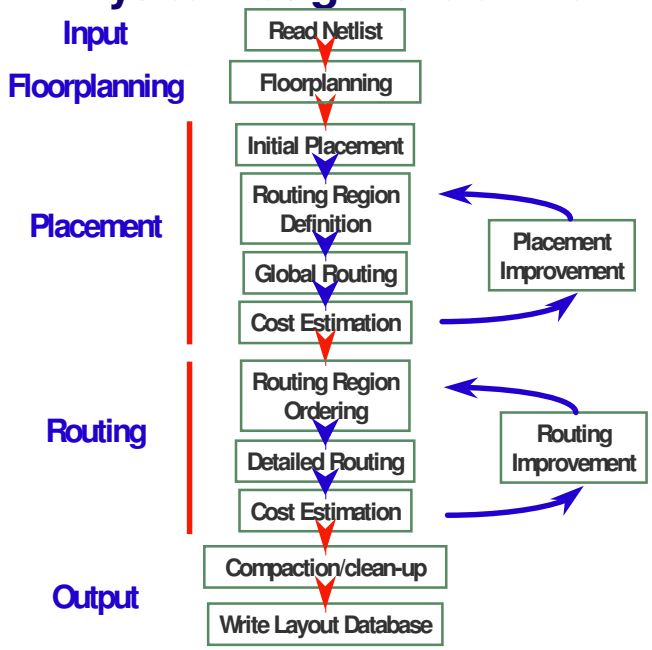
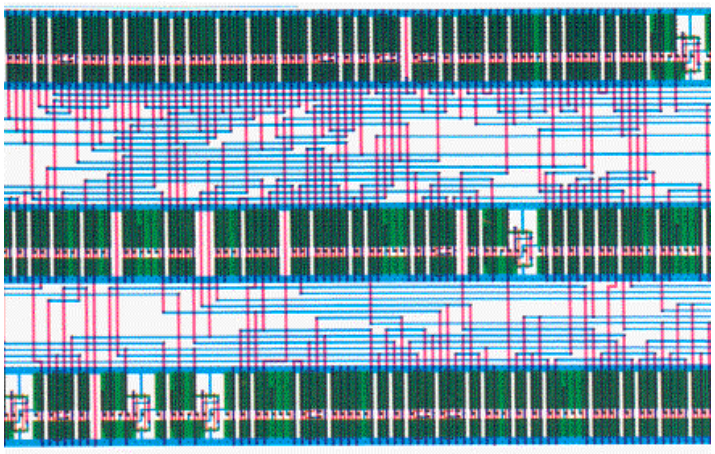
# Routing in Integrated Circuits

Prof. Jason Cong
CS, UCLA
Prof. Kurt Keutzer
Prof. Richard Newton
Michael Orshansky
EECS
UC Berkeley

## Physical Design: Overall Flow

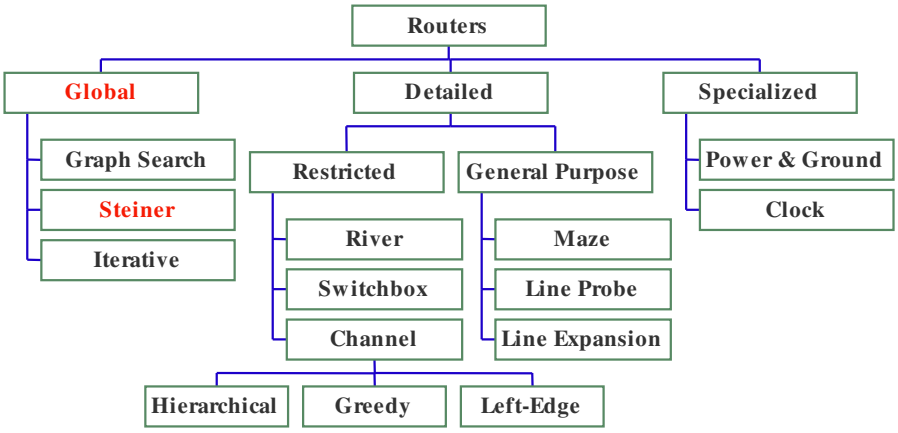


# Standard Cell Layout





# Taxonomy of VLSI Routers

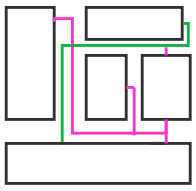


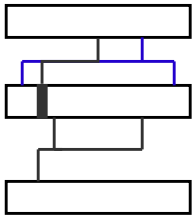
# Global Routing Formulation

**Given** (i) Placement of blocks/cells
(ii) channel capacities

**Determine**
Routing topology of each net

**Optimize**
(i) max # nets routed
(ii) min routing area
(iii) min total wirelength





In general cell design or standard cell design, we are able to move blocks or cell rows, so we can guarantee connections of all the nets.

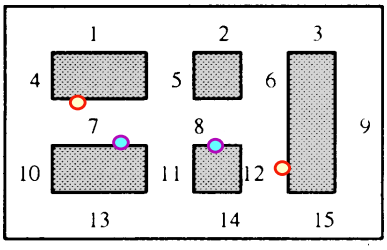
In gate array design, exceeding channel capacity is not allowed.

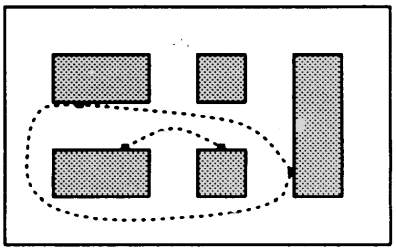
With more (>3) metal layers over-the-cell routing is allowed easing the problem.



# Global Routing

- Provide guidance to detailed routing (why?)
- Objective function application-dependent



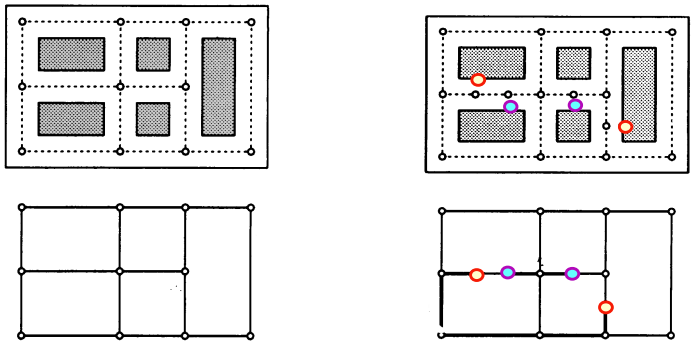
# Graph Models for Global Routing

- Global routing problem is a graph problem
- Model routing regions, their adjacencies and capacities as graphs
- Choice of model depends on algorithm
- Grid graph model
  - Grid graph represents layout as a $h \times w$ array, vertices are layout cells, edges capture cell adjacencies, zero-capacity edges represent blocked cells.
- Channel intersection graph model



# Graph Models for Global Routing

- Channel Intersection Graph most common approach
  - Edges are channels, vertices are channel intersections (CI), v1 and v2 are adjacent if there exists a channel between ($CI_1$ and $CI_2$). Graph can be extended to include pins.



# Global Routing Approaches

- Can route nets:
  - Sequentially, e.g. one at a time
  - Concurrently, e.g. simultaneously all nets
- Sequential approaches
  - Sensitive to ordering
  - Usually sequenced by
    - Criticality
    - Number of terminals
- Concurrent approaches
  - Computationally hard
  - Hierarchical methods used



# Sequential Approaches

- Solve a single net routing problem
- Differ depending on whether net is two- or multi-terminal
- Two-terminal algorithms
  - Maze routing algorithms
  - Line probing
  - Shortest-path based algorithms
- Multi-terminal algorithms
  - Steiner tree algorithms

# Two-Terminal Routing: Maze Routing

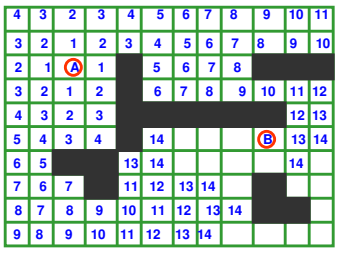
- Maze routing finds a path b/w source (s) and target (t) in a planar graph
- Grid graph model is used to represent block placement
- Available routing areas are unblocked vertices, obstacles are blocked vertices
- Finds an optimal path
- Time and space complexity O($h \times w$)



# Maze Routing

**Basic idea -- wave propagation method(Lee, 1961)**

- **Breadth-first search**
- **Back-tracing after finding the shortest path**
- **guarantee to find the shortest path**



# Problems with Maze Routing

Slow: for each net, we have to search N×N grid

Memory: total layout grid needs to be kept NxN

Improvements

- Simple speed-up
- Minimum detour algorithm (Hadlock, 1977)
- Fast maze algorithm (Soukup, 1978)
  - depth-first search until obstacle
  - breadth-first at obstacle
  - until target is reached
- Will find a path if it exists, may be sub-optimal
- Typical speed-up 10-50x

Further improvements

- Maze routing infeasible for large chips
- Line search (Mikami & Tabuchi, 1968; Hightower, 1969)



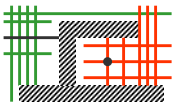
# Line-Probe Algorithms

*Mikami&Tabuchi IFIPS Proc, Vol H47, pp 1475-1478, 1968*

Mikami+Tabuchi's algorithm

- Generate search lines from both source and target (level-0 lines)
- From every point on the level-i search lines, generate perpendicular level-(I+1) search lines
- Proceed until a search line from the source meet a search line from a target
- Will find the path if it exists, not guaranteed to find the shortest path

Time and space complexity: *O(L)*, where L is the number of line segments



# Hightower's algorithm

**Difference: generate search level-(i+1), search lines which are extendable beyond the obstacle.**

**Faster, but does not guarantee a connection**

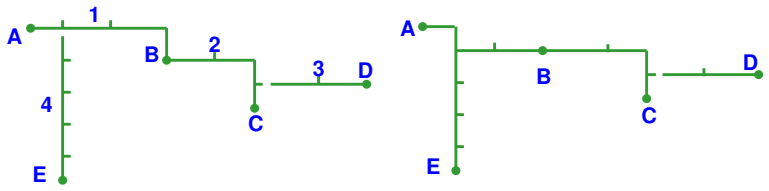


# Connecting a Multi-Terminal Net

**In general, maze and line-probe routing is not suitable to multi-terminal nets**

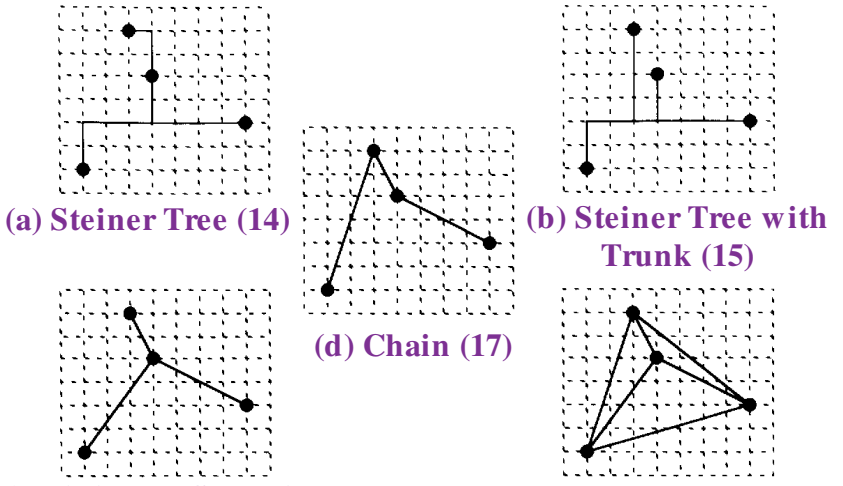
**Several attempts made to extend to multi-terminal nets**

- **Connect one terminal at a time**
- **Use the entire connected paths as source to expand.**
- **Improve the quality of the solution (remove a segment and re-connect)**




- **Results are sub-optimal**
- **Inherit time and memory cost of maze and line-probe algorithms**

# Multi-terminal Nets: Different Routing Options




Cost is determined by routing model



# Steiner Tree Based Algorithms

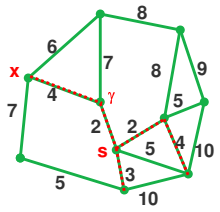
- Tree interconnecting a set of points (demand points, D) and some other (intermediate) points (Steiner points, S)
- If S is empty, Steiner Minimum Tree (SMT) equivalent to Minimum Spanning Tree (MST)
- Finding MST is NP-complete
- Many excellent heuristics
- Constraint of underlying Grid Graph being defined by intersection of horizontal and vertical lines through demand points, leads to Rectilinear Minimum Steiner Tree (RMST) and Rectilinear Steiner Tree RST problems
- Can compute MST and modify to approximate RMST
  - Build MST
  - Rectilinearize each edge of MST

# Minimum Spanning Trees

Given a weighted graph
Find a spanning tree whose weight is minimum

**Prim's algorithm**

start with an arbitrary node S
T←{s}
while T is not a spanning tree
{ find the closet pair x∈V-T, y∈T
  add (x,y) to T

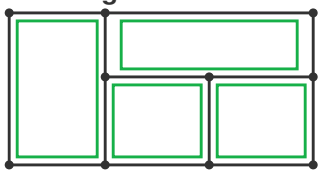



runs in $O(n^2)$ time
very simple to implement
always gives a tree of minimum cost



# Applying Spanning/Steiner Tree Algorithms

- General cell design: channel intersection graphs



- Standard cell/Gate array/Sea-of-gate design rectilinear steiner/spanning trees or grid graphs

# Problems with Sequential Routing Algorithms

Net ordering:

- we have to route net by net, but it is difficult to determine the best net ordering!
- Difficult to predict/avoid congestion

What can be done:

- Use other routers
  - channel/switchbox routers
  - hierarchical routers
- Rip-up and re-route



# Global Routing: Concurrent Approaches

- Can formulate routing problem as *integer programming problem*, solve simultaneously for all nets

**Given**

(i) Set of Steiner trees for each net
(ii) Placement of blocks/cells
(iii) Channel capacities

**Determine**

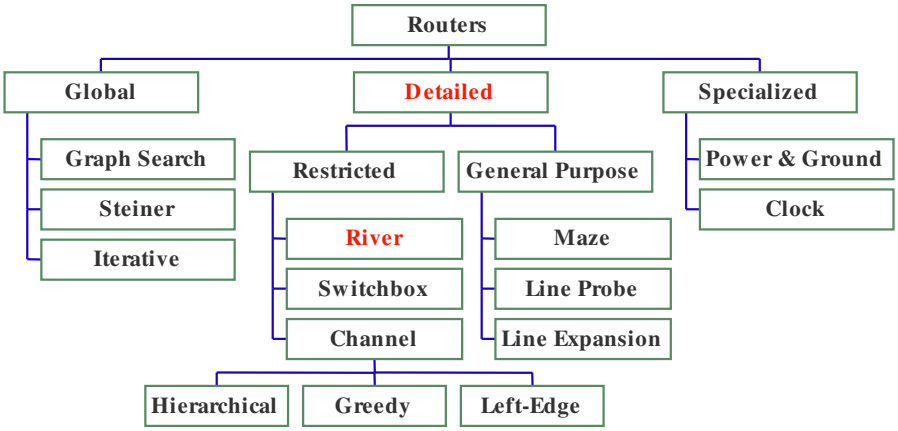
Select a Steiner tree for each net w/o violating channel capacities

**Optimize**

Min total wirelength

- Simulated annealing also can be used

# Taxonomy of VLSI Routers

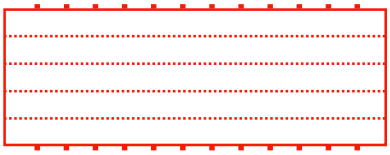



X { gridded, gridless}

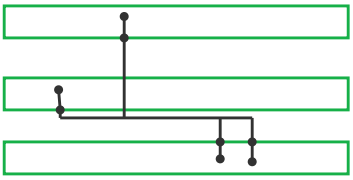
EE244 23

# Gridded and Gridless

- Gridded

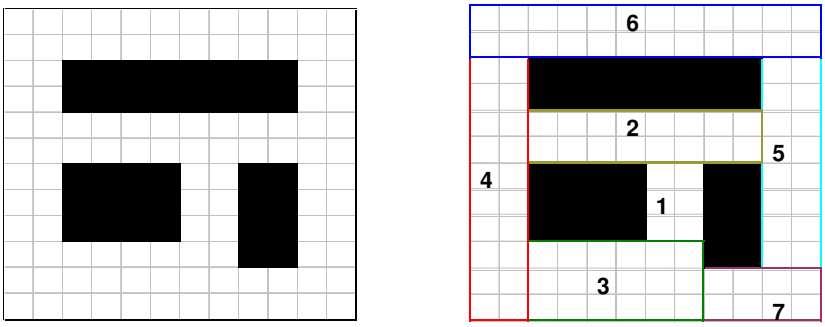


- Gridless aka area based



EE244 24

## Transition to Detailed:Routing Region Ordering

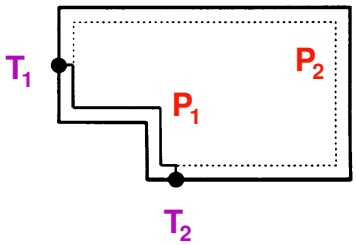



- Determine ordering for detailed routing regions
- Use of Slicing Structure for ordering (R. Otten)
  - for macro-block designs



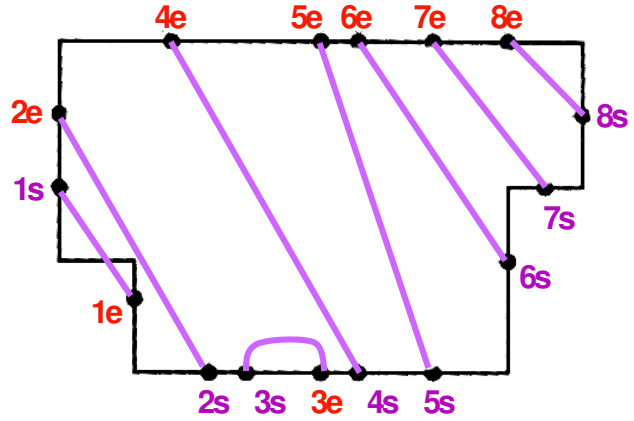
## One Layer Routing: General River-Routing

- For clock, power, ground routing still may need to solve single-layer routing
- Two possible paths per net along boundary
- path is alternating sequence of horizontal and vertical segments connecting two terminals of a net
- Consider starting terminals and ending terminals
- Assume every path counter-clockwise around boundary



# One Layer Routing: General River-Routing

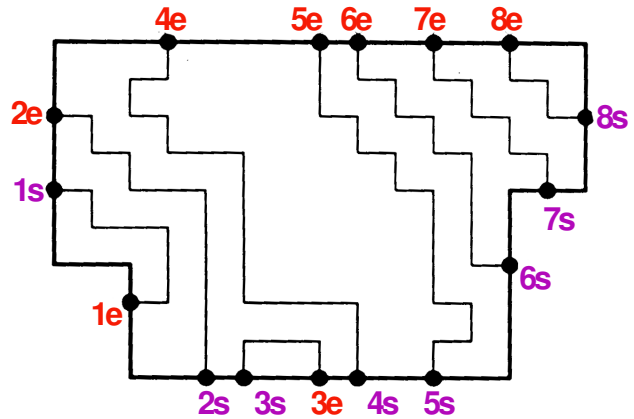
- Create circular list of all terminals ordered counterclockwise according to position on boundary




EE244 27

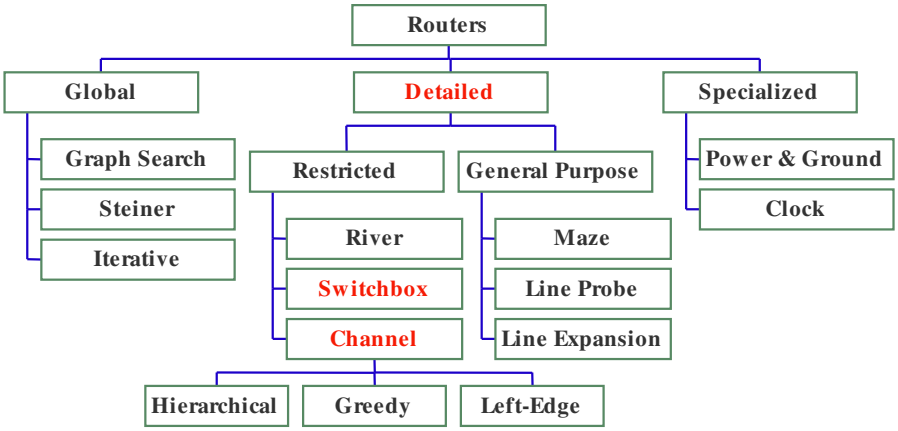
# One Layer Routing: General River-Routing

- Boundary-Packed Solution
- Flip corners to minimize wire length




EE244 28

# Taxonomy of VLSI Routers






# Channel/Switch Box Routing Algorithms

- Graph theory based algorithm
  Yoshimura and Kuh
- Greedy algorithm
  Rivest and Fiduccia

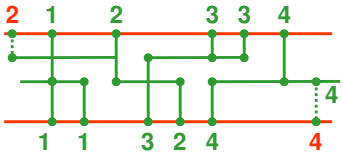
} Channel routing

- Maze routing and its variations
  Lee, Robin, Soukup, Ohtsuki
- Hierarchical wire routing
  Burstein and Pelavin

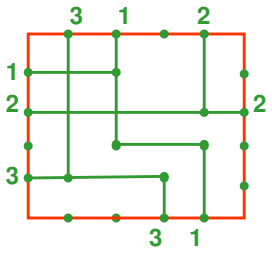
} Channel / switchbox and general area routing

# Channel vs Switchbox

- Channel may have exits at left and right sides, but exit positions are not fixed
- We may map exits to either lower or upper edge of a channel
- One dimensional problem




- Terminal positions on all four sides of a switchbox are fixed
- Two dimensional problem




Switchbox routing is more difficult!



# Channel Routing Problem

Input: Pins on the lower and upper edge

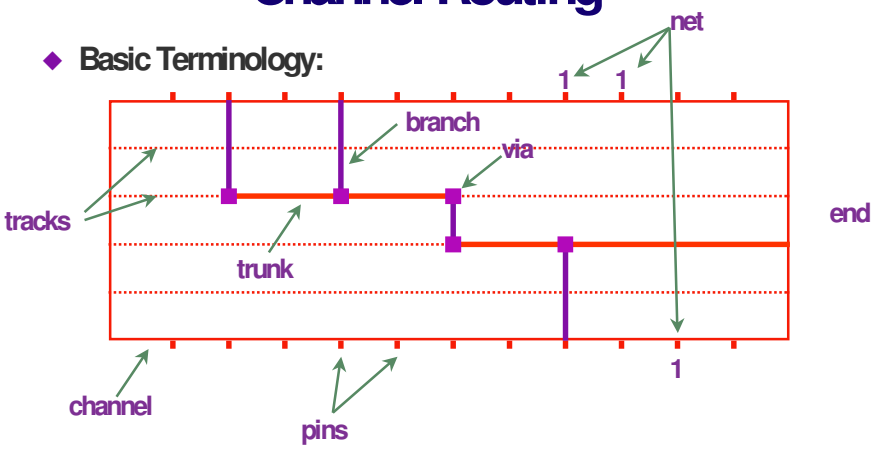
Output: Connection of each net

Constraints (Assumption)
- (i) grid structure
- (ii) two routing layers. One for horizontal wires, the other for vertical wires
- (iii) vias for connecting wires in two layers

Minimize:
- (i) # tracks (channel height)
- (ii) total wire length
- (iii) # vias

# Channel Routing

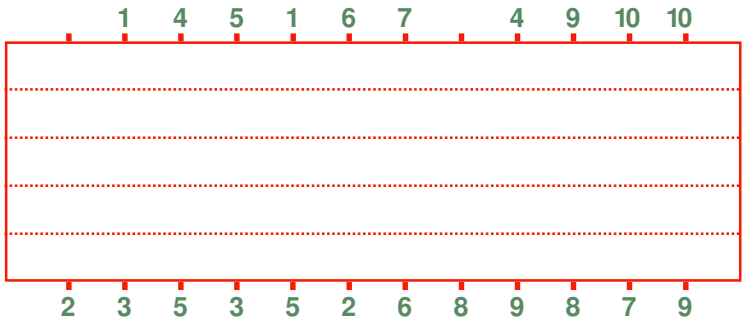
- Basic Terminology:




- Fixed pin positions on top and bottom edges
- Classical channel: no nets leave channel
- Three-sided channel possible



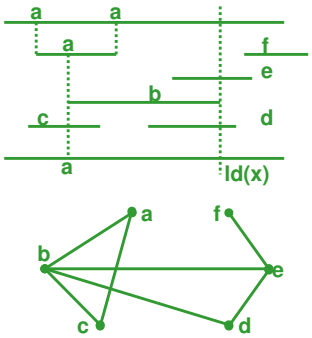
# Channel Routing




- Graph representations:
  - Vertical Constraint Graph (VCG)
  - Horizontal Interval Graph (HIG)

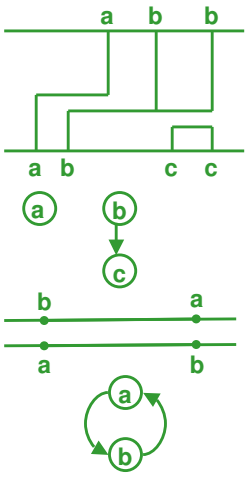
# Horizontal Constraint Graph (HCG)

- Node $v_i$: represents a horizontal interval spanned by net I
- There is an edge b/w $v_i$ and $v_j$ if horizontal intervals overlap
- No two nets with a horizontal constraint may be assigned to the same track
- Maximum clique of HCG establishes a lower bound on # of tracks:
  # tracks ≥ maximum clique of HCG



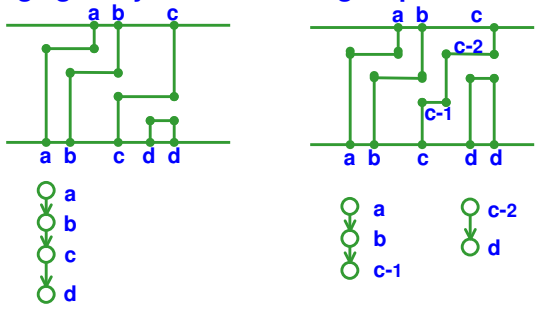

# Vertical Constraint Graph (VCG)

- Node: represents a net
- edge (a1→a2): if at some column,
  net a1 has a terminal on the upper edge
  net a2 has a terminal on the lower edge
  a1→a2 means that net a1 has to be above a2
- Establishes a lower bound:
  # tracks ≥ longest path in VCG
- VCG may have a cycle !

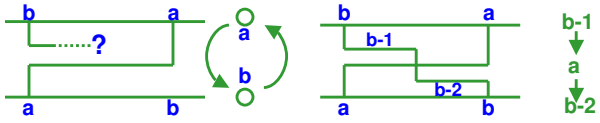


# Doglegs in Channel Routing

- Doglegs may reduce the longest path in VCG



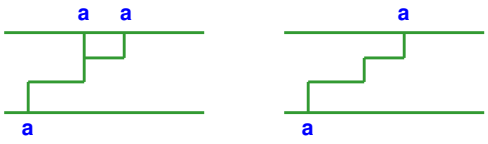

- Doglegs break cycles in VCG





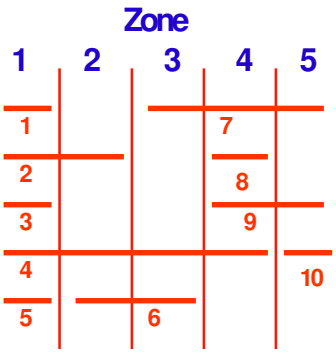

# Doglegs in Channel Routing (Cont'd)

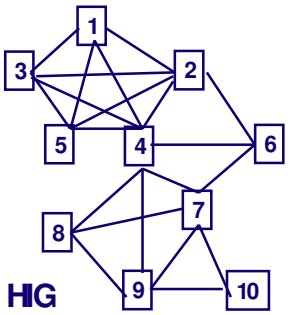
- Restricted Dogleg vs unrestricted dogleg



# Constraint Graph Based Algorithm: "Merging of Nets" (Yoshimura & Kuh)

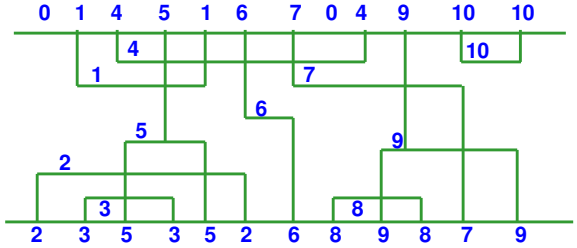
- On the assumption of no cyclic constraints, nets that can be placed on the same track can be merged in the VCG, simplifying the VCG.
- Nets can be organized into zones, further simplifying the problem

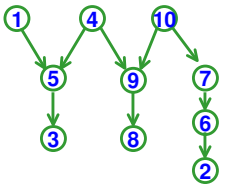


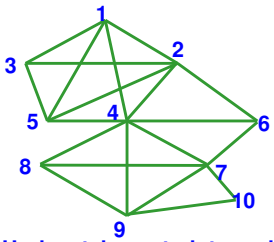

EE244 39

# Characterizing Channel Routing Problem



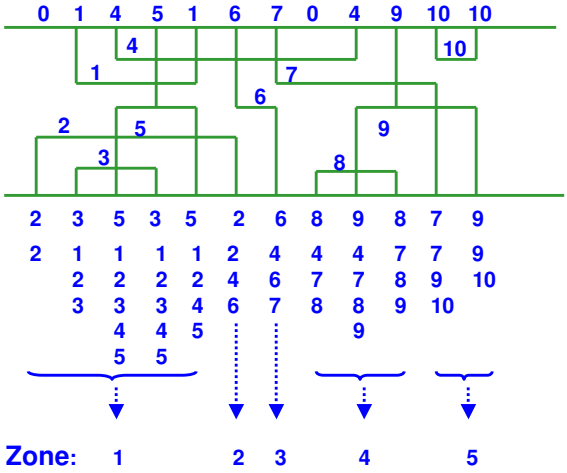

Vertical constraint graph $G_v$




Horizontal constraint graph

The channel routing problem is completely characterized by the vertical constraint graph and the horizontal constraint graph.

## Zone Representation of Horizontal Segments




Zones are maximum cliques in the horizontal interval graph.
Each net *must* be routed on a different track.

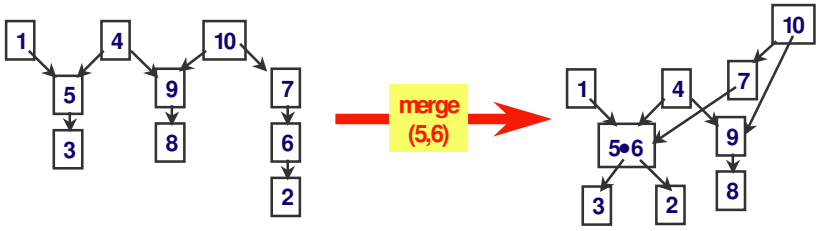


## Merging of Nets

- Definition: Let *i* and *j* be nets for which the following holds:
  - (a) *i* and *j* are not adjacent in the HIG
  - (b) There is no direct path between *i* and *j* in the VCG

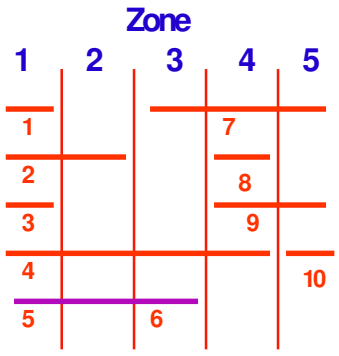
  Then these nets can be assigned to the same track and hence they can be merged in the VCG
- Merging Operation:
  - (1) Combine nodes *i* and *j* into node *i•j* in VCG
  - (2) Update zone representation such that *i•j* occupies zones of *i* and *j*

# Merging of Nets: Example

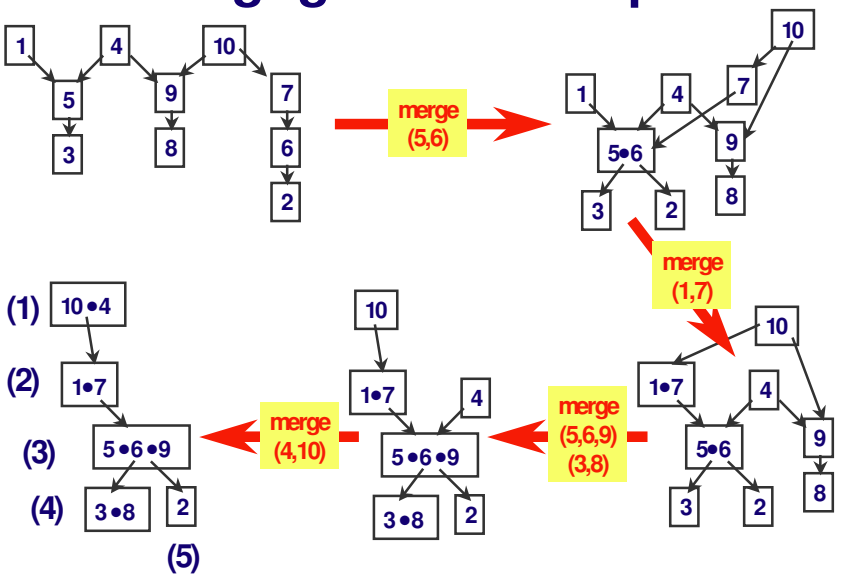





# Updating of Zone Representation

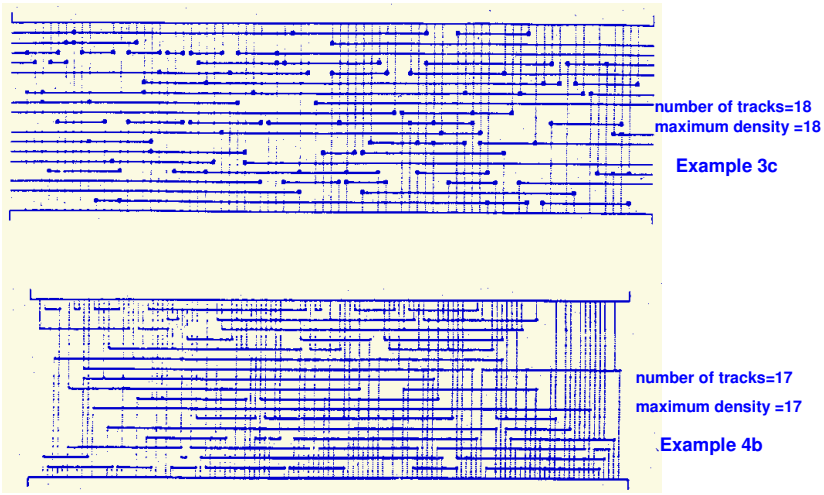


# Merging of Nets: Example

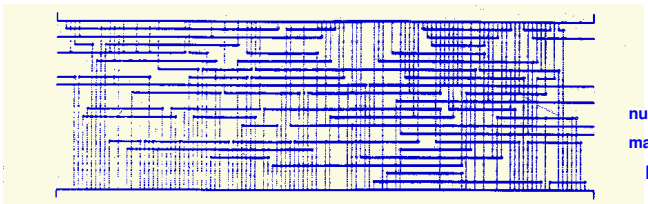





# Routing Examples by Y-K's Algorithm



# Routing Examples by Y-K's Algorithm
(Cont'd)



number of tracks=20
maximum density =20
Example 5

Deutsch's Difficult example without dogleg

number of tracks=28
maximum density =19



# Yoshimura and Kuh's Method

**Source:**

**"Efficient Algorithms for Channel Routing"**

**by T. Yoshimura and E. Kuh**

***IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems.***

**Vol. CAD-1, pp25-35, Jan 1982**

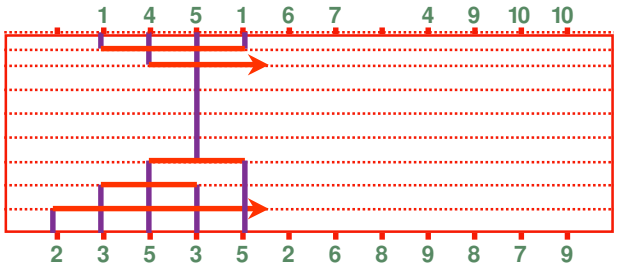
# Greedy Channel Router

*R.L. Rivest and C.M. Fiduccia " A Greedy Channel Router", 19th DAC, 1982 P418-424*

- A simple linear time algorithm
- Guarantee the completion of all the nets (may extend to right-hand side of the channel)
- Produce both restricted doglegs and unrestricted doglegs



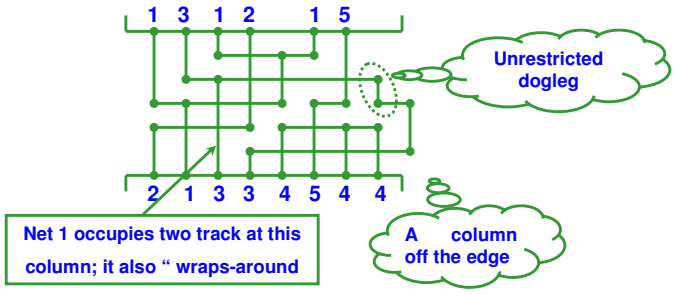
# Greedy Router: Rivest & Fiduccia




- Proceed column by column (left to right)
- Make connections to all pins in that column
- Free up tracks by collapsing as many tracks as possible to collapse nets
- Shrink range of rows occupied by a net by using doglegs
- If a pin cannot enter a channel, add a track
- O(pins) time

# Comments on Greedy Router
## (Rivest&Fiduccia 1982)

- Always succeeds (even if cyclic conflict is present);
- Allows unrestricted doglegs;
- Allows a net to occupy more than 1 track at a given column;
- May use a few columns off the edge;





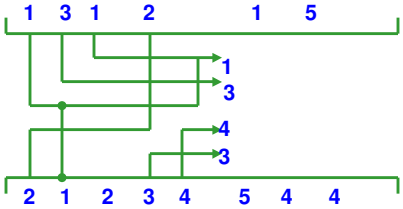

# Overview of Greedy Router

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

```
c:=0;
while (not done) do
    begin
        c:=c+1;
        complete wiring at column c;
    end;
```

In general, at a point in a net may be

(1) empty (net 5)
(2) unsplit (nets 1,4)
(3) split (net 3)
(4) completed (net2)



# Parameters to Greedy Router

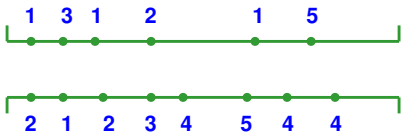
- Initial-channel-width icw
- Minimum-jog-length mjl (Why?)
- Steady-net-constant snc
- Usually start icw as d. the density
- Mjl controls the number of vias, use a large mjl for fewer vias
- Snc also controls # of vias (typical value=10 Why?)



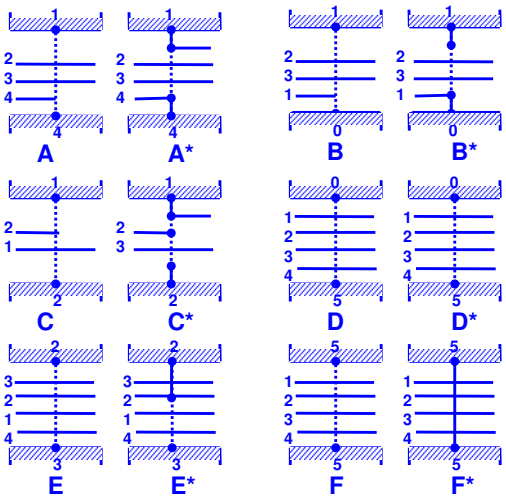
# Operations at Each Column

At each column, the greedy router tries to maximize the utility of the wiring produced:

A: Make minimal feasible top/bottom connections;
B: Collapse (connect) split nets;
C: Move split nets closer to one another;
D: Raise rising nets/lower falling nets, I.e. bring nets closer to destination terminal;
E: Widen channel when necessary;
F: Extend to next column;

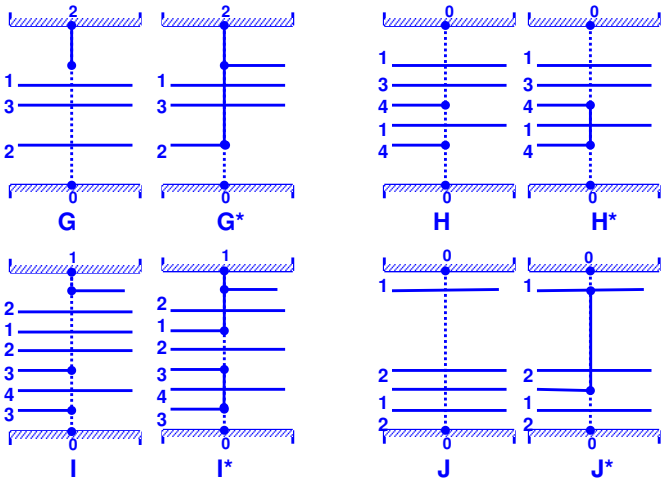


## (A) Make Minimal Feasible top/bottom Connetions

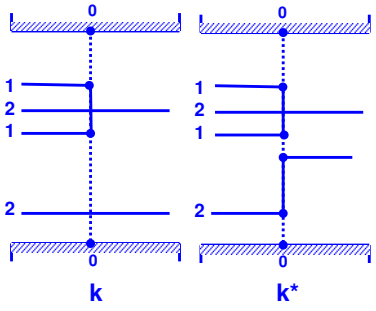





## ( B ) Collapse/(Connect) Split Nets

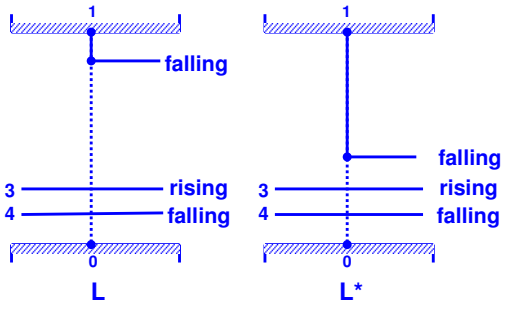


# ( C ) Move Split Nets Closer

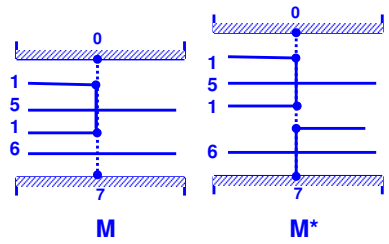





# (D) Rising/Falling

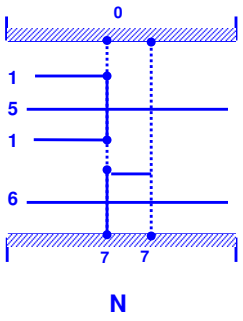


# (E) Insert New Track

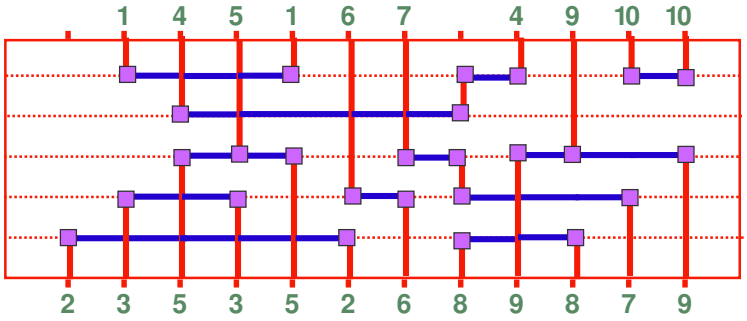





# (F) Extend to Next Column



# Greedy Routing Example






# Experimental Results

- Runs very fast
- 20-track solution to the Deutsch's Difficult Example

# Current Status

- All router technologies discussed in this lecture are employed
- Current focus is on using ungridded/area-based routers to do detailed routing
- Chip-level routing is a top concern. Performance is limited by chip-level nets
- Timing driven routing is important
- Safe (noise resistant/immune) routing is another important topic



# Extras

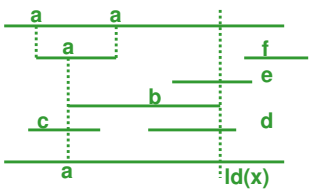
# Greedy Router

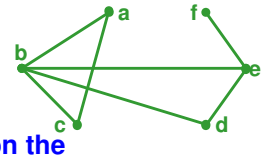
- Three Parameters Control Algorithm:
  - Initial_Channel_Width, Minimum_Jog_length, Steady_Net_Constant
- Rule-Based Approach
  - Rule 1: Make feasible top and bottom connections in minimal manner
  - Rule 2: Free up as many tracks as possible by collapsing split nets
  - Rule 3: Add jogs to minimize distance between split nets (but no shorter than minimum_jog_length)
  - Rule 4: Add jogs where possible to raise rising nets and lower falling nets
  - Rule 5: Widen channel if needed to make top or bottom connections. Add tracks in the middle of the channel
  - Rule 6: Extend the channel to a new column to complete unconnected net segments.



# Channel Density

- Local density at column C
  ld(C) = # nets split by column C
- Channel Density
  d = max ld( C )
  all C
- Each net spans over an interval
- Horizontal Constraint Graph(HCG)
  an *undirected* graph such that:
  vertex : net
  edge: <n_j, n_k>, if intervals I_j, I_k intersect
  I.e. two nets would overlap if placed on the same track
- Size of max clique in HCG= channel density
- A lower bound:
  # tracks ≥ channel density



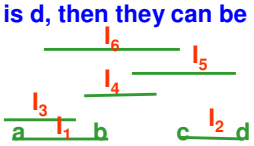



# Interval Packing

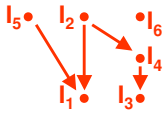
**Thm: If the density of a set of intervals is d, then they can be packed into d tracks.**

**Proof: $I_1=(a,b)$ $I_2=(c,d)$**

**Define: $I_1<I_2$ iff $b<c$ or $I_1=I_2$**




- **reflective: $I_1<I_1$**
- **anti-symmetric: $I_1<I_2$, $I_2<I_1 \Rightarrow I_1=I_2$**
- **transitive: $I_1<I_2$, $I_2<I_3 \Rightarrow I_1<I_3$**




- **The interval set with binary relation < forms a partially ordered set (POSET)!!**
  **Intervals in a track⇔ they form a chain**
  **Intervals intersecting a common column ⇒ antichain**

**Dilworth's theorem (1950): If the max antichain of a POSET is d, then the POSET can be partitioned into d chains**



# Left-Edge Algorithm for Interval Packing

**Repeat**

  **create a new track t**

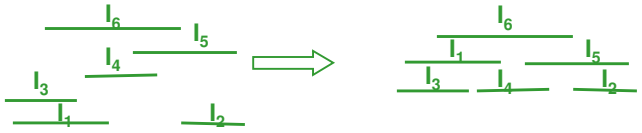
  **Repeat**

    **put leftmost feasible interval to t**

  **until no move feasible interval**

**until no move interval**

**Interval are sorted according to their left endpoints**




**O(nlogn) time algorithm. Greedy algorithm works!**

# References


- Introduction to Algorithms, T. Cormen, C. Lesierson, R. Rivest, The MIT Press, Second Printing, 1996.
  Chapter 24, Minimum Spanning Trees 503-513

- Winter, P. " Steiner Problem in Networks: A Survey" Networks, Vol. 17, 1987

- Cong, He, Koh, Madden, "Performance Optimization of VLSI Interconnect Layout", Integration, the VLSI Journal, Vol. 21 ,1996, pp.1-94 (Sect.3)