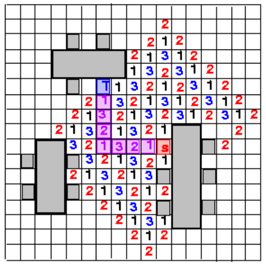
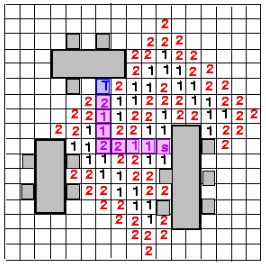
Reducing Memory Requirement

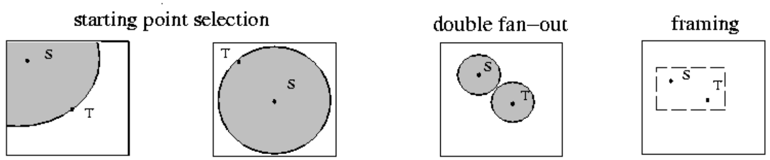
- Akers's Observations (1967)
  - Adjacent labels for *k* are either *k*-1 or *k*+1.
  - Want a labeling scheme such that each label has its preceding label different from its succeeding label.
- Way 1: coding sequence 1, 2, 3, 1, 2, 3, …; states: 1, 2, 3, *empty*, *blocked* (3 bits required)
- Way 2: coding sequence 1, 1, 2, 2, 1, 1, 2, 2, …; states: 1, 2, *empty*, *blocked* (need only 2 bits)


Sequence: 1, 2, 3, 1, 2, 3, ...


Sequence: 1, 1, 2, 2, 1, 1, 2, 2, ...

Reducing Running Time

- Starting point selection: Choose the point farthest from the center of the grid as the starting point.
- Double fan-out: Propagate waves from both the source and the target cells.
- Framing: Search inside a rectangle area 10--20% larger than the bounding box containing the source and target.
  - Need to enlarge the rectangle and redo if the search fails.

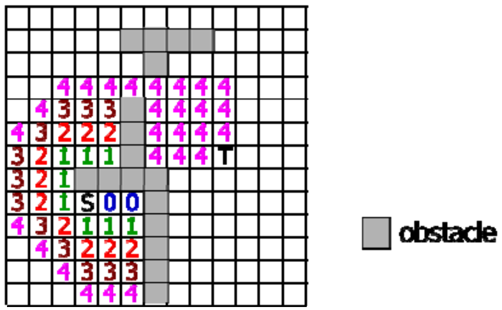


Hadlock's Algorithm

- Hadlock, "A shortest path algorithm for grid graphs," *Networks*, 1977.
- Uses detour number (instead of labeling wavefront in Lee's router)
  - Detour number, *d*(*P*): # of grid cells directed **away from** its target on path *P*.
  - *MD*(*S, T*): the Manhattan distance between *S* and *T*.
  - Path length of *P*, *l*(*P*): *l*(*P*) = *MD*(*S, T*) + 2 *d*(*P*).
  - *MD*(*S, T*) fixed! ⇒ Minimize *d*(*P*) to find the shortest path.
  - For any cell labeled *i*, label its adjacent unblocked cells **away from** *T* *i*+1; label *i* otherwise.
- Time and space complexities: *O*(*MN*), but substantially reduces the # of searched cells.
- Finds the shortest path between *S* and *T*.

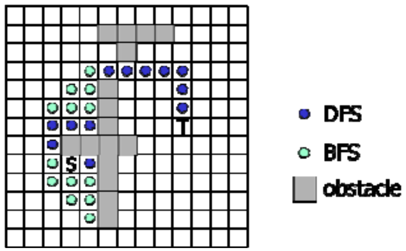
Hadlock's Algorithm (cont'd)

- *d*(*P*): # of grid cells directed **away from** its target on path *P*.
- *MD*(*S*, *T*): the Manhattan distance between *S* and *T*.
- Path length of *P*, *l*(*P*): *l*(*P*) = *MD*(*S*, *T*) + 2*d*(*P*).
- *MD*(*S*, *T*) fixed! ⇒ Minimize *d*(*P*) to find the shortest path.
- For any cell labeled *i*, label its adjacent unblocked cells **away from** *T* *i*+1; label *i* otherwise.



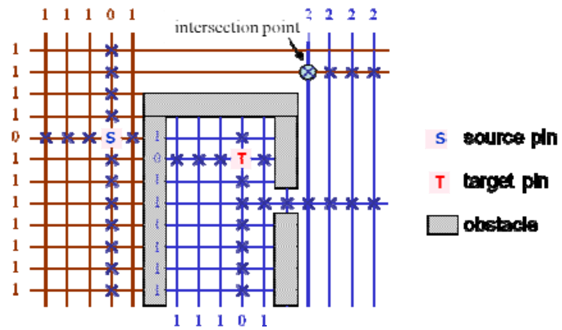
Soukup's Algorithm

- Soukup, “Fast maze router,” DAC-78.
- Combined breadth-first and depth-first search.
  - Depth-first (**line**) search is first directed toward target *T* until an obstacle or *T* is reached.
  - Breadth-first (Lee-type) search is used to “bubble” around an obstacle if an obstacle is reached.
- Time and space complexities: *O*(*MN*), but 10~50 times faster than Lee's algorithm.
- Find **a** path between *S* and *T*, but may not be the shortest!



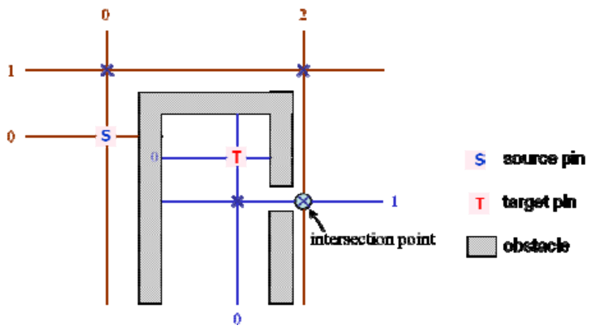
Mikami-Tabuchi's Algorithm

- Mikami & Tabuchi, “A computer program for optimal routing of printed circuit connectors,” *IFIP*, H47, 1968.
- Every grid point is an escape point.



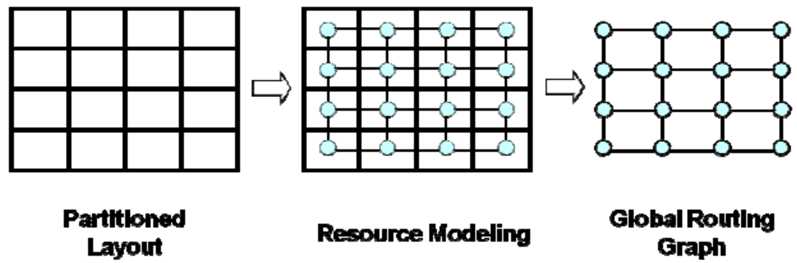
Hightower's Algorithm

- Hightower, “A solution to line-routing problem on the continuous plane,” DAC-69.
- A single escape point on each line segment.
- If a line parallels to the blocked cells, the escape point is placed just past the endpoint of the segment.



Global Routing Graph

- Each cell is represented by a vertex.
- Two vertices are joined by an edge if the corresponding cells are adjacent to each other.



Global-Routing Problem

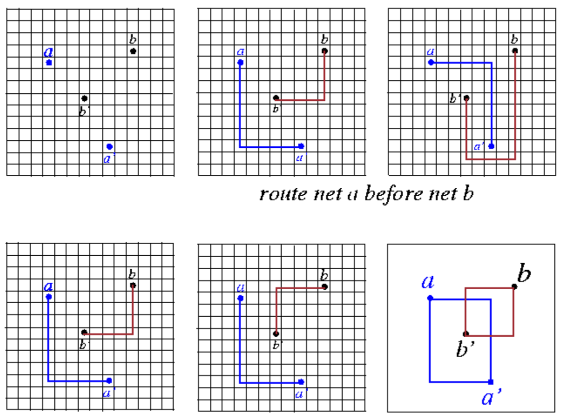
- Given a netlist $N=\{N_1, N_2, \ldots, N_n\}$, a routing graph $G=(V,E)$, find a Steiner tree T_i for each net N_i, $1 \le i \le n$, such that $U(e_j) \le c(e_j)$, $\forall\, e_j \in E$ and $\sum_i L(T_i)$ is minimized, where
  - $c(e_j)$: capacity of edge e_j
  - $x_{ij}=1$ if e_j is in T_i; $x_{ij}=0$ otherwise
  - $U(e_j) = \sum_i x_{ij}$: # of wires that pass through the channel corresponding to edge e_j
  - $L(T_i)$: total wirelength of Steiner tree T_i
- For high performance, the maximum wirelength $\max_i L(T_i)$ is minimized (or the longest path between two points in T_i is minimized).

Classification of Global-Routing Algorithms

- Sequential approach:
  - Select a net order and route nets sequentially in the order
  - Earlier routed nets might block the routing of subsequent nets
  - Routing quality heavily depends on net ordering
  - Strategy: Heuristic net ordering + rip-up and rerouting
- Concurrent approach:
  - All nets are considered simultaneously
    - E.g., 0-1 integer linear programming (0-1 ILP)

Net Ordering

- Net ordering greatly affects routing solutions.
- In the example, we should route net b before net a.



route net a before net b

route net b before net a

Net Ordering (cont'd)

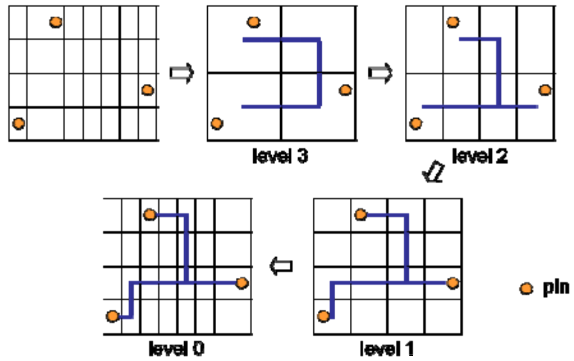
- Order the nets in the ascending order of the # of pins within their bounding boxes.
- Order the nets in the ascending (descending) order of their lengths if routability (timing) is the most critical metric.
- Order the nets based on their timing criticality.

Rip-Up and Re-routing

- Rip-up and re-routing is required if a global or detailed router fails in routing all nets.
- Approaches: the manual approach? the automatic procedure?
- Two steps in rip-up and re-routing
  1. Identify bottleneck regions, rip off some already routed nets.
  2. Route the blocked connections, and re-route the ripped-up connections.
- Repeat the above steps until all connections are routed or a time limit is exceeded.

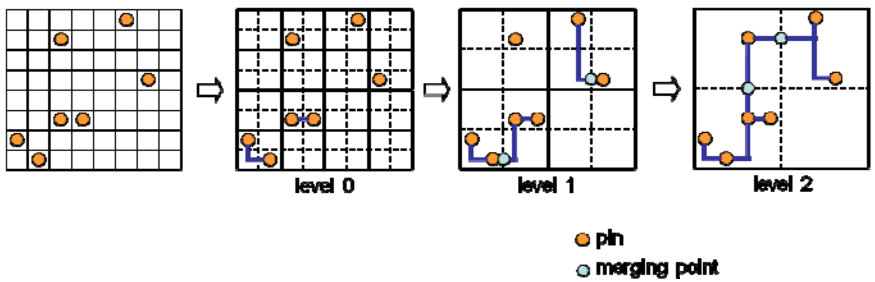
Top-down Hierarchical Global Routing

- Recursively divides routing regions into successively smaller **super cells**, and nets at each hierarchical level are routed sequentially or concurrently.



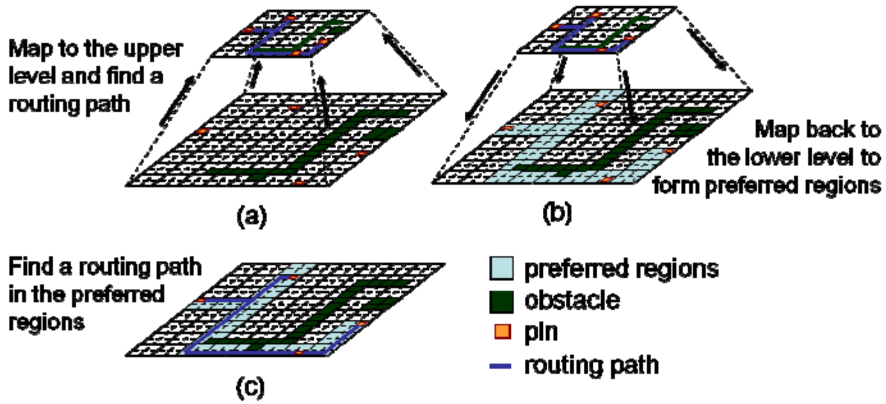
Bottom-up Hierarchical Global Routing

- At each hierarchical level, routing is restrained within each super cell individually.
- When the routing at the current level is finished, every four super cells are merged to form a new larger super cell at the next higher level.



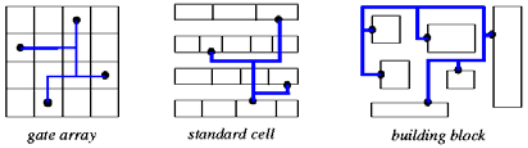
Hybrid Hierarchical Global Routing

- (1) neighboring propagation, (2) preference partitioning, and (3) bounded routing

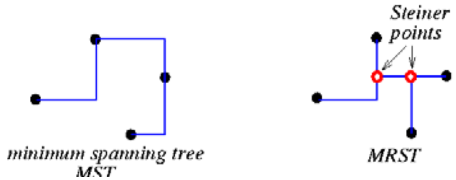


The Routing-Tree Problem

- **Problem:** Given a set of pins of a net, interconnect the pins by a "routing tree."

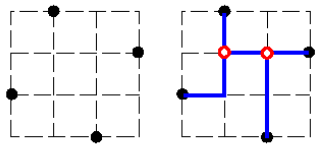


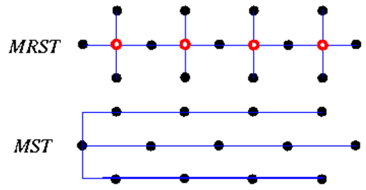
- **Minimum Rectilinear Steiner Tree (MRST) Problem:** Given *n* points in the plane, find a minimum-length tree of rectilinear edges which connects the points.
- $MRST(P) = MST(P \cup S)$, where *P* and *S* are the sets of original points and Steiner points, respectively.



Theoretical Results for the MRST Problem

- **Hanan's Thm:** There exists an MRST with all Steiner points (set *S*) chosen from the intersection points of horizontal and vertical lines drawn points of *P*.
  - Hanan, "On Steiner's problem with rectilinear distance," *SIAM J. Applied Math.*, 1966.
- **Hwang's Theorem:** For any point set *P*, $\frac{Cost(MST(P))}{Cost(MRST(P))} \leq \frac{3}{2}$.
  - Hwang, "On Steiner minimal tree with rectilinear distance," *SIAM J. Applied Math.*, 1976.
- Best existing approximation algorithm: Performance bound 61/48 by Foessmeier *et al*.

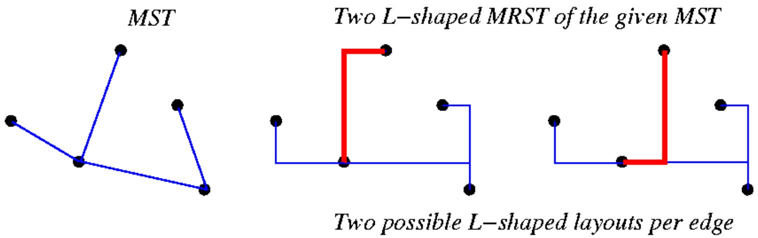

Hanan grid



Cost(MST)/Cost(MRST) –> 3/2

Coping with the MRST Problem

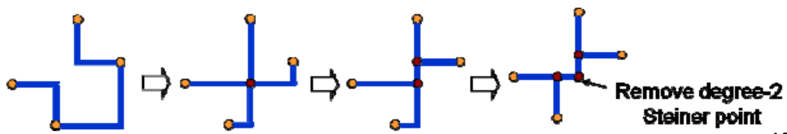
- Ho, Vijayan, Wong, "New algorithms for the rectilinear Steiner problem,"
  1. Construct an MRST from an MST.
  2. Each edge is straight or L-shaped.
  3. Maximize overlaps by dynamic programming.
- About 8% smaller than *Cost*(*MST*).



Iterated 1-Steiner Heuristic for MRST

- Kahng & Robins, "A new class of Steiner tree heuristics with good performance: the iterated 1-Steiner approach," *ICCAD*-90.

Algorithm: Iterated_1-Steiner(*P*)
***P*: set of *n* points.**
1 begin
2 $S \leftarrow \emptyset$;
    /* $H(P \cup S)$: set of Hanan points */
    /* $\Delta MST(A, B) = Cost(MST(A)) - Cost(MST(A \cup B))$ */
3 while ($Cand \leftarrow \{x \in H(P \cup S) \mid \Delta MST(P \cup S, \{x\}) > 0\} \neq \emptyset$) do
4   Find $x \in C$ and which maximizes $\Delta MST(P \cup S), \{x\})$;
5 $S \leftarrow S \cup \{x\}$;
6   Remove points in *S* which have degree ≤ 2 in $MST(P \cup S)$;
7 return $MST(P \cup S)$;
8 end

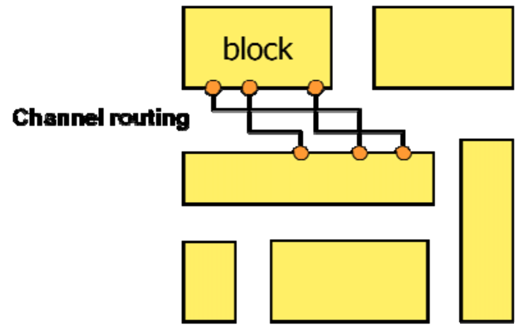


Outline

- Partitioning
- Floorplanning
- Placement
- Routing
  - Global rounting
  - Detailed routing
- Compaction

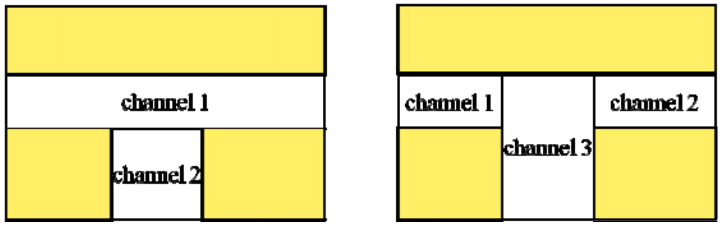
Channel Routing

- In earlier process technologies, channel routing was pervasively used since most wires were routed in the free space (*i.e.*, routing channel) between a pair of logic blocks (cell rows)



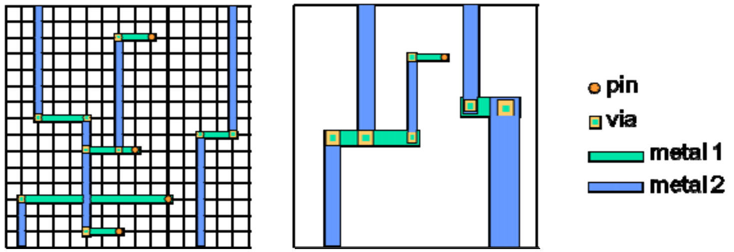
Routing Region Decomposition

- There are often various ways to decompose a routing region.
- The order of routing regions significantly affects the channel-routing process.



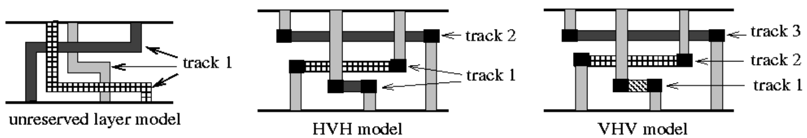
Routing Models

- **Grid-based model:**
  - A grid is super-imposed on the routing region.
  - Wires follow paths along the grid lines.
  - **Pitch:** distance between two gridded lines
  - **Gridless model:**
  - Any model that does not follow this "gridded" approach.



Models for Multi-Layer Routing

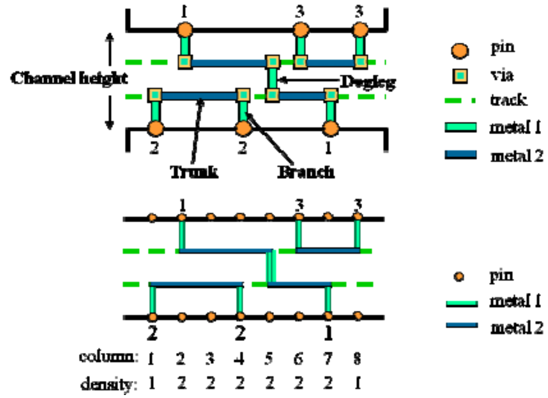
- **Unreserved layer model:** Any net segment is allowed to be placed in any layer.
- **Reserved layer model:** Certain type of segments are restricted to particular layer(s).
  - Two-layer: HV (Horizontal-Vertical), VH
  - Three-layer: HVH, VHV



3 types of 3−layer models

Terminology for Channel Routing

- Local density at column *i*, *d(i)*: total # of nets that crosses column *i*.
- **Channel density:** maximum local density
  - # of horizontal tracks required ≥ channel density.

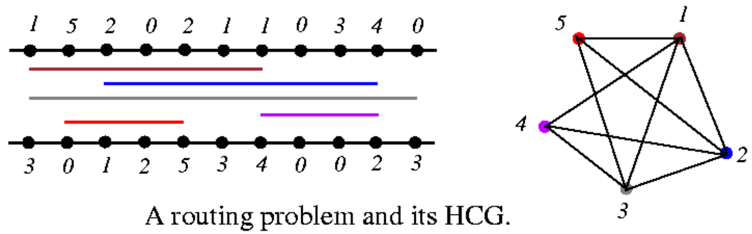


Channel Routing Problem

- Assignments of horizontal segments of nets to tracks
- Assignments of vertical segments to connect the following:
  - horizontal segments of the same net in different tracks, and
  - terminals of the net to horizontal segments of the net.
- Horizontal and vertical constraints must not be violated
  - Horizontal constraints between two nets: the horizontal span of two nets overlaps each other.
  - Vertical constraints between two nets: there exists a column such that the terminal on top of the column belongs to one net and the terminal on bottom of the column belongs to another net.
- Objective: Channel height is minimized (i.e., channel area is minimized).

Horizontal Constraint Graph (HCG)

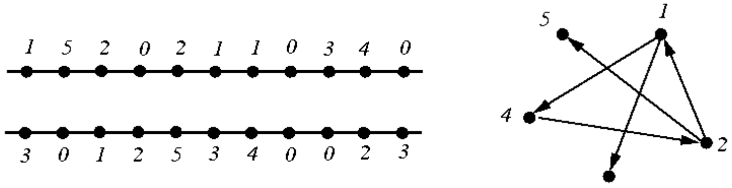
- HCG $G = (V, E)$ is **undirected** graph where
  - $V = \{ v_i \mid v_i$ represents a net $n_i \}$
  - $E = \{(v_i, v_j) \mid$ a horizontal constraint exists between n_i and $n_j\}$.
- For graph *G*: vertices ⇔ nets; edge (*i*, *j*) ⇔ net *i* overlaps net *j*.



A routing problem and its HCG.

Vertical Constraint Graph (VCG)

- VCG $G = (V, E)$ is **directed** graph where
  - $V = \{ v_i \mid v_i$ represents a net $n_i\}$
  - $E = \{(v_i, v_j) \mid$ a vertical constraint exists between n_i and $n_j\}$.
- For graph G: vertices ⇔ nets; edge $i \rightarrow j$ ⇔ net i must be above net j.



A routing problem and its VCG.

2-L Channel Routing: Basic Left-Edge Algorithm

- Hashimoto & Stevens, "Wire routing by optimizing channel assignment within large apertures," DAC-71.
- **No vertical constraint.**
- HV-layer model is used.
- **Doglegs are not allowed.**
- Treat each net as an interval.
- Intervals are sorted according to their left-end *x*-coordinates.
- Intervals (nets) are routed one-by-one according to the order.
- For a net, tracks are scanned from top to bottom, and the first track that can accommodate the net is assigned to the net.
- Optimality: produces a routing solution with the minimum # of tracks (if no vertical constraint).

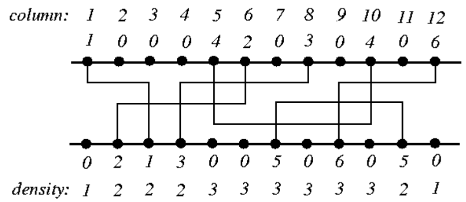
Basic Left-Edge Algorithm

Algorithm: Basic_Left-Edge(*U*, *track*[*j*])
U: set of unassigned intervals (nets) *I*1, ..., *In*;
Ij=[*sj*, *ej*]: interval *j* with left-end *x*-coordinate *sj* and right-end *ej*;
track[*j*]: track to which net *j* is assigned.

```
begin
U ← {I1, I2 , ..., In};
t ← 0;
while (U ≠ ∅ ) do
    t ← t + 1;
    watermark ← 0;
    while (there is an Ij ∈ U s.t. sj > watermark) do
      Pick the interval Ij ∈ U with sj > watermark,
      nearest watermark;
      track[j] ← t;
     watermark ← ej;
     U ← U - {Ij};
end
```

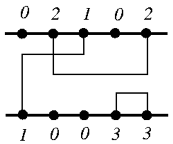

Basic Left-Edge Example

- $U = \{I_1, I_2, ..., I_6\}$; $I_1 = [1, 3]$, $I_2 = [2, 6]$, $I_3 = [4, 8]$, $I_4 = [5, 10]$, $I_5 = [7, 11]$, $I_6 = [9, 12]$.
- $t = 1$:
  - Route I_1: *watermark* = 3;
  - Route I_3 : *watermark* = 8;
  - Route I_6: *watermark* = 12;
- $t = 2$:
  - Route I_2 : *watermark* = 6;
  - Route I_5 : *watermark* = 11;
- $t = 3$: Route I_4

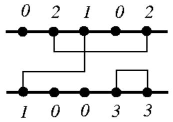


Basic Left-Edge Algorithm

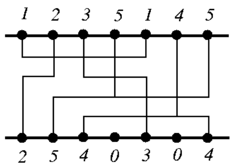
- If there is no vertical constraint, the basic left-edge algorithm is optimal.
- If there is any vertical constraint, the algorithm no longer guarantees optimal solution.

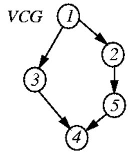


result from basic left−edge algorithm 3 tracks



optimal routing: 2 tracks





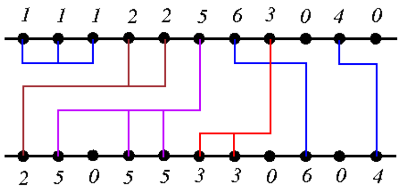
Constrained Left-Edge Algorithm

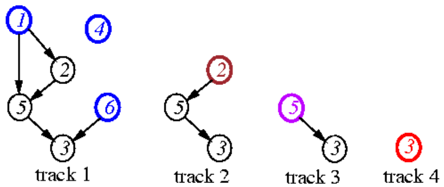
```
Algorithm: Constrained_Left-Edge(U, track[j])
U: set of unassigned intervals (nets) I1, ..., In;
Ij=[sj, ej]: interval j with left-end x-coordinate sj and right-end ej;
track[j]: track to which net j is assigned.

begin
U ← { I1, I2, ..., In};
t ← 0;
while (U ≠ ∅) do
    t ← t + 1;
    watermark ← 0;
    while (there is an unconstrained Ij ∈ U s.t. sj > watermark) do
    Pick the interval Ij ∈ U that is unconstrained,
      with sj > watermark, nearest watermark;
      track[j] ← t;
      watermark ← ej;
      U ← U - {Ij};
end
```


Constrained Left-Edge Example

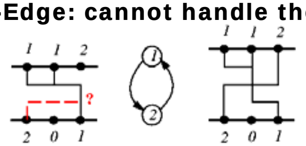
- I_1 = [1, 3], I_2 = [1, 5], I_3 = [6, 8], I_4 = [10, 11], I_5= [2, 6], I_6 = [7, 9].
- Track 1: Route I_1 (cannot route I_3); Route I_6; Route I_4.
- Track 2: Route I_2;
- Track 3: Route I_5.
- Track 4: Route I_3.



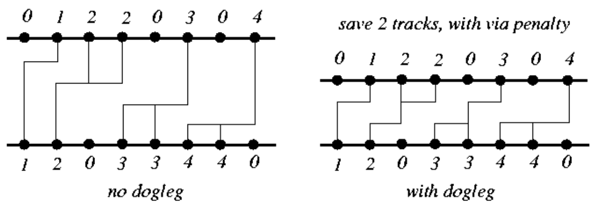


Dogleg Channel Router

- Deutch, "A dogleg channel router," 13rd DAC, 1976.
- **Drawback of Left-Edge: cannot handle the cases with constraint cycles.**

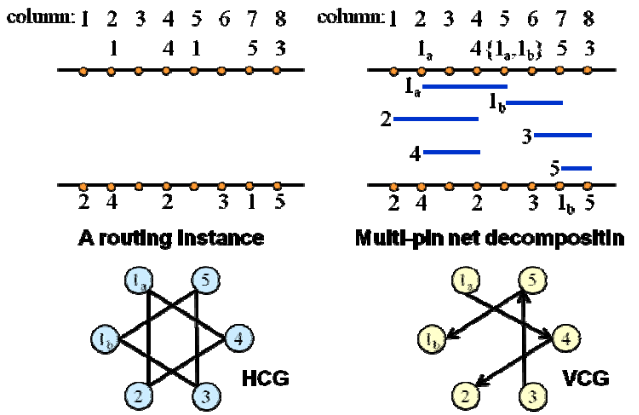


- **Drawback of Left-Edge: the entire net is on a single track.**
  - **Doglegs** are used to place parts of a net on different tracks to minimize channel height.
  - Might incur penalty for additional vias.



Dogleg Channel Router

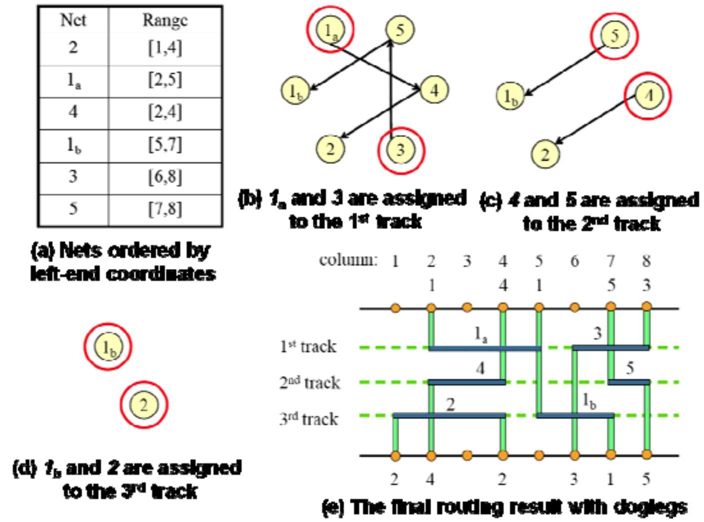
- Each multi-pin net is broken into a set of 2-pin nets.
- Modified Left-Edge Algorithm is applied to each subnet.



Dogleg Channel Routing Example

Net	Range
2	[1,4]
1_a	[2,5]
4	[2,4]
1_b	[5,7]
3	[6,8]
5	[7,8]

(a) Nets ordered by left-end coordinates



Modern Routing Considerations

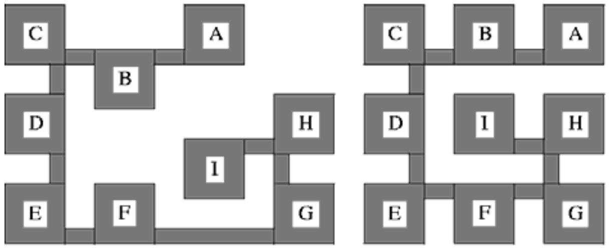
- Signal/power Integrity
  - Capacitive crosstalk
  - Inductive crosstalk
  - IR drop
- Manufacturability
  - Process variation
  - Optical proximity correction (OPC)
  - Chemical mechanical polishing (CMP)
  - Phase-Shift Mask (PSM)
- Reliability
  - Double via insertion
  - Process antenna effect
  - Electromigration (EM)
  - Electrostatic discharge (ESD)

Outline

- Partitioning
- Floorplanning
- Placement
- Routing
- Compaction

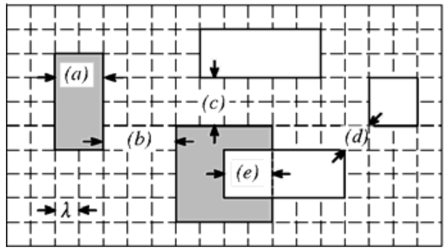
Layout Compaction

- Course contents
  - Design rules
  - Symbolic layout
  - Constraint-graph compaction



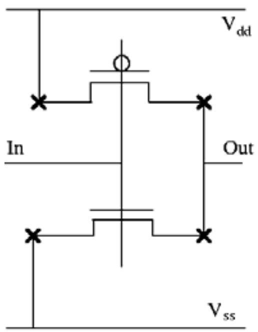
Design Rules

- **Design rules:** restrictions on the mask patterns to increase the probability of successful fabrication.

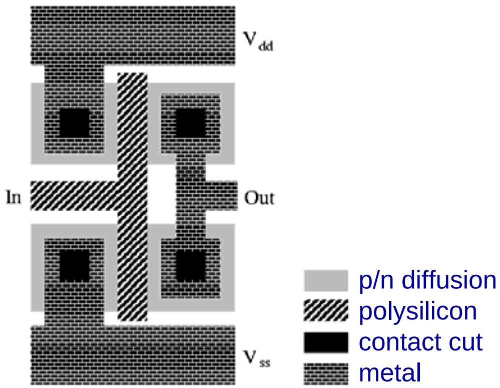


- Patterns and design rules are often expressed in λ rules.
- Most common design rules:
  - minimum-width rules (valid for a mask pattern of a specific layer): (a).
  - minimum-separation rules (between mask patterns of the same layer or different layers): (b), (c).
  - minimum-overlap rules (mask patterns in different layers): (e).

CMOS Inverter Layout Example



Symbolic layout



Geometric layout

Symbolic Layout

- Geometric (mask) layout: coordinates of the layout patterns (rectangles) are absolute (or in multiples of λ).
- Symbolic (topological) layout: only relations between layout elements (below, left to, etc) are known.
  - Symbols are used to represent elements located in several layers, e.g. transistors, contact cuts.
  - The *length*, *width* or *layer* of a wire or other layout element might be left unspecified.
  - Mask layers not directly related to the functionality of the circuit do not need to be specified, e.g. n-well, p-well.
- The symbolic layout can work with a technology file that contains all design rule information for the target technology to produce the geometric layout.

Compaction and Its Applications

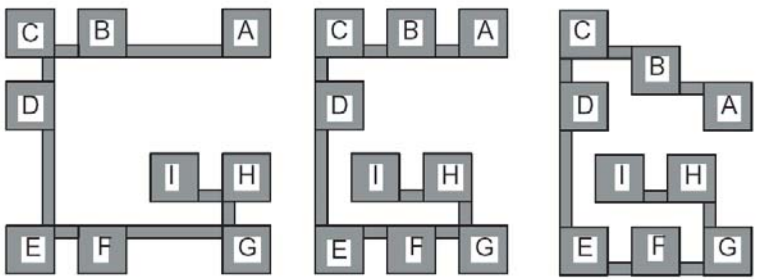
- A compaction program or compactor generates layout at the mask level. It attempts to make the layout as dense as possible.
- Applications of compaction:
  - Area minimization: remove redundant space in layout at the mask level.
  - Layout compilation: generate mask-level layout from symbolic layout.
  - Redesign: automatically remove design-rule violations.
  - Rescaling: convert mask-level layout from one technology to another.

Aspects of Compaction

- Dimension:
  - 1-dimensional (1D) compaction: layout elements only are moved or shrunk in one dimension (*x* or *y* direction).
    - Is often performed first in the x-dimension and then in the y-dimension (or vice versa).
  - 2-dimensional (2D) compaction: layout elements are moved and shrunk simultaneously in two dimensions.
- Complexity:
  - 1D compaction can be done in polynomial time.
  - 2D compaction is NP-hard.

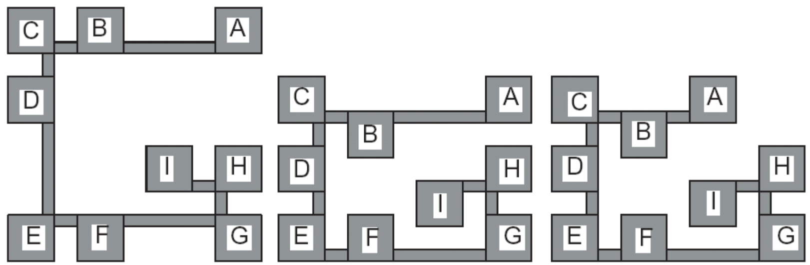
1D Compaction: *X* Followed By *Y*

- Each square is 2 λ * 2 λ, minimum separation is 1 λ.
- Initially, the layout is 11 λ * 11 λ.
- After compacting along the x direction, then the y direction, we have the layout size of 8 λ * 11 λ.



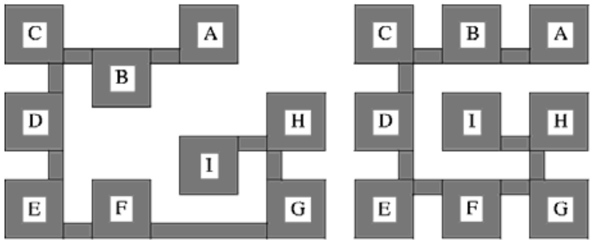
1D Compaction: *Y* Followed By *X*

- Each square is 2 λ * 2 λ, minimum separation is 1 λ.
- Initially, the layout is 11 λ * 11 λ.
- After compacting along the y direction, then the x direction, we have the layout size of 11 λ * 8 λ.



2D Compaction

- Each square is 2 λ * 2 λ, minimum separation is 1 λ.
- Initially, the layout is 11 λ * 11 λ.
- After 2D compaction, the layout size is only 8 λ * 8 λ.

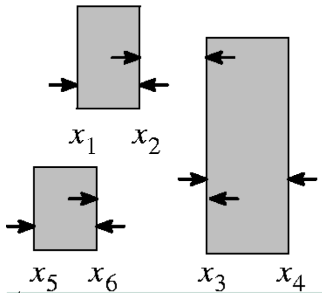


- Since 2D compaction is NP-complete, most compactors are based on repeated 1D compaction.

Inequalities for Distance Constraints

- Minimum-distance design rules can be expressed as inequalities.

  $x_j - x_i \geq d_{ij}$.

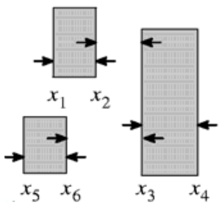


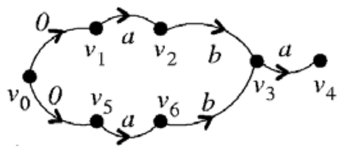
- For example, if the minimum width is a and the minimum separation is b, then

  $x_2 - x_1 \geq a$
  $x_3 - x_2 \geq b$
  $x_3 - x_6 \geq b$

The Constraint Graph

- The inequalities can be used to construct a constraint graph $G(V, E)$:
  - There is a vertex v_i for each variable x_i.
  - For each inequality $x_j - x_i \geq d_{ij}$ there is an edge (v_i, v_j) with weight d_{ij} .
  - There is an extra source vertex, v_0; it is located at $x = 0$; all other vertices are at its right.
- If all the inequalities express minimum-distance constraints, the graph is acyclic (DAG).
- The longest path in a constraint graph determines the layout dimension.

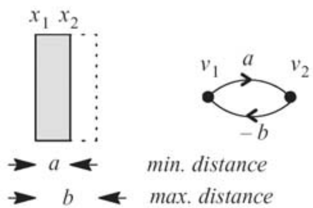


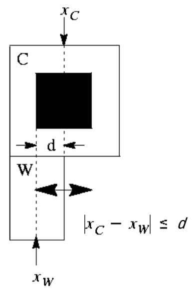


constraint graph

Maximum-Distance Constraints

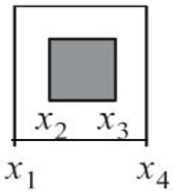
- Sometimes the distance of layout elements is bounded by a maximum, e.g., when the user wants a maximum wire width, maintains a wire connecting to a via, etc.
  - A maximum distance constraint gives an inequality of the form: $x_j - x_i \leq c_{ij}$ or $x_i - x_j \geq -c_{ij}$
  - Consequence for the constraint graph: backward edge
    - (v_j, v_i) with weight $d_{ji} = -c_{ij}$; the graph is not acyclic anymore.
- The longest path in a constraint graph determines the layout dimension.

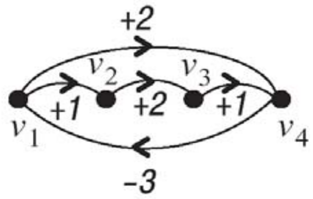




Longest-Paths in Cyclic Graphs

- Constraint-graph compaction with maximum-distance constraints requires solving the longest-path problem in cyclic graphs.
- Two cases are distinguished:
  - There are positive cycles: No feasible solution for longest paths. We shall detect the cycles.
  - All cycles are negative: Polynomial-time algorithms exist.



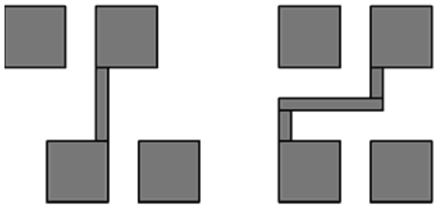


Longest and Shortest Paths

- Longest paths become shortest paths and vice versa when edge weights are multiplied by –1.
- Situation in DAGs: both the longest and shortest path problems can be solved in linear time.
- Situation in cyclic directed graphs:
  - All weights are positive: shortest-path problem in P (Dijkstra), no feasible solution for the longest-path problem.
  - All weights are negative: longest-path problem in P (Dijkstra), no feasible solution for the shortest-path problem.
  - No positive cycles: longest-path problem is in P.
  - No negative cycles: shortest-path problem is in P.

Remarks on Constraint-Graph Compaction

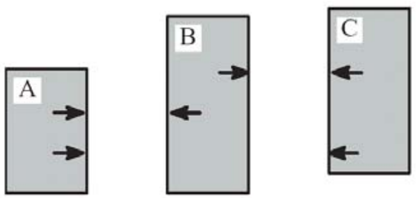
- Noncritical layout elements: Every element outside the critical paths has freedom on its best position => may use this freedom to optimize some cost function.
- Automatic jog insertion: The quality of the layout can further be improved by automatic jog insertion.



- Hierarchy: A method to reduce complexity is hierarchical compaction, e.g., consider cells only.

Constraint Generation

- The set of constraints should be irredundant and generated efficiently.
- An edge (v_i, v_j) is redundant if edges (v_i, v_k) and (v_k, v_j) exist and $w((v_i, v_j)) \leq w((v_i, v_k)) + w((v_k, v_j))$.
  - The minimum-distance constraints for (A, B) and (B, C) make that for (A, C) redundant.



- Doenhardt and Lengauer have proposed a method for irredundant constraint generation with complexity $O(n \log n)$.