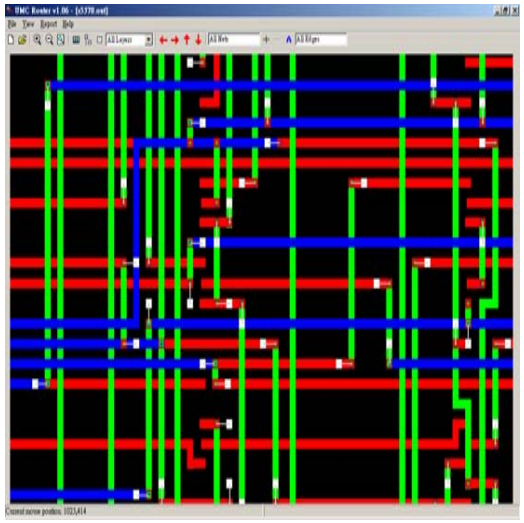
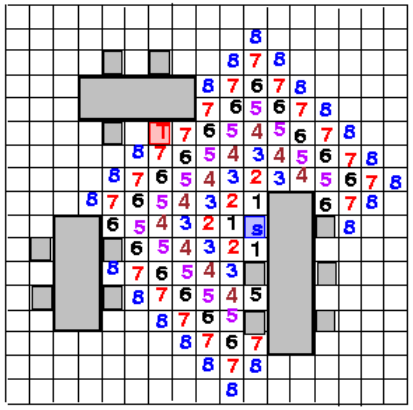
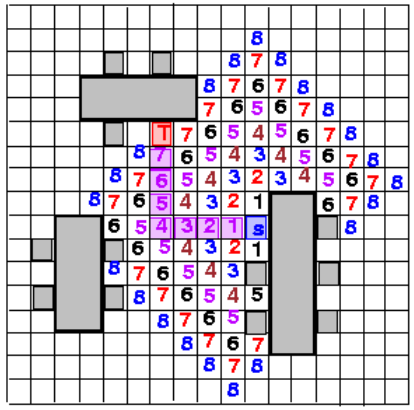
Unit 6: Routing

- Course contents:
  - Maze/A*-search routing
  - Global routing
  - Routing trees
  - Channel routing
  - Full-chip routing
- Readings
  - W&C&C: Chapter 12
  - S&Y: Chapters 5, 6, and 7





Filling



Retrace

Routing

placement

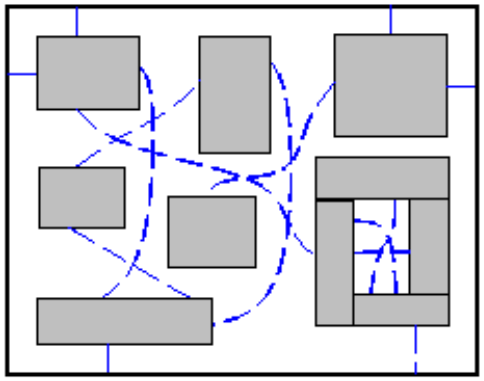
- Generates a "loose" route for each net.
- Assigns a list of routing regions to each net without specifying the actual layout of wires.

global routing

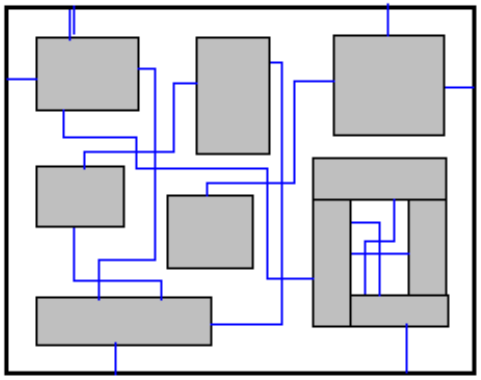
detailed routing

- Finds the actual geometric layout of each net within the assigned routing regions.

compaction



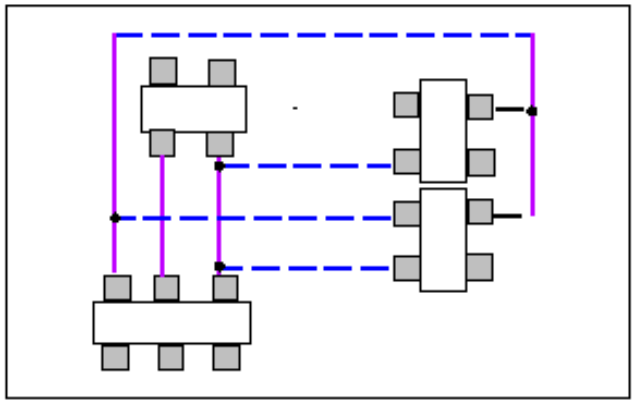
Global routing



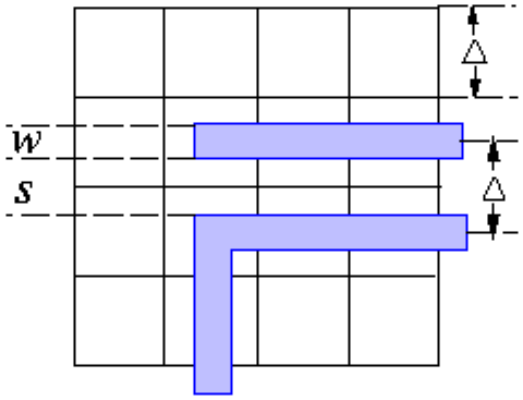
Detailed routing

Routing Constraints

- 100% routing completion + area minimization, under a set of constraints:
  - Placement constraint: usually based on fixed placement
  - Number of routing layers
  - Geometrical constraints: must satisfy design rules
  - Timing constraints (performance-driven routing): must satisfy delay constraints
  - Signal integrity constraint: Crosstalk?
  - Manufacturability & reliability: CMP, MPL, EUV, antenna effect?

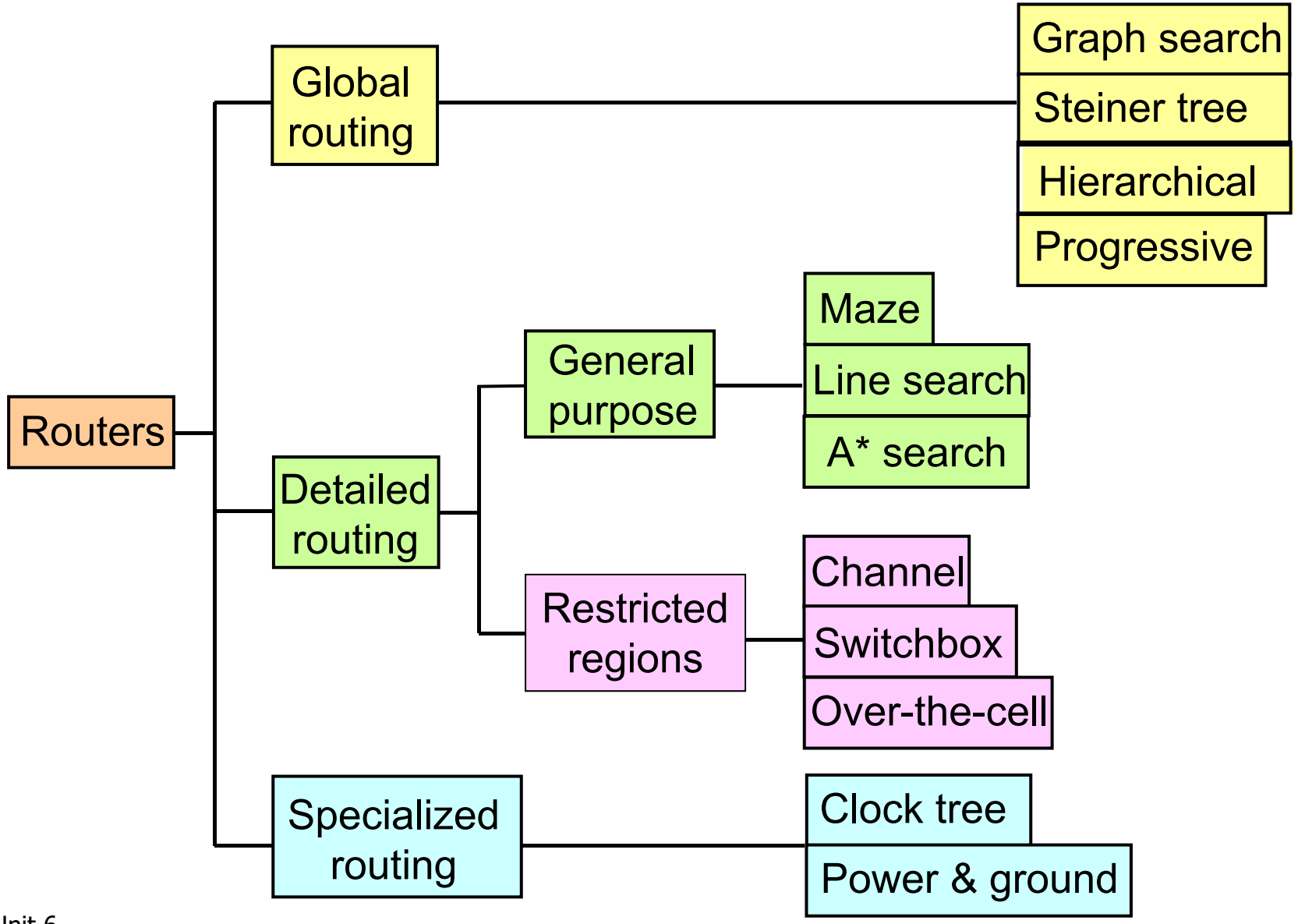


Two–layer routing



Geometrical constraint

Classification of Routers

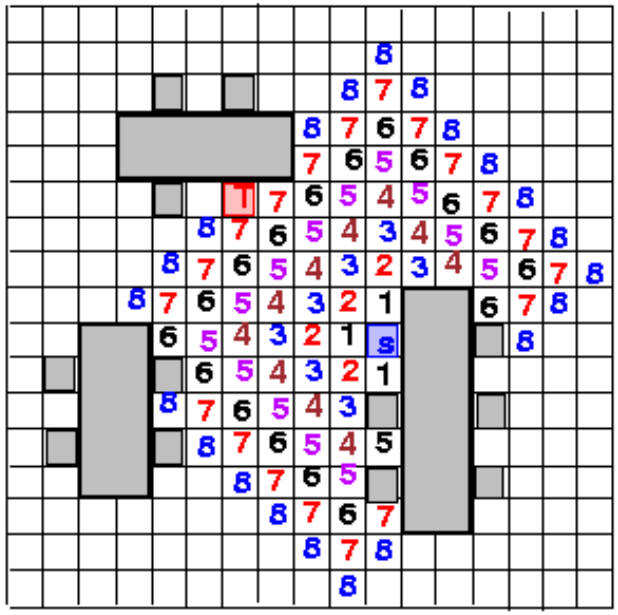


Maze Router: Lee Algorithm

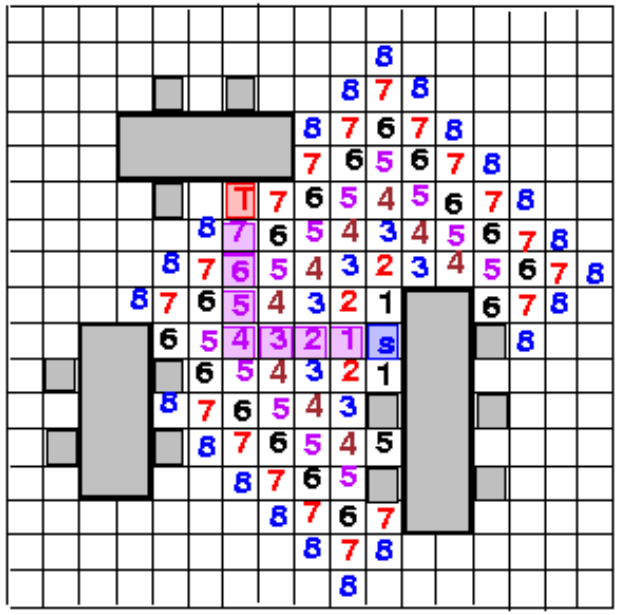
- Lee, “An algorithm for path connection and its application,” *IRE Trans. Electronic Computer*, EC-10, 1961.
- Discussion mainly on single-layer routing
- **Strengths**
  - Guarantee to find connection between 2 terminals if it exists.
  - Guarantee minimum path.
- **Weaknesses**
  - Requires large memory for dense layout
  - Slow
- Applications: global routing, detailed routing

Lee Algorithm

- Find a path from *S* to *T* by “wave propagation”.



Filling

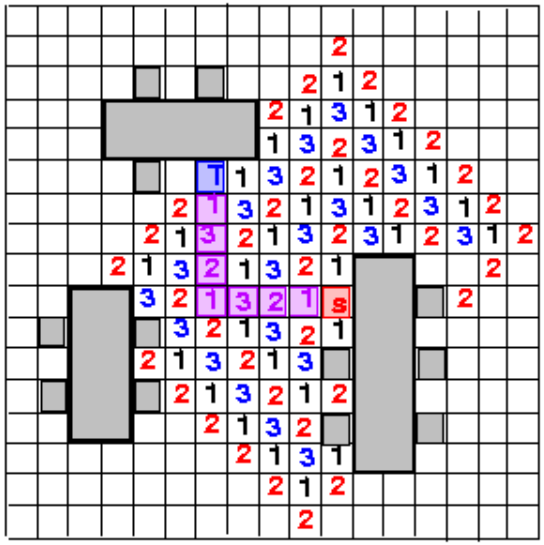


Retrace

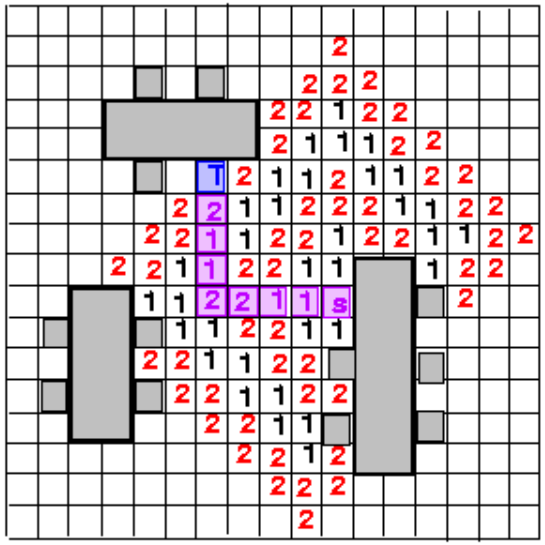
- Time & space complexity for an $M \times N$ grid: $O(MN)$ (**huge!**)

Reducing Memory Requirement

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



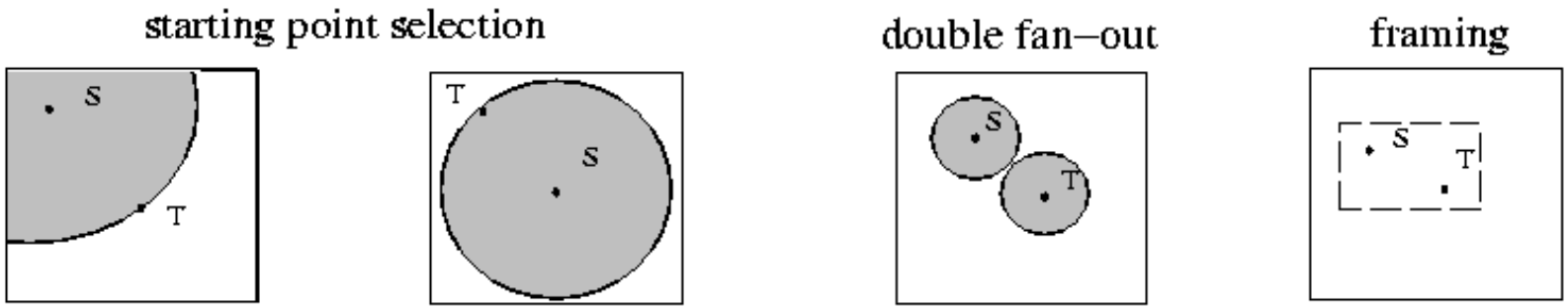
Sequence: 1, 2, 3, 1, 2, 3, ...



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

Reducing Running Time

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

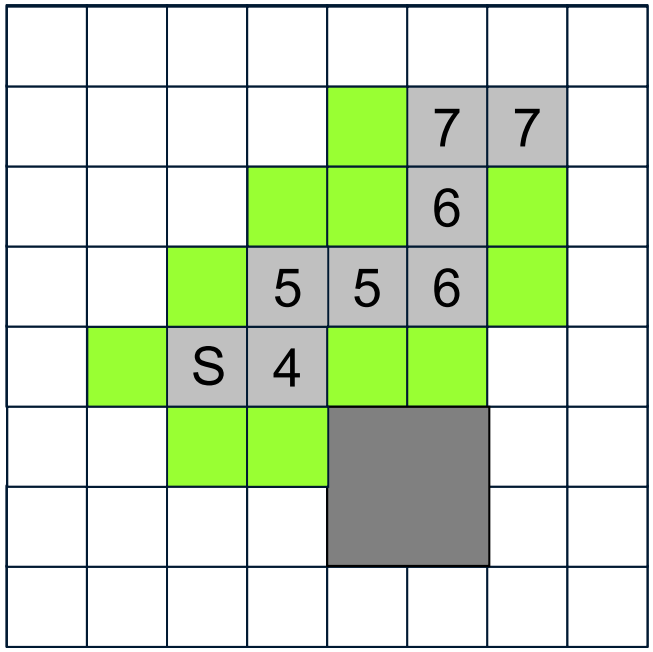


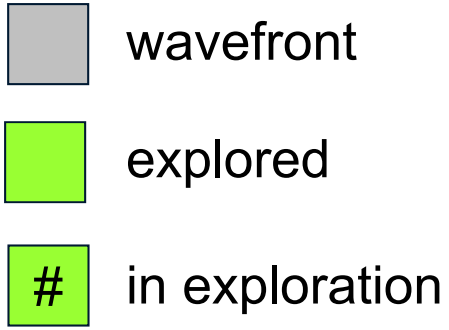
A*-Search Routing

- Maze search is also called **blind search** because it searches the routing region in a blind way.
- Dijkstra's shortest path algorithm is **BFS-based**
  - Might waste time/memory in inferior paths, considering only real costs
- A*-search is also called **best-first search**, which uses function $f(x) = g(x) + h(x)$ to evaluate the cost of a path x
  - $g(x)$: the cost from the source to the current node of x
  - $h(x)$: the *estimated* cost from the current node of x to the target
- A*-search first searches the routes that is most likely to lead towards the target.
  - BFS is a special case of A*-search where $h(x) = 0$ for all x
- Property: If $h(x)$ is ***admissible***, then A*-search is optimal
  - Admissibility: the router never overestimates the actual cost from the current node to the target

Example of A*-Search Routing

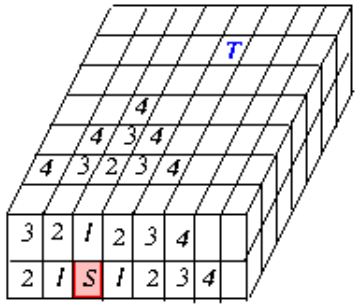
- Cost function for a path x: $f(x) = g(x) + h(x)$
  - $g(x)$: the label from the source S to the current node of x (*i.e.*, the label used in maze routing)
  - $h(x)$: $\max(\text{dist}_x(T, x), \text{dist}_y(T, x))$



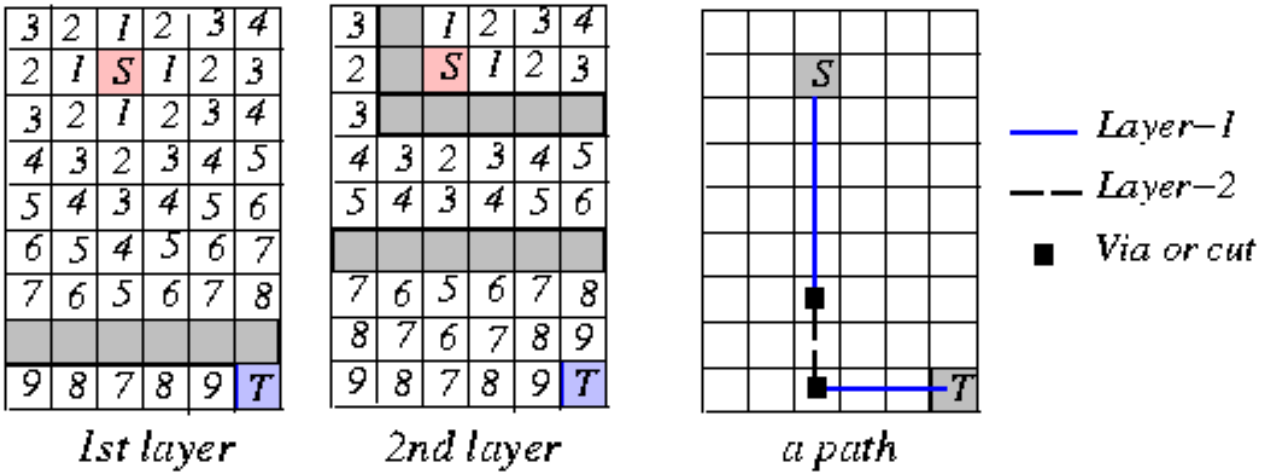


Multi-layer Routing

- 3-D grid:

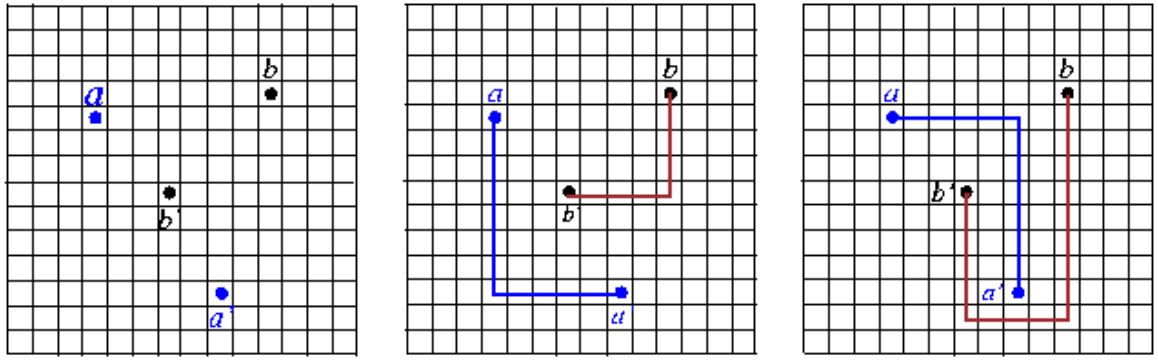


- Two planner arrays:
  - Neglect the weight for inter-layer connection through via.
  - Pins are accessible from both layers.

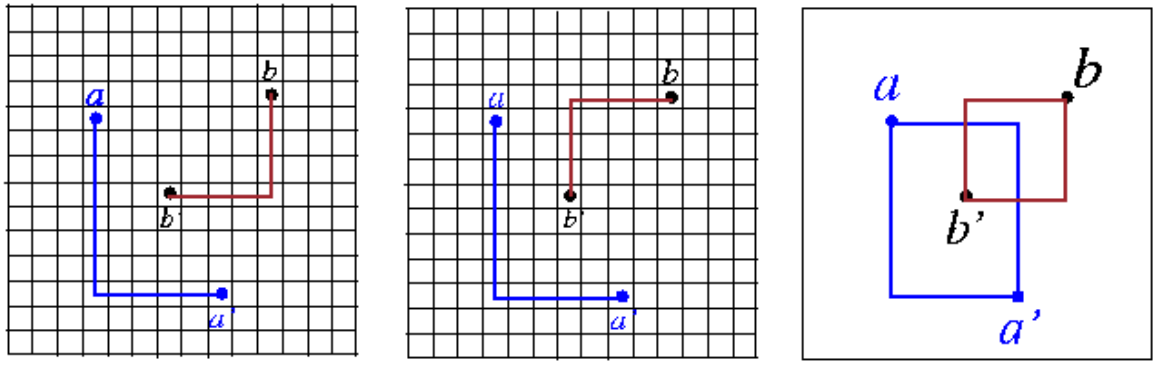


Net Ordering

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



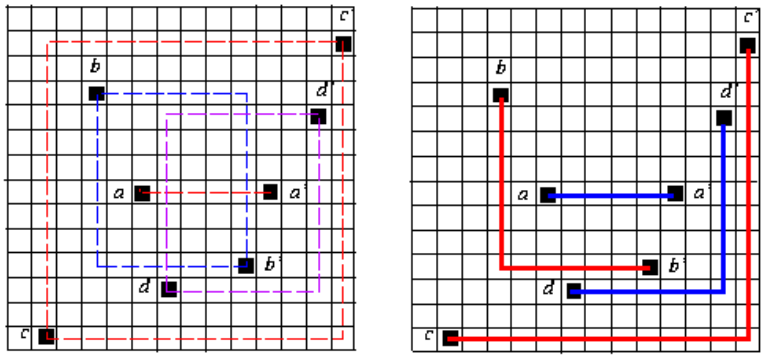
route net a before net b



route net b before net a

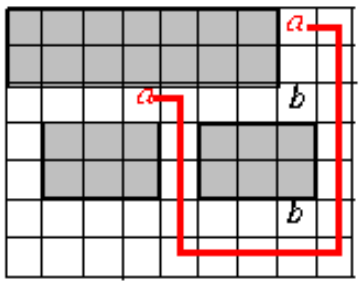
Net-Ordering Handling

- Order the nets in the ascending order of the # of pins within their bounding boxes.
- Order the nets in the ascending (or descending??) order of their lengths.
- Order the nets based on their timing criticality.

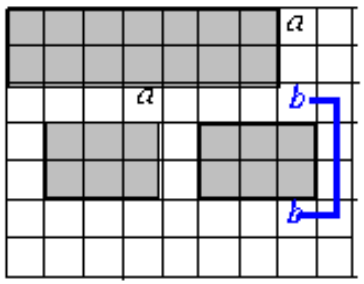


routing ordering: a (0) –> b (1) –> d (2) –> c (6)

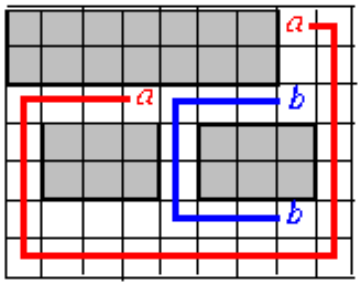
- A mutually intervening case:



a prevents routing of b



b prevents routing of a



a feasible routing

Rip-Up and Re-routing

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

INR (Iterative Negotiation-based Ripping-up/rerouting)

- Proposed in **PathFinder** [McMurchie and Ebeling, FPGA'95] to reduce overflows
- Spreads congested wires iteratively
- At the *i*-th iteration, the cost of a global-routing edge *e*:

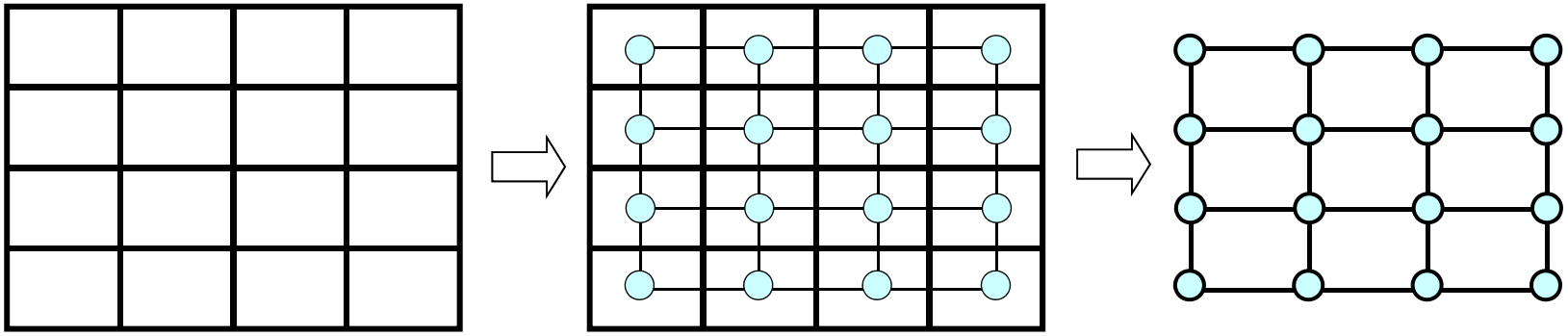
$$(b_e + h_e^{(i)}) \cdot p_e$$

  - b_e: base cost of using *e* (say, delay and/or length of *e*)
  - p_e: # of nets passing *e*
  - $h_e^{(i)}$: historical cost on *e*, $h_e^{(i)} = \begin{cases} h_e^{(i-1)} + 1 & \text{if } e \text{ has overflow} \\ h_e^{(i-1)} & \text{otherwise} \end{cases}$

- Used by recent academic global routers, Archer [ICCAD'07], BoxRouter [ICCAD'07], FastRoute [ICCAD'06], FGR [ICCAD'07], NCTUgr [ASP-DAC'09], NTHU-Route [ICCAD'08] , NTUgr [ICCAD'08], SGR [DAC'10], etc.
- INR may get stuck as the number of iterations increases [Ozdal, ICCAD'07] [Gao et al., ASP-DAC'08]

Modeling and Global Routing Graph

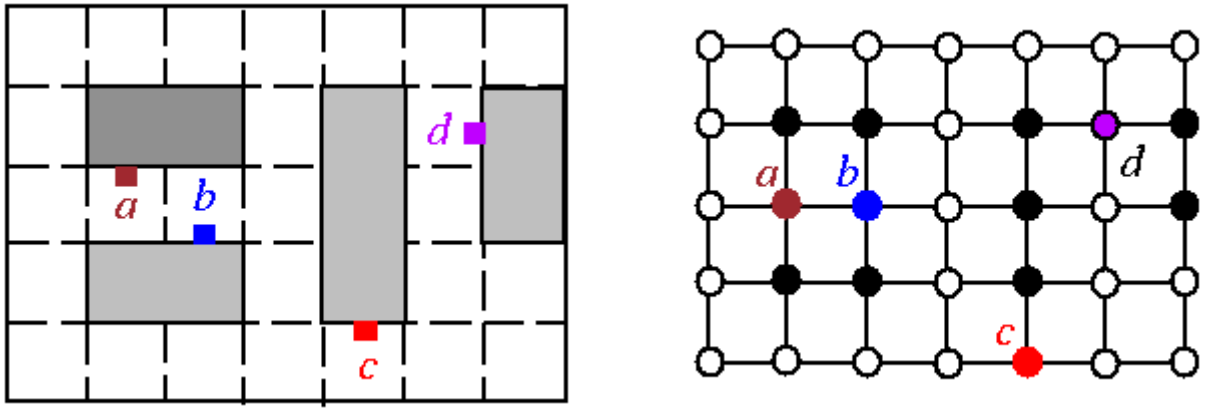
- Often model a chip as a graph such that the graph topology can represent the routing structure and resource in the chip
- Global routing graph
  - Each cell is represented by a vertex.
  - Two vertices are joined by an edge if the corresponding cells are adjacent to each other.



Partitioned Layout | **Resource Modeling** | **Global Routing Graph**

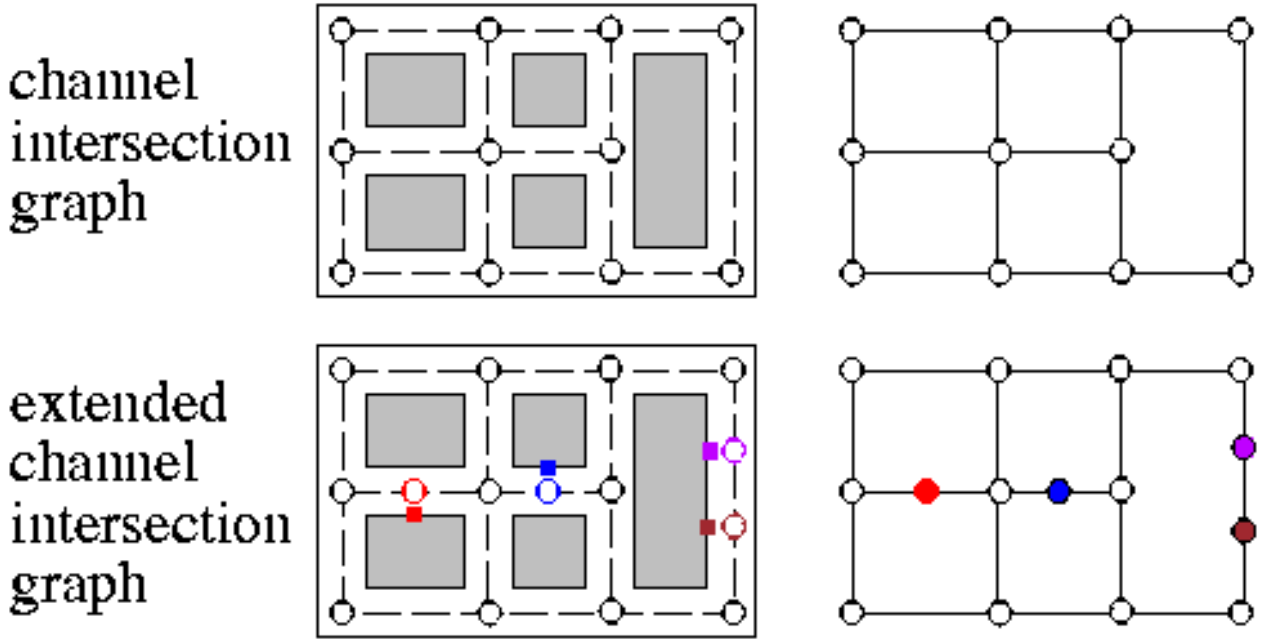
Model for Global Routing: Grid Graph

- Each cell is represented by a vertex.
- Two vertices are joined by an edge if the corresponding cells are adjacent to each other.
- The occupied cells are represented as filled circles, whereas the others are as clear circles.



Model: Channel Intersection Graph

- Channels are represented as edges.
- Channel intersections are represented as vertices.
- Edge weight represents channel capacity.
- Extended channel intersection graph: terminals are also represented as vertices.

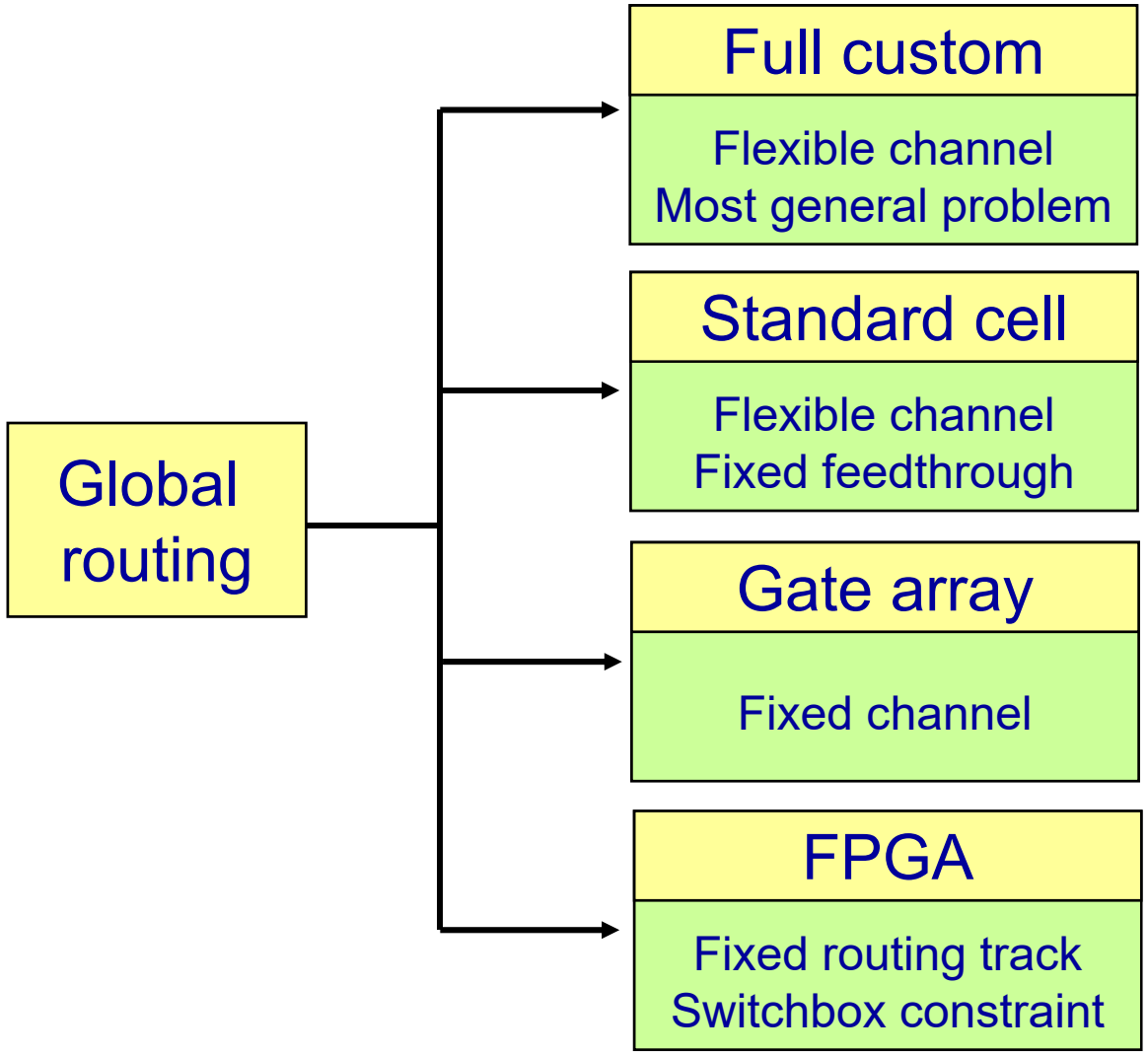


Global-Routing Problem

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

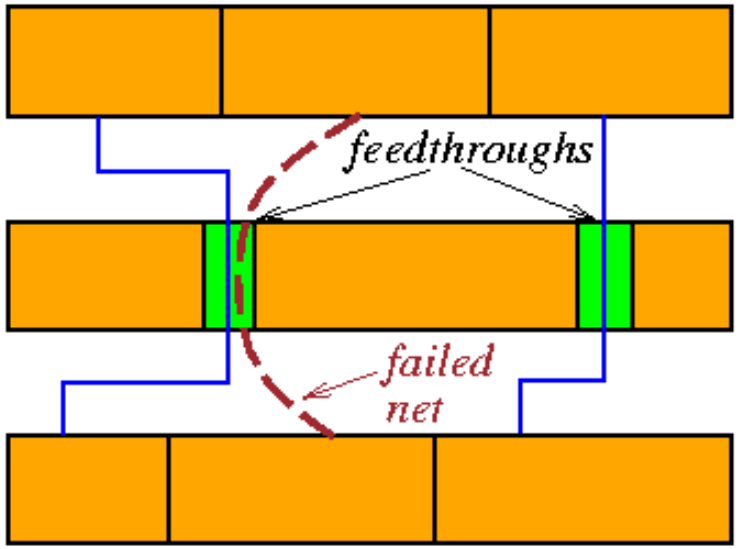
- For high performance, the maximum wirelength ($\max_{i=1}^{n} L(T_i)$) is minimized (or the longest path between two points in T_i is minimized).

Global Routing in Different Design Styles



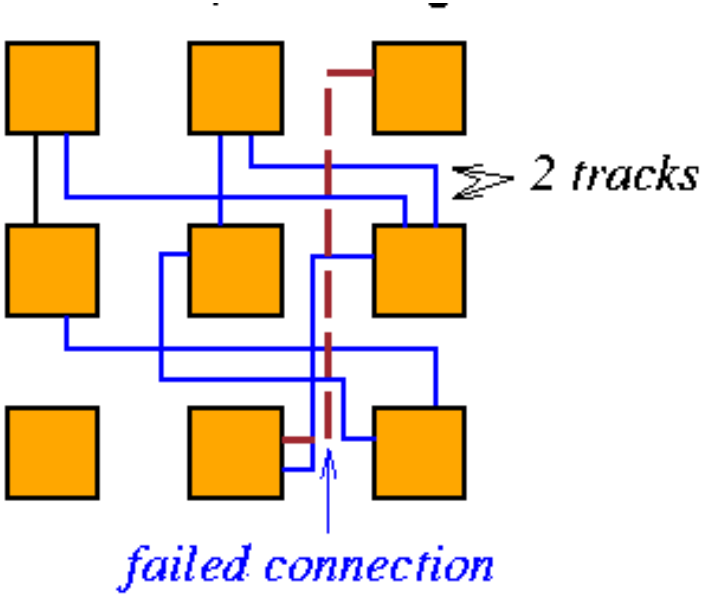
Global Routing in Standard Cells

- Objective
  - Minimize total channel height and total wirelength (over-the-cell routing).
  - Assignment of **feedthrough**: Placement? Global routing?
- For high performance,
  - Minimize the maximum wire length.
  - Minimize the maximum path length.



Global Routing in Gate Arrays

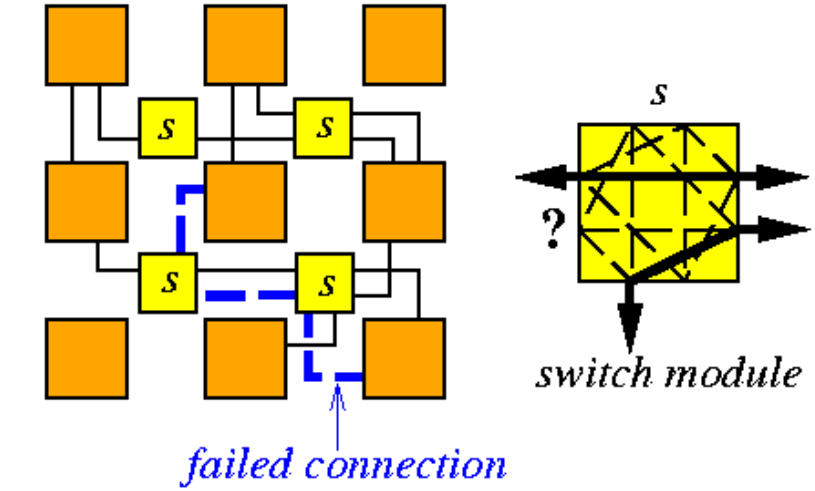
- Objective
  - **Guarantee 100% routability.**
- For high performance,
  - Minimize the maximum wire length.
  - Minimize the maximum path length.



Each channel has a capacity of 2 tracks.

Global Routing in FPGAs

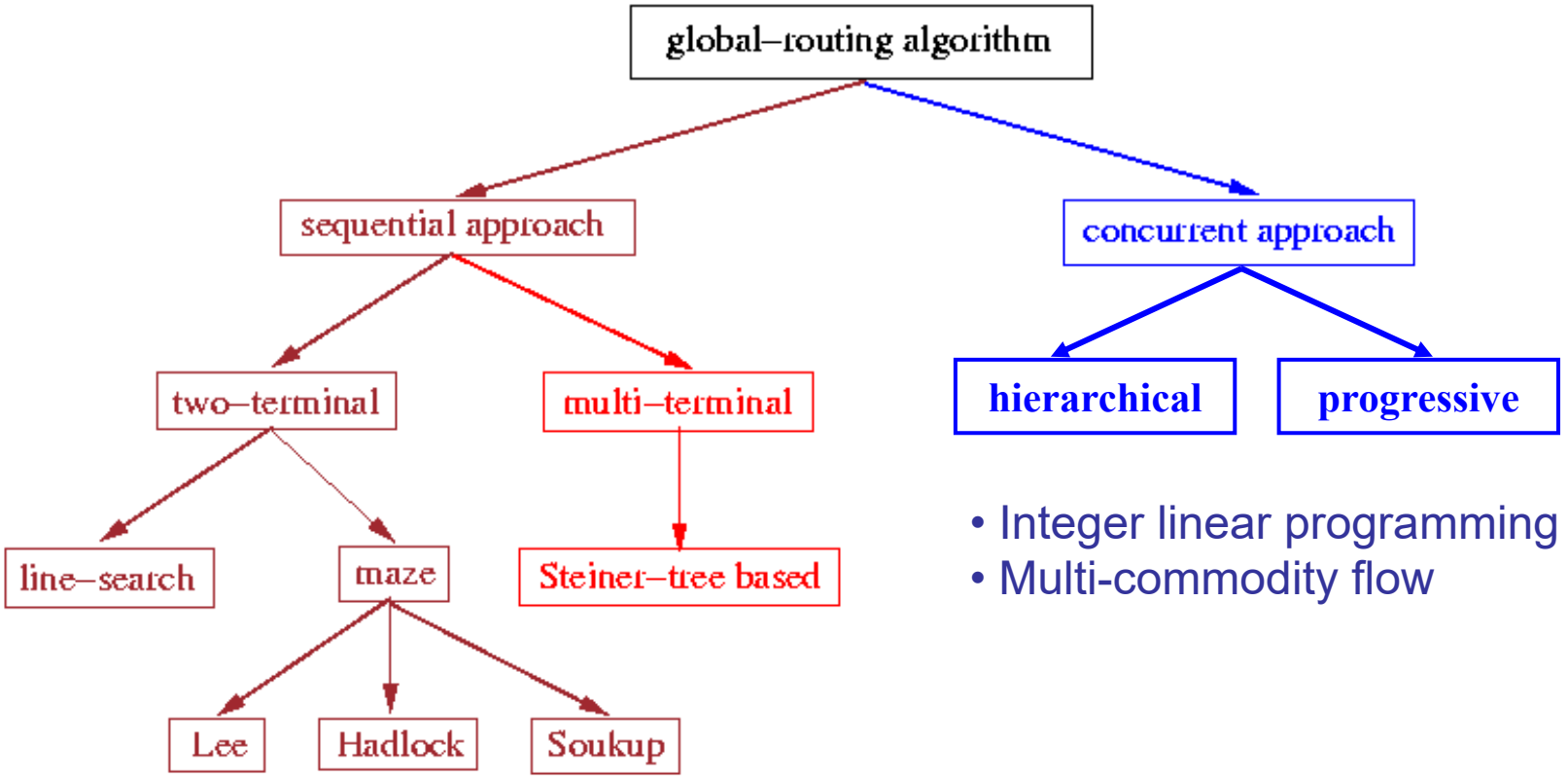
- Objective
  - Guarantee 100% routability.
  - Consider **switch-module architectural constraints.**
- For performance-driven routing,
  - **Minimize # of switches used.**
  - Minimize the maximum wire length.
  - Minimize the maximum path length.



Each channel has a capacity of 2 tracks.

Classification of Global-Routing Algorithms

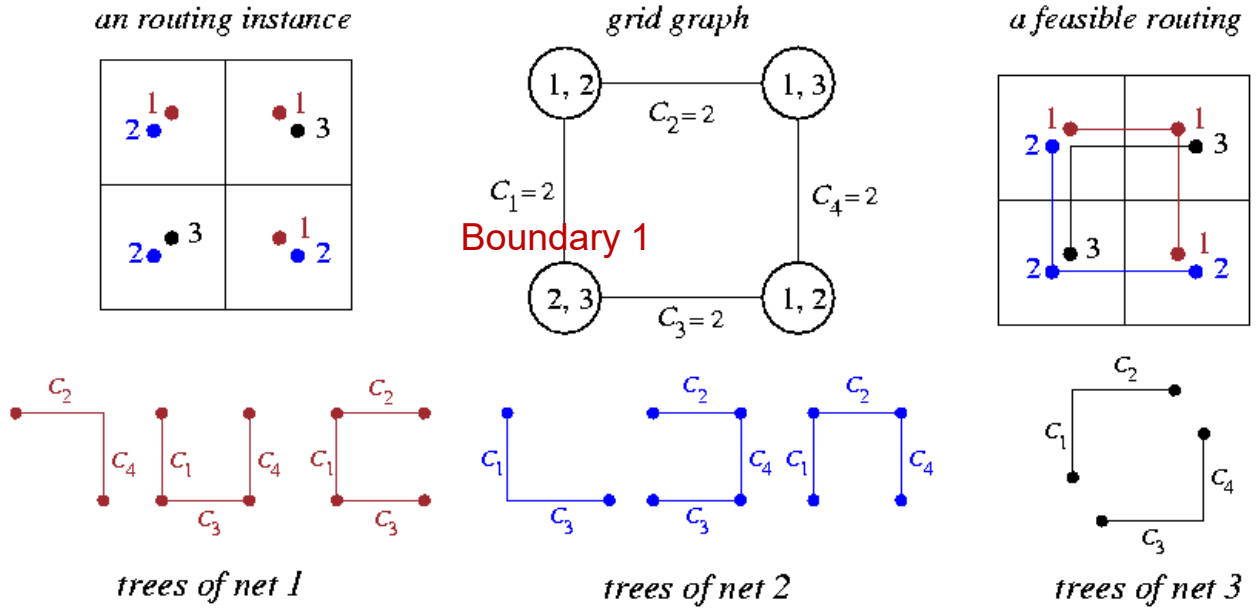
- **Sequential approach:** Assigns priority to nets; routes one net at a time based on its priority (net ordering?).
- **Concurrent approach:** All nets are considered at the same time (complexity?)



- Integer linear programming
- Multi-commodity flow

Global Routing by Integer Programming

- Suppose that for each net i, there are n_i possible trees t_1^i, t_2^i, …, t_{ni}^i to route the net.
- Constraint I: For each net i, only one tree t_j^i will be selected.
- Constraint II: The capacity of each cell boundary c_i is not exceeded.
- Minimize the total tree cost.
- **Question:** Feasible for practical problem sizes?
  - **Keys:** Hierarchical approach, progressive approach, multilevel framework



An Integer-Programming Example

Boundary	t_1^1	t_2^1	t_3^1	t_1^2	t_2^2	t_3^2	t_1^3	t_2^3
B1	0	1	1	1	0	1	1	0
B2	1	0	1	0	1	1	1	0
B3	0	1	1	1	1	0	0	1
B4	1	1	0	0	1	1	0	1

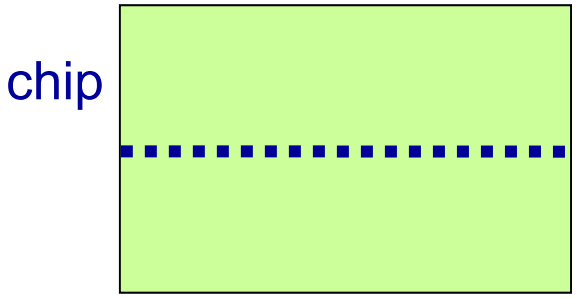
- $g_{i,j}$: cost of tree $t_j^i \Rightarrow$ $g_{1,1}$=2, $g_{1,2}$=3, $g_{1,3}$=3, $g_{2,1}$= 2, $g_{2,2}$=3, $g_{2,3}$=3, $g_{3,1}$=2, $g_{3,2}$=2.

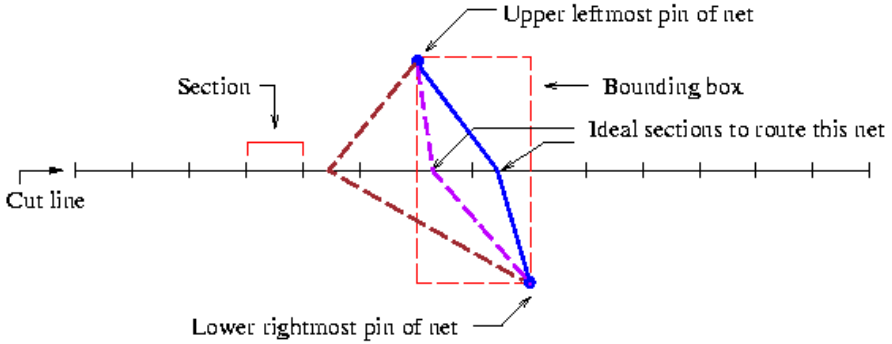
Minimize $2x_{1,1} + 3x_{1,2} + 3x_{1,3} + 2x_{2,1} + 3x_{2,2} + 3x_{2,3} + 2x_{3,1} + 2x_{3,2}$
subject to

$$
\begin{aligned}
x_{1,1} + x_{1,2} + x_{1,3} &= 1 \quad (Constraint\ I : t^1) \\
x_{2,1} + x_{2,2} + x_{2,3} &= 1 \quad (Constraint\ I : t^2) \\
x_{3,1} + x_{3,2} &= 1 \quad (Constraint\ I : t^3) \\
x_{1,2} + x_{1,3} + x_{2,1} + x_{2,3} + x_{3,1} &\leq 2 \quad (Constraint\ II : B1) \\
x_{1,1} + x_{1,3} + x_{2,2} + x_{2,3} + x_{3,1} &\leq 2 \quad (Constraint\ II : B2) \\
x_{1,2} + x_{1,3} + x_{2,1} + x_{2,2} + x_{3,2} &\leq 2 \quad (Constraint\ II : B3) \\
x_{1,1} + x_{1,2} + x_{2,2} + x_{2,3} + x_{3,2} &\leq 2 \quad (Constraint\ II : B4) \\
x_{i,j} &= 0, 1, 1 \leq i, j \leq 3
\end{aligned}
$$

Hierarchical Global Routing

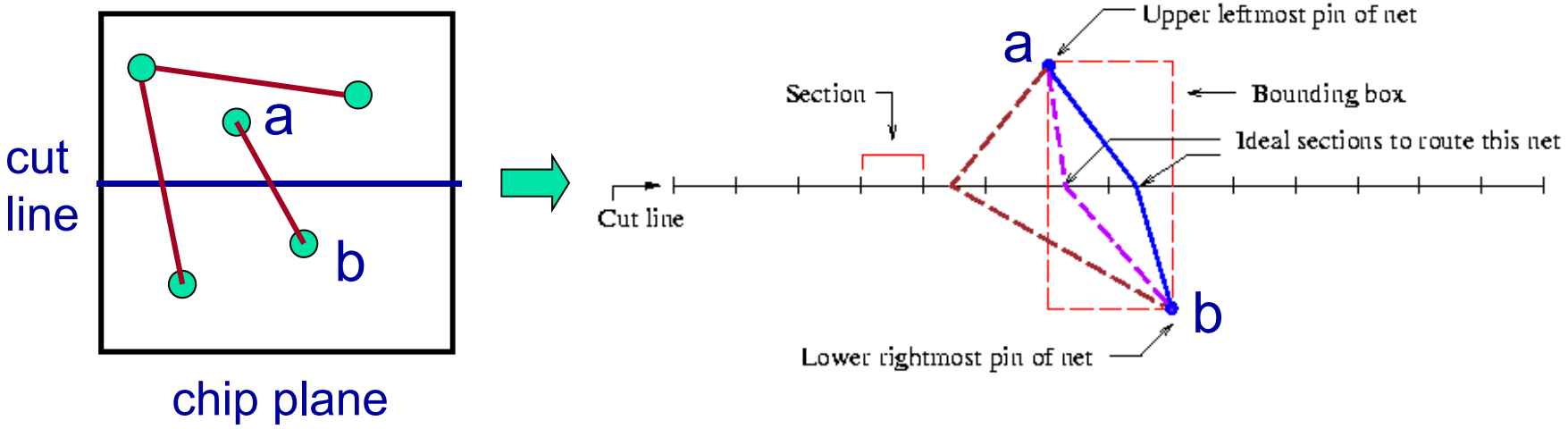
- Marek-Sadowska, “Router planner for custom chip design,” ICCAD-86.
- At each level of the hierarchy, an attempt is made to minimize the cost of nets crossing cut lines.
- At the lowest level of the hierarchy, the layout surface is divided into $R \times R$ grid regions with boundary capacity equal to C tracks.
- Let R_l be the # of grid regions of a given cut line l; a cut line can be divided into $M = R_l / C$ sections.
- 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 $\sum_{i=1}^{N} \sum_{j=1}^{M} w_{ij} x_{ij}.$





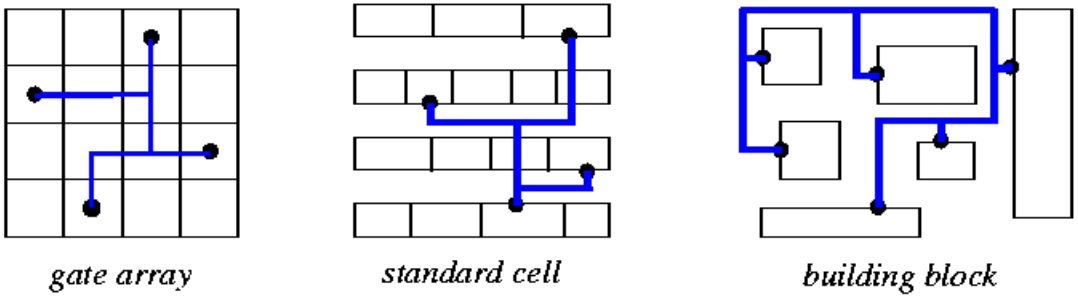
Hierarchical Global Routing (cont'd)

- 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: $\sum_{j=1}^{M} x_{ij} = 1, 1 \le i \le N.$
    - Reasonable assumption that simplifies the problem
  - 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 $\sum_{i=1}^{N} \sum_{j=1}^{M} w_{ij} x_{ij}$.

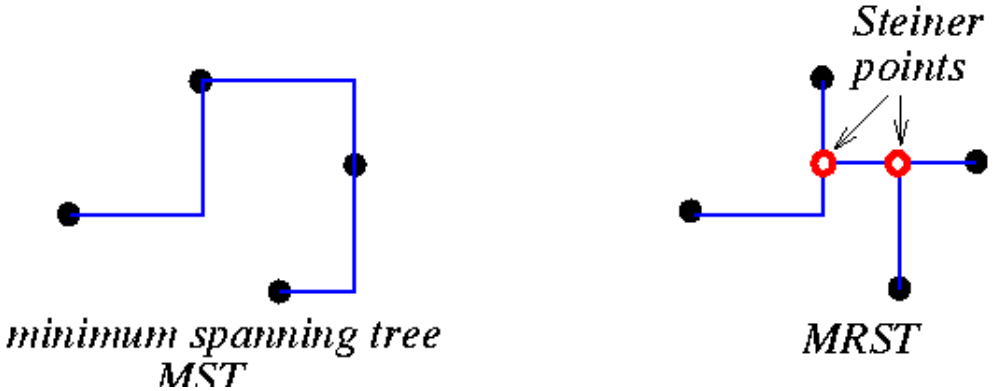


The Routing-Tree Problem

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

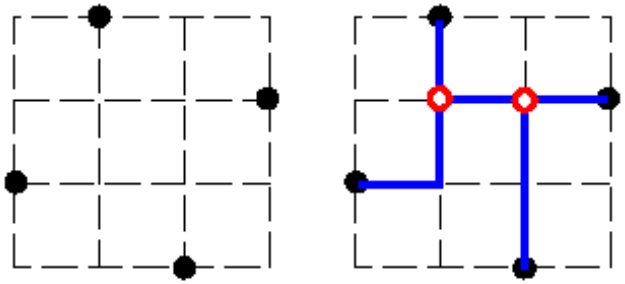


- **Minimum Rectilinear Steiner Tree (MRST) Problem:** Given n points in the plane, find a minimum-length tree of rectilinear edges which connects the points.
- $MRST(P) = MST(P \cup S)$, where P and S are the sets of original points and Steiner points, respectively.

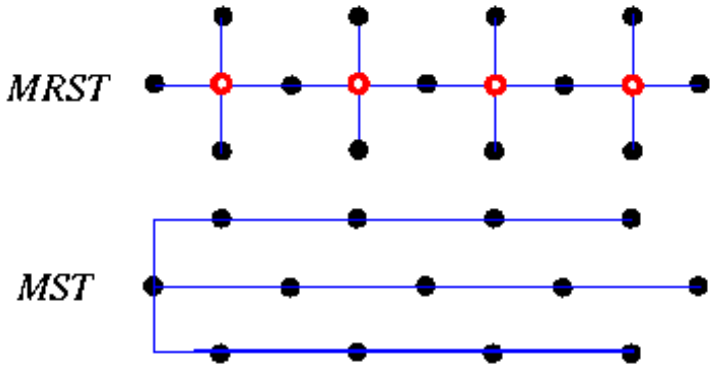


Theoretic Results for the MRST Problem

- **Hanan's Thm:** There exists an MRST with all Steiner points (set *S*) chosen from the intersection points of horizontal and vertical lines drawn points of *P*.
  - Hanan, "On Steiner's problem with rectilinear distance," *SIAM J. Applied Math.*, 1966.
- **Hwang's Theorem:** For any point set *P*, $\frac{Cost(MST(P))}{Cost(MRST(P))} \leq \frac{3}{2}$.
  - Hwang, "On Steiner minimal tree with rectilinear distance," *SIAM J. Applied Math.*, 1976.
- Better approximation algorithm with the performance bound 61/48
  - Foessmeier *et al*, "Fast approximation algorithm for the rectilinear Steiner problem," Wilhelm Schickard-Institut für Informatik, TR WSI-93-14, 93.



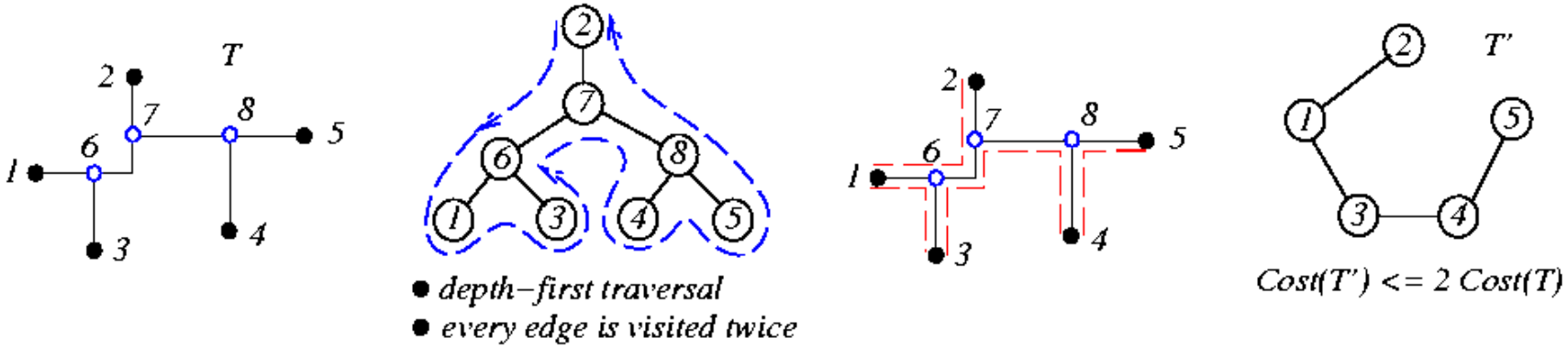
Hanan grid



Cost(MST)/Cost(MRST) –> 3/2

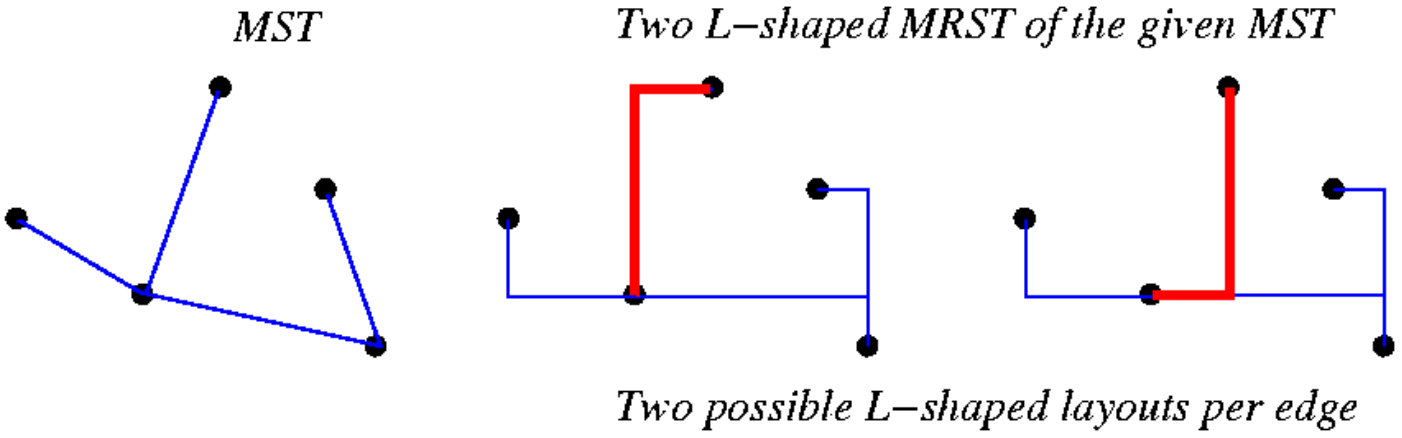
A Simple Performance Bound

- Easy to show that $\frac{Cost(MST(P))}{Cost(MRST(P))} \leq 2$
- Given any MRST T on point set P with Steiner point set S, construct a spanning tree T' on P as follows:
  1. Select any point in T as a root.
  2. Perform a depth-first traversal on the rooted tree T.
  3. Construct T' based on the traversal.



Coping with the MRST Problem

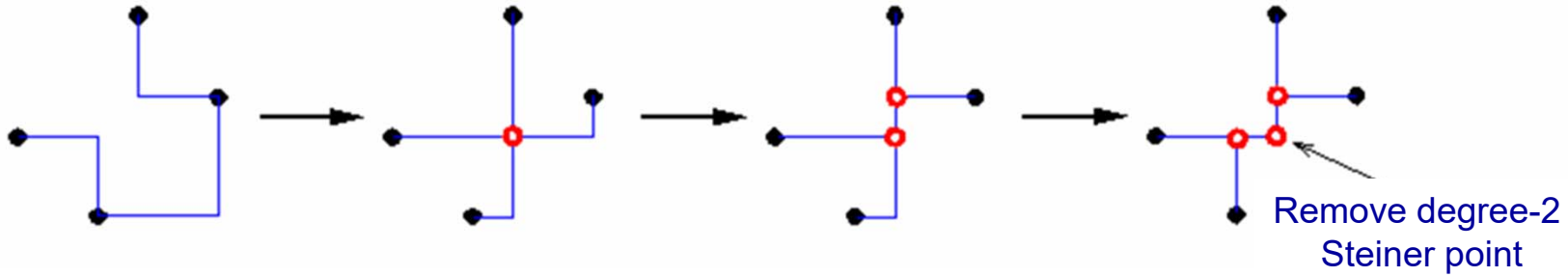
- Ho, Vijayan, Wong, “New algorithms for the rectilinear Steiner problem,” TCAD-90.
  1. Construct an MRST from an MST.
  2. Each edge is straight or L-shaped.
  3. Maximize overlaps by dynamic programming.
- About 8% smaller than *Cost*(*MST*).



Iterated 1-Steiner Heuristic for MRST

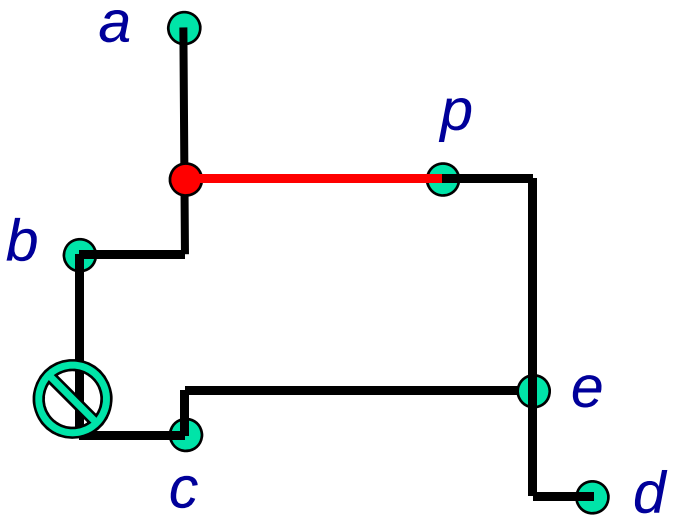
- Extended by Kahng & Robins, "A new class of Steiner tree heuristics with good performance: the iterated 1-Steiner approach," *ICCAD*, 1990.

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

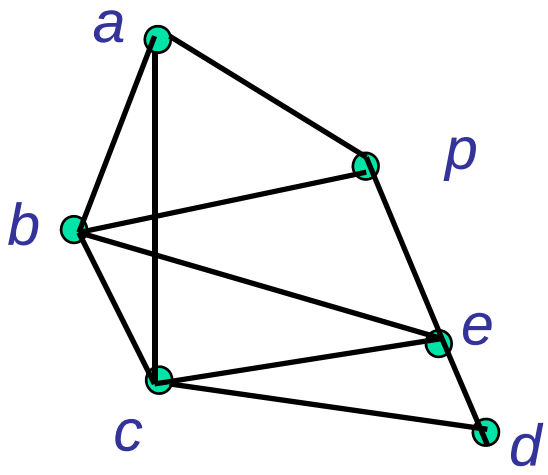


Spanning-Graph-Based Steiner Minimal Trees

- Zhou, "Efficient Steiner tree construction based on spanning graphs," ISPD-03 (TCAD-04)
- Select Borah *et al.*'s edge-substitution approach as the basis (TCAD-94)
  - Connect a point to an edge and delete the longest edge on the created cycle
- Enhance it with Zhou *et al.*'s spanning graph (ISPD-03)



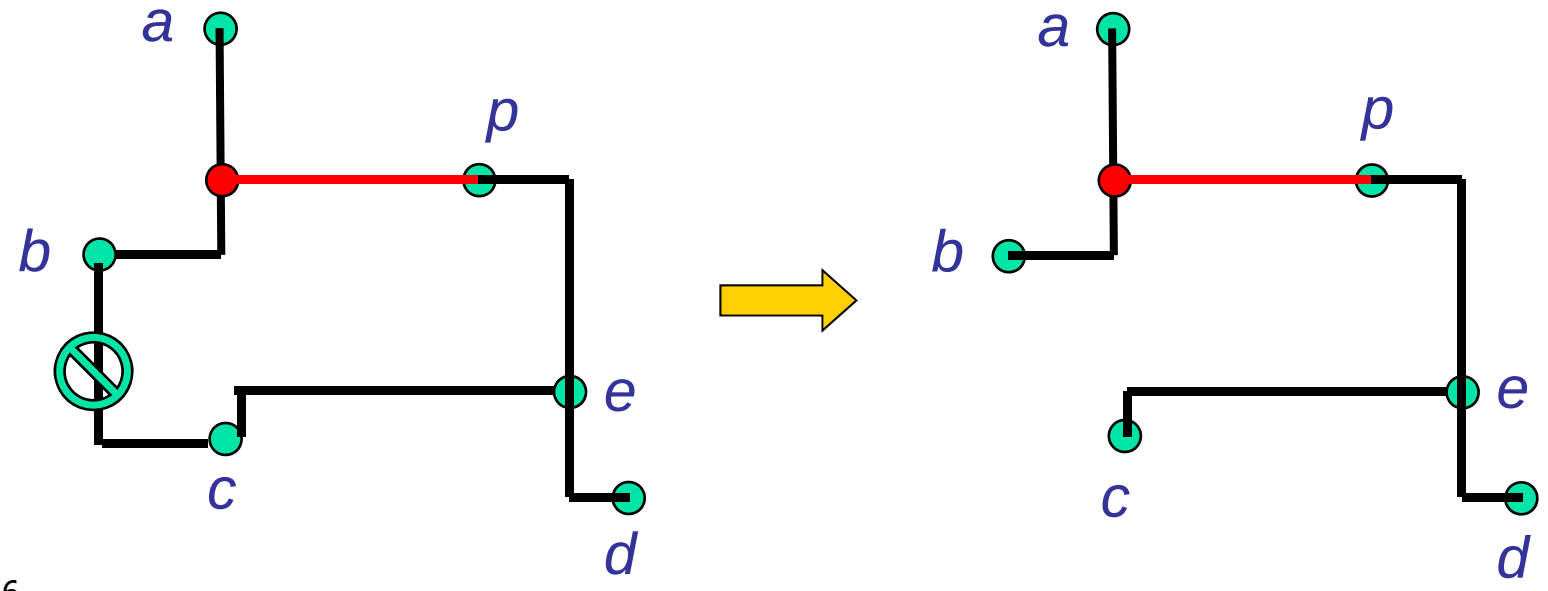
Edge substitution



Spanning graph

Edge-Substitution Approach

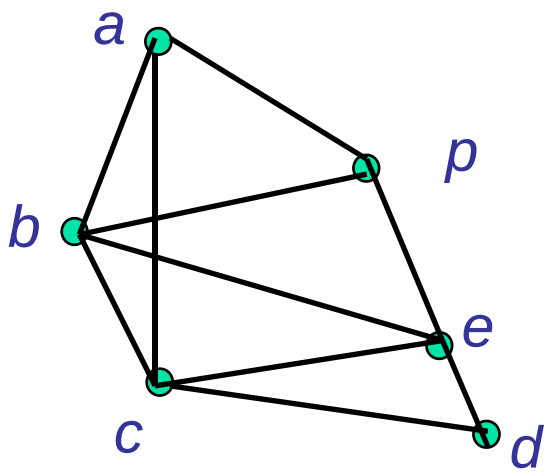
- Borah, Owens, and Irwin's algorithm for the rectilinear SMT (TCAD-94)
  - Start with a minimal spanning tree
  - Iteratively consider connecting a point to a nearby edge (e.g., point *p* to edge (*a, b*))
  - Delete the longest edge on the cycle thus formed (e.g., edge (*b, c*))



Implementing the Edge-Substitution Approach

- How to construct the initial spanning tree?
- How to find the edge-point pair candidates for connection?
- How to find the longest edge on the formed cycle?

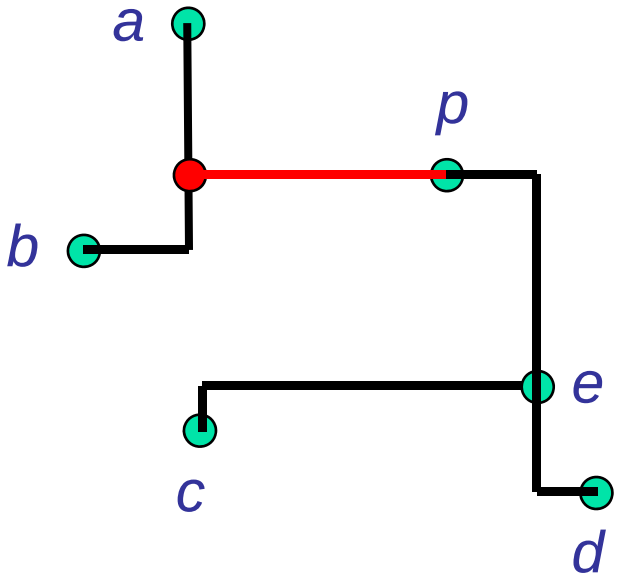
Spanning graphs may answer the questions!!

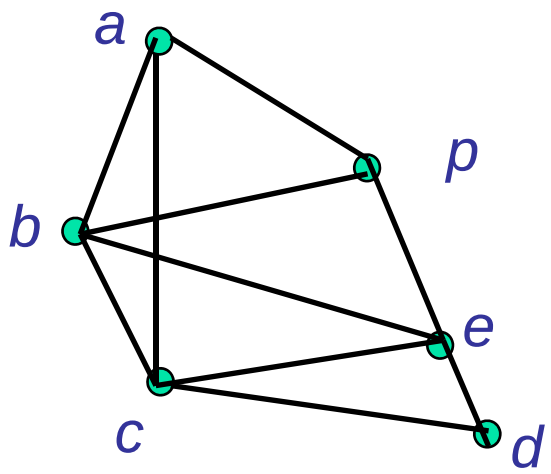


Spanning graph

Spanning Graph

- An MST on the graph is an MST of the given points
  - geometrical MST = spanning graph + graph MST
- Spanning graphs represent geometrical proximity information
  - Also provide candidates for point-edge connection
  - E.g., no edge (*b*, *d*) since *d* is blocked by edge (*c*, *e*)
- It is a sparse graph: $O(n)$ edges

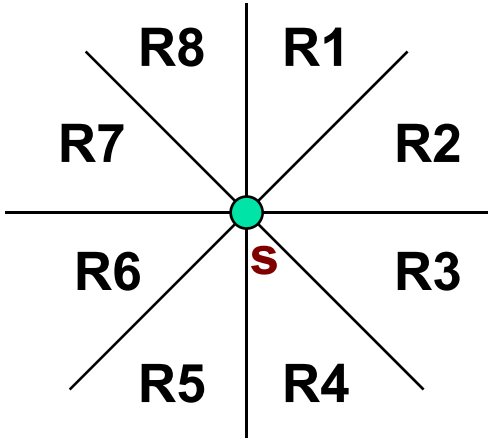


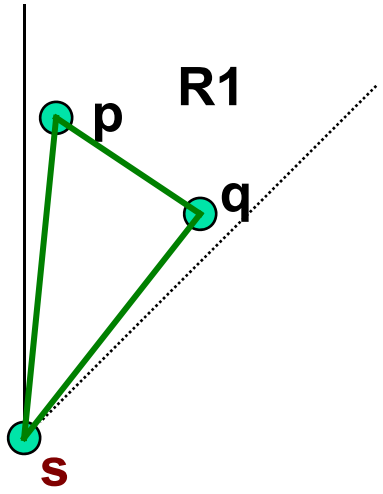


Spanning graph

Octal Partition for Finding a Spanning Graph

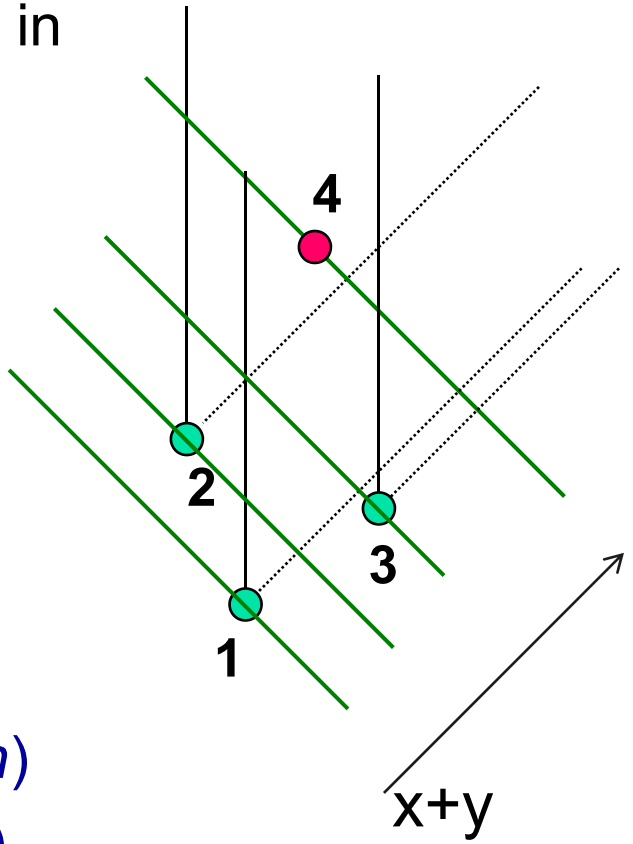
- Find a spanning graph in $O(n \lg n)$ time
  - Yao, SIAM J. Computing, 1982
- For each point s, the plane is divided into 8 octal regions
- $||pq|| < \max(||ps||, ||qs||)$
- Only the closest point in each region needs to be connected to s



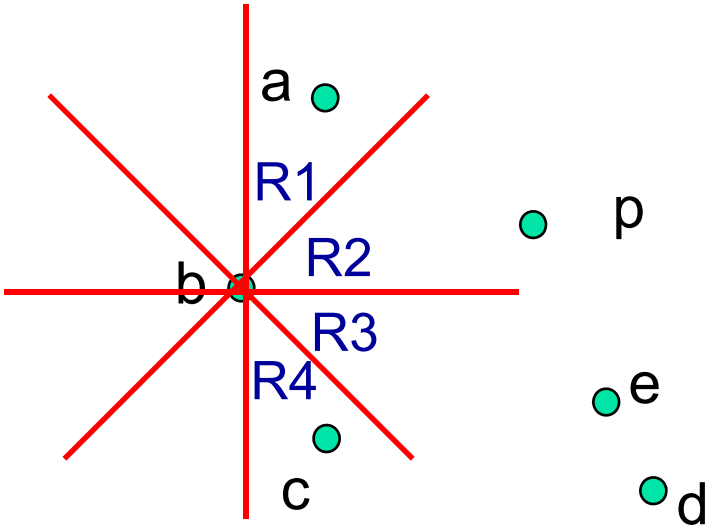


Finding Closest Points in R1

- Sweep points by increasing $x+y$
- Keep points waiting for closest point in A (active set)
- Checking current point with A
  - make connections
  - delete connected points
  - add current point in A
- Active set can be linearly ordered according to x
- Use a binary search tree
  - Finding an insertion place: O(lg n)
  - Deleting/inserting a point: O(lg n)
- Each point is inserted and deleted at most once: $O(n \lg n)$

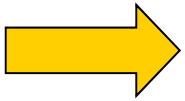


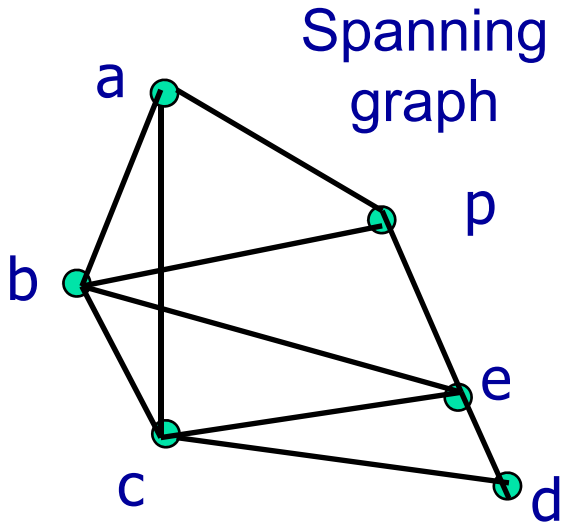
Finding the Neighbors



For each point, use the octal partition to find its neighbors.

	a	b	c	d	e	p
R1		a				
R2		p	e			
R3	p	e	d			
R4	c	c			d	e





Rectilinear Steiner Minimal Tree Algorithm

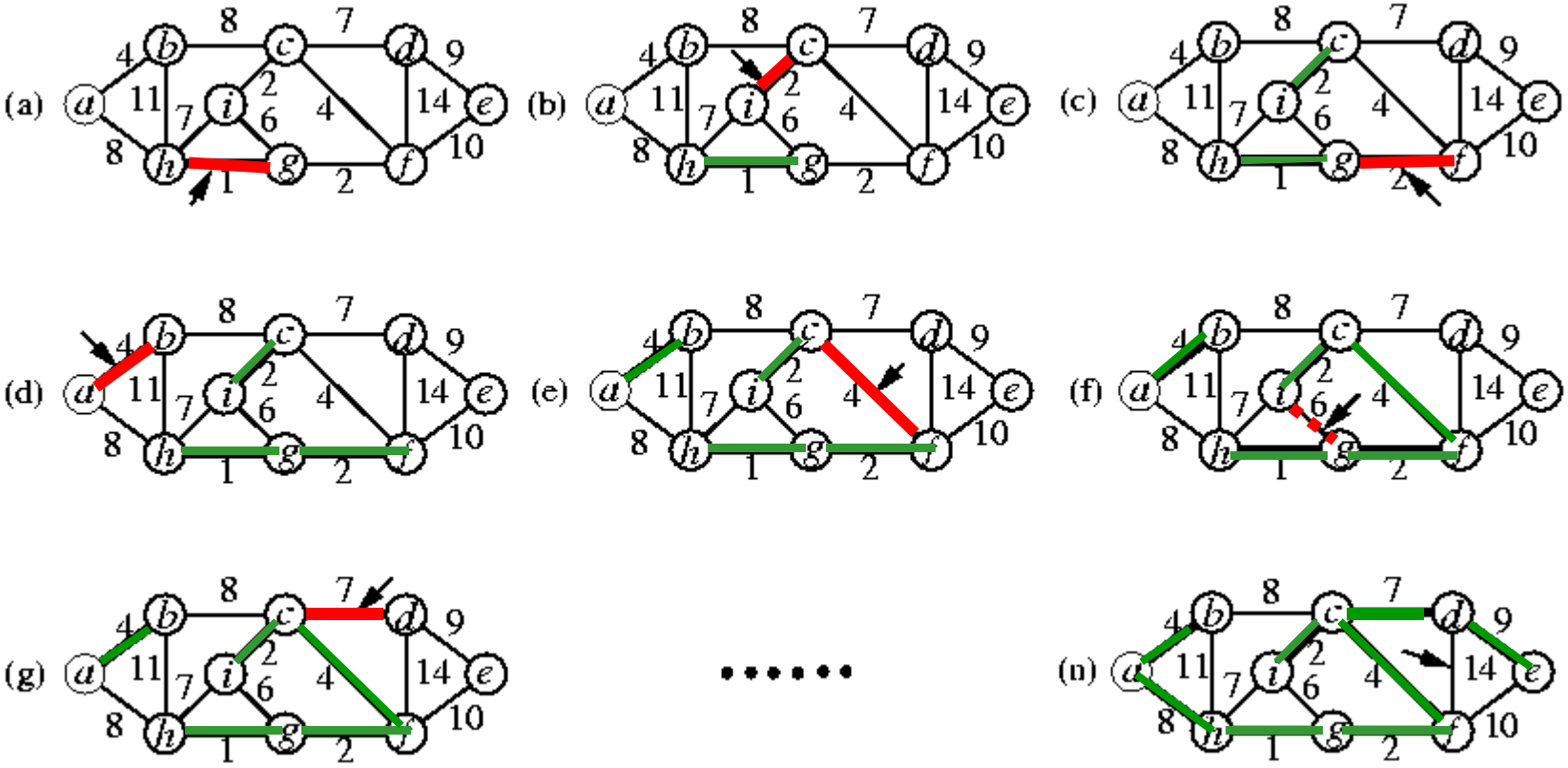
- Construct a rectilinear spanning graph
- Sort the edges in the graph
- Use Kruskal to build MST, at the same time
  - Build the merging binary tree and
  - Find possible point-edge candidates (by the spanning graph)
- Use least common ancestors on the merging binary tree to find the longest edges in cycles
- Iteratively update point-edge connections

Kruskal's MST Algorithm

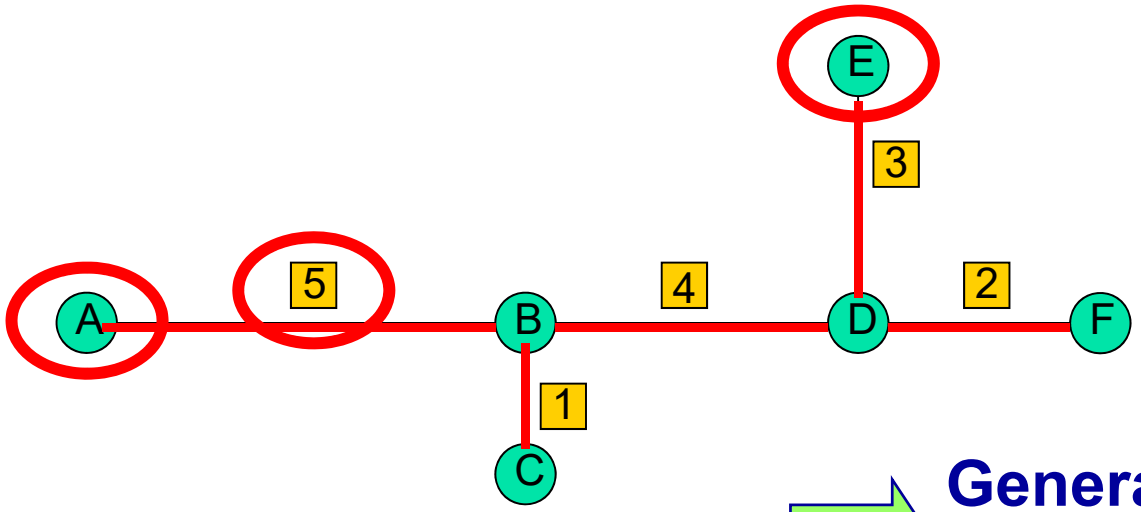
```
MST-Kruskal(G,w)
1. A ← ∅;
2. for  each vertex v ∈ V[G]
3.     Make-Set(v);
4. Sort the edges of E[G] by nondecreasing weight w;
5. for  each edge (u,v) ∈ E[G], in order by nondecreasing weight
6.     if Find-Set(u) ≠ Find-Set(v)
7.         A ← A ∪ {(u, v)};
8.         Union(u,v);
9. return A
```

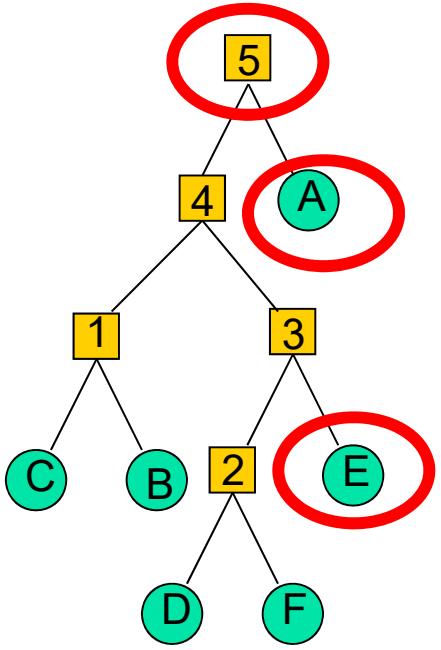
- Add a safe edge at a time to the growing **forest** by finding an edge of least weight (from those connecting two trees in the forest).
- Time complexity: $O(E \lg E + V)$ (= $O(E \lg V + V)$, why?)
  - Lines 2--3: $O(V)$.
  - Line 4:$O(E \lg E)$.
  - Lines 5--8: $O(E)$ operations on the disjoint-set forest, so $O(E\ \alpha(E, V))$.

Example: Kruskal's MST Algorithm



Finding the Longest Edge on a Cycle



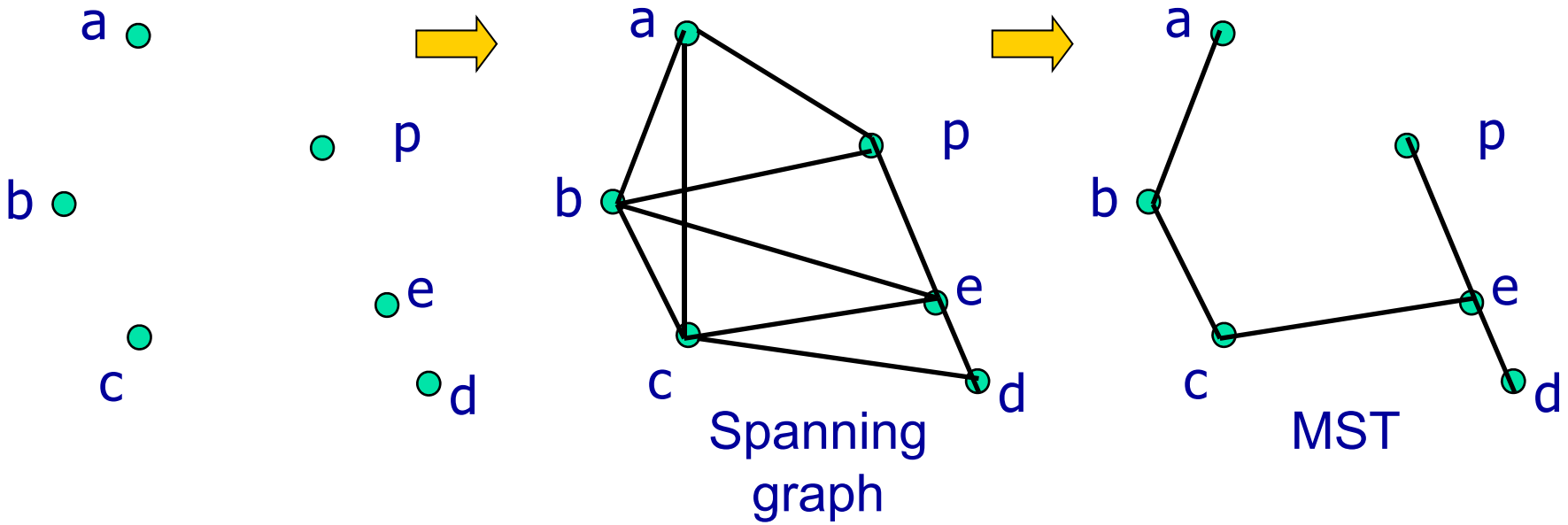


Generate the merging binary tree corresponding to the minimal spanning tree

The longest edge for this pair is the least common ancestor

Example Spanning Graph Construction

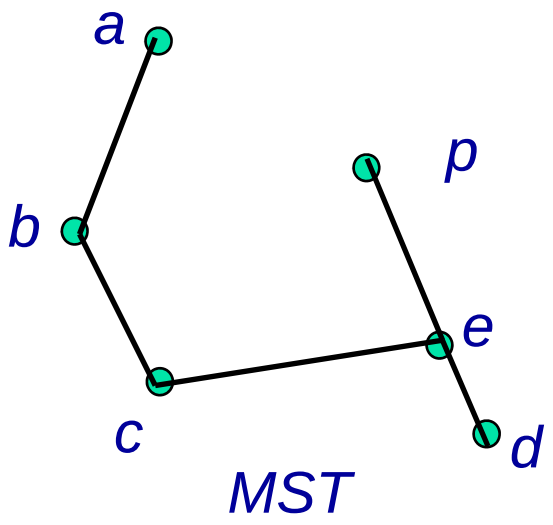
1. Construct a rectilinear spanning graph
2. Use Kruskal's algorithm to build MST
3. Update point-edge connections iteratively



Point-Edge Connection Updating

1. Construct a rectilinear spanning graph
2. Use Kruskal's algorithm to build MST
3. Update point-edge connections iteratively

point	edge	delete edge	gain
a	(b,c)	(a,b)	3
c	(a,b)	(b,c)	3
p	(a,b)	(b,c)	3
p	(a,b)	(c,e)	2
c	(d,e)	(c,e)	1
e	(b,c)	(c,e)	1
p	(a,b)	(e,p)	1
p	(b,c)	(c,e)	1
p	(c,e)	(e,p)	1

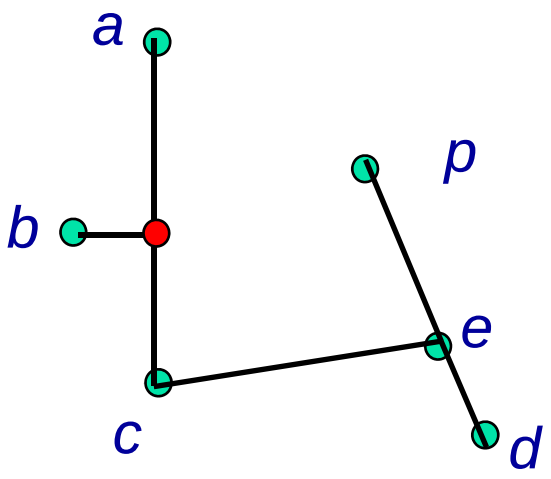


Building the table according to the spanning graph

Point-Edge Connection Updating (cont'd)

1. Construct a rectilinear spanning graph
2. Use Kruskal's algorithm to build MST
3. Update point-edge connections iteratively
   - Connect the point-edge pair with the max gain (e.g., point *a* to edge *(b, c)*)
   - Delete related point-edge candidates (point *c* to edge (*a, b*), etc.)

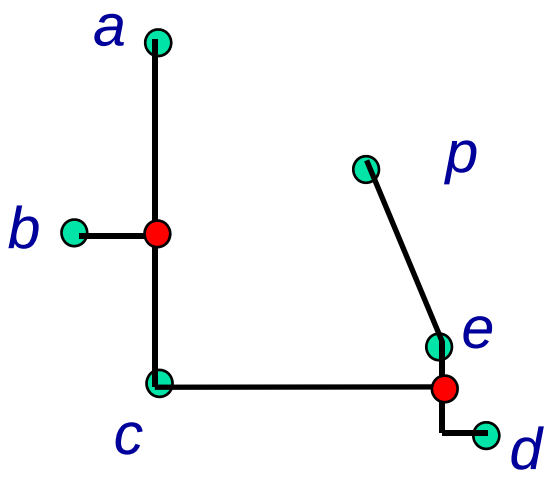
point	edge	delete edge	gain
a	(b,c)	(a,b)	3
c	(a,b)	(b,c)	3
p	(a,b)	(b,c)	3
p	(a,b)	(c,e)	2
c	(d,e)	(c,e)	1
e	(b,c)	(c,e)	1
p	(a,b)	(e,p)	1
p	(b,c)	(c,e)	1
p	(c,e)	(e,p)	1



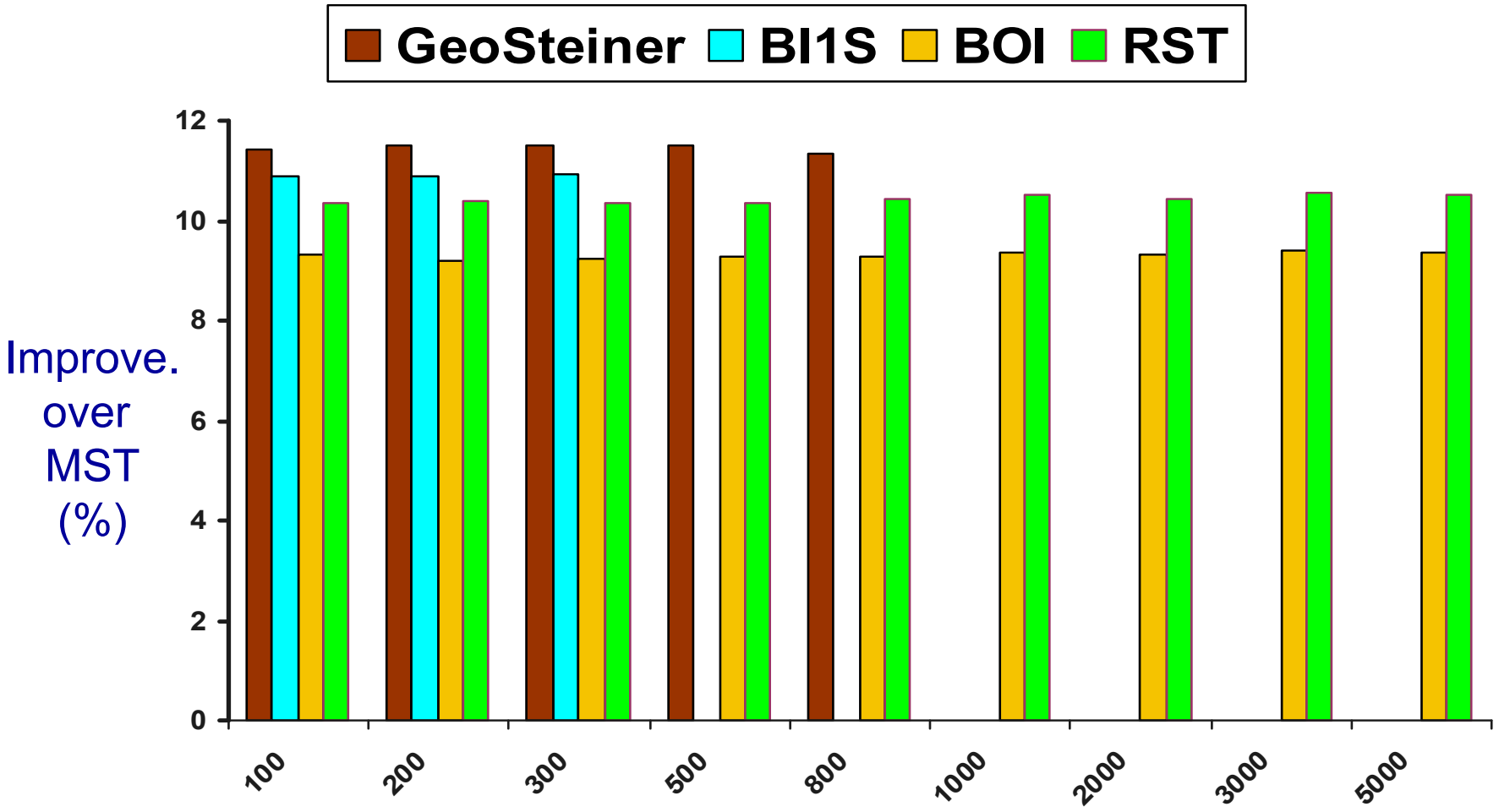
Point-Edge Connection Updating (cont'd)

1. Construct a rectilinear spanning graph
2. Use Kruskal's algorithm to build MST
3. Update point-edge connections iteratively
   - Find the remaining point-edge pair with the max gain (point *c* to edge *(d, e)*)

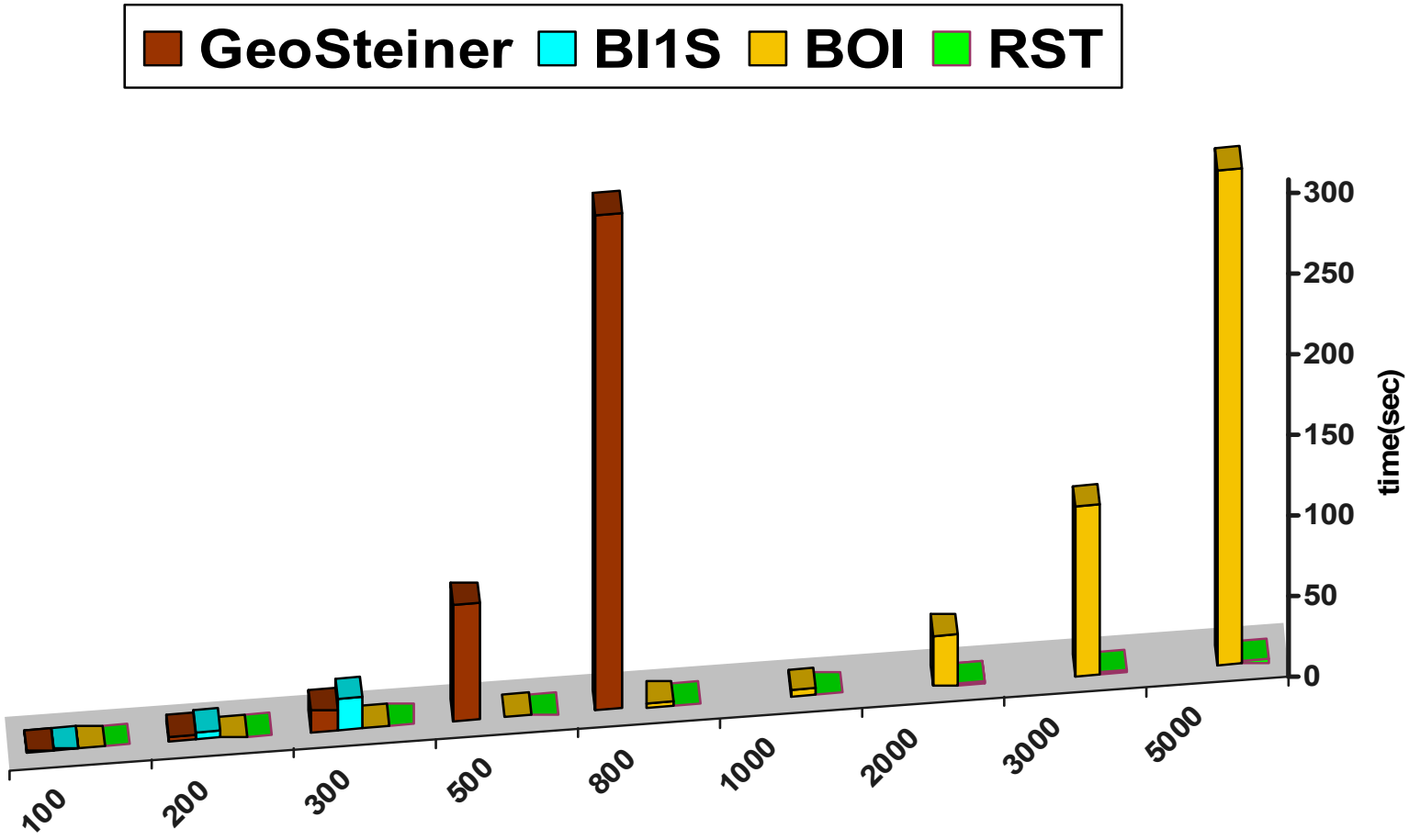
point	edge	delete edge	gain
a	(b,c)	(a,b)	3
c	(a,b)	(b,c)	3
p	(a,b)	(b,c)	3
P	(a,b)	(c,e)	2
c	(d,e)	(c,e)	1
e	(b,c)	(c,e)	1
p	(a,b)	(e,p)	1
p	(b,c)	(c,e)	1
p	(c,e)	(e,p)	1



Quality Comparison



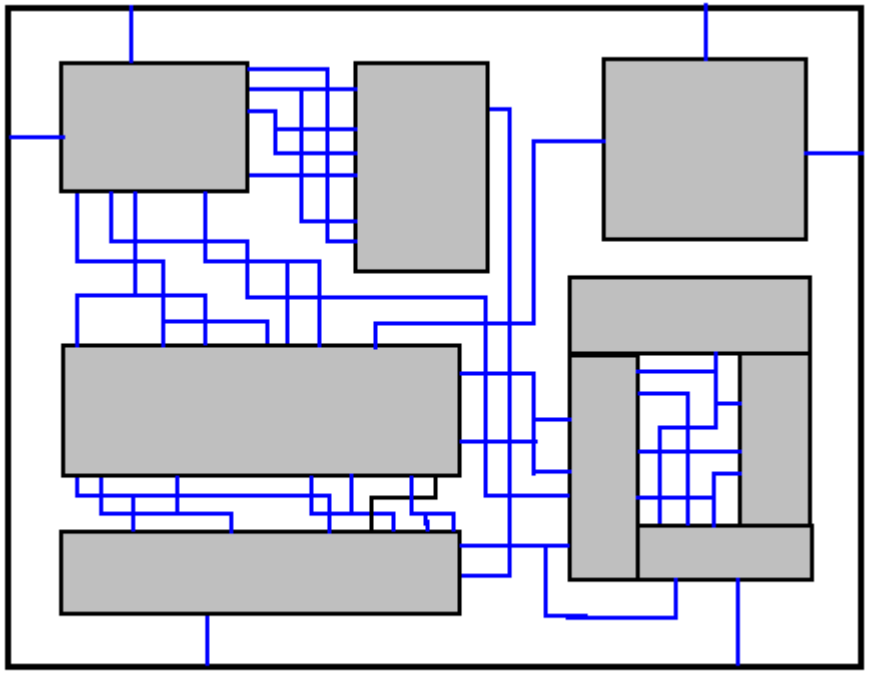
Running-Time Comparison



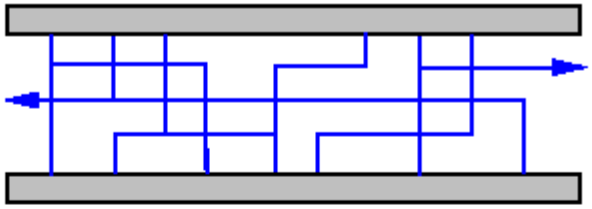
Summary on the RSMT Algorithms

- Optimal algorithms:
  - Hwang, Richards, Winter [ADM-92]
  - Warme, Winter, Zachariasen [AST-00] *GeoSteiner package*
- Near-optimal algorithms:
  - Griffith et al. [TCAD-94] *Batched 1-Steiner heuristic (BI1S)*
  - Mandoiu, Vazirani, Ganley [ICCAD-99]
- Low-complexity algorithms:
  - Borah, Owens, Irwin [TCAD-94] *Edge-based heuristic, O(n log n)*
  - Zhou [ISPD-03] *Spanning graph based, O(n log n)*
- Algorithms targeting low-degree nets (VLSI applications):
  - Soukup [Proc. IEEE-81] *Single Trunk Steiner Tree (STST)*
  - Chen et al. [SLIP-02] *Refined Single Trunk Tree (RST-T)*
  - **Chu [ICCAD-04, TCAD-08]: very efficient & effective *O(n log n)-time algorithm*** (see Appendix A)
- Obstacle-avoiding RSMT construction
  - Lin, et al. [ISPD-07, TCAD-08] (see Appendix B), Ajwani, et al., [TCAD-11]; many more
- Multilayer (3D) obstacle-avoiding RSMT construction
  - Lin, et al. [ICCAD-07, TCAD-08], Liu, et al. [TCAD-14]; many more

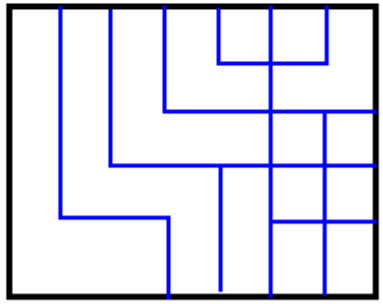
Detailed Routing: Channel/Switchbox Routing



Detailed routing



channel routing



switchbox routing

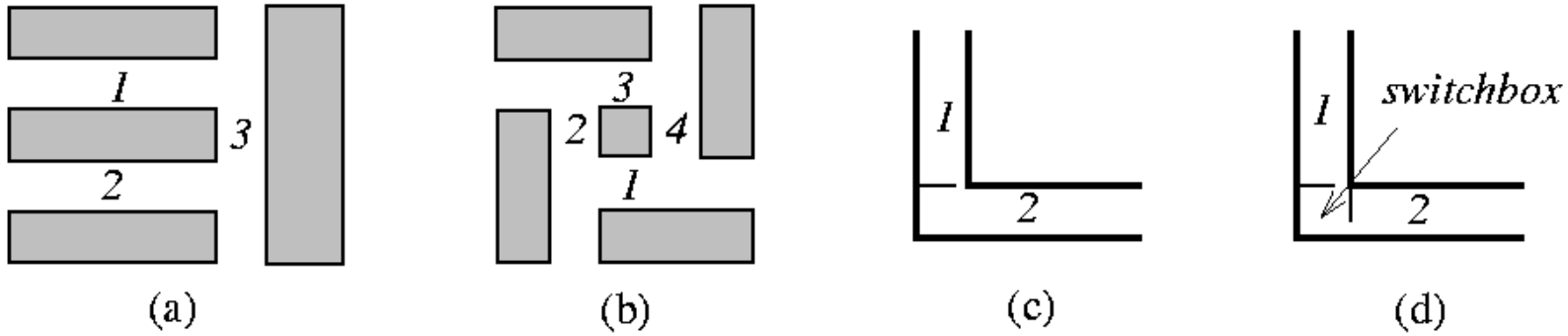
Order of Routing Regions and L-Channels

a) No conflicts if 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.

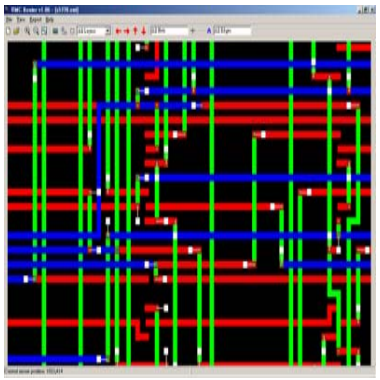


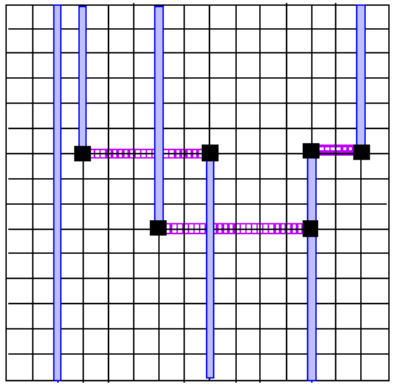
Routing Considerations

- 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)

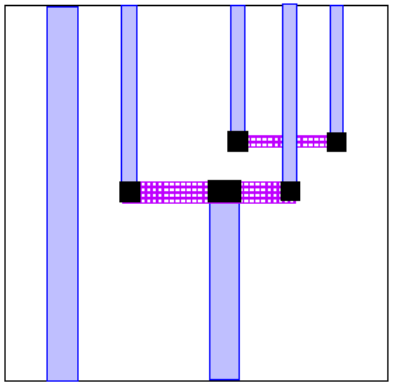
Routing Models

- **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.

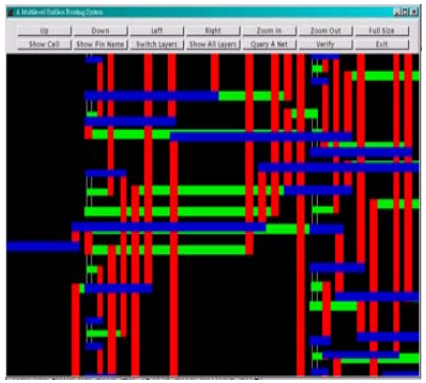




grid–based

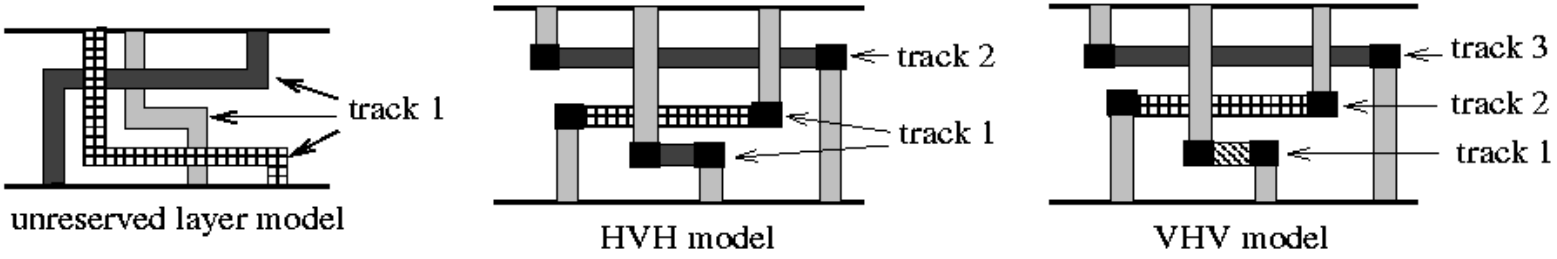


gridless



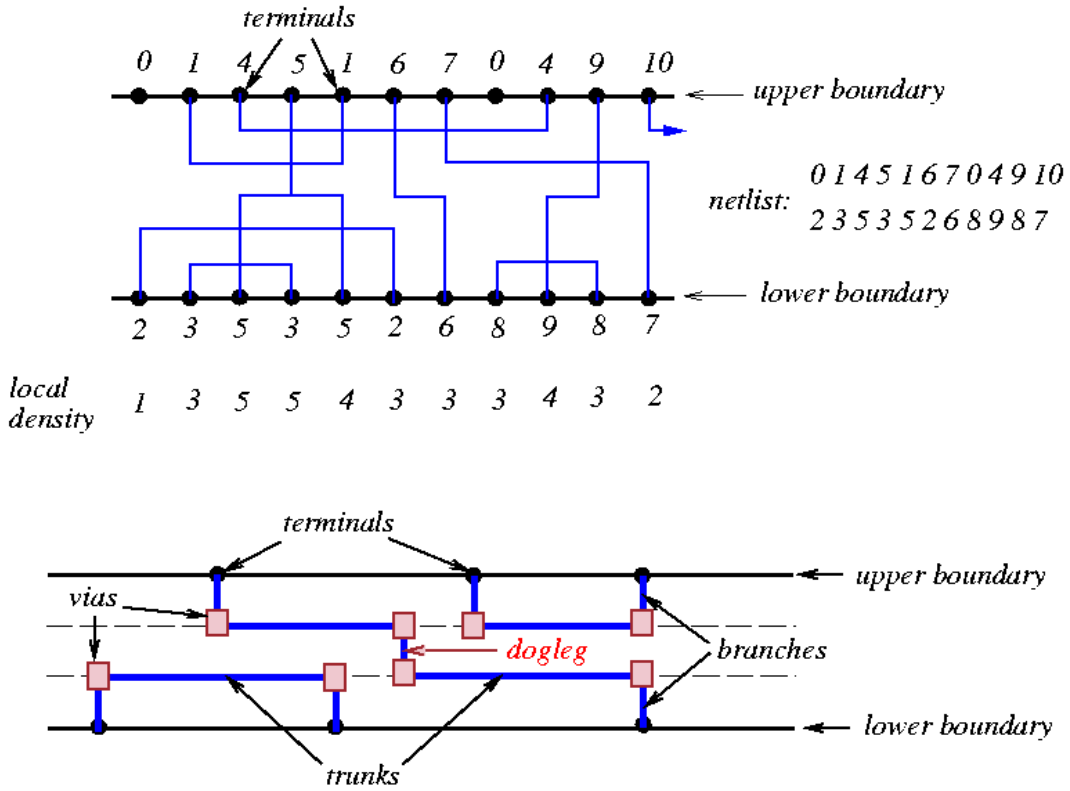
Models for Multi-Layer Routing

- **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 layers: HV (horizontal-Vertical), VH
  - Three layers: HVH, VHV
  - Modern design can afford 10+ layers



3 types of 3–layer models

Terminology for Channel Routing Problems



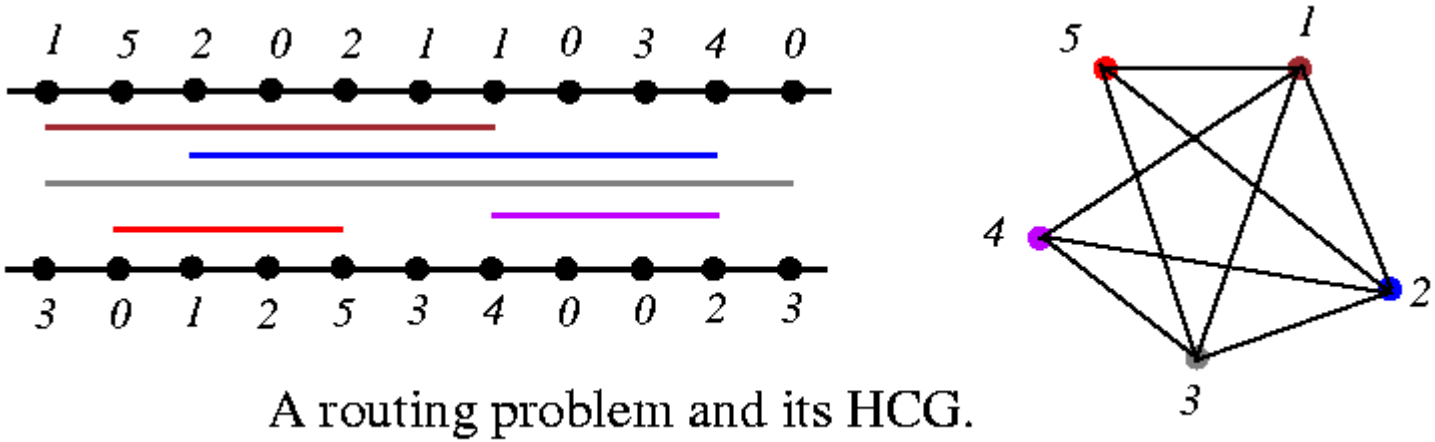
- Local density at column *i*, *d(i)*: total # of nets that crosses column *i*.
- Channel density: maximum local density
  - # of horizontal tracks required ≥ channel density.

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).

Horizontal Constraint Graph (HCG)

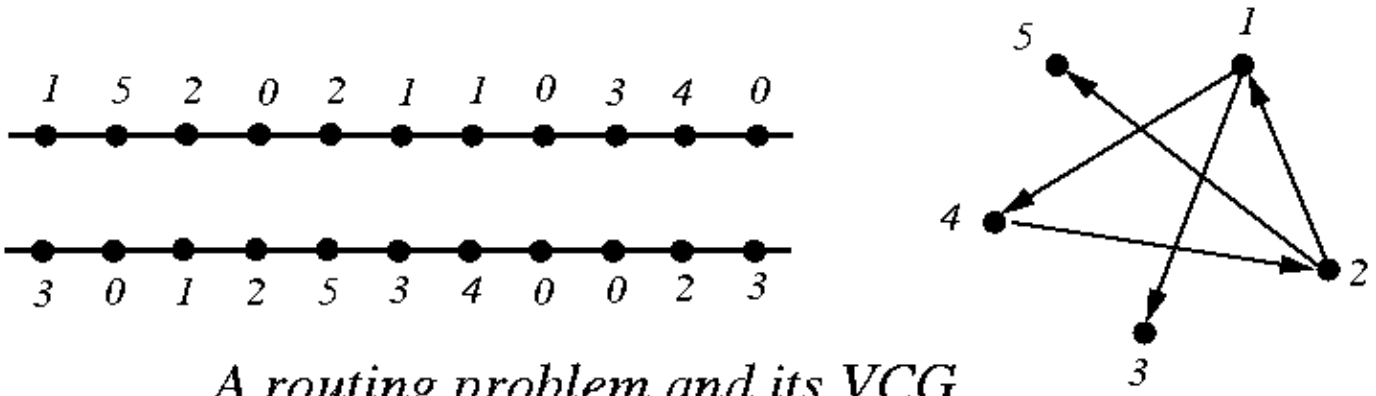
- HCG $G = (V, E)$ is undirected graph where
  - $V = \{ v_i \mid v_i$ represents a net $n_i\}$
  - $E = \{(v_i, v_j) \mid$ a horizontal constraint exists between n_i and $n_j\}$.
- For graph G: vertices $\Leftrightarrow$ nets;
  edge $(i, j) \Leftrightarrow$ net i overlaps net j.



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-Layer Channel Routing: Basic Left-Edge Algorithm

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

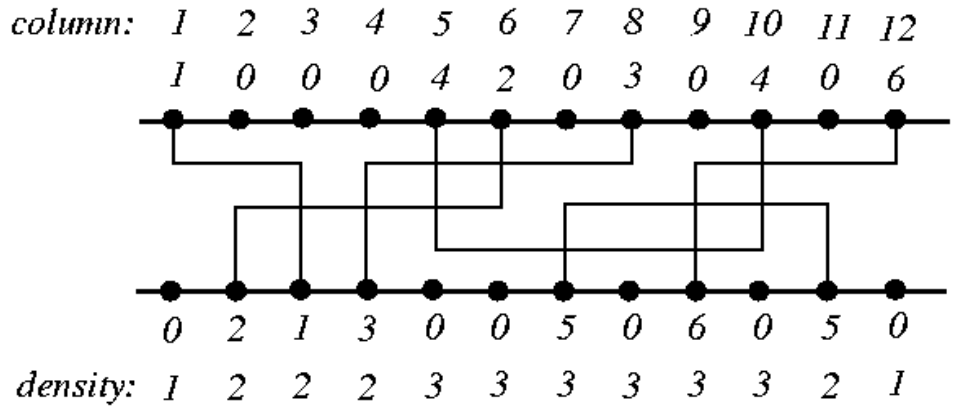
Basic Left-Edge Algorithm

Algorithm: Basic_Left-Edge(*U, track*[*j*])
U: set of unassigned intervals (nets) $I_1, \ldots, I_n$;
$I_j=[s_j, e_j]$: 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
```

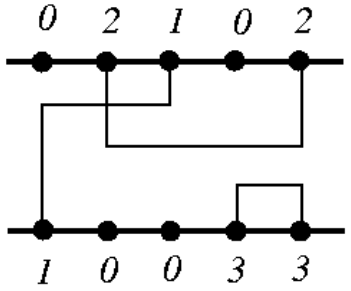
Basic Left-Edge Example

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

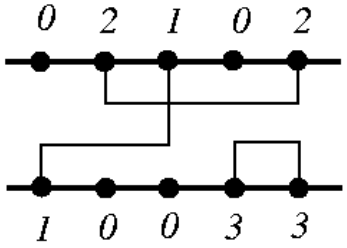


Basic Left-Edge Algorithm

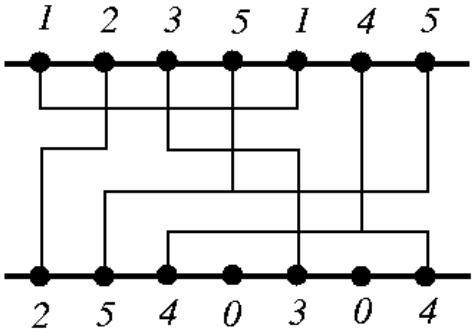
- If there is no vertical constraint, the basic left-edge algorithm is optimal.
- If there is any vertical constraint, the algorithm no longer guarantees optimal solution.

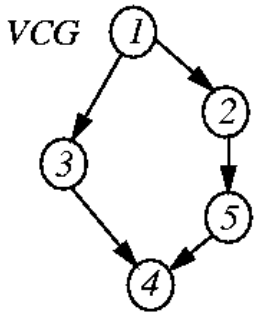


result from basic left–edge algorithm 3 tracks



optimal routing: 2 tracks





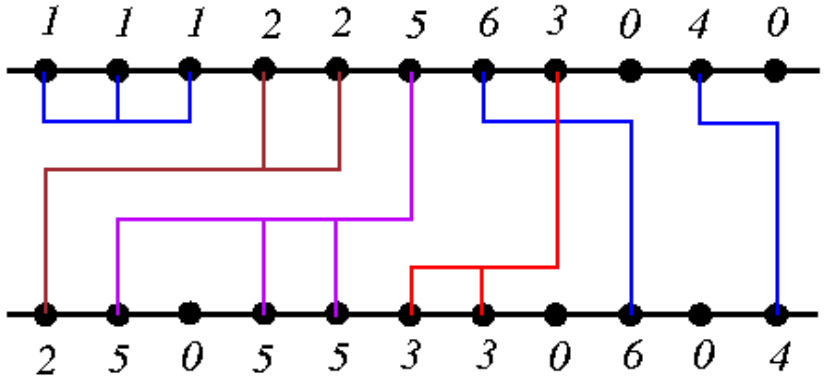
Constrained Left-Edge Algorithm

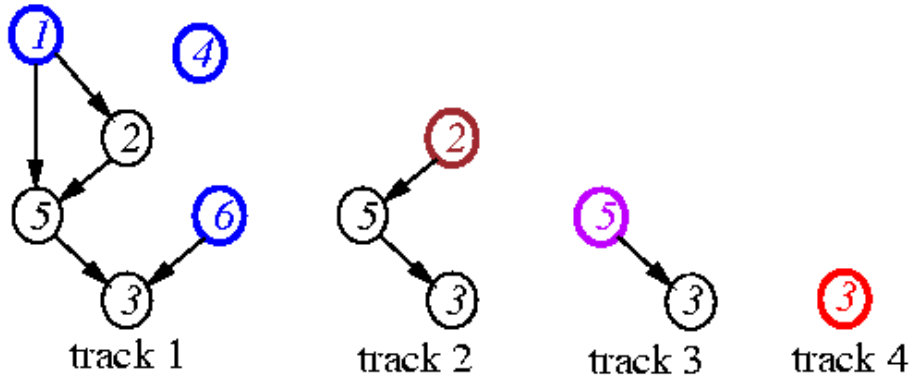
Algorithm: Constrained_Left-Edge(U, $track[j]$**)**
U: set of unassigned intervals (nets) I_1, …, I_n;
I_j=[s_j, e_j]: interval j with left-end x-coordinate s_j and right-end e_j;
$track[j]$: track to which net j is assigned.

```
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
```

Constrained Left-Edge Example

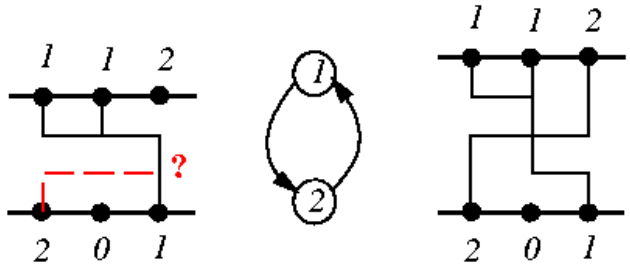
- I_1 = [1, 3], I_2 = [1, 5], I_3 = [6, 8], I_4 = [10, 11], I_5= [2, 6], I_6 = [7, 9].
- Track 1: Route I_1 (cannot route I_3); Route I_6; Route I_4.
- Track 2: Route I_2; cannot route I_3.
- Track 3: Route I_5.
- Track 4: Route I_3.



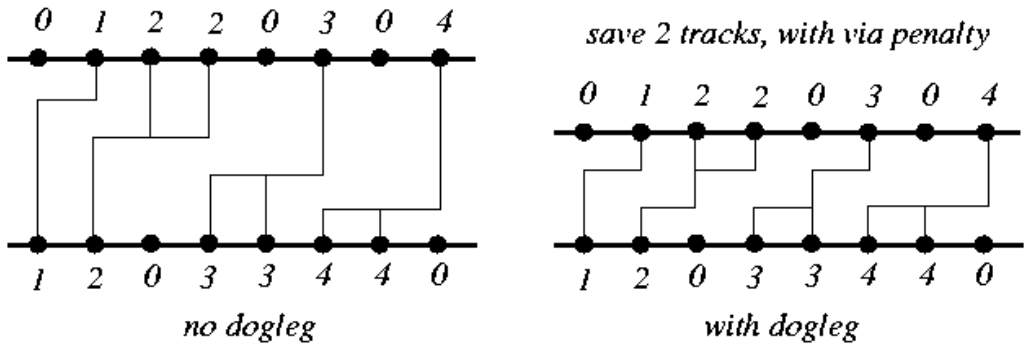


Dogleg Channel Router

- 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.

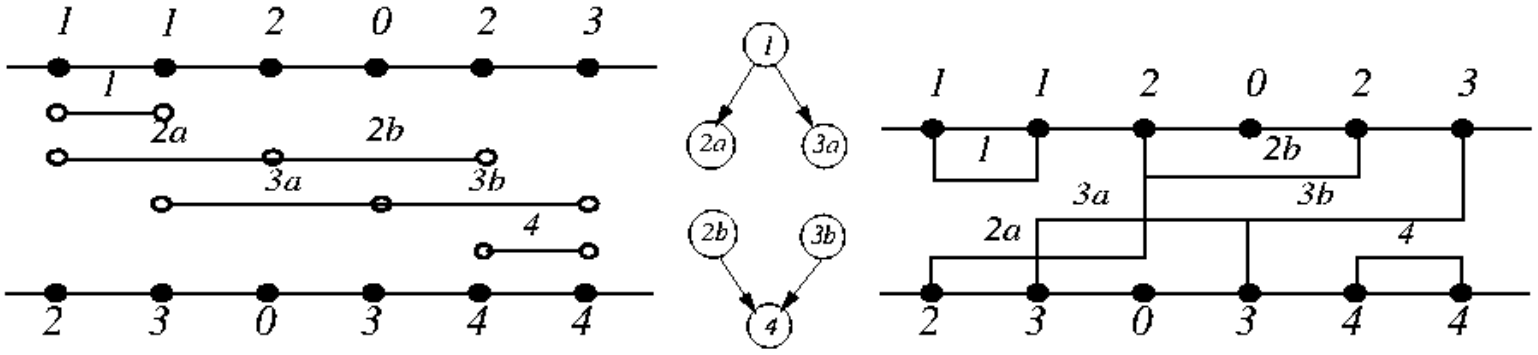


- **Drawback of Left-Edge: the entire net is on a single track.**
  - **Doglegs** are used to place parts of a net on different tracks to minimize channel height.
  - Might incur penalty for additional vias.



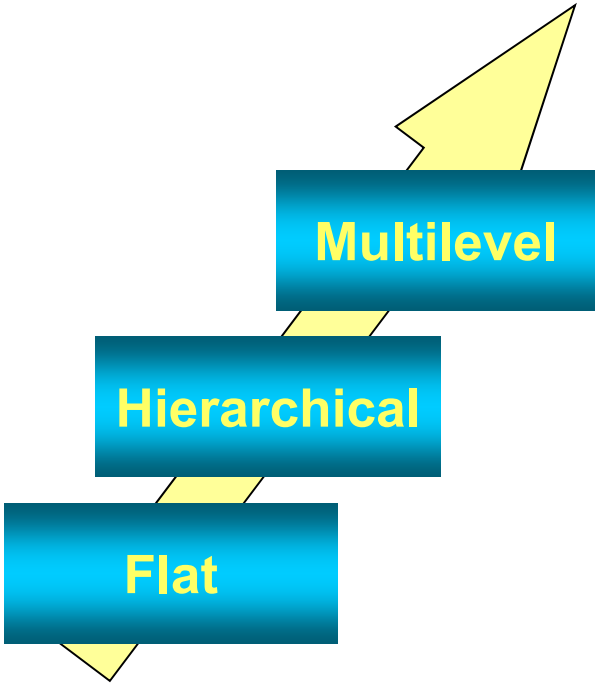
Dogleg Channel Router

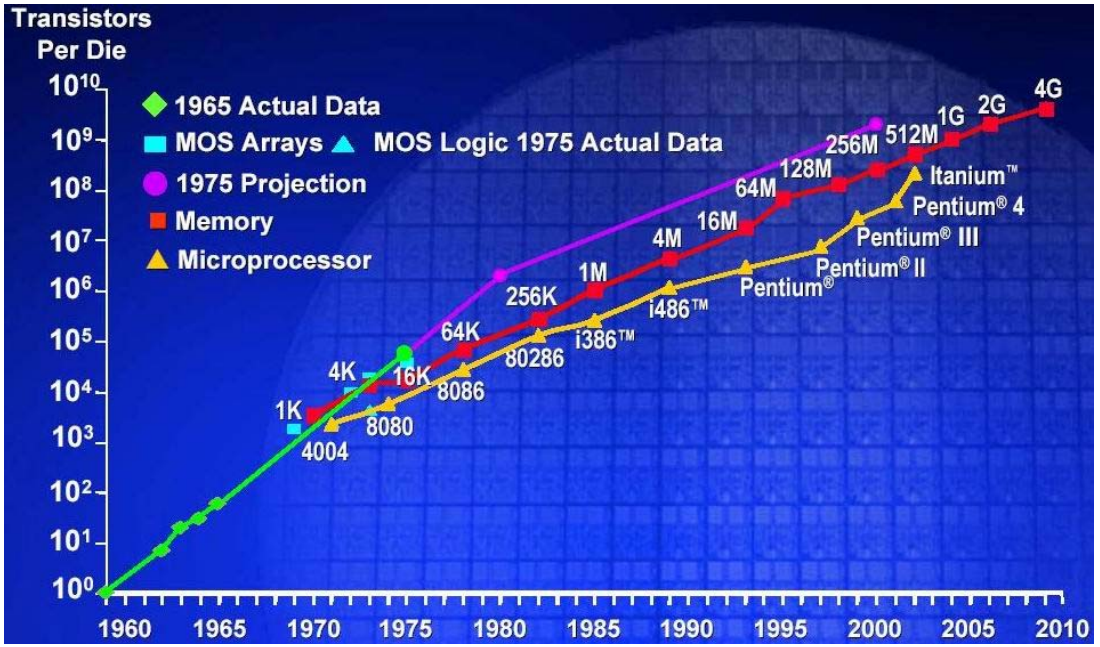
- Each multi-terminal net is broken into a set of 2-terminal nets.
- Two parameters are used to control routing:
  - Range: Determine the # of consecutive 2-terminal subnets of the same net that can be placed on the same track.
  - Routing sequence: Specifies the starting position and the direction of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

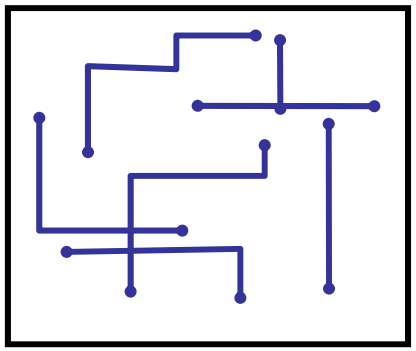




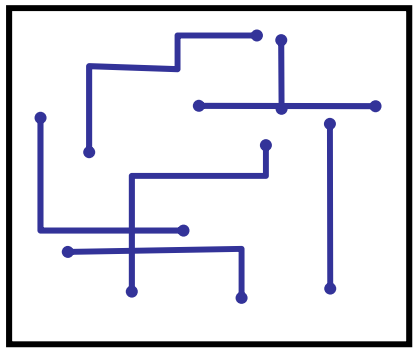
Source: Intel (ISSCC-03)

Flat Routing Framework

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



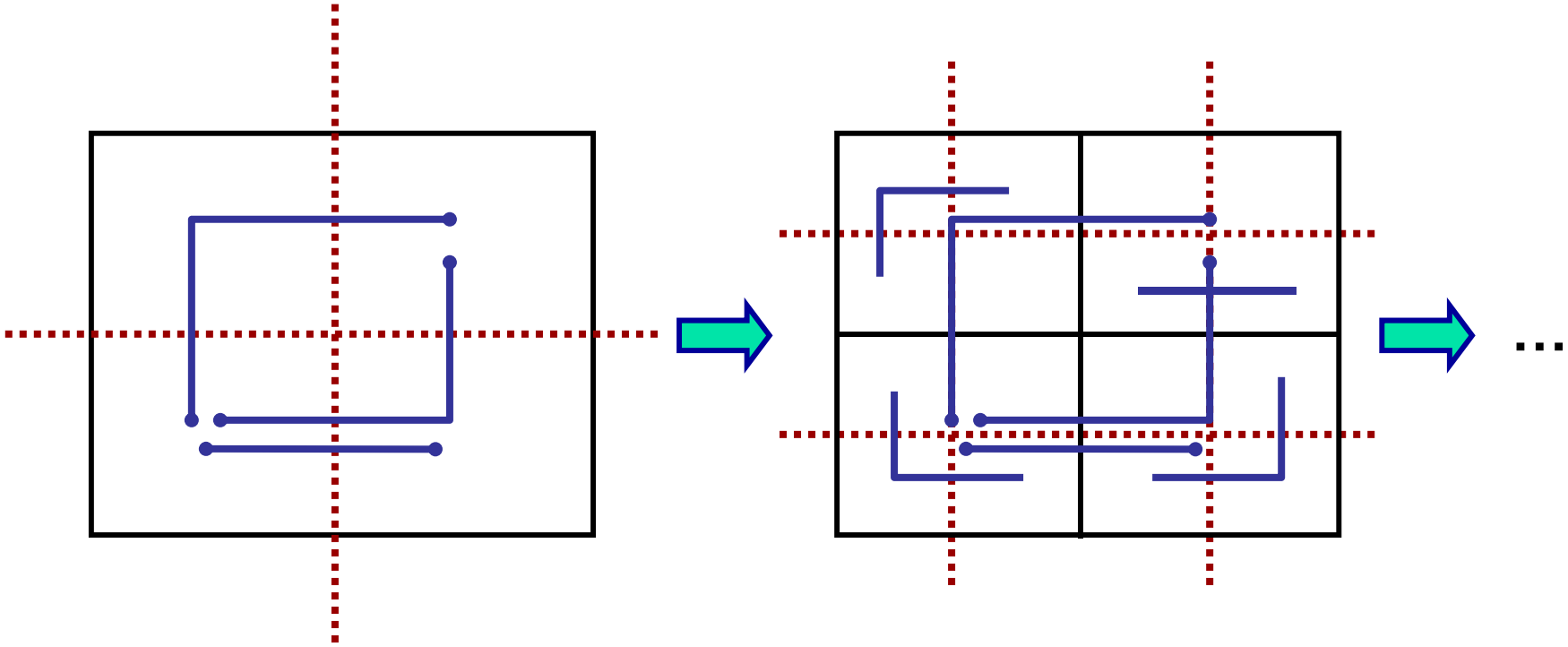
Sequential



Concurrent

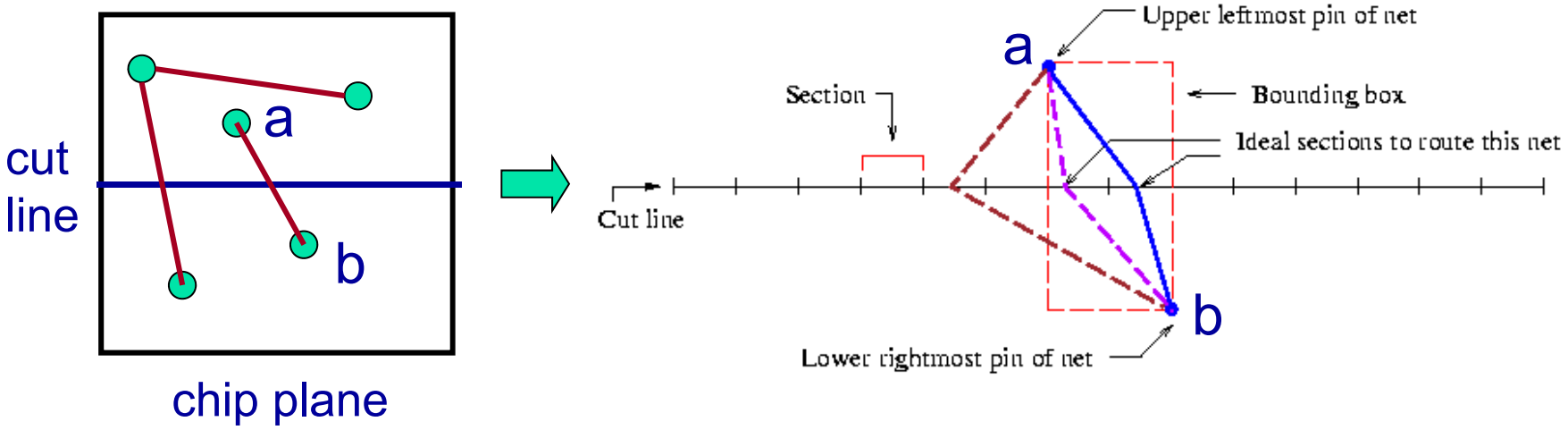
Hierarchical Routing Framework

- 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.



Hierarchical Routing Revisited

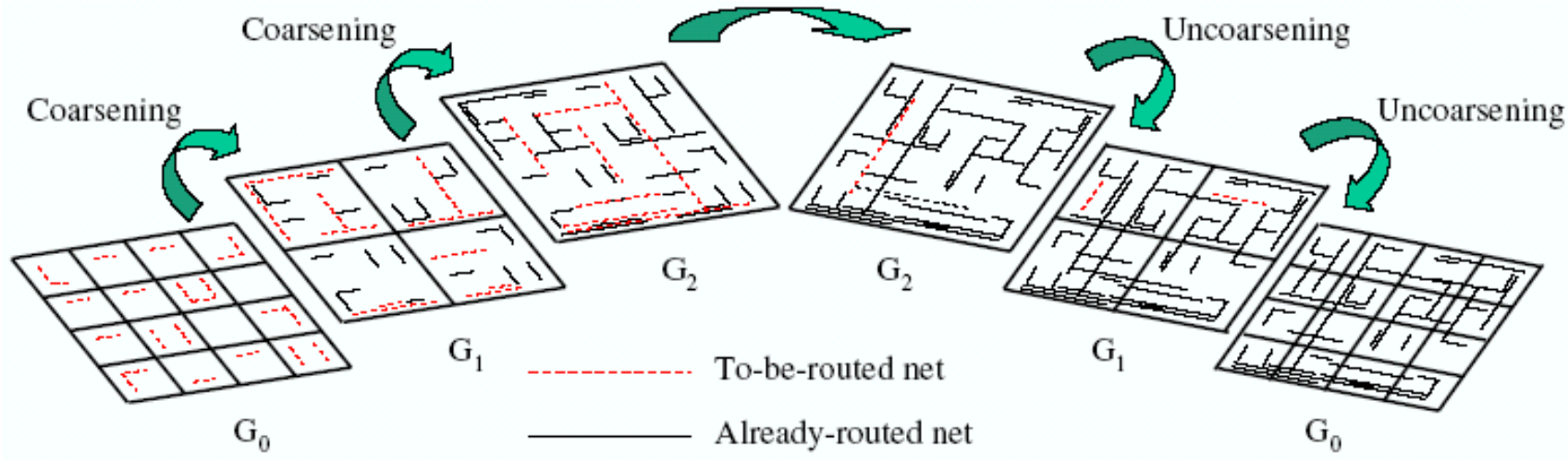
- Global routing can be formulated as a linear assignment problem:
  - $x_{i,j}$ = 1 if net i is assigned to section j; $x_{i,j}$ = 0 otherwise.
  - Each net crosses the cut line exactly once: $\sum_{j=1}^{M} x_{ij} = 1, 1 \le i \le N.$
  - Capacity constraint of each section: $\sum_{i=1}^{N} x_{ij}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



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.

A Multilevel Full-Chip Routing Framework

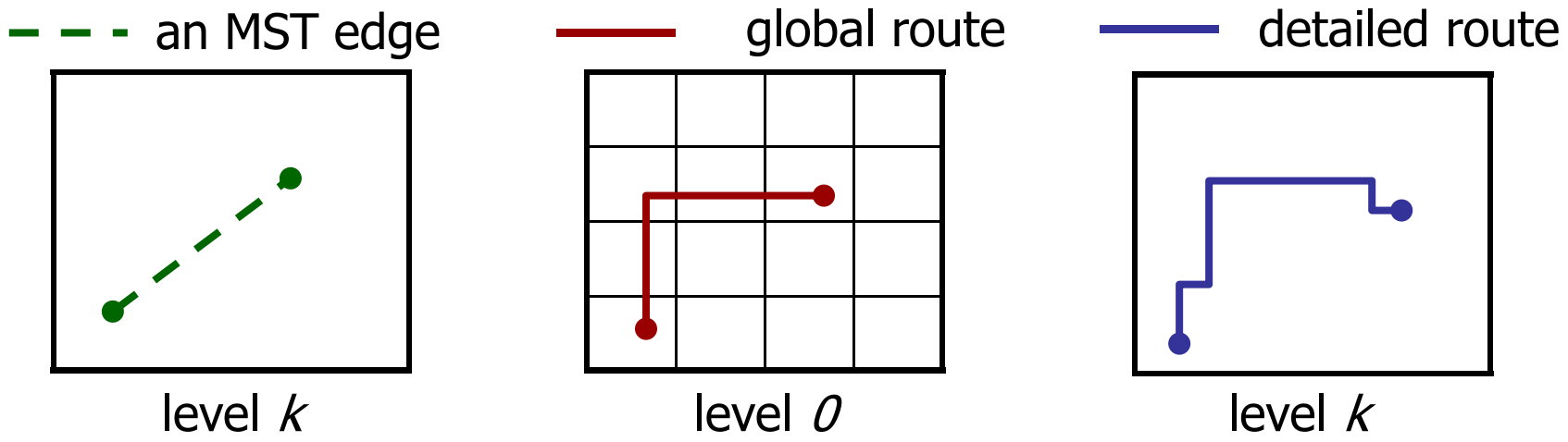


Perform global pattern routing and Dijkstra's shortest path detailed routing for local connections and then estimate routing resource for the next level.

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

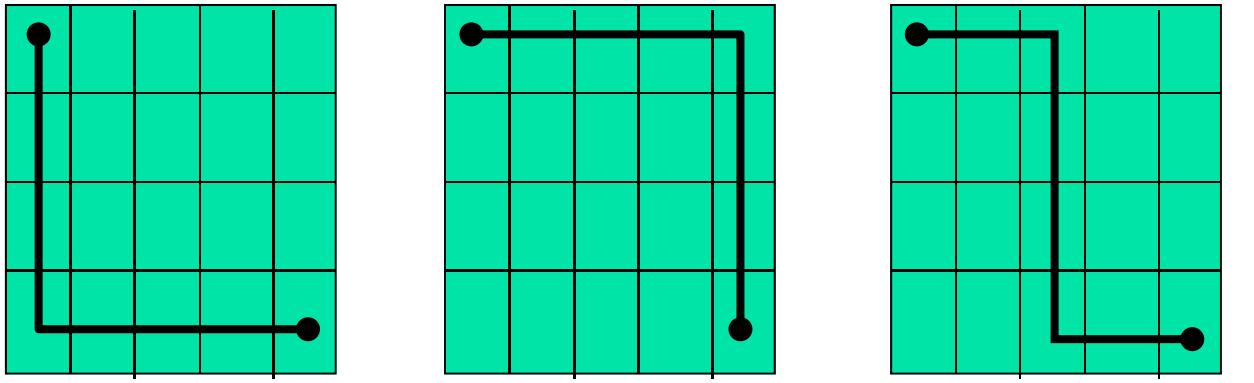
Coarsening Stage

- 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.



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

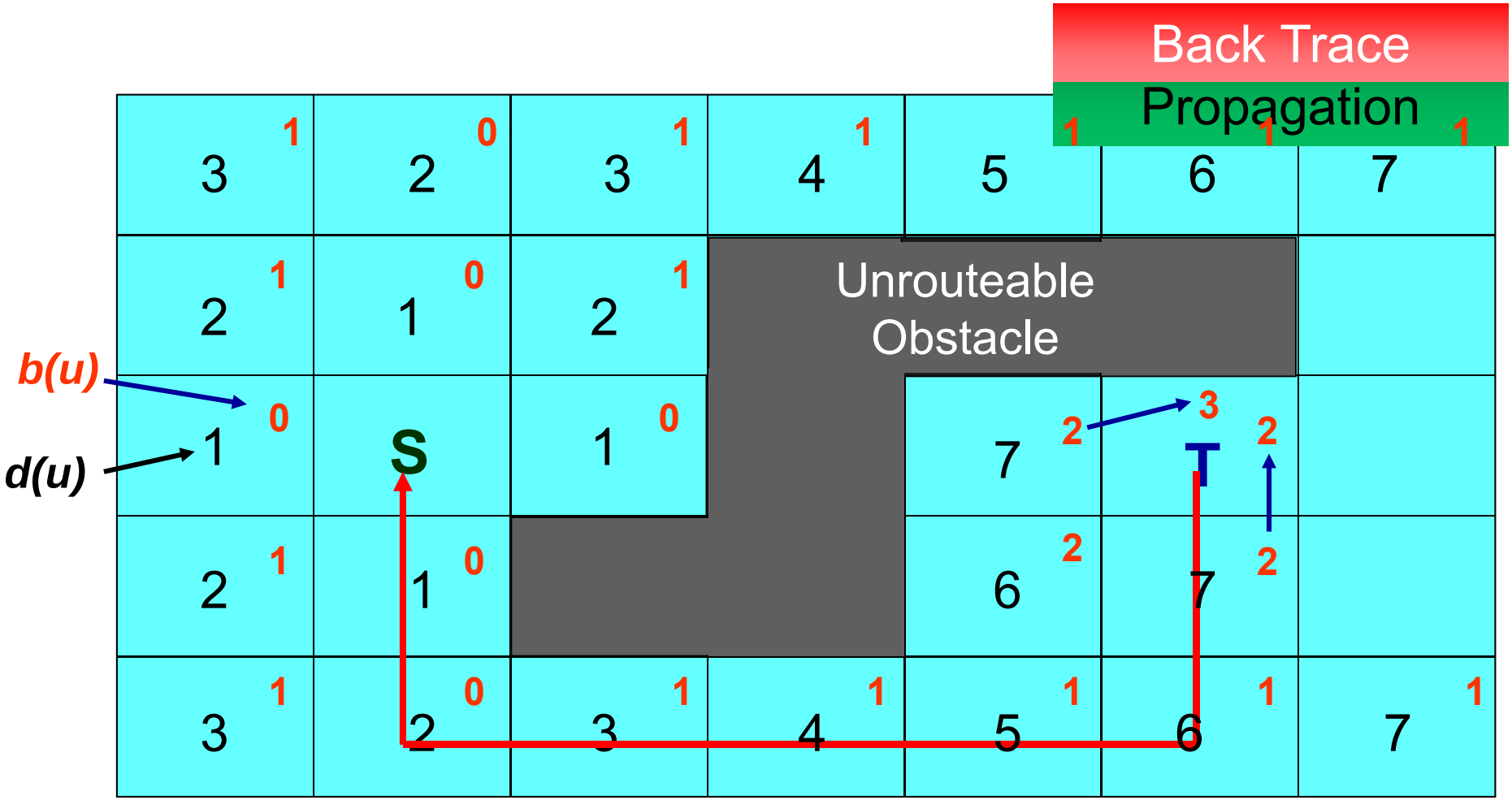
Upper L-Shaped connection

Z-Shaped connection

Detailed Routing

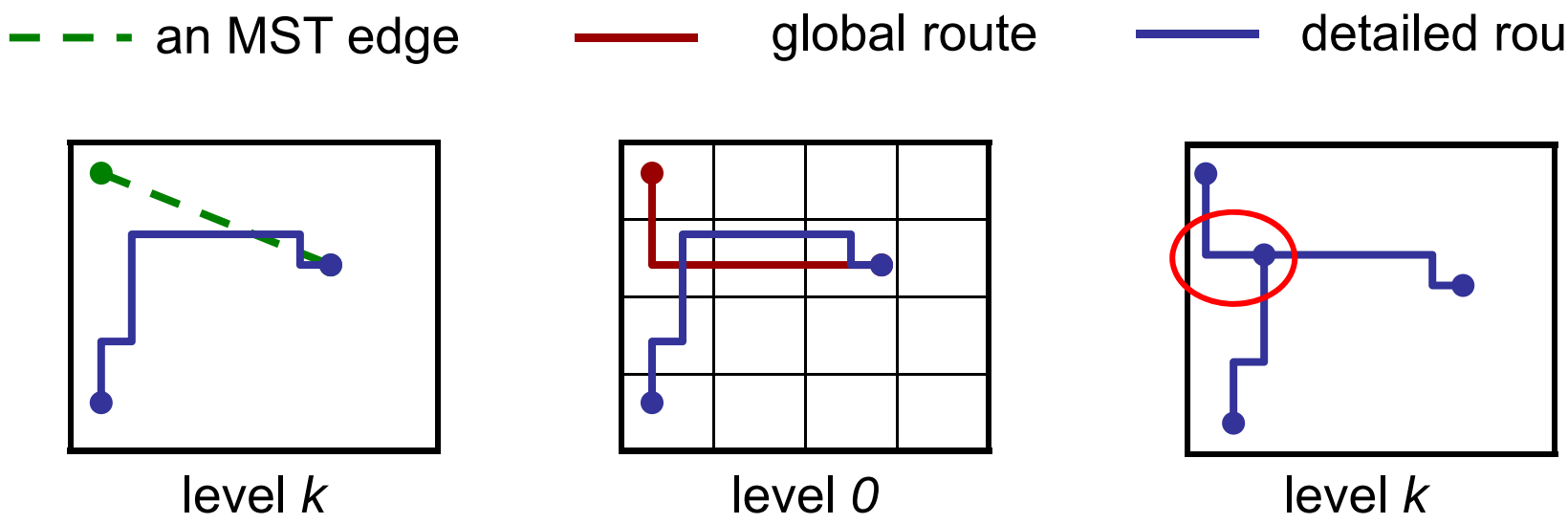
- Via minimization
  - Modify the maze router to minimize the number of bends.
- Local refinement
  - Apply general maze routing to improve the detailed routing results.
- Resource estimation
  - Update the edge weights of the routing graph after detailed routing.

Via Minimization



Local Refinement

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



Resource Estimation

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

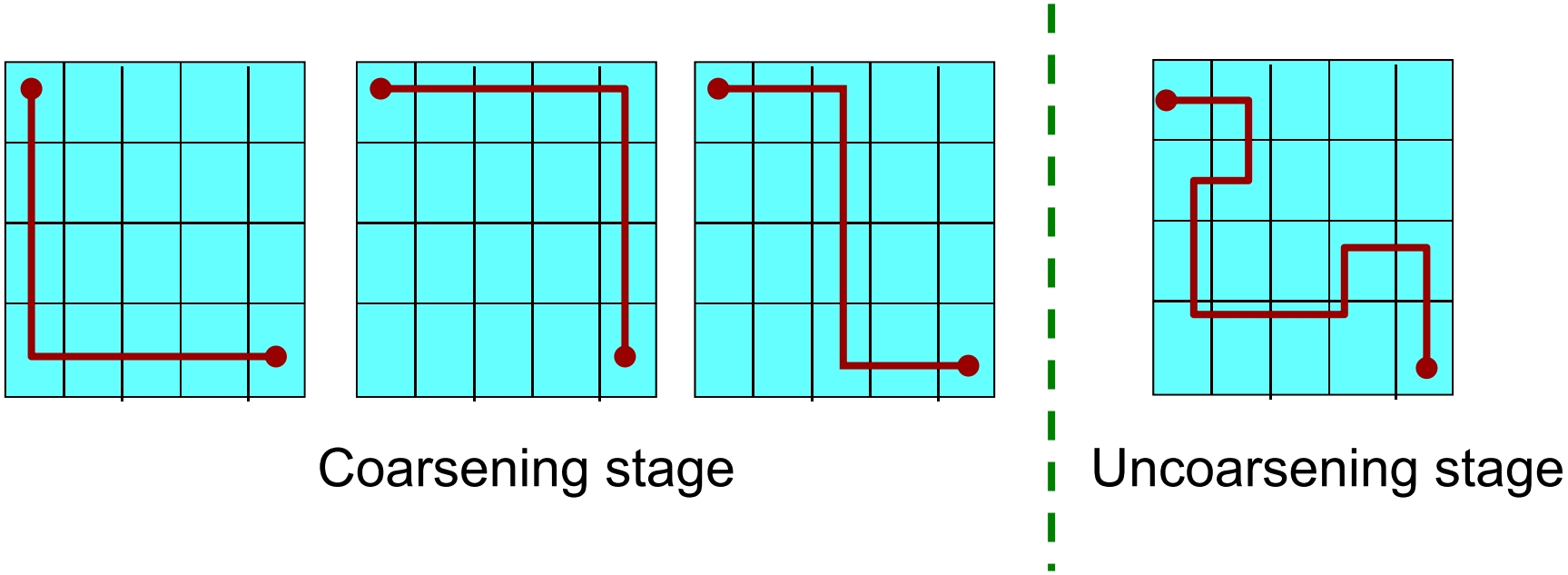
$$C_e = \frac{1}{2^{(p_e - d_e)}},$$

where p_e and d_e are the capacity and density, respectively.

- Update the congestion of routed edges to guide the subsequent global routing.

Uncoarsening Global Routing

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



Routing Comparisons

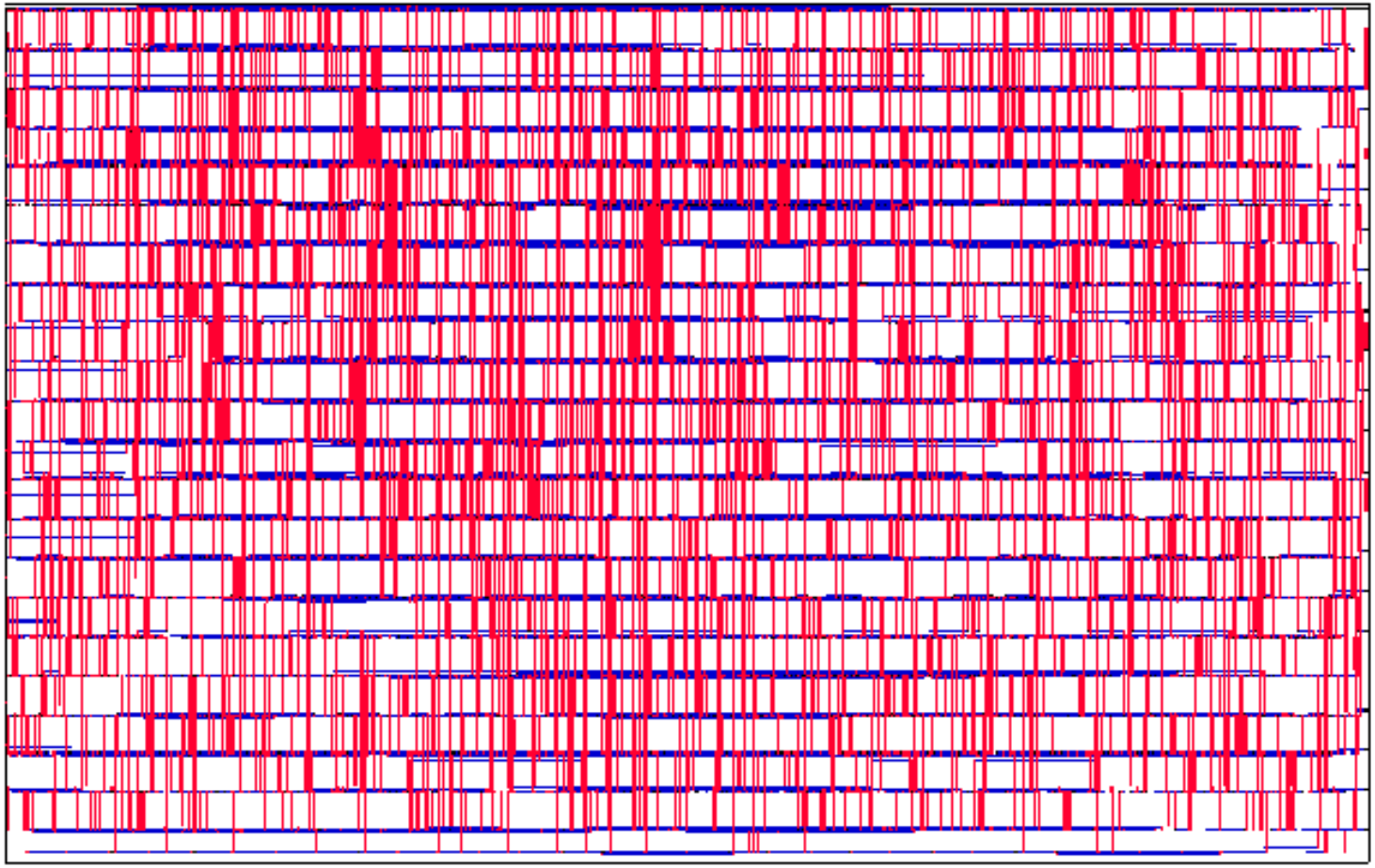
- 100% routing completion for all (11) benchmark circuits
  - Three-level routing: 0 completion (ISPD-2K)
  - Hierarchical routing: 2 completions (ICCAD-2001)
  - Previous multilevel routing: 2 completions (ICCAD-2001)
- Can complete routings using even fewer routing layers.

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.

Routing Solution for Prim2

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

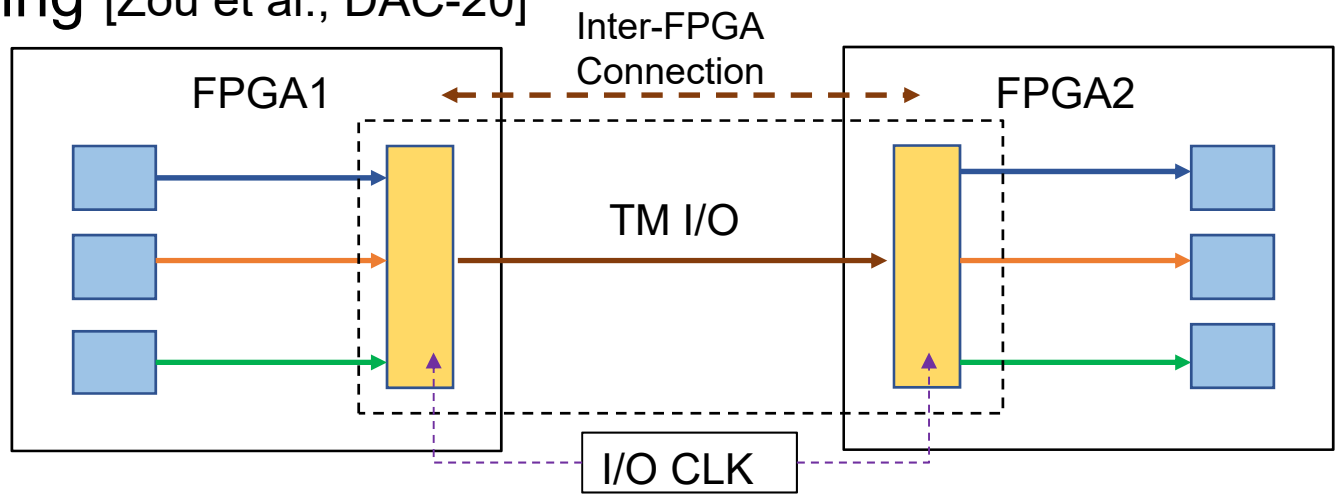


Summary: Detailed Routing

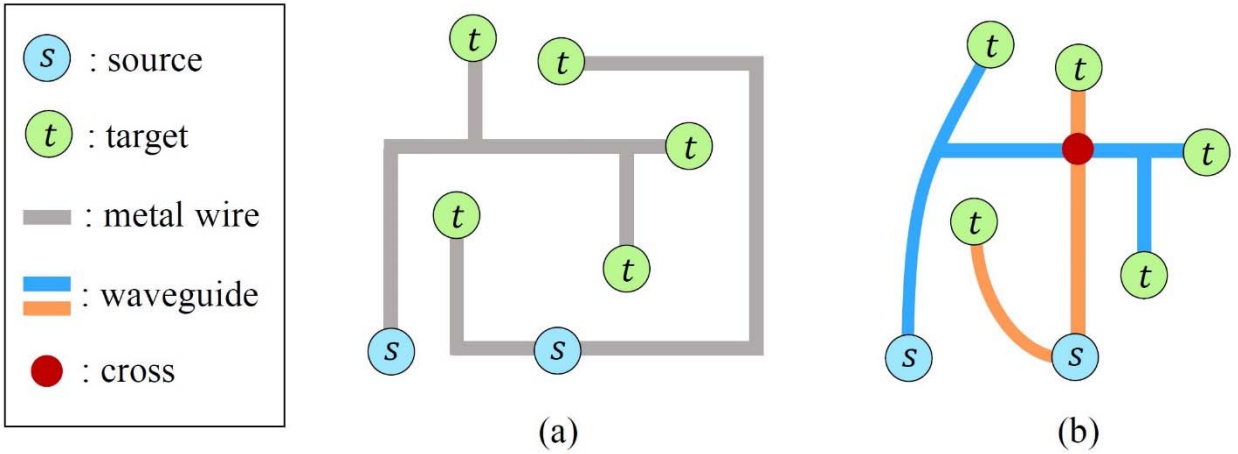
- Channel routing is considered a well-solved problem.
- Hierarchical and multilevel are keys to handle large-scale routing problems.
- Routing frameworks: go parallel??
  - Λ-shaped routing: ICCAD-02 (TCAD-03); ASP-DAC-05 (TCAD-07)
  - V-shaped routing: ASP-DAC-06
  - Two-pass bottom-up routing: DAC-06 (TCAD-08)
- Routing considerations for nanometer technology
  - Noise (crosstalk) & electro-migration constraints
  - Buffer insertion for timing optimization
  - Additional design rules: antenna effect, redundant via, OPC (optical proximity correction), CMP (chemical mechanical polishing), double/multiple patterning, e-beam, EUV (extreme ultraviolet lithography), directed self-assembly (DSA), etc.
- Initial detailed routing: ISPD-18 & -19 contests

Emerging Routing Problems

- Time-division multiplexing based system-level FPGA routing [Zou et al., DAC-20]



- On-chip optical routing [Lu, Yu, Chang, DAC-20]



Appendix A:

Other Maze Routing Algorithms

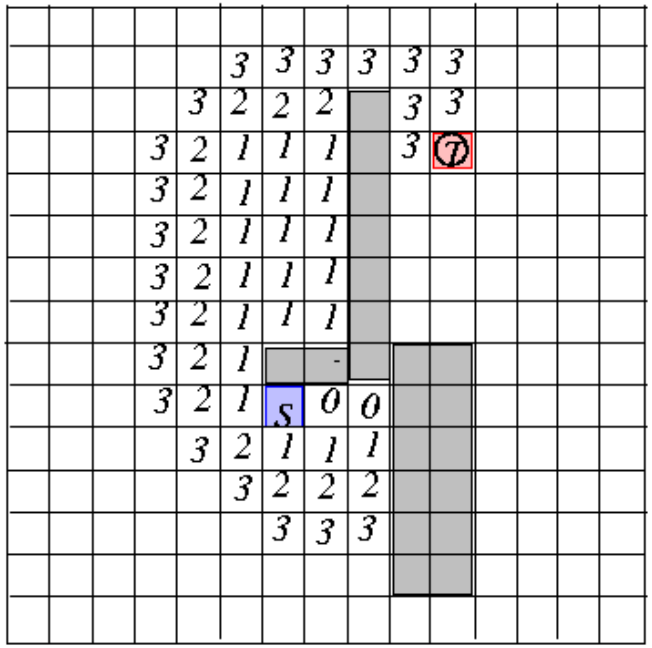
	Maze routing			Line search	
	Lee	Soukup	Hadlock	Mikami	Hightower
Time	$O(MN)$	$O(MN)$	$O(MN)$	$O(L)$	$O(L)$
Space	$O(MN)$	$O(MN)$	$O(MN)$	$O(L)$	$O(L)$
Finds path if one exists?	yes	yes	yes	yes	no
Is the path shortest?	yes	no	yes	no	no
Works on grids or lines?	grid	grid	grid	line	line

Hadlock's Algorithm

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

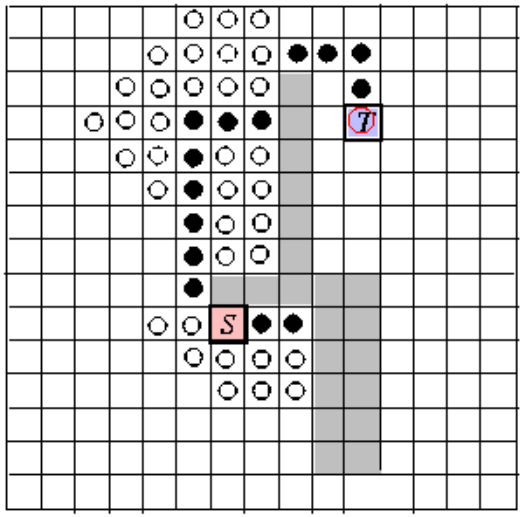
Hadlock's Algorithm (cont'd)

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

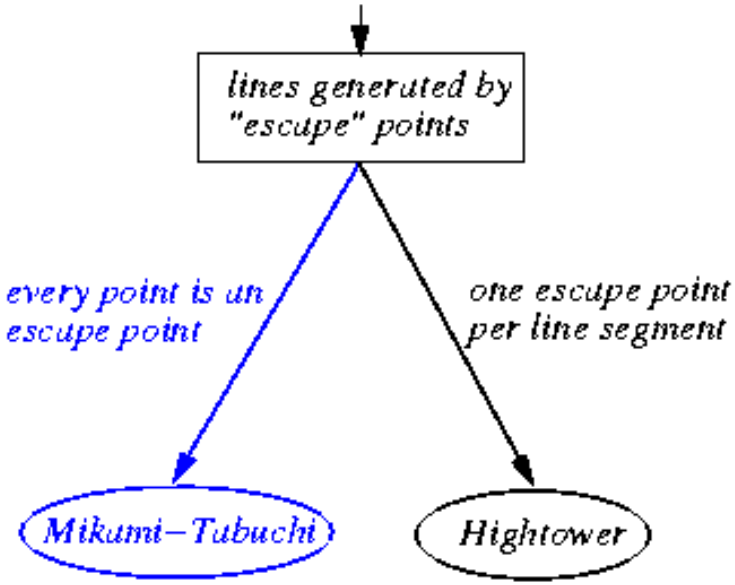


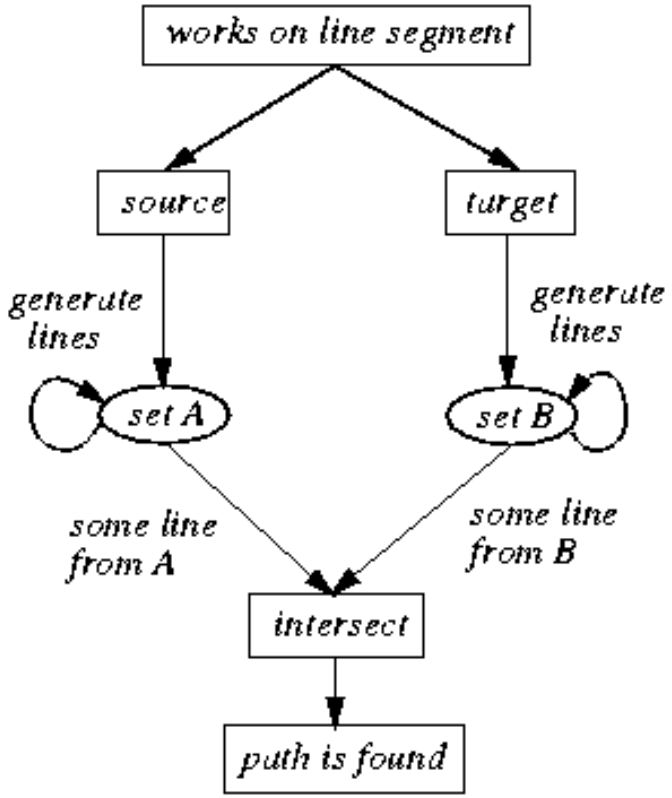
Soukup's Algorithm

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



Features of Line-Search Algorithms

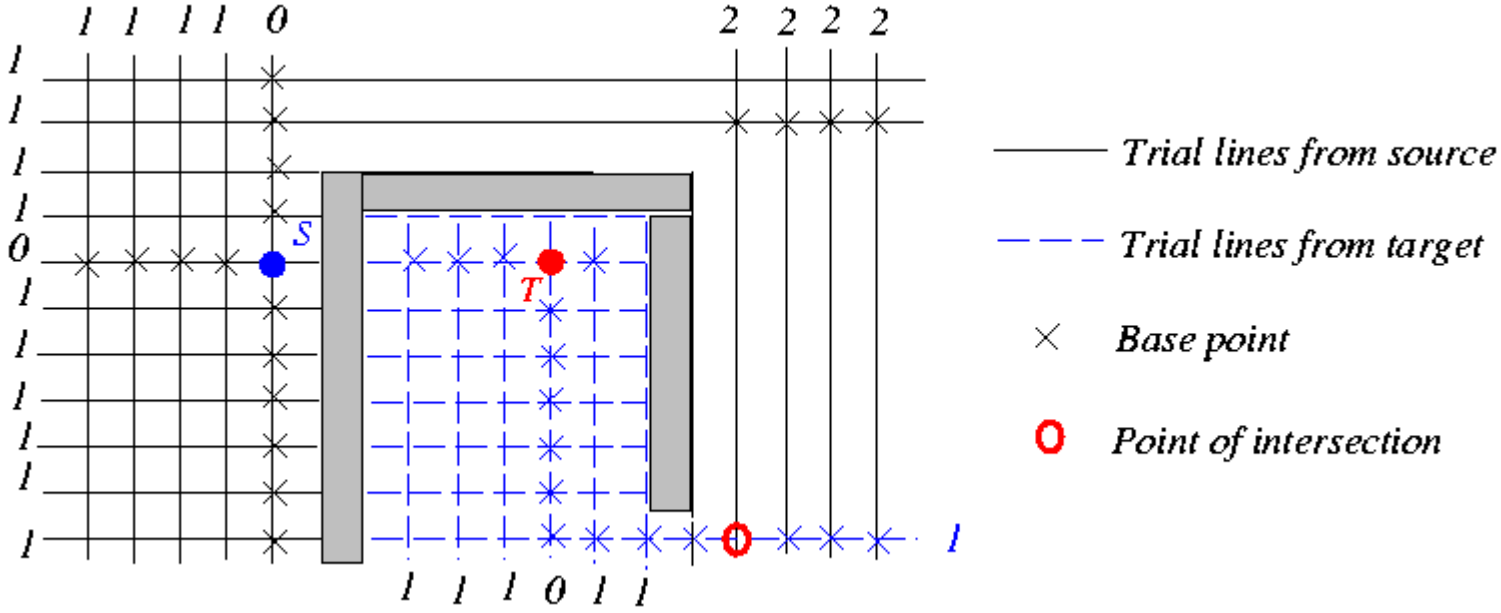




- Time and space complexities: $O(L)$, where L is the # of line segments generated.

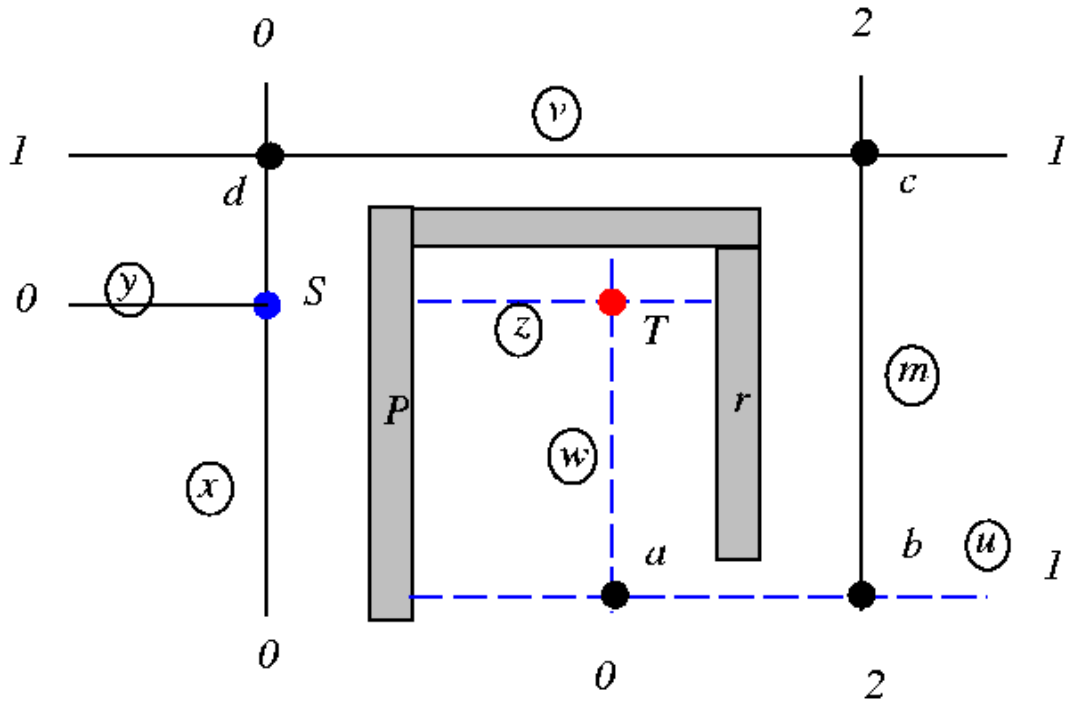
Mikami-Tabuchi's Algorithm

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



Hightower's Algorithm

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



Comparison of Algorithms

	Maze routing			Line search	
	Lee	Soukup	Hadlock	Mikami	Hightower
Time	$O(MN)$	$O(MN)$	$O(MN)$	$O(L)$	$O(L)$
Space	$O(MN)$	$O(MN)$	$O(MN)$	$O(L)$	$O(L)$
Finds path if one exists?	yes	yes	yes	yes	no
Is the path shortest?	yes	no	yes	no	no
Works on grids or lines?	grid	grid	grid	line	line

- Soukup, Mikami, and Hightower all adopt some sort of line-search operations $\Rightarrow$ cannot guarantee shortest paths.

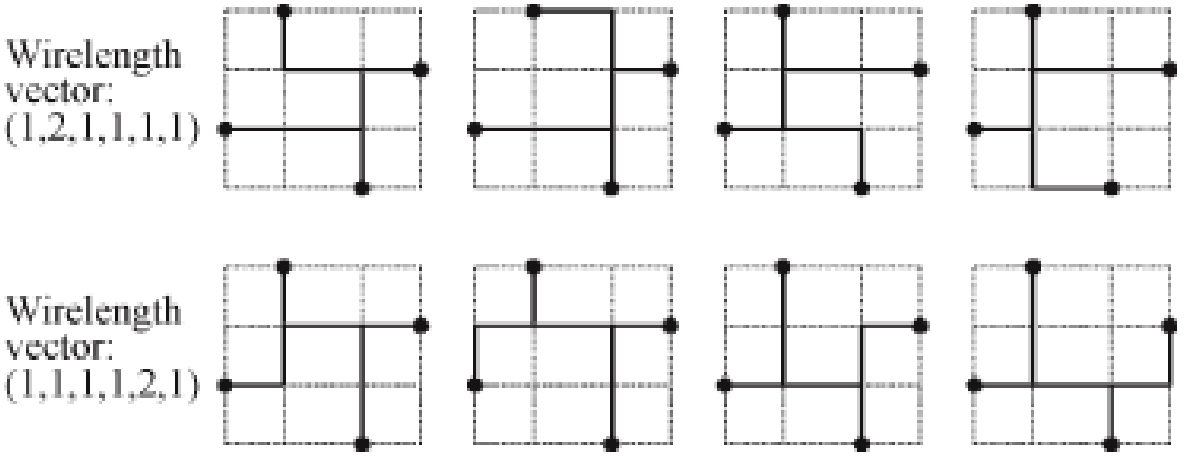
Appendix B:

FLUTE: Fast Lookup Table Based Wirelength Estimation Technique

Chris Chu

ICCAD-04, TCAD-08

(Slides provided by the authors)



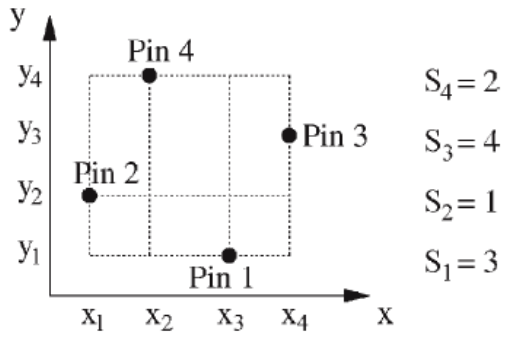
FLUTE Features

- Fast and accurate rectilinear Steiner minimum tree (RSMT) construction technique
- Finds optimal RSMTs for up to nine pins
- Near minimal (~1%) RSTs for larger nets
- Based on a precomputed lookup table for low degree nets
- Uses a net-breaking technique for high degree nets
- O(n log n) runtime

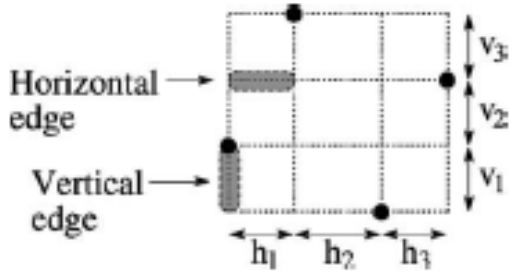
Net Representation

- Degree-n nets are partitioned into n! groups
- Potentially optimal wirelength vectors (POWVs) represent linear combination of distances between adjacent pins
- Few POWVs for each group are precomputed and stored in a table
- A potentially optimal Steiner tree (POST) associated with each POWV is also stored
- To find the optimal RSMT of a net
  - wirelengths corresponding to POWVs for the net group are computed
  - the POST of the POWV with minimum wirelength is returned

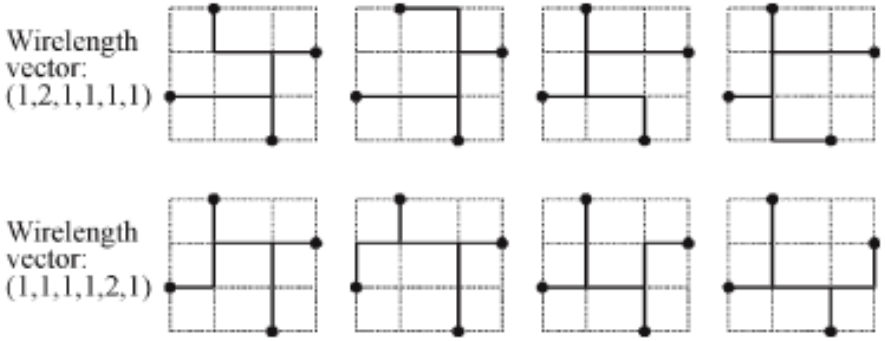
POWV and POST



Pins for a sample net

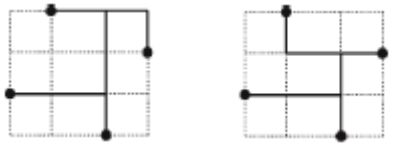


POWV: #segments in h_i/v_i



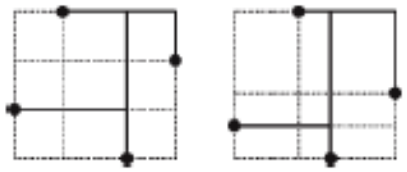
All POSTs for the sample net

- (1,2,1,1,1,1,2) cannot be shorter than (1,2,1,1,1,1,1)
- Wirelength needs to be calculated for (1,2,1,1,1,1,1) vs. (1,1,1,1,1,2,1)



FLUTE-Based RSMT Computation

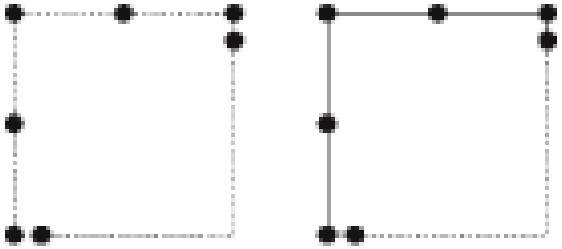
- RSMT is easy to find if all the POWVs and POSTs are precomputed in a lookup table
- Infinite number of different nets
- Nets that can share the same POWVs are grouped together
- Steiner trees are topologically equivalent if they can be transformed into each other by changing the edge lengths
  - Same position sequence



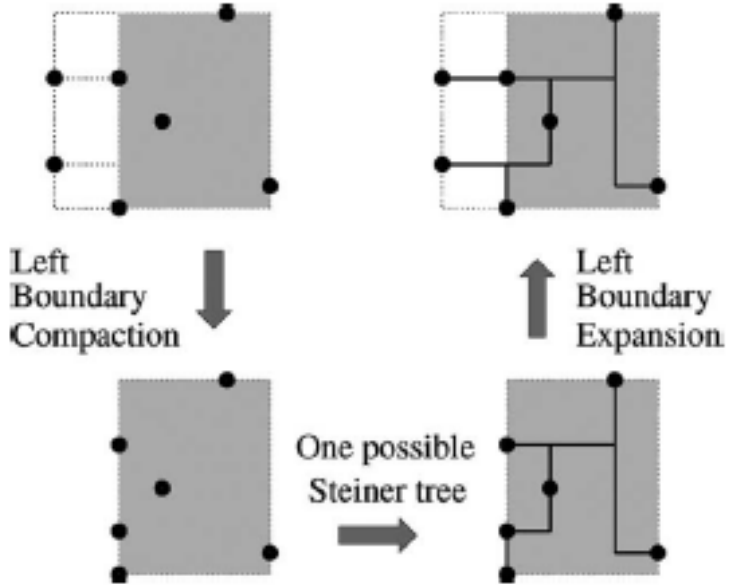
- To obtain RSMT for a net, look up the vectors for the corresponding group from the table
- Wirelength is computed based on vector entries and edge lengths
- Minimum wirelength vector's POST gives RSMT

Lookup Table Generation

- Boundary size reduction – to reduce the number of Steiner trees generated
  - Grid size is reduced by compacting one of the four boundaries (pins on boundary shifted to the adjacent grid line)
  - Original Steiner trees are generated by expanding

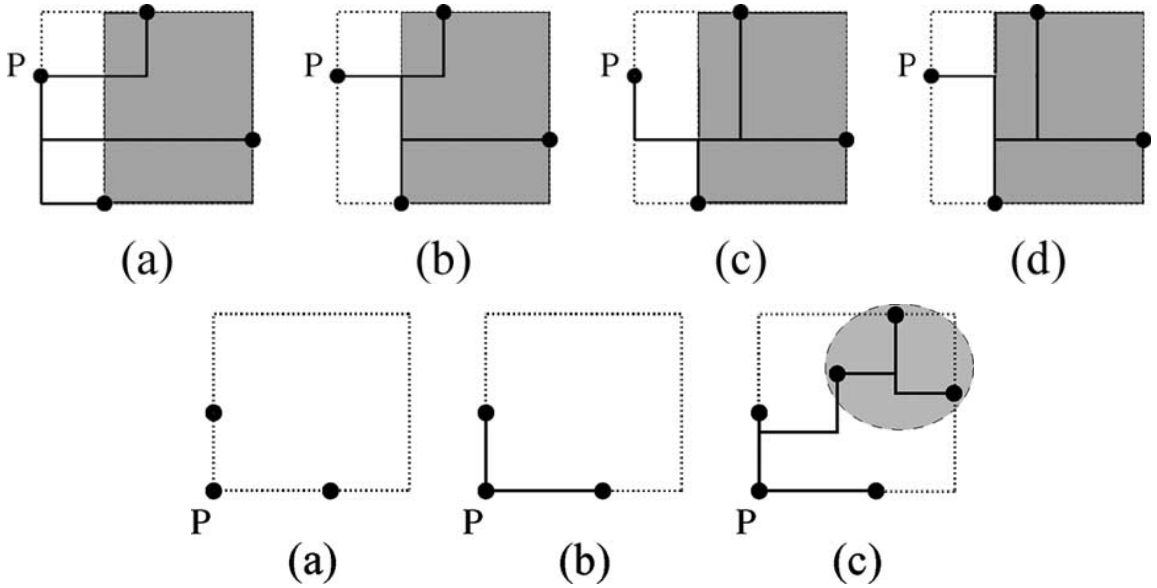


A non-compactible grid



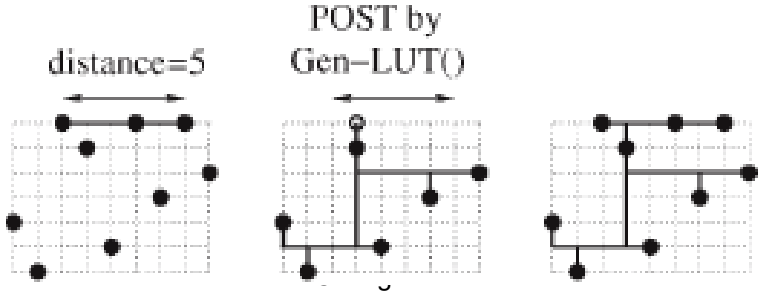
Grid Compactibility

- A grid is compactible if it has boundary with only one pin
- A grid is compactible if it has a corner with one pin and both boundaries adjacent have exactly one other pin
- A grid is compactible if it has up to six pins on all four boundaries



POWV Generation

- For a grid with seven pins, boundary compaction together with near-ring structures can generate all POWVs
  - Near-ring structure: bounding box that surrounds the grid with edges connecting one of the 7 pairs of the adjacent pins removed.
- For eight or more pins Connect-adj-pins() used to generate extra trees
  - It connects two or more adjacent pins on the same boundary and introduces a branch along that boundary
  - Then pins are replaced by a pseudo-pin
  - Gen-Lut() function is called to generate POSTs of reduced grid
  - Original POSTs are generated by connecting the branch with the POSTs of reduced grid
- Boundary compaction together with Connect-adj-pins() is sufficient to generate all POWVs for nets with up to 10 pins

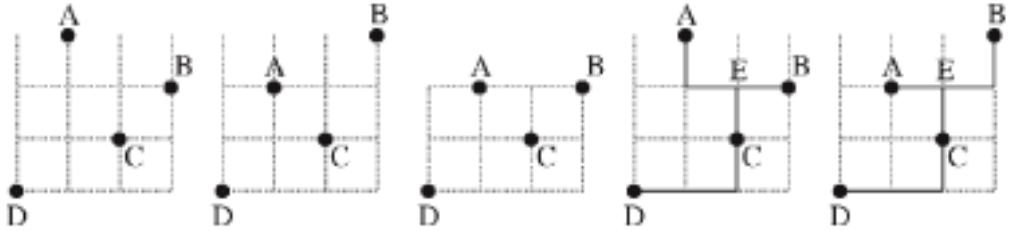


Algorithm: Lookup Table Generation

Gen-LUT(G) Algorithm
Input: A grid G with some pins at grid nodes
Output: One POST for each POWV of the group associated with G
begin
If G is simple enough,
generate and **return** the set of POSTs for G
else if any boundary b contains only one pin,
return Expand-b(Gen-LUT(Compact-b(G))
else if there is a corner with one pin such that both its adjacent boundaries b1 and b2 have one other pin,
return Prune(Expand-b1(Gen-LUT(Compact-b1(G)))
U (Expand-b2(GEN-LUT(Compact-b2(G)))
else
if there are 7 pins with all 7 pins on boundaries,
S = {Trees with near-ring structure connecting all pins}
else if there are more than 8 pins with more than seven pins on boundaries
S = Connect-adj-pins(G,d) where d = # of pins -3
return Prune(S U Expand-left(Gen-LUT(Compact-left(G))) U
Expand-right(Gen-LUT(Compact-right(G))) U
Expand-top(Gen-LUT(Compact-top(G))) U
Expand-bot(Gen-LUT(Compact-bot(G))))
end

Lookup Table Reduction

- In order to reduce storage requirements for POWVs and associated POSTs in the lookup table, similar POWVs can be grouped together and differences can be stored
  - Numbers of POWVs or the POSTs do not decrease
- Alternatively, equivalence of groups can be exploited to generate less POWVs and POSTs
  - Storage requirements and table generation time decreases



  - Boundary reduction causes two different nets to have same set of POWVs
  - POSTs can also be shared
  - Equivalency of groups can reduce the number of groups generated and stored by 25.8x

Wirelength Computation Speedup

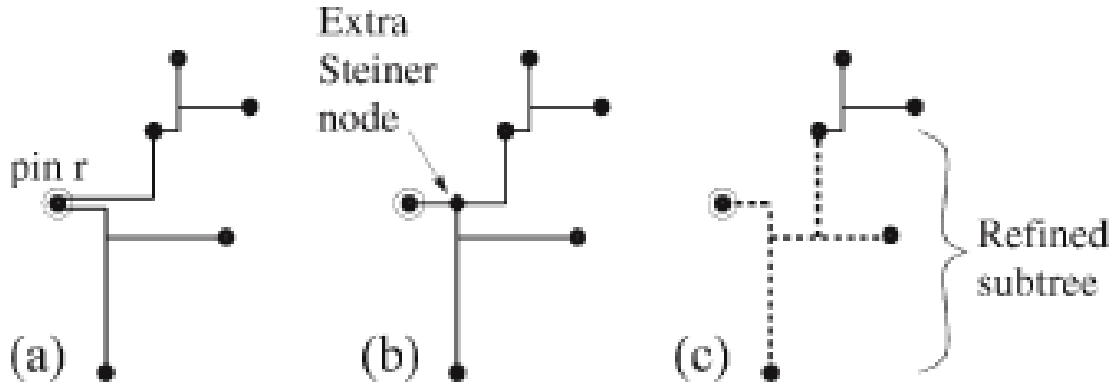
- Instead of adding edge lengths (to reduce number of addition operations), an MST-based approach can be used to reduce minimum wirelength computation time
  - q nodes corresponding to q POWVs in the set and one more node to represent wirelength vector (1,…,1)
  - Weight of each edge is the number of addition/subtraction required to convert from wirelength of one vector to other
  - Total edge weight of MST gives the number of additions/subtractions required to compute POWVs
  - Based on the fact that most POWVs are similar to each other
    - i.e., wirelength computations differ by a few additions/subtractions
  - Can significantly speed up evaluation of high-degree nets

Net Breaking

- Net Breaking for High-degree Nets
  - Table lookup approach is only practical for low-degree nets due to storage and computation time requirements
  - A lookup table is constructed up to degree D = 9
  - Higher-degree (>9) nets are broken into subnets with degrees 2 to 9
- Net Breaking Heuristics
  - If a net is broken at r, then two subnets: pin 1 to r, and pin r to n
  - A score is computed to evaluate the most desirable way of breaking (based on if the edge is likely to be counted in both partitions or not)
  - Score $S(r) = S_1(r) - \alpha S_2(r) - \beta S_3(r) - \gamma S_4(r)$
    (α=0.3, β=7.4/(n+10), γ=4.8/(n-1))
  - $S_1(r) = y_{r+1} - y_{r-1}$ (it is better to break at pin r if large)
  - $S_2(r) = 2(x_3 - x_2)$ if $s_r = 1$ or 2
    $= x_{sr+1} - x_{sr-1}$ if $3 \le s_r \le n-2$
    $= 2(x_{n-1} - x_{n-2})$ if $s_r = n - 1$ or n
  - $S_3(r) = |s - (n+1)/2| \times h + | r - (n+1)/2| \times v$, $h = (x_{n-1} - x_2)/(n-3)$ and $v = (y_{n-1} - y_2)/(n-3)$
  - $S_4(r)$ = total HPWL of the two subnets (direct way to predict resulting wirelength)

RSMT Construction

- After subtrees of subnets are generated, they are combined to form ST
  - Redundant edges detected in constant time
  - Removed via extra Steiner node
  - Local refinement technique using FLUTE to reduce wirelength by reconstructing subtree connecting the neighborhood of breaking pin

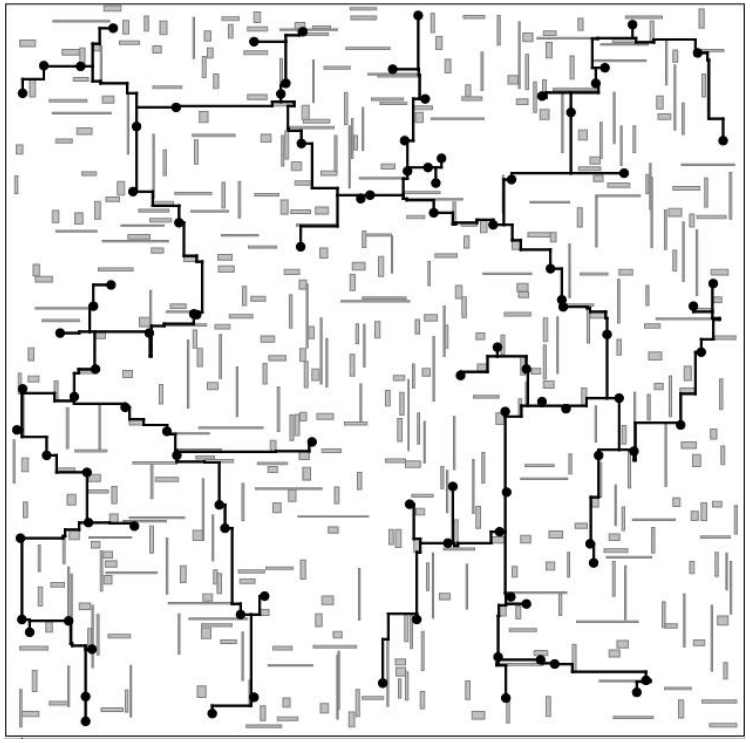


Appendix C:

Obstacle-Avoiding Rectilinear Steiner Tree Construction

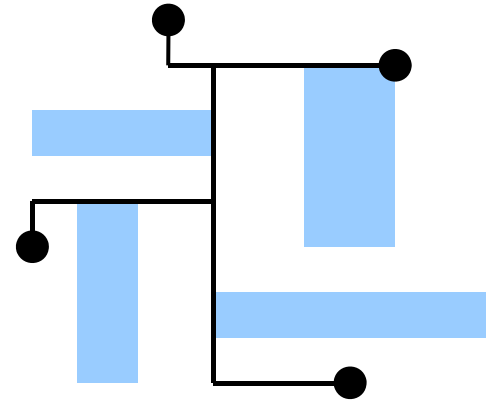
Lin, Chen, Li, Chang, and Yang

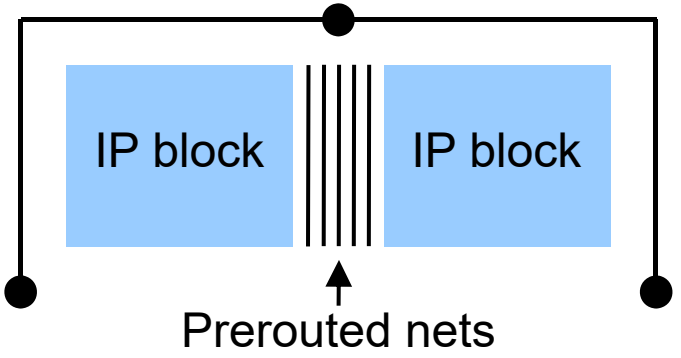
ISPD-07, TCAD-08



Introduction to OARSMT Problem

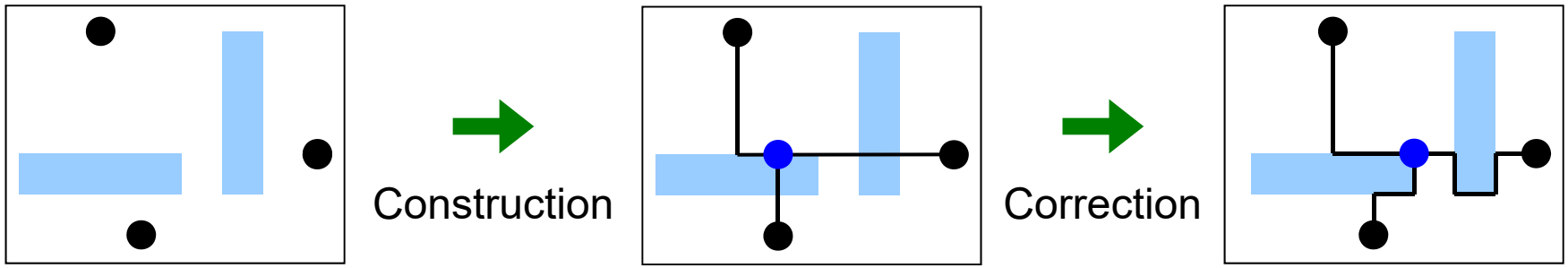
- Given a set of pins and a set of obstacles, an obstacle-avoiding rectilinear Steiner minimal tree (OARSMT)
  - Connect those pins, possibly through some Steiner points
  - Use only vertical and horizontal edges
  - Avoid running through any obstacle
  - Have a minimal total wirelength
- It becomes more important than ever for modern nanometer IC designs.
  - The design needs to consider numerous routing obstacles incurred from
    - Prerouted nets
    - Large-scale power networks
    - IP blocks, etc





Construction-by-Correction Approach

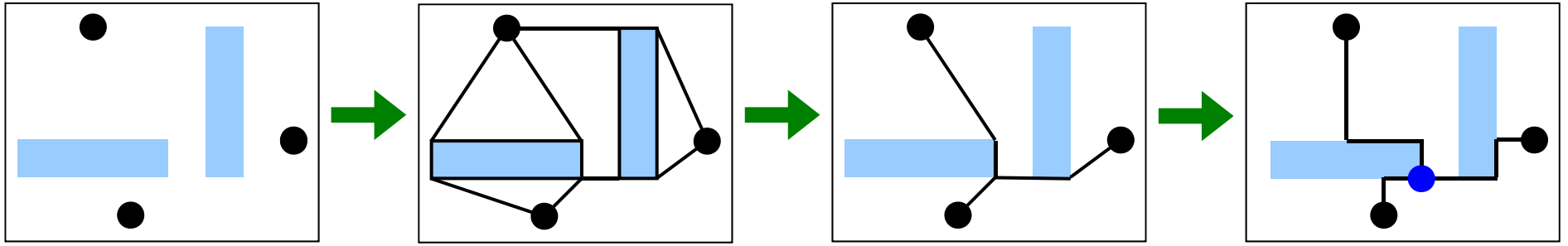
- Construct an initial tree without considering obstacles
- Correct edges overlapping obstacles



- Lack a global view of obstacles
- Have a limited solution quality
- Feng et al., ISPD-06
  - Construct OARSMT in the λ-geometry plane

Connection-Graph Based Approach

- Construct a connection graph in which there is a desired OARSMT
- Apply searching techniques to find the desired OARSMT

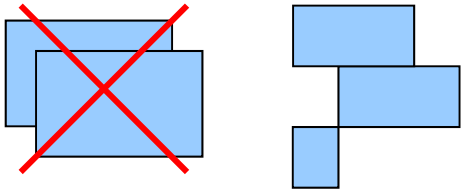


- Prune many redundant edges to reduce problem size
- Obtain much better solution quality
- Shen et al., ICCD-05
  - Achieve good results but can be further improved

Problem Formulation

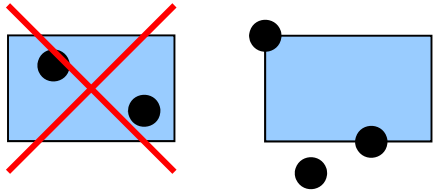
- Given m pins $\{p_1, p_2, \ldots, p_m\}$ and k obstacles $\{o_1, o_2, \ldots, o_k\}$, construct an OARSMT such that the total wirelength of the tree is minimized.

Obstacles overlap each other



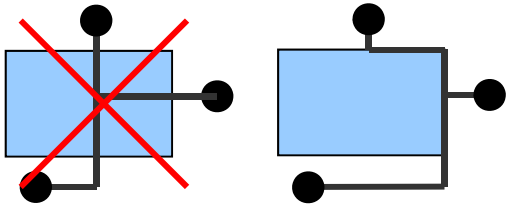
Obstacles are point-touched or line touched

A pin locates inside an obstacle



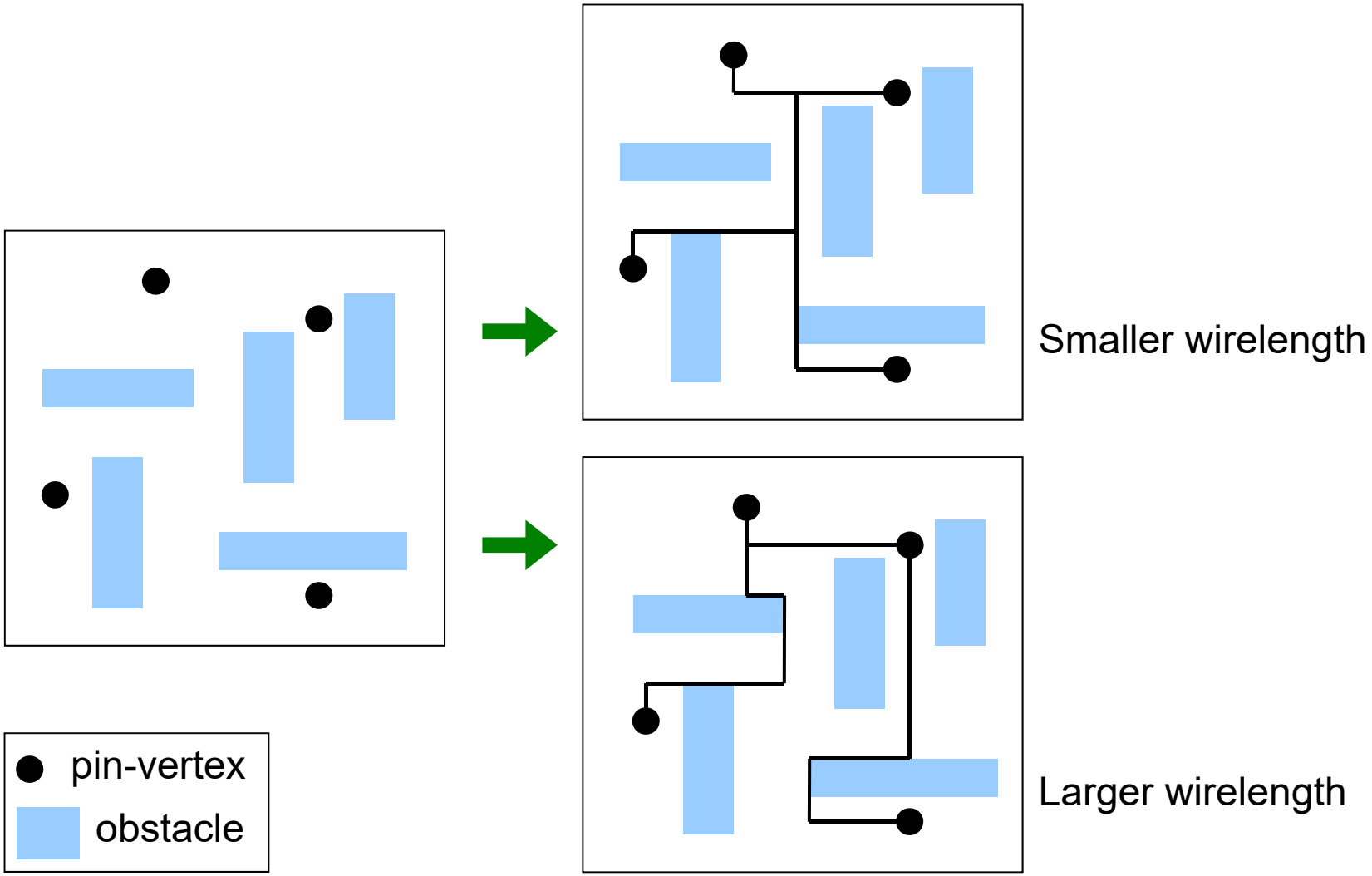
A pin is at the corner or on the boundary of an obstacle

An edge intersects an obstacle

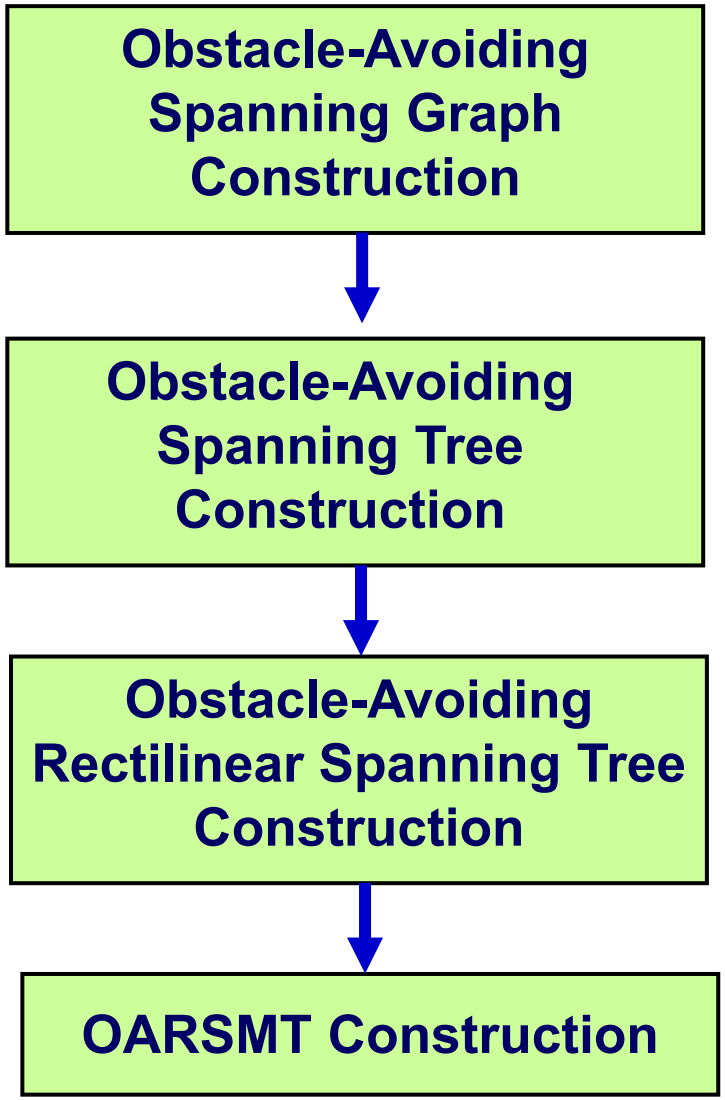


An edge is point-touched or line-touched with an obstacle

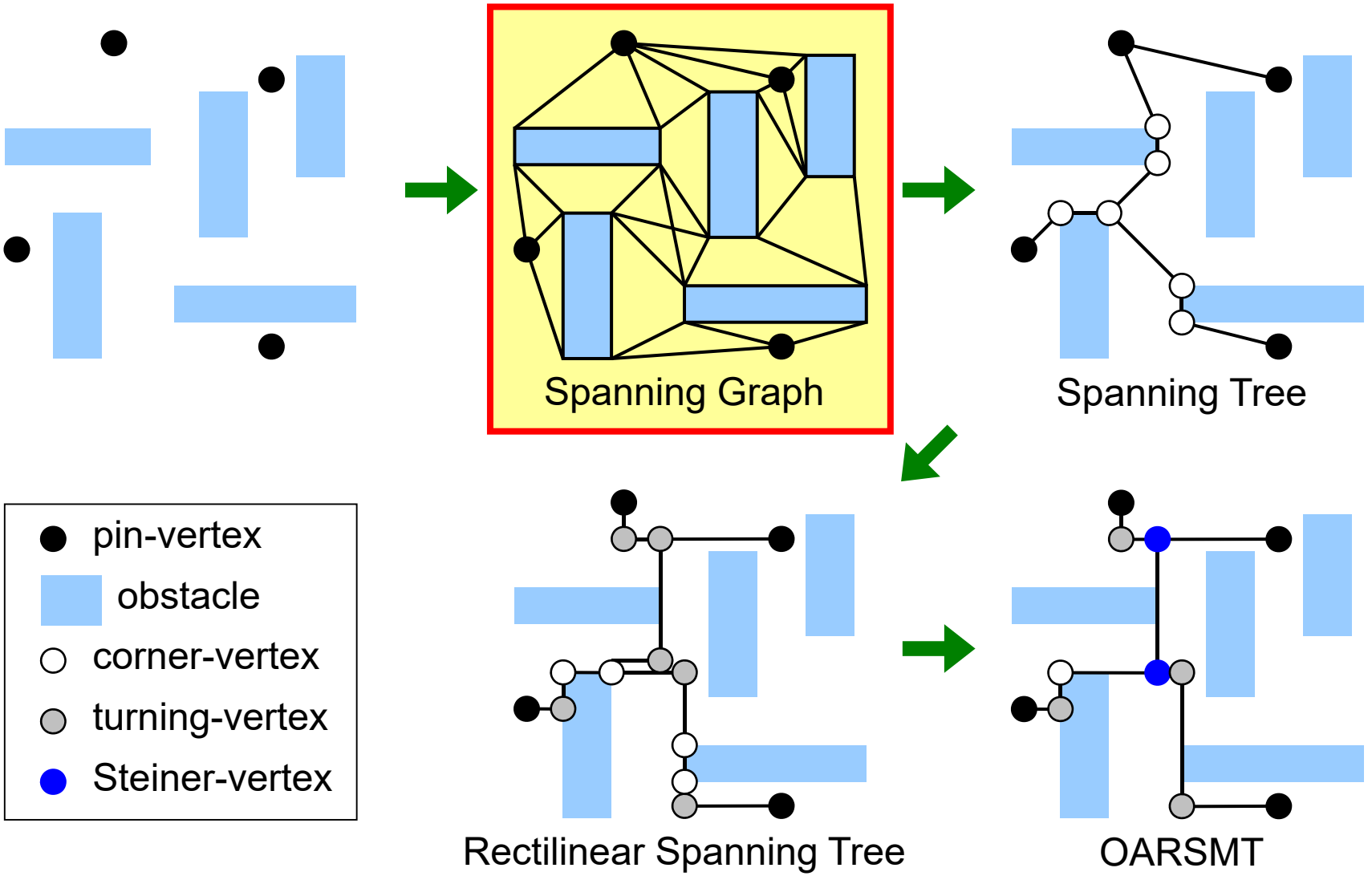
An OARSMT Example



Algorithm Flow

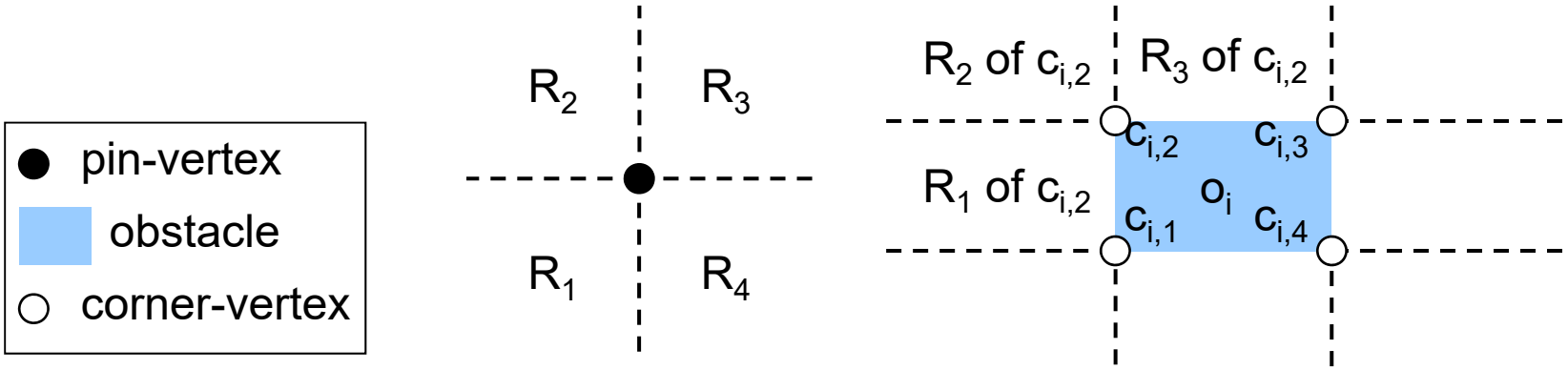


Algorithm Flow

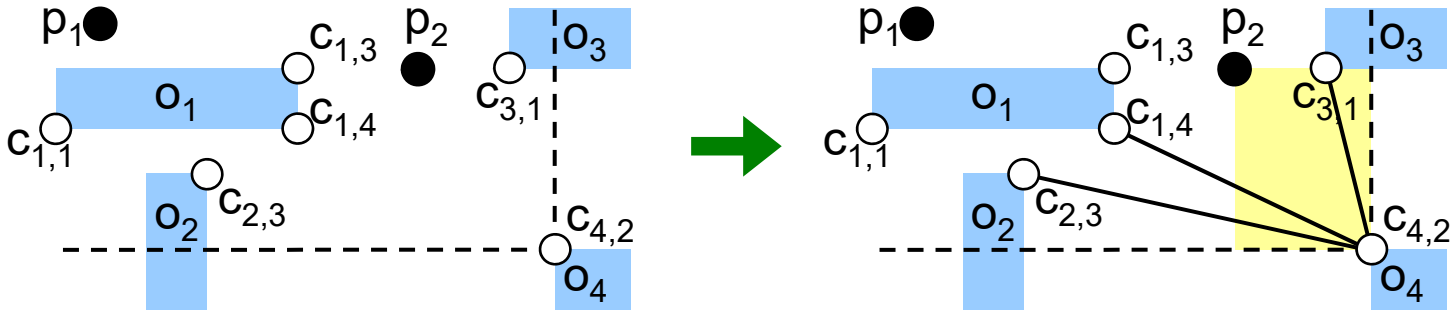


Spanning Graph Construction

- The plane is divided into four regions for each vertex.



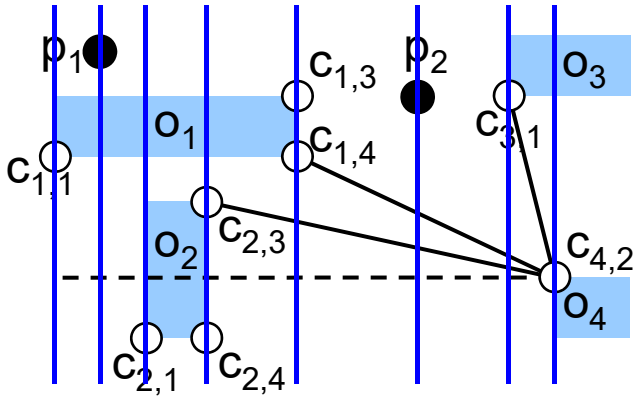
- v_1 and v_2 are connected if no other vertex or obstacle is inside or on the boundary of the bounding box of v_1 and v_2.



Spanning graph for R_2 of $c_{4,2}$

Spanning Graph Construction – Example

- Apply a sweeping line algorithm for each region
- Use blocking information to check a vertex is blocked or not



Candidate set

$c_{1,1}$
$c_{2,3}$
$c_{1,4}$
p_2
$c_{3,1}$

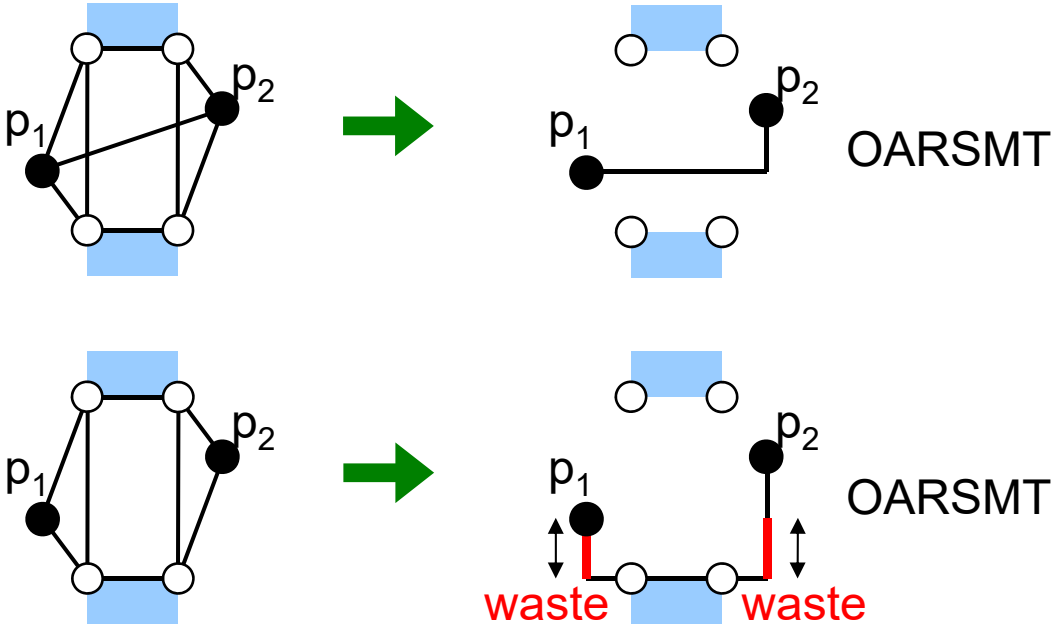
Blocking information

o_1
o_2
o_3

Spanning graph for R_2 of $c_{4,2}$

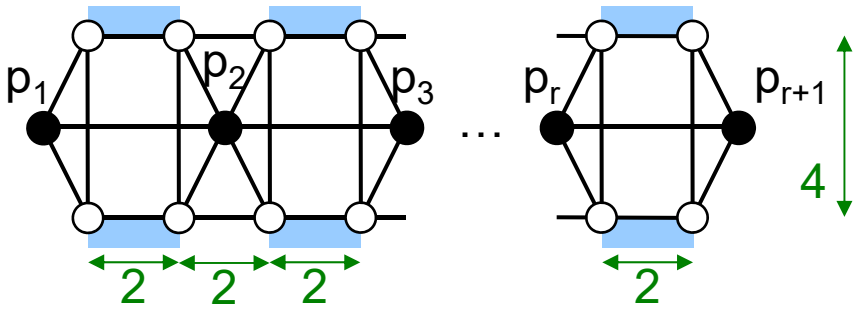
Properties of Our Spanning Graph (1/2)

- Our spanning graph guarantees a shortest path of any two vertices.

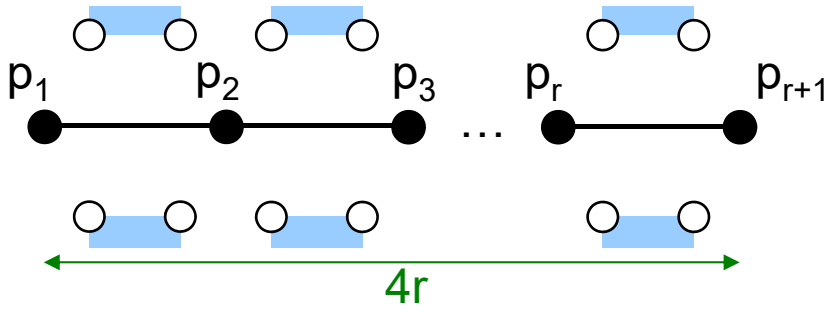


- Spanning graph of Shen et al. (ICCD-05) does not guarantee it.

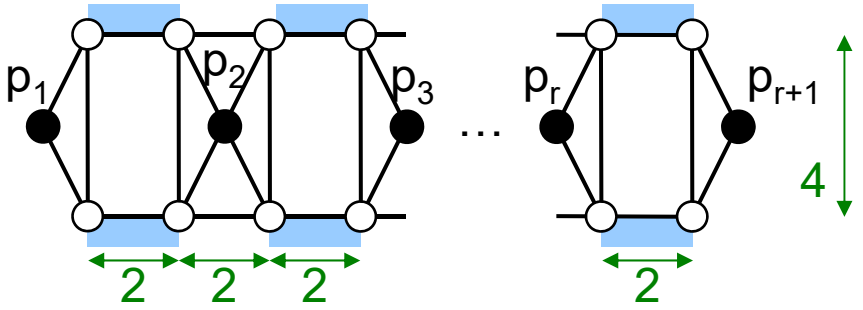
Properties of Our Spanning Graph (2/2)



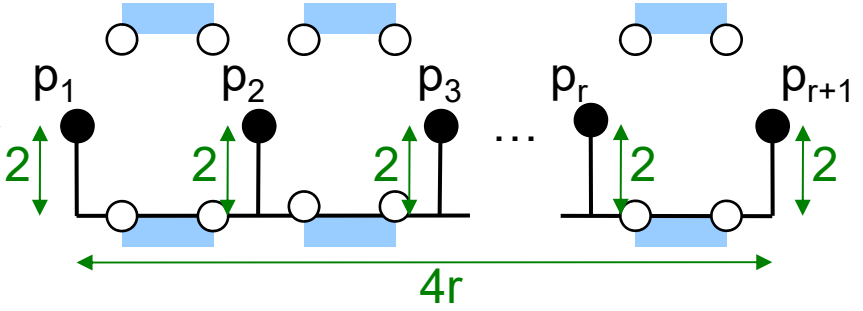
Our spanning graph



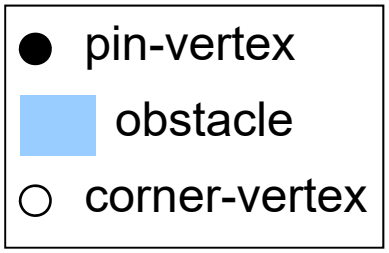
OARSMT total wirelength = 4r



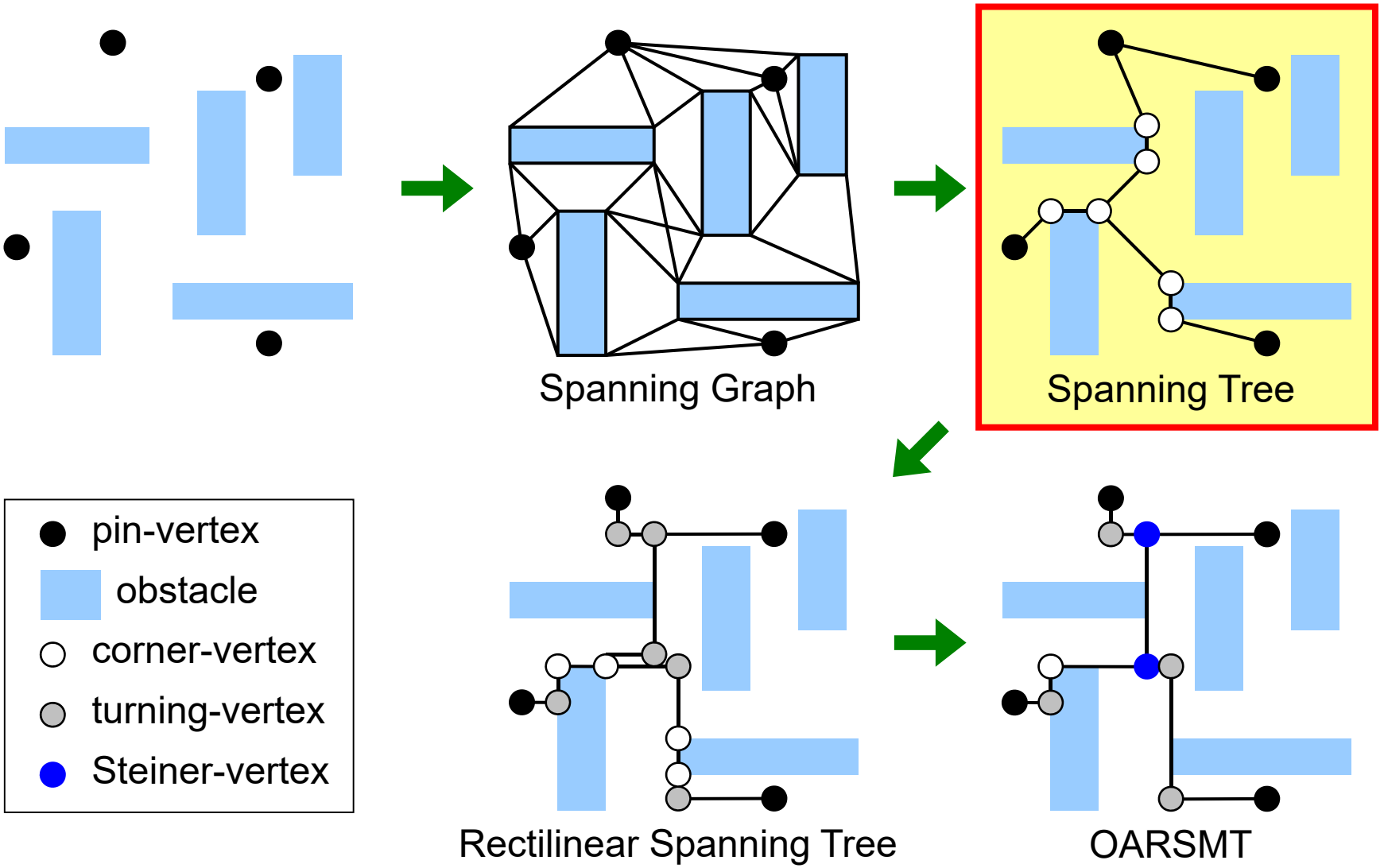
Spanning graph of Shen et al.



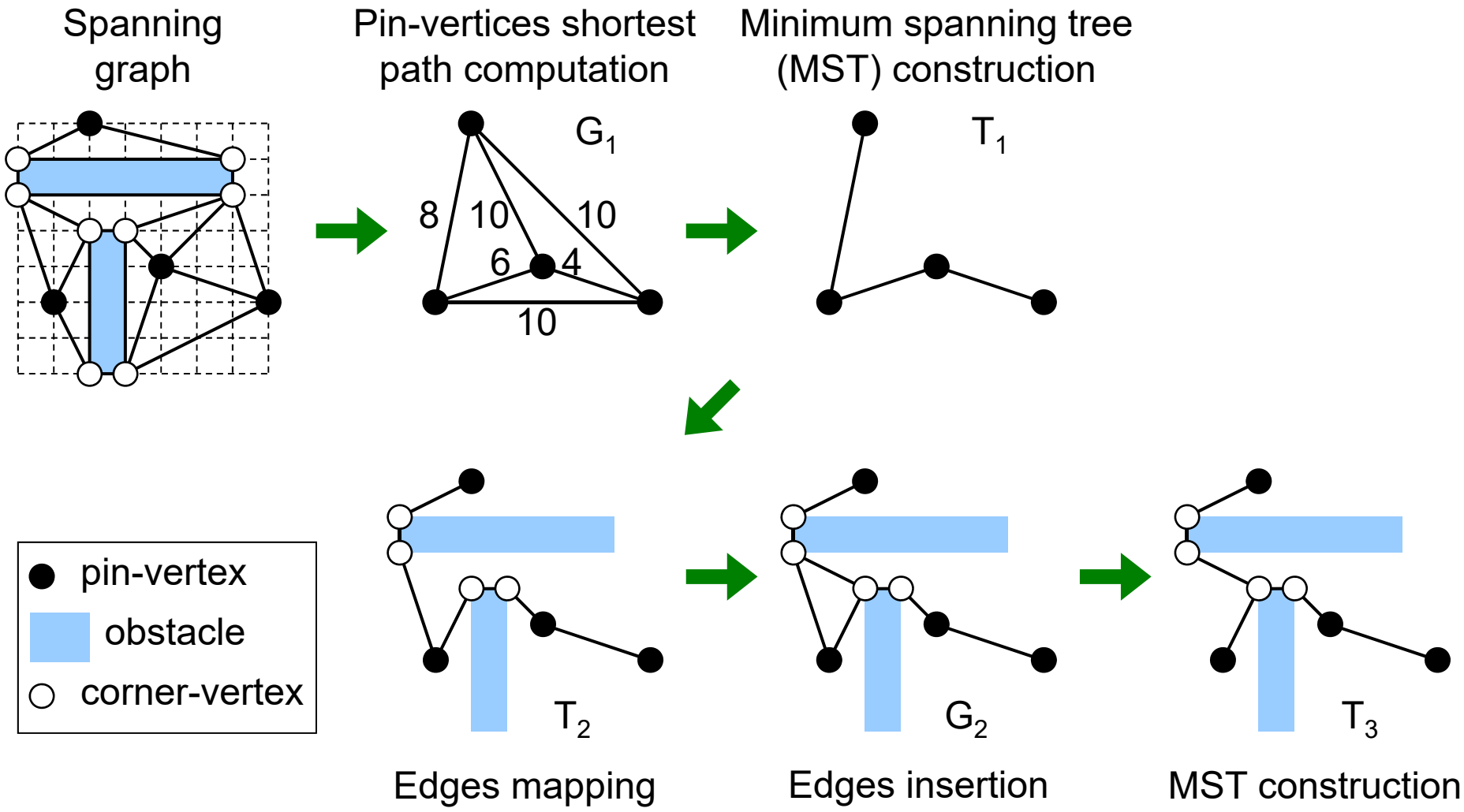
OARSMT total wirelength = 6r+2



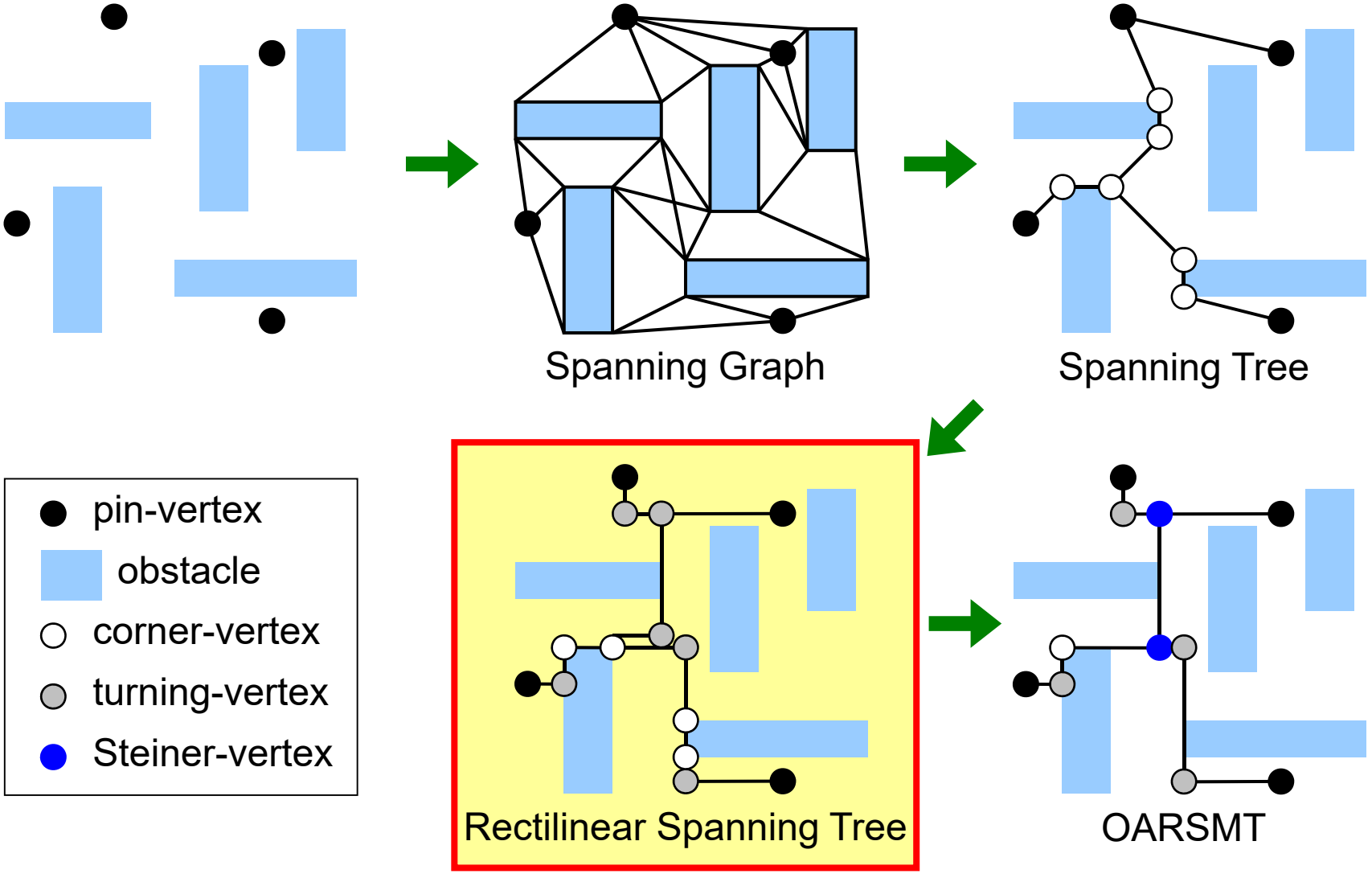
Algorithm Flow



Spanning Tree Construction



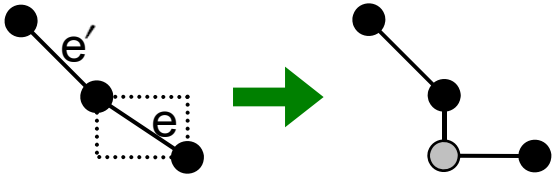
Algorithm Flow



Rectilinear Spanning Tree Construction

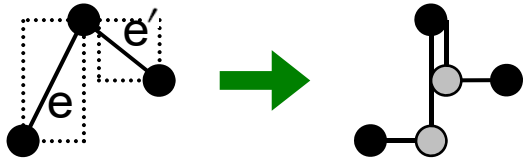
- Transform slant edges into vertical and horizontal edges
  - Longer edges are transformed first.
  - Three cases for a slant edge e and its neighboring edge e':

1. They are in opposite regions.



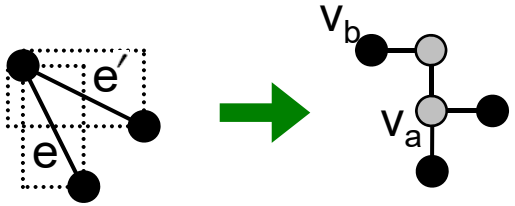
e is transformed randomly

2. They are in neighboring regions.



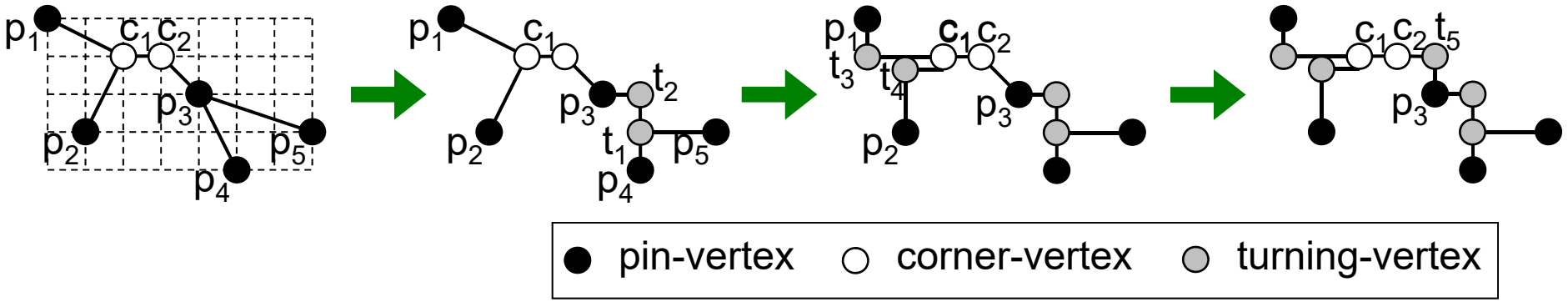
e and e' are transformed with edge overlap

3. They are in the same region.

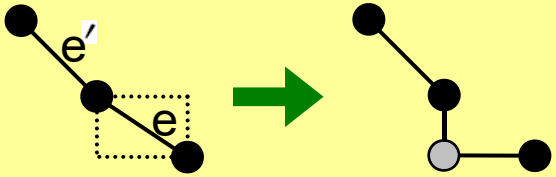


e and e' are transformed with edge overlap
(v_a,v_b) is transformed randomly

Rectilinear Spanning Tree Construction – Example

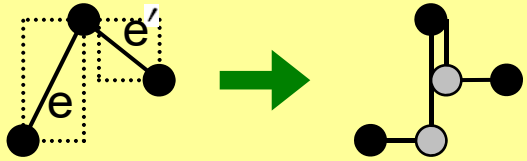


1. They are in opposite regions.



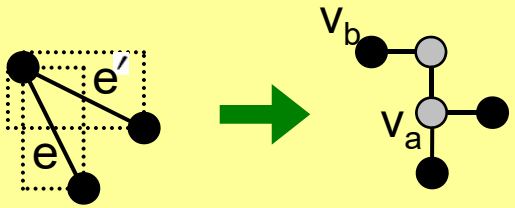
e is transformed randomly

2. They are in neighboring regions.



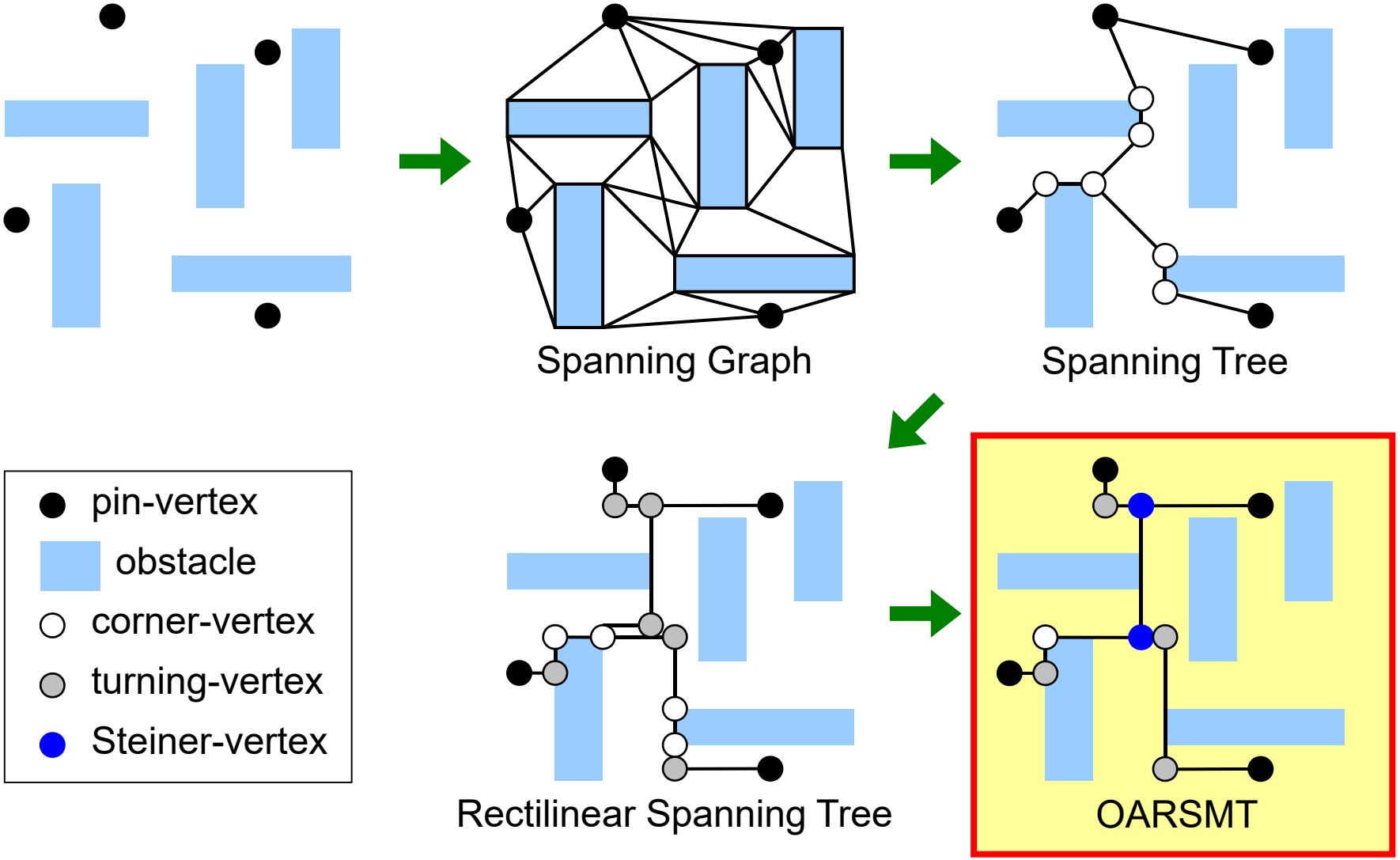
e and e’ are transformed with edge overlap

3. They are in the same region.



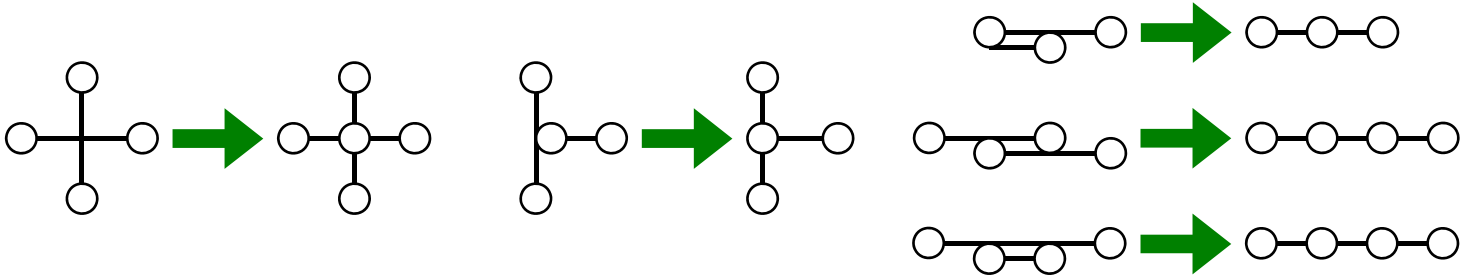
e and e’ are transformed with edge overlap
(v_a, v_b) is transformed randomly

Algorithm Flow

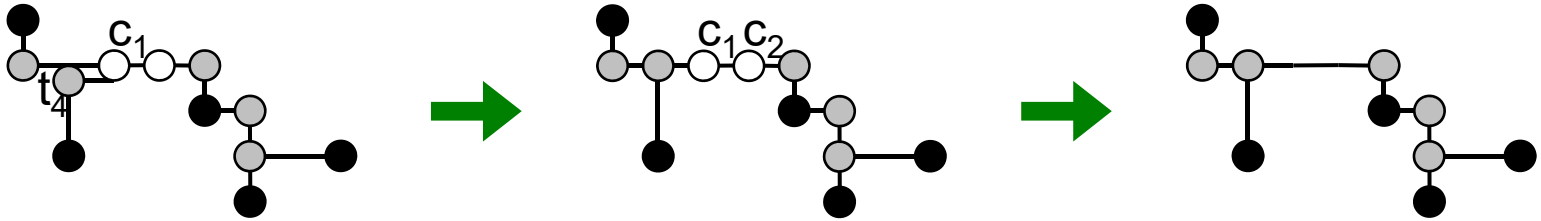


OARSMT Construction (1/2)

- Overlapping edge removal

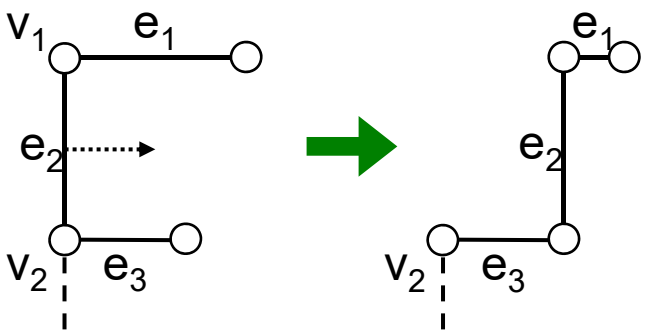


- Redundant vertex removal
  - A redundant-vertex is a non-pin-vertex with the degree of 2, and the two edges connecting to it are parallel.

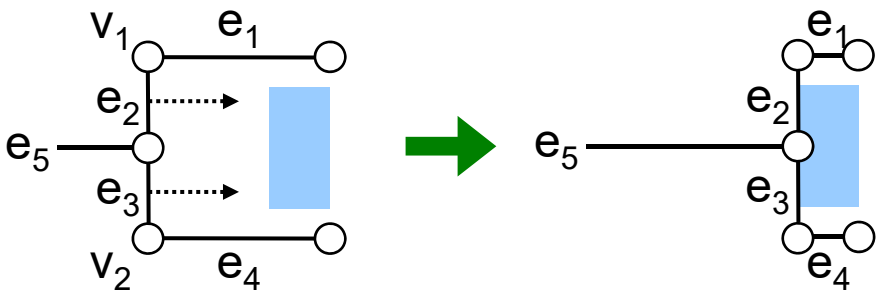


OARSMT Construction (2/2)

- U-shaped pattern refinement
  - A vertex satisfies the “U-shaped pattern refinement rules” if it is not a pin-vertex, and its degree is 2.
  - Two cases for the U-shaped pattern refinement:



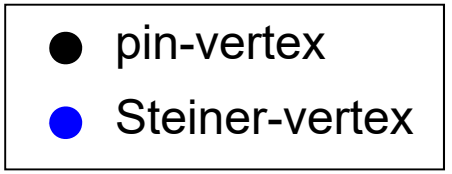
One of the vertices v_1 and v_2 must satisfy the refinement rule.

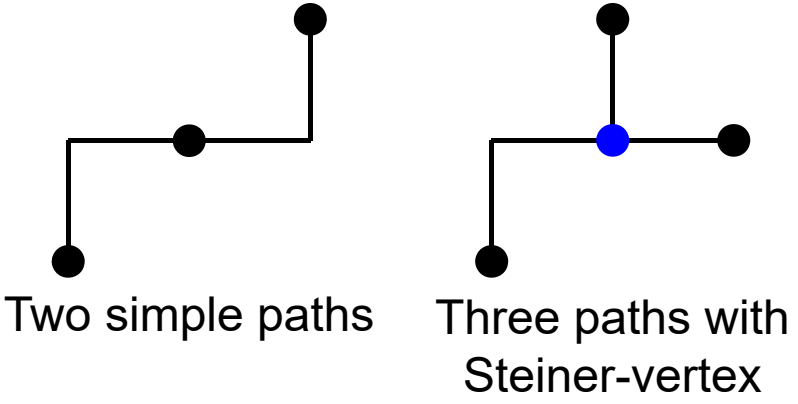


Both vertices v_1 and v_2 must satisfy the refinement rules.

Properties of Our OARSMT

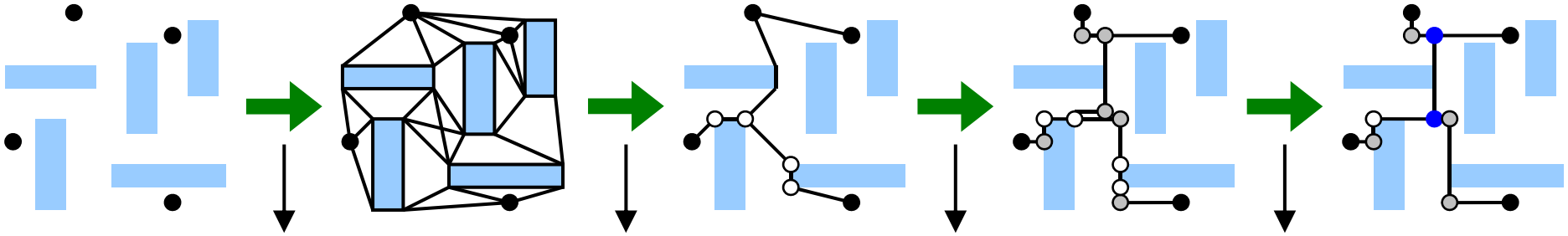
- Our OARSMT is an optimal solution for:
  - Any 2-pin net
  - Any 3-pin net without obstacles
  - Any net whose topology of an optimal solution contains only simple paths





- They are not guaranteed by any previous work.
- They give the sufficient but not necessary conditions for an optimal solution.
  - More optimal solutions may still be generated in other cases.

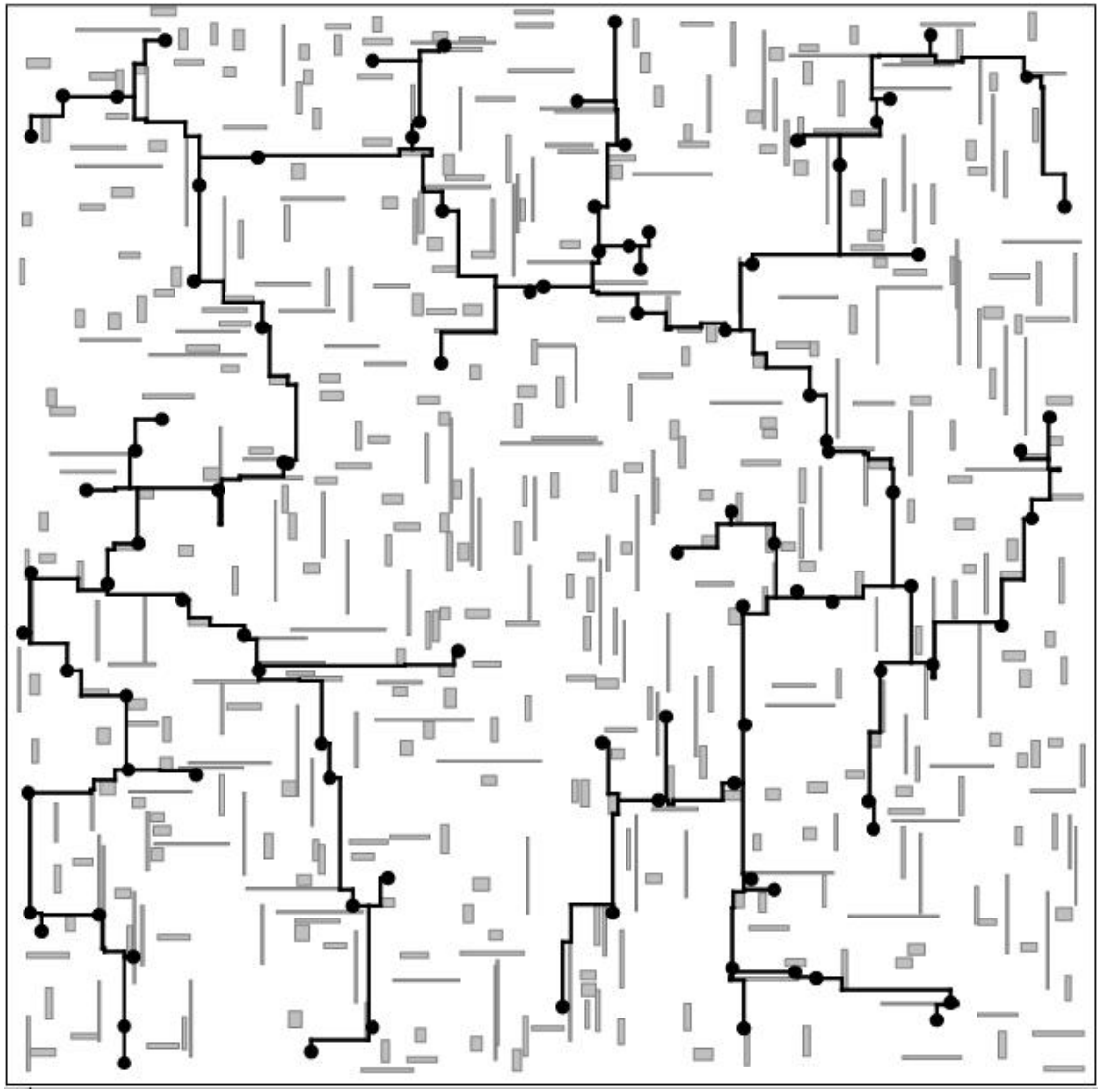
Complexity



Spanning Graph Construction	Spanning Tree Construction	Rectilinear Spanning Tree Construction	OARSMT Construction
Worst Case			
$O(n^2 \lg n)$	$O(n^3)$	$O(n \lg n)$	$O(n^2)$
Practical Applications (# of edges in spanning graph is O(n).)			
$O(n \lg n)$	$O(n^2 \lg n)$	$O(n \lg n)$	$O(n^2)$

- Overall time complexity
  - $O(n^3)$ in the worst case
  - $O(n^2 \lg n)$ for practical applications
  - n: # of pin-vertices and corner-vertices

Experimental Results – Routing Result of rt3



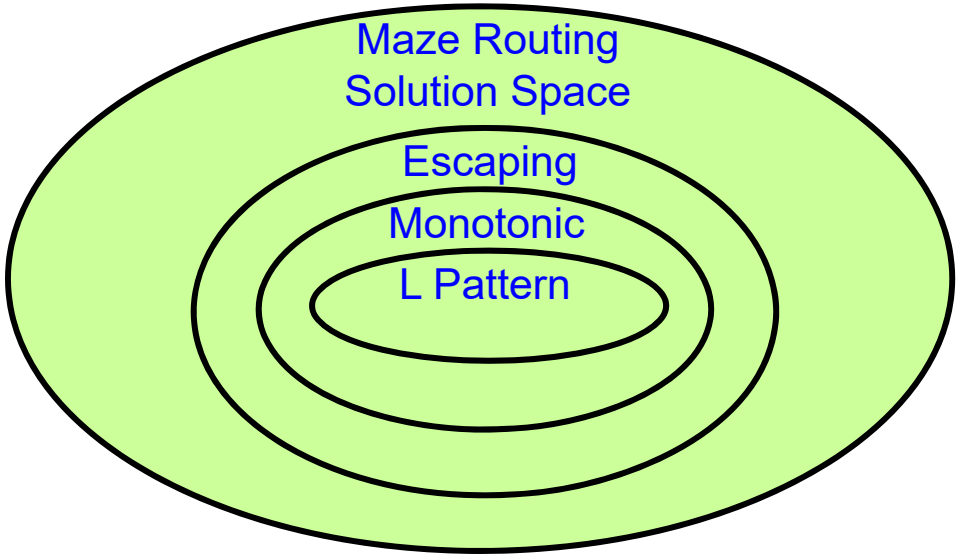
Appendix D:

Modern Global Routing & NCTU-GR

Wen-Hao Liu, Yih-Lang Li, et al.

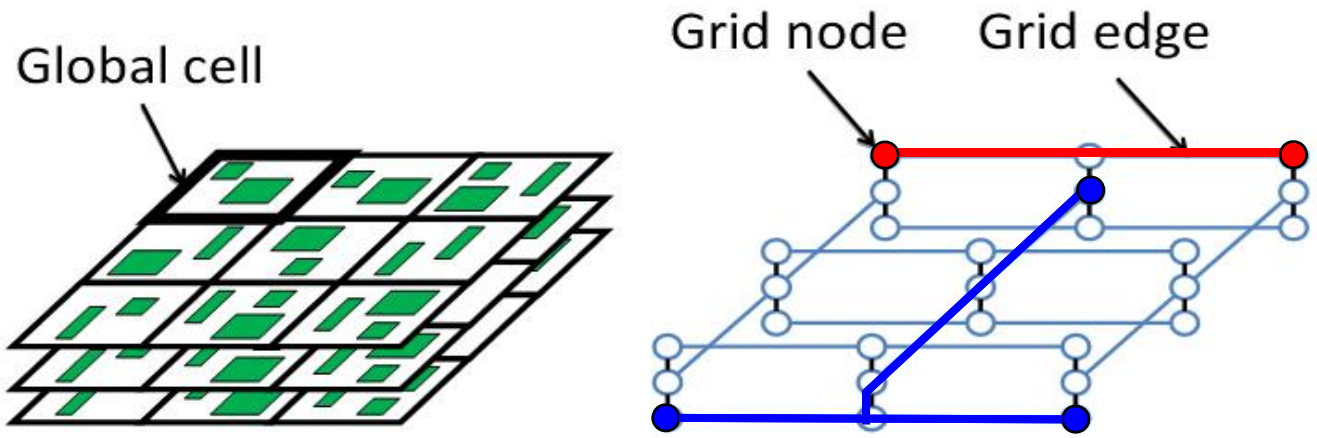
DAC'10, TVLSI-12 (NCTU-GR), TCAD-13 (NCTU-GR 2.0)

(Slides provided by Dr. Wen-Hao Liu, with revisions)



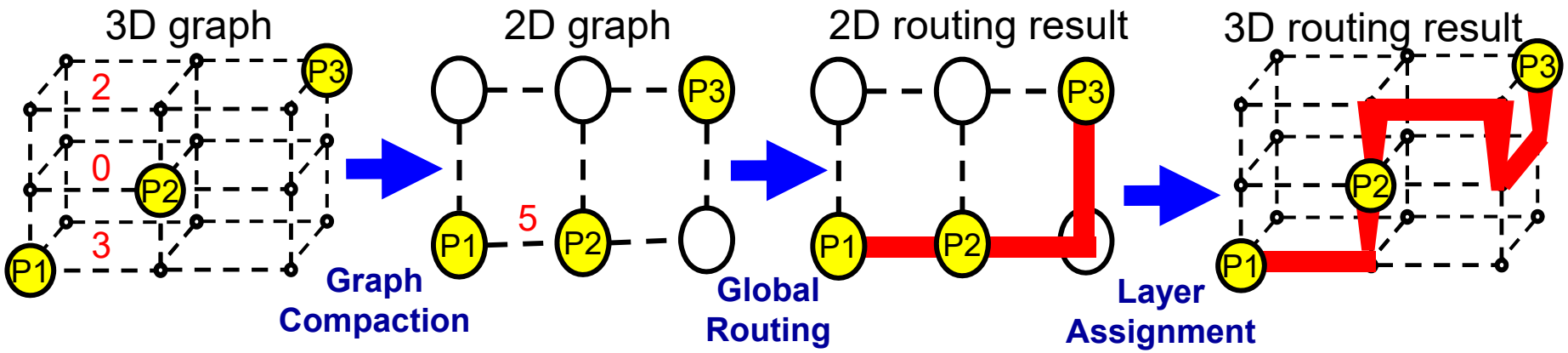
Global Routing Problem

- Given a grid graph and a set of nets (each net consists of a set of pins).
- The goal of global routing is to identify the global routing paths to connect each net.
- The optimization order is to minimize overflows, total wirelength (wire edges and vias), and finally runtime.

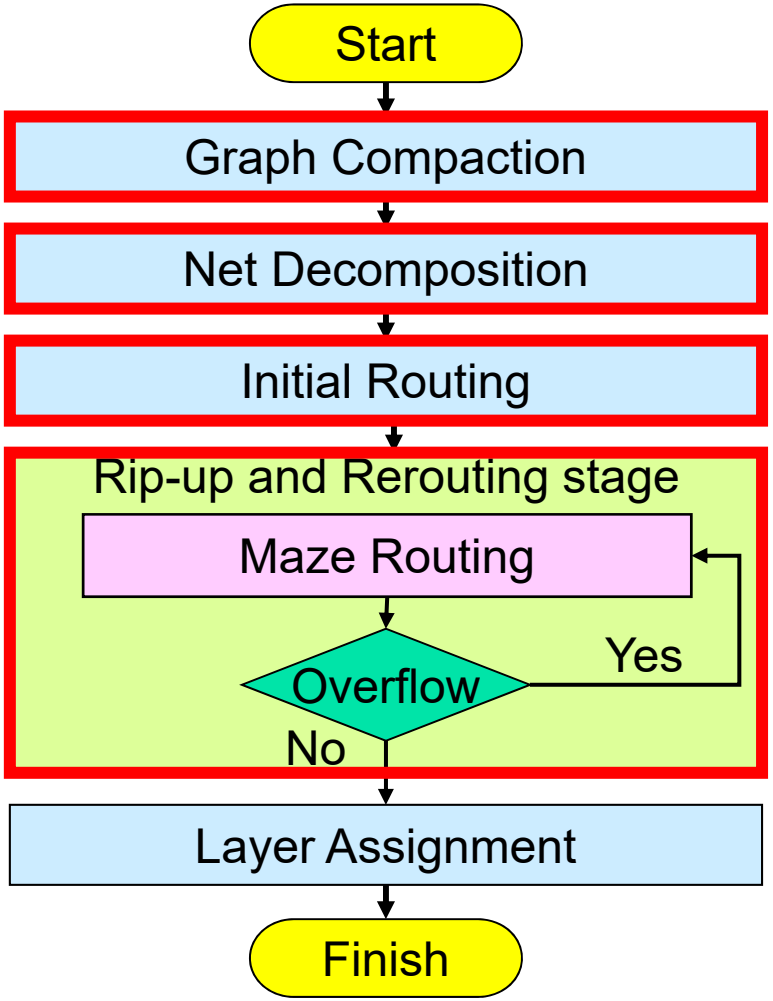


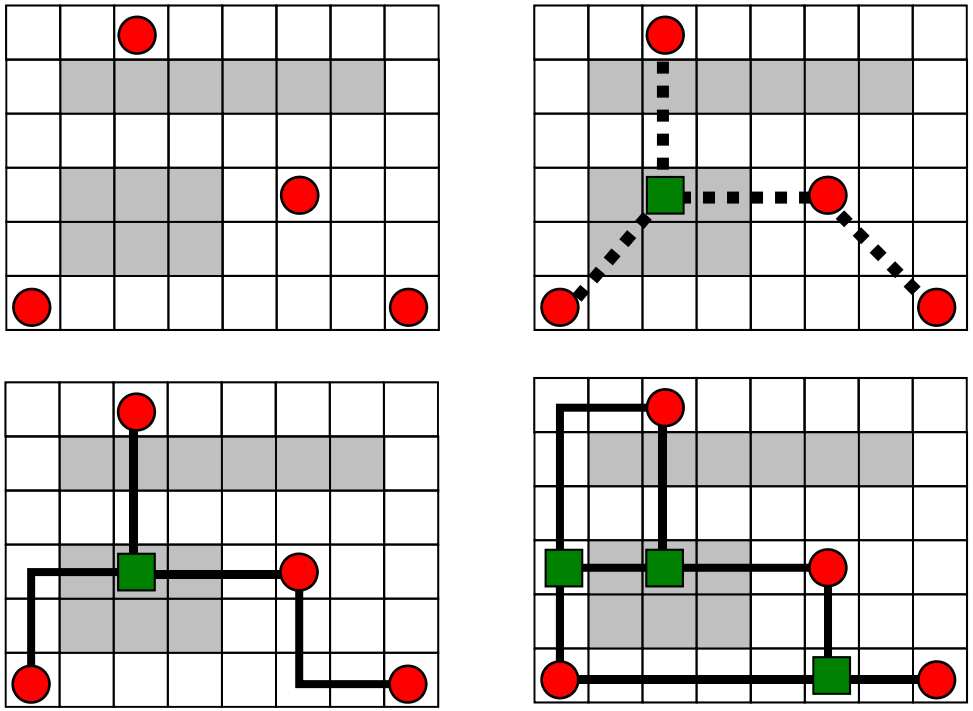
Global Routing Strategies

- Two strategies are generally adopted to resolve the global routing problem.
  - Directly perform global routing on a 3D graph.
  - 2D global routing with layer assignment.
- Due to the runtime concern, most global routers adopt the second strategy.



Mainstream Global Routing Flow





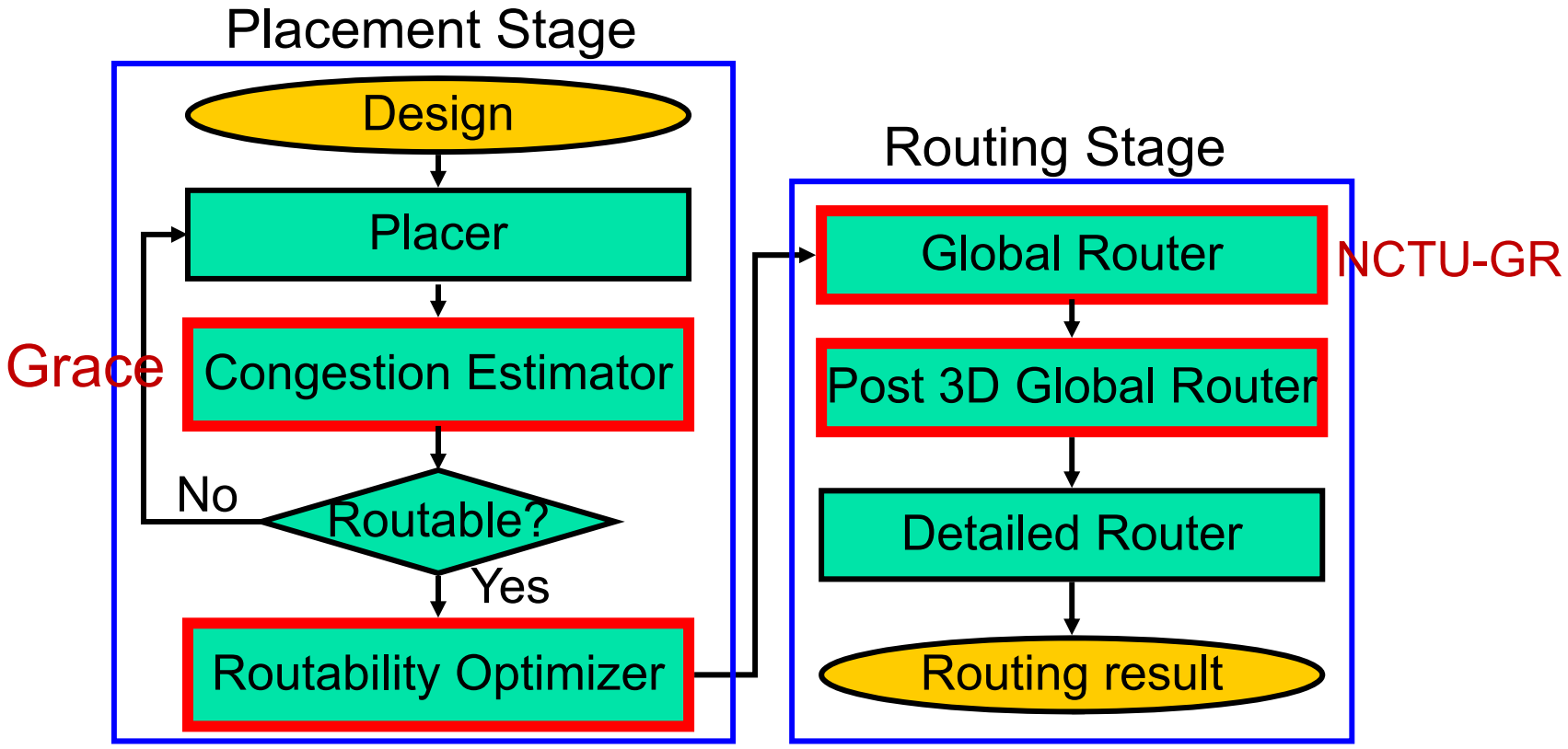
Global Routing Studies

- The studies of existing global routing researches
  - Net decomposition [6,7,8,14]
  - Routing nets ordering [2,7,10,13]
  - Rip-up and rerouting scheme [2,8,11,13,16]
  - Routing cost formulation [1,2,3,6,7,8,10,13,14,15]
  - Routing algorithms [4,6,9,10,13,15,19]
  - Layer assignment approaches [6,7,12,13]
  - Multi-threaded routing [14,16,17,18]

NTHU-Route [1,2]	FastRoute [3-6]	FGR [7,8]	MGR [9]	NTUgr [10]
Box-Router [11,12]	NCTU-GR [13,14]	Archer [15]	GRIP [16,17]	HybridGR [18]
Maize-Router [19]				

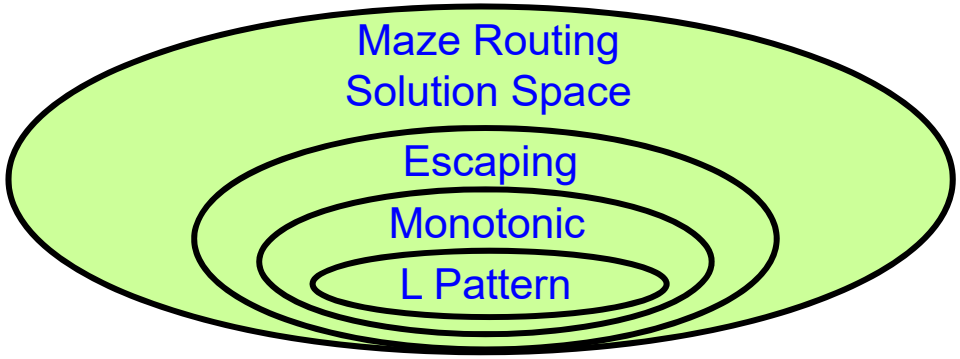
Routability-Driven Placement & Routing

- Add a congestion estimator during placement for routability consideration



Routing Effectiveness vs. Runtime

- Effectiveness and runtime of each routing stage for an industry design
  - Grid size : 444 x 518
  - Net number : 1.8M

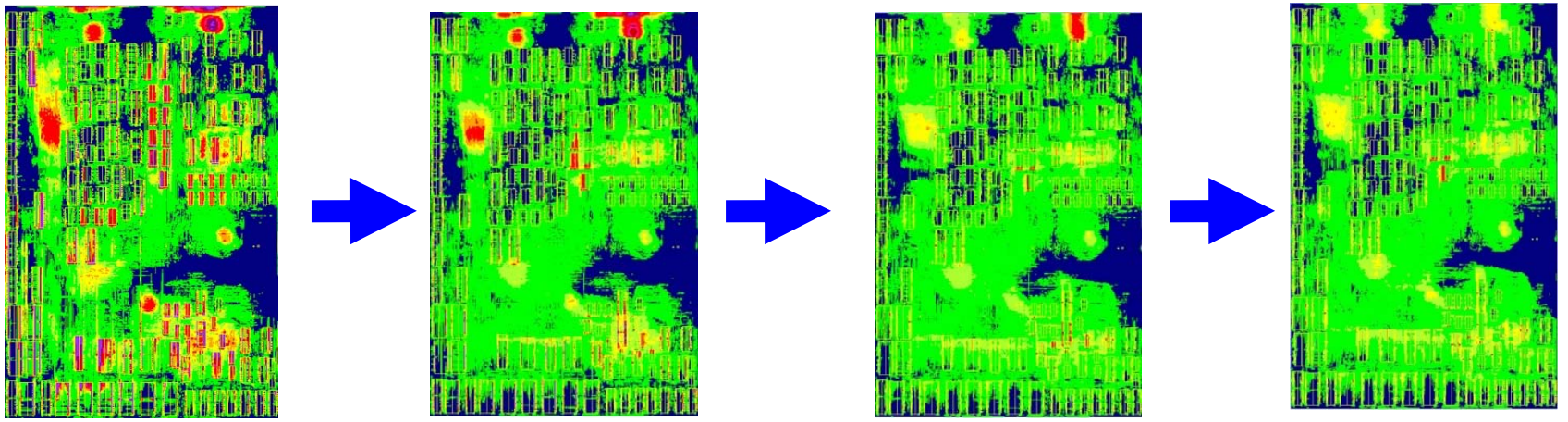


L-shaped Pattern Routing
Route 100 % nets
11 seconds

Monotonic Routing
Reroute 11% nets
14 seconds

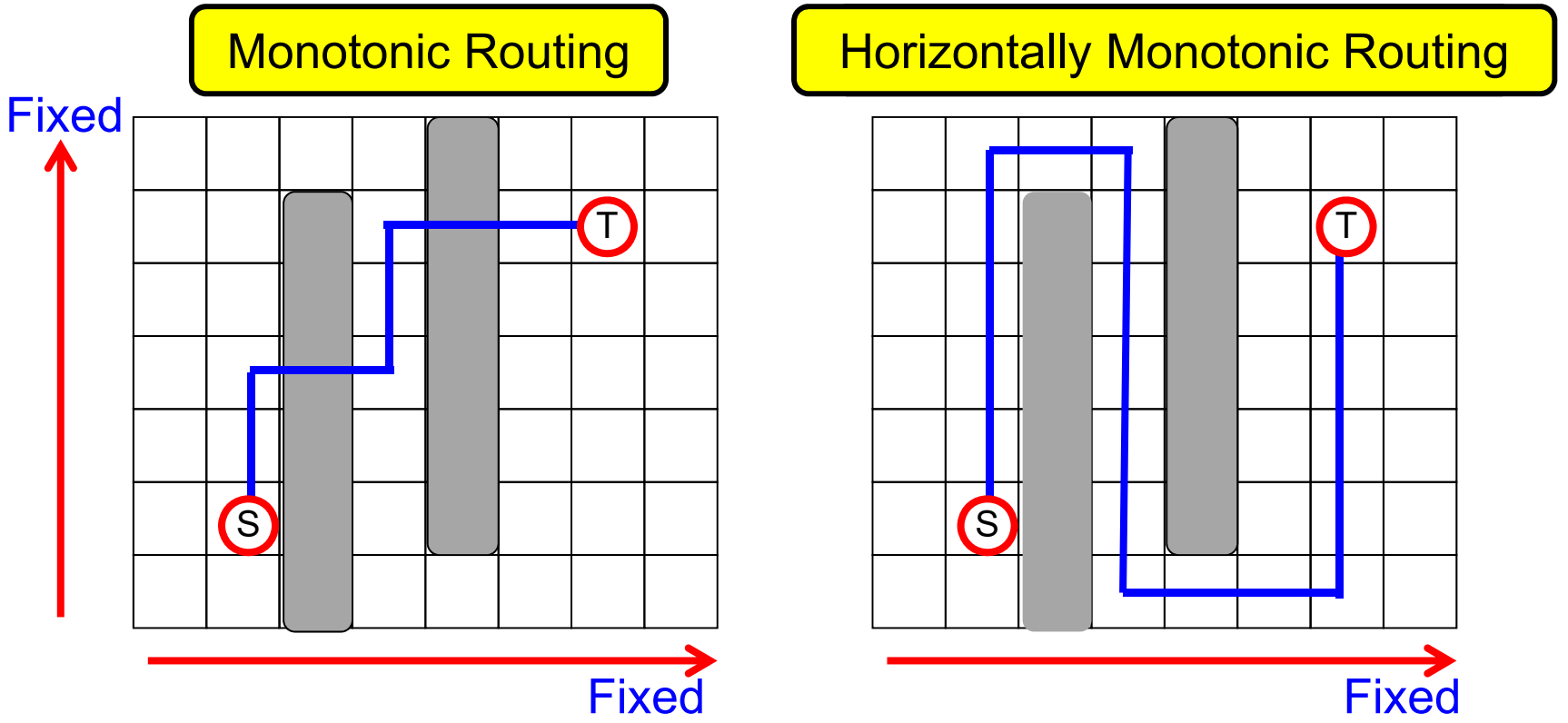
Escaping Routing
Reroute 3% nets
23 seconds

Maze Routing
Reroute 1% nets
120 seconds



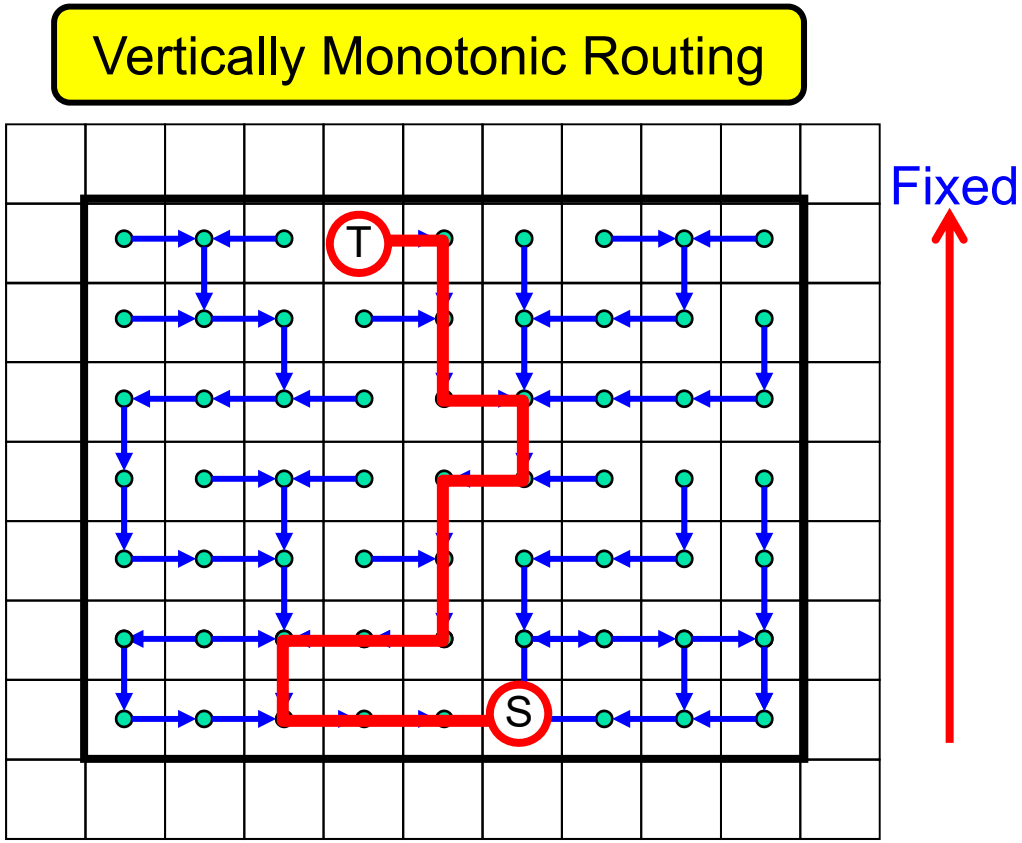
Unilateral Monotonic Routing

- **Unilateral monotonic routing** can seek a detoured path and running in the time complexity linear to its searching region



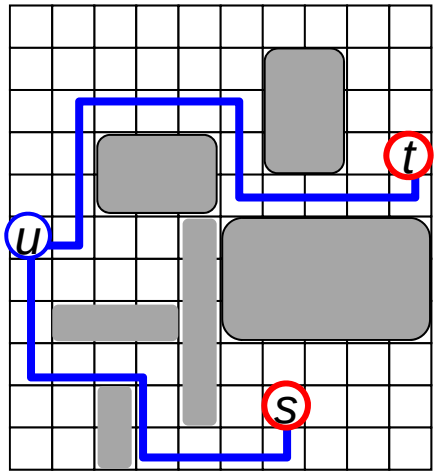
Unilateral Monotonic Routing

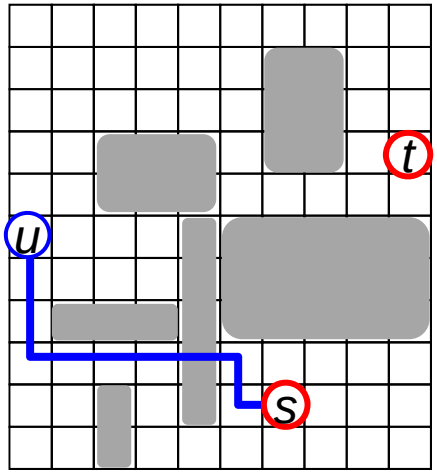
- Unilateral monotonic routing is based on dynamic programming.
- Given a bounding box *B*, unilateral monotonic routing identifies a parent for each node within *B* in a bottom-up manner.
- The time complexity of unilateral monotonic routing is $O(|B|)$.



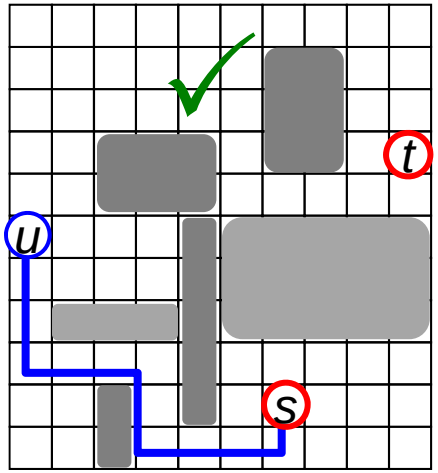
Hybrid Unilateral Monotonic (HUM) Routing

- HUM path consists of two paths $P_{s,u}$ and $P_{t,u}$
  - $P_{s,u}$ is either horizontally or vertically monotonic path from s to internal node u.
  - $P_{t,u}$ is either the horizontally or vertically monotonic path from t to internal node u.

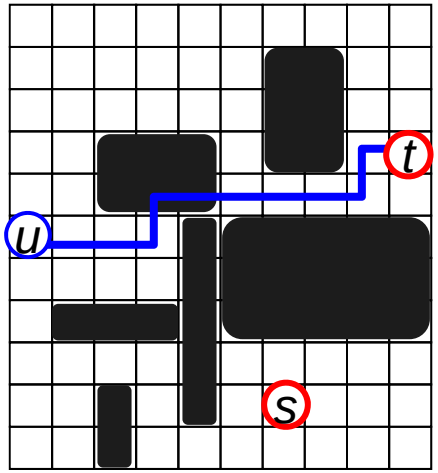




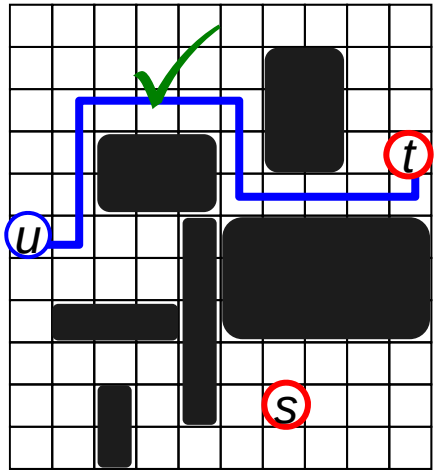
Vertically monotonic path from s to u



Horizontally monotonic path from s to u



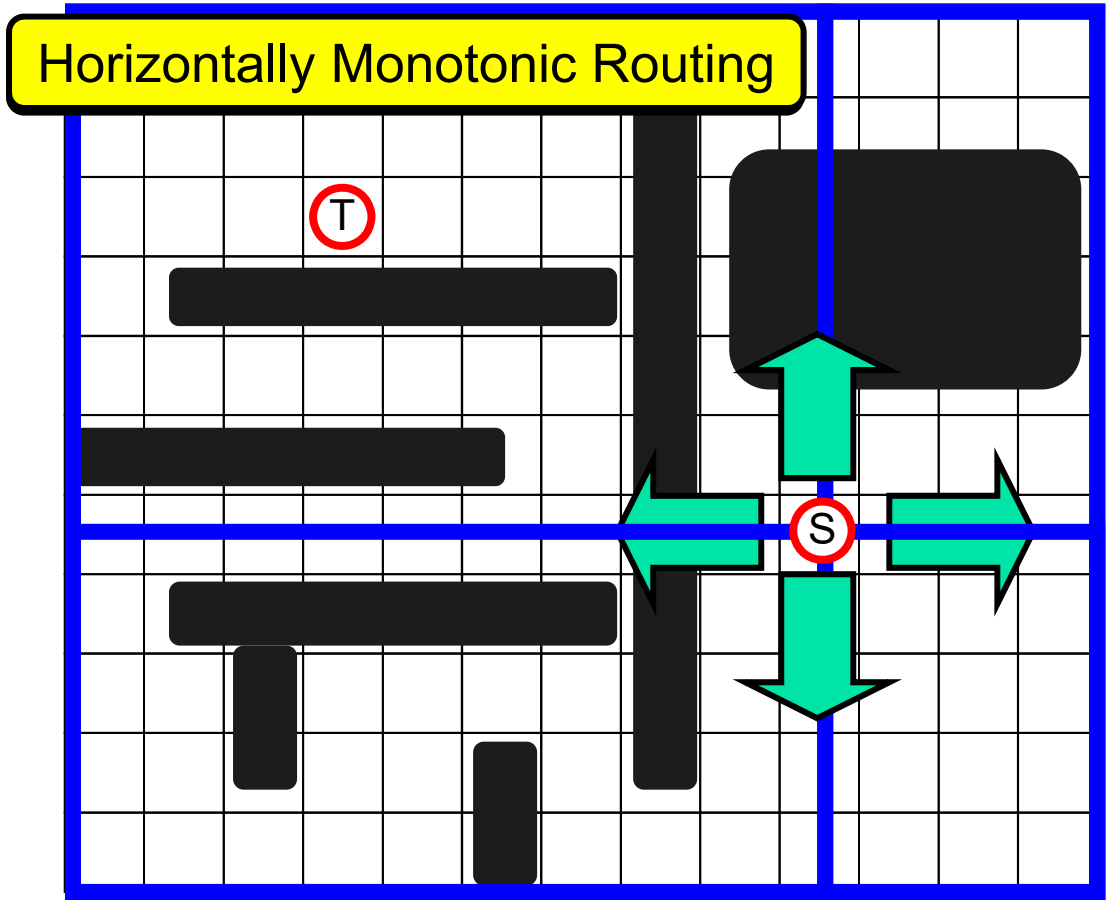
Vertically monotonic path from t to u



Horizontally monotonic path from t to u

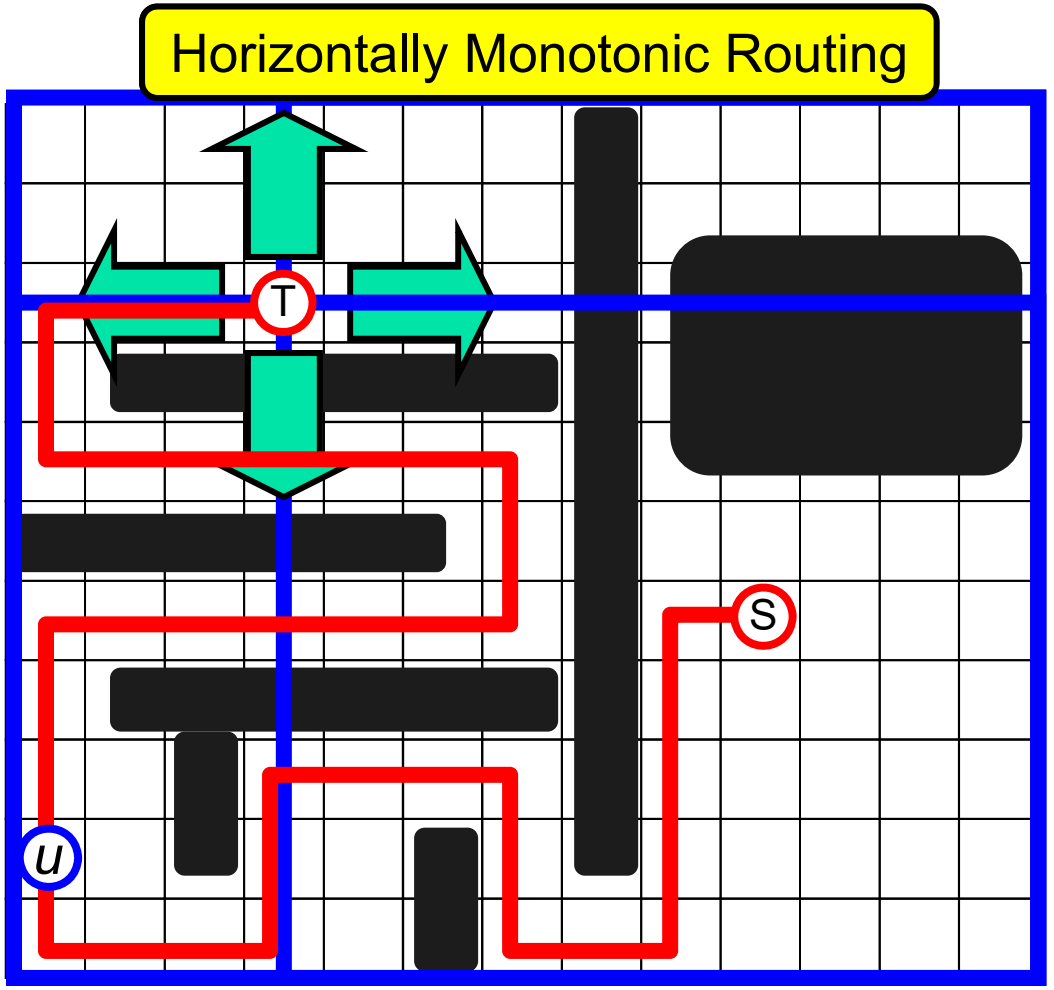
HUM Routing

- Perform four shots of unilateral monotonic routing to try all values of *u* in the bounding box, and then pick the best one to be internal node.
- Perform horizontally monotonic routings from *S* to the left and right boundaries.
- Perform vertically monotonic routings from *S* to the top and bottom boundaries.



HUM Routing

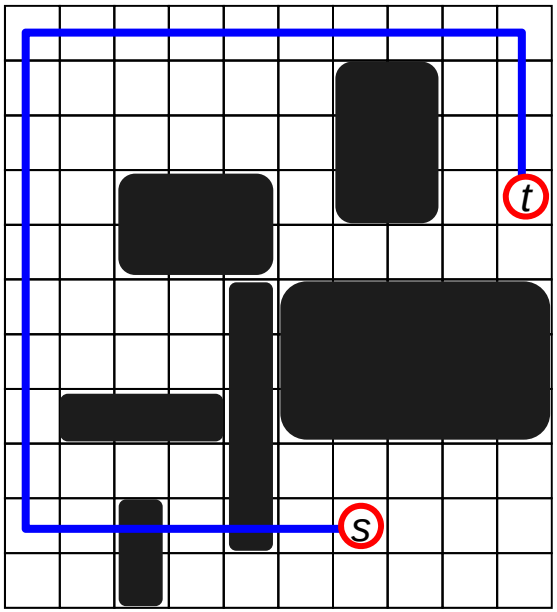
- Perform horizontally monotonic routings from *T* to the left and right boundaries.
- Perform vertically monotonic routings from *T* to the top and bottom boundaries.
- Select the best node *u* to minimize the routing cost of $P_{s,u}$ + $P_{t,u}$



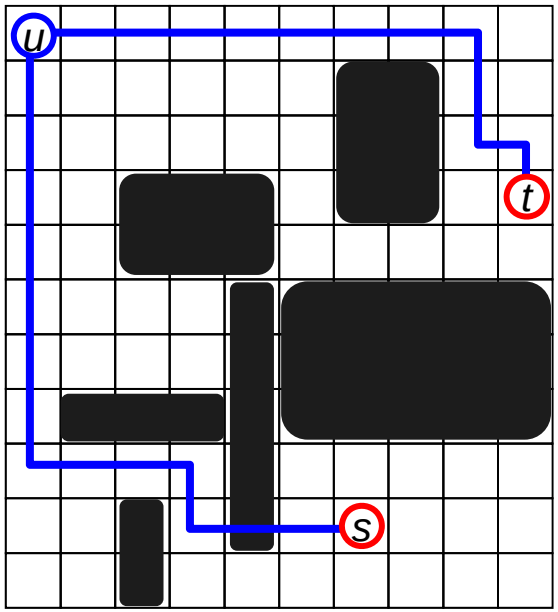
Comparison

- The time complexities of 3-bend routing, escape routing and HUM routing are same, but HUM can get a better routing result.

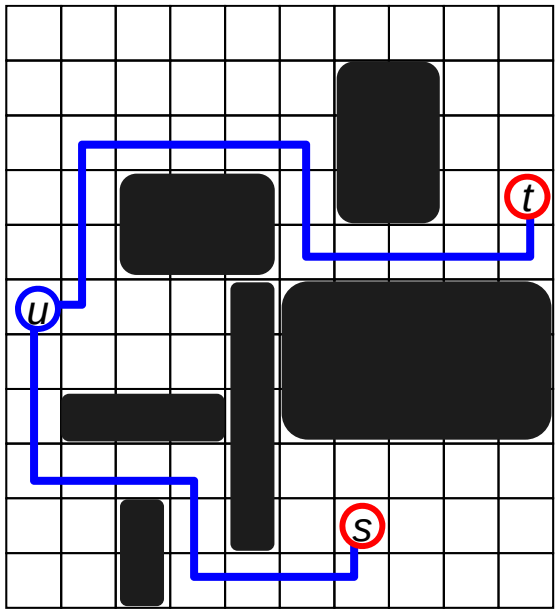
3-bend routing



Escape routing

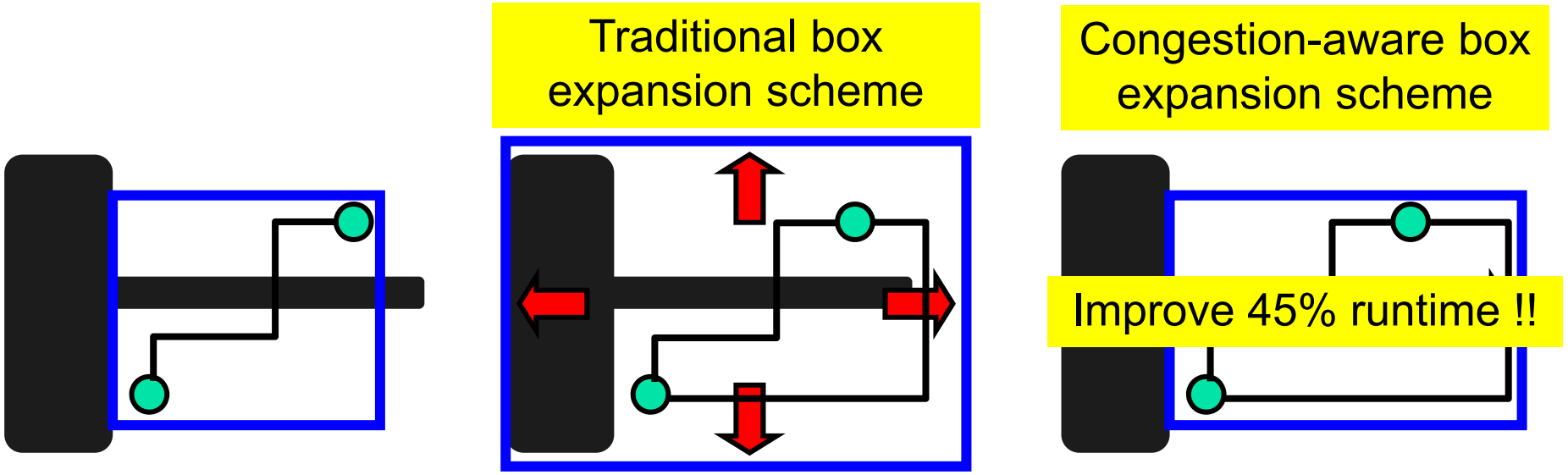


HUM routing

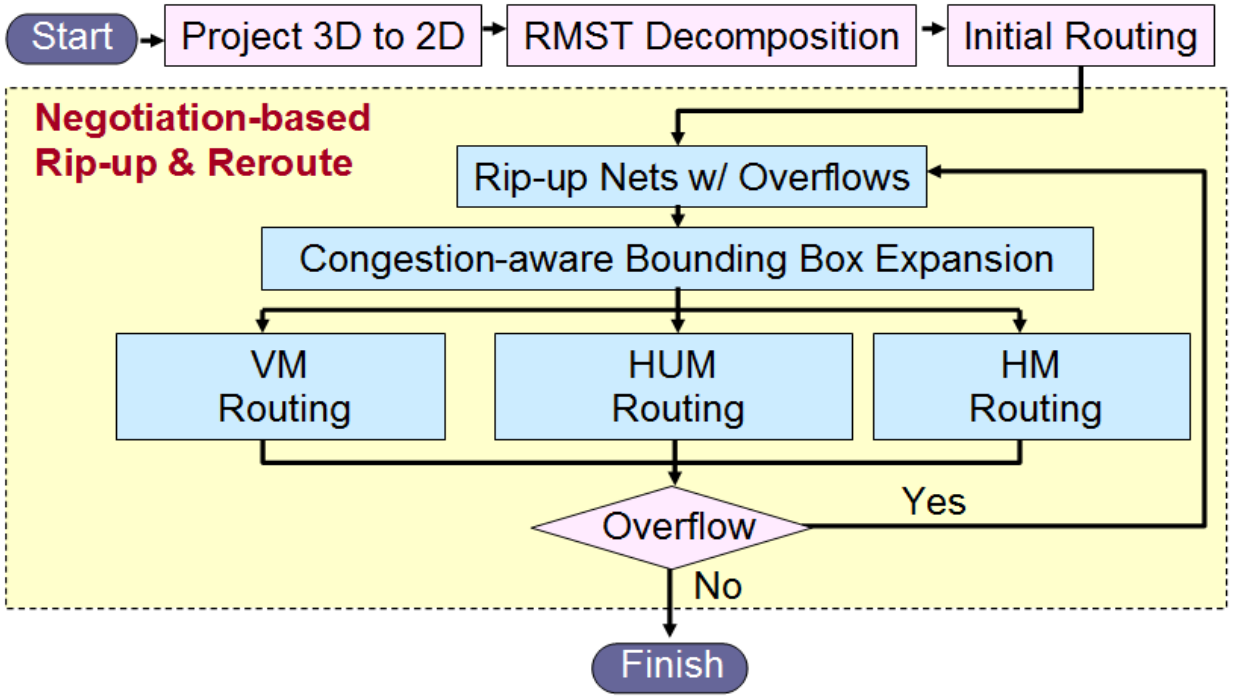


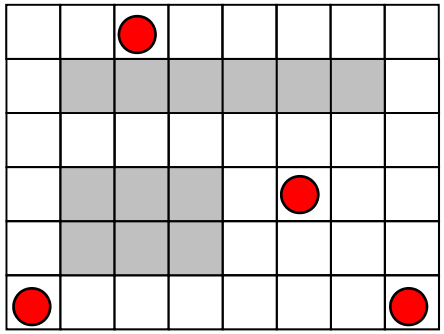
Congestion-aware Bounding Box Expansion

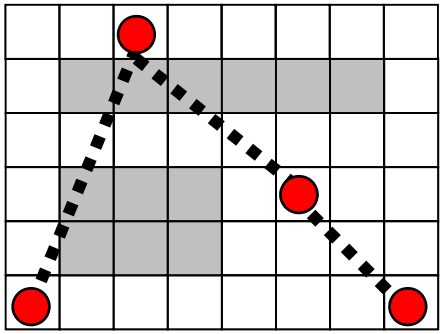
- If a net passes through a congested region, the bounding box of the net expands and then the net is rerouted.
- Over expanding the bounding box may increase the runtime, so use a congestion-aware bounding box expansion scheme to avoid over expanding.

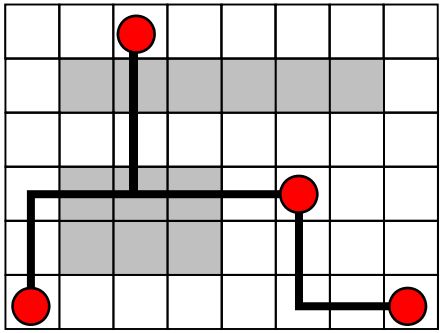


Grace: Fast Congestion Estimator

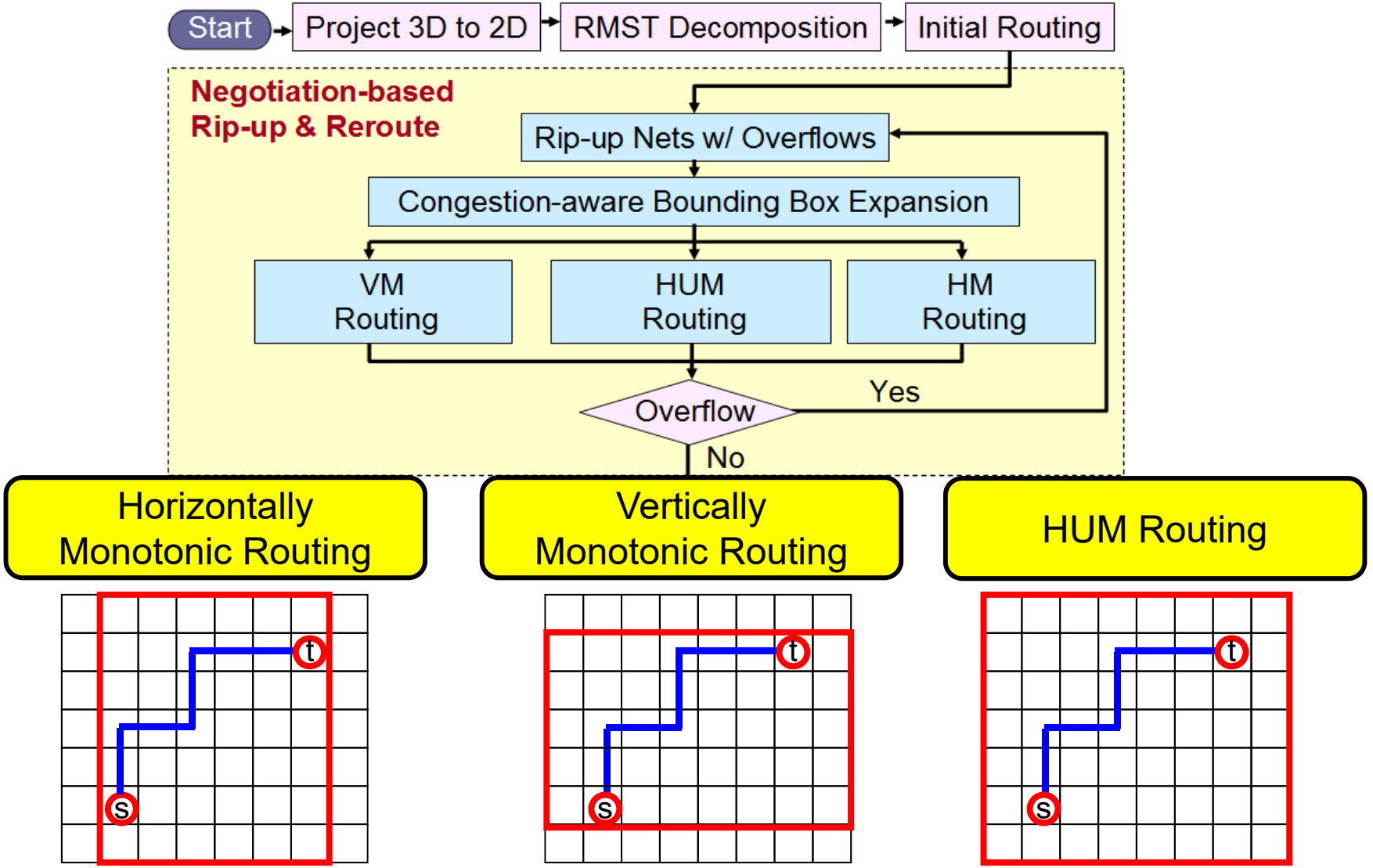




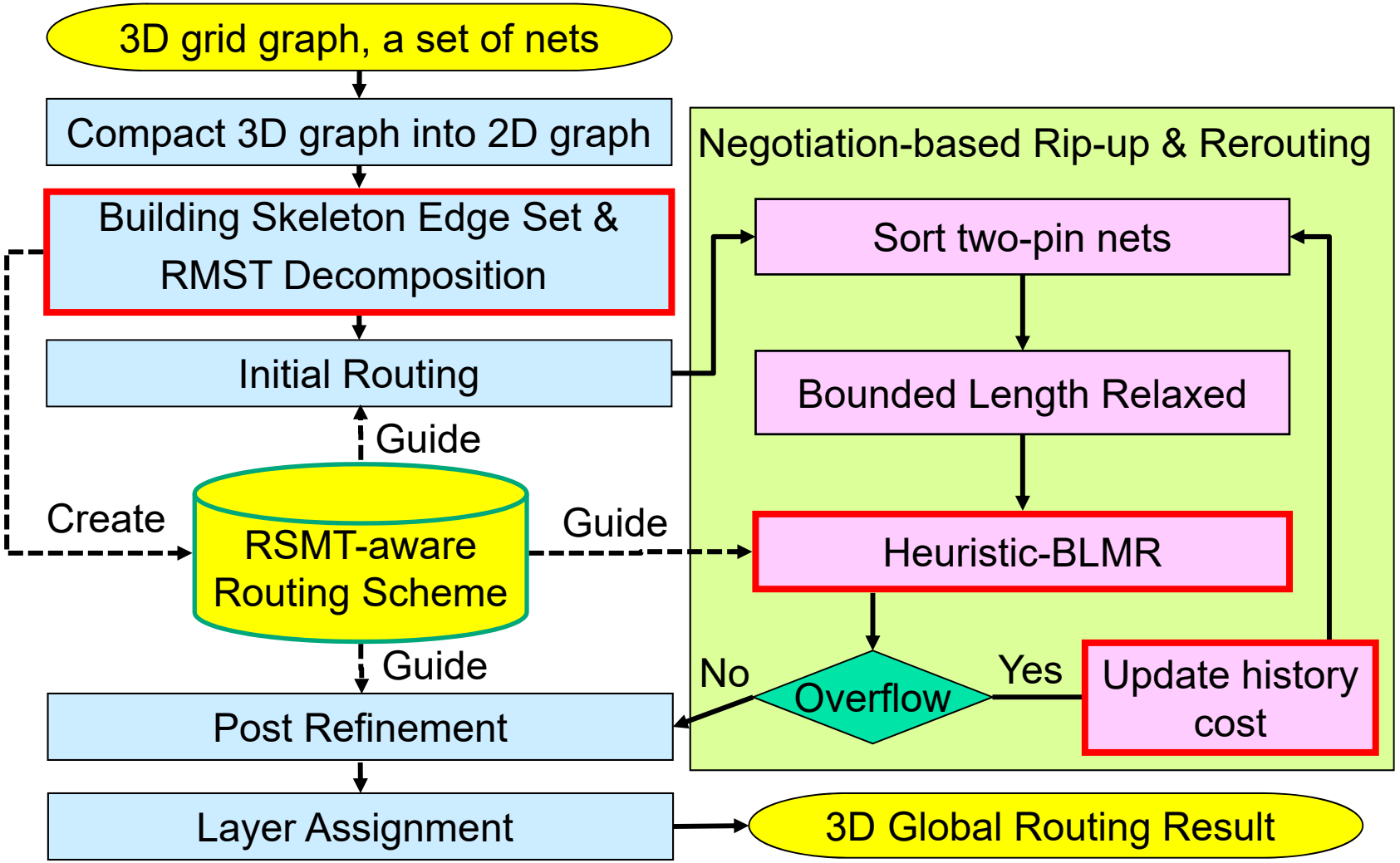




Grace: Fast Congestion Estimator

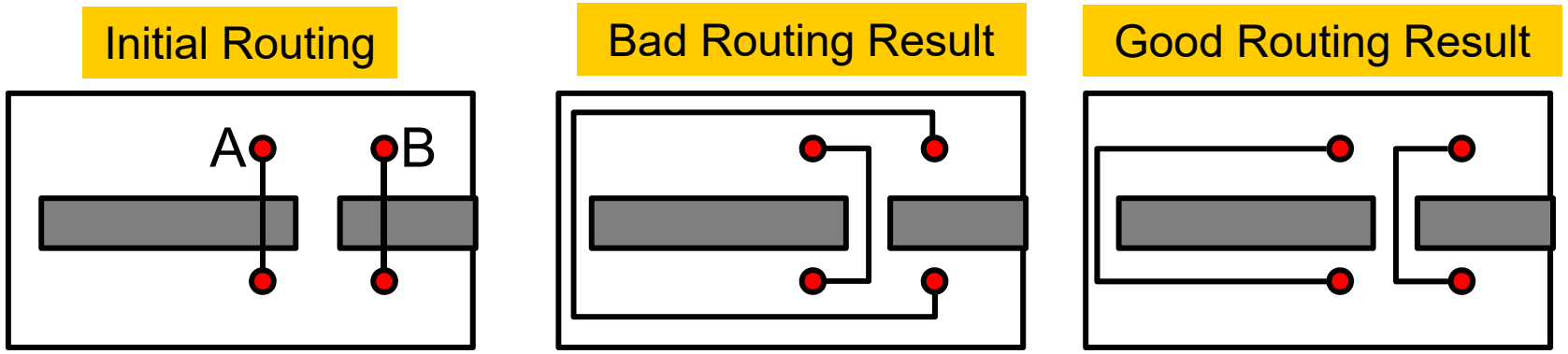


NCTU-GR 2.0 Design Flow



How to Get Good Routing Results

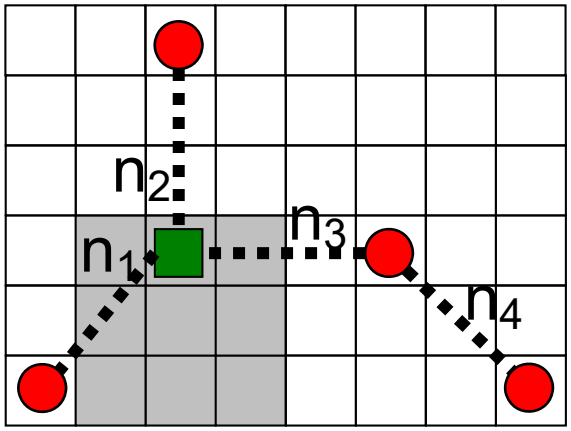
- Carefully utilizing routing resource can get a routing result with better congestion, wirelength and runtime
  - Which grid edge is a critical routing resource
  - Who needs a critical routing resource
  - How to avoid wasting routing resources



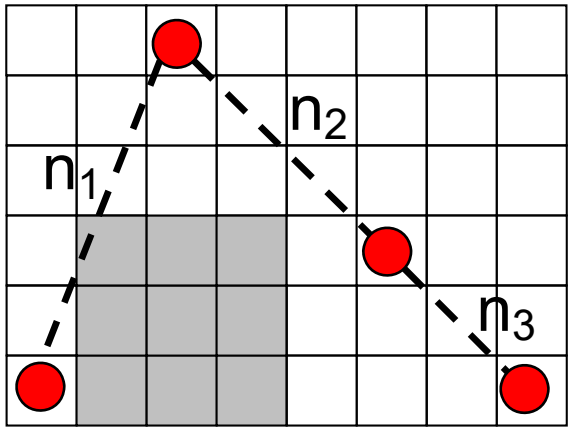
RSMT-aware Routing Scheme

- Rectilinear Steiner minimum tree (RSMT) has shorter wire length, but has less routing flexibility and more complicated data structure
- Rectilinear minimum spanning tree (RMST) offers better flexibility, but may have longer routing wirelength

RSMT Decomposition



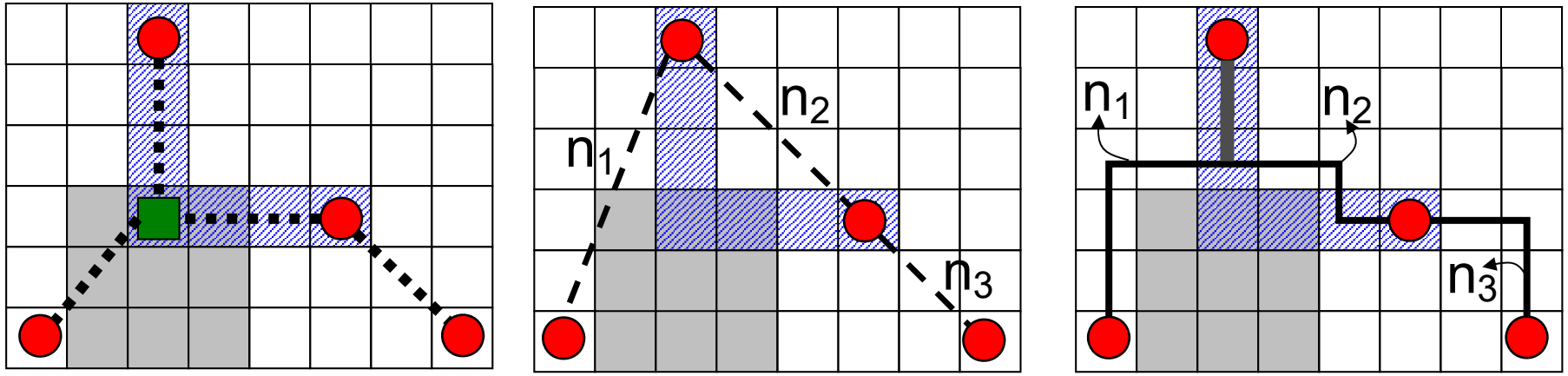
RMST Decomposition



RSMT-aware Routing Scheme

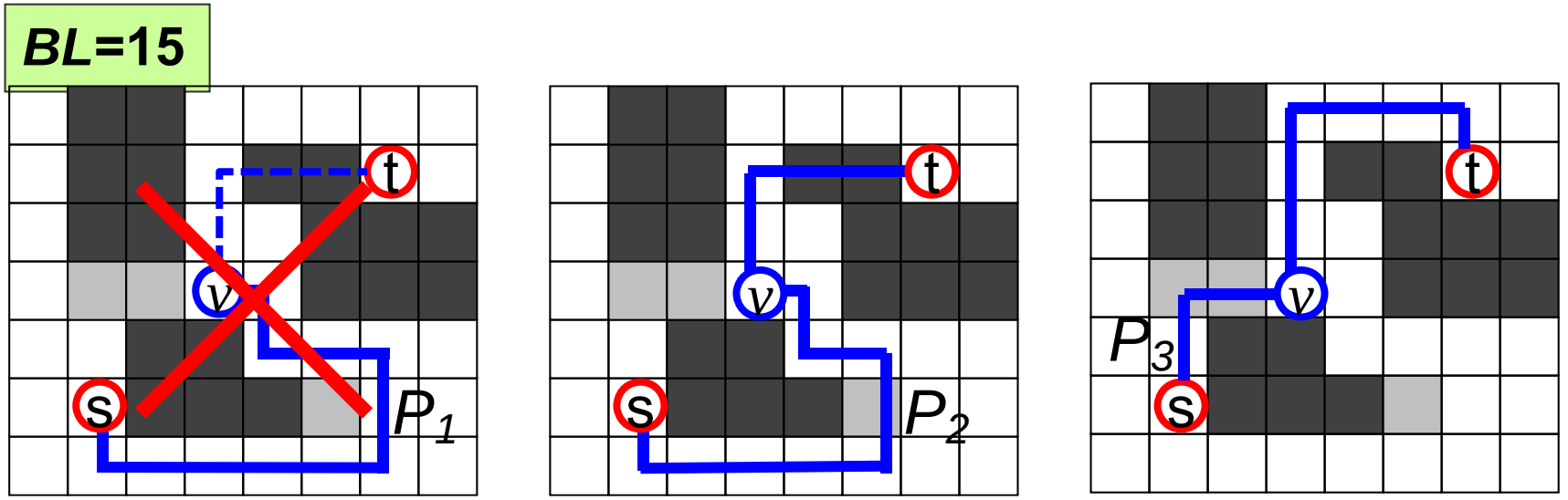
- Decompose each multi-pin net to two-pin subnets by RMST, and then guide routing by an ideal RSMT
  - Strike a good balance between flexibility and wirelength
  - Restore the routing solution to the idea RSMT as the congestions are eliminated
  - Every net knows which grid edges are critical to itself

Encourage the routing paths passing through the **skeleton edges** of RSMT



Bounded-Length Maze Routing (BLMR)

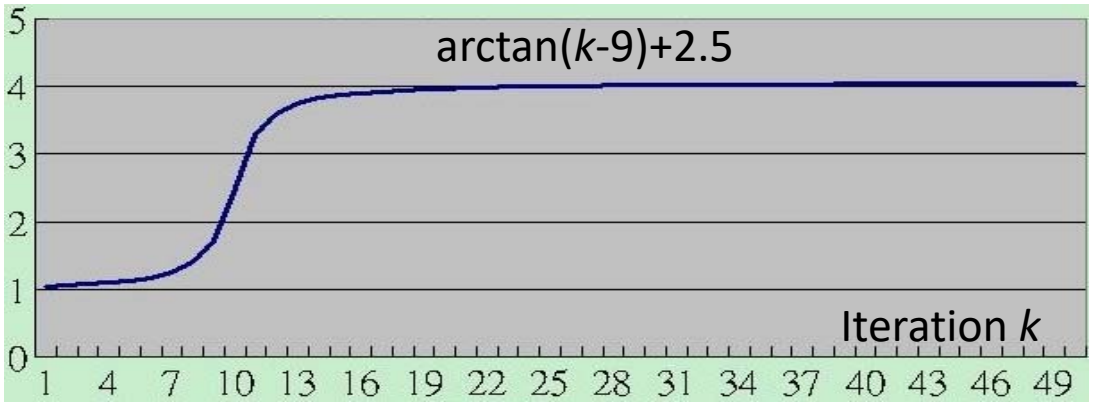
- Heuristic-BLMR can identify a minimal-cost path from *s* to *t* such that wirelengh of the found path cannot exceed *BL*
  - Discard path $P_{s,v}$ if wl($P_{s,v}$)+Manh(v,t) > *BL*
  - Preserve a *better* path if more than one path from *s* to *v*

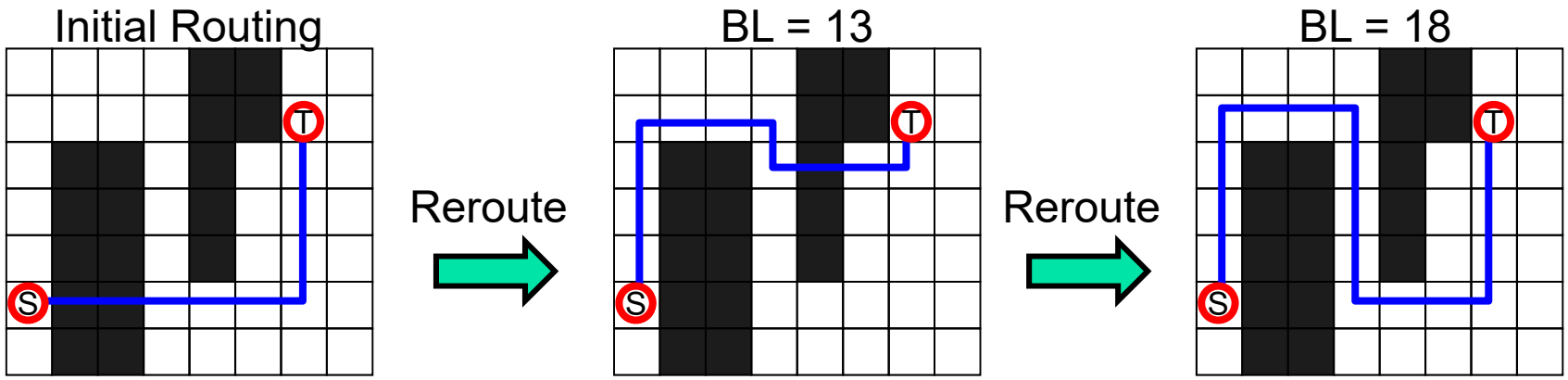


Bounded-Length Relaxation

- The ***BL*** of net *n* at iteration *k* is formulated as follows:

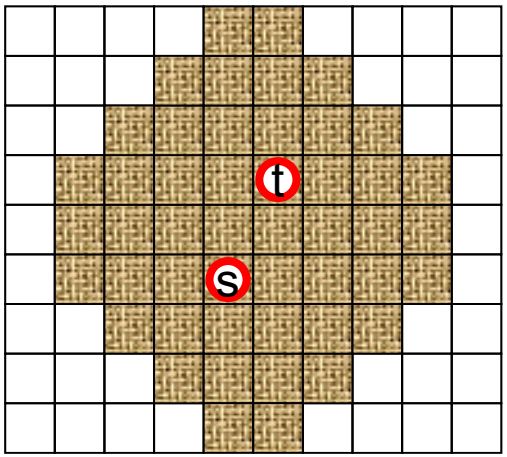
$$BL_n^k = Manh(s_n, t_n) \times (\arctan(k - \alpha) + \beta)$$

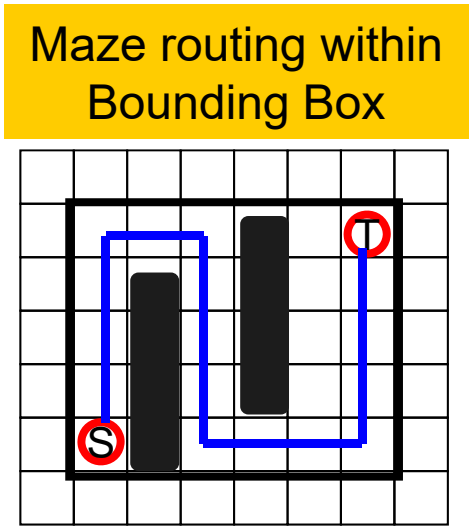


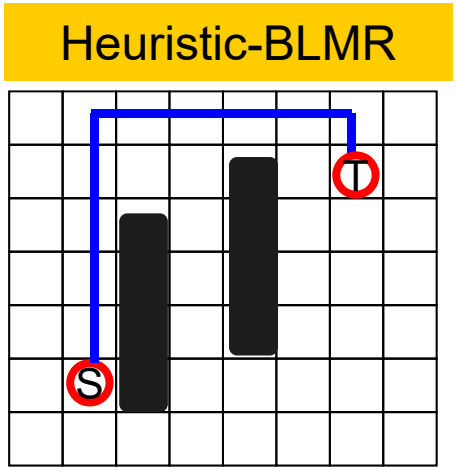


Advantages of Heuristic-BLMR

- Specifically control the routing wirelength
- Restrict the searching region for accelerating routing
- Has good routing resource utilization by avoiding redundant detours.







Dynamically Adjusted History Cost

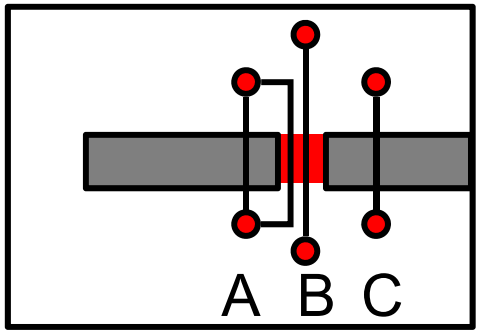
- A grid edge with high history cost implies that its routing resource is critical
- The proposed history cost can more precisely capture the criticality of grid edges

$$h_e^{k+1} = \begin{cases} h_e^k + h_{inc} & \text{if } e \text{ is overflowed} \\ h_e^k & \text{otherwise} \end{cases}$$

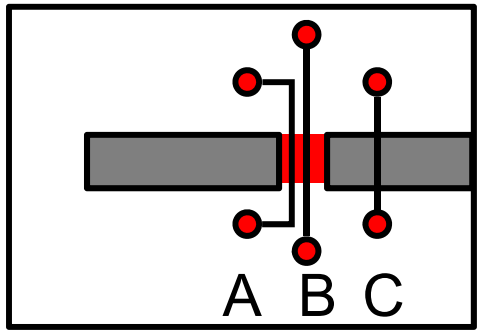
conventional

$$h_e^{k+1} = h_e^k + of_e^k$$

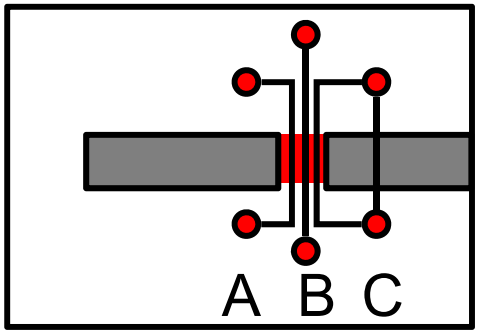
NCTU-GR



$of_e^k = 1$



$of_e^k = 2$



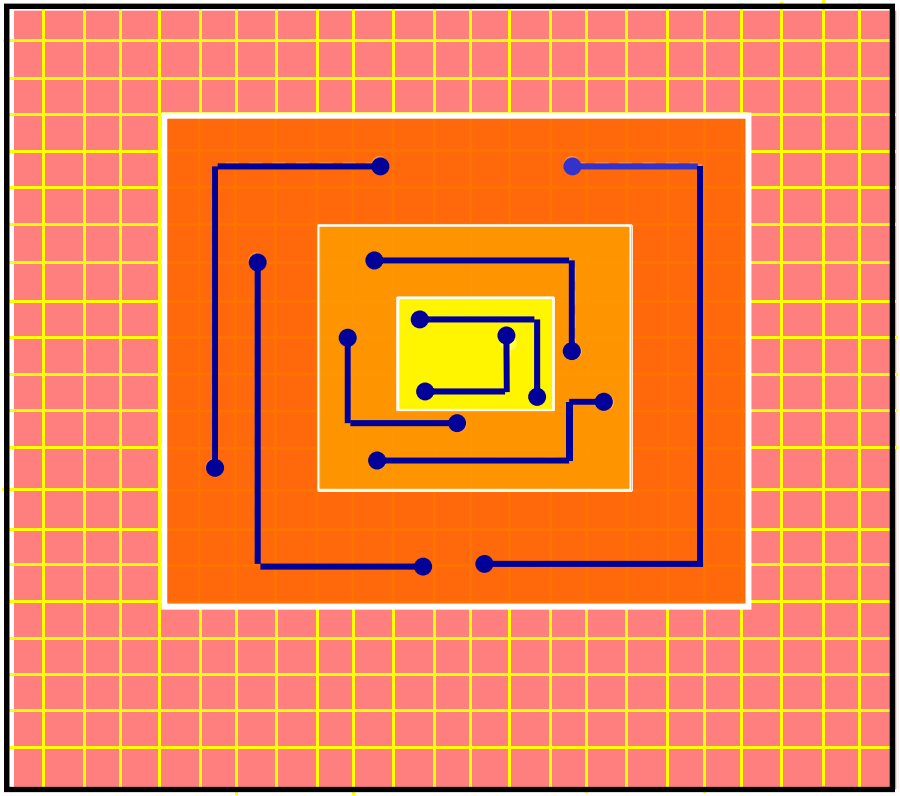
$of_e^k = 3$

Appendix E:

Box Router

Cho and Pan

DAC 2006

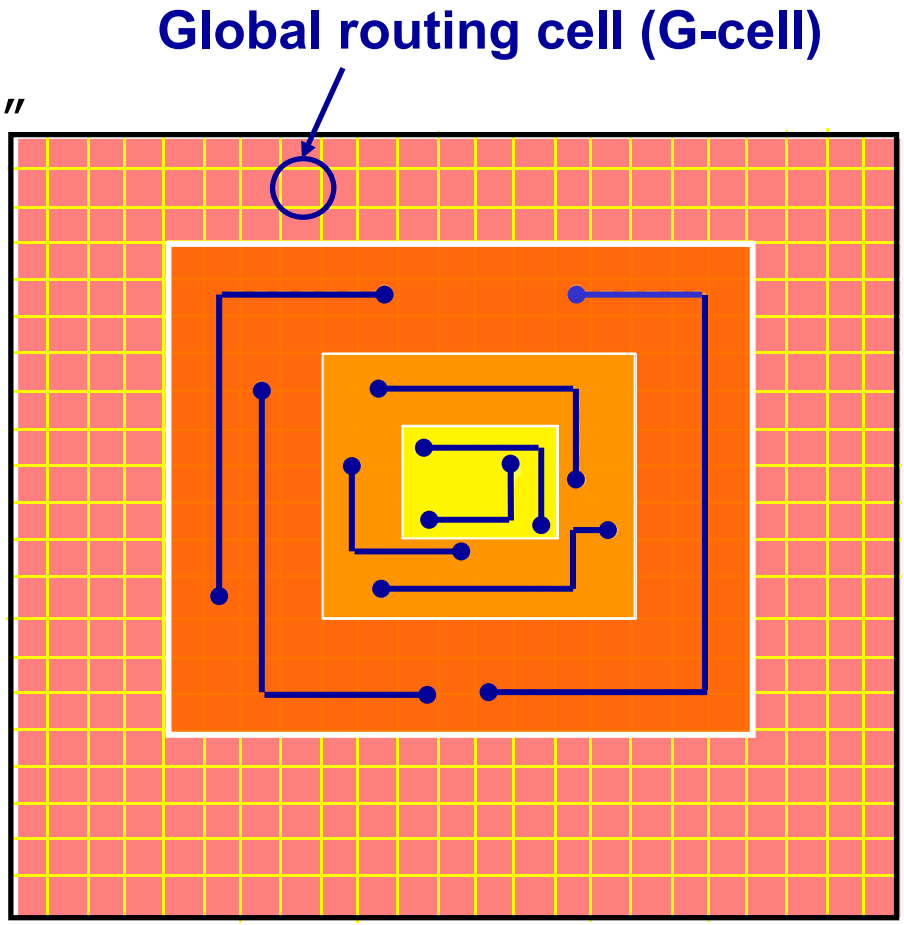


BoxRouter: Progressive Global Routing

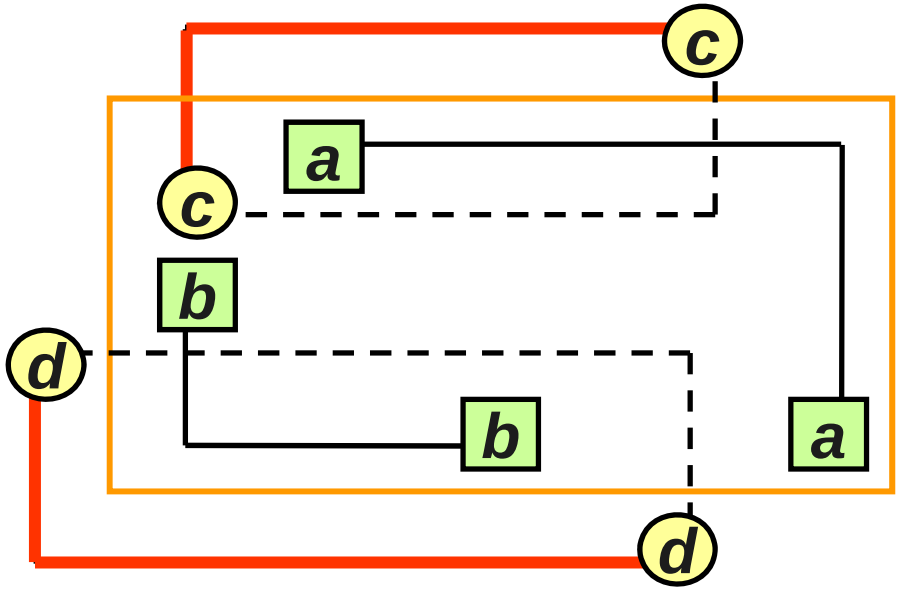
- Cho & Pan, "BoxRouter: A new global router based on box expansion and progressive ILP," DAC-2006.

1. Incremental box expansion
   - From the most congested region (擒賊先擒王)
2. Progressive ILP
   - With adaptive maze routing
3. Post routing
   - Rerouting w/o rip-up

- Try to minimize the total routing overflow

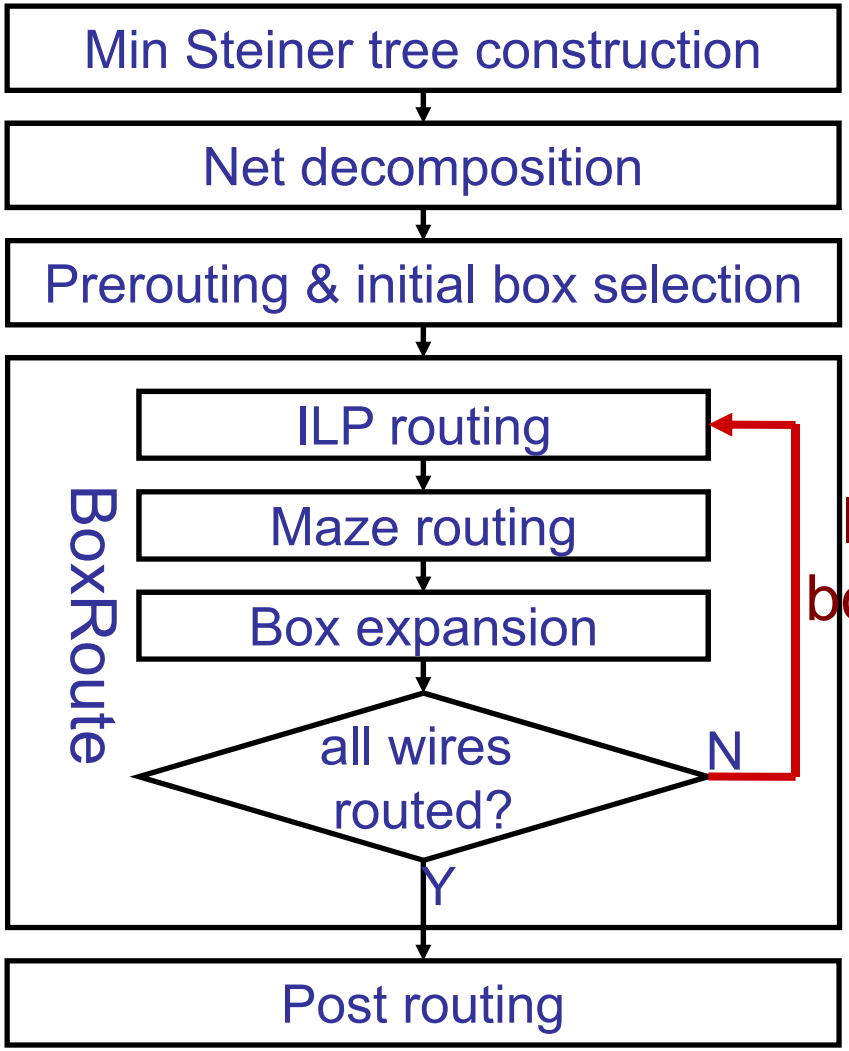


Motivation for Box Expansion

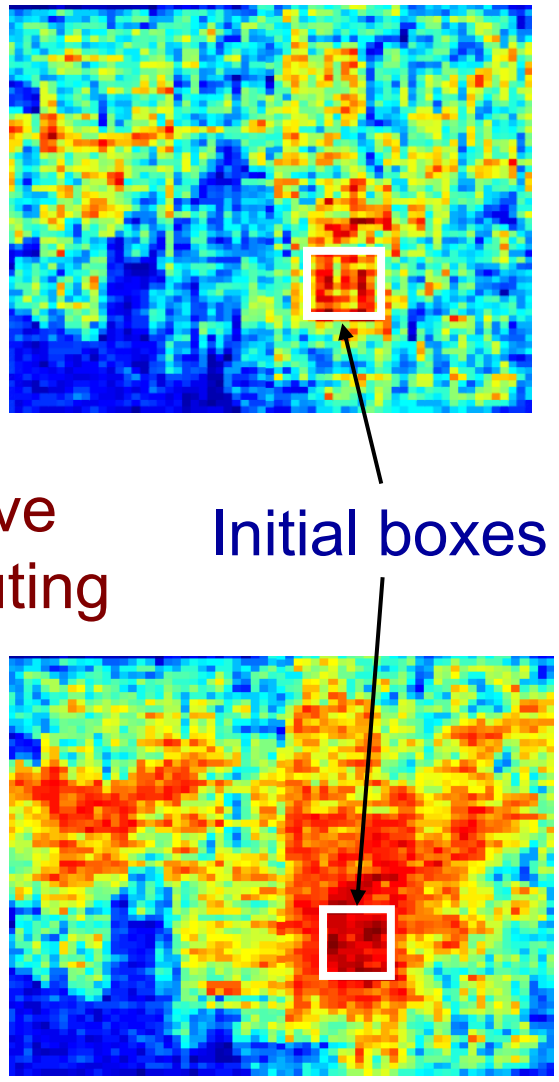


- Different resource management for wires inside/outside of a box
- Box expansion pushes/diffuses congestion outwards progressively

Overall Flow of BoxRouter

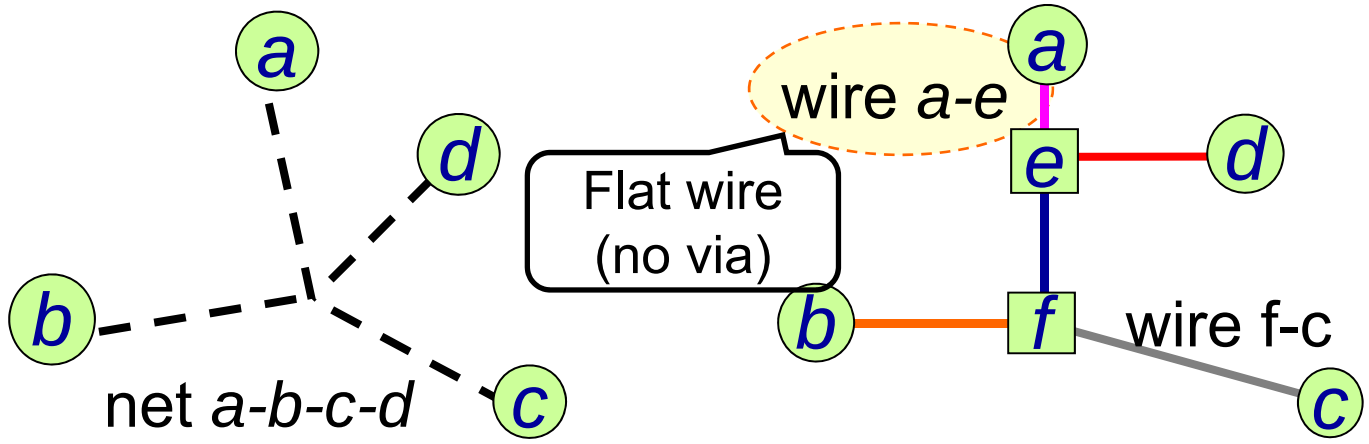


Iterative box routing



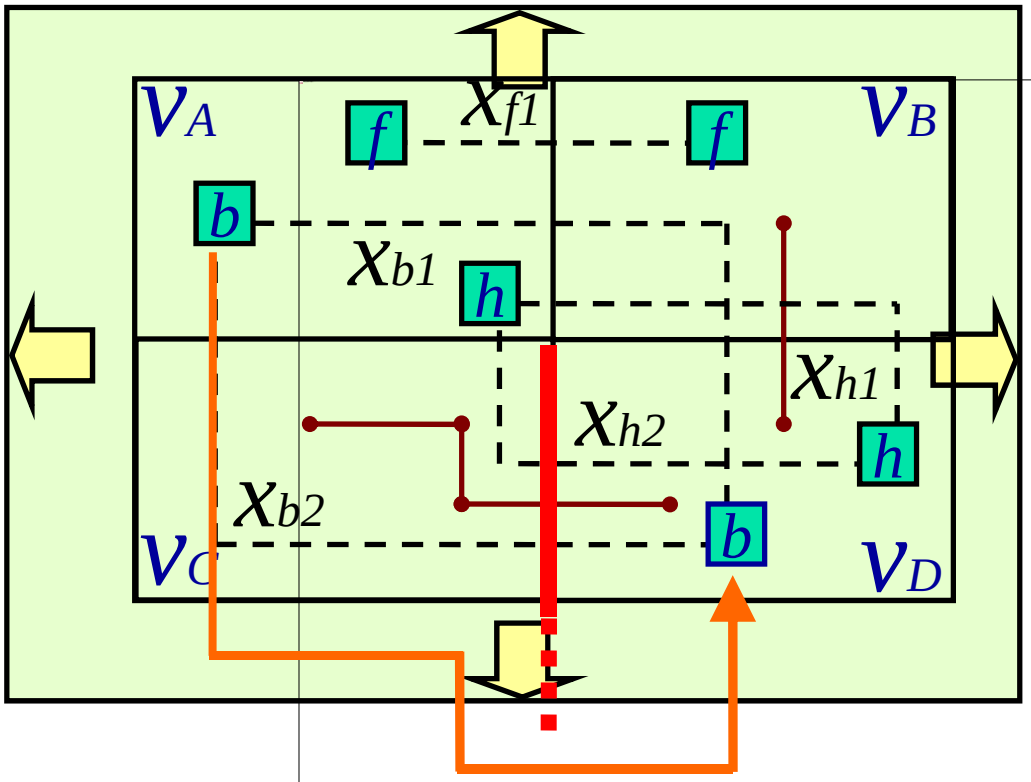
Initial boxes

Steiner Tree & Net Decomposition & Prerouting



- Minimum Steiner Tree: FLUTE [ICCAD-04], Spanning graph [ISPD-03], Geosteiner 3.1
- Decompose into 2-pin wires
  - Each wire becomes an atomic routing object
- Preroute as many flat wires as possible: 60% of nets
  - Overall congestion captured, runtime improved
- Start with the box of the most congested region

BoxRouting in Action



Maze router **[DAC03]**

$$max: \; x_{b1} + x_{b2} + x_{f1} + x_{f2} + x_{h1} + x_{h2}$$

$$s.t: \; x_{b1}, x_{b2}, x_{f1}, x_{f2}, x_{h1}, x_{h2} \in \{0,1\}$$

$$x_{b1} + x_{f1} + x_{h1} \le C_{AB}$$

$$x_{b1} + x_{h1} \le C_{BD}$$

routing cap. constraint

$$x_{b2} + x_{h2} \le C_{AC}$$

$$x_{b2} + x_{h2} \le C_{CD}$$

$$x_{b1} + x_{b2} \le 1$$

$$x_{f1} \le 1, x_{f2} = 0$$

$$x_{h1} + x_{h2} \le 1$$

solution constraint

solution

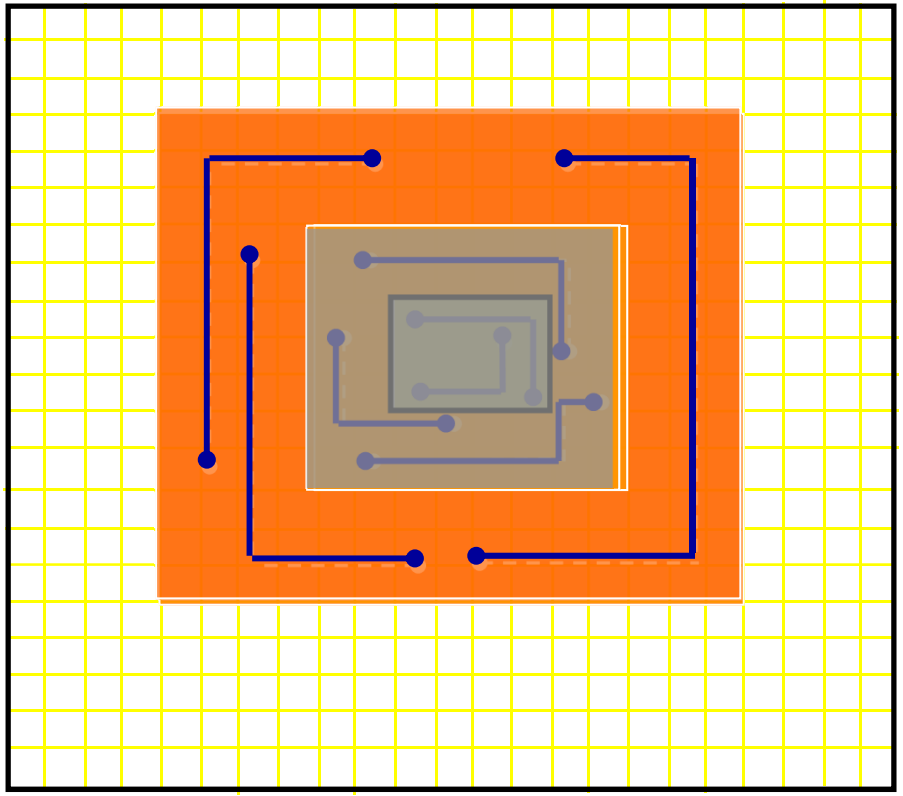
$$x_{b1} = x_{b2} = 0$$

$$x_{f1} = 1$$

$$x_{h1} = 1$$

Progressive ILP with Box Expansion

- ILP typically incurs very high time complexity
  - How to reduce the time complexity?
- Key: Progressive ILP
  - Incorporate the solution from inner boxes
  - Solve it between two consecutive boxes



Comparison between ILP Formulations

$$max: \sum \{x_{i1} + x_{i2}\}$$
$$s.t: x_{i1}, x_{i2} \in \{0,1\} \quad \forall i \in W_{box}$$
$$x_{i1} + x_{i2} \le 1 \quad \forall i \in W_{box}$$
$$x_{i2} = 0 \quad \forall i \in W_{box} \cap W_{flat}$$
$$\sum_{e \in x_{ij}} x_{ij} \le C_e \quad \forall i \in W_{box}$$

- ILP formulation with constant bounds
  - Maximize # routed nets (min. total overflow)
  - Work together with maze routing

$$min: D \quad \forall i \in W_{box}$$
$$s.t: x_{i1}, x_{i2} \in \{0,1\} \quad \forall i \in W_{box}$$
$$x_{i1} + x_{i2} = 1 \quad \forall i \in W_{box}$$
$$x_{i2} = 0 \quad \forall i \in W_{box} \cap W_{flat}$$
$$\sum_{e \in x_{ij}} x_{ij} \le D \quad \forall i \in W_{box}$$

- ILP formulation with bounds as objective
  - Minimize max overflow

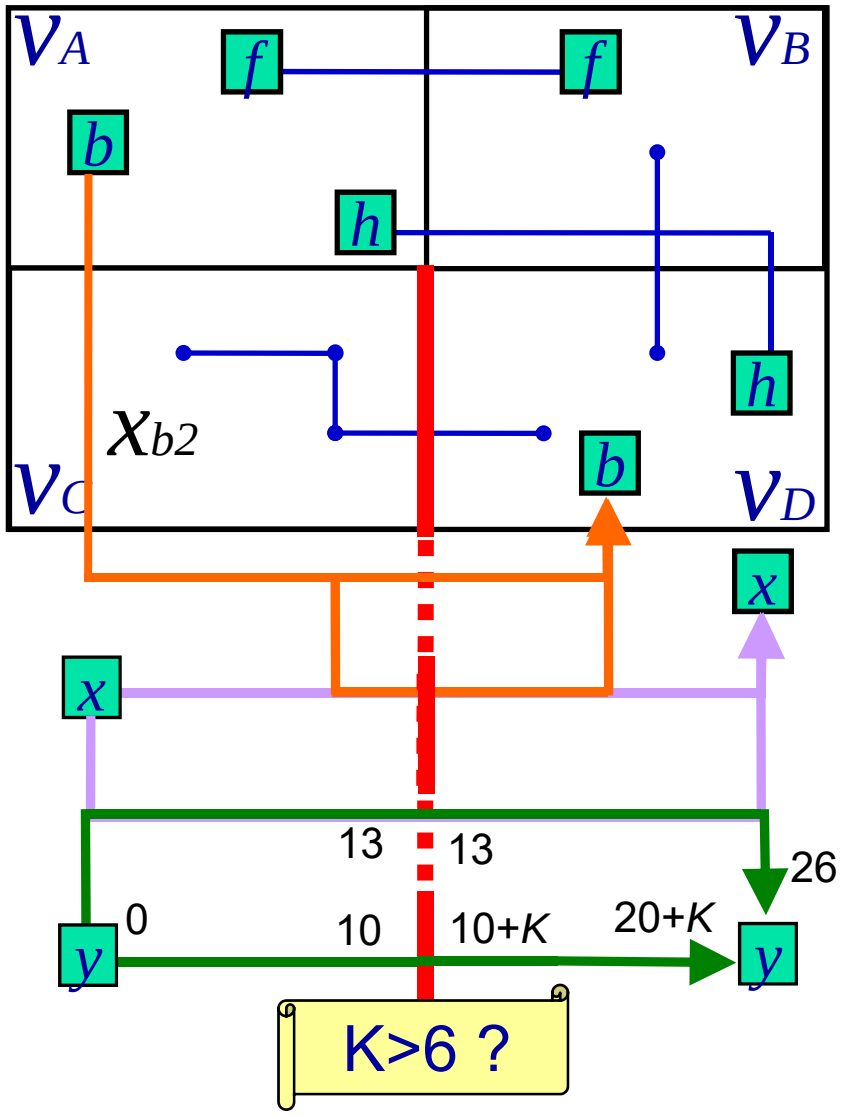
Much slower!!

(Try not to use formulation of this kind!!)

Post Routing

- Reroute w/o ripping up other nets
  - In the same order as box expansion
  - Enhance routing path
- Tradeoff between wirelength and congestion
  - Controlled by a parameter *K*

  K ↑ => overflow↓, wirelength↑

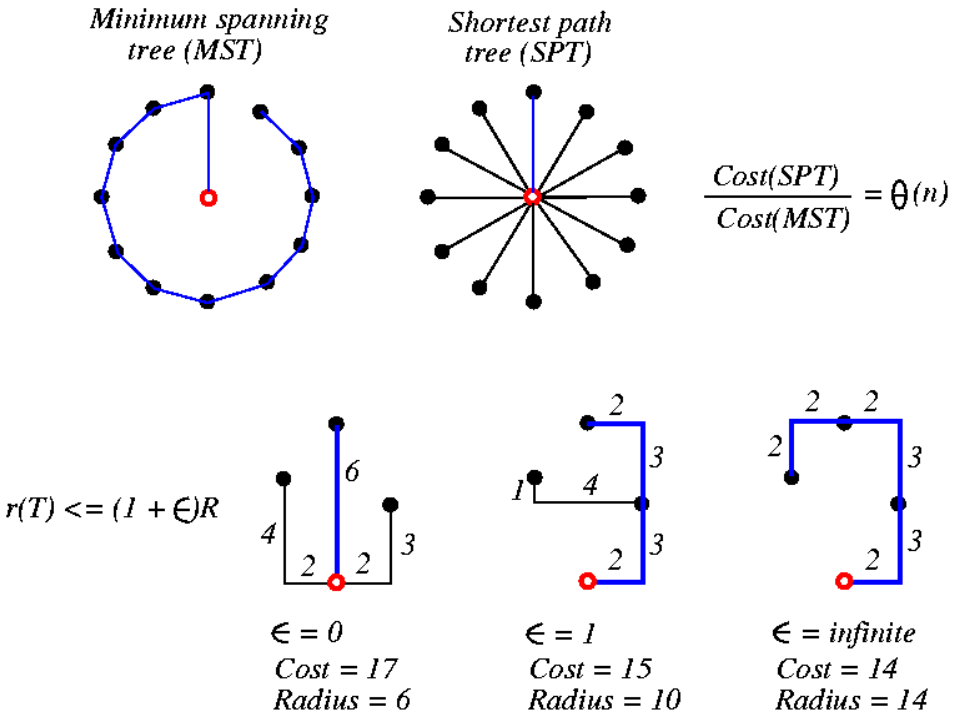


Appendix F:

Bounded-Radius Minimum Spanning Tree

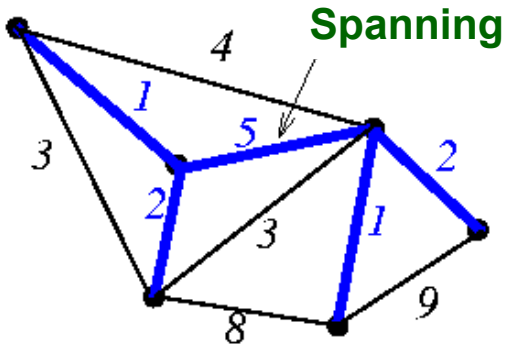
Cong, Kahng, Robins, Sarrafzadeh, Wong

ICCD-91, TCAD-92



Bounded-Radius Minimum Spanning Tree

- **Problem:** Given a parameter $\varepsilon \geq 0$ and a signal net with radius R (diameter D), find a minimum-cost spanning tree T with radius $r(T) \leq (1 + \varepsilon)R$ ($d(T) \leq (1 + \varepsilon)D$).
  - Awerbuch, *et al*., "Cost-sensitive analysis of communication protocols," *ACM Symp. Principles of Distributed Computing*, 1990.
  - Cong, Kahng, Robins, Sarrafzadeh, Wong, "Performance-driven global routing for cell based IC's," ICCD-91 (& TCAD, June 1992).
- R (D): the maximum length among all shortest paths from the source (among every pair of nodes).
- MST (minimum spanning tree) $\leftrightarrow$ minimum cost; SPT (shortest path tree) $\leftrightarrow$ minimum radius.
- Question: How to find a spanning tree with a good trade-off between cost and radius?

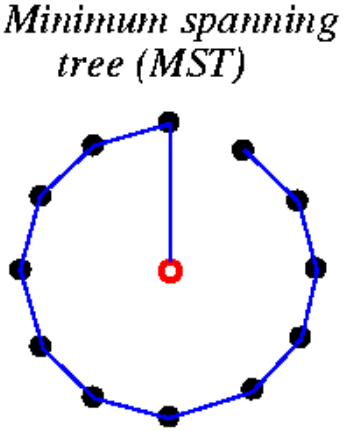


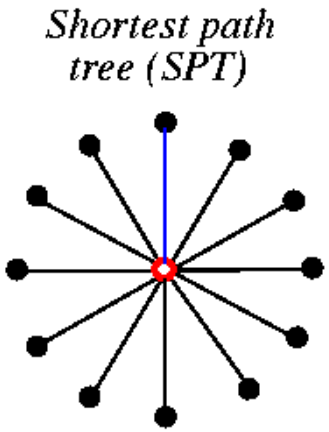
$Cost(T)=1+2+5+1+2=11$

$Radius(T)=max\{1+2, 1+5+1, 1+5+2\}=8$

MST vs. SPT Trade-off

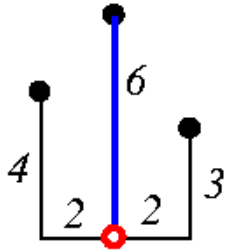
- *Cost*(*SPT*) may be $\Omega(|n|)$ times larger than *Cost*(*MST*).



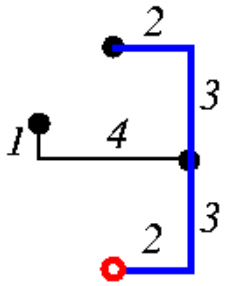


$$\frac{Cost(SPT)}{Cost(MST)} = \theta(n)$$

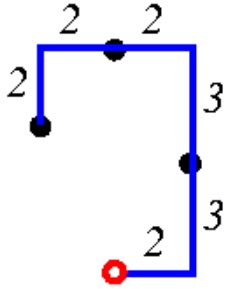
$r(T) <= (1 + \epsilon)R$



$\epsilon = 0$
Cost = 17
Radius = 6



$\epsilon = 1$
Cost = 15
Radius = 10

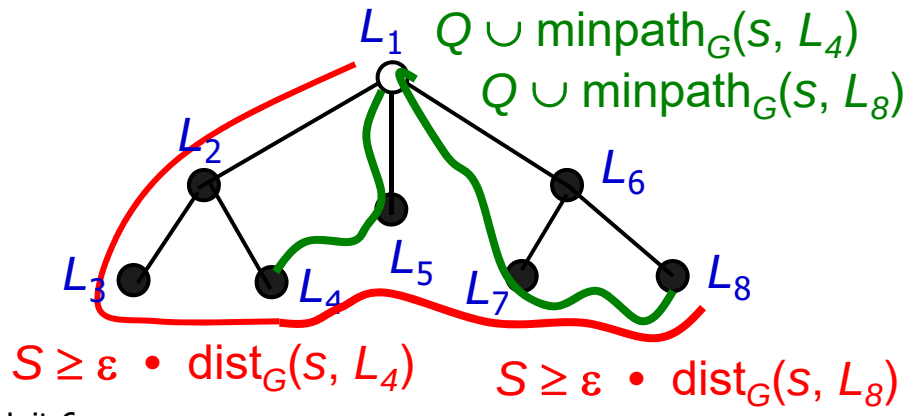


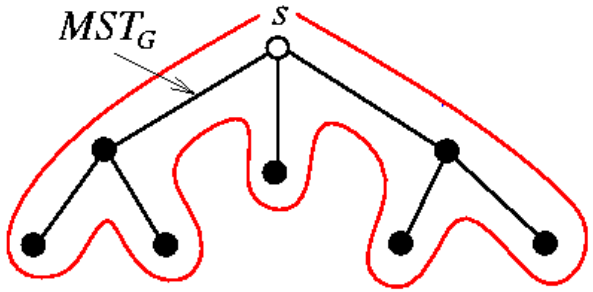
$\epsilon = infinite$
Cost = 14
Radius = 14

Bounded Radius Bounded Cost Spanning Tree

Algorithm: Bounded-Radius_Bounded-Cost_Spanning_Tree(*G*)

```
1 begin
2 Compute MST_G and SPT_G;
3 Q ← MST_G;
4 L ← depth-first tour of MST_G;
5 S ← 0;
6 for i ← 1 to |L| - 1
7     S ← S + cost(L_i, L_{i+1});
8     if S ≥ ε • dist_G(s, L_{i+1}) then
9         Q ← Q ∪ minpath_G(s, L_{i+1});
10        S ← 0;
11 T = shortest path tree of Q;
12 end
```





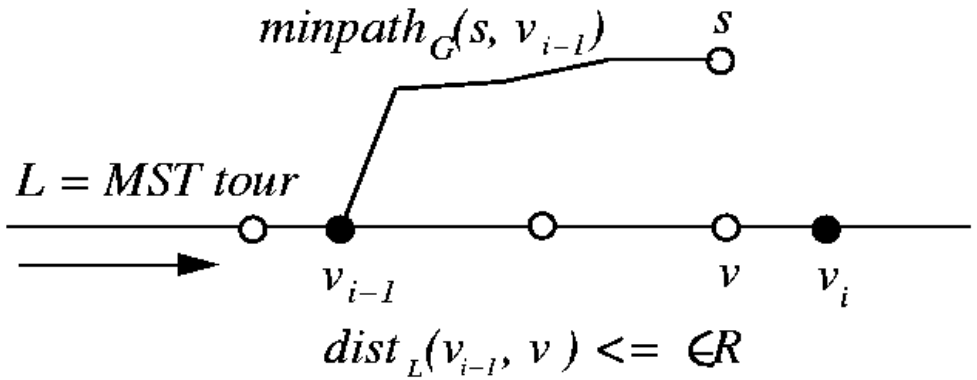
L = depth–first tour of MST

cost(L) <= 2 cost(MST_G)

Bounded-Radius Bounded-Cost Spanning Tree

- For any weighted graph G and parameter ε, the routing tree T constructed by the algorithm has radius $r(T) \leq (1+\varepsilon)R$.
- v_{i-1}: the last node before v on L for which we added $minpath_G(s, v_{i-1})$ to Q.

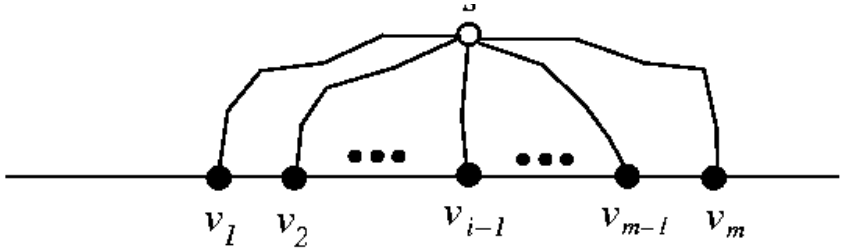
$$\begin{aligned} dist_T(s,v) &\leq dist_T(s,v_{i-1}) + dist_L(v_{i-1},v) \\ &\leq dist_G(s,v_{i-1}) + \epsilon R \\ &\leq R + \epsilon R \\ &= (1+\epsilon)R \end{aligned}$$



Bounded-Radius Bounded-Cost Spanning Tree

- For any weighted graph G and parameter ε, the routing tree T constructed by the algorithm has cost $cost(T) \le (1+2/\varepsilon)cost(MST_G)$.
- Let $v_1, v_2, \ldots, v_m$ be the set of nodes to which the algorithm added shortest paths from source s.

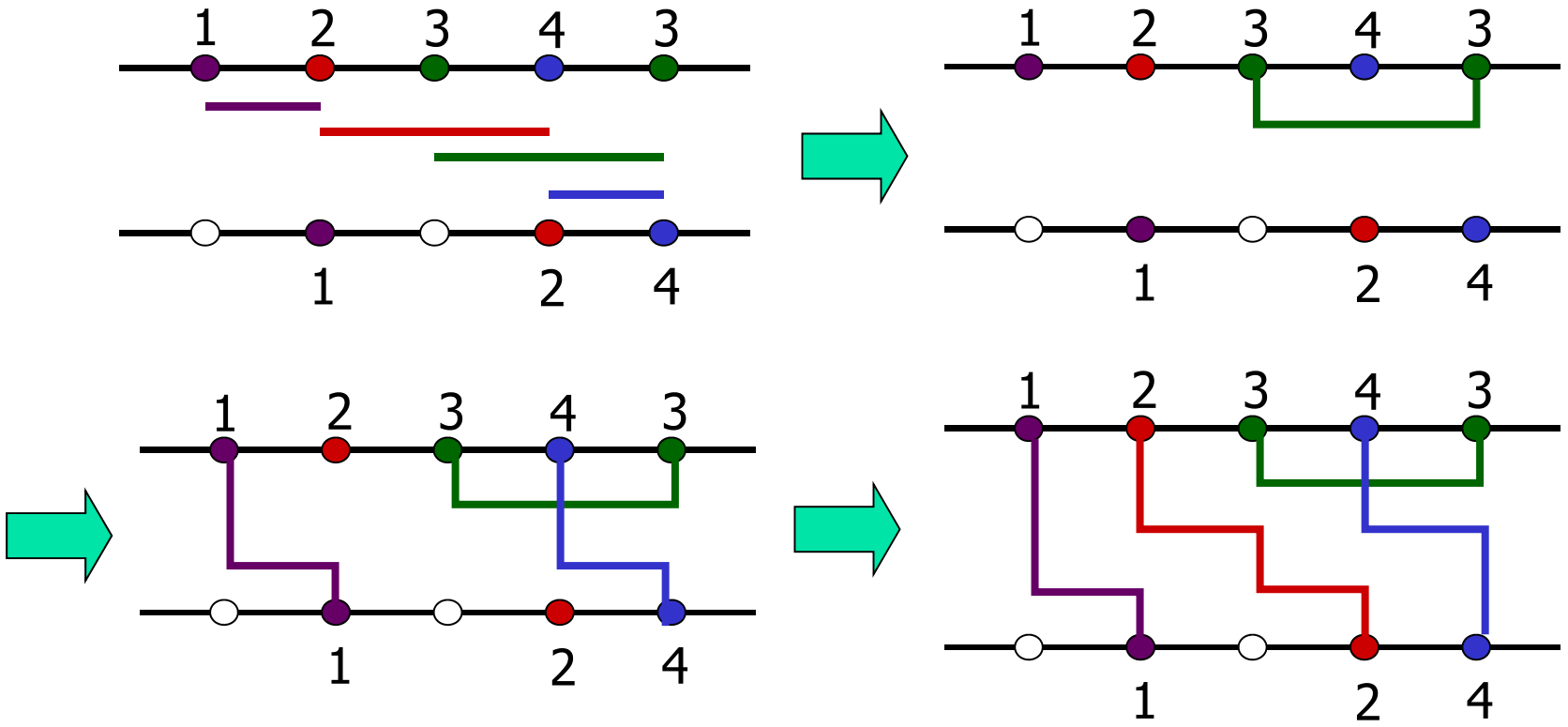
$$
\begin{aligned}
cost(T) &\le cost(MST_G) + \sum_{i=1}^{m} dist_G(s, v_i) \\
dist_L(v_{i-1}, v_i) &\ge \epsilon \cdot dist_G(s, v_i) \\
cost(T) &\le cost(MST_G) + \sum_{i=1}^{m} \frac{dist_G(v_{i-1}, v_i)}{\epsilon} \\
&\le cost(MST_G) + \frac{cost(L)}{\epsilon} \\
&\le cost(MST_G) + \frac{2 \cdot cost(MST_G)}{\epsilon} \\
&\le (1 + \frac{2}{\epsilon}) cost(MST_G)
\end{aligned}
$$



Appendix G:

Robust Channel Router

Yoeli, TCAD-91



Robust Channel Router

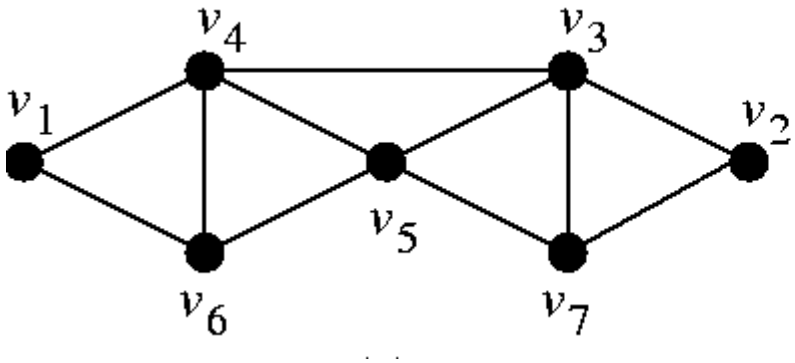
- Yoeli, “A robust channel router,” IEEE TCAD, 1991.
- Alternates between top and bottom tracks until the center is reached.
- The working side is called the *current side*.
- Net weights are used to guide the assignment of segments in a track, which
  - favor nets that contribute to the channel density;
  - 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.

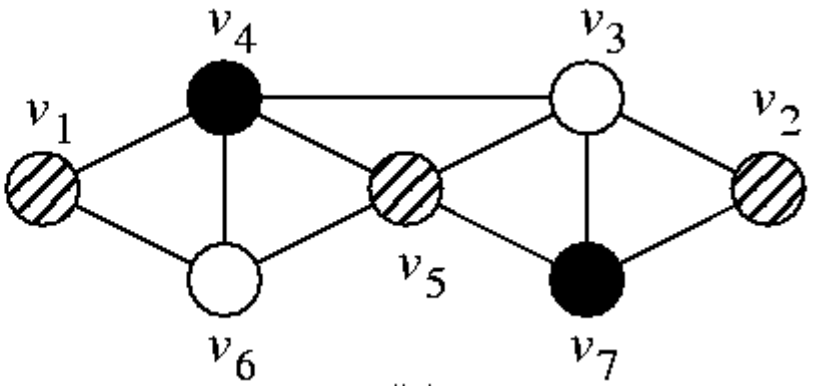
Robust Channel Router

- 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??)

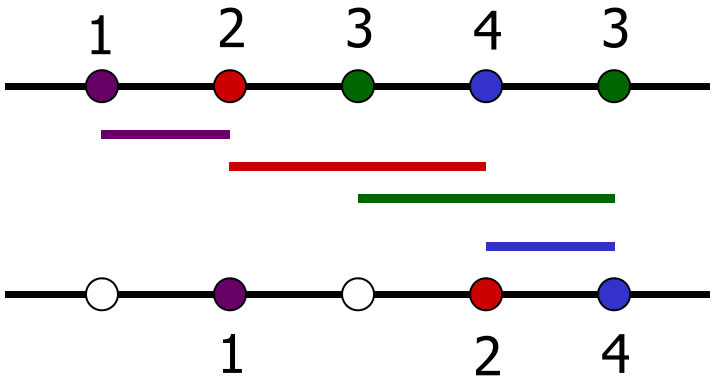
Interval Graphs

- 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.





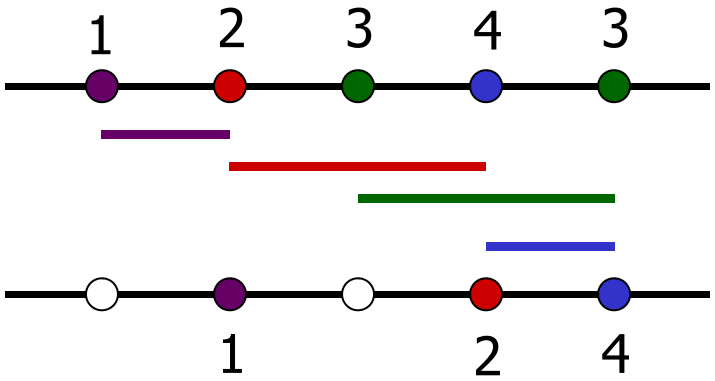
Weight Computation



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

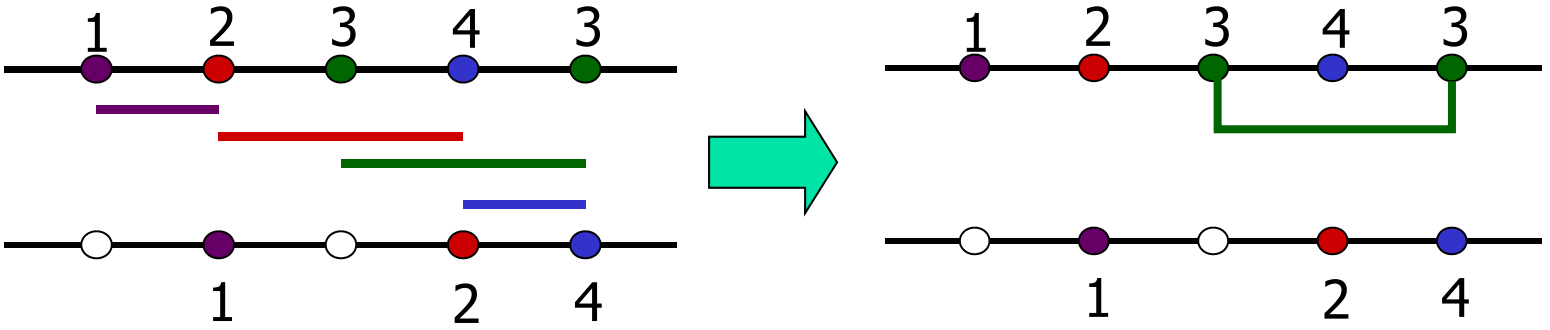
Weight Computation (cont'd)



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

Top-Row Net Selection

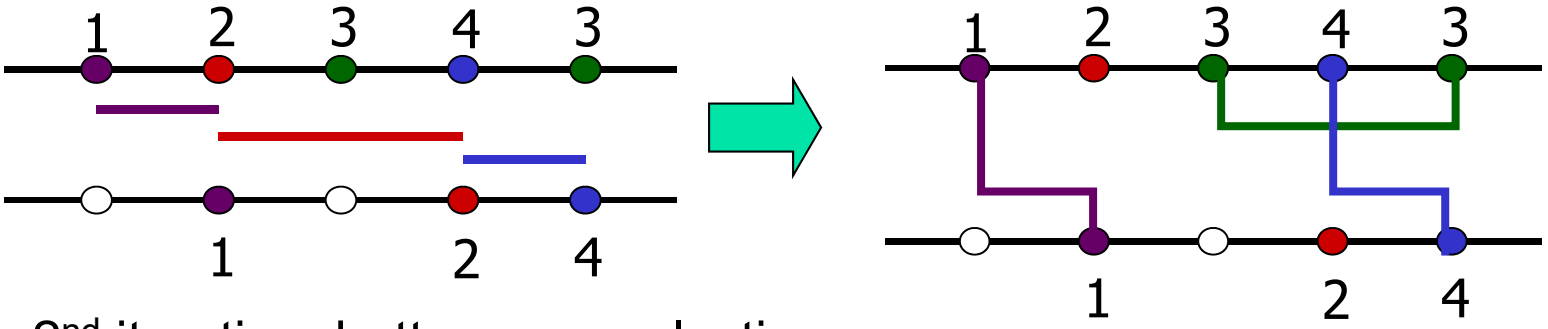


- $w_1 = -9$, $w_2 = 987$, $w_3 = 1004$, $w_4 = 993$.
- A net *n* with a rightmost terminal at position *c* is taken into the solution if: total[$c - 1$] < w_n + total[$x_{n_{min}} - 1$].

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.)

Bottom-Row Net Selection

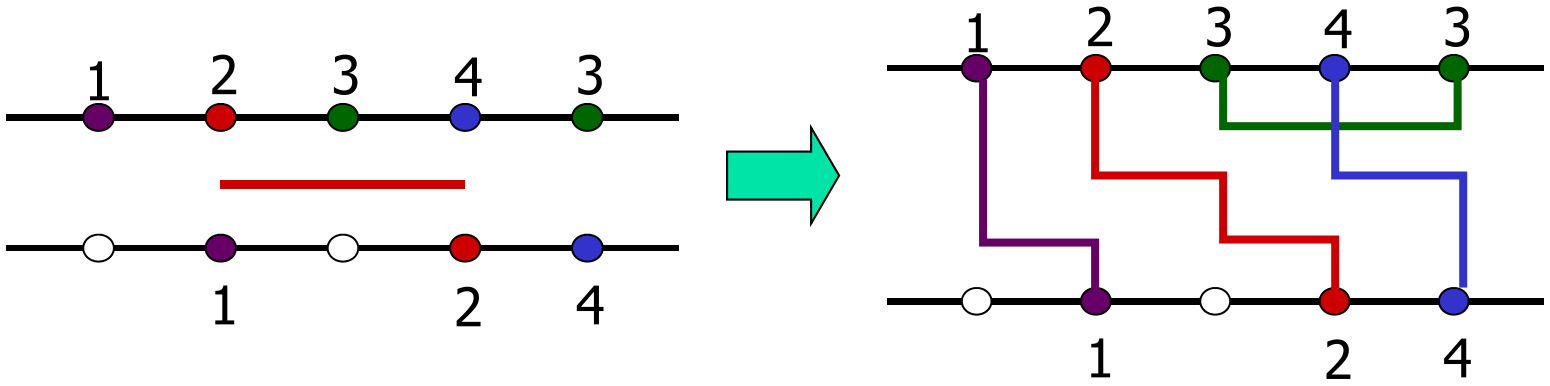


- 2nd iteration: bottom-row selection
  - w_1 = (1000) + (2) + (0) = 1002
  - w_2 = (1000) + (2) + (-5 * 2) = 992
  - w_4 = (1000) + (1) + (-5 * 2) = 991

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.

Maze Routing + Rip-up & Re-route



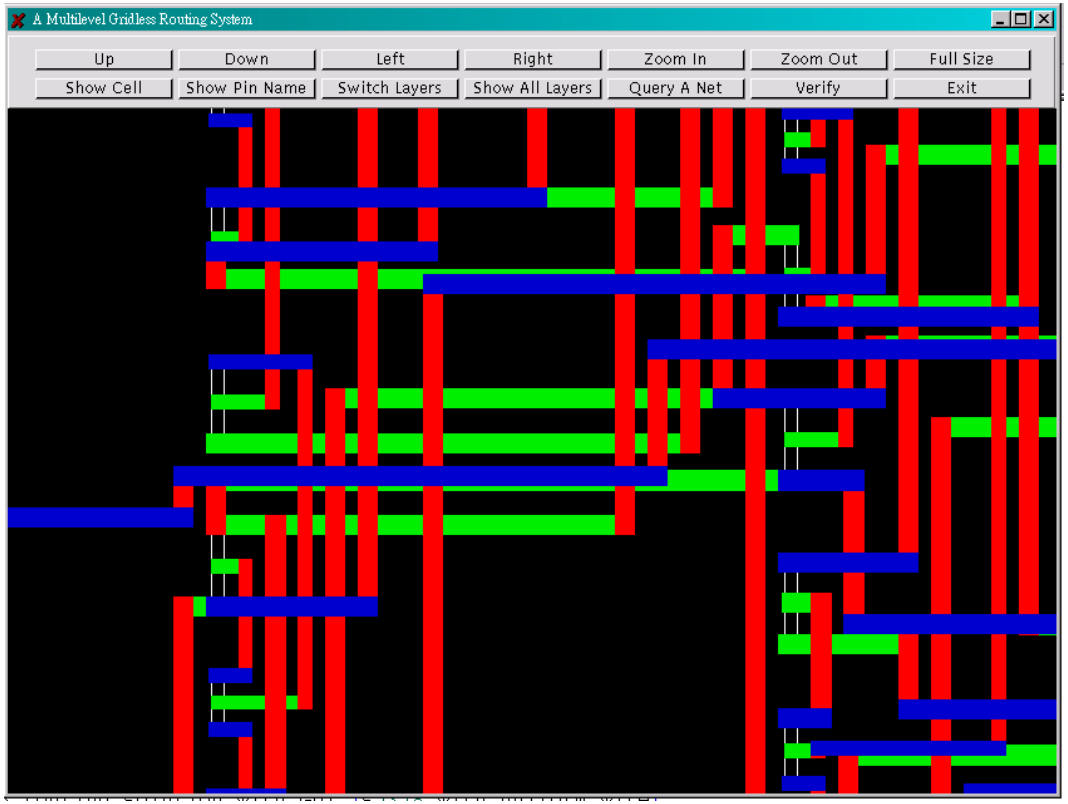
- 3rd iteration
  - Routing net 2 in the middle row leads to an infeasible solution.
  - Apply maze routing and rip-up and re-route nets 2 and 4 to fix the solution.

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"
}
```

Appendix H:
Gridless Routing

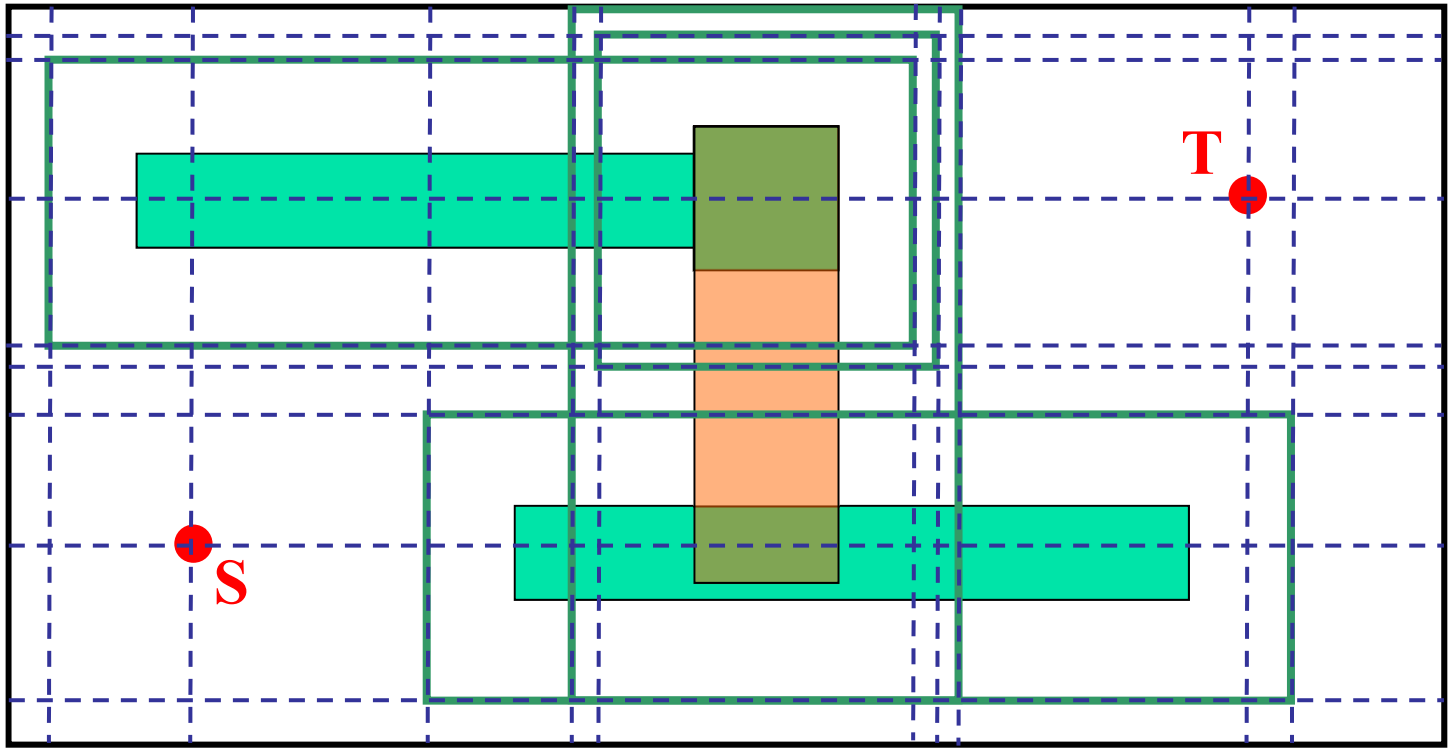


Gridless Full-Chip Routing

- Chen and Chang, “Multilevel full-chip gridless routing considering optical proximity correction,” ASPDAC-05 (TCAD-08).
- 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.

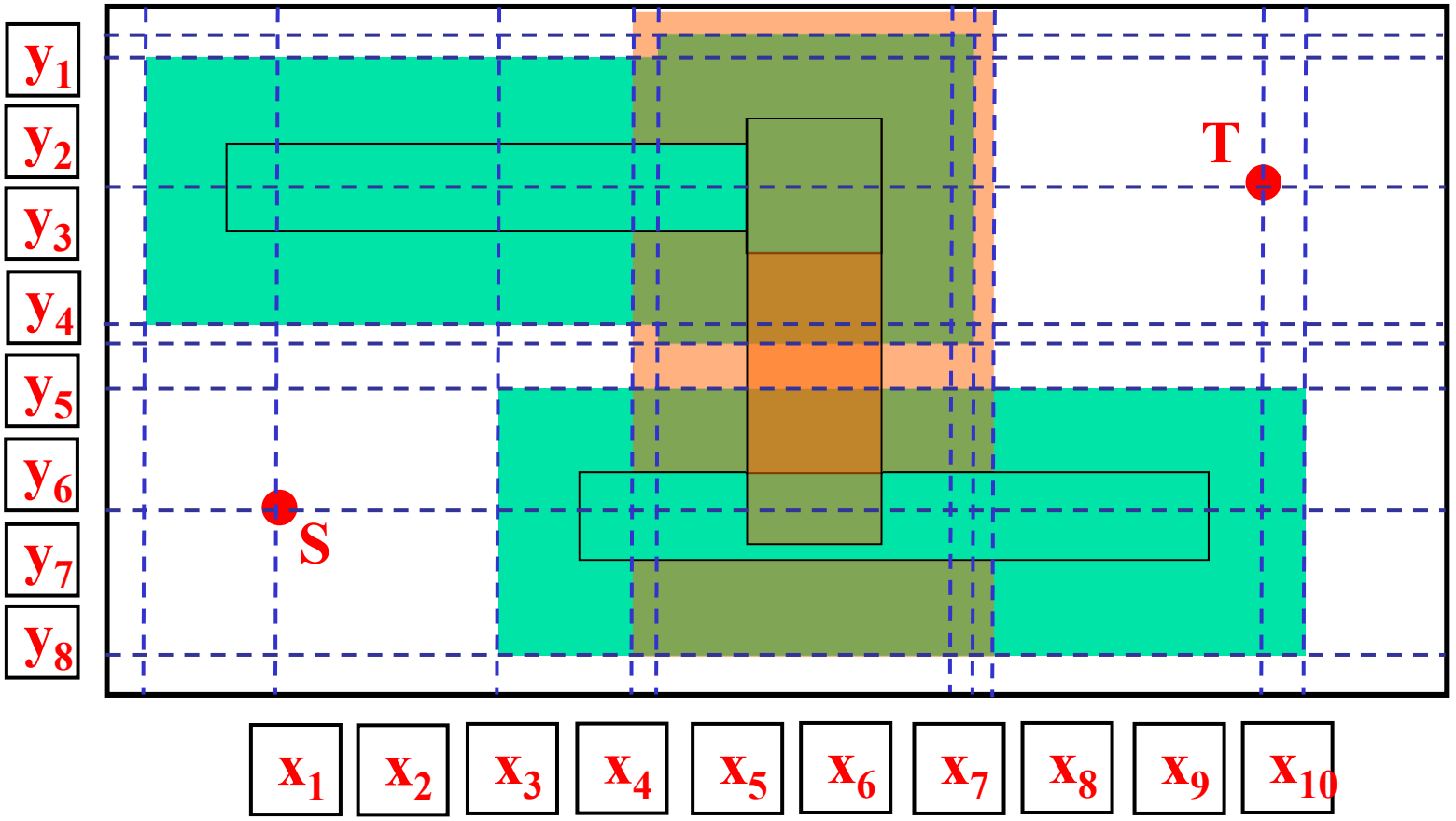
Implicit Connection Graph

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



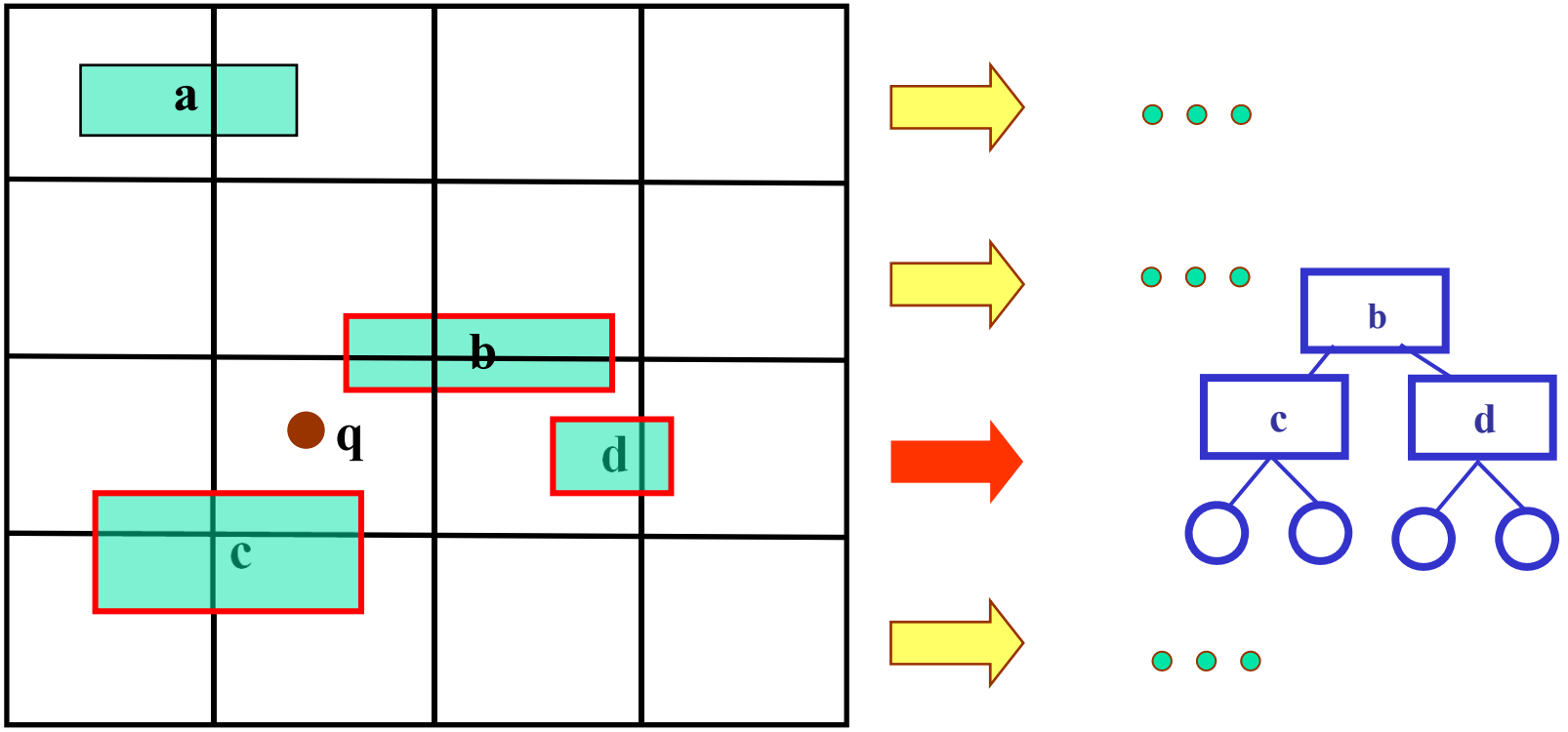
Implicit Representation of G_S

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



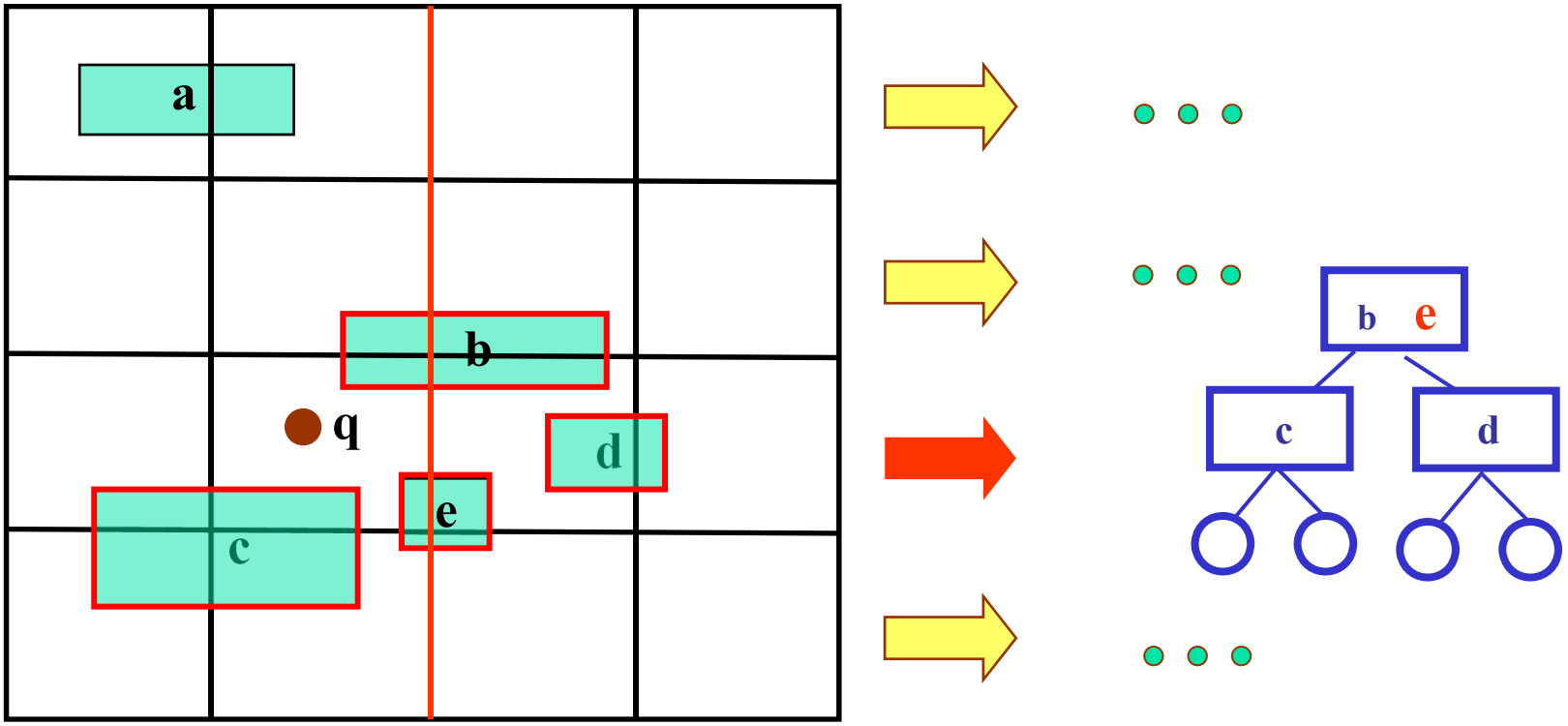
Interval Tree

- 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 stored in the leaf nodes.
- Question: Is *q* in free space? Need average $O(\lg n)$ time to answer.



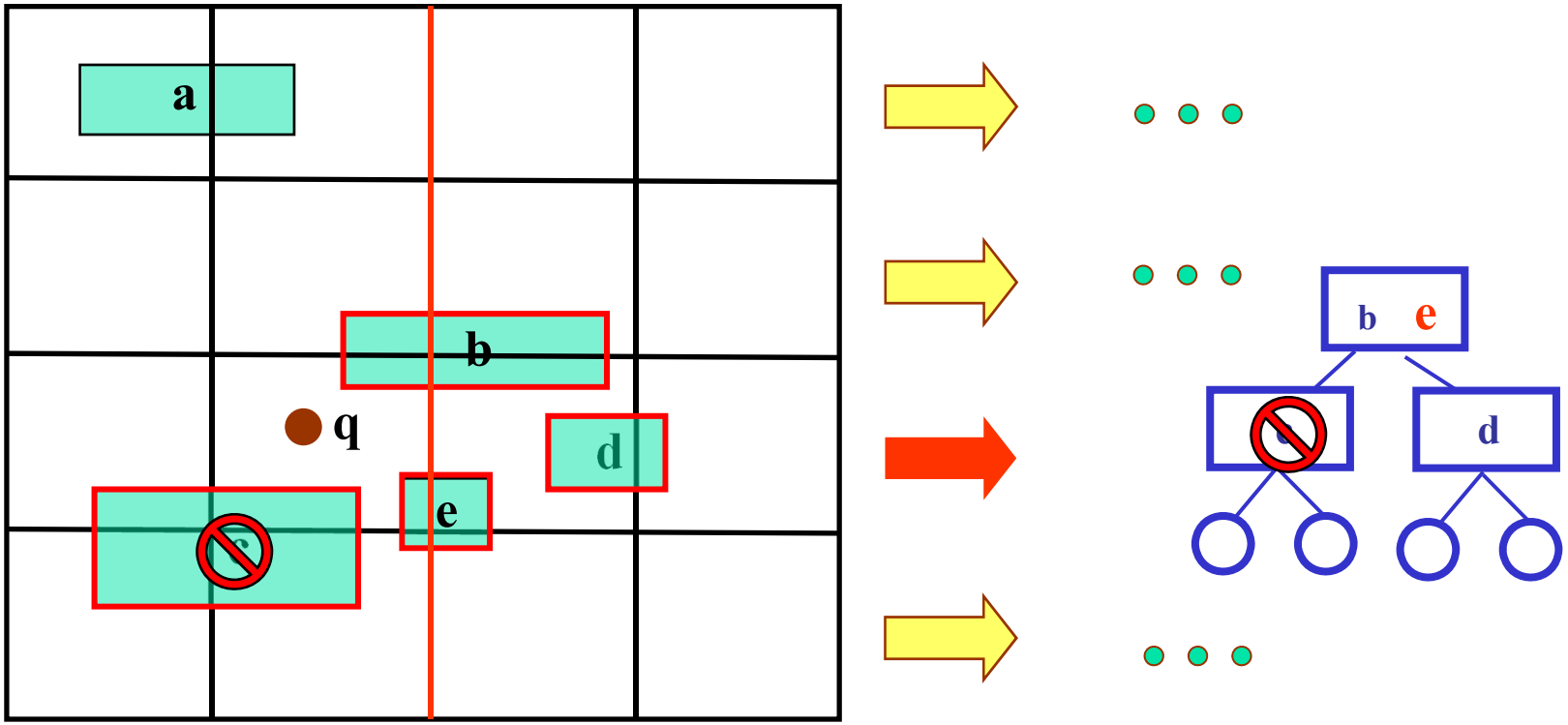
Block Insertion

- 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: $O(\lg n)$.



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: $O(\lg n)$.



Complexity Summary

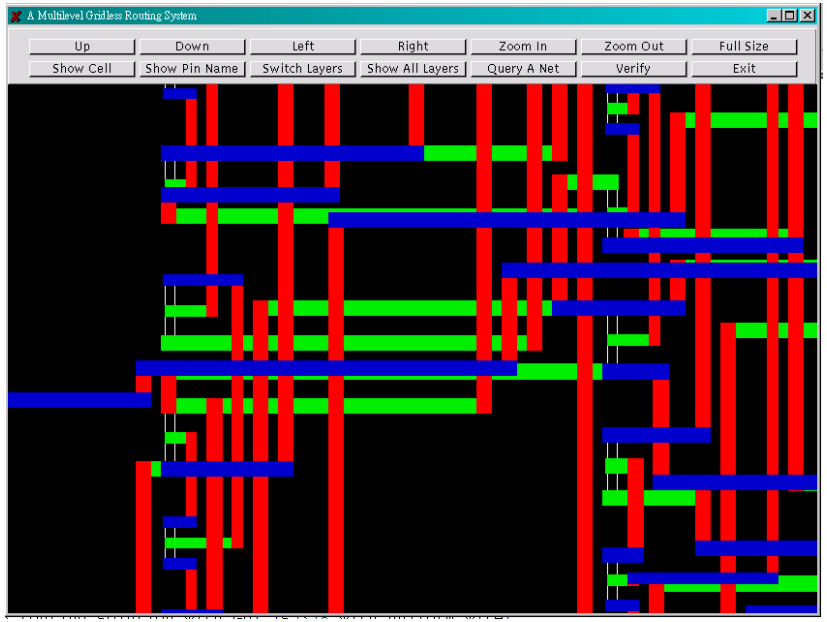
- Interval (query) tree

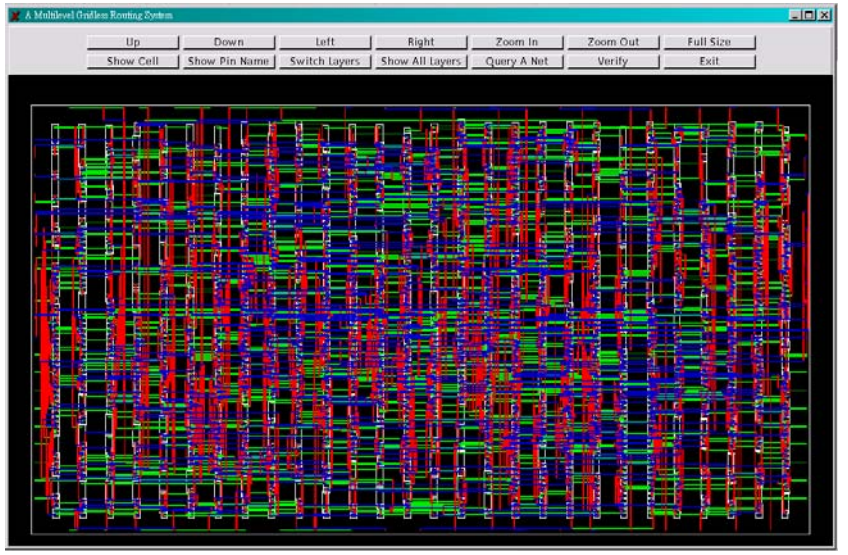
	Average Complexity
Memory Usage	**O(n^2)**
Block Insertion	**O(lg n)**
Block Deletion	**O(lg n)**
Point Finding	**O(lg n)**
Neighbor Finding	**O(1)**

n: number of blocks

Routing Solution for s5378

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





Courtesy of Tai-Chen Chen