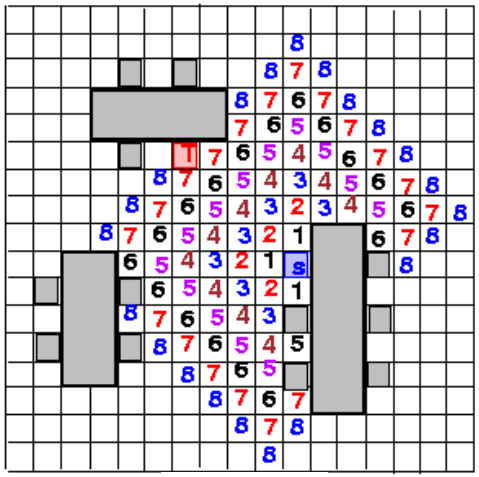
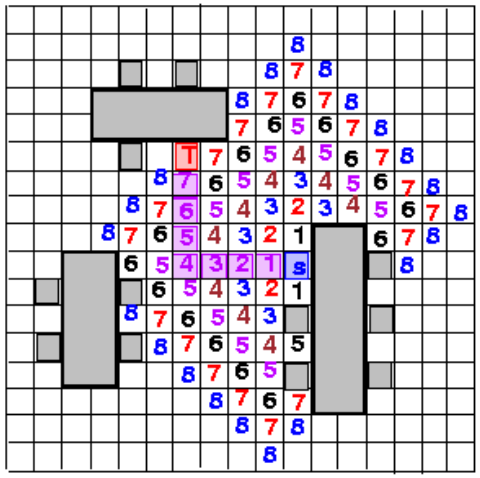
Unit 6: Global and Maze Routing

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



Filling



Retrace

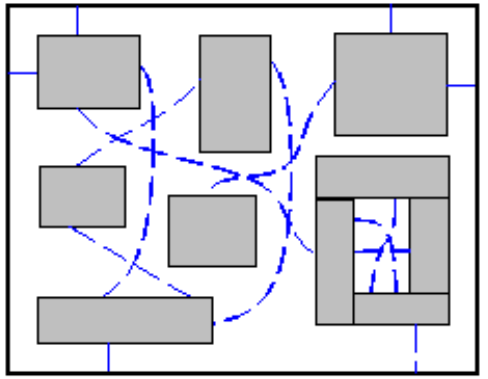
Courtesy of Prof. Y.-W. Chang

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.

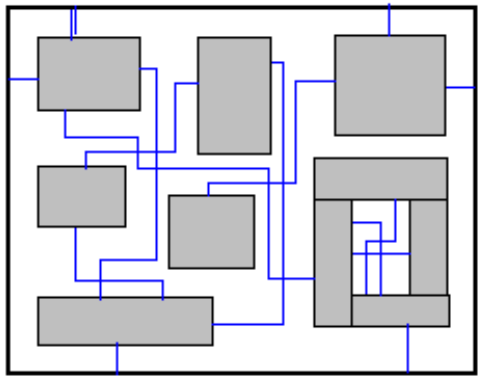
global routing



Global routing

detailed routing

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



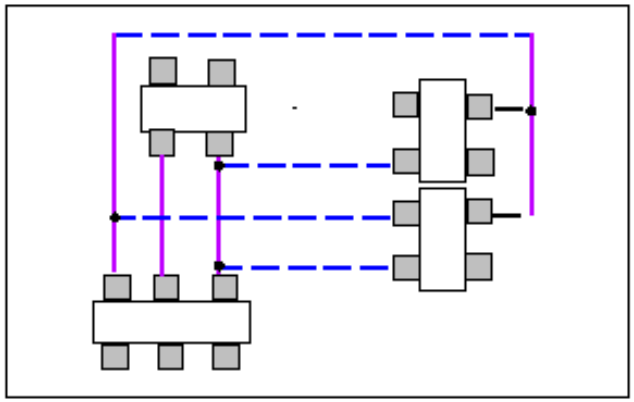
Detailed routing

compaction

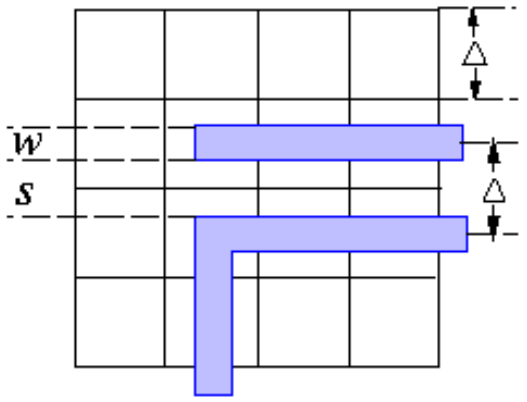
Courtesy of Prof. Y.-W. Chang

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?

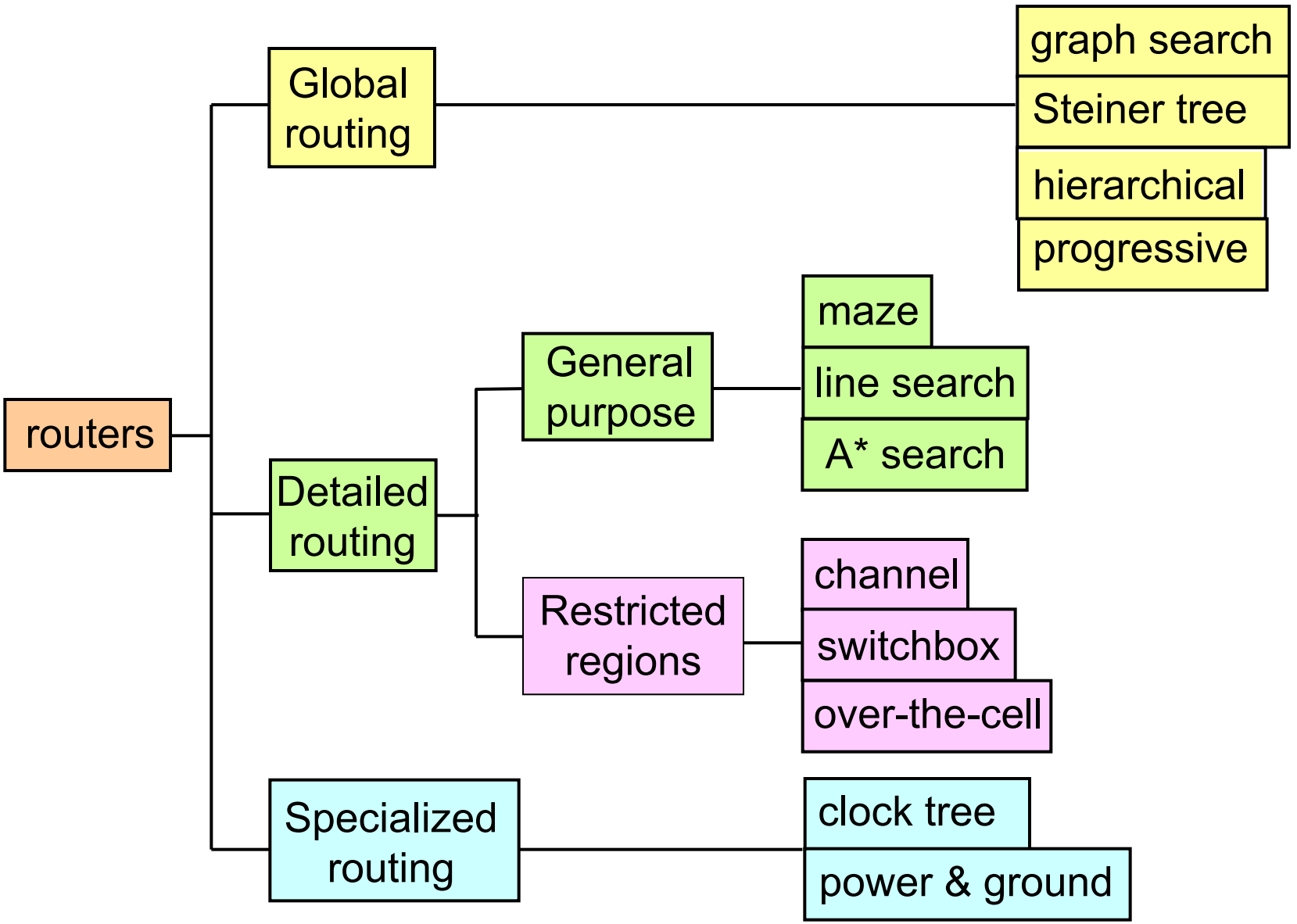


Two–layer routing



Geometrical constraint

Classification of Routers



Courtesy of Prof. Y.-W. Chang

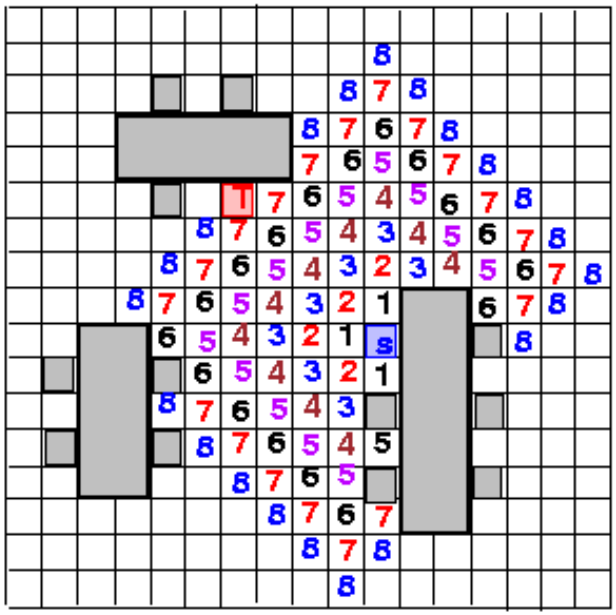
Maze Router: Lee Algorithm

- 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

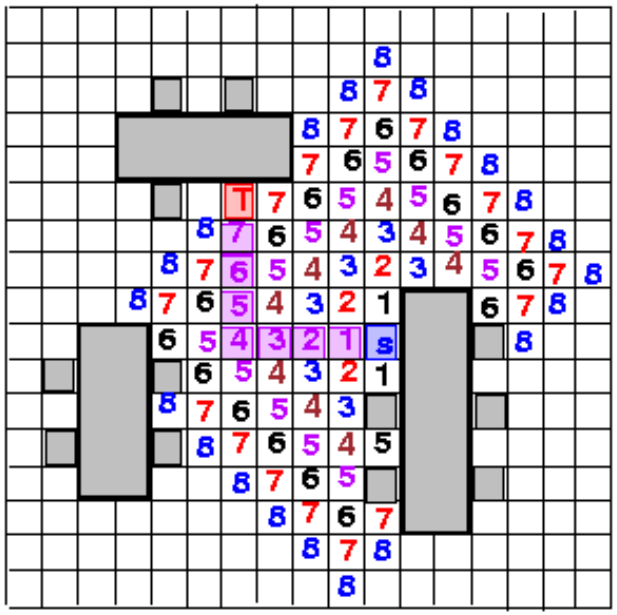
Courtesy of Prof. Y.-W. Chang

Lee Algorithm

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



Filling

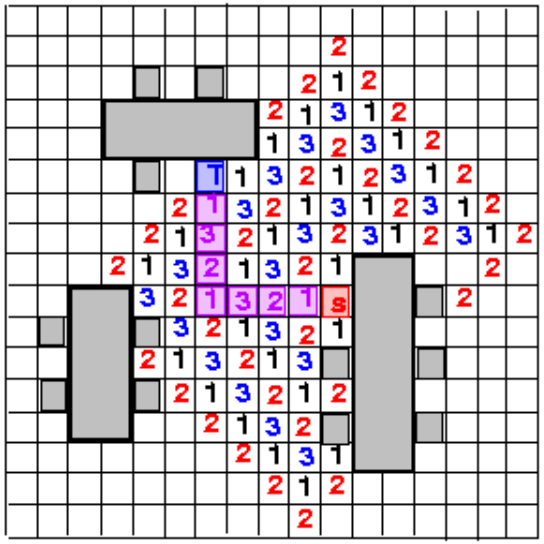


Retrace

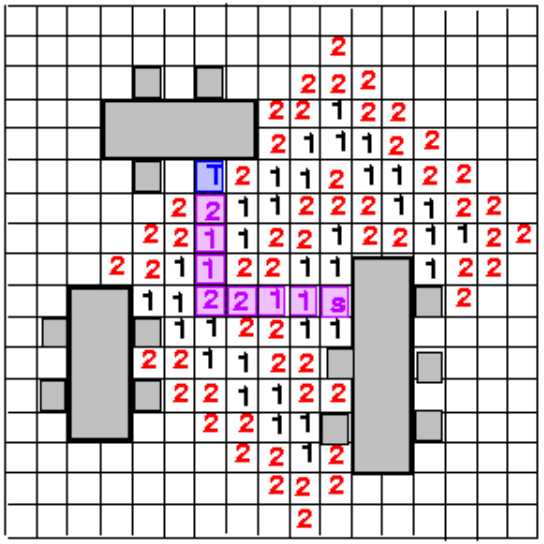
- Time & space complexity for an $M \times N$ grid: $O(MN)$ (**huge!**)

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, ...

Courtesy of Prof. Y.-W. Chang

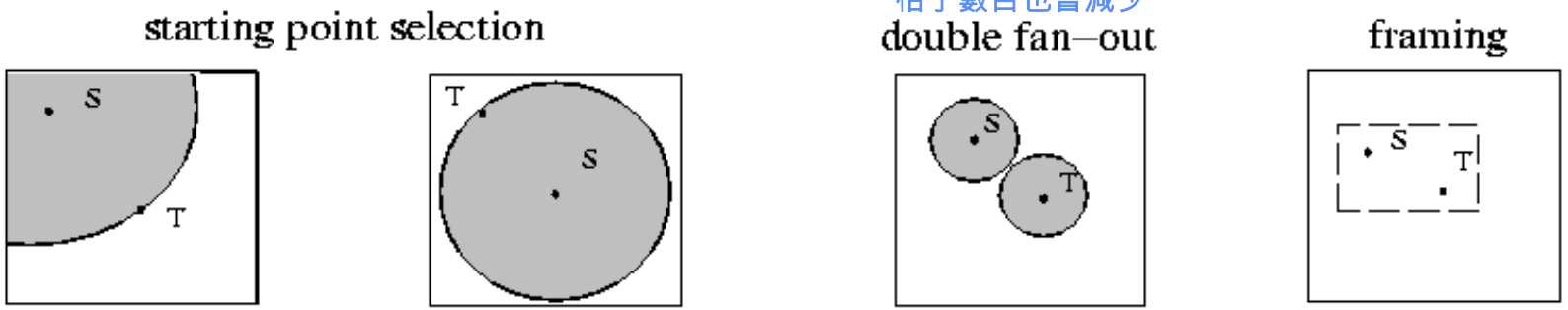
Reducing Running Time

- 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.

左邊所需要填的格子數目比較少，因此速度會較快

兩邊輪流擴散，填的格子數目也會減少

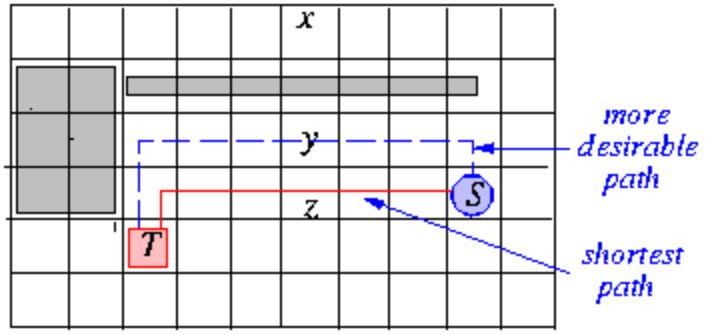
路徑可能不存在此frame中，會導致routing fail(可能可以在一開始沒什麼線的時候用)

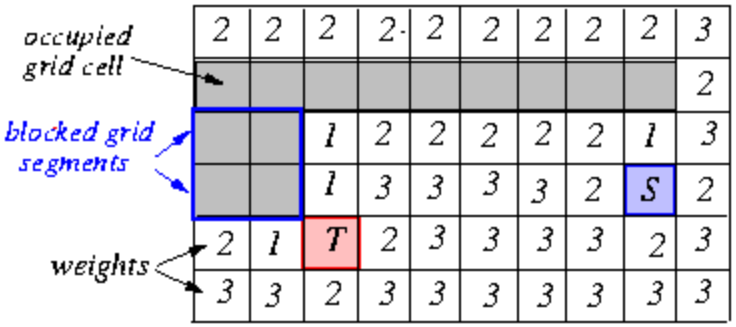


Routing on a Weighted Grid

- Motivation: finding more desirable paths
- *weight*(*grid cell*) = *# of unblocked grid cell segments* - 1

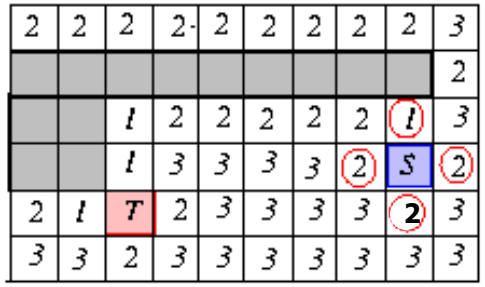
越靠近障礙物的weight越小，因此走線會傾向貼著障礙物走(總weight越小越好)



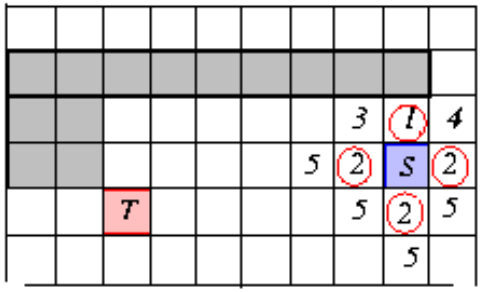


Courtesy of Prof. Y.-W. Chang

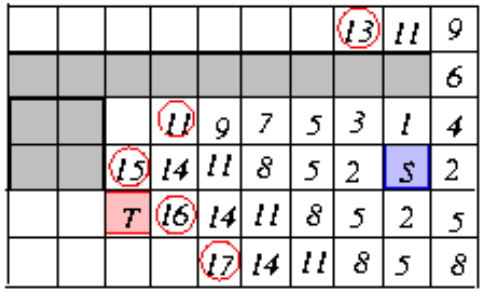
Example Routing on a Weighted Grid

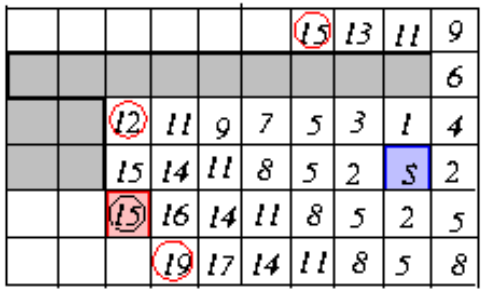


initialize cell weights

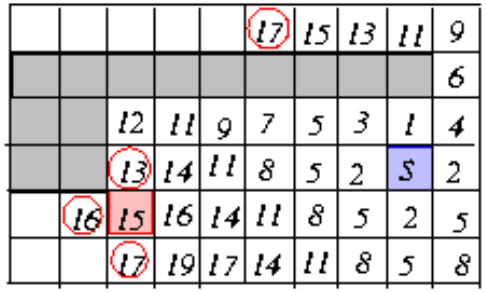


wave propagation

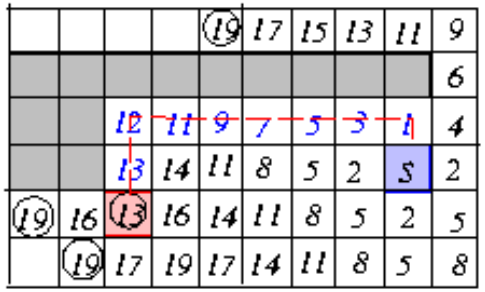




first wave reaches the target



finding other paths



min−cost path found

Courtesy of Prof. Y.-W. Chang

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.

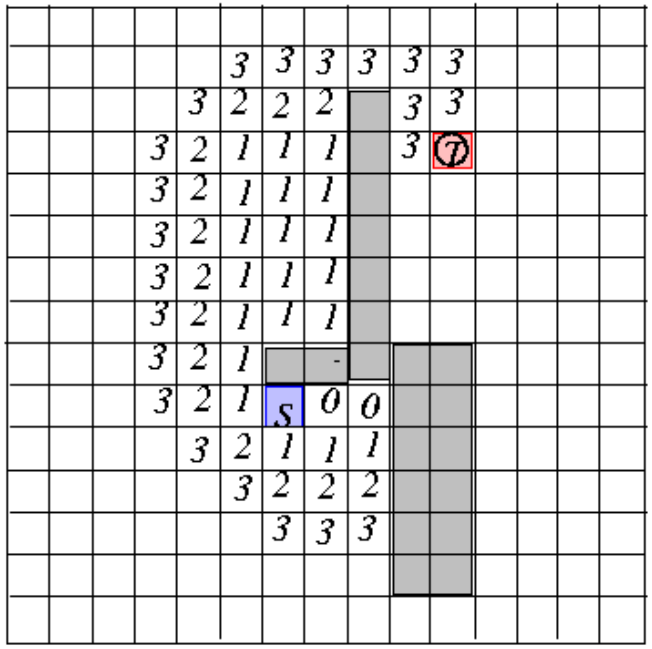
優化目標: 減少detour number(繞遠路的所浪費的距離)

即detour
why 2倍? 可能是因為要額外存一張能trace back的map

Courtesy of Prof. Y.-W. Chang

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.

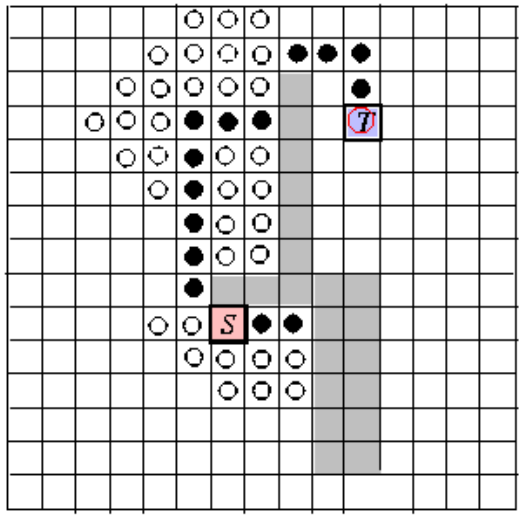


往target擴散，detour就維持原本的數字，
若往反方向，則每步detour都會+1
-->target方面: weight = 0
-->target反向: weight = 1

Courtesy of Prof. Y.-W. Chang

Soukup's Algorithm

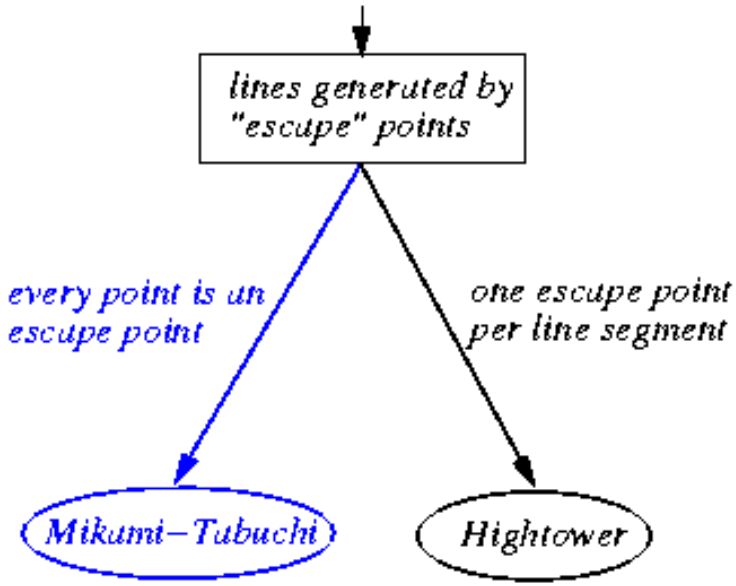
- Soukup, “Fast maze router,” DAC-78.
- Combined breadth-first and depth-first search.
  - Depth-first (**line**) search is first directed toward target *T* until an obstacle or *T* is reached.
  - Breadth-first (Lee-type) search is used to “bubble” around an obstacle if an obstacle is reached.
- Time 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!

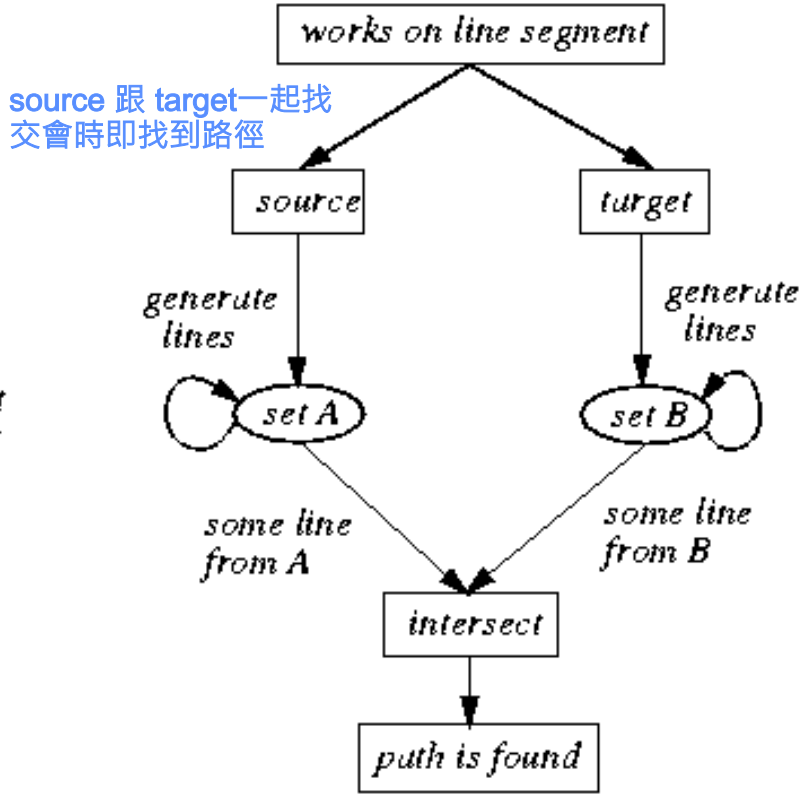


往target方向用DFS，撞到障礙物或是往target反方向時用BFS

Courtesy of Prof. Y.-W. Chang

Features of Line-Search Algorithms

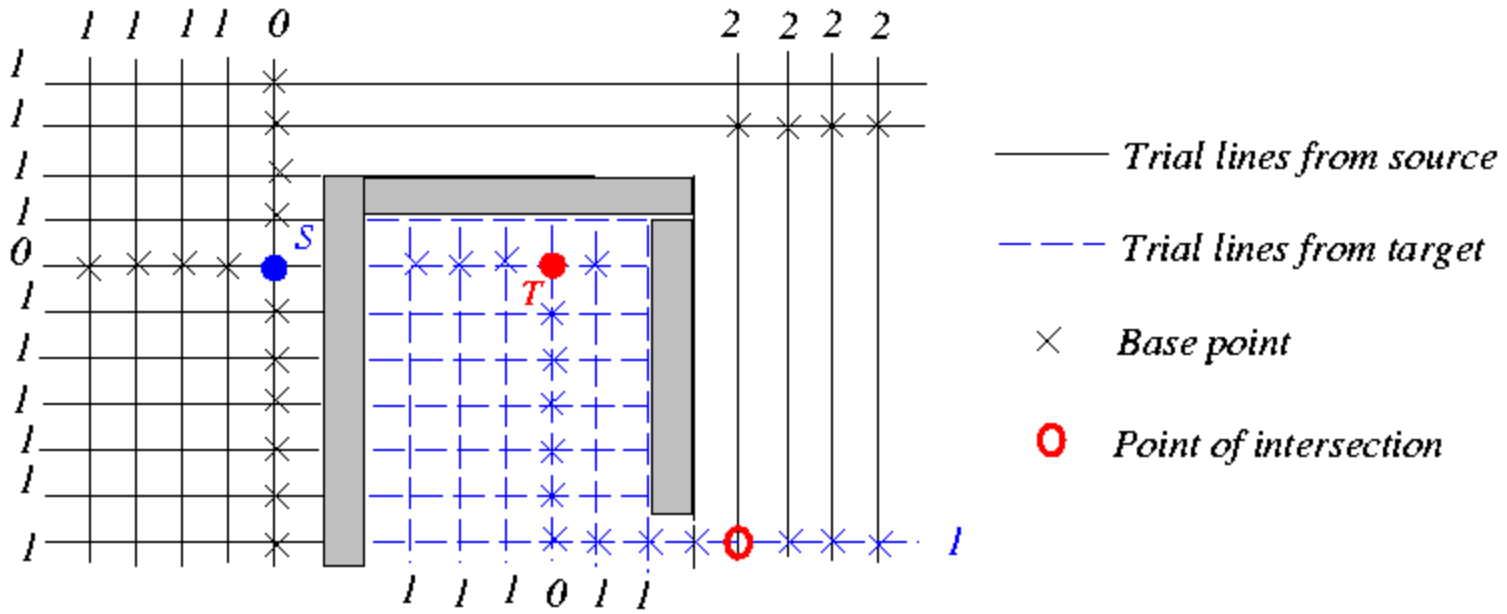




- Time and space complexities: $O(L)$, where L is the # of line segments generated. 一次看一條線

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.

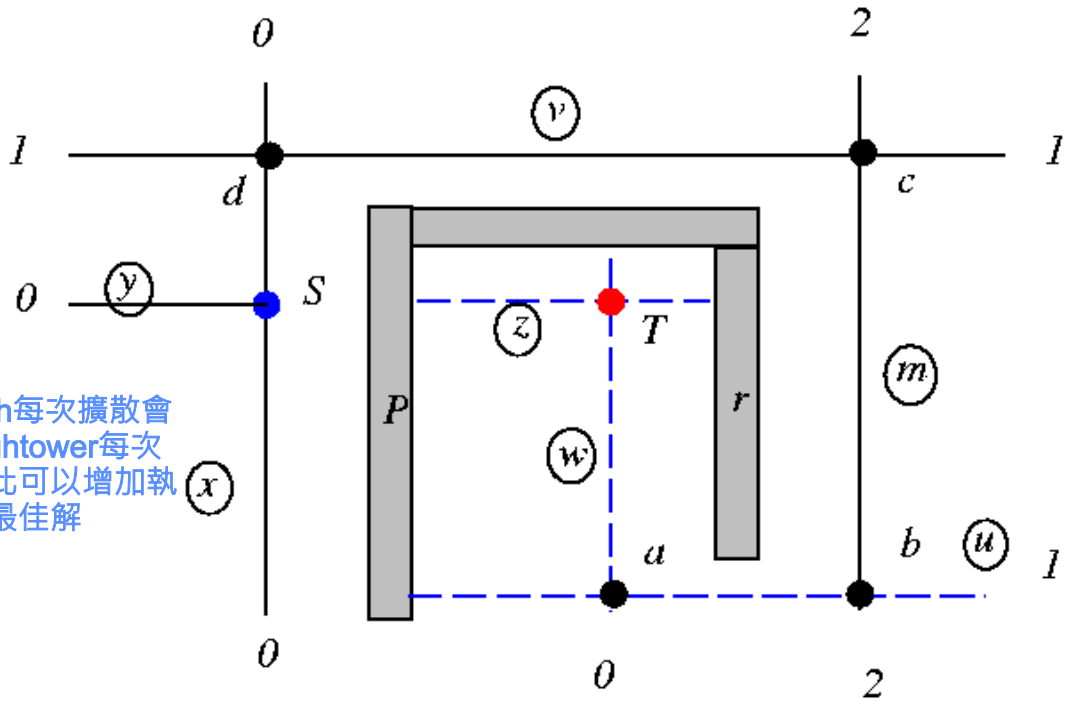


0: source的水平與鉛直線
1: 與0的線相互垂直的線
除了從source出發，也能從target出發

Courtesy of Prof. Y.-W. Chang

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.



相較於一開始的line search每次擴散會有很多選擇需要檢查，Hightower每次擴散都只看一種選擇，如此可以增加執行速度，但可能會找不到最佳解

Courtesy of Prof. Y.-W. Chang

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.

A*-Search Routing

- Maze search is also called **blind search** since it searches the routing region in a blind way.
- A*-search is also called the **best-first search**
  - 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
- Good property:
  - If $h(x)$ is *admissible* (never overestimates the actual cost from the current node to the target), then A*-search is optimal

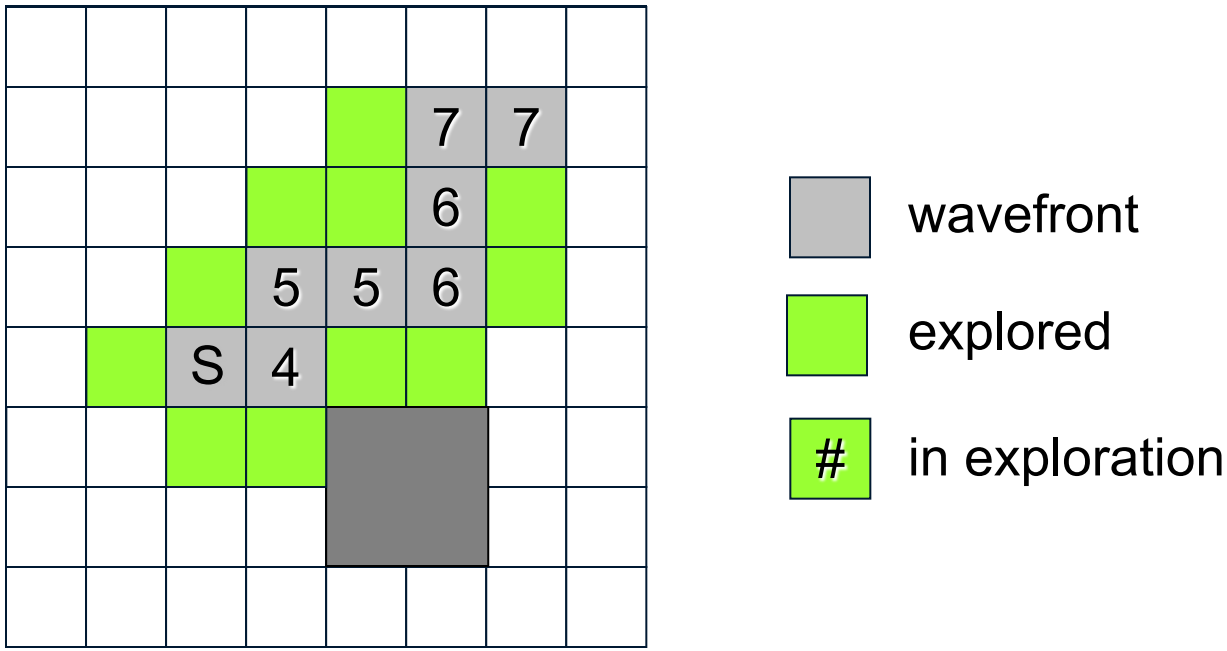
f(x) = g(x) + h(x)

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

lee's algorithm

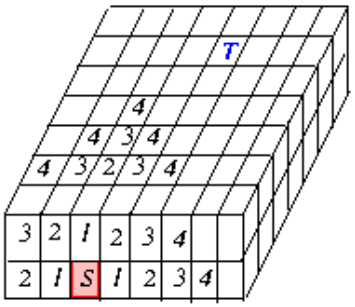
Example of A*-Search Routing

- 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 lable 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

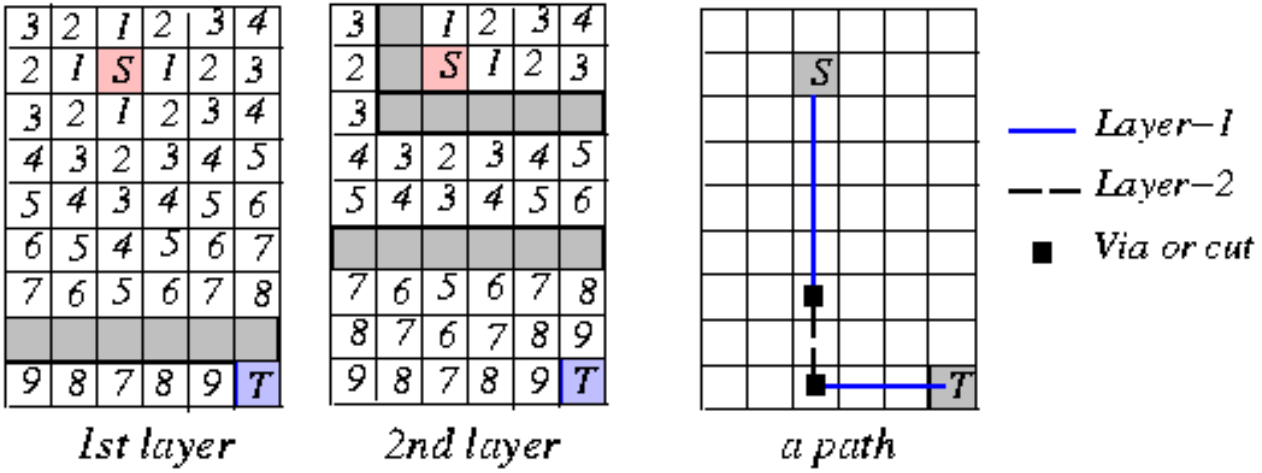


Multi-layer Routing

- 3-D grid:



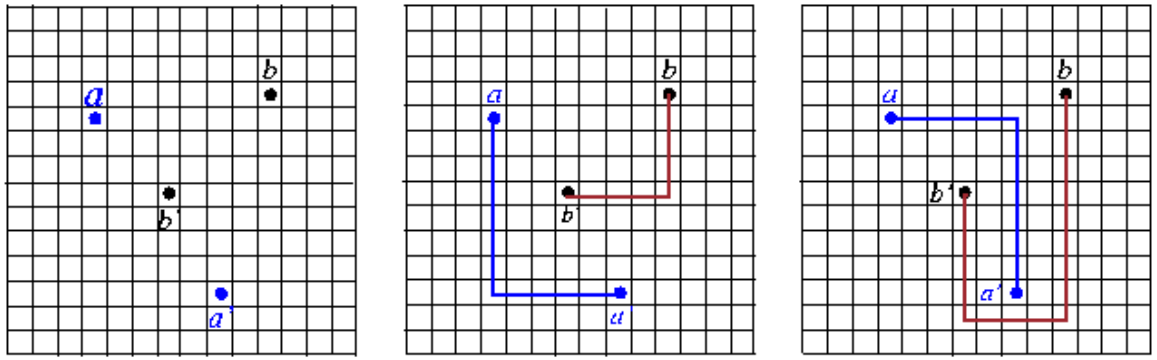
- Two planner arrays:
  - Neglect the weight for inter-layer connection through via.
  - Pins are accessible from both layers.



Courtesy of Prof. Y.-W. Chang

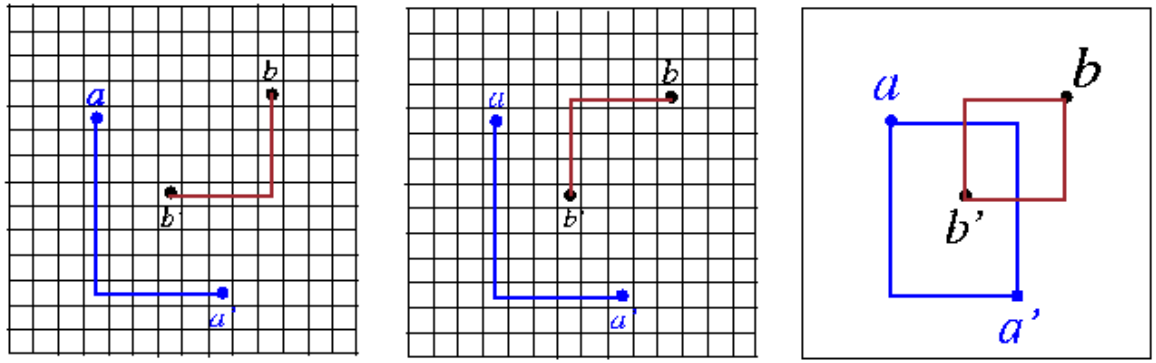
Net Ordering

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



route net *a* before net *b*

把兩條net都框出其bounding box，若bounding box有包到其他人的pin的話，被包住的那個人要先繞

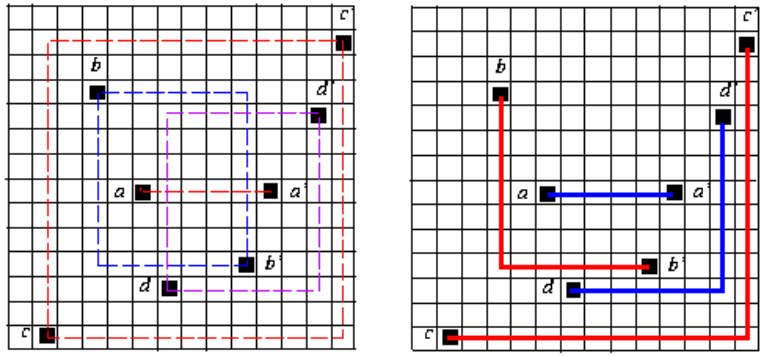


route net *b* before net *a*

Courtesy of Prof. Y.-W. Chang

Net Ordering (cont'd)

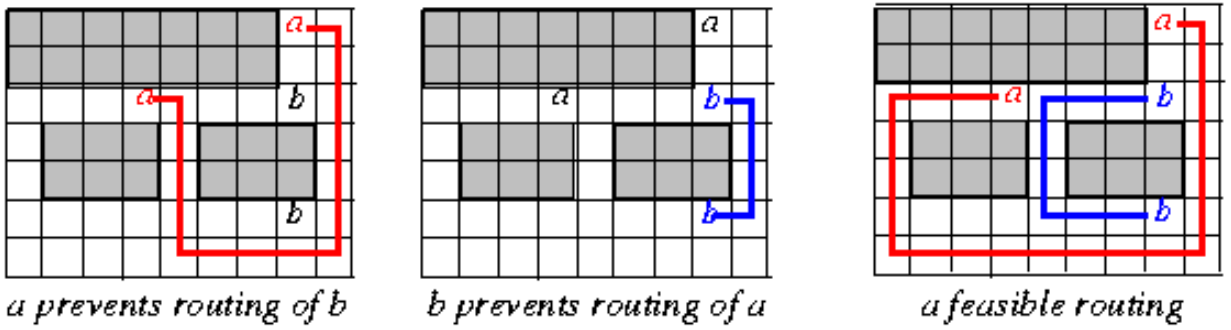
- Order the nets in the ascending order of the # of pins within their bounding boxes.
- Order the nets in the ascending (or descending??) order of their lengths.
- Order the nets based on their timing criticality.



a沒有包到別人-->先繞
b只有包住a-->第二個繞
d包到a與b-->第三個繞
c包到所有人的pin-->最後繞

- A mutually intervening case:

有時候不能繞最短路徑才能繞出來
完成繞線 >> 繞線長度



Rip-Up and Re-routing

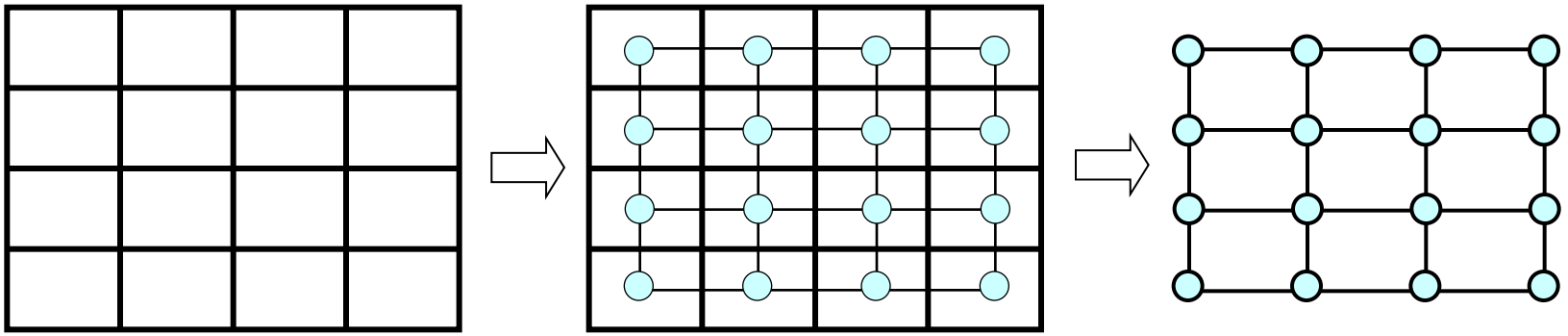
- Rip-up and re-routing is required if a global or detailed router fails in routing all nets.
- Approaches: the manual approach? the automatic procedure?
- Two steps in rip-up and re-routing
  1. Identify bottleneck regions, rip off some already routed nets.
  2. Route the blocked connections, and re-route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

Courtesy of Prof. Y.-W. Chang

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.

格子大小會決定maze routing的時間



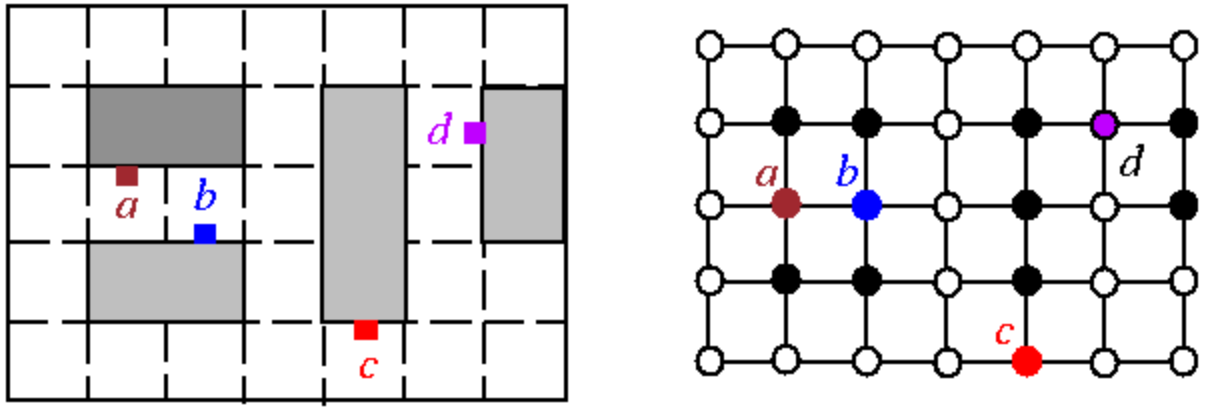
Partitioned Layout

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.



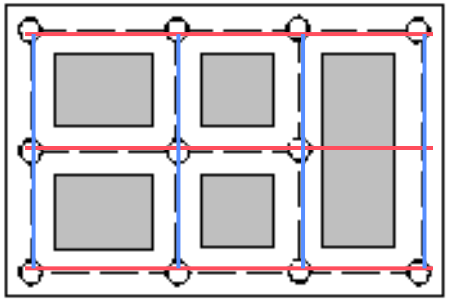
Courtesy of Prof. Y.-W. Chang

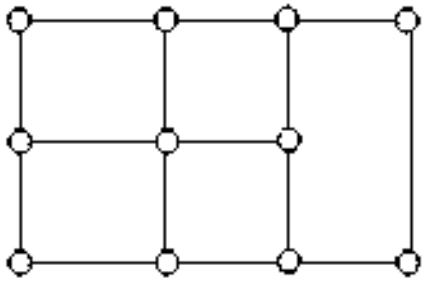
Graph Model: Channel Intersection Graph

- 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.

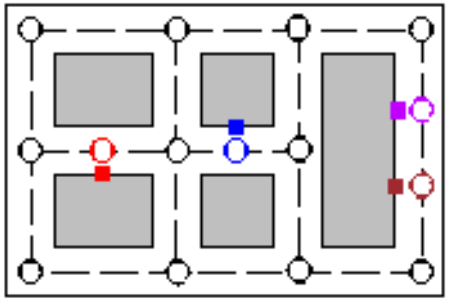
edge 相交的地方才加上vertex(紅藍交界處)
-->vertex數量會變少，可能可以加速maze routing

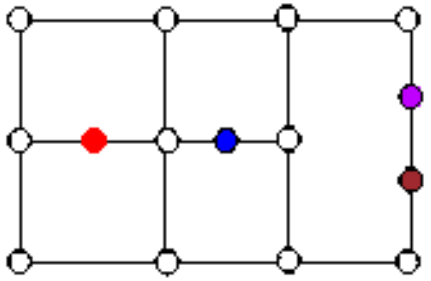
channel intersection graph





extended channel intersection graph





Courtesy of Prof. Y.-W. Chang

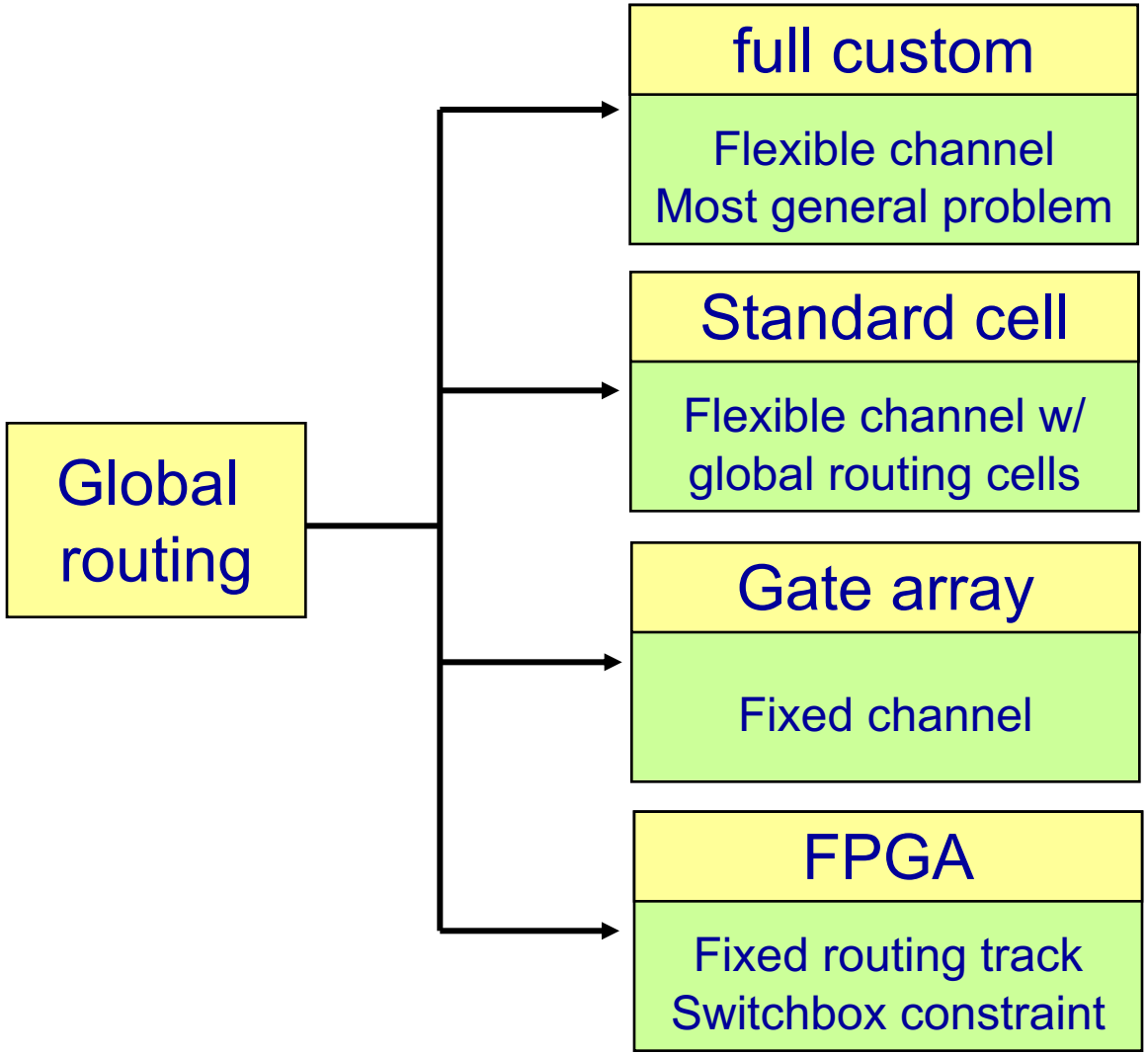
Global-Routing Problem

- 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 capacity constraint
  U: 當前vertex通過的edge數量
  - $c(e_j)$: 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). 一些高速晶片需要盡量縮短最長的繞線距離

Courtesy of Prof. Y.-W. Chang

Global Routing in different Design Styles

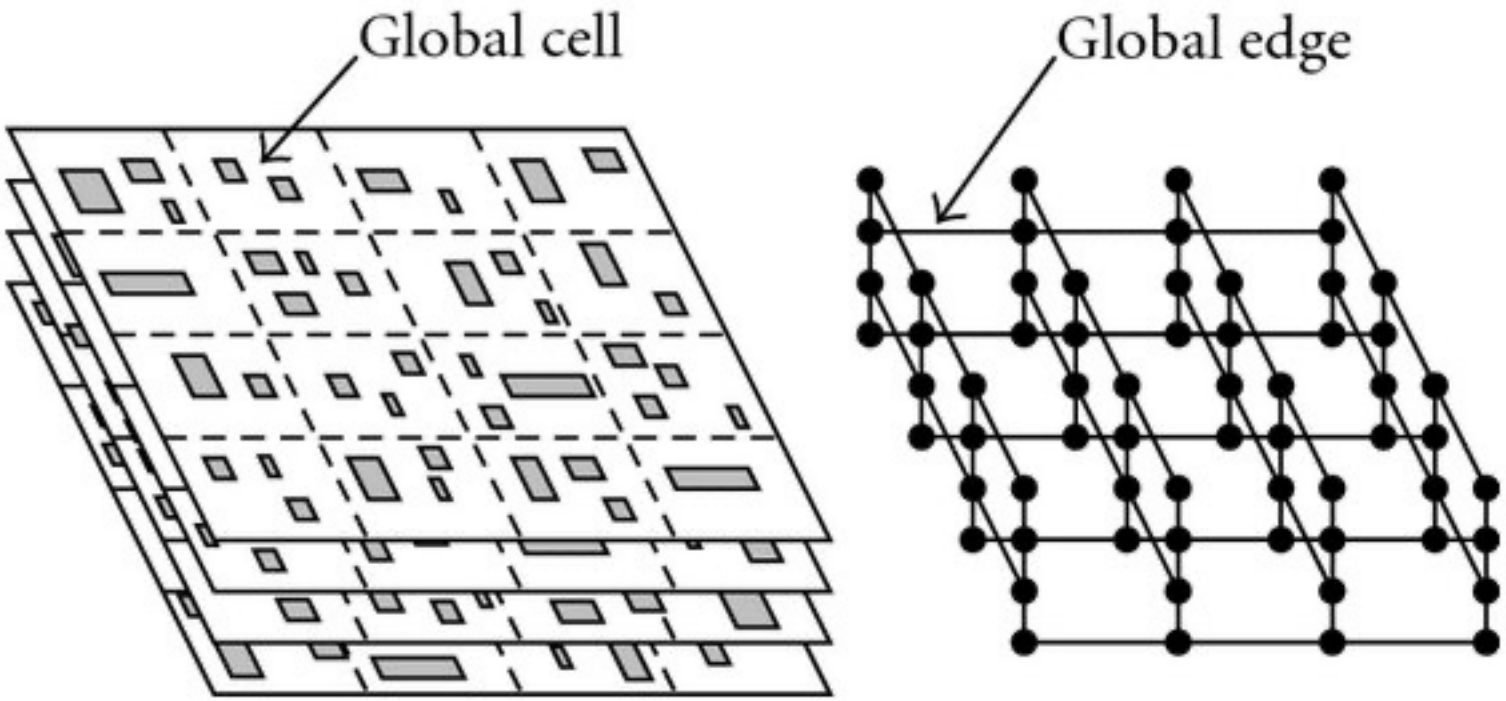


Courtesy of Prof. Y.-W. Chang

Global Routing in Cell-based Design

- Objective
  - Minimize the maximum wire length.
- For high performance,
  - Minimize the maximum path length.

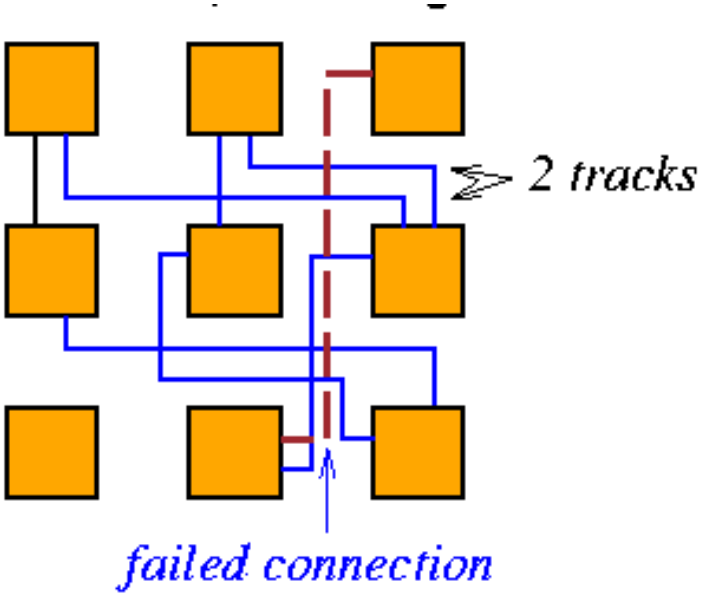
每個G-Cell就是一個vertex
相鄰的vertex間會有一個edge



Courtesy of Prof. Y.-W. Chang

Global Routing in Gate Array

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

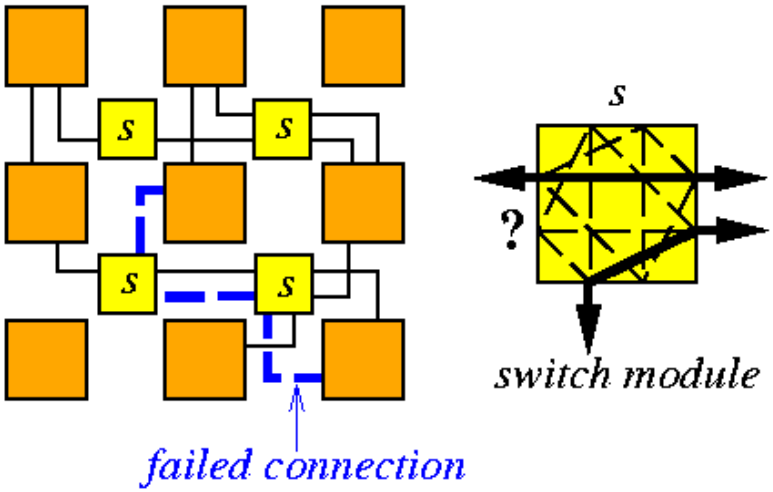


Each channel has a capacity of 2 tracks.

Courtesy of Prof. Y.-W. Chang

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.

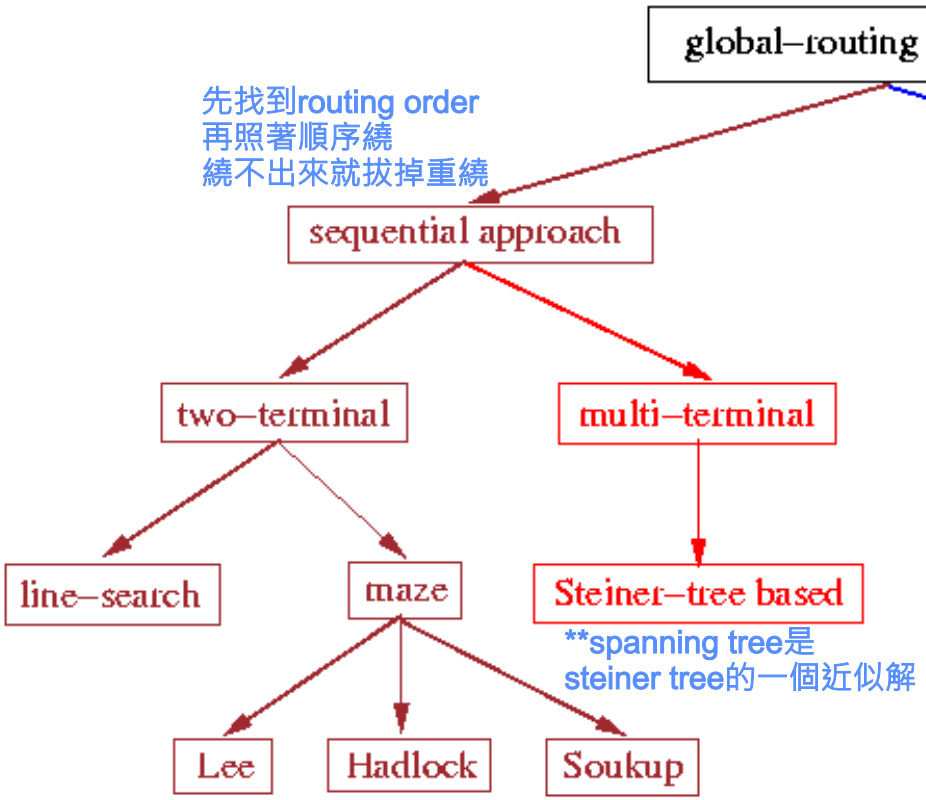


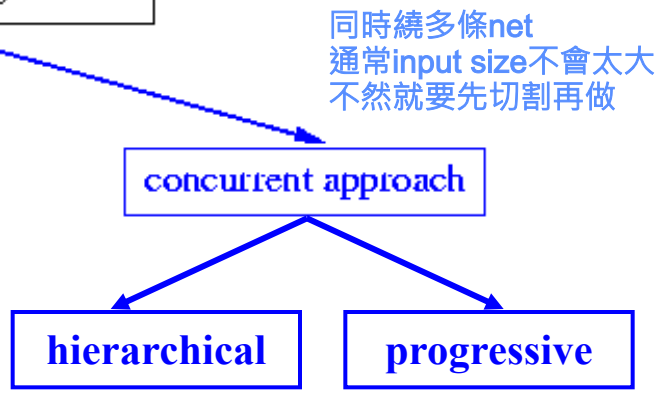
Each channel has a capacity of 2 tracks.

Courtesy of Prof. Y.-W. Chang

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?)



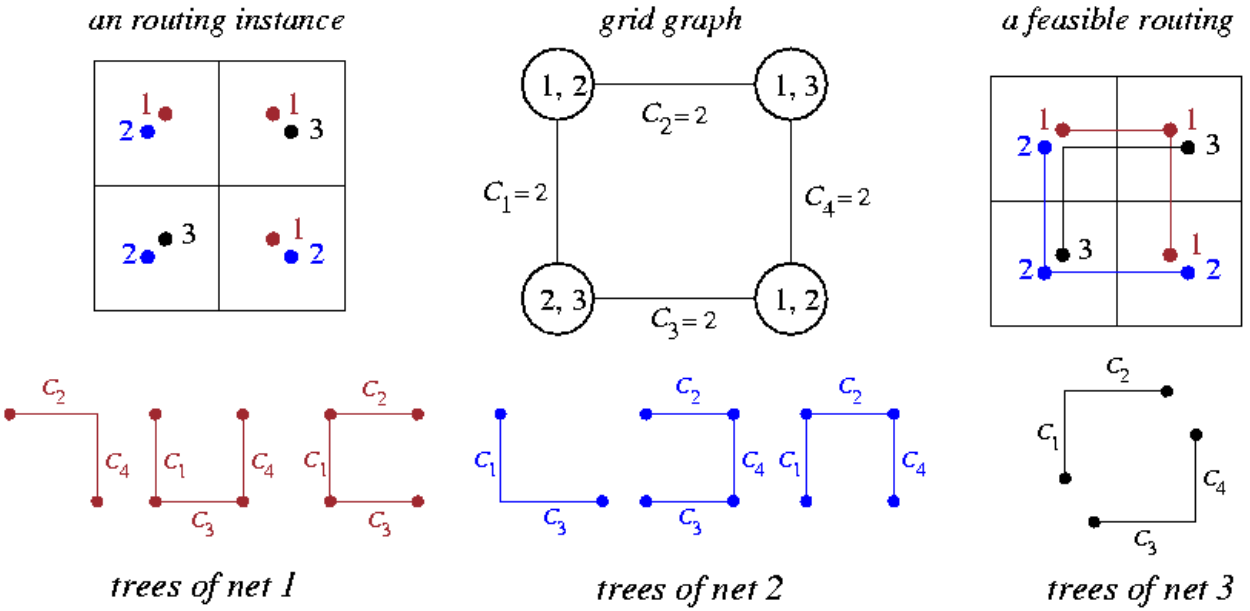


- Integer linear programming
- Multi-commodity flow

Courtesy of Prof. Y.-W. Chang

Global Routing by Integer Programming

- Suppose that for each net i, there are n_i possible trees $t_1^i, t_2^i, \ldots, t_{ni}^i$ to route the net. 每條net有已知的幾種solution，去找出最適合的組合
- Constraint I: For each net i, only one tree t_j^i will be selected. 每條net一定炫選出其中一組組合
- Constraint II: The capacity of each cell boundary c_i is not exceeded.
- Minimize the total tree cost. 繞線不能超過vertex的capacity
- **Question:** Feasible for practical problem sizes?
  - **Key:** Hierarchical approach!



Courtesy of Prof. Y.-W. Chang

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.

將前頁中的constraint轉換成數學式

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 x11,x12,...都是前一頁中的某一種solution
比如說x11代表net1的第一種solution，若x11=0，就代表我們沒選他

從net1的三種解中挑出一種解

$$x_{1,1} + x_{1,2} + x_{1,3} = 1 \quad (Constraint\ I : t^1)$$

$$x_{2,1} + x_{2,2} + x_{2,3} = 1 \quad (Constraint\ I : t^2)$$

$$x_{3,1} + x_{3,2} = 1 \quad (Constraint\ I : t^3)$$

把會經過C1的所有solution相加
不能超過C1的constraint

$$x_{1,2} + x_{1,3} + x_{2,1} + x_{2,3} + x_{3,1} \le 2 \quad (Constraint\ II : B1)$$

$$x_{1,1} + x_{1,3} + x_{2,2} + x_{2,3} + x_{3,1} \le 2 \quad (Constraint\ II : B2)$$

$$x_{1,2} + x_{1,3} + x_{2,1} + x_{2,2} + x_{3,2} \le 2 \quad (Constraint\ II : B3)$$

$$x_{1,1} + x_{1,2} + x_{2,2} + x_{2,3} + x_{3,2} \le 2 \quad (Constraint\ II : B4)$$

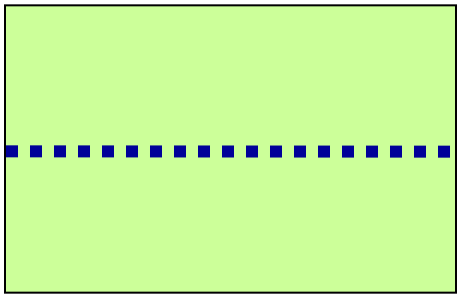
$$x_{i,j} = 0, 1, 1 \le i, j \le 3$$

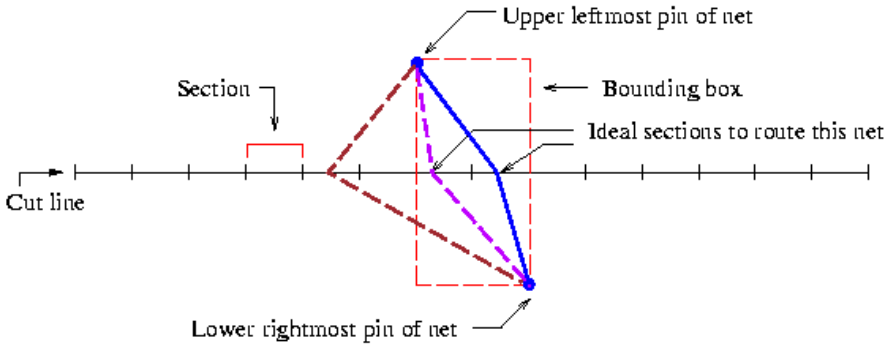
解完之後就會知道我們該挑每條net的哪一種解

Courtesy of Prof. Y.-W. Chang

Hierarchical Global Routing

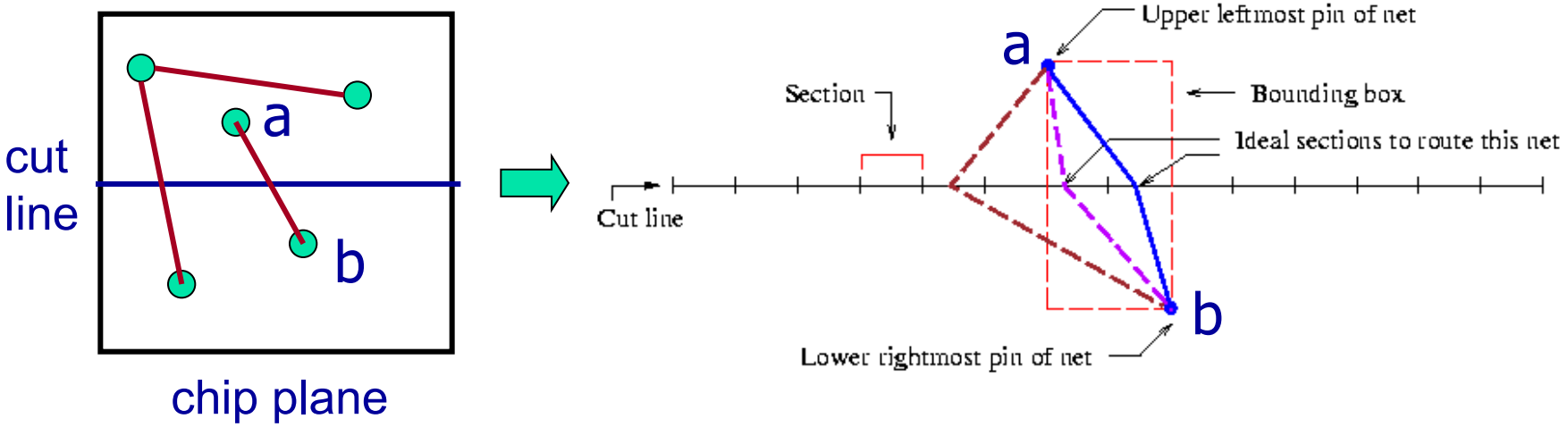
- Marek-Sadowska, “Router planner for custom chip design,” ICCAD-86.
- At each level of the hierarchy, an attempt is made to minimize the cost of nets crossing cut lines.
- At the lowest level of the hierarchy, the layout surface is divided into $R \times R$ grid regions with boundary capacity equal 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: $\sum_{j=1}^{M} x_{ij} = 1, 1 \le i \le N.$
  - Capacity constraint of each section: $\sum_{i=1}^{N} x_{ij} \le C, 1 \le j \le M.$
  - w_{ij}: cost of assigning net *i* to section *j*. Minimize $\sum_{i=1}^{N} \sum_{j=1}^{M} w_{ij} x_{ij}$.

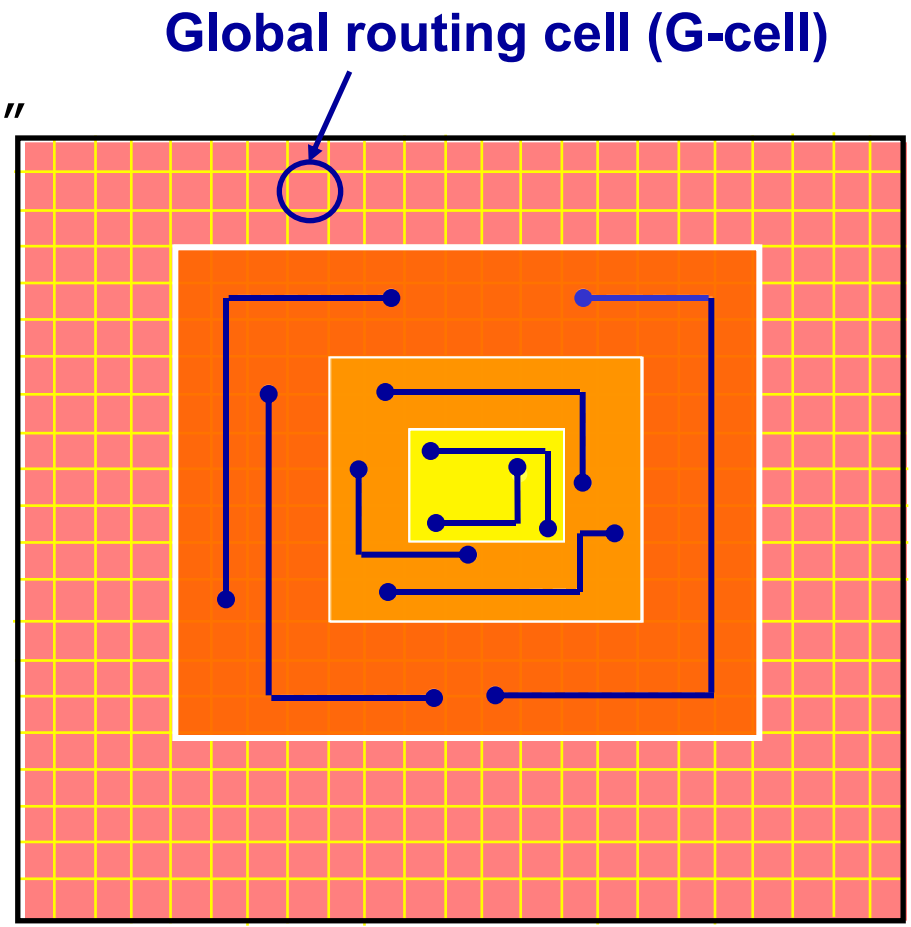


Courtesy of Prof. Y.-W. Chang

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

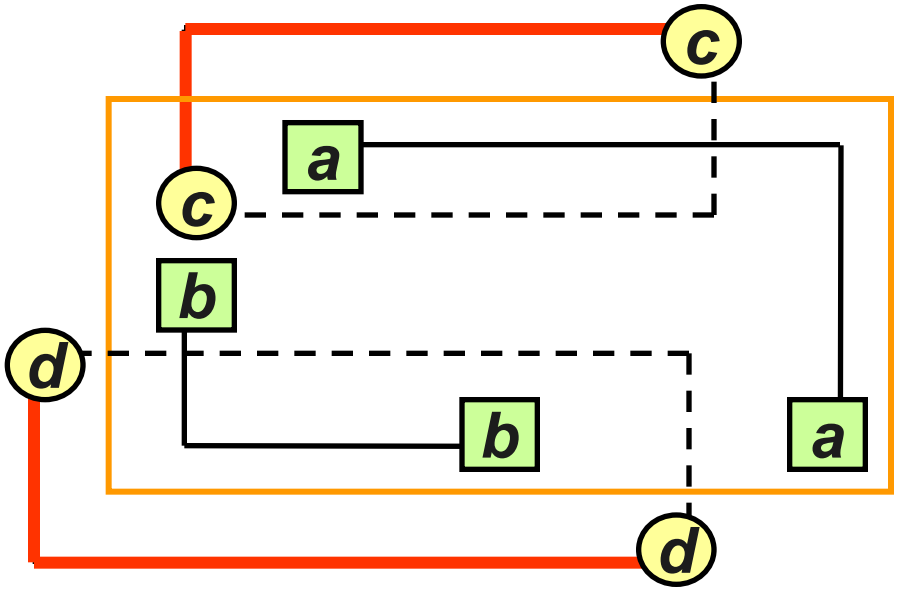
1. Incremental box expansion
   - From the most congested region (擒賊先擒王)
2. Progressive ILP 從蛋黃區開始繞(最多bounding box覆蓋的地方)
   - With adaptive maze routing
3. Post routing
   - Rerouting w/o rip-up



- Try to minimize the total routing overflow

Courtesy of Prof. Y.-W. Chang

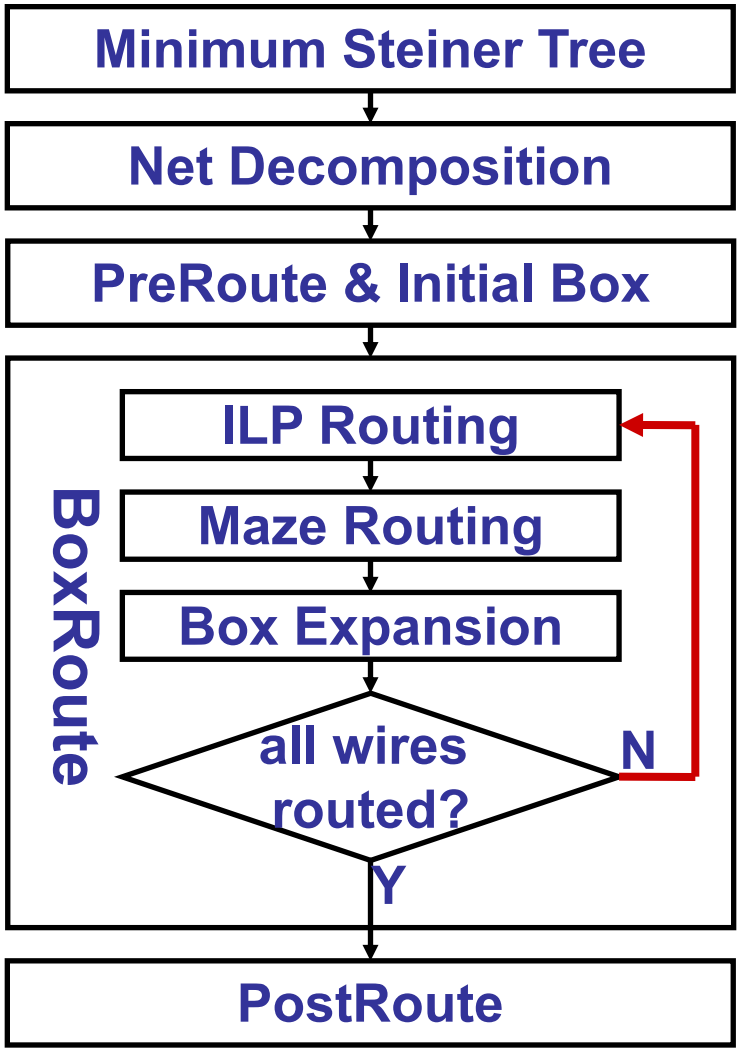
Motivation for Box Expansion



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

Courtesy of Prof. Y.-W. Chang

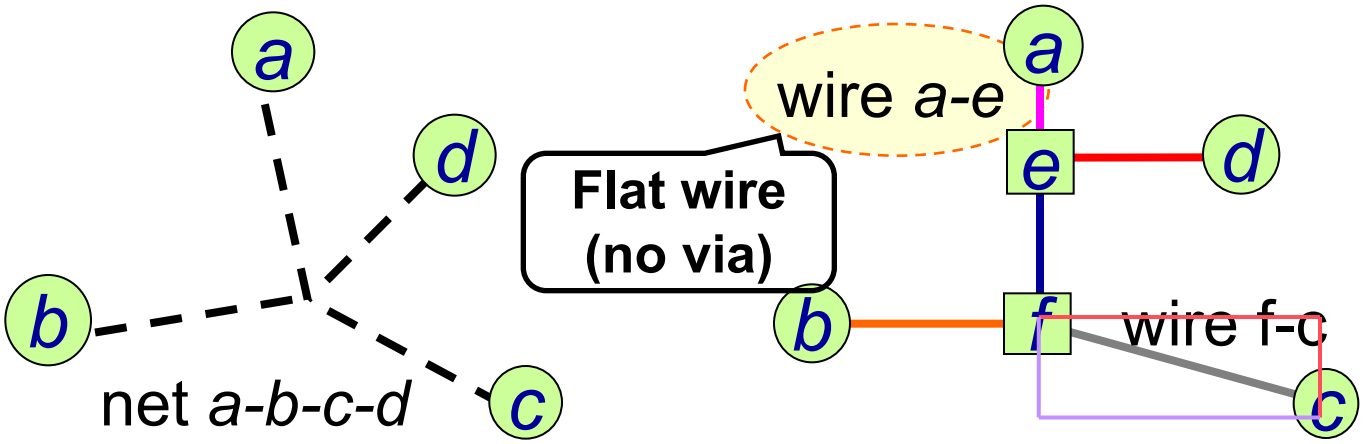
Overall Flow of BoxRouter



Iterative box routing

Courtesy of Prof. Y.-W. Chang

Minimal Steiner Tree & Net Decomposition



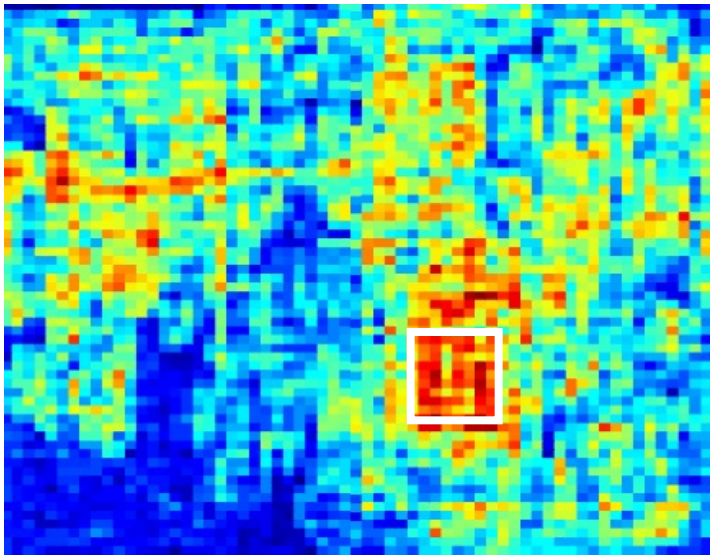
直線的two-pin net才需要ILP，直線的就直接連起來就可以了
只有f-c這種斜線的才會有"上L(紅色)'或是"下L(紫色)"的選擇

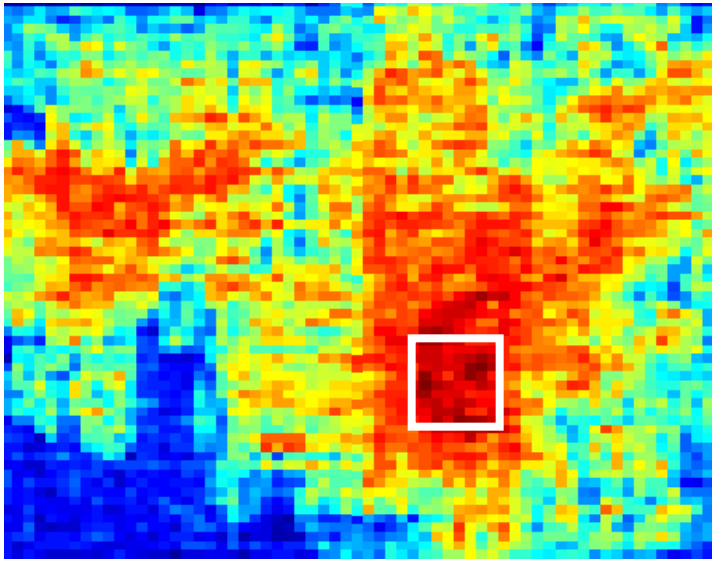
- Minimum Steiner Tree
  - FLUTE [ICCAD-04], Spanning graph [ISPD-03], Geosteiner 3.1
  - Decompose into 2-pin wires
  - Each wire becomes an atomic routing object

Courtesy of Prof. Y.-W. Chang

PreRouting

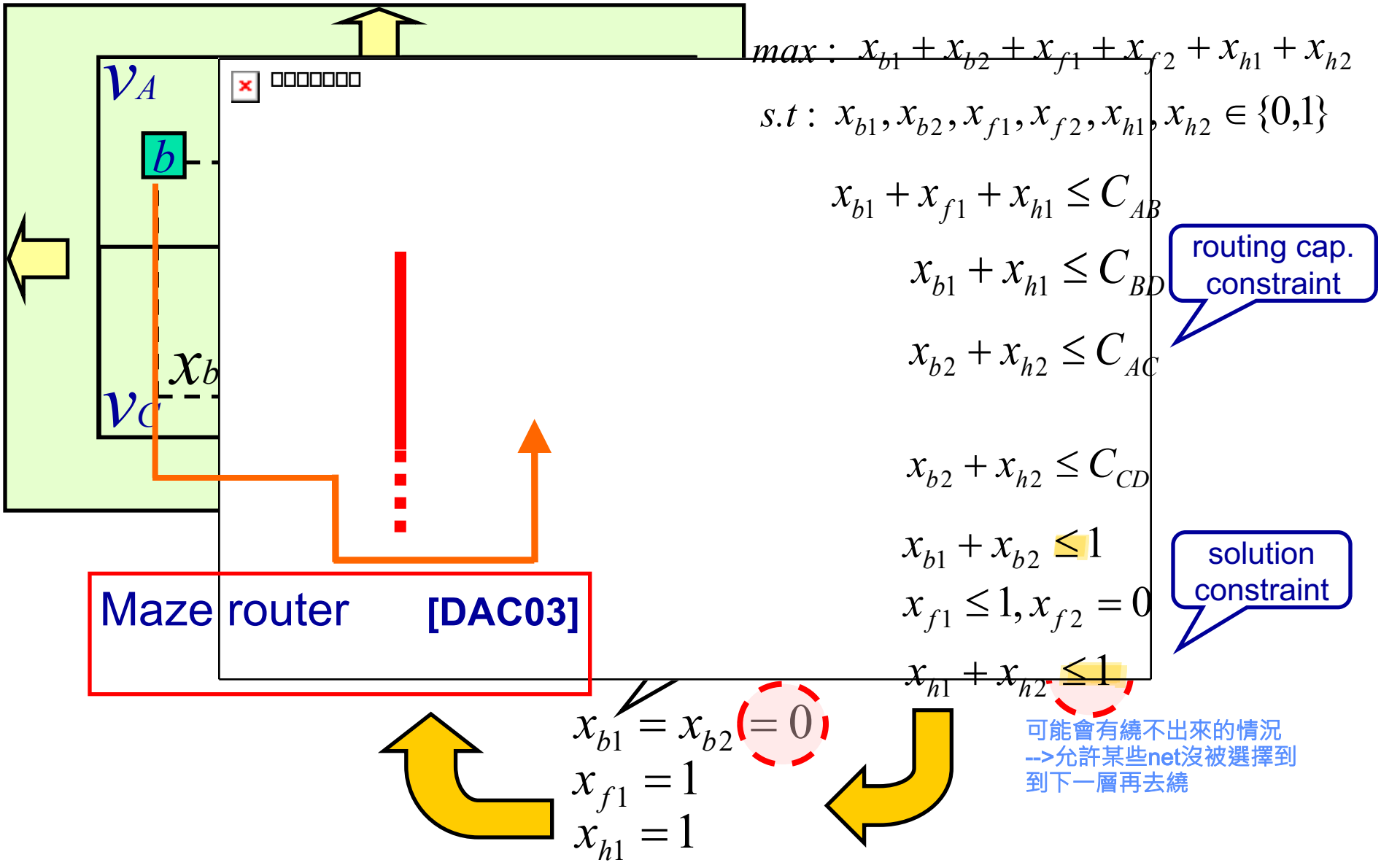
- Preroute as many flat wires as possible
  - 60% of wirelength can be prerouted
- Benefits
  - Overall congestion captured
  - Runtime improved
- Starting “Box” of BoxRouting



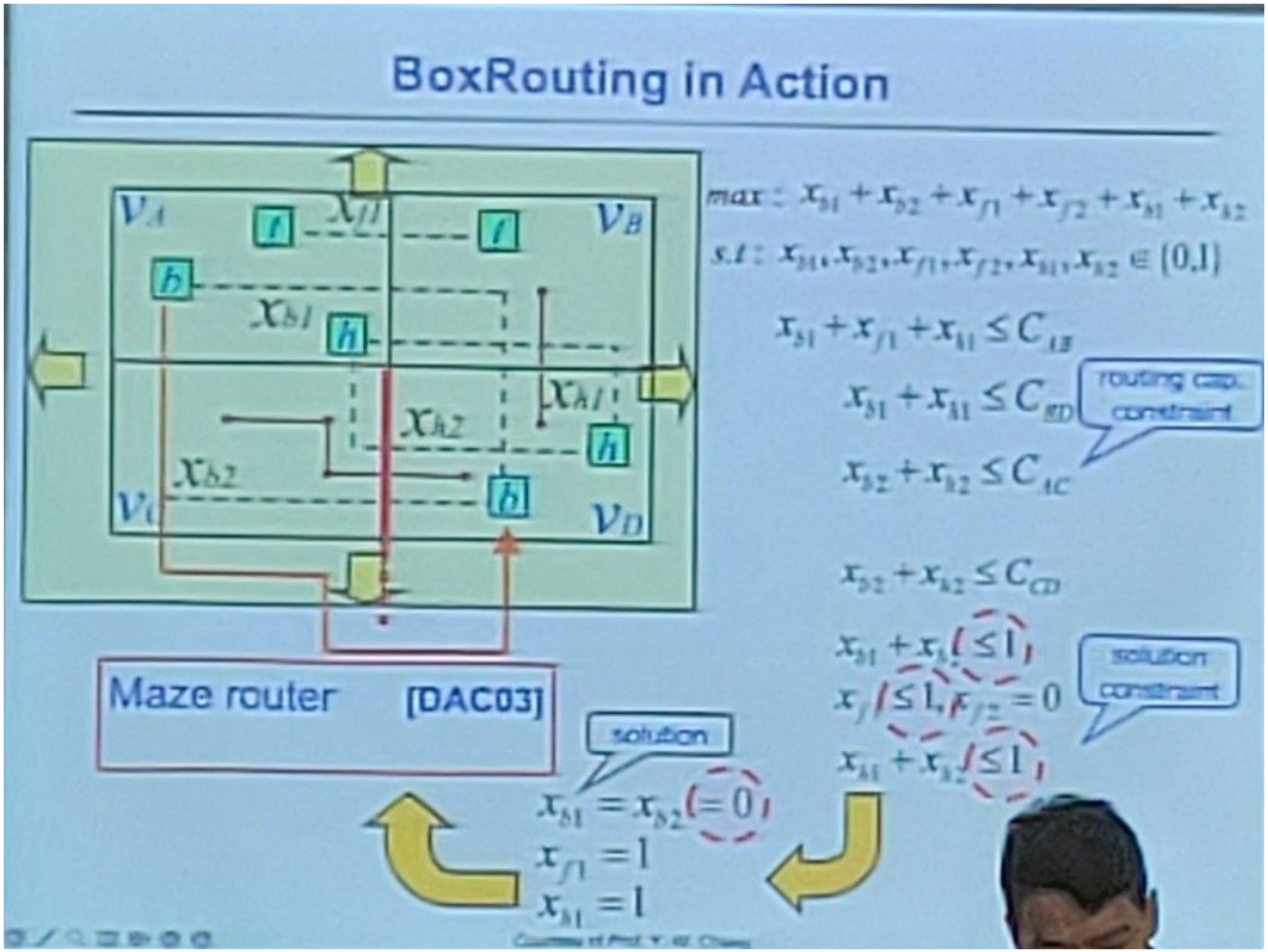


Courtesy of Prof. Y.-W. Chang

BoxRouting in Action



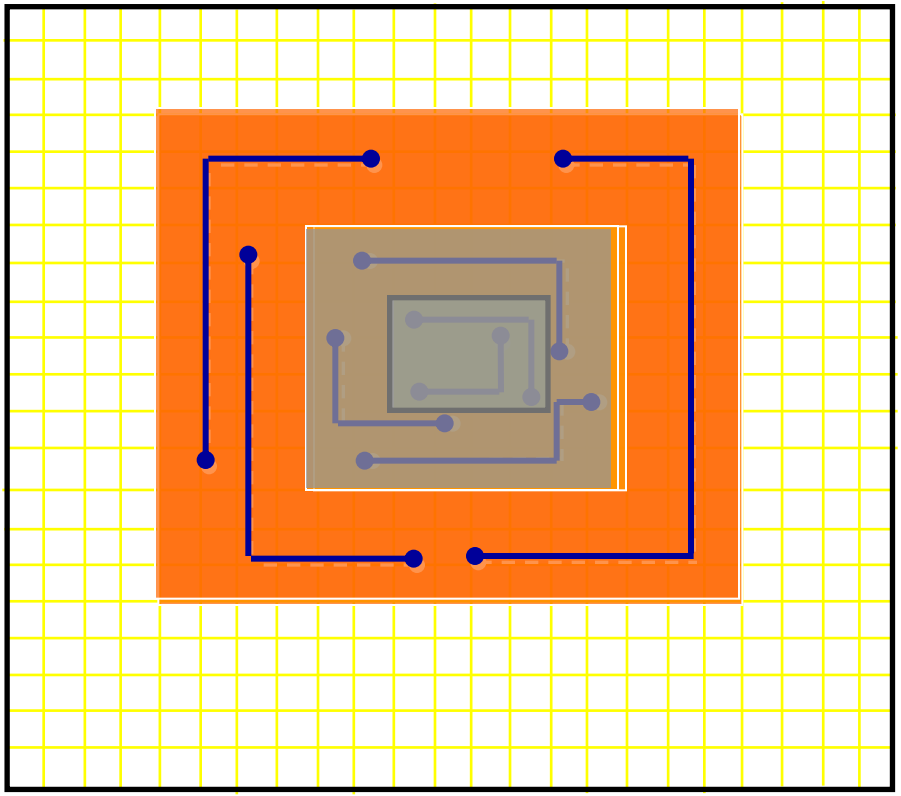
Courtesy of Prof. Y.-W. Chang



圖中的b就是無法被繞出來的，因此需要detour處理(繞遠路)

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



Courtesy of Prof. Y.-W. Chang

Proposed ILP vs. Conventional ILP

$$max: \sum \{x_{i1} + x_{i2}\}$$

$$s.t: x_{i1}, x_{i2} \in \{0,1\} \quad \forall i \in W_{box}$$

$$x_{i1} + x_{i2} \le 1 \quad \forall i \in W_{box}$$

$$x_{i2} = 0 \quad \forall i \in W_{box} \cap W_{flat}$$

$$\sum_{e \in x_{ij}} x_{ij} \le C_e \quad \forall i \in W_{box}$$

$$min: D \quad \forall i \in W_{box}$$

$$s.t: x_{i1}, x_{i2} \in \{0,1\} \quad \forall i \in W_{box}$$

$$x_{i1} + x_{i2} = 1 \quad \forall i \in W_{box}$$

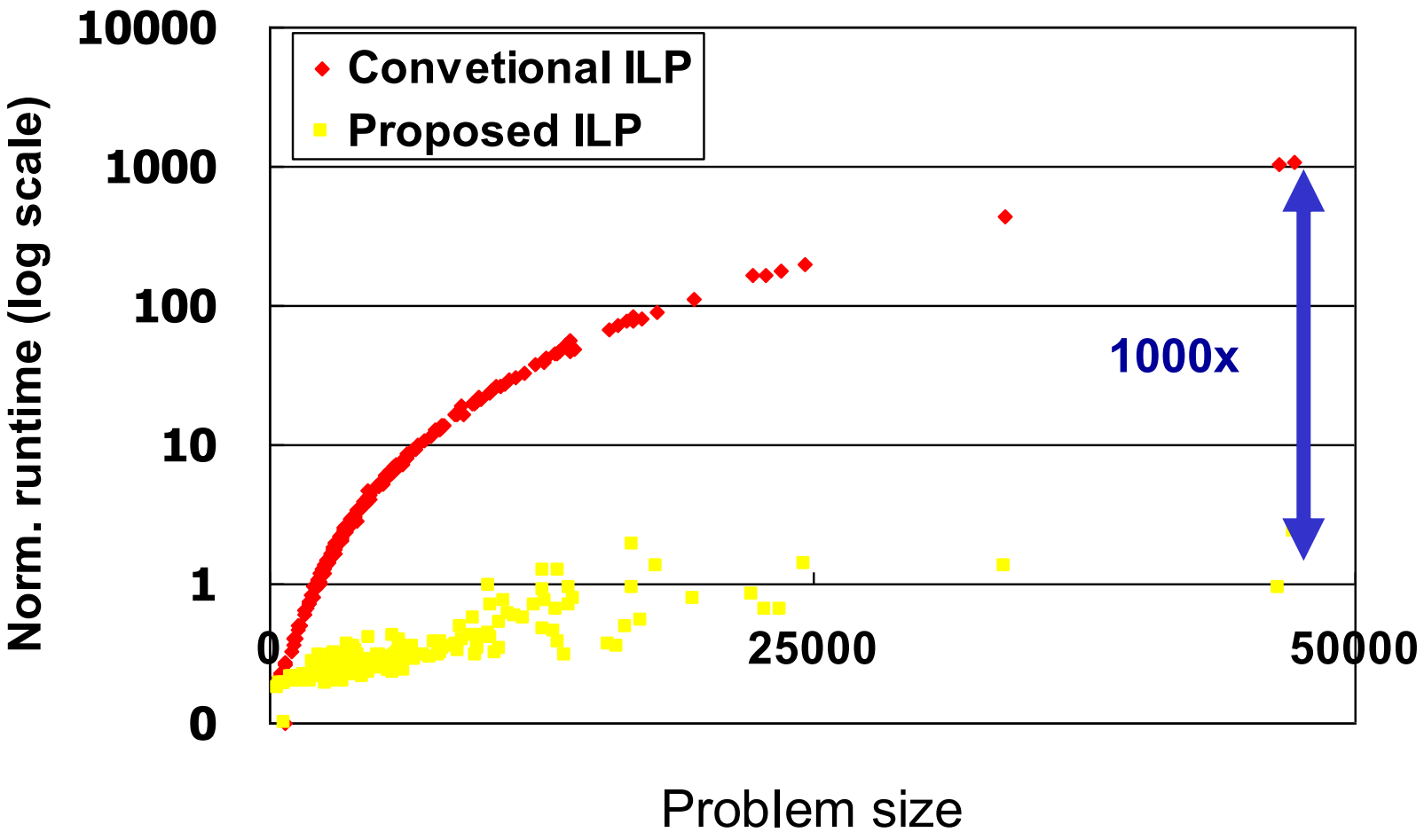
$$x_{i2} = 0 \quad \forall i \in W_{box} \cap W_{flat}$$

$$\sum_{e \in x_{ij}} x_{ij} \le D \quad \forall i \in W_{box}$$

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

Courtesy of Prof. Y.-W. Chang

Proposed ILP vs. Conventional ILP



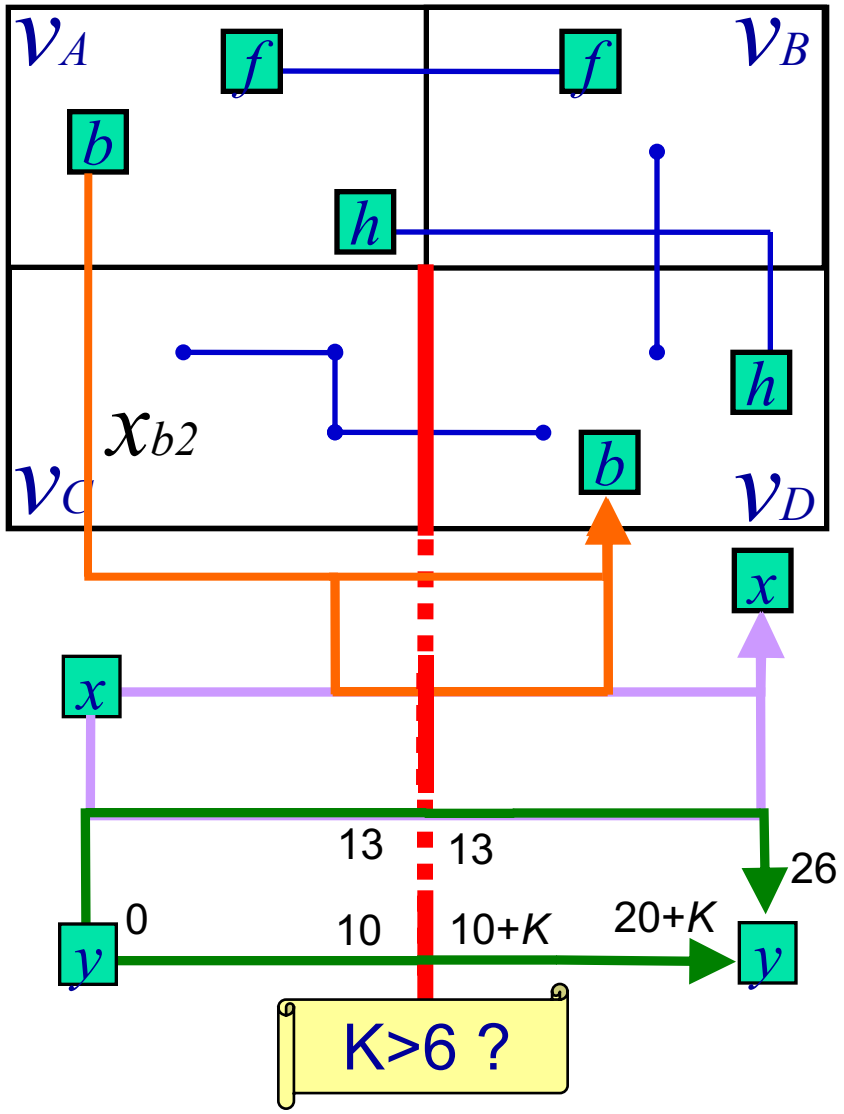
GNU Linear Programming Kit 4.10, Industrial circuit (7mm x 7mm)

Courtesy of Prof. Y.-W. Chang

跳過

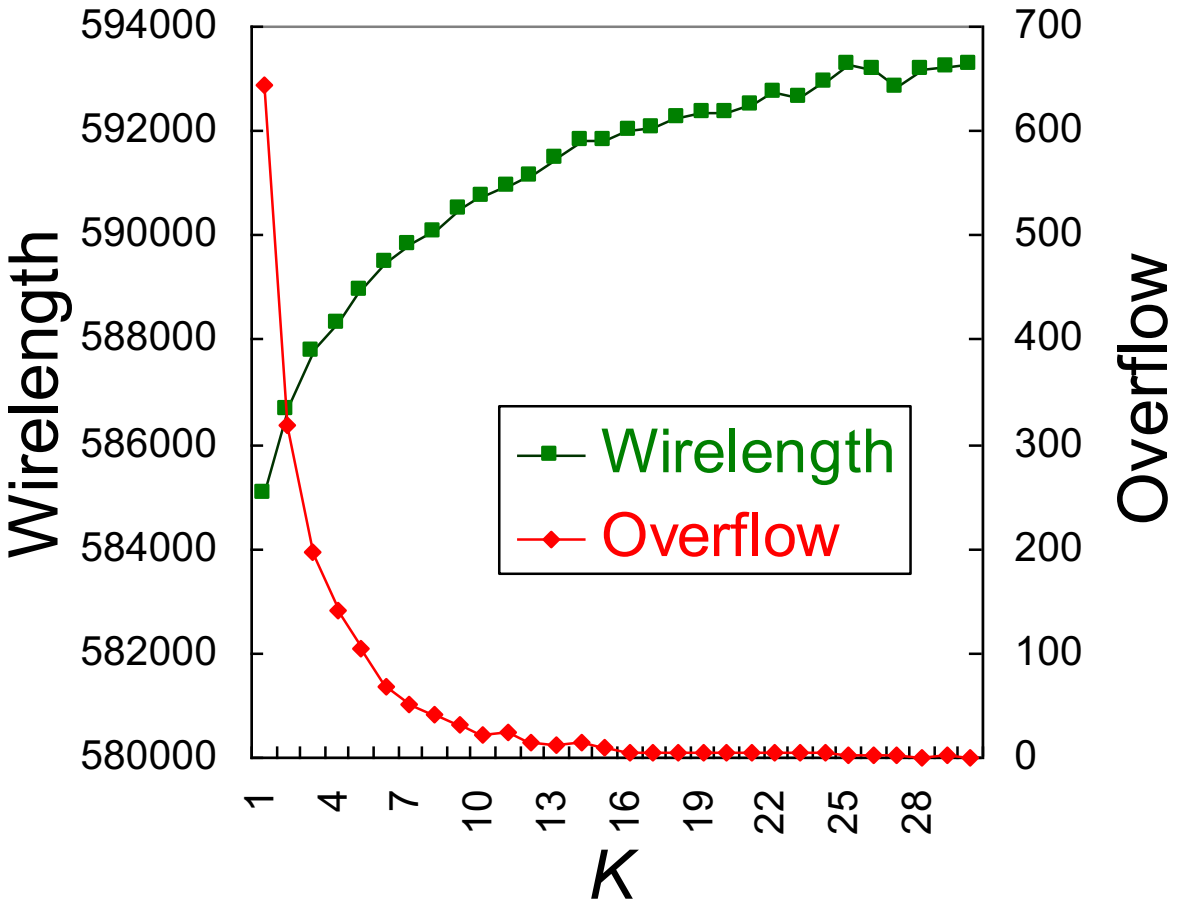
PostRouting

- 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*



Courtesy of Prof. Y.-W. Chang

PostRoute



- Smooth tradeoff: $K \uparrow$ => overflow↓, wirelength↑

Courtesy of Prof. Y.-W. Chang

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*:

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

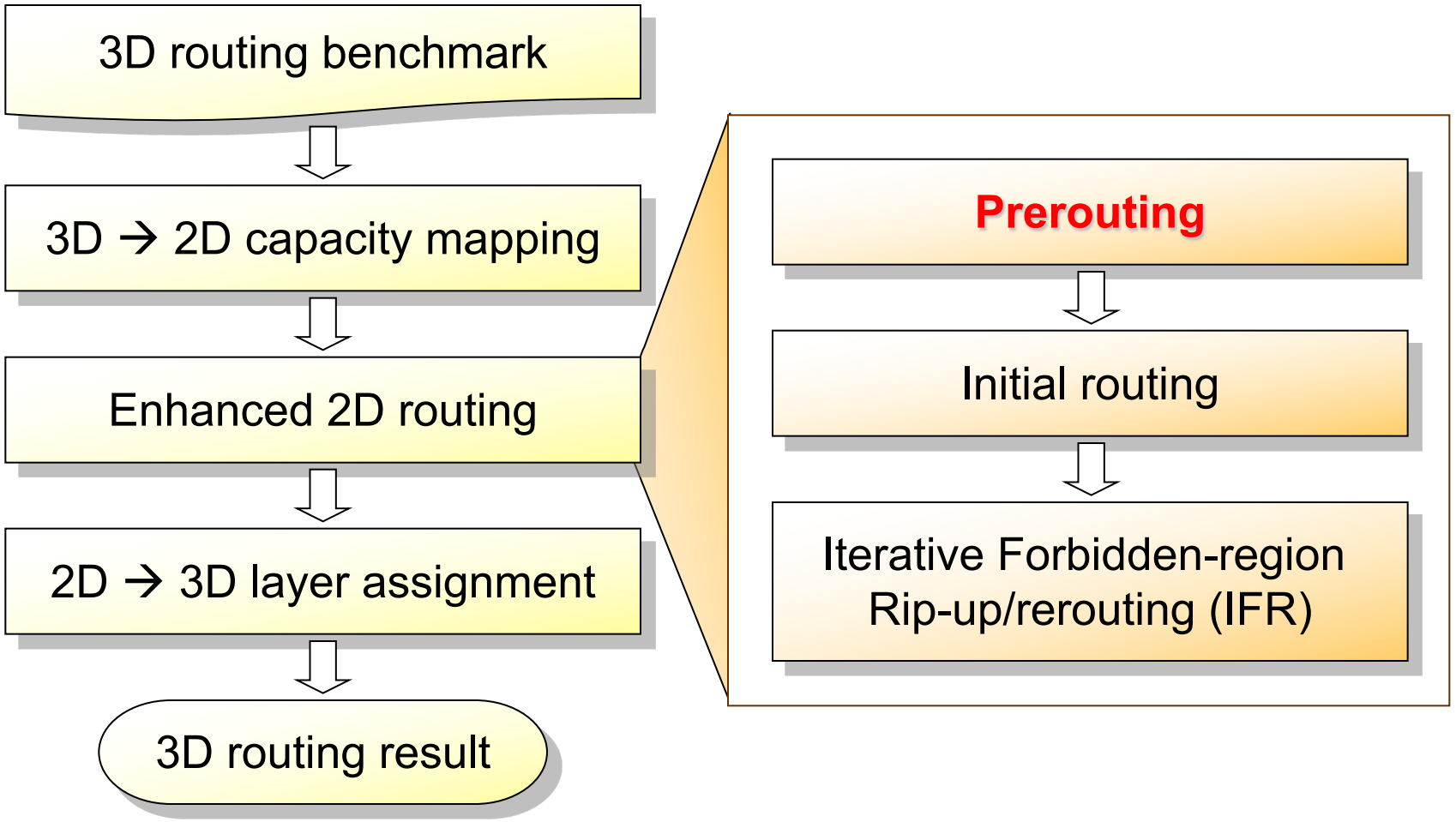
  - b_e: base cost of using *e*,
  - p_e: # of nets passing *e*,
  - $h_e^{(i)}$: historical cost on *e*, $h_e^{(i)} = \begin{cases} h_e^{(i-1)} + 1 & \text{if } e \text{ has overflow} \\ h_e^{(i-1)} & \text{otherwise} \end{cases}$

- 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]

Courtesy of Prof. Y.-W. Chang

NTUgr Global Routing Flow

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

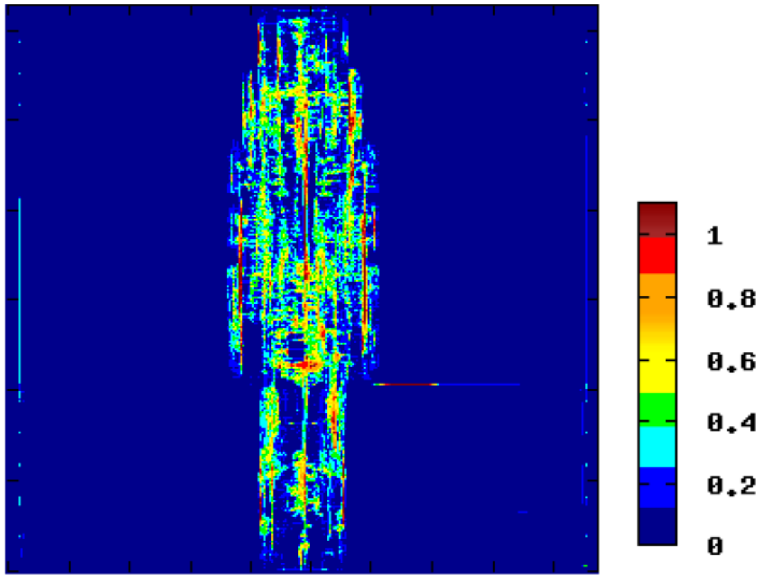


Courtesy of Prof. Y.-W. Chang

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)

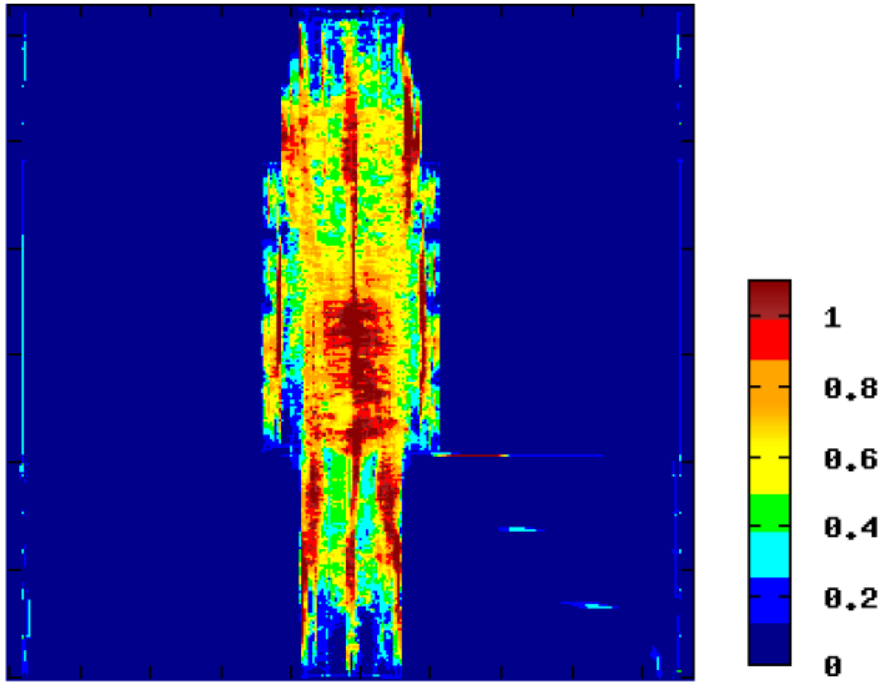


Courtesy of Prof. Y.-W. Chang

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)

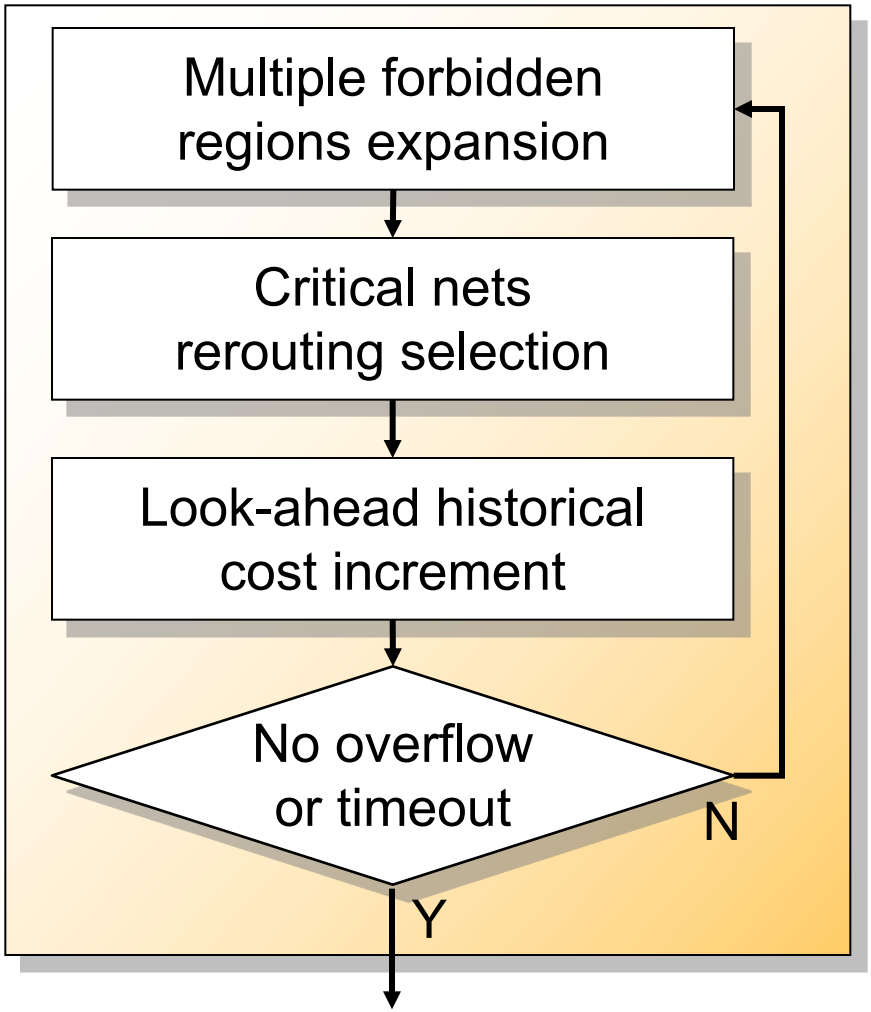


Courtesy of Prof. Y.-W. Chang

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

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

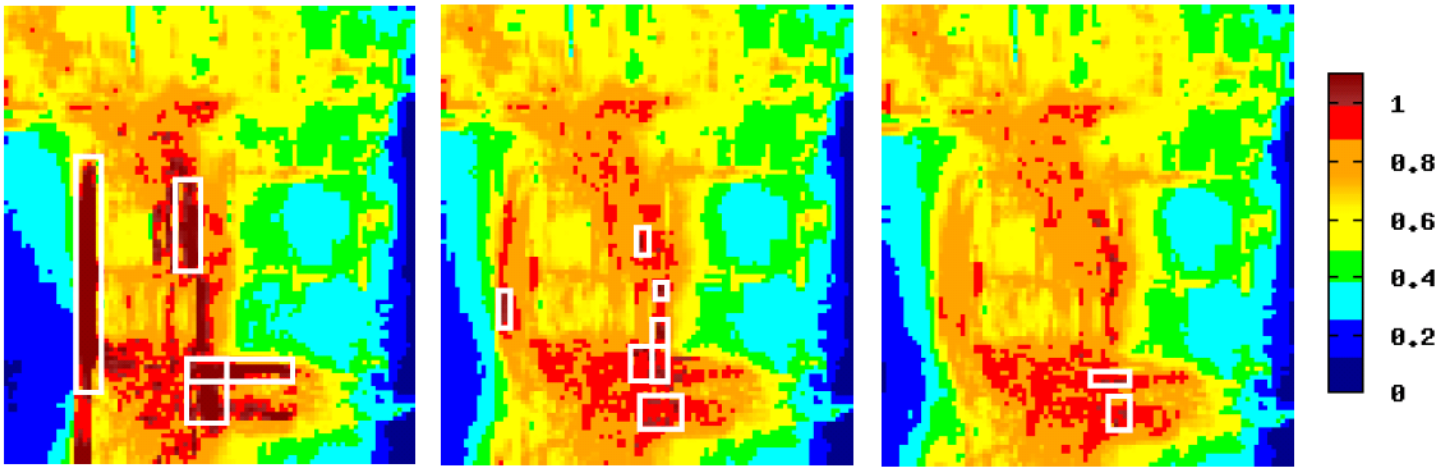
IFR:



Courtesy of Prof. Y.-W. Chang

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



Forbidden-region routing of adaptec5

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

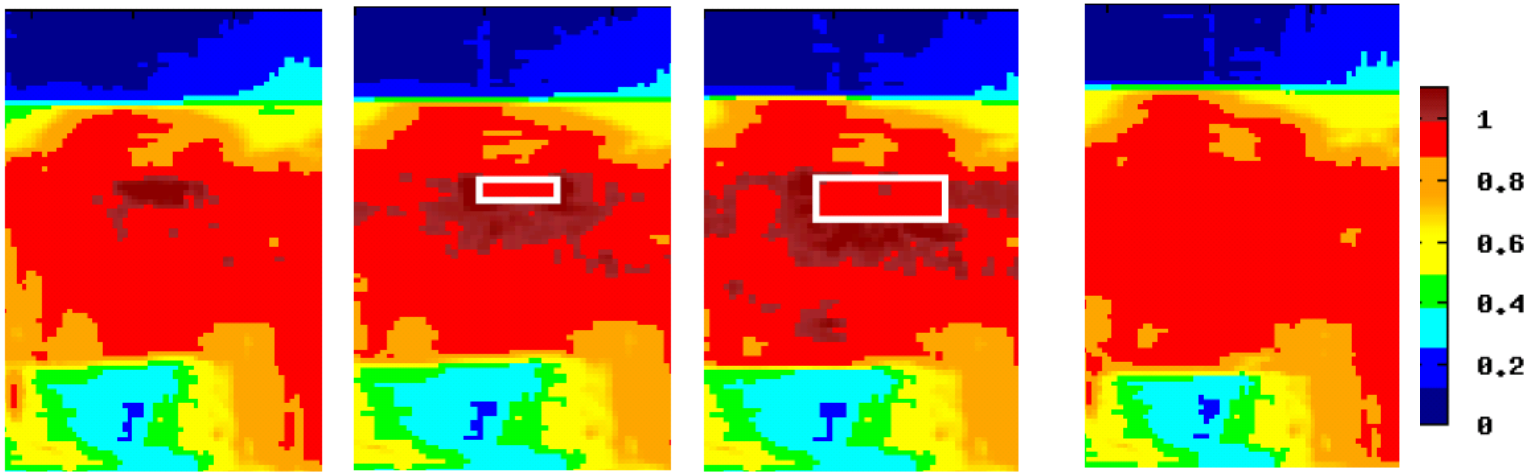
$h_e^{(i)}$: historical cost of e

$$b_e = \begin{cases} 1/(c_e - d_e) & \text{if } d_e < c_e \\ \xi & \text{if } d_e = c_e \\ \xi + d_e / c_e & \text{if } d_e > c_e \end{cases}$$

(ξ = penalized base cost of e; $\xi = 3$ in NTUgr)

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



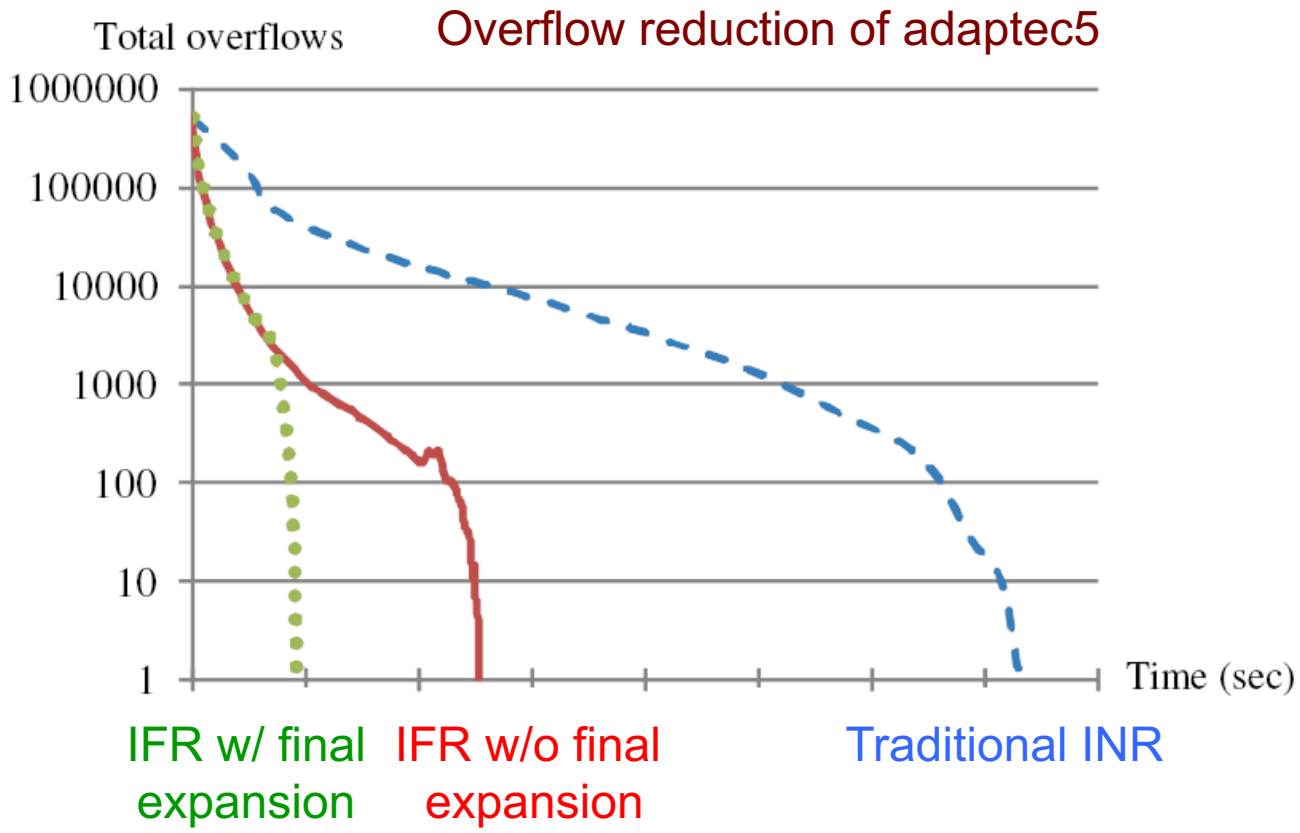
(i)-th iteration (i+1)-th iteration (i+2)-th iteration final iteration

Forbidden-region routing of bigblue3

Courtesy of Prof. Y.-W. Chang

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



Courtesy of Prof. Y.-W. Chang

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

Courtesy of Prof. Y.-W. Chang

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

Courtesy of Prof. Y.-W. Chang

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

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

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

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

Courtesy of Prof. Y.-W. Chang

跳過終止

Results on ISPD'08 Benchmarks

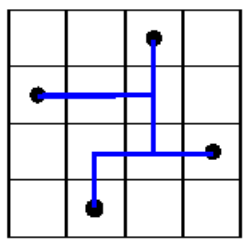
- Compared with the winners of ISPD'08 global routing contest
  - Runtime is averagely the same as that of NTHU-Route 2.0
  - Overflow is better than FastRoute 3.0

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

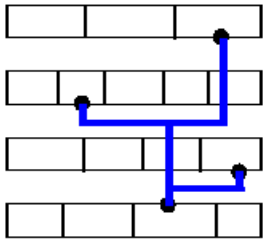
Courtesy of Prof. Y.-W. Chang

The Routing-Tree Problem

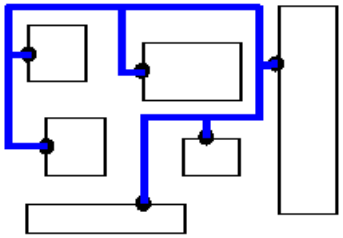
- **Problem:** Given a set of pins of a net, interconnect the pins by a "routing tree."



gate array



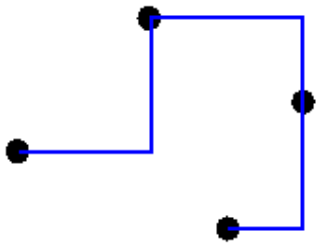
standard cell



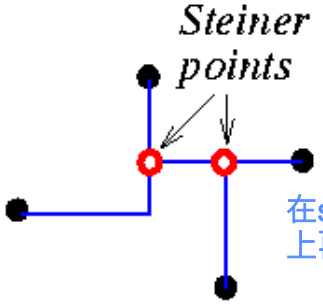
building block

- **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.

spanning tree可以在polynomial time找到最佳解



minimum spanning tree
MST



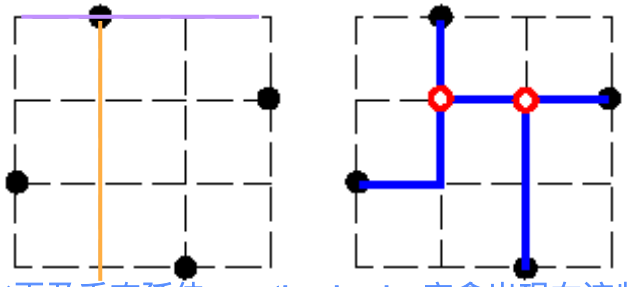
MRST

在spanning tree的基礎上再加入steiner point

Courtesy of Prof. Y.-W. Chang

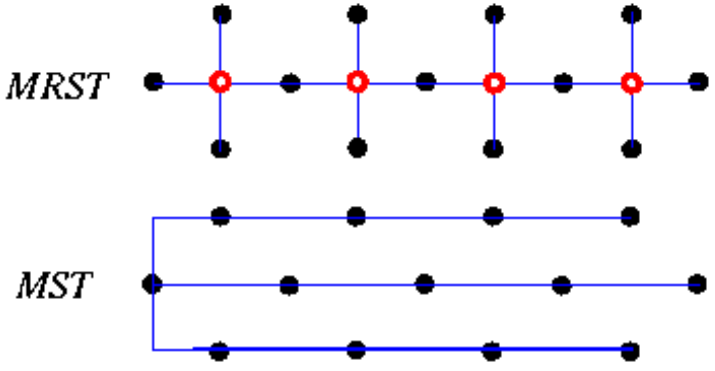
Theoretic Results for the MRST Problem

- **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*, $\frac{Cost(MST(P))}{Cost(MRST(P))} \leq \frac{3}{2}$. spanning tree的cost大概是steiner tree的1.5倍
  - 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.



把每個pin做水平及垂直延伸，optimal sol一定會出現在這些路徑上
steiner point也會出現在這些延伸線的交界處上

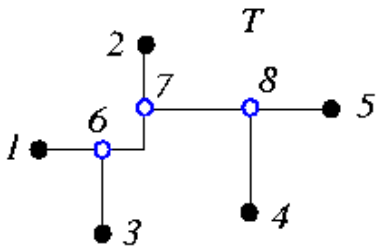
Hanan grid

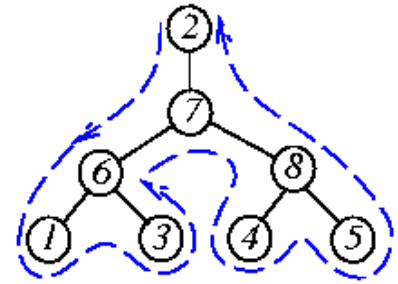


Cost(MST)/Cost(MRST) –> 3/2

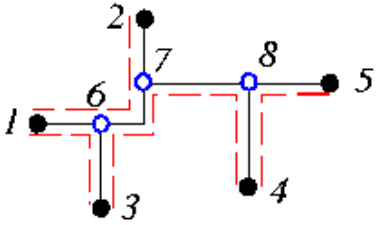
A Simple Performance Bound

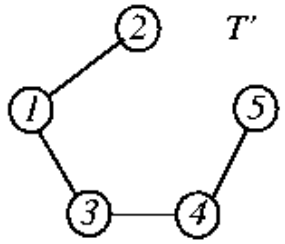
- Easy to show that $\frac{Cost(MST(P))}{Cost(MRST(P))} \leq 2$
- Given any MRST T on point set P with Steiner point set S, construct a spanning tree T' on P as follows:
  1. Select any point in T as a root.
  2. Perform a depth-first traversal on the rooted tree T.
  3. Construct T' based on the traversal.





- *depth–first traversal*
- *every edge is visited twice*

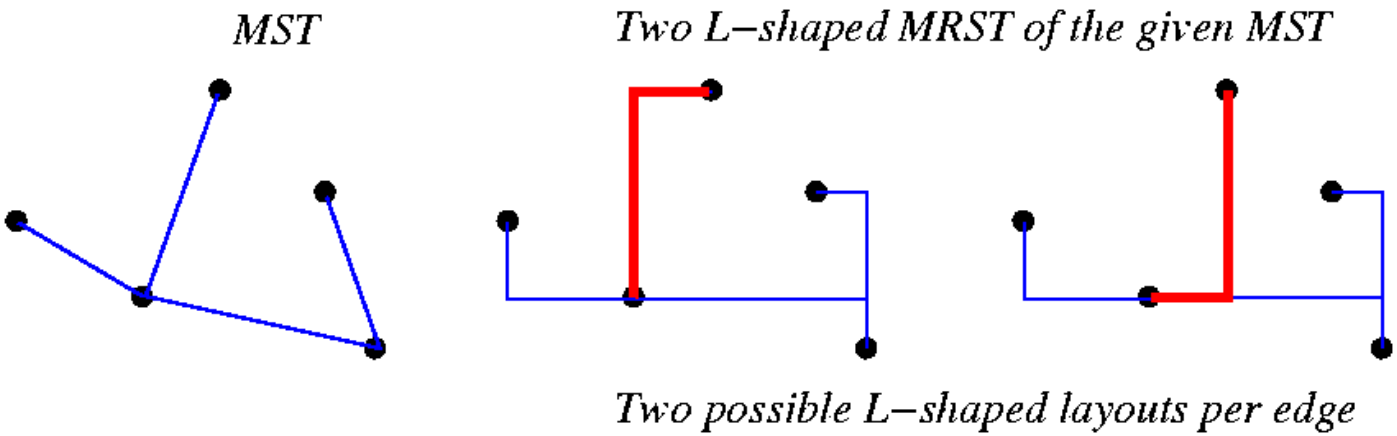




Cost(T') <= 2 Cost(T)

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. 把每個spanning tree的edge 轉換成直線或是L shape
  3. Maximize overlaps by dynamic programming. 應該有點類似slicing tree的處理方式 先bottom up 再 top down (用DP去尋最佳解)
- About 8% smaller than *Cost*(*MST*).



Courtesy of Prof. Y.-W. Chang

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 $S \leftarrow \varnothing$;

/* $H(P \cup S)$: set of Hanan points */

/* $\Delta MST(A, B) = Cost(MST(A)) - Cost(MST(A \cup B))$ */

3 **while** ($Cand \leftarrow \{x \in H(P \cup S) | \Delta MST(P \cup S, \{x\}) > 0\} \neq \varnothing$) **do**

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

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

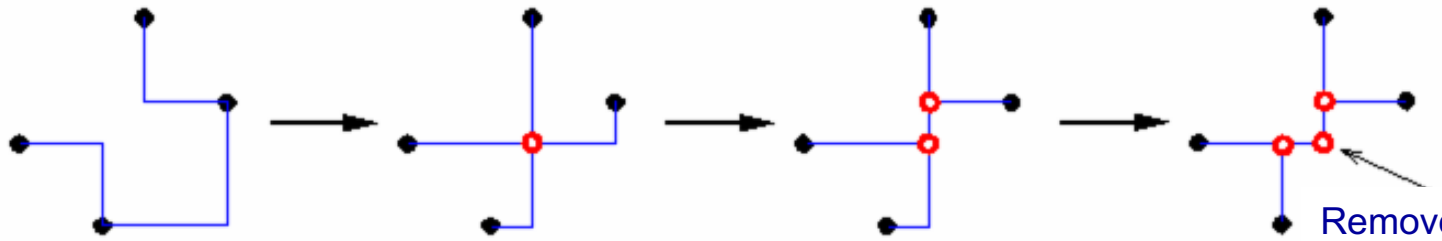
6 Remove points in *S* which have degree ≤ 2 in $MST(P \cup S)$;

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

8 **end**

計算steiner point的位置(每次loop都會計算一個steiner point)如果cost降低就將此steiner point加入一直持續到無法再降低cost為止

得到的解不一定是最佳解，因為steiner point的擺放順序也會影響，因此這也只是一種heuristic

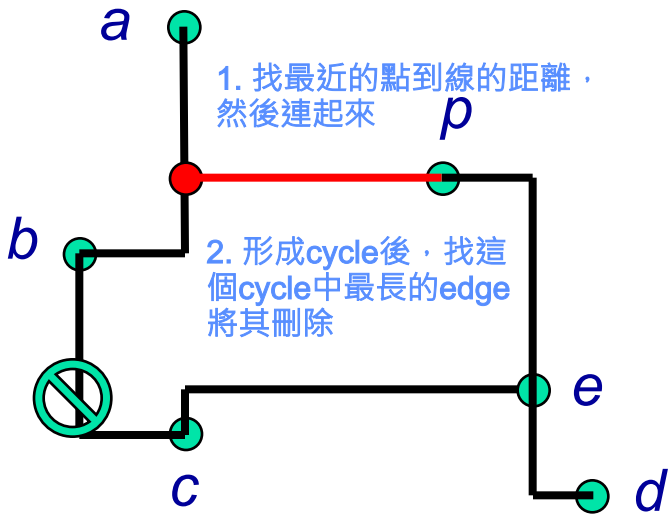


Courtesy of Prof. Y.-W. Chang

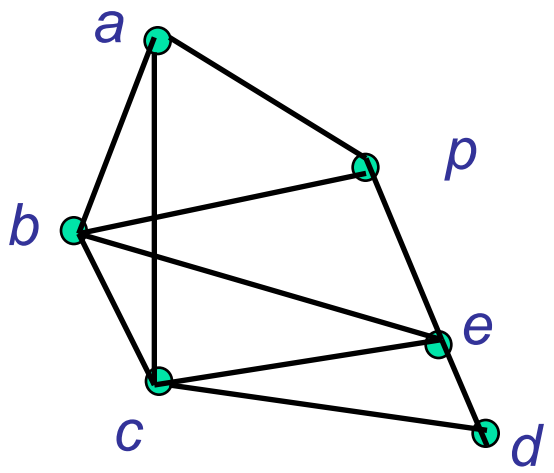
最牛逼的找steiner point方法

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



Spanning graph

Courtesy of Prof. Y.-W. Chang

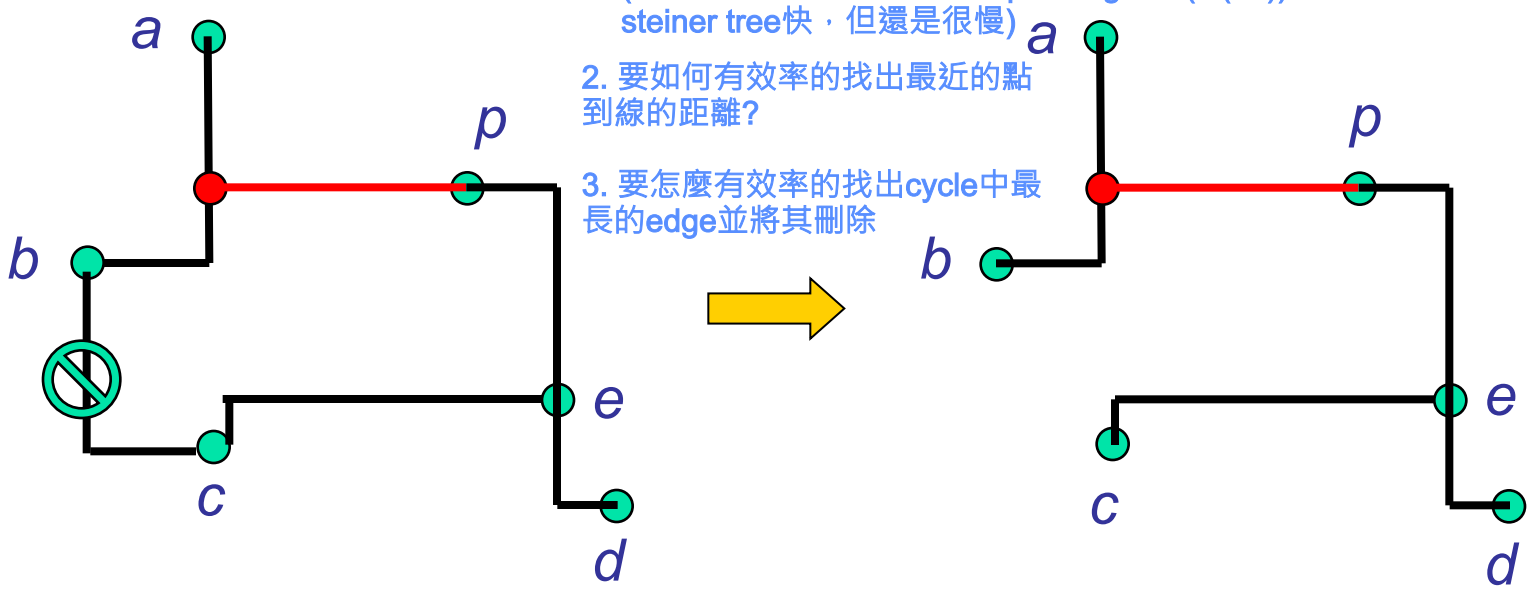
Edge-Substitution Approach

- 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*))

1. 要怎麼有效率的建立initial spanning tree? (當node很多的時候，即使spanning tree(O(n^2))比steiner tree快，但還是很慢)

2. 要如何有效率的找出最近的點到線的距離?

3. 要怎麼有效率的找出cycle中最長的edge並將其刪除



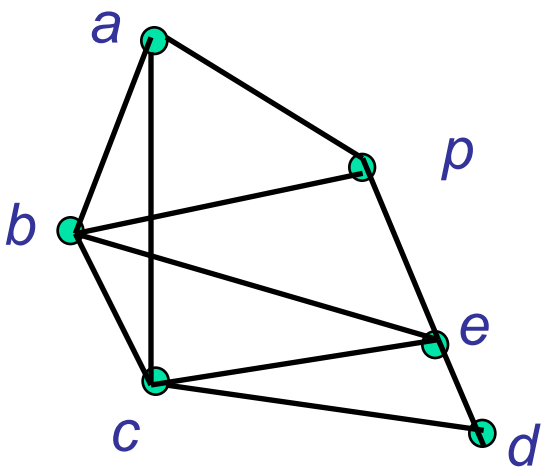
Courtesy of Prof. Y.-W. Chang

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!!

Kruscal演算法是從complete graph開始操作，edge的數量會很多
但我們可以對complete進行優化，spanning graph可以將原本的complete graph進行縮減，卻又可以保證最佳解仍在其中，可以大幅減少數量級
因此可以先做spanning graph後再進行Kruscal

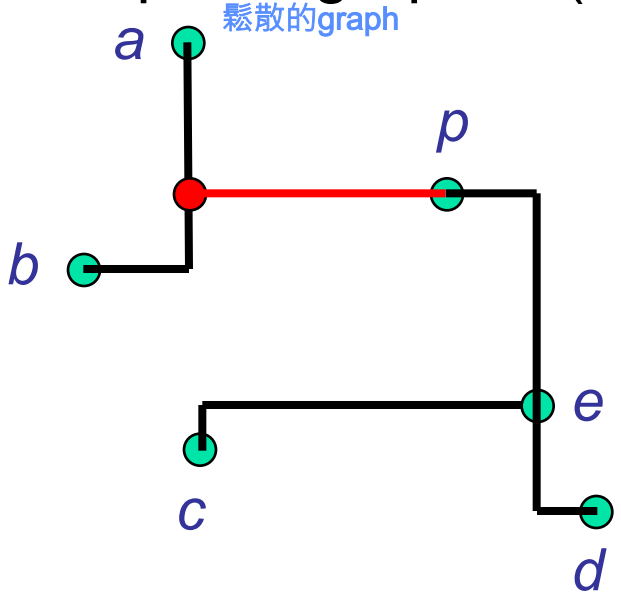


Spanning graph

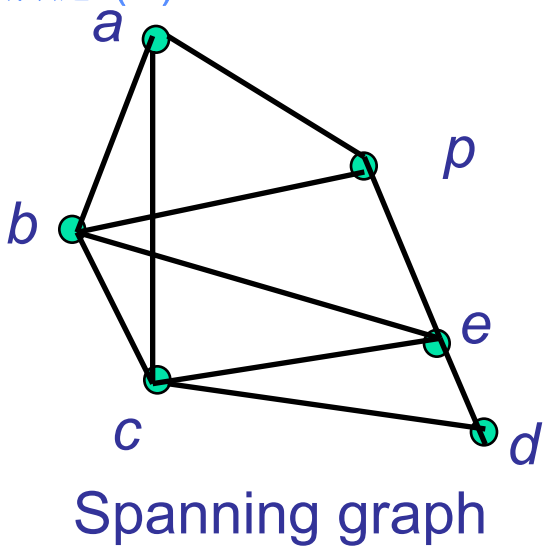
Kruscal演算法:
1. 先長出C_2^n個edge，並將其按照長度排列(ascending)
2. 先選最短的，然後第二短...，遇到會形成cycle的edge要跳過
3. 將選到的edge納入集合中，最後的集合就是spanning tree

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



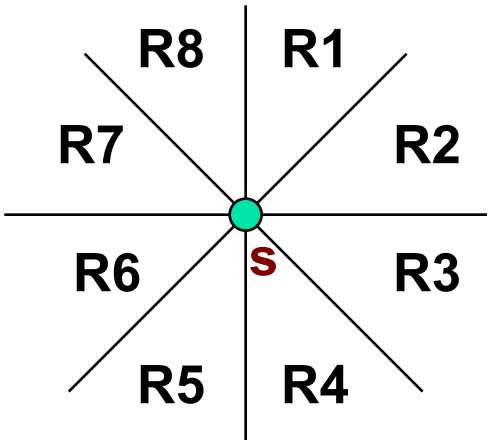
n是node的數量，complete graph因為是C_2^n，所以是$O(n^2)$

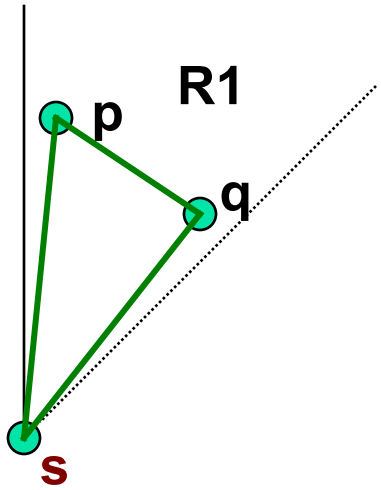


Spanning graph

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*

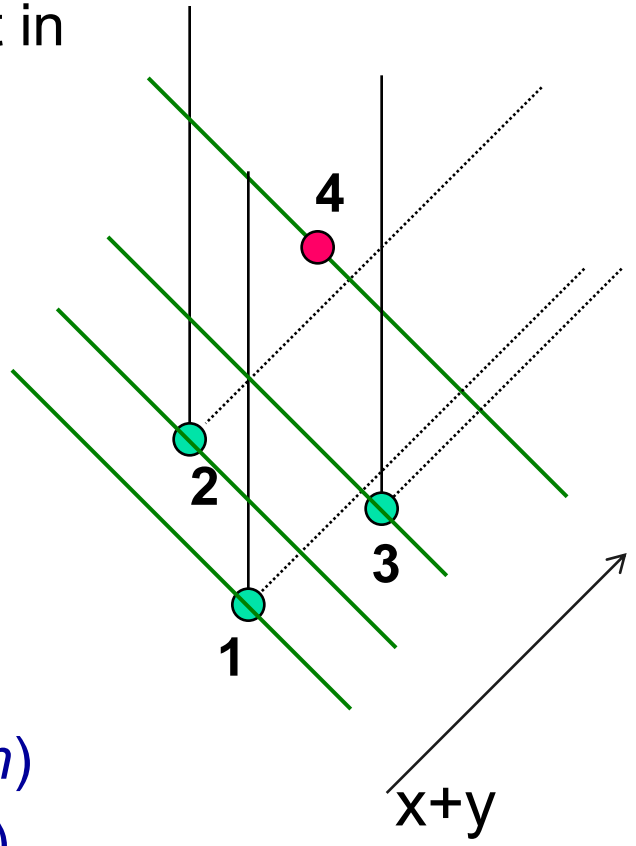




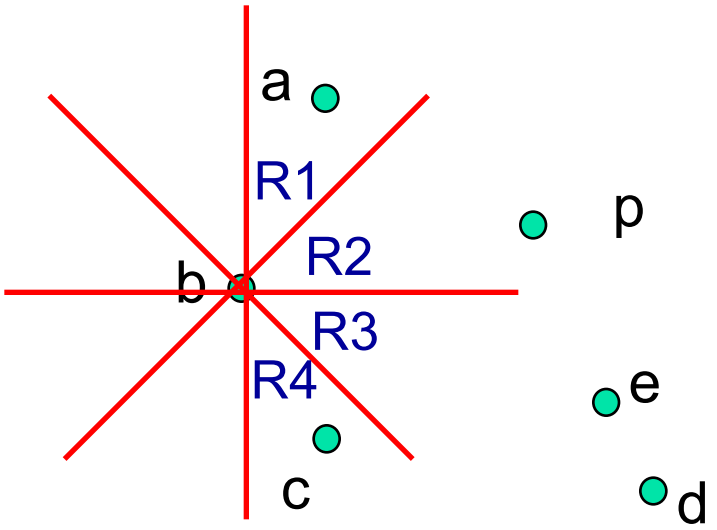
Courtesy of Prof. Y.-W. Chang

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 order 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



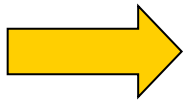
對應第一個問題的解法:
spanning graph

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

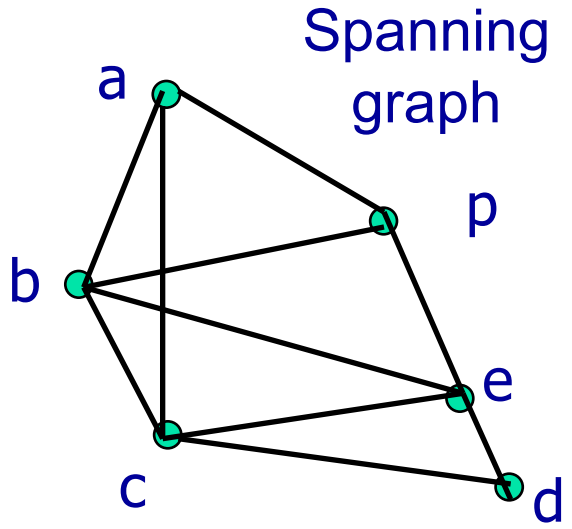
將當前考慮的node為原點(0,0)，去切割象限
然後找出每個象限中離原點最近的node，最後再把他們連起來就可以得到spanning graph

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

每個node都要做一次



Spanning graph



對應第二個問題的解法

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

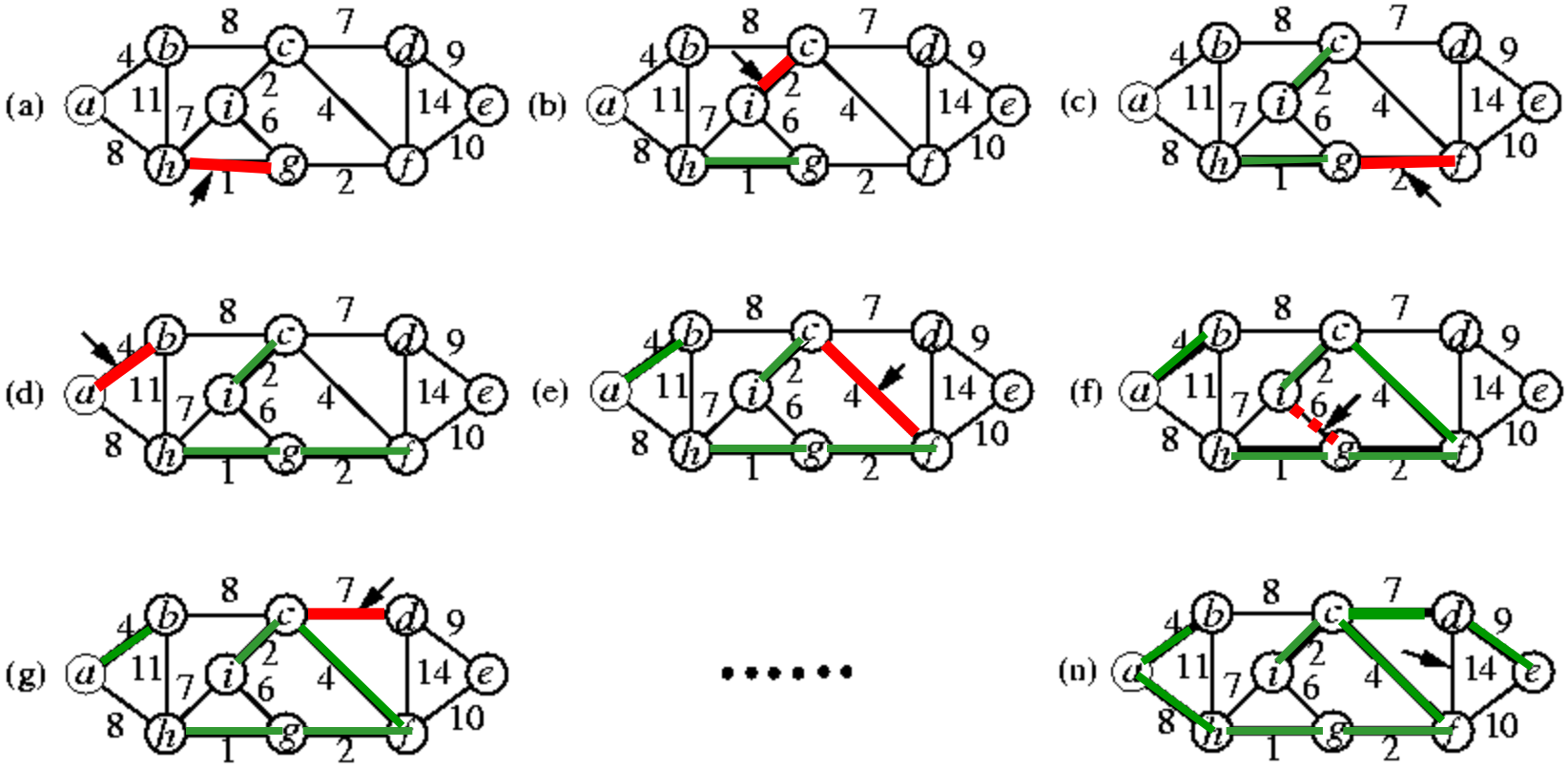
Courtesy of Prof. Y.-W. Chang

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;
```

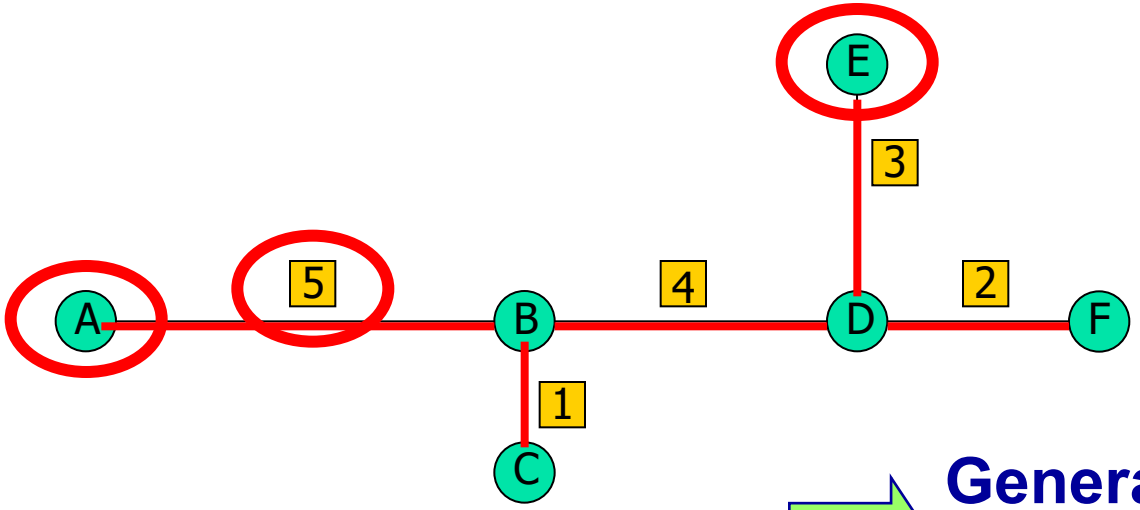
- Add a safe edge at a time to the growing **forest** by finding an edge of least weight (from those connecting two trees in the forest).
- Time complexity: $O(E \lg E + V)$ (= $O(E \lg V + V)$, why?)
  - Lines 2--3: $O(V)$.
  - Line 4:$O(E \lg E)$.
  - Lines 5--8: $O(E)$ operations on the disjoint-set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

Example: Kruskal's MST Algorithm

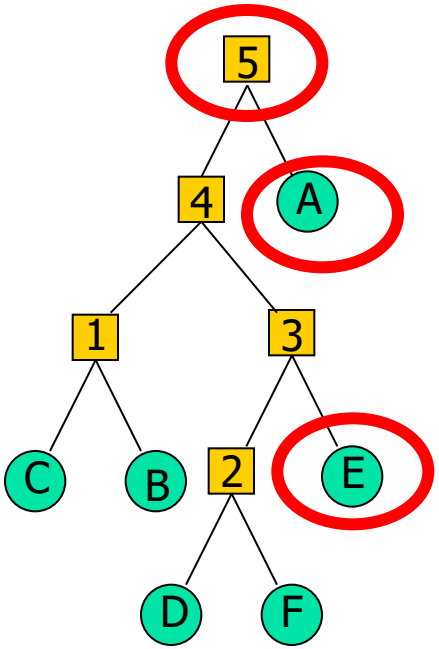


Courtesy of Prof. Y.-W. Chang

Finding the Longest Edge on a Cycle



1. 先選最短的edge(B、C)，將其弄成1號tree
2. 選到D、F，結合成2號tree
3. 選到D、E，但因為D已經被弄進2號tree了，所以E會跟2號tree結合成3號tree
.....



Generate the merging binary tree corresponding to the minimal spanning tree

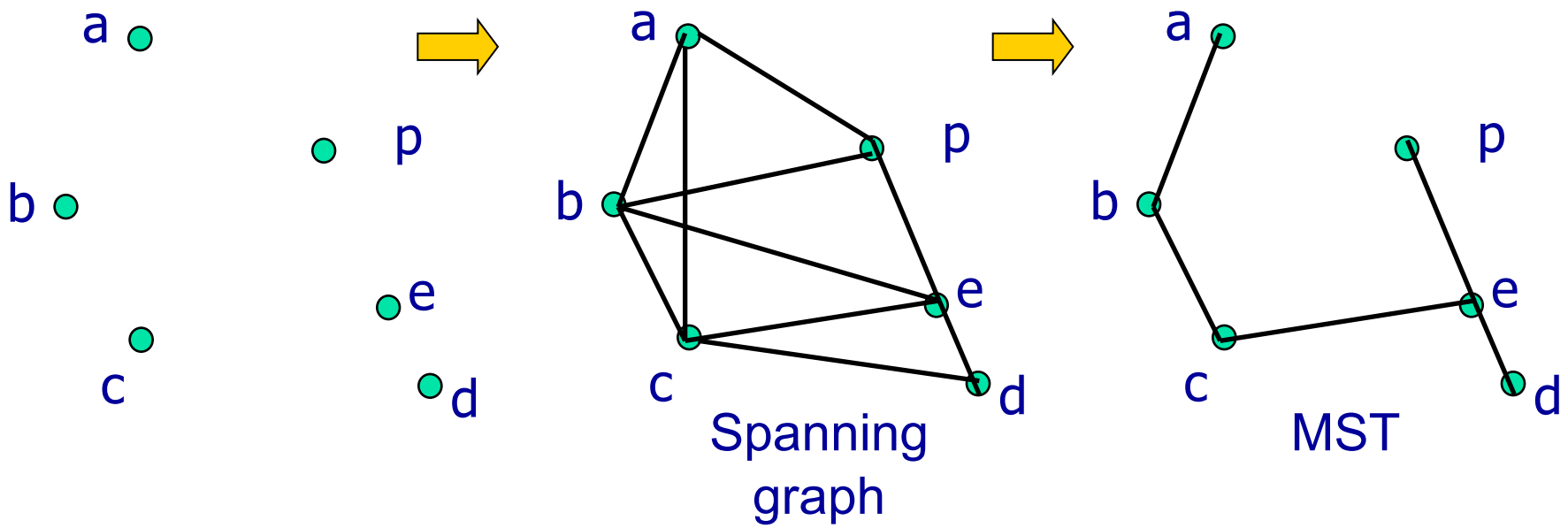
The longest edge for this pair is the least common ancestor

兩個node之間最長的路徑就是他們"最近的共同祖先"
以A、E為例，他們間的最長edge就是5號edge

Courtesy of Prof. Y.-W. Chang

Example Spanning Graph Construction

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

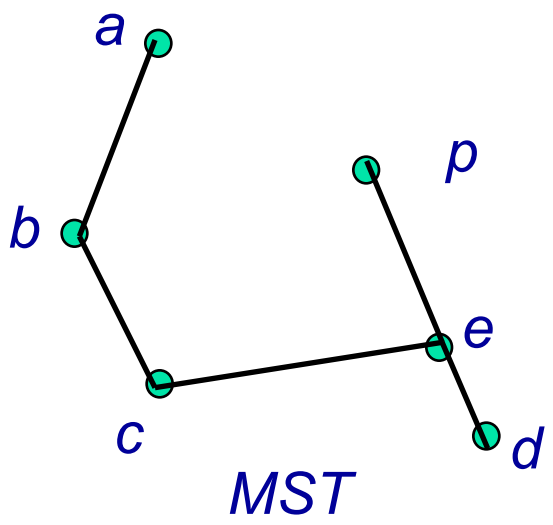


Courtesy of Prof. Y.-W. Chang

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

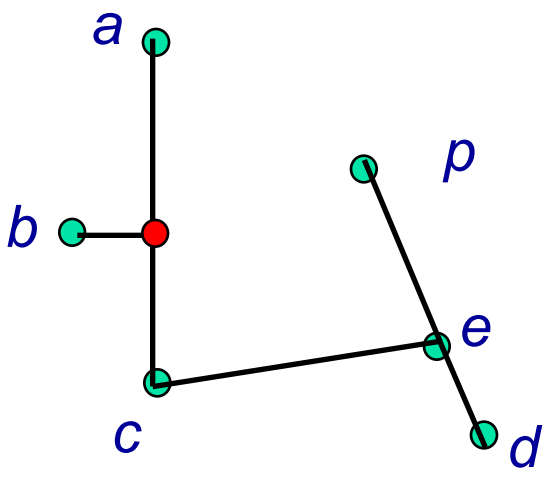


Building the table according to the spanning graph

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

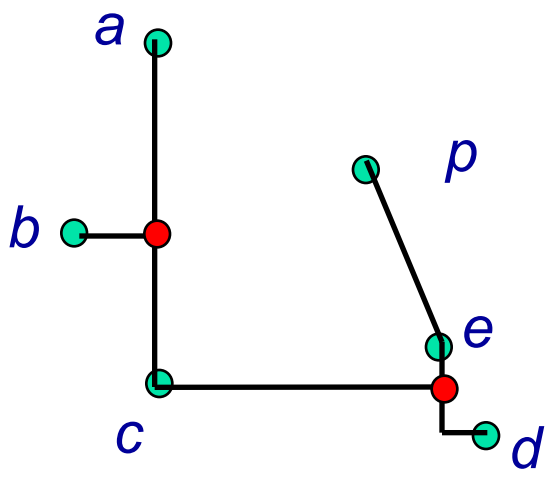


Point-Edge Connection Updating (cont'd)

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

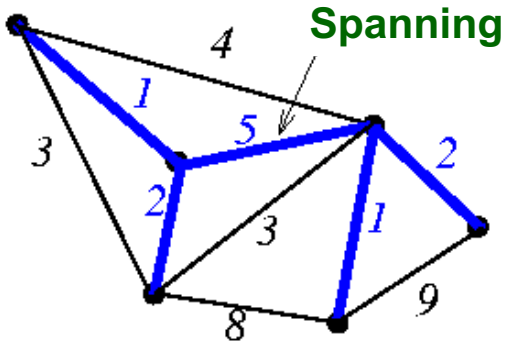
- Find the remaining point-edge pair with the max gain (point *c* to edge *(d, e)*)

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?



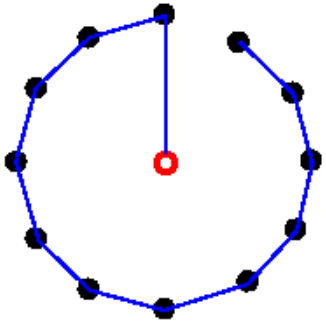
$Cost(T)=1+2+5+1+2=11$

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

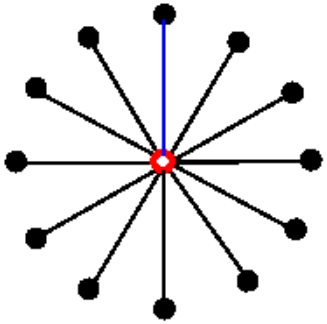
MST vs. SPT Trade-off

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

Minimum spanning tree (MST)



Shortest path tree (SPT)

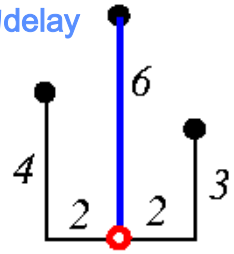


$$\frac{Cost(SPT)}{Cost(MST)} = \theta(n)$$

minimum spanning tree雖然繞線長度是最短的，但是實際在電路中的delay卻不一定是最少的
以左圖為例，他的繞線距離是最短的，但是他從第一個node到最後一個node的delay會很長
右圖中雖然繞線距離比較長，但是他的delay performance卻比較好

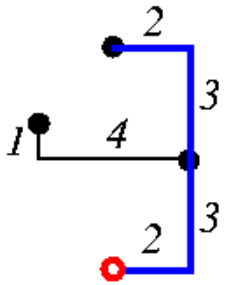
$$r(T) <= (1 + \epsilon)R$$

因此我們可以設置一個lower bound來避免左圖的情況發生



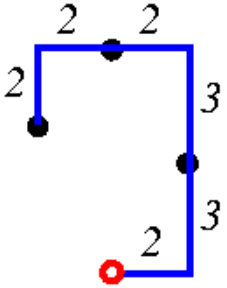
$\epsilon = 0$

Cost = 17
Radius = 6



$\epsilon = 1$

Cost = 15
Radius = 10



$\epsilon = infinite$

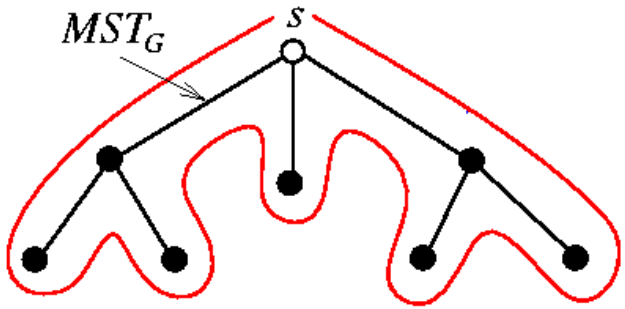
Cost = 14
Radius = 14

可以透過調整epsilon來決定spanning tree要怎麼長

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
```



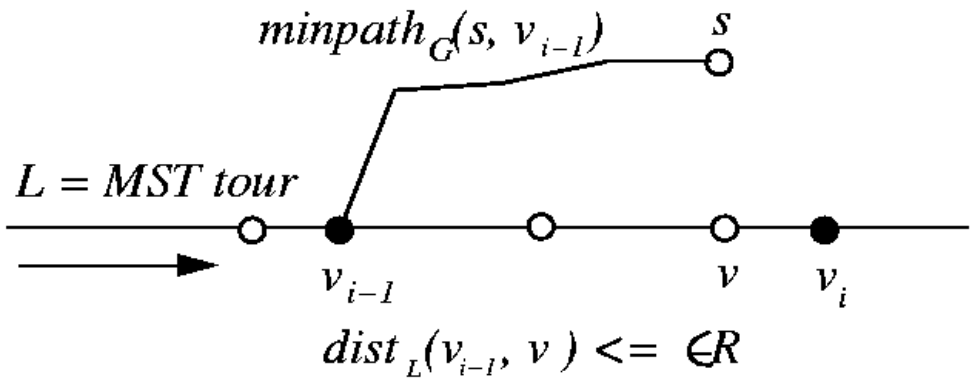
L = *depth–first tour of MST*

$cost(L) <= 2\ cost(MST_G)$

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.

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

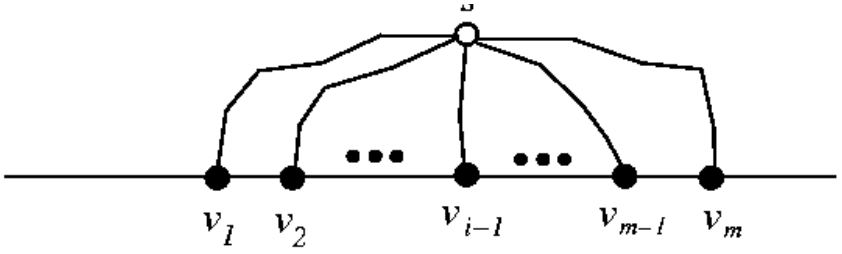


Courtesy of Prof. Y.-W. Chang

Bounded-Radius Bounded-Cost Spanning Tree

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

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



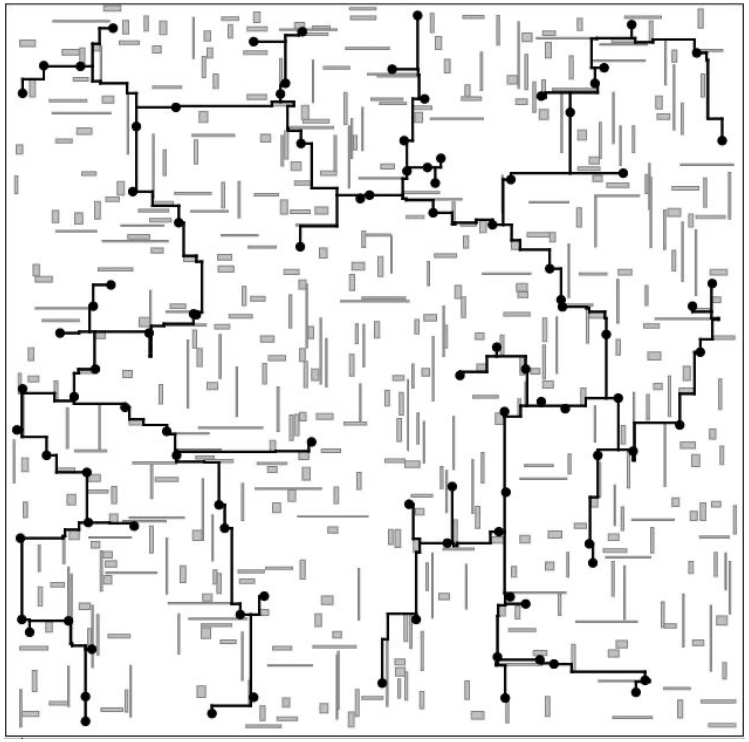
Courtesy of Prof. Y.-W. Chang

Appendix A:

Obstacle-Avoiding Rectilinear Steiner Tree Construction

Lin, Chen, Li, Chang, and Yang

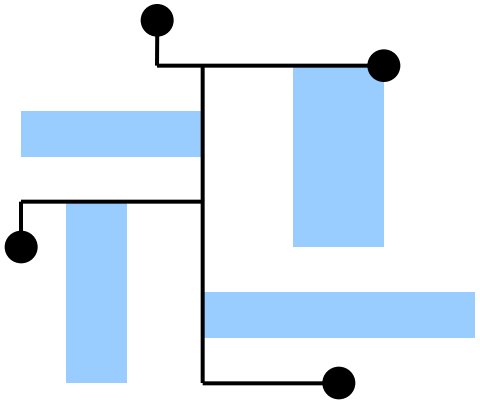
ISPD-07

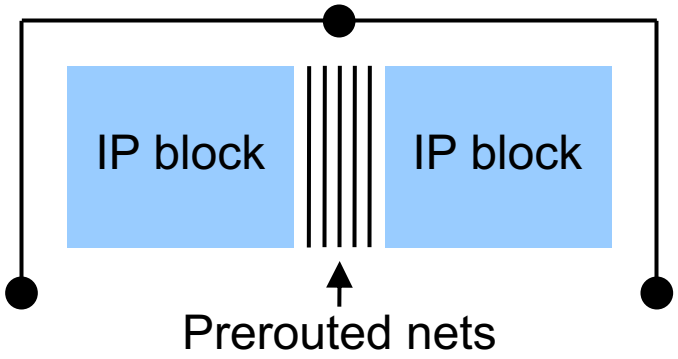


Courtesy of Prof. Y.-W. Chang

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

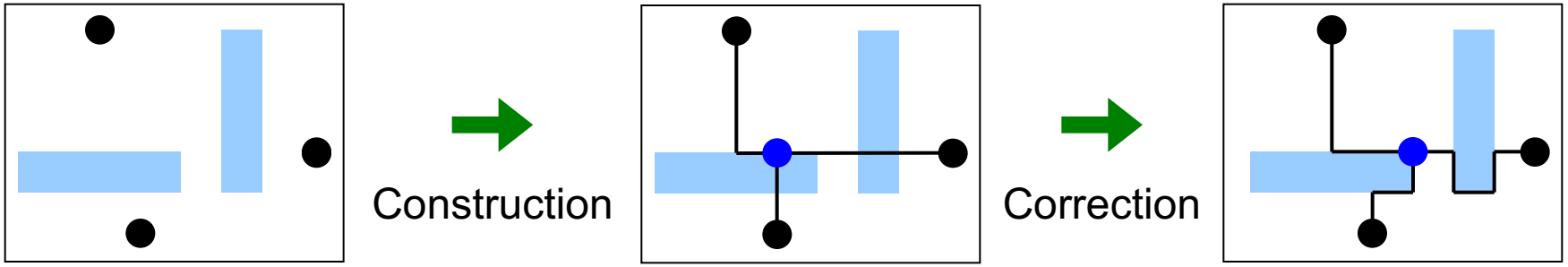




Courtesy of Prof. Y.-W. Chang

Construction-by-Correction Approach

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

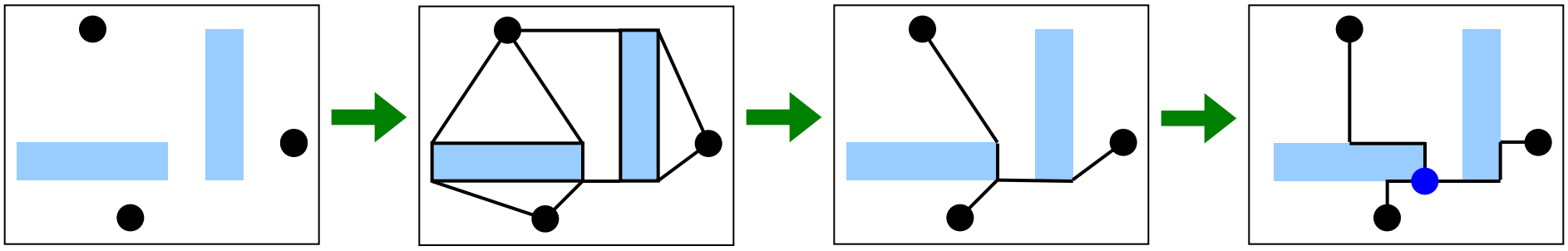


- Lack a global view of obstacles
- Have a limited solution quality
- Feng et al., ISPD-06
  - Construct OARSMT in the λ-geometry plane

Courtesy of Prof. Y.-W. Chang

Connection-Graph Based Approach

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



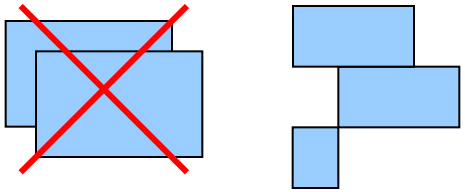
- Prune many redundant edges to reduce problem size
- Obtain much better solution quality
- Shen et al., ICCD-05
  - Achieve good results but can be further improved

Courtesy of Prof. Y.-W. Chang

Problem Formulation

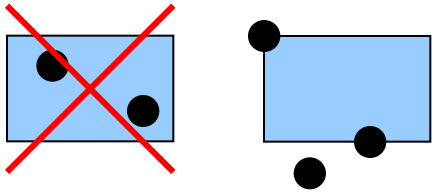
- Given m pins $\{p_1, p_2, \ldots, p_m\}$ and k obstacles $\{o_1, o_2, \ldots, o_k\}$, construct an OARSMT such that the total wirelength of the tree is minimized.

Obstacles overlap each other



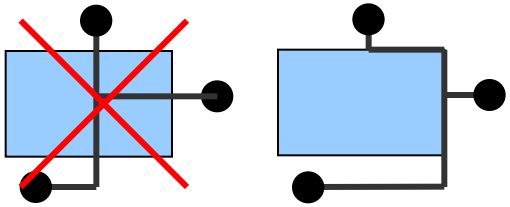
Obstacles are point-touched or line touched

A pin locates inside an obstacle



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

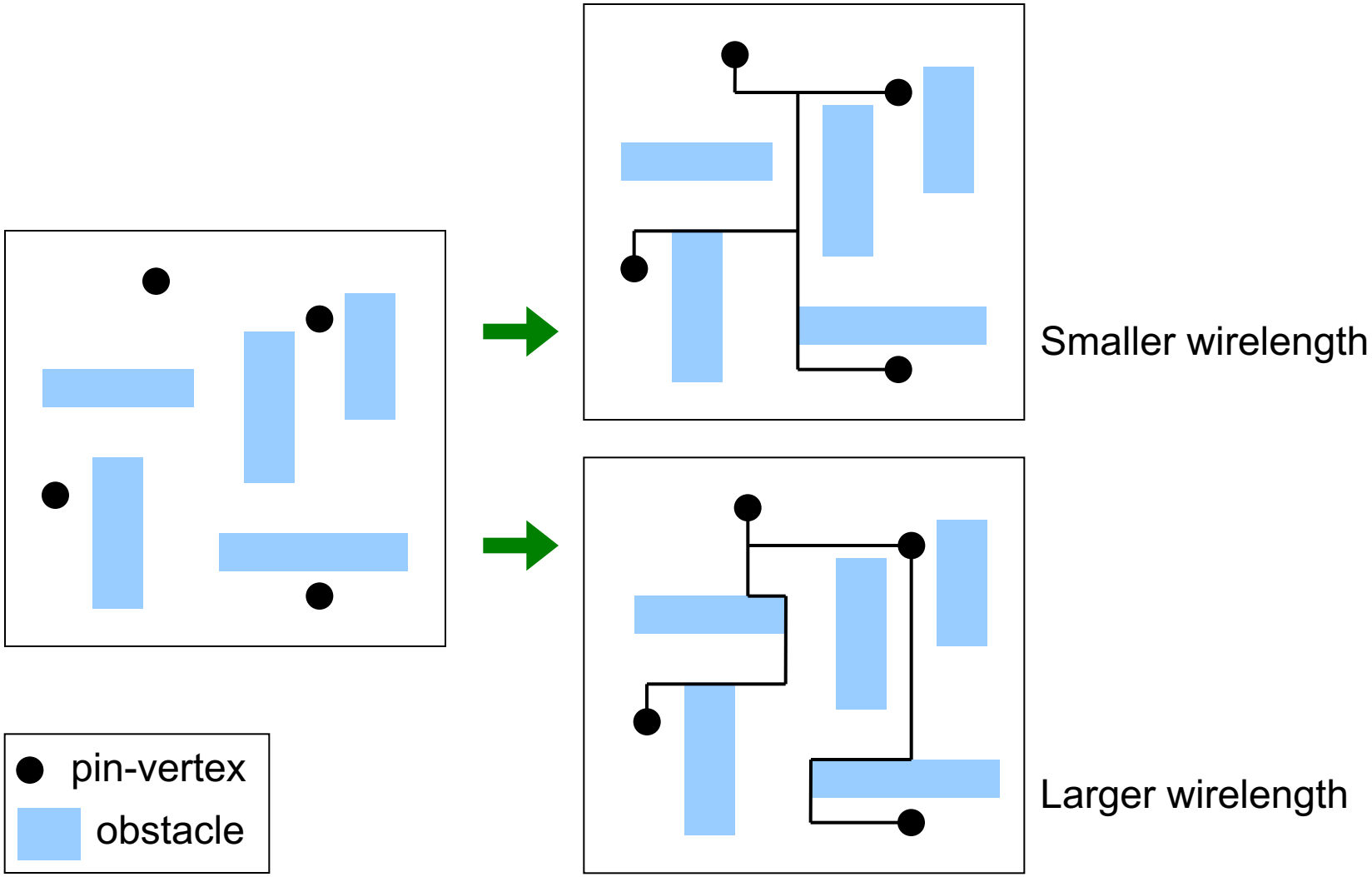
An edge intersects an obstacle



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

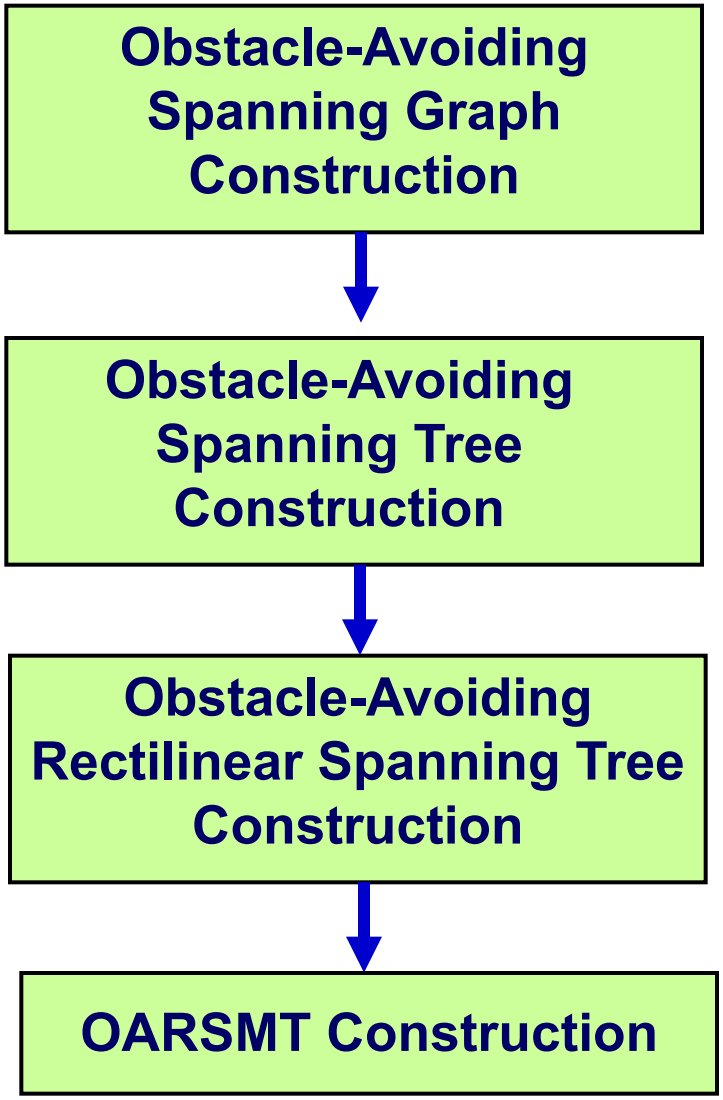
Courtesy of Prof. Y.-W. Chang

An OARSMT Example



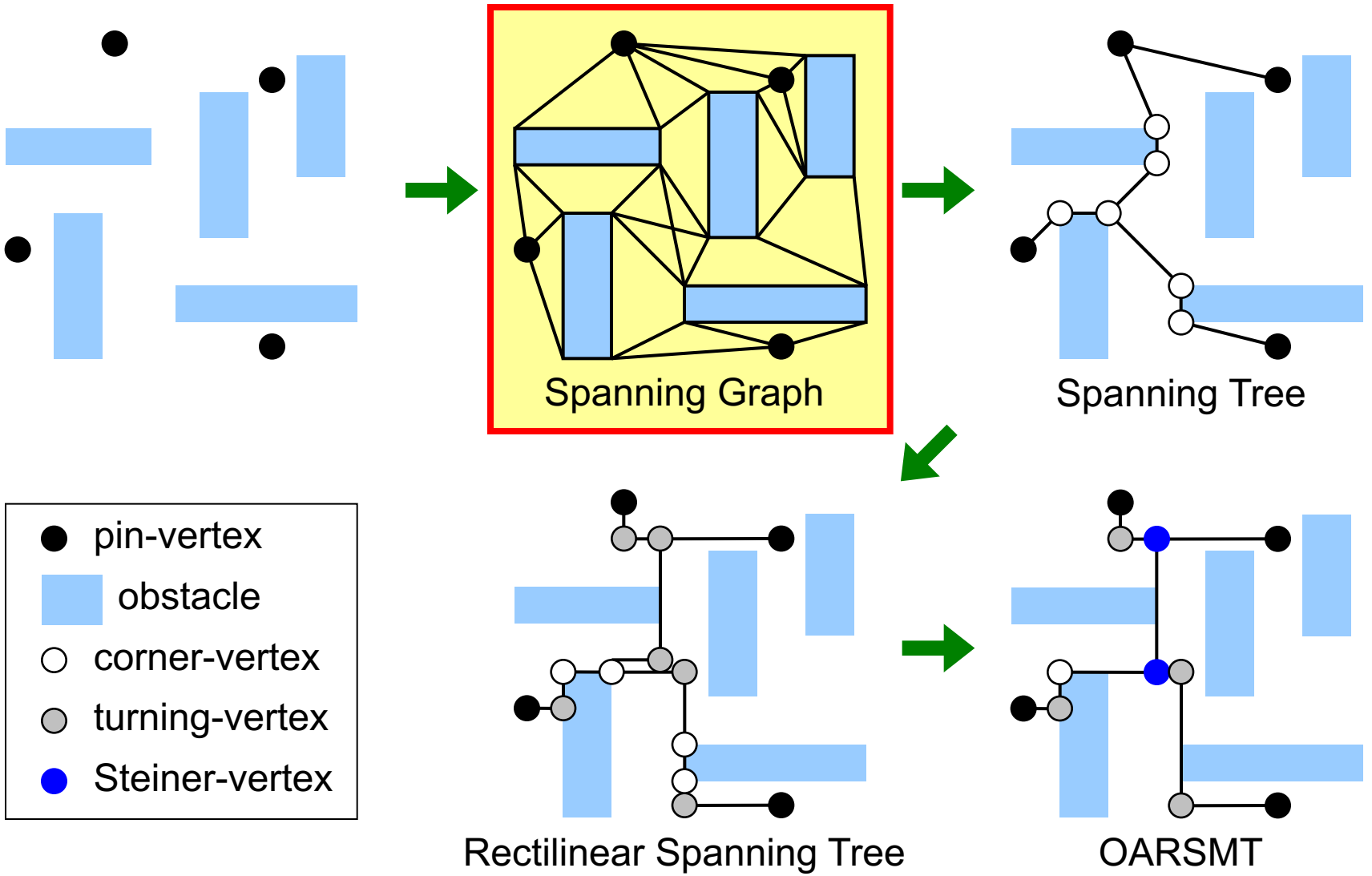
Courtesy of Prof. Y.-W. Chang

Algorithm Flow



Courtesy of Prof. Y.-W. Chang

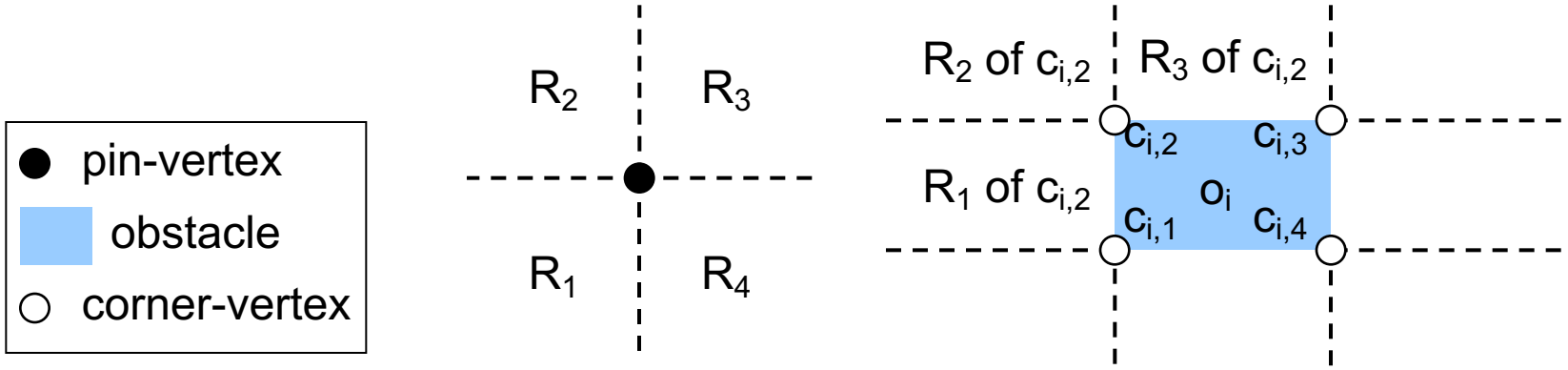
Algorithm Flow



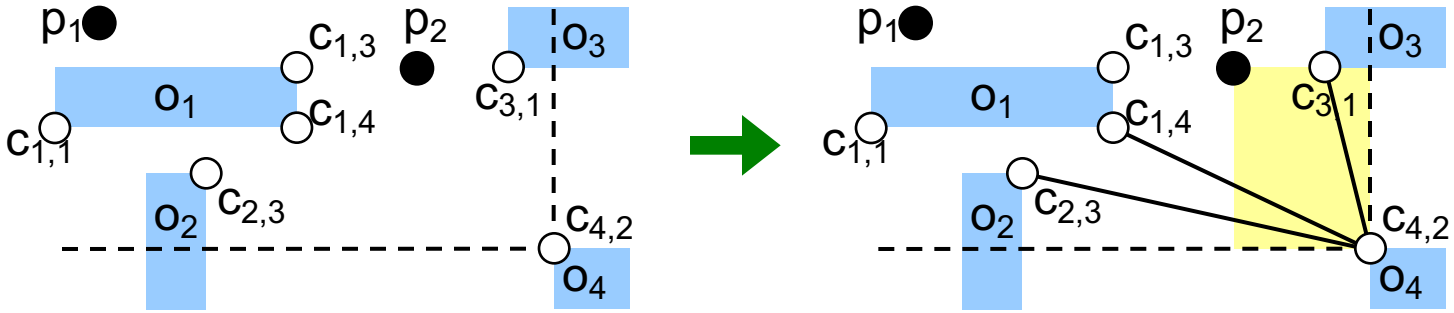
Courtesy of Prof. Y.-W. Chang

Spanning Graph Construction

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



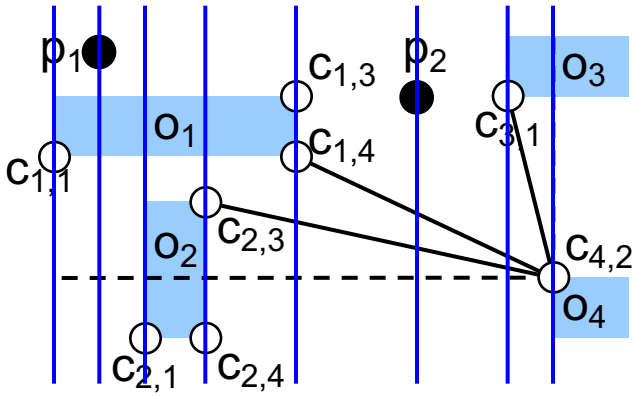
- v_1 and v_2 are connected if no other vertex or obstacle is inside or on the boundary of the bounding box of v_1 and v_2.



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
$c_{3,1}$

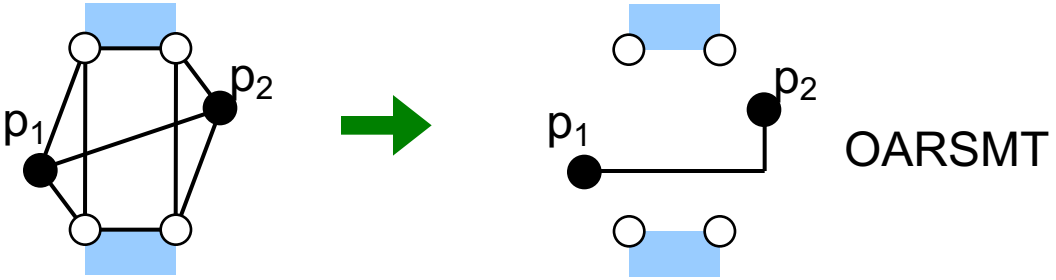
Blocking information

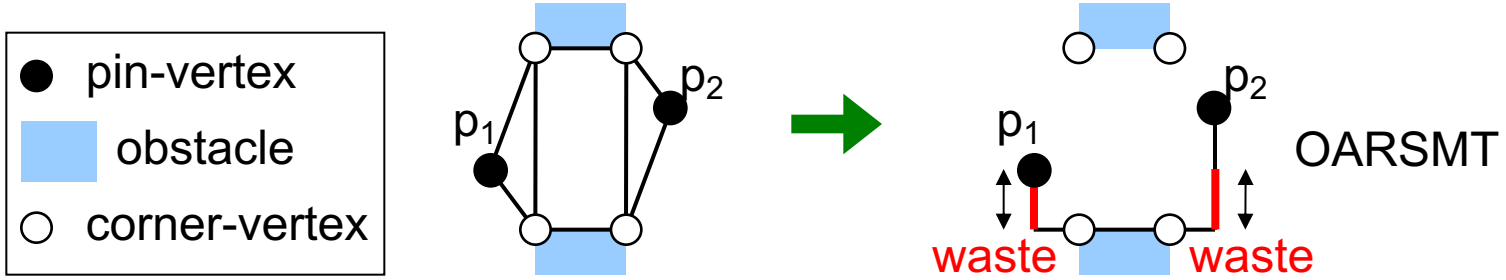
o_1
o_2
o_3

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.

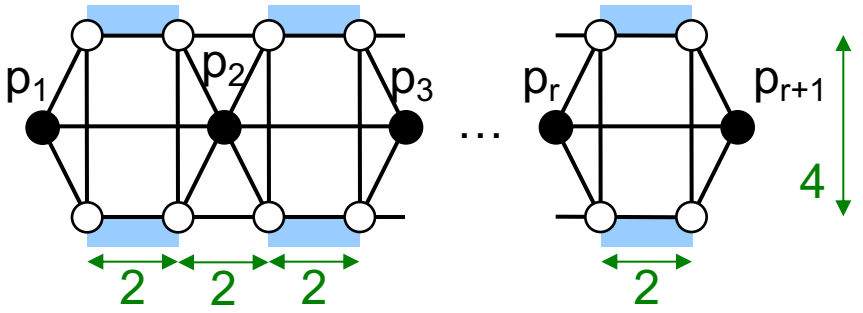




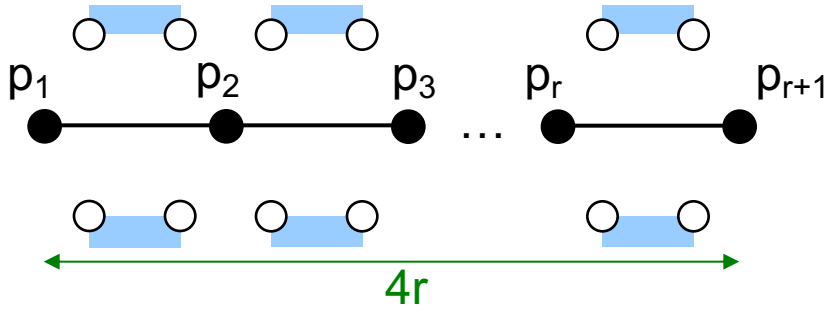
- Spanning graph of Shen et al. (ICCD-05) does not guarantee it.

Courtesy of Prof. Y.-W. Chang

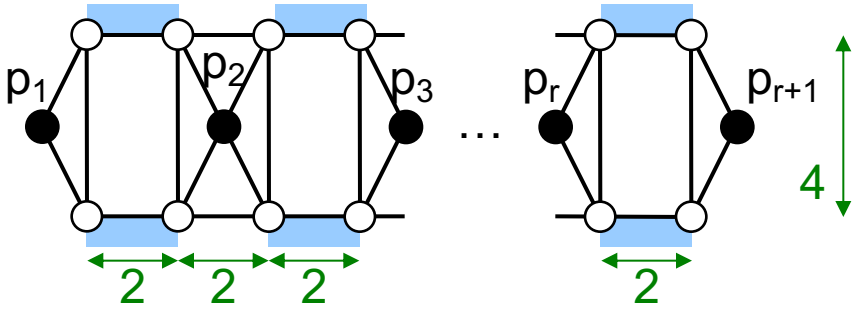
Properties of Our Spanning Graph (2/2)



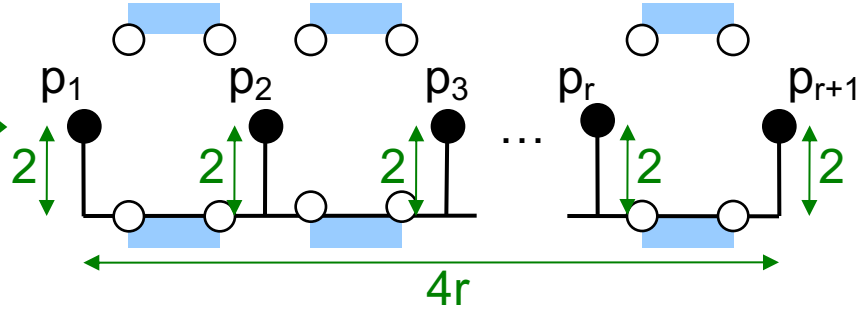
Our spanning graph



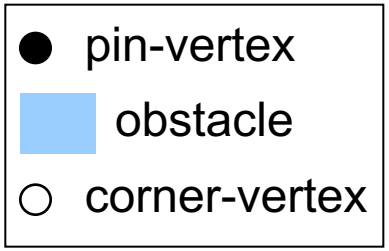
OARSMT total wirelength = 4r



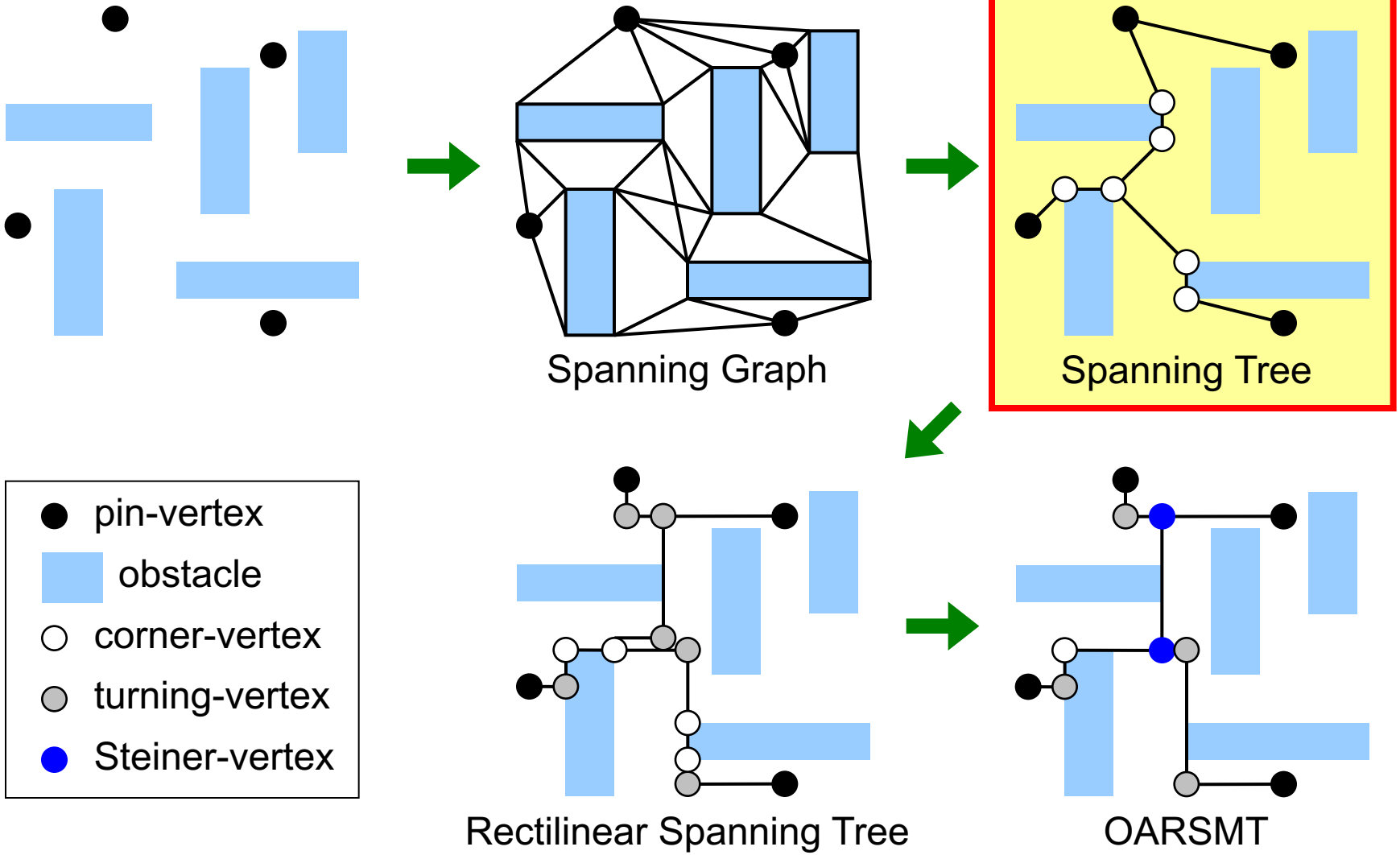
Spanning graph of Shen et al.



OARSMT total wirelength = 6r+2

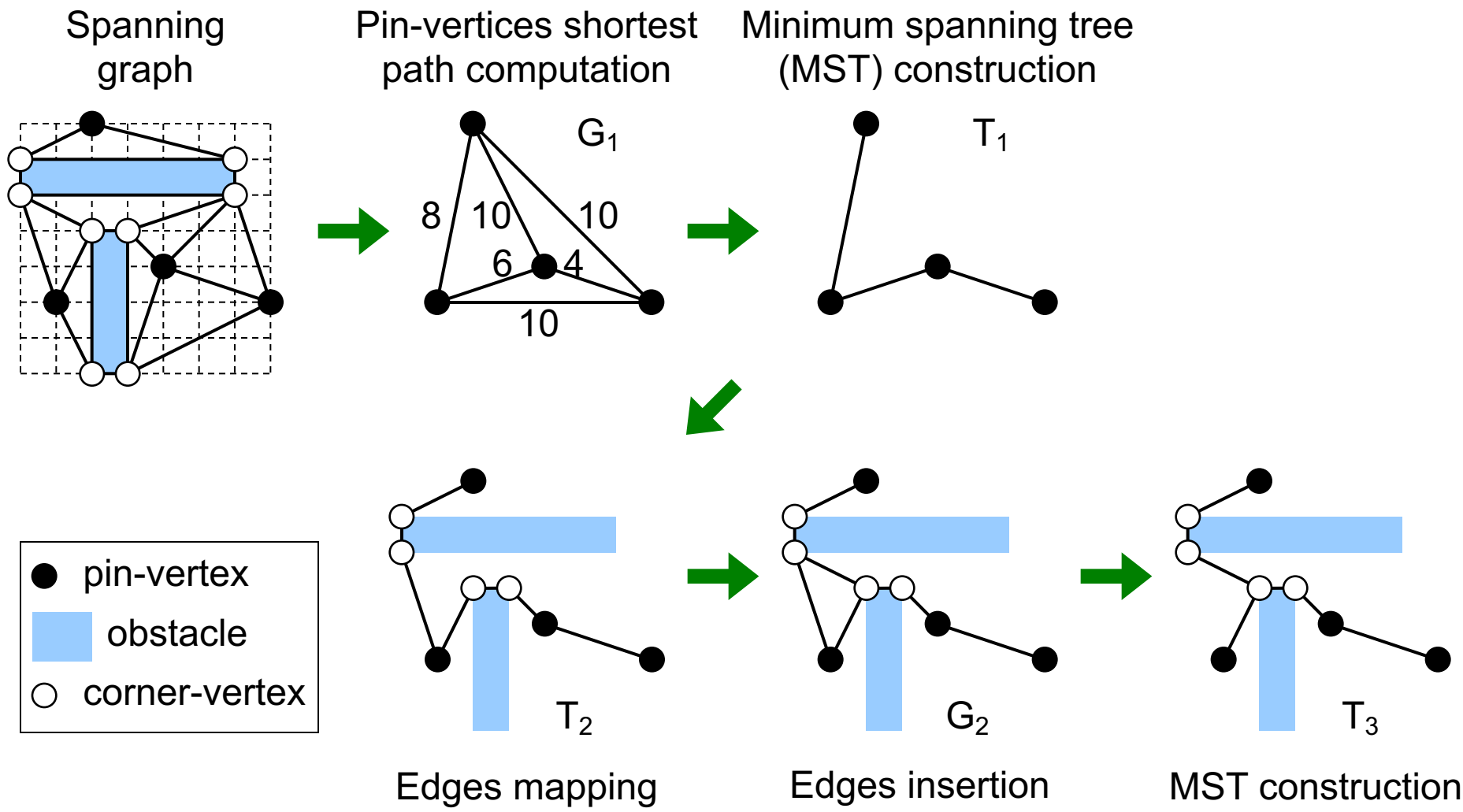


Algorithm Flow



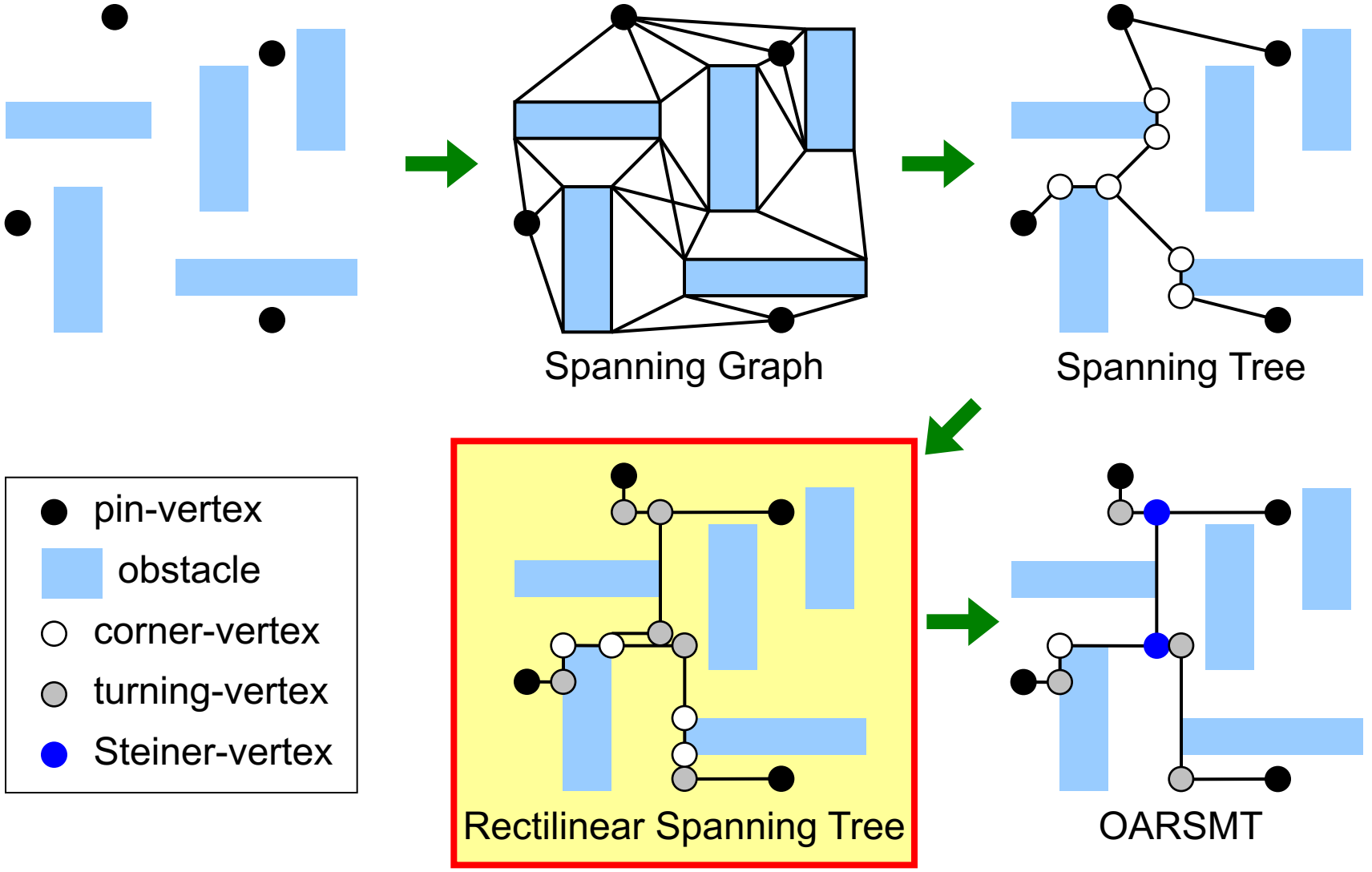
Courtesy of Prof. Y.-W. Chang

Spanning Tree Construction



Courtesy of Prof. Y.-W. Chang

Algorithm Flow

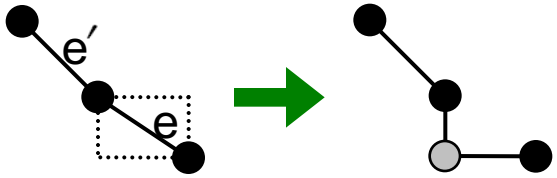


Courtesy of Prof. Y.-W. Chang

Rectilinear Spanning Tree Construction

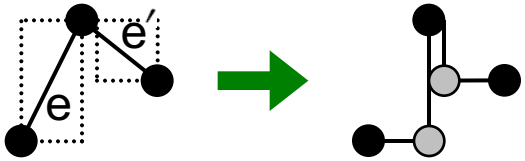
- Transform slant edges into vertical and horizontal edges
  - Longer edges are transformed first.
  - Three cases for a slant edge e and its neighboring edge e':

1. They are in opposite regions.



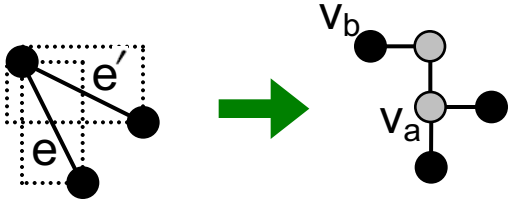
e is transformed randomly

2. They are in neighboring regions.



e and e' are transformed with edge overlap

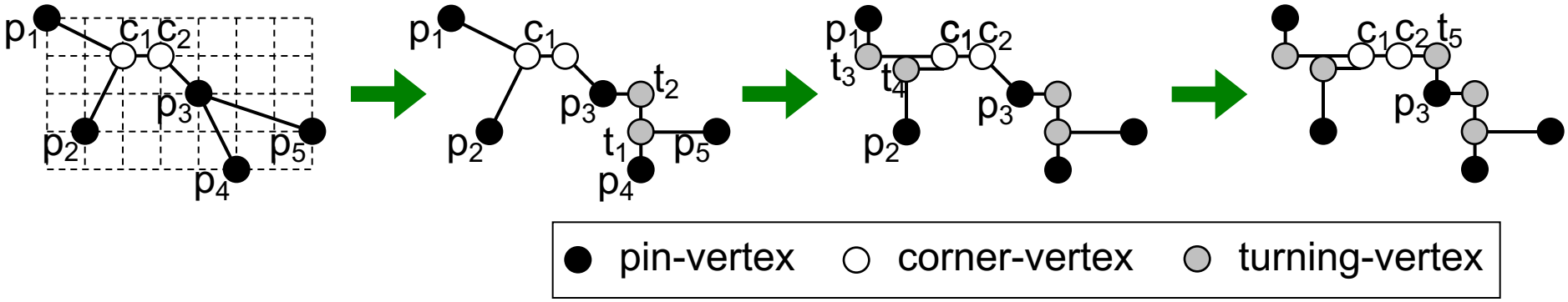
3. They are in the same region.



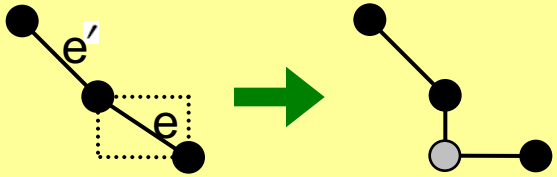
e and e' are transformed with edge overlap
(v_a,v_b) is transformed randomly

Courtesy of Prof. Y.-W. Chang

Rectilinear Spanning Tree Construction – Example

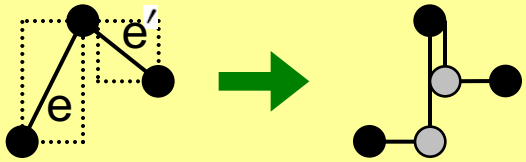


1. They are in opposite regions.



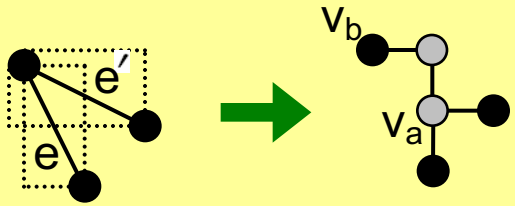
e is transformed randomly

2. They are in neighboring regions.



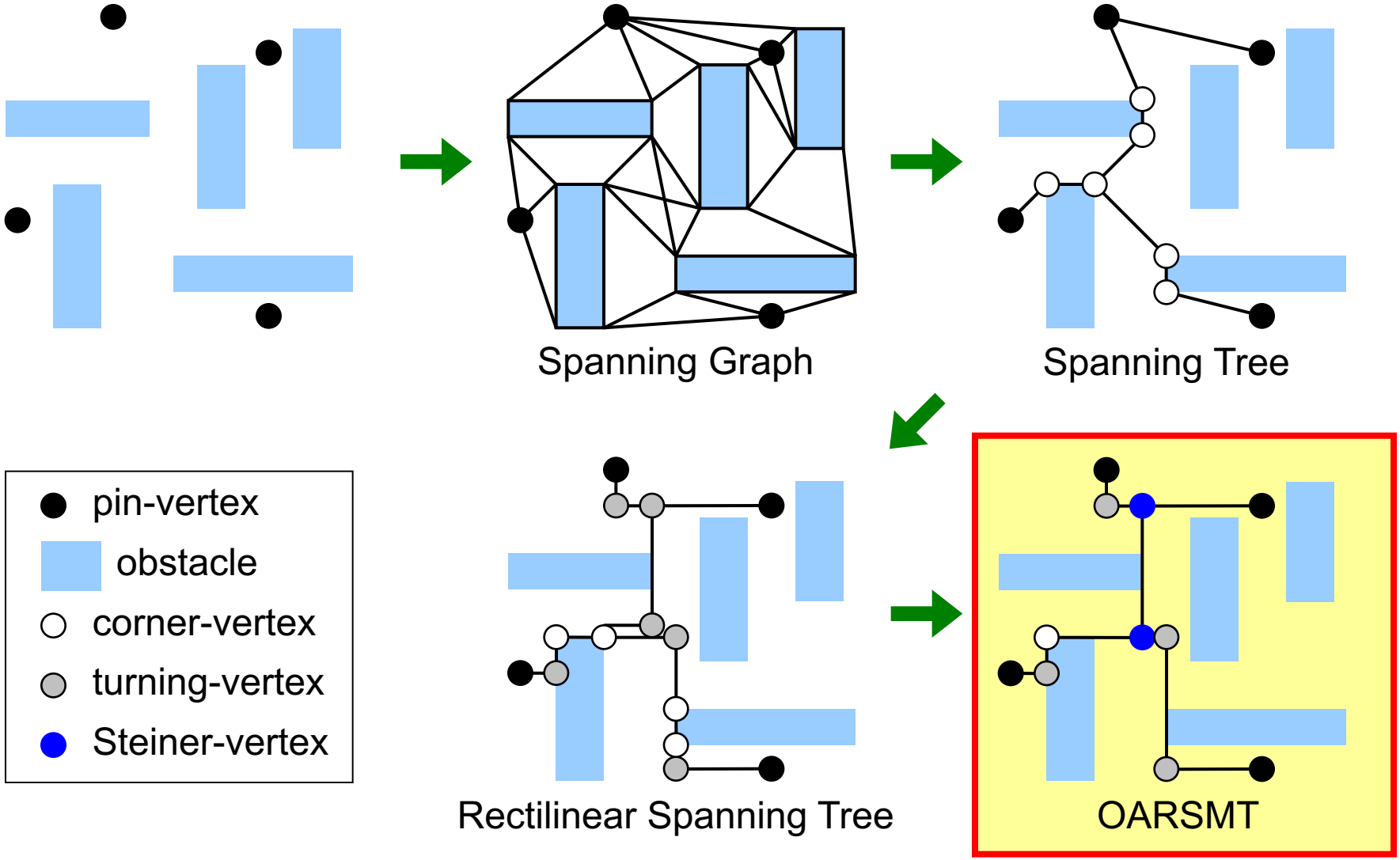
e and e' are transformed with edge overlap

3. They are in the same region.



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

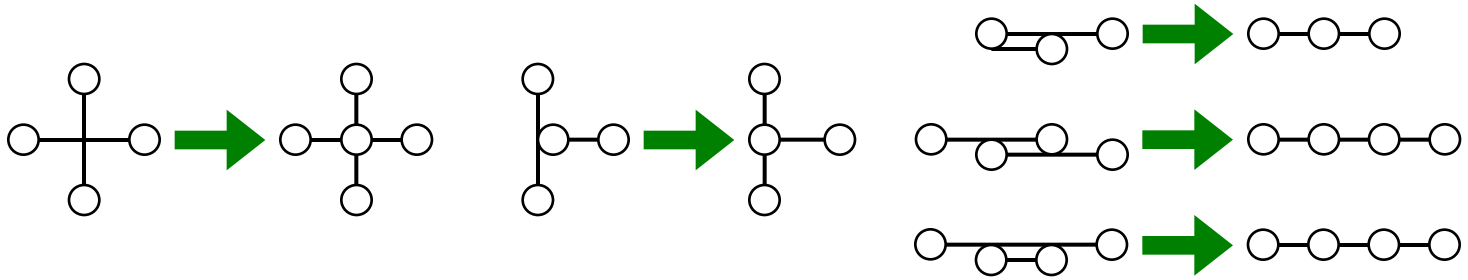
Algorithm Flow



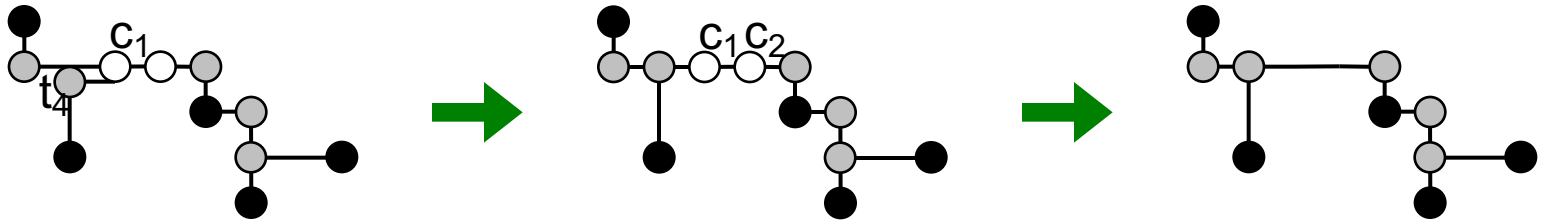
Courtesy of Prof. Y.-W. Chang

OARSMT Construction (1/2)

- Overlapping edge removal



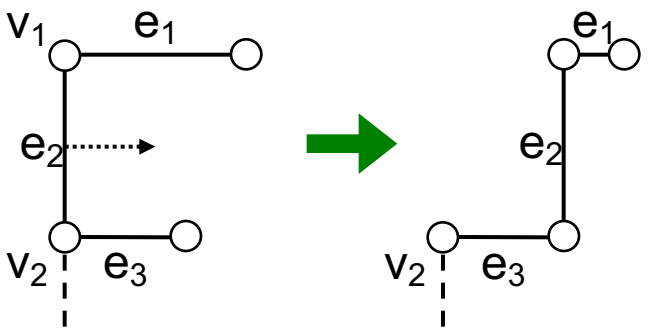
- Redundant vertex removal
  - A redundant-vertex is a non-pin-vertex with the degree of 2, and the two edges connecting to it are parallel.



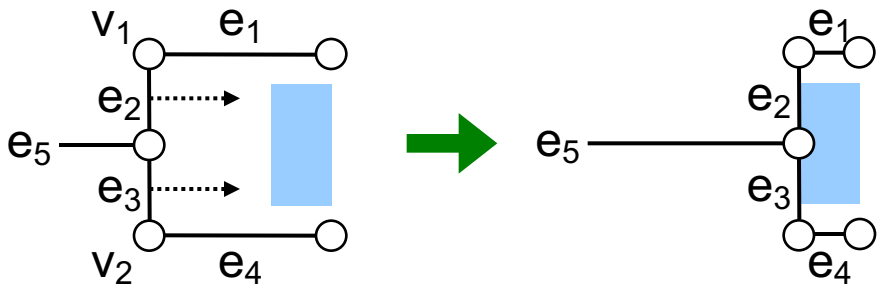
Courtesy of Prof. Y.-W. Chang

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.

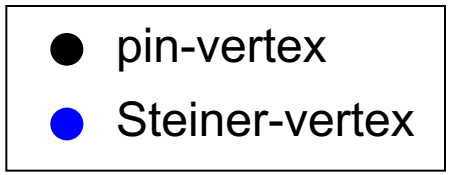


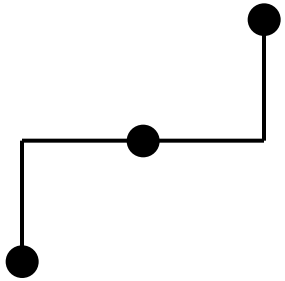
Both vertices v_1 and v_2 must satisfy the refinement rules.

Courtesy of Prof. Y.-W. Chang

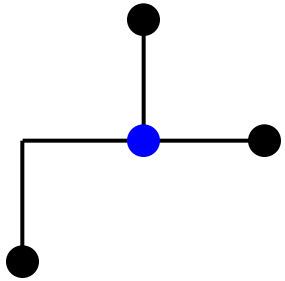
Properties of Our OARSMT

- 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





Two simple paths

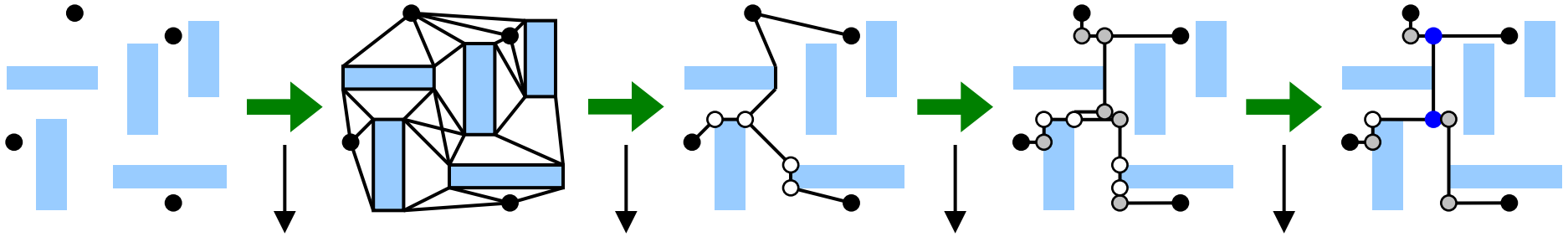


Three paths with Steiner-vertex

- 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.

Courtesy of Prof. Y.-W. Chang

Complexity

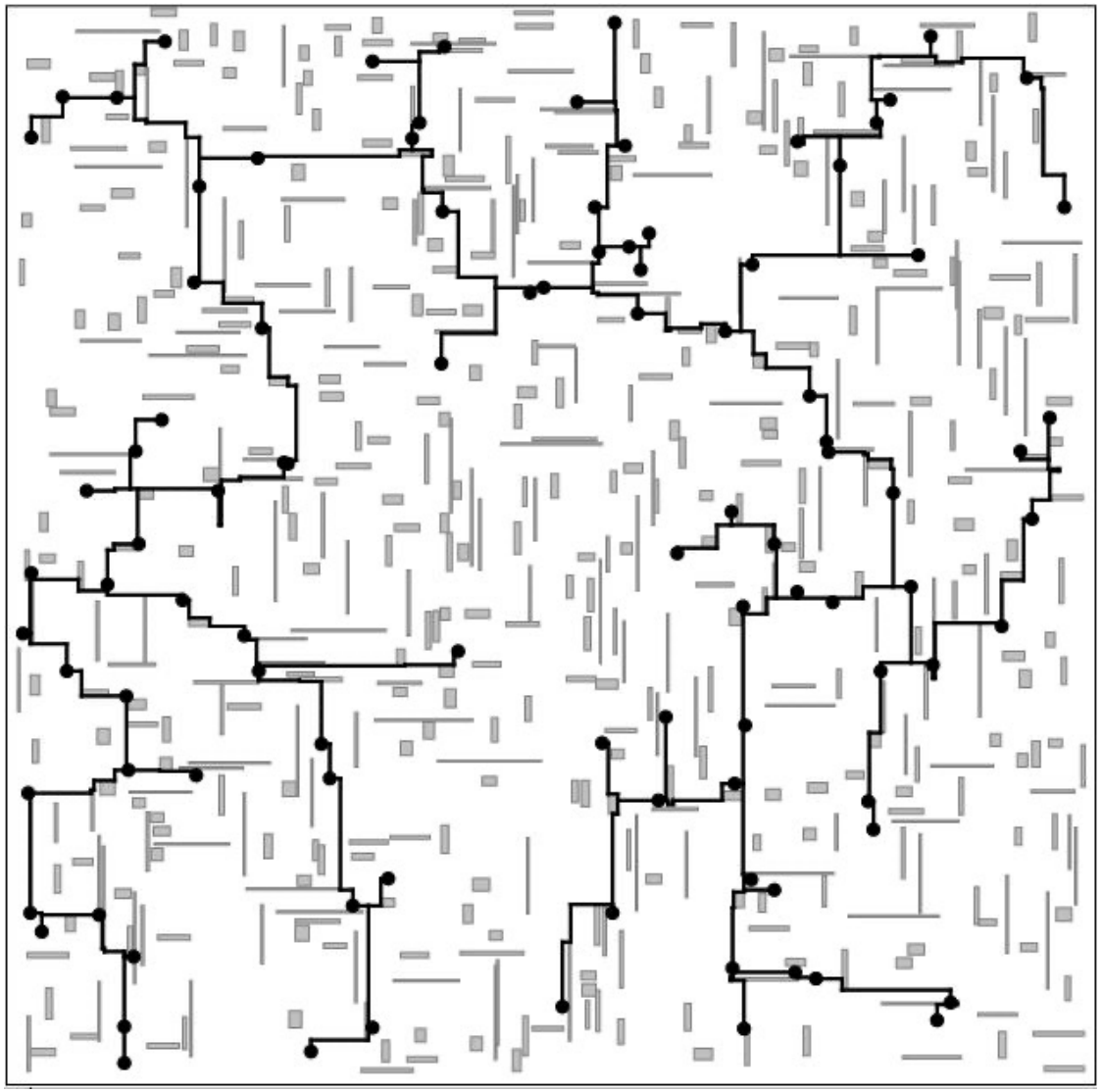


Spanning Graph Construction	Spanning Tree Construction	Rectilinear Spanning Tree Construction	OARSMT Construction
Worst Case			
$O(n^2 \lg n)$	$O(n^3)$	$O(n \lg n)$	$O(n^2)$
Practical Applications (# of edges in spanning graph is O(n).)			
$O(n \lg n)$	$O(n^2 \lg n)$	$O(n \lg n)$	$O(n^2)$

- 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

Courtesy of Prof. Y.-W. Chang

Experimental Results – Routing Result of rt3



Courtesy of Prof. Y.-W. Chang