Routing Algorithms for VLSI Design

by

R. Venkateswaran and P. Mazumder

CSE-TR-90-91

Computer Science and Engineering Division
Room 3402 EECS Building

THE UNIVERSITY OF MICHIGAN

Department of Electrical Engineering and Computer Science
Ann Arbor, Michigan 48109-2122
USA

ROUTING ALGORITHMS FOR VLSI DESIGN

R. Venkateswaran P. Mazumder
Dept. of Electrical Engg. and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122

Contents

Abstract

The area of automatic routing is very vast and of considerable importance. A lot of work has been done in this area and a lot remains to be done. As newer technologies emerge, either existing routing algorithms will have to be adapted to cope with the new routing constraints, or else new algorithms will have to be developed. In either case, much can be learnt by studying and understanding existing algorithms. This article attempts to put things in perspective by presenting and contrasting several representative routing strategies.

1 Introduction

The layout process typically is divided into a *placement* phase followed by a *routing* phase. During placement, the various components are placed onto the substrate surface so as to minimize cost criteria such as the total estimated wire length. The routing phase is the one that determines the actual course of the wires connecting the cells that have been placed. Routing has to abide by certain *design rules* that ensure a technically functioning circuit. For instance, some rules specify physical constraints such as the widths and clearances of conductor paths. An abridged flowchart for automated layout is shown in Fig. 1.

In addition, other constraints that a good router should consider are minimizing the total wiring area required, minimizing long paths that could cause excessive propagation delay, prioritizing the layout of critical signals such as clock lines, *etc.* The structure of the routing phase is also greatly influenced by the design and fabrication technology. The following terminology will be adhered to in the rest of this article:
Net: A set of pins or terminals that are to be electrically connected. This information is provided in a *netlist*.
Via: A hole or other mechanism used to provide physical connections between different layers.
Block or *cell*: Refers to a component made up of transistors or gates.

There are many ways of classifying routers. One way of classifying routers is as follows:

One-wire-at-a-time routers : As the name suggests, these routers make one connection at a time. The wiring order is often critical because previously laid wires can affect the routability of later nets. Important members of this class are the **maze** routers, **line-probe** routers and the **line-expansion** routers.

Area-restricted routers : Here, the router has to complete all interconnections within a (relatively) small routing region of restricted area. **Channel**, **switchbox** and **river** routers fall in this class. In addition, river routers require all nets to be routed on a single layer. The main distinguishing feature of area-restricted routers is that they consider all the nets simultaneously. Thus, the resulting routing is net-order independent.

Global routers : As opposed to the first two styles, which are sometimes referred to as *detailed* routing, global routing essays to provide a routing plan in which each net is assigned to a particular routing region(s). Exact wiring of the nets within the region is left to a detailed router. The main idea is to balance the congestion across the routing regions, minimize overall wire lengths and on the whole make the job of the detailed router easier.

Specialized routers : Certain wires need special treatment. For example, while routing power lines, factors such as voltage drops, metal migration assume greater significance. Clock lines have to be adjusted for skew. Consequently, algorithms that do signal routing are incapable of handling these additional constraints.

Others : In this category can be placed **topological** via minimization routers, layer-assignment routers, single-row routers, routers for **analog** layout, PC-board routers and others, each one fine-tuned to meet the problem specifications.

Layout styles play a very big factor in determining the suitability of a particular routing scheme. The following five styles are representative of the major styles:

Full-custom design : Here, the designer is pretty much free to place and route cells arbitrarily. No assumption is made as to the location or shapes of the cells. Routing for full-custom is *two-phase*: consisting of a global routing phase followed by detailed routing.

Standard-cell design : In this layout style, cells are placed in horizontal rows; each cell has the same height and has fixed connections on the left and right sides (clock and/or power) that abut with the neighboring cells. A channel separates two rows and is used for wiring purposes. Wires can switch between non-adjacent rows of standard cells at specified locations between the cells called *feedthroughs*. One problem of standard-cell global routers, therefore, is to determine an optimal set of feedthroughs that minimizes wire length and channel congestion. Another factor that the global router considers relates to *equivalent* pins, which are terminals that are connected within a cell and thus are electrically indistinguishable. Detailed routing is normally performed by a channel router. All interconnections can eventually be made because of the flexibility in determining the positions of the rows, cells and feedthroughs.

Gate-array design : Here, a cell-based design is mapped onto a prefabricated, two-dimensional array of uncommitted transistors called *cells* or *gates*. The array is then personalized for a particular circuit function by depositing interconnections on the top layers of metallization. The cells can be arranged in rows creating horizontal channels for inter-cell wiring or they can be arranged in a matrix form with both vertical and horizontal channels. In the latter case the cells are referred to as *islands*. Routing in gate-arrays is similar to the methods employed for standard cell design with one major difference, *viz.*, the fact that the channels have fixed width. This means that all connections may not always be completed. A good global router is therefore of paramount importance in gate-array design.

Sea-of-gates design : Sea-of-gate technologies are successors to the gate-array design style. The characteristic features include the fact that there are no discernible routing channels, *i.e.*The whole real estate is now filled with transistors. There are no discernible routing channels. This enables the realization of dense logic such as random-access memories directly on the master. In the absence of routing channels, interconnects have to be completed either by routing through existing gates, or by adding more metal or polysilicon interconnection layers. There are problems associated with either solution. The former reduces the gate utilization; the latter increases the mask count and increases fabrication time and cost. For sea-of-gates arrays, a multilayer maze router or an over-the-cell router is preferable to do detailed routing so as to make full use of the wiring area.

Printed-circuit-board design : Though, similar in appearance to an integrated circuit design problem, the routing criteria are different. In the MOS layout, the resistivity and capacitance per unit path length are not the same for different layers. The metal layer, which has low resistivity, is used whenever possible for routing. Polysilicon is mainly used to cross under the metal. On the other hand, main routing layers on PCB's have essentially the same properties. Change in layers is therefore dictated more by the ensuing increase in production cost and decrease in reliability, than by any electrical properties of the layer. Wave reflections, ringing due to impedance mismatch, and cross-talk between lines are factors that have to be considered while designing high-speed PCBs. Design rules relating to paths, lands and pads have to be complied with. With the advent of surface mounted technology, devices can be placed on both sides of the board (traditionally called the component and the solder sides). This means that routing has truly become a three-dimensional problem. Also, the number of wiring layers can easily exceed a dozen. Channel routers that solve a two-dimensional routing problem are therefore hardly sufficient. By far, maze routing has been the router of choice for printed circuit board because of their flexibility in adapting to manifold technological factors.

The rest of this article is organized in accordance with the router classification that was presented earlier. Some of the commercial layout tools that use some of these algorithms are also mentioned in the conclusion.

2 Maze Routing

Maze routers form a broad and adaptable class of routers also known as Lee routers, or Lee-Moore routers after their inventors. Maze routers can be employed for both detailed as well as global routing. The maze

Algorithm 1 LEE-MOORE MAZE ROUTER FOR A SINGLE LAYER

Input : A rectangular grid of dimensions $M \times N$ with
initial blockages, if any, marked.
The netlist of all two-pin nets to be routed in this grid.

Output Grid with all nets routed, if possible.

Begin

For each net in the netlist do

1. *Initialization* : CCS (current cell set) $\leftarrow$ Source cell;
NCS (neighbor cell set) $\leftarrow \Phi$;
wavefront $\leftarrow$ 1;

2. Label all cells in CCS with value of *wavefront*;
If target cell has been added to NCS then
goto Step 5;

3. *Wave Expansion:* For each cell in CCS, add all its unexpanded
neighbors to NCS;

4. If NCS is empty then
this net is not routable; goto Step 6;
Else
CCS $\leftarrow$ NCS; NCS $\leftarrow \Phi$; *wavefront* $\leftarrow$ *wavefront* $+1$;
goto Step 2;

5. *Backtrace*: Retrace the shortest path to the source, starting from the target,
by considering neighboring cells in the descending order
of labels. If there are more than one cell with label $l-1$
that are adjacent to a cell with label l, then choose the one
that causes no change in direction (no bends) in the path
traced out so far.

6. *Label Clearance*: Reset all labeled cells, except those used
for the path just found, to be *empty*.

end for

End

router is so called because it operates on a uniform rectangular gridded area with an incremental spacing of Δ. This spacing is essentially the center-to-center constraint that is assumed to be standardized for all nets. It is characterized by two values: a minimum width w and a minimum clearance c; such that $\Delta = w + c$. All the pins and terminals of nets are also assumed to be located at the grid intersection points.

2.1 The Original Lee Maze Router

This router, first published in 1961 [10], was intended for routing on a single layer, even though it will be seen later that the extension to more than one layer is straightforward. It consists of three phases a *wave propagation* phase (due to its resemblance to the wave created by dropping a pebble into a pool of water), a *backtrace* phase and a *label clearance* phase.

Example

In Fig. 2, the numbers in the grid indicate the labels assigned to the cells during wave propagation phase. In this case a uniform weight of 1 is assigned for each expansion, and hence the label of the cell also corresponds to the length of the shortest path from that cell to the source. The target, in this case, is seen to be reached by a path of length 10. Fig. 3 shows the result after the backtrace and label clearance phase is over. The new net is shown in a lighter shade and will act as a block for other connections.

Discussion:

1. Maze routing has two main advantages over other routing techniques. First, because of its exhaustive search (breadth-first) characteristic, it guarantees to find a source-to-target path, if one exists. Secondly, paths can be optimized with respect to complex objective functions, such as via-consumption, cross-over and pin-blocking.

2. Maze-routing as shown above has also several disadvantages. First, it requires large amounts of storage. For an $N \times N$ grid, it requires $O(N^2)$ memory. Some non-trivial amount of temporary storage is also needed to keep the positions of the cells in the current wavefront. Secondly, maze routers have long execution times. The wave propagation phase and the label clearance phase take $O(l^2)$ time each to route a path of length l. Thirdly, despite guaranteed path-finding abilities, they typically consider one net at a time. This poses the problem of net-ordering: paths routed earlier can block later paths. A non-optimal order can result in the maze-router being unable to make certain connections. For example, Fig. 4(a) shows a case [47] where the routed net B blocks net C from being connected. The correct solution is shown in Fig. 4(b). Finally, maze-routers require all the features (pins, wires, vias) to be on-grid. So any feature that lies between cells (off-grid) have to be considered as blocking all cells they impinge upon. This typically causes wastage in routing area.

3. Commercial maze-routers are widely available for multi-layer carriers. Because of their path-finding abilities, they are widely used as overflow routers to embed those final nets that are unroutable by other means. The maze-running paradigm is also utilized by several popular routers, including those specializing in channel or switch box routing.

2.2 Reduction of Memory Requirements

As mentioned earlier, the Lee algorithm requires a unit of memory space be allocated for each cell (grid point). For a single-layer problem, this at least involves one bit for indicating whether the cell is free or blocked and k bits for the label associated with it during the wave propagation phase. For an $N \times N$ grid, where path lengths of up to N can be expected, $k = \log_2 N$.

In practice, however for shortest-path problems, k can be made a fixed constant like 2 or 3, and is independent of N. Such a coding scheme was first introduced by Akers [9]. He noted that any cell containing a number X, has adjacent to it only cells that are filled or empty, or else contain either $(X-1)$ or $(X+1)$. Hence, any sequence that guarantees that the predecessor of any symbol in the sequence is different from its successor, can be used instead. An example of such a sequence is $1, 1, 2, 2, 1, 1, 2, 2, \ldots$ Here, the cells on every two consecutive wavefronts are assigned the same label, *viz.* 1 or 2. In order to retrace the optimum path, it is merely necessary to start at the proper point in the $1, 1, 2, 2 \ldots$ sequence. This can be achieved using a single counter. In Fig. 5, the target is labeled 1. The retraced path is indicated by the dotted line. Since, at each stage each cell is in only four states- filled, empty, has a 1, or has a 2- it follows that only 2 bits are needed per cell.

Other efficient labeling sequences also exist, such as the $1, 2, 3, 1, 2, 3 \ldots$ sequence. Yet another strategy is to label the empty cell with $\leftarrow$ (left), $\rightarrow$ (right), $\uparrow$(up), $\downarrow$ (down), which indicates the direction of wave propagation. For a 2-bit representation, in case a cell is reached from more than one direction, any one of them can be picked.

Though, significant memory reductions can be achieved in this fashion, there still exists a trade-off between number of bits used per cell and the efficiency of the algorithm. Some redundant information, for instance, may go a long way in simplifying both the label clearance phase as well as programming. For multi-layer extensions to Lee algorithm, additionally information about vias has to be maintained for each cell. One such coding scheme uses $(M+3)$ bits per cell: one bit for indicating via availability, two bits for the labeling sequence (including a state for unlabeled cells), and the remaining M bits for indicating the presence or absence of a block on any given layer. For improved performance, the Geyer's router [36] (to be discussed later), uses $(2M+3)$ bits per cell.

2.3 Speed Improvements

Since the wave propagation phase is computationally the most expensive, all speed-up techniques are related to this phase. Software algorithms attempt either to prune the search space by reducing the number of cells

---

Algorithm 2 WAVE PROPAGATION PHASE OF HADLOCK ROUTER

Begin

1. *Initialization* : CCS (current cell set) ← Source cell;
   NCS (neighbor cell set) ← Φ;
   $MinDetour \leftarrow 0$;
2. If target cell has been added to NCS then
   Proceed to Backtrace Phase;
3. For each cell in CCS, add all its unexpanded
   neighbors to NCS, along with their d(detour) values;
   Sort NCS entries in ascending order of d values;
   $MinDetour \leftarrow$ Minimum d value in NCS
4. CCS ← all cells in NCS with d value = $MinDetour$;
   NCS ← NCS - CCS;
   goto Step 2;

End

---

searched or by assigning priorities to wavefronts and expanding those with higher priority first. On the other hand, hardware accelerators can be developed that can expand all active cells in parallel, ideally reducing the $O(l^2)$ complexity to one of $O(l)$ time.

2.3.1 Software approaches

Akers suggests three possible methods to reduce the number of cells being labeled

- For a given pair of terminals, start wavefront propagation from the one that is farthest from the center of the grid. The idea here is that if the source is close enough to the frame, the area of wave propagation gets bounded by it.
- Start simultaneous wave propagation from both source and target till a common point of contact is reached. This typically reduces the area of propagation by half.
- By restricting the expansion to lie within an artificial rectangular bounding box, typically 20% larger than the one defined by the terminal pair, we can considerably reduce the number of cells searched. The idea here is the fact that most nets are routable near their minimal lengths, and do not extend beyond the minimum bounding rectangle. This notion is referred to as *framing*. The size of the rectangle can be statically fixed or varied incrementally, depending on whether we could route the net within the bounding box or not.

Two software algorithms that lead to considerable speed-up have been developed by Hadlock [35] and Soukup [26], respectively. Both of these can be considered as *goal-directed* search variants of the Lee algorithm.

2.3.2 Hadlock's minimum detour algorithm

Given a source cell A and a target B, M(A,B) is the Manhattan distance on the grid, between the two points. The *detour number* d(P) of any path P between A and B, is the number of cells on P that are directed away from the target B. Clearly, the length of such a path is,

$$length(P) = M(A, B) + 2d(P)$$

Clearly $length(P)$ will be a minimum, if and only if d(P) is minimized. This is the basic idea of Hadlock's algorithm. It proceeds by modifying Step 2 of the basic Lee Algorithm as follows: instead of labeling each

---

Algorithm 3 WAVE PROPAGATION PHASE OF SOUKUP ROUTER
Begin

```
1. Initialization :  CCS (current cell set) ← Source cell;
                     NCS (new cell set) ← Φ;
2. If target cell has been added to NCS then
       Proceed to Backtrace Phase;
3. For  each cell in CCS do
       a. If there exists a free neighbor n that is closer to the target then
          Start a line segment in this direction, adding all cells
          on this line to the end of CCS. Stop extending the line
          upon encountering a blocked cell or when no progress is made
          towards the target
       b. Lee expansion:  Add all the unexpanded neighbors to NCS
4. If NCS is empty then
       no path exists -- flag error;
   Else
       CCS ← NCS; NCS ← Φ; goto Step 2;
```

End

---

cell with the current wavefront number (distance from the source), the detour numbers with respect to a particular target are used. Only, the wave propagation phase is outlined below. The other phases remain unchanged.

Discussion: This algorithm maintains the maze-routing guarantee of a shortest path. The running time now varies between $O(N)$ to $O(N^2)$ depending upon the distribution of the blockages and the given netlist. For a general weighted graph, a priority queue like data structure is required. However, for a uniform-weighted graph, Hadlock has devised a two-stack solution. Fig. 6 shows the detour numbers entered in the searched cells. For the path P indicated by the dotted line, we have $d(P) = 3$ and $M(A, B) = 8$. Thus $l(P) = 8 + 2 \times 3 = 14$.

2.3.3 Soukup's fast maze algorithm

It guarantees a connection if one exists, while at the same time it runs 10–50 times faster than the regular Lee algorithm on typical two-layer boards. In this algorithm, a line segment starting from the source is extended towards the target as much as possible, unless an obstacle is encountered earlier. The cells on this line segment are searched first. Again, if any cell approaching the target is found, then another line probe originating from that cell is made; otherwise, Lee-type wave propagation is performed till such a cell can be found or the target is met.

Discussion:The algorithm can be easily coded. The CCS and NCS (the original paper calls these the old and new reach sets respectively) are best represented using stacks. This is because in Step 3, the next cell which we wish to expand is the one last added to CCS. The idea here is that the cells on the line segments are nearer the target in a sense, and hence wave expansion around them first may lead to a solution faster. A cell is *free* in Step 3a if either it is not blocked or reached during the line search procedure. The quality of routing is sometimes sub-optimal, both in length of the path and in the number of bends used. This is the price to be paid for the improved speed.

2.4 Hardware Accelerators for Maze Routing

Maze-routing is also the only major routing technique that lends itself easily to be parallelized. Several researchers have, therefore, proposed a variety of architectures suitable for maze routing. Advances in VLSI technology have made such routers feasible.

The typical framework of most routing engines is a 2-dimensional array of processing elements that are locally interconnected. There may be a one-to-one or many-to-one mapping between grid cells and processing elements. The wavefront expansion phase is the most time–intensive portion of the algorithm. So, all mappings assign adjacent grid cells to neighboring processors, so that each wavefront can be quickly expanded to the next. The processing elements work under the supervision of an array control unit that also interfaces to the host computer.

Grid Folding

Most implementations incorporate a notion of grid-folding: grids larger than the physical array are folded into several planes and mapped onto the storage available at each node. Such routing engines are often referred to as *virtual*, in analogy to the computer memory hierarchy. The Wire Routing Machine (WRM) designed by Nair *et al.* [75], and the Distributed Array Processor (DAP) by Adshead *et al.* [33] are two such machines.

Other implementations, on the other hand are *fixed*: there is a one-to-one correspondence between the size of the grid (number of cells) and the number of processing elements in the array. The L-machine of Breuer and Shamsa [32], and the Bit Mapped Processor (BMP) of Blank *et al.* [11] are in this category. Each can be used to solve problems on grids of size less than or equal to 1024×1024 cells, which is also the number of processing elements present in the system.

The *fixed* architectures are usually much faster than their *virtual* counterparts. In the *virtual* implementations, several cells get mapped onto the same processing element. Additional storage is required at each node to keep track of this information. Secondly, during wavefront propagation, the address of the grid cell being expanded, has to be sent to the neighboring processing elements . This can be as much as $2 \log_2 N$ bits for an $N \times N$ grid. For a *fixed* size array however, one bit of information is clearly sufficient. Thus the cost for accessing storage elements and additional communication overheads tend to slow down the expansion phase on a *virtual* machine. On the other hand, the *fixed* machines are less flexible and can only solve a certain size of problems.

Complexity of the processing elements

The L-machine employs simple state-machines (L-cells), while the BMP implementation consists of bit-serial addressable processors each with a little memory. On the other hand, the WRM is made up of an 8×8 array of general purpose Z80 microprocessors with 15Kb of memory per node.

There is again a trade-off between performance and speed. Simple-minded processing elements cannot cater to complex path functions and heuristics required by production-quality maze routers. However, they can often be custom designed to greatly accelerate the execution speed. The WRM can accommodate a weighted grid framework with expected wiring cost and demand; the L-machine and BMP require unit-cost assumptions. However, the performance improvements on the small WRM (8×8) prototype were at best modest: basic problems required roughly one minute to execute on either the WRM or a mainframe computer. Assuming that a few minutes of computation time is easily obtainable on the mainframe without much burden, the added complexity of the special hardware may become unacceptable in spite of its obvious cost-effectiveness.

Another advantage of using simple processing elements is with respect to VLSI implementation. By eliminating off-chip communication delays, it is possible to obtain an order of magnitude improvement in speed. Breuer and Shamsa, have also proposed a Modular L-Machine, wherein the full array of 1024×1024 processing elements is partitioned into smaller subarrays, each of which can be designed using a single chip. To reduce pin count, these subarray chips communicate serially with each other.

Mapping and Interconnection Topologies

This obviously concerns only *virtual* architectures. A good mapping of grid cells to processors in the array is imperative in order to achieve maximum parallelism. A good mapping will attempt to assign the cells in the current wavefront to as many different processing elements as possible. One characteristic of any of the interconnection topologies to be discussed later is the fact that they are all wrapped around symmetric structures. The minimum distance between two occurrences of the same processing element along any straight line (including diagonal lines) is referred to as the *interprocessor cycle period*. For uniform weighted grids,

Table 1: Evaluation of interconnection topologies for maze routing

Interconnection Topology†	Interprocessor Cycle Period	Multi-Layer Routing
Square Mesh	$\sqrt{N}$	No
Twisted Torus	$\frac{N}{2}$	No
Hexagonal Mesh	N	Yes

† N is the total number of processors present in each topology

the wavefronts typically are diamond-shaped. Therefore, one heuristic is to optimize processor assignment across 45 and 135° lines, by having as large an *interprocessor cycle period* as possible.

The traditional interconnection topologies that have been employed include the *square mesh* – WRM, and the *toroidal* topology – Suzuki *et al* [27]. Current processing technologies allow up to 4 layers (2 metal and 2 poly) for routing. Interconnection topologies such as a square mesh or toroid, assign the same processor to all cells with the same (x, y) co-ordinates, but on different layers. This implies that wavefront expansion has to step through the layers sequentially. This limitation is true of any topology where each processor has only 4 nearest neighbors. The problem of via placement is also quite severe. Most of these problems are resolved by the Hexagonal Array Machine – HAM [29]. Here, two of the six neighbors of each processing element are mapped to grid cells on adjacent layers. This allows concurrent expansions on multiple layers and also assigns costs to each via. This will be further explained in a later section. A brief comparison of the three topologies considered is shown in Table 2.4.

Other Architectures: Rutenbar [28] has mapped the maze-routing algorithms onto a cytocomputer architecture that has traditionally been employed for image–processing applications. The specific framework used was a linear systolic array, where N processors, referred to as *stages*, are connected in a pipeline fashion. Each processor has internal buffering to store each cell neighborhood (3×3 array of cells). The entire routing grid streams through the processors in raster (row) order. All stages can work in tandem, expanding the input wavefront by one step in each direction.

Bus-structured machines have also been constructed. Damm *et al.* [31] have built a Lee-routing engine by modifying a commercial minicomputer. A special cell-memory, a hardware 'kernel' of routing operations, and an interconnecting bus were added to optimize performance. Sahni *et al.* [30] have also reported a three processor pipeline machine, with several memory banks to reduce the memory access times.

There exists a trade-off between these approaches and the processor array approaches (WRM, HAM, etc.). The hardware costs for the former is much less. Increasing the number of processors is also easy in a linear array as compared to a wrap-around mesh topology. However, the processor-array approaches are much faster. For example, consider a simple problem of searching for a path of length L in the grid shown in Fig. 7. In the worst case, with the source at the center of the grid, all the cells within a Manhattan distance of L from the source will have to be expanded. Hence, an ordinary software implementation needs:

$$T_{software} = 2L^2 + 2L + 1 = O(L^2).$$

A systolic array, with S stages and a 3×3 subarray at each stage, needs L/S passes (assume $L > S$) to perform L wavefront expansions. The total time needed is:

$$T_{sys} = (L/S)[S(2L + 3) + (2L + 1)^2] = O(L^3/S).$$

Now consider the hexagonal array structure with N processors. When $N < L$, the maximum number of cells on the rth wavefront that can be assigned to the same processor is $4r/N + 1$. Hence, the total time needed is:

$$T_{HAM} = \sum_{r=1}^{L} (4r/N + 1) = O(L^2/N).$$

Algorithm 4 WAVE PROPAGATION PHASE OF GEYER'S MULTI-LAYER ROUTER

```
Begin
    1. Initialization : CCS (current cell set) ← Source cell;
                        NCS (neighbor cell set) ← Φ;
                        MarkSeq ← 1;
                        Set the V,LP,LA and MS fields appropriately;
    2. If target cell has been added to NCS then
          Proceed to Backtrace Phase;
    3. For each cell i in CCS
           NBS ← all neighbors of i that are unmarked or marked on current cycle;
           NCS ← NCS + NBS;
           For each j in NBS
               MS[j] = MarkSeq
               If V[i] = 0 then
                  LP[j] = LP[i] or LP[j]
               Else
                  LP[j] = LA[j] or LP[j];
    4. CCS ← NCS;NCS ← Φ;
       Set MarkSeq to the next value in the 1,2,3 sequence;
       goto Step 2;
End
```

Thus, HAM offers nearly an order of magnitude better performance than the systolic array.

2.5 Multi-Layer Maze Routing

The algorithms, hitherto presented, are suitable for one layer if wires can go in both directions; or for two layers wherein one of the layers consist of vertical connections and the other horizontal routes. But, for true multi-layer routing, with $n > 2$ layers, we need to employ a 3-dimensional version of the Lee algorithm wherein, cells on adjacent layers are also labeled during the propagation phase. However, if care is not taken, memory and time requirements can easily get out of hand.

2.5.1 Geyer's multi-layer maze router

The idea behind this algorithm [36] is to advance wavefronts on all layers simultaneously, ignoring the cost for inter-layer connections through vias. The trick here is to label an unmarked square whenever another cell (accessible through a via) at the same position gets labeled.

For M-layer routing, Geyer proposed a coding scheme utilizing (2M + 3) bits per cell. This includes:

- 1-bit V field, which is set to 1 to denote a potential via
- 2-bit MS field, which records a $1, 2, 3$ marking sequence
- M-bit LA field, where the ith bit is 1 if the cell on the ith layer is unoccupied
- M-bit LP field, where the ith bit is 1 to indicate that a potential connection path to a marked cell has been found for the cell on the ith layer

Discussion: Though the Geyer's router is very fast as compared to the general Lee algorithm, it does not assure path completion if a path exists. Once a cell has been entered during a cycle, it cannot be re-entered on different layers in a later cycle. Consequently, any path that requires entering a cell on one layer and reentering the same cell on different layers in a later cycle will not be found in this algorithm. Experience has, however, shown that the occurrence of such paths are indeed rare, and hence the results are generally

Table 2: Memory contents during expansion phase of Geyer's algorithm

	1				2				3				4			
	V	LP	LA	MS	V	LP	LA	MS	V	LP	LA	MS	V	LP	LA	MS
1	0	01	01	2	0	01+01	01	3	1	11+01	11	1	1	11+11	11	2
2	1	11	11	1	0	01	01	2	1	11+01	11	3	1	11+11	11	1
3	1	11	S	3	0	01	01	1	1	01	11	2	1	11+11	11	3
4	0	10	10	1	0		00		0	10	10	3	1	10+10	10	1
5	1	10	11	2	0	01	01	3	0		00		0	01	01	1
6	1	11	11	3	1	01+11	11	1	0	10	10	2	1	10	11	3
7	0	01	01	1	0	01+01	01	2	0		00		0	01	01	1
8	1	01	11	2	1	01+11	11	3	0	10	10	1	1	10+01	T	2

acceptable. It is also easy to see how the algorithm can be modified to handle other constraints, such as layer restriction for special nets, bigger area for vias, *etc.* An example for a 2-layer situation is shown in Fig. 8.

2.5.2 HAM: a hardware accelerator for multi-layer routing

As mentioned earlier, HAM (which stands for Hexagonal Array Machine) [29], is one practical solution to get good and fast solutions to the multi-layer routing problem.

A C-wrapped hexagonal mesh of dimension e consists of $3e(e-1)+1$ ($= N$, say) processors, labeled from 0 to $3e(e-1)$, such that each processor s has six neighbors $[s+1]_N$, $[s+3e-1]_N$, $[s+3e-2]_N$, $[s+3e(e-1)]_N$, $[s+3e^2-6e+2]_N$, and $[s+3e^2-6e+3]_N$, where $[a]_b$ denotes a (mod b). Furthermore, for each of the three directions, if R_0 is the top row, R_1 the second row, and so on till R_{2e-2}, then the last processor in R_i is connected to the first processor in $R_{[i+e-1]_{2e-1}}$.

Fig. 9 shows a 3-dimensional hexagonal array with 19 processors. The wraparound connections are indicated but not drawn. For routing purposes, we need to map the processor array onto our multi-layer grid, so that each grid cell, with coordinates $< x, y, z >$ is assigned to a processor p_{xyz}. For the HAM, we have

$$p_{xyz} = [a + w_z[z]_N + w_y[y]_N + x]_N,$$

where a is the processor assigned to the grid cell (0,0,0), $w_y = 3e-1$ and $w_z = 3e-2$ and $[a]_b$ refers to a mod b.

The interface between the processor array and the host computer is the responsibility of a HAM Control Unit (HCU). Given a routing problem, the HCU will load the memories of the processors with the relevant information, such as whether that cell is blocked or is the source of a net. Wavefront expansion is achieved by the processors updating their local store based on the information sent and received to and from their six neighbors. HAM operates in an SIMD mode, *i.e.*, all processors work under a common clock, under the overall supervision of the HCU. Various architectural features are exploited to achieve maximum speed.

Example: Consider a 4 × 4 3-layer grid. Fig. 10(a) shows the processor assignment to the 64 grid points, obtained by using the formula $p_{xyz} = [14 + 7[z]_{19} + 8[y]_{19} + x]_{19}$. This is obtained by substituting $e = 3$ and $a = 14$ in the general expression. Fig. 10(b) shows the initial configuration with two nets, A and B, that have already been routed and a third net C that remains to be routed. The example has been purposely constructed such that this interconnection cannot be completed without using the third layer.

Fig. 11(a) is the snapshot at the culmination of the wave propagation phase of the modified Lee algorithm. The shaded areas indicate the previous obstacles. The number in each square is the label associated with that grid point, *i.e.*, all cells with a label i would lie on the ith wavefront. For this example, the wave propagation phase requires 11 clock cycles. There are also no processor conflicts. This is crucial towards attaining linear speedup. Table 3 shows the active processors in each clock cycle. The distinction between layers is made solely to facilitate understanding. Thus, for instance, during clock cycles 3,4 and 9, a common wavefront exists over all three layers. However, as all the active processors are distinct, this expansion is possible in a single clock cycle. Fig. 11(b) shows the final 2-via 1-bend solution.

Table 3: Processor assignments during wave propagation cycles

Clock	Processor Assignments		
Cycle	Layer 1	Layer 2	Layer 3
1	9	–	–
2	10,17	2	–
3	11,18	3	14
4	0,12	4	3,15
5	–	5	4,11,16
6	–	–	0,5,12,17
7	–	0	1,6,13
8	–	1,8	2,14
9	15	2,9	3
10	16	10	–
11	17	–	–

2.6 A Minimum Bend Maze Router

A path with fewer bends is generally preferred. This is especially true in situations where a bend translates into a via. One way is to modify the cost function of the maze router to reflect bends in the path. However, such an approach is awkward at best and not very practical. A more thorough approach was proposed by Hu and Shing [38].

2.6.1 The $\alpha - \beta$ routing algorithm

This algorithm finds the minimum cost path between two points, where the cost is determined by the number of links and corners in the path. The cost in traveling an arc is α, while the cost of a via (bend) is β. It utilizes a depth-first search strategy and can be considered to be a generalization of Dijkstra's labeling algorithm for a grid-graph G [126], when $\beta \neq 0$.

Two nodes joined by an arc in the grid-graph G are *neighbors* to each other. For any node (x_i, y_i), we define its *future cost* as $|x_t - x_i| + |y_t - y_i|$, where (x_t, y_t) is the target node. The *current cost* of (x_i, y_i) is the actual minimum-cost path that has been found to it from the source (x_s, y_s). The sum of the current and future costs of a node give its *potential cost.*

For any node, the directions that lead to neighbors with nonincreasing (increasing) future costs are defined as *forward* (*backward*, respectively) directions. Two nodes are *h-connected* (*v-connected*) if they are neighbors and have the same y (x)-coordinates. All h-connected or v-connected nodes that have the same potential costs are said to be *brothers*; the one with a smaller future cost is said to be *younger* than the other.

The $\alpha - \beta$ router is a labeling algorithm. Each node v_j in G has two potential costs (or simply called labels) associated with it. The labels p_j (q_j) denotes the potential cost of a path from the source v_s to v_j, where v_j is h-connected (v-connected) to the last intermediate node in the path. To label an unlabeled node v_j from a labeled node v_i means to draw a directed arc from v_i to v_j in G and assign a potential cost(label) to v_j. Every label in G is either *temporary* or *permanent.*

Example: In Fig. 12(a), a sample routing region is shown. Two possible paths from the source to target are also indicated. The dotted path is the shortest length path (length 10), but it also involves 6 bends. The Lee algorithm would, however, pick this path because of its optimal length nature. The other path has a length 14 but has only 2 bends. Suppose the cost of a bend is 3 times that of an arc, *i.e.*, $\beta = 3$ and $\alpha = 1$. Then in this case, clearly the latter path is cheaper (cost $= 14 \times 1 + 2 \times 3 = 20$) than the first (cost $= 10 \times 1 + 6 \times 3 = 28$).

Fig. 12(b) shows the DAG constructed in the process. Note that if the source is assumed to be at (0,0).

Algorithm 5 LABELING PHASE OF $\alpha - \beta$ ROUTER

Input: An undirected grid graph G, the source node v_s and target node v_t

Output: A DAG with permanently labeled nodes including the target

Begin

1. *Initialization*:
   Set $p_s = q_s = k$ where $k = |x_s - x_t| + |y_s - y_t|$;
   Set p_j and q_j for all other nodes to ∞;
   Mark p_s and q_s as permanent; Set DAG $\leftarrow v_s$;
   Set p_j (q_j) of all v_s's h-connected (v-connected) brothers v_j to k. Starting from oldest to youngest, mark p_j and q_j permanent and include the nodes v_j into the DAG.
2. For each node v_j which has a newly marked permanent label do
   Let v_k be a neighbor of v_j.
   - Set $p_k = \min(p_k, p_j, q_j + \beta)$ if v_k is h-connected from v_j in the forward direction;
   - Set $p_k = \min(p_k, p_j + 2\alpha, q_j + \beta + 2\alpha)$ if v_k is h-connected from v_j in the backward direction;
   - Set $q_k = \min(q_k, q_j, p_j + \beta)$ if v_k is v-connected from v_j in the forward direction;
   - Set $q_k = \min(q_k, q_j + 2\alpha, p_j + \beta + 2\alpha)$ if v_k is v-connected from v_j in the backward direction;
3. Stop if there is no temporary label $< \infty$ or if the target is $\in$ DAG;
4. Pick the node v_j with the smallest temporary label (In case of a tie the node with smallest future cost is chosen);
   Mark label permanent and include v_j in DAG, if it is not already present;
5. If p_j (q_j) is chosen in Step 4 then set p_k (q_k) of all v_j's h-connected (v-connected) brothers v_k to p_j (q_j).
   Starting from oldest to youngest, mark these labels permanent and include the nodes v_k into the DAG, if they are not there already;
   Goto Step 2.

End

Algorithm 6 MIKAMI-TABUCHI LINE-SEARCH ALGORITHM
Begin

```
1. Initialization :
        i = 0;
        CSP (Current Source-originated Probes)= Hor.  and Ver.  probe from source point;
        CTP (Current Target-originated Probes)= Hor.  and Ver.  probe from target point;
        OSP (Old Source-originated probes) = Φ;
        OTP (Old Target-originated probes) = Φ;
2a. If any probe p ∈ CSP intersects any probe q ∈ CTP+OTP then
      goto Step 5;
2b. If any probe q ∈ CTP intersects any probe p ∈ CSP+OSP then
      goto Step 5;
3.  OSP ← OSP + CSP;
    OTP ← OTP + CTP;
    i = i+1;
4. Generate probes perpendicular to each probe p in CSP (CTP), starting from
   each grid intersection with p.  Let these be level i probes
   Replace CSP(CTP) with all the new probes generated thus.
   goto Step 2a;
5. Backtrace from the point of intersection of the two probes p and q
   to both of the terminals to generate the required path.
```

End

then the target coordinates are (6,4). Hence $k = 6 + 4 = 10$. Nodes (1,0),(0,1), (0,2), (0,3) and (0,4) are the brothers of (0,0). All of them have the same potential cost of 10. Node (0,4) which has the smallest future cost amongst them is the youngest. Node (1,1) gets assigned a label $p_j = 13$ because it is h-connected to (0,1) in the forward direction. Similarly (1,1) has $q_j = 13$ because it is v-connected to (1,0) in the forward direction. Node (0,5) has $q_j = 12$ since it is v-connected to (0,4) in the backward direction. We stop once the target is included in the DAG. The path which is detected is shown in Fig. 12(a).

Discussion: The $\alpha - \beta$ router has a worst-case complexity of $O(n \log n)$. This is because at least one label becomes permanent in each iteration. Hence, we would have to make at most $O(n)$ passes. The dominating factor in each pass is to find the temporary label with minimum cost. This can be done in $O(\log n)$ time with a priority queue. The need to keep two costs at each node arises because two paths with different costs can reach the node, but the costlier one may eventually turn out to be cheaper due to the additional cost of turning 90° for the other path.

The cost functions can be tailored to various situations. For instance, we can define the potential cost of a node as the sum of the current cost and twice the future cost. This would make the search make a bee-line towards the target even faster. However, the path found may not be optimal with respect to the current cost. Horizontal and Vertical arcs can be assigned different costs.

2.7 Line–Probe Routers

One of the main disadvantages of the Lee maze router is the enormous amount of time and space that it needs. An interesting strategy, commonly referred to as *line-probe* or *line-search* routers, was independently developed by Mikami-Tabuchi [37] and Hightower [15]. These routers can be expected to run much faster than the maze-running algorithms, when the routing area is not congested. They store only features already placed on the routing surface, instead of the entire grid. This can also reduce the memory needs.

The basic idea of line-probe routers is to project probes (line-segments) from a point, extending them until either an obstacle or the edge of the routing area is met. Probes are either horizontal or vertical in orientation. The point from which a probe is projected is referred to as an *escape* point.

Discussion: The Mikami approach is in fact a breadth-first search like the Lee algorithm. It is also a gridded approach. It guarantees a solution, if one exists. However, it does not guarantee to find the shortest length path. In general, if a source-originated probe p of level i intersects a target-originated probe of level j, the resultant path will have $i+j+1$ bends. Minimizing the number of bends is possible by an appropriate level scheduling of probes.

Proper data-structures should be designed to efficiently search for obstructions to a probe. A typical organization is as two sorted lists: one for vertical lines and the other for horizontal lines. All co-horizontal and co-vertical lines are grouped together. Since a horizontal probe is only limited by vertical lines, such an organization would allow us to ignore lines parallel to the direction of the probe in the search for obstructions. For n initial obstacles and e probes, the storage required is, therefore, $O(n+e)$. The worst case time complexity could be as large as $O(e(n+e))$. For a congested grid of size $N \times N$, e and n could each be as large as $O(N^2)$. Hence the total time complexity gets to be $O(N^4)$. Thus, it becomes evident that the Mikami approach is suitable only when the number of probes generated is small. And even then, the gain seems to be more in terms of space than time.

A similar algorithm was developed by Hightower [15] around the same time as the Mikami algorithm. The main difference was in the manner in which the new probes were generated. Hightower's algorithm adds only one additional probe (*escape line*) to each existing probe. He describes three escape processes to allow the router to find a path around different types of obstacles. For instance, in case of a probe parallel to a wiring segment, the new probe is placed just beyond the endpoint of the segment. Thus, the Hightower router will generate fewer escape lines than the Mikami algorithm. Still, in the worst case, the number of escape lines could get as large as $O(n^2)$. However, when the number of bends in the path is small, the Hightower router is generally faster. For a path with l bends, we would have to generate at most $4l$ probes. Hence, for a linked list implementation, the total time is $\sum_{i=1}^{4l}(n+i) = O(nl+l^2)$. The Hightower router however does not guarantee finding a path, even if one exists. Some claim that this is better than finding a complex path that may block many future connections. They contend that such nets are best left eventually to a cleanup router.

For a simple routing problem shown in Fig. 13, we find that both the Mikami and Hightower algorithm discover two paths, one comprising of 2 bends (minimum) and the other consisting of three bends. However, the Mikami algorithm generates 22 escape points in contrast to the 4 generated by the Hightower algorithm. In case of less congested grids, while employing the Mikami algorithm, one also has to seriously consider bounding the routing region to prune the generation of unnecessary escape points.

2.8 Line-Search Routing

Ohtsuki, *et al.* [127], has designed a new line-search algorithm that is guaranteed to run in $O(n\log^2(n))$ using $O(n)$ space, given an n-vertex rectilinear routing region. The main idea is to use the priority search tree data structure, first proposed by McCreight [128] to efficiently solve the line-intersection problem.

Given a set of vertical segments, find a horizontal slice line such that the number of line segment endpoints above and below the bisector are balanced. We associate the root of an external tree with this bisector and the line segments intersecting it. The latter are represented as two priority search trees, one for the cut segments below the bisector and the other for those below. The left and right subtrees of the root is associated with those line segments that lie entirely below or above the bisector. The process is recursively repeated till each line segment is cut by a bisector. A similar construction is possible for horizontal wire segments too. The main advantage of such an *embedded* priority search tree structure is in solving the problem : for a set D of horizontal(vertical) line segments and a vertical (horizontal) line segment query h, find all line segments in D, or the minimum (maximum) line segment, that intersect h. Typically $O(\log n)$ priority search trees need to be searched, each of which takes $O(\log n)$ time. Thus if k intersections exist, the total time needed is $O(\log^2 n + k)$. Ohtsuki's algorithm uses this data structure extensively.

Since, each escape line is considered for intersection only once, and there are at most $O(n)$ escape lines, the total running time is $O(n\log^2 n)$. A simple problem is illustrated in Fig. 14.

Algorithm 7 Ohtsuki's Line-Search Algorithm
Begin

1. *Initialization* :
   Extend each edge of the routing region in both directions as much as possible. These are called escape lines. In addition generate two escape lines h_T and v_T incident at the target T. All the horizontal escape lines H are placed in one embedded priority search tree as described above. Similarly, the vertical escape lines V are placed in another. Horizontal and vertical line segments incident at the source S are generated and placed in a queue.
2. Pick up a line from the queue. Using the embedded priority search tree, enumerate all escape lines that intersect it.
3. If one of them is either h_T or v_T, then backtrace to generate a path
4. Otherwise, delete the enumerated line segments from the priority search tree and put them into the queue. Goto step 2.

End

2.9 Rip-Up and Rerouting

Autorouters which route one net at a time, such as the mazerunning algorithms, suffer from a fundamental drawback in that they provide no feedback or anticipation to avoid conflict between nets or to ensure that some early connection will not prevent the successive connections. Typically, such failed interconnections account for less than 5% of the total number. However, the manual effort required to complete these final few nets is often quite considerable. Hence, nowadays a lot of emphasis is being placed on routing tools that have automatic rip-up and rerouting ability so as to achieve 100% successful interconnection. One approach adopted by Mighty (see the section on switchbox routers) [66] and Dee in [13] was to push aside existing nets so as to make room for a blocked net. Mighty also tries and removes existing interconnections so as to connect the blocked net earlier. We shall briefly discuss here, a graph-theoretic approach to rip-up and rerouting as reported first by Suzuki *et al.* [27].

Consider the example shown in Fig. 4(a). The failure of the Lee algorithm in routing net C can be attributed to the fact that its terminal pair has been blocked by some already routed nets. Such a blockage can be described by the graph-theoretic term *minimal separator.* * The essential minimal separators for our purpose are those consisting only of the cells occupied by existing interconnections. Among them, the two particular ones of interest are M_S (reachable from the starting pin of the blocked net C) and M_T (reachable from the target pin of net C). These can be easily computed by a simple modification of the basic Lee algorithm. Fig. 15 shows the two sets M_S and M_T marked with a $\times$ and O respectively. It so happens that both these sets contain the same three cells. This is not true of a general case.

Minimal Separator Computation: Propagate a wave starting from S, marking the reached occupied cells as a. Another wave is propagated from T, marking the reached occupied cells as b. All cells which are labeled with both a and b are relabeled as c. A new wave starting from a (similarly b) and proceeding through both empty and occupied cells is propagated until it hits a cell marked b (a, respectively); the latter is remarked as d (e, respectively). At this point, we will have the two minimal separators given by $M_S = \{\{c\} \cup \{e\}\}$ and $M_T = \{\{c\} \cup \{d\}\}$.

The second step is to rank the previously routed nets W by a *reroute effect index*, $P(W)$. The larger the value of the index, the higher is the chance of successful rerouting of W. Let $N_S(W)$ and $N_T(W)$ denote the number of cells of W contained in M_S and M_T. Also let $L_S(W) = 1$ (or $L_T(W) = 1$) if the terminal pair of W lies on opposite sides with respect to M_S (or M_T); otherwise it is equal to 0. Then

$$P(W) = \min\{N_S(W) - L_S(W), N_T(W) - L_T(W)\}$$

For our example, we have $P(A) = \min 0 - 0, 3 - 0 = 0$ and $P(B) = \min 3 - 0, 3 - 0 = 3$. Hence, clearly it is

*For any two nodes S and T in a graph, the minimal S-T separator M_{ST} is defined as the smallest set of nodes such that any path connecting S and T will contain at least one node from M_{ST}

better to reroute net B. In fact, for the rerouting of a net W to be successful, we have to have $P(W) \geq 1$. Note also that removing a net which satisfies this criteria, does not guarantee a successful rerouting. This case occurs when removal of a blockage still cannot create a path between the terminal pair under consideration because of other preassigned nets. In such case, we could try to rip up two nets simultaneously and to reroute them. The reroute effect index for two preassigned nets W_1 and W_2 is defined by

$$P(W_1, W_2) = \min\{N_S(W_1) - L_S(W_1) + \{N_S(W_2) - L_S(W_2), N_T(W_1) - L_T(W_1)N_T(W_2) - L_T(W_2)\}$$

Another way of ensuring success is to reroute preassigned nets traversing as few of the minimal separators as possible. This idea has been incorporated into the *X router* as part of the WIRES system [3]. The hardware accelerators described earlier can also be used to provide additional speed up.

3 Channel Routers

3.1 Introduction

Channel routing is one of the most important and possibly widely used methodology for wire routing of large integrated circuits. Prior to channel routing, a channel decomposition and global routing phase is carried out that decomposes the entire routing region into several smaller and tractable channels. Some terms that will be frequently used are (see Fig. 16):

A *channel* or routing region is a rectilinear polygonal region between circuit blocks that are to be connected. All terminals are fixed on two opposite sides of the channel, typically the top and bottom edges.
A *trunk* is the horizontal part of a net. It is connected to the top and bottom of the channel by vertical wires called *branches*.
A *track* is used to refer to the line along which a trunk is placed. A *row* is synonymous to a track when there are more than two layers.
If a net can be split between several tracks, then the vertical wire segments connecting the horizontal pieces are called *doglegs* or *jogs*.
The *width* of a channel equals the number of horizontal tracks available for routing.

Mathematically, the channel routing problem for n layers can be stated as:

> Given a set of layers $l_1, l_2, \ldots l_n$, two ordered lists T_L and B_L consisting of terminals on the top and bottom edges of the channel, and two unordered lists L_L and R_L specifying the nets that exit on the left and right edges of the channel respectively, perform the connections specified by the terminal lists T_L, B_L, L_L, and R_L as per the given design rules specifications.

The goals may also vary depending on the layout style. In gate-array chips, the width of the channel is fixed and the main objective is to connect all the nets. In standard-cell chips and custom-designed chips, we can always increase the width of a channel to ensure the feasibility of the routing; hence, the main objective is to minimize the channel width while connecting all the nets.

3.2 Channel Routing Models

Depending on the number of layers available for routing, as well as, the permissible grid structure, we can form the following classification

Planar Model : This model is very restrictive, in the sense that wires cannot cross each other. The model is mainly used for such tasks as routing the chip inputs and outputs to the pads on the chip boundary. The model also goes under the name of *river* routing, since rivers cannot cross one another without their waters merging. Efficient polynomial-time algorithms are available to solve this problem.

Manhattan Model : Most of the detailed routers, we shall present, belong to this category. Wires are allowed to run in the horizontal or vertical directions. Furthermore, wires may cross one another, but may not bend away from one another forming what are referred to as ***knock-knees*** (see Fig. 17).

Knock-knee Model : This is similar to the Manhattan model, except that knock-knees are also allowed. The approach, though theoretically attractive, owing to its resemblance to multicommodity flow problems, is not widely used in practice. The presence for knock-knees complicates the layer-assignment problem. It is not amenable to fabrication technologies that require assignment of preferred layers to wiring layers. When three or four wiring layers are used, it follows that there is no horizontal overlap between wires. Further, if vertical overlaps are also excluded (such an assignment is always possible for four layers), it is referred to as the *k-layer knock-knee model.* The 4-layer knock-knee model also suffers from sparsity of the resulting layout.

Multi-layer Model : We have already defined the multi-layer knock-knee model. A less restrictive model is called a *$(L \mid k)$-Layer Model.* This allows wires that are at least L layers apart to run on top of one another anywhere in the grid graph comprising of k layers. The other important model is referred to as the *Directional Multilayer Model.* These models attach a preferred wiring direction, horizontal (H) or vertical (V), to layers. Thus a k-layer model could be specified as a string of letters from $\{H, V\}$ that define the wiring direction on layers $1, 2 \ldots$ Thus a HVH model refers to one wherein wires run in the H direction on layers 1 and 3; and in the V direction on layer 2.

For the Manhattan routing model, where doglegs are not allowed, two constraints can be identified:

Horizontal constraints : These exist when the set of columns spanned by two nets intersect, since their trunks cannot be both assigned to the same track on the same layer.

Vertical Constraints : These are present when terminals for two different nets a and b, exist at the same column i. If the terminal for net a lies on the top side of the channel, then we say that net a is *above* net b, or equivalently net b is *below* net a.

Two graphs, known as the *Horizontal Constraint Graph* (HCG) and the *Vertical Constraint Graph* (VCG), are often quite useful. In each case the node set is the set of nets in the routing problem.

With every net i, associate an interval or zone I(i), where left point of I(i) is the minimum column number k where a terminal for i occurs and right point is likewise the maximum column. HCG is then the intersection graph of this system of intervals. In other words, there exists an edge between two nodes a and b in the HCG, iff the nets they represent have a horizontal overlap. For the VCG, a directed edge is drawn from node a to node b, if net a is *above* net b.

Let $vmax$ be the length of the longest path in the VCG from a source to a sink. This provides a measure of the difficulty in satisfying all of the vertical constraints. Note that if the VCG is cyclic, it is impossible to eliminate all vertical constraint violations, using the Manhattan two-layer H-V wiring model. Similarly, the number of nets that span a specific column is called the *local* density of the column. A maximal clique in the HCG defines a *zone*. The clique number is the zone density. The maximum value of zone density over all the zones is known as the *channel* or *global* density. The global density, denoted d, gives an absolute lower bound on the number of tracks required.

The two constraint graphs for the channel routing problem of Fig. 18, are shown in Fig. 19 and Fig. 20. This particular problem can be solved without using doglegs, since the VCG is acyclic ($vmax = 2$). The HCG has 4 maximal cliques, *viz.*, 123, 234, 349, and 356789. This means that the channel density is 6 and thus any 2-layer solution will require at least 6 tracks.

A better estimate of the lower bound t_{min}, taking into account the vertical constraints and assuming uniform technology, would be $t_{min} = \max(\lceil \frac{d}{n_h} \rceil, vmax)$, where n_h is the number of horizontal layers available for routing. Thus, for the HVH model, we will need at least $\max(\lceil \frac{d}{2} \rceil, vmax)$ tracks. However, for the VHV model, d rows are both necessary and sufficient to do the routing. This is because, we can assign all upward vertical segments to one V layer, and all downward vertical segments to the second V layer, thereby eliminating all vertical constraints [56]. Thus, choice of the proper wiring model is critical to minimizing the number of rows required. Amongst HV, HVH, and VHV wiring models, we would therefore

- use VHV when $d < vmax$;
- use two-layer routing when $vmax < d < 2 \cdot vmax$; and
- use HVH when $d > 2 \cdot vmax$.

Algorithm 8 LEFT-EDGE ALGORITHM
Begin

1.Let d be the channel density as given by the HCG;
Let $t_1, \ldots, t_d$ denote the routing tracks.
2.Sort the set of intervals, computed in the construction
of the HCG in the ascending order of absiccas of their left ends.
Let $I_1, I_2, \ldots, I_n$ denote the sorted set.
3.Assign I_1 to track t_1.
3a.For $j = 2$ to n do
Assign net I_j to an arbitrary available track t_j.
4.Once, the horizontal tracks are assigned, the vertical branches are
simply assigned to either the TOP or BOT layer depending
on whether they are upward or downward directed.

End

The two-layer channel routing problem has been shown to be *NP-complete* [55] even when all nets consist of just two terminals. Hence good heuristic algorithms are required. We shall discuss the Left Edge Algorithm, the dogleg router (Deutsch), the greedy router (Rivest) and YACR2 (Reed). All four algorithms are based on the Manhattan routing model.

3.3 The Left-Edge Algorithm

This algorithm, popularly known as LEA, was introduced by Hashimoto and Stevens [46], to solve the two-layer channel routing problem for PCBs. It yields optimal routing using the minimum number of tracks. However, it can be applied only in the absence of vertical constraints. Furthermore, it does not allow doglegs and the H-V wiring model is strictly enforced. These conditions may seem too restrictive for modern channel routing problems. But for PCBs, it is often possible to *jog* one of the conflicting vertical segments, since there was enough space between adjacent pins for this purpose.

Discussion: Step 3a) is always possible because otherwise it would imply that the left edge of I_j intersects with previously assigned intervals at every track $t_1, \ldots, t_d$. This would make the density $d + 1$. A simple example, with $d = 4$ is shown in Fig. 21(a) and (b). The worst case complexity of LEA is $O(n \log d)$, where d is the channel density. The sorting of the nets by their starting abscissae can be done in $O(n)$ time by employing a radix-sort technique over the interval [0,n].

An optimal solutions to the constrained LEA problem, with no doglegs, was first proposed by Kernighan [48]. He suggested the use of a branch and bound technique. The idea is to exhaustively enumerate all possible assignments of net segments to tracks satisfying vertical constraints. The search space can be pruned by eliminating nodes that are guaranteed to have a higher cost than the best one found so far. Further compaction is made possible by allowing doglegs. The dogleg router, which is the next to be discussed, illustrates the modifications, to the basic LEA algorithm, required to allow vertical constraints, doglegs etc.

3.4 The Dogleg Router

The Dogleg Router was introduced by Deutsch in 1976 [39] and inspired the development of more powerful heuristics for the general channel routing problem.

The necessity for doglegs is rather a moot point and is quite data dependent. From an electrical standpoint each dogleg adds one or two contacts and this increases the capacitance. From a mathematical standpoint, if a dogleg occurs at a position where the local density is equal to the channel density, then we need an extra track to perform the routing which increases the area and is contrary to our objectives. Thus, it is necessary to be judicious while introducing doglegs.

Algorithm 9 DOGLEG ROUTER
Begin

1. Split each k-terminal ($k > 2$) net into $k-1$ two-terminal nets. Choose a *range* parameter, which determines the number of adjacent two-terminal subnets of an original net that have to be placed on the same track. Choose a routing sequence. The first track can start at any one of the four corners of the channel. The second will then be at one of the two corners at the opposite end of the channel. Thus one of the 8 such sequences would be *Top Left* → *Bottom Right* ...
2. Apply the Left Edge algorithm to this collection of two-terminal nets taking into account the *range* and routing sequence constraints. Subnets of the same net are allowed to share the same track .

End

Deutsch observed that the primary reason why routers use more tracks than the theoretical minimum (channel density) is due to a few critical nets, like clock lines, which are heavily connected to both sides of the channel and cause long constraint chains. In such cases, doglegs can aid in breaking up the chains, leading to better solutions. Also since these nets have numerous terminals, it is sufficient to limit doglegs only to terminal positions. This significantly reduces the number of doglegs than will otherwise be used. Furthermore, each dogleg now requires only one additional contact, thus reducing the capacitance.

Another situation where doglegs can be attractive, is in the presence of constraint loops. Doglegs can break such loops, and make routing possible. For example, the problem of routing two nets A and B, where A is above B at the first column and vice versa in the next column requires the use of a dogleg. This should again be taken with a pinch of salt, for the loops could have been eliminated by suitably reflecting or mirroring the terminal positions.

Discussion: The *range* parameter keeps the number of doglegs introduced to a minimum. Setting the *range* to a very high value will typically lead to a non-dogleg solution. Of course, an exception is made when the number of subnets of a net that remain to be placed is less than the *range*. Varying the routing sequence can also reduce the vertical height of the channel and tends to reduce the average capacitance of a net.

Dogleg routers have been extended to handle off-grid terminals, such as when polycells with one terminal spacing share the channel with PLA's or pads with a different terminal spacing. The complexity of the dogleg router is essentially that of LEA. Hence, it can be implemented to run in $O(n \log d)$, where d is the channel density. Dogleg routers give superior results over LEA, often requiring very few tracks over the channel density. Their major drawback is that they do not operate with cyclical vertical constraints, which is a problem where manual intervention is not possible, such as in silicon compilers.

Example: The following example illustrates the general dogleg algorithm with off-grid terminals. All spacings are in terms of $\delta = 1.25$ microns. The value of *range* is set to 1. A *Top Left* → *Bottom Left* routing sequence is employed. The dogleg solution is shown in Fig. 22(b).

The resulting solution uses two doglegs, three tracks and 14 contacts. The total area required is $32\delta \times 28\delta = 896\delta^2 = 1120\ sq.mm$. Note that since the terminals were misaligned, the number of vertical constraints increased. In general, this can have a significant effect on the quality of routing.

3.5 Greedy Channel Router

The *greedy* router is the work of Fiduccia and Rivest [53]. It is the first of two routers that we present that are general-purpose extensions of LEA. They are also ***unrestricted*** dogleg routers meaning that doglegs can occur at any column, not necessarily where a terminal pin of the doglegged net occurs.

The greedy router performs a left-to-right column-by-column scan of the entire channel, completing the wiring within a given column before proceeding on to the next. In each column, the router employs several *greedy* heuristic measures to optimize the channel width. Hence, its name. However, it can sometimes "overkill" a problem, using an excessive number of vias and vertical jogs than are necessary. Nonetheless, it

Algorithm 10 GREEDY CHANNEL ROUTER
Begin:

1. While there remain nets to be routed do
   a) Make vertical connections between the pins at the top and bottom of the channel and the nearest track that is either empty or contains the same net as the pin being connected.
   Preference in this matter is given to an empty track, even though this may lead to a net occupying more than one track. If both pins in the current column belong to the same net, and no other pins in the channel are to be attached to this net, a vertical line is drawn connecting them. No action is taken, however, if a pin cannot be connected to existing tracks

   b) Add jogs between tracks assigned to the same net.
   The track farthest from the center of the channel is always released. If more than one set of tracks can be combined, combine the set of tracks that will release tracks farthest from the center of the channel.

   c) Add vertical connecting jogs (doglegs) to reduce the distance between tracks assigned to the same net have not yet been combined.

   d) Add jogs to the circuit to move nets closer to the edge of the channel where their next terminals lie.
   Preference is given to nets having terminals nearest to the current column.

   e) If necessary, add additional tracks to accommodate nets that could not
   The new tracks are placed as close to the middle of the circuit as possible. After inserting a track, the remaining tracks are renumbered. Previous routings, however, are left unchanged.
2. Extend all nets which have pins remaining to be connected to the next column with appropriate horizontal wiring
   end while

End

always provides a solution even in the presence of cyclic vertical constraints, often within one track of the optimal solution.

Discussion: Rivest and Fiduccia observed that the *initial-channel-width* used by the router has a significant effect on the quality of the solution produced. Too small a value for this parameter would require a large number of tracks to be added to the circuit, making poor use of the tracks added later to the previously routed columns. Values too large also tended to waste circuit area. One suggestion is to use the channel density as an initial guess and use the result to perform other iterations. Also, a way to reduce unnecessary vias and vertical jogs, is to employ two parameters: (i) a *minimum-jog-length* specifying the smallest vertical jog that can be made, and (ii) a *steady-net-constant* which prevents a jog from being added in Step 1(d), if the net has two terminals on opposite sides that are less than this many columns away.

The fundamental data structure needed for the router is a set $Y(E)$ for each net E of y-coordinates of the tracks currently occupied by the net E. The algorithm performs a fairly constant amount of work for each column. Steps 1*b*) and 1*c*) do require exhaustive search to identify tracks that can be combined or brought closer together. In practice, however, most columns contain only a few such sets of tracks. The lookahead performed in Step 1*d*) is also curtailed by the *steady-net-constant* value. The overall complexity, on the average, therefore, is $O(c)$, where c is the number of columns in the channel.

Example: Fig. 23(a)-(d) demonstrate the effect of the various heuristics outlined. The example uses a

Algorithm 11 YACR2
Begin

1. Reserve d tracks at first. Let c be a column of maximum density d.
2. *Select* and *Assign* to available tracks, nets that cross c, those that are to its right and finally those that are to its left.
3. Place all vertical segments in columns where there are no VCVs;
4. Place the remaining vertical segments using the *maze*1, *maze*2, and *maze*3 patterns in that order. If none succeed add an extra track at each possible location across the channel and re-apply *maze*2. If still unsuccessful add a track and Goto Step 2;

End

minimum-jog-length of 1 and a zero *steady-net-constant*. The final solution needs 6 tracks, even though the channel density is only 5. This is because of its *greedy* nature, the router searches for local optimums, consequently creating situations where it becomes necessary to increase the width of the channel. The routing also produces 18 vias, many of which are redundant if we use a 2-layer grid model.

3.6 YACR2: Yet Another Channel Router

YACR2 is the work of Reed *et al.* [52]. The algorithm works in two phases: a horizontal track assignment phase, which is a modified LEA, followed by a maze-running phase. In the first phase, horizontal segments are packed in a channel of width equal to the density, in such a way that the number of vertical violations are minimized. All admissible (non-conflicting) vertical segments are then embedded. The other interconnections are performed by the invocation of three different maze-running techniques.

Discussion: The *select* and *assign* procedures are fairly complex and incorporate several heuristics that have been found to yield excellent practical results. Let p be the longest path involving net n in the VCG. Let there be t nodes before n and b nodes after n in p. Let $S(n)$ be the set of tracks where the horizontal segment for n causes no horizontal violations. Let $T(n), B(n)$ and $P(n)$ represent the top t, bottom b, and the remaining tracks respectively.

The first choice is to try and assign net n to a track in $S(n) \cap P(n)$ so that there will be definitely no VCV at this time. The second choice is to pick a track from $S(n) \cap T(n) \cap B(n)$. This ensures that the vertical segments connecting the top and bottom pins on the column overlap for as few rows as possible. As a last resort, a track in $S(n) \cap \{T(n) \cup B(n)\}$ is picked. In case of a tie, the track closest to the *ideal* row is chosen.

The *maze-running* techniques used in YACR2, in reality, only do *pattern* matching. The patterns are designed so as to avoid the overhead of a complete maze search and also help minimize the number of additional vias that get introduced. In Fig. 25-27, a vertical constraint violation is present at column j, and net a is connected to the top edge and net b to the bottom edge of the channel at j. Also a and b have been assigned tracks p and q, where p is above track q.is

All three patterns attempt to eliminate the VCV at column j by doglegging one net in an adjacent column. Note that *maze1* allows a short *horizontal* jog on the vertical layer. Thus, YACR2 is not a true Manhattan router, but rather uses a two-layer model. These patterns can easily be checked by suitable geometrical methods and are quite fast. A balanced tree like dynamic data structure can be used to maintain the set of intervals crossing the sweep line.

3.7 Multi-Layer Channel Routers

Though multilayer channel routers with unlimited overlap are possible, they often produce unsatisfactory results in practice. Hence, most attention has been directed towards the directional and the (L out of k)-layer models. We present three routers for directional multilayer channel routing: (i) a net-merging approach by Chen and Liu [56], (ii) Converting Manhattan routing to channel routings with 3 or 4 layers [58], and (iii) Chameleon[41], a multi-layer router based on YACR2-like ideas.

Algorithm 12 HVH CHANNEL ROUTING USING HV SOLUTIONS
Begin

```
1.Call a two-layer channel router to obtain a two-layer solution S;
2.Perform optimal track permutation on S;
3.Do a relative-order-rearrangement of S;
4.While S still has vertical violations on tracks do
        If S has two adjacent singular tracks then
           remove all adjacent vias between them and reconnect using a
           maze router if possible;
        Else If S has singular tracks then
                do singular track shifting;
        Else
                insert empty track;
5.  Transform S into a three-layer solution.
```

End

3.8 Extension of HV Solutions to HVH Models

Cong *et al.* [58] have developed a new technique for transforming the two-layer routing solution into one for a three-layer HVH model. The idea is to merge tracks without introducing any new vertical constraints. The result of this process is a *perfect pairing*. Some tracks may need to be paired up with an empty track. These are referred to as *singular* tracks.

Discussion: *Track Permutation*: The key problem here is to get a perfect pairing using a minimum of empty tracks. Given the two-layer routing solution S, we proceed by constructing a *track-order graph TOG(S)*. This is a directed graph, in which each vertex v_i corresponds to a track t_i ($1 \leq t_i \leq w$) and there is a directed edge (v_i, v_j) if at some column, there is a via on track t_i above a via on track t_j. If, $TOG(S)$ is treated as scheduling constraints, the problem is similar to the two-processor scheduling problem for which efficient polynomial, in fact linear-time, algorithms [130] have been developed. † If, in the solution, job j_i is assigned a processor during the time interval x, then track t_i is placed in the xth track pair. An empty track, therefore, corresponds to a processor being idle during the corresponding time interval.

Relative-order-rearrangement of S This step attempts to reduce the number of vertical violations between adjacent tracks by reordering tracks within a pair. The method used is to find the shortest path in a directed graph called the *track-group-order* graph $TGG(S)$. The graph is constructed as follows: For each track group containing a singular track t_k, there is a vertex labeled k. If the track group contains a pair of tracks t_i and t_j, then we introduce two nodes ij (corresponding to placing track t_i above t_j) and node ji (corresponding to placing track t_j above t_i). There is a directed edge (u, v), if their corresponding track groups are adjacent and the one corresponding to u is above the one corresponding to v. The weight of each edge is the number of adjacent vias between the two track groups when the track arrangement is consistent with the corresponding vertex's labeling. In addition, a start node s and a finish node t are added. s (t) is connected to all nodes corresponding to the first (last respectively) track groups by zero-weight edges.

Singular Track Shifting: This procedure is aimed towards bringing together singular tracks, with the hope of performing local rerouting to fold them together. Shifting is not the same as permutation. It does not reorder the tracks, but only modifies the folding pairs. Supposing we have four tracks a, b, c, and d in the channel after the permutation procedure. Tracks b and c may be paired up by the local rerouting procedure. Therefore, we have track a, track-pair (b, c), and track d as our final solution. Singular track shifting will undo the folding of b and c and then try to pair up a and b. If we succeed, c and d are now singular and have greater chance to be folded by local rerouting.

Complexity: Let c be the number of columns in the given channel routing problem and let w be the number of tracks used in the two-layer solution. The above algorithm can then be shown to run in $O(w^3c^2)$ if the

† A two-processor scheduling is a problem in which a set of unit-length jobs, with prespecified scheduling constraints, are to be selected for execution on any two of the processors such that they minimize the completion time of all jobs.

two-layer solution allows unrestricted doglegs; and in time $O(w^2c^2)$ if the two-layer solution allows only restricted doglegs. This is because during each iteration, the maze router takes $O(wc)$ time in the worst case to eliminate a pair of adjacent vias. There are at most $O(wc)$ ($O(c)$) adjacent vias in the case of unrestricted (restricted) doglegs. Furthermore, at most $O(w)$ iterations are required.
Example: A 6-track two-layer solution for the problem of Fig. 23(a), as obtained by the greedy router, is shown again in Fig. 28. The induced perfect pairing obtained by the processor scheduling strategy is shown in Fig. 29. In the TGG shown in Fig. 30, $s \rightarrow 13 \rightarrow 2 \rightarrow 46 \rightarrow 5 \rightarrow t$ is the minimum cost path of length 1. Fig. 31 shows the final optimal three-track HVH wiring solution after singular track shifting.

3.9 Merging Algorithm for Three-Layer Channel Routing

Merging two nets means that they share a track. Two nets i and j can be merged if they have no mutual horizontal or vertical constraints. Merging amounts to

1. Modifying the VCG by coalescing the nodes i and j into one super-node $i \cdot j$, and
2. Substituting the zones of net i and net j by the smallest interval, $i \cdot j$, that includes both.

Yoshimura and Kuh [59] had suggested the merging heuristic for solving the 2-layer Manhattan channel routing problem. Chen and Liu extended this approach to handle 3-layer HVH or VHV type channel routing [56]. The latter is uninteresting as no vertical constraints exist and LEA suffices. For the HVH approach, the vertical constraints still exist. Merging can be of two kinds:

1. *Serial Merging*: Two nets i and j can be serially merged (placed on the same track and layer), if they do not impose horizontal and vertical constraints on each other.
2. *Parallel Merging*: Two nets i and j can be parallel merged (placed on the same track but on different layers), if they do not have vertical constraints but have some horizontal overlap.

Discussion: The basic idea is that the merge operation cannot introduce a cycle into the VCG. However, it can increase its depth, and since this determines a lower bound on the minimum channel width, the idea is to minimize the increase. The key step therefore, lies in the choice of the two nodes to be merged. The authors, in their paper, have discussed some heuristics that have yielded good results. and take $O(n^2)$ worst-case time for n nets.
Example: Figs. 32 and 33 illustrate the HVH routing solution for Fig. 18. We assume that prior to merging, no nets have been placed on any track. In the zone representation thin lines denote nets placed on the top horizontal layer, while broken lines indicate nets placed on the bottom horizontal layer. The final solution is shown in Fig. 34. It uses only 4 tracks. Since, the channel density is 6 for this example, this yields a saving of 2 tracks over the best possible 2-layer solution.

3.10 Chameleon: A Multi-Layer Channel Router

Chameleon, developed by Braun *et al.* [41], is a very general multi-layer channel router. It addresses several important, though often ignored, issues such as nonuniform technology, obstacles, stacked contacts and the like. It is therefore interesting to consider the Chameleon routing strategy in detail.

The goal of the partitioning phase is to generate subproblems such that the overall area of the wiring is minimized. This can be achieved by maximizing the number of tracks per unit area in the channel, and by keeping the difficulty of routing each of the subproblems at a minimum.

The first step is to determine the wiring model that is to be employed. For uniform technology, the number of tracks per unit area can be maximized by maximizing the number of horizontal layers. For example, when there are 5 layers, one HV and one HVH partition is generated; while for 6 layers, two HVH partitions are preferred. The problem becomes more difficult for nonuniform technology. Both line widths as well as line-to-line spacing for the different layers has to be individually considered in an effort to arrive at an optimal partitioning. Because the tracks on adjacent layers can overlap, new vertical constraints are introduced. Chameleon uses two strategies to deal with this: a *greedy* strategy and a *conservative* strategy. Greedy strategy assumes that all of the horizontal tracks are usable without incurring an area penalty regardless of overlaps. That is , any additional VCVs that are introduced by overlapping tracks are ignored

Algorithm 13 MERGING ALGORITHM FOR HVH ROUTING

Begin

```
1.  For (i=1; i<Number of Zones; i++){
        Initialization:
        P = {set of nets that end at zone i}
        Q = {set of nets that start at zone i+1}
        T =P + {set of nets that are placed on a track such
            that there are no horizontal segments of other nets on the other layer}
2.  While Q not empty do
        Choose a node in the VCG corresponding to a net m ∈ Q, such that
        it lies on the longest path in the VCG and is farthest from either
        source or sink.
        Q = Q −{m};
        If |Q| ≤ |T| then
           3a.  P' = T;
        Else
           3b.  P' = T + Q;

        4.  Find a net n in P' which is neither an ancestor or descendent
        of m in the VCG. Thus these two can be placed on the same
        track.  Also choose the one which minimizes any increase in the maximum
        depth of VCG as this determines also the number of tracks required.
        If no such n exists then
           5a.  Place m on new track; T = T +{m};
        Else
           If n ∈ P then
              5b.  serial merge (m,n); T= T−{n};
           Else
            If n ∈ T then
              5c.  parallel merge (m,n); T= T−{n};
        Else
           If n ∈ Q then
              5d.  parallel merge (m,n); Q = Q−{n};
   end while;
   end for;
End
```

Algorithm 14 CHAMELEON
Begin

Partitioning:

1. Determine which routing model (HVH or HV) is used in each partition.
2. Determine the cost function parameters for each partition.
3. For all nets do
   Place the net in the partition with least cost;

Detailed routing:

For all partitions do

4. Assign horizontal segments to the row that minimizes the number of vertical constraint violations;
5. Connect the terminals to the horizontal segments using a general three-dimensional maze router.

End

during partitioning. The conservative strategy, on the other hand, treats as available only those tracks that do not overlap. The authors have, however, been unable to determine if any one of the two is uniformly superior to the other. Hence, the user is given the ability to fine-tune the partitioning procedure between the two extremes.

The second step is to assign the nets to a partition. There are two factors that affect this process. The first factor is the per-partition upper bound on the channel density, and the other is a measure to characterize the routing difficulty for each partition, such as, the number of cyclic vertical constraints and the length of the longest path in the VCG. The cost function $J(i)$ while assigning a net to the ith partition, used by Chameleon is:

$$J_U(i) = d_i/h_i + p_i + \alpha_1 C_i + \alpha_2 \max(0, d_i - D_i) + \alpha_3 \max 0, p_i - P_i)$$

Here, d_i is the channel density, h_i is the number of H-layers in the ith partition, p_i is the length of the longest path in the VCG, D_i and P_i are the bounds on the density and longest path, and C_i is the number of cycles in the VCG. The constants α_1, α_2, and α_3 can be used to stress the relative importance of each term. For example, if the detailed router cannot handle cycles, it is necessary to set α_1 to a very high value as compared to α_2 and α_3.

For nonuniform technologies, the cost function needs to reflect the penalties of track overlaps. Consequently, the modified cost function becomes

$$J_{NU}(i) = d_i/T_i + p_i + \alpha_1 C_i + \alpha_2 \max(0, d_i - (T_i^{H_1} + T_i^{H_2}) + \alpha_3 \max 0, p_i - \max(T_i^{H_1}, T_i^{H_2})$$

Here, T_i is the number of tracks per unit area, $T_i^{H_1}$ and $T_i^{H_2}$ are the bounds on the number of tracks on the first and second horizontal layers (if three layers). The latter bounds are computed so that the final area of each partition is as balanced as possible.

Using the appropriate cost functions, we assign the nets to the various partitions. At each iteration of Step 3, the net is assigned to all partitions and the cost function is evaluated. This involves adding the net to the VCG of each partition, counting the number of cycles and vertical violations introduced, and find the density of each partition. The current net is then removed from all but the lowest-cost partition. When all the nets have been assigned, the partitioning phase is then complete.

Detailed Routing: Chameleon uses the same ideas as espoused by YACR2 to route both two- and three-layer partitions. In the first phase, the horizontal segments of the nets are assigned to rows by using a modified left-edge algorithm. Each segment is placed such that a cost function that accounts for vertical constraint violations is minimized. A three-dimensional table, denoted $A = [a_{nrl}]$, is used to represent the cost of assigning net n to row r of layer l. In considering net m for assignment to layer k, its ideal position is is selected by choosing row r such that a_{mrk} is minimized among all rows.

Chameleon, then uses a general three-dimensional maze router to perform the vertical connections. YACR2, on the other hand, uses only a limited set of patterns. Hence, Chameleon outperforms YACR2 on some 2-layer problems as well, albeit at higher cost. The maze routing operates by causing a three-dimensional wave to propagate from the starting cell in all directions. A sort of bounding-box approach is employed, and the cost function is tailored so as to produce routing that is unlikely to block unrouted columns.

Discussion: The demerit of the general maze router have already been discussed. The same apply to Chameleon as well. However, the routing solutions are uniformly of a high quality. Even though, the three-dimensional matrix representation represents a lot of storage, it can be used to easily tackle other problems, such as stacked contacts, obstacles within the channel, etc. Some of them merely require us to tag the corresponding entries with a high cost. It has been integrated in a synthesis system being developed at the University of Berkeley.

4 Switchbox Routing

In building-block layout, when four channels meet at a "+" (cross) intersection, the region of intersection defines a rectangular routing region called a *switch box*. Thus a switch box can be thought of as a channel with fixed pins on all four sides. The switch box routing problem is harder than the channel routing problem. Whereas the main goal of channel routing is to minimize channel width, the chief goal of switch box routing is to make 100% connections according to the specified net list. Maze routers can also be applied but they fail to check ahead to avoid unnecessary blocking of other terminals.

Four algorithms that use different techniques are now presented in some detail. A comparitive analysis is made at the end. WEAVER [62] is an elaborate rule-based expert system router, that often produces excellent quality routing at the cost of excessive computation time. BEAVER [64] is an excellent heuristic router with much promise. MIGHTY [66] is based on an incremental routing strategy that allows modifications and rip-up of nets that may impede the routing of other nets. SILK [65]is based on a technique known as simulated evolution.

4.1 AI-Based Switchbox Routing

WEAVER [62], is a knowledge-based expert system, that has been demonstrated to route several channel and switchbox problems successfully. Since there are several metrics that need to be optimized, WEAVER provides an *expert* on each. These experts observe the current problem state and suggest their solution as far as the criteria which are important to them are concerned. A scheduler or *focus of attention* expert then decides on the best action based on all the suggestions received. This, in AI terminology, is referred to as a *blackboard* architecture. Fig. 35 shows WEAVER's architecture and the different experts. Each expert consists of a set of rules written in OPS5 and as such continuously performs a recognize-and-act cycle. The figures in Fig. 35 represent the number of rules implemented for that expert.

The experts in WEAVER can play the role of a consultant or a planner. As a consultant, they criticize the suggestions of other experts. As a planner, they provide solutions to how a net should be routed. For instance, the wire-length expert, in the role of a planner, might suggest routing the longest length wire first. The vertical/horizontal constraint graph expert, in the role of a consultant, might criticize this decision because it foresees that based on the constraints routing that net now may block future nets from being routed.

In each cycle, the scheduler first invokes the pattern router as a planner. If the latter comes up with a suggestion, it is accepted and the constraint propagation expert is called to update the problem state. Other experts are also allowed to update their private scratch memory. The pattern router expert, in WEAVER parlance, is an example of an *autonomous* expert: one whose decision is accepted without any criticism.

If the pattern router fails to come up with a plan, then the wire-length expert is invoked. Its suggestion is however criticized by the Steiner tree expert, the constraint graph expert, the merging expert and the congestion expert. In this scenario, the wire-length router is a *decision-making* expert, while the others act as *decision-modifying* experts. A fourth category constitute the *tie-breaking* experts such as the congestion expert, whose function is only to prioritize the other expert's decisions. After the decision is made as to

Algorithm 15 BEAVER
Begin

```
1.Initialization: Initialize the control information
  Initialize the corner priority queue.
2.Call the corner router.
3.If there are any unrealized nets then
     Initialize linesweep priority queue
     Call line-sweep router
     4.If there are any unrealized nets then
          Relax control constraints
          Reinitialize linesweep priority queue
          Call line-sweep router
          5.If there are any unrealized nets then
               Initialize thread priority queue
               Call thread router.
6.Perform layer assignment.
```

End

which net and which portion of it is to be routed, the set of rules responsible for actually laying out the net are activated followed of course, by the constraint propagation expert.

Discussion: WEAVER combines both algorithmic techniques (such as constraint handling) with simple deductive and expert knowledge. This has often resulted in very good quality routing. WEAVER does not need to impose unnecessary constraints such as imposed by a H-V wiring model. It allows easy user interaction. It can however be extremely time-consuming. To reduce the search space, WEAVER uses the minimum rectilinear Steiner tree in routing and design based on a locally optimum yet invalid state in artificial intelligence domain.

Several improvements to WEAVER have been suggested in [62]. These include extending WEAVER to more than two layers, making it gridless, improving the domain knowledge, using intersection graph techniques to efficiently solve the layering and via minimization problems, *etc.* A comparison with other switch box routers will be made at a later period.

4.2 Computational-Geometry Based Switchbox Routing

BEAVER employs three methods to solve a routing problem – a *corner router*, a *line sweep router*, and a *thread router*. Each of these routers is given a priority queue of nets to be routed. BEAVER has a *track-control parameter* that is used by both the corner and line-sweep routers. It represents the percentage of the tracks, extending from the terminals of the net, that is reserved for that net. If the control parameter is less than 50%, the middle portion of the row or column is available to any net. On the other hand, if the control parameter is greater than 50%, there is an overlap in control in the middle portion between nets with terminals on the opposite sides of the routing region. Fig. 36(b) illustrates this concept for the sample switch box problem shown in Fig. 36(a). For the sake of illustration, it is assumed that nets 2 and 3 have 50% control, nets 1 and 4 have 80% control and net 5 has 40% control. Only the control for the horizontal tracks are shown. Similar control will also exist for the columns. The authors contend that the best results are obtained when the control is set to its maximal value, since it allows a crucial terminal to extend just sufficiently outward to realize the connection, without overly blocking future nets.

Corner router: This is the router that is routes most of the nets which form a *corner* connection. A corner connection is formed by two terminals belonging to the same net, that lie on adjacent sides of the switch box and with no other terminals belonging to the net, that lie between them on the adjacent sides. The corner router is also the preferred router since it is the fastest and also because its connections tend to be part of the minimum rectilinear Steiner tree for the nets.

During the initialization of the corner priority queue, each net is checked to see if it has a corner connection. Corner connections can be simply made by examining the control of the terminals that comprise

the corner. If the sections of the control overlap, then the corner can be realized. Nets with one or two corners need no further checks. However, straightforward connections of four corner nets can introduce cycles. There are two types of cycles as shown in Fig. 37. An *overlap* cycle is removed by routing only one of the overlapping corners. A *four-terminal* cycle can be removed by routing only three of the corners. When the corner router has to decide upon one of two such corners to route, the one with least impact on the routability of other nets is preferred.

Fig. 39(a) shows the switch box with corner connections for nets 1 and 4 realized. Each realized corner connection consists of a wire with a single bend. Although net 2 is also a single-corner net, it cannot be immediately routed as control of grid point (2,1) belongs to net 3.

Linesweep router: The linesweep router is invoked when no more corner connections can be made. However, if after the current net's linesweep realization, some other corner connections become realizable, then the linesweep router is temporarily suspended until the corner priority queue gets emptied. For each net five possibilities are considered: a wire with a single bend, a single straight-line wire, a dogleg connection with a unit-length cross-piece, three wires arranged as a horseshoe, and three wires in a stair arrangement. These are shown in Fig. 38. To reduce the number of vias, straight line wires are preferred to dogleg connections, doglegs to single-bend connections and single bend to two-bend connections.

In looking for its connections, the linesweep router uses the computational geometry technique of *plane-sweeping*. One approach, using scan-lines to find straight line connections between disjoint subnets of the net in question, was due to [131]. It works in $O(n \log n + k)$ time. BEAVER uses three scan lines that sweep the plane in tandem: one across the columns, one across the rows and a third to detect doglegs. Also, it employs bounding functions to reduce the computational complexity. If some nets still remain unrealized, the control of existing nets is reduced and the process is repeated a second time. All remaining nets are then routed by the thread router.

Fig. 39(b) shows the switch box after the line-sweep and corner routing. At first nets 6 and then 3 get realized by the linesweep router. The completion of net 3 allows net 2 to extend its control to column 2 and hence net 2 gets realized by a call to the corner router. Finally net 5 gets realized. At this point only net 7 remains to be routed.

Thread router: This router is invoked very sparingly. It is a maze-type router that seeks to find minimal-length connections to realize the remaining nets. Since, this router does not restrict its connections to any preferred form, it will find a connection if one exists. Whenever a net i consists of more than one routable subnets, a maze- expansion, on the lines of the Soukup router, is initiated. The expansion starts from the smallest subnet s that has not undergone thread routing to the largest realized subnet t. During the maze-expansion both layers are used in an endeavor to minimize the wire-length and number of vias. Fig. 39(c) shows the switch box with all the nets routed.

Layer Assignment: This phase primarily aims at minimizing the number of additional vias introduced. Since, at this stage, all grid points that any wire passes through are known, it is possible to optimally assign unlayered wires to a particular layer. BEAVER can also very easy extend this to achieve metal-maximization. A simple set of heuristics, based on coloring the grid points as red or black, is presented in [64].

4.3 Rip-up and Reroute-Based Routing

The Mighty router [66] is based on an incremental routing strategy. It employs maze-running but has an additional feature of modifying already-routed nets. Some nets are even ripped-up and rerouted. It is this feature of Mighty, that makes it suitable for channel and switch box routing. The cost functions used for maze-routing penalizes long paths and those requiring excessive vias. It consists of two entities: a *path-finder* and a *path-conformer*.

Discussion It is possible for the router to go into a loop in the modification phase. This can be avoided by using some sort of a time-out mechanism. The authors have shown that in the worst case, the algorithm needs no more than $O(k^3pnL)$ time, where p and k are the number of pins and nets respectively and L is the complexity of the maze-running algorithm.

Algorithm 16 MIGHTY
Begin

1. Extend all pins on the boundaries of the region inside by one unit;
   Initialize list $L \leftarrow \phi$.
2. *Path Finder*:
   For each net Do
      Find a minimum cost path for this net using a maze-router;
      Record this in the list L;
3. Sort L in increasing value of costs;
4. While L is not empty Do
   Get the next path p from L;
   4a. If none of the grid cells in p have been occupied then
      Implement p; goto Step 8;
   4b. Else invoke the path-finder to find a new feasible minimum cost path
      connecting two unconnected subnets of the net;
      Let δ be the increase in cost for the new path p';
   5. If $\delta < MAXINCREASE$ then
      Implement p' ; goto Step 8;
   6. *Weak Modification*: Push implemented nets around so as to
      obtain a "good" connection for the given net;
   7. If Step 6 fails then
      Strong Modification: Remove an existing connection and try
      to obtain a "good" connection;
   8. Remove p from L;
   end while;

End

Algorithm 17 SILK
Begin

1. *Initial Router*: Use a greedy algorithm to obtain a solution for the switchbox. Such a solution will normally contain several violations.
2. *Mutation*: This is the rip-up and reroute phase. Compute the total wire length and the number of failed-nets;
3. While there remain some failed-nets do
   4. Score (*Evaluate*) each net and normalize the scores
   5. Use Natural Selection to choose candidates for rerouting
   6. Route the nets chosen in Step 5 using a maze-running approach

End

4.4 Simulated Evolution for Switchbox Routing

SILK [65], is based on the principle of evolution, which has been successfully applied to the placement problem. The algorithm operates on one configuration, and the nets in that configuration constitute the population. A mutation is a random change in the routing. The evaluation step is used to score each net and determine the goodness of the current configuration. This goodness value is also used to probabilistically determine whether the net is to be ripped and rerouted.

Details The greedy router used to obtain the initial configuration is a simpler version of the one in [63]. The grid is processed one column at atime and the complexity of this step is $O(l \times r \times c)$, where l,r, and c are the number of layers, rows and columns respectively. For each net, the lower bound on the wire length is calculated according to the half-perimeter of the smallest rectangle that encloses all pins of the net. Also, the number of failed connections can be easily determined by counting the number of grid cells that have been multiply assigned to different nets.

The chief step of evolution is the evaluation and subsequent natural selection phase. The basic idea is to evaluate each net in the current generation (configuration) and determine the survivability rate of each net over the next generation by giving the nets with lower score a higher probability of survival. Those nets that fail to survive are replaced with new nets that have different characteristics (paths). The score for each net is the weighted sum of three different factors.

$$Score(i) = \alpha \cdot (\#of\ violations) + \beta \cdot (\#of\ vias - \#of\ pins) + \gamma \cdot (Wire\ length/lower\_bound)$$

The parameters α, β, and γ can be tuned to fit the requirements. Setting α to a high value directs the mutation toward a feasible solution. Similarly β and γ can be used to discourage excessive vias and wire length. In practice, the authors have found setting $\alpha = [1.0, 3.0]$, $\beta = 0.1$ and $\gamma = 1.0$ to yield satisfactory results.

These scores are then normalized to lie between 0 and 1 so that they can be compared with a random number also between 0 and 1. A net is marked to be rerouted if its normalized score is greater than the random number generated for it. This is the *natural selection* step.

The marked nets are routed using a Lee type mazerunner algorithm with appropriate cost functions. Once, a feasible solution has been obtained, it is possible to improve the solution further by repeating the rerouting step. However, only feasible solutions are now accepted. Also, in the scoring function, β and γ assume increased importance. The authors also report that the initial configuration is not critical to the performance of the algorithm.

4.5 A Comparison of the Switch Box Routers

Some other notable efforts to solve the switchbox problem have been reported by Luk (Greedy Switchbox Router - GSR)[63] and Hamachi *et al.* (Detour) [61]. Both these approaches are extensions of the greedy approach for channel routing [53]. They include additional heuristics to include nets with pins on the *right* side of the region and those having pins only on the *right* edge. Detour, in addition began at the point of

Table 4: Comparison of switch box routers

Example	Router	# of rows	# of cols	# of vias	Length	Time
Burstein's switchbox	GSR	16	23	58	577	1
	WEAVER	15	23	41	531	1390
	Mighty	15	22	39	541	4
	SILK	15	22	36	528	69
	BEAVER	15	23	35	547	1
Terminal Intensive switchbox	GSR	16	23	68	632	?
	WEAVER	16	23	49	615	1847
	Mighty	16	23	50	629	?
	SILK	16	23	49	616	?
	BEAVER	16	23	46	632	1
Augmented Dense switchbox	GSR	18	16	36	527	?
	WEAVER	18	16	31	517	?
	Mighty	17	16	32	530	?
	SILK	17	16	29	516	?
	BEAVER	18	16	27	529	1
Modified Dense switchbox	GSR	n/a	n/a	n/a	n/a	n/a
	WEAVER	17	16	29	510	?
	Mighty	17	16	29	510	924
	SILK	17	16	29	510	?
	BEAVER	17	16	26	510	1

maximum density and started routing both ways. It also included heuristics to account for obstacles in the routing region. Part of its popularity stemmed from the fact that it was included as part of the MAGIC layout system.

In this section, we will compare the performance of the switch box routers that have been presented earlier. Table 4 shows their performance on some of the popularly used benchmark problems. The timing statistics should be interpreted with caution since many of these routers were run on different machines and in different programming languages (eg. WEAVER in OPS5).

BEAVER seems to be the fastest of the lot. The time performance of Mighty and GSR is comparable to that of BEAVER. SILK is slower than these three, but is faster than WEAVER. Part of the reason for this is in the fact that BEAVER and Mighty use several strategies to route all nets: the fastest (eg. corner router) usually accounting for most of the nets. SILK, on the other hand, has to score and normalize each net during each iteration. This takes time. WEAVER does an exhaustive search of the solution space and as such is extremely slow.

SILK and WEAVER generally use less wire than the other three. Of course, there exists a trade-off between the time permitted and the routing quality. BEAVER is also generally better than GSR because of its three-tier strategy as opposed to GSR's straightforward greedy method.

BEAVER uses the least number of vias among all algorithms. This is to be expected since BEAVER tries to avoid vias at several places (eg. when searching for connections in the linesweep router; the layer-assignment phase). SILK is the second best. The scoring function and also the post-optimization phase help to minimize the vias. Mighty and WEAVER use a comparable number of vias whereas GSR does not use any via-minimization technique after generating a solution.

WEAVER and Mighty can handle irregularly shaped rectilinear routing regions. They are also very effective with respect to channel routing. The 4-dimensional grid representation for SILK is general enough for it to handle such cases too. BEAVER on the other hand becomes quite slow when used for channel routing. The reason is that the corner router, its fastest router, is no longer applicable. Both BEAVER and SILK can however handle obstacles within the routing region. This is done by modifying the grid status in SILK and by updating the track-control information in BEAVER. Mighty, in addition, can also handle floating pins (*i.e.* pins whose positions are not decided prior to the start of the routing operation). Of all

these routers, SILK seems to be the best approach for multi-layer switch box problems.

BEAVER and Mighty are insensitive to the orientation of the routing area (*eg.* which side is considered *up*). GSR has been known to be quite sensitive to the orientation. SILK does not discuss this problem. Of all the routers, WEAVER alone can guarantee to route all routable nets. However, it often takes an enormous amount of time.

In conclusion, BEAVER seems to be a good first router to use on a routing problem. It does not take too much time, uses few vias and a reasonable amount of wire length. WEAVER can be considered for the few pathological problems that BEAVER may fail to route completely. SILK also seems promising, especially for multi-layer solutions. However, some more understanding of the effect of the control parameters is required. Mighty is good for simple channel-like problems with floating pins and obstacles.

5 Single-Layer Routing

A necessary condition for single-layer routing is that wires should not cross. Thus, the detailed routing graph can be replaced by a continuous planar routing region (gridless planar routing). As was mentioned earlier, planar routing is also referred to as *river routing*. If the routing region is in the form of a channel, with each net having one terminal on each side of the channel, then the problem is also called *strait-typed* river-routing. The separation between the two boundaries is referred to as the *width* of the strait.

5.1 The General River Routing Algorithm

Only two-terminal nets are considered. All paths are assumed to be counterclockwise along the boundary. For a net $\{T1, T2\}$, there are two possible paths along the boundary. The starting terminal for the net is chosen to correspond to the shorter path. The following algorithm is due to Hsu[67].

A planarity check is first required to determine the feasibility of a single-layer solution. This can be efficiently determined using a stack. Initially the stack is empty. Assume L is a circular list of all terminals ordered in a counterclockwise direction according to their positions on the boundaries. Starting at any terminal on L, all terminals in L are sequentially checked. Each new terminal is first matched with the top element of the stack. If the two are the same then the top of the stack is popped, else the new terminal is pushed onto the stack. If after processing one cycle of L, the stack is empty, then there is no crossover between the nets.

The next stage is to generate the proper order for the nets to be routed. Since, initially all nets are routed as close to the boundary as possible, this will ensure that a net is routed only after all nets which have one or more terminals in between have been routed. This stage can also be performed using the same stack used for planarity-testing.

The third stage is to form paths for each net in the order determined. This routing is done by contouring along the outer boundary from the starting terminal in the counterclockwise direction, with the path staying as close as possible to the boundary formed by the edges of the routing region or the wiring paths already defined.

The final phase, called *corner minimization* is a heuristic step to reduce the number of bends and to shorten wiring paths. It is done in reverse net order by flipping the corners toward the inside of the routing region. A corner is flipped only if all design rule constraints are met. The eight types of corners that can occur are shown in Fig. 40. Thus, cases a, b, c and d can be transformed to cases e, f, g and h by flipping the corner to the inside.

Example: Fig. 41(a) is a general river routing problem that can easily be seen to be routable on one layer. The starting terminals for each net is also shown. A net-order of {1,3,4,2} is obtained if we begin at the starting terminal of net 1. Fig. 41(b) shows the solution after the path-placement phase. The final routing after corner mnimization is shown in Fig. 41(c).

5.2 Strait-Typed River Routing

This algorithm is fairly simple. A net whose top terminal (t_i) is to the left of its bottom terminal (b_i) is called a *falling* net. If t_i is to the right of b_i, it is called a *rising* net. Otherwise, the net is said to be *trivial*. The minimum width required for a solution can be exactly determined in polynomial time using Alg 19.

Algorithm 18 NET-ORDERING FOR GENERAL RIVER ROUTING
Begin

1. For each net n, assign a starting (s_n) and ending(e_n) terminal.
2. *Initialization*: $i = 1$;Mark all s_n as *not pushed*;
   Mark all e_n as *not matched*;
3. While $i \leq$ number of nets Do
   - 3a. Pick a terminal T from the circular list L ;
   - 3b. If T is a starting terminal and *not pushed* then
     Push it on the stack and mark it *pushed*
   - 3c. Else if T is a *not matched* ending terminal then
     If T and the top of the stack correspond to the same net then
     Pop the top element; Mark T as *matched*;
     Assign the number i to the net; $i++$;
     Set T to the next terminal on L; goto 3b;

end while

End

Algorithm 19 STRAIT-TYPED RIVER ROUTING
Begin

For each net proceeding from left to right Do

1. If it is a trivial net then
   simply make a vertical connection between the two terminals
2. If it is a falling net then
   go downward from the top terminal as much as possible
   then contour around the boundary or previously laid wire
   until the bottom terminal is reached
3. If it is a rising net then
   go upward from the bottom terminal as much as possible
   then contour around the boundary or previously laid wire
   until the top terminal is reached

end for

End

Discussion: The basic principle behind the algorithm is that no space to the left of routing segments of a net can be used by the nets to its right. This leaves maximum available space to the right of routing segments that are still to be placed. If B is the number of boundary segments and T is the number of terminals, then the time complexity of Alg 19 is $O(T(B+T)^2)$ and it needs $O((B+T)^2)$ space. Furthermore, both the river routers discussed here guarantee to find a solution if one exists.

6 Global Routing

The goal of global, or loose, routing is to provide a routing plan in which each net is assigned to a particular routing region(s). The aim is not only to make 100% connections, but also to minimize the total wire length, consider the priority of signals , and to balance the congestion across the routing regions, *etc.* The latter is important to facilitate the job of the detailed router in finding the precise route of every net within their assigned routing regions. Global routing is thus very important to obtain a good overall layout.

The nature of the routing regions depend on the layout style. In gate array and standard cells, it is formed principally by the horizontal channels that run between the rows of gates (cells). The interchannel connection is by means of vertical tracks on the sides or by feedthroughs. Feedthroughs and equivalent pins generally complicate routing. The major difference between the two layout styles is the fact that in gate-arrays the channel height (distance between adjacent rows of gates) is fixed and the primary goal is to make 100% interconnections. In standard cell, the height is variable and can be adjusted so as to minimize the overall chip area. This difference may explain why rip-up techniques are more widely used for gate arrays. For macro cells, global routing becomes meaningful only after a routing region definition and ordering (RRDO) phase is performed. The routing regions are known only after the placement of the modules is done. Regions can be regular channels, L-shaped channels, switchboxes or general maze structure.

6.1 Global Routing Models

Sequential Approach : This approach, though simple, generally suffers from the net-ordering problem. A global routing graph is first defined that depicts the relationships between the routing regions and the pins to be connected. There are two approaches. Preas [51] defined a *channel intersection graph* in which each vertex represents a channel intersection and each edge represents a channel or channel subsection. Hightower [47] defined his graph as comprising of channels as vertices and channel adjacencies as edges. By assigning appropriate costs to these edges, it is possible to formulate the global routing problem as a shortest path finding algorithm for a two-pin net. A common form of the edge weight is expressed by

$$w = aL + \frac{b}{c^{T+1}}$$

where a, b and c are parameters to be fine-tuned for a particular application, L is the length of the channel, and T is the number of available tracks. Since T changes after each net is routed, the edge weights will have to be updated. Such a cost function therefore favors less congested and shorter channels.

For multi-pin nets, finding the optimal path in the global routing graph corresponds to the Steiner tree problem. The Steiner tree permits the vertices of the connection graph to be at pins as well as at locations other than pins (called Steiner points), and places no restrictions on the degree of the vertices. Fig. 42 shows the minimum Steiner tree of weight 4 interconnecting points A, C and E. Note that F is the Steiner point.

Unfortunately, Steiner tree problems, including those on rectilinear grid graphs, as is commonly encountered in VLSI problems, are NP-complete and heuristics are often used. There exist simple heuristic algorithms [133], for $n \leq 5$, where n is the number of terminals of a net. Also, for special graphs such as $2 \times N$ graphs and series-parallel graphs, there are linear time algorithms. These graphs have been found to be very useful in hierarchical routing and in combining floorplanning with global routing [85, 84].

Simultaneous Approach : Many approaches to simultaneous routing of all wires reformulate the global routing problem as an integer linear program. This problem can be considerably simplified using a

Style	Routing Model	Most suited Technology	Reference
Iterative	Sequential	Gate Arrays	Ting[78]
Spanning Tree	Quasi-simultaneous	Std. Cell	Winter[83]
Hierarchical	Simultaneous	Gate Arrays	Burstein[71]
Greedy	Simultaneous	Std. Cell	Sechen[82]

Table 5: Global Routers

hierarchical approach, wherein the given large routing problem is decomposed into a number of tractable subproblems; the partial solutions to which can be combined to yield a solution to the original problem instance. Hierarchical decomposition can be *top-down*[71] or *bottom-up*[74]. A simulated annealing approach to global wiring was studied in [79]. Here again the global routing problem was formulated as a 0-1 programming problem. The cost function was equal to the sum of the squares of the number of wires assigned to each channel. In this way, the algorithm penalized congestion. Each move corresponds to switching one net from one path to another. Such an approach, in essence, restricts the solution space to a small subset of the full set allowed by the original problem definition. In this way, we hope to obtain an acceptable solution at a reasonable cost.

A third approach was presented by Nair [87]. It is essentially an extension to the Lee algorithm. A cost is associated with each boundary of a cell in addition to the cells themselves. Via costs are also considered. Each cell is then given a *score* in the forward pass. The score represents the sum of the costs of all the cell boundaries traversed by the cheapest path from the source to that cell. The backtrace phase is similar to the standard Lee algorithm. Very high costs can be associated with boundaries that correspond to blockages to discourage their selection. The various speedup techniques mentioned with respect to the Lee algorithm are also applicable here. This algorithm has very high time complexity. The best implementation, using priority queues requires $O(n^2 \log n)$ time for each net in a $n \times n$ weighted grid.

The approach used in [85] modules are hierarchically clustered in a bottom-up fashion, according to their size and connectivity. Each cluster can have at most five components. By constraining the nature of the floorplans, the Steiner tree problem for each of the clusters could be exactly solved in linear time. The global routing results at each level of the hierarchy is used to give the terminal positions at the next level.

6.2 Iterative Global Routing Algorithm

The algorithm starts by dividing the entire chip into *global* cells. The router assigns all the nets to selected global cells without considering the detailed routing within a global cell (see Fig. 43). Nets cannot exceed the boundary capacities. There are two phases for the algorithm:
(i) *Initial Routing*: Each net is allowed to take its *best* route, as if it were the only net to be routed.
(ii) *Iterative Rerouting*: Select the nets that exceed boundary capacities in the first phase and reroute them to remove all violations.

Discussion: The bipartite graph in Algorithm 21 consists of two sets of vertices: one set representing the set of nets crossing at least one overflow boundary and the other representing the set of overflow boundaries. Each boundary node has a weight equal to the associated overflow number. This algorithm can be considered as a demand-supply problem for which good heuristic solutions exist. However, the location of the global cell boundaries is crucial to the performance. The algorithm fails to consider feedthroughs and equivalent pins. The problem of congestion at feedthroughs can become quite difficult to address. The global router is followed by a vertical assignment step, wherein a vertical track is assigned to each net segment crossing a horizontal global cell boundary. An extension of Munkres' algorithm [132] is used for this purpose. Final detailed routing is then performed by the channel or maze routers.
Example: See Fig. 44. Assume that supply at all boundaries is 2. The initial routing causes *overflow* 1, 2. 1, and 2 at the boundaries indicated by T1, T2, T3, and T4. For the iterative routing phase, we let $k = 2$.

Algorithm 20 INITIAL ROUTING

Begin

1. Use a suitably modified Lee's algorithm to construct a global route for each net.
2. Construct a *boundary-demand* matrix to reflect the demand (*i.e.* the number of nets crossing the boundary) at each boundary
3. Using the boundary-demand matrix, generate an *overflow* matrix that reflects the overflow at each boundary. Overflow can be expressed simply as the difference between supply and demand at a boundary.

End

Algorithm 21 ITERATIVE REROUTING

Begin

1. If there exists no overflow boundaries then
   stop and report success
   Else
   Select the set B of the k largest overflow boundaries, for some k
2. *Net Selection*
   2a) Find all loops formed by the boundaries in B
   2b) For each such loop, find a double-crossing net, if one exists.
   If such a net exists, choose that net.
   2c) If there exists a loop, but no double-crossing net, then report the fact that no improvement can be achieved.
   2d) If a loop does exist, then use a simple greedy algorithm to select a set of nets using the bipartite graph representation.
3. Reroute selected nets. This can be done either by the Lee algorithm, one net at a time, or by rerouting all selected nets independently -- as in the initial routing step -- and then check if there is an overall improvement n the total overflow count. If so, accept the solution and goto step 1. If not, goto step 2.

End

Algorithm 22 HIERARCHICAL LOOSE ROUTING
Begin

```
1.Create hierarchy(hlevel, h) where hlevel is the number of levels;
2.Create the priority queue pq, of nets still to be routed;
3.For i = 1 to hlevel Do
      For all nets n in pq Do
          4.Create the subgraph G'_n of G_n by deleting all edges that
          have weight greater than h(i)
          5.Compute a Minimum Spanning Tree for G'_n
          6.If n has been completely routed then
             remove n from pq
      end for
  end for
```

End

The final (routable) solution obtained is shown in Fig. 46.

6.3 Spanning Tree Approach

This approach, developed by Winter and Mlynski [83], is also hierarchical in nature. Define a *channel* edge (i, j) as an edge connecting two adjacent terminals i and j lying in the same channel c, with no other terminal of the same net in c between them. Define a *transit* edges as one linking two terminals that are neighbors in different channels and have no other terminal lying within the bounding rectangle defined by them. A feedthrough connection is necessary to realize a transit edge. Define for a net n, its *net graph* $G_n = (V_n, E_n)$, where V_n is the set of terminals of the net and E_n is made up of channel and transit edges for n. Fig. 47 shows a net with eight terminals. Fig. 48(a) shows the corresponding net graph with transit edges indicated by broken lines. BD is a channel edge; BE is not since terminal D is in between. Similarly, BF is a transit edge; BG is not because of D.

Discussion: The number of hierarchical levels *hlevel* chosen should be inversely related to the number of local (as opposed to global) connections. The idea behind the hierarchical decomposition is to minimize the net-ordering problem. Weights assigned to channel edges reflect the density of the channel between the terminals, while for transit weight edge, both average channel density and the sum of all the feedthrough densities over all the feedthroughs that lie in between are taken into account. A good density estimation is therefore necessary to obtain accurate weights. The choice of nets used in Step 3 of Algorithm 22 is critical in reducing excess net congestion. This is done by associating a net with a *degree of freedom* f based on its routing alternatives.

$$f = \begin{cases} n(e_c) - n(T) + 1, & \text{for } n(F) = 0 \\ n(e_c) - n(T) + n(F) + 2 & \text{for } n(F) > 0 \end{cases}$$

Here, $n(e_c)$ is the number of channel edges in the net graph of the net, $n(T)$ is the number of terminals of the net, and $n(F)$ is the minimum number of necessary feedthroughs. The net with the least degree of freedom is routed first. An example is shown in Fig. 51. Here, net 1 would be considered for routing before net 2.

Example: Fig. 48(b) shows a minimum spanning tree for our example. The corresponding embedding is shown in Fig. 50.

6.4 Hierarchical Global Routing

This router, developed by Burstein and Pelavin [71], is based on the *divide and conquer* philosophy. The hierarchical approach is to divide the routing region into a 2×2 grid, lump all pins in each grid in the center of the 4 grids, decide on the appropriate pattern to route this 2×2 grid, subsequently building up the wire routes for every level of chip bisectioning hierarchy. The problem can be reduced to that of global wiring

Algorithm 23 HIERARCHICAL ROUTING BY INTEGER PROGRAMMING
Begin:

1. Partition the $2 \times N$ grid into two (usually in halves) parts.
   Form a 2×2 grid (Fig. 53) by coalescing all cells in each row
   to the left and right of the cut line into one supercell.
   Compute values for the horizontal and vertical capacities and via bounds
   for this 2×2 setup based on the location of the cut.

2. Now, construct the following system of equations:

$$\begin{array}{llll} \sum_j x(i,j) + X(i) & = & K(i); & \text{(The number of nets of type } i\text{)} \\ \sum_{(i,j)\in V1} x(i,j) + Y_1 & = & v1 & \sum_{(i,j)\in H1} x(i,j) + Y_2 = h1 \\ \sum_{(i,j)\in V2} x(i,j) + Y_3 & = & v2 & \sum_{(i,j)\in H2} x(i,j) + Y_4 = h2 \\ \sum_{(i,j)\in A} x(i,j) + Z_1 & = & M(v1,h1) & \sum_{(i,j)\in B} x(i,j) + Z_2 = M(v2,h1) \\ \sum_{(i,j)\in C} x(i,j) + Z_3 & = & M(v1,h2) & \sum_{(i,j)\in D} x(i,j) + Z_4 = M(v2,h2) \end{array}$$

   X_i, Y_i, Z_i are artificial variables that represent the total number of overflowed nets,
   the number of leftover wiring tracks, and the via restrictions respectively.
   H1 denotes nets crossing the upper vertical boundary. V1, H2, and V2 are
   analogously defined. Thus the goal is to
   $min \quad \sum_{i=1}^{11} X_i$, $max \quad \sum_{i=1}^{4} Y_i$, *and* $max \quad \sum_{i=1}^{4} Z_i$

3. Now consider the original $2 \times N$ grid. Introduce new terminal cells on the
   main grid adjacent to the crossed vertical boundary of the 2×2 grid for
   every net. Now solve the two smaller problems of size $2 \times k$ and $2 \times (N-k)$
4. Repeat till all wireable nets are routed.

End

within a $2 \times N$ grid. Two approaches to the $2 \times N$ wiring problem have been suggested. The first is built upon the exact solution of the problem for a 2×2 grid using an integer linear programming approach, and the second is based on a more traditional one net-at-a-time dynamic programming approach.

6.4.1 The integer programming approach

Let there be $h(i,j)$ horizontal tracks between cells $G(i,j)$ and $G(i,j+1)$. Let $v(i)$ be the number of vertical tracks between $G(1,i)$ and $G(2,i)$. Let $m(k,r)$ be the maximum vias that is permitted in a cell with k horizontal and r vertical tracks.

For a 2×2 grid, there can be 6 types of two-terminal nets, 4 types of three-terminal nets and 1 type of a four-terminal net as shown in Fig. 52 (a). A net can be routed in several ways for each type (see Fig. 52(b)). Let $x(i,j)$ denote the number of nets of type i that are to be routed in the jth possibility. An integer programming approach is employed to solve a system of equations to determine the values of $x(i,j)$.
Discussion: All nets are simultaneously considered for routing. This avoids the problem of net-ordering. Routability is also independent of the order of terminals within each net The size of the underlying integer programming problem is fixed and independent of the number of wires to be routed. This will therefore require a time of $C + O(m)$ where m is the number of nets, and C is the time for the IP problem. The algorithm is amenable to parallel implementation, where the different processors can handle a different 2×2 subproblem. Wiring quality and the number of overflows however may deteriorate when channel capacities are not uniform. The dynamic programming algorithm can then be used to route the overflow nets.

6.4.2 The dynamic programming approach

This is a slower and more traditional one net-at-a-time routing approach. The *divide* and *conquer* philosophy is in reducing a $(m \times n)$ grid into two parts of size $(\lceil m/2 \rceil \times n)$ and $(\lfloor m/2 \rfloor \times n)$ respectively. By treating

---

Algorithm 24 DYNAMIC PROGRAMMING $2 \times N$ ALGORITHM
Begin

1. *Initialization*: Denote by FIRST and LAST the abscissae of the leftmost and rightmost terminal cells.
   Tree $T1 \leftarrow G(1, FIRST)$; Tree $T2 \leftarrow G(2, FIRST)$;
   Tree $T3 \leftarrow$ minimal cost path between $G(1, FIRST)$ and $G(2, FIRST)$ and containing only of cells with abscissa $\leq$ FIRST;
   Forest $T4 \leftarrow$ cells $G(1, FIRST)$and $G(2, FIRST)$;

2. For i = FIRST+1 to LAST do
   (a)Enumerate the cheapest way of adding to one of $T1(i-1), T2(i-1), T3(i-1)$ and $T4(i-1)$(i)cell $G(1,i)$, (ii)cell $G(2,i)$, (iii) both cells $G(1,i)$ and $G(2,i)$ These are the new $T1(i)$, $T2(i)$ and $T3(i)$ respectively.
   (b)$T4(i)$ is defined as a minimal cost forest, consisting of two trees, one containing cell $G(1,i)$ and the other cell $G(2,i)$. It is either a trivial extension of $T4(i-1)$ or has one component identical to $T1(i)$ or $T2(i)$ and the other component is the isolated node $G(1,i)$ or $G(2,i)$

3. The final net is given by the cheapest among $Tj(LAST)$ $[j = 1, 2, 3]$ and $T4(LAST)$ with the two sub-trees connected by a minimum cost path consisting of cells with abscissae $\geq$ LAST.

End

---

each part as a supercell, it is easy to reduce this to a $2 \times N$ routing problem. The solution to the latter combined with the solution to the two parts gives us the final wiring. The $2 \times N$ routing algorithm is a modification of the Steiner tree finding algorithm developed in [129].

Discussion: The costs used for horizontal crossings (HC), vertical crossings (VC) and vias (VIAC) need to be carefully chosen to obtain good solutions. The authors suggest using the following cost-functions:

$$\begin{aligned} VC(HC)(B) &= (C1)^{C(B)-W(B)} + C2 \\ VIAC(B) &= (C3)^{C(G)-W(G)} + C4 \end{aligned}$$

$W(B)$ is the initial capacity of boundary B and $C(B)$ is the current number of nets crossing B. $W(G)$ is the maximal number of vias allowed in cell G and $C(G)$ is the current number of nets turning in G. The constants $C1$, $C2$, $C3$, $C4$ are experimentally determined and fine-tuned.

At every level of hierarchy, we obtain routes of all nets. So a net is never positioned exactly before all other nets are routed globally at least on previous levels of hierarchy. The complexity can be estimated as follows. The time spent in routing a single net at any level of hierarchy is proportional to the total length of horizontal segments of this net (assuming a constant time for detouring). This is bounded by $O(n)$, for a grid of size $m \times n$. Also, there are $log_2 m$ levels of hierarchy. So for N nets, the worst case time for the algorithm, assuming all nets expand throughout the length of the channel, is $O(N \cdot n \cdot \log_2 m)$.

6.5 Greedy Global Routers

The global routing algorithm to be discussed also has a strong resemblance to the popular simulated annealing strategy, even though it has no concept of temperature. The approach has been incorporated in the Thunderbird layout package [82]. The two salient features of the algorithm include:

- It has no dependence on the net order
- Electrically-equivalent pins are utilized to minimize the total wire length.

Algorithm 25 GREEDY GLOBAL ROUTER FOR STANDARD CELLS

Begin

```
For all nets do
  1.L ← list of pin clusters sorted by their average x-coordinate
  2.Construct the net graph G(V,E) , where V is the set of pin clusters
      and E is initially empty.
  3.While (L not empty) do
          P ← next unexamined pin cluster in L
          Find M,U and D (if they exist) such that they are the nearest cluster
          to the right of P and lie on the same row, row above and row below
          that of P respectively
          Let M_x,U_x and D_x be the x locations of M,U and D respectively
          ( their value is set to ∞ if such a cluster actually does not exist)
          If (D_x < M_x)
            add an edge between P and D in G
          If (U_x < M_x)
            add an edge between P and U in G
          If (M_x ≤ U_x and M_x ≤ D_x)
            add an edge between P and M in G
  end while
  3.T ← minimum spanning tree of G
  4.For each edge (u,v) in T do
        Choose one pin from cluster u and another from cluster v to form an
        actual net segment.  If the net segment is switchable, then a second net
        segment connecting the clusters u and v is formed in the other channel
end for
  5.Place all net segments selected in Step 4 in their respective channels.
        A switchable net segment is assigned to the channel which has the least
        congestion along the span of the net segment.
  6.While (stopping criterion is not met) do
          Select a switchable net segment and place it in the opposite channel
          ΔC =TCD(new state)- TCD(previous state);
          Accept this move only if ΔC < 0 ; else reject it
  end while
End
```

To recapitulate, standard cells are often designed using a dual-entry style (*i.e.* its signals appear on both the top and bottom side) with several equivalent signal pins called *clusters*. (see Fig. 55). The portion of a net connecting two pin clusters is referred to as a *net segment*. If the net segment can be implemented in either the channel above the row or below it, then it is further classified as a *switchable net segment*. Let the total channel density for a set of channels be denoted by TCD.

Discussion: Steps 1 to 4 attempt to find the shortest routes for a given net by considering all the shortest length segments connecting two neighboring pin clusters. The net graph for the example in Fig. 55 is shown in Fig. 56(a). Note the existence of two switchable net segments in the second and fourth rows. A minimum spanning tree for the same is shown in Fig. 56(b).

If N_{sw} represent the number of switchable net segments, then the algorithm stops when either the number of new states equal to $N_{sw}A_{sw}$ have been attempted or if a consecutive number of proposed new states equal to $\mu N_{sw}A_{sw}$ have been rejected. A_{sw} can be considered to be a bound on the number of attempted new states per switchable net segment. For practical circuits a suitable choice for μ is $\simeq 3$ and for A_{sw} is 30.

6.6 Global Routing for Building Block Layout

The advantage with the gate array or standard cell layout is that the channels are predefined and symmetric. With building block or macro-cell layout, this no longer holds true. As was mentioned in Section s:gintro, global routing for macro cells becomes meaningful only after a routing region definition and ordering (RRDO) phase is performed. Henceforth, we shall refer to the routing regions as channels and to the blocks as tiles.
Channel Definition: The following terminology is useful: A tile can be a ***block*** (representing cells) or a *space* (representing empty routing space). A ***horizontal tile plane*** is a maximal horizontal strip obtained by extending each horizontal edge of a block until it encounters a vertical edge of another block or the edge of the layout. Same with ***vertical tile planes***. The ***interval*** of a horizontal tile is the range of X-coordinates covered by the tile (Y-coordinates for vertical tiles). A space tile is called ***dominant*** if its interval includes the intervals of all its adjacent space tiles. Nondominant tiles are called ***subordinate*** tiles.
Floorplan Graph: The adjacencies between blocks and wiring regions is captured by constructing a floorplan graph. This graph is used as the primary data structure for routing region definition and ordering. The floorplan graph F has an edge called a ***wall*** for each dominant horizontal or vertical tile. The intersection of two walls is called a ***wall junction***. A wall junction that is incident to only one wall segment is referred to as *external*, the rest are called ***internal***. A ***wall segment*** is the portion of the wall between two adjacent junctions. A *room* is a region bounded by walls but containing no walls. If a room does not correspond to a block, it is called an ***empty room***. An observation to make is that walls can meet only either in a T shape connection or a (*cross*) + connection. Fig. 57(a) and (b) show the tile planes and Fig. 58 the floorplan graph for an example layout. In the absence of empty rooms, there is a one-to-one correspondence between wall segments and element channels, and between intersecting areas of adjacent channels and wall junctions.
Channel Ordering: Consider two channels intersecting in a T shape: the channel on the vertical part of the T intersection must be routed before the one on the horizontal part. This is because, if we route the region corresponding to the horizontal part first, we would have a set of pins whose positions are not specified, on one of the sides where fixed pins should be located. If the vertical part is routed first, then the channel router specifies, when it terminates, the exact locations of the otherwise floating pins.

To route a + junction, we can either invoke a switchbox router or a general maze router. Alternatively, we can represent it as a pair of T-junctions. There are two ways of doing this as is shown in Fig. 59. However, the two are not equivalent. For example, because of the precedence relationship, the representation shown in Fig. 59(b) has a feasible order ($e \rightarrow d \rightarrow c \rightarrow b \rightarrow a$) while the one in Fig. 59(c) has a cyclic relationship ($a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$) and thus causes a conflict. Hence, an optimization problem can be defined where the variables are the choices of representations for the cross connections and the objective function is the minimization of the number of cyclic relations among the channels. Other criteria influencing the conversion are discussed in [88].

Dai [86] proposed a scheme to resolve such cyclic conflicts by bisecting the floorplan graph along an L-shaped wall with two external connections. This corresponds to an L-shaped channel and since this is more difficult to route than a straight channel, the algorithm tries to minimize the number of such L-shaped channels introduced. A corner $c1$ is said to be ***horizontally dependent*** on another corner $c2$, if the corner junction of $c1$ is on the horizontal part of the wall of $c2$. Similarly, for vertical dependency. If a corner is neither horizontally nor vertically dependent on any other corner, it is called a ***totally independent*** corner. The routing order for the problem in Fig. 58 is shown in Fig. 60.
Discussion: The ordering and independence of the routing region described above are properties of the ***slicing structure*** concept. A placement, wherein the slices are straight lines has the nice property that the relation defined on the walls is always acyclic. One such algorithm is the min-cut algorithm due to Breuer. Each cut corresponds to a channel and those corresponding to the cuts producing the leaves of the tree constructed by the algorithm, are the first to be routed since they have pins on fixed positions on the sides facing the blocks separated by the cut. The placement tree thus automatically provides the channel ordering in a bottom-up fashion. Other popular placement algorithms such as simulated annealing need not produce a slicing structure, and hence require more rigorous treatment as described above. Another approach, suggested by Otten [5] is to convert a non-slicing placement into a slicing one by applying a suitable ***shrinking*** procedure. Modules are uniformly shrunk maintaining their aspect ratio until one or more slices bisecting the layout appears. The L-shaped channels can be considered as arising due to one-bend slices. In general, we can have k-bend slices. Routing the resulting routing regions often require use of maze

Algorithm 26 Routing Region Definition and Ordering
Input: A Floor Plan Graph F
Begin

```
1.Initialize stack S as the channel definition stack;
2.While ∃ a wall w, bounded by two external junctions Do
        Bisect the graph along w, (wall slicing);
        Make all junctions along w, external;
        Push straight channel definition corresponding to w, on S;
3.If there are no more internal junctions then
        stop; pop S for a feasible routing order
  Else find an independent corner c if one exists,
   Else set c to a vertically or horizontally independent corner;
4.Cut the corner c chosen above. All junctions on it are external;
  Push corresponding L-shaped channel definition on S;
  goto Step 2.
```

End

or switchbox routers. The example, we have considered, corresponds to a slicing structure. Hence, there are no cyclic conflicts. However, the existence of an empty room causes a minor problem since it violates the one-to-one correspondence between walls and routing regions. This is accounted for by ignoring one of the wall segments bounding the empty room when we define channels corresponding to the walls of the room.

Subsequent to the RRDO phase, a global router, such as those presented earlier can be invoked. Note how the RRDO phase has a close link to the placement and floorplanning phase. The BEAR system [81] tries to break the loop between floorplanning and global routing by performing both hand-in-hand so to speak. After a channel order has been determined, one channel is routed and the results used to modify the floorplan directly. Then we adapt the global routing, and proceed to route the next channel.

7 Minimum Via Topological Routing

Topological routers [68, 92, 90], perform routing in a continuous plane without having a notion of an underlying detailed routing graph. The main focus of attention is in the topological arrangement of wire crossings that leads to a minimum number of vias. All nets are assumed to be of zero-width and all vias are assumed to take zero area. Hence, the solution proposed may not be physically feasible because of design-rule constraints. A *geometrical mapping* step is therefore required to map the topological solution onto the layout plane using as small an area as possible. There are two flavors of via-minimization algorithms;

Unconstrained via minimization problem (UVM) : Given a routing region and a set of terminals of signal nets, find the interconnection topology and layer assignments for all signal nets that will minimize the number of vias used. This tackles both the two-dimensional routing problem and the via-minimization problems at the same time from a global point of view.

Constrained via minimization problem (CVM) : Given a collection of paths representing the interconnection wire segments without completely specifying the layers to which they belong, find a layer assignment for all the wires that minimizes the number of vias required. The major limitation of the CVM approach is that it is limited by the quality of the original routing solution.

Ciesielski [96] proposed an integer programming formulation to this problem. Other approaches were proposed in [94, 95]. Recently, Du and Chang [93] proposed another heuristic algorithm based on graph bi-partitioning. Naclerio [91] has shown that the CVM problem is NP-complete.

Algorithm 27 TOPOLOGICAL UNCONSTRAINED VIA MINIMIZATION
Begin

1. Generate the interconnection graph for the area being routed.
2. Check to see if this interconnection graph is planar. If so, set $E^*=\Phi$ and $G_p = G$ and go to step 4.
3. Find a maximal planar subgraph G_p of G and defined the set of edges removed as E^*.
4. Assign the sets on the inside of the Hamiltonian of G_p to one layer and those on the outside of the Hamiltonian to the other layer.
5. If $E^*=\Phi$, end the algorithm; the network has been routed.
6. Find the region connectivity graph G_I.
7. Select an edge $e_i \in E^*$ corresponding to a net and determine the vertices of G_I which correspond to the regions containing the endpoints of e_i.
8. Find a path in G_I connecting the two regions. Record which lines in Layer 1 cross the path and the order in which they do so.
9. Modify the graph G_I to reflect the regions split by the path just found.
10. If all the edges in E^* have not been placed go to Step 7; else end the algorithm.

End

7.1 Topological UVM Routing

In this section, we present Marek-Sadowska's [92]approach to the two-layer UVM problem. Actually, the paper addresses only a subset of UVM referred to as UVM*, where the routing region is free of obstacles and all nets have only two-terminals.

Discussion: The *Hamiltonian circuit* of a graph consists of the terminals placed along a circle maintaining the same relative positions to each other as in the original routing region. The *interconnection graph* G for a circuit, consists of its Hamiltonian and an edge connecting each pair of terminals that belong to the same net. Nets drawn on the inside of the Hamiltonian can be considered as being routed in one plane; nets drawn on the outside of the circuit can be considered to lie in the other routing plane. The problem of finding a minimum cardinality set of edges to planarize a given graph is NP-Complete. Therefore, some heuristic is usually employed to find a sub-optimal set of edges for removal in a more manageable amount of time. The *region connectivity graph* G_I, mentioned in Step 6 of Alg 27 is defined as a set of nodes corresponding to regions within the Hamiltonian circuit of G_p with edges defined between nodes that correspond to pairs of neighboring regions.

Example: Fig. 61shows an UVM* routing problem and its interconnection graph. For this example, removal of nets N_6 and N_7 is sufficient to planarize the interconnection graph. This is shown in Fig. 62. The region connectivity graph is shown in Fig. 63. The final topological routing of nets is shown in Fig. 65.

A more recent paper by Xiong and Kuh [90] presents a divide-and-conquer solution to the UVM problem, that can also handle the case of multi-terminal nets.

7.2 Topological CVM Routing

Xiong and Kuh [90] have proposed a fairly efficient CVM algorithm. Given a planar two-layer routing solution, define:

A *cross point* is any point of intersection of two different nets.

A *via candidate* is any location on a net (usually between two adjacent cross points) where a via may be introduced.

A *net segment* is the part of a net spanning from one via candidate to either another via candidate or a terminal.

Algorithm 28 TOPOLOGICAL CONSTRAINED VIA MINIMIZATION
Begin

1. Construct the planar representation P of the geometric routing.
2. Identify all the cross points, via candidates and net segments in P.
3. Let x_{nl} be a {0,1} variable designating the layer (0 or 1) that is to be assigned to segment l of net n.
4. Construct the *constraint graph* and check if all clusters are feasible
5. Solve the following {0,1} integer programming problem:
   Minimize the function f where

$$f = \sum_{\text{all via candidates}} W(k) \times |x_{ni} - x_{nj}| \ , \quad i \neq j$$

   subject to the constraints

$$x_{ni} + x_{mj} = 1 \ , \quad n \neq m \text{ for all cross points.}$$

End

The *degree* of a via candidate is the number of net segments intersecting at that via candidate.

Discussion: At each cross point, exactly two segments belonging to different nets intersect. It is obvious that these two segments have to be assigned different layers. Hence, the constraint equation of the form $x_{ni} + x_{mj} = 1 \ , \quad n \neq m$. In the constraint graph, a net segment corresponds to a node and an edge exists between two nodes if and only if the two corresponding net segments meet at a cross point. The constraint graph will in general consist of several connected components called *clusters*. A cluster is feasible if and only if the corresponding connected component is two-colorable (or contains no odd cycles).

The objective function can be understood as follows. A via candidate will be really a via, if and only if, there is at least one net segment that is assigned to a different layer than the others at that via candidate. Thus, for every pair of net segments at a via candidate, the existence of a non-zero term $|x_{ni} - x_{nj}| \ , \ i \neq j$ signifies the presence of a via. In case of via candidates with degree $k > 2$, a weighting function $W(k) = 2/k(k-1)$ is introduced to properly weight the contribution of each of the $k(k-1)/2$ pairs of net segments.

The objective function is quadratic in nature because of the presence of the absolute value sign. This makes the solution even harder. One way is to transform the problem to an appropriate integer linear program. An $O(n \log n + k)$ algorithm based on the latter approach, where n is the number of layout objects and k is the total number of cross points, has been reported.

Knock-knee connections and wire overlap can be easily handled by treating these as additional cross points. It is also possible to add additional constraints to force critical nets on a predetermined layer or limit the number of vias on a particular net.

Example: As an example, the two-layer H-V routing solution obtained in Fig. 24 is used. This solution apparently requires 18 vias. The planar representation is shown in Fig. 66(a). In Fig. 66(b), the 26 cross points are indicated by crosses and the 12 via candidates are indicated by shaded circles. The maximum degree of a via candidate is 4 (formed by the intersection of the net segments N_{5a}, N_{5b}, N_{5c}, and N_{5d}). Fig. 67 shows the constraint graph for this example. Note that all the 7 maximal clusters are feasible. The final result, which uses only 9 vias, is shown in Fig. 68.

8 Specialized Routers

In this section, we briefly overview some of the specialized schemes that have been practiced to route power and clock lines.

8.1 Power and Ground Routing

Routing of power and ground nets is more involved than the routing of regular signal nets.

Voltage Drops : Due to the finite conductance of the interconnect and contacts, there exists a voltage drop from the supply point to any feeding point (sink node) or between sink nodes. This has an adverse effect on the noise margins of the cells.

Metal Migration : The p/g nets carry much more current than other nets. Hence, the problem of metal migration, which occurs when the current density exceeds the maximum limits, is even more pronounced.

Wire Widths : The attribute of a topology that determines whether the above two factors are satisfied or not is the unknown widths of the various segments of the topology. However, there are bounds on the width of a net. The lower bound could be determined by technology, the upper bound could be due to floorplanning. These have also to be taken into account.

Since metal has the least resistivity as compared to polysilicon or diffusion, the p/g nets are usually laid out entirely on the metal layer(s) of the chip. Since, contacts (vias) also significantly add to the parasitics, it is also advisable to have a planar single-layer implementation of the p/g network.

Routing of p/g network consists of two main tasks: a) construction of the interconnection topology, and b) determination of the widths of the various segments of the topologies. There can be several types of topologies such as: a rooted tree with its base at the pad and the leaves at the modules' terminals, a forest of trees with more than one supply pad, or a general graph with several pads and containing one or more cycles.

8.1.1 Planar routing of power/ground networks

Syed and El Gamal [102]proved the necessary and sufficient conditions for a planar routing of p/g nets using single pads: two nets can be routed on a single layer without crossover if and only if there exists a cut for each block in the chip that separates the terminals of one net from the terminals of the other net as shown in Fig. 69.

The p/g nets can then be grown as interdigitated trees. Applying simple *traffic rules* to the free channels between modules prevents the two trees from crossing. If the power pad (pins) is assumed to be at the upper left corner of the chip (block) and the ground pad (pins) is assumed to be at the diagonally opposite corner, then the relevant traffic rules as given in [102] are:

- In every horizontal channel, the ground wire must be above the power wire. Similarly, in every vertical channel, the ground wire must be to the left of the power wire.
- The rules at T intersections are shown in Fig.70(a)-(d).
- The rule at a + intersections is shown in Fig.70(e).

These rules are illustrated in Fig. 71(a) and (b). Fig. 71(a) shows the available segments for the p/g nets according to these rules. Depending on the actual pin distribution, some of these segments may be unnecessary (Fig. 71(b)).

Other Approaches: Another approach is tree growing [101, 100]. In [101], one tree extends from the left edge of the chip and the other from the right. Another approach grows, at each step, the branch of a tree that would least hinder the growing of the other tree. In [100], the approach was to find a Hamiltonian cycle that passes through every module once, dividing the VDD terminals from the ground terminals. This splits the chip into a VDD region and a GND region. Signal-routing techniques are employed to find a short Steiner tree, lying within the cycle, that connects the VDD pins to the VDD pad. Subsequently, the Hamiltonian cycle is deleted and the GND pins and the GND pad are connected to form a GND tree, thus avoiding a crossing of the trees. However, both the Hamiltonian cycle and Steiner-tree operations are computationally very expensive.

In [107], a scan-line technique is employed to sweep across the layout, and simultaneously determine the two trees. This method has the advantage of avoiding pad and pin placement restrictions. Moreover, it is very efficient both in space and time, as it reduces a two-dimensional problem into a one-dimensional problem on a scan line. The drawback of the approach is that it does not attempt to optimize routing parameters such as total wire length or area.

For more complex chips, we usually have multiple power pads. Some of the pads may even supply the same voltage level. Hence, we need to extend these algorithms to handle more than two networks. [102] mentions the case of a three and four net problem. A more exhaustive treatment can be found in [106]. For multi-pad and multi-layer p/g distribution systems, graph structures are also preferred to enhance system reliability and signal stability.

8.1.2 Power/ground net sizing

The area requirements for p/g nets depend on the voltage drop, current density and other constraints. As these limits become more restrictive, the area increases and can become quite large. Thus, minimizing p/g routing area is very important. In the case of normal signal nets, the current they carry is very small. Hence, they can be routed using minimum-width wires as required by the technology. Thus minimizing the total wire length also ensures minimizing the area needed to route them. The same is not true for p/g nets. In fact routing all segments with a constant width does not guarantee the minimum area.

It is reasonable to assume that the average and peak currents drawn by each of the logic modules is known. Based on this information, and the topology of the p/g network (tree or graph), it is possible to determine the widths of the branches in the tree or graph by formulating the problem as a nonlinear optimization problem. The following notations are used:

I_i	current through branch i
r_i	resistance of branch i
l_i	length of branch i
w_i	width of branch i
V_i	Voltage of node i (a node is either a junction of three or more branches, or the terminal at which the block is fed
Δv_i	voltage drop across branch i
ρ	the sheet resistance of the layer on which branch i lies
σ	electromigration constant (*i.e.* maximum branch i current $\leq \sigma \times w_i$)
V_{min}, W_{min}	constants determining minimum voltage level at a node in the power net and minimum wire width respectively
n_b	Number of branches
A	Total area for the power net

Objective Function: $A = \sum_{i=1}^{n_b} l_i w_i \quad = \sum_{i=1}^{n_b} \frac{\rho I_i l_i^2}{\Delta v_i}$.

Constraints:

Voltage drop: $V_i \geq V_{min}$,

Metal migration: $\frac{|I_{imax}|}{w_i} = \frac{|I_{imax}|\, r_i}{\rho l_i} \leq \sigma$ or, $\max |\Delta v_i| \leq \rho l_i \sigma$

Minimum width: $w_i = \frac{\rho I_i l_i}{\Delta v_i} \geq W_{min}$

Note, that in this formulation, the variables being solved for are the non-leaf currents I_i and the node voltages V_i. The objective function is non-linear. All the constraints are linear in nature. In the presence of general graph topologies additional constraints such as Kirchoff's current and voltage laws may be required. Computing an exact solution with very high accuracy is not usually required. The chief requirement is that of a numerically stable and efficient algorithm that converges to near optimal solutions. Most of the general optimization techniques applicable for this problem can be classified into three broad categories: method of feasible direction, penalty function methods, and augmented Lagrangian techniques. The method of feasible directions generates a sequence of feasible points, each better than its predecessor, by moving in the direction of feasible arcs associated with the constrained region. The details of these methods can be found in [99, 103], and are beyond the scope of this survey. The interior-penalty function approach has been reported in [105, 104].

Discussion Most of the discussion above has been directed towards the custom chip design style. For gate arrays, the power and ground nets are prewired in the design of the masterslice. The designer has only to

avoid these wires while routing the signal nets. In the case of standard cells, power and ground wirings are defined as part of the cells and can be connected to the pad rings by simple patterns. The normal scheme for routing such regular arrays is shown in Fig. 72(a). However, if the voltage drop constraints dominate the others, might be possible to optimize the area by employing a star distribution [105]. This is shown in Fig. 72(b).

8.2 Clock Distribution Schemes

Within most VLSI circuits, data transfer between functional elements is synchronized by a single control signal, the processing clock. The clock signal often constrains the timing and performance of the circuit. It is obvious that in the absence of a proper clock distribution strategy, different clock lines can have greatly varying lengths, leading to large skews and delays. Thus, special techniques are often required for routing clock lines, especially in high-performance circuits.

Ramanathan in [110] has identified the four major reasons for the timing uncertainties in a VLSI circuit:

1. Differences in lengths of lines that deliver the clock signal to the different functional modules
2. Differences in delays caused by active elements, such as clock buffers, inserted in the clock lines
3. Parametric differences in resistivity and dielectric that determine the line delays
4. Differences in threshold voltages of different elements

Wann and Franklin [109] proposed a distribution scheme that ensures equal lengths for all clock paths when the elements are identical and placed in the form of a regular array. But for a building-block layout, such a scheme is not suitable. It is interesting to note that delay and skew often have conflicting requirements. A minimum skew layout may have long lines that have larger delays and vice versa.

The approach in [110] uses the hierarchy created by the clock buffers to parallelize the distribution of the clock signals. Given a floorplan graph, we assign a weight to each edge equal to the length of the associated channel. Then we add nodes corresponding to the input clock terminals of the sub-modules and the output clock terminals of the clock buffers and edges connecting these nodes to the original floorplan graph. The weights of these new edges include a factor for the delay associated with the previous active stage, plus the maximum delay to the input of the buffer driving it. The next step is to identify an *entry point*, say e from which the primary clock is to be distributed. The entry point is selected in such a way that the delay from the farthest sub-module from that entry point is minimized. The optimal layout of the clock lines from e is subsequently determined by an exhaustive search of all paths with intelligent pruning. The pruning of the search space occurs while forming the set of possible vertices that can be appended to the current partial path. Vertices are disregarded if the shortest path to the vertex conclusively shows that there is no better completion of the current partial path. These techniques are also amenable to parallelization for further speedup. However, it is expected that in any large VLSI circuit, the time required for clock synchronization would be insignificant compared to the total time required to route all other signal nets.

Another approach to clock distribution is called *Clock Tree Synthesis*, a new option for Tangate, Cadence's place-and-route software for large gate arrays. As the name suggests, the algorithm aims to synthesize optimally shaped clock trees for minimizing or matching delays. There are three main steps involved: determination of the tree shape, clustering and balanced-wire global routing. Fig. 73 shows a two-level clock-tree of equal strength drivers with fanouts of 4,5 and 3 respectively.

The tree shape is based on an interactive delay estimator that returns delay values through each level of buffering, accounting for the capacitance and resistance of the interconnect wiring. The tree is then constructed one level at a time. All input pins at that level are clustered and drivers are assigned to each cluster. The original net is partitioned among the clusters. Within each net in the clock-tree a new balanced-wire global routing algorithm is used. The objective here is to equalize the length of the wire from the source pin of the net to each sink (see Fig. 74). The solutions are adaptive changing in response to congestion across the design.

Description	Cost
Heading towards target using proper direction	1
Heading away from target using proper direction	3
Heading towards target using wrong direction	5
Heading away from target using wrong direction	8
Making a corner	5
Adding a via	9
Using the cells next to any pads except target and origin	7

Table 6: Costs for maze algorithm

9 PC board routers

PC-board routers need to consider trace lengths, routing angles, heat dissipation, and cross-talk between signal paths. In addition, the plethora of different devices- surface-mounted and pin-grid array packages- with different numbers and spacing of pins make the place-and-route task very onerous. PC-boards also typically employ fineline dimensions and multiple layers. Channels tend to be fixed in width. Often to cope with complex boards, a a simple but fast algorithm is used to make the initial connections and is followed by more sophisticated but slower algorithms.

9.1 Heuristic Routers

A *heuristic* or *pattern recognition* router is written with a specific design problem in mind. The router needs only to search the board for predetermined configurations and then uses the appropriate trace pattern without having to execute a complicated mathematical algorithm for each trace.

Heuristic routers are fast. A 45° pattern router is commonly employed to route memory arrays, where each pin is connected to the equivalent pin on the next component in line and so on. However, the completion rate is low. Even if a component is displaced slightly, the pattern may not meet the required criteria and may fail.

9.2 Grid-Based Routers

These include the line probe and maze routers that were discussed in Section 2. Maze routers are generally much preferred to other type of routers for complex boards because of its ability to incorporate an array of cost functions that meet required specifications. Thus, even though line probe routers are faster, they lack the information needed to yield good quality routing. Double-sided surface-mounted devices can also be handled by the general three-dimensional maze-running model. Some typical penalties that are used for the Lee algorithm for PC-board routing, as given in [116], is reproduced in Table 6. It is not uncommon to consider 30 or more penalties.

9.3 Gridless Routers

A consequence of the gridded approach is that all the structures on the board, such as pads, copper area, traces, tooling holes, *etc.*, have to be placed within a cell or cells. Today's boards contain a mixture of IC packages, connectors, pin-grid arrays, *etc.*, with radically different pad spacings. Consequently, an uniform grid cannot yield a very dense layout. Simply subdividing a grid into fractional grids does not solve the problem.

Thus gridless routers have an important role to play. A true gridless router works by determining trace paths based solely on design rules and location of obstacles on the board. However, such an implementation is not very practical because of the high cost involved. At each stage, the gridless router has to perform complicated calculations regarding the optimum positioning. In some sense, the placement quality also determines the degree of success that the gridless router achieves. A better way is to conceptually divide the

grid into a number of varying-sized rectangles which are customized to the components and other restrictions on the board. This is also referred to as a *plastic grid* and has been incorporated into the Flexible Field Router offered by Cadnetix.

9.4 Single-Row Routing

This is an important problem that often arises in the layout of multilayer PC-boards using DIP components and backplanes. The problem can be stated as: Given the position of pins, which are all on one line (called the *reference* line), and the wiring list L, connect all the nets and minimize the number of tracks on either side of the reference line (referred to as the *upper* and *lower* streets).

Single-row routing allows efficient *a priori* estimation of the inherent routability of a given instance.It also produces wire layouts that are more amenable to automatic fabrication, since all conductors on a given layer are predominantly either horizontal or vertical. Fig. 75 shows a realization for the problem instance: $n = 9, L = \{(1,4,7),(2,8),(3,9)(5,6)\}$. As can be seen from the solution traces are allowed to cross the reference line in between terminals. Details can be found in [114]. Raghavan proposed another solution to single-row routing [115] with backward moves permitted. This meant that nets were permitted to extend beyond their endpoints.

10 Autorouters for Analog VLSI

Layout for analog VLSI is much more complicated than that for digital logic. The main reason is that the layout affects the functionality of the circuit. Some of the important constraints that are peculiar to analog design are:

- *Device Matching*: Some circuits, say a differential amplifier, needs the two transistors and resistors to be placed as near to one another as possible to minimize device mismatch.
- *Routing parasitics*: Care must be paid to routing parasitics on analog signals as they are critical to such features as capacitor ratios and sensitive summing nodes of operational amplifiers.
- *Signal net isolation*: Digital signals and large swing analog signals need to be isolated from sensitive analog signals. Floating or virtual ground notes are particularly sensitive.
- *Crosstalk*: Crosstalk can arise when two signals run in parallel over a long distance. This can also lead to circuit malfunction. Such wires need to be isolated from each other as well.
- *Timing/power distribution*: Power wires to high-power circuits must be thick to allow large currents to flow. Length of the wires interconnecting simultaneously active components have also to be controlled.

10.1 Analog Routing for Standard-Cell Layout

As mentioned above, sensitive signal nets need to be properly shielded for correct analog operation. A leaf-cell floorplan template is shown in Fig. 76. Like, their digital counterparts, the cells have fixed heights and variable widths. All digital signals and clocks enter the cell from the *insensitive* side. The analog signals and power lines enter the cell from the *sensitive* side at the bottom. Most intercell connections are by abutment.

The placement algorithms organize these leaf cells in rows with the sensitive side of each cell on the same side of the row. The floorplan (Fig. 77) is obtained by flip repeating these rows to obtain *sensitive* and *insensitive* channels. Clustering is used as much as possible for the cells of a subcircuit so that they occur within the same row. Such an arrangement helps to avoid unwanted couplings and minimize parasitics.

In the ILAC (Interactive Layout of Analog CMOS circuits) system [119], a global router is first invoked to determine for each net the path to follow through the routing channels. It is best-first maze-search based and handles net sensitivities, planarity for the power nets and channel congestion. Subsequent to this each channel is independently routed by a detailed channel router. The router routes nets one at a time in a given priority order to accomplish the required analog isolation. A similar approach is also taken in the LTX2 system [118]. For an insensitive channel, opamp biases and supplies to tubs are routed first; large swing analog signals are routed next; then analog ground is routed followed finally by digital signals. The analog

ground shields the large swing analog nets from the digital nets. In the sensitive channels, the power nets are routed first to ensure planarity. Fig. 78 shows the routing order for both types of channels. The cost function for ranking paths includes distance, penalties for crossing or running adjacent to noisy or sensitive nets, for switching layers, and such like. The channel router is also made capable of handling different wire widths and is gridless.

10.2 Analog Routing for Macro-Cell Layout

ANAGRAM [120], is part of a framework called ACACIA, developed at CMU for the design of common analog integrated circuit modules. ANAGRAM transforms sized schematics into mask geometry. It operates in the style of a macro-cell place and route system. The macro-cells can be individual devices such as FETs or groups of devices with special matching requirements. These building blocks are placed using a simulated annealing algorithm. A new analog-sensitive line-expansion router then connects all signal paths and power grids. In addition to the conventional incremental cost of a segment, the router incorporates first-order models of parasitic coupling effects(e.g. crosstalk on parallel runs, cross-overs, the ability of some nets to shield the unwanted interactions of noisy and sensitive analog paths, and resistive ground loops) and strives to place mutually interacting wiring paths sufficiently far apart. Like the digital routing scenario, the effect of wire-ordering is still critical even when sensitive nets are routed before noisy ones. Hence, ANAGRAM, performs a random ripup and reroute of all wiring in the hope of eliminating additional interactions.

LADIES, [124], also is applicable for macro-cell layout. It first decomposes the routing areas for each layer into rectangles and constructs an adjacency graph, using which it performs shortest-path global routing. The river routing algorithm is used to do the detailed wiring. A rule-based program is finally run to improve the layout.

Other Approaches: OPASYN [122], is restricted to the synthesis of operational amplifiers and uses the MIGHTY switchbox router for this purpose. BLADES [125], is a rule-based system that has been able to synthesize a few circuits such as op-amps, current and voltage sources.

11 Conclusion

Full-custom IC layout has come a long way in the eighties from the so-called *polygon-pushing* graphics systems, like GDS, where each transistor and wire was laid down by hand, to the modern layout systems like Mentor's Chip Station, Daisy's Chipmaster, Valid's Scaldstar VLSI system and Seiko's SX9000 system. These editors offer, in addition, integration with schematics and simulation tools. Automatic design-rule checkers and extraction tools are also typically provided. Extraction makes it possible to back-annotate resistance and capacitance values to schematic and simulation programs. For instance, LSI Logic provides an interconnect-delay estimator, a hierarchical floor-planner, and an RC-tree extraction tool which are integrated with their layout editor. Another innovation in IC design was *symbolic layout*. This meant that users could represent transistors, wires, and entire cells as symbols which were then loosely placed on the grid. A *compactor*, such as ECAD's [‡] SYMBAD, could then be used to shove these elements together in accordance to the design-rules.

Layout methodologies have had an influence on the routing algorithm used. For instance, channel routing is mostly used for standard-cell layout. Early commercial offerings included the LTX System of Bell Labs, CALMP and SCII from Silvar-Lisco and Tancell from Tangent Systems. Two good university place-and-route packages for standard-cell layout were the TimberWolfSC package from Berkeley and the VPNR package developed at Duke and MCNC. By a clever choice of equivalent pins and feedthrough locations, these systems do a good job of minimizing the total routing area. Though early gate-array place-and-route packages, such as Silvar-Lisco's 1981 GARDS, could handle only 5000 gates, the newer release, *viz.* AVANT GARDS, boasts of a capacity of up to 250,000 gates with a 100% routing guarantee. In addition to the standard channeled architecture, it also supports sea-of-gates design.

The first PCB CAD systems from vendors such as Computervision (now part of Prime Computer) and APPLICON (now part of Schlumberger) were simply automated drafting systems that ran in batch-mainframe or minicomputer environments. In 1985, Racal-Redalc came out with a gridless router called

[‡]ECAD and Tangent Systems are now part of Cadence

Company	Product and Cost	Hosts	Layout Styles	Routing features
Daisy Systems	GATEMASTER $20K	IBM PC AT	GA: c+sg; 1 and 2 routing layers	MCGP; Supports interactive editing
ECAD Inc.	SYMBAD $50K-$100K	VAX, Apollo, Sun, Tektronix	SC, BI, Gridless algorithms, Symbolic	GCSP, supports ripup, does power and ground sizing
LSI Logic Corp.	LDS3 Layout $300K	Sun, IBM, VAX, Silicon Graphics station	GA: c+sg; SC, BI	MGCSP; interactive; clock routing
Mentor Graphics	GATE Station, Cell Station $50K to $110K	Apollo	GA, SC, 2 routing layers	MCGLP; interactive
Seattle Silicon	SuperGlue $70K-$150K	Apollo, Valid SCALDSTAR	SC, BI, 1 or 2 routing layers (CMOS)	GCP; interactive
Seattle Silicon	Analog/APAR+	Apollo	Analog place and route	power analysis; buffer sizing; interactive
Silvar-Lisco	AVANT GARDS $60K+	Apollo, Digital, Sun, IBM	GA: c+sg+row,column,island; Mixed block and any size cell; 3 routing layers max.	Line-search and maze-cleanup router; interactive
Valid Logic Systems	Compose $25K	Sun	SC, Full-custom blocks, No restrictions on circuit size; 2.5 layers visible, unlimited layers supported	GMP
ABBREVIATION:			GA: Gate Array; SC: Standard Cell; BI: Block Interconnect; c+sg: channeled + sea-of-gates	M: Maze; C: Channel; S: Switchbox; G: Global; P: Power and ground; L: Line-probe

Table 7: Representative Commercial Automatic IC Layout Systems

VISULA, which divided the board into "obstacles" and "frontiers" instead of fixed grids. This approach decreased the memory requirements as compared to the traditional maze-router and could accommodate more densely packed traces, surface-mounted devices and fine-line technologies. Mentor's Board Station offered features such as two-sided simultaneous placement, true multilayer routing, variable grids according to pin pitch, and an ability to annotate properties from schematics to drive automatic placement and routing. The Allegro package from Valid Logic Systems provided expert routing systems and tools for high-speed board designs. Several vendors, such as PADS-PCB from CAD Software Inc., Master Designer from P-CAD, and Tango-Route from Accel Technologies now offer fairly good PCB layout packages for the IBM PC. Representative commercial systems that cater to analog design include Analog Artist Design System from Cadence and Chipcrafter/MAX from Seattle Silicon

Hardware engines have also been studied with an eye to speed up the maze routing process. It is believed that these will become prevalent as designers move to sub 25-mil carriers and double-sided surface mounting. Two commercial offerings from Daisy/Cadnetix include the Route Engine and the Voodoo router. The latter can accommodate up to ten i860 processors and 96 Mbytes of memory to solve multi-layer routing problems. A 5 to 20% speedup is claimed over more conventional routers. The WIRES system, developed at Waseda University, also consists of a hardware router to speed up rip-up and reroute operations.

Given the increasing complexity of current designs that can easily consist of upward of a hundred thousand transistors, the hunt for more efficient routing schemes will continue. Better heuristics are necessary if good quality solutions are to be obtained within a reasonable amount of time. The degree of conflict between such routing heuristics and those associated with the other phases of layout, such as placement, will finally determine the 'optimality' of the resulting solution. This means that the efficacy of one layout system can only be evaluated *relative* to that of another.

Newer technologies and layout styles also impose different requirements on the router. For example, with multiple wiring layers, the very concept of a channel loses much significance. The traditional two-layer channel router is no longer sufficient. Though a few multi-layer channel routers have been developed, maze routers still are predominantly used for multi-layer wiring. The areas of timing-driven layout and mixed-mode layout can also benefit substantially from further research. Though detailed routing is a very diverse field with a plethora of field-proven heuristics, many of the combinatorial and mathematical theories have yet to be properly understood.

References

[1] M. Lorenzetti and D. S. Baeder, "Routing," in *Physical Design Automation of VLSI Systems* (B. T. Preas and M. J. Lorenzetti, eds.), ch. 5, pp. 157–210, Menlo Park, CA 94025: The Benjamin/Cummings Publishing Company, Inc., 1988.

[2] J. Soukup, "Circuit layout," *Proc. of the IEEE*, vol. 69, pp. 1281–1304, Oct. 1981.

[3] E. S. Kuh and T. Ohtsuki, "Recent advance in VLSI layout," *Proceedings of the IEEE*, vol. 78, pp. 237–263, Feb. 1990.

[4] A. Sangiovanni-Vincentelli, *Design Systems for VLSI circuits*, ch. Automatic Layout of integrated circuits, pp. 113–195. NATO ASI Series, Dordrecht, the Netherlands: Martinus Nijhoff, 1987.

[5] R. Otten, *Design Systems for VLSI circuits*, ch. Layout Compilation, pp. 439–472. NATO ASI Series, Dordrecht, the Netherlands: Martinus Nijhoff, 1987.

[6] R. Goering, "Design automation," *High Performance Systems*, pp. 20–40, Dec. 1989.

[7] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley-Teubner, John Wiley and Sons, 1990.

[8] L. Abel, "On the ordering of connections for automatic wire routing," *IEEE Trans. on Computers*, pp. 1227–1233, Nov. 1972.

[9] S. B. Akers, "A modification of Lee's path connection algorithm," *IEEE Trans. on Electronic Computers*, pp. 97–98, Feb. 1967.

[10] C. Y. Lee, "An algorithm for path connections and its applications," *IEEE Trans. on Electronic Computers*, vol. VEC-10, pp. 346–365, Sept. 1961.

[11] T. Blank, M. Stefik, and W. van Cleemput, "A parallel bit map processor architecture forDA algorithms," in *Proc. 18th Design Automation Conference*, pp. 837–845, June 1981.

[12] W. A. Dees and R. J. Smith II, "Performance of interconnection rip-up and reroute strategies," in *Proc. 18th Design Automation Conference*, pp. 382–390, June 1981.

[13] W. A. Dees and P. G. Karger, "Automated rip-up and reroute techniques," in *Proc. 19th Design Automation Conference*, pp. 432–439, June 1982.

[14] W. Heyns, W. Sansen, and H. Beke, "A line-expansion algorithm for the general routing problem with a guaranteed solution," in *Proc. 17th Design Automation Conference*, pp. 243–249, June 1980.

[15] D. W. Hightower, "A solution to the line routing problem on a continuous plane," in *Proc. 6th Design Automation Conference*, pp. 1–24, June 1969.

[16] D. W. Hightower, "The interconnection problem -a tutorial," in *Proc. 10th Design Automation Conference*, pp. 1–21, June 1973.

[17] D. W. Hightower, "The Lee router revisited," in *Proc. Intl. Conf. on Computer Design*, pp. 136–139, Oct. 1983.

[18] D. W. Hightower, "A manually driven line-search router in a line editor," in *Digest Intl. Conf. on Computer Design*, pp. 276–278, Nov. 1985.

[19] B. L. Hulme and J. A. Wisniewski, "A comparision of shortest path algorithms applied to sparse graphs," Tech. Rep. NUREG/CR-0293,SAND78-1411, Sandia Laboratories, National Technical Information Service,Springfield, VA 22161, Aug. 1978.

[20] A. Iosupovici, "A class of array architectures for hardware grid routers," *IEEE Trans. On Computer-Aided Design*, vol. CAD-5, pp. 245–255, Apr. 1986.

[21] T. Ohtsuki, "Maze-running and line-search algorithms," in *Layout Design and Verification* (T. Ohtsuki, ed.), pp. 99–131, Amsterdam: North Holland, 1986.

[22] G. Persky, D. N. Deutsch, and D. G. Schweikert, "LTX a system for the directed automatic design of LSI circuits," in *Proc. 13th Design Automation Conference*, pp. 399–407, June 1976.

[23] F. Rubin, "The Lee path connection algorithm," *IEEE Trans. On Computer-Aided Design*, vol. c-23, pp. 907–914, Sept. 1974.

[24] F. Rubin, "An iterative technique for printed wire routing," in *Proc. 11th Design Automation Conference*, pp. 308–313, June 1974.

[25] R. A. Rutenbar, T. N. Mudge, and D. E. Atkins, "A class of cellular architectures to support physical design automation," *IEEE Trans. On Computer-Aided Design*, vol. CAD-3, pp. 264–278, Oct. 1984.

[26] J. Soukup, "Fast maze router," in *Proc. 15th Design Automation Conference*, pp. 100–102, June 1978.

[27] K. Suzuki, Y. Matsunaga, M. Tachibana, and T. Ohtsuki, "A hardware maze router with applications to interactive rip-up and reroute," *IEEE Trans. On Computer-Aided Design*, vol. CAD-5, pp. 466–476, Oct. 1986.

[28] R. A. Rutenbar and D. E. Atkins, "Systolic routing hardware: Performance evaluation and optimization," *IEEE Trans. On Computer-Aided Design*, vol. 7, pp. 397–410, Mar. 1988.

[29] R. Venkateswaran and P. Mazumder, "A hexagonal array machine for multi-layer wire routing," *IEEE Trans. On Computer-Aided Design*, vol. 9, pp. 1096–1112, Oct. 1990.

[30] S. Sahni and Y. Won, "A hardware accelerator for maze routing," in *Proc. 24th Design Automation Conference*, pp. 800–806, June 1987.

[31] E. Damm and H. Gethoeffer, "Hardware support for automatic routing," in *Proc. 19th Design Automation Conference*, pp. 219–223, June 1982.

[32] M. A. Breuer and K. Shamsa, "A hardware router," *Journal of Digital Systems*, vol. IV, no. 4, pp. 393–408, 1981.

[33] H. G. Adshead, "Employing a distributed array processor in a dedicated gate-array layout system," in *Proc. ICCC*, pp. 411–414, Oct. 1982.

[34] J. H. Hoel, "Some variations of Lee's algorithm," *IEEE Trans. on Computers*, vol. C-25, pp. 19–24, Jan. 1976.

[35] F. O. Hadlock, "A shortest path algorithm for grid graphs," in *Netsworks*, pp. 323–334, 1977.

[36] J. M. Geyer, "Connection routing algorithm for printed circuit boards," *IEEE Transactions on Circuit Theory*, vol. CT-18, pp. 95–101, Jan. 1971.

[37] K. Mikami and K. Tabuchi, "A computer program for optimal routing of printed circuit connectors," in *"Proc. of IFIPS"*, pp. 1475–1478, 1968.

[38] T. C. Hu and M. T. Shing, "The $\alpha - \beta$ routing," in *VLSI: Circuit Layout Theory* (T. C. Hu and E. S. Kuh, eds.), pp. 139–144, IEEE Press, 1985.

[39] D. N. Deutsch, "A dogleg channel router," in *Proc 13th Design Automation Conference*, pp. 425–433, June 1976.

[40] D. Braun, J. Burns, S. Devadas, H. K. Ma, K. Mayaram, F. Romeo, and A. Sangiovanni-Vincentelli, "Chameleon: a new multi-layer channel router," in *Proc. 23rd Design Automation Conference*, pp. 495 502, June 1986.

[41] D. Braun, J. Burns, S. Devadas, H. K. Ma, K. Mayaram, F. Romeo, and A. Sangiovanni-Vincentelli, "Techniques for multilayer channel routing," *IEEE Trans. On Computer-Aided Design*, vol. 7, pp. 698–711, June 1988.

[42] P. Bruell and P. Sun, "A greedy three layer channel router," in *Digest Intl. Conf. on Computer-Aided Design*, pp. 298–300, Nov. 1985.

[43] M. Burstein and R. Pelavin, "Hierarchical channel router," in *Proc. 20th Design Automation Conference*, pp. 591–597, June 1983.

[44] W. A. Dees and M. J. Lorenzetti, "Channel routing in a gate array environment," in *Proc. IEEE Intl. Conf. on Computer Design*, pp. 458–462, Oct. 1985.

[45] D. N. Deutsch, "Compacted channel routing," in *Digest Intl. Conf. on Computer-Aided Design*, pp. 223–225, Nov. 1985.

[46] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *Proc. 8th Design Automation Conference*, pp. 155–163, 1971.

[47] D. W. Hightower and R. Boyd, "A generalized channel router," in *Proc. 17th Design Automation Conference*, pp. 12–21, June 1980.

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

[49] U. Lauther, "Channel routing in a general cell environment," in *VLSI 85, VLSI Design of Digital Systems* (E. Horbst, ed.), pp. 393–403, Amsterdam, the Netherlands: Elsevier Science Publishers (North Holland), 1986.

[50] G. Persky, C. Enger, and D. M. Selove, "The Hughes automated layout system- automated LSI/VLSI layout based on channel routing," in *Proc. 18th Design Automation Conference*, pp. 22–28, June 1981.

[51] B. T. Preas, *Placement and Routing Algorithms for Hierarchical Integrated Circuit Layout.* PhD thesis, Stanford University, Aug. 1979.

[52] J. Reed, A. Sangiovanni-Vincentelli, and M. Santomauro, "A new symbolic channel router: YACR2," *IEEE Trans. On Computer-Aided Design*, vol. CAD-4, pp. 208–219, July 1985.

[53] R. L. Rivest and C. M. Fiduccia, "A 'greedy' channel router," in *Proc. 19th Design Automation Conference*, pp. 418–424, June 1982.

[54] J. Royle, M. Palczewski, H. VerHeyen, N. Naccache, and J. Soukup, "Geometrical compaction in one dimension for channel routing," in *Proc. 24th Design Automation Conference*, pp. 140–145, June 1987.

[55] T. G. Szymanski, "Dogleg channel routing is NP-complete," *IEEE Trans. On Computer-Aided Design*, vol. CAD-4, pp. 31–41, Jan. 1985.

[56] Y. K. Chen and M. L. Liu, "Three-layer channel routing," *IEEE Trans. On Computer-Aided Design*, vol. CAD-3, pp. 156–163, Apr. 1984.

[57] H. W. Leong, D. F. Wong, and C. L. Liu, "A simulated-annealing channel router," in *Proc. IEEE International Conf. on Computer-Aided-Design*, pp. 226–228, 1985.

[58] J. Cong, D. F. Wong, and C. L. Liu, "A new approach to three or four layer channel routing," *IEEE Trans. On Computer-Aided Design*, vol. 7, pp. 1094–1104, Oct. 1988.

[59] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing," *IEEE Trans. On Computer-Aided Design*, vol. CAD, pp. 180–190, Jan. 1982.

[60] K. Aoshima and E. S. Kuh, "Multichannel optimization in gate array LSI layout," in *Proc. ISCAS*, pp. 1005–1008, 1983.

[61] G. T. Hamachi and J. K. Ousterhout, "A switchbox router with obstacle avoidance," in *Proc. 21st Design Automation Conference*, pp. 173–179, June 1984.

[62] R. Joobbani, *An Artificial Intelligence Approach to VLSI Routing.* MA 02061: Kluwer Academic Publishers, 1986.

[63] W. K. Luk, "A greedy switch box router," *Integration, the VLSI journal*, vol. 3, pp. 129–149, 1985.

[64] J. P. Cohoon and P. L. Heck, "BEAVER: A computational geometry based tool for switchbox routing," *IEEE Trans. On Computer-Aided Design*, vol. 7, pp. 684–697, June 1988.

[65] Y. L. Lin, Y. C. Hsu, and F. S. Tsai, "SILK: A simulated evolution router," *IEEE Trans. On Computer-Aided Design*, vol. 8, pp. 1108–1114, Oct. 1989.

[66] H. Shin and A. Sangiovanni-Vincentelli, "Mighty: a rip-up and reroute detailed router," in *ICCAD*, pp. 2–5, Nov. 1986.

[67] C. P. Hsu, "General river routing algorithm," in *Proc. 20th Design Automation Conference*, pp. 378–583, June 1983.

[68] C. P. Hsu, "Minimum via topological routing," *IEEE Trans. On Computer-Aided Design*, vol. CAD-2, pp. 235–246, Oct. 1983.

[69] M. Marek-Sadowska and T. T. Tarng, "Single layer routing for VLSI: analysis and algorithms," *IEEE Trans. On Computer-Aided Design*, vol. CAD-2, pp. 246–259, Oct. 1983.

[70] R. Y. Pinter, "River routing: methodology and analysis," in *The Third CalTech Conference on VLSI*, pp. 141–163, Mar. 1983.

[71] M. Burstein and R. Pelavin, "Hierarchical wire routing," *IEEE Trans. On Computer-Aided Design*, vol. CAD-2, pp. 223–234, Oct. 1983.

[72] G. T. Hamachi and J. K. Ousterhout, "Magic's obstacle avoiding global router," in *Chapel Hill Conf. on VLSI*, (Rockville,Maryland), pp. 145–164, Computer Science Press Inc., 1985.

[73] J. Kessenich and G. Jackoway, "Global forced hierarchical router," in *Proc. 23rd Design Automation Conference*, pp. 798–802, June 1986.

[74] J. T. Li and M. Marek-Sadowska, "Global routing for gate array," *IEEE Trans. On Computer-Aided Design*, vol. CAD-3, pp. 298–307, Oct. 1984.

[75] R. Nair, S. J. Hong, S. Liles, and R. Villani, "Global wiring on a wire-routing machine," in *Proc 19th Design Automation Conference*, pp. 224–231, June 1982.

[76] J. Ousterhout, G. Hamachi, R. Mayo, W. Scott, and G. Taylor, "Magic: a VLSI layout system," in *Proc. 21st Design Automation Conference*, pp. 154–159, June 1984.

[77] A. M. Patel, N. L. Soong, and R. K. Korn, "Hierarchical VLSI routing: an approximate routing procedure," *IEEE Trans. On Computer-Aided Design*, vol. CAD-4, pp. 121–126, Apr. 1985.

[78] B. S. Ting and B. N. Tien, "Routing techniques for gate array," *IEEE Trans. On Computer-Aided Design*, vol. CAD-2, pp. 301–312, Oct. 1983.

[79] M. P. Vecchi and S. Kirkpatrick, "Global wiring by simulated annealing," *IEEE Trans. On Computer-Aided Design*, vol. CAD-2, pp. 215–222, Oct. 1983.

[80] J. G. Xiong, "Algorithms for global routing," in *Proc. 23rd Design Automation Conference*, pp. 824-830, June 1986.

[81] W. M. Dai, H. H. Chen, *et al.*, "BEAR: a new building block layout system," in *ICCAD*, pp. 34–37, Nov. 1987.

[82] C. Sechen, D. Braun, and A. Sangiovanni-Vincentelli, "Thunderbird: A complete standard cell layout package," *IEEE Journal of Solid-State Circuits*, vol. 23, pp. 410–420, Apr. 1988.

[83] K. Winter and D. A. Mlynski, "Hierarchical loose routing for gate arrays," *IEEE Trans. On Computer-Aided Design*, vol. CAD-6, pp. 810–819, Sept. 1987.

[84] W. K. Luk, P. Sipda, *et al.*, "A hierarchical global wiring algorithm for custom chip design," *IEEE Trans. On Computer-Aided Design*, vol. CAD-6, pp. 518–533, July 1987.

[85] W. M. Dai and E. S. Kuh, "Simultaneous floorplanning and global routing for hierarchical building-block layout," *IEEE Trans. On Computer-Aided Design*, vol. CAD-6, pp. 828–837, Sept. 1987.

[86] W. M. Dai, T. Asano, and E. S. Kuh, "Routing region definition and ordering scheme for building-block layout," *IEEE Trans. On Computer-Aided Design*, vol. CAD-4, pp. 189–197, July 1985.

[87] R. Nair, "A simple yet effective technique for global wiring," *IEEE Trans. On Computer-Aided Design*, vol. CAD-6, pp. 165–172, Mar. 1987.

[88] H. Cai and R. Otten, "Conflict-free channel definition in building-block layout," *IEEE Trans. On Computer-Aided Design*, vol. 8, pp. 981–988, Sept. 1989.

[89] D. T. Lee, S. J. Hong, and C. K. Wong, "Number of vias: A control parameter for global wiring of high-density chips," *IBM Journal of Research and Development*, vol. 25, no. 4, pp. 261–271, 1981.

[90] X.-M. Xiong and E. S. Kuh, "A unified approach to the via minimization problem," *IEEE Trans. On Computer-Aided Design*, vol. 36, pp. 190–204, Feb. 1989.

[91] N. J. Naclerio, S. Masuda, and K. Nakajima, "The via minimization problem is NP-complete," *IEEE Trans. on Computers*, vol. 38, pp. 1604–1608, Nov. 1989.

[92] Marek-Sadowska, M., "An unconstrained topological via minimization problem for two-layer routing," *IEEE Trans. On Computer-Aided Design*, vol. CAD-3, pp. 184–190, July 1984.

[93] K. C. Chang and H. C. Du, "Efficient algorithms for layer-assignment problems," *IEEE Trans. On Computer-Aided Design*, pp. 67–78, Jan. 1987.

[94] Y. Kajitani, "On via-hole minimization of routing on 2-layer boards," in *Proc. ICCC*, pp. 295–298, 1980.

[95] K. R. Stevens and W. M. VanCleemput, "Global via elimination in generalized routing environment," in *Proc. IEEE Symposium on Circuits and Systems*, (Tokyo, Japan), pp. 689–692, June 1979.

[96] M. J. Cielelski and E. Kinnen, "An optimum layer assignment for routing in ICs and PCBs," in *Proc. 18th Design Automation Conference*, pp. 733–737, June 1981.

[97] J. M. Ho, M. Sarrafzadeh, G. Vijayan, and C. K. Wong, "Layer assignment for multi-chip modules," *IEEE Trans. On Computer-Aided Design*, vol. 9, pp. 1272–1277, Dec. 1990.

[98] S. Chowdury and M. A. Breuer, "The construction of minimum area power and ground nets for VLSI circuits," in *Proc. 22nd Design Automation Conference*, pp. 794–797, June 1985.

[99] S. Chowdury, "An automated design of minimum area IC power/ground nets," in *Proc. 24th Design Automation Conference*, pp. 223–229, June 1987.

[100] A. Moulton, "Laying the power and ground wires on a VLSI chip," in *Proc. 20th Design Automation Conference*, pp. 754–755, June 1983.

[101] H. Rothermal and D. Mlynski, "Computation of power supply nets in VLSI layout," in *Proc. 18th Design Automation Conference*, pp. 37–42, June 1981.

[102] Z. Syed and A. El Gamal, "Single layer routing of power and ground networks in integrated circuits," *Journal of Digital Systems*, vol. 6, no. 1, pp. 53–63, 1982.

[103] R. Dutta and M. Marek-Sadowska, "Automatic sizing of power/ground networks in VLSI," in *Proc. 26th Design Automation Conference*, pp. 783–786, June 1989.

[104] S. Chowdury, "Optimum design of reliable IC power networks having general graph topologies," in *Proc. 26th Design Automation Conference*, pp. 787–790, June 1989.

[105] S. Chowdury and M. A. Breuer, "Optimum design of IC power/ground nets subject to reliability constraints," *IEEE Trans. On Computer-Aided Design*, vol. 7, pp. 787–796, July 1988.

[106] M. Marek-Sadowska, "Pad assignment for power nets in VLSI circuits," *IEEE Trans. On Computer-Aided Design*, vol. CAD-6, pp. 550–560, July 1987.

[107] X. Xiong and E. S. Kuh, "The scan line approach to power and ground routing," in *ICCAD*, pp. 6–10, Nov. 1986.

[108] S. Haruyama and D. Fussell, "A new area-efficient power routing algorithm for VLSI layout," in *ICCAD*, pp. 38–41, Nov. 1987.

[109] D. F. Wann and M. A. Franklin, "Asynchronous and clocked control structures for VLSI based interconnection networks," *IEEE Trans. on Computers*, vol. C-32, pp. 284–293, Mar. 1983.

[110] P. Ramanathan and K. G. Shin, "A clock distribution scheme for non-symmetric VLSI circuits," in *IEEE International Conf. on Computer-Aided-Design*, pp. 398–401, Nov. 1989.

[111] S. Han and S. Sahni, "Single-row routing on narrow streets," *IEEE Trans. On Computer-Aided Design*, vol. CAD-3, pp. 235–241, July 1984.

[112] T. T. Tarng, M. Marek-Sadowska, and E. S. Kuh, "An efficient single row routing algorithm," *IEEE Trans. On Computer-Aided Design*, vol. CAD-3, pp. 178–183, July 1984.

[113] P. Lund, *PCB Precision Artwork Generation and Manufacturing Methods.* Bishop Graphics, Inc., 1986.

[114] E. S. Kuh, T. Kashiwabara, and T. Fujisawa, "On optimum single-row routing," *IEEE Trans. on Circuits and Systems*, vol. CAS-26, pp. 361–368, June 1979.

[115] R. Raghavan and S. Sahni, "Single row routing," *IEEE Trans. on Computers*, vol. C-32, pp. 209–220, Mar. 1983.

[116] R. Anastasi, "Routing algorithms and grids for advanced board design," *VLSI Systems Design*, pp. 54–58, Sept. 1987.

[117] M. Kayal *et al.*, "SALIM: A layout generation tool for analog ICs," in *IEEE Custom Integrated Circuits Conf.*, pp. 7.5.1–7.5.4, May 1988.

[118] C. D. Kimble *et al.*, "Autorouted analog VLSI," in *ICICC*, pp. 72–76, 1985.

[119] J. Rijmenants *et al.*, "ILAC: An automated layout tool for analog CMOS circuits," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 417–425, Apr. 1989.

[120] L. R. Carley *et al.*, "ACACIA: The CMU analog design system," in *ICICC*, pp. 4.3.1–4.3.5, May 1989.

[121] E. Berkcan *et al.*, "From analog design description to layout," in *ICICC*, (Schenectady, NY 12301), pp. 4.3.1–4.3.5, General Electric Co., May 1989.

[122] H. Y. Koh, C. H. Sequin, and P. R. Gray, "Automatic layout generation for CMOS operational amplifiers," in *ICCAD*, 1988.

[123] R. Harjani, R. A. Rutenbar, and L. C. Carley, "OASYS: A framework for analog circuit synthesis," *IEEE Trans. On Computer-Aided Design*, vol. 8, pp. 1247–1266, Dec. 1989.

[124] M. Mogaki *et al.*, "LADIES: An automatic layot system for analog LSIs," in *ICCAD*, pp. 450–453, Nov. 1989.

[125] F. El-Turky and E. E. Perry, "BLADES: An AI approach to analog circuit design," *IEEE Trans. On Computer-Aided Design*, vol. 8, pp. 680–692, June 1989.

[126] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[127] T. Asano, M. Sata, and T. Ohtsuki, "Computational geometry algorithms," in *Layout Design and Verification* (T. Ohtsuki, ed.), pp. 295–345, Amsterdam: North Holland, 1986.

[128] E. M. McCreight, "Priority search trees," *SIAM Journal of Computing*, vol. 14, pp. 257–276, May 1985.

[129] A. Aho, M. R. Garey, and F. K. Hwang, "Rectilinear steiner trees: efficient special case algorithms," in *Netsworks, 7*, pp. 37–58, 1977.

[130] E. G. Coffman and R. L. Graham, "Optimal scheduling for two processor systems," *Acta Informatica*, vol. 1, pp. 200–213, 1972.

[131] J. L. Bentley and T. Ottmann, "Algorithm for reporting and counting geometric intersections," *IEEE Trans. on Computers*, vol. C-28, pp. 643–647, Sept. 1979.

[132] J. Mukres, "Algorithms for the assignment and transportation problems," *J. SIAM*, vol. 5, pp. 32–38, Mar. 1957.

[133] M. Hanan, "On steiner's problem with rectilinear distance," *SIAM Journal of Applied Math.*, vol. 14, no. 2, pp. 255–265, 1966.

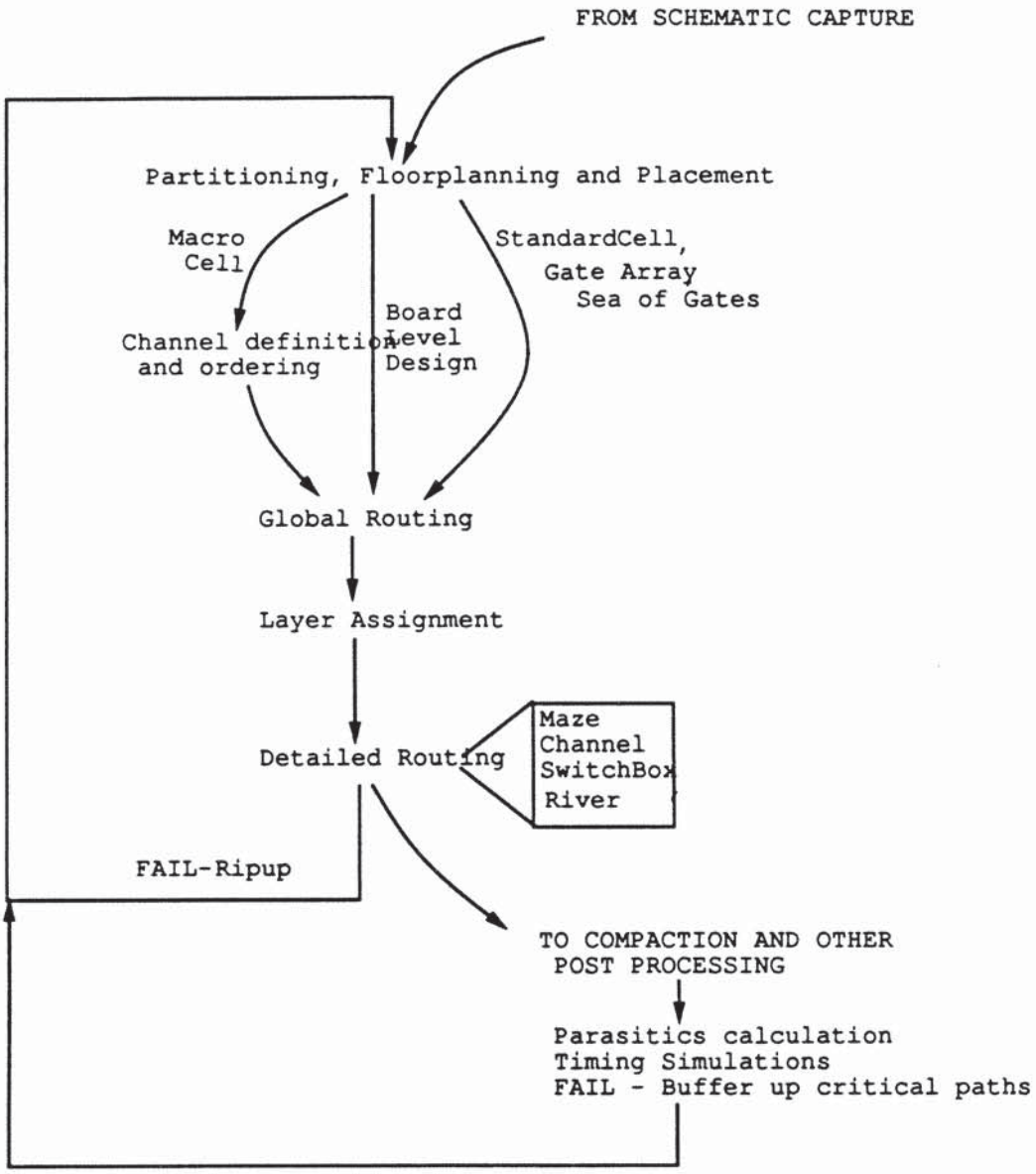


Figure 1: Steps in automated layout

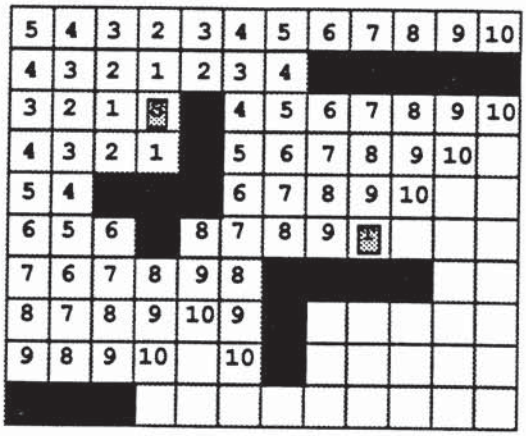


Figure 2: Lee wave propagation phase

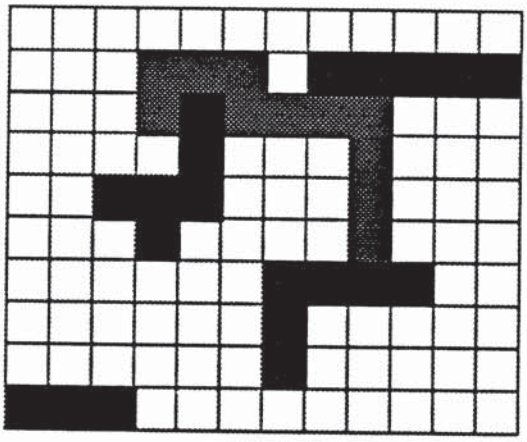


Figure 3: Final Step of the algorithm

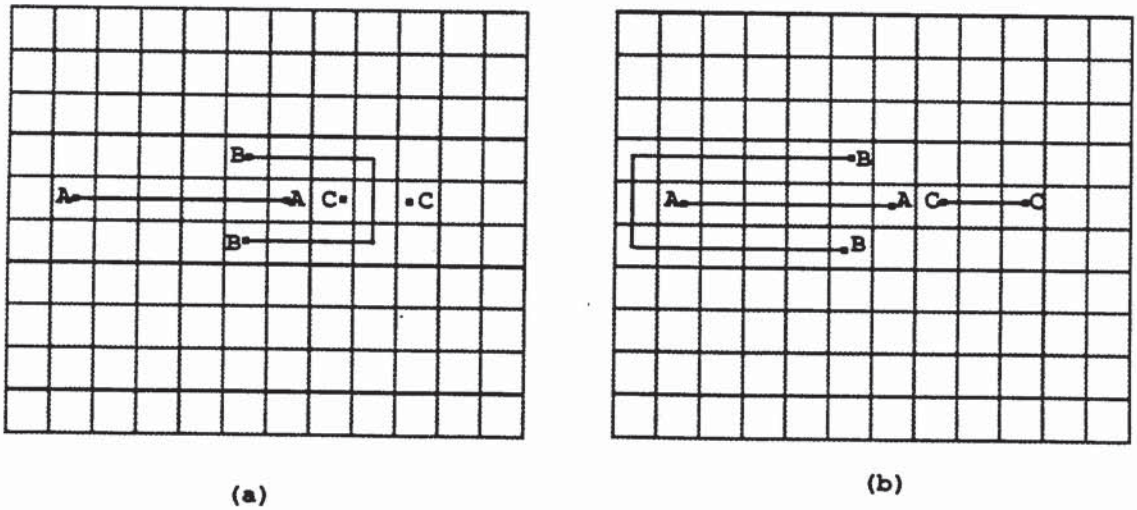


Figure 4: (a) An incomplete solution by the Lee algorithm (b) Correct Solution

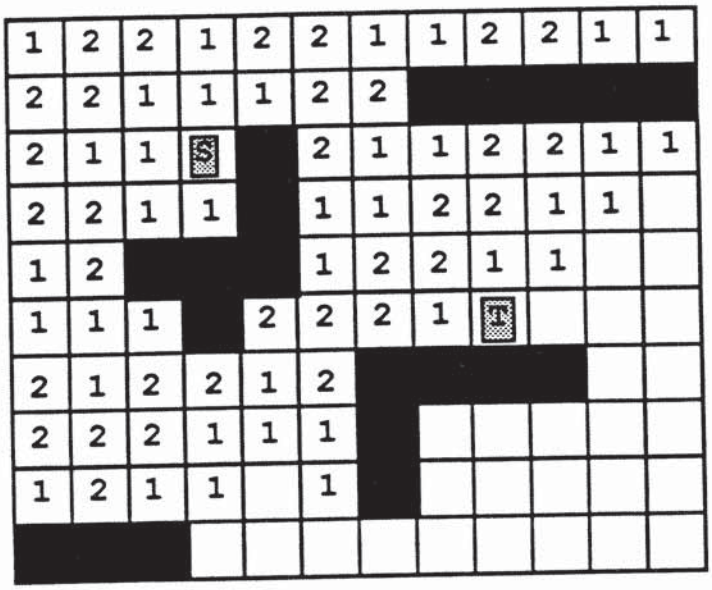


Figure 5: A 1,1,2,2 labeling sequence. The target is labeled 1

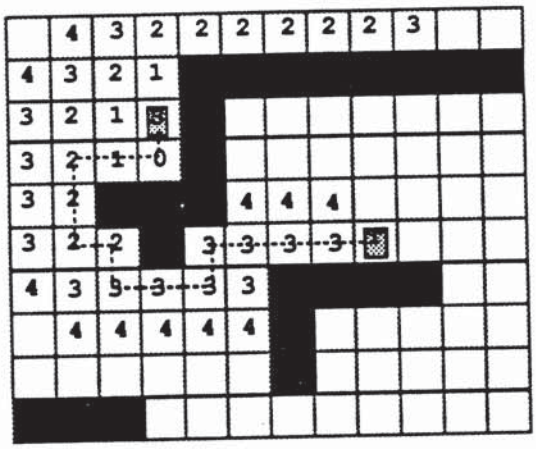


Figure 6: Detour numbers

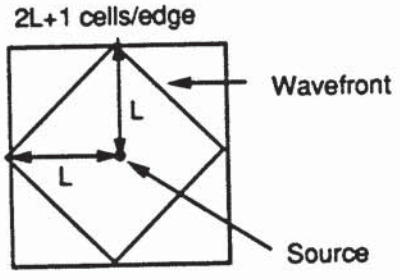


Figure 7: Sample problem for comparing maze-routers

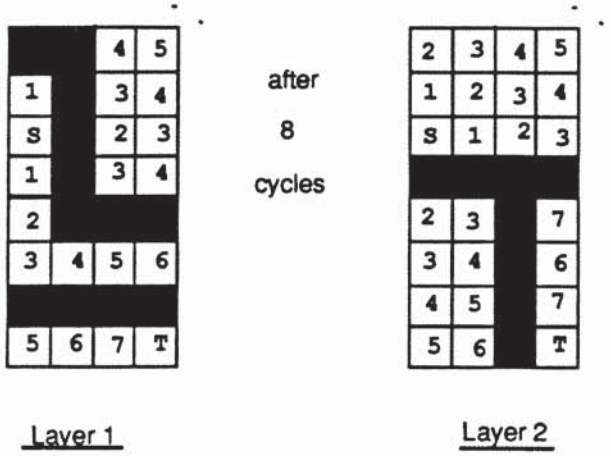


Figure 8: 2-layer routing based on Geyer's algorithm

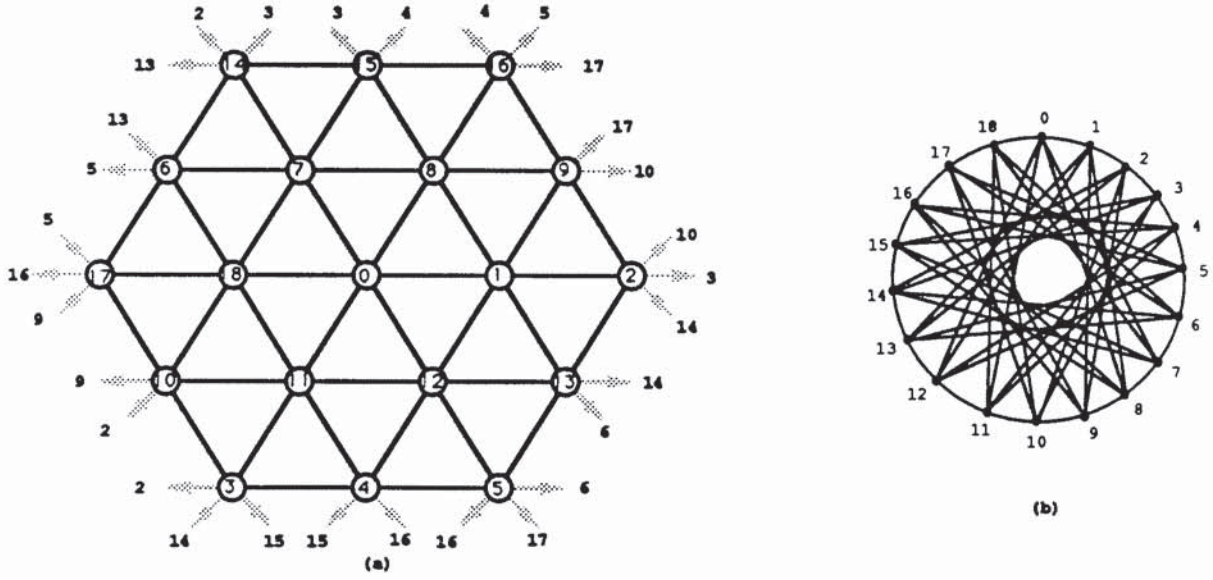


Figure 9: (a) A wraparound hexagonal mesh of dimension 3 (b) its chordal equivalent

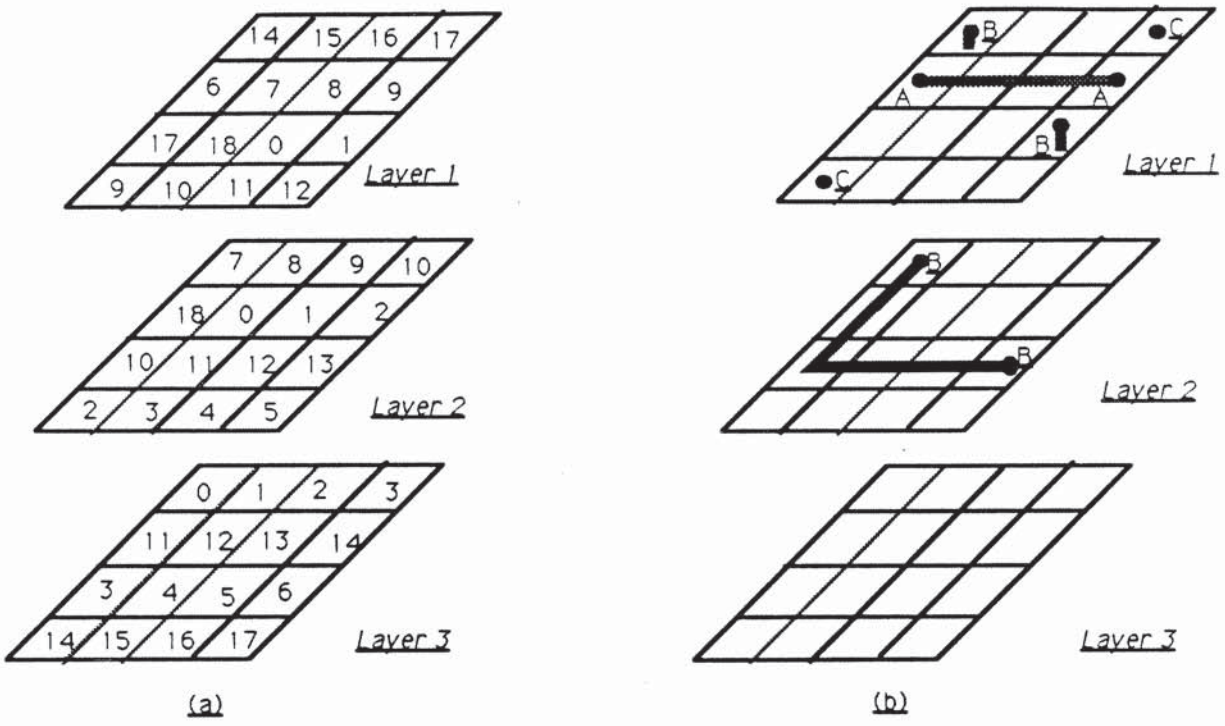


Figure 10: (a) Processor Assignments for a 3-layer 4 × 4 grid (b) Initial Netlist

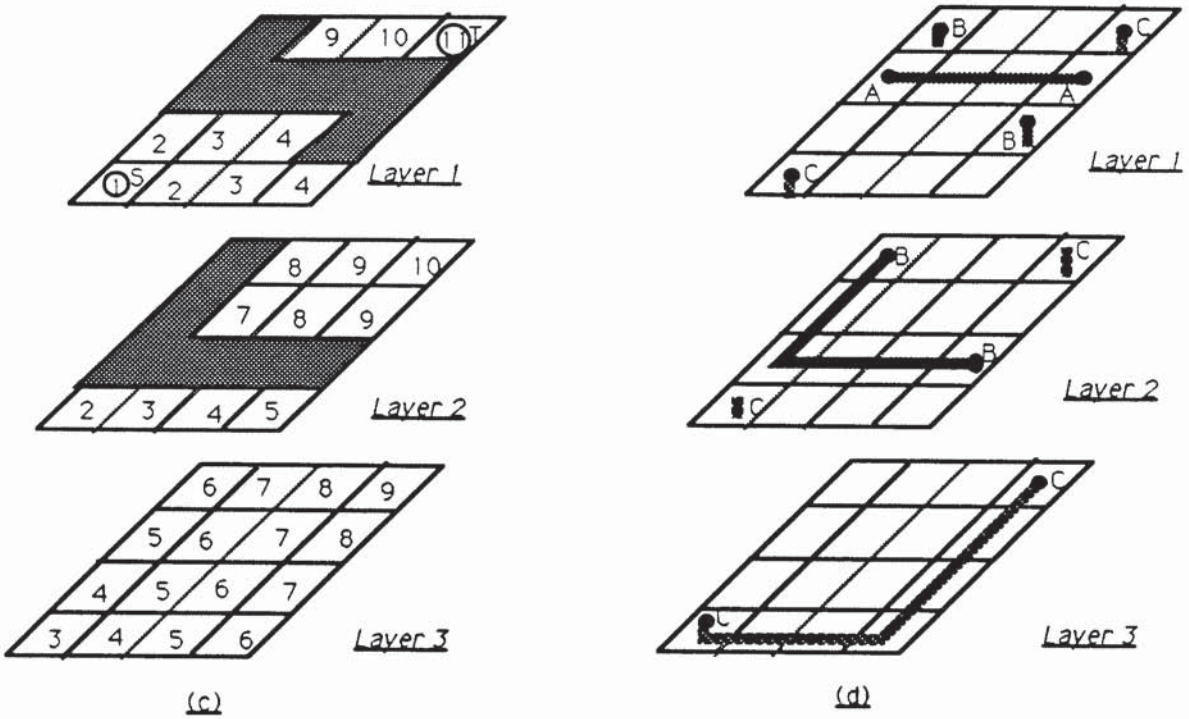
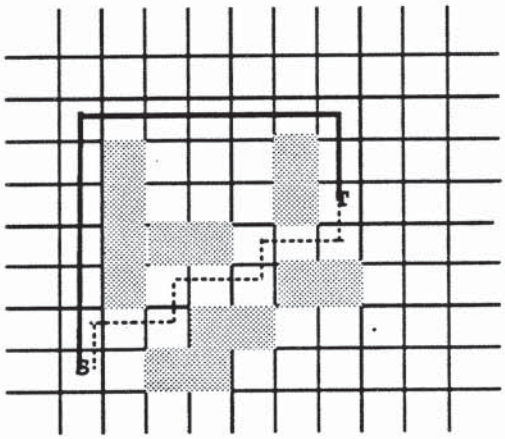


Figure 11: (a)Snapshot of wavefronts on the 3 layers (b) Final wiring



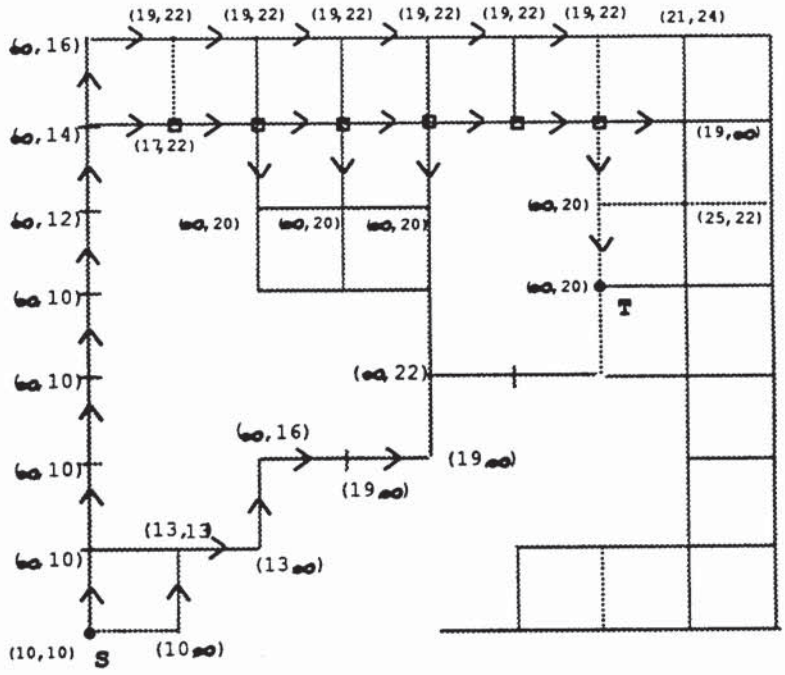
Path found by $\alpha-\beta$ router

Path found by Maze router

(a) Sample Problem

$\alpha = 1$

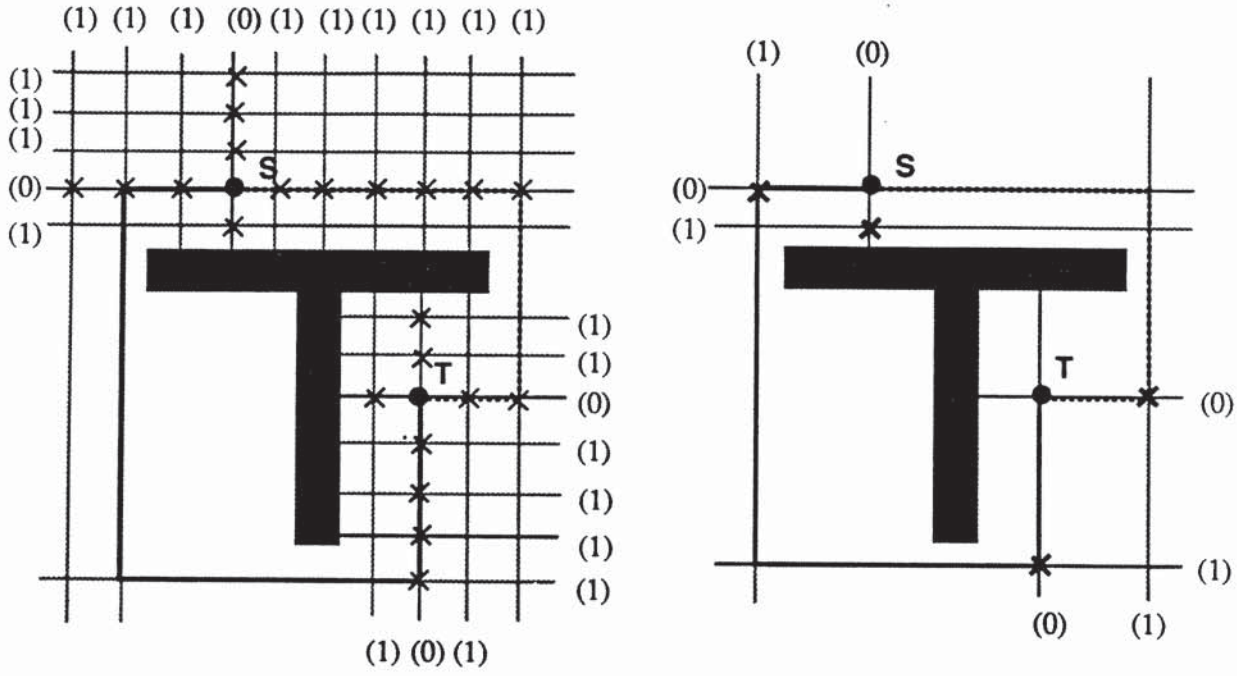
$\beta = 3$



(All nodes marked □ are labeled (17,22))

(b) DAG representation with (p_j, q_j) values shown

Figure 12: The $\alpha - \beta$ router

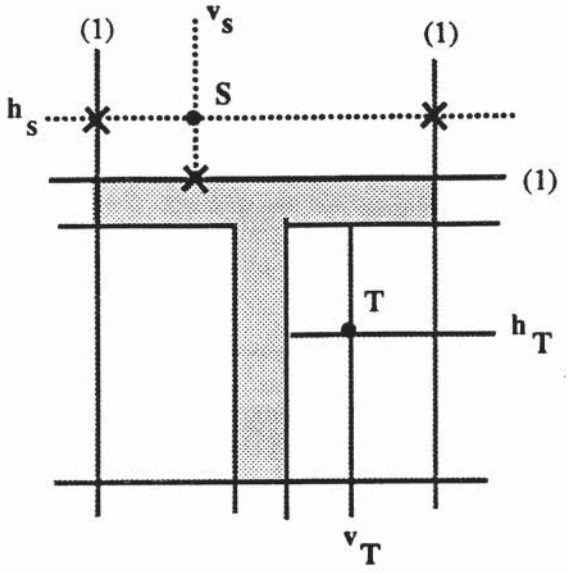


(a) Mikami-Tabuchi Algorithm

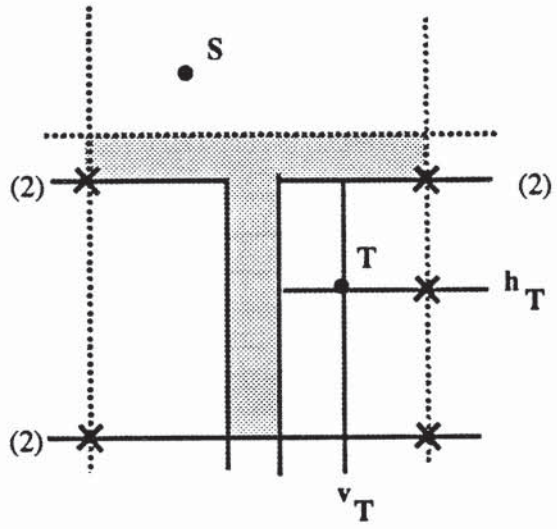
(b) Hightower Algorithm

× Escape point

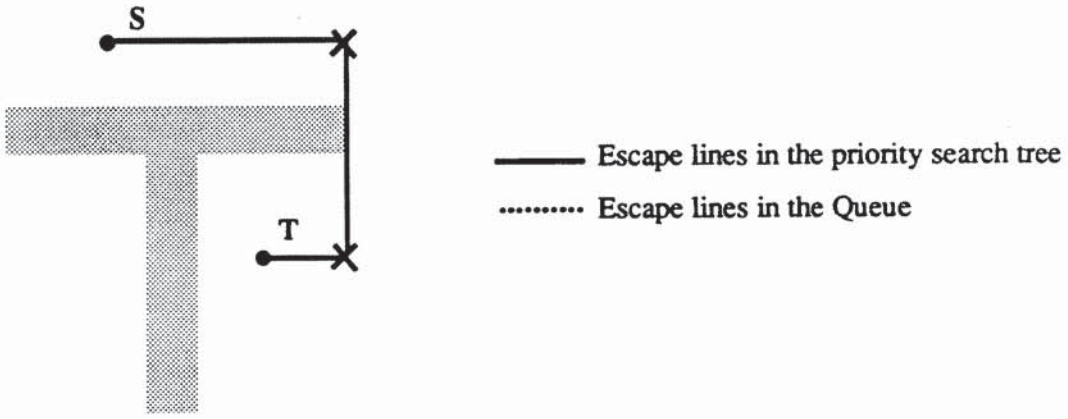
Figure 13: Application of Mikami and Hightower algorithm on a simple problem



(a) Escape Lines of Level 1



(b) Escape Lines of Level 2



(c) Generated Minimum Bend Path

Figure 14: Application of Ohtsuki's Line-Search Algorithm

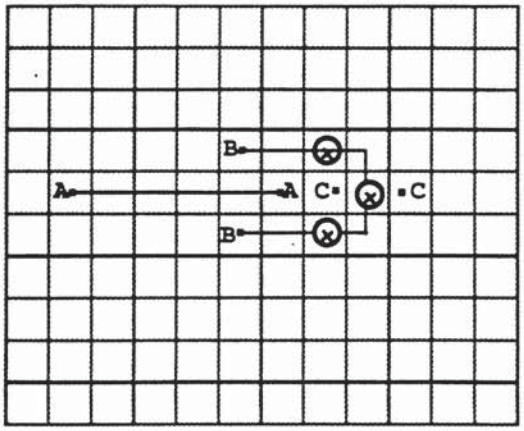
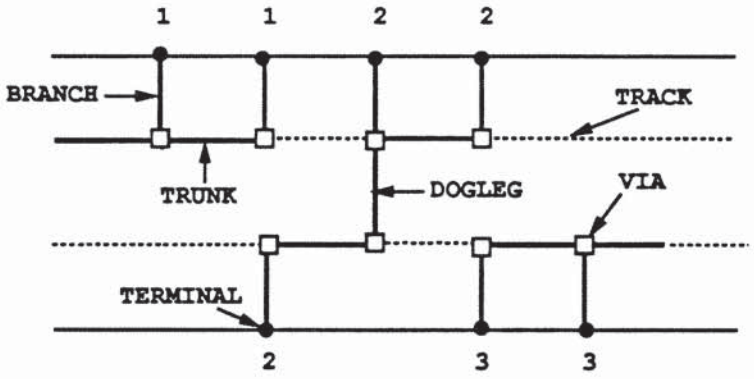


Figure 15: Minimal Separators



$T_L = \{1,1,2,2,0\}$ $B_L = \{0,2,0,3,3\}$

$L_L = \{1\}$ $R_L = \{3\}$

Figure 16: Channel Routing Terminology

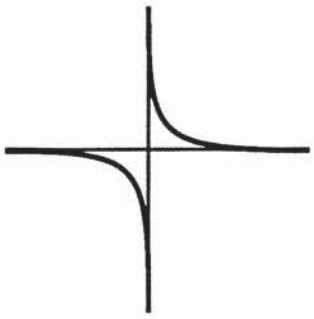


Figure 17: A knock-knee

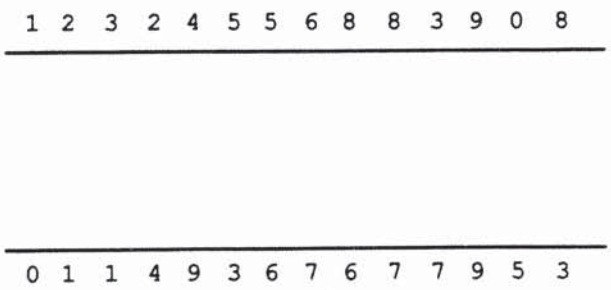


Figure 18: A simple channel routing problem P_{ch}

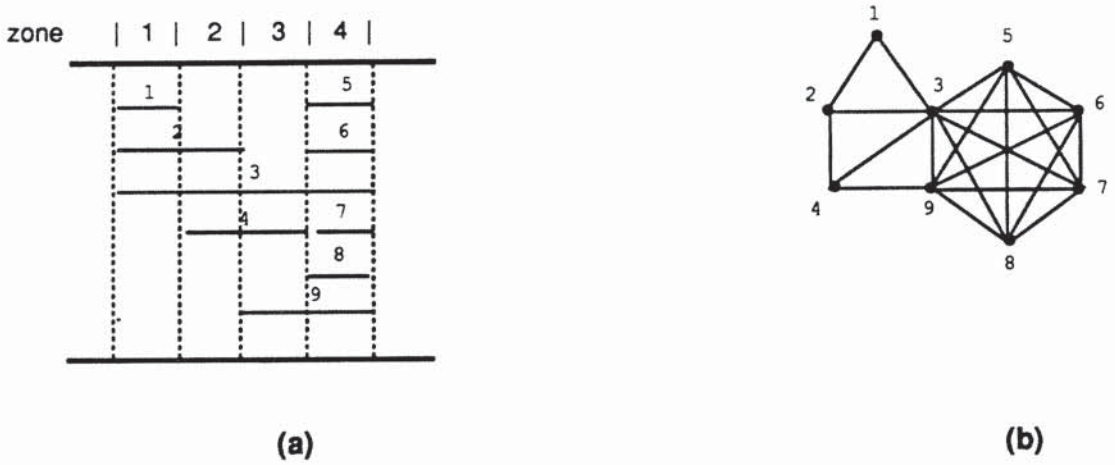


Figure 19: (a) Zone representation and (b) Horizontal Cover Graph for P_{ch}

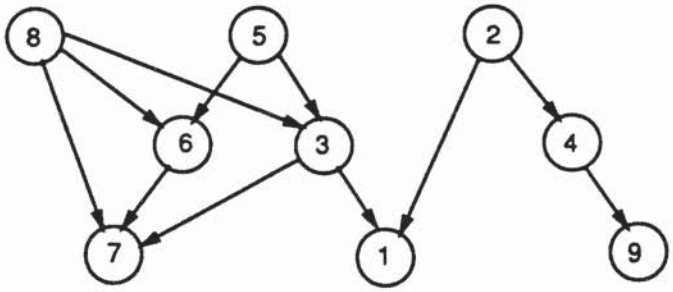


Figure 20: Vertical Constraint Graph for P_{ch}

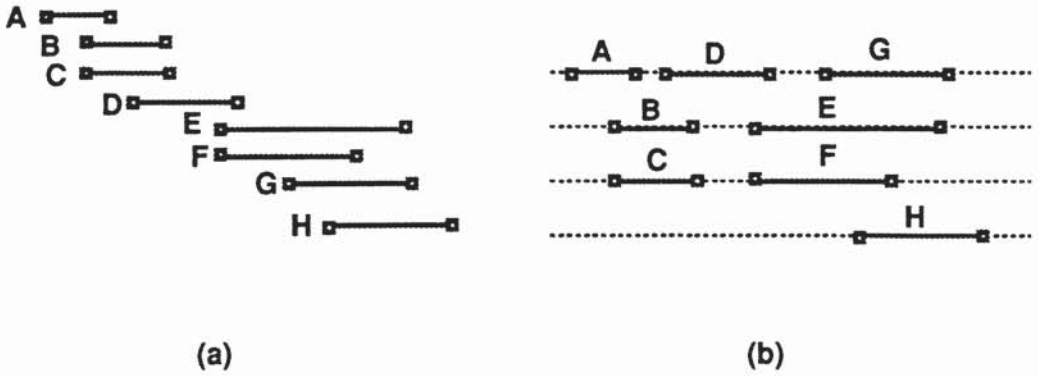
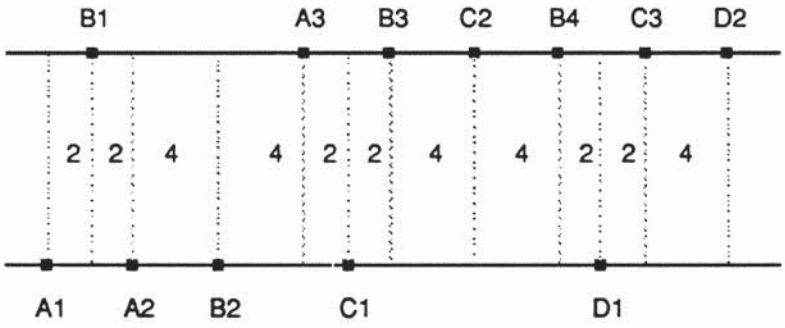
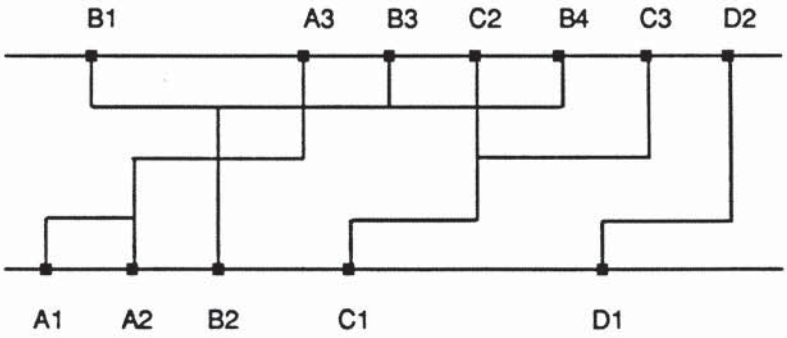


Figure 21: (a) Segments sorted by starting abscissae (b) Segments assigned to tracks by LEA



(a) Netlist with terminal spacings shown



(b) General Dogleg Router Solution

Figure 22: Dogleg Router

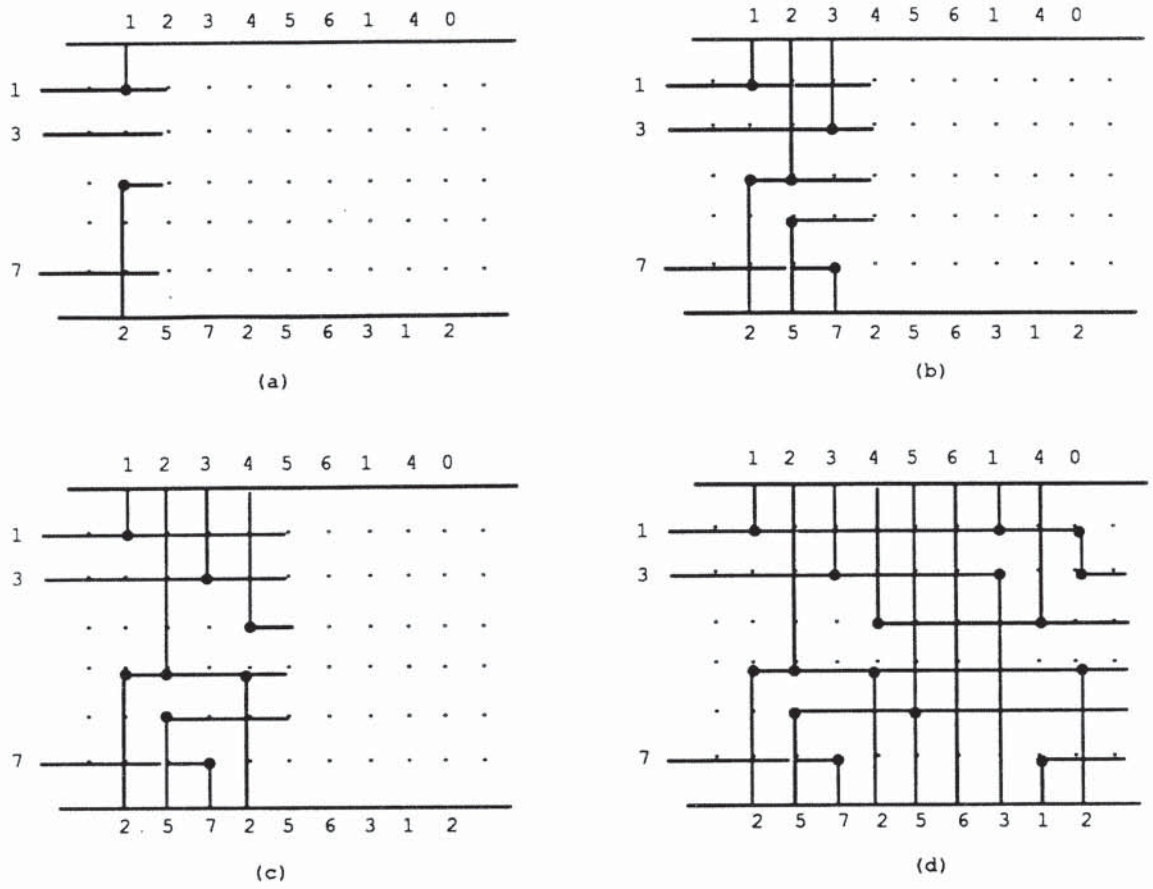


Figure 23: Channel after routing of (a) column 1 (b)column 3 (c)column 4 (d) column 9

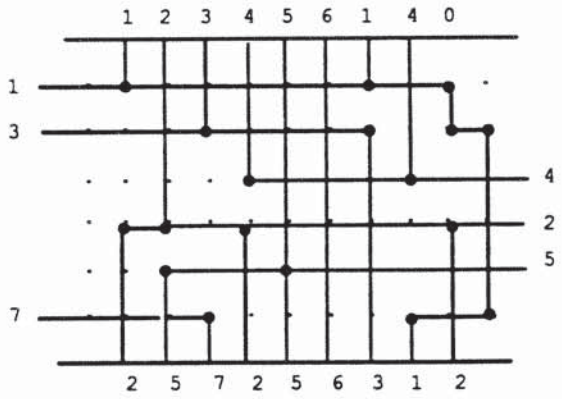


Figure 24: Final routing

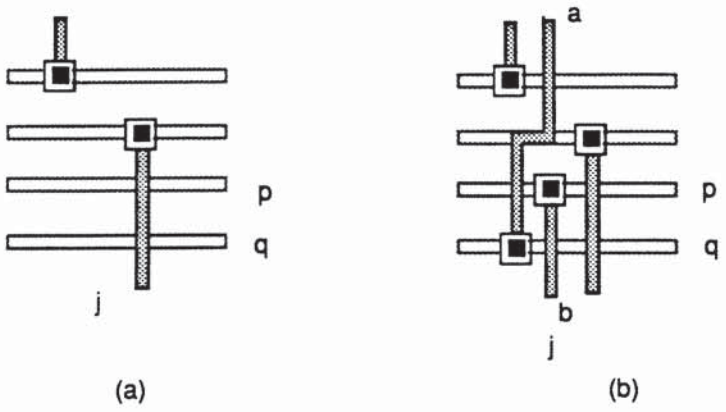


Figure 25: (a)Before maze1 routing (b) After maze1 routing

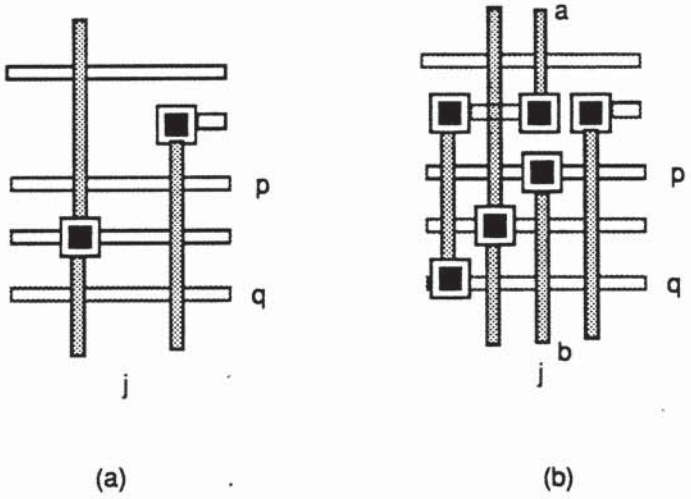


Figure 26: (a)Before maze2 routing (b) After maze2 routing

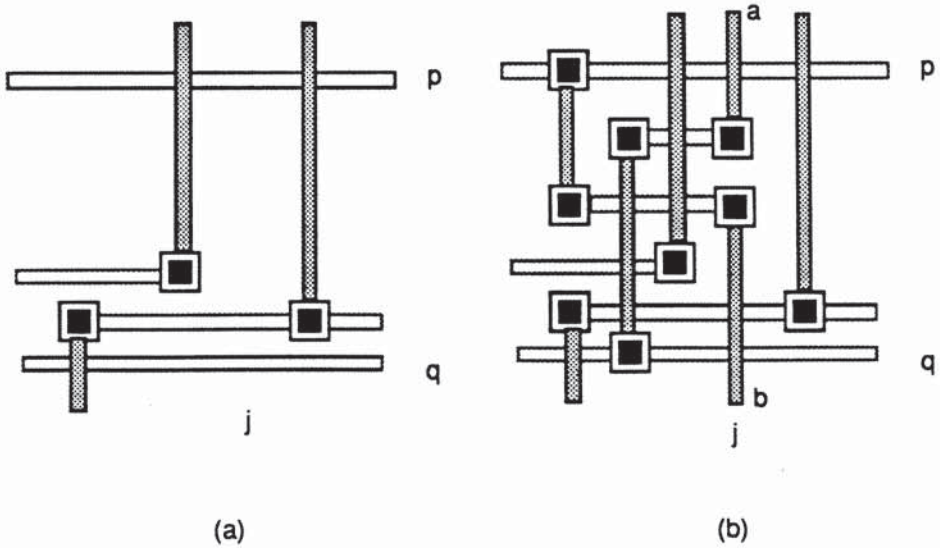


Figure 27: (a)Before maze3 routing (b) After maze3 routing

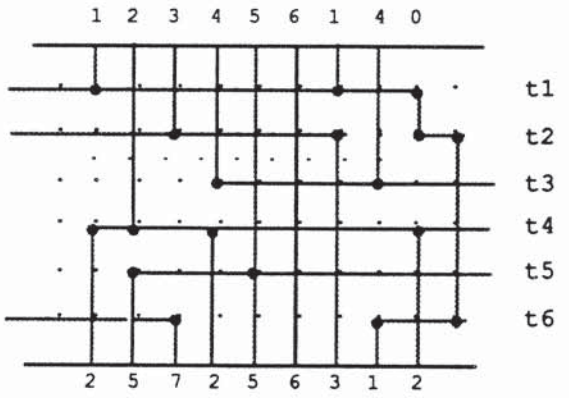
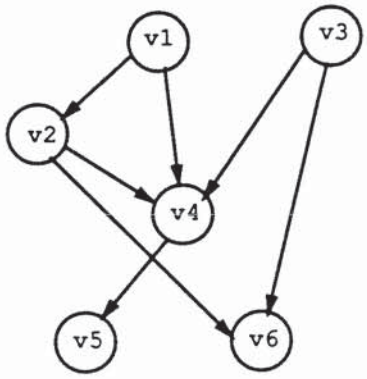


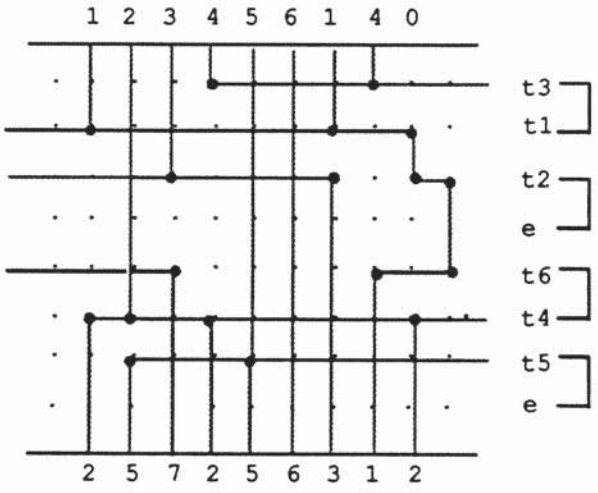
Figure 28: An example of a two-layer routing solution



(a)

	P1	P2
time = 1	t3	t1
2	t2	
3	t6	t4
4	t5	

(b)



(c)

Figure 29: (a)Track Ordering Graph (b) An optimal solution for a two-processor schedule (c) Induced perfect pairing

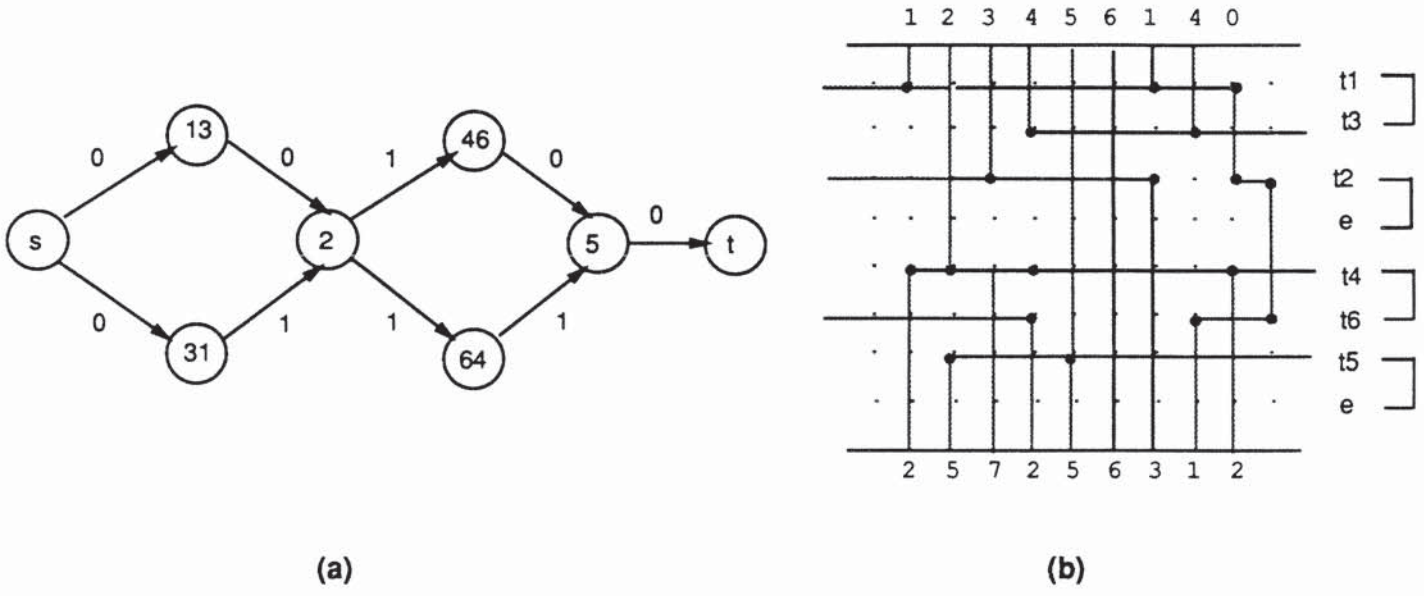


Figure 30: (a) Track-group Order Graph (b) Solution after relative track ordering arrangement

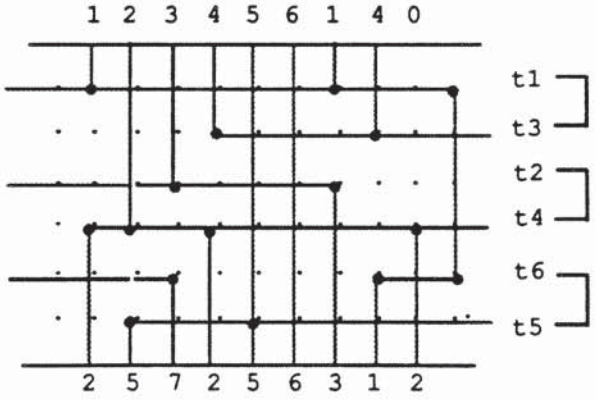


Figure 31: Final 3-track HVH solution after singular track shifting and rerouting

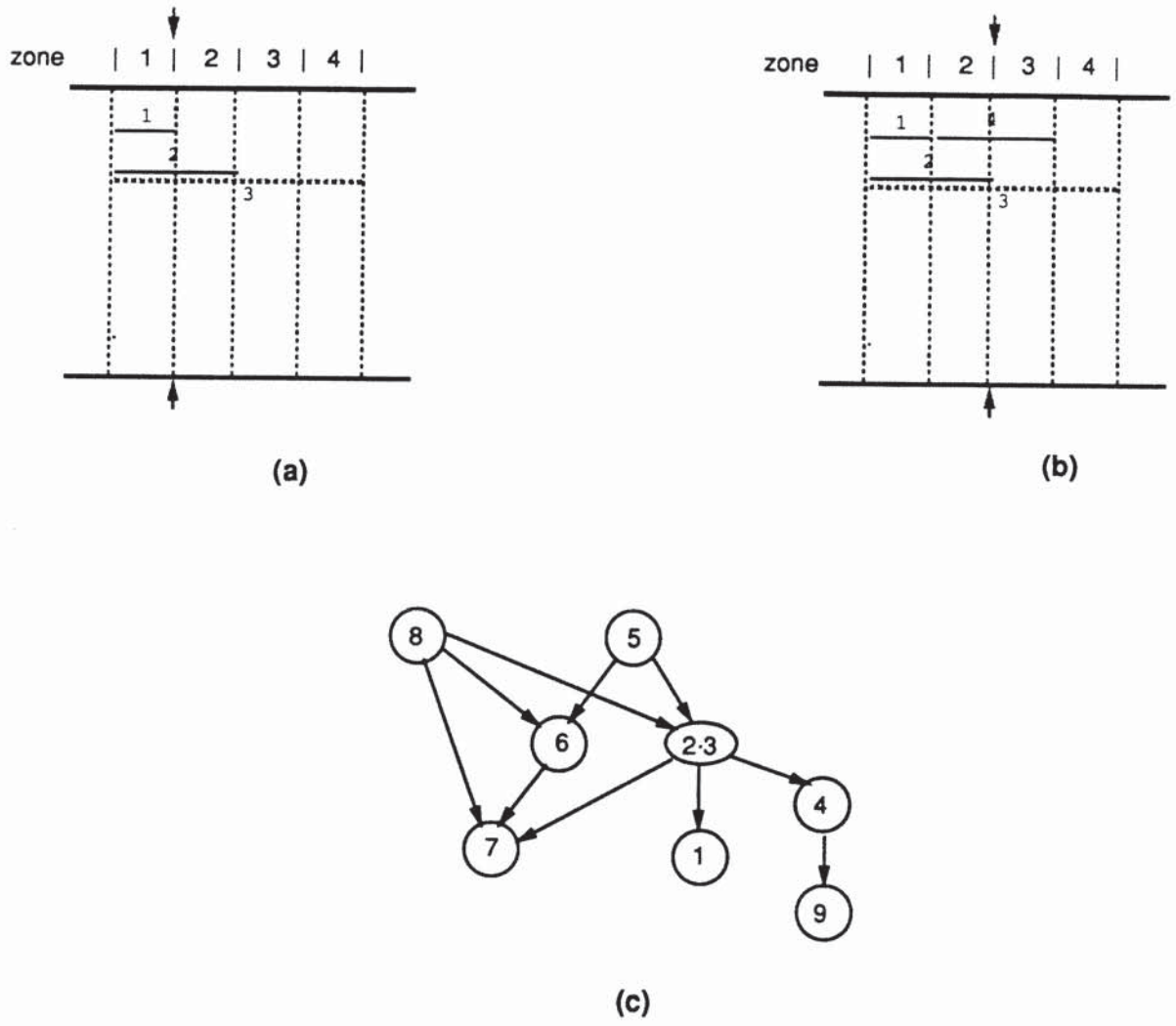
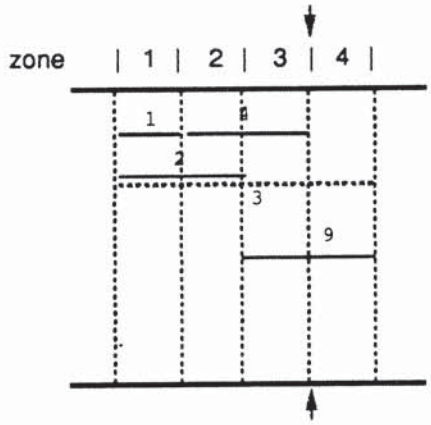
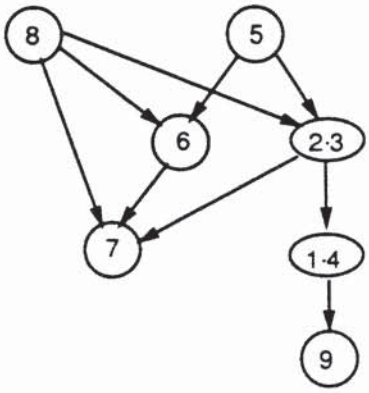


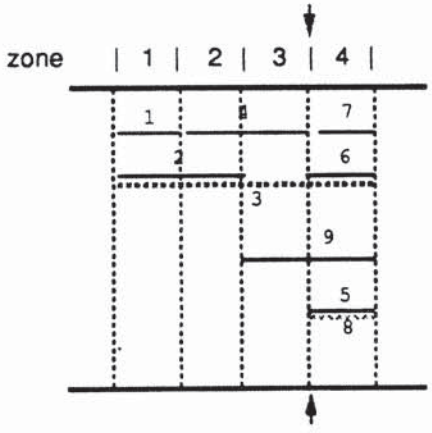
Figure 32: Zone representation after merging nets in (a) zone 1 (b) zone 2 (c) VCG after the second iteration



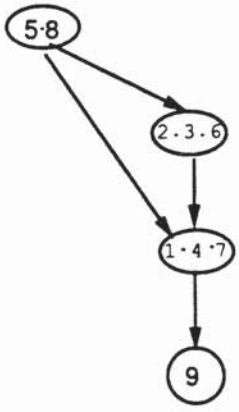
(a)



(b)



(c)



(d)

Figure 33: Zone representation after merging nets in (a) zone 3 (b) zone 4 VCG after the (c)third (d) final iteration

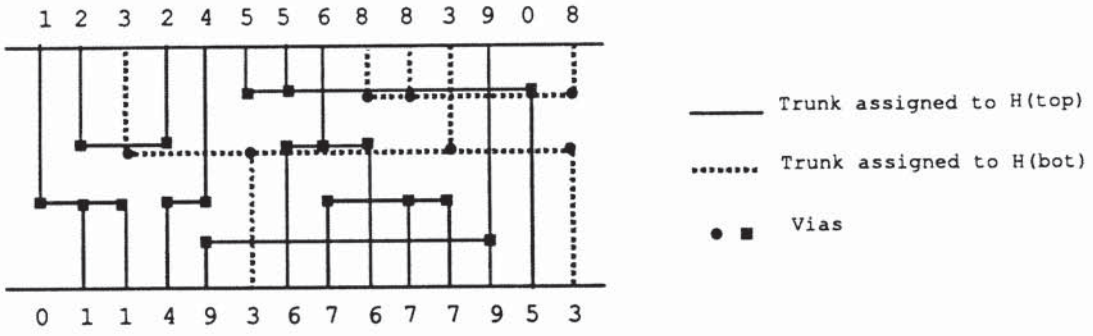


Figure 34: Three-layer HVH solution to P_{ch}

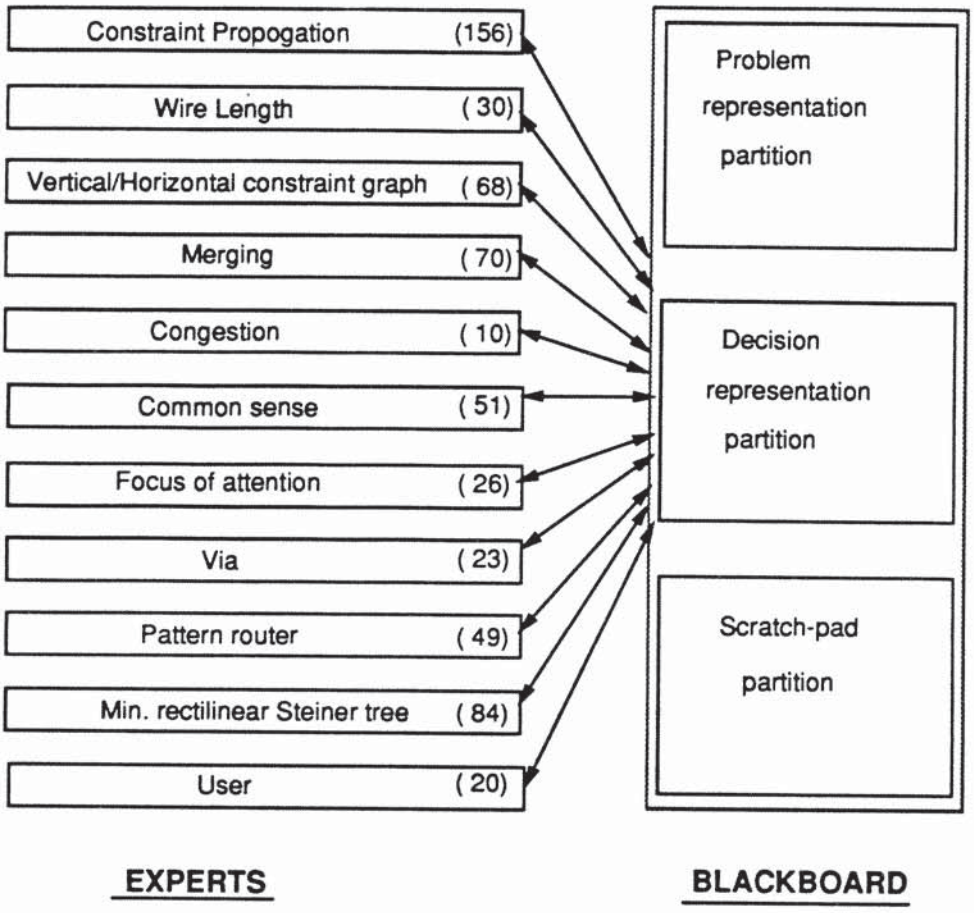


Figure 35: WEAVER architecture

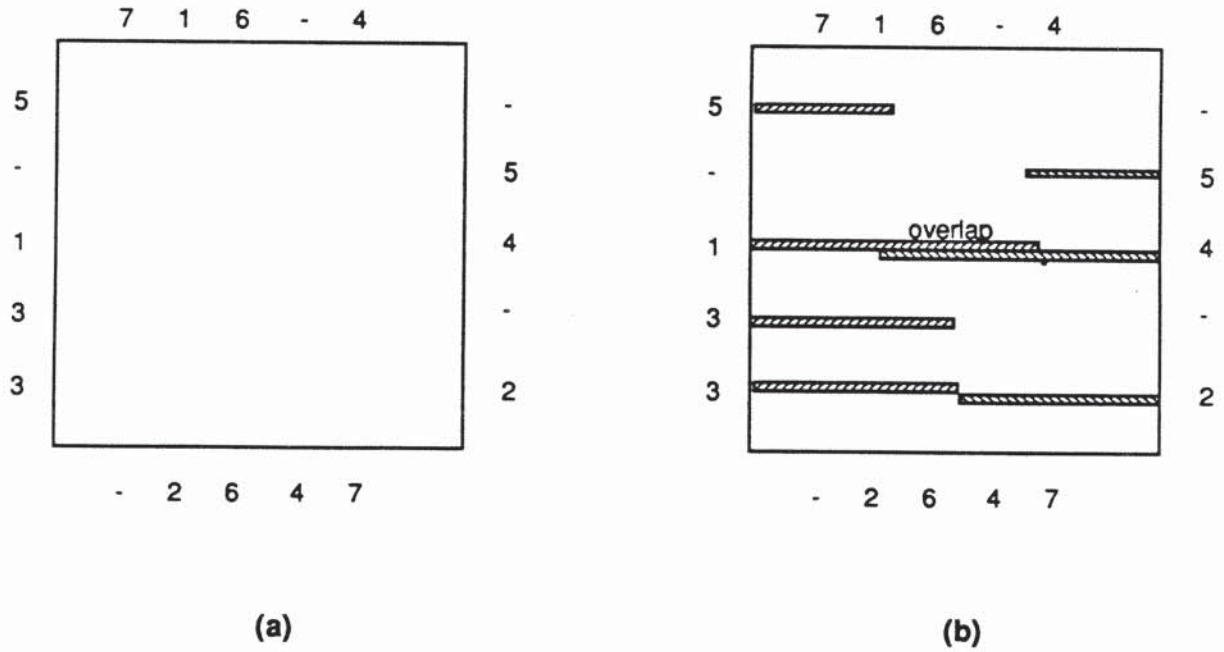


Figure 36: (a) Simple switch box problem (b) Extent of Horizontal control

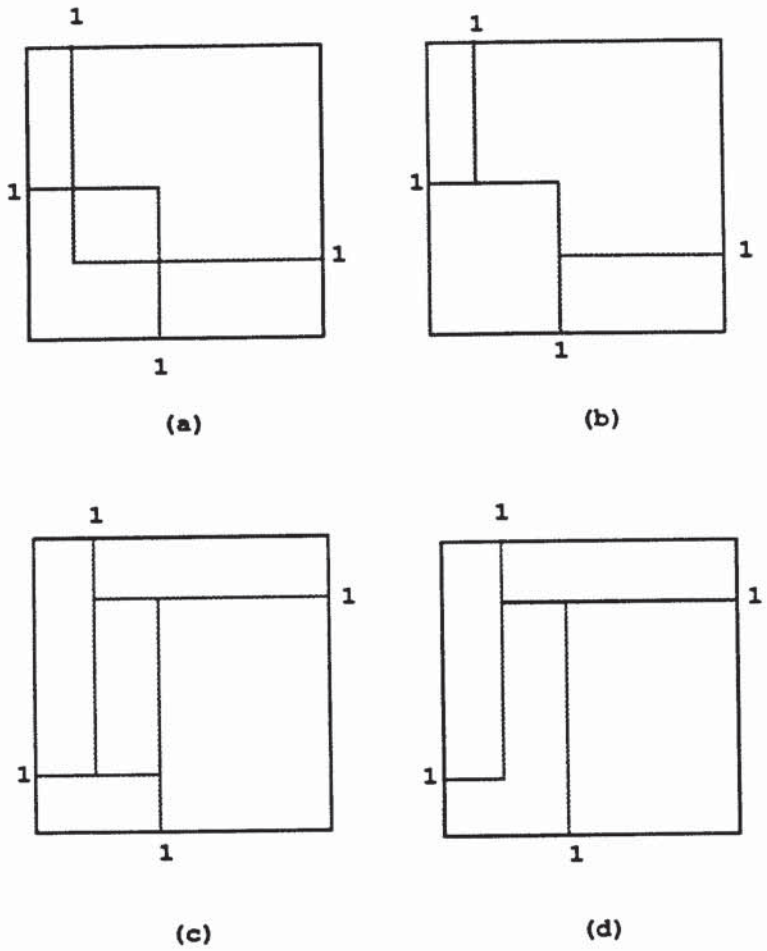


Figure 37: (a)Example of an overlap cycle and (b) its solution (c) Example of a four-terminal cycle and (d) its solution .

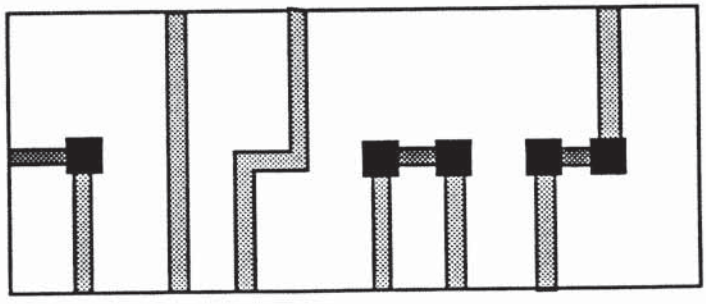


Figure 38: Prototype linesweep connections

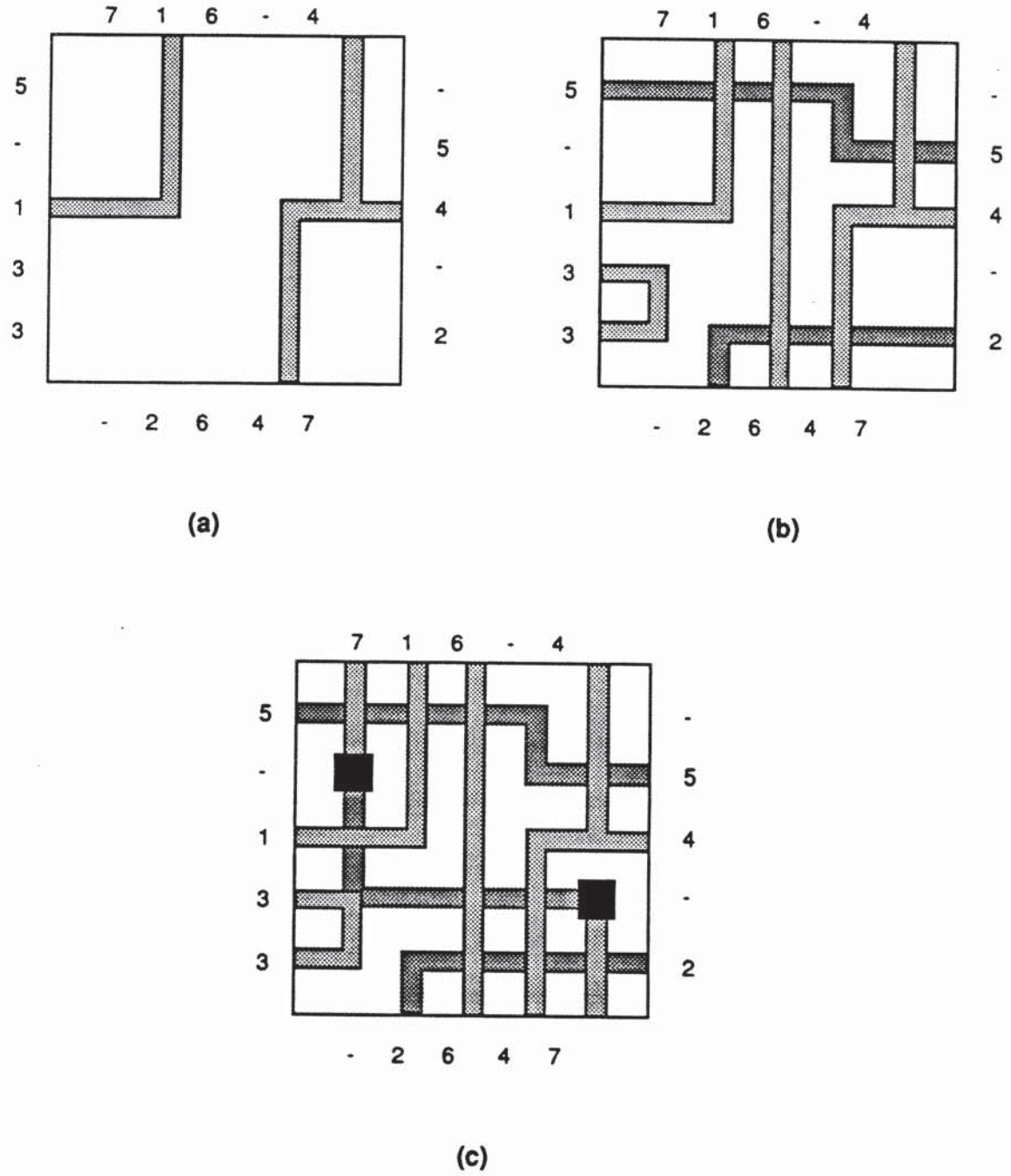


Figure 39: (a)After corner routing (b) After linesweep routing (c) Final solution after thread routing

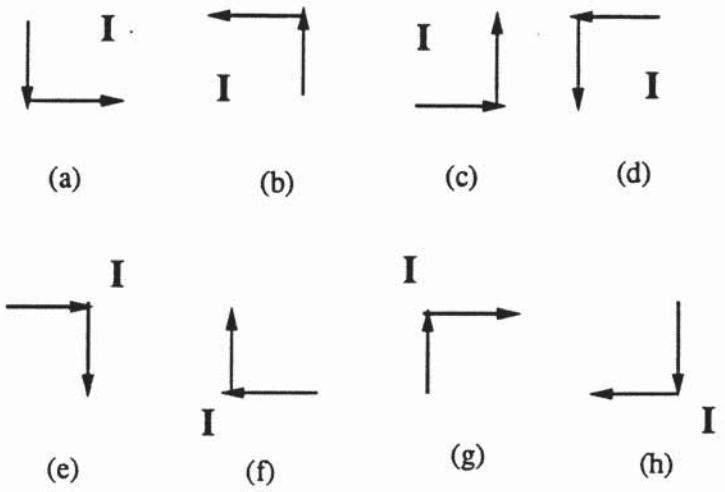


Figure 40: Types of corners in River Routing

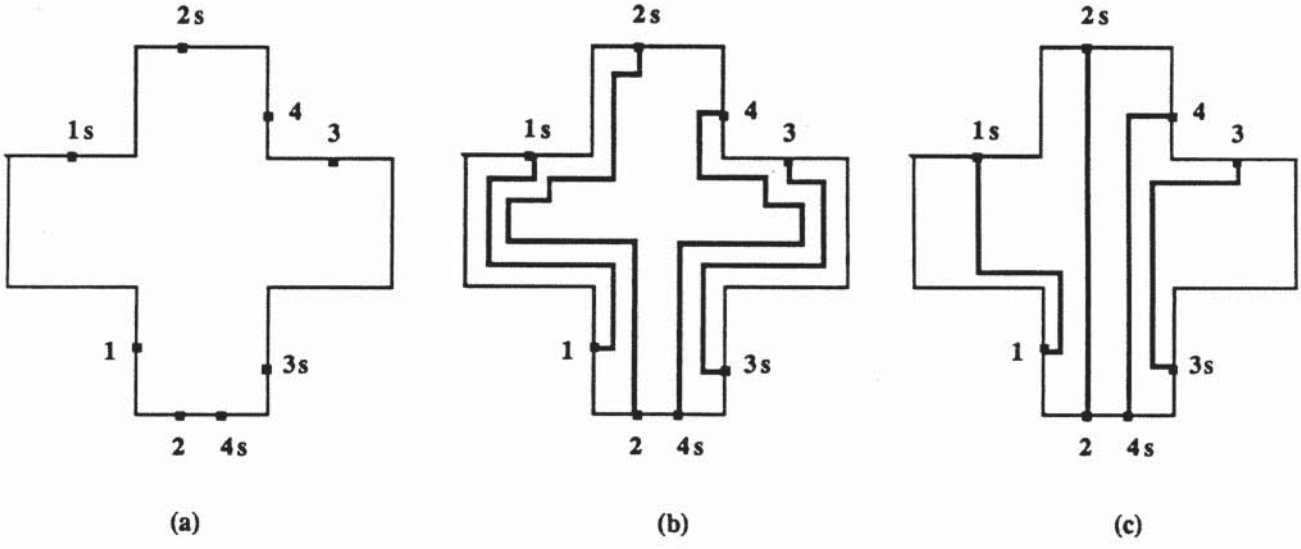


Figure 41: (a)General River Routing Problem (b) After initial routing (c) After corner minimization

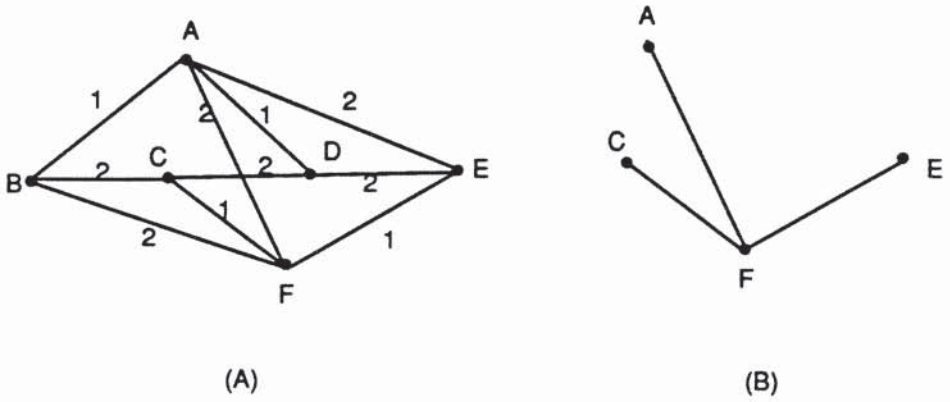


Figure 42: (a) Channel Intersection Graph (b) minimum Steiner tree for a net $N_1 = \{A, C, E\}$

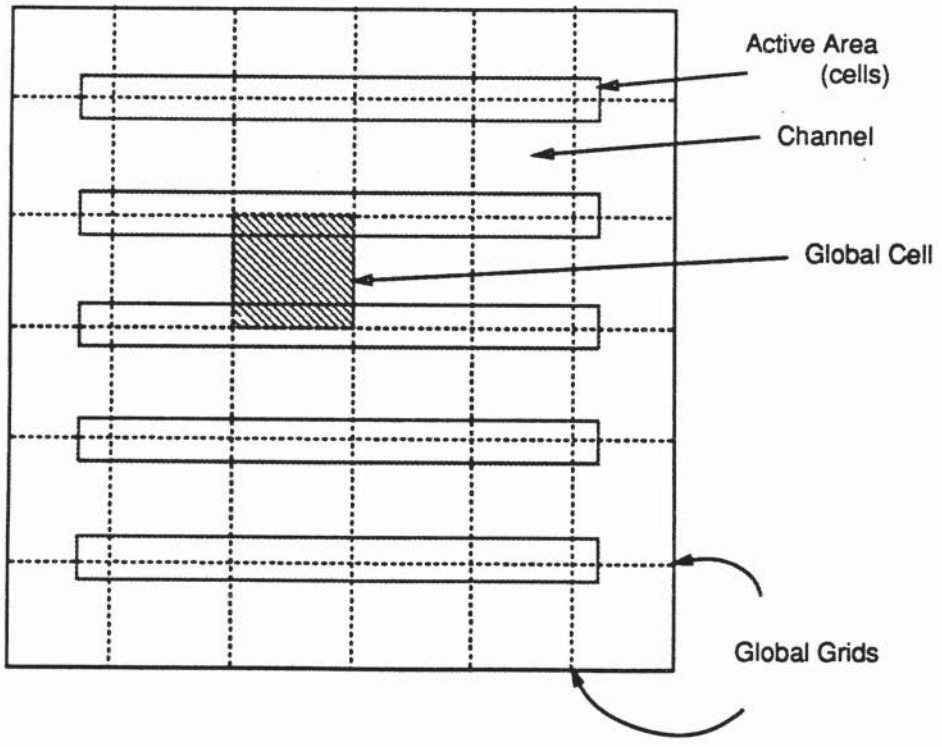


Figure 43: Global routing environment [78]

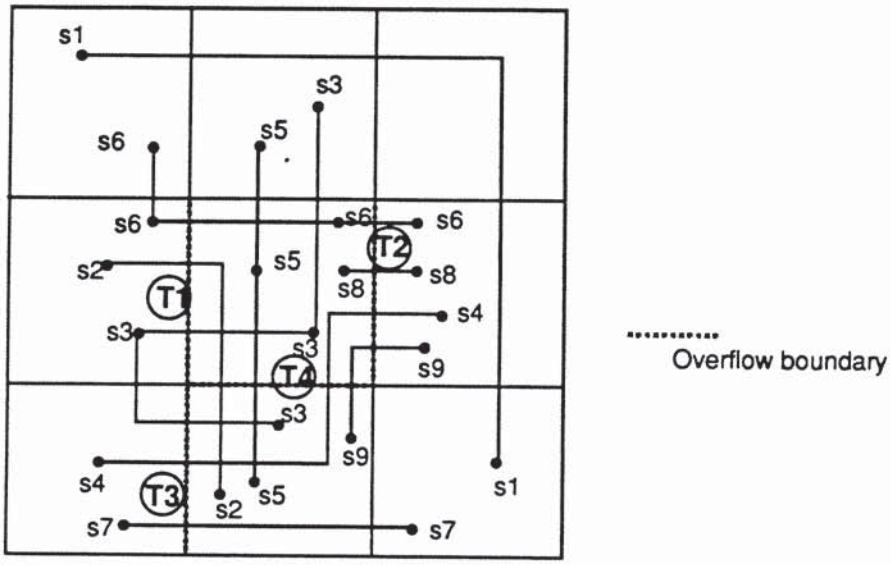


Figure 44: Initial Routing Phase

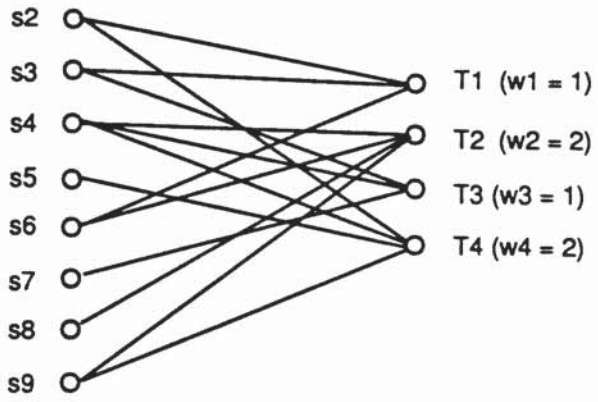


Figure 45: Bipartite graph representation of net and boundary

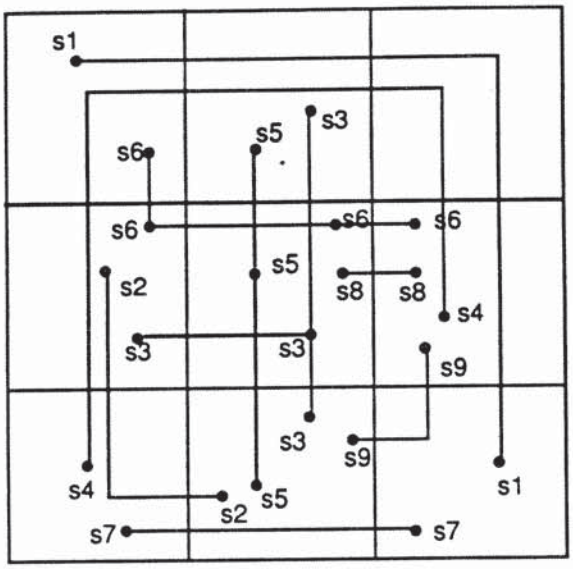


Figure 46: Final Solution

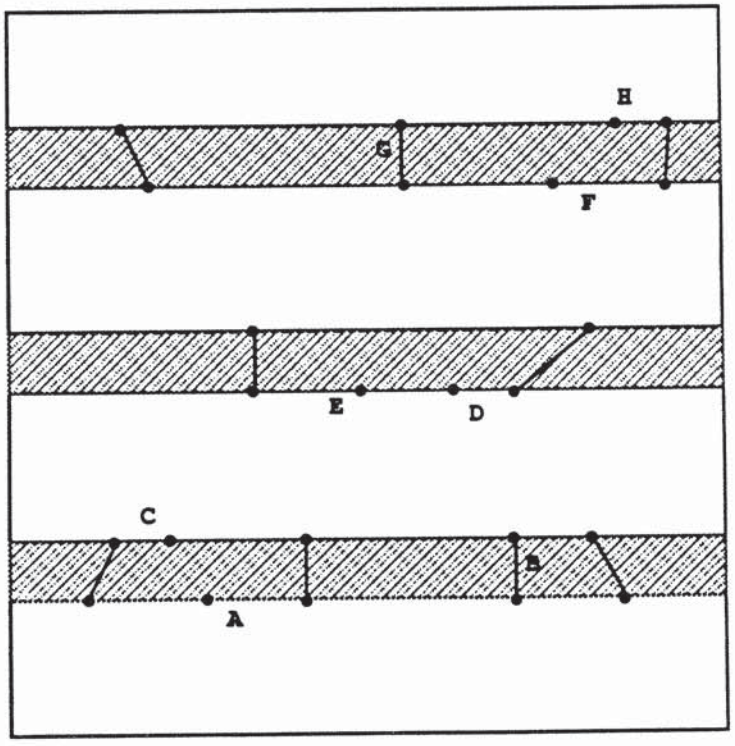


Figure 47: Loose routing problem of a net

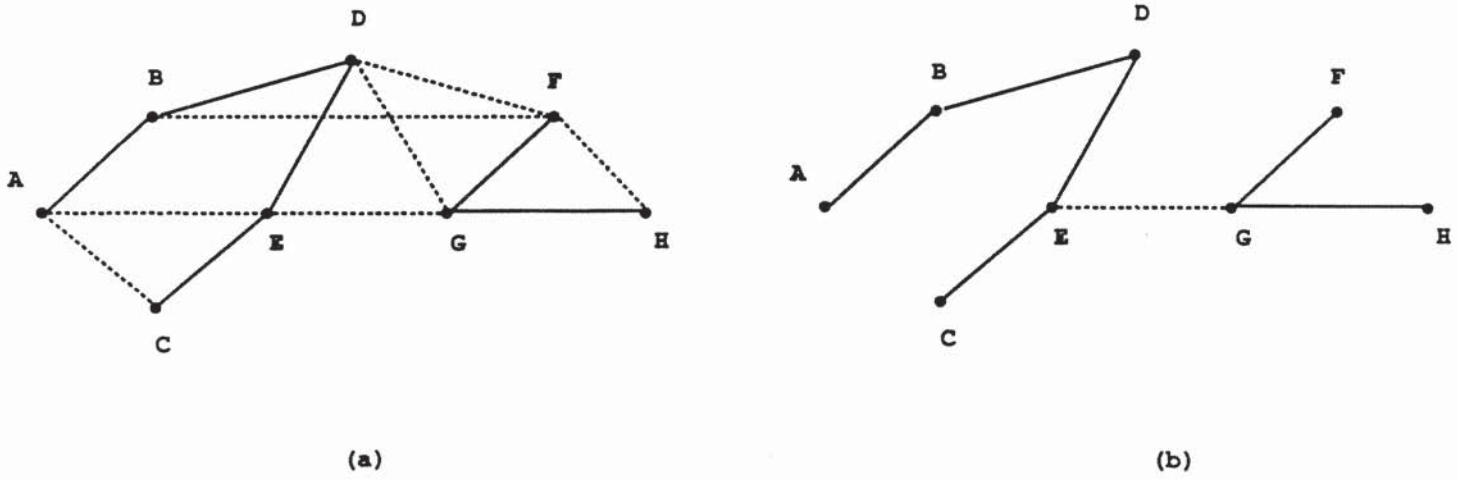


Figure 48: (a)Net graph (b)Minimum spanning Tree

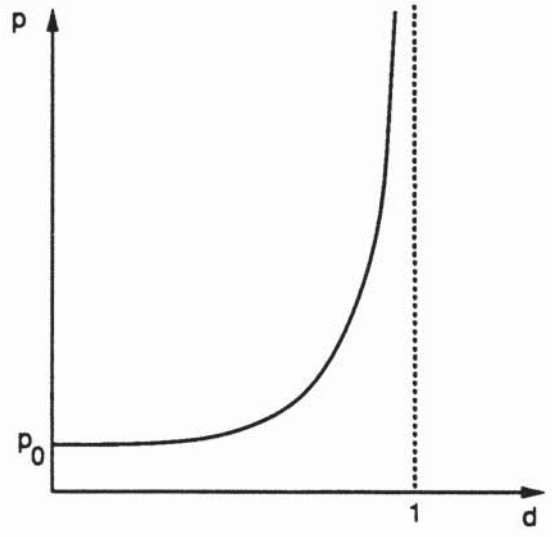


Figure 49: Penalty function for the channel density

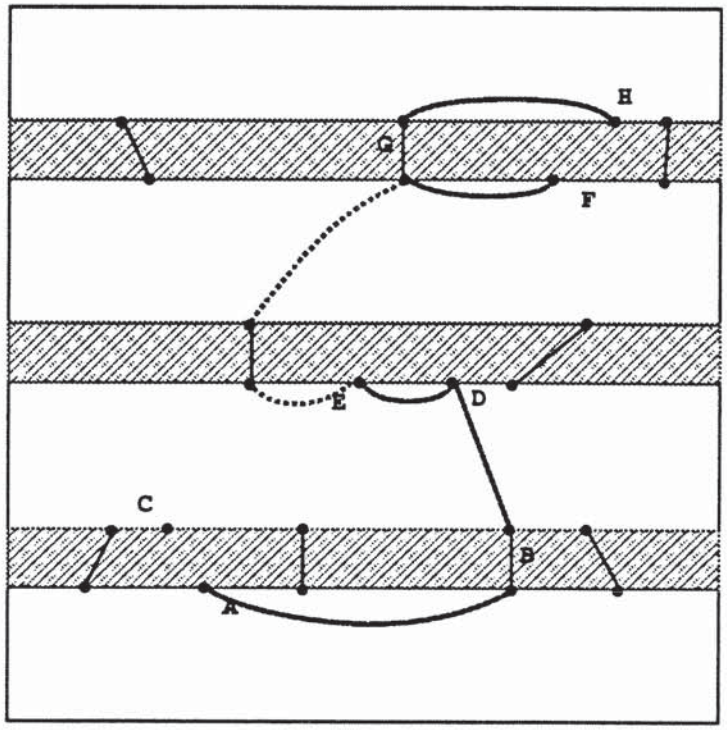
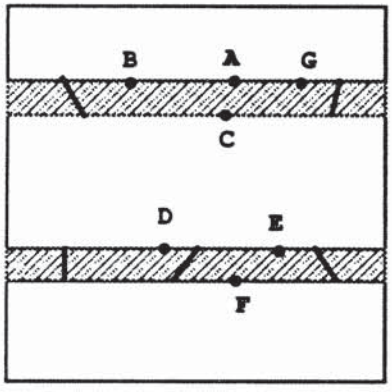


Figure 50: Loose Routing Solution



NET	n(e)	n(T)	n(F)	f
1	1	2	0	0
2	2	5	2	1

NET 1 = {A, B}
NET 2 = {C, D, E, F, G}

Figure 51: Net Ordering by Degree of Freedom

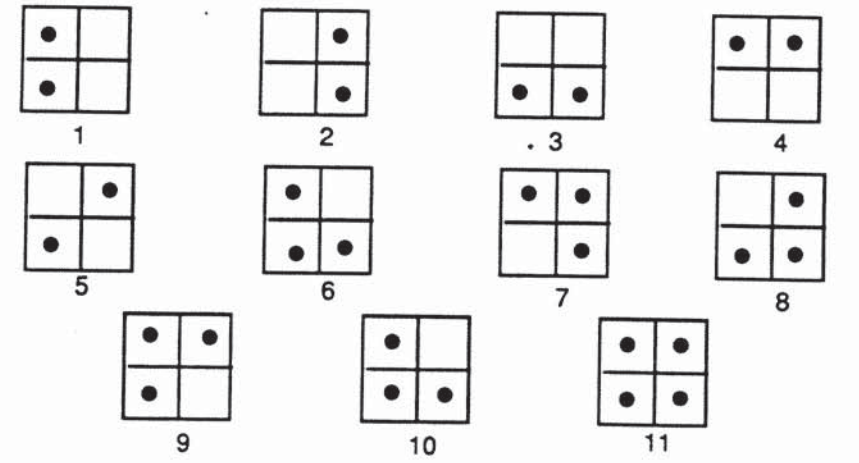


Figure 52: (a) Types of nets (b)4-terminal net wiring for a 2×2 grid

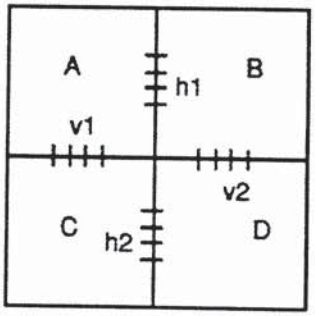


Figure 53: Reduced Two by Two grid wiring problem

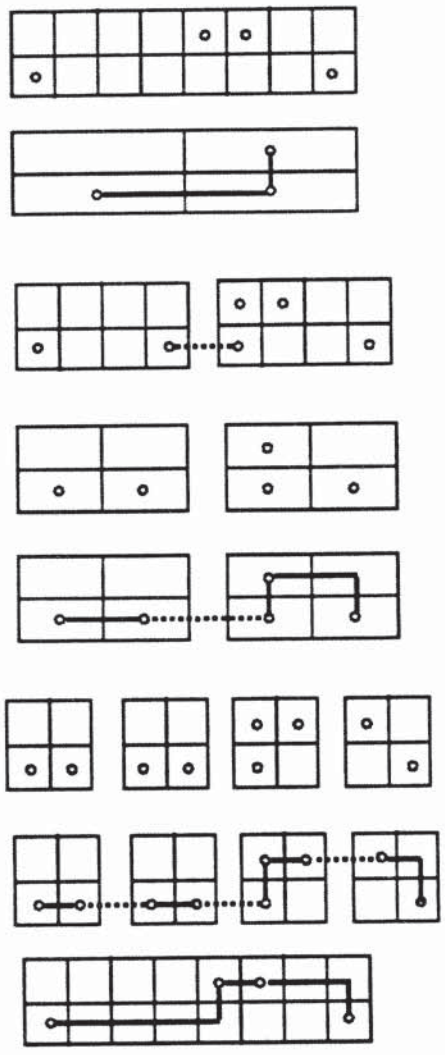


Figure 54: A step by step routing of a 2 × 8 grid using the Hierarchical LP approach

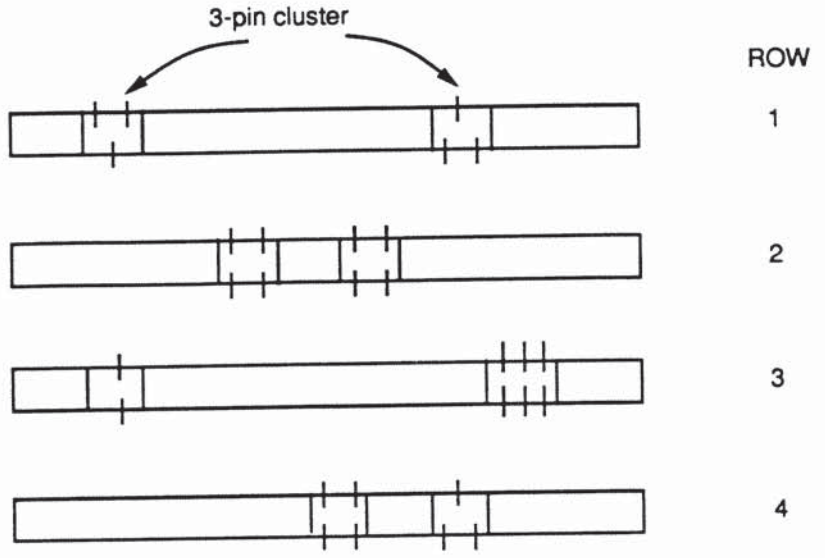


Figure 55: (a) Standard Cell problem showing a net with 8 pin clusters

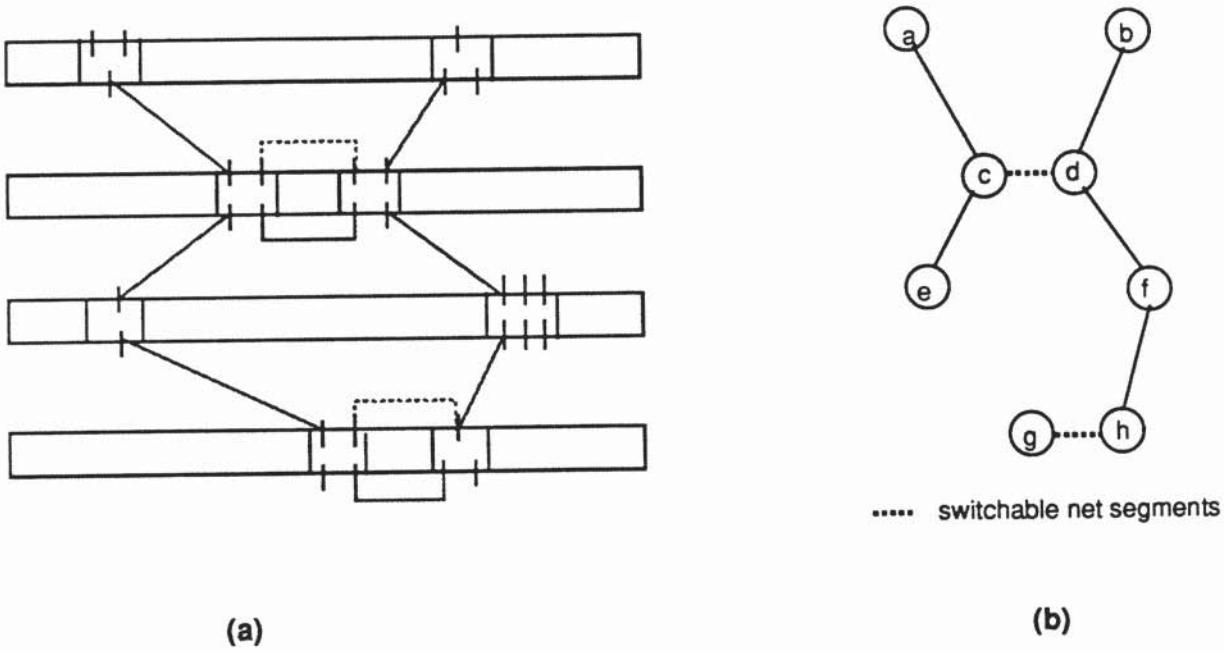


Figure 56: (a) Graph of possible net segments for a single net (b) Corresponding MST

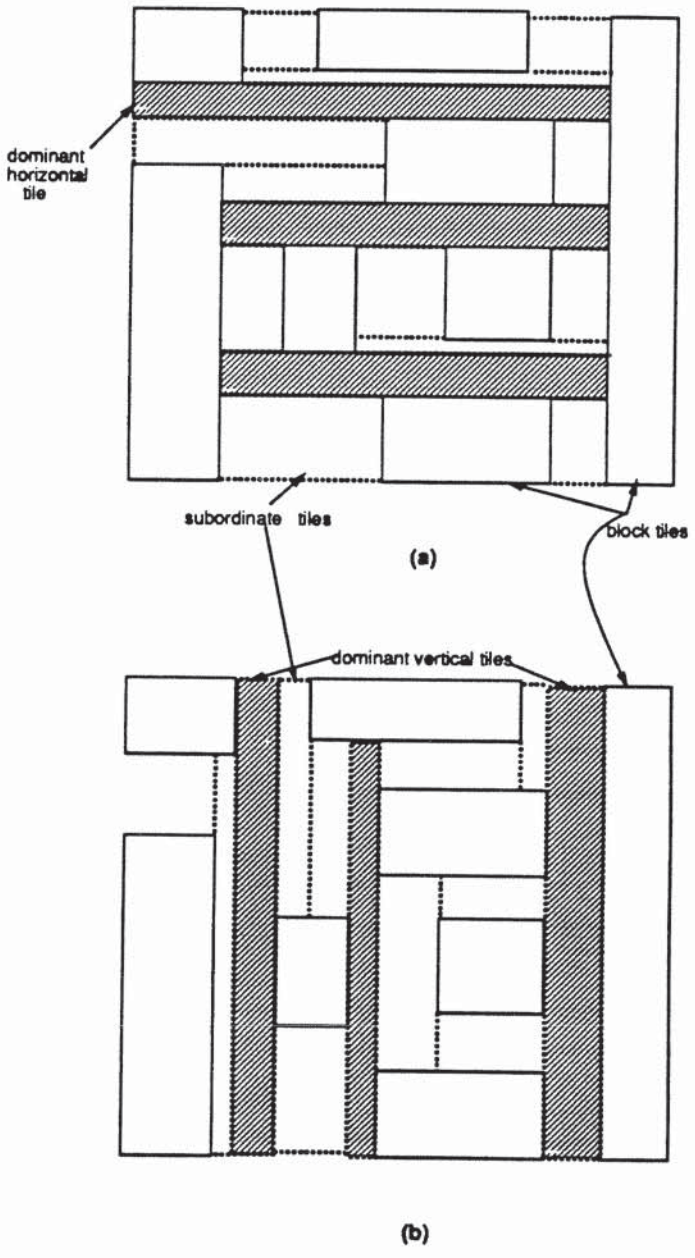


Figure 57: (a) Horizontal tile planes (b) Vertical Tile Planes

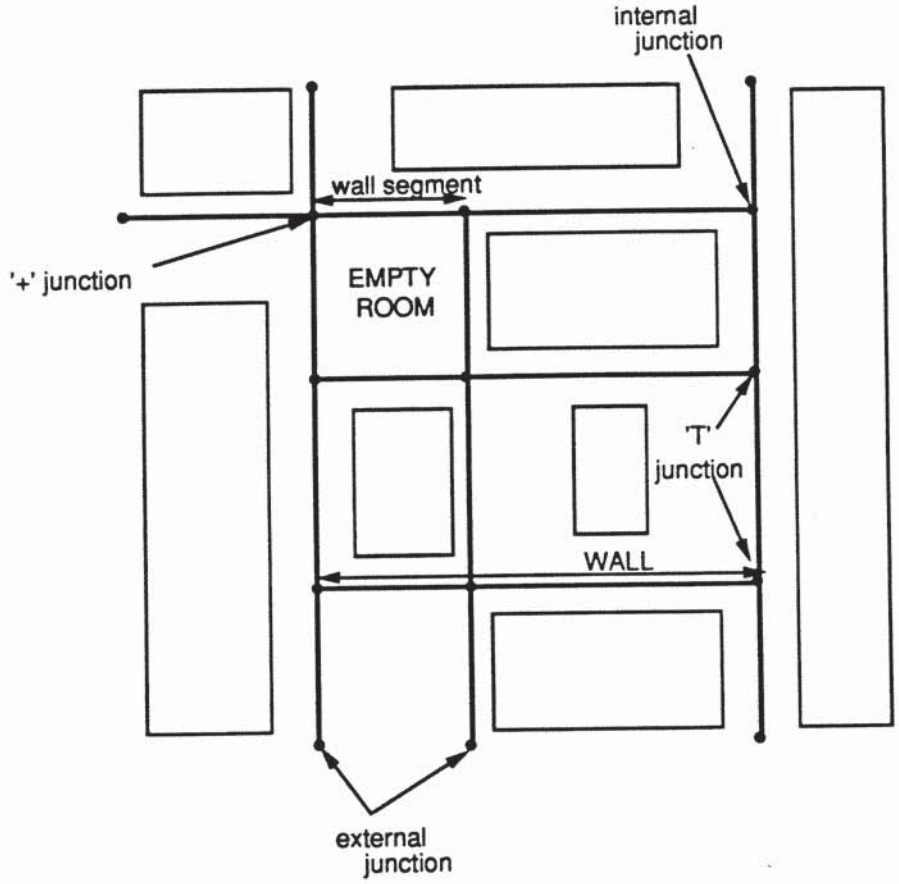


Figure 58: Floor plan graph

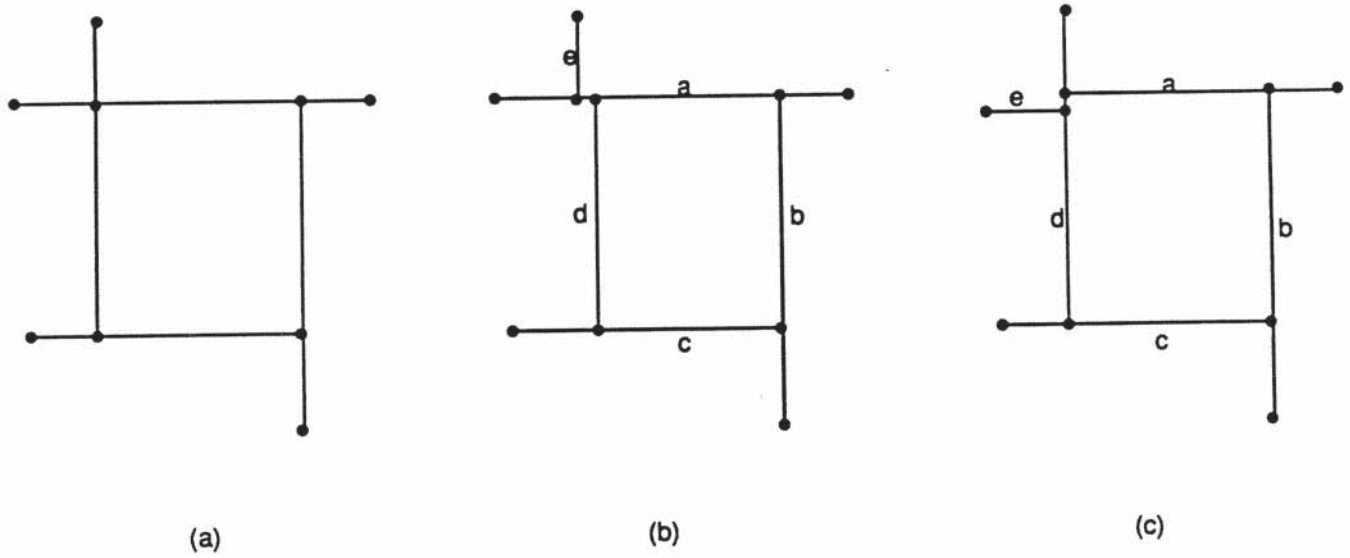
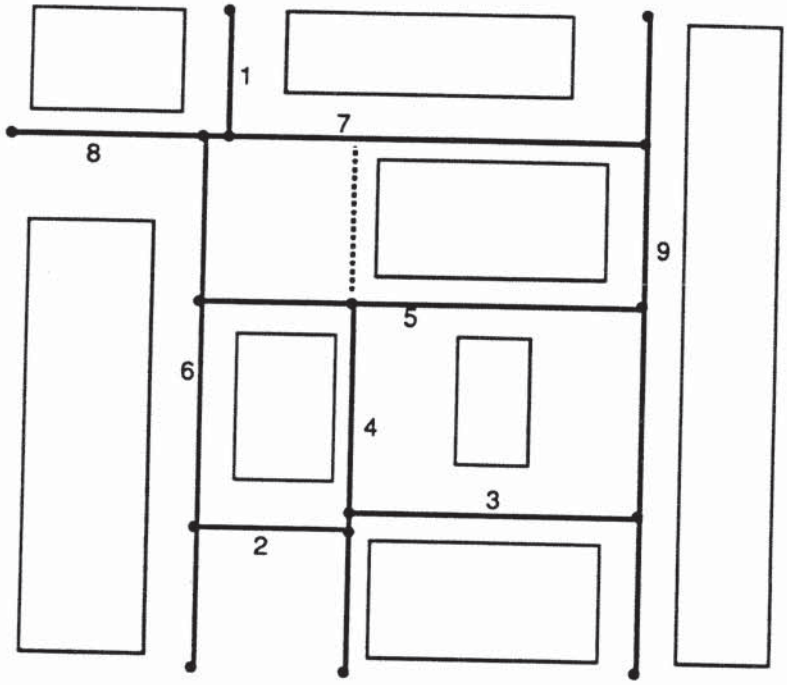
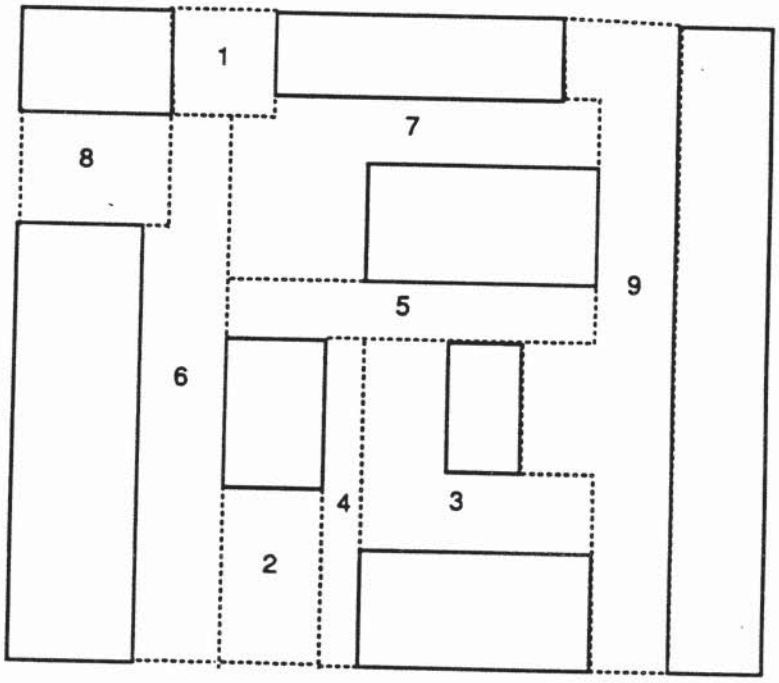


Figure 59: (a)Floorplan Graph, (b) and (c)Two possible + to T conversions



(a)



(b)

Figure 60: Channel definition and ordering

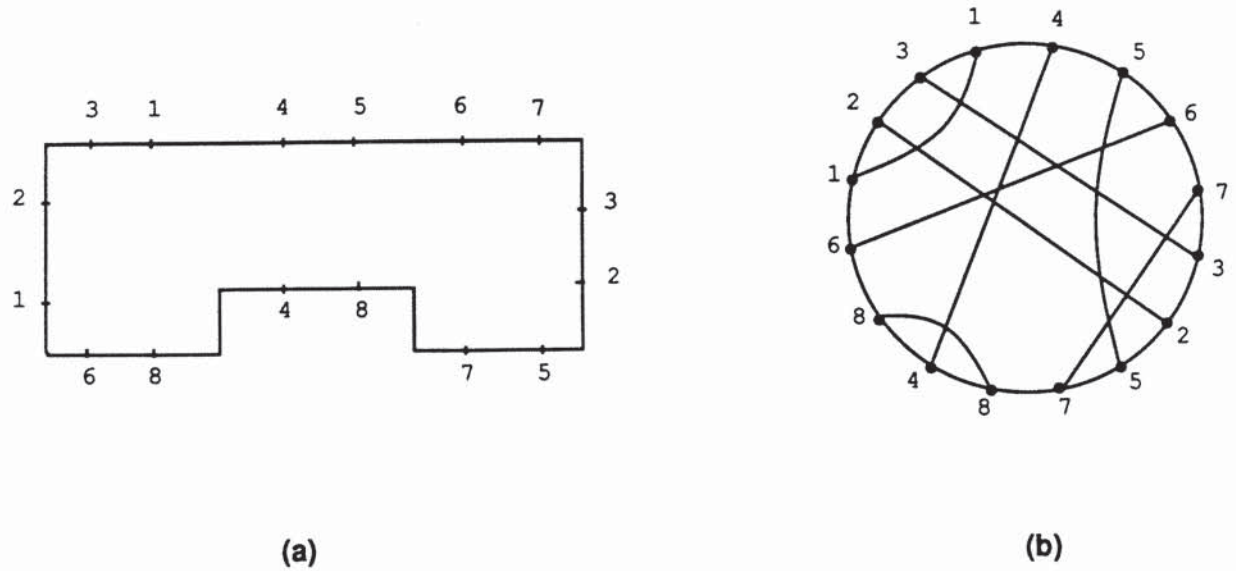


Figure 61: (a) Example UVM* routing problem (b) Interconnection graph G

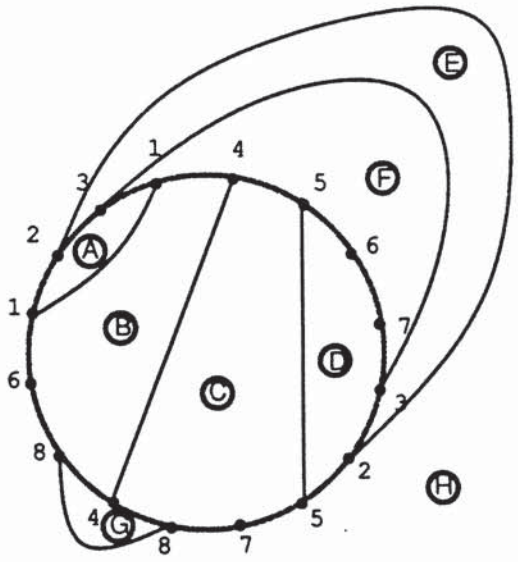


Figure 62: G_p, planar graph obtained by removing $\{N_6, N_7\}$

[b]

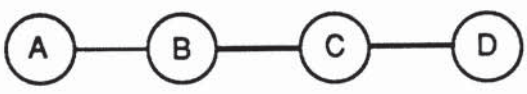
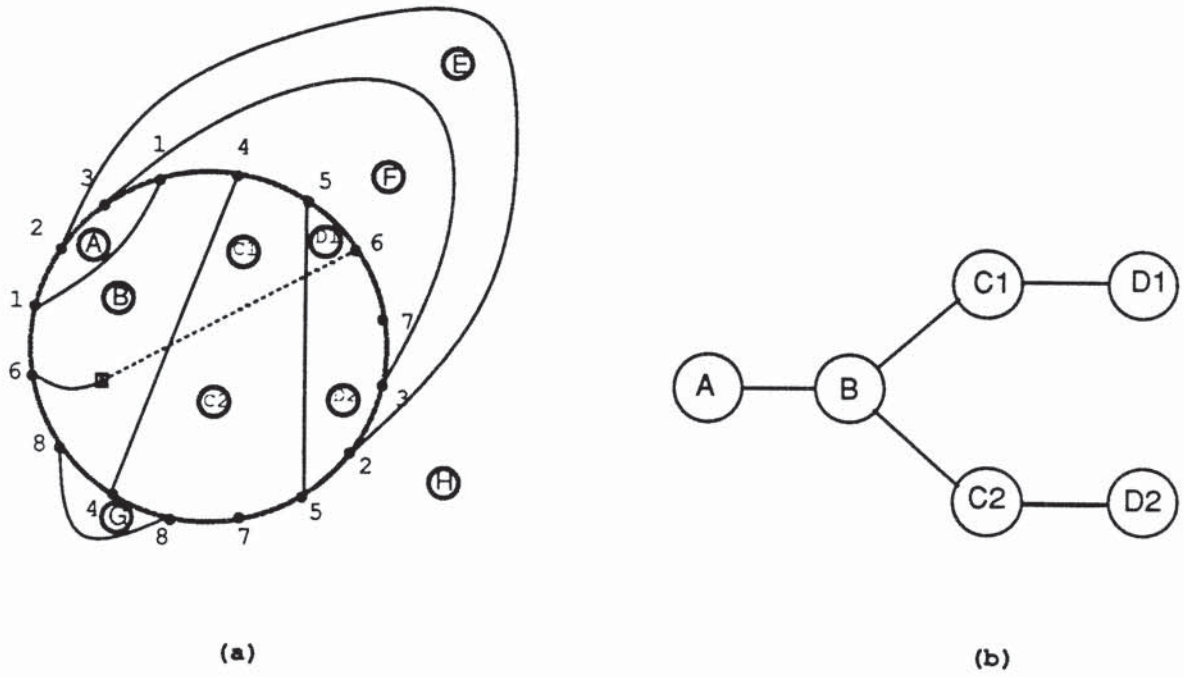


Figure 63: The region connectivity graph G_I



(a) (b)

Figure 64: (a) G_p after N_6 is routed (b) The modified G_I after N_6 is routed

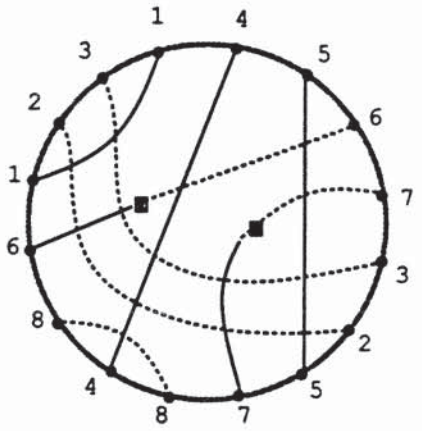
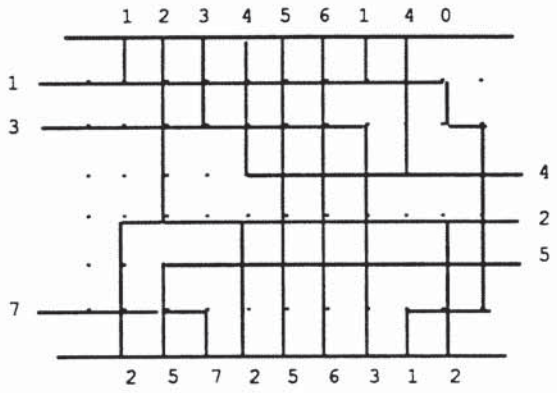
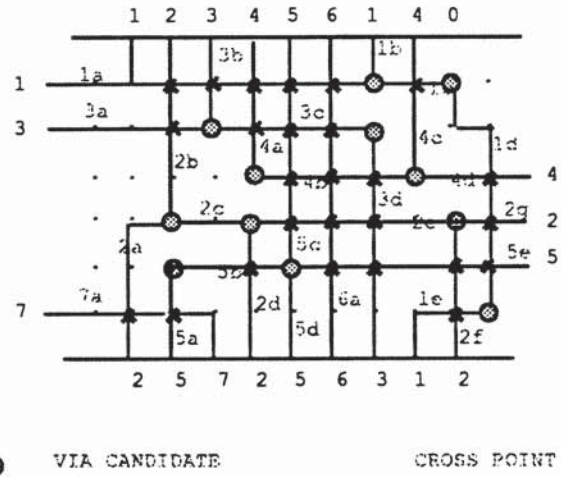


Figure 65: The final topological routing



(a)



(b)

Figure 66: (a) Planar representation of a 2-layer H-V routing solution (b) Cross points and via candidates

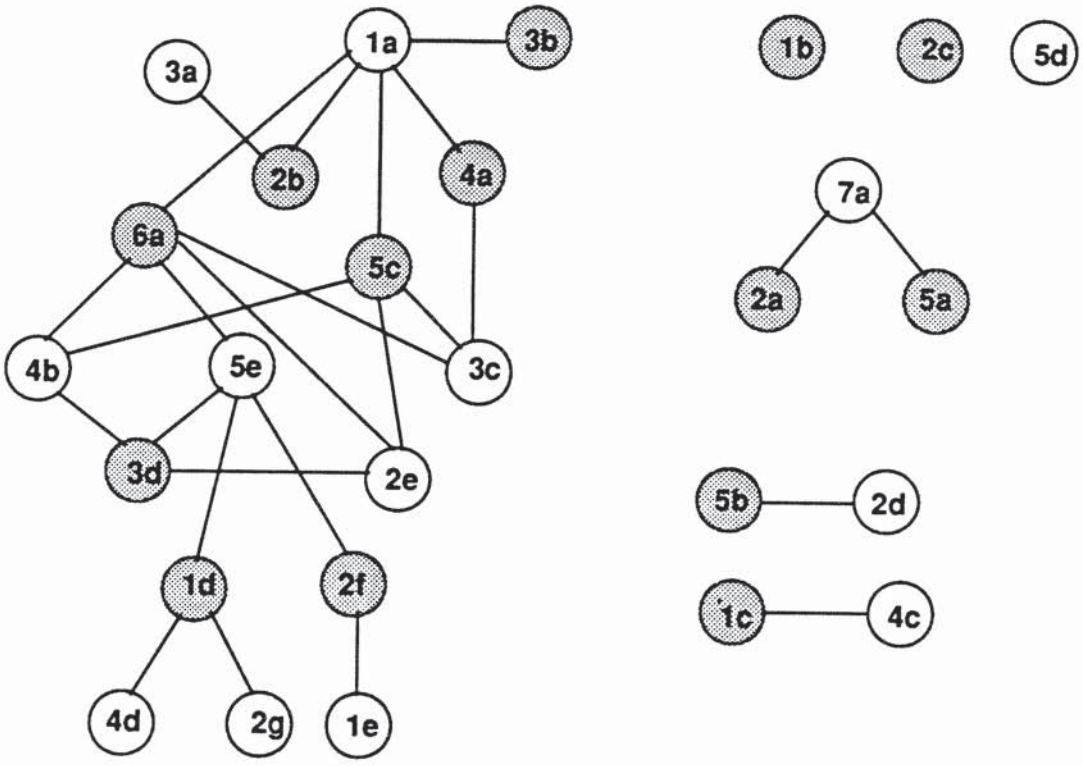


Figure 67: Constraint graph

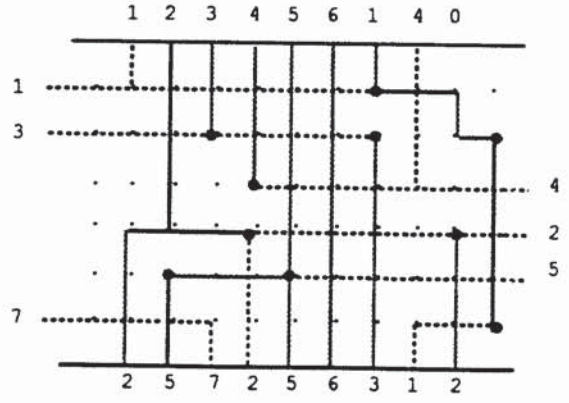


Figure 68: Final solution after via minimization

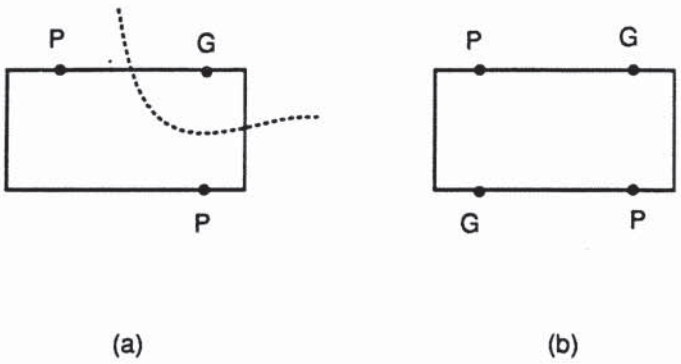


Figure 69: (a) Block for which a cut exists that separates power and ground pins (b)Block for which no such cut exists

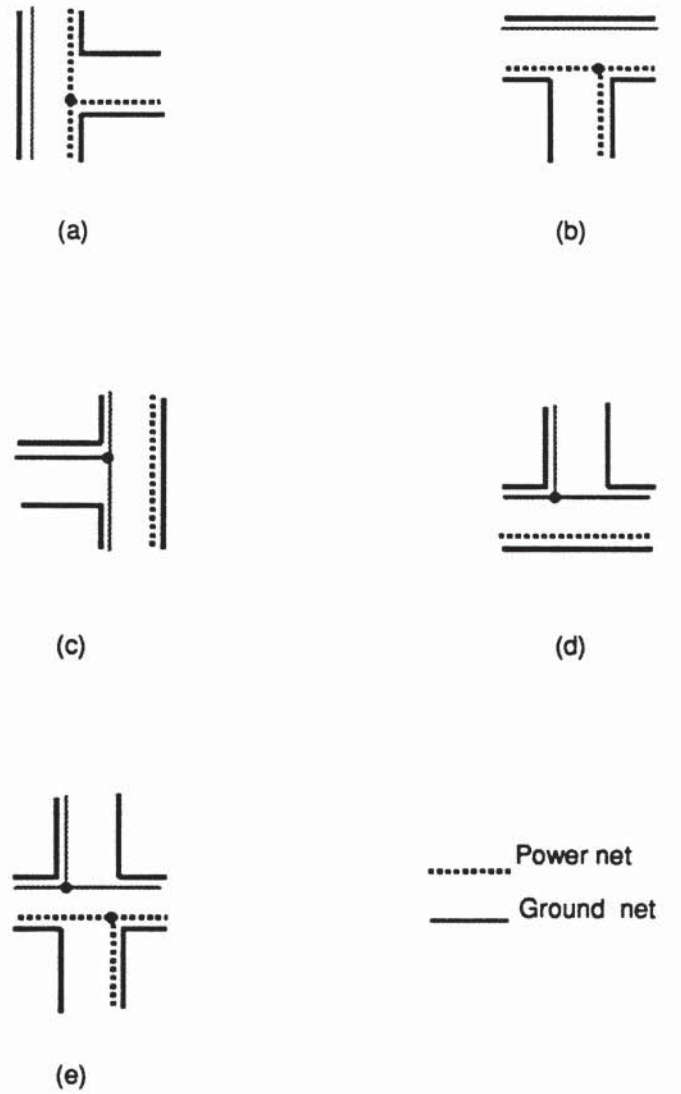
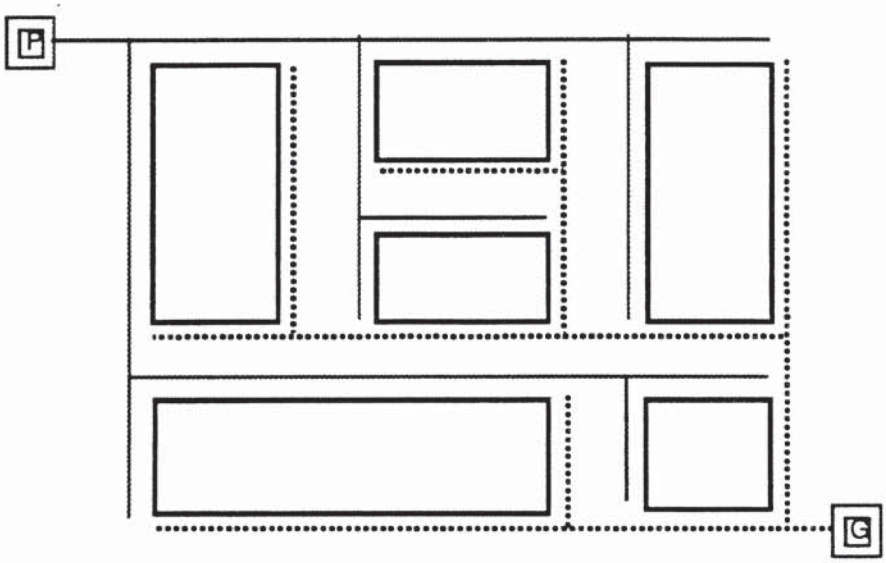
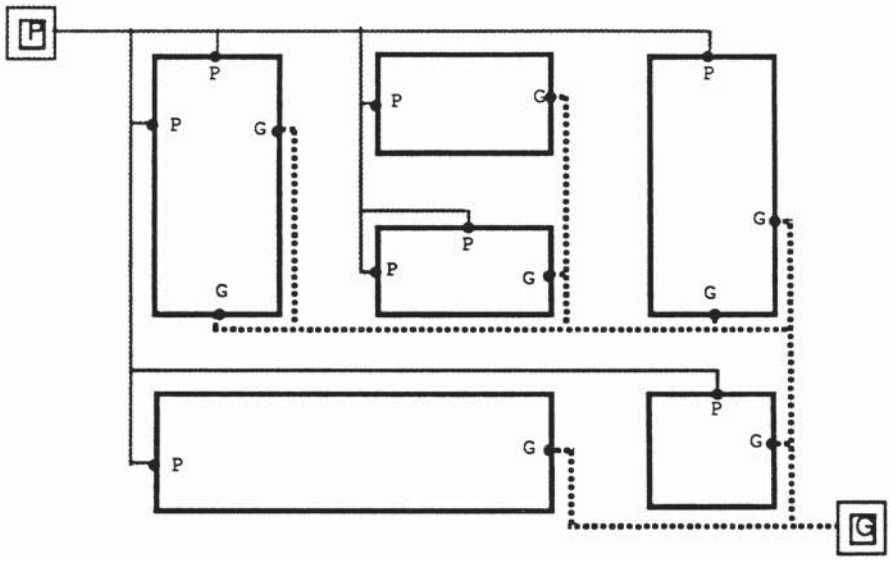


Figure 70: (a)–(d) Rules for T intersections (e) Rule for + intersection

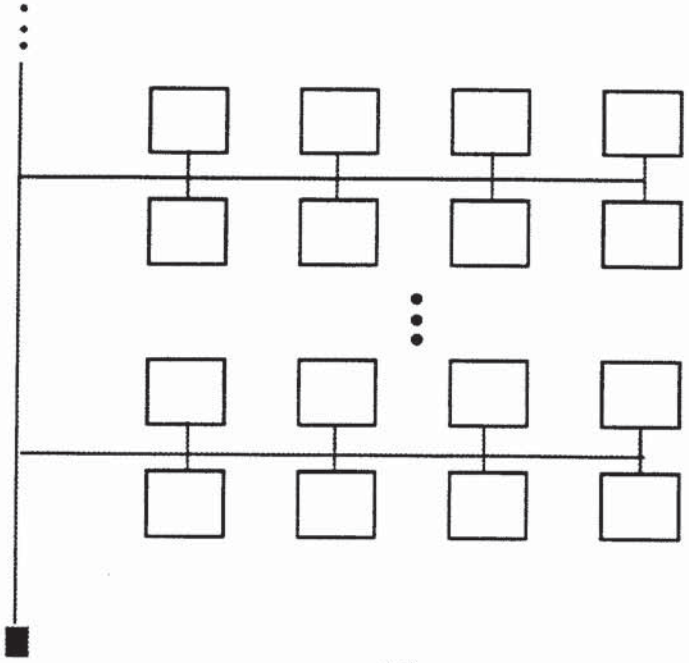


(a)

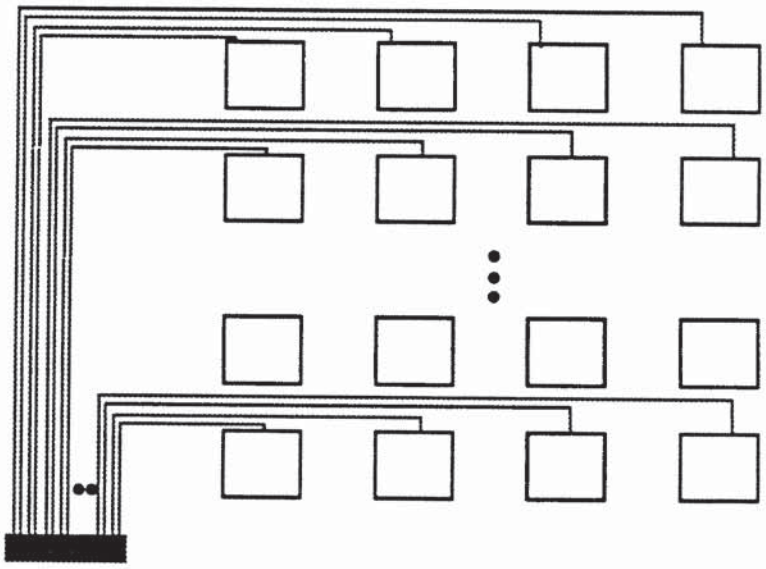


(b)

Figure 71: (a) Traffic Rules illustrated (b) A sample p/g routing example



(a)



(b)

Figure 72: (a) Normal scheme (b) Star scheme of routing regular cells [105]

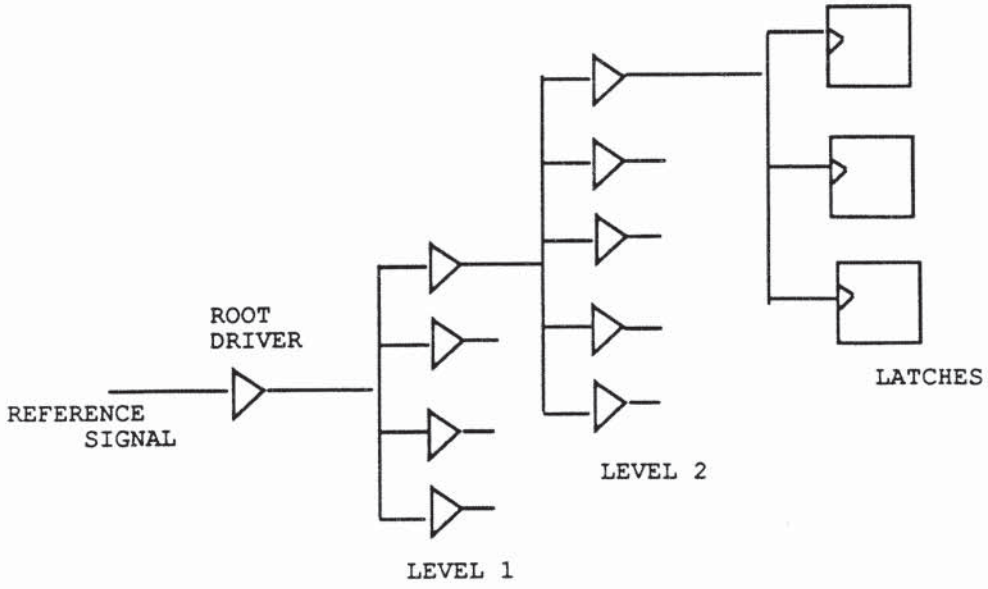


Figure 73: A 2-level Clock Tree

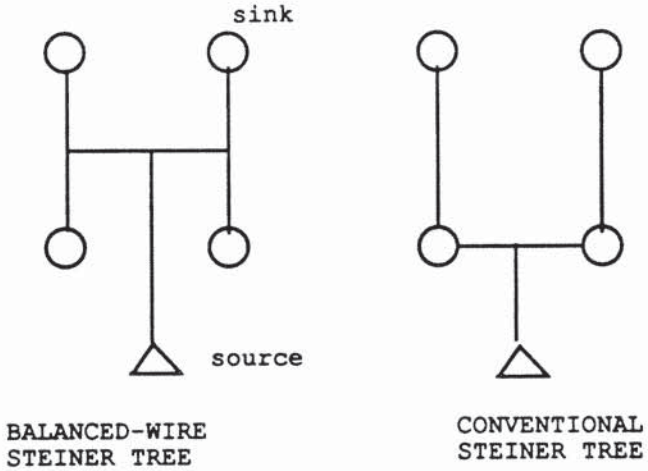


Figure 74: Balanced Wire Steiner Trees

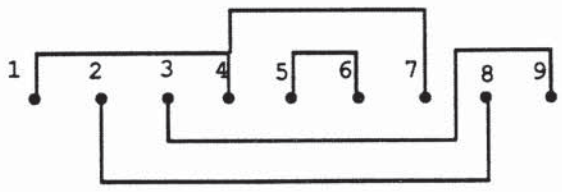


Figure 75: Realization of a 4-street single row routing problem

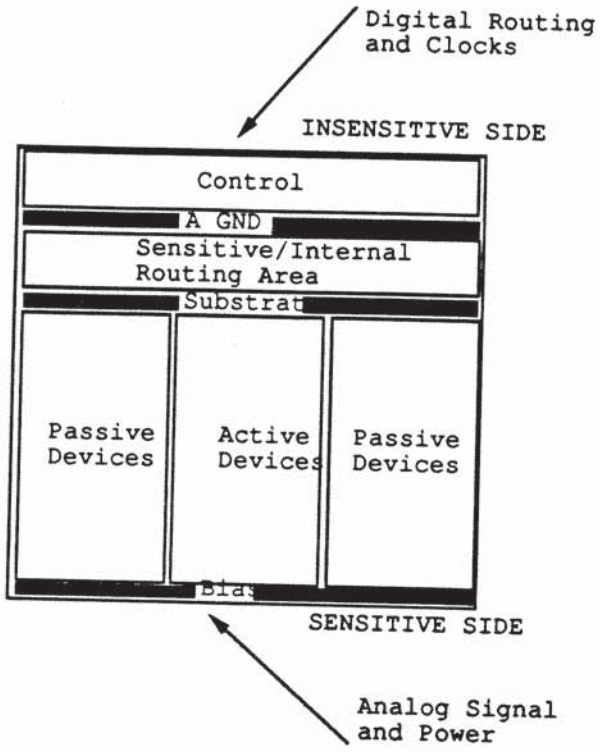


Figure 76: Analog Leaf-cell Floorplan Template

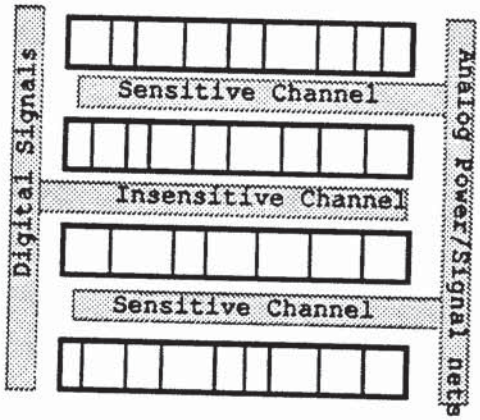
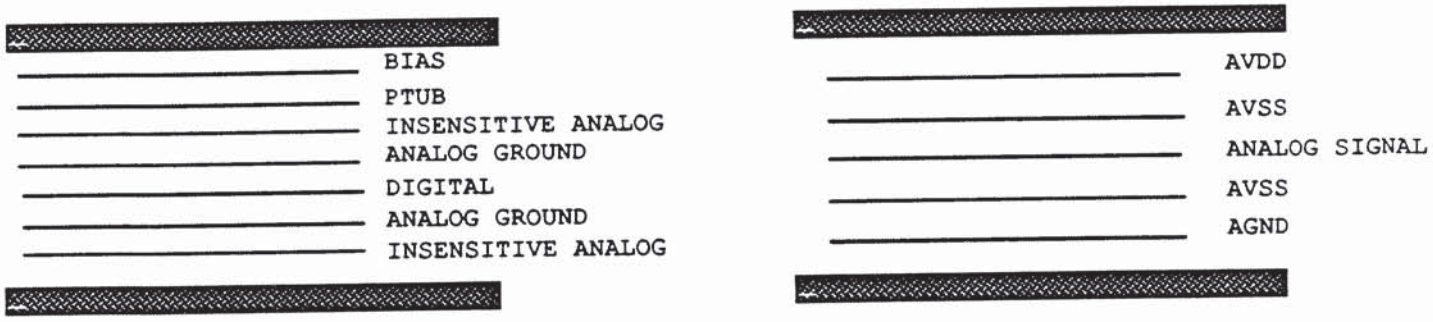


Figure 77: Overall Analog Standard-cell Floorplan and Routeplan



(a) Insensitive Channel

(b) Sensitive Channel

Figure 78: Analog Routeplan in (a) Sensitive Channels (b) Insensitive Channels