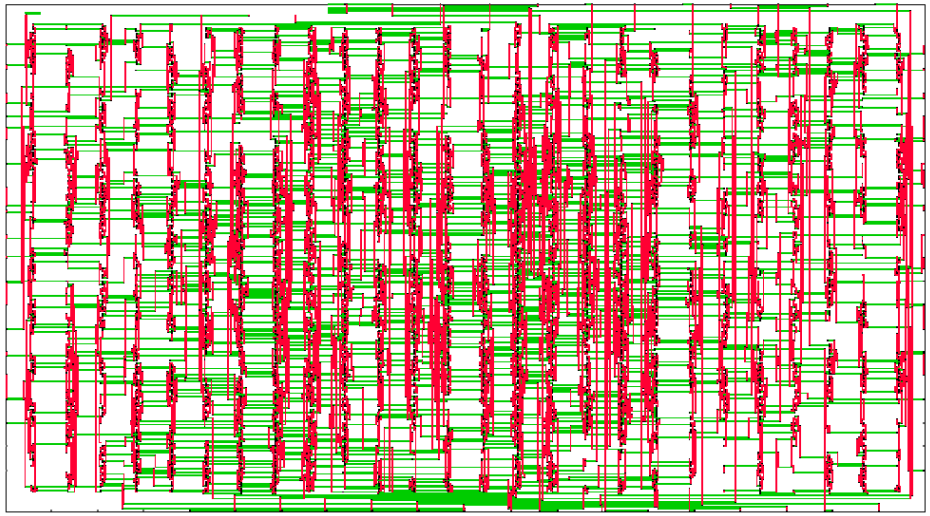
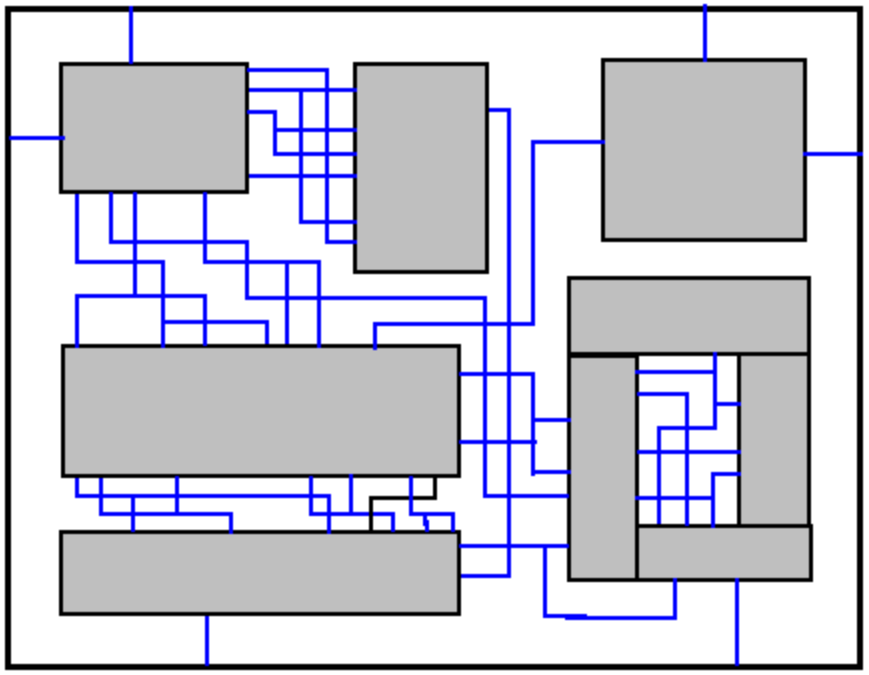
Unit 7: Detailed Routing

- Course contents:
  - Channel routing
  - Full-chip routing
- Readings
  - W&C&C: Chapter 12
  - S&Y: Chapter 7

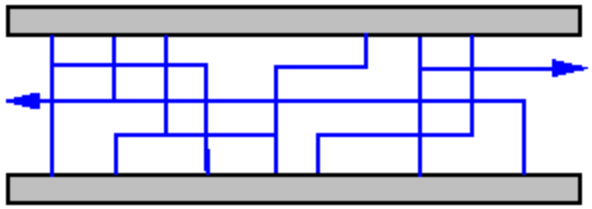


Courtesy of Y.-W. Chang

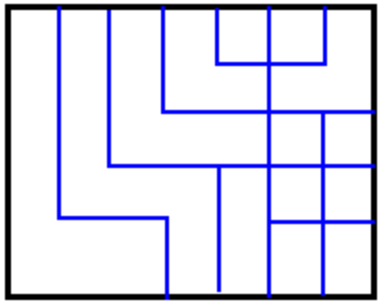
Channel/Switchbox Routing



Detailed routing



channel routing



switchbox routing

Courtesy of Y.-W. Chang

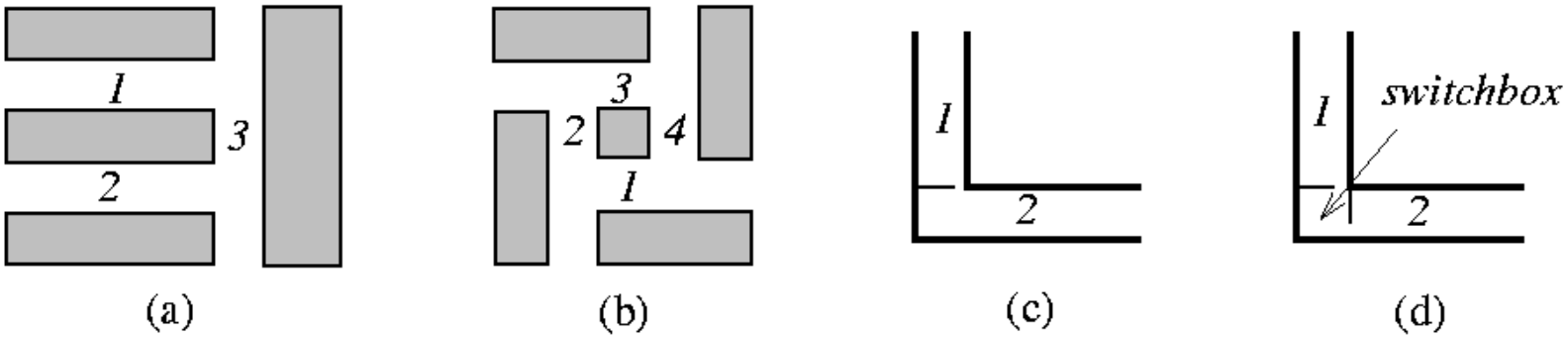
Order of Routing Regions and L-Channels

(a) No conflicts in case of routing in the order of 1, 2, and 3.

(b) No ordering is possible to avoid conflicts.

(c) The situation of (b) can be resolved by using L-channels.

(d) An L-channel can be decomposed into two channels and a switchbox.



Courtesy of Y.-W. Chang

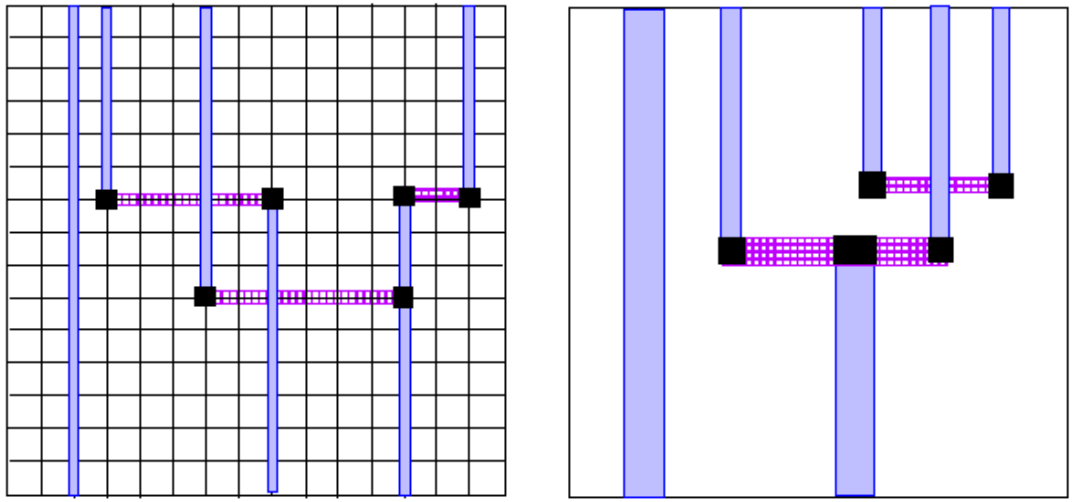
Routing Considerations

- Number of terminals (two-terminal vs. multi-terminal nets)
- Net widths (power and ground vs. signal nets)
- Via restrictions (stacked vs. conventional vias)
- Boundary types (regular vs. irregular)
- Number of layers (two vs. three, more layers?)
- Net types (critical vs. non-critical nets)

Courtesy of Y.-W. Chang

Routing Models

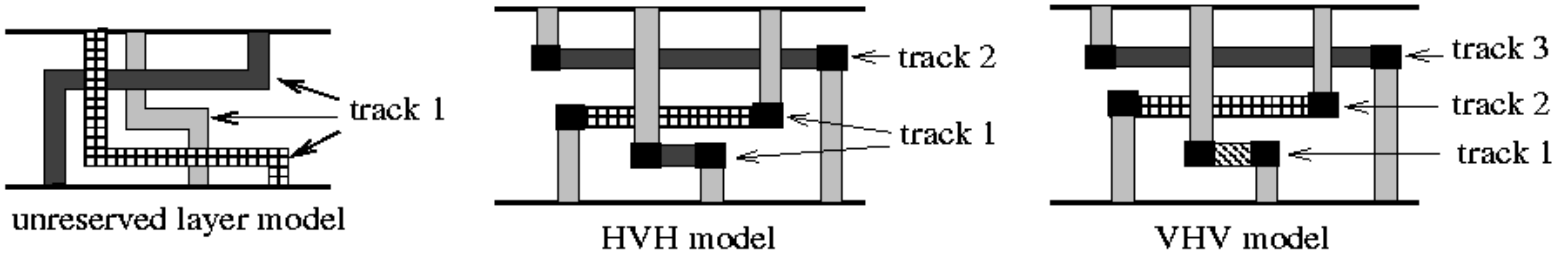
- **Grid-based model:**
  - A grid is super-imposed on the routing region.
  - Wires follow paths along the grid lines.
  - **Pitch:** distance between two gridded lines
- **Gridless model:**
  - Any model that does not follow this "gridded" approach.



Courtesy of Y.-W. Chang

Models for Multi-Layer Routing

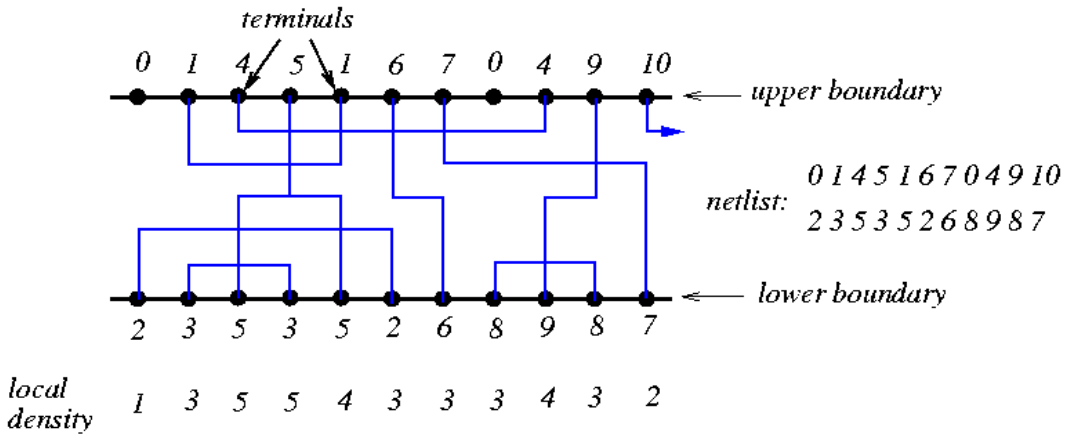
- **Unreserved layer model:** Any net segment is allowed to be placed in any layer.
  - Less popular, but wrong-way jogs?
- **Reserved layer model:** Certain type of segments are restricted to particular layer(s).
  - More popular model
  - Two-layer: HV (horizontal-Vertical), VH
  - Three-layer: HVH, VHV

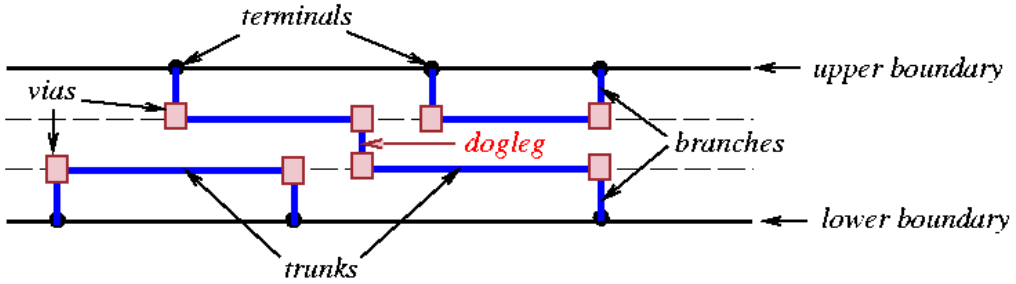


3 types of 3−layer models

Courtesy of Y.-W. Chang

Terminology for Channel Routing Problems





- 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.

Courtesy of Y.-W. Chang

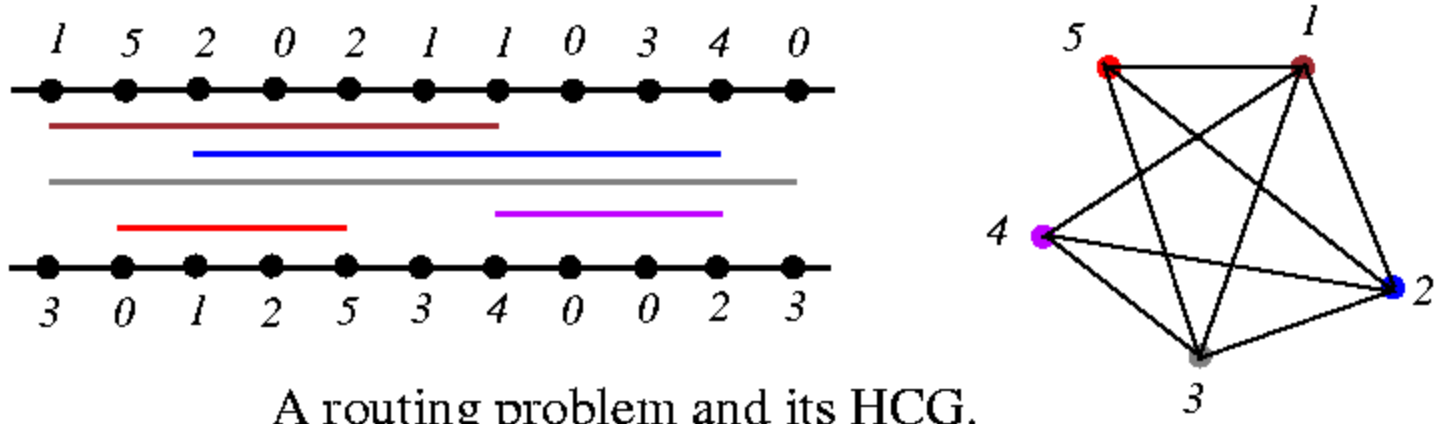
Channel Routing Problem

- Assignments of horizontal segments of nets to tracks.
- Assignments of vertical segments to connect
  - horizontal segments of the same net in different tracks, and
  - the 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 the other net.
- Objective: Channel height is minimized (i.e., channel area is minimized).

Courtesy of Y.-W. Chang

Horizontal Constraint Graph (HCG)

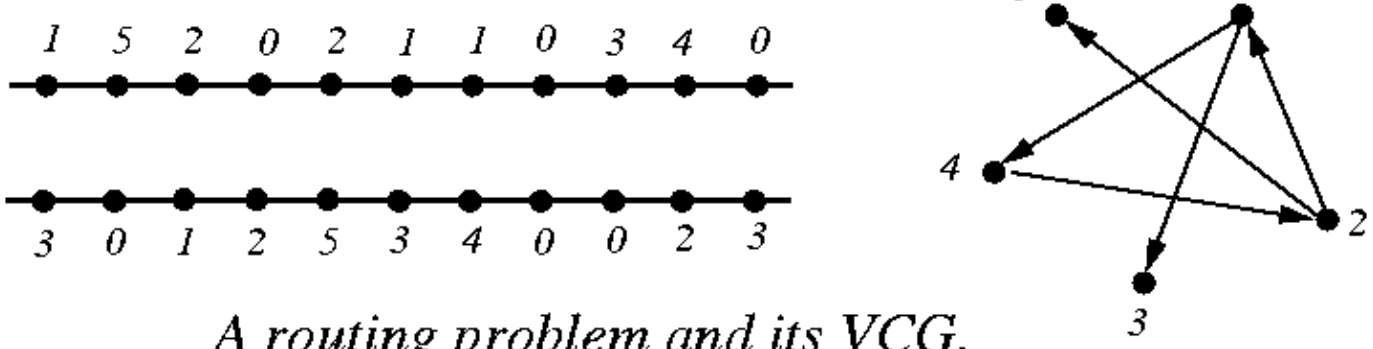
- HCG $G = (V, E)$ is undirected graph where
  - $V = \{ v_i \mid v_i$ represents a net $n_i\}$
  - $E = \{(v_i, v_j) \mid$ a horizontal constraint exists between n_i and $n_j\}$.
- For graph G: vertices ⇔ nets; edge (i, j) ⇔ net i overlaps net j.



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 $\Leftrightarrow$ nets; edge $i \rightarrow j \Leftrightarrow$ 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).

Courtesy of Y.-W. Chang

Basic Left-Edge Algorithm

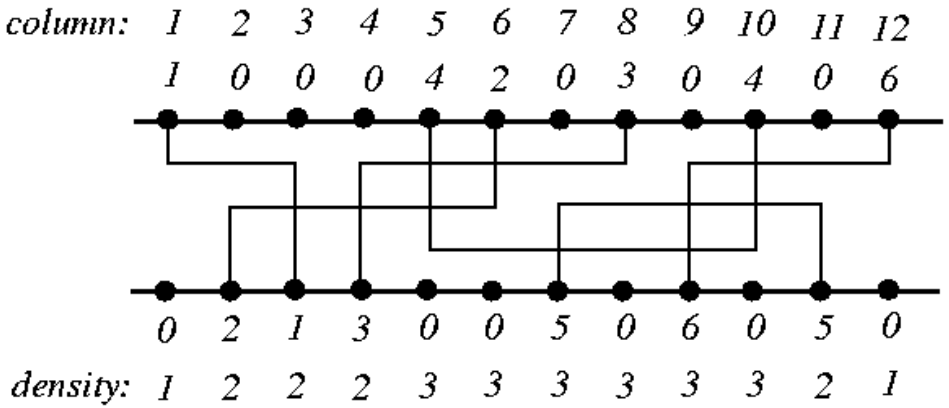
Algorithm: Basic_Left-Edge(*U, track*[*j*])
U: set of unassigned intervals (nets) $I_1, \ldots, I_n$;
$I_j=[s_j, e_j]$: interval *j* with left-end *x*-coordinate s_j and right-end e_j;
track[*j*]: track to which net *j* is assigned.

```
1  begin
2  U ← {I1, I2 , …, In};
3  t ← 0;
4  while (U ≠ ∅ ) do
5      t ← t + 1;
6      watermark ← 0;
7      while (there is an Ij ∈ U s.t. sj > watermark) do
8          Pick the interval Ij ∈ U with sj > watermark,
           nearest watermark;
9          track[j] ← t;
10         watermark ← ej;
11         U ← U - {Ij};
12 end
```


Courtesy of Y.-W. Chang

Basic Left-Edge Example

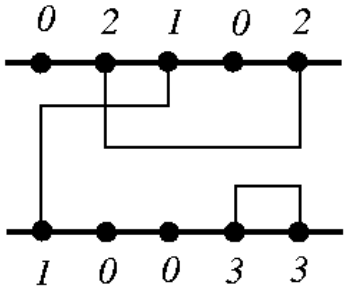
- $U = \{I_1, I_2, \ldots, I_6\}$; $I_1 = [1, 3]$, $I_2 = [2, 6]$, $I_3 = [4, 8]$, $I_4 = [5, 10]$, $I_5 = [7, 11]$, $I_6 = [9, 12]$.
- t =1:
  - Route I_1: *watermark* = 3;
  - Route I_3 : *watermark* = 8;
  - Route I_6: *watermark* = 12;
- t = 2:
  - Route I_2 : *watermark* = 6;
  - Route I_5 : *watermark*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



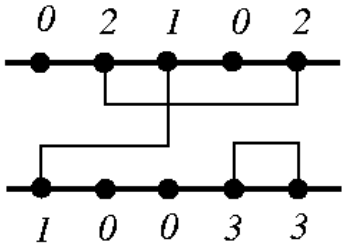
Courtesy of Y.-W. Chang

Basic Left-Edge Algorithm

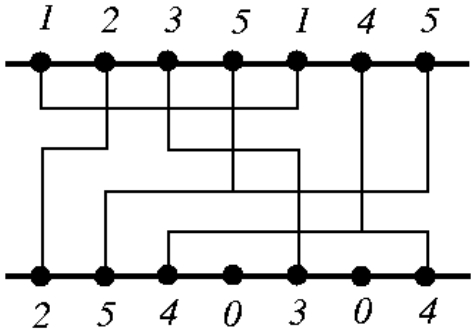
- If there is no vertical constraint, the basic left-edge algorithm is optimal.
- If there is any vertical constraint, the algorithm no longer guarantees optimal solution.

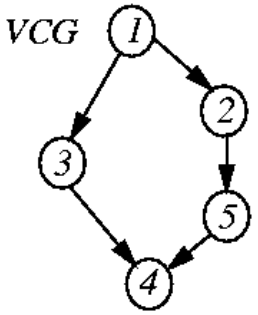


result from basic left–edge algorithm 3 tracks



optimal routing: 2 tracks





Courtesy of Y.-W. Chang

Constrained Left-Edge Algorithm

Algorithm: Constrained_Left-Edge(U, $track[j]$**)**

U: set of unassigned intervals (nets) I_1, …, I_n;

$I_j=[s_j, e_j]$: interval j with left-end x-coordinate s_j and right-end e_j;

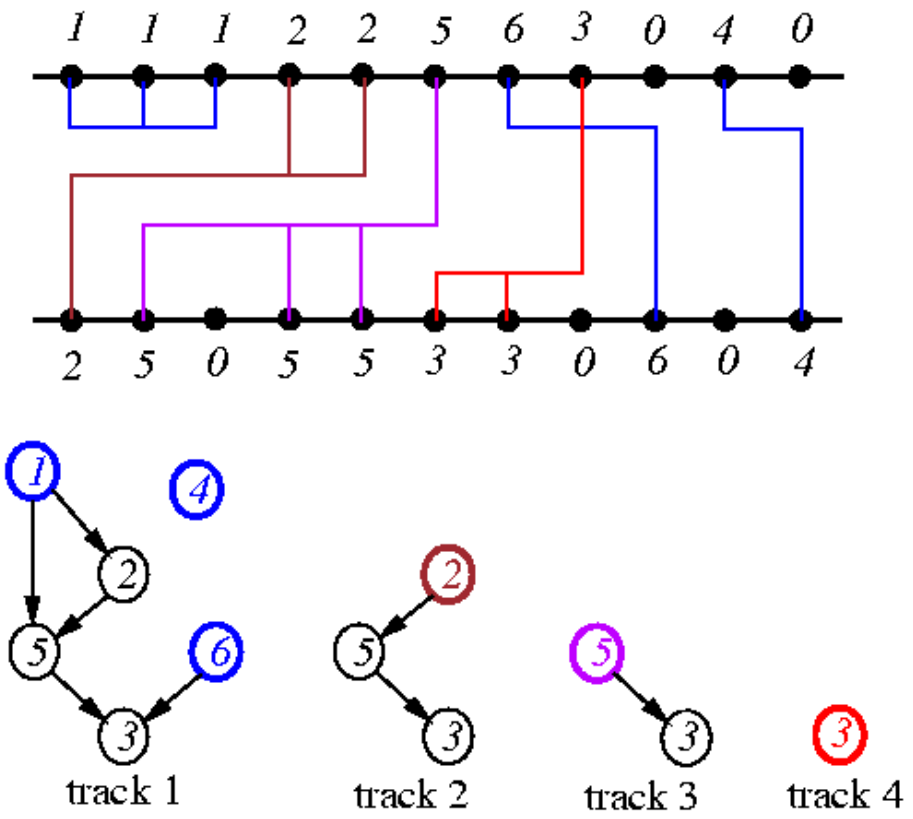
$track[j]$: track to which net j is assigned.

```
1 begin
2 U ← { I1, I2, …, In};
3 t ← 0;
4 while (U ≠ ∅) do
5     t ← t + 1;
6     watermark ← 0;
7     while (there is an unconstrained Ij ∈ U s.t. sj > watermark) do
8     Pick the interval Ij ∈ U that is unconstrained,
          with sj > watermark, nearest watermark;
9         track[j] ← t;
10        watermark ← ej;
11        U ← U - {Ij};
12 end
```

Courtesy of Y.-W. Chang

Constrained Left-Edge Example

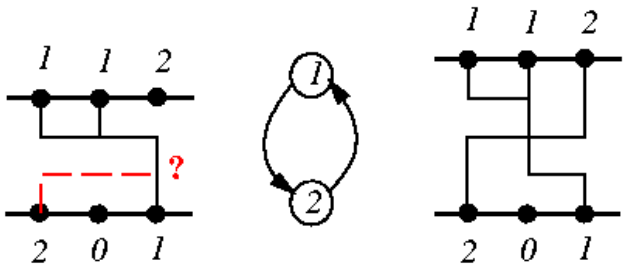
- I_1 = [1, 3], I_2 = [1, 5], I_3 = [6, 8], I_4 = [10, 11], I_5= [2, 6], I_6 = [7, 9].
- Track 1: Route I_1 (cannot route I_3); Route I_6; Route I_4.
- Track 2: Route I_2; cannot route I_3.
- Track 3: Route I_5.
- Track 4: Route I_3.



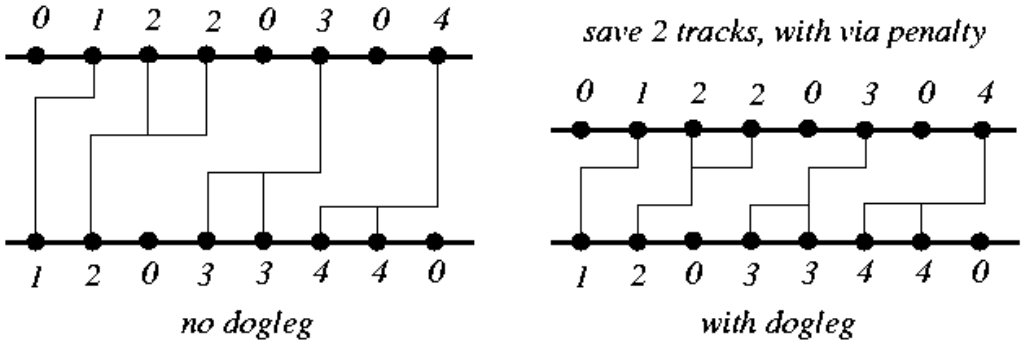
Courtesy of Y.-W. Chang

Dogleg Channel Router

- Deutsch, “A dogleg channel router,” 13rd DAC, 1976.
- **Drawback of Left-Edge: cannot handle the cases with constraint cycles.**
  - **Doglegs** are used to resolve constraint cycle.

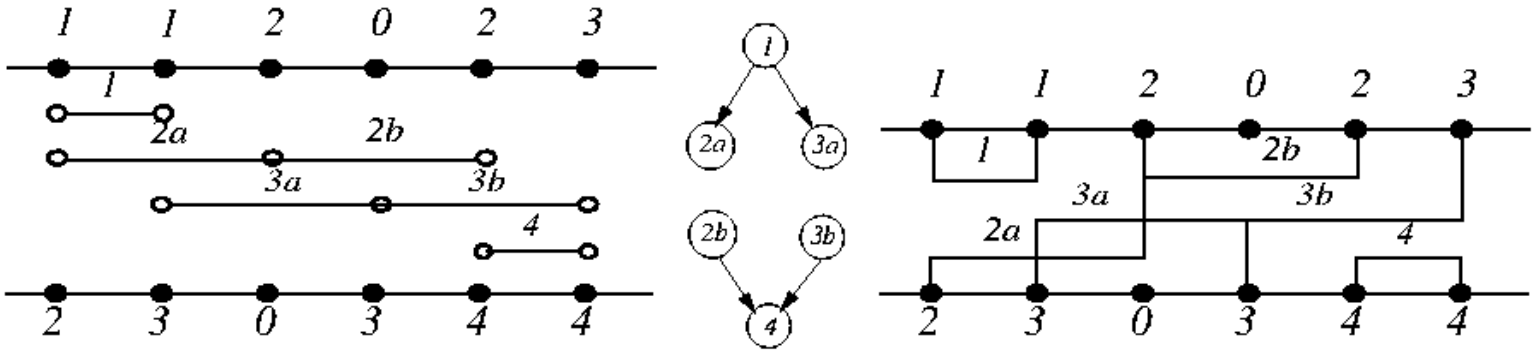


- **Drawback of Left-Edge: the entire net is on a single track.**
  - **Doglegs** are used to place parts of a net on different tracks to minimize channel height.
  - Might incur penalty for additional vias.



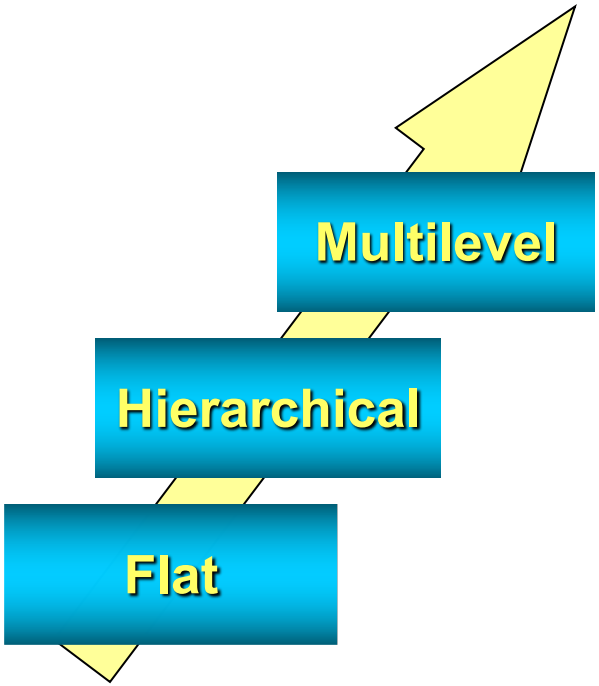
Dogleg Channel Router

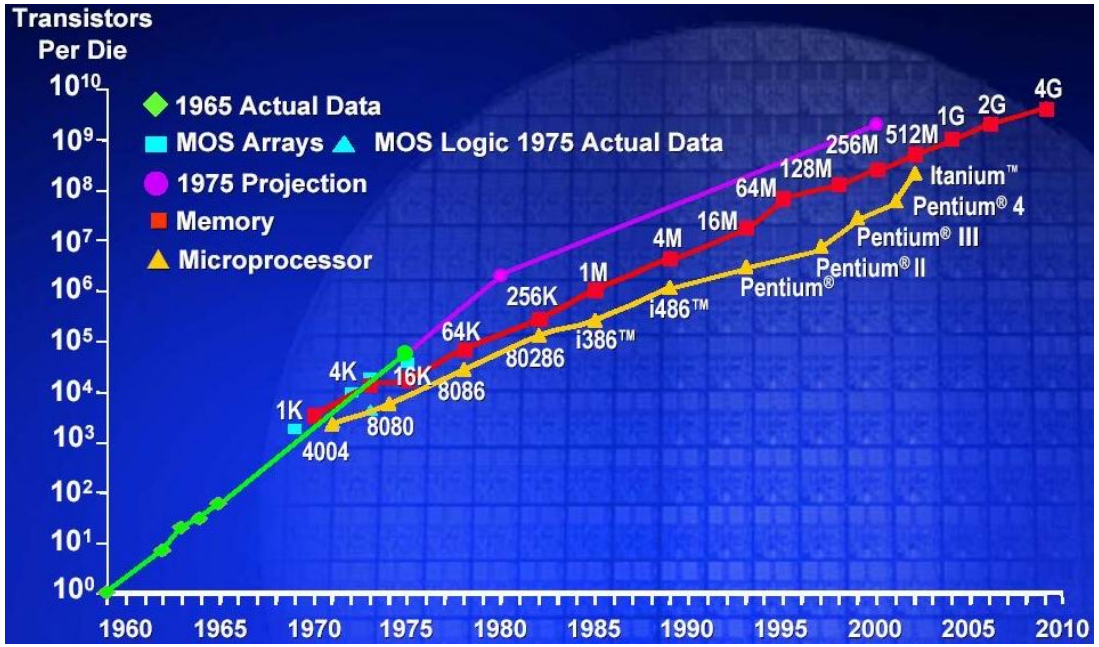
- Each multi-terminal net is broken into a set of 2-terminal nets.
- Two parameters are used to control routing:
  - Range: Determine the # of consecutive 2-terminal subnets of the same net that can be placed on the same track.
  - Routing sequence: Specifies the starting position and the direction of routing along the channel.
- Modified Left-Edge Algorithm is applied to each subnet.



Framework Evolution

- Billions of transistors may be fabricated in a single chip for nanometer technology.
- Need frameworks for very large-scale designs.
- Framework evolution for EDA tools: Flat ➔ Hierarchical ➔ Multilevel



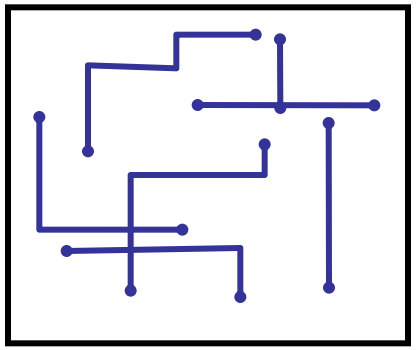


Source: Intel (ISSCC-03)

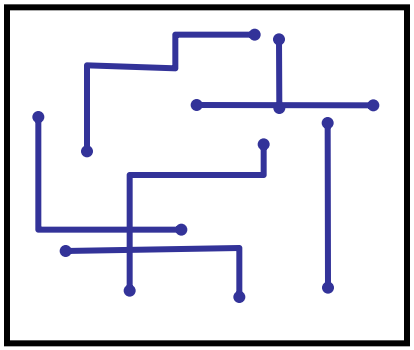
Courtesy of Y.-W. Chang

Flat Routing Framework

- Sequential approaches
  - Maze searching
  - Line searching
- Concurrent approaches
  - Network-flow based algorithms
  - Linear assignment formulation
- Drawback: hard to handle larger problems



Sequential

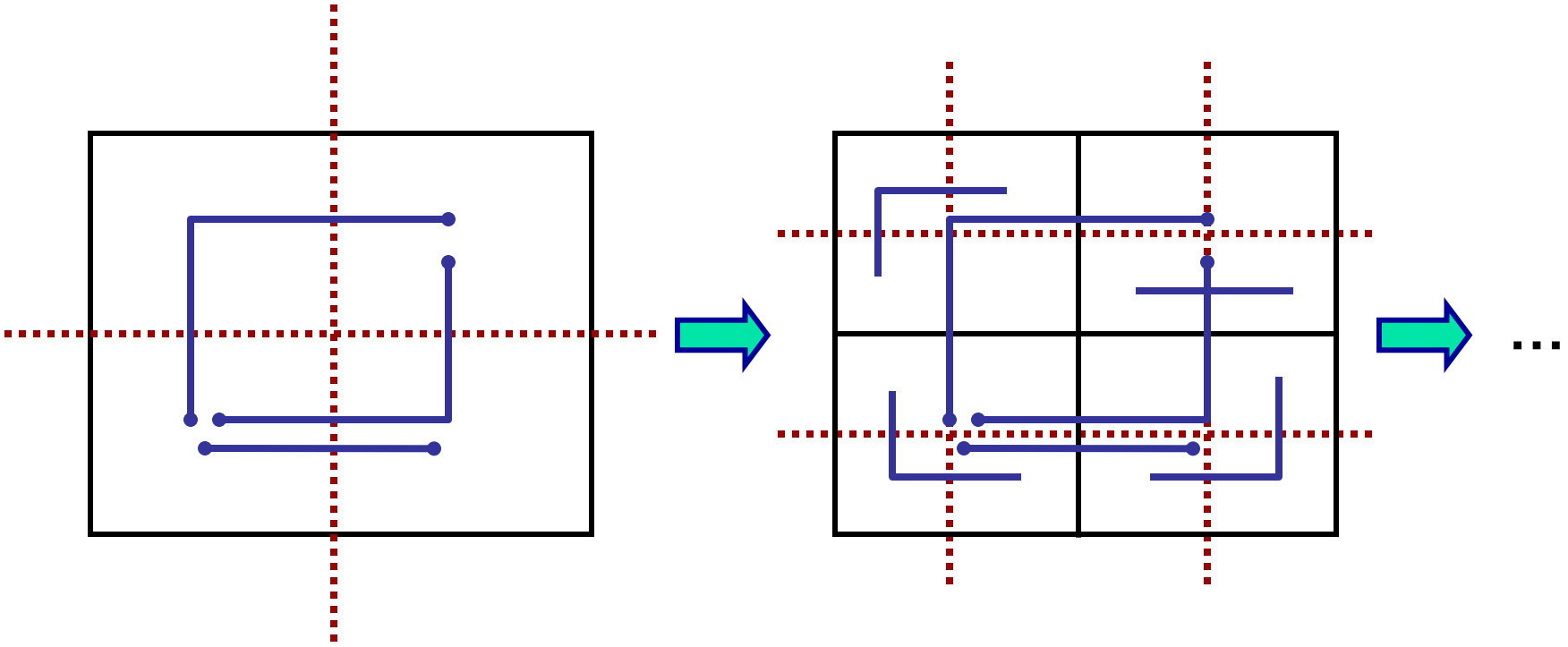


Concurrent

Courtesy of Y.-W. Chang

Hierarchical Routing Framework

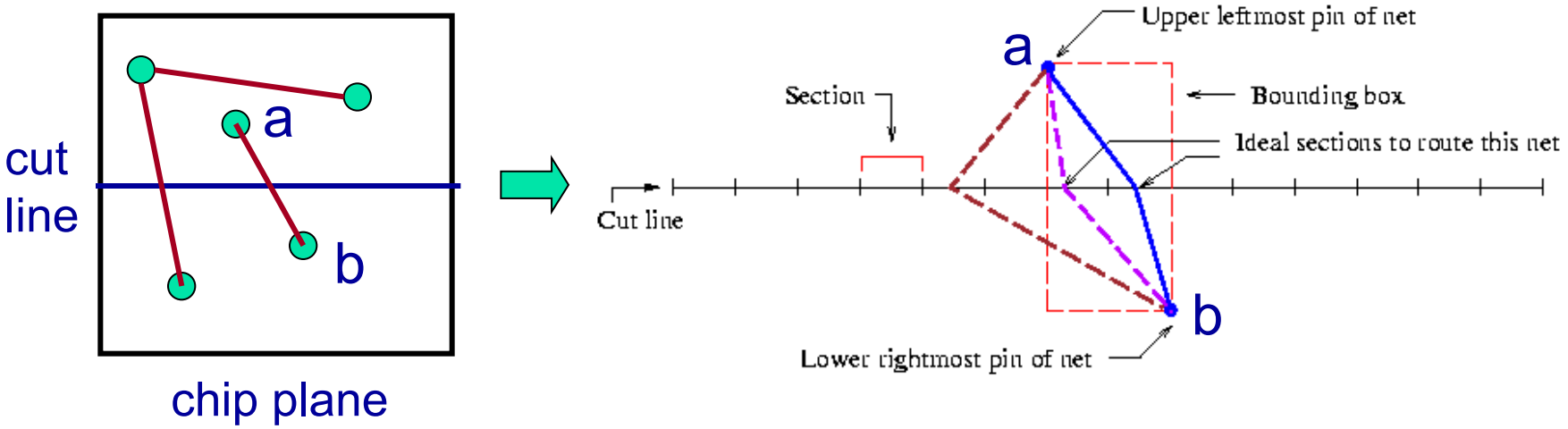
- The hierarchical approach recursively divides a routing region into a set of subregions and solve those subproblems independently.
- Drawbacks: lack the global information for the interaction among subregions.



Courtesy of Y.-W. Chang

Hierarchical Routing Revisited

- Global routing can be formulated as a linear assignment problem:
  - $x_{i,j}$ = 1 if net *i* is assigned to section *j*; $x_{i,j}$ = 0 otherwise.
  - Each net crosses the cut line exactly once: $\sum_{j=1}^{M} x_{ij} = 1, 1 \le i \le N.$
  - Capacity constraint of each section: $\sum_{i=1}^{N} x_{ij} \le C, 1 \le j \le M.$
  - w_{ij}: cost of assigning net *i* to section *j*. Minimize $\sum_{i=1}^{N} \sum_{j=1}^{M} w_{ij} x_{ij}$.



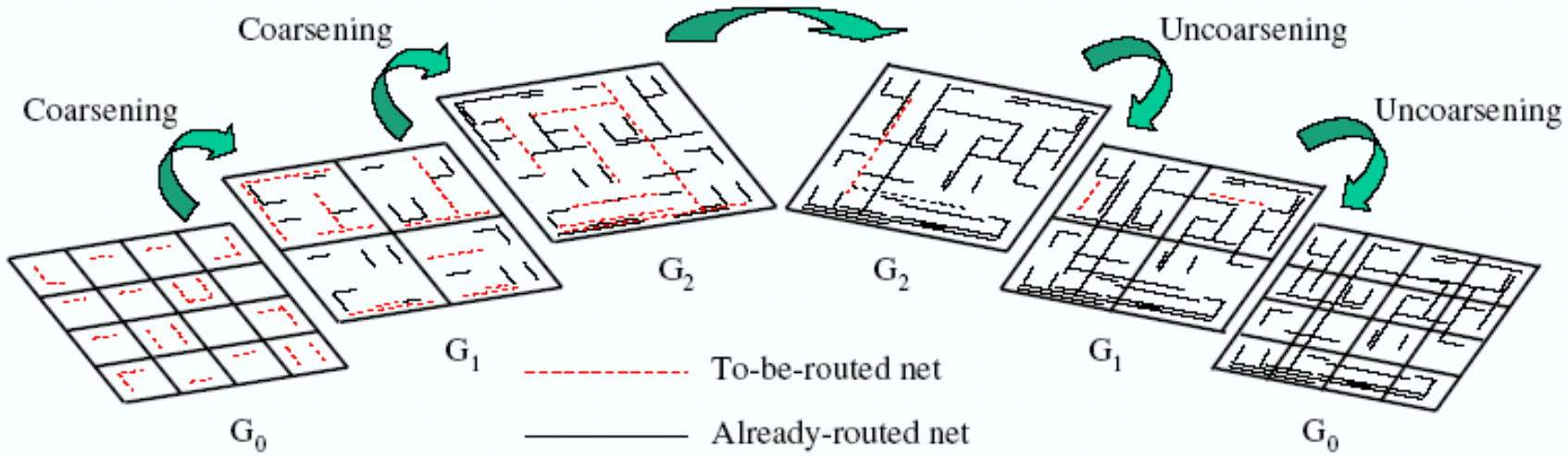
Courtesy of Y.-W. Chang

Multilevel Full-Chip Routing Framework

- Lin and Chang, “A novel framework for multilevel routing considering routability and performance,” ICCAD-2002 (TCAD, 2003).
- Multilevel framework: coarsening followed by uncoarsening.
- Coarsening (bottom-up) stage:
  - Constructs the net topology based on the minimum spanning tree.
  - Processes routing tiles one by one at each level, and only local nets (connections) are routed.
  - Applies two-stage routing of global routing followed by detailed routing.
  - Uses the L-shaped & Z-shaped pattern routing.
  - Performs resource estimation after detailed routing to guide the routing at the next level.
- Uncoarsening (top-down) stage
  - Completes the failed nets (connections) from the coarsening stage.
  - Uses a global and a detailed maze routers to refine the solution.

Courtesy of Y.-W. Chang

A Multilevel Full-Chip Routing Framework



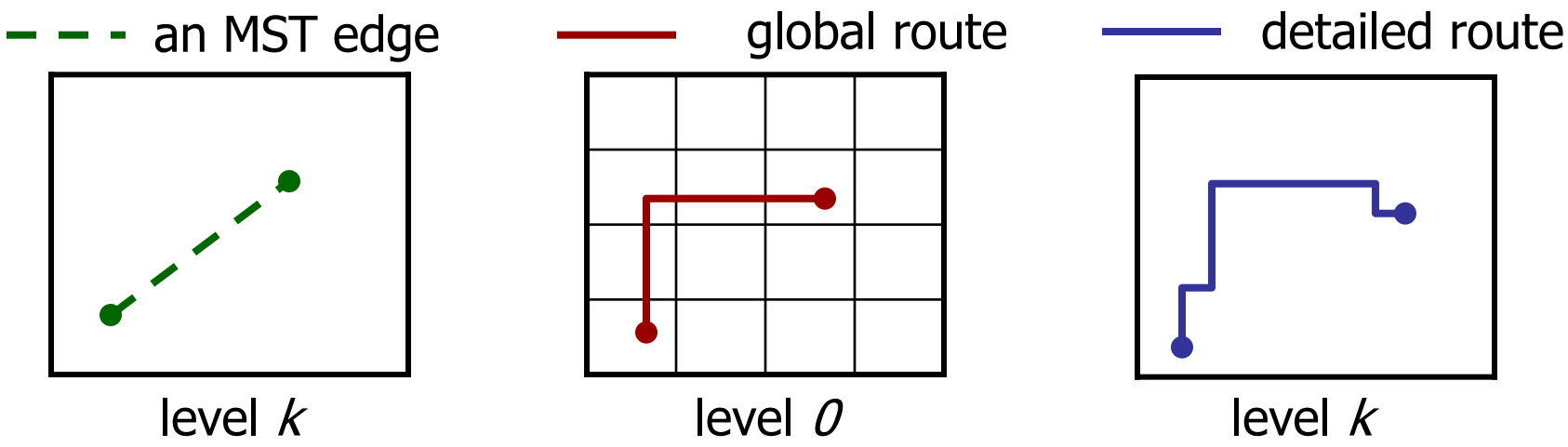
Perform global pattern routing and Dijkstra's shortest path detailed routing for local connections and then estimate routing resource for the next level.

Use global maze routing and Dijkstra's shortest path detailed routing to reroute failed connections and refine the solution.

Courtesy of Y.-W. Chang

Coarsening Stage

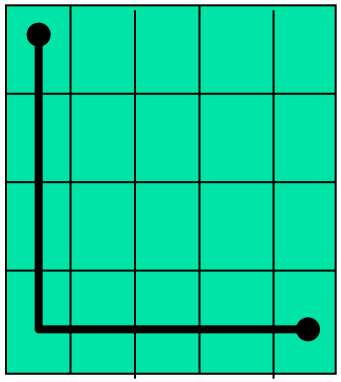
- Build MSTs for all nets and decompose them into two-pin connections.
- Route **local nets (connections)** from level 0.
  - Two-stage routing (global + detailed routing) for a local net.



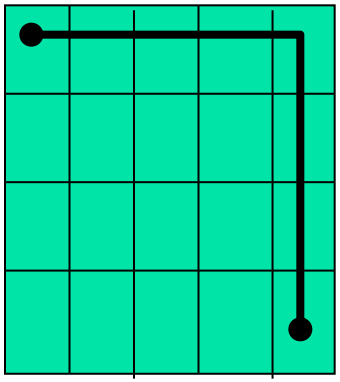
Courtesy of Y.-W. Chang

Global Routing

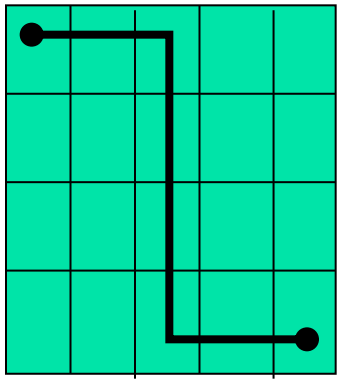
- Apply pattern routing for global routing
  - Use L-shaped and Z-shaped connections to route nets.
  - Has lower time complexity than maze routing.



Lower L-Shaped connection



Upper L-Shaped connection



Z-Shaped connection

Courtesy of Y.-W. Chang

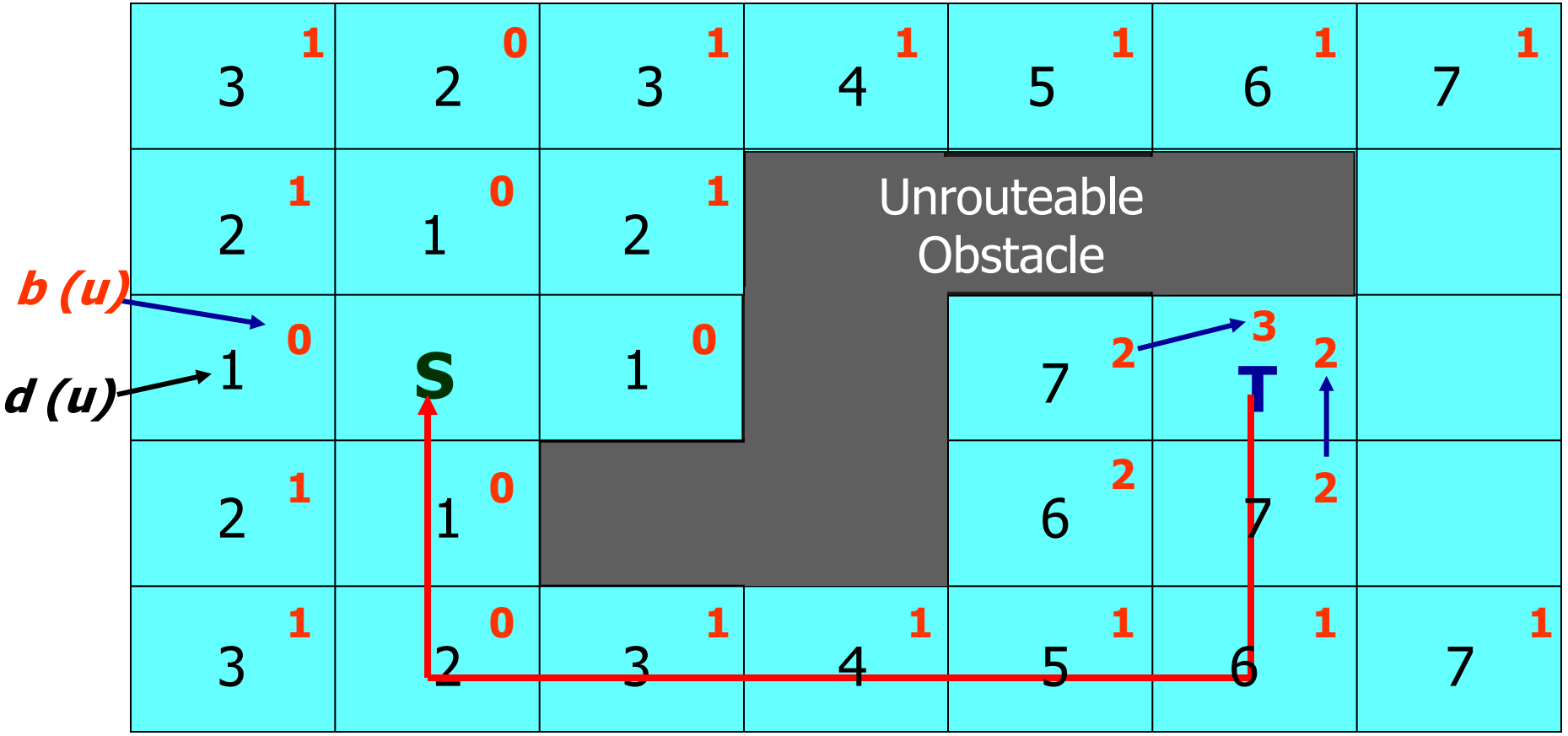
Detailed Routing

- Via minimization
  - Modify the maze router to minimize the number of bends.
- Local refinement
  - Apply general maze routing to improve the detailed routing results.
- Resource estimation
  - Update the edge weights of the routing graph after detailed routing.

Courtesy of Y.-W. Chang

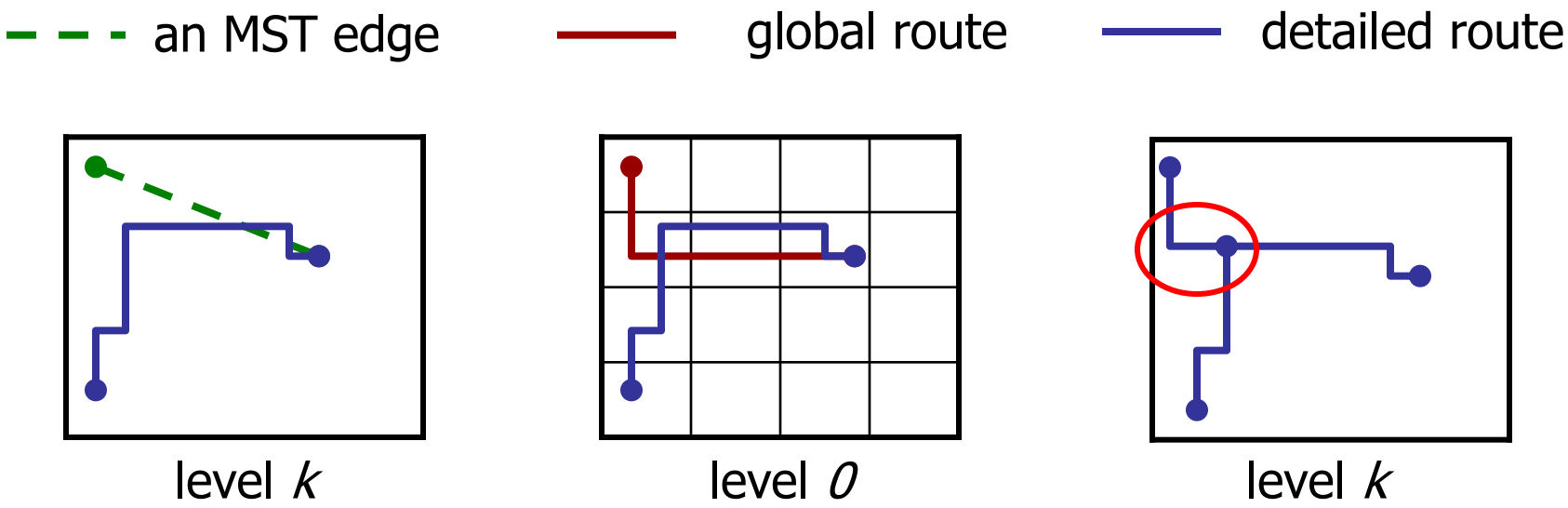
Via Minimization

Back Trace



Local Refinement

- Local refinement improves detailed routing results by merging two connections which are decomposed from the same net.



Courtesy of Y.-W. Chang

Resource Estimation

- Global routing cost is the summation of congestions of all routed edges.
- Define the congestion, C_e, of an edge e by

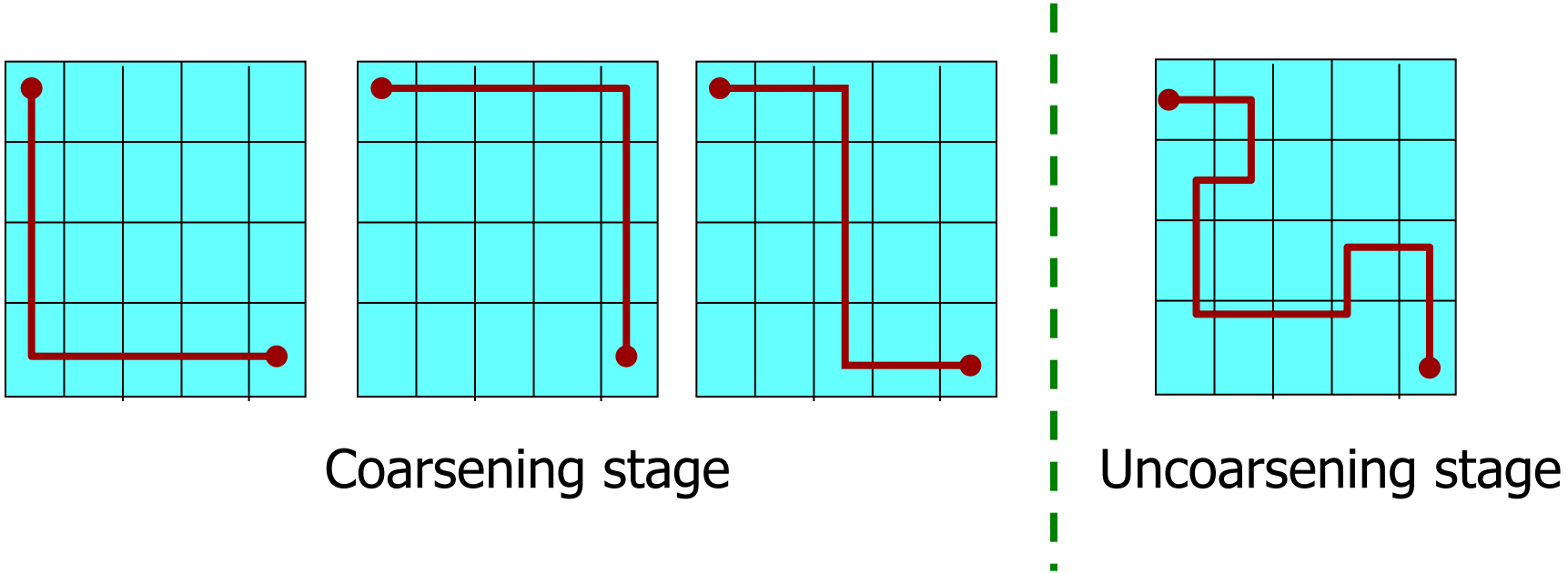
$$C_e = \frac{1}{2^{(p_e - d_e)}},$$

  where p_e and d_e are the capacity and density, respectively.
- Update the congestion of routed edges to guide the subsequent global routing.

Courtesy of Y.-W. Chang

Uncoarsening Global Routing

- Use maze routing.
- Iterative refinement of a failed net is stop when a route is found or several tries have been made.



Courtesy of Y.-W. Chang

Routing Comparisons

- 100% routing completion for all (11) benchmark circuits
  - Three-level routing: 0 completion (ISPD-2K)
  - Hierarchical routing: 2 completions (ICCAD-2001)
  - Previous multilevel routing: 2 completions (ICCAD-2001)
- Can complete routings using even fewer routing layers.

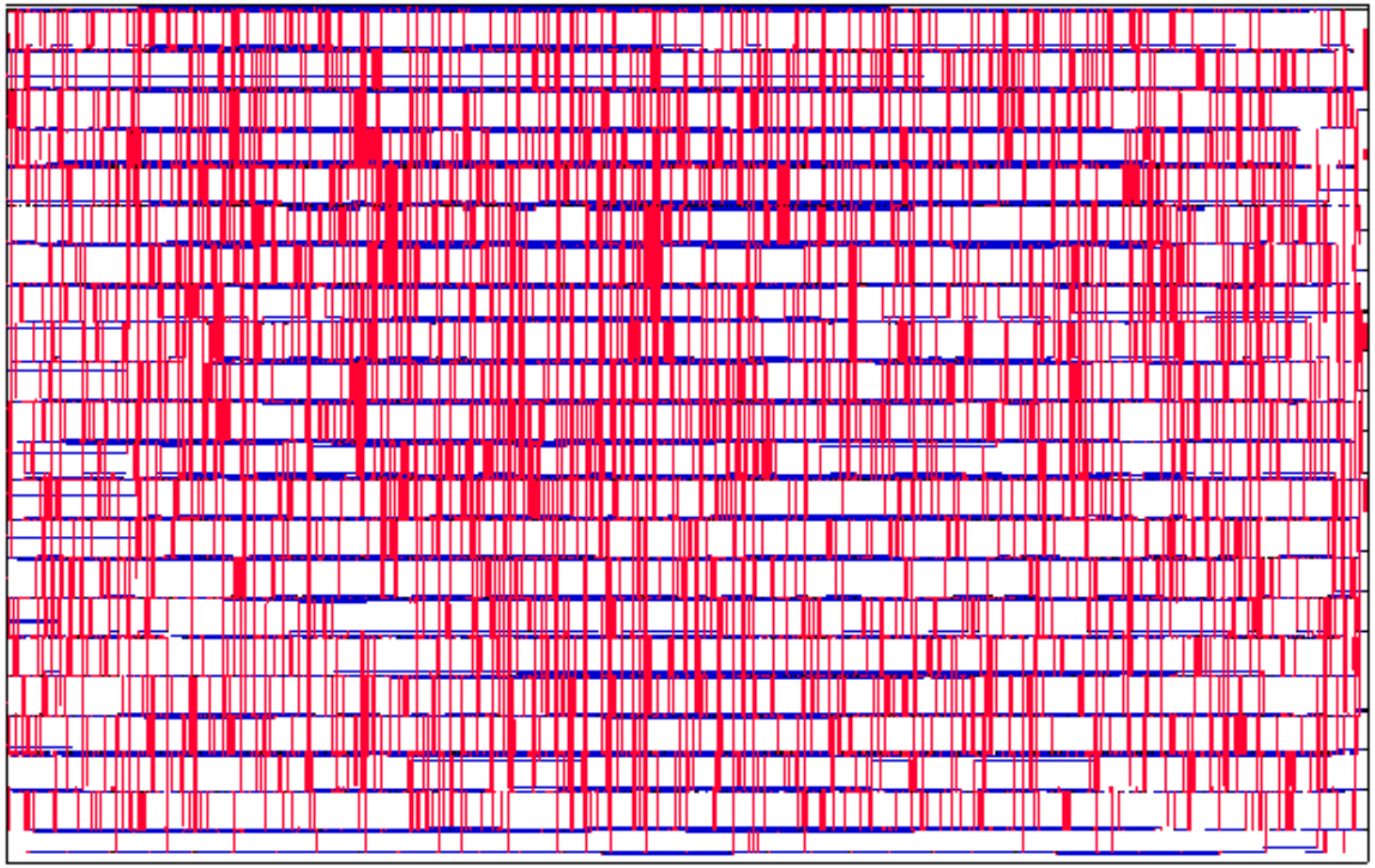
Ex.	#Layers	(A) Three-Level Routing			(B) Hierarchical Routing with Ripup and Replan			(C) Results of [9]			(D) Our Results		
		Time(s)	#Rtd. Nets	Cmp. Rates	Time(s)	#Rtd. Nets	Cmp. Rates	Time(s)	#Rtd. Nets	Cmp. Rates	Time(s)	#Rtd. Nets	Cmp. Rates
Mcc1	4	933.2	1499	88%	947.9	1600	94.5%	436.7	1683	99.4%	204.7	1694	100%
Mcc2	4	12333.6	5451	72.3%	10101.4	7161	95.6%	7644.8	7474	99.1%	7203.3	7541	100%
Struct	3	406.2	3530	99.4%	324.5	3551	100%	316.8	3551	100%	151.5	3551	100%
Prim1	3	239.1	2018	99.0%	353.0	2037	100%	350.2	2037	100%	165.4	2037	100%
Prim2	3	1331	8109	98.9%	2423.8	8194	100%	2488.4	8196	100%	788.2	8197	100%
S5378	3	430.2	2607	83.4%	57.9	2964	94.9%	54.0	2963	94.8%	10.9	3124	100%
S9234	3	355.2	2467	88.9%	40.7	2564	92.4%	41.0	2561	92.3%	7.7	2774	100%
S13207	3	1099.5	6118	87.5%	161.9	6540	93.5%	188.8	6574	94.0%	38.2	6995	100%
S15850	3	1469.1	7343	88.2%	426.1	7874	94.6%	403.4	7863	94.5%	57.5	8321	100%
s38417	3	3560.9	19090	90.8%	754.6	19596	93.2%	733.6	19636	93.3%	137.6	21035	100%
S38584	3	7086.5	25642	91.0%	1720	26461	93.9%	1721.6	26504	94.1%	316.7	28177	100%
avg.				89.8%			95.7%			96.5%			100%

Table 3: Comparison among (A) the three-level routing [10], (B) the hierarchical routing [9], (C) the multilevel routing [9], and (D) our multilevel routing. Note: (A),(B),(C) ran on a 440 Mhz Sun Ultra-5 with 384 MB memory, (D) ran on a 450Mhz Sun Sparc Ultra-60 with 2GB MB.

Courtesy of Y.-W. Chang

Routing Solution for Prim2

- 0.18um technology, pitch = 1 um, 8109 nets.
- Two layers, 100% routing completion.



Courtesy of Y.-W. Chang

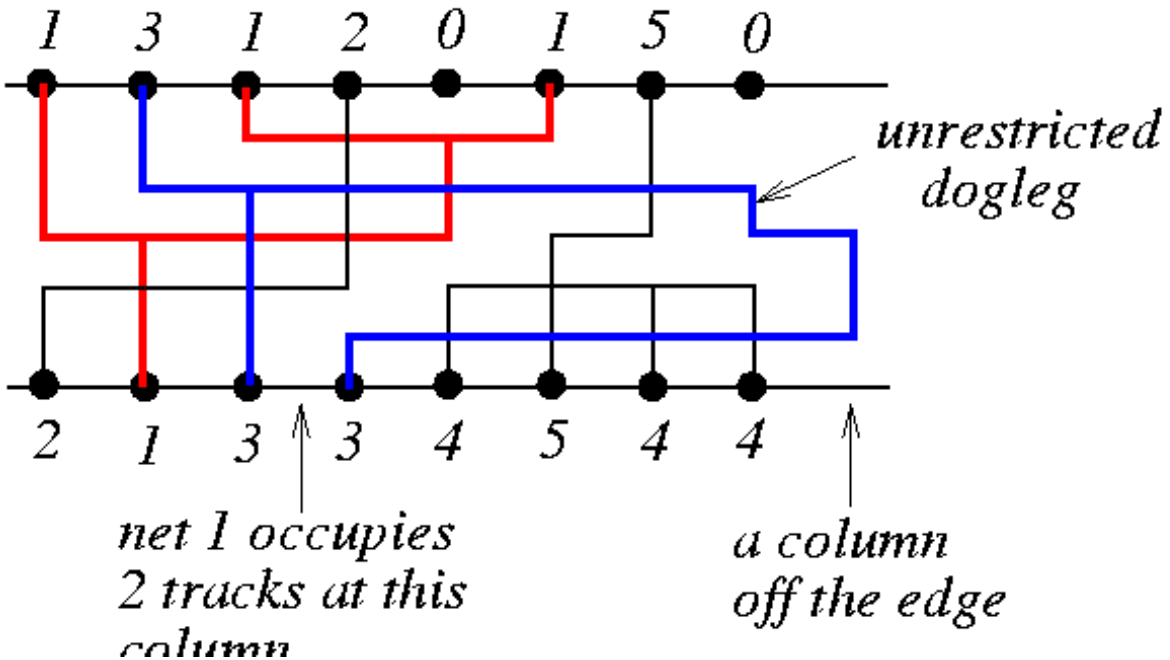
Summary: Detailed Routing

- Channel routing is considered a well solved problem.
- Hierarchical and multilevel are keys to handle large-scale routing problems.
- Routing frameworks
  - Λ-shaped routing: ICCAD-02 (TCAD-03); ASP-DAC-05 (TCAD-07)
  - V-shaped routing: ASP-DAC-06
  - Two-pass bottom-up routing: DAC-06 (TCAD-08)
- Routing considerations for nanometer technology
  - Noise (crosstalk) constraints
  - Buffer insertion for timing optimization
  - Additional design rules: antenna effect, redundant via, OPC (optical proximity correction), CMP (chemical mechanical polishing), double patterning, e-beam
  - Electromigration constraints

Courtesy of Y.-W. Chang

Appendix A:

Other Channel Routing Algorithms



Courtesy of Y.-W. Chang

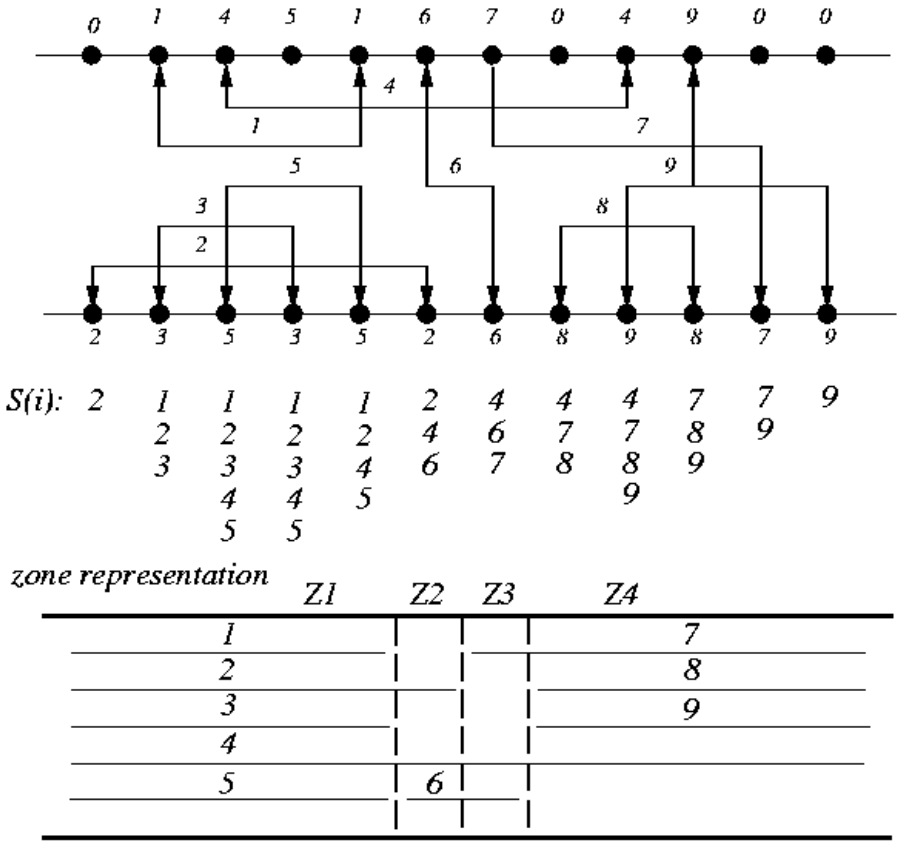
Yoshimura-Kuh (YK) Algorithm

- Yoshimura & Kuh, "Efficient algorithms for channel routing," IEEE TCAD, Jan. 1982.
- YK algorithm considers both HCG and VCG.
- Nets are assigned to minimize the effect of vertical constraint chains in VCG.
- Does not allow **"unrestricted"** doglegs and cannot handle vertical constraint cycles.
- Algorithm consists of two major steps:
  - Zone representation of horizontal segments.
  - Merging of nets.
- The two steps are carried out to minimize vertical constraints and track assignment.
- Same idea can be extended to three-layer channel routing (Chen & Liu, IEEE TCAD, 1984)

Courtesy of Y.-W. Chang

Zone Representation of Horizontal Segments

- Zones are maximal clique in the interval graph of horizontal net segments.
- $S(i)$: set of nets whose horizontal segments intersect Column i.
- Zone numbers are assigned to the columns at which $S(i)$ is maximum.

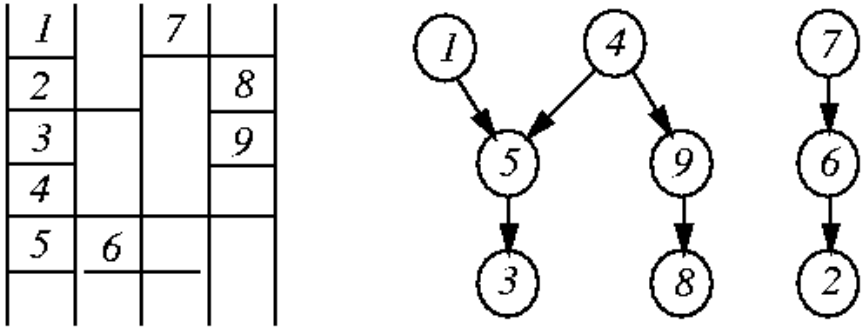


Courtesy of Y.-W. Chang

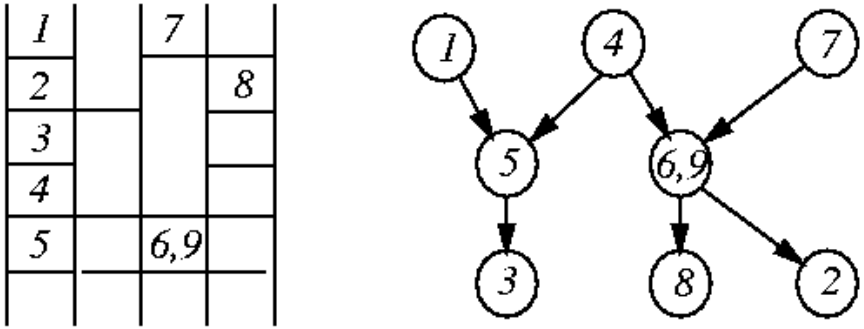
Merging of Nets

- Two nets n_i and n_j can be merged if
  - there is no edge between v_i and v_j in HCG,
  - no directed path exists between v_i and v_j in VCG.

nets 6 and 9 can be merged:

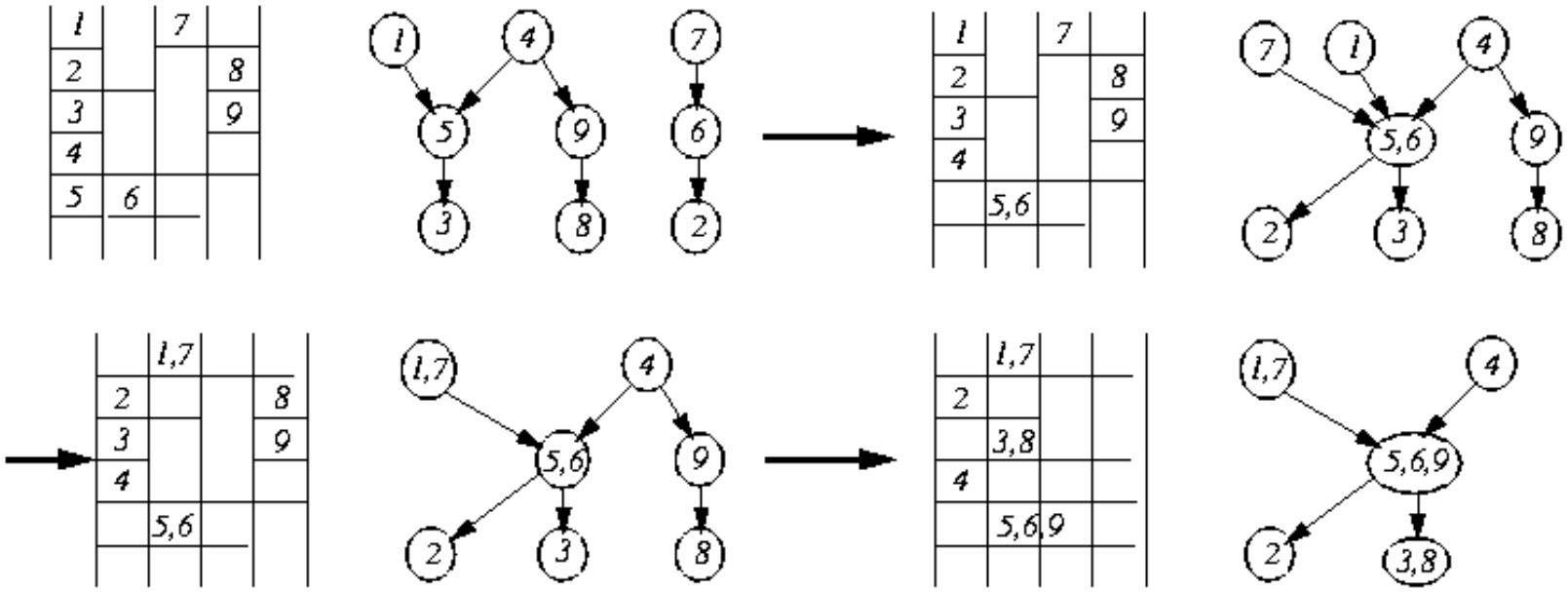


updated graph and zone representation:



Courtesy of Y.-W. Chang

Zone Processing

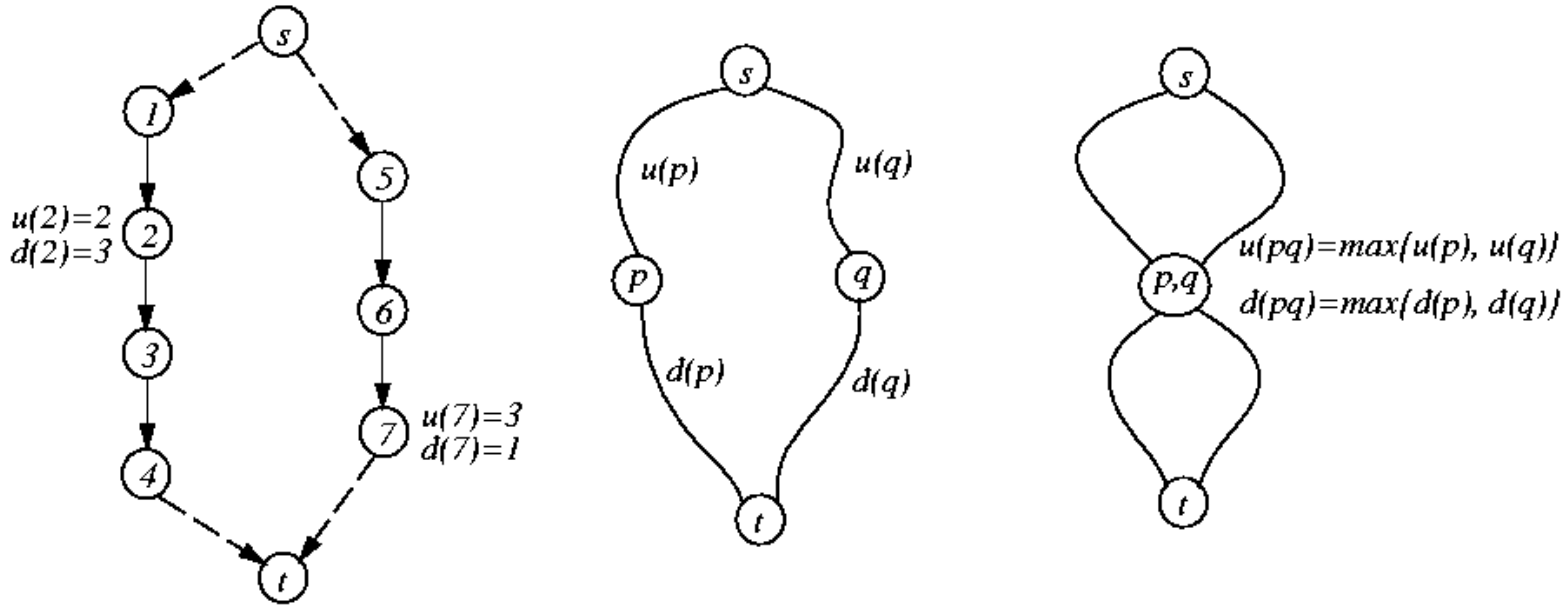


- Track 1 (Track 2): net (1, 7) or net 4; track 3: net (5, 6, 9); track 4 (track 5): net 2 or net (3, 8)

Courtesy of Y.-W. Chang

Minimizing the Longest Path

- Merge 2 nodes so as to minimize the increase of the longest path length in the VCG.
- Heuristic rule to select nets to merge sequentially.



$$increase\ in\ longest\ path = max\{u(p), u(q)\} + max\{d(p), d(q)\} - max\{u(p)+d(p), u(q)+d(q)\}$$

Courtesy of Y.-W. Chang

Algorithm for Merging Nets

Algorithm: Net_Merging(z_s, z_t)
z_s: Leftmost zone;
z_t: rightmost zone;

1 **begin**
2 $L \leftarrow \{\}$;
3 $z_s \leftarrow$ leftmost zone; $z_t \leftarrow$ rightmost zone;
4 **for** $z \leftarrow z_s$ **to** z_t **do**
5 $L \leftarrow L$ + {nets which terminate at or before zone z};
6 $R \leftarrow$ {nets which begin at zone z+1};
7 Merge L and R so as to minimize the increase of the longest path in the vertical constraint graph;
8 $L \leftarrow L$ - $\{n_1, n_2, \ldots, n_j\}$, where $\{n_1, n_2, \ldots, n_j\}$ are the nets merged at Step 7;
9 **end**

Courtesy of Y.-W. Chang

Implementation

- Procedure to select 2 nodes for merging
  - For $m \in LEFT$ (set of nodes on the left path)

$$f(m) = C_\infty \times \{u(m) + d(m)\} + \max\{u(m), d(m)\}, C_\infty \gg 1$$

  - For $n \in RIGHT$, $m \in LEFT$

$$g(n,m) = C_\infty \times h(n,m) - \{\sqrt{u(m) \times u(n)} + \sqrt{d(m) \times d(n)}\}, where$$
$$h(n,m) = \max\{u(n), u(m)\} + \max\{d(n), d(m)\} - \max\{u(n) + d(n), u(m) + d(m)\}.$$

- Two steps
  - Find m^* ∈ *LEFT* which maximizes $f(m)$
  - Find n^* ∈ *RIGHT* which minimizes $g(n, m^*)$

$$f(m^*) = C_\infty \times \underbrace{\{u(m^*) + d(m^*)\}}_{lies\ on\ a\ longest\ path} + \underbrace{\max\{u(m^*), d(m^*)\}}_{farthest\ away\ from\ s\ or\ t}$$

$$g(n^*, m^*) = \underbrace{C_\infty \times h(n^*, m^*)}_{min\ increase\ in\ longest\ path\ length} - \underbrace{\{\sqrt{u(m) \times u(n)} + \sqrt{d(m) \times d(n)}\}}_{\frac{u(m^*)}{d(m^*)} \approx \frac{u(n^*)}{d(n^*)}}$$

Courtesy of Y.-W. Chang

Algorithm for the Implementation

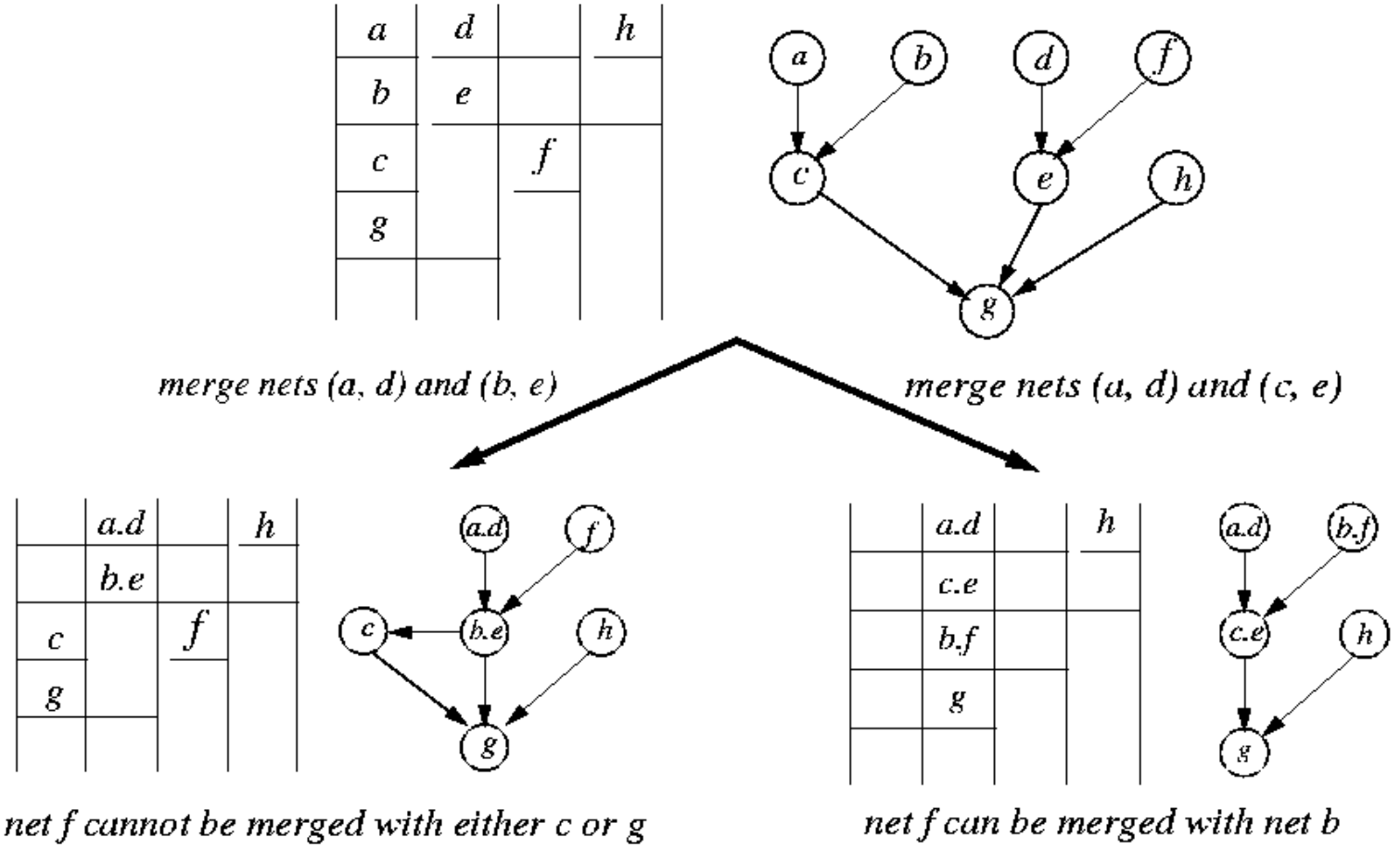
Algorithm: Merge_Implementation(*LEFT*, *RIGHT*)
LEFT: left *s*-to-*t* path;
RIGHT: right *s*-to-*t* path;

1 **begin**
2 **while** *LEFT* $\neq \varnothing$ **do**
3 Among *LEFT*, find m^* which maximizes $f(m)$;
4 Among *RIGHT*, find n^* which minimizes $g(n, m^*)$, and which is neither ancestor nor successor of m^*;
5 Merge n^* and m^*;
6 Remove m^* and n^* from *LEFT* and *RIGHT*, respectively;
7 **end**

Courtesy of Y.-W. Chang

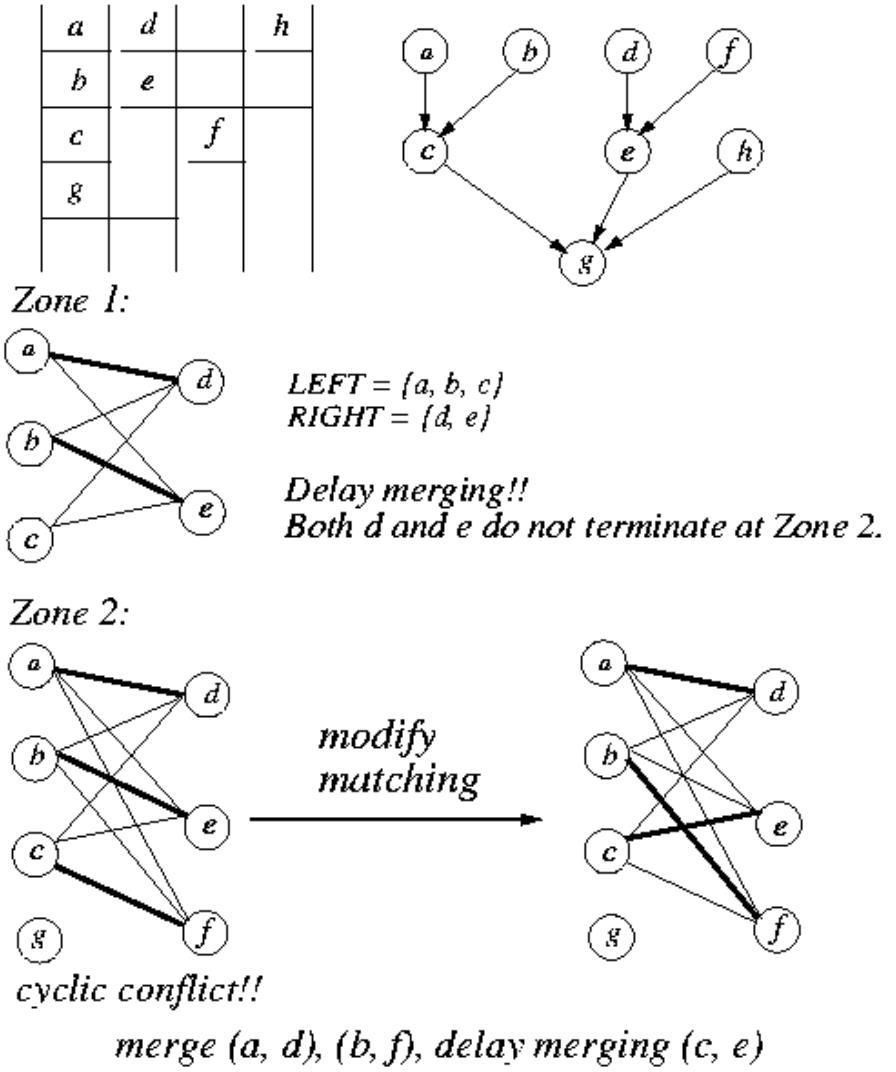
Merging Considerations

- A merging of 2 nodes may block subsequent mergings.



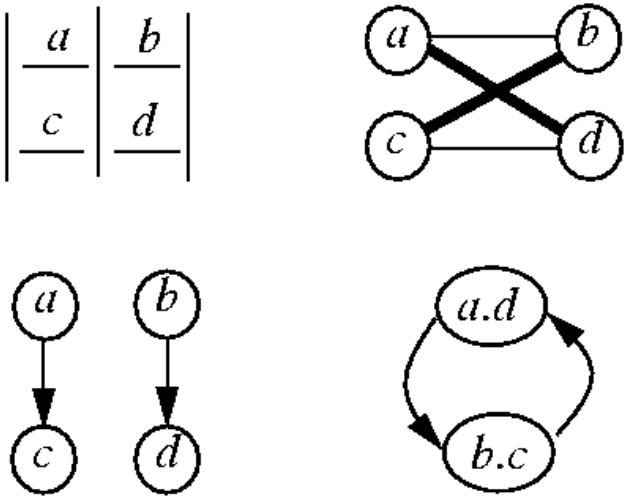
Second Approach Based on Matching

- Global merging of nets using algorithms for maximum cardinality matching.
- Delay net merging.



Cyclic Conflicts

- Simultaneous merging can produce cyclic conflicts.



- How to modify a matching with cyclic conflicts to get a feasible matching?

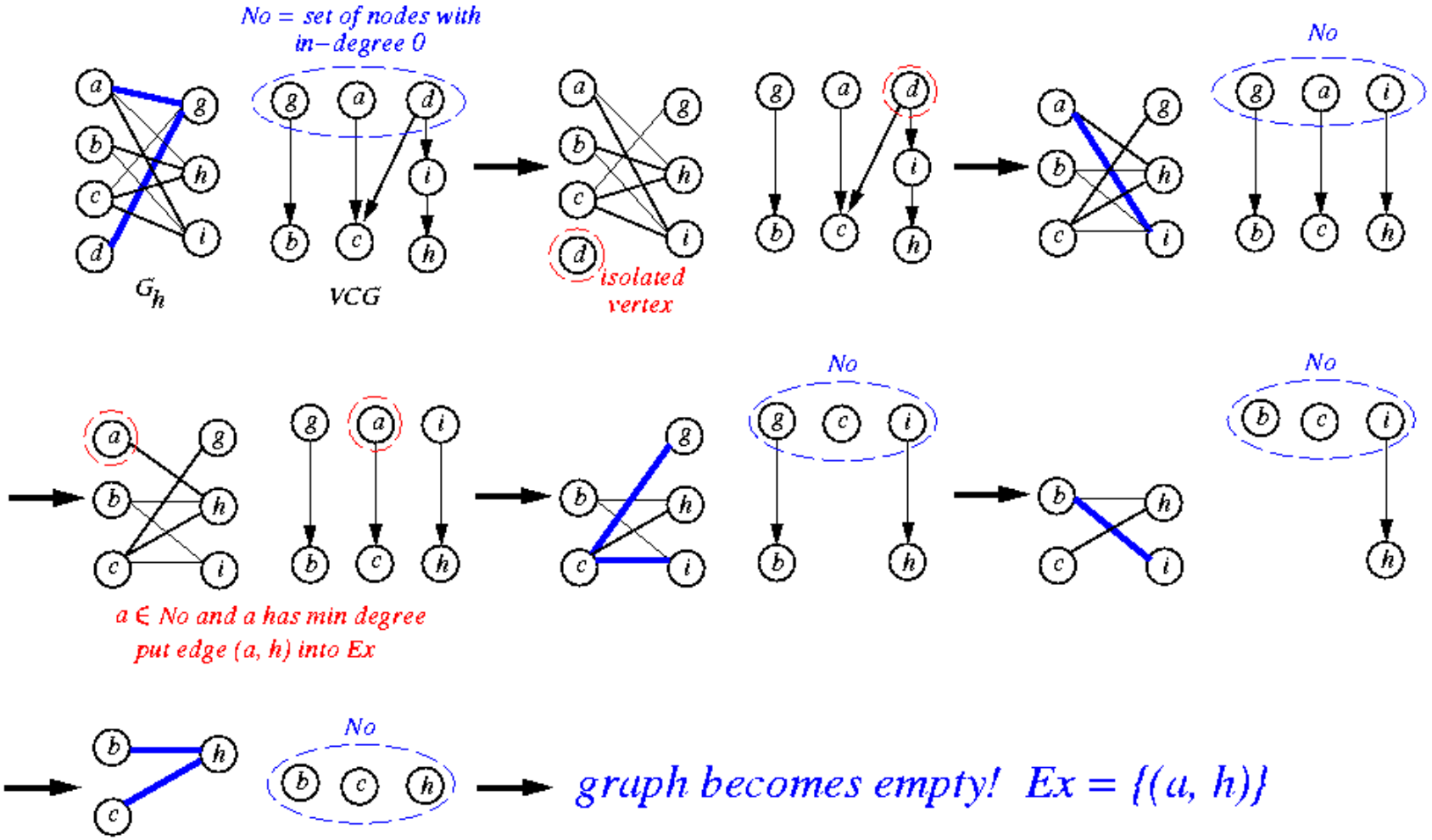
Courtesy of Y.-W. Chang

Feasible Matching

- Bipartite graph $G_h = (N, E_h)$: G_h $\xrightarrow{algorithm}$ E_x = a set of edges in G_h.
- The merging corresponding to any matching on G_h is feasible $\Leftrightarrow$ $E_x = \varnothing$.
- The merging corresponding to any matching on $G_r = (N, E_h - E_x)$ is feasible.
- Procedures to get a feasible matching
  - Find a matching in G_h.
  - Apply an algorithm to the graph $G_m = (N, M)$ (M: set of edges in the matching)
  - If $E_x = \varnothing$ then the matching is feasible!
  - Else apply an algorithm to G_h and obtain E'_x; compute a new matching in $(N, E_h - E'_x)$.

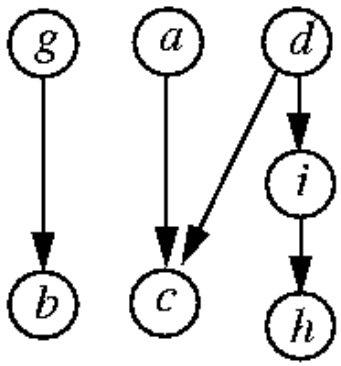
Courtesy of Y.-W. Chang

Identifing E_x

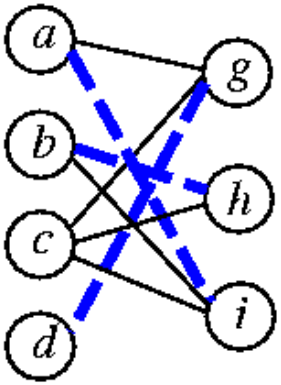


Feasibility Condition

- The merging corresponding to any matching on $G_r = (N, E_h - E_x)$ is feasible (no cycle in VCG).

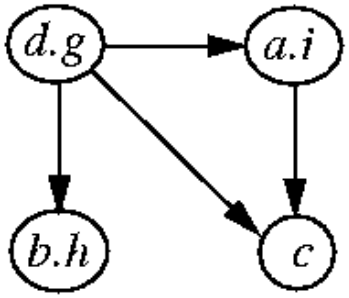


original VCG



arbitrary matching

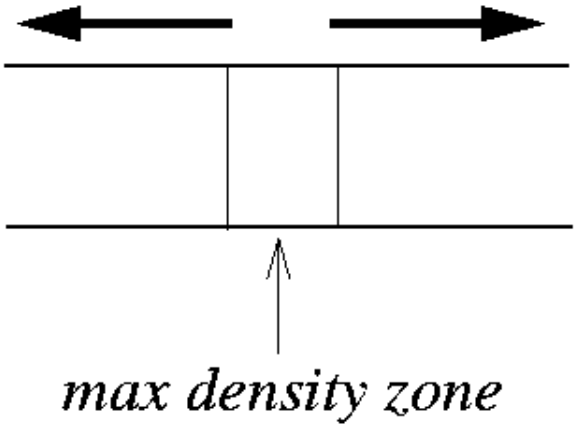
$Ex = \{(a, h)\}$



no cycle in VCG

Comments on the YK Algorithm

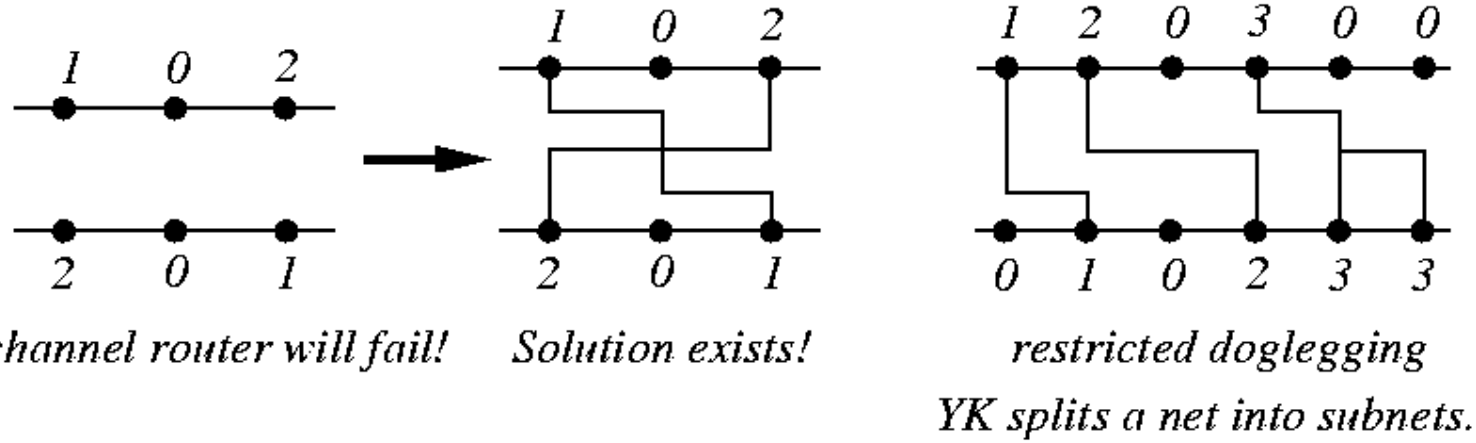
- Use a process "merging of subnets" to avoid unnecessary introduction of dogleg.
  - Subnet *i* and subnet *j* can be merged only if merging subnet *i* and subnet *j* will not increase the longest path length passing through node *i* and node *j* on VCG.
- Need not start at Zone 1. Can obtain better results by starting at the maximum density zone.
  - Chan, "A new channel routing algorithm," *CIT VLSI Design Conf.*, 1983.



Courtesy of Y.-W. Chang

Restricted vs. Unrestricted Doglegging

- **Unrestricted doglegging:** Allow a dogleg even at a position where there is no pin.
- **Restricted doglegging:** Allow a dogleg only at a position where there is a pin belonging to that net.
- The YK channel router does not allow unrestricted doglegging.



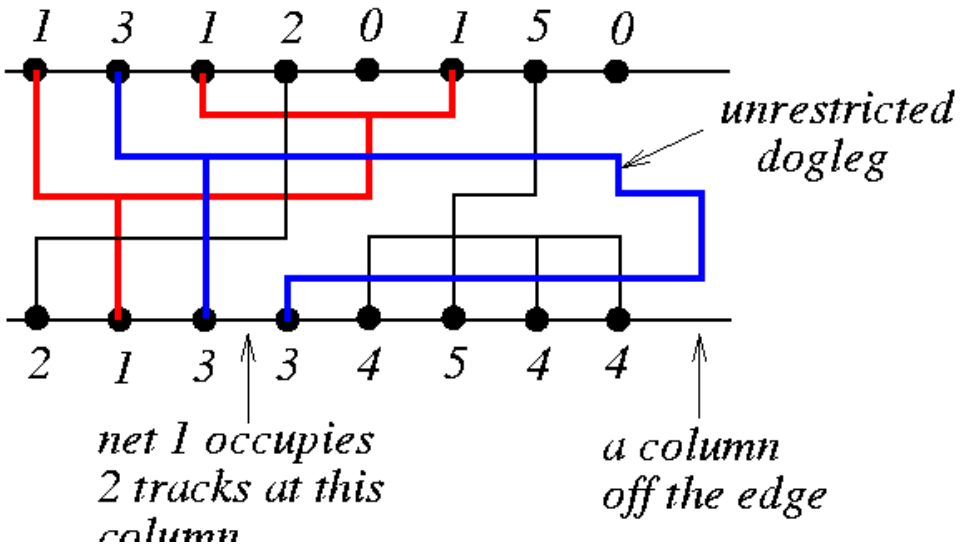
YK channel router will fail! *Solution exists!* *restricted doglegging*

YK splits a net into subnets.

Courtesy of Y.-W. Chang

Greedy Channel Router

- Rivest & Fiduccia, "A greedy channel router," *DAC-82*, (*IEEE TCAD*, May 1983).
- Always succeed (even if cyclic conflict is present)
- Allows unrestricted dogleg
- Allows a net to occupy more than 1 track at a given column.
- May use a few columns off the edge.



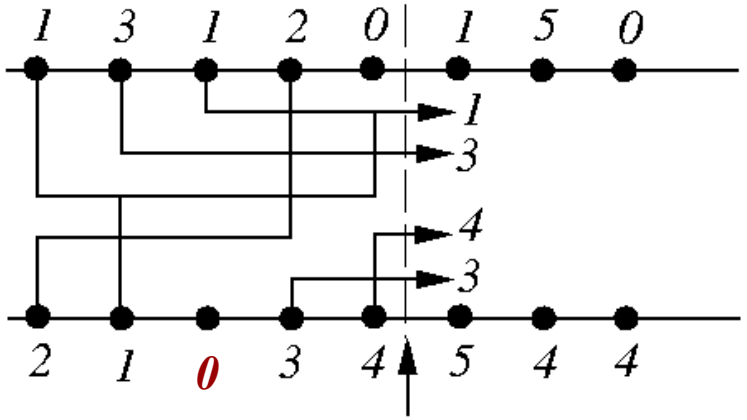
Courtesy of Y.-W. Chang

Overview of Greedy Router

- Left-to-right, column-by-column scan.

```
1 begin
2 c ← 0;
3 while (not done) do
4     c ← c + 1;
5     Complete wiring at column c;
6 end
```

- In general, a net may be
  1. empty (net 5)
  2. unsplit (nets 1, 4)
  3. split (net 3)
  4. completed (net 2)



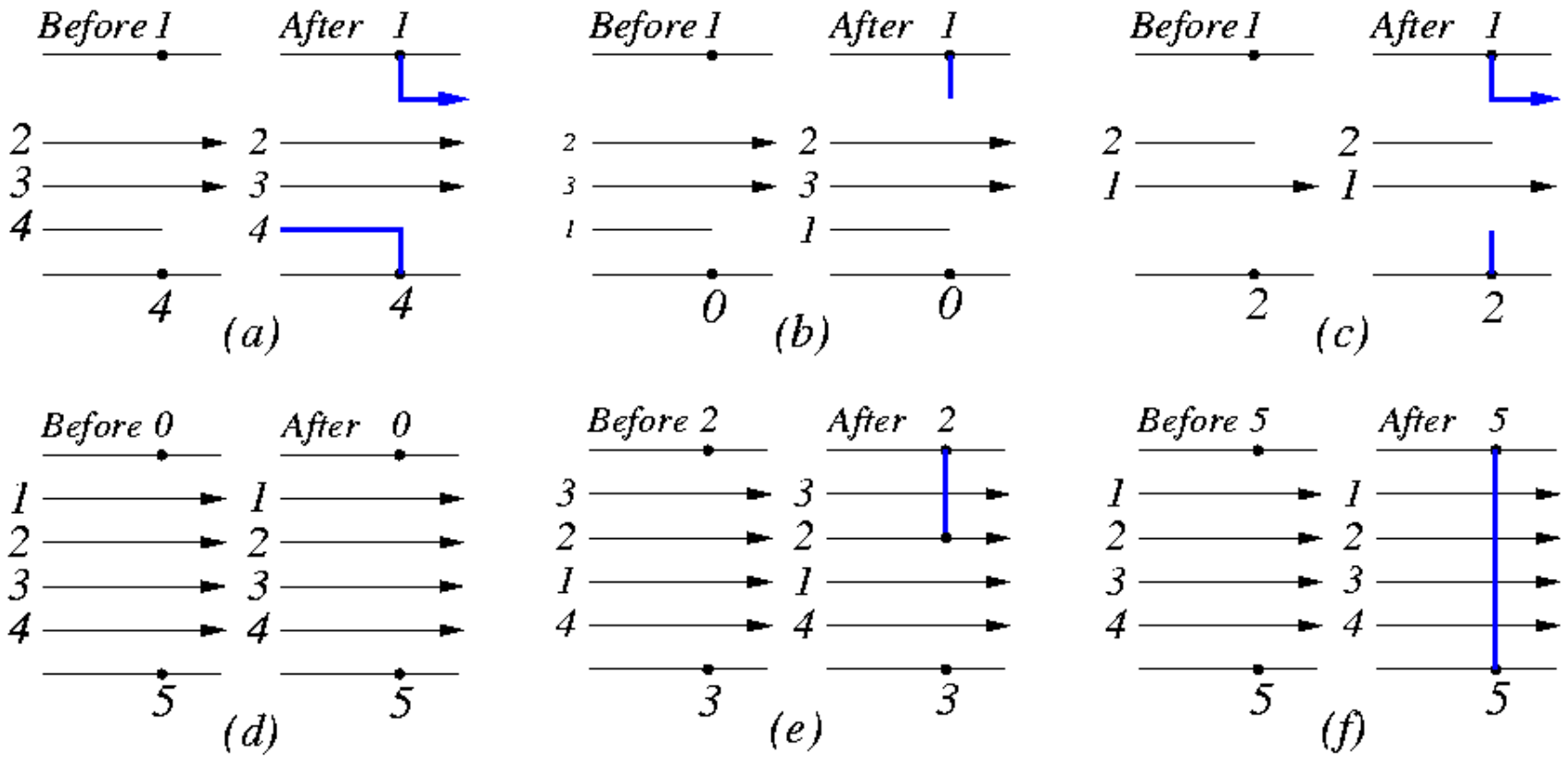
Courtesy of Y.-W. Chang

Greedy Heuristics

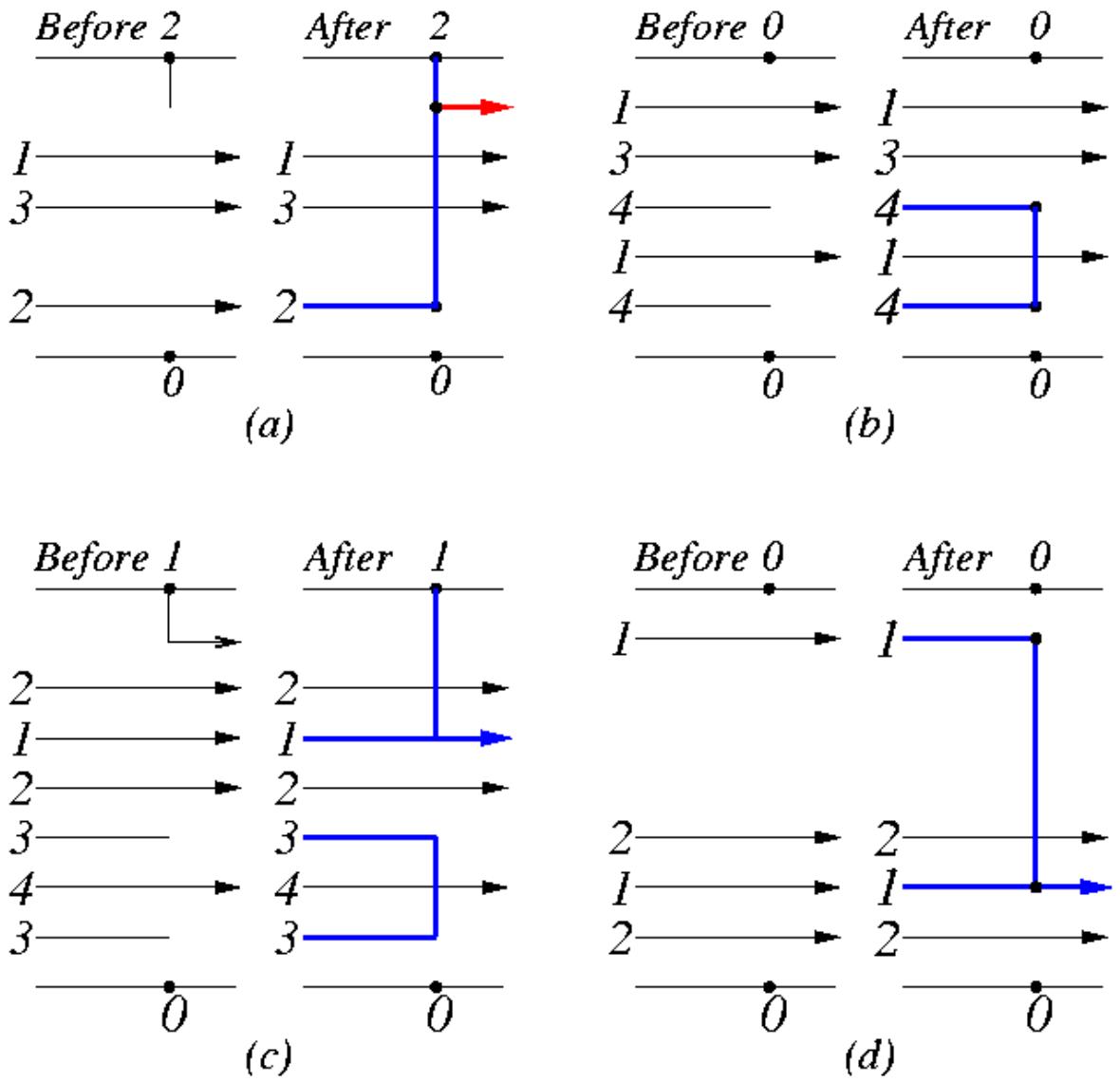
- At each column, the greedy router tries to maximize the utility of the wiring produced:
  - A: Make minimal feasible top/bottom connections;
  - B: Collapse split nets;
  - C: Move split nets closer to one another;
  - D: Raise rising nets/Lower falling nets;
  - E: Widen channel when necessary;
  - F: Extend to next column.

Courtesy of Y.-W. Chang

A: Make Minimal Feasible Top/Bottom Connections



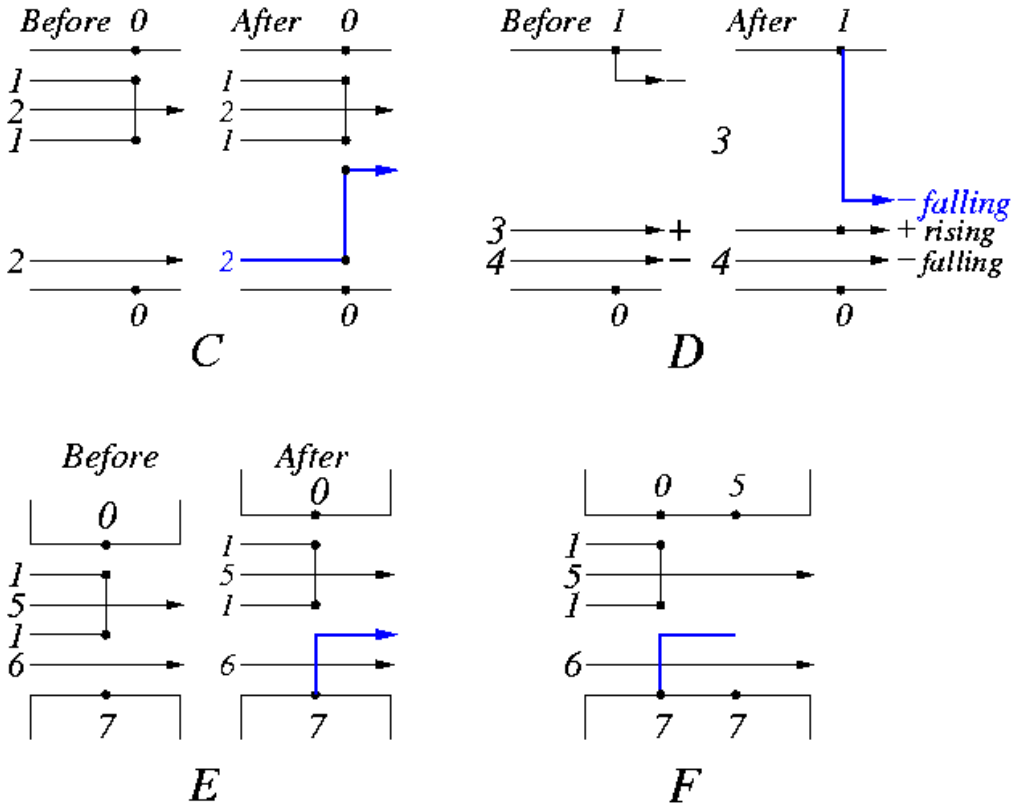
B: Collapse Split Nets



Courtesy of Y.-W. Chang

Heuristics: C, D, E, and F

- C: Move split nets closer to one another;
- D: Raise rising nets/Lower falling nets;
- E: Widen channel when necessary;
- F: Extend to next column.



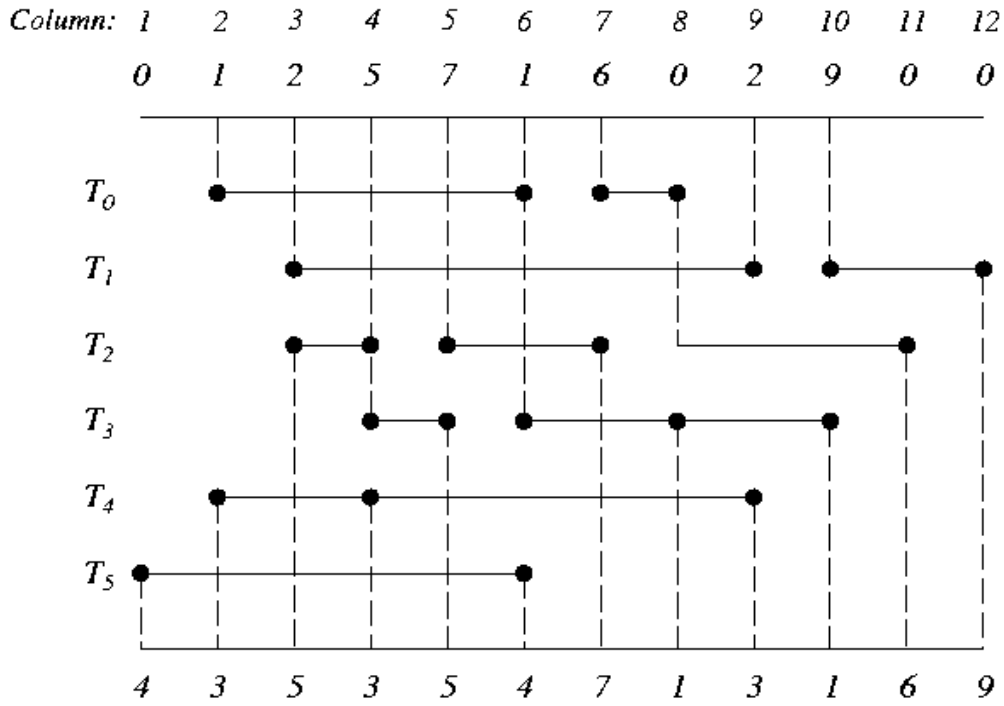
Courtesy of Y.-W. Chang

Parameters to Greedy Router

- Initial-channel-width: *icw*
- Minimum-jog-length: *mjl*
- Steady-net-constant: *snc* (window size in terms of # of columns; determines # of times a multipin net changes tracks)
- Usually start *icw* as *d*, the density.
- *mjl* controls the number of vias, use a large *mjl* for fewer vias.
- *snc* also controls # of vias. Typically, *snc* = 10.

Courtesy of Y.-W. Chang

Greedy Routing Example



- C_3: Connect pin 5 to T_3 → Jog net 5 from T_3 to T_2 (since net 5 is rising).
- C_4: Connect pin 5 to T_2 → Jog net 5 from T_2 to T_3 (since net 5 is falling).
- C 6: Connect pin 1 to T_0 → Jog net 1 from T_0 to T_3 (since net 1 is falling).
- C 7: Connect pin 7 to T_5 → Merge tracks T_2 and T_5 (last pin 7).
- C 8: Connect pin 1 to T_5 → Jog net 6 from T_0 to T_2 and net 1 from T_5 to T_3.
- … … … … … …

Courtesy of Y.-W. Chang

Comparison of Two-Layer Channel Routers

	Left-edge	Dogleg	YK	Greedy
Layer assignment	reserved	reserved	reserved	reserved
Dogleg	not allowed	allowed	allowed (restricted)	allowed
Vertical constraint	not allowed	allowed	allowed	allowed
Cyclic constraint	not allowed	not allowed	not allowed	allowed

Courtesy of Y.-W. Chang

Comparison of Two-Layer Channel Routers

- Comparison using the benchmark example: **Deutsch's "difficult example."**
- Channel density: 19

Routers	Tracks	Vias	wire length
Left-edge	31	290	6526
Dogleg	21	346	5331
YK	20	403	5381
Greedy	20	329	5078
Hierarchical	19	336	5023
YACR2	19	287	5020

Courtesy of Y.-W. Chang

Robust Channel Router

- Yoeli, “A robust channel router,” IEEE TCAD, 1991.
- Alternates between top and bottom tracks until the center is reached.
- The working side is called the *current side*.
- Net weights are used to guide the assignment of segments in a track, which
  - favor nets that contribute to the channel density;
  - favor nets with terminals at the current side;
  - penalize nets whose routing at the current side would cause vertical constraint violations.
- Allows unrestricted doglegs by rip-up and re-route.

Courtesy of Y.-W. Chang

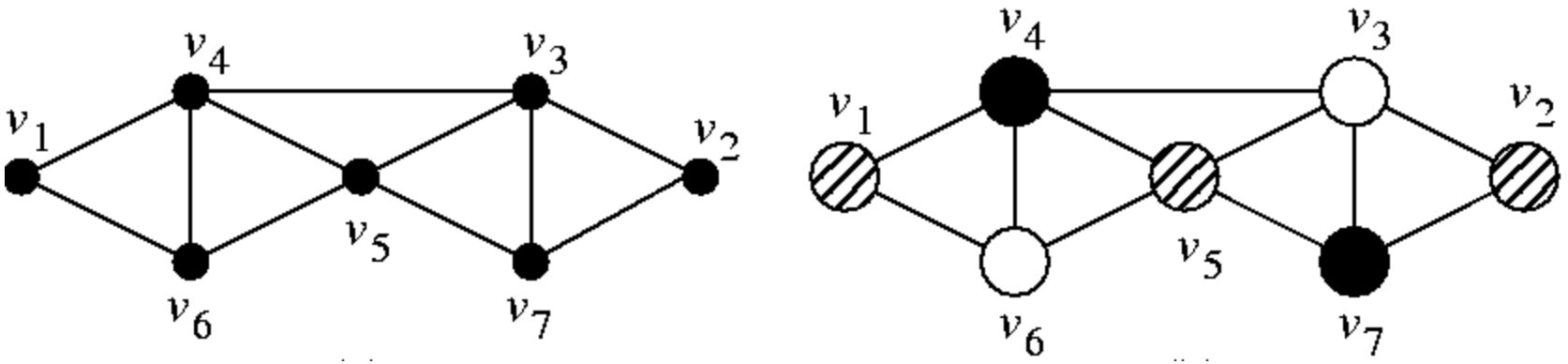
Robust Channel Router

- Select the set of nets for the current side by solving the maximum weighted independent set problem for interval graphs.
  - NP-complete for general graphs, but can be solved efficiently for interval graphs using dynamic programming.
- Main ideas:
  - The interval for net i is denoted by $[x_{i_{min}}, x_{i_{max}}]$; its weight is w_i.
  - Process channel from left to right column; the optimal cost for position c is denoted by total[c];
  - A net n with a rightmost terminal at position c is taken into the solution if total[$c - 1$] < w_n + total[$x_{n_{min}} - 1$].
- Can apply maze routers to fix local congestion or to post-process the results. (Why not apply maze routers to channel routing directly??)

Courtesy of Y.-W. Chang

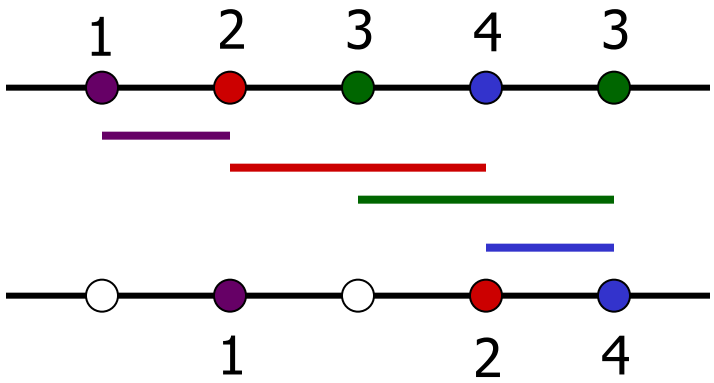
Interval Graphs

- There is a vertex for each interval.
- Vertices corresponding to overlapping intervals are connected by an edge.
- Solving the track assignment problem is equivalent to finding a **minimal vertex coloring** of the graph.



Courtesy of Y.-W. Chang

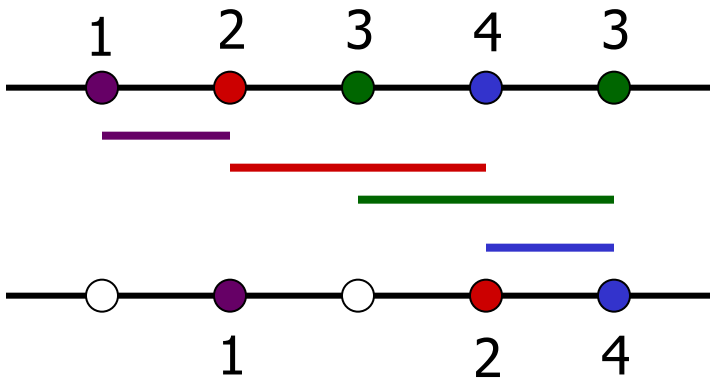
Weight Computation



d(1) = 1
d(2) = 2
d(3) = 2
d(4) = 3 (nets 2, 3, 4)
d(5) = 2

- Computation of the weight w_i for net *i*:
  1. favor nets that contribute to the channel density: add a large *B* to w_i.
  2. favor nets with current side terminals at column *x*: add d(*x*) to w_i.
  3. penalize nets whose routing at the current side would cause vertical constraint violations: subtract *K*d(*x*) from w_i, *K* = 5 ~ 10.
  - Assume *B* = 1000 and *K* = 5 in the 1st iteration (top side):
    - w_1 = (0) + (1) + (-5 * 2) = -9
    - Net 1 does not contribute to the channel density
    - One net 1 terminal on the top
    - Routing net 1 causes a vertical constraint from net 2 at column 2 whose density is 2

Weight Computation (cont'd)

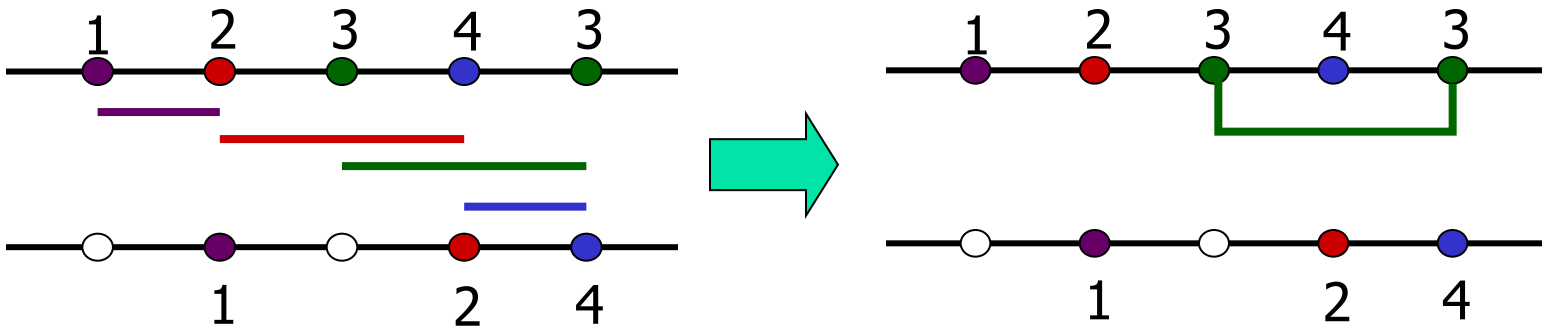


d(1) = 1
d(2) = 2
d(3) = 2
d(4) = 3 (nets 2, 3, 4)
d(5) = 2

- Computation of the weight w_i for net *i*:
  1. favor nets that contribute to the channel density: add a large *B* to w_i.
  2. favor nets with current side terminals at column *x*: add d(*x*) to w_i.
  3. penalize nets whose routing at the current side would cause vertical constraint violations: subtract *K*d(*x*) from w_i, *K* = 5 ~ 10.
  - Assume *B* = 1000 and *K* = 5 in the 1st iteration (top side):
    - w_1 = (0) + (1) + (-5 * 2) = -9
    - w_2 = (1000) + (2) + (-5 * 3) = 987
    - w_3 = (1000) + (2+2) + (0) = 1004
    - w_4 = (1000) + (3) + (-5 * 2) = 993

Courtesy of Y.-W. Chang

Top-Row Net Selection



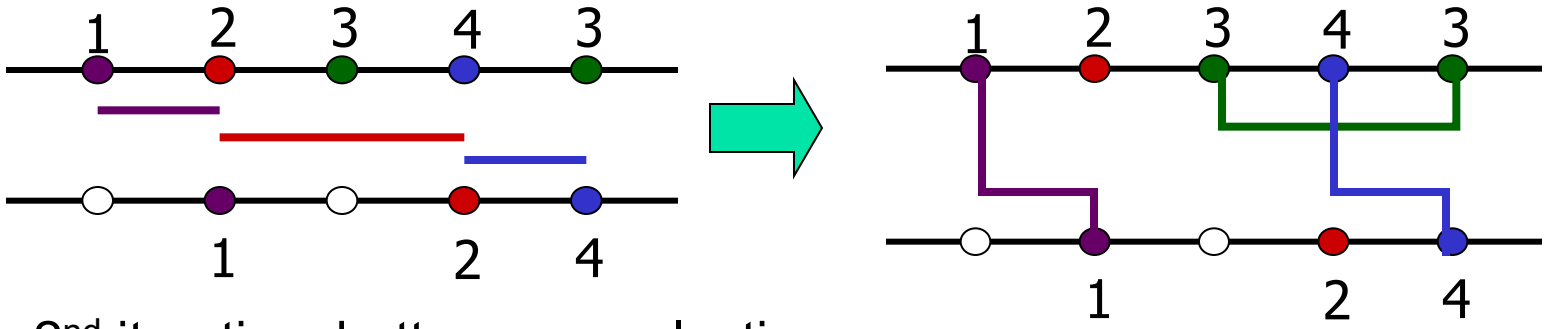
- $w_1 = -9$, $w_2 = 987$, $w_3 = 1004$, $w_4 = 993$.
- A net *n* with a rightmost terminal at position *c* is taken into the solution if: total[$c - 1$] < w_n + total[$x_{n_{min}} - 1$].

total[1] = 0	selected_net[1] = 0
total[2] = max(0, 0-9) = 0	selected_net[2] = 0
total[3] = 0	selected_net[3] = 0
total[4] = max(0, w_2+total[1]) = 987	selected_net[4] = 2
total[5] = max(987, 0+1004, 0+993) = 1004	selected_net[5] = 3

- Select nets backwards from right to left and with no horizontal constraints: Only net 3 is selected for the top row. (Net 2 is not selected since it overlaps with net 3.)

Courtesy of Y.-W. Chang

Bottom-Row Net Selection



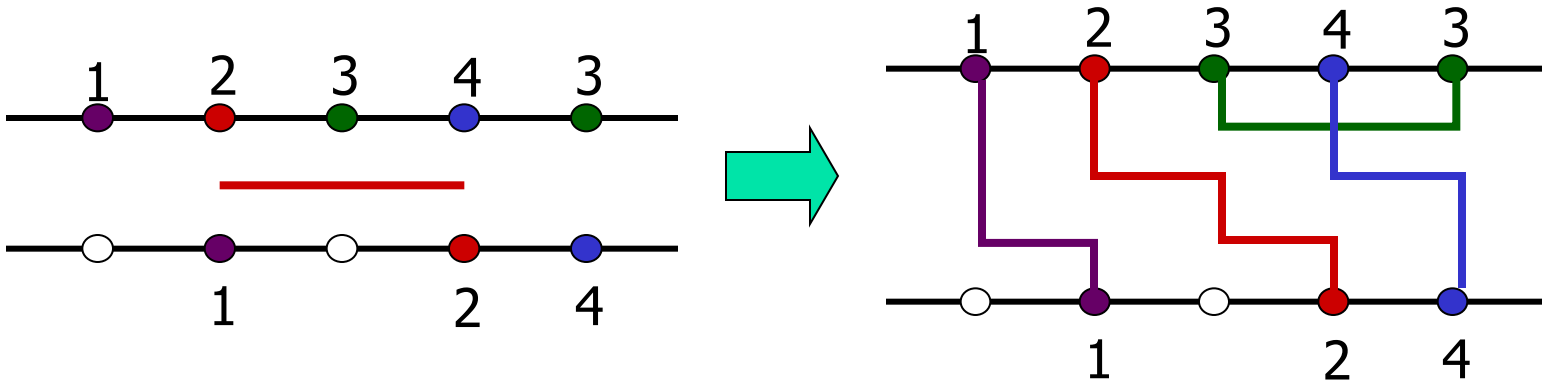
- 2nd iteration: bottom-row selection
  - w_1 = (1000) + (2) + (0) = 1002
  - w_2 = (1000) + (2) + (-5 * 2) = 992
  - w_4 = (1000) + (1) + (-5 * 2) = 991

total[1] = 0	selected_net[1] = 0
total[2] = max(0, 0+1002) = 1002	selected_net[2] = 1
total[3] = 1002	selected_net[3] = 0
total[4] = max(1002, 0+992) = 1002	selected_net[4] = 0
total[5] = max(1002, 1002+991) = 1993	selected_net[5] = 4

- Nets 4 and 1are selected for the bottom row.

Courtesy of Y.-W. Chang

Maze Routing + Rip-up & Re-route



- 3rd iteration
  - Routing net 2 in the middle row leads to an infeasible solution.
  - Apply maze routing and rip-up and re-route nets 2 and 4 to fix the solution.

Courtesy of Y.-W. Chang

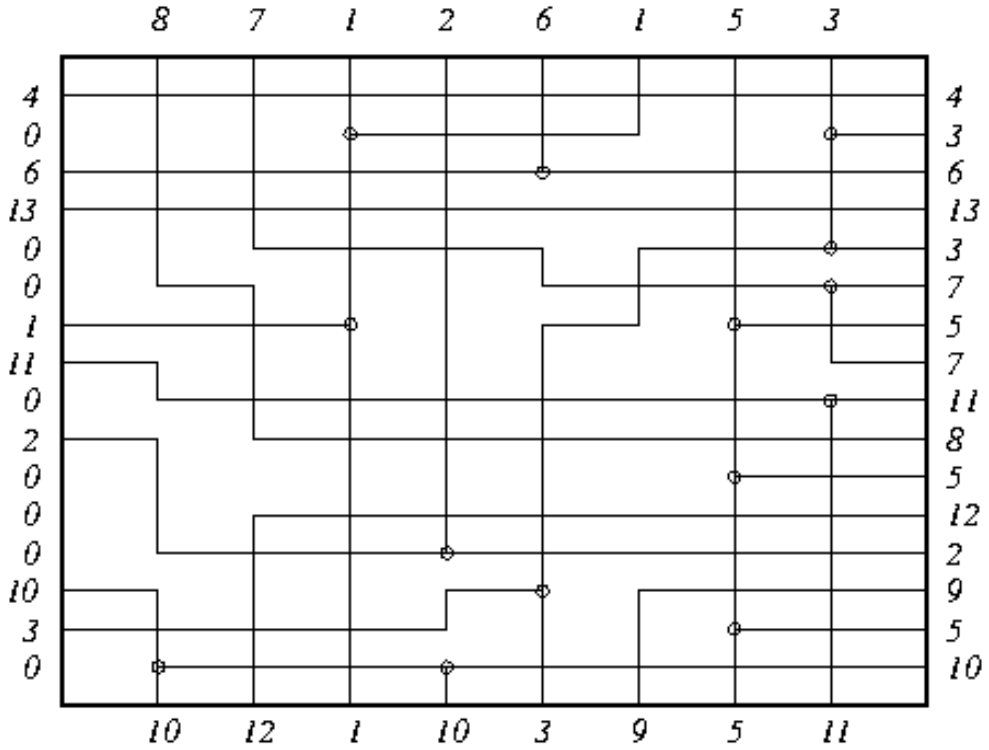
Robust Channel Router

```
robust_router (struct netlist N)
{
  set of int row;
  struct solution S;
  int total[channel_width + 1], selected_net[channel_width +
  int top, height, c, r, i;

  top ← 1;
  height ← density(N);
  for (r ← 1; r ≤ height; r ← r + 1) {
    for all "nets i in netlist N"
      w_i ← compute_weight(N, top);
    total[0] ← 0;
    for (c ← 1; c ≤ channel_width; c ← c + 1) {
      selected_net[c] ← 0;
      total[c] ← total[c − 1];
      if ("some net n has a top terminal at position c")
        if (w_n + total[x_{n_min} − 1]) > total[c]) {
          total[c] ← w_n + total[x_{n_min} − 1]);
          selected_net[c] ← n;
        }
      if ("some net n has a bottom terminal at position c")
        if (w_n + total[x_{n_min} − 1]) > total[c]) {
          total[c] ← w_n + total[x_{n_min} − 1]);
          selected_net[c] ← n;
        }/* if */
    }/* for */
    row ← ∅;
    c ← channel_width;
    while (c > 0)
      if (selected_net[c]) {
        n ← selected_net[c];
        row ← row ∪ {n};
        c ← x_{n_min} − 1;
      }
      else
        c ← c − 1;
    solution ← solution ∪ {row};
    top ← !top;
    N ← "N without the nets selected in row"
  }/* for */
  "apply maze routing to eliminate possible vertical constraint violations"
}
```

Courtesy of Y.-W. Chang

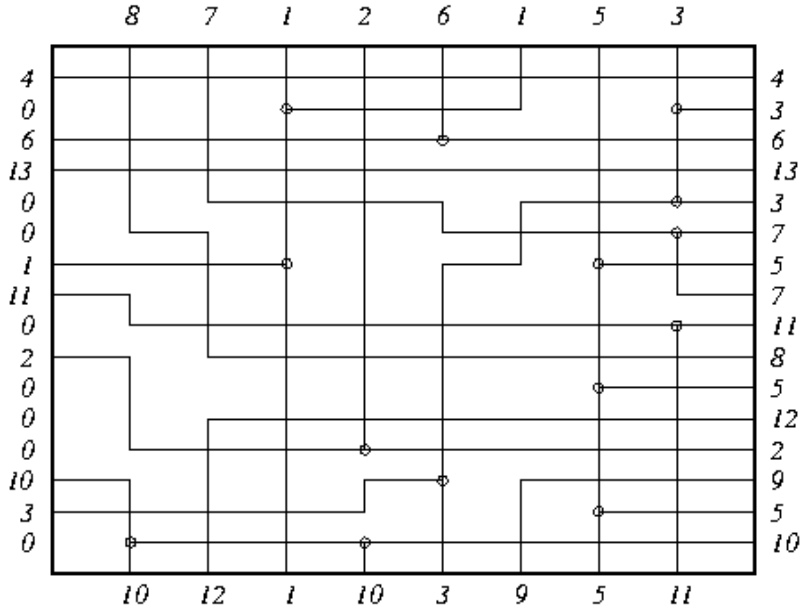
Appendix B:
Switchbox Routing



Courtesy of Y.-W. Chang

Greedy Switchbox Router

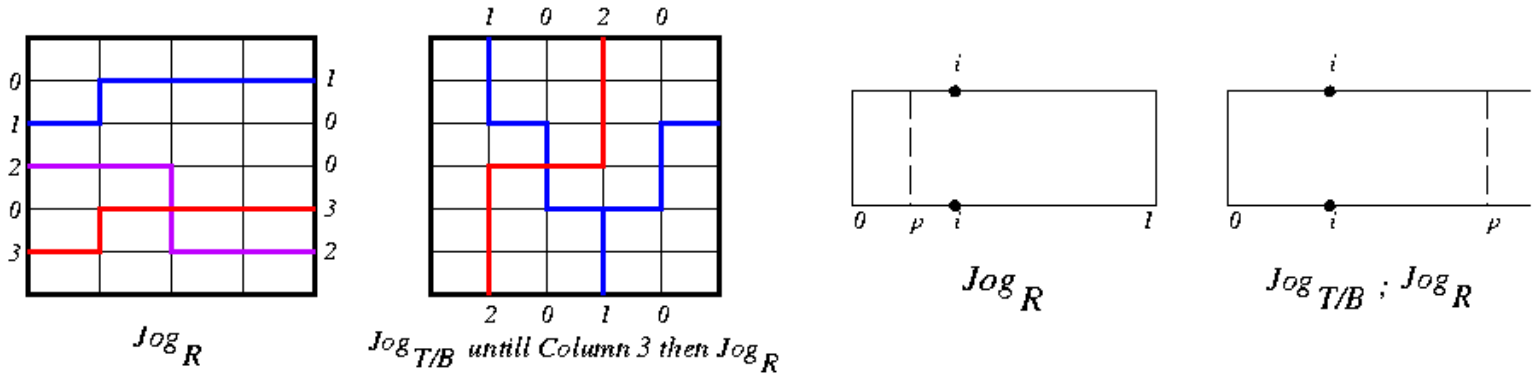
- Luk, “A greedy switchbox routing,” *INTEGRATION, the VLSI Journal*, 1985.
- Based on the greedy channel router (Rivest & Fiduccia).
- Terminals on left boundary are brought to the 1st column as horizontal tracks.
- Nets are jogged to target rows in additional to jogging to next top or bottom terminals.
- Various jogging schemes are applied.



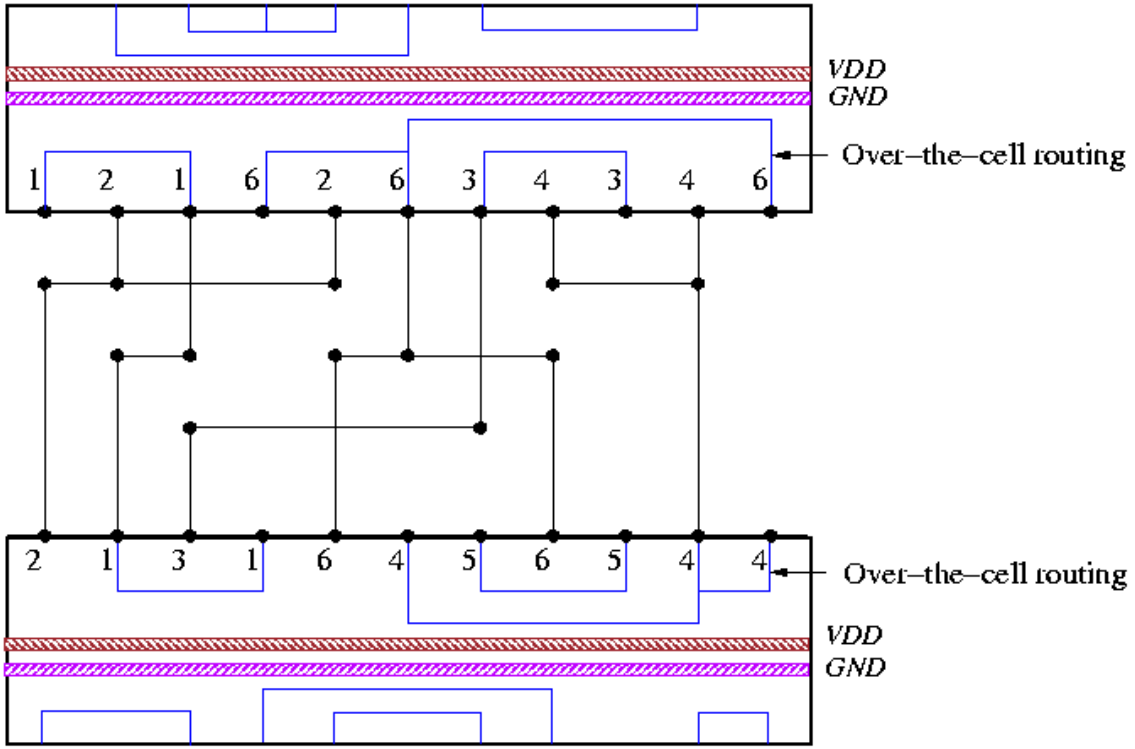
Courtesy of Y.-W. Chang

Jogging Schemes

- Jog_R: Nets with terminals on the **right** boundary; the jog is performed until matching the right positions.
- $Jog_{T/B}$: Nets with terminals only on the **top** and **bottom**; similar to the greedy channel router.
- $Jog_{T/B}$: Jog_R: First perform $Jog_{T/B}$; switch to Jog_R when the last top or bottom terminal appears.



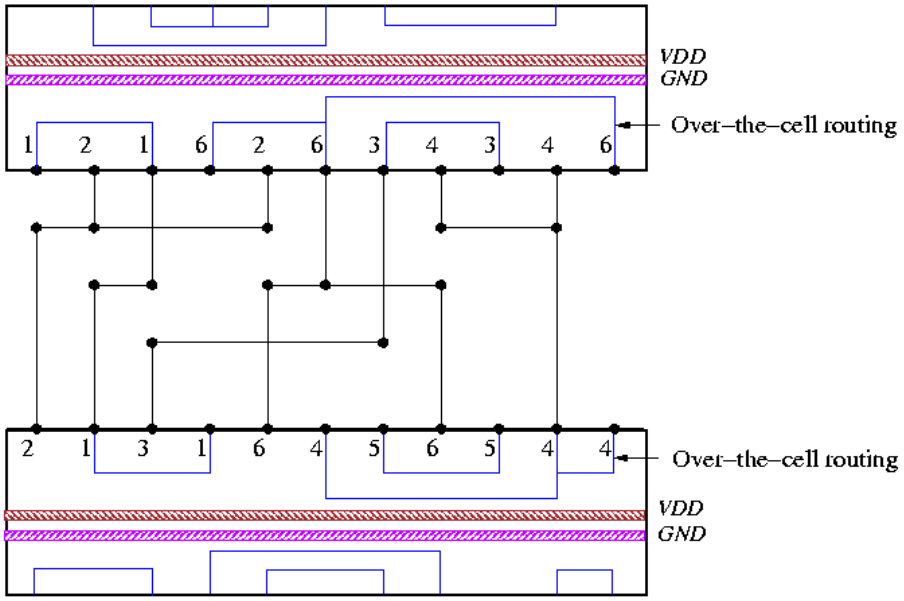
Appendix C:
Over-the-Cell Routing



Courtesy of Y.-W. Chang

Over-the-Cell Routing

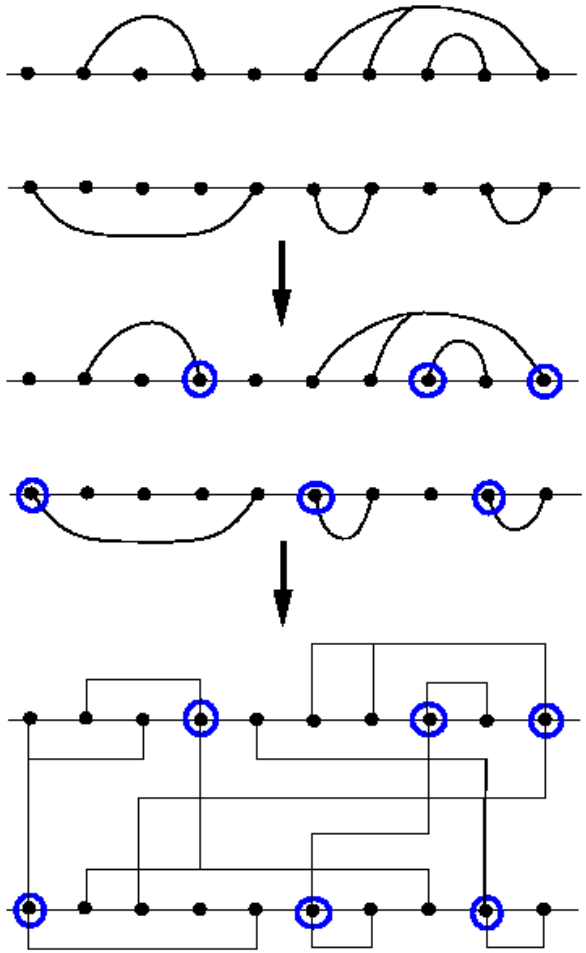
- Routing over the cell rows is possible due to the limited use of the 2nd (M2) metal layers within the cells.
- Divide the over-the-cell routing problem into 3 steps: (1) routing over the cell, (2) choosing the net segments, and (3) routing within the channel.
- Reference: Cong & Liu, "Over-the-cell channel routing," *IEEE TCAD*, Apr. 1990.



Courtesy of Y.-W. Chang

Over-the-Cell Channel Routing

- Cong & Liu, "Over-the-cell channel routing," *IEEE TCAD*, Apr. 1990.



Select over–the–cell nets
use Supowit's Max. Independent Set algorithm for circle graph
(solvable in $O(c^3)$ time,
c: # of columns)

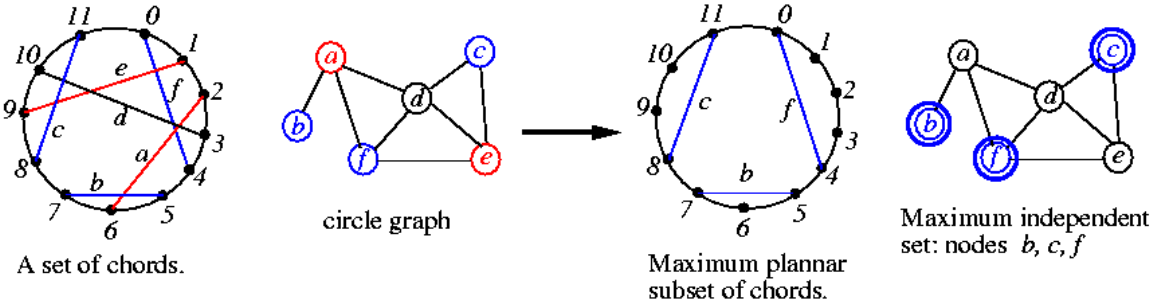
Select terminals among "equivalent" ones for regular channel routing
(Goal: minimize channel density
NP–complete!)

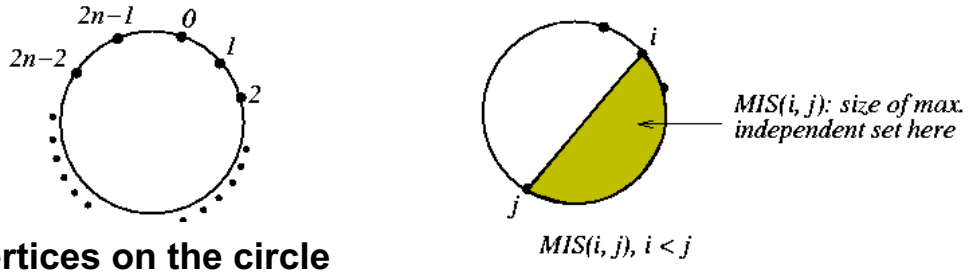
Plannar routing for over–the–cell nets
+
Regular channel routing

Courtesy of Y.-W. Chang

Supowit's Algorithm

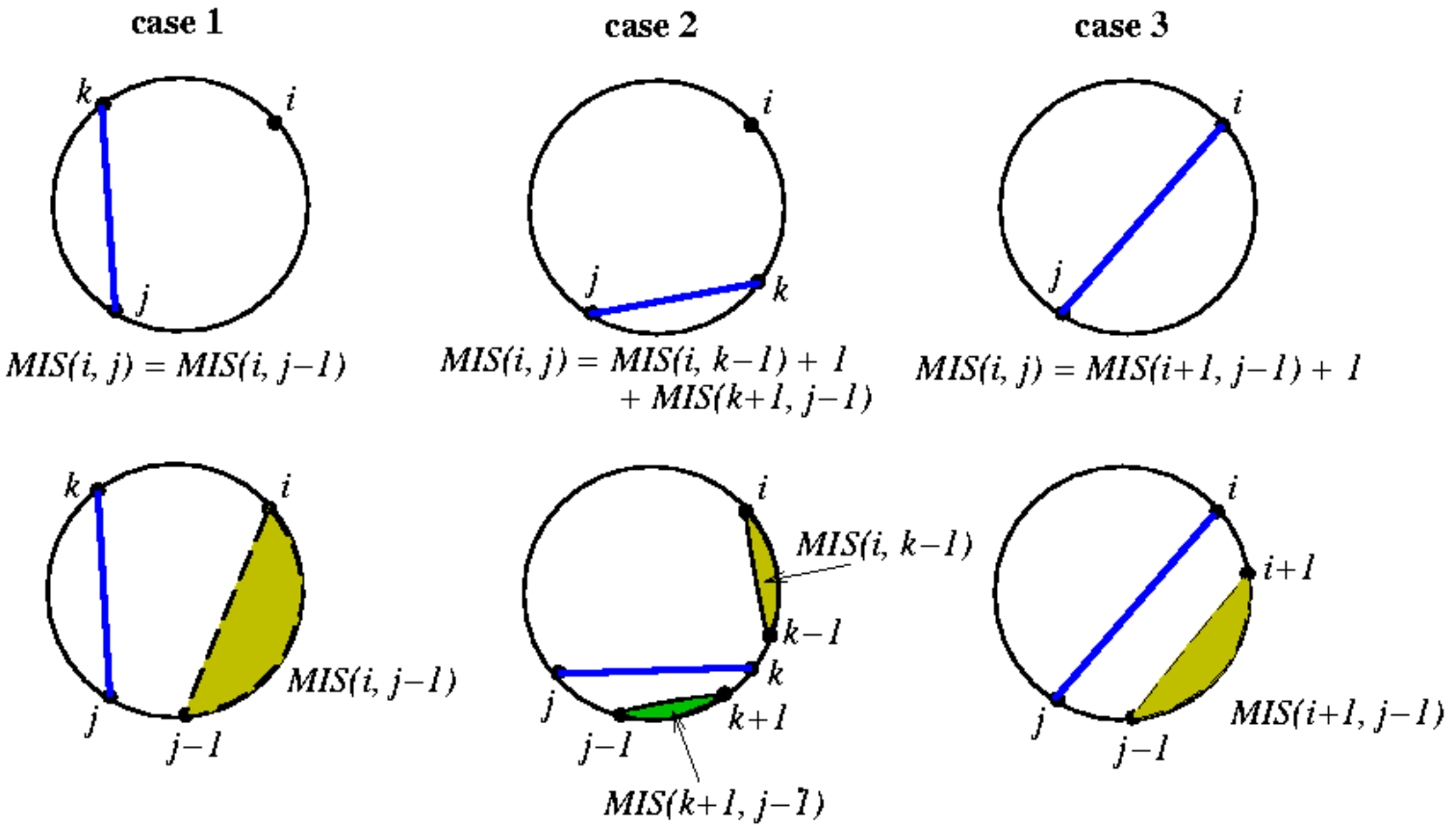
- Supowit, “Finding a maximum planar subset of a set of nets in a channel,” *IEEE TCAD*, 1987.
- Problem: Given a set of chords, find a maximum planar subset of chords.
  - Label the vertices on the circle 0 to 2*n*-1.
  - Compute *MIS*(*i, j*): size of maximum independent set between vertices *i* and *j*, *i* < *j*.
  - Answer = *MIS*(0, 2*n*-1).



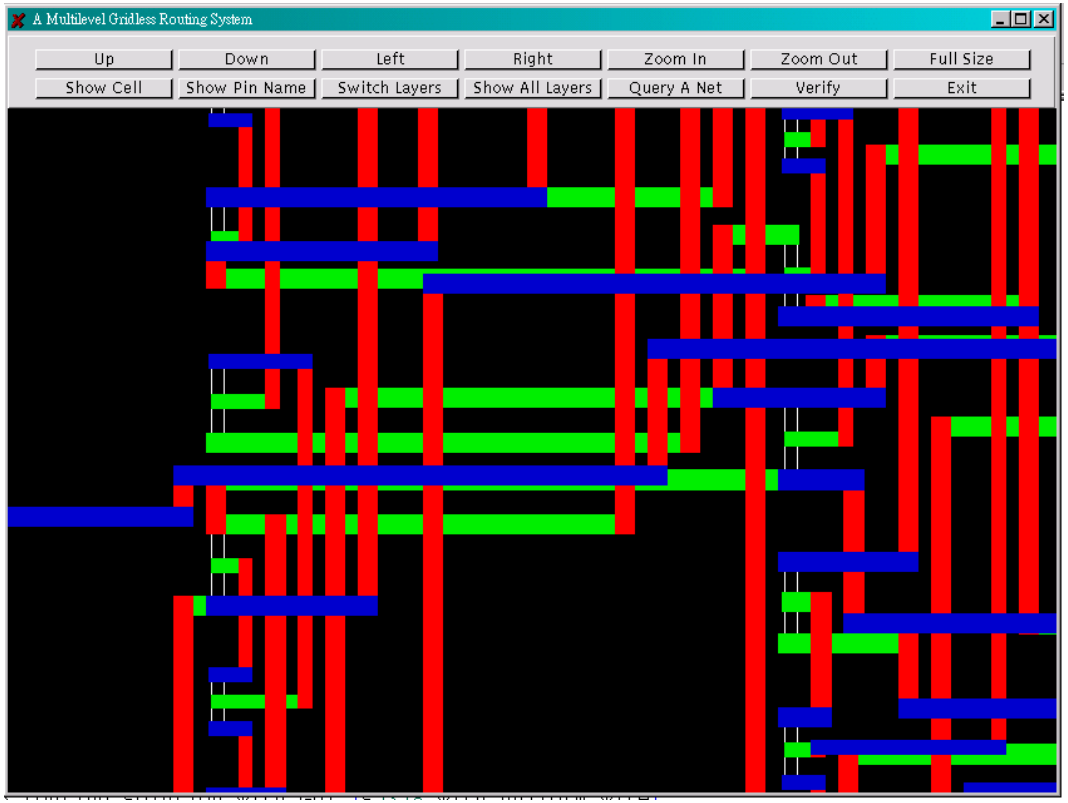


Dynamic Programming in Supowit's Algorithm

- Apply dynamic programming to compute *MIS*(*i, j*).



Appendix D:
Gridless Routing



Courtesy of Y.-W. Chang

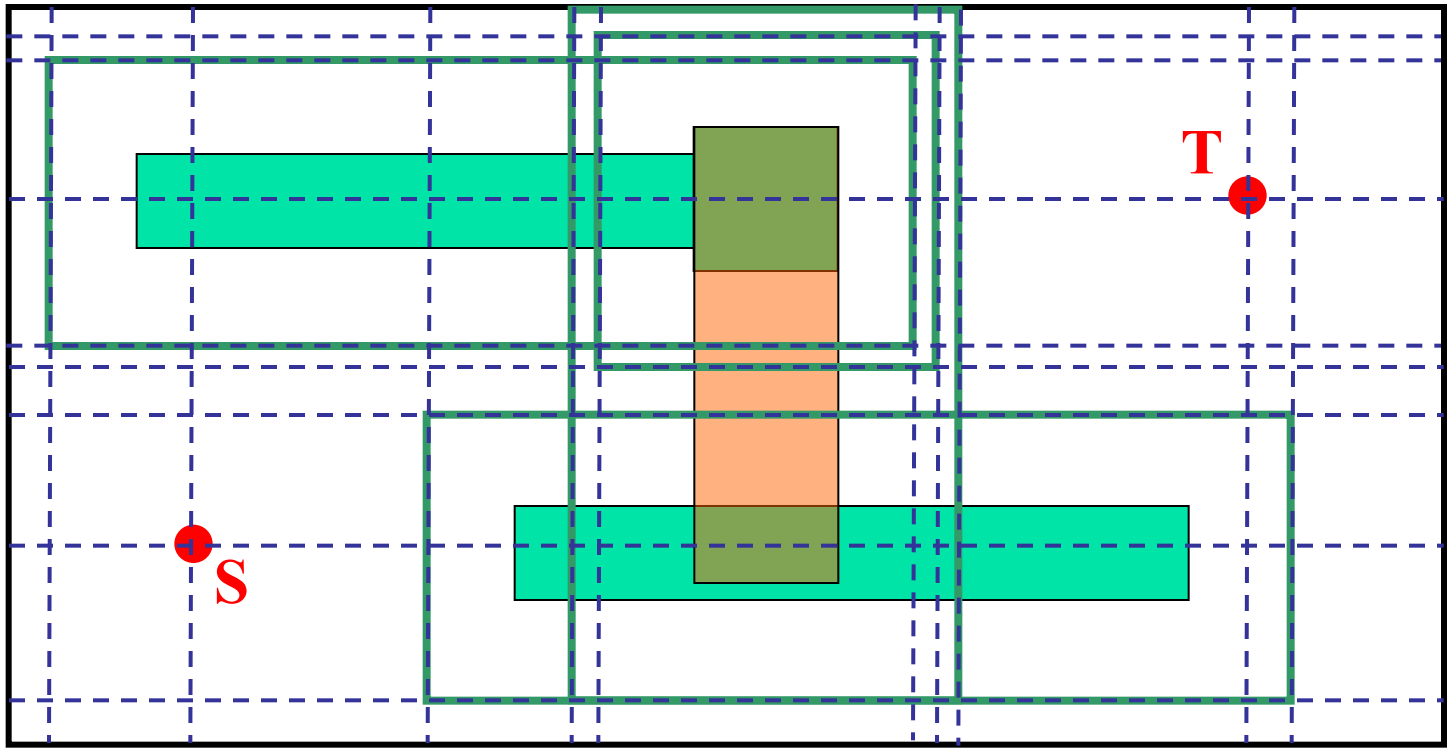
Gridless Full-Chip Routing

- Chen and Chang, “Multilevel full-chip gridless routing considering optical proximity correction,” ASPDAC-05.
- Is based on the multilevel routing framework.
- Applies the implicit connection graph to transform the gridless structure into a “grid”-like structure.
- Adopts the interval tree to do range query for identifying obstacles and available spaces.
- Congestion metric is based on available routing space in a region.
- Gridless routing is needed for handling some nanometer effects like OPC, metal slotting, etc.

Courtesy of Y.-W. Chang

Implicit Connection Graph

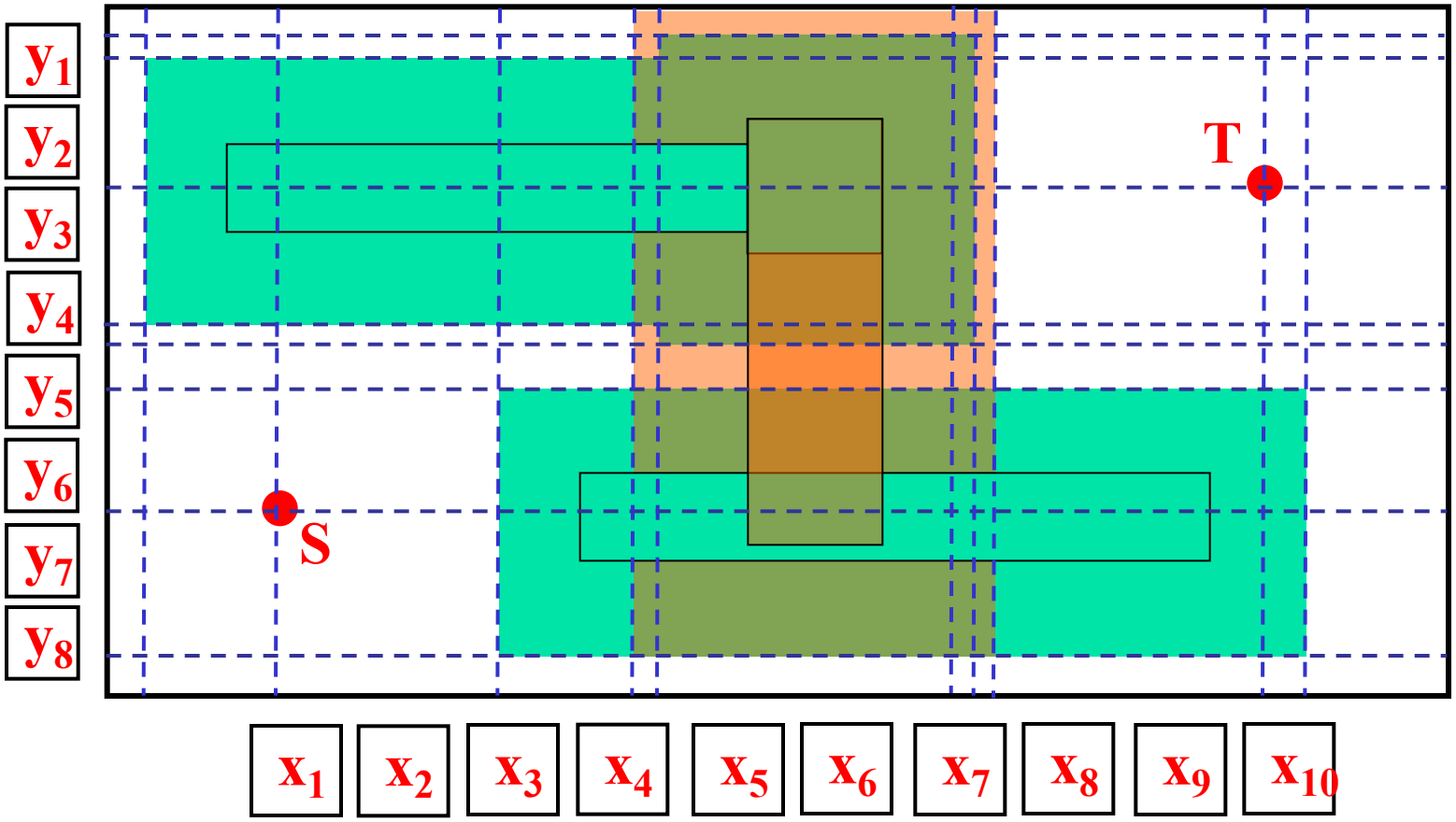
- Given a set of obstacles and a source *s* and sink *t*
- G_S is an orthogonal grid graph



Courtesy of Y.-W. Chang

Implicit Representation of G_S

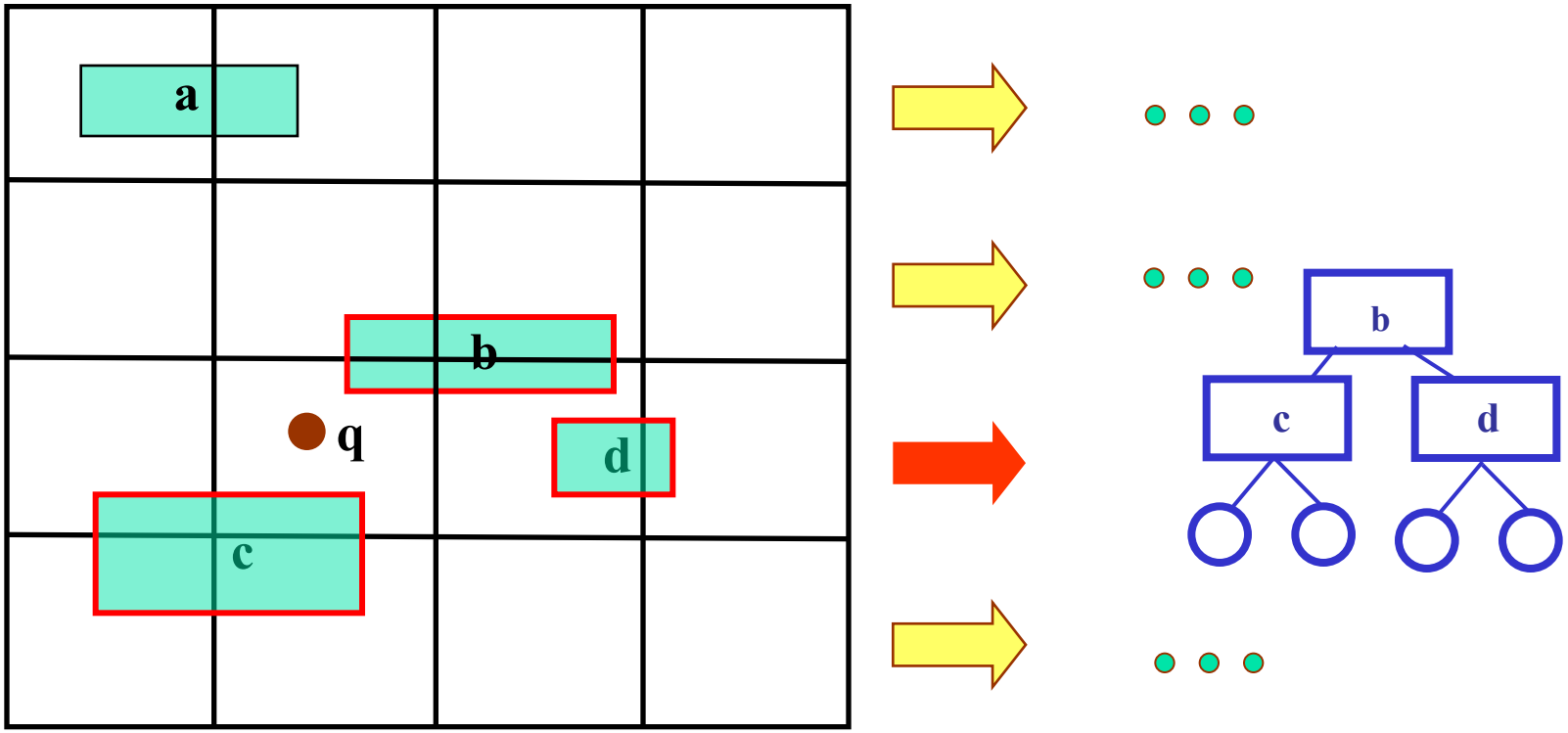
- Store *x, y* coordinates into two sorted arrays
- O(*n*) space & O(*n* lg *n*)-time pre-construction (*n*: # of rectangles)



Courtesy of Y.-W. Chang

Interval Tree

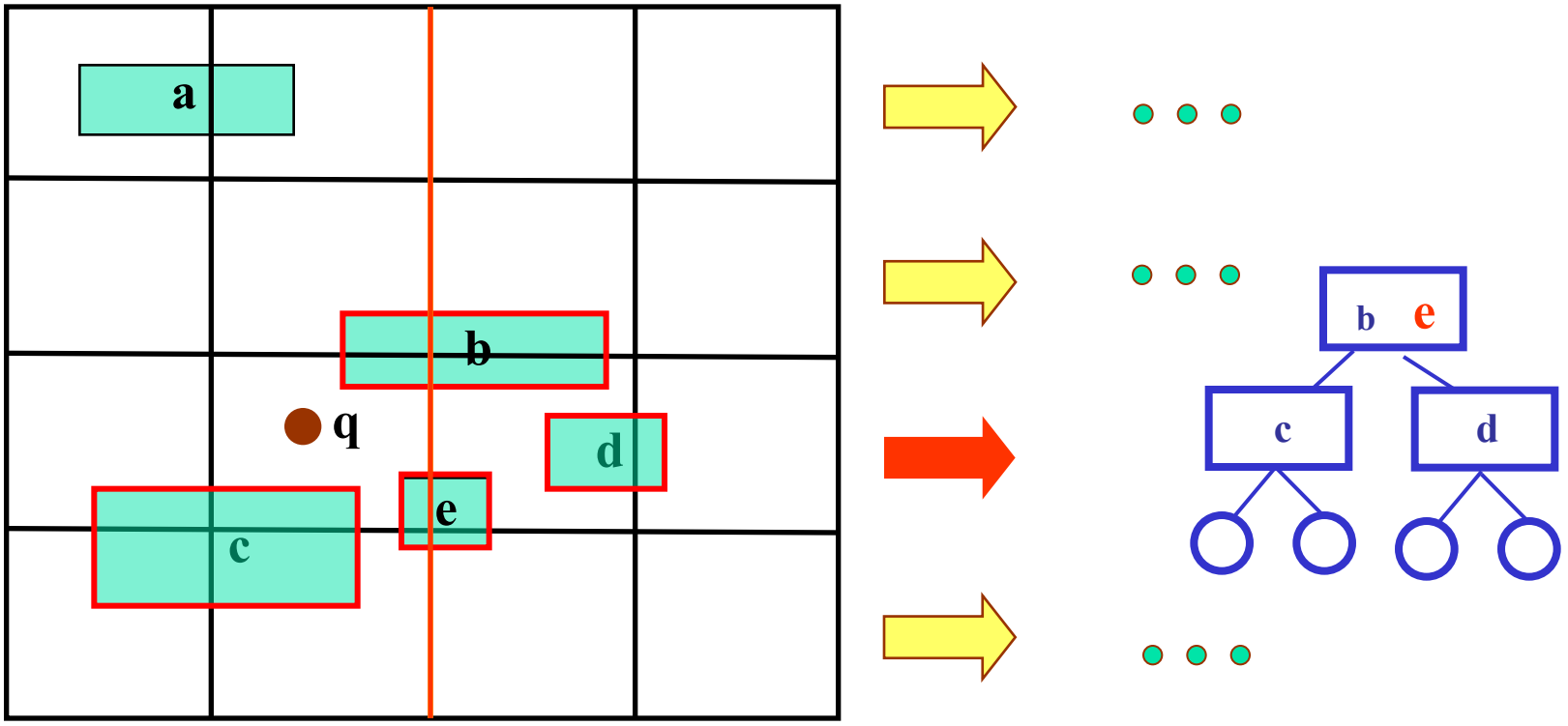
- Partition the chip plane into rows and keep a query tree for each row.
- The query tree stores cut blocks in the internal nodes.
- Uncut blocks are stores in the leaf nodes.
- Question: Is *q* in free space? Need average *O*(lg *n*) time to answer.



Courtesy of Y.-W. Chang

Block Insertion

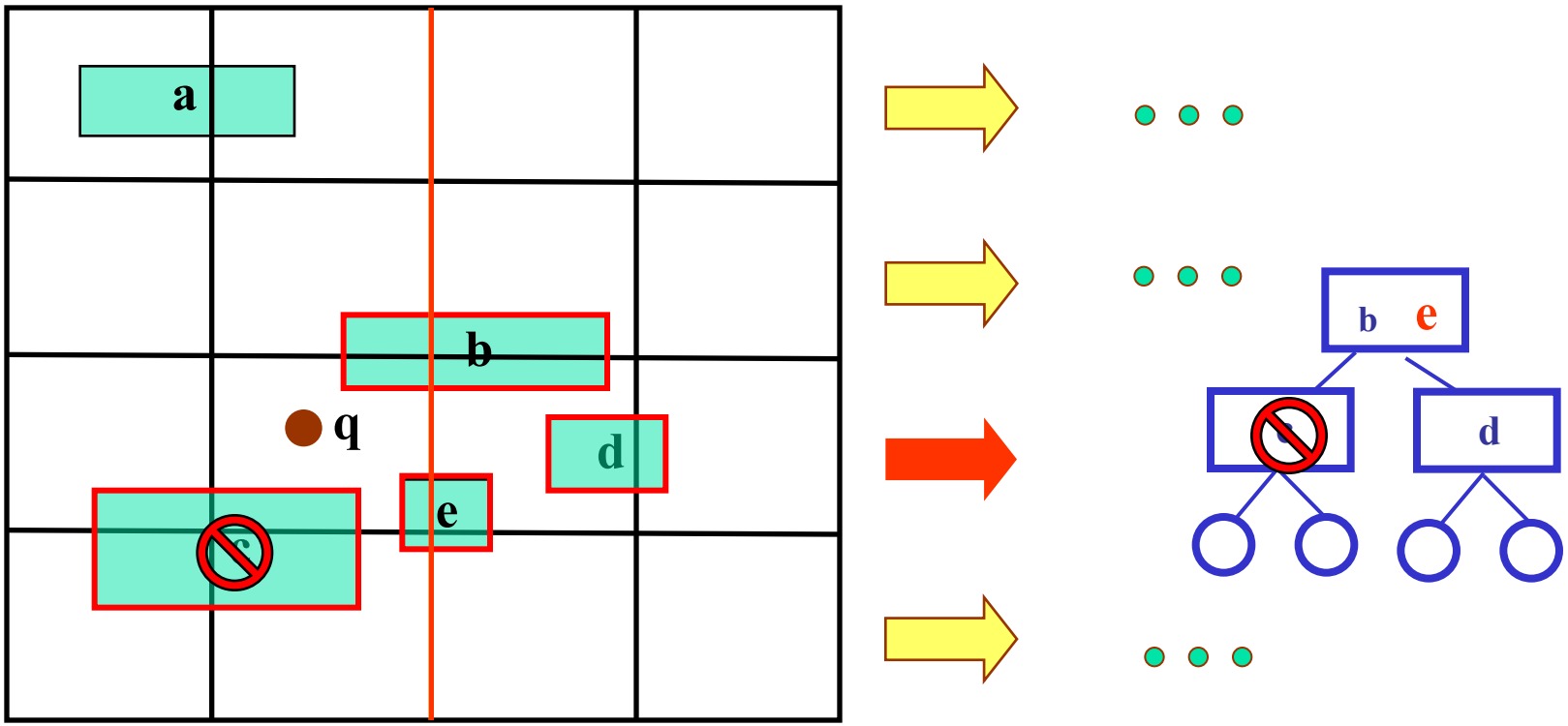
- To insert a block (say, block *e*), we traverse the query tree from the root down to an internal node until it is cut, or a leaf node if it is uncut.
- Average time complexity: $O(\lg n)$.



Courtesy of Y.-W. Chang

Block Deletion

- To delete a block (say, block *c*), we traverse the query tree from the root down to a node that stores the block name and delete the block.
- Average time complexity: $O(\lg n)$.



Courtesy of Y.-W. Chang

Complexity Summary

- Interval (query) tree

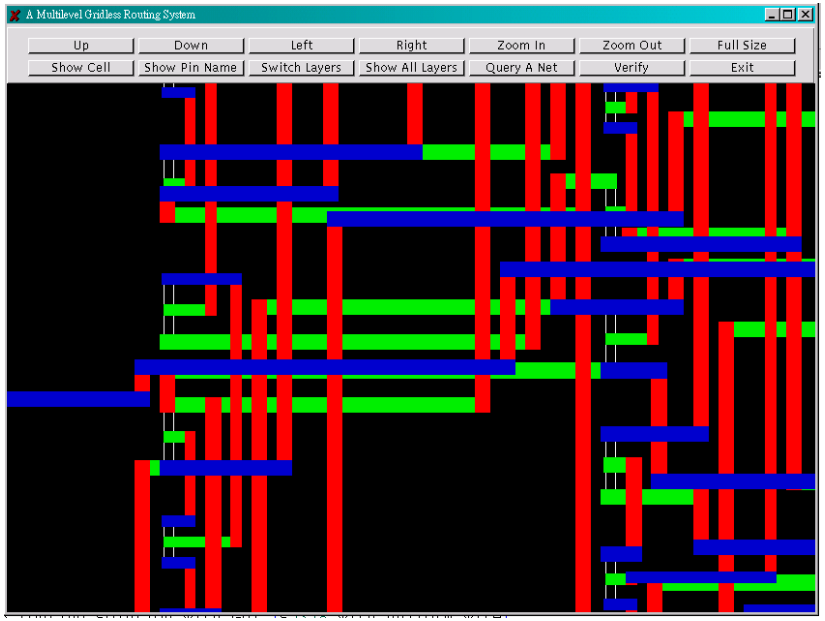
	Average Complexity
Memory Usage	**O(n^2)**
Block Insertion	**O(lg n)**
Block Deletion	**O(lg n)**
Point Finding	**O(lg n)**
Neighbor Finding	**O(1)**

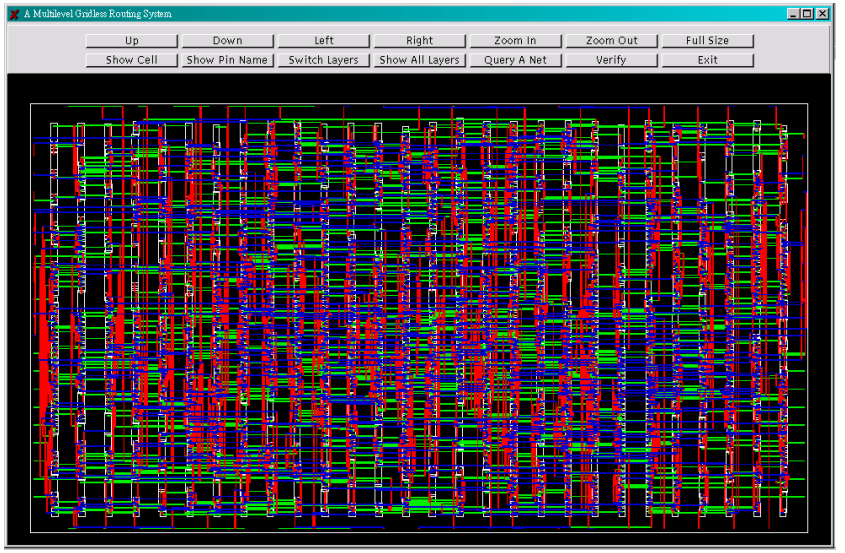
n: number of blocks

Courtesy of Y.-W. Chang

Routing Solution for s5378

The routing solution of ”s5378” with non-uniform (above) and uniform (below) nets

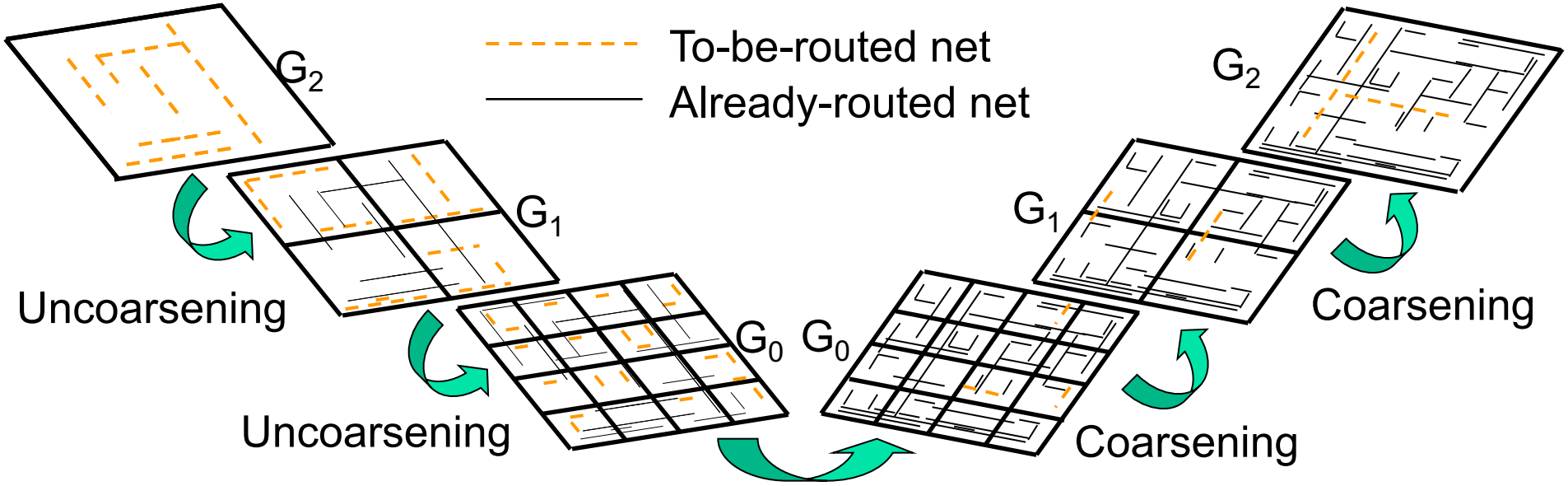




Courtesy of Tai-Chen Chen

Courtesy of Y.-W. Chang

Appendix E:
V-Shaped Gridless Routing

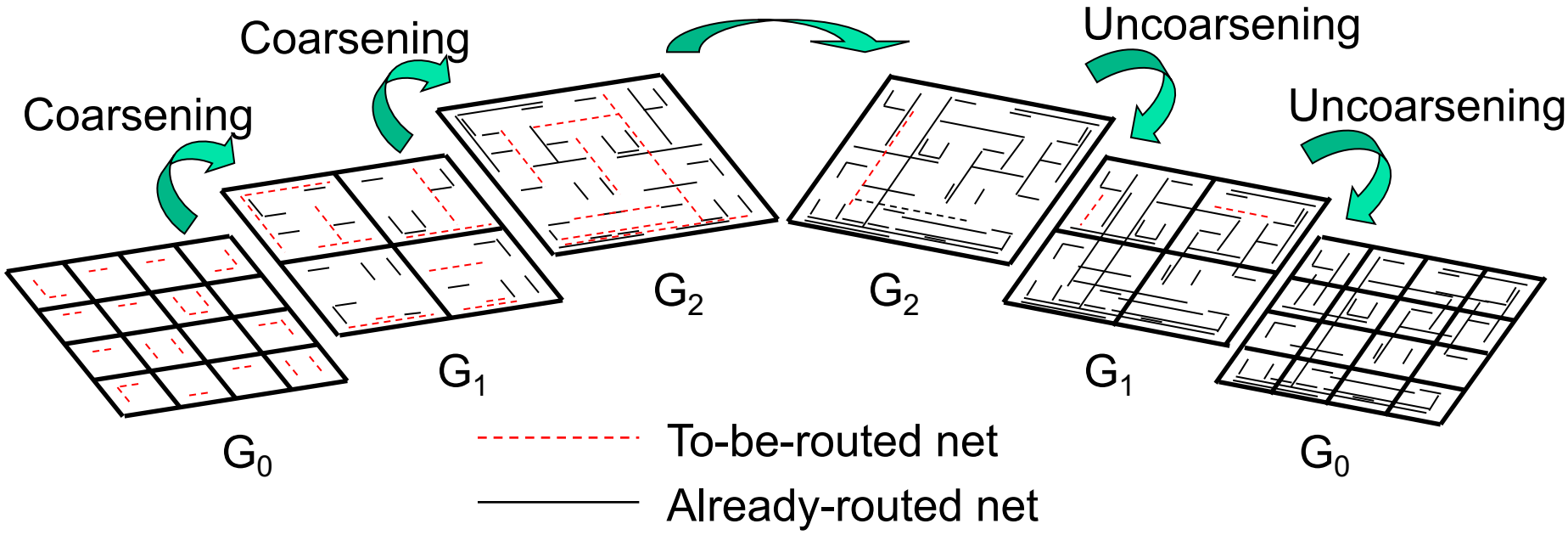


Courtesy of Y.-W. Chang

Λ-Shaped Multilevel Routing

Perform global pattern routing and Dijkstra's shortest path detailed routing for local connections and then estimate routing resource for the next level.

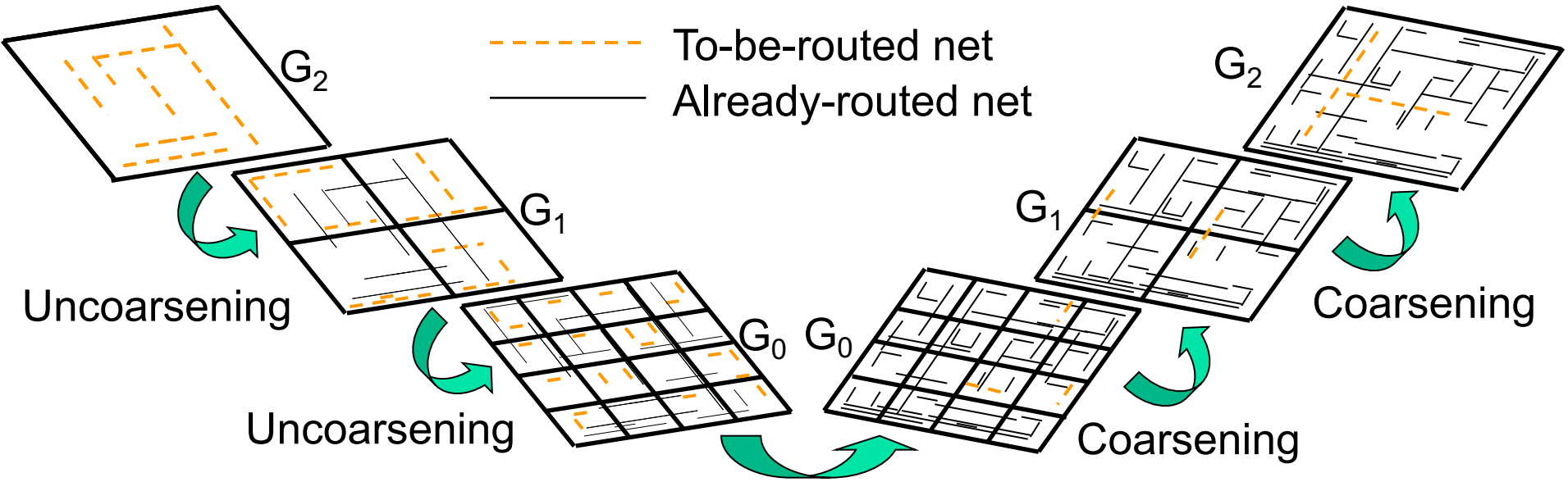
Use global maze routing and Dijkstra's shortest path detailed routing to reroute failed connections and refine the solution.



Does not have the view of the global configuration at the earlier stages.

Courtesy of Y.-W. Chang

V-Shaped Multilevel Framework



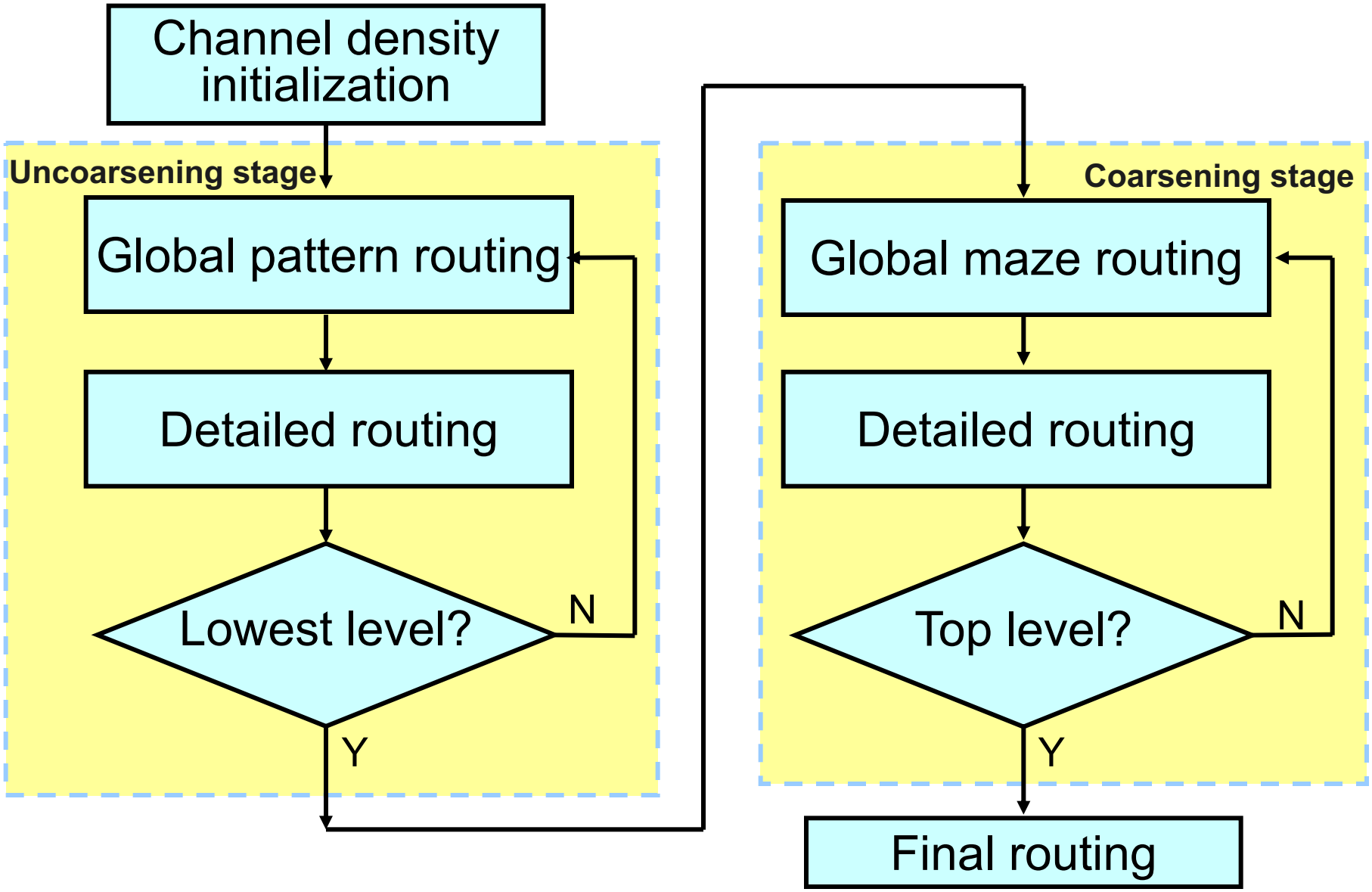
Perform global pattern routing and Dijkstra's shortest path detailed routing for local nets and then estimate routing resource for the next level.

Use global maze routing and Dijkstra's shortest path detailed routing to reroute failed connections and refine the solution.

Consider the global effects at the earlier stages.

Courtesy of Y.-W. Chang

Design Flow



Courtesy of Y.-W. Chang

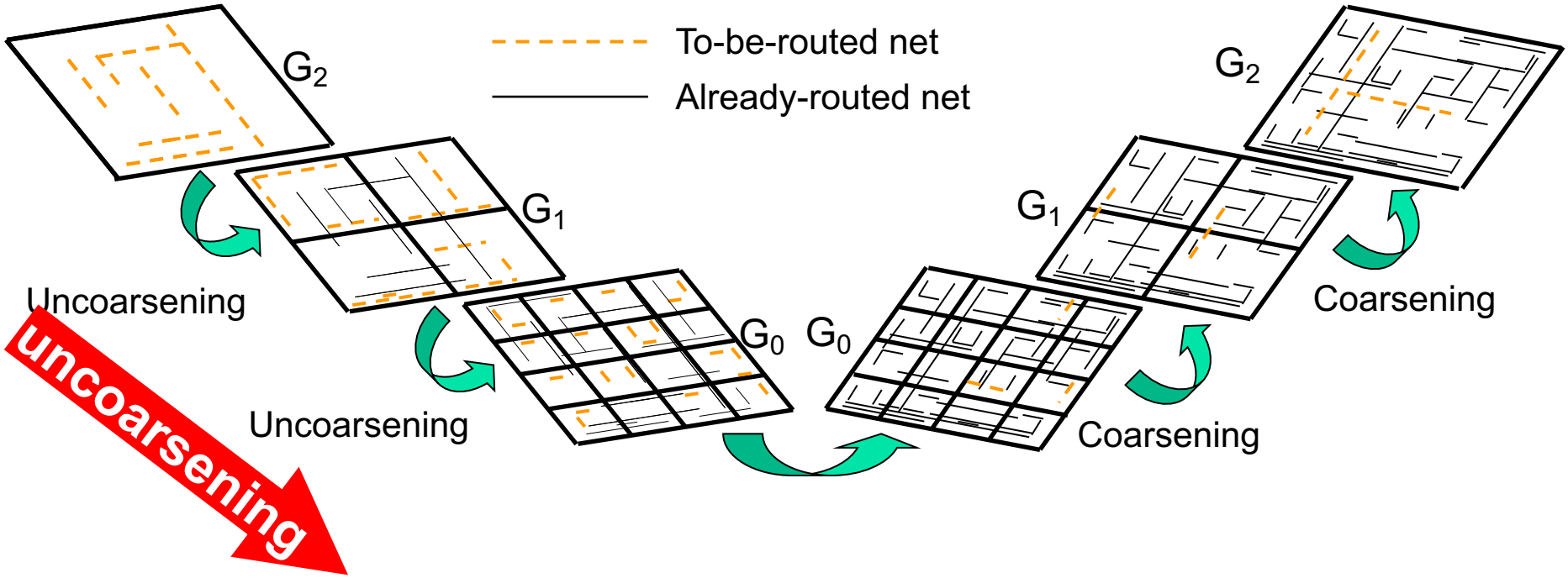
Channel Density Initialization and Updated

- Making the global routing, detailed routing, and resource estimation interact with each other can significantly improve routing completion rates
  - Only guide the latter nets passing through the area with lower congestion
  - Cannot avoid determining the bad global path of an early routed net without considering the routing resource of succeeding nets.
- Initialize the congestion map based on the pin distribution and the global-path prediction of all nets
- Update the congestion map dynamically based on both the already routed nets and the estimated unrouted nets
- Have better congestion control throughout the whole routing process

Courtesy of Y.-W. Chang

Uncoarsening Stage

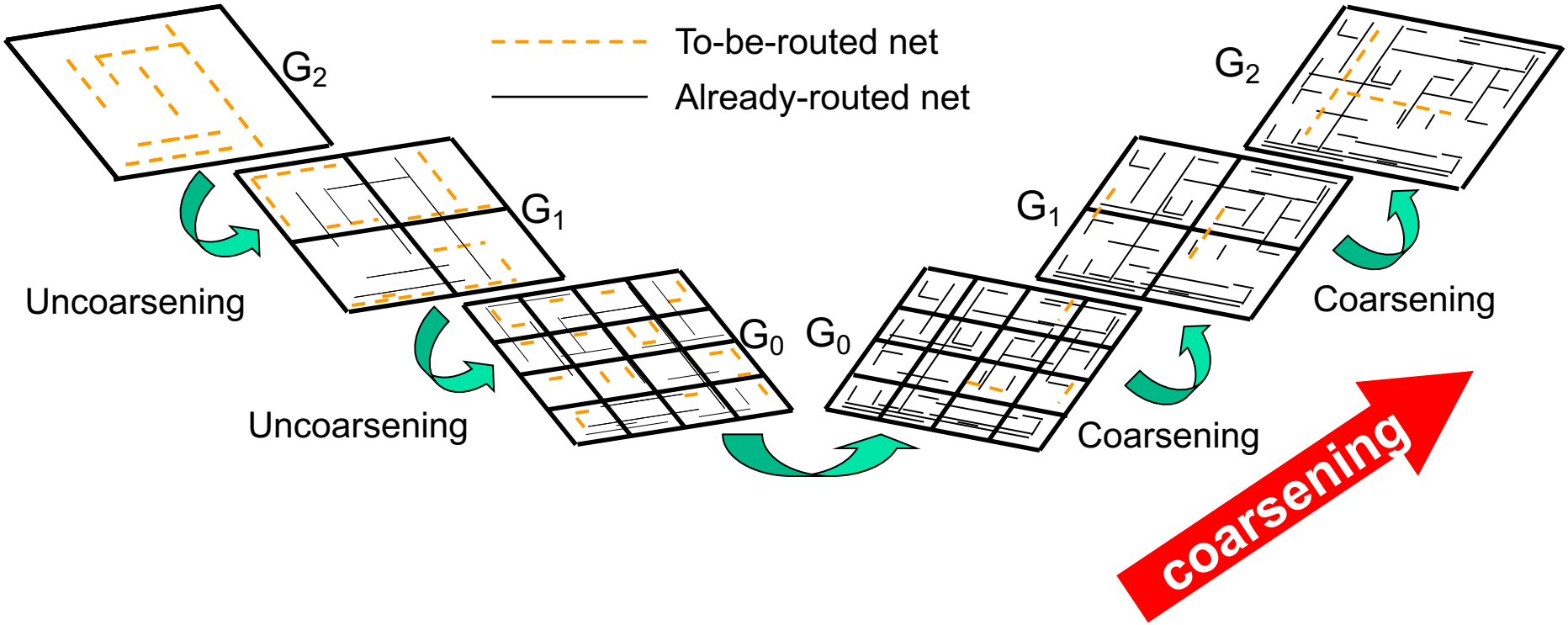
- Global effect is the highest consideration in this stage
  - Longer nets are more critical
- Start from the coarsest tiles of level k; route level nets at each level



Courtesy of Y.-W. Chang

Coarsening Stage

- Routability is the highest consideration in this stage
  - Shorter connections are more critical
- Repeat the same steps as the uncoarsening stage
  - Perform maze routing at the global routing stage



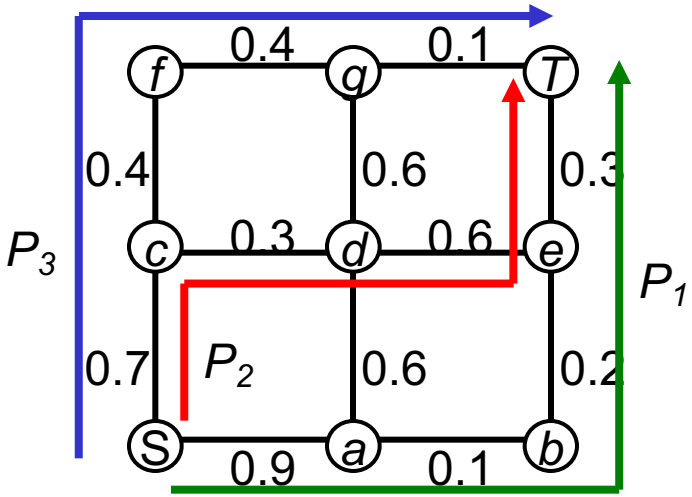
Courtesy of Y.-W. Chang

Cost Function for Global Routing

- The cost function is the sum of the maximum channel congestion and the average of the total path congestion
  - Can avoid selecting a path with lower total path congestion and higher channel congestion path
  - Can avoid selecting a path with higher overall path congestion when two path have the same maximum channel congestion

$$\alpha(R_e) = \max_{e \in R_e} c_e + \frac{1}{|R_e|} \sum_{e \in R_e} c_e$$

c_e: the congestion of edge e



P1 has the minimal total path congestion

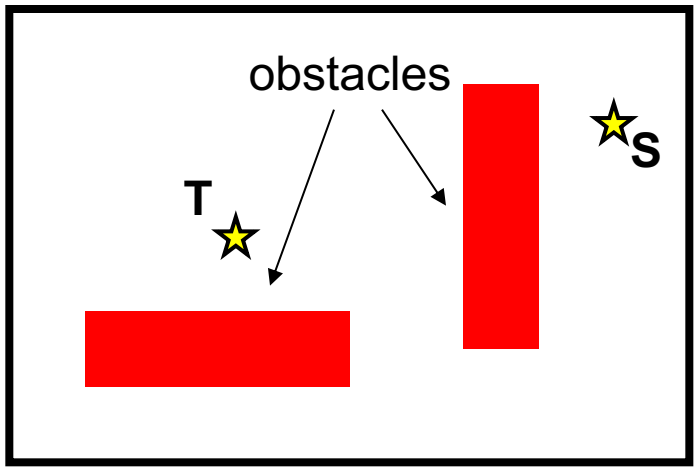
P2 and P3 have the same maximum channel congestion

P3 has the minimum cost

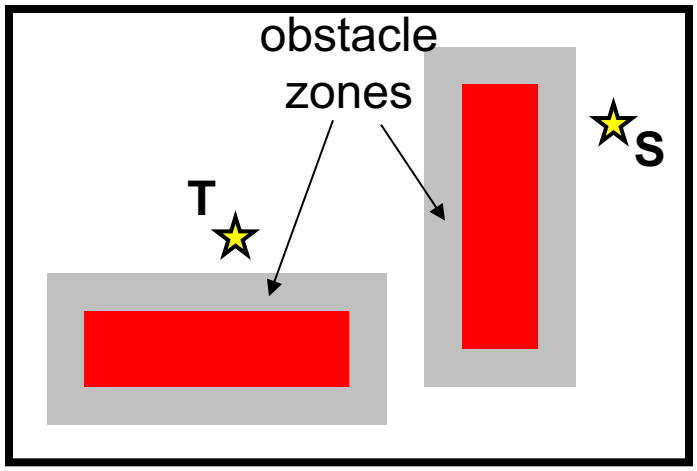
Courtesy of Y.-W. Chang

Triple-Line Graph (TLG) Model

- Find a design-rule-correct path and avoid redundant wires
- Construct the obstacle zones (gray areas) from the obstacles by taking design rules into account
  - expand the obstacle for a range which is the sum of the obstacle spacing and the half width of the routing wire
  - The area outside of the obstacle zones is available for placing the center lines of wires and mid-points of contacts



a routing example

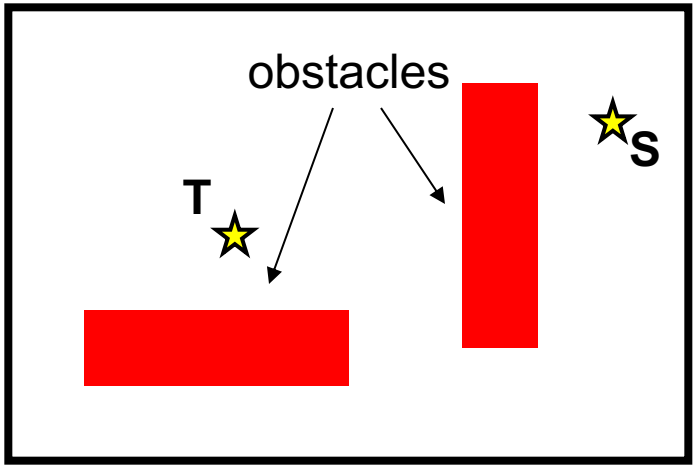


obstacle zones construction

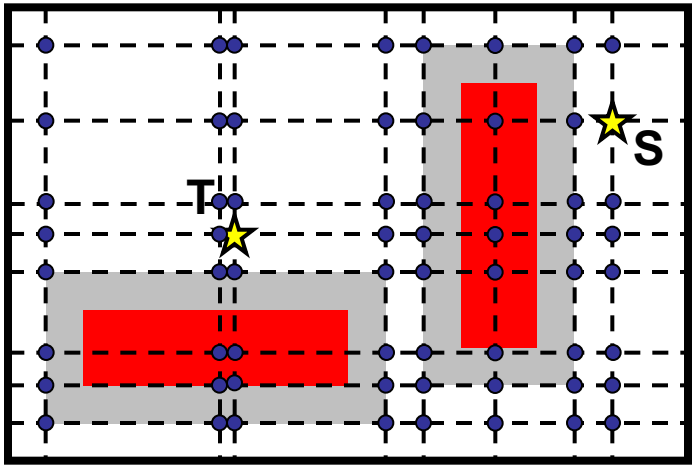
Courtesy of Y.-W. Chang

TLG Model (Cont'd)

- Collect all x-coordinates and y-coordinates of
  - the source and the target
  - the boundaries of all obstacle zones
  - the centers of all obstacle zones (P-lines and C-lines)
- Generate a vertical (horizontal) dashed line for each x-coordinate (y-coordinate)
- Construct a connection graph
  - a node in the connection graph denotes an intersection of a horizontal and a vertical dashed lines



a routing example

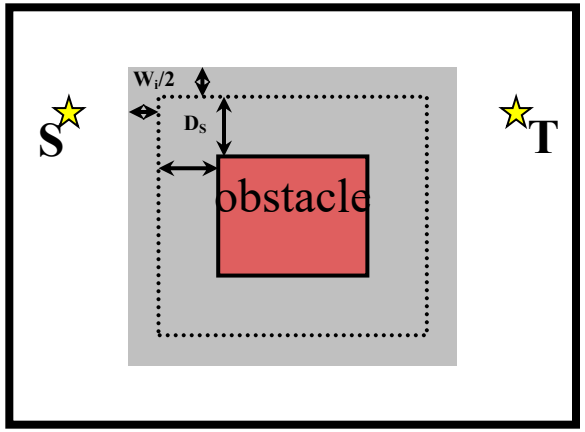


a connection graph

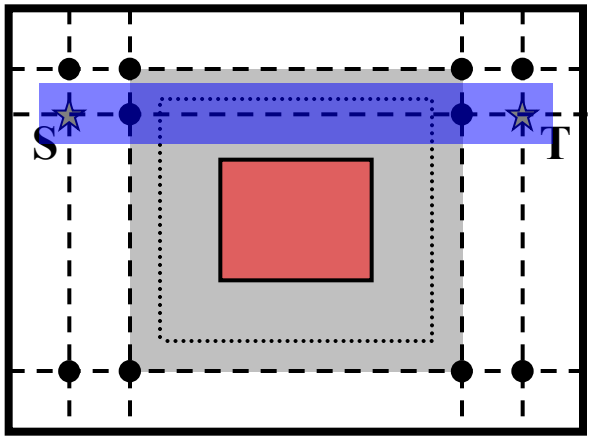
Courtesy of Y.-W. Chang

P-Lines

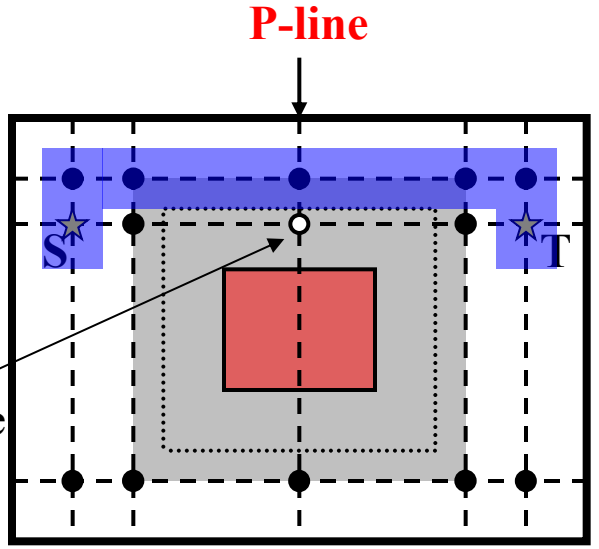
- To avoid design-rule-incorrect paths
  - pass the center of the obstacle zone
  - Is perpendicular to the routing direction of the obstacle zone



a routing example



A design-rule-incorrect path

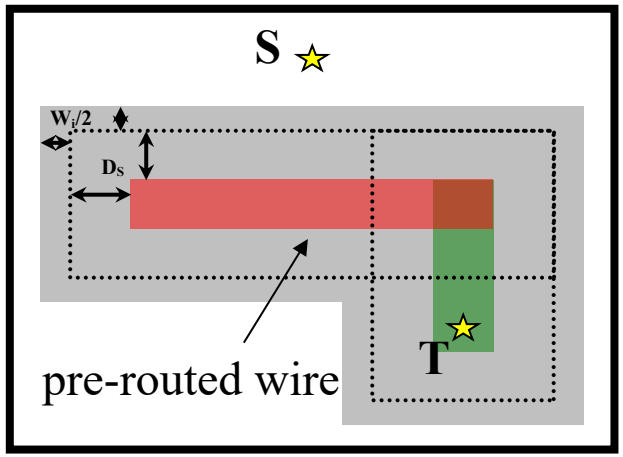


A design-rule-correct path

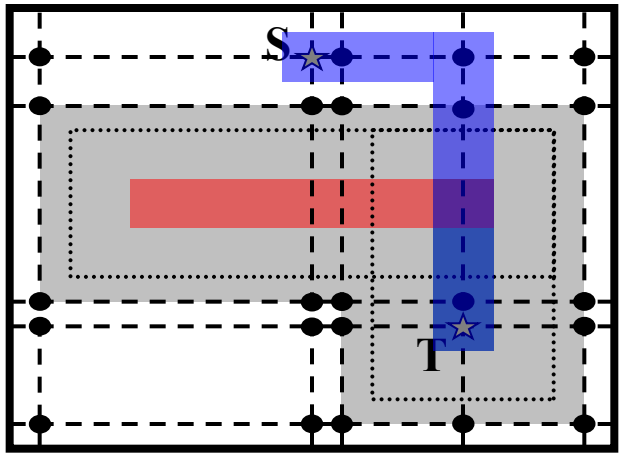
Courtesy of Y.-W. Chang

C-Lines

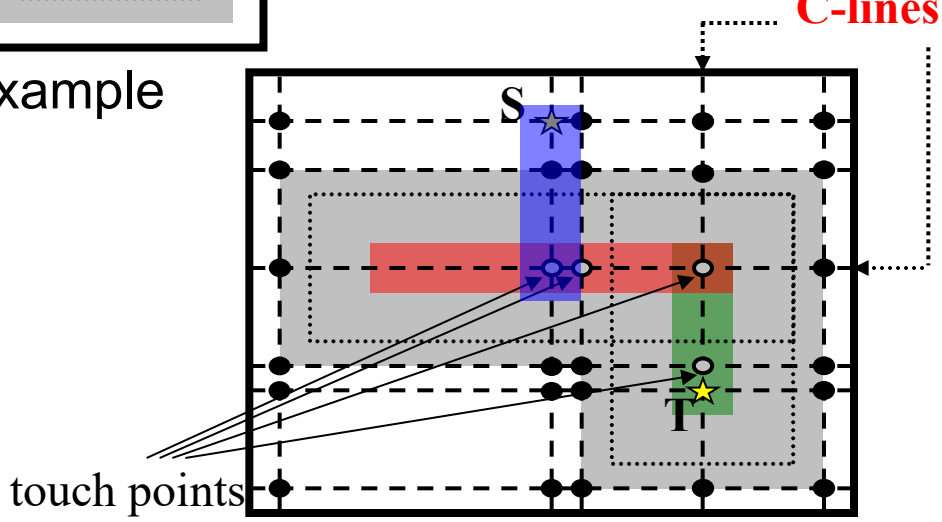
- To avoid redundant wires
  - pass the center of the obstacle zone
  - Is parallel to the routing direction of the obstacle zone



a routing example



A path with an redundant wire



A path without redundant wires

Courtesy of Y.-W. Chang