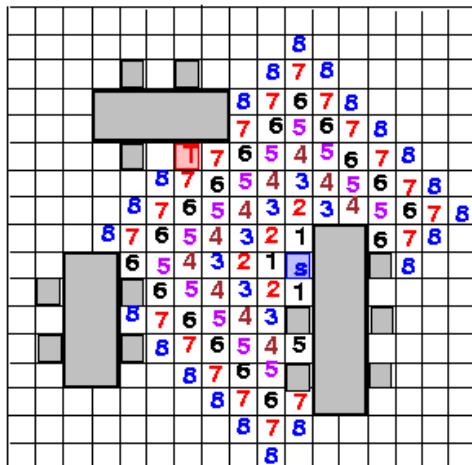
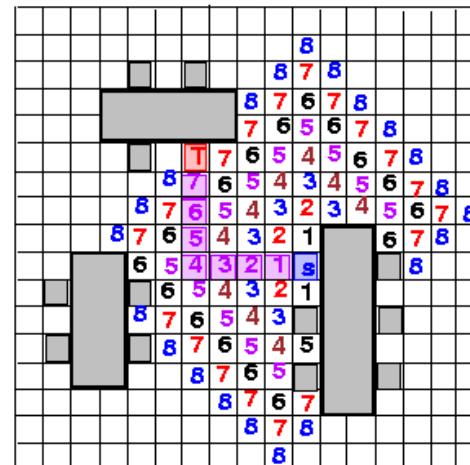


Unit 6: Global and Maze Routing

- Course contents:
 - Maze routing
 - Global routing
 - Routing trees
- Readings
 - W&C&C: Chapter 12
 - S&Y: Chapters 5 and 6

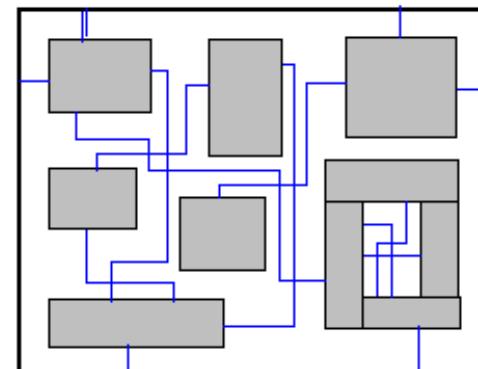
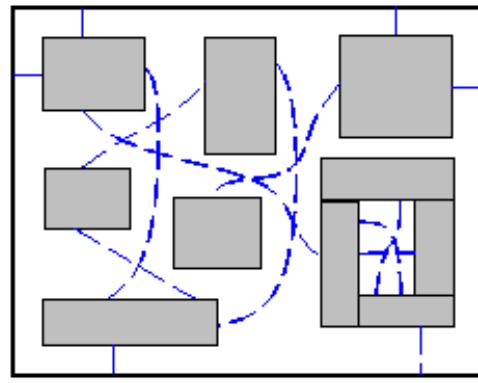
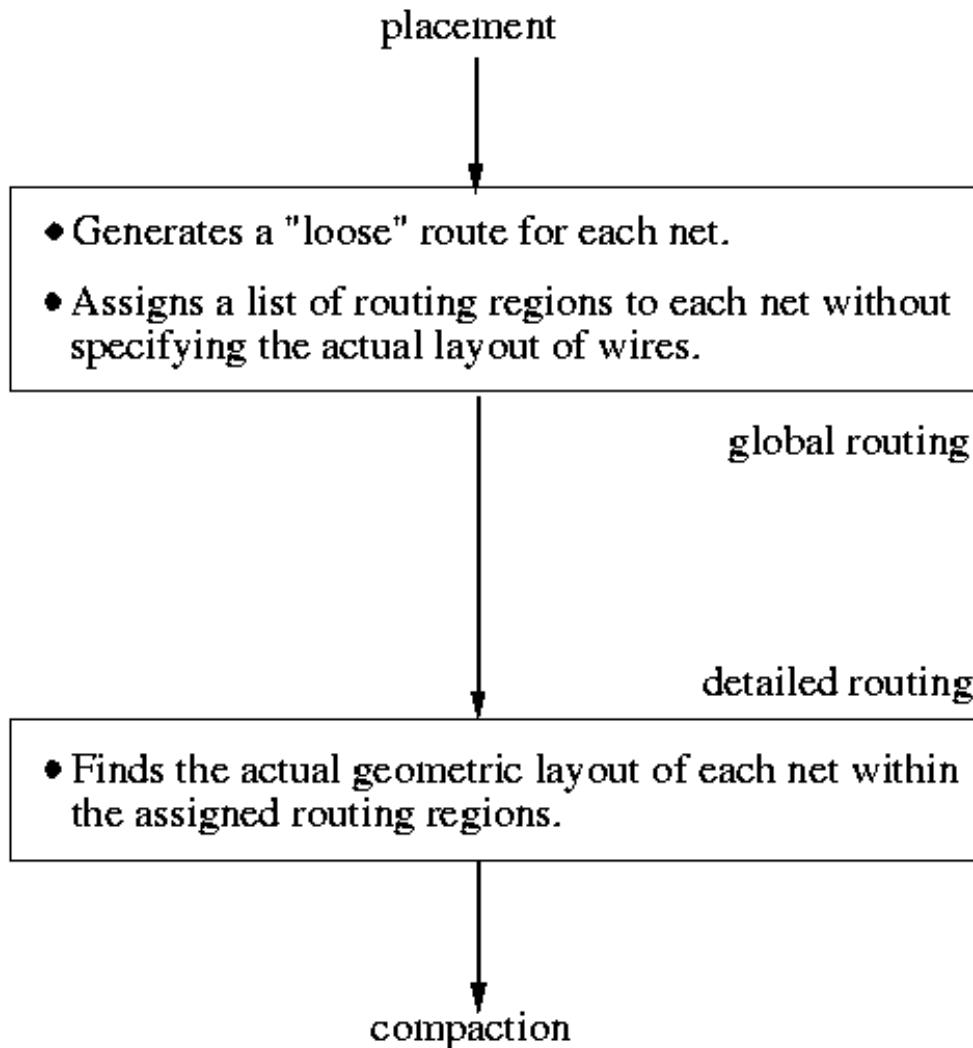


Filling



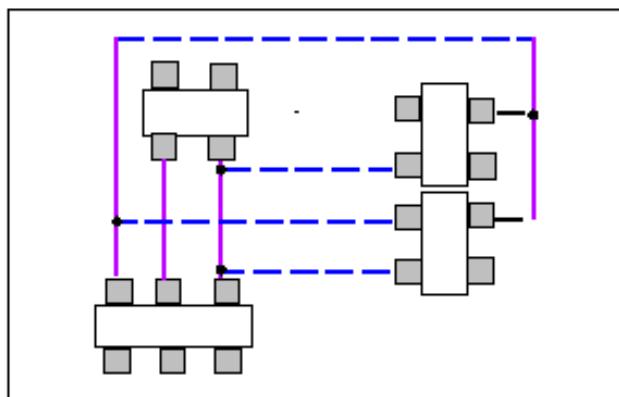
Retrace

Routing

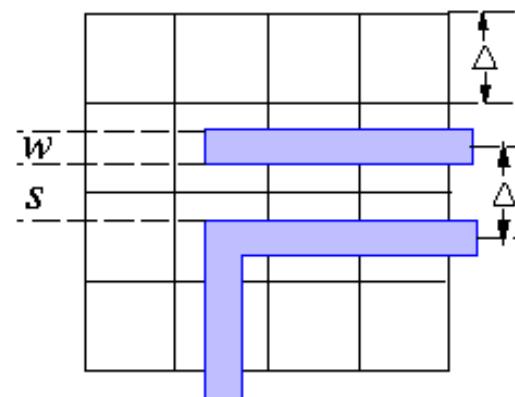


Routing Constraints

- 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
 - Crosstalk?
 - Design for manufacturability, reliability, and printability?

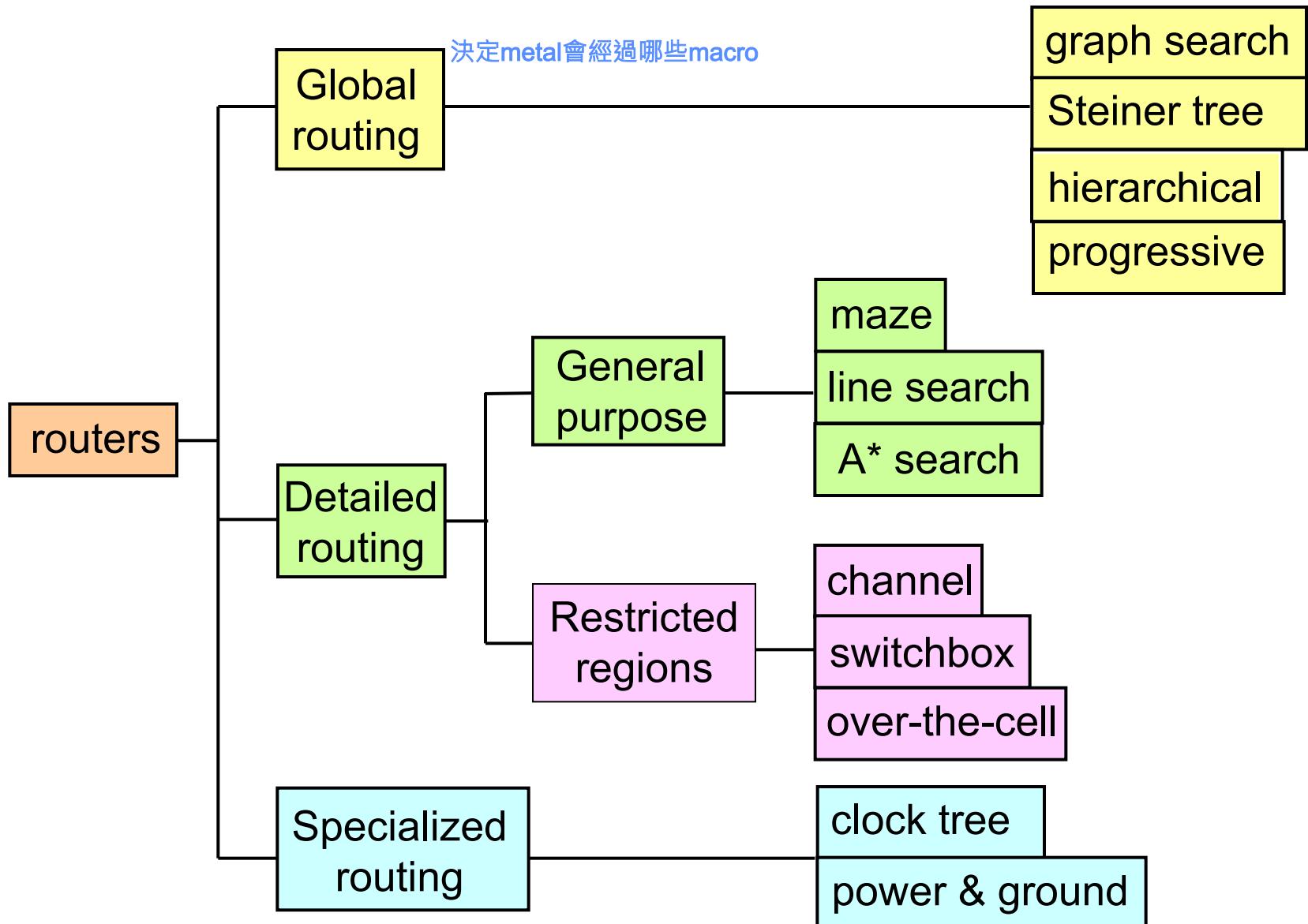


Two-layer routing



Geometrical constraint

Classification of Routers

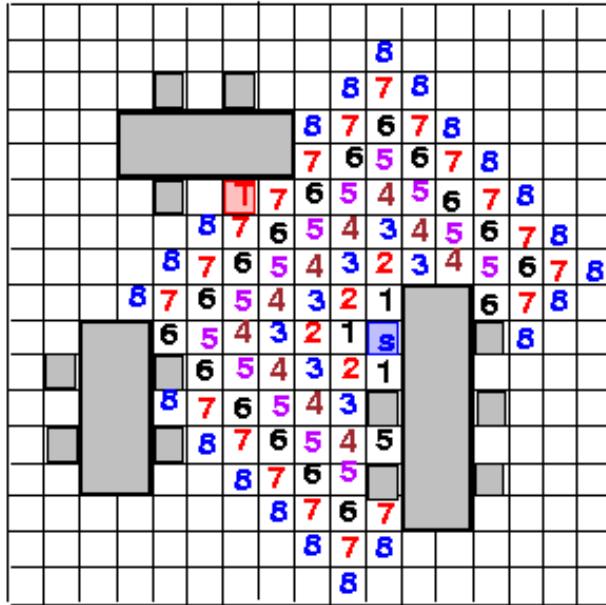


Maze Router: Lee Algorithm

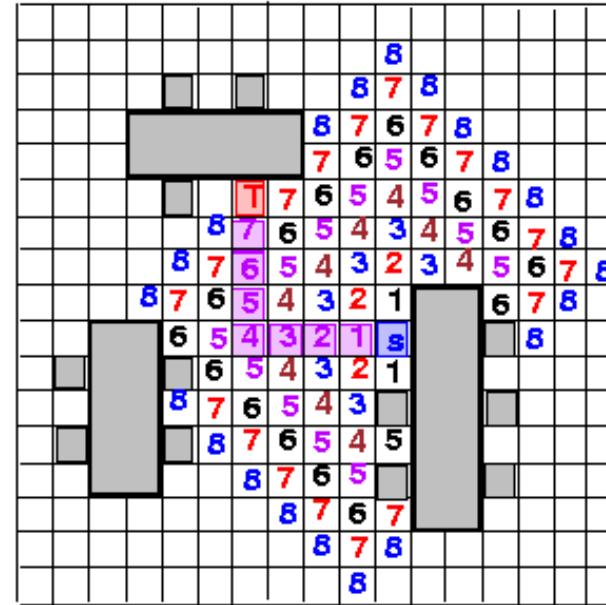
- Lee, “An algorithm for path connection and its application,” *IRE Trans. Electronic Computer*, EC-10, 1961.
- Discussion mainly on single-layer routing
- **Strengths**
 - Guarantee to find connection between 2 terminals if it exists.
 - Guarantee minimum path.
- **Weaknesses**
 - Requires large memory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

Lee Algorithm

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



Filling

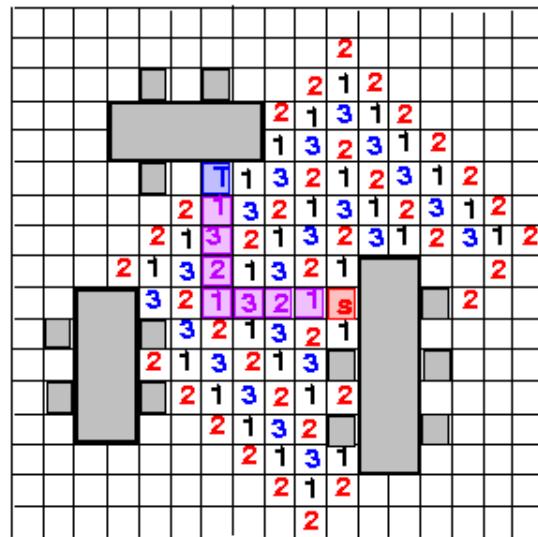


Retrace

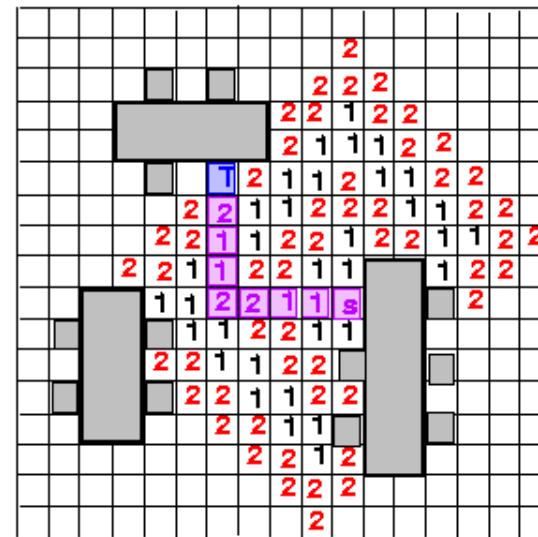
- Time & space complexity for an $M \times N$ grid: $O(MN)$ (**huge!**)

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, ...

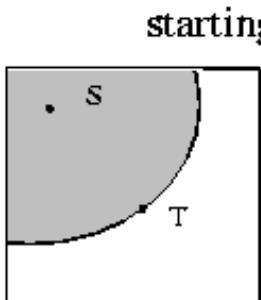


Sequence: 1, 1, 2, 2, 1, 1, 2, 2, ...

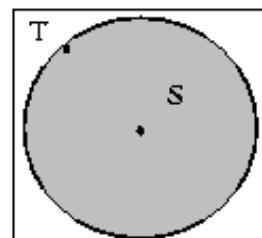
Reducing Running Time

- 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.

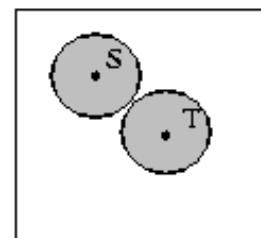
左邊所需要的格子數目比較少，因此速度會較快



starting point selection

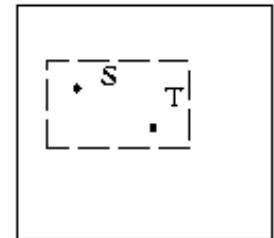


兩邊輪流擴散，填的
格子數目也會減少
double fan-out



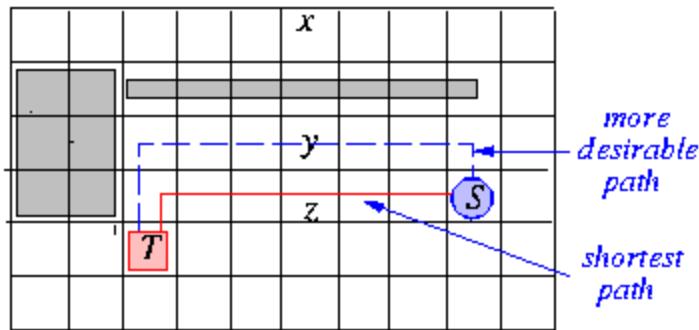
路徑可能不存在此frame中，會導
致routing fail(可能可以在一開始沒
什麼線的時候用)

framing



Routing on a Weighted Grid

- Motivation: finding more desirable paths
- $\text{weight}(\text{grid cell}) = \# \text{ of unblocked grid cell segments} - 1$
越靠近障礙物的weight越小，因此走線會傾向貼著障礙物走(總weight越小越好)



2	2	2	2	2	2	2	2	2	3
2	2	2	2	2	2	2	2	2	2
1	2	2	2	2	2	2	2	1	3
1	3	3	3	3	3	3	2	1	2
2	1	T	2	3	3	3	3	2	3
3	3	2	3	3	3	3	3	3	3

occupied grid cell
blocked grid segments
weights

Example Routing on a Weighted Grid

2	2	2	2	2	2	2	2	2	3
									2
	1	2	2	2	2	2	2	1	3
	1	3	3	3	3	2	S	2	
2	1	T	2	3	3	3	3	2	3
3	3	2	3	3	3	3	3	3	3

initialize cell weights

							3	1	4
						5	2	S	2
		T				5	2	5	
								5	

wave propagation

									13
								11	9
									6

					15	13	11	9	
									6
					12	11	9	7	5
					15	14	11	8	5
					15	16	14	11	8
					19	17	16	14	11

first wave reaches the target

					17	15	13	11	9
					12	11	9	7	5
					13	14	11	8	5
					16	15	16	14	11
					17	19	17	14	11

finding other paths

					19	17	15	13	11
					12	11	9	7	5
					13	14	11	8	5
					16	15	16	14	11
					19	17	19	17	14

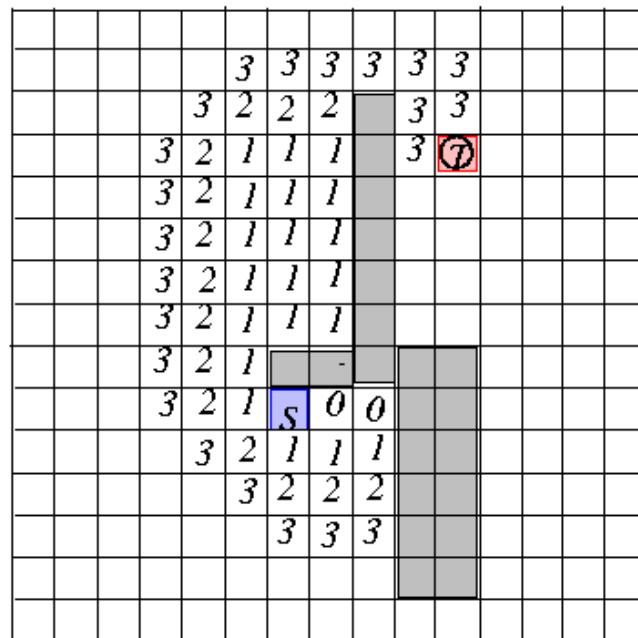
min-cost path found

Hadlock's Algorithm

- Hadlock, “A shortest path algorithm for grid graphs,” *Networks*, 1977.
- Uses **detour number** (instead of labeling **wavefront** in Lee's router)
 優化目標: 減少 detour number(繞遠路的所浪費的距離)
 - 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 , $I(P)$: $I(P) = MD(S, T) + 2 d(P)$.
 即 detour
 why 2倍? 可能是因為要額外存一張能 trace back 的 map
 - $MD(S, T)$ fixed! \Rightarrow Minimize $d(P)$ to find the shortest path.
 - For any cell labeled i , label its adjacent unblocked cells **away from T** ; 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)

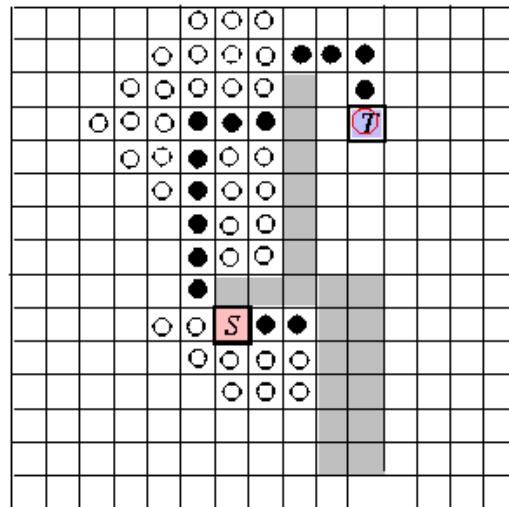
- $d(P)$: # of grid cells directed **away from** its target on path P .
- $MD(S, T)$: the Manhattan distance between S and T .
- Path length of P , $I(P)$: $I(P) = MD(S, T) + 2d(P)$.
- $MD(S, T)$ fixed! \Rightarrow Minimize $d(P)$ to find the shortest path.
- For any cell labeled i , label its adjacent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



往target擴散，detour就維持原本的數字，
若往反方向，則每步detour都會+1
-->target方面: weight = 0
-->target反向: weight = 1

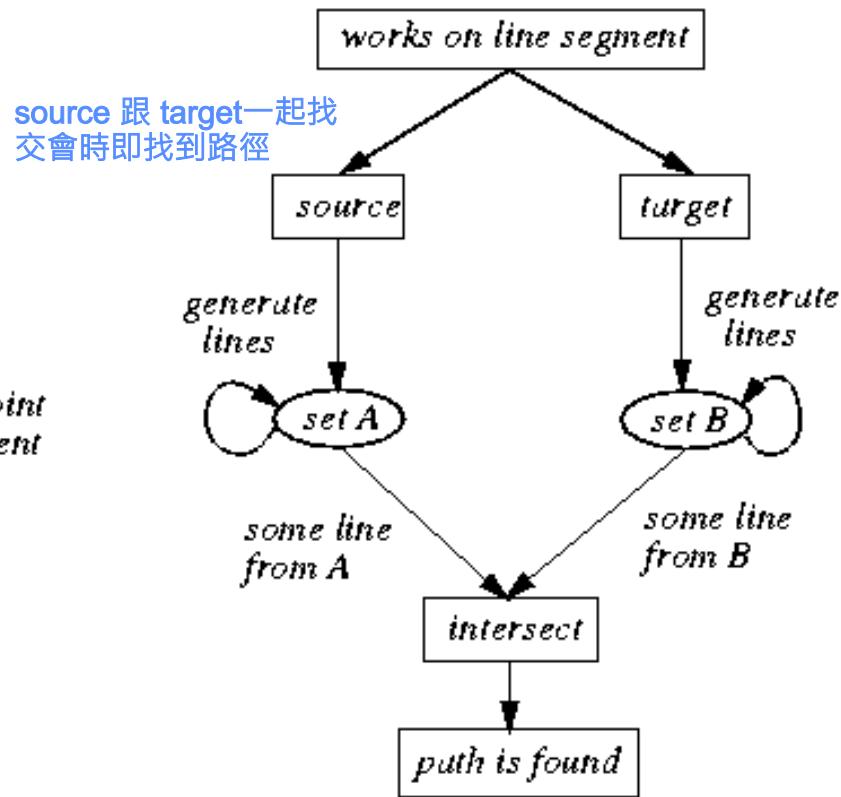
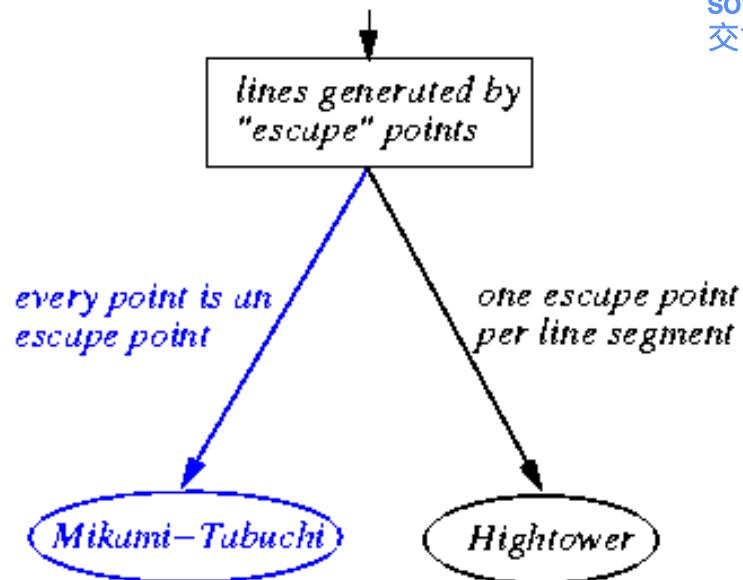
Soukup's Algorithm

- 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!



往target方向用DFS，撞到障礙物或是往target反方向時用BFS

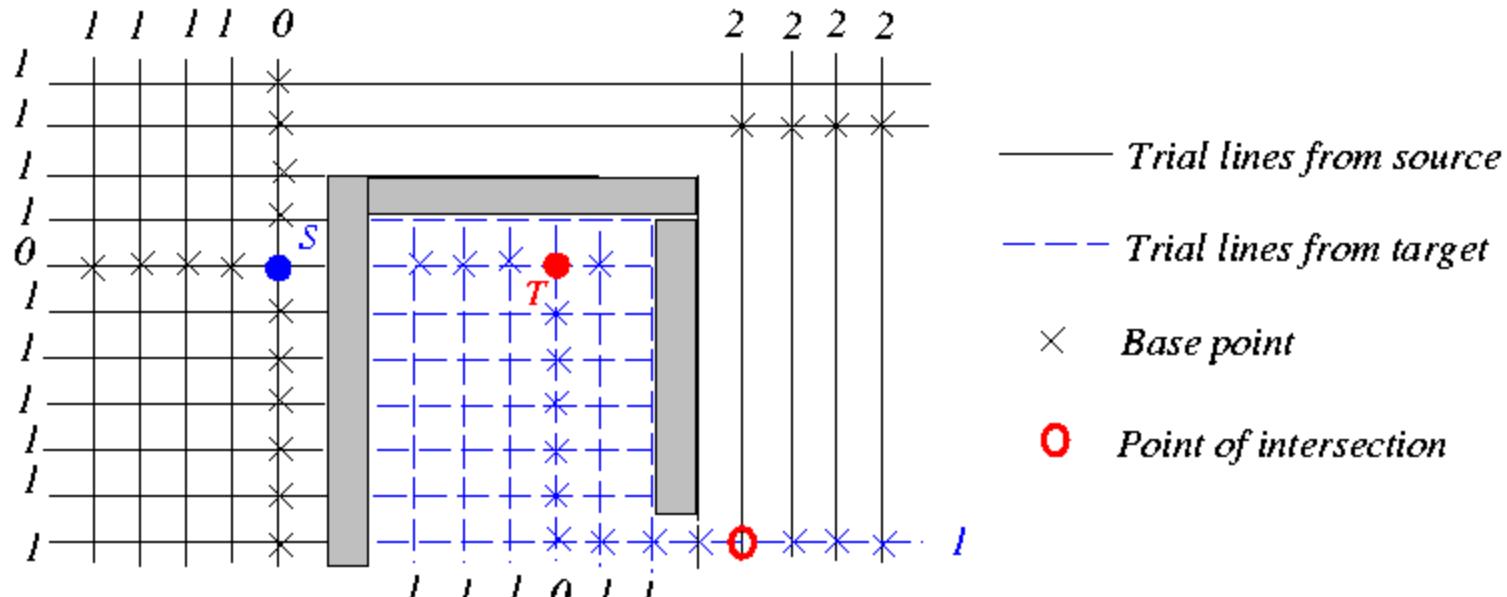
Features of Line-Search Algorithms



- Time and space complexities: $O(L)$, where L is the # of line segments generated. 一次看一條線

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.



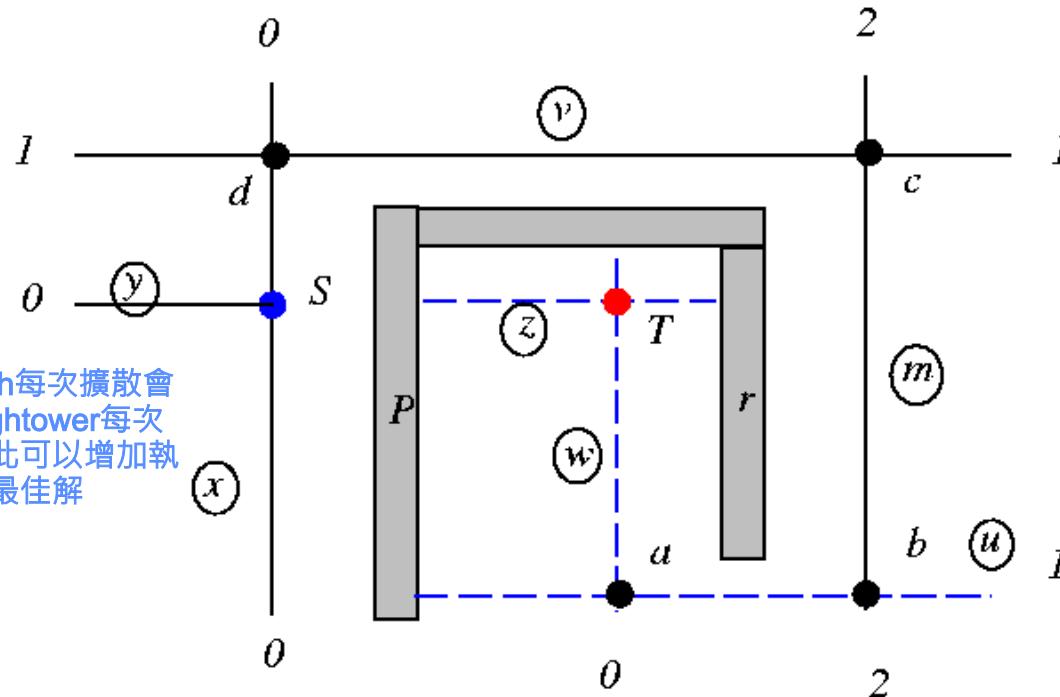
0: source的水平與鉛直線

1: 與0的線相互垂直的線

除了從source出發，也能從target出發

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

看論文的時候可以多多整理不同論文的東西，用成表格來比較

	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

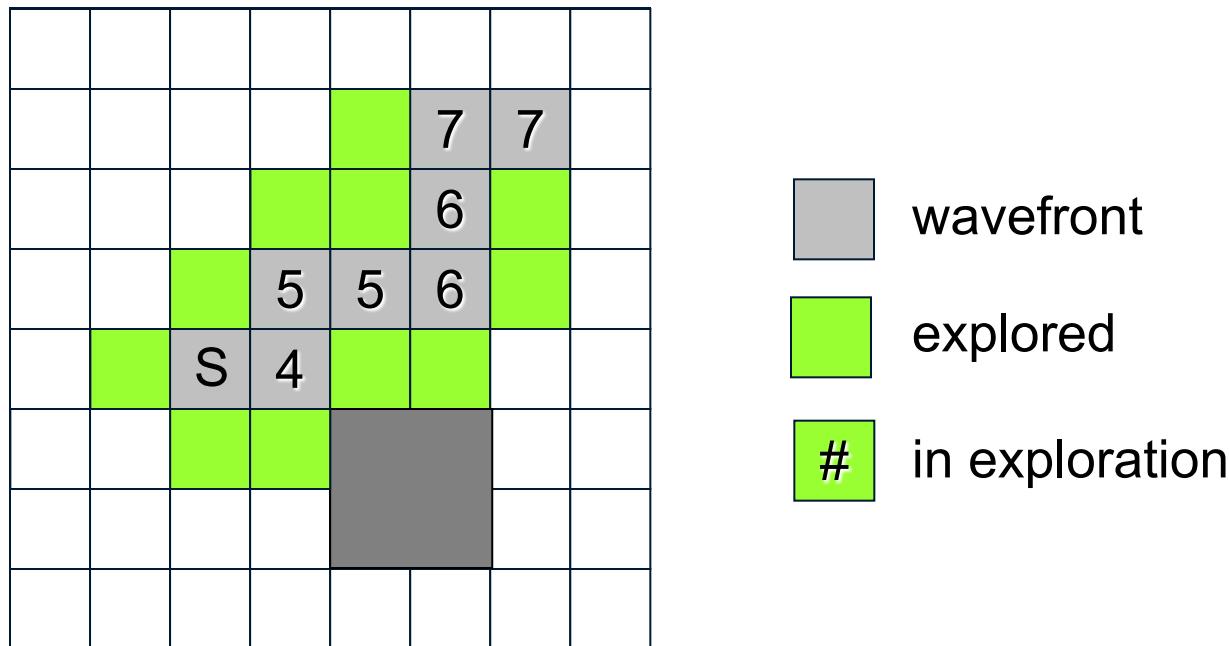
- Soukup, Mikami, and Hightower all adopt some sort of line-search operations \Rightarrow cannot guarantee shortest paths.

A*-Search Routing

- Maze search is also called **blind search** since it searches the routing region in a blind way.
 $f(x) = g(x) + h(x)$
- A*-search is also called the **best-first search**
用function來引導搜尋方向，重點就是 $h(x)$ 。如果沒有 $h(x)$ 就跟普通的blind search(lee's algorithm)一樣
 - 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
lee's algorithm
- Good property:
 - If $h(x)$ is *admissible* (never overestimates the actual cost from the current node to the target), then A*-search is optimal

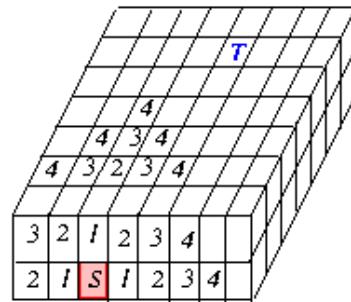
Example of A*-Search Routing

- 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)
 - $h(x)$: $\max(\text{dist}_x(T, x), \text{dist}_y(T, x))$ $h(x)$ 沒設計好可能會找不到最佳解or執行時間變長

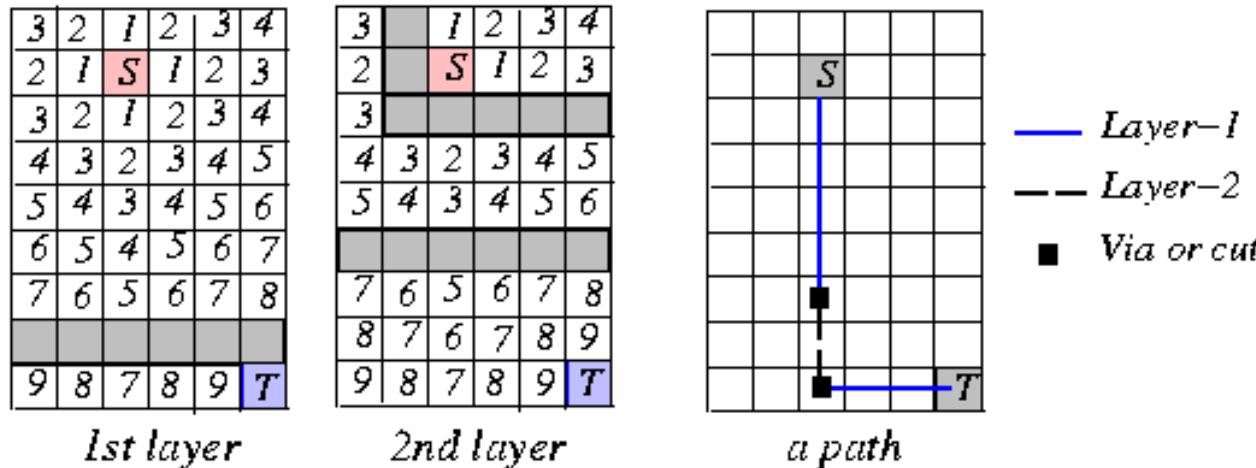


Multi-layer Routing

- 3-D grid:

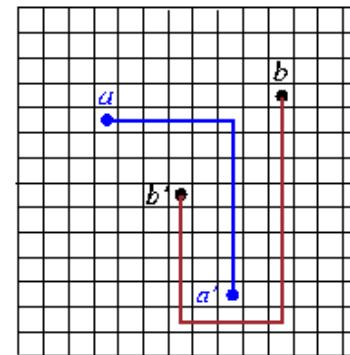
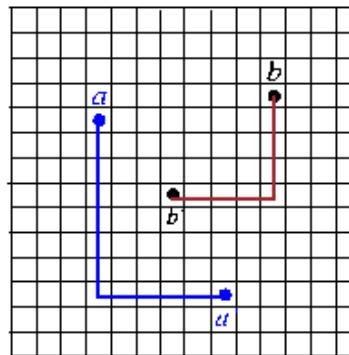
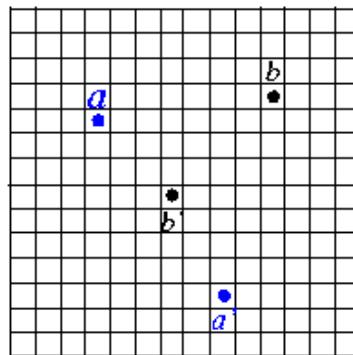


- Two planner arrays:
 - Neglect the weight for inter-layer connection through via.
 - Pins are accessible from both layers.

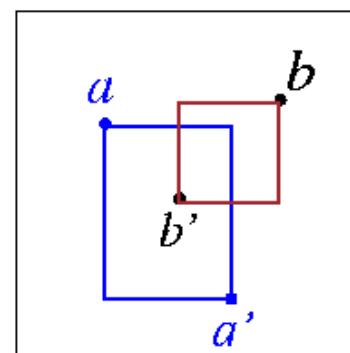
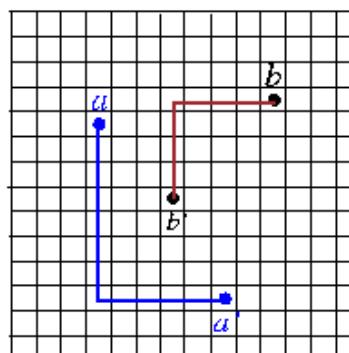
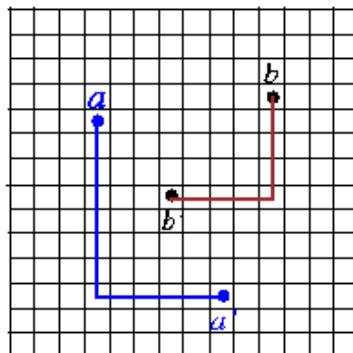


Net Ordering

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



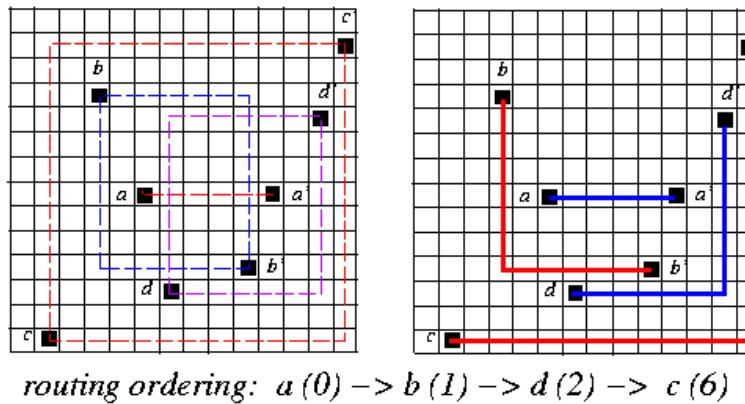
route net a before net b



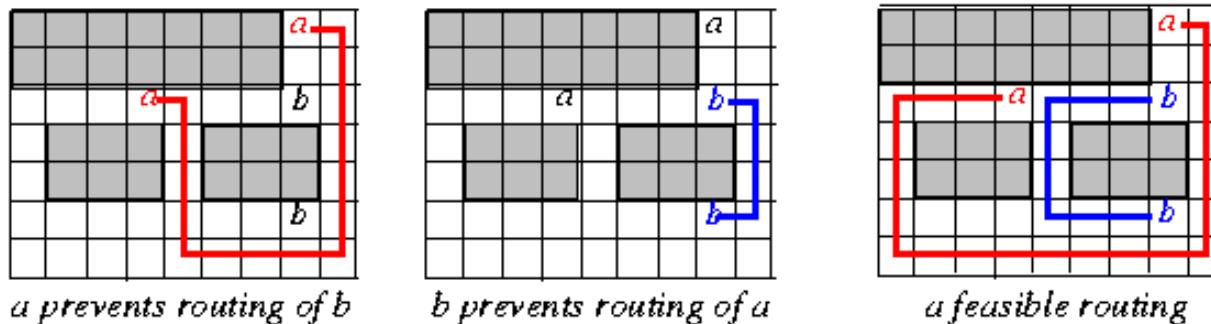
route net b before net a

Net Ordering (cont'd)

- 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.



- A mutually intervening case:

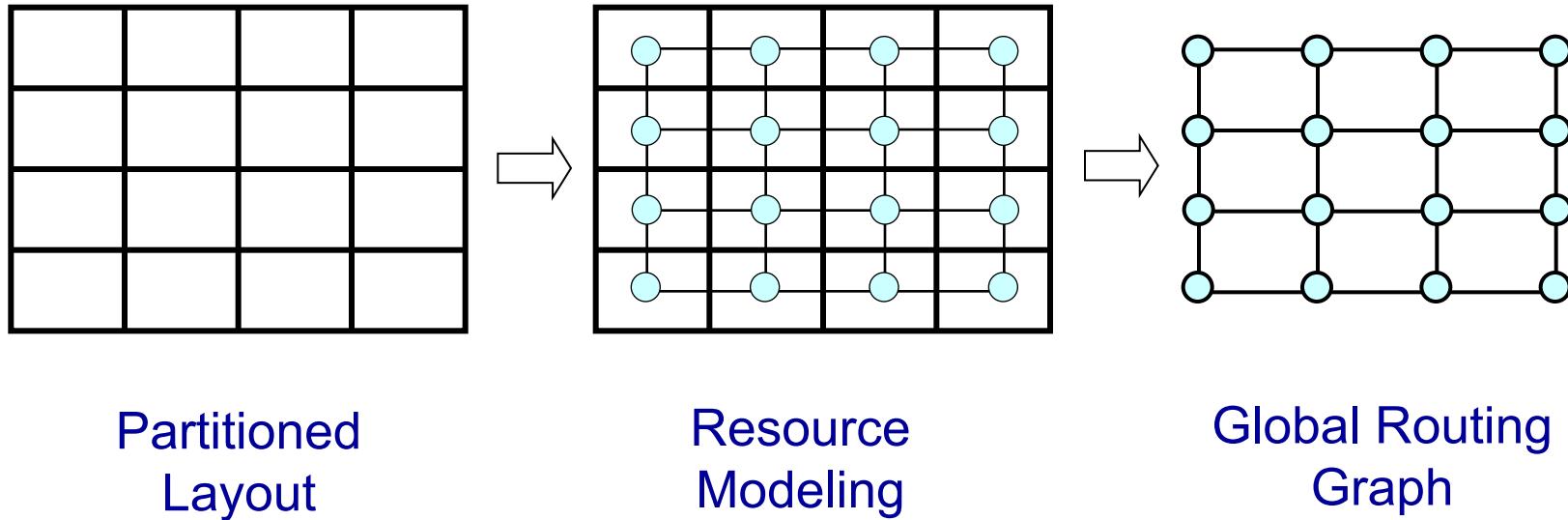


Rip-Up and Re-routing

- 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.

Graph Models: 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.



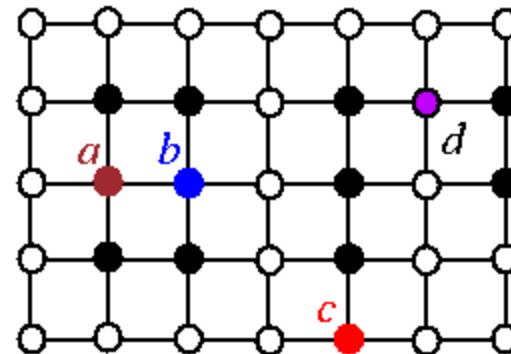
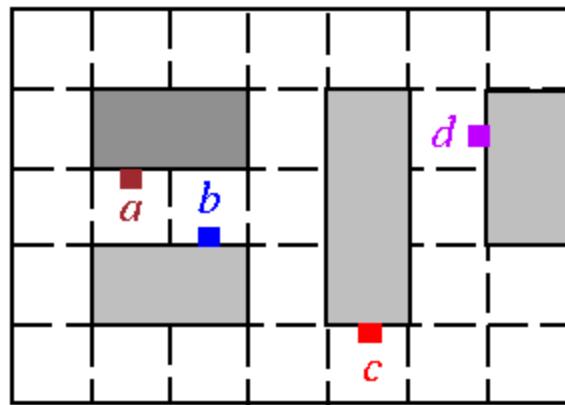
Partitioned
Layout

Resource
Modeling

Global Routing
Graph

Graph Models for Global Routing: Grid Graph

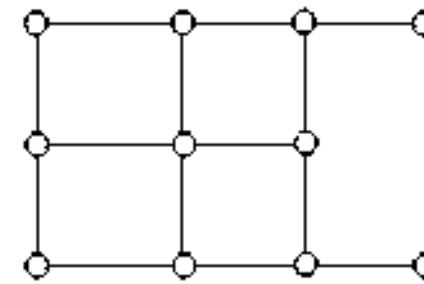
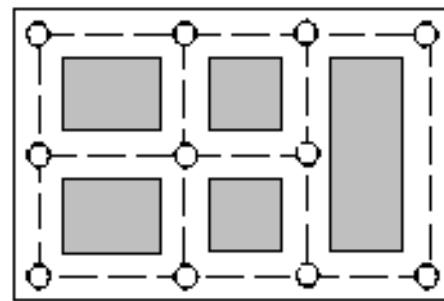
- Each cell is represented by a vertex.
- Two vertices are joined by an edge if the corresponding cells are adjacent to each other.
- The occupied cells are represented as filled circles, whereas the others are as clear circles.



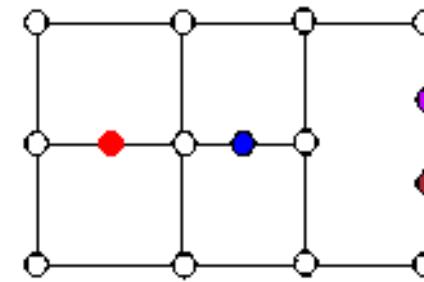
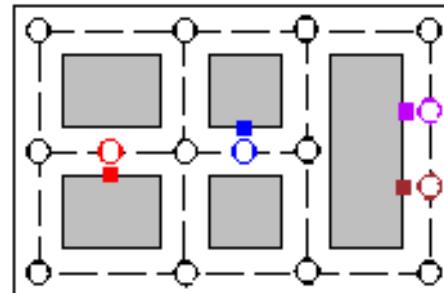
Graph Model: Channel Intersection Graph

- 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.

channel
intersection
graph



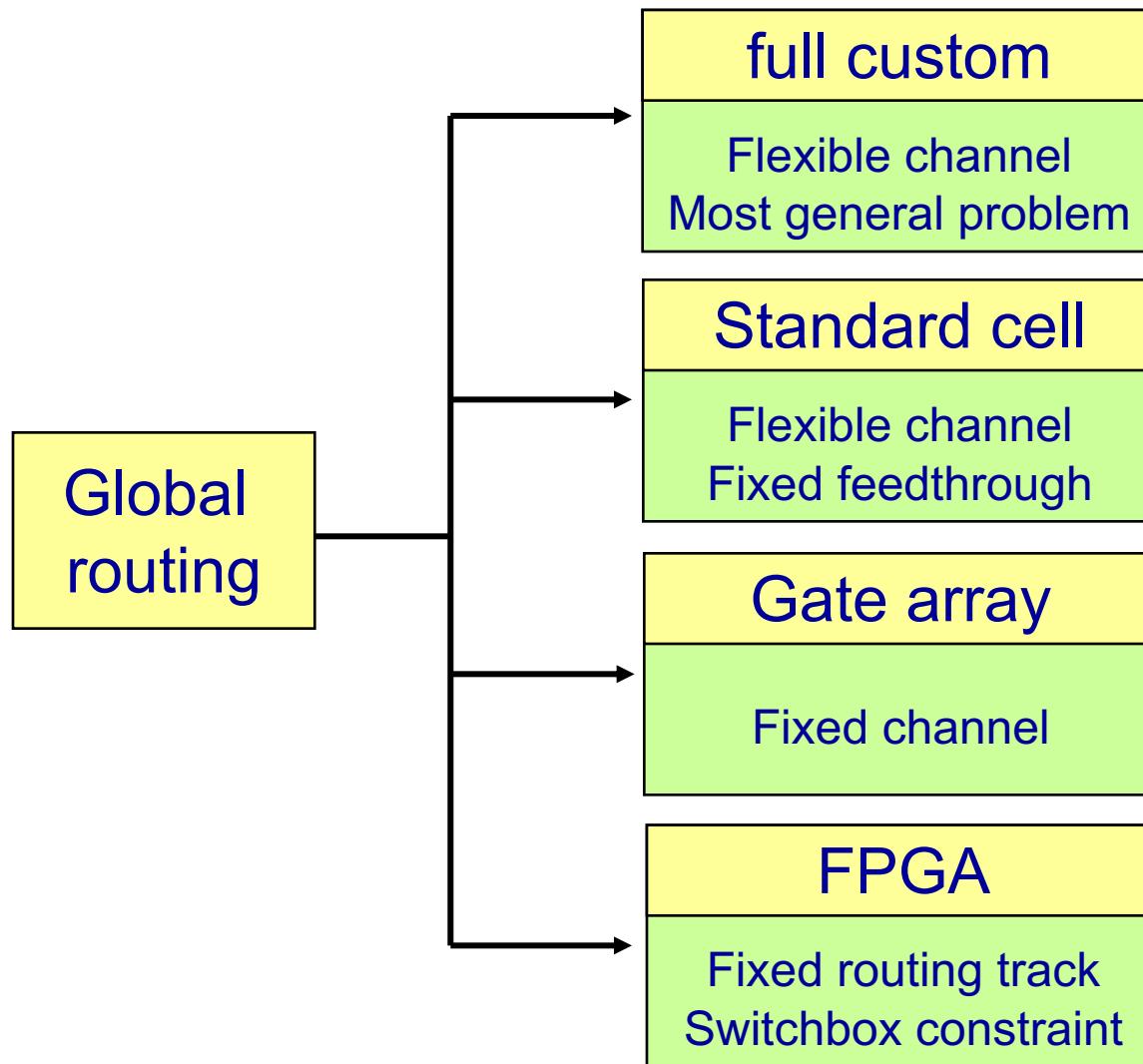
extended
channel
intersection
graph



Global-Routing Problem

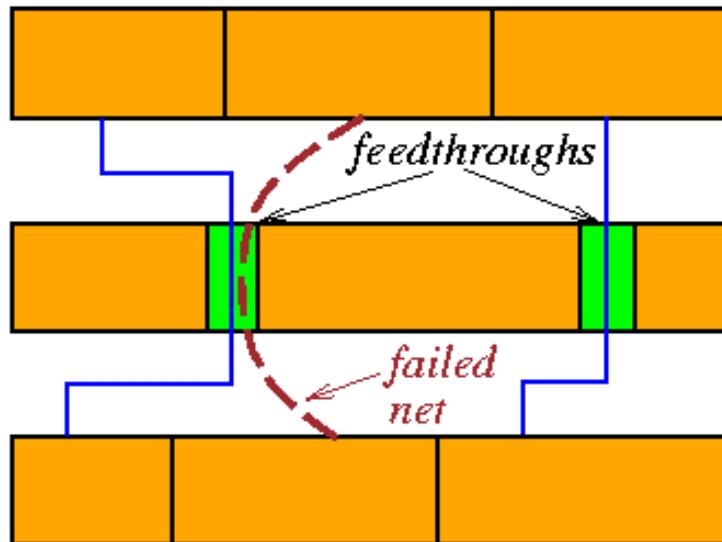
- Given a netlist $N = \{N_1, N_2, \dots, N_n\}$, a routing graph $G = (V, E)$, find a Steiner tree T_i for each net N_i , $1 \leq i \leq n$, such that $U(e_j) \leq c(e_j)$, $\forall e_j \in E$ and $\sum_{i=1}^n L(T_i)$ is minimized, where
 - $c(e_j)$: capacity of edge e_j
 - $x_{ij}=1$ if e_j is in 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



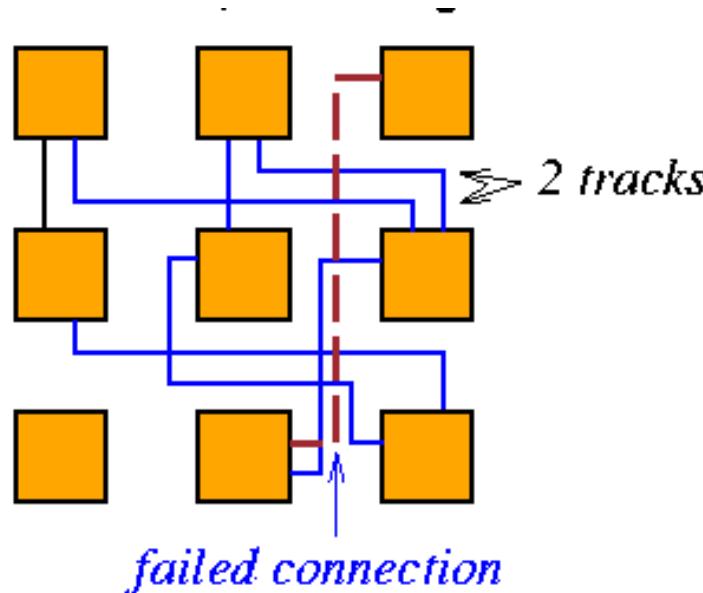
Global Routing in Standard Cell

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



Global Routing in Gate Array

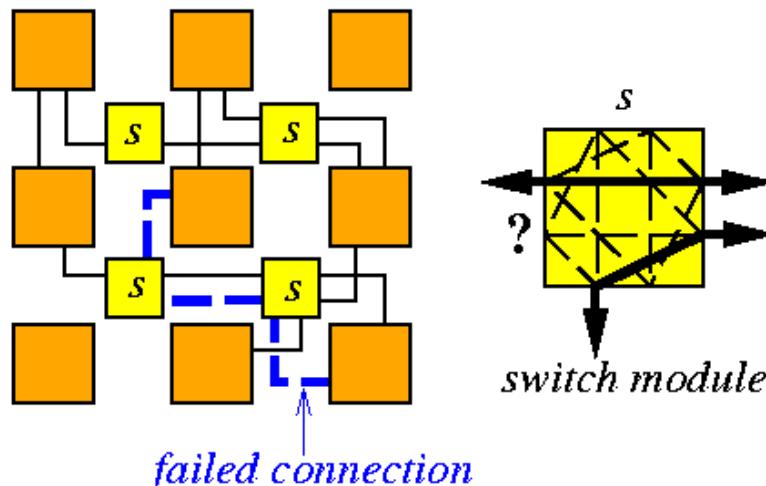
- Objective
 - **Guarantee 100% routability.**
- For high performance,
 - Minimize the maximum wire length.
 - Minimize the maximum path length.



Each channel has a capacity of 2 tracks.

Global Routing in FPGA

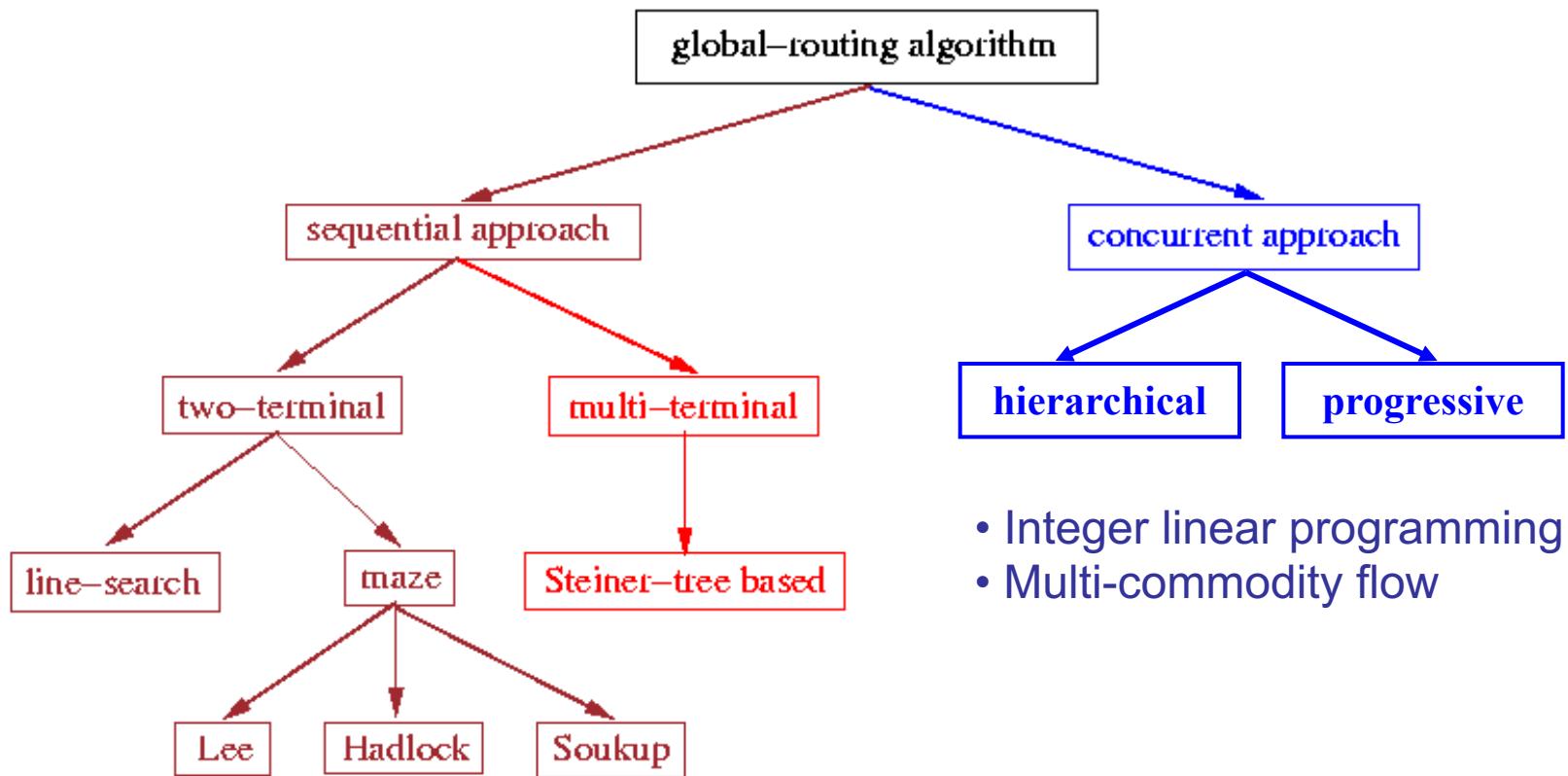
- Objective
 - Guarantee 100% routability.
 - Consider **switch-module architectural constraints**.
- For performance-driven routing,
 - **Minimize # of switches used.**
 - Minimize the maximum wire length.
 - Minimize the maximum path length.



Each channel has a capacity of 2 tracks.

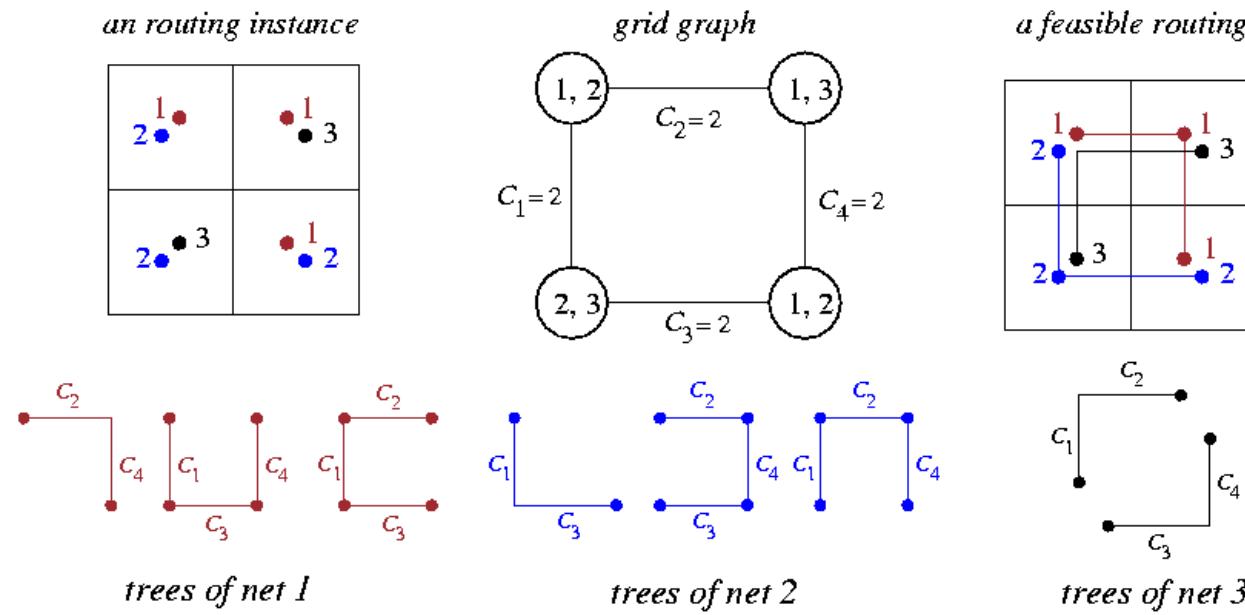
Classification of Global-Routing Algorithm

- **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?)



Global Routing by Integer Programming

- Suppose that for each net i , there are n_i possible trees $t_1^i, t_2^i, \dots, t_{n_i}^i$ 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?
 - Key:** Hierarchical approach!



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

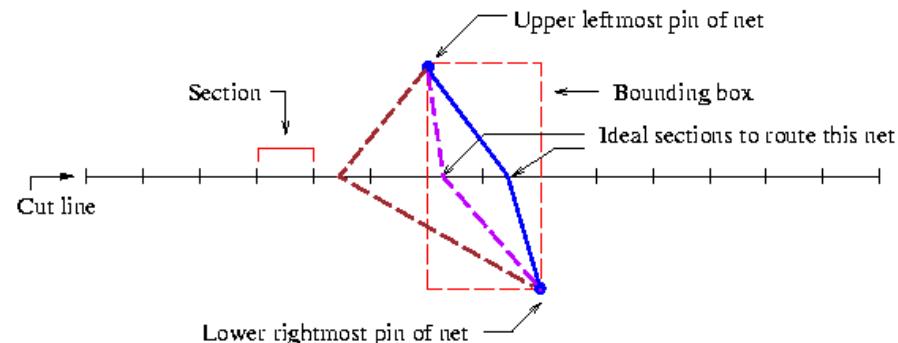
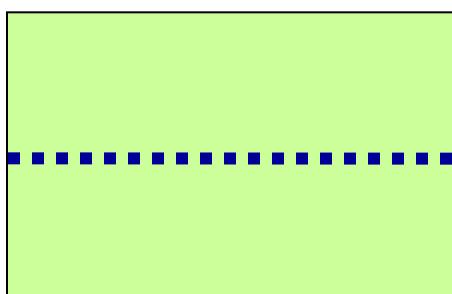
- $g_{i,j}$: cost of tree $t_j^i \Rightarrow g_{1,1}=2, g_{1,2}=3, g_{1,3}=3, g_{2,1}=2, g_{2,2}=3, g_{2,3}=3, g_{3,1}=2, g_{3,2}=2$.

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

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

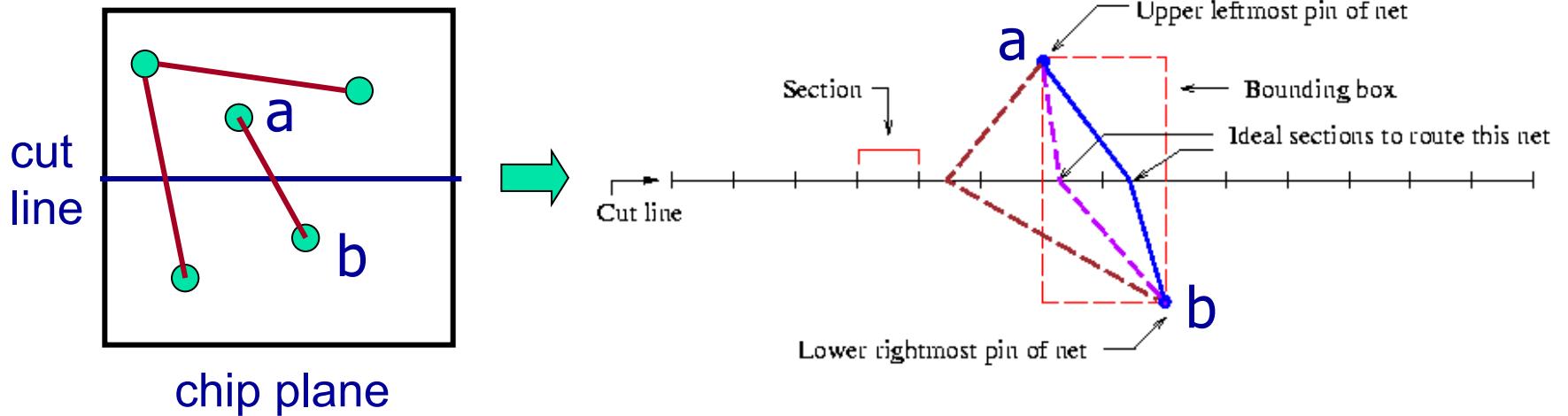
Hierarchical Global Routing

- 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_i be the # of grid regions of a given cut line i ; a cut line can be divided into $M = R_i / C$ 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: $\sum_{j=1}^M x_{ij} = 1, 1 \leq i \leq N$.
 - Capacity constraint of each section: $\sum_{i=1}^N x_{ij} \leq C, 1 \leq j \leq M$.
 - 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: $\sum_{j=1}^M x_{ij} = 1, 1 \leq i \leq N$.
 - Capacity constraint of each section: $\sum_{i=1}^N x_{ij} \leq C, 1 \leq j \leq M$.
 - w_{ij} : cost of assigning net i to section j . Minimize $\sum_{i=1}^N \sum_{j=1}^M w_{ij} x_{ij}$.



BoxRouter: Progressive Global Routing

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

- Slides modified from the DAC presentation

1. Incremental box expansion

- From the most congested region (擒賊先擒王)

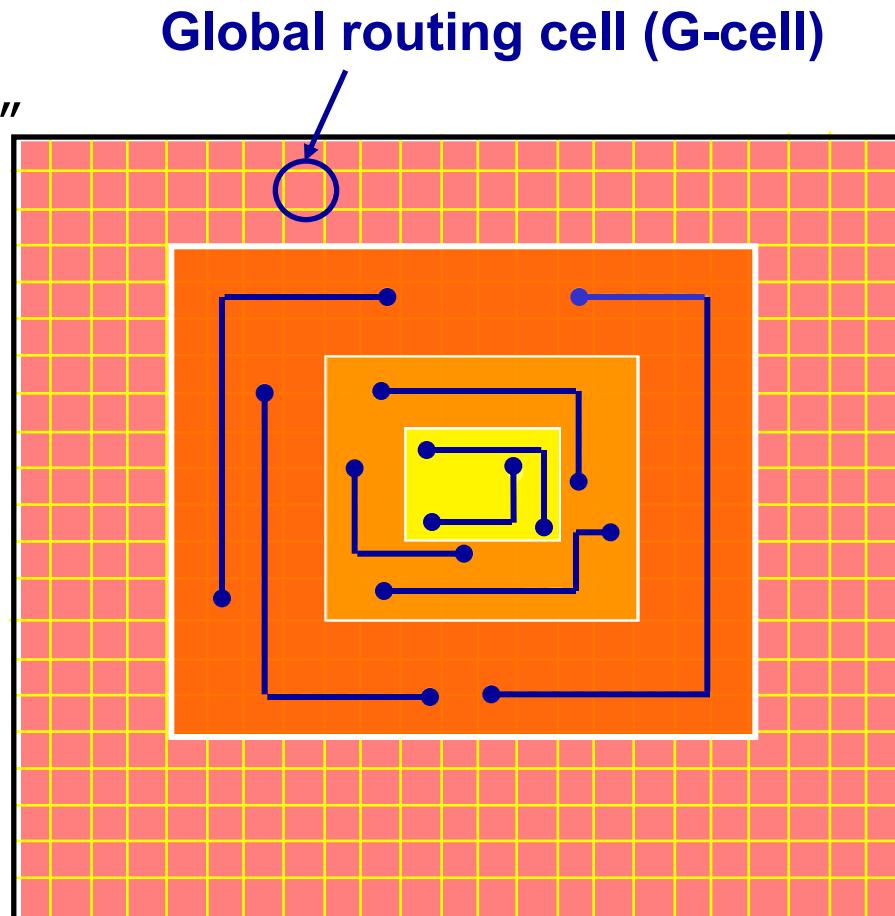
2. Progressive ILP

- With adaptive maze routing

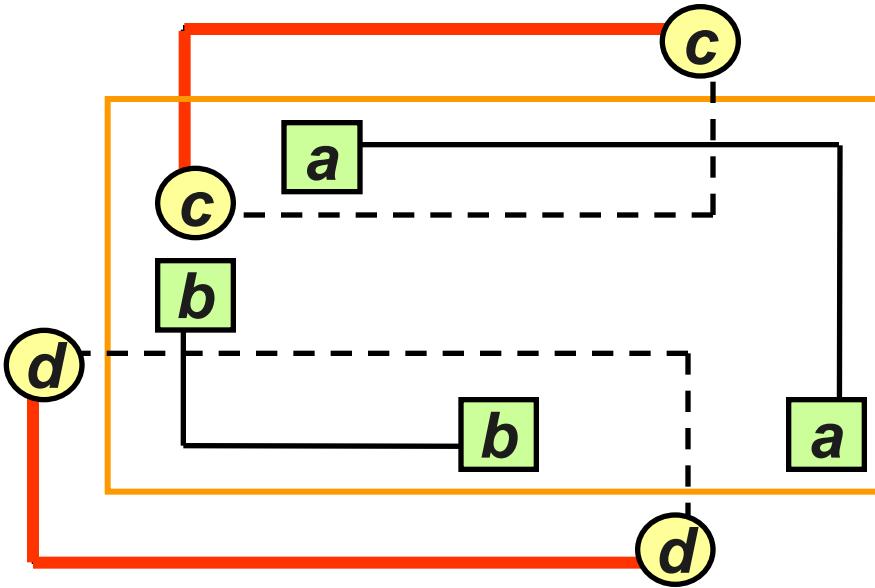
3. Post routing

- Rerouting w/o rip-up

- Try to minimize the total routing overflow

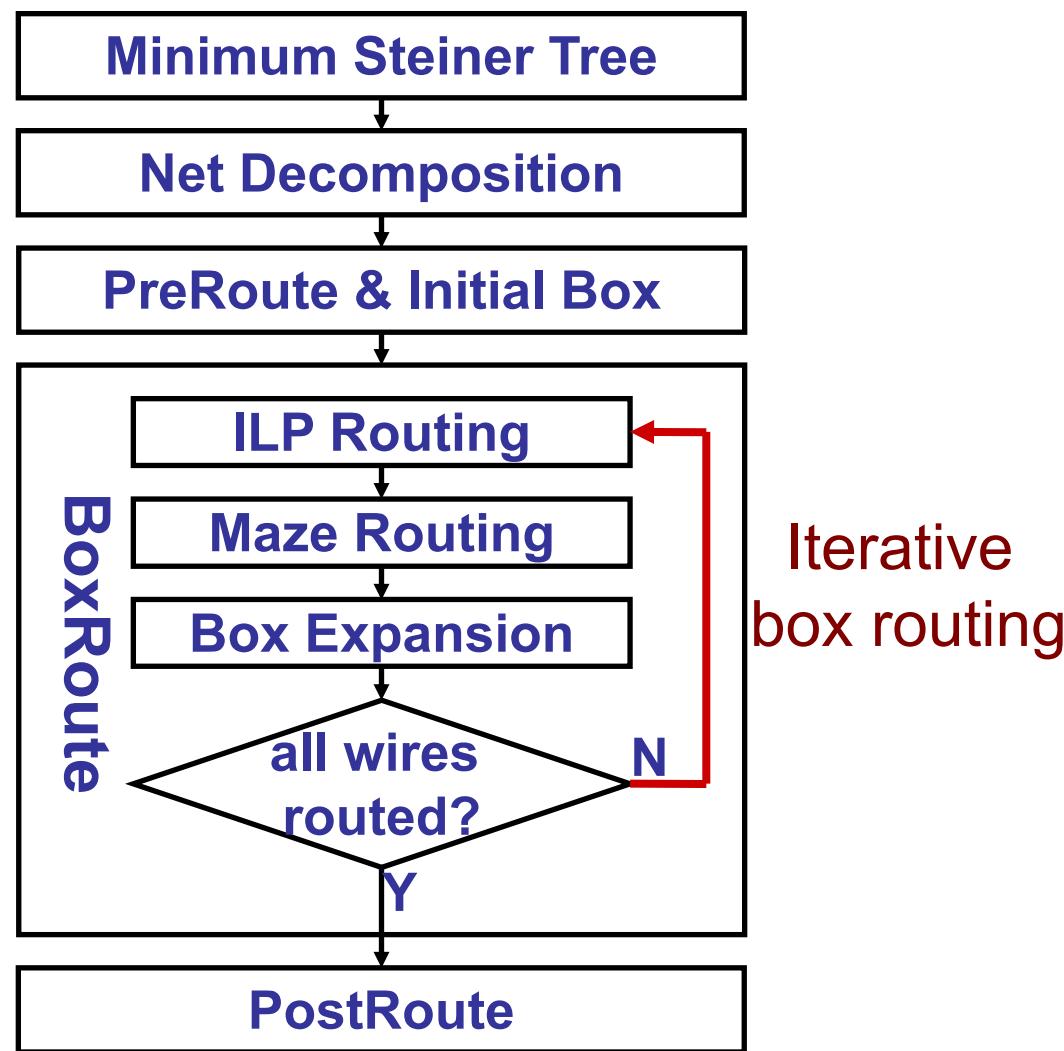


Motivation for Box Expansion

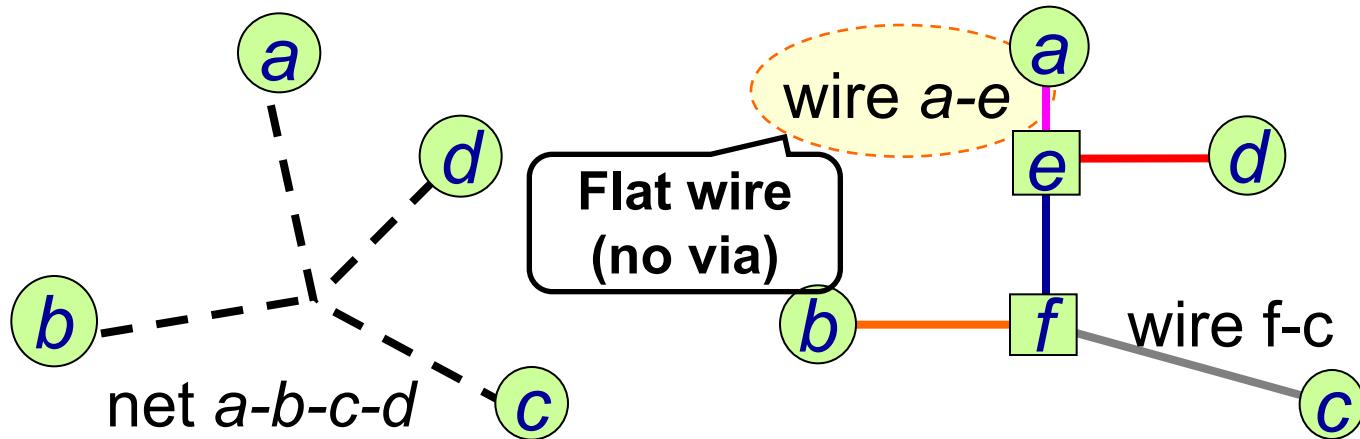


- Different resource management for wires inside/outside of a box
- Box expansion **pushes/diffuses** congestion outwards progressively

Overall Flow of BoxRouter



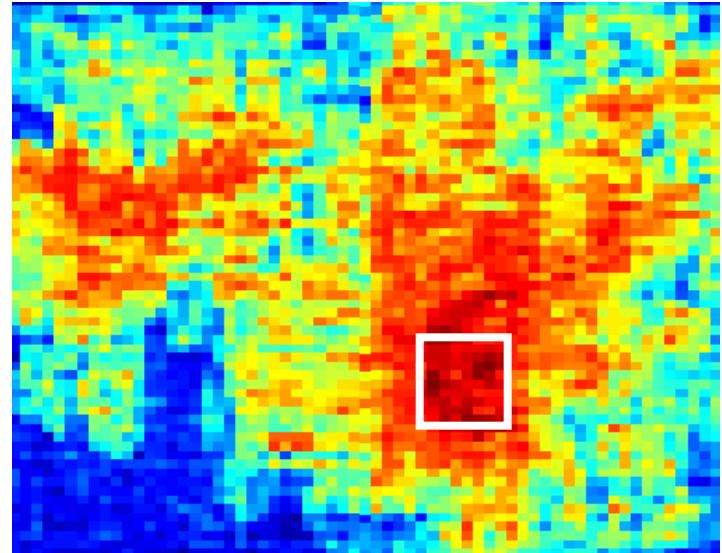
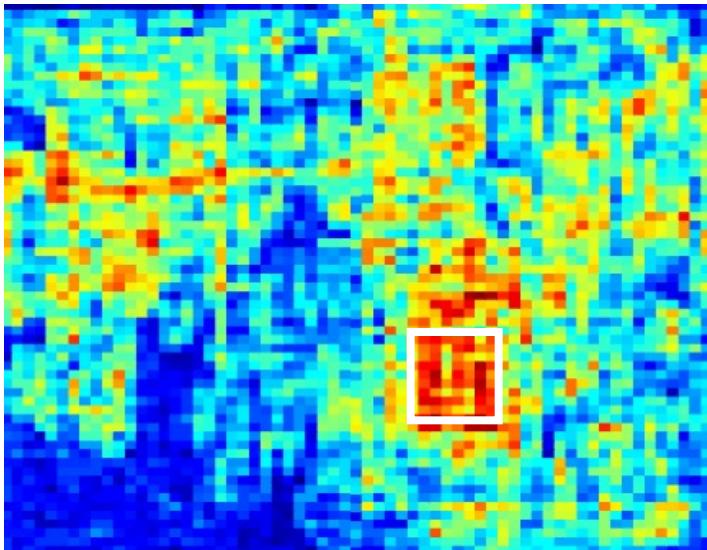
Minimal Steiner Tree & Net Decomposition



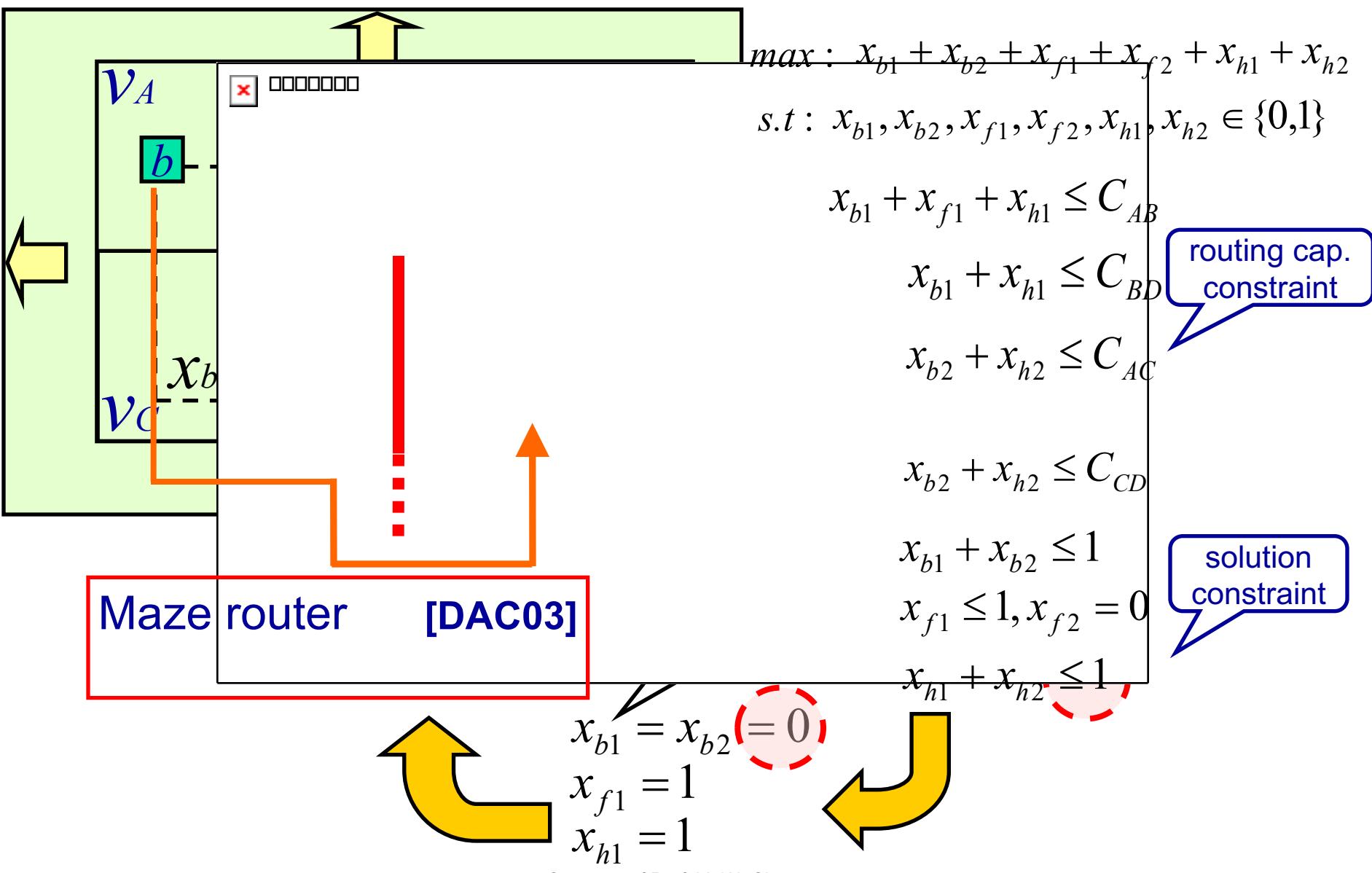
- Minimum Steiner Tree
 - FLUTE [ICCAD-04], Spanning graph [ISPD-03], Geosteiner 3.1
 - Decompose into 2-pin wires
 - Each wire becomes an atomic routing object

PreRouting

- Preroute as many flat wires as possible
 - 60% of wirelength can be prerouted
- Benefits
 - Overall congestion captured
 - Runtime improved
- Starting “Box” of BoxRouting

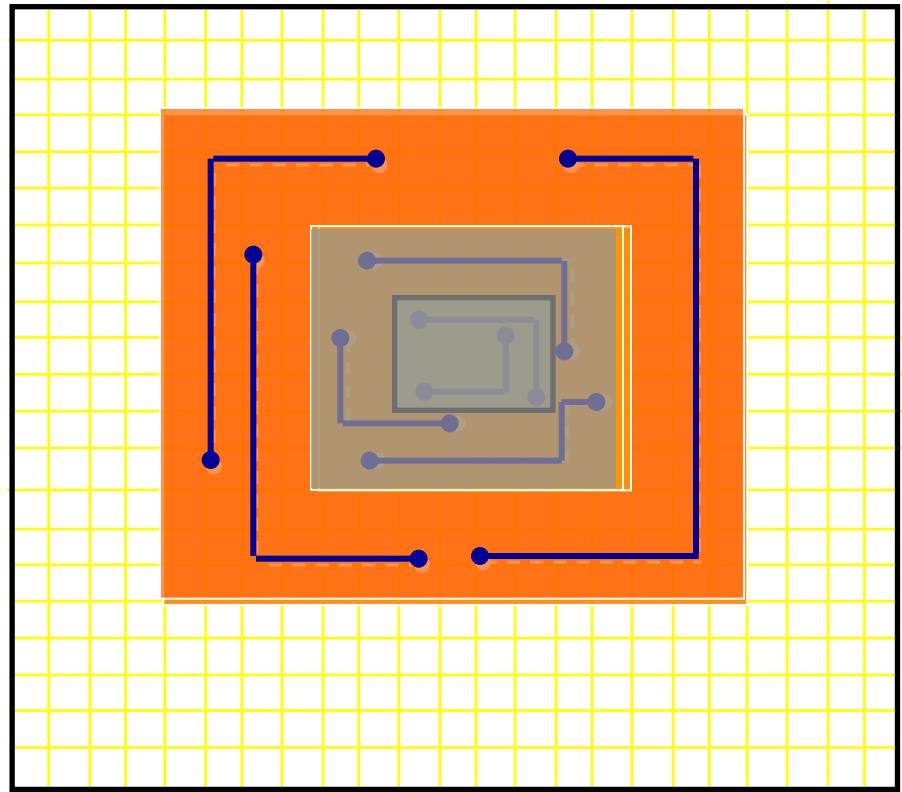


BoxRouting in Action



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



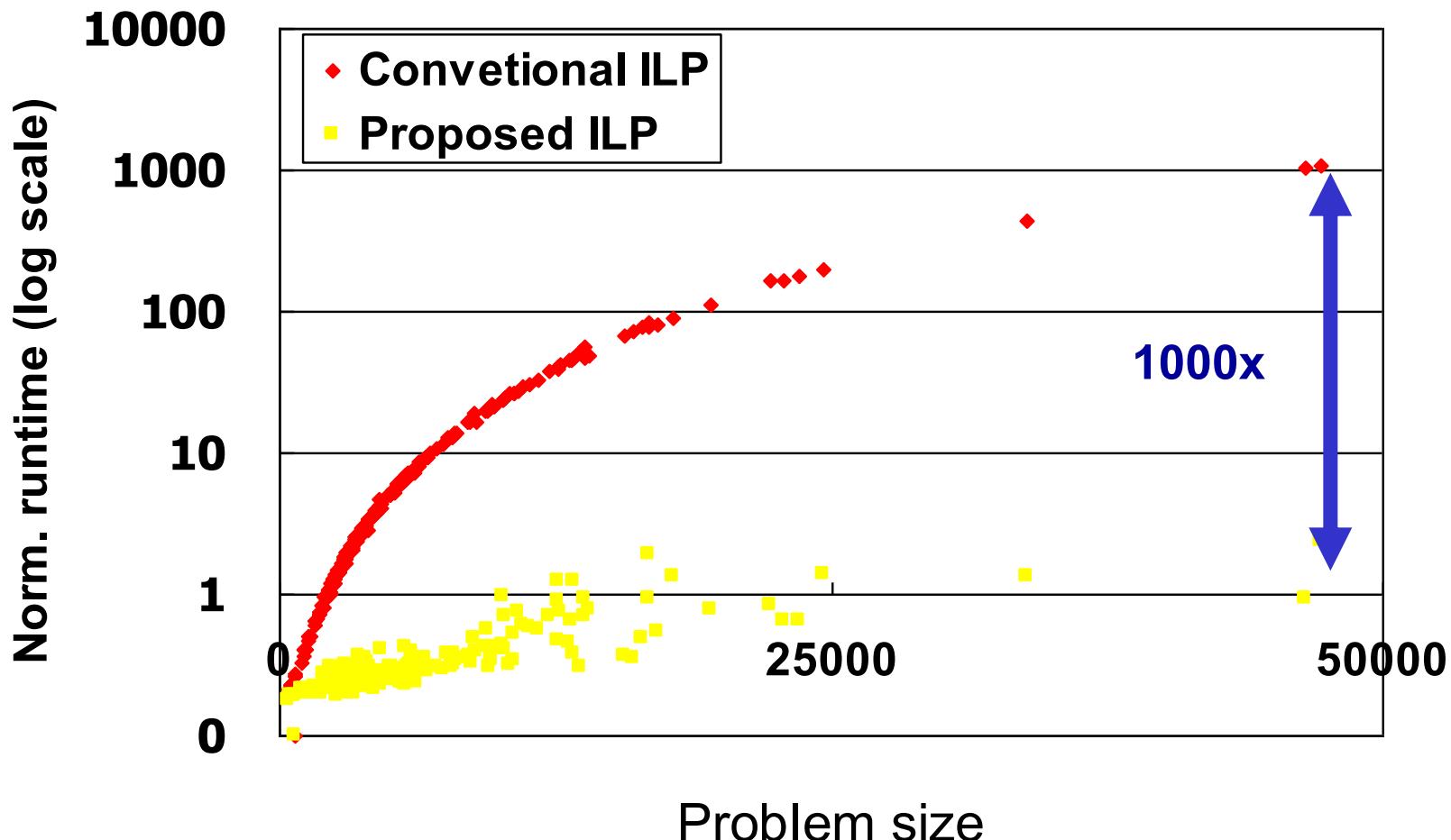
Proposed ILP vs. Conventional ILP

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

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

- Proposed ILP formulation
 - Maximize # routed nets (min. total overflow)
 - Work together with maze router
- Conventional ILP formulation
 - Minimize max overflow [ICCAD-01]

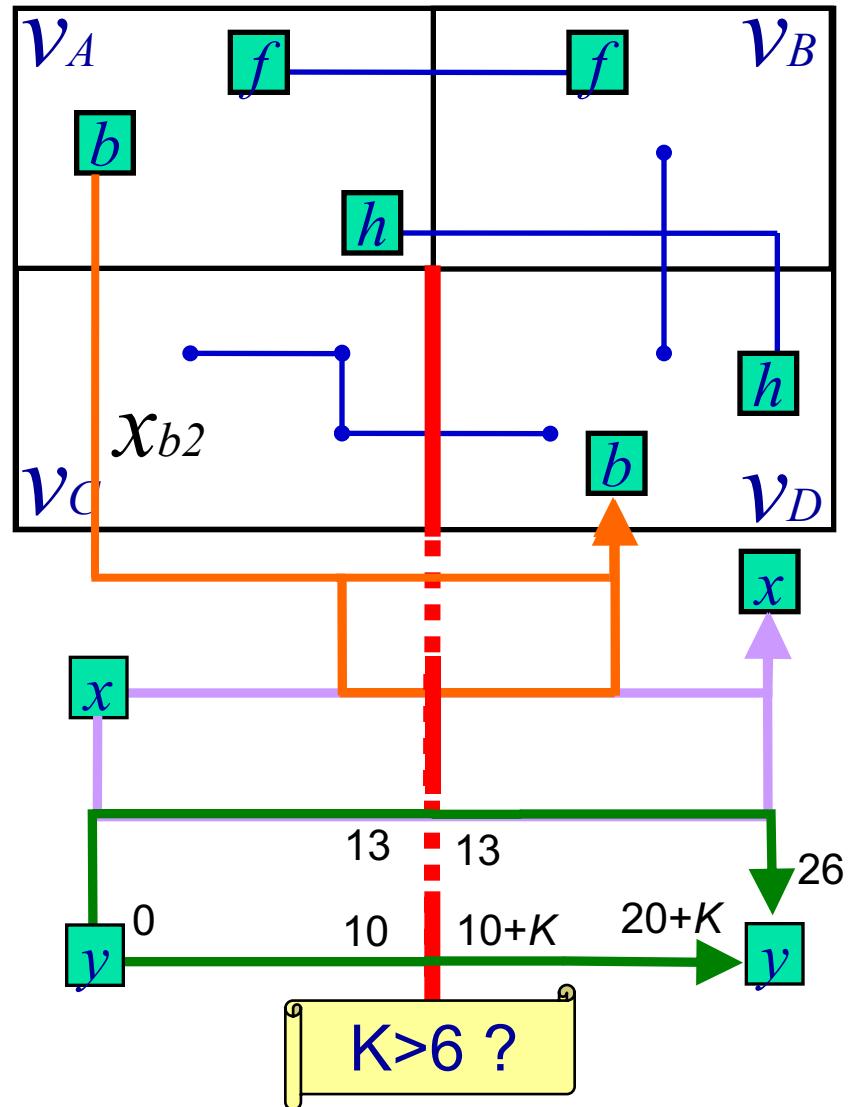
Proposed ILP vs. Conventional ILP



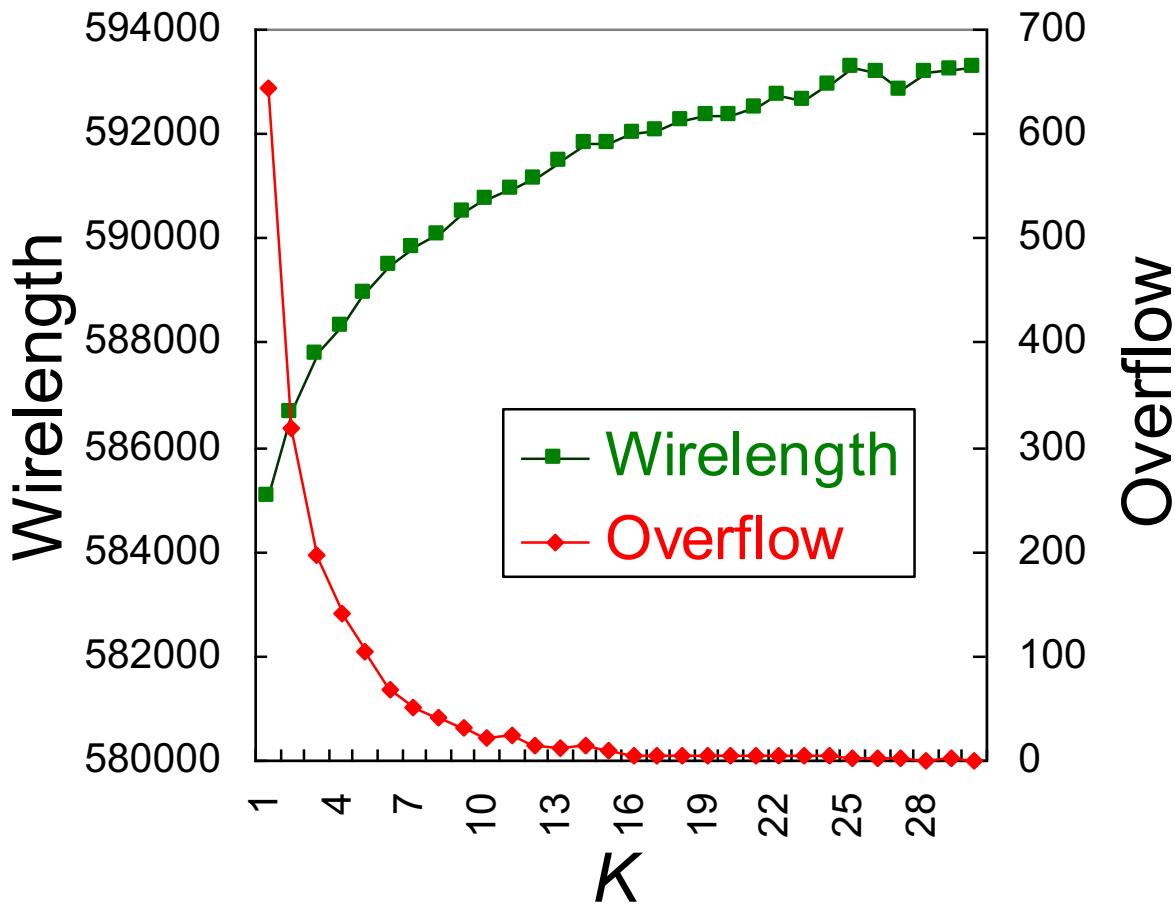
GNU Linear Programming Kit 4.10, Industrial circuit (7mm x 7mm)

PostRouting

- Reroute w/o rip-up other nets
 - In the same order as box expansion
 - Enhance routing path
- Smooth tradeoff
 - Between wirelength and congestion
 - Controlled by a parameter K



PostRoute



- Smooth tradeoff: $K \uparrow \Rightarrow \text{overflow} \downarrow, \text{wirelength} \uparrow$

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

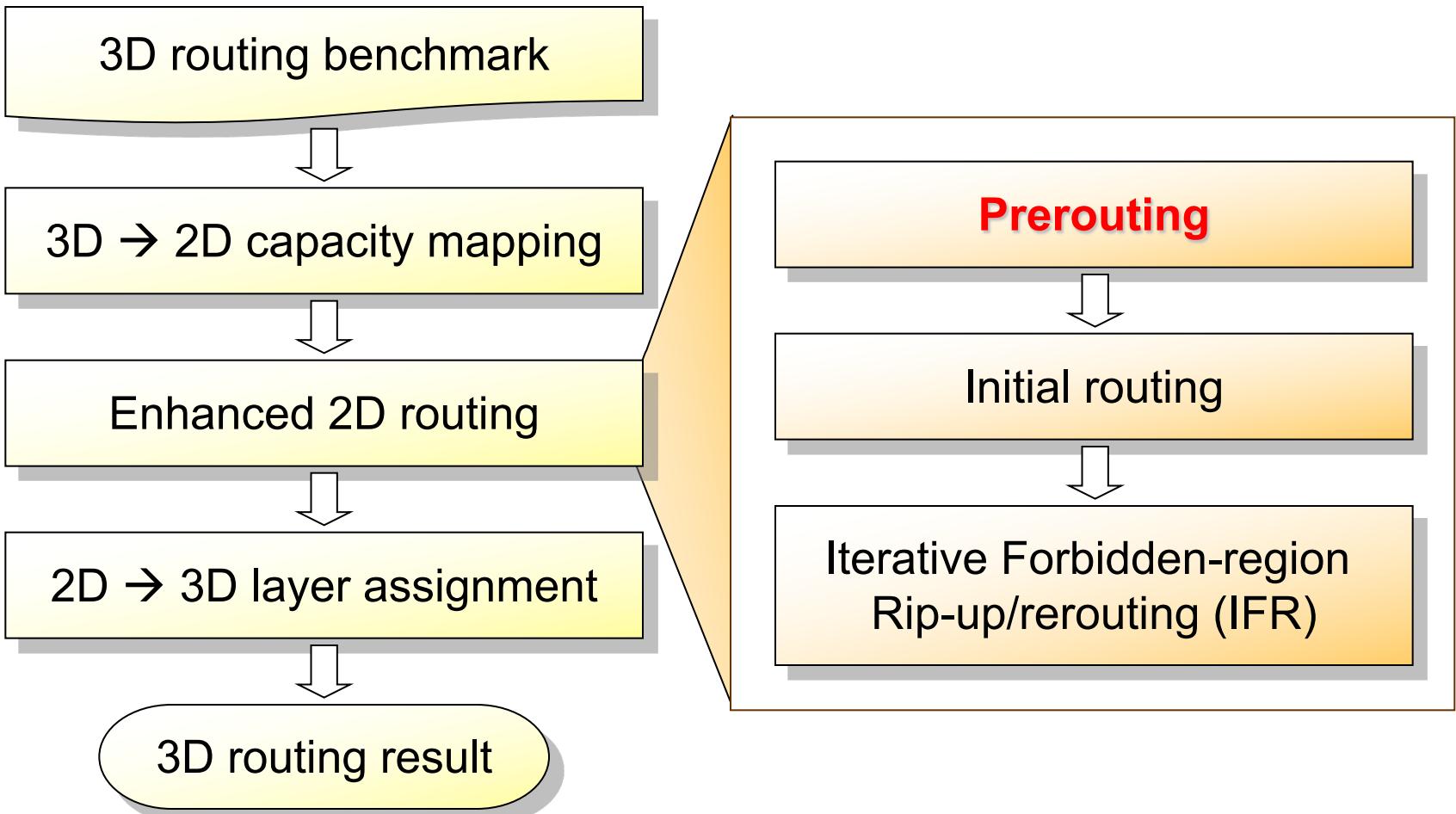
- Proposed in **PathFinder** [McMurchie and Ebeling, FPGA'95] to reduce overflows
- Spreads the congested wires iteratively
- At the (i)-th iteration, the cost of a global edge e:

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

- b_e : base cost of using e,
- p_e : # of nets passing e,
- $h_e^{(i)}$: historical cost on e,
$$h_e^{(i)} = \begin{cases} h_e^{(i-1)} + 1 & \text{if } e \text{ has overflow} \\ h_e^{(i-1)} & \text{otherwise} \end{cases}$$
- Also used by recent academic global routers, Archer [ICCAD'07], BoxRouter [ICCAD'07], FastRoute [ICCAD'06, ASPDAC'07, TCAD'08], FGR [ICCAD'07], NTHU-Route [ASPDAC'08, ICCAD'08]
- INR may get stuck as the number of iterations increases [Ozdal, ICCAD'07] [Gao et al., ASPDAC'08]

NTUgr Global Routing Flow

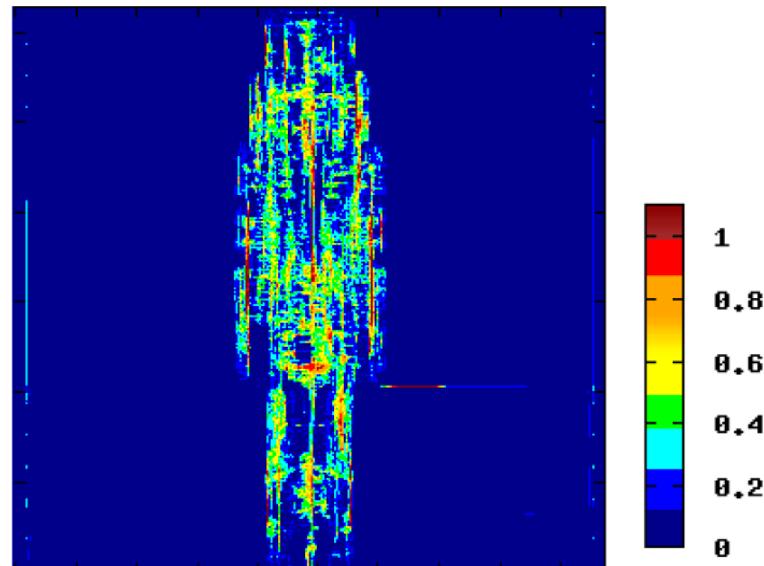
- Hsu, Chen & Chang, ICCAD-08, ASP-DAC-09, TCAD-10



Prerouting

1. Congestion-hotspot historical cost pre-increment
 - Identify the high-pin-density tiles (#pin exceeds total tile capacity)
 - Increase the historical cost lying around these tiles by 10
 - To avoid other nets passing through these congested tiles
2. Small bounding-box area routing
 - Route the less-flexibility nets with smaller bounding-box area

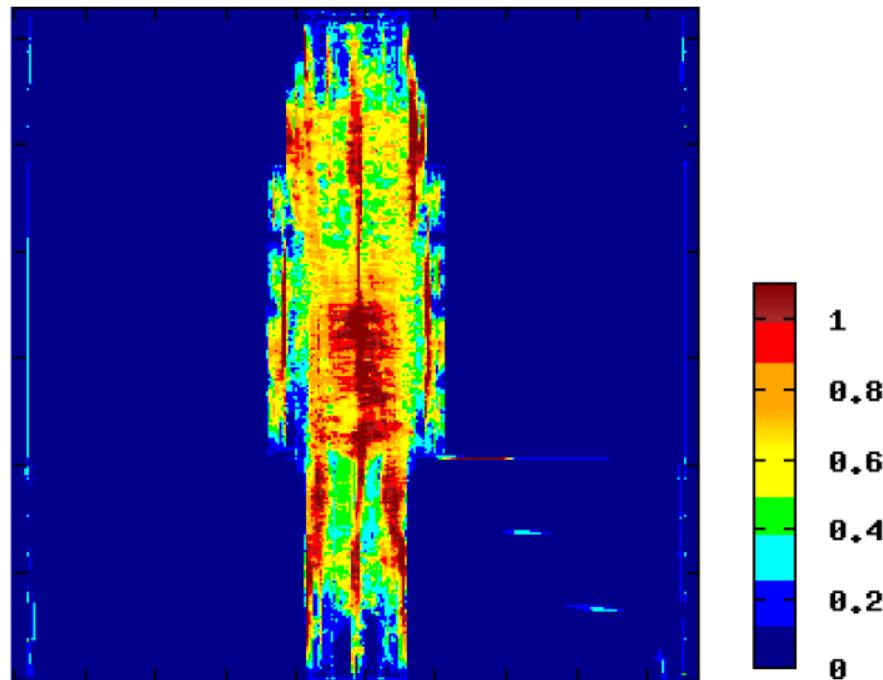
Prerouting of newblue3
(49.22% routed nets,
74374 overflows)



Initial Routing

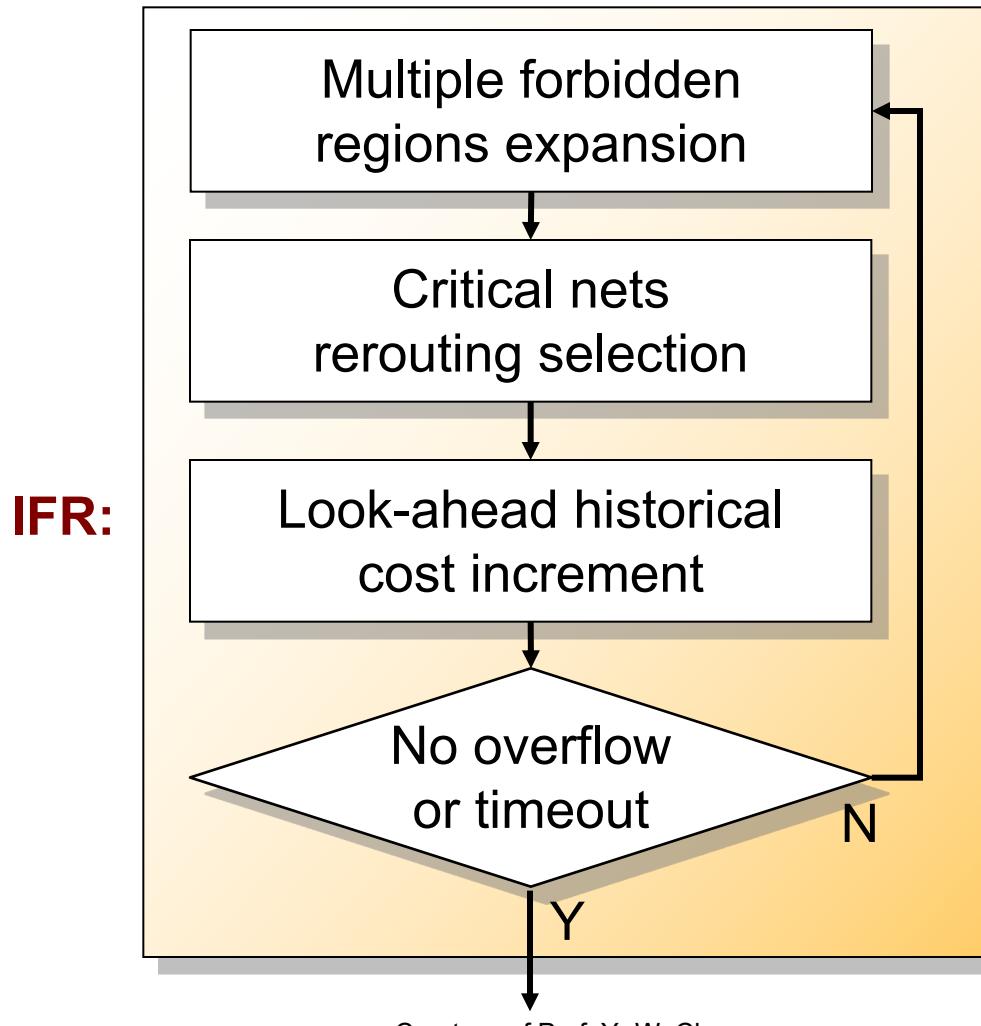
- The first stage completing all nets in the whole chip
- Apply **iterative monotonic routing** until the overflow improvement is less than 5% of the previous iteration

Initial routing of newblue3
(100% routed nets,
306082 overflows)



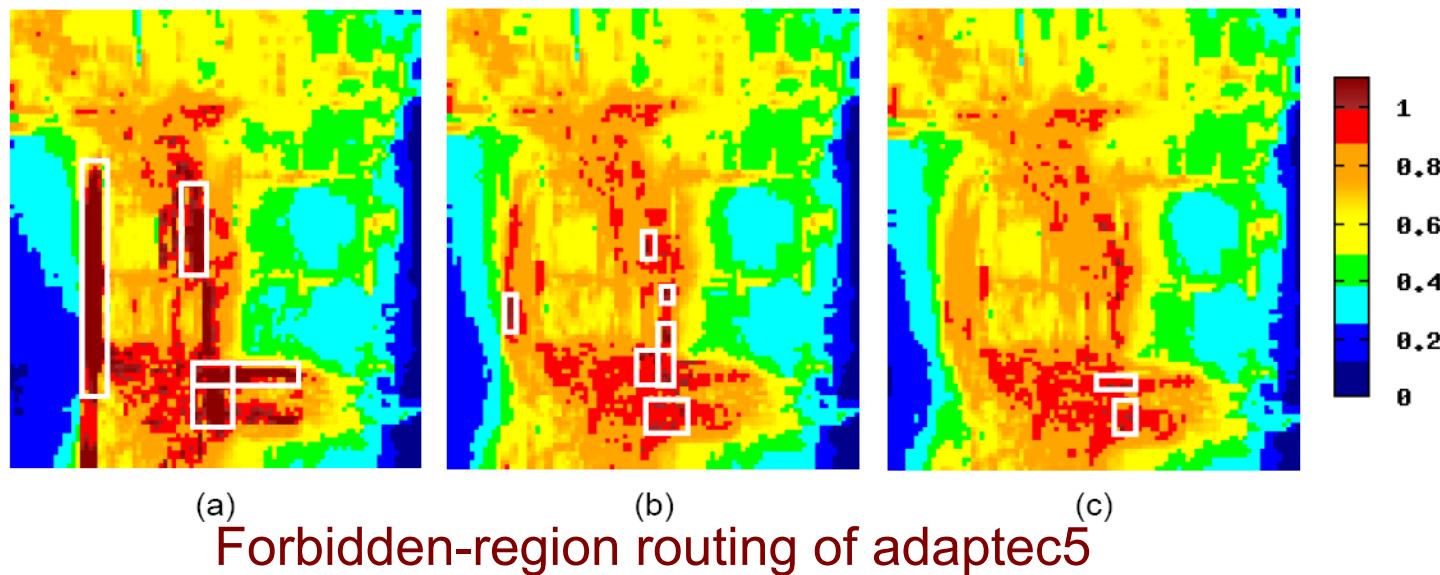
Iterative Forbidden-region Rip-up/rerouting (IFR)

- An enhanced flow over the traditional INR
- Perform iteratively until no overflow or timeout



Multiple Forbidden-Regions Construction

- At each iteration of IFR, new **forbidden regions** are constructed from the most congested regions
 - Initially contains two adjacent tiles w.r.t. the most congested edge
 - Expand the region until the average congestion of each boundary is smaller than a threshold (overlap is allowed)
- Apply a special cost metric for nets in forbidden regions
 - Introducing new overflows within these regions is almost forbidden by incurring a large penalty



Cost Considering Forbidden Regions

- The cost function of a global edge e :

$$\text{cost}(e) = \begin{cases} P_n \cdot (d_e / c_e) & \text{if } e \in \text{forbidden regions and is congested} \\ b_e \cdot (1 + h_e^{(i)}) & \text{otherwise} \end{cases}$$

P_n : forbidden region penalty ($=1000$ in NTUgr)

d_e : routing demand of e

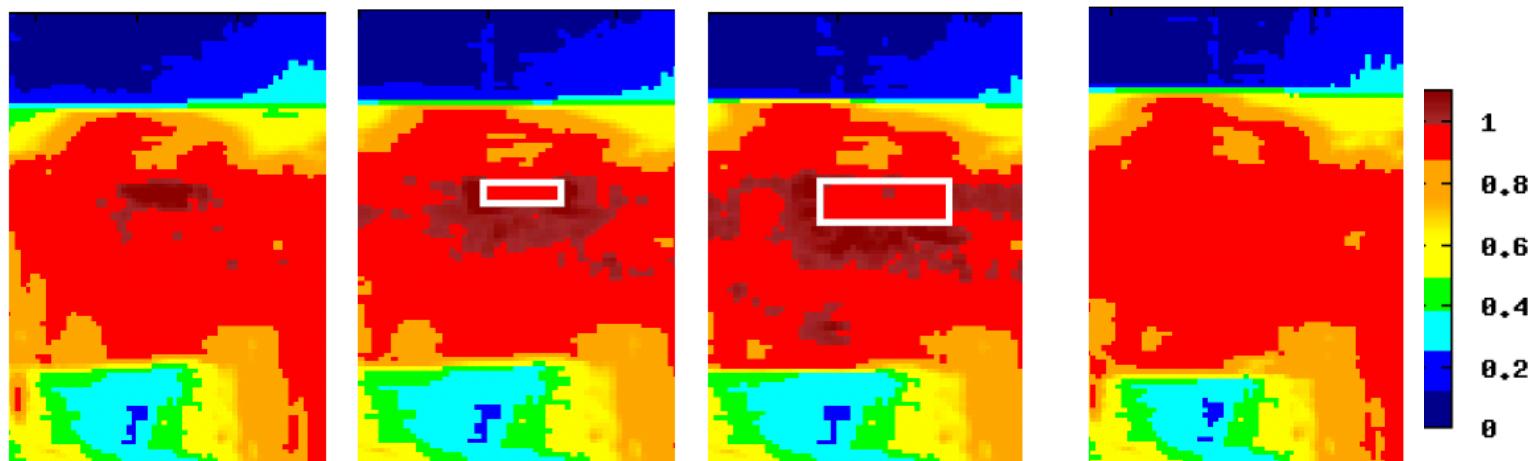
c_e : routing capacity of e

$h_e^{(i)}$: historical cost of e

$$b_e = \begin{cases} 1/(c_e - d_e) & \text{if } d_e < c_e \\ \xi & \text{if } d_e = c_e \quad (\text{penalized base cost of } e) \\ \xi + d_e/c_e & \text{if } d_e > c_e \quad (\xi = 3 \text{ in NTUgr}) \end{cases}$$

Region Propagation Leveling

- Applied when # of overflows stops decreasing (get stuck at the local optima)
 - Stop creating new forbidden regions
 - Expand all forbidden regions at the previous iteration simultaneously

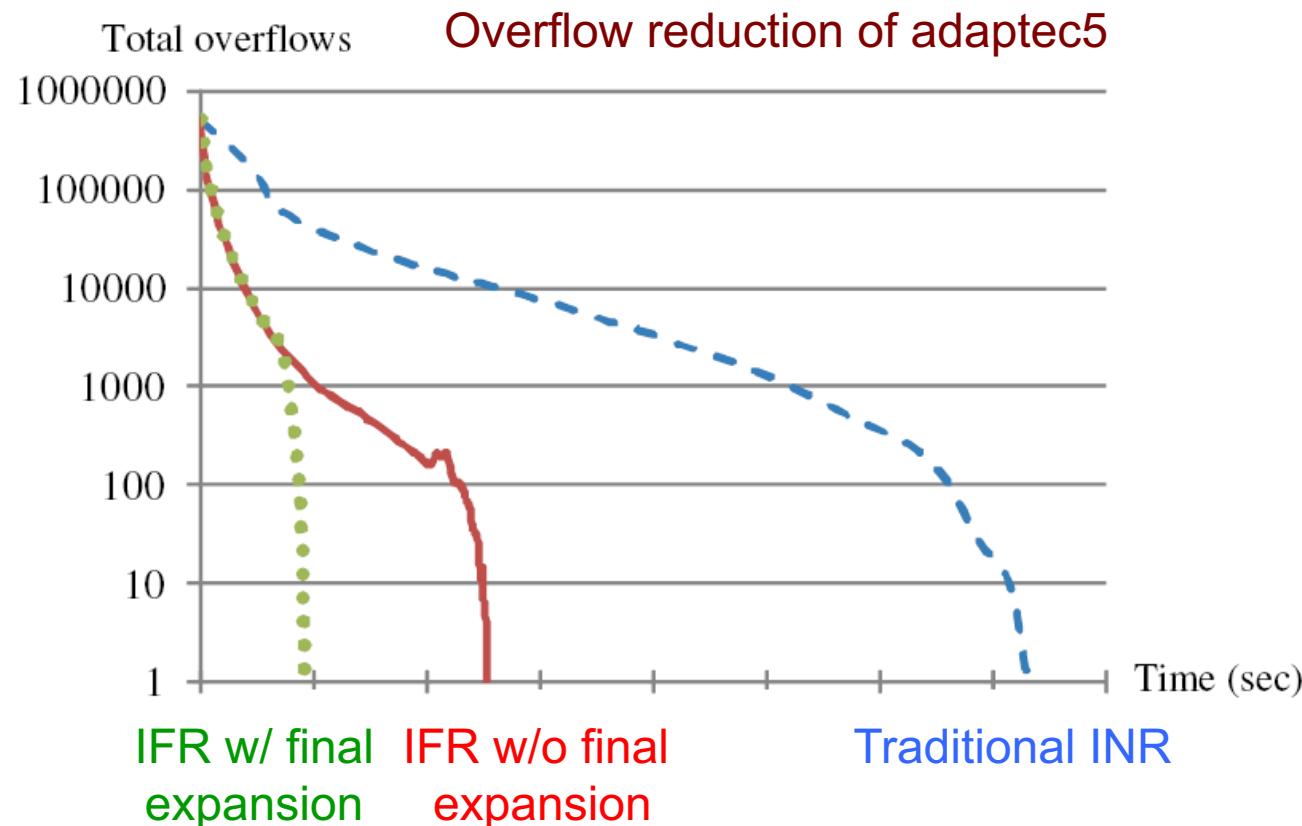


(i)-th iteration (i+1)-th iteration (i+2)-th iteration final iteration

Forbidden-region routing of bigblue3

Final Expansion of Forbidden Regions

- Applied when # of overflows < 0.5% of initial overflow
- Expand the forbidden region to the whole routing graph to quickly reduce the remaining overflows



Comparisons of Congested Region Handling

	BoxRouter	NTHU-Route 1.0	NTUgr (Ours)
Terminology	Box	Congested region	Forbidden region
Shape	Rectangular	Rectangular	Rectilinear
# of regions	Single box	Single region	Multiple regions
Objective	Performing progressive ILP	Selecting rerouting nets	Performing different cost functions
Simultaneous expansion	No	No	Yes

Critical Nets Rerouting Selection

- To speed up the rip-up/rerouting process
- Only rip-up/reroute the **critical nets** in each iteration
- The critical nets are those nets with overflows or small remaining capacity:

$$\min_{e \in n} \{c_e - d_e\} \leq 1$$

c_e : routing capacity of e

d_e : routing demand of e

Look-Ahead Historical Cost Increment

- For the **near-overflow global edges** (those edges would have overflow if more N demands are added), increase their historical cost in advance

$$h_e^{(i)} = \begin{cases} h_e^{(i-1)} + 1 & \text{if } d_e + N > c_e \text{ (} e \text{ is near-overflow)} \\ h_e^{(i-1)} & \text{otherwise} \end{cases}$$

$N \geq 1$: The look-ahead historical-cost update scheme

- Setting $N = 1$ in NTUgr results in better quality and with about 2x runtime speedup

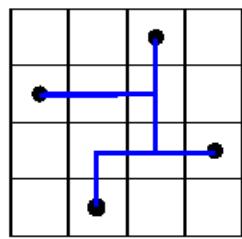
Results on ISPD'08 Benchmarks

- Compared with the winners of ISPD'08 global routing contest
 - Runtime is averagely the same as that of NTHU-Route 2.0
 - Overflow is better than FastRoute 3.0

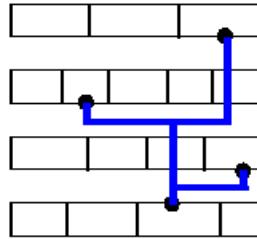
Circuit	NTHU-Route 2.0				FastRoute 3.0				NTUgr (Ours)			
	Overflow	Max Overflow	ISPD'08 WL (e ⁵)	CPU (min)	Overflow	Max Overflow	ISPD'08 WL (e ⁵)	CPU (min)	Overflow	Max Overflow	ISPD'08 WL (e ⁵)	CPU (min)
adaptec1	0	0	53.5	7.9	0	0	55.5	1.8	0	0	57.4	4.5
adaptec2	0	0	52.3	1.7	0	0	53.1	0.4	0	0	53.7	1.1
adaptec3	0	0	131.1	8.0	0	0	133.3	1.7	0	0	135.0	4.4
adaptec4	0	0	121.7	2.3	0	0	122.2	0.6	0	0	123.7	1.2
adaptec5	0	0	155.6	17.2	0	0	160.9	4.7	0	0	159.9	15.3
bigblue1	0	0	56.3	9.8	0	0	58.3	3.9	0	0	60.0	18.1
bigblue2	0	0	90.6	9.9	142	4	98.2	11.2	0	0	91.2	248.3
bigblue3	0	0	130.8	4.3	0	0	131.7	3.1	0	0	133.5	4.0
bigblue4	182	2	230.8	125.6	206	2	243.4	41.3	188	8	242.8	413.1
newblue1	0	0	46.5	5.1	76	2	49.0	12.9	6	2	49.3	977.5
newblue2	0	0	75.7	1.1	0	0	76.3	0.3	0	0	76.9	0.6
newblue3	31454	204	106.5	129.1	31650	734	109.3	31.5	31024	408	188.3	884.0
newblue4	152	2	129.9	67.1	226	4	135.7	9.9	142	2	143.8	1118.1
newblue5	0	0	231.7	14.2	0	0	241.2	5.5	0	0	244.9	20.5
newblue6	0	0	177.0	13.6	0	0	186.6	4.0	0	0	186.6	21.3
newblue7	68	2	353.6	140.6	588	6	358.6	189.9	310	2	372.2	1445.5
Comp.	-	-	1.00	3.48	-	-	1.03	1.00	-	-	1.04	3.01

The Routing-Tree Problem

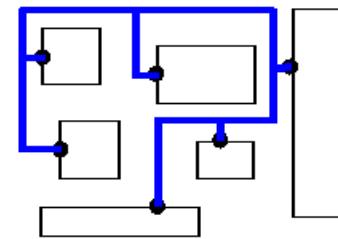
- **Problem:** Given a set of pins of a net, interconnect the pins by a “routing tree.”



gate array

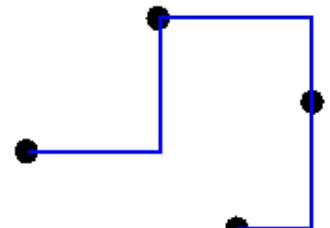


standard cell

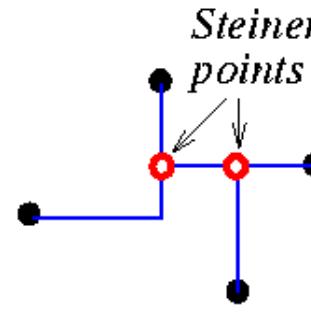


building block

- **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.



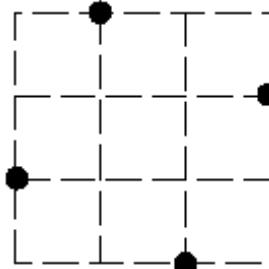
minimum spanning tree
MST



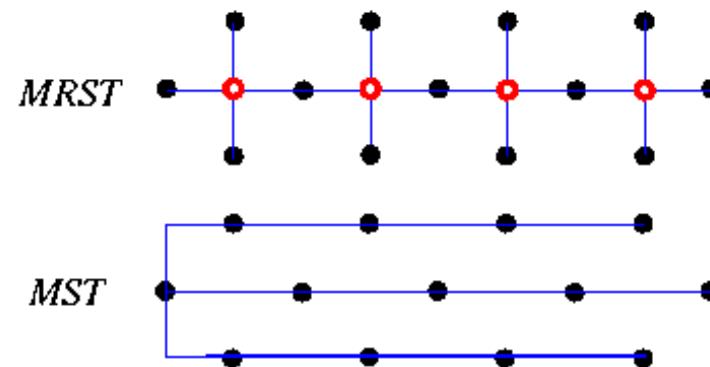
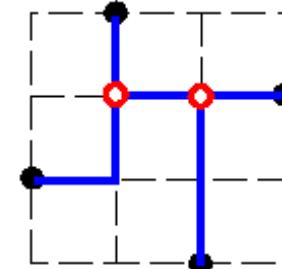
MRST

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 through points of P .
 - Hanan, “On Steiner's problem with rectilinear distance,” *SIAM J. Applied Math.*, 1966.
- **Hwang's Theorem:** For any point set P , $\frac{\text{Cost}(MST(P))}{\text{Cost}(MRST(P))} \leq \frac{3}{2}$.
 - 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.



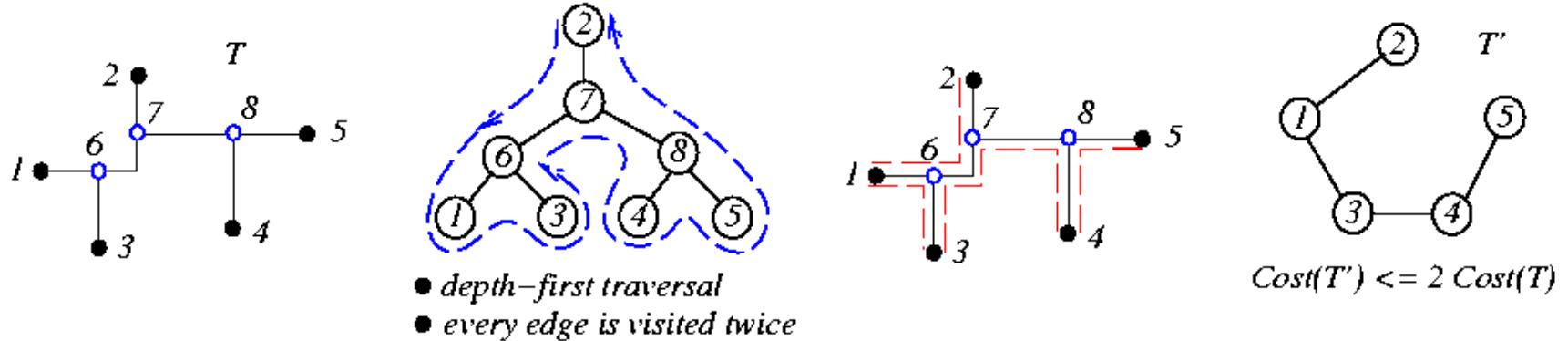
Hanan grid



$$\text{Cost}(MST)/\text{Cost}(MRST) \rightarrow 3/2$$

A Simple Performance Bound

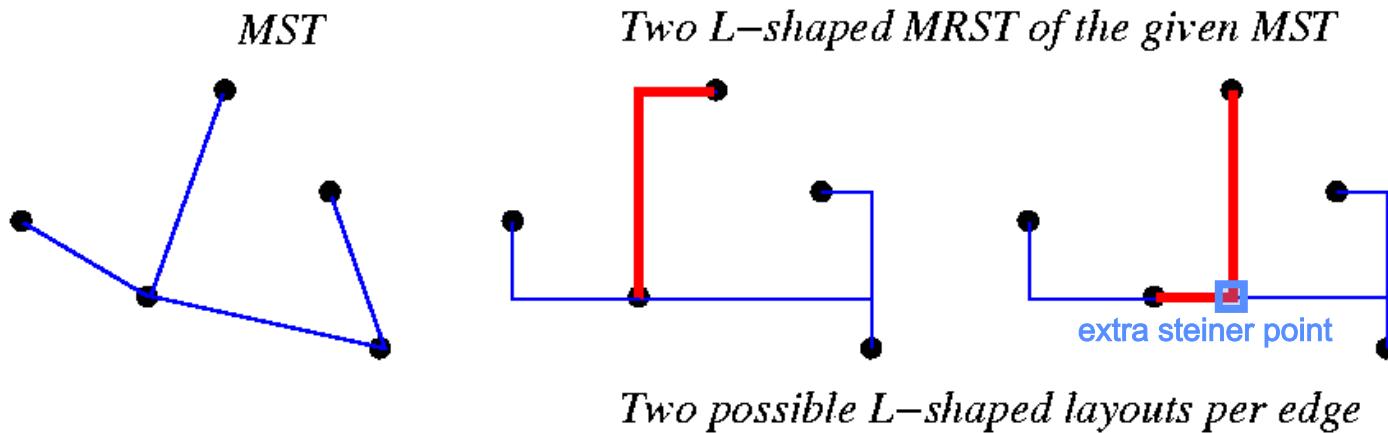
- Easy to show that $\frac{Cost(MST(P))}{Cost(MRST(P))} \leq 2$
- Given any MRST T on point set P with Steiner point set S , construct a spanning tree T' on P as follows:
 1. Select any point in T as a root.
 2. Perform a depth-first traversal on the rooted tree T .
 3. Construct T' based on the traversal.



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 $\text{Cost}(\text{MST})$.

遇到斜線要轉換成L型，再
透過DP去做最佳解的搜尋



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**

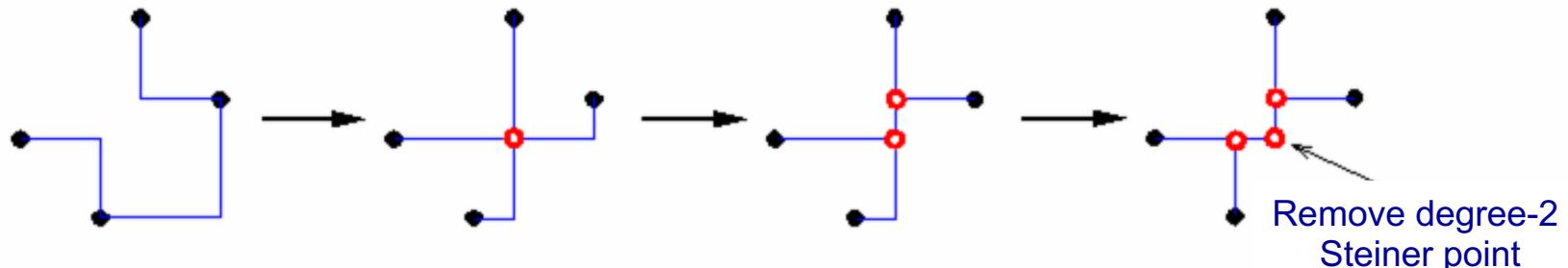
4 Find $x \in Cand$ which maximizes $\Delta MST(P \cup S, \{x\})$;

5 $S \leftarrow S \cup \{x\}$;

6 Remove points in S which have degree ≤ 2 in $MST(P \cup S)$;

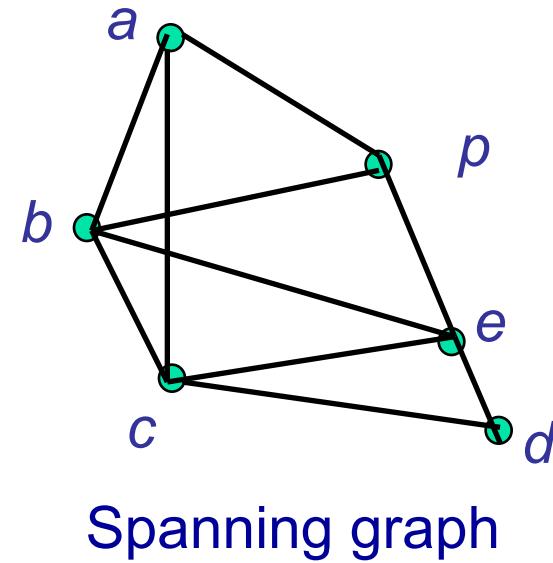
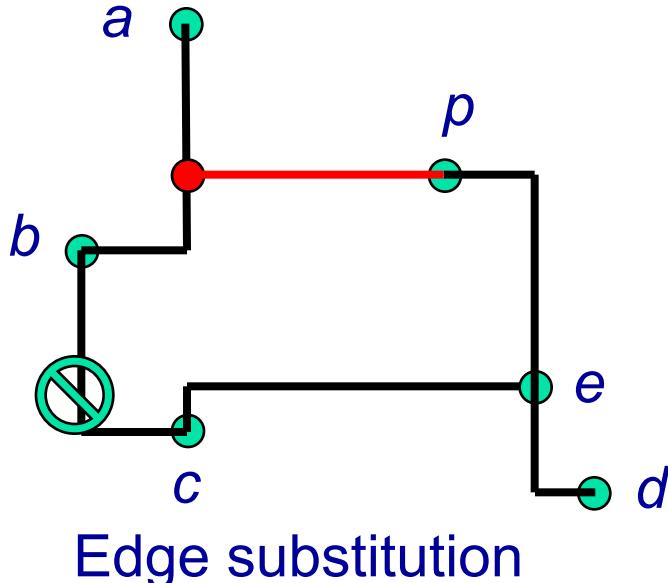
7 **Output** $MST(P \cup S)$;

8 **end**



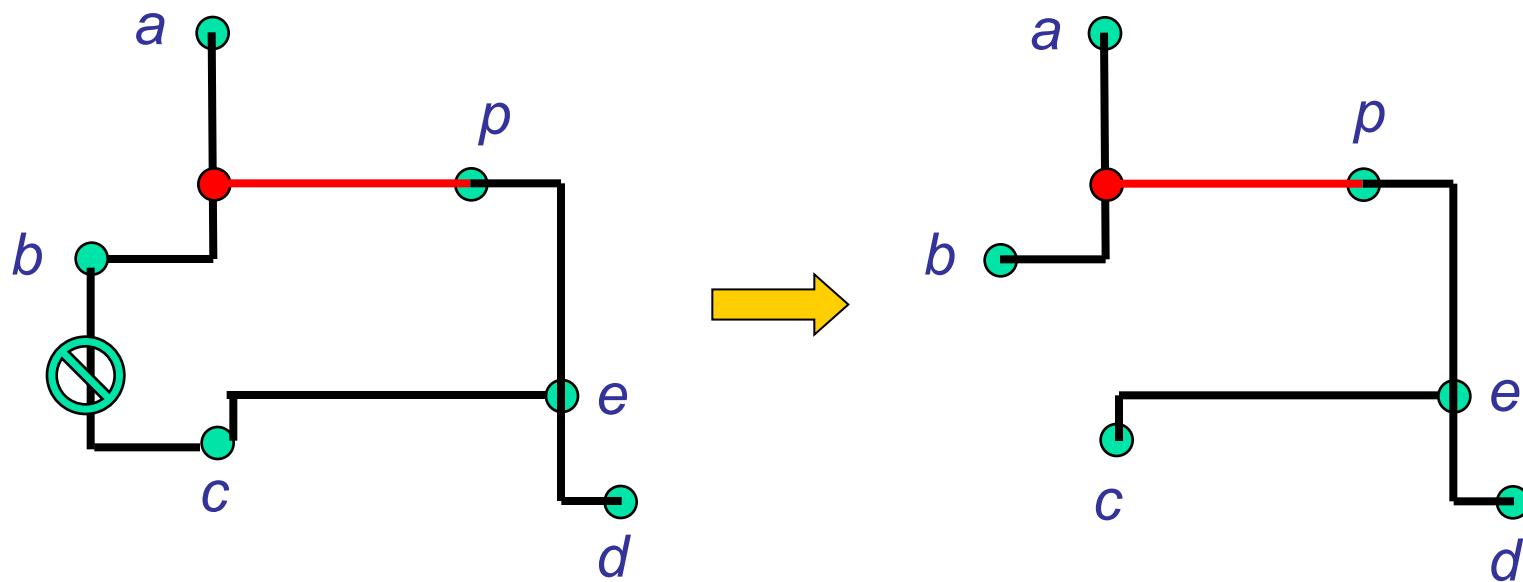
Steiner Minimal Trees Based on Spanning Graphs

- 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 Approach

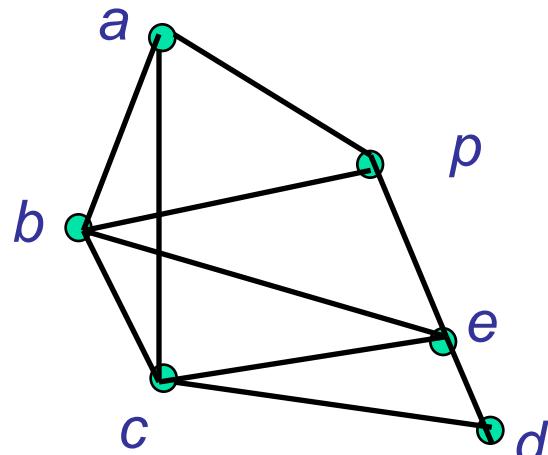
- Borah, Owens, and Irwin's algorithm for the rectilinear SMT (TCAD-94)
 - Start with a minimal spanning tree
 - Iteratively consider connecting a point to a nearby edge (e.g., point p to edge (a, b))
 - Delete the longest edge on the cycle thus formed (e.g., edge (b, c))



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?

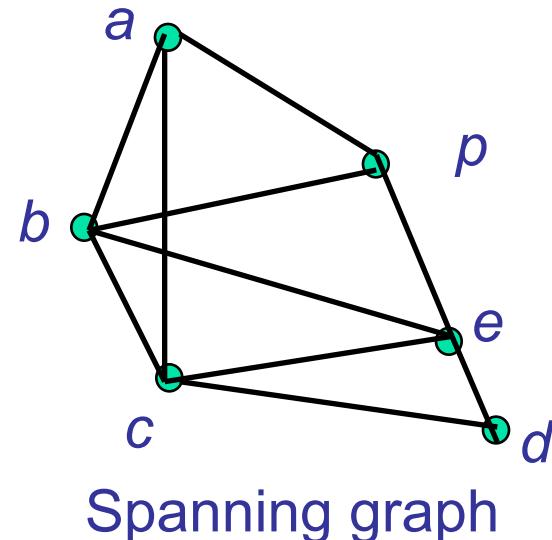
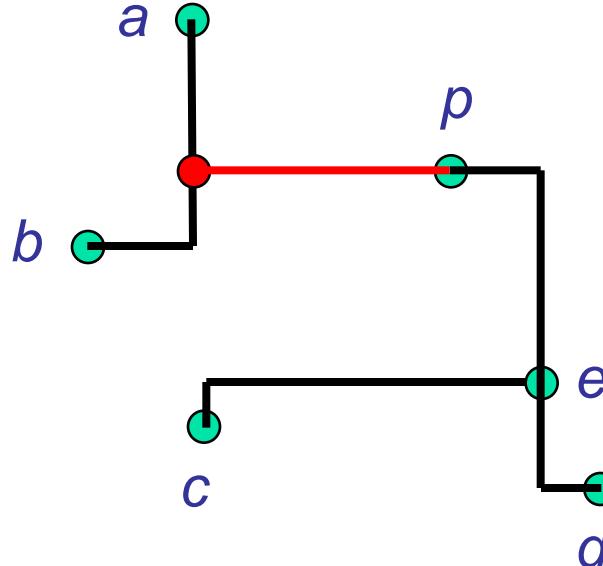
Spanning graphs may answer the questions!!



Spanning graph

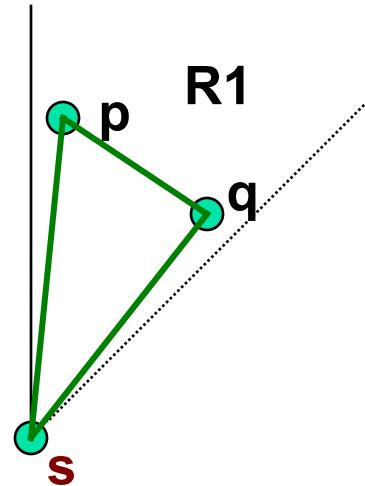
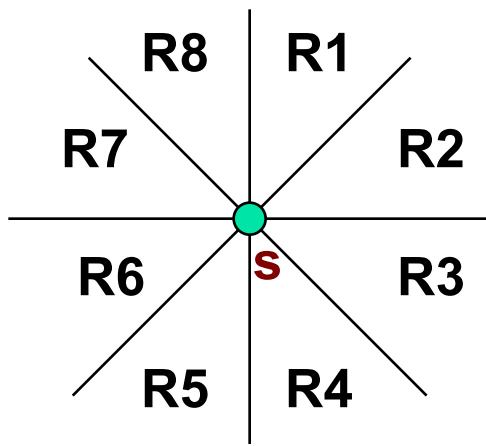
Spanning Graph

- 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



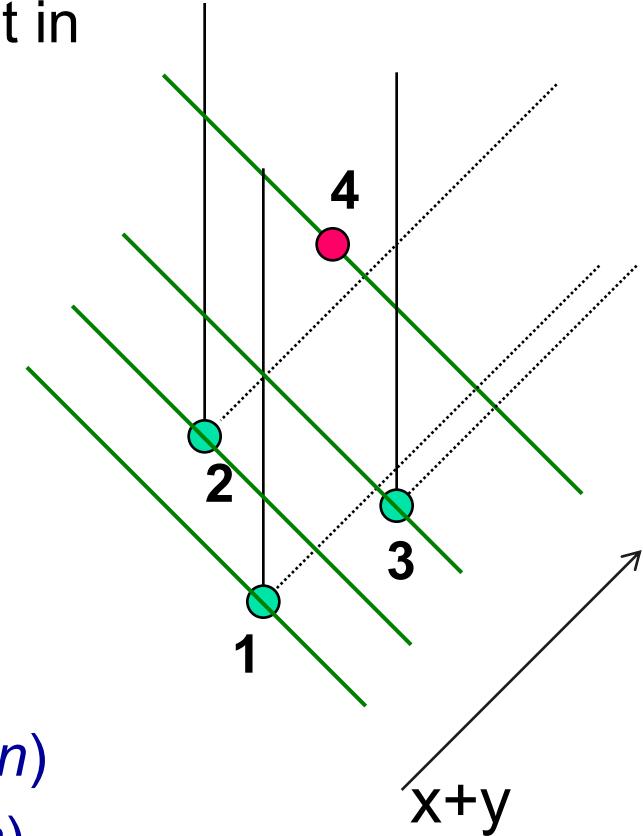
Octal Partition for Finding a Spanning Graph

- 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

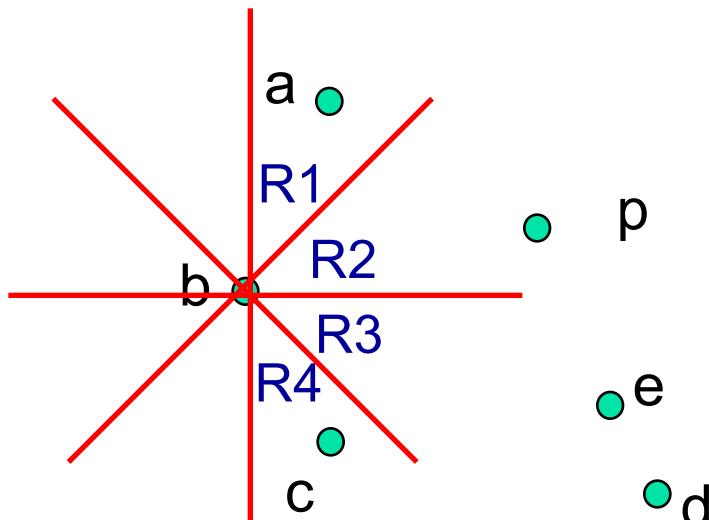


Finding Closest Points in R¹

- 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: $O(n \lg n)$

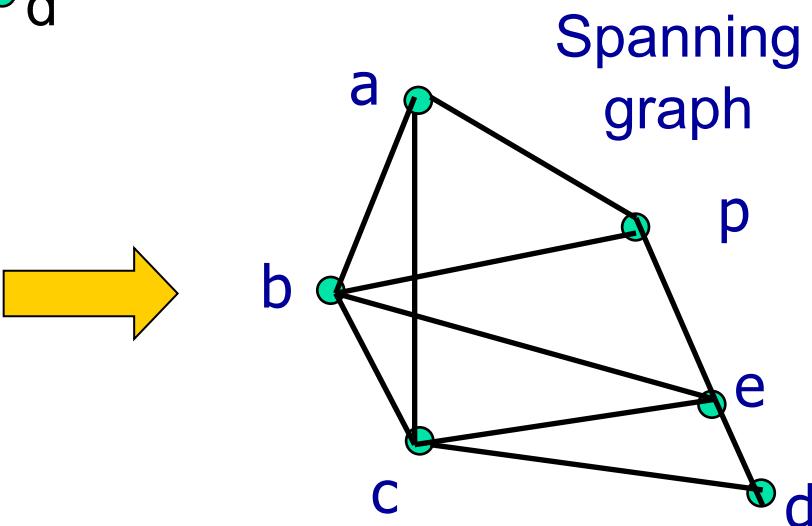


Finding the Neighbors



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

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



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
 - 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

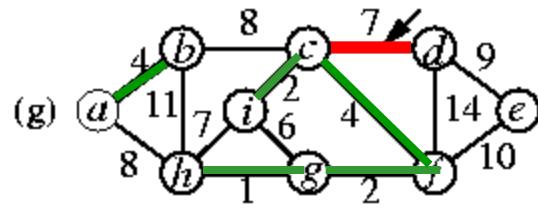
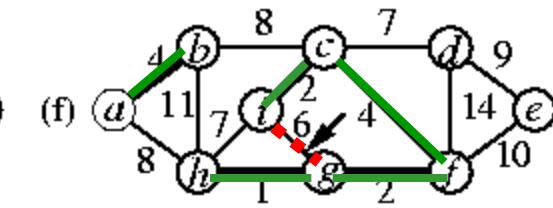
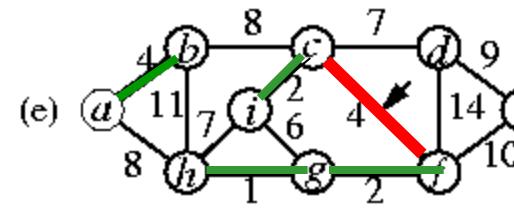
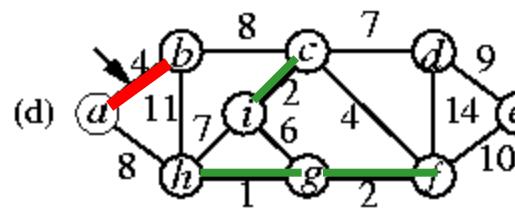
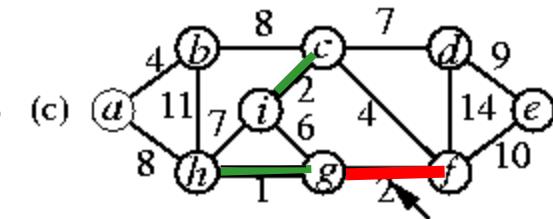
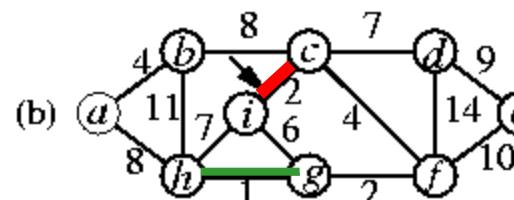
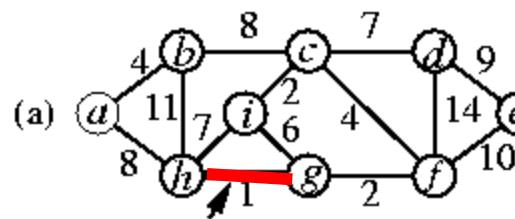
Kruskal's MST Algorithm

```
MST-Kruskal( $G, w$ )
```

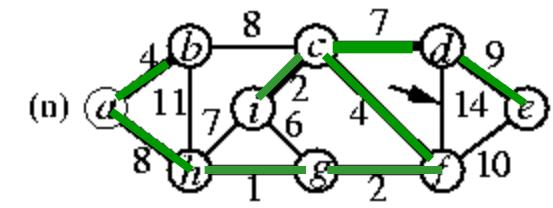
1. $A \leftarrow \emptyset$;
2. **for** each vertex $v \in V[G]$
3. Make-Set(v);
4. sort the edges of $E[G]$ by nondecreasing weight w ;
5. **for** each edge $(u, v) \in E[G]$, in order by nondecreasing weight
6. **if** Find-Set(u) \neq Find-Set(v)
7. $A \leftarrow A \cup \{(u, v)\}$;
8. Union(u, v);
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 $O(E \alpha(E, V))$.

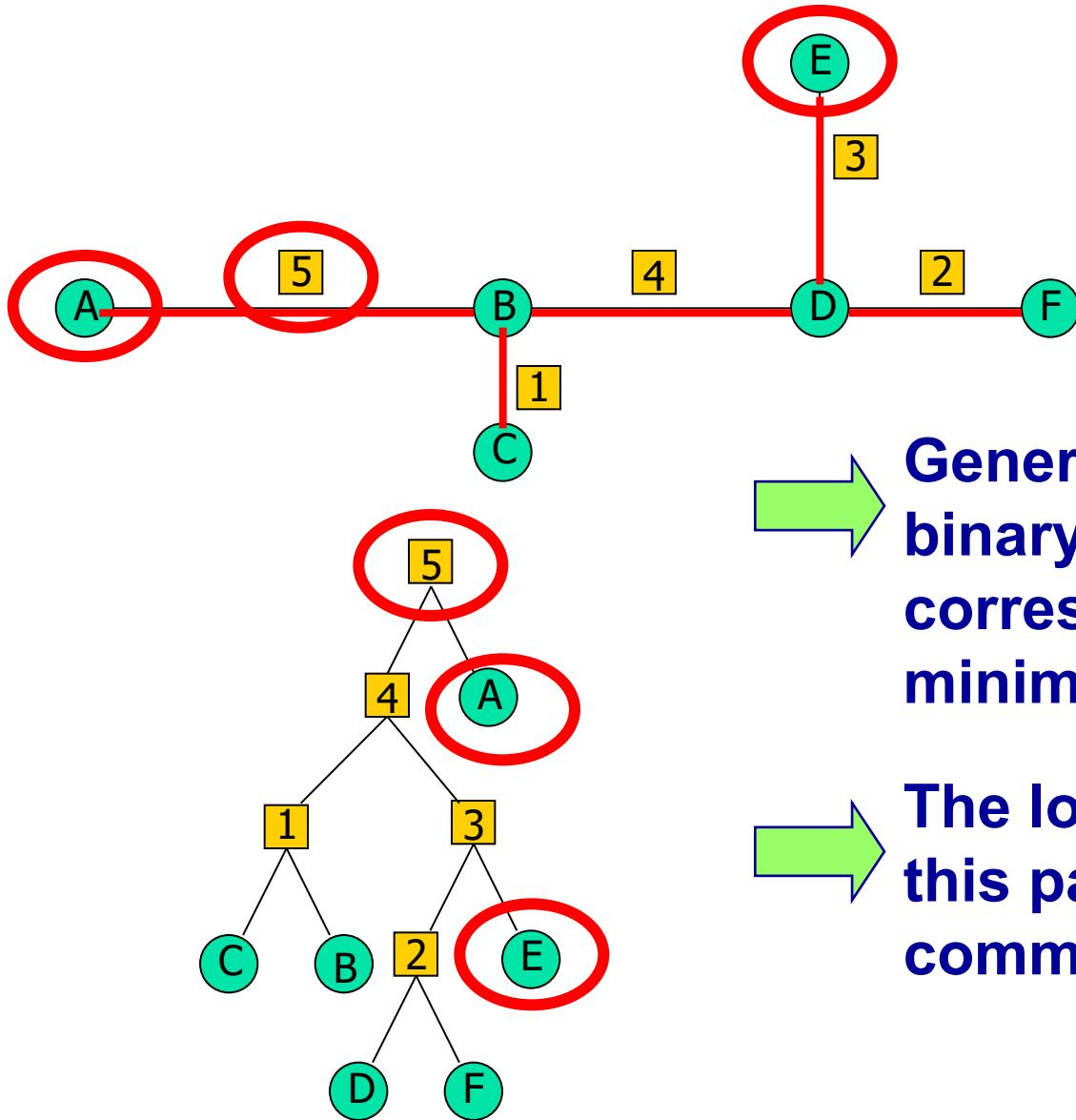
Example: Kruskal's MST Algorithm



.....



Finding the Longest Edge on a Cycle

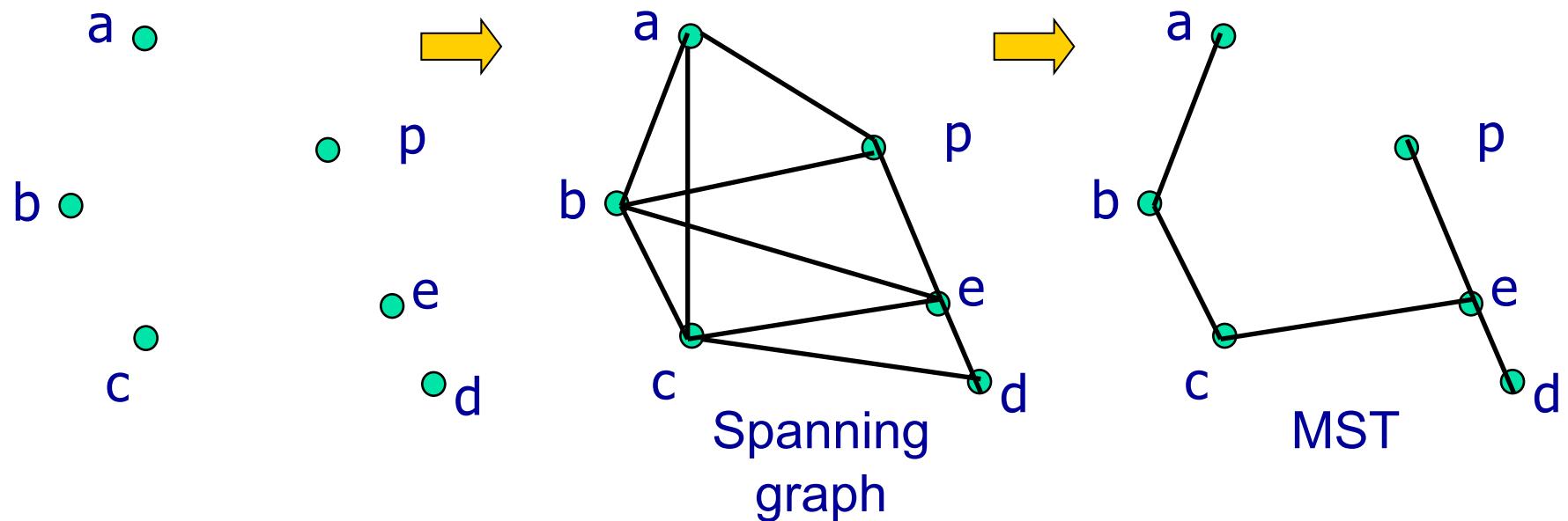


Generate the merging
binary tree
corresponding to the
minimal spanning tree

The longest edge for
this pair is the least
common ancestor

Example Spanning Graph Construction

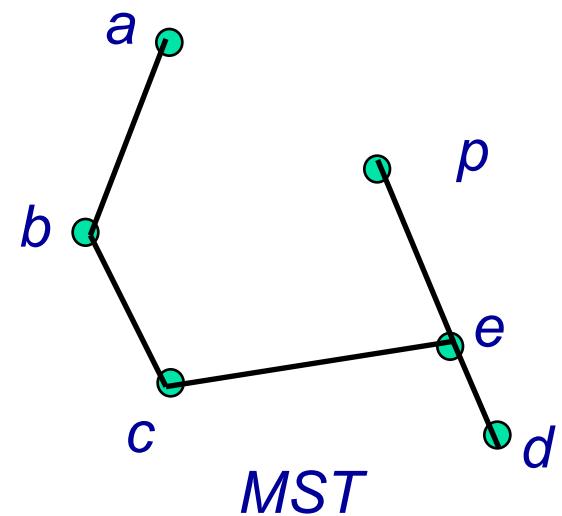
1. Construct a rectilinear spanning graph
2. Use Kruskal's algorithm to build MST
3. Update point-edge connections iteratively



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

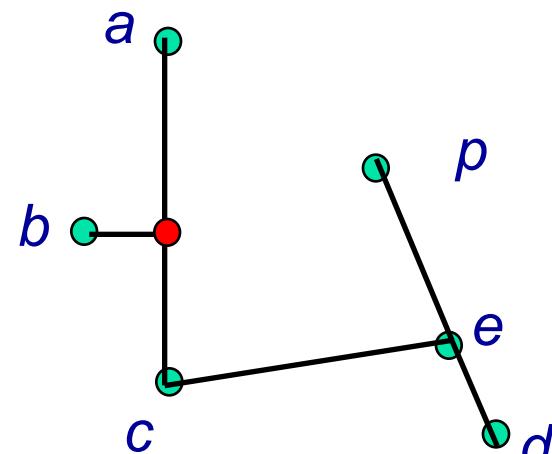


Building the table according to the spanning graph

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.)

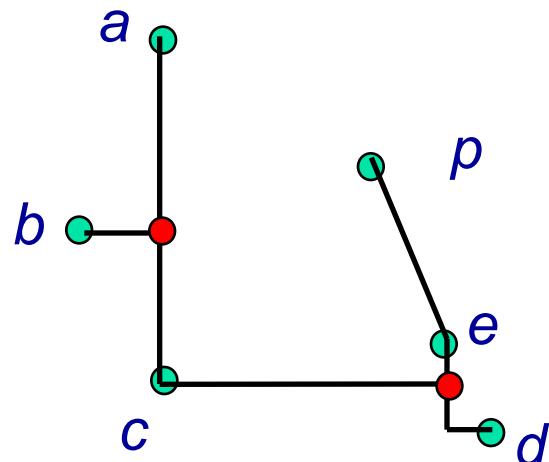
point	edge	delete edge	gain
a	(b,c)	(a,b)	3
c	(a,b)	(b,c)	3
p	(a,b)	(b,c)	3
P	(a,b)	(c,e)	2
c	(d,e)	(c,e)	1
e	(b,c)	(c,e)	1
p	(a,b)	(e,p)	1
p	(b,c)	(c,e)	1
p	(c,e)	(e,p)	1



Point-Edge Connection Updating (cont'd)

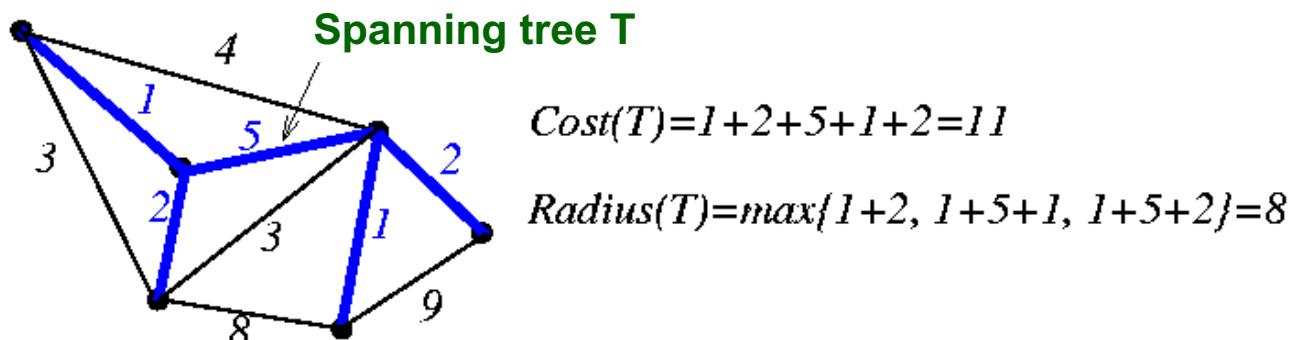
1. Construct a rectilinear spanning graph
2. Use Kruskal's algorithm to build MST
3. Update point-edge connections iteratively
 - Find the remaining point-edge pair with the max gain (point c to edge (d, e))

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



Bounded-Radius (-Diameter) Minimum Spanning Tree

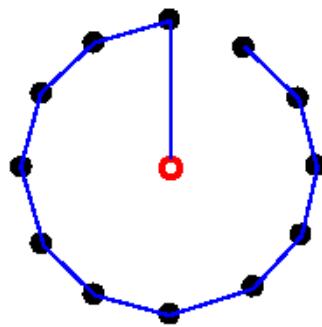
- **Problem:** Given a parameter $\varepsilon \geq 0$ and a signal net with radius R (diameter D), find a minimum-cost spanning tree T with radius $r(T) \leq (1 + \varepsilon)R$ ($d(T) \leq (1 + \varepsilon)D$).
 - Awerbuch, et al., “Cost-sensitive analysis of communication protocols,” *ACM Symp. Principles 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?



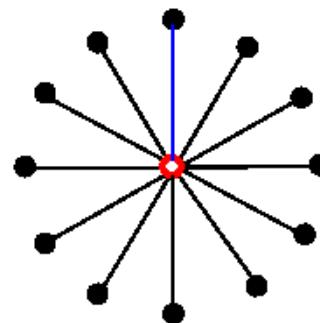
MST vs. SPT Trade-off

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

Minimum spanning tree (MST)

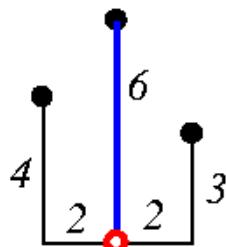


Shortest path tree (SPT)

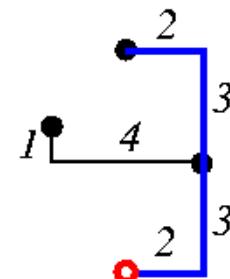


$$\frac{\text{Cost}(SPT)}{\text{Cost}(MST)} = \Theta(n)$$

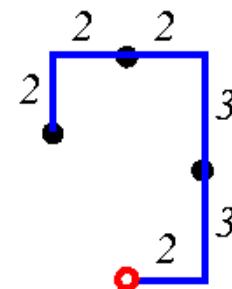
$$r(T) \leq (1 + \epsilon)R$$



$$\begin{aligned}\epsilon &= 0 \\ \text{Cost} &= 17 \\ \text{Radius} &= 6\end{aligned}$$



$$\begin{aligned}\epsilon &= 1 \\ \text{Cost} &= 15 \\ \text{Radius} &= 10\end{aligned}$$

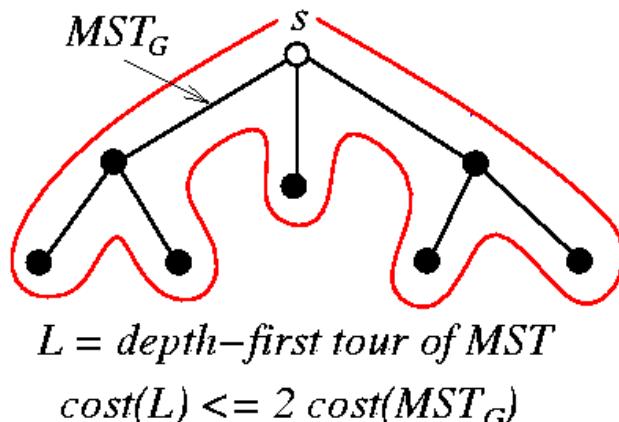


$$\begin{aligned}\epsilon &= \text{infinite} \\ \text{Cost} &= 14 \\ \text{Radius} &= 14\end{aligned}$$

Bounded Radius Bounded Cost Spanning Tree

Algorithm: Bounded-Radius_Bounded-Cost_Spanning_Tree(G)

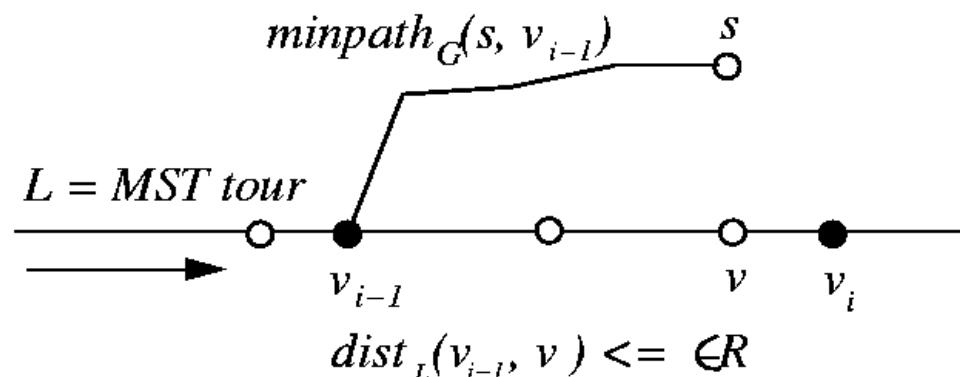
```
1 begin
2 Compute  $MST_G$  and  $SPT_G$ ;
3  $Q \leftarrow MST_G$ ;
4  $L \leftarrow$  depth-first tour of  $MST_G$ ;
5  $S \leftarrow 0$ ;
6 for  $i \leftarrow 1$  to  $|L| - 1$ 
7    $S \leftarrow S + \text{cost}(L_i, L_{i+1})$ ;
8   if  $S \geq \varepsilon \cdot \text{dist}_G(s, L_{i+1})$  then
9      $Q \leftarrow Q \cup \text{minpath}_G(s, L_{i+1})$ ;
10     $S \leftarrow 0$ ;
11  $T =$  shortest path tree of  $Q$ ;
12 end
```



Bounded-Radius Bounded-Cost Spanning Tree

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

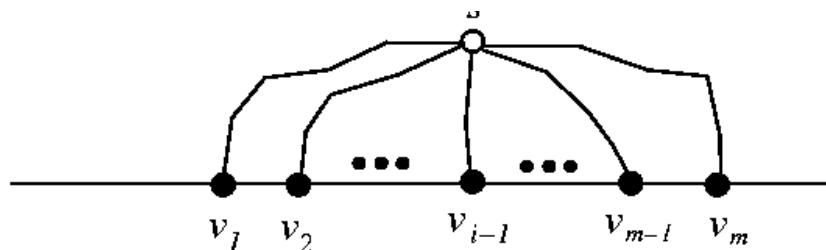
$$\begin{aligned} \text{dist}_T(s, v) &\leq \text{dist}_T(s, v_{i-1}) + \text{dist}_L(v_{i-1}, v) \\ &\leq \text{dist}_G(s, v_{i-1}) + \epsilon R \\ &\leq R + \epsilon R \\ &= (1 + \epsilon)R \end{aligned}$$



Bounded-Radius Bounded-Cost Spanning Tree

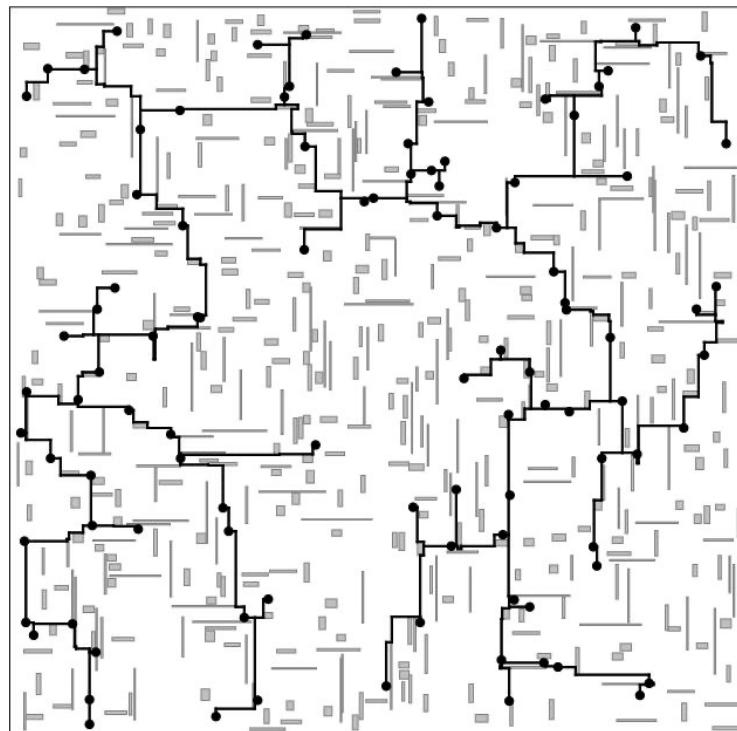
- For any weighted graph G and parameter ϵ , the routing tree T constructed by the algorithm has cost $\text{cost}(T) \leq (1+2/\epsilon)\text{cost}(\text{MST}_G)$.
- Let v_1, v_2, \dots, v_m be the set of nodes to which the algorithm added shortest paths from source s .

$$\begin{aligned}\text{cost}(T) &\leq \text{cost}(\text{MST}_G) + \sum_{i=1}^m \text{dist}_G(s, v_i) \\ \text{dist}_L(v_{i-1}, v_i) &\geq \epsilon \cdot \text{dist}_G(s, v_i) \\ \text{cost}(T) &\leq \text{cost}(\text{MST}_G) + \sum_{i=1}^m \frac{\text{dist}_G(v_{i-1}, v_i)}{\epsilon} \\ &\leq \text{cost}(\text{MST}_G) + \frac{\text{cost}(L)}{\epsilon} \\ &\leq \text{cost}(\text{MST}_G) + \frac{2 \cdot \text{cost}(\text{MST}_G)}{\epsilon} \\ &\leq (1 + \frac{2}{\epsilon}) \text{cost}(\text{MST}_G)\end{aligned}$$



Appendix A: Obstacle-Avoiding Rectilinear Steiner Tree Construction

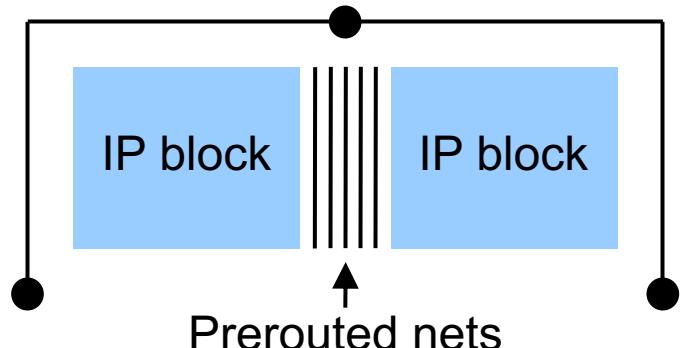
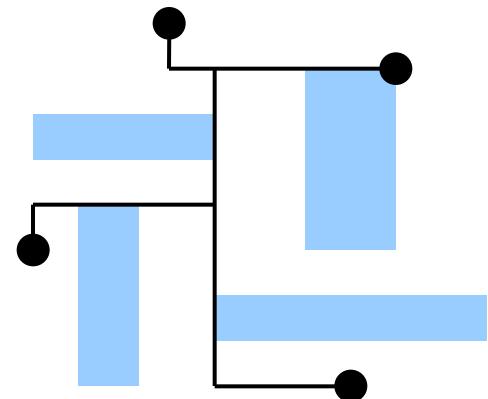
Lin, Chen, Li, Chang, and Yang
ISPD-07



Courtesy of Prof. Y.-W. Chang

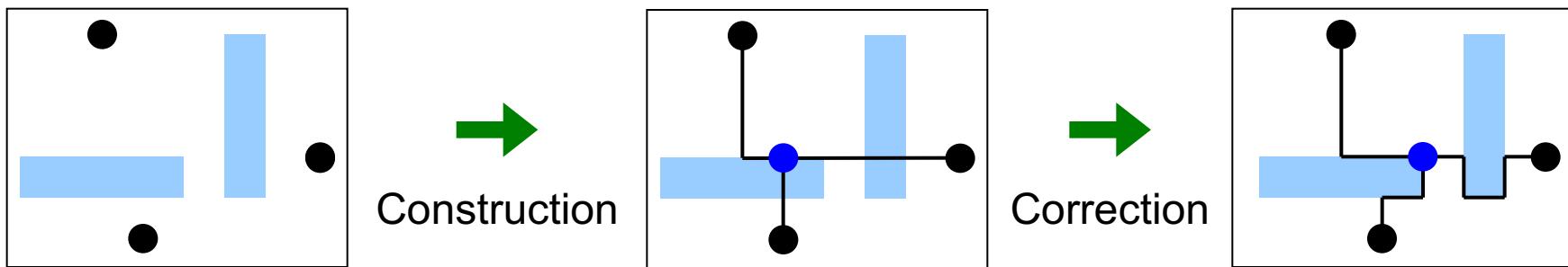
Introduction to OARSMT Problem

- 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

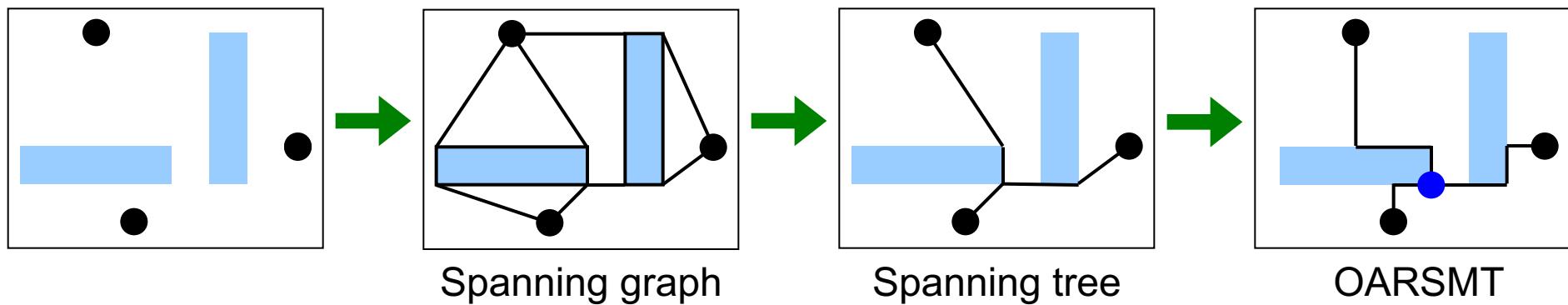
- Construct an initial tree without considering obstacles
- Correct edges overlapping obstacles



- 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

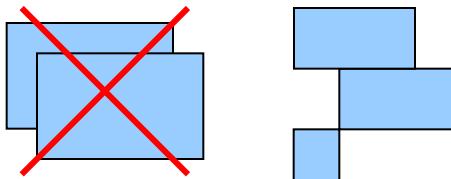


- Prune many redundant edges to reduce problem size
- Obtain much better solution quality
- Shen et al., ICCD-05
 - Achieve good results but can be further improved

Problem Formulation

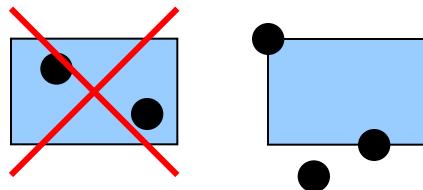
- Given m pins $\{p_1, p_2, \dots, p_m\}$ and k obstacles $\{o_1, o_2, \dots, o_k\}$, construct an OARSMT such that the total wirelength of the tree is minimized.

Obstacles overlap each other



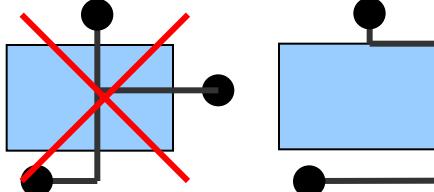
Obstacles are point-touched or line touched

A pin locates inside an obstacle



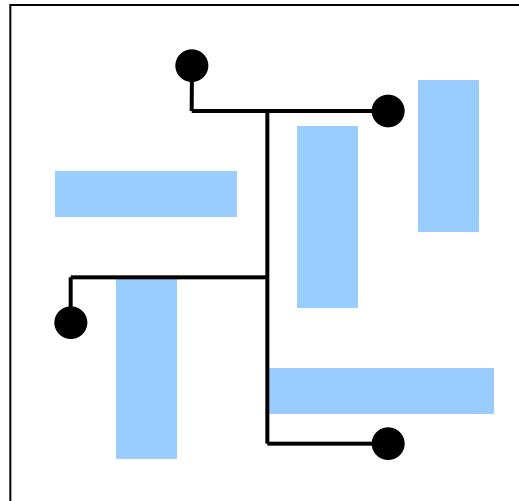
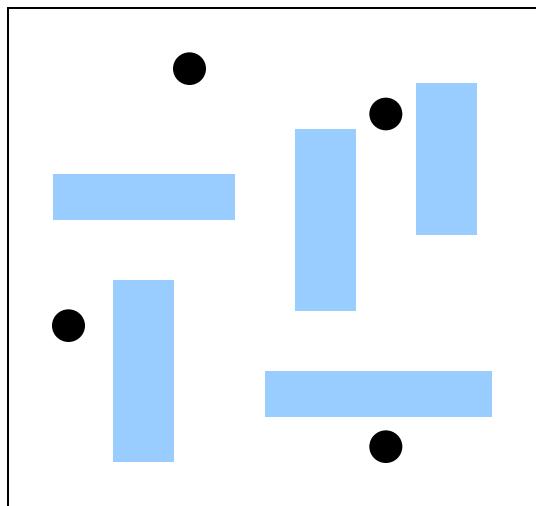
A pin is at the corner or on the boundary of an obstacle

An edge intersects an obstacle

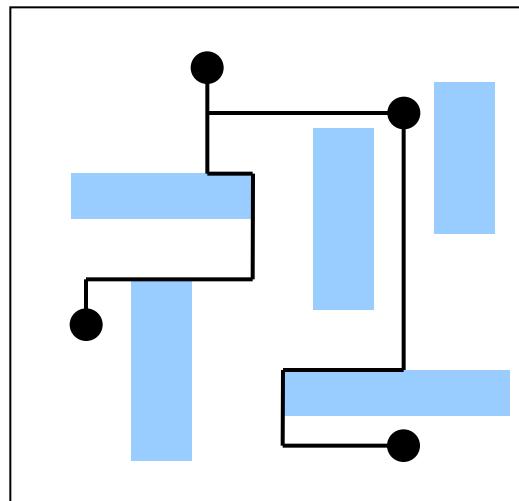


An edge is point-touched or line-touched with an obstacle

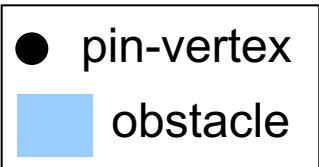
An OARSMT Example



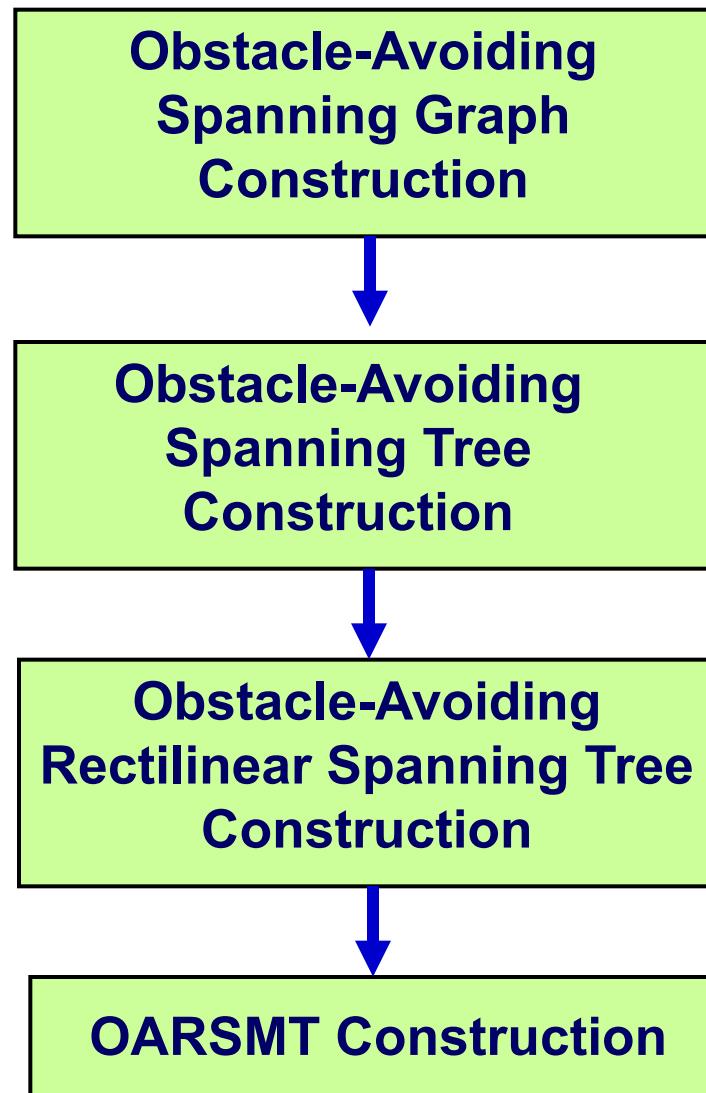
Smaller wirelength



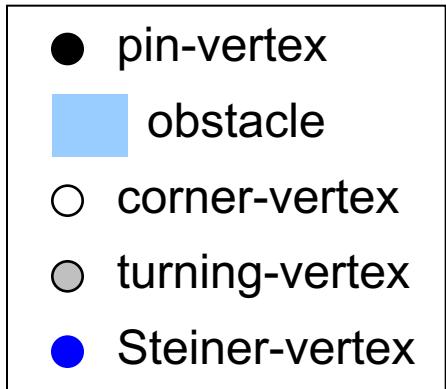
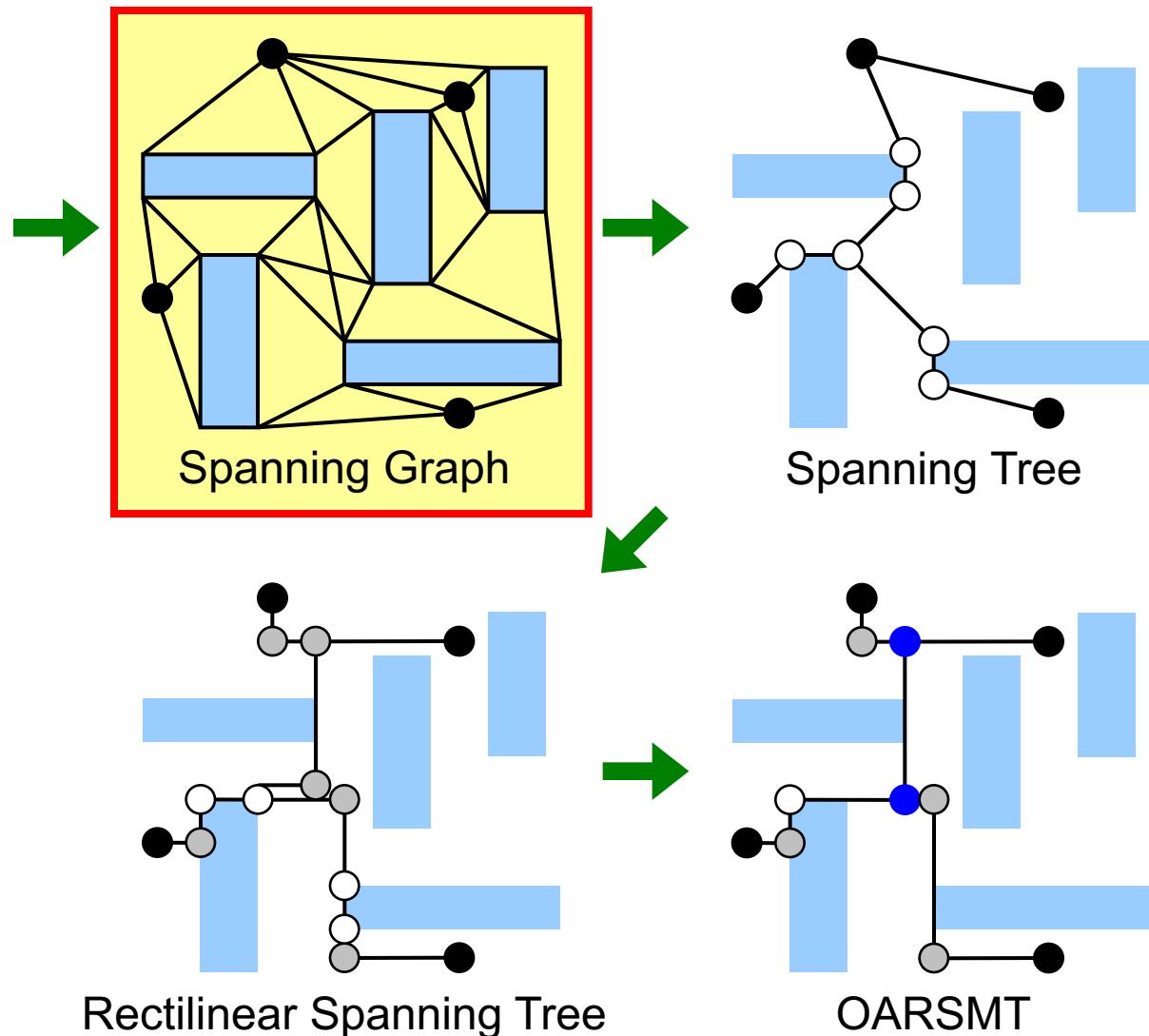
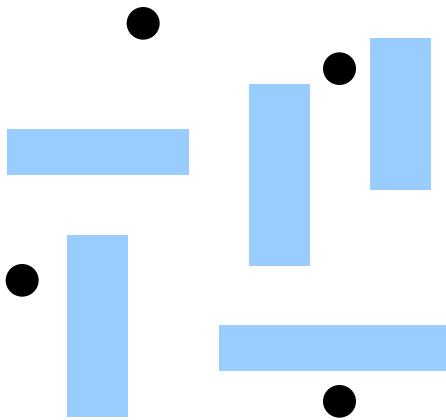
Larger wirelength



Algorithm Flow

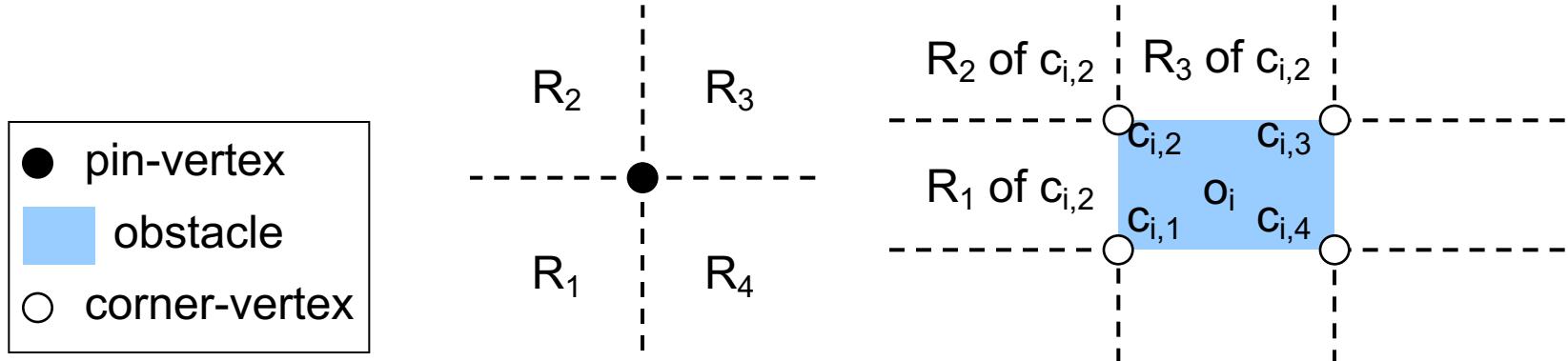


Algorithm Flow

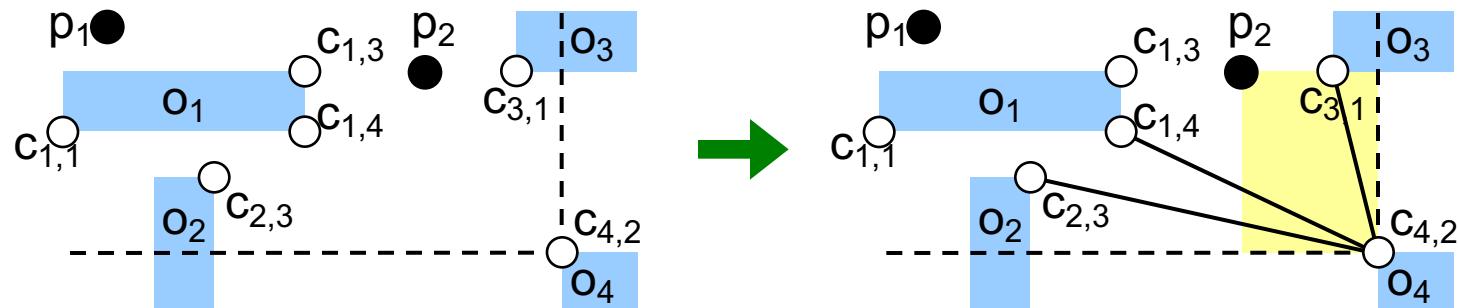


Spanning Graph Construction

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



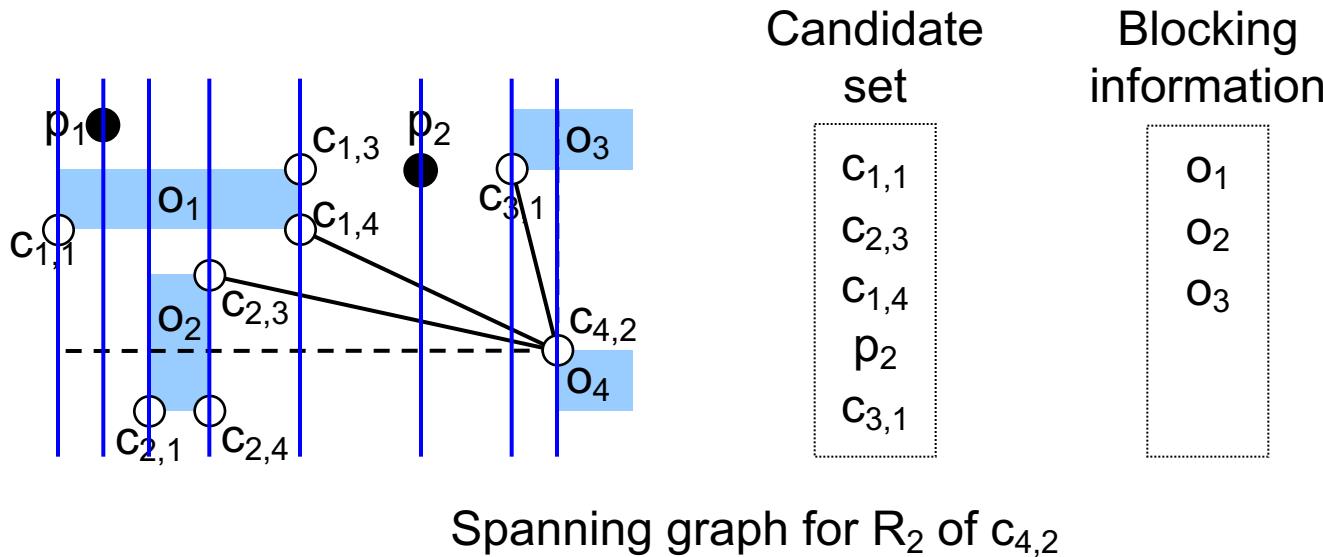
- v_1 and v_2 are connected if no other vertex or obstacle is inside or on the boundary of the bounding box of v_1 and v_2 .



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

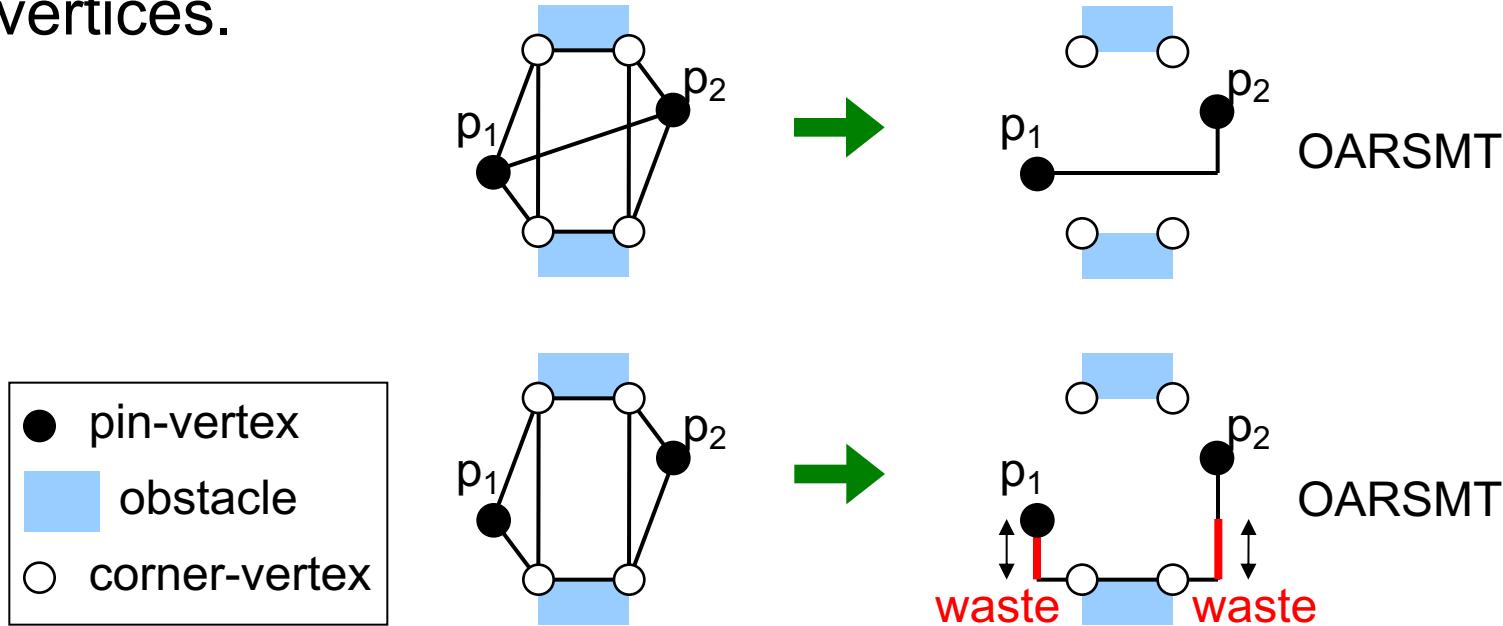
Spanning Graph Construction – Example

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



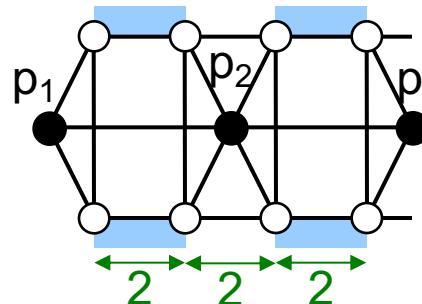
Properties of Our Spanning Graph (1/2)

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

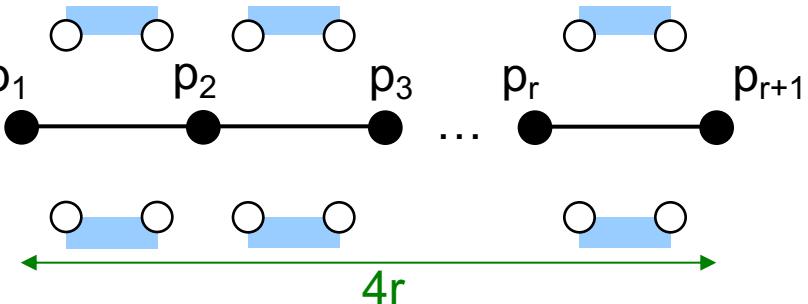
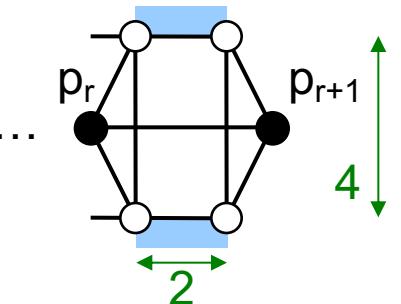


- Spanning graph of Shen et al. (ICCD-05) does not guarantee it.

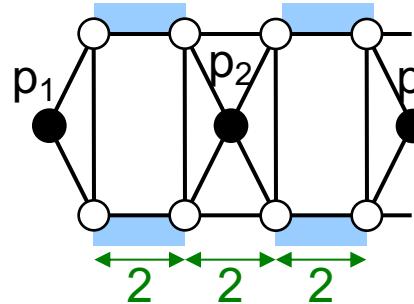
Properties of Our Spanning Graph (2/2)



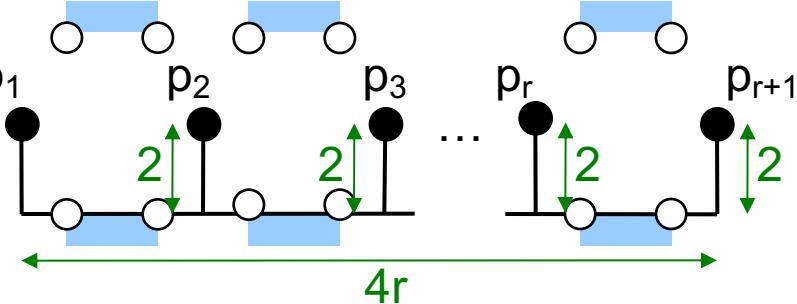
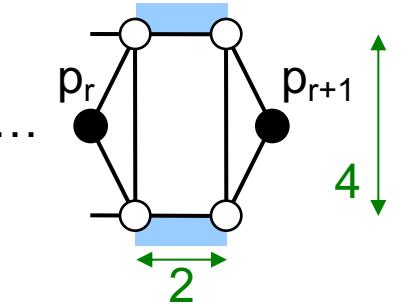
Our spanning graph



OARSMT total wirelength = $4r$



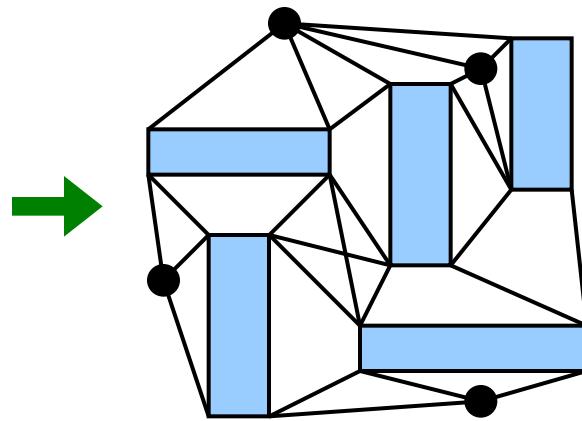
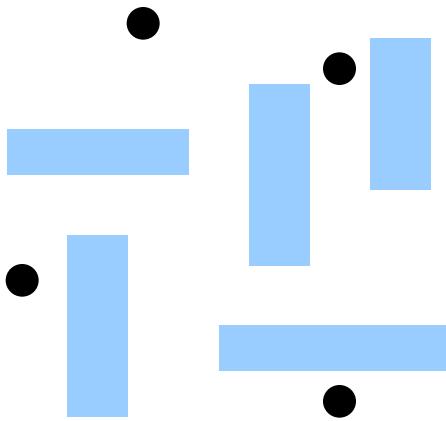
Spanning graph of Shen et al.



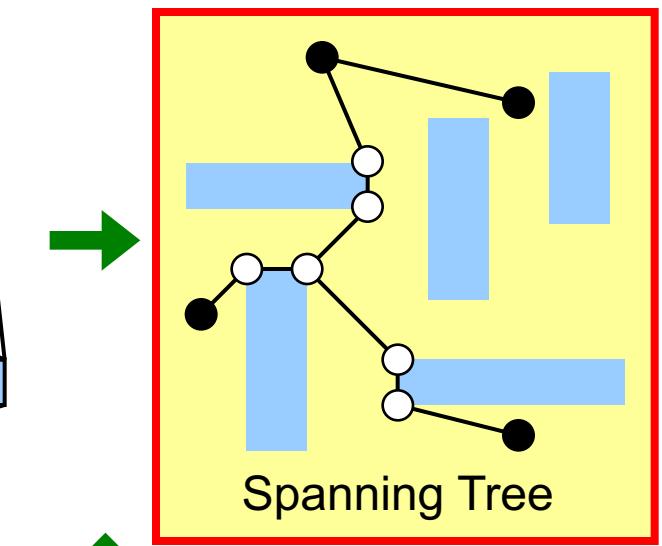
OARSMT total wirelength = $6r+2$

- pin-vertex
- obstacle
- corner-vertex

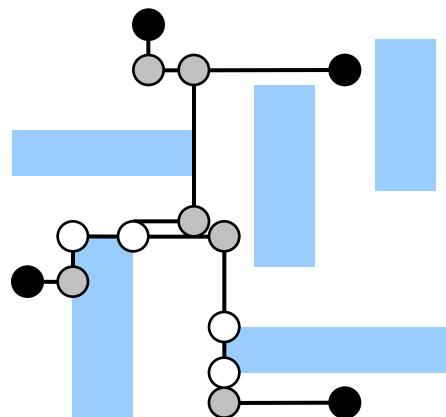
Algorithm Flow



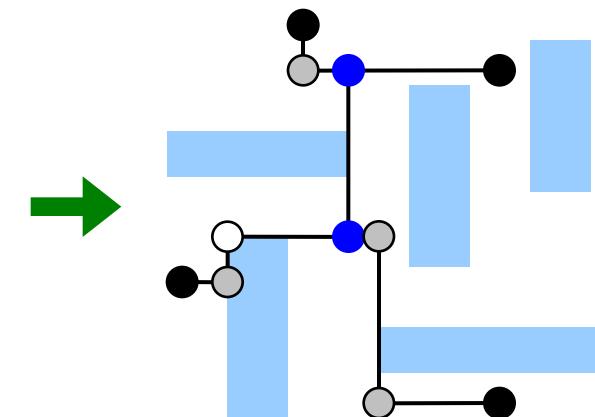
Spanning Graph



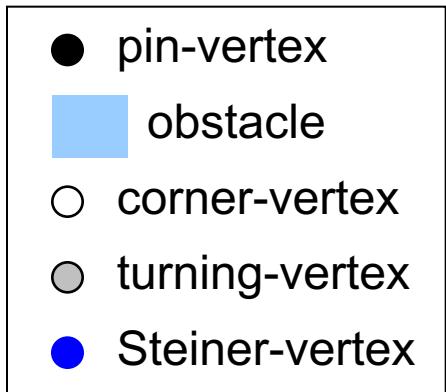
Spanning Tree



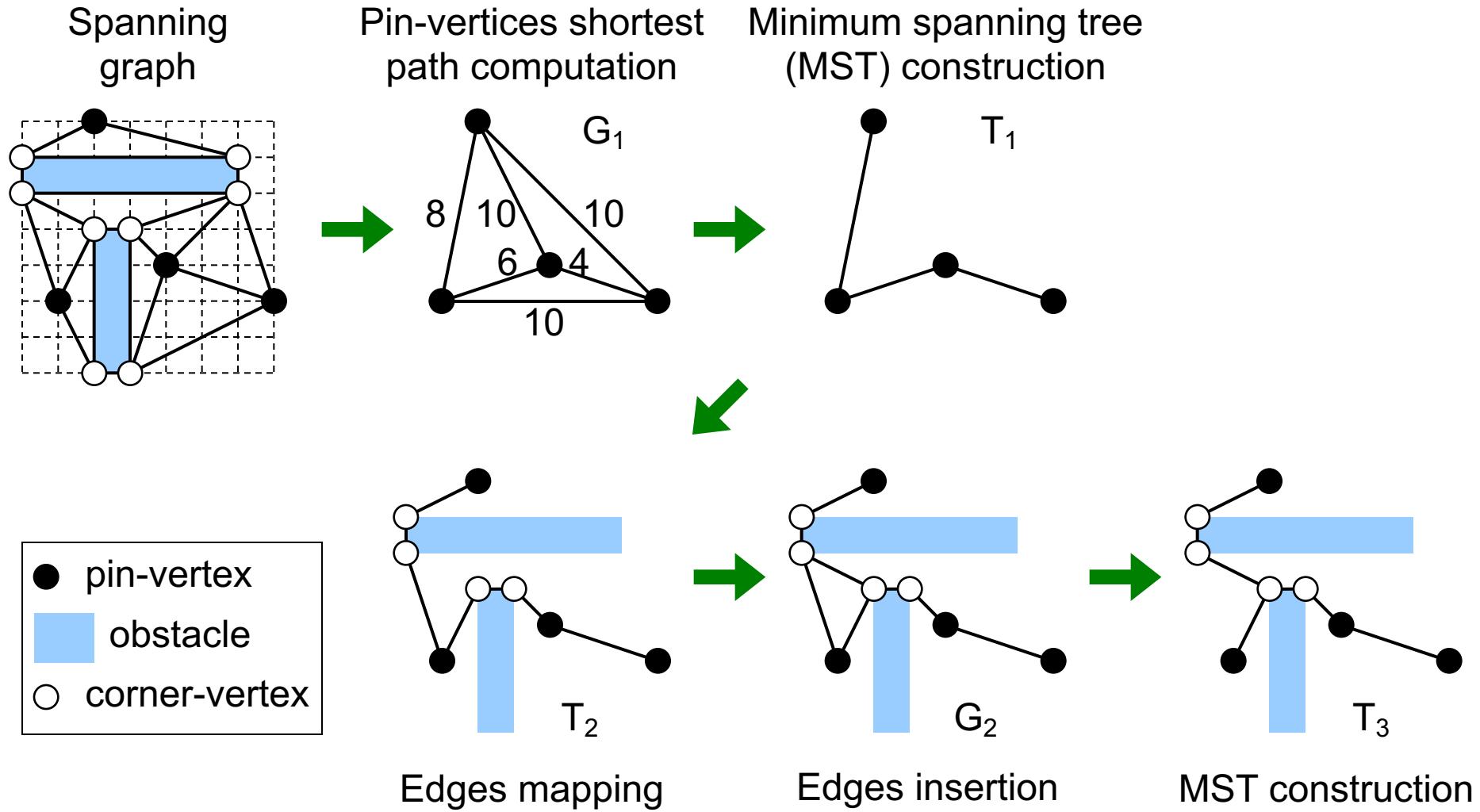
Rectilinear Spanning Tree



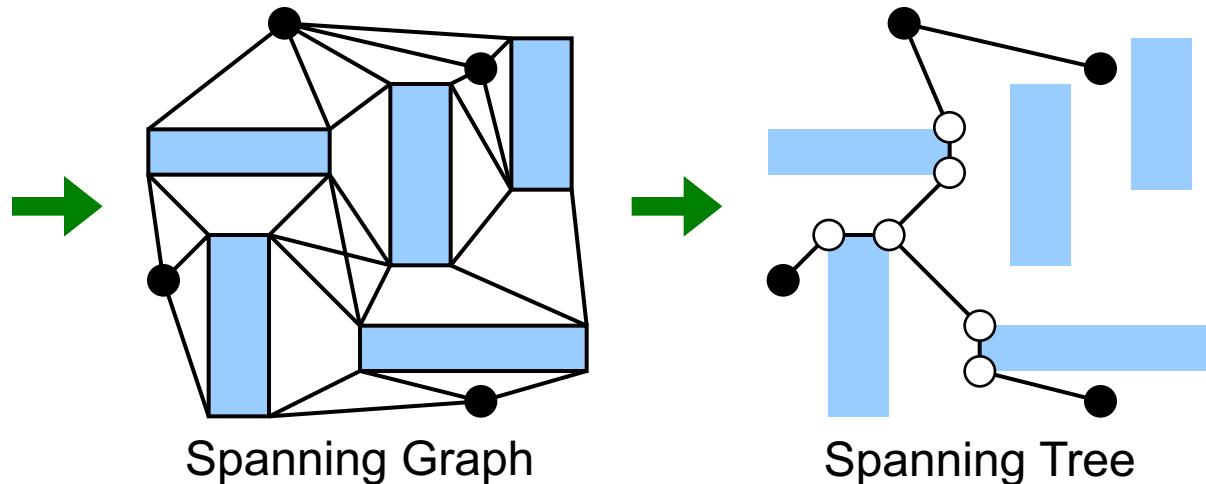
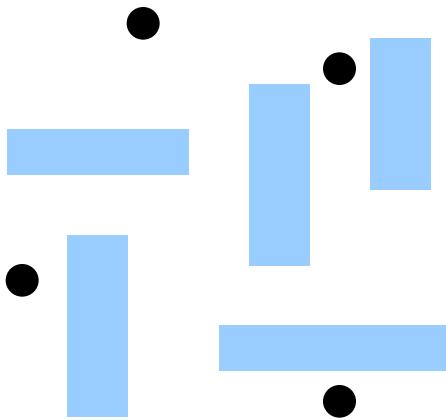
OARSMT



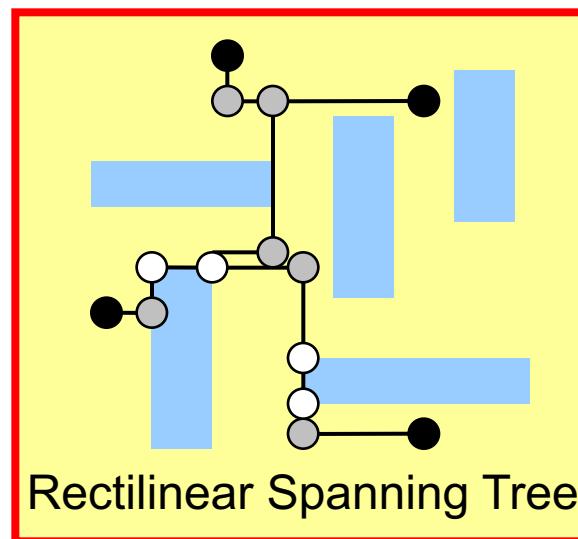
Spanning Tree Construction



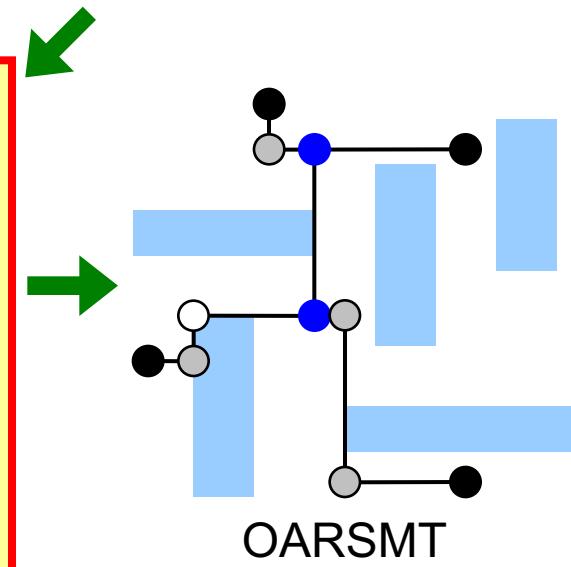
Algorithm Flow



- pin-vertex
- obstacle
- corner-vertex
- turning-vertex
- Steiner-vertex



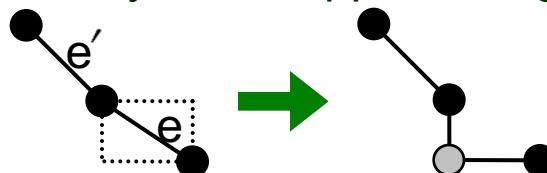
Rectilinear Spanning Tree



OARSMT

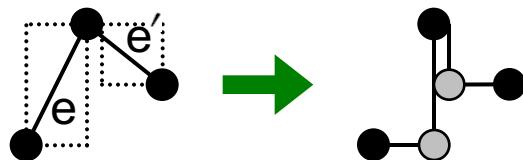
Rectilinear Spanning Tree Construction

- 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' :
 - They are in opposite regions.



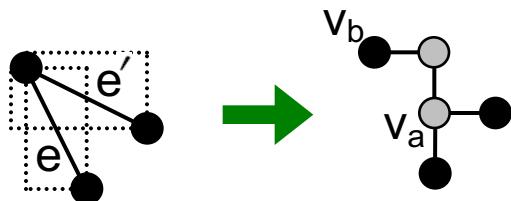
e is transformed randomly

- They are in neighboring regions.



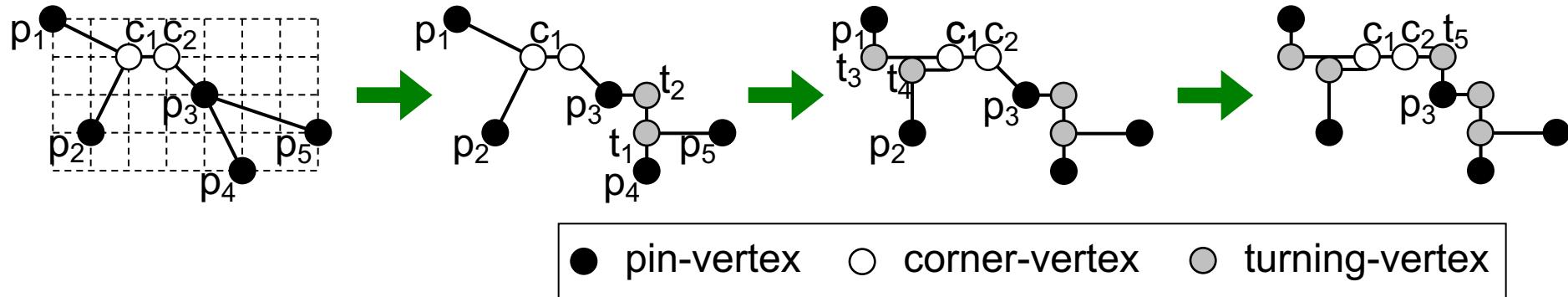
e and e' are transformed with edge overlap

- They are in the same region.



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

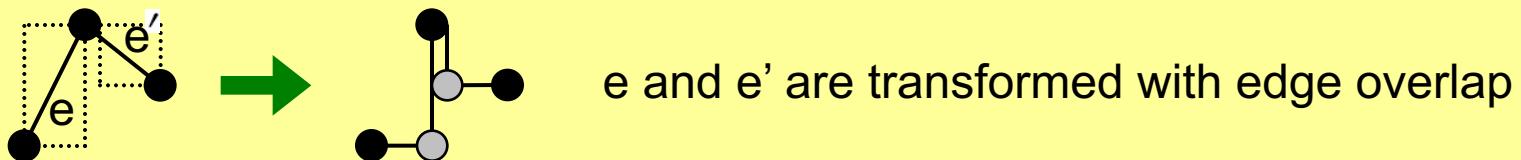
Rectilinear Spanning Tree Construction – Example



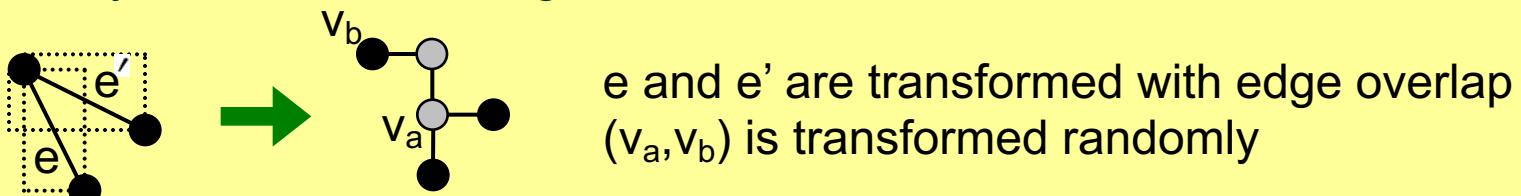
1. They are in opposite regions.



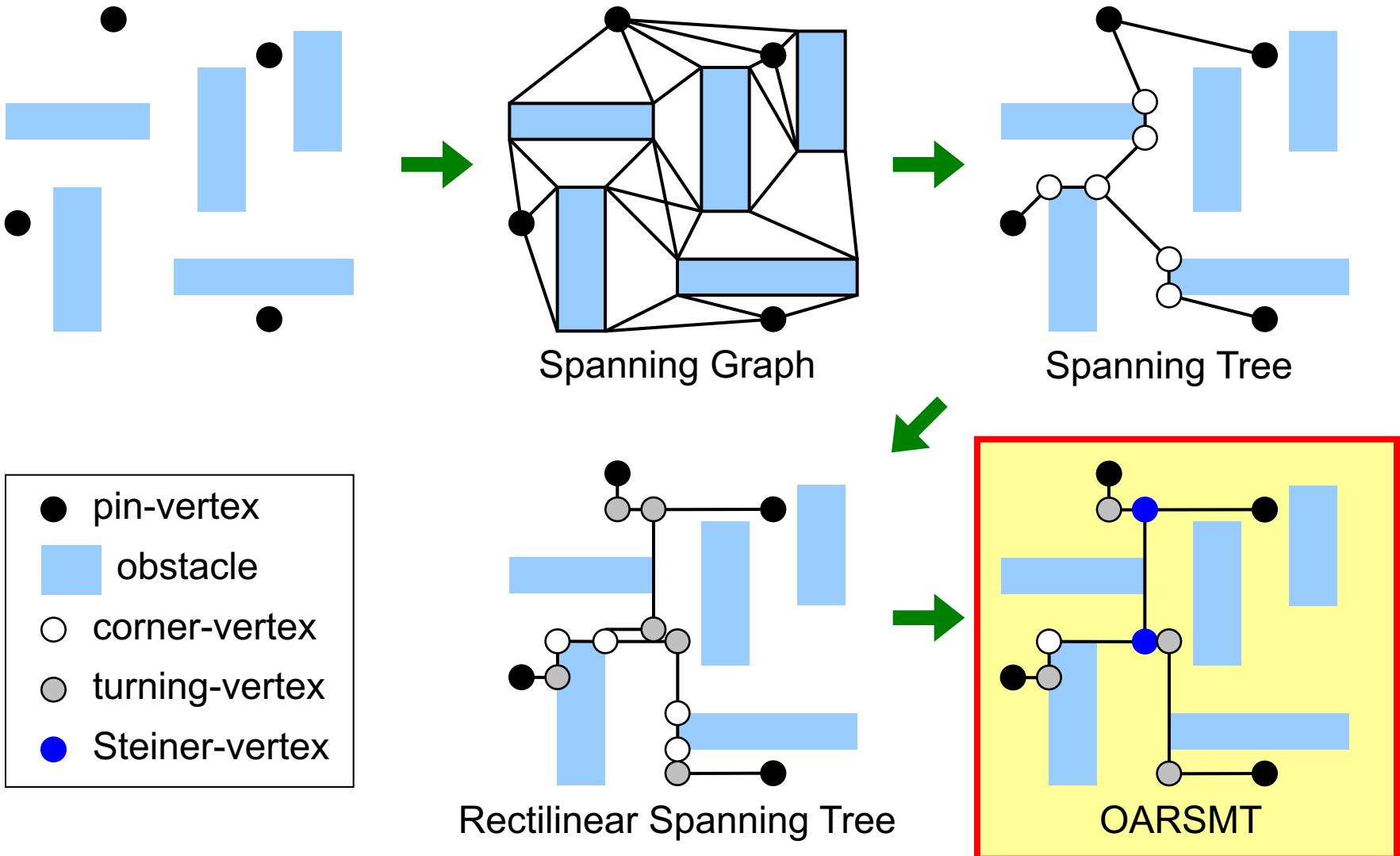
2. They are in neighboring regions.



3. They are in the same region.

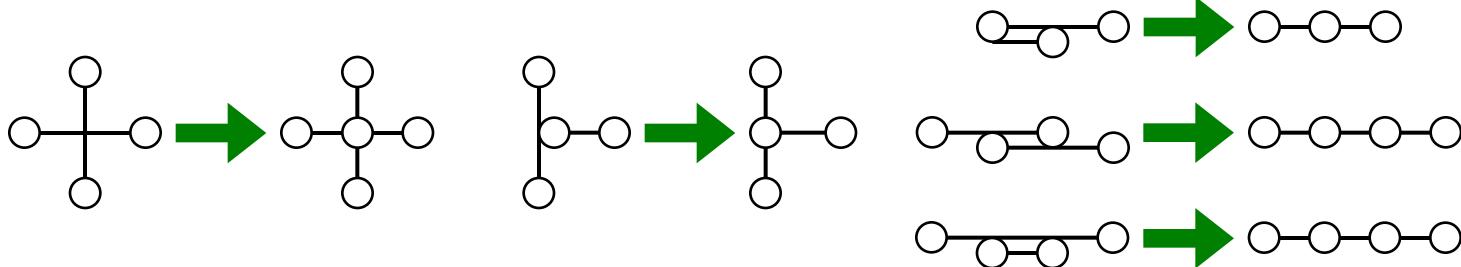


Algorithm Flow



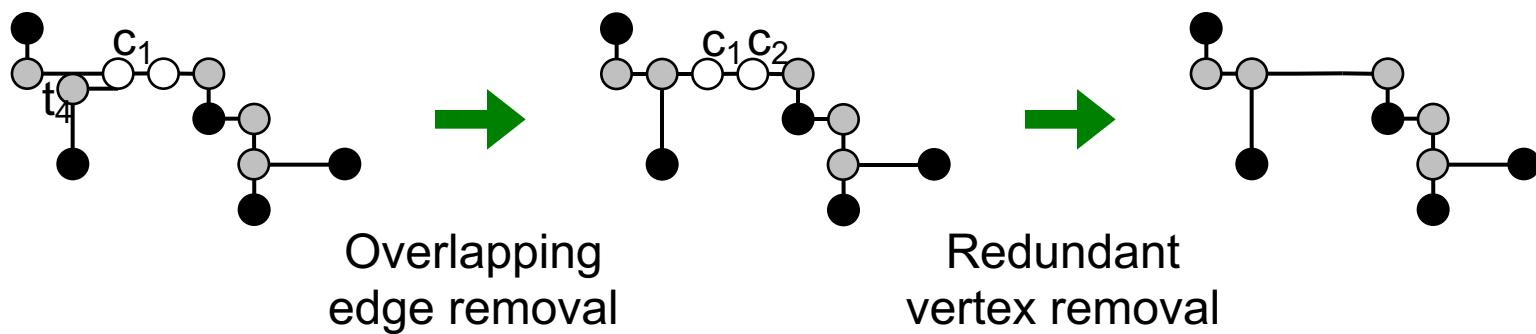
OARSMT Construction (1/2)

- Overlapping edge removal



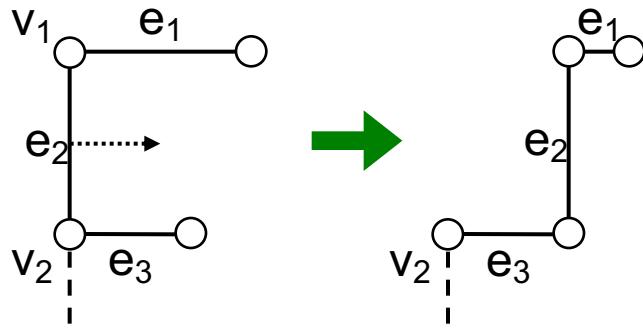
- Redundant vertex removal

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

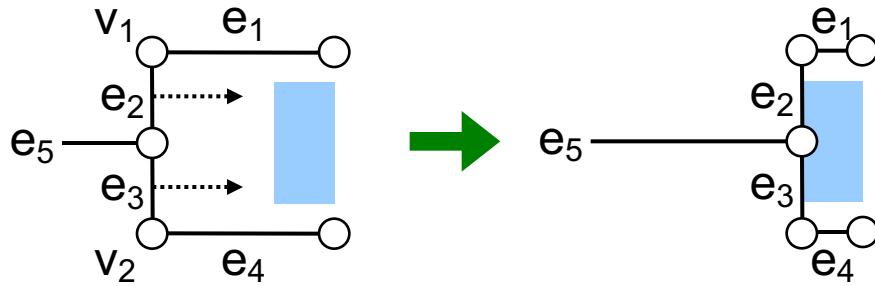


OARSMT Construction (2/2)

- 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:



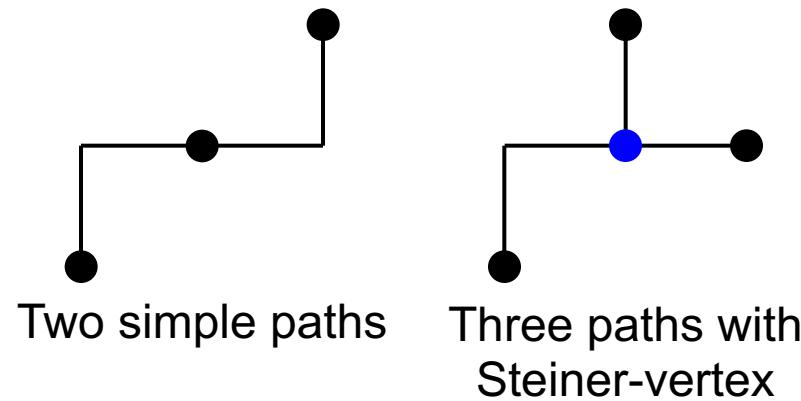
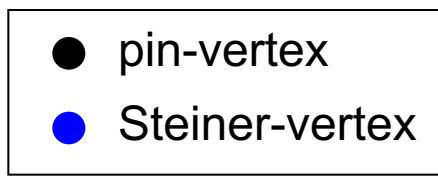
One of the vertices v_1 and v_2 must satisfy the refinement rule.



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

The diagram illustrates the four stages of spanning tree construction:

- Spanning Graph Construction:** A set of blue rectangles representing obstacles and black dots representing vertices. A complex graph structure is overlaid, connecting the vertices.
- Spanning Tree Construction:** The graph from the first stage is simplified into a single spanning tree structure.
- Rectilinear Spanning Tree Construction:** The spanning tree is converted into a rectilinear spanning tree, which follows a grid-like path around the obstacles.
- OARSMT Construction:** The final step, shown with a blue dot indicating the start vertex, where the rectilinear spanning tree is refined or optimized.

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

