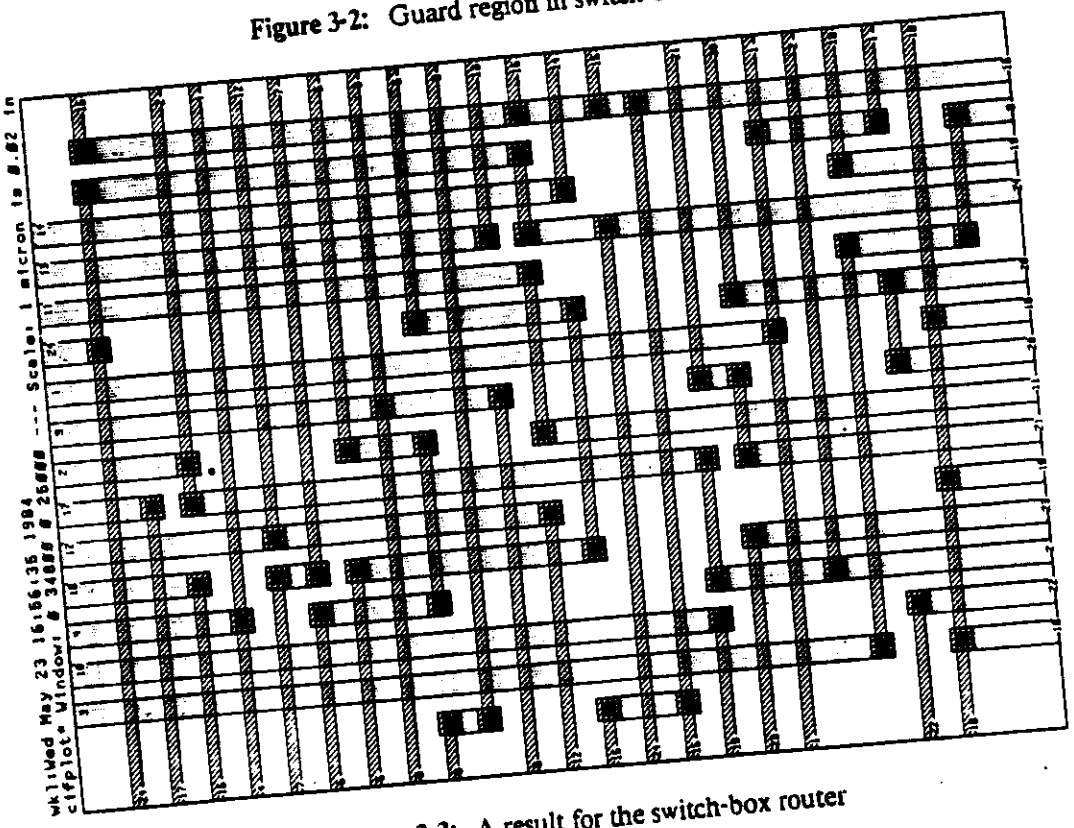


Figure 3-3: A result for the switch-box router

3.4.2. Terminal-intensive example

We discuss another example in which all the grid points on the four edges are completely occupied by terminals. We call such case *terminal-intensive.*

$$R=\{0,1,2,....,24\}\times\{0,1,2,....,17\}$$

$$NET=\{1,2,....,24\}$$

$$\mathbf{c}_1=[15,5,2,4,12,7,6,9,5,8,13,15,14,19,15,21,20,1,2,19,1,18,3]$$

$$\mathbf{c}_2=[24,17,16,4,7,6,5,9,8,8,9,12,6,24,15,10,23,1,10,3,22,18,11]$$

$$\mathbf{c}_3=[8,3,10,4,16,12,17,2,9,1,24,11,13,14,24,15]$$

$$\mathbf{c}_4=[10,18,22,2,23,18,21,11,20,18,20,3,24,19,3,15]$$

The density distributions are

$$\boldsymbol{d}_1^+ =(15,15,16,15,14,14,15,14,14,14,13,12,10,9,8,8,9,11,11,11,11,12,12,12,12)$$

$$\boldsymbol{d}_2^+ =(18,18,18,18,17,17,15,16,15,16,16,16,16,16,16,18,18,17,17)$$

And the augmented densities (D_2^+, D_1^+) for the two edge pairs are (18,16).

The routing result is shown in the following Figure 3-4. It requires a grid of 23×16. It is optimal in the sense of the lower bound based on the terminal number, but is not in the sense of that based on the augmented density. The result is obtained by using a constant jog-to-target threshold of 1 instead of the distance dependent threshold scheme.

3.4.3. Dense example

The last example attains the lower bound set by the terminal number. In general, it is more difficult to attain the lower bound that is determined by the augmented density, especially those with a density distribution $\boldsymbol{d}^+$ uniformly dense through out the region. We call such case a *dense* switch-box routing problem. The heuristic for finding a good scan direction tries to scan from the end that is denser towards to the end that is less dense. This can be seen from the example SW1 and the terminal-intensive example. In both cases, one of the two density distributions has a less dense end.

We illustrate another example in which the two density distributions are uniformly dense.

$$R=\{0,1,2,....,17\}\times\{0,1,2,....,19\}$$

$$NET=\{1,2,....,19\}$$

e_1=[15,5,2,4,12,7,19,9,6,8,13,15,1,3,18,0]

e_2=[18,17,16,4,7,6,19,2,5,0,9,10,12,0,14,0]

e_3=[0,8,3,10,4,16,12,17,2,19,9,1,0,11,13,14,18,15]

e_4=[0,8,3,10,6,19,12,2,7,9,11,1,5,14,13,0,18,15]

The density distributions are

d_1^* =(16,16,17,16,15,15,16,16,16,16,16,16,16,16,16,16,16,16)

d_2^* =(13,13,14,15,15,14,14,13,13,12,12,13,14,14,14,14,14,13,14,14)

The two values of track availability are respectively $s_1 = t_1/D_2^* = 16/15 = 1.067$, $s_2 = t_2/D_1^* = 18/17 = 1.059$. The uniformness of the two density distributions and the closeness of s_1 and s_2 to 1 shows that this routing problem is *dense*. $s_2 < s_1$ suggests to scan in the direction along the d_2 distribution. Actual experimental results show that scanning along the other d_1 direction is better: along d_1 (e_1-->e_2) requires a 16×18 grid and along d_2 (e_3-->e_4) requires a 16×20 grid. This discrepancy may be due to s_1 and s_2 being too close to each other.

The grid size based on the augmented density lower bound is $(D_2^*, D_1^*)=(15,17)$. The actual grid size is 16×18, and is 1 track in both directions from the augmented density lower bound. The result is shown in Figure 3-5. We do not know whether or not the augmented density bound can be achieved.

4. Conclusion

We have presented an efficient algorithm and implementation for a switch-box router for VLSI layout. The objective of this experiment is intended to come up with a set of routers and eventually integrate them into an automatic placement and wiring tool for custom VLSI design. In addition, the router is designed to work independently for assembling custom cells, and communicate with other tools such as a layout editor via a simple interface. Thus it serves as a tool for both immediate usage and future need.

The greedy channel routing algorithm can be nicely extended to handle switch-box routing efficiently. It functions by relaxing some of the non-critical operations of the channel router, namely jog-bypass-conflict and jog-for-join, and modifying them into some operations, e.g. jog-to-right-target, that are basic in solving the switch-box constraints. The expected running time is proportional to $M(N+N_{net})$, where M,N and N_{net} are respectively the number of columns, rows and nets in the routing region. The scan direction is crucial to the algorithm and we have proposed good heuristic in finding it.

The basic control structure of the algorithm can be easily modified to study the effect of different

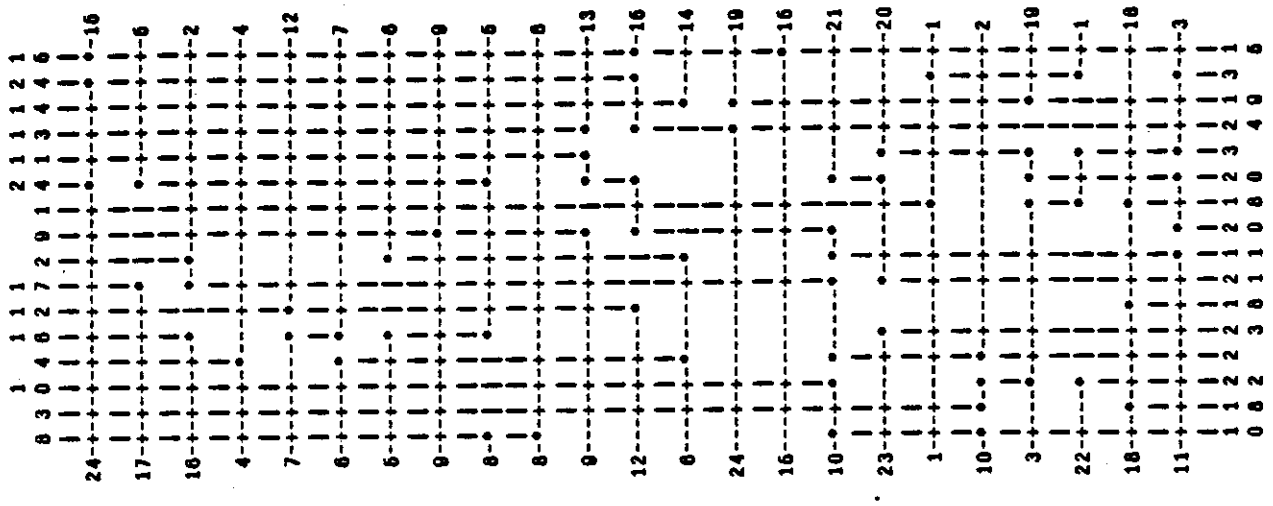


Figure 3-4: A terminal-intensive example

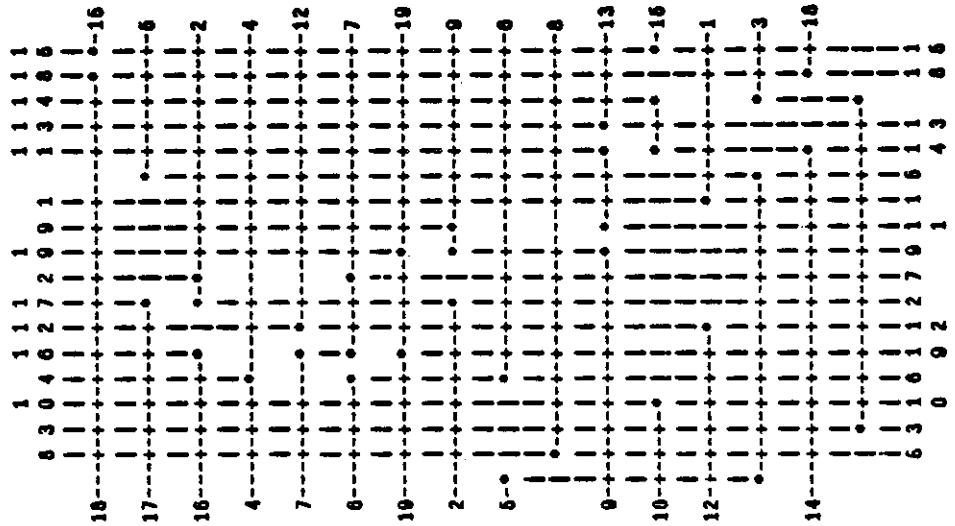


Figure 3-5: A dense switch-box routing problem

heuristics on the routability for a given switch-box routing problem, and tradeoff between routability and its execution time. Though the router always succeeds in giving a route by extending the routing region, the problem of finding routability of such switch-box router remains open.

Acknowledgements

Thanks are due to Hank Walker for contributing the CAESAR to router interface and comments on the report.

References

[1] Burstein, M. and Pelavin, R.
Hierarchical wire routing.
IEEE Trans. on Computer-Aided Design CAD-2(4):223-234, October, 1983.

[2] LaPaugh, A. S.
Algorithms for integrated circuit layout: an analytic approach.
PhD thesis, Dept. of EE and CS, MIT, December, 1980.
MIT VLSI memo 80-38.

[3] Luk, W. K.
ROUTER: A set of routing tools for VLSI layout.
April, 1984.
Computer Science Dept., Carnegie-Mellon University, VLSI doc. V155, April, 1984.

[4] Rivest, R. L. and C. M. Fiduccia.
A greedy channel router.
In *19th Design Automation Conference*, pages 418-424. IEEE, 1982.

[5] Soukup, J.
Circuit layout.
Proc. of IEEE 69(10):1281-1304, October, 1981.