Routing

Routing

placement

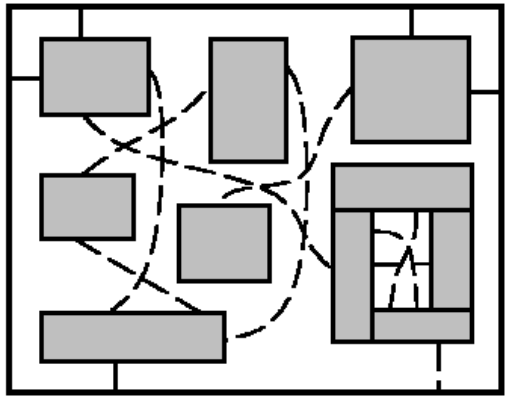
- Generates a "loose" route for each net.
- Assigns a list of routing regions to each net without specifying the actual layout of wires.

global routing

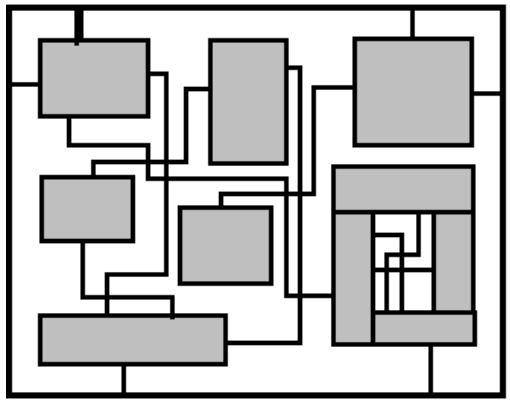
- Finds the actual geometric layout of each net within the assigned routing regions.

detailed routing

compaction



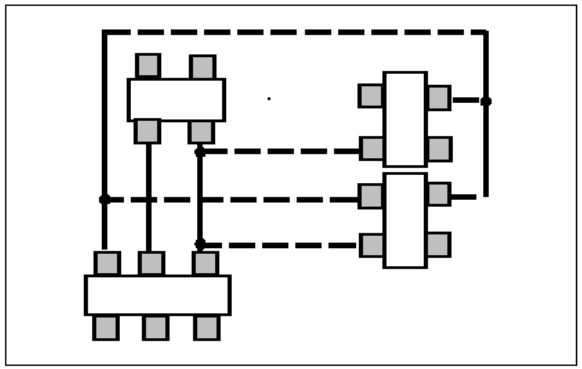
Global routing



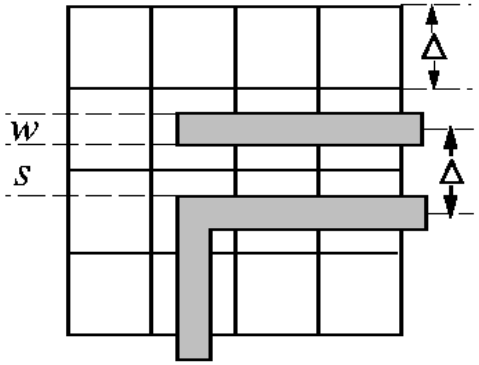
Detailed routing

Routing Constraints

- 100% routing completion + area minimization, under a set of constraints:
  - Placement constraints: usually based on fixed placement
  - Number of routing layers
  - Geometrical constraints: must satisfy design rules
  - Timing constraints (performance-driven routing): must satisfy delay constraints
  - Others



Two-layer routing



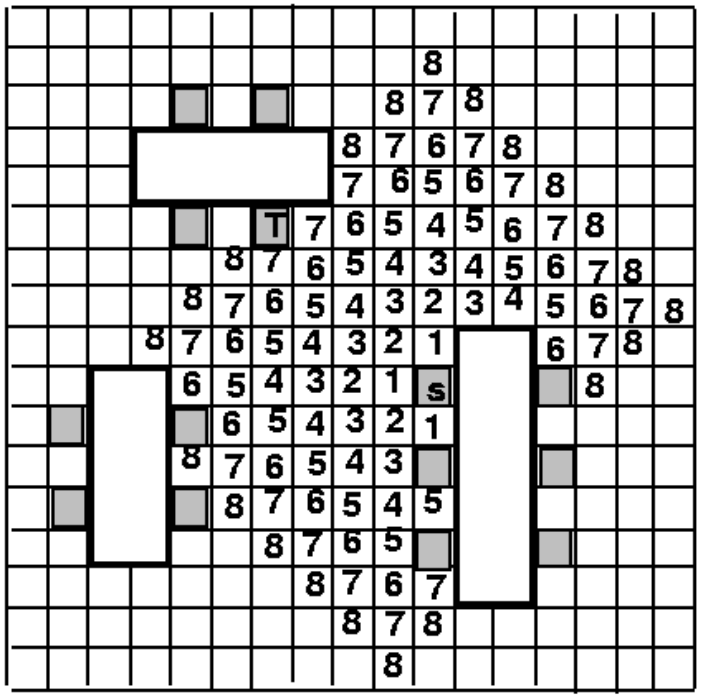
Geometrical constraint

Maze Router: Lee Algorithm

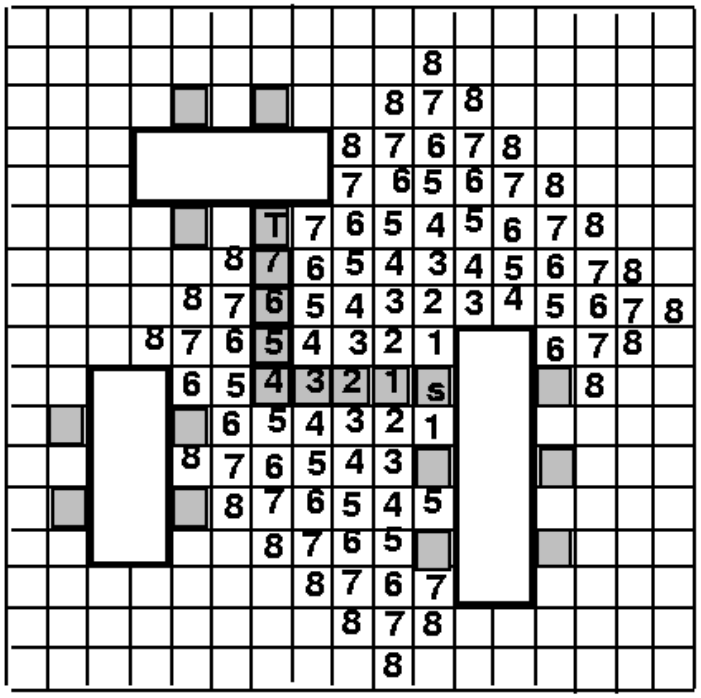
- Lee, “An algorithm for path connection and its application,” *IRE Trans*. Electronic Computer, EC-10, 1961.
- Discussion mainly on single-layer routing
- Strengths
  - Guarantee to find connection between 2 terminals if it exists.
  - Guarantee minimum path.
- Weaknesses
  - Requires large memory for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

Lee Algorithm

- Find a path from *S* to *T*.



Filling (Wave propagation)

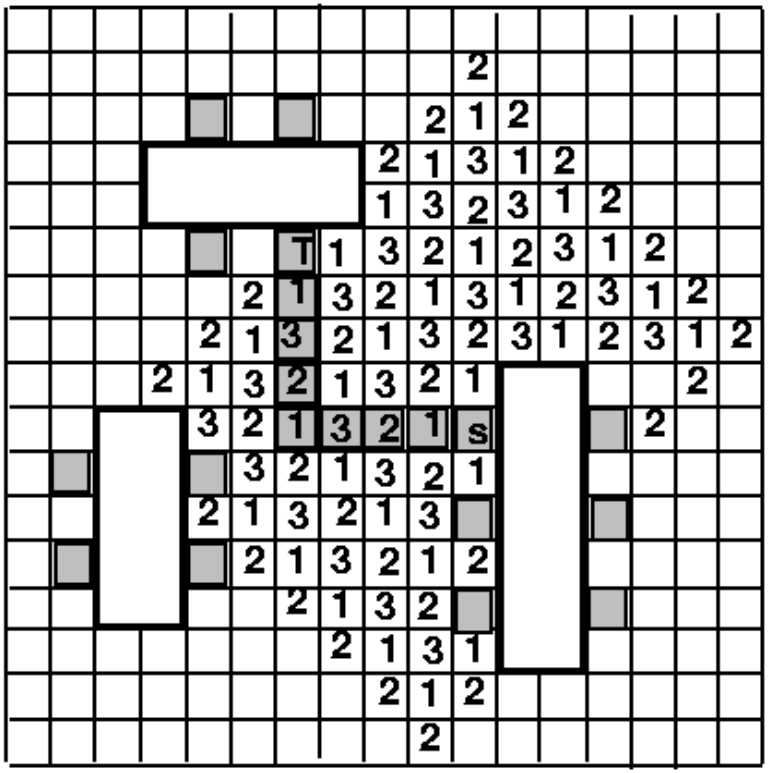


Retrace

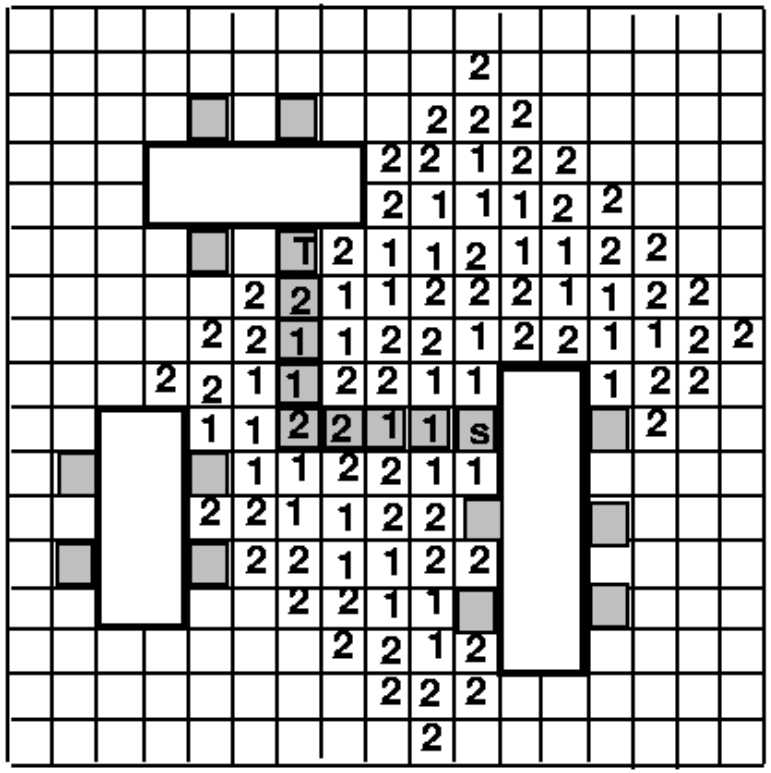
- Time & space complexity for an *M***N* grid: *O*(*MN*) (**huge!**)

Reducing Memory Requirement

- Akers's Observation (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)



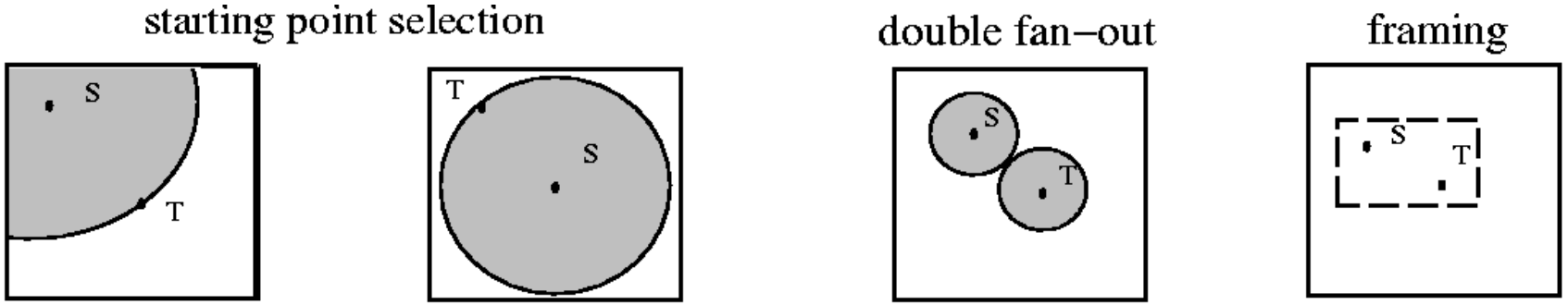
Sequence: 1, 2, 3, 1, 2, 3, ...



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

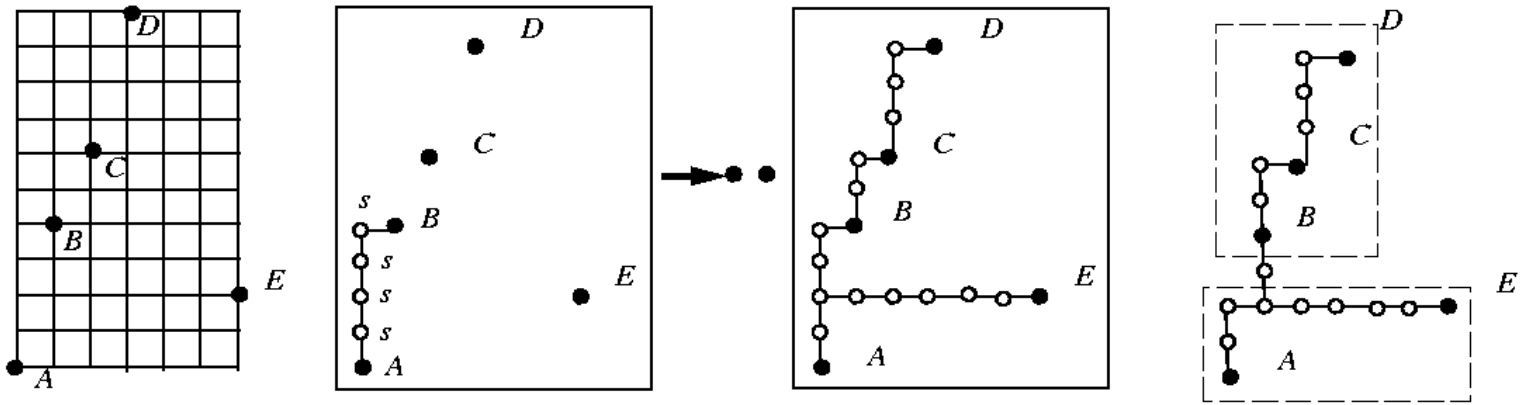
Reducing Running Time

- **Starting point selection:** Choose the point farthest from the center of the grid as the starting pint.
- **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.



Connecting Multi-Terminal Nets

- Step 1: Propagate wave from the source s to the closet target.
- Step 2: Mark ALL cells on the path as *s*.
- Step 3: Propagate wave from ALL *s* cells to the other cells.
- Step 4: Continue until all cells are reached.
- Step 5: Apply heuristics to further reduce the tree cost.

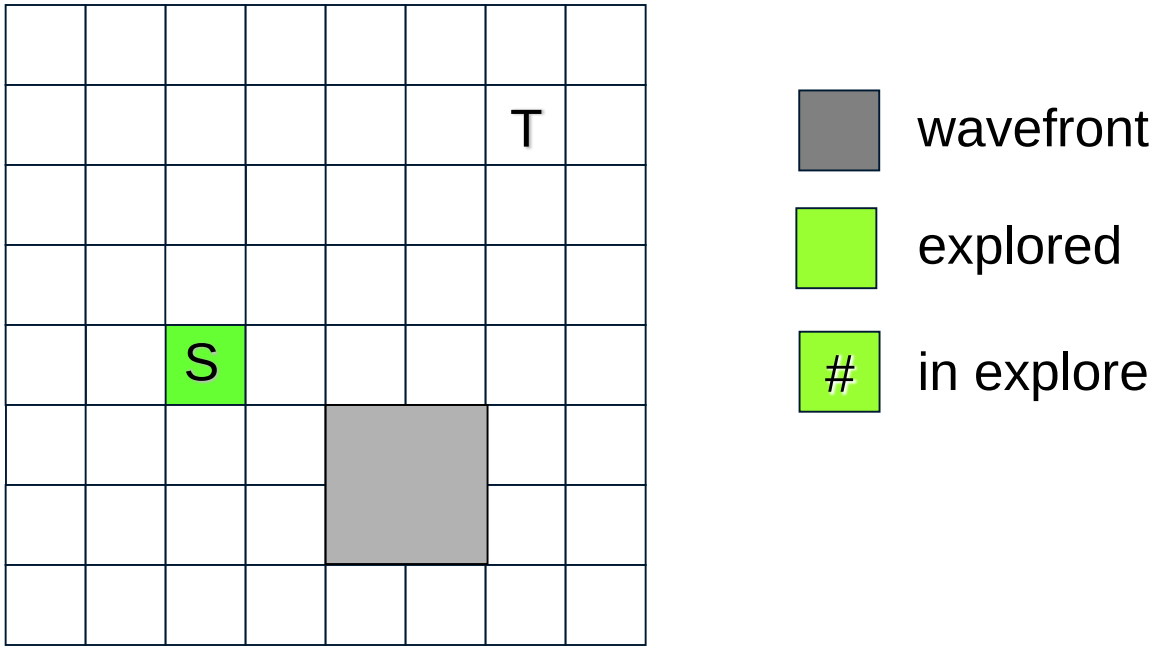


A*-Search Routing

- 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 route 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

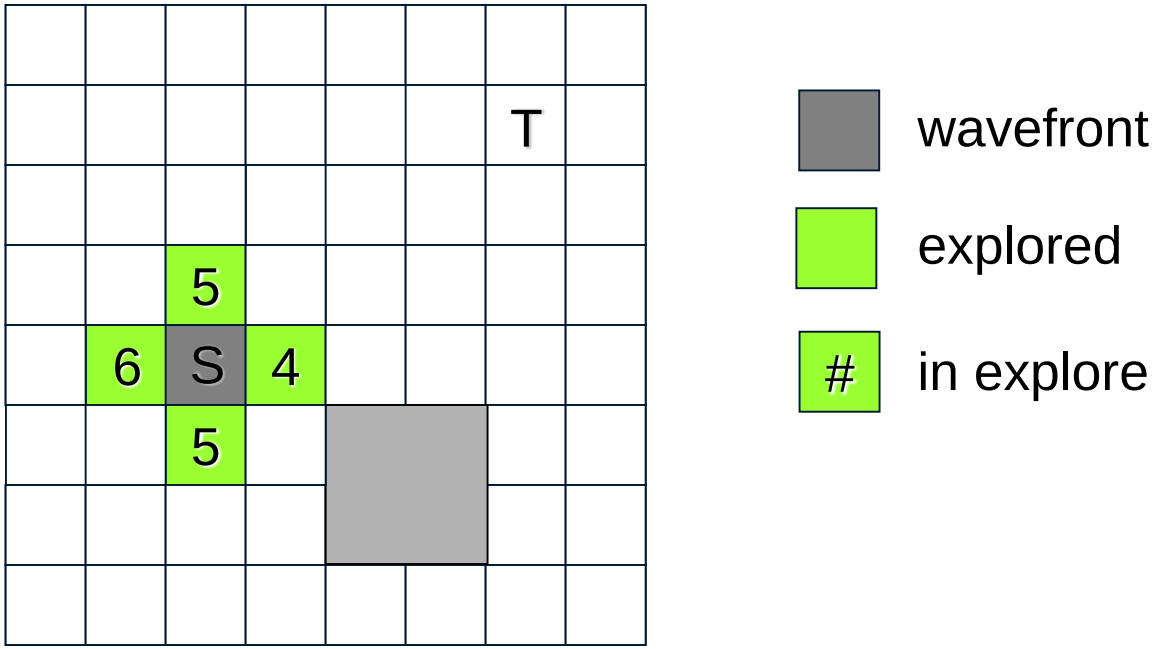
An Example of A*-Search Routing

- 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)
  - $h(x)$: $\max(\text{dist}_x(T, x), \text{dist}_y(T, x))$



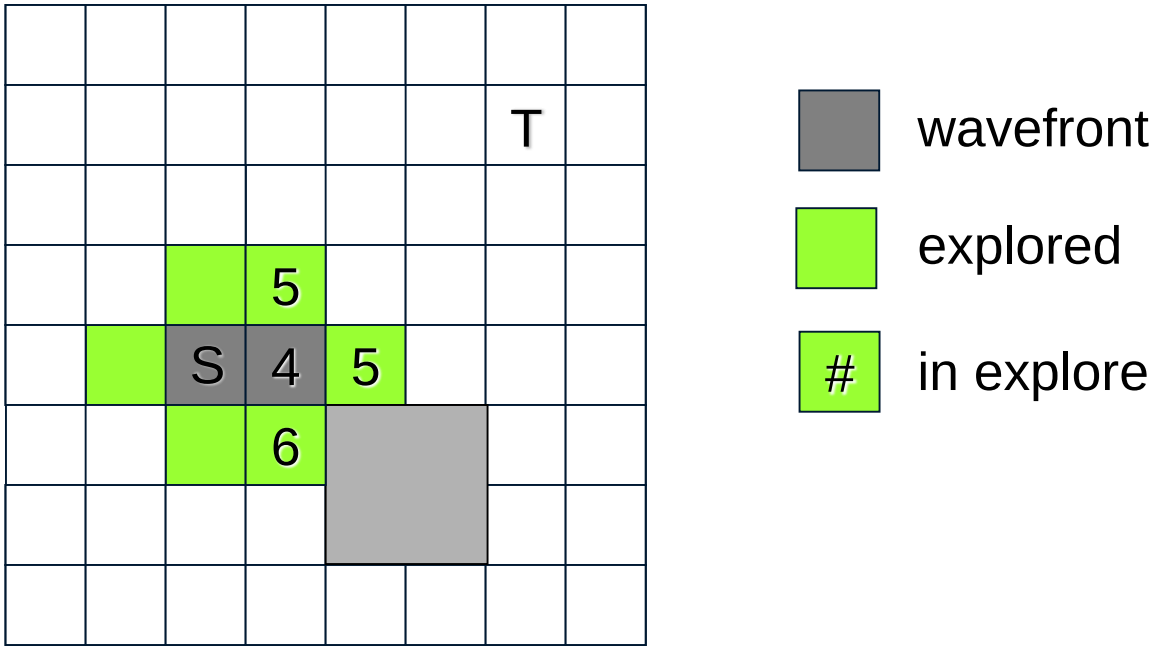
An Example of A*-Search Routing

- 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)
  - $h(x)$: $\max(\text{dist}_x(T, x), \text{dist}_y(T, x))$



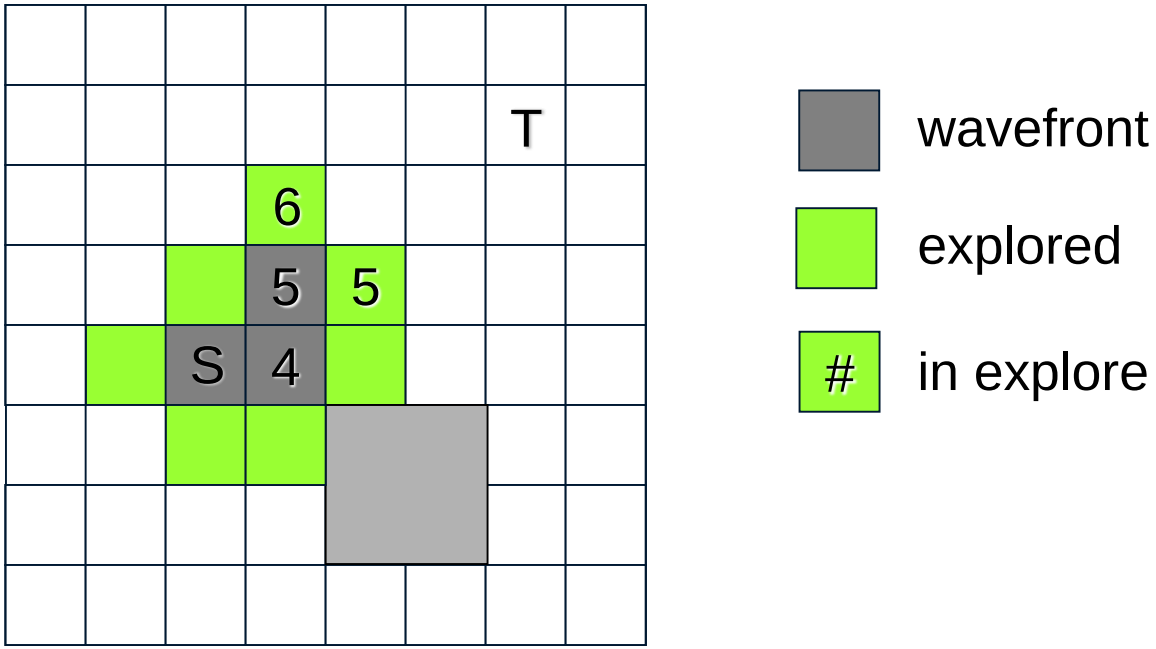
An Example of A*-Search Routing

- 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)
  - $h(x)$: $\max(\text{dist}_x(T, x), \text{dist}_y(T, x))$



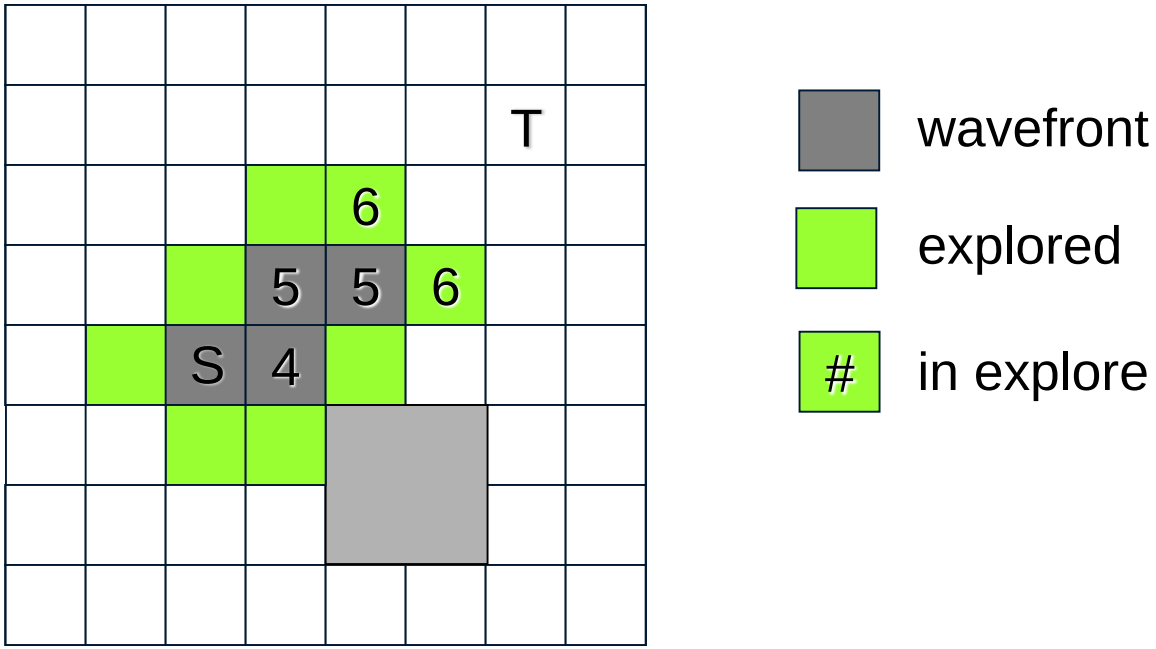
An Example of A*-Search Routing

- 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)
  - $h(x)$: $\max(\text{dist}_x(T, x), \text{dist}_y(T, x))$



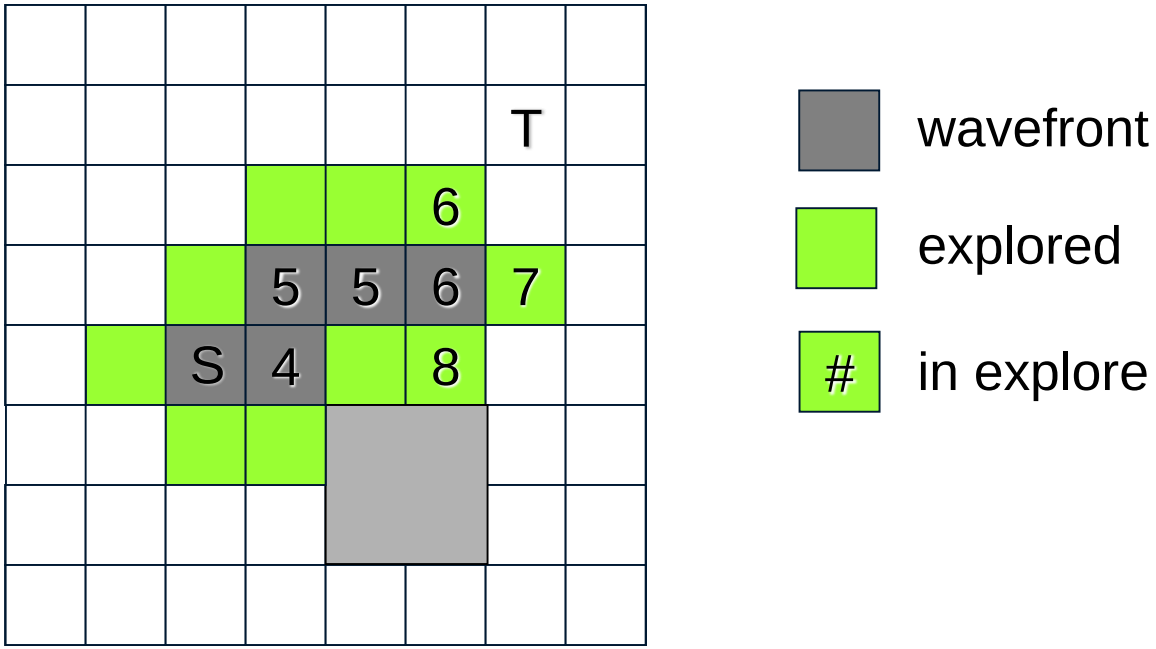
An Example of A*-Search Routing

- 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)
  - $h(x)$: $\max(\text{dist}_x(T, x), \text{dist}_y(T, x))$



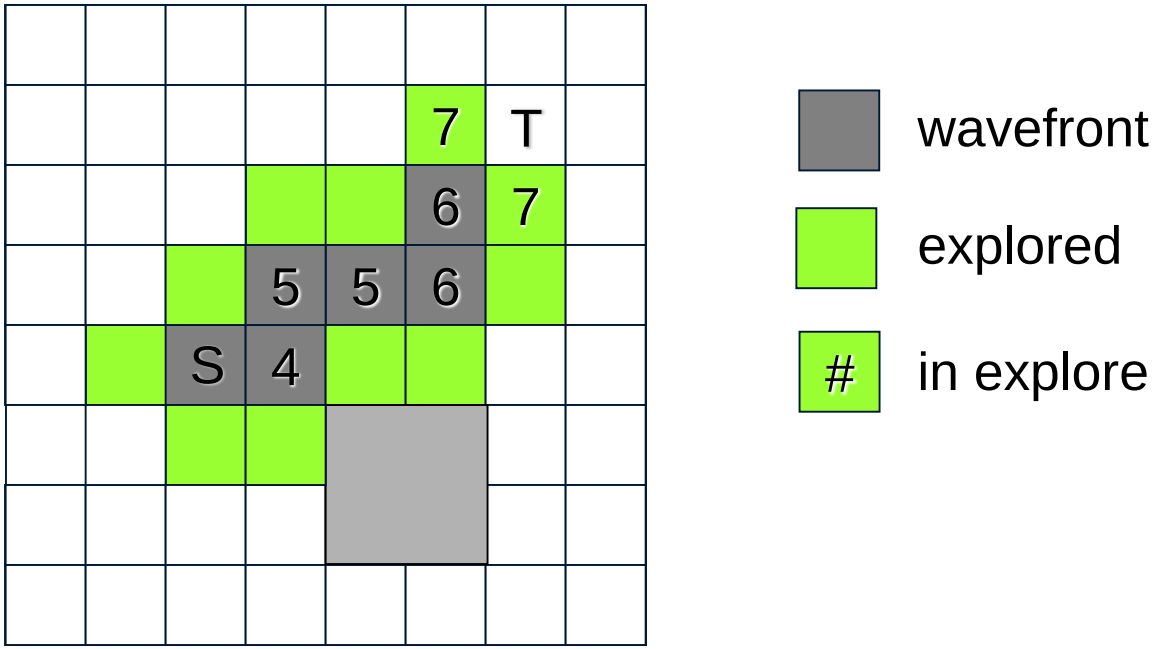
An Example of A*-Search Routing

- 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)
  - $h(x)$: $\max(\text{dist}_x(T, x), \text{dist}_y(T, x))$



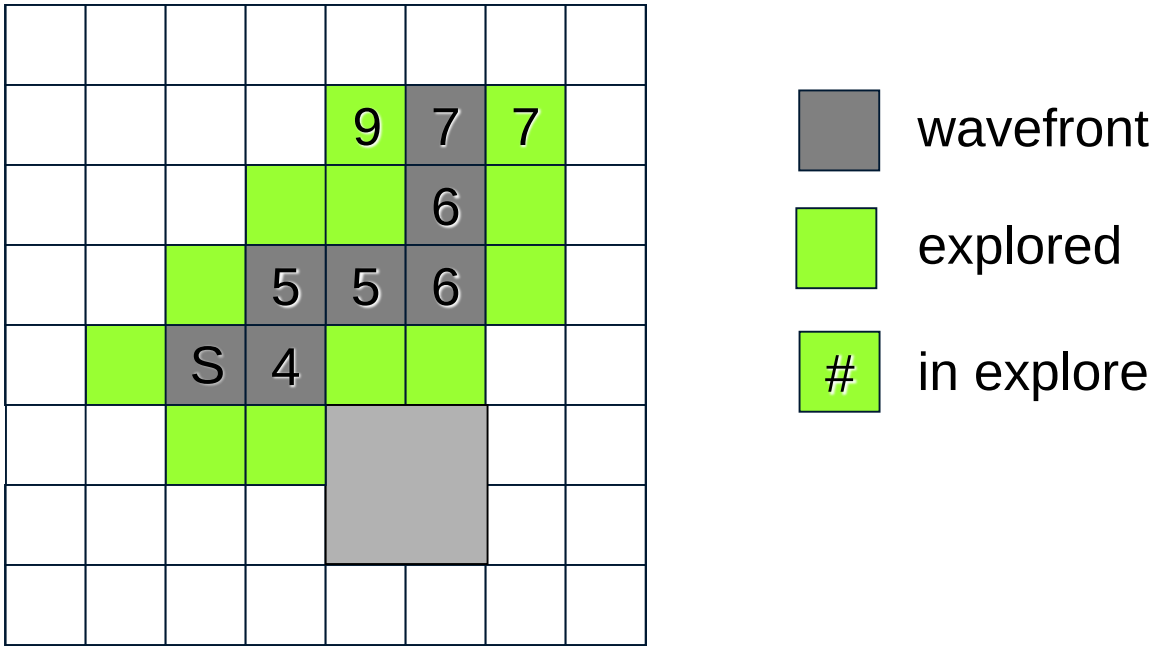
An Example of A*-Search Routing

- 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)
  - $h(x)$: $\max(\text{dist}_x(T, x), \text{dist}_y(T, x))$



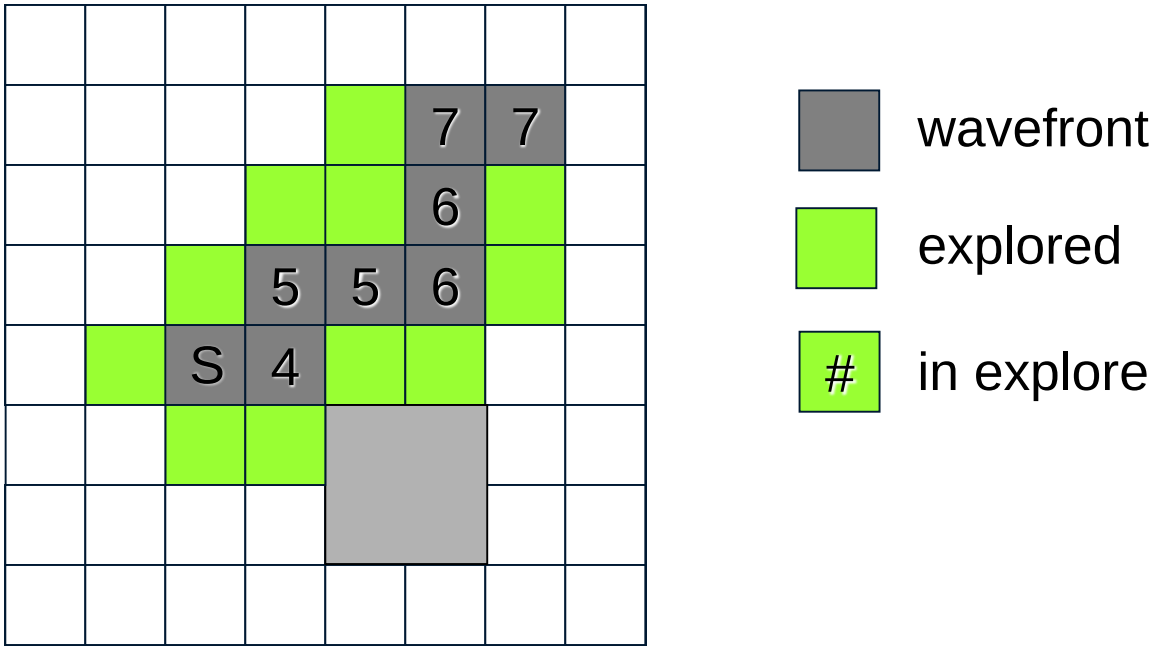
An Example of A*-Search Routing

- 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)
  - $h(x)$: $\max(\text{dist}_x(T, x), \text{dist}_y(T, x))$



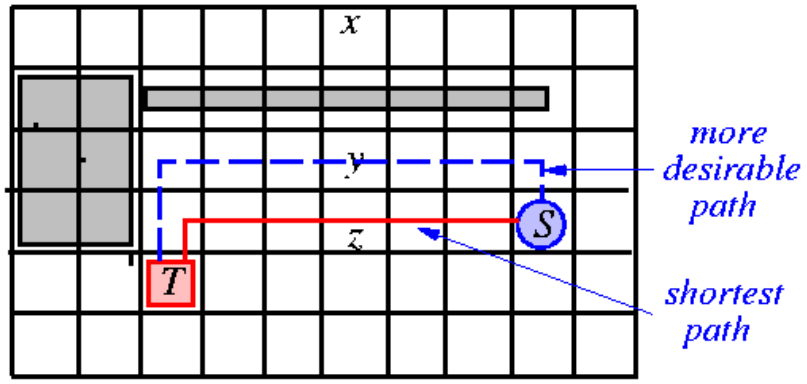
An Example of A*-Search Routing

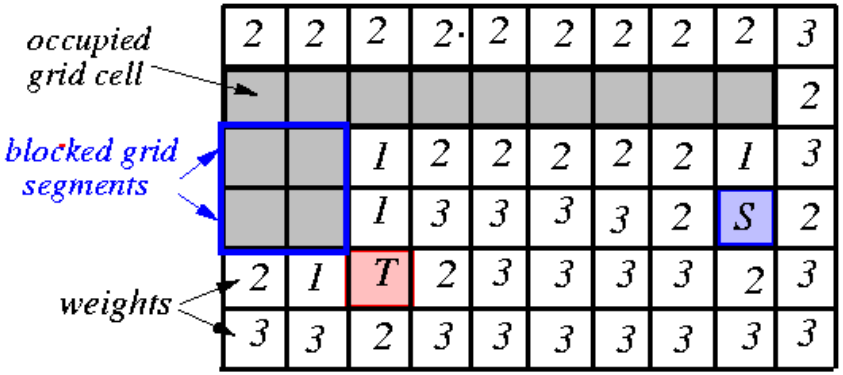
- Cost function for a path x: $f(x) = g(x) + h(x)$
  - $g(x)$: the label from the source S to the current node of x (*i.e.*, the lable used in maze routing)
  - $h(x)$: $\max(\text{dist}_x(T, x), \text{dist}_y(T, x))$



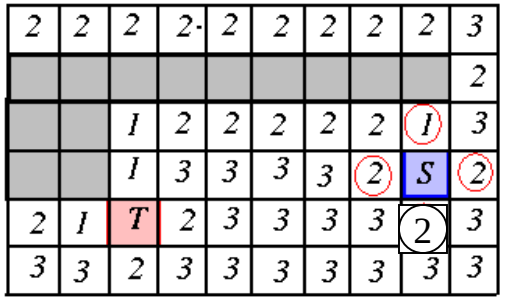
Routing on a Weighted Grid

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

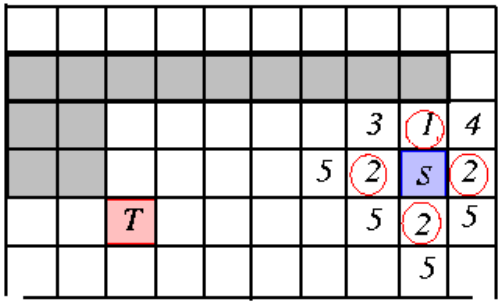




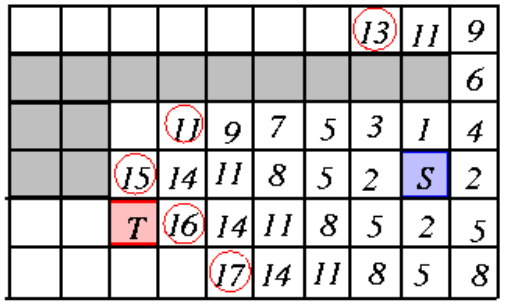
A Routing Example on a Weighted Grid

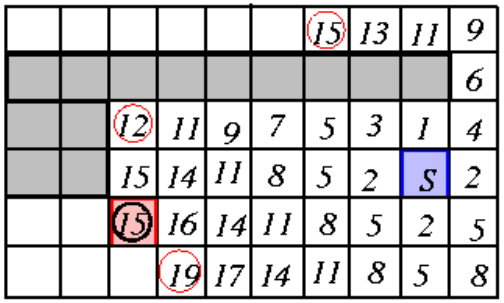


initialize cell weights

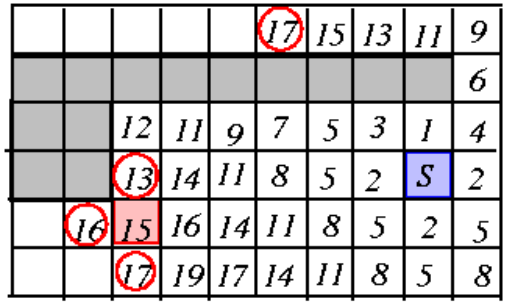


wave propagation

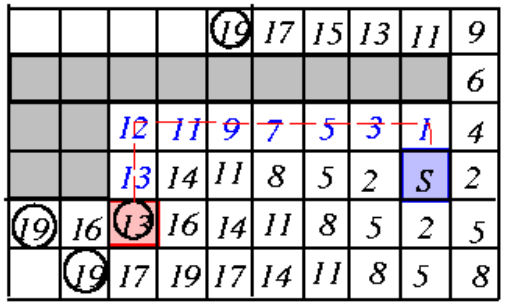




first wave reaches the target



finding other paths

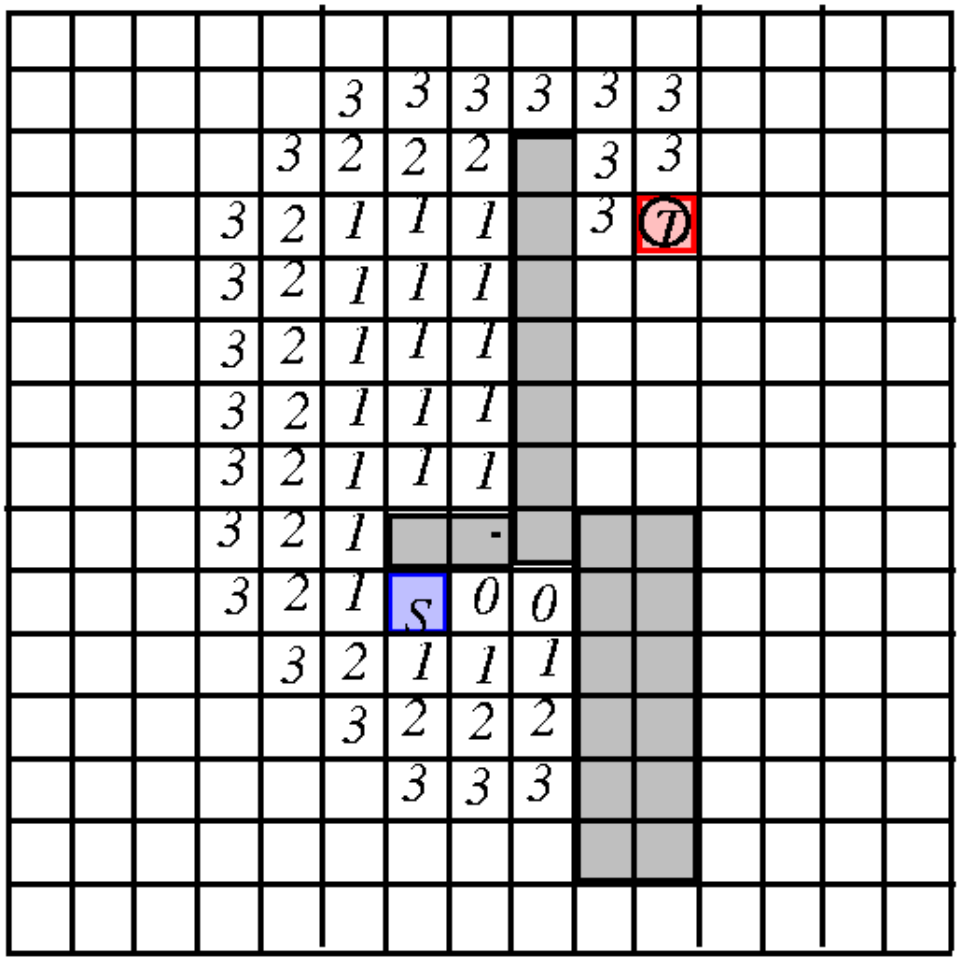


min–cost path found

Hadlock's Algorithm

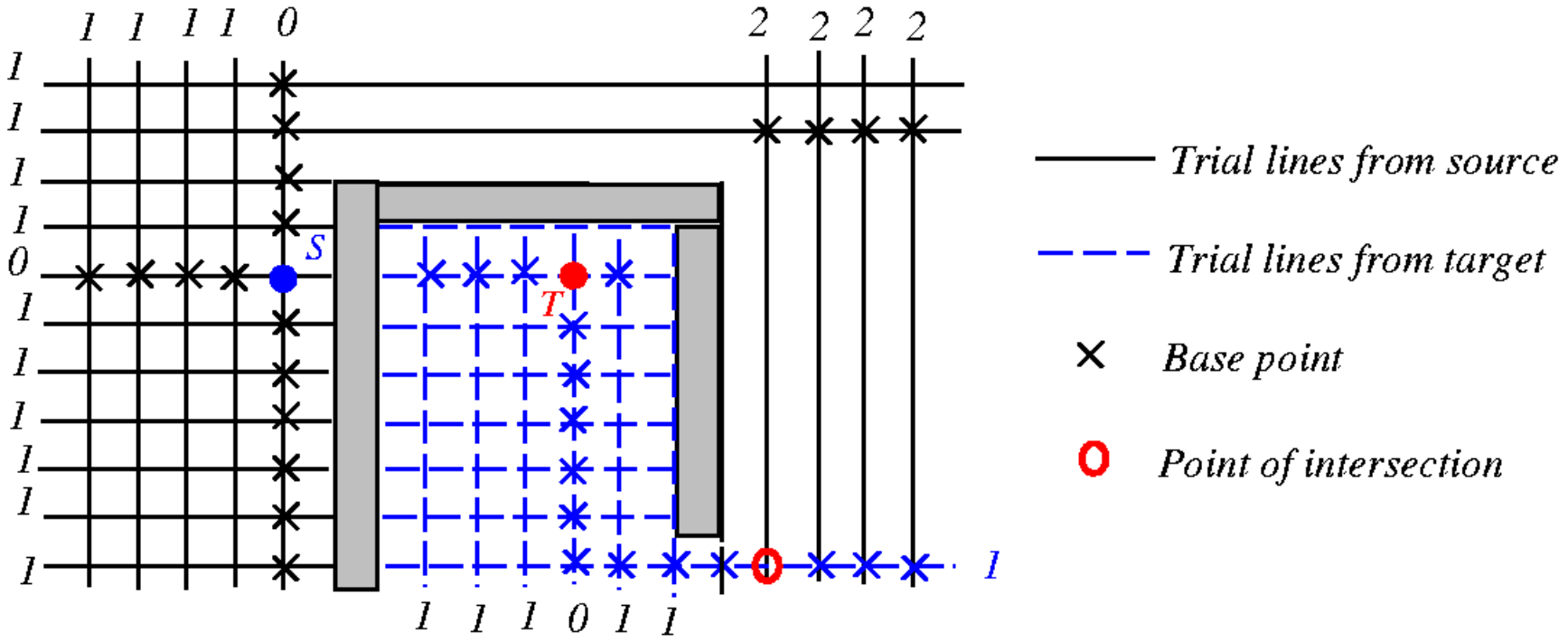
- Hadlock, "A shortest path algorithm for grid graph," *Networks*, 1977.
- Uses detour number (instead of labeling wavefront in Lee's router)
  - Detour number, $d(P)$: # of grid cells directed **away from** its target on path P.
  - $MD(S,T)$: the Manhattan distance between S and T.
  - Path length of P, $l(P)$: $l(P) = MD(S,T) + 2d(P)$.
  - $MD(S,T)$ 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.
- 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)



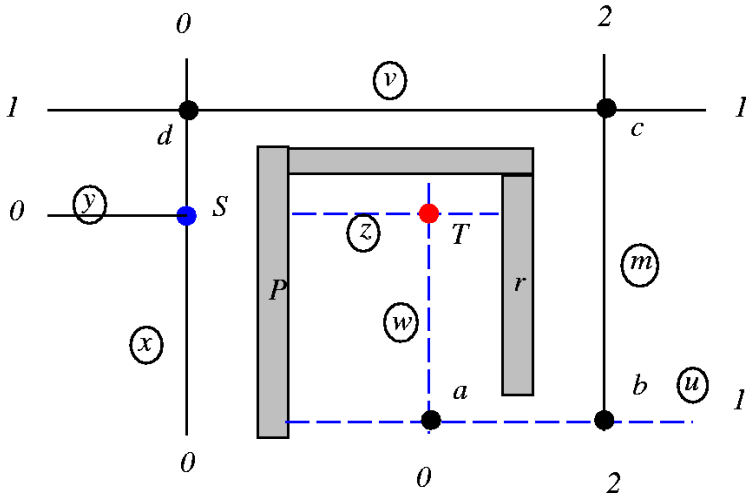
Mikami-Tabuchi's Algorithm

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



Hightower's Algorithm

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

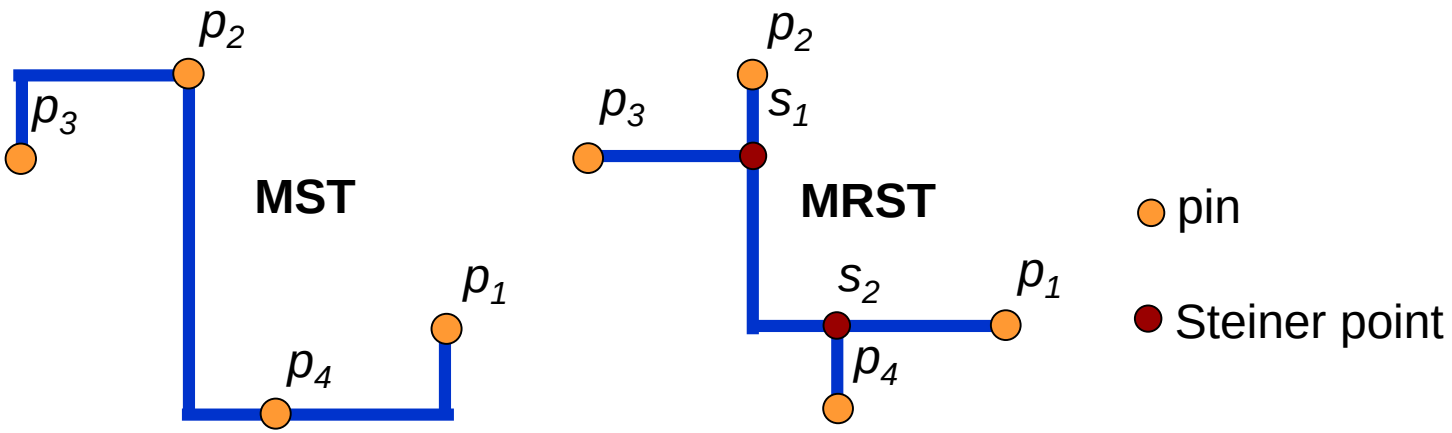


Routing Tree

- If all nets are two-pin ones, we can apply a general-purpose routing algorithm to handle the problem, such as maze, line-search, and A*-search routing.
- For three or more multi-pin nets, one approach is to *decompose* each net into a set of two-pin connections, and then routes the connections one-by-one.

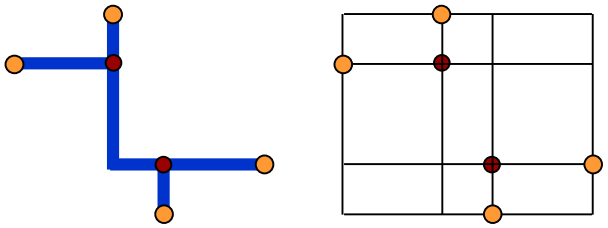
The Routing-Tree Problem

- **Problem:** Given a set of pins of a net, interconnect the pins by a "routing tree."
- **Minimum Spanning Tree (MST):** a minimum-length tree of edges connecting all the pins
- **Minimum Rectilinear Steiner Tree (MRST) Problem:** Given n points in the plane, find a minimum-length tree of rectilinear edges which connects the points. (Very useful in routing of VLSI circuits, but NP-hard)
- $MRST(P) = MST(P \cup S)$, where P and S are the sets of original points and Steiner points, respectively.



Theoretic Results for the RSMT Problem

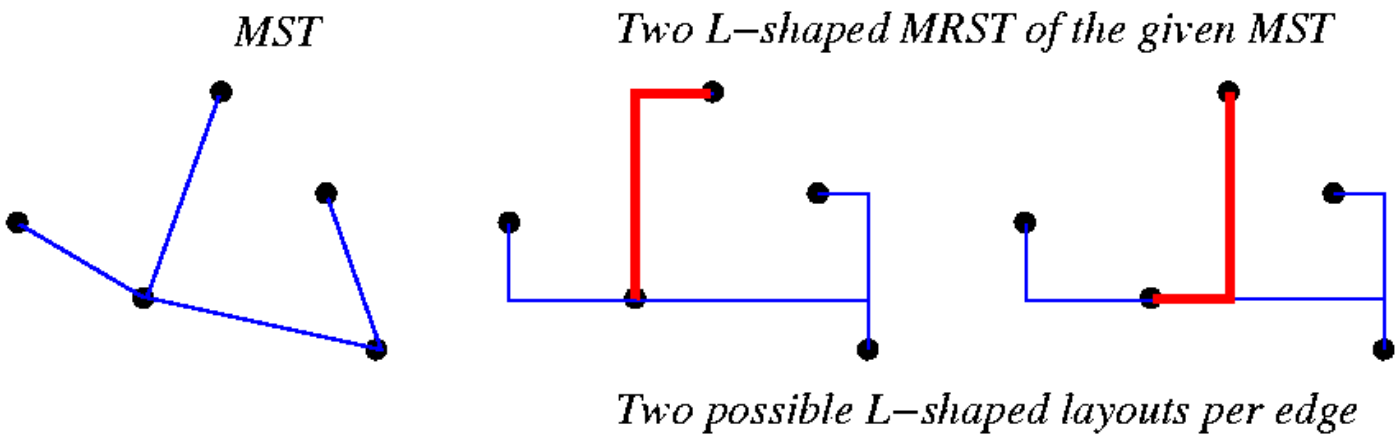
- **Hanan's Thm:** There exists an MRST with all Steiner points (set *S*) chosen from the intersection points of horizontal and vertical lines drawn from points of *P*.
  - Hanan, "On Steiner's problem with rectilinear distance," *SIAM J. Applied Math.*, 1966.



- **Hwang's Theorem:** For any point set *P*, $\frac{Cost(MST(P))}{Cost(MRST(P))} \leq \frac{3}{2}$.
  - Hwang, "On Steiner minimal tree with rectilinear distance," *SIAM J. Applied Math.*, 1976.
- Better approximation algorithm with the performance bound 61/48
  - Foessmeier *et al*, "Fast approximation algorithm for the rectilinear Steiner problem," Wilhelm Schickard-Institut für Informatik, TR WSI-93-14, 93.

Minimum Spanning Tree Based Heuristic

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



Repeated 1-Steiner Tree Heuristic

- A. B. Kahng and G. Robins, "A new class of Steiner tree heuristics with good performance: the iterated 1-Steiner approach," ICCAD, 1990.
- **1-Steiner tree problem**: the minimal Steiner tree problem with the restriction that the tree contains only a single Steiner point.
- The optimal solution of the 1-Steiner tree problem can be found efficiently.
- The **repeated 1-Steiner tree heuristic** gives provably good results (but no optimal solution is guaranteed).

```
(set of struct vertex, set of struct edge)
  steiner(set of struct vertex P)
{
  set of struct vertex T;
  set of struct edge E, F;
  int gain;

  E ← prim(P);
  (T, F, gain) ← 1-steiner(P, E);
  while (gain > 0) {
      P ← T;
      E ← F;
      (T, F, gain) ← 1-steiner(P, E);
  }
  return (P, E);
}
```

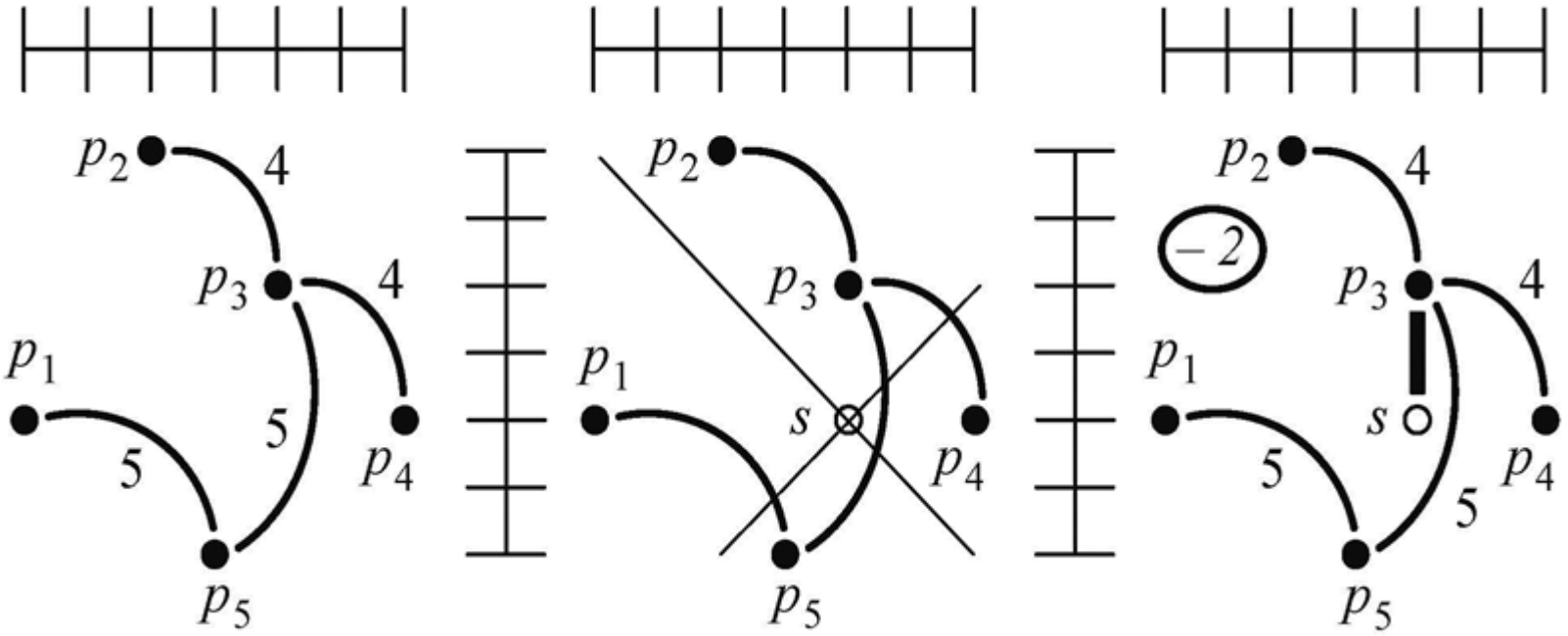
1-Steiner: Try All Hanan Points

```
(set of struct vertex, set of struct edge, int)
  1-steiner(set of struct vertex V, set of struct edge E)

  maxgain ← 0;
  for each s ∈ "Hannan points of V" {
    (W, F, gain) ← spanning_update(V, E, s);
    if (gain > maxgain) {
      maxgain ← gain;
      maxpoint ← s;
    }
  }
  if (maxgain > 0) {
    (W, F, gain) ← spanning_update(V, E, s);
    return (W, F, maxgain);
  }
  else return (V, E, 0);
}
```

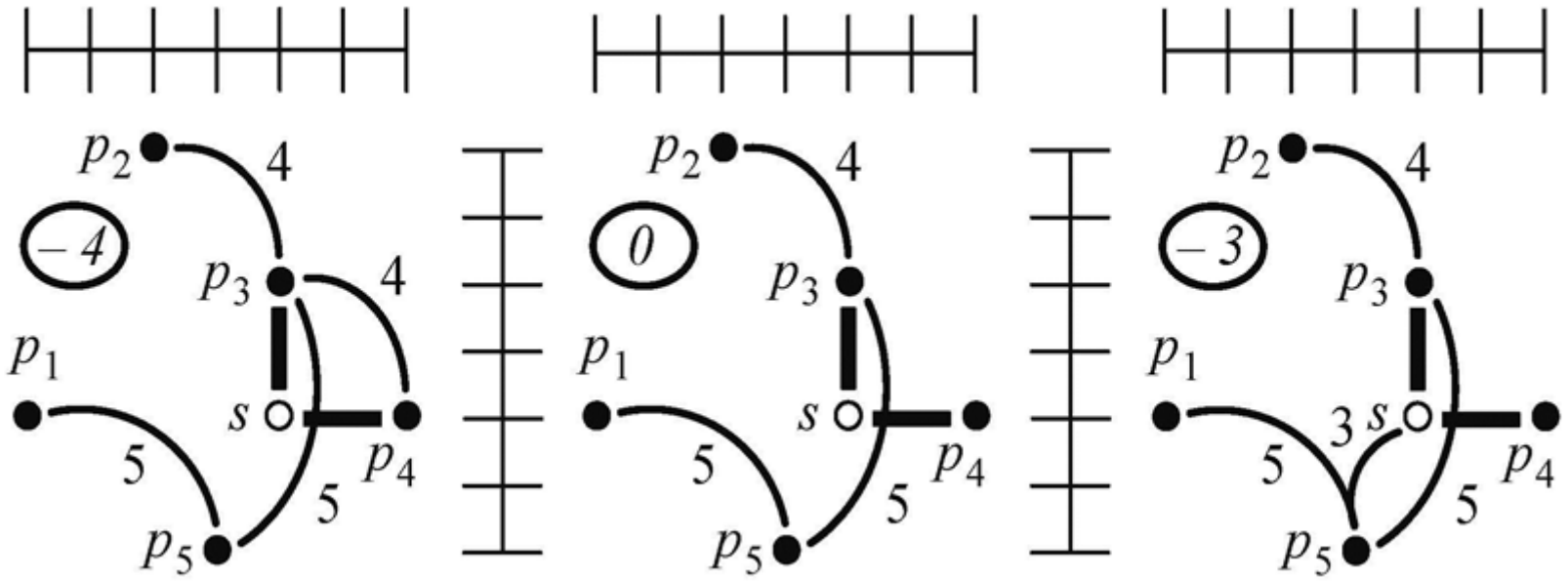
Getting Spanning Tree Incrementally

```
(set of struct vertex, set of struct edge, int)
  spanning_update(set of struct vertex V, set of struct edge E, struct vertex s)

  delta ← 0;
  V ← V ∪ {s};
  for each d ∈ {north, east, south, west} {
    u ← closest_point(V, s, d);
    delta ← delta - distance(s, u);
    E ← E ∪ {(s, u)};
    if (cycle(V, E)) {
      (v, w) ← largest_cycle_segment(V, E);
      E ← E \ {(v, w)};
      delta ← delta + distance(v, w);
    }
  }
  return (V, E, delta);
}
```

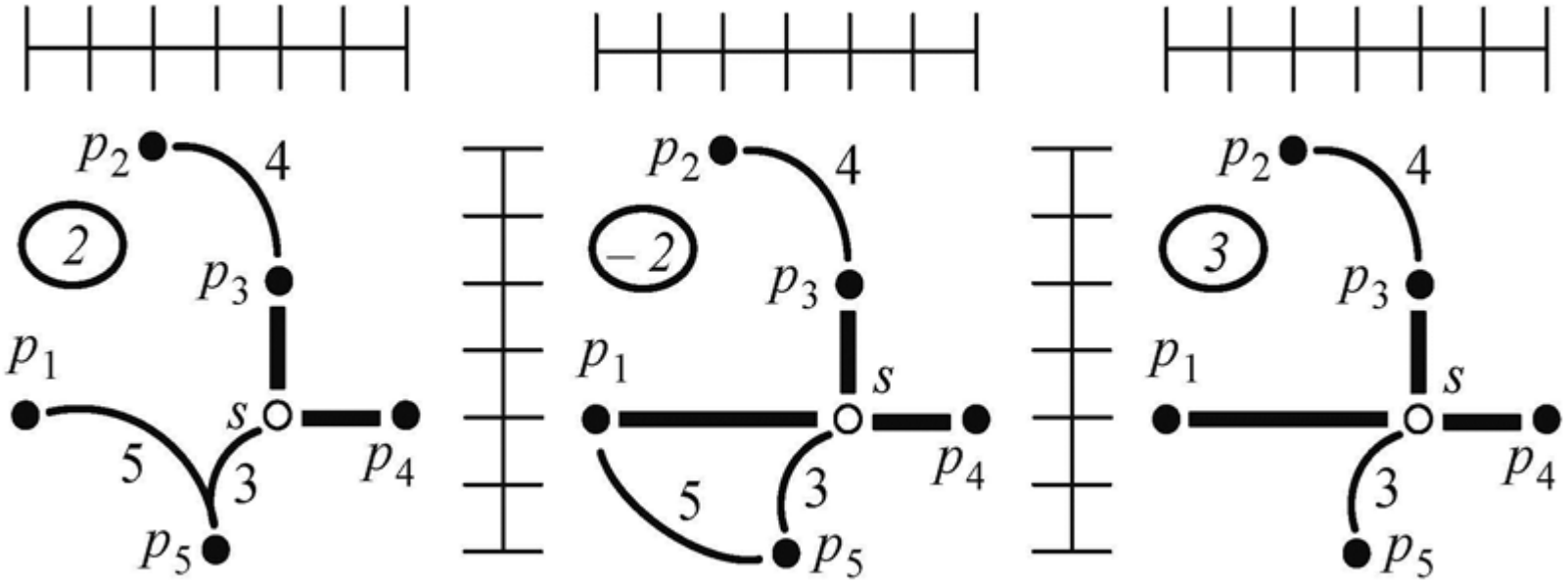
Example



Example (cont'd)



Example (cont'd)



FLUTE: Fast LookUp Table Estimation

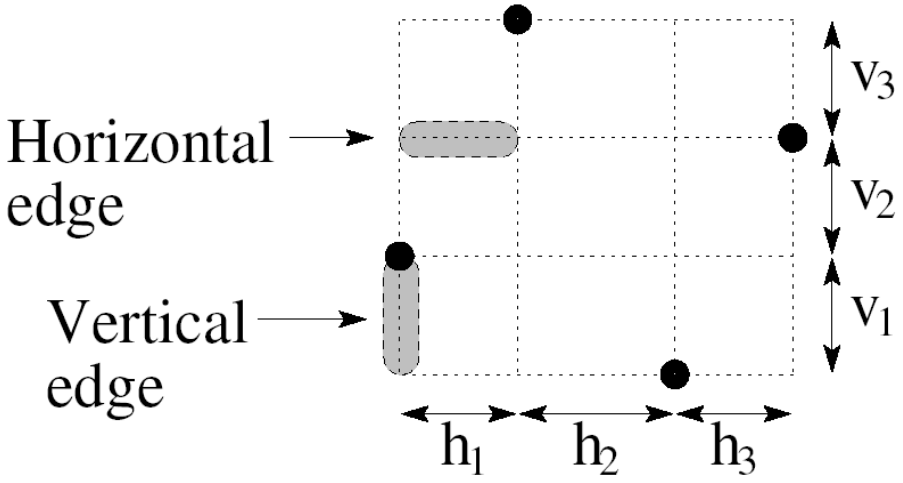
- Related publications
  - C. Chu, “FLUTE: Fast Lookup Table Based Wirelength Estimation Technique”, ICCAD 2004. (FLUTE 1.0)
  - C. Chu and Y.-C. Wong, “Fast and Accurate Rectilinear Steiner Minimal Tree Algorithm for VLSI Design”, ISPD 2005. (FLUTE 2.0)
  - C. Chu and Y.-C. Wong, “FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design”, TCAD 2008. (FLUTE 2.5)
  - Y.-C. Wong and Chris Chu, “A Scalable and Accurate Rectilinear Steiner Minimal Tree Algorithm”, VLSI-DAT 2008. (FLUTE 3.0)

Overview

- Basic idea
  - Lookup table to handle nets with a few pins
  - Net breaking technique to recursively break large nets
- Low degree nets are handled extremely well
  - Optimal and extremely efficient for nets up to 9 pins
  - Still very accurate and fast for nets up to 100 pins
- Suitable for VLSI applications
  - Over all 1.57 million nets in 18 IBM circuits [ISPD 98]
    - More accurate than Batched 1-Steiner heuristic
    - Almost as fast as minimum spanning tree construction

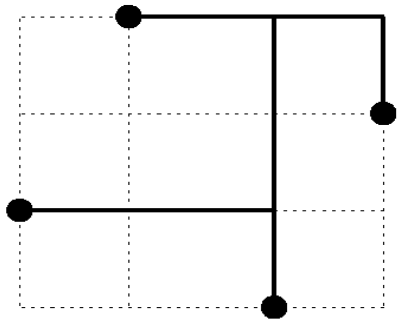
Preliminaries

- Consider routing along Hanan grid
- **Observation:** An optimal RSMT can always be broken down into a set of horizontal edges and vertical edges
- Define edge lengths h_i and v_i



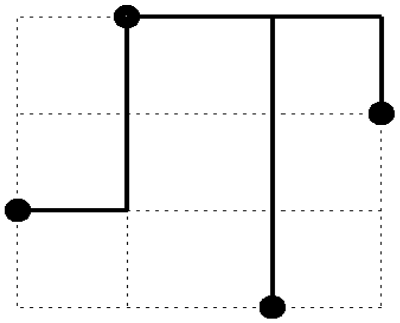
Wirelength Vector (WV)

- **Observation:** WL can be written as a linear combination of edge lengths with positive integral coefficients



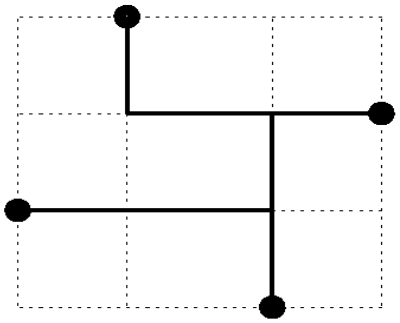
$$WL = h_1 + 2h_2 + h_3 + v_1 + v_2 + 2v_3$$

(1, 2, 1, 1, 1, 2)



$$WL = h_1 + h_2 + h_3 + v_1 + 2v_2 + 3v_3$$

(1, 1, 1, 1, 2, 3)



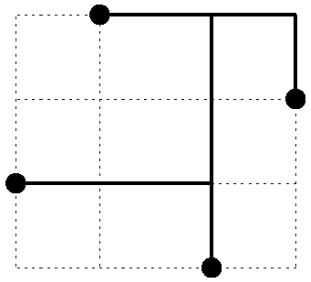
$$WL = h_1 + 2h_2 + h_3 + v_1 + v_2 + v_3$$

(1, 2, 1, 1, 1, 1)

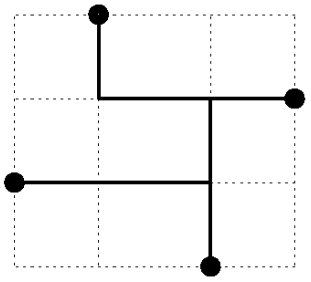
- WL can be expressed as a vector of the coefficients called Wirelength Vector (WV)

Potentially Optimal WV (POWV)

- Optimal WL can be found by enumerating all WVs
- However, most WVs can never produce optimal WL
  - (1, 2, 1, 1, 1, $\underline{2}$) is redundant as it always produces a larger WL than (1, 2, 1, 1, 1, $\underline{1}$)



(1, 2, 1, 1, 1, 2)



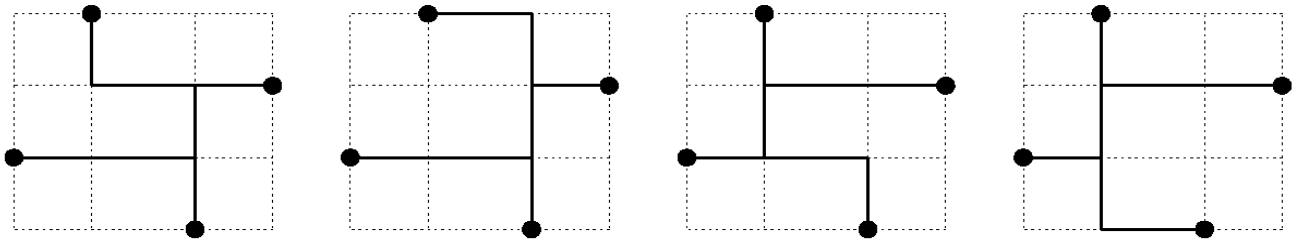
(1, 2, 1, 1, 1, 1)

- Potentially Optimal Wirelength Vector (POWV) is a WV that *may* produce optimal WL

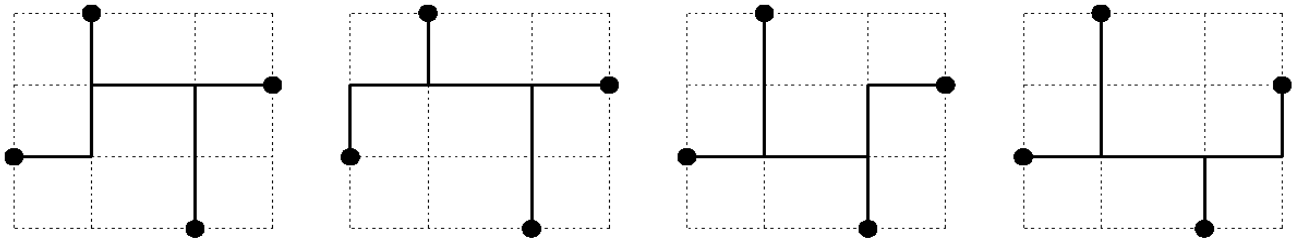
Number of POWVs

- For any net
  - # of possible routing solutions is huge
  - # of WVs is much less
  - **# of POWVs is very small**
- For example, only 2 POWVs for the net below

POWV (1,2,1,1,1,1)

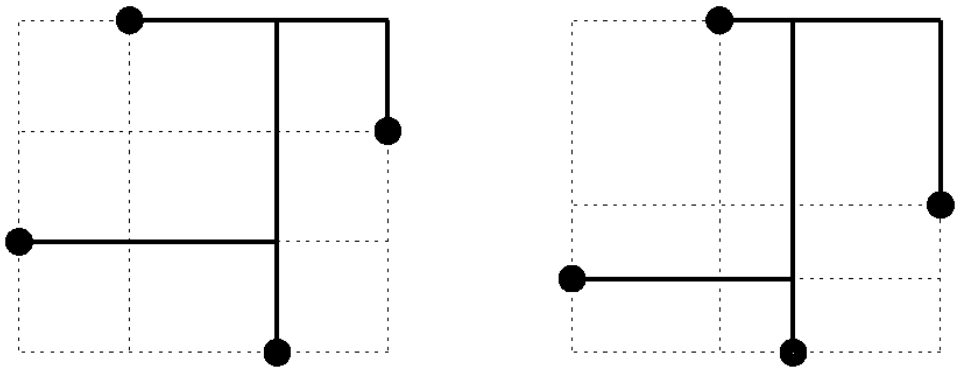


POWV (1,1,1,1,2,1)



Sharing of POWVs Among Nets

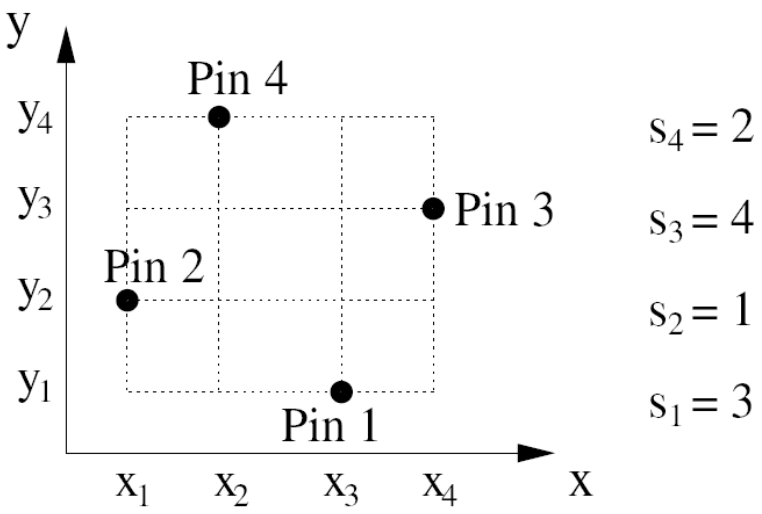
- To find optimal WL, we can pre-compute all POWVs and store them in a lookup table
- However, there are infinite number of different nets
- Try to group together nets that can share the same set of POWVs
- For example, these following two nets share the same set of POWVs:



Grouping by Position Sequence

- Define position sequence $s_1 s_2 \ldots s_n$ to be the list of ranks of pins in x-coordinate

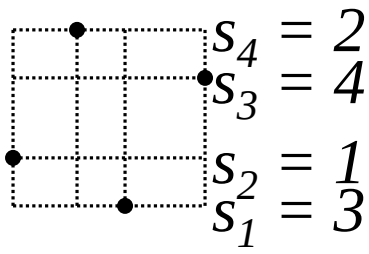
Position sequence
= 3142



- **Lemma:** The set of all degree-n nets can be divided into n! groups according to the position sequence such that all nets in each group share the same set of POWVs

Steps for WL Estimation

- Given a net
  1. Find the position sequence

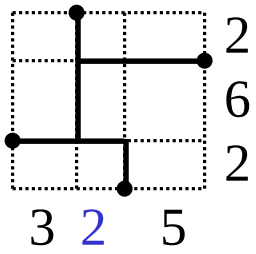


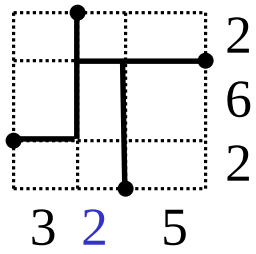
Position sequence: *3142*

  2. Get the POWVs from LUT

POWVs:
(1,2,1,1,1,1) (1,1,1,1,2,1)

  3. Find the edge lengths





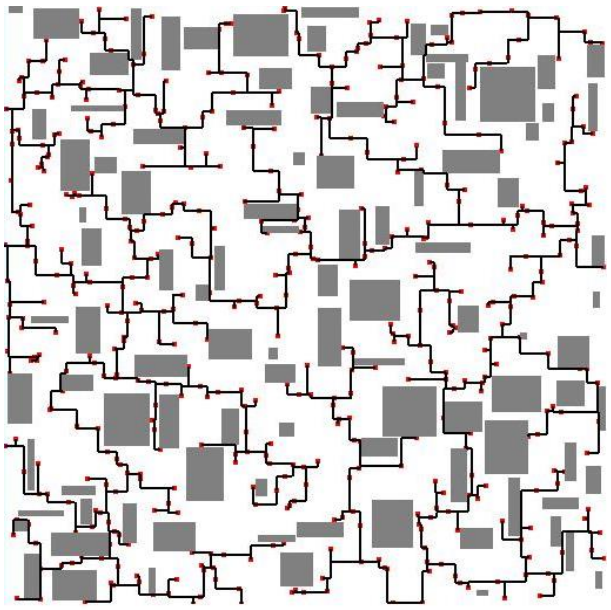
  4. Find WL for each POWV and return the best

HPWL + 2 = 22 HPWL + 6 = 26

Return

Obstacle-Avoiding Rectilinear Steiner Minimal Tree

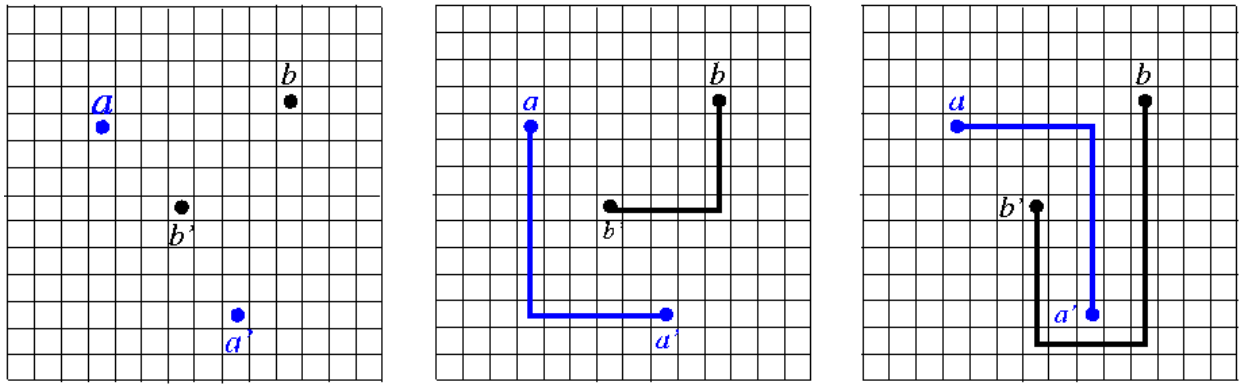
- Obstacles: macro/IP blocks, power/ground network, etc.
  - Rectangular shapes vs. rectilinear shapes
- Single routing layer



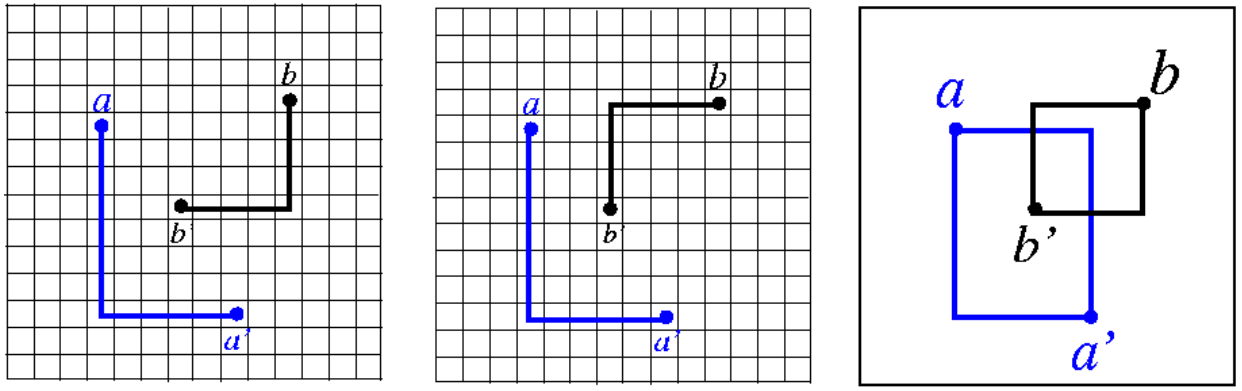
- Multiple routing layers
  - Preferred directions vs. non-preferred directions

Net Ordering

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



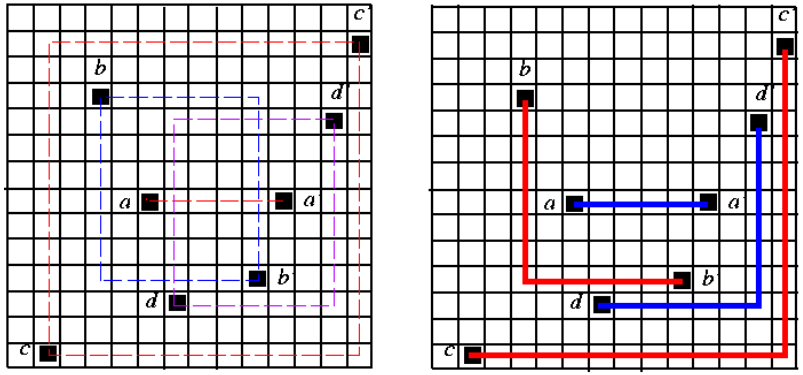
route net a before net b



route net b before net a

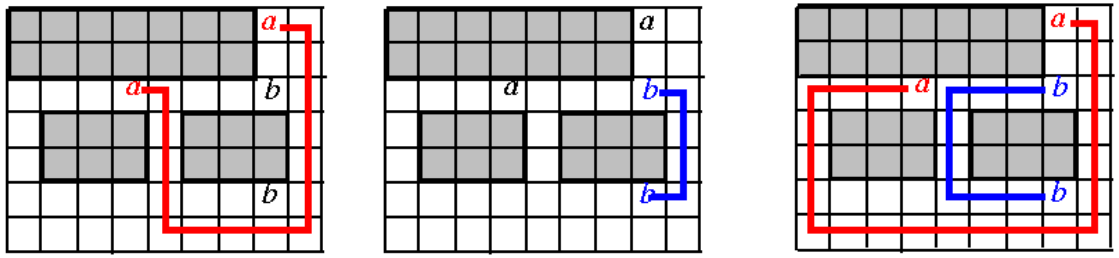
Net Ordering (cont'd)

- Order the nets in the ascending order of the # of pins within their bounding boxes.



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

- Order the nets in the ascending (or descending??) order of their length.
- Order the nets based on their timing criticality.
- A mutually intervening case:



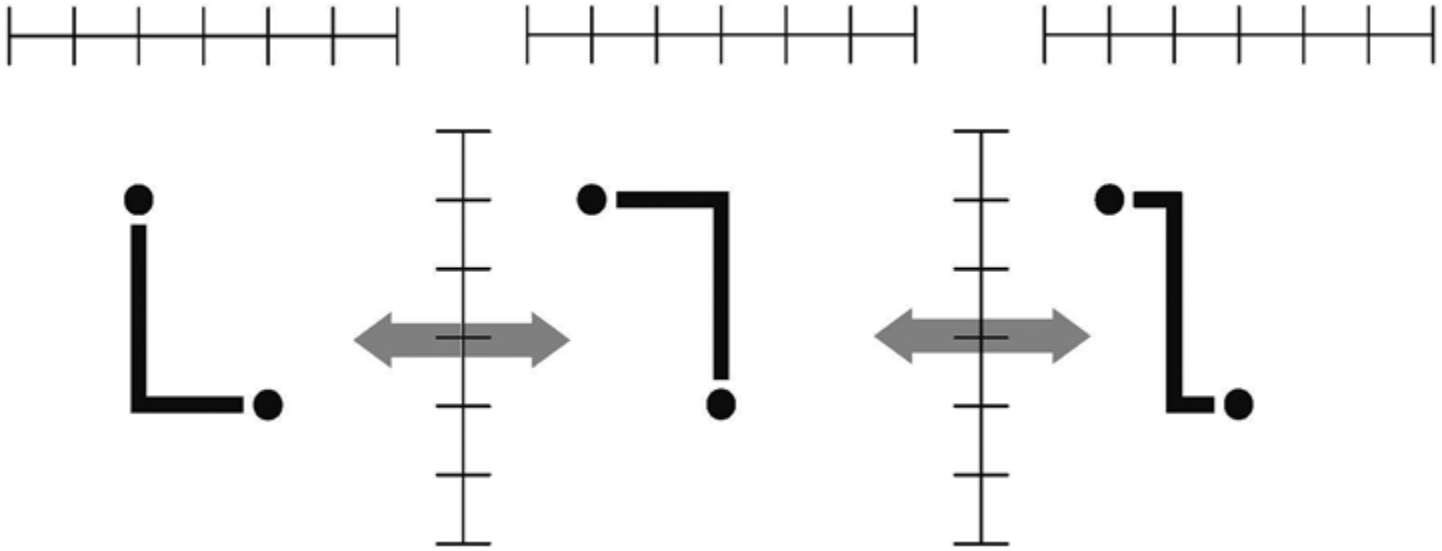
a prevents routing of b *b prevents routing of a* *a feasible routing*

Rip-Up and Re-routing

- Rip-up and re-routing is required if a global or detailed router fails in routing all nets.
- 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.

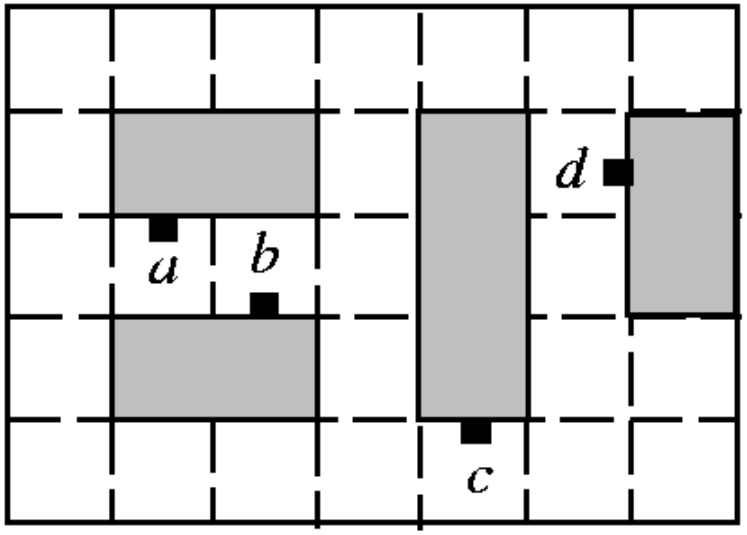
Local Transformations

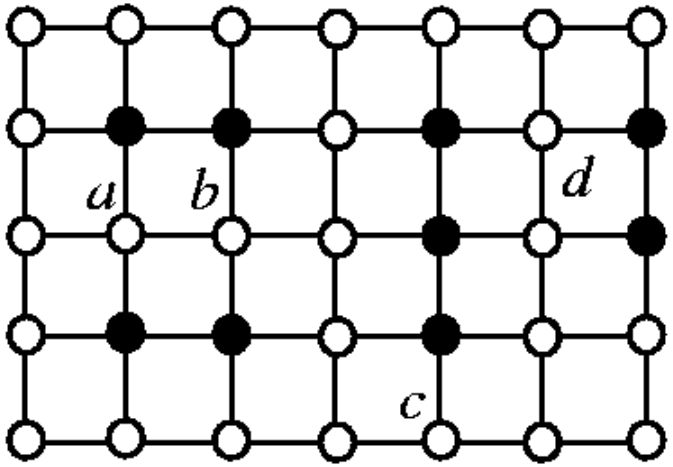
- Over-congestion after net-independent Steiner-tree construction can be eliminated by local transformations (e.g. guided by simulated annealing) or by ripping up a net and rerouting it by maze routing



Graph Models for Global Routing: Grid Graph

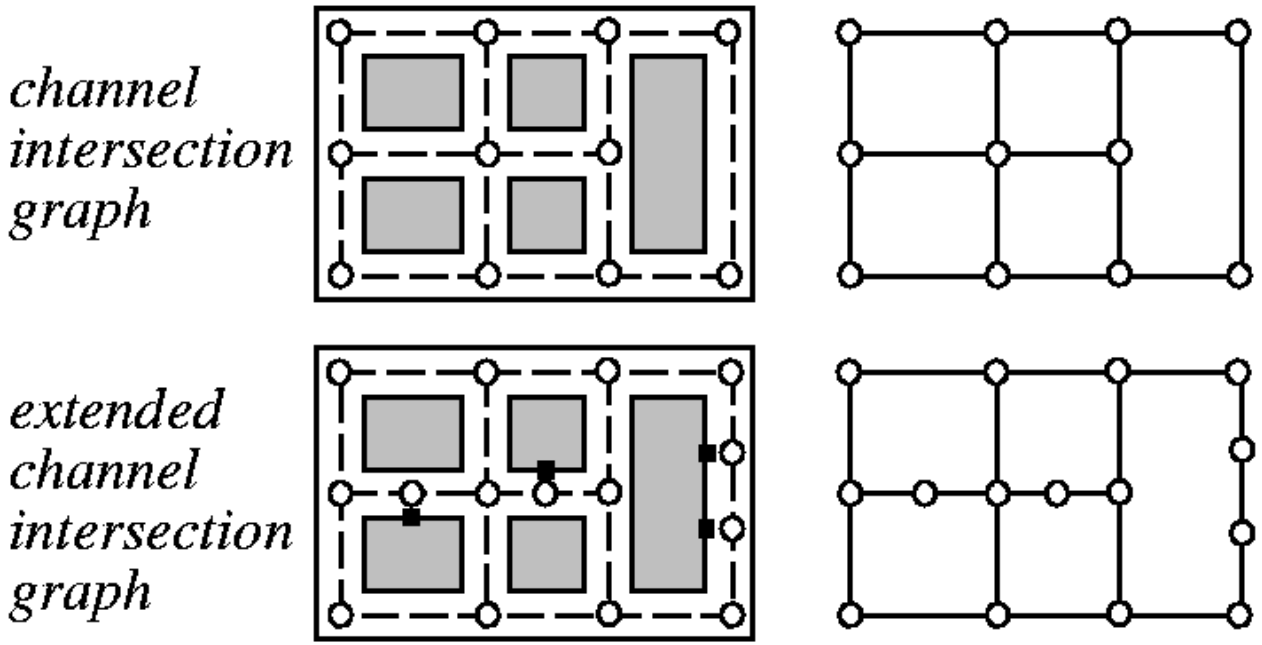
- Each grid cell is represented by a vertex.
- Two vertices are joined by an edge if the corresponding grid cells are adjacent to each other.
- The occupied grid cells are represented as filled circles, whereas the others are as clear circles.





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.



Global Routing Problem

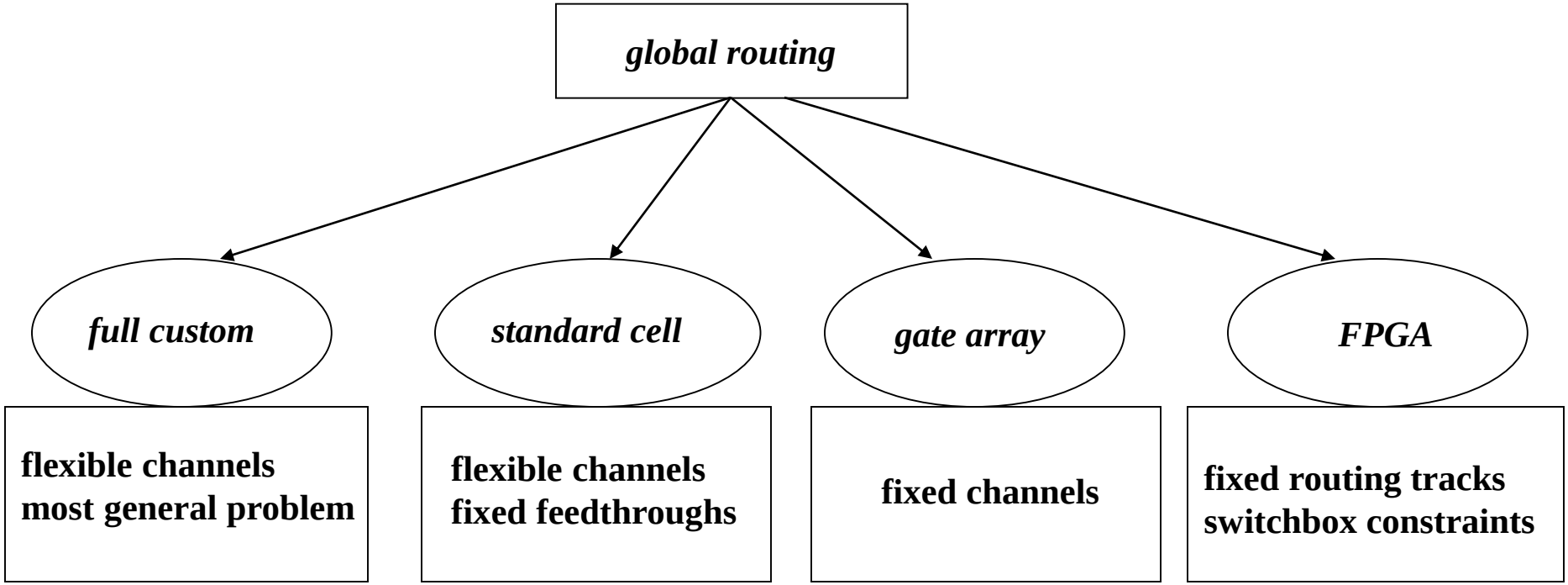
- Given a netlist $N=\{N_1,N_2,\ldots,N_n\}$, a routing graph $G=(V,E)$, find a Steiner tree T_i for each net N_i, $1 \le i \le n$, such that $U(e_j) \le c(e_j)$, $\forall e_j \in E$ and $\sum_{i=1}^{n} L(T_i)$ is minimized,

  where

  - $c(e_j)$: capacity of edge e_j;
  - $x_{ij}=1$ if e_j is in T_i; $x_{ij}=0$ otherwise;
  - $U(e_j)= \sum_{i=1}^{n} x_{ij}$: # of wires that pass through the channel corresponding to edge e_j;
  - $L(T_i)$: total wirelength of Steiner tree T_i.

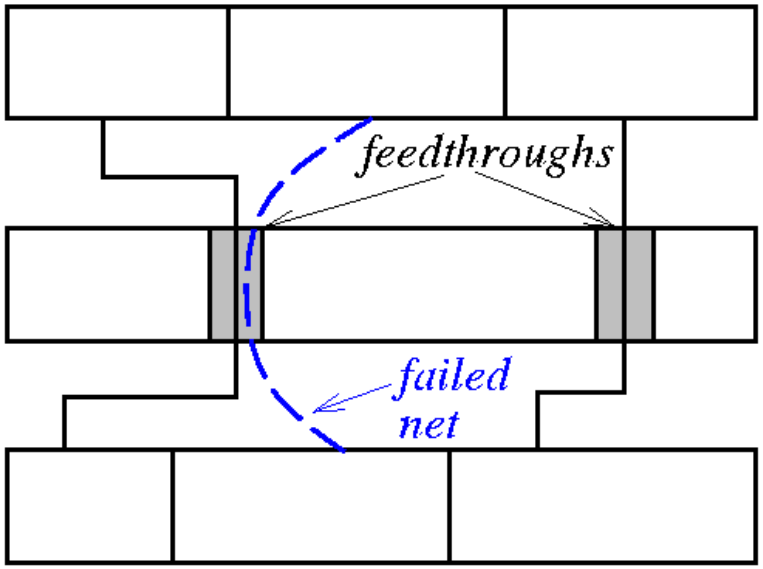
- For high-performance, the maximum wirelength $(\max_{i=1}^{n} L(T_i))$ is minimized (or the longest path between two points in T_i is minimized).

Global Routing in Different Design Styles



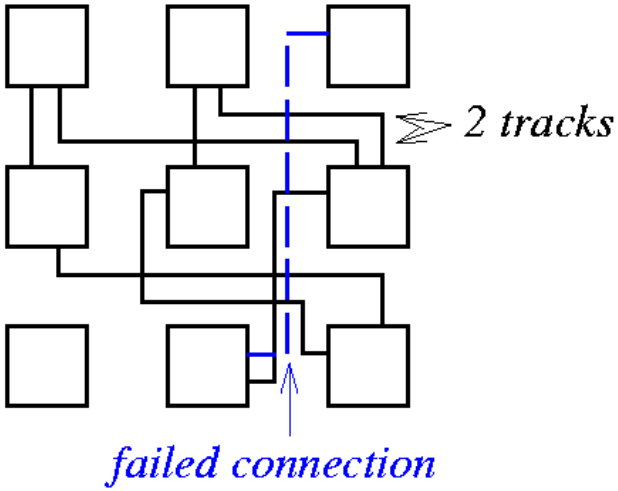
Global Routing in Standard Cell

- Objective
  - Minimize total channel height.
  - Assignment of **feedthroughs**.
- For high performance,
  - Minimize the maximum wire length.
  - Minimize the maximum path length.



Global Routing in Gate Array

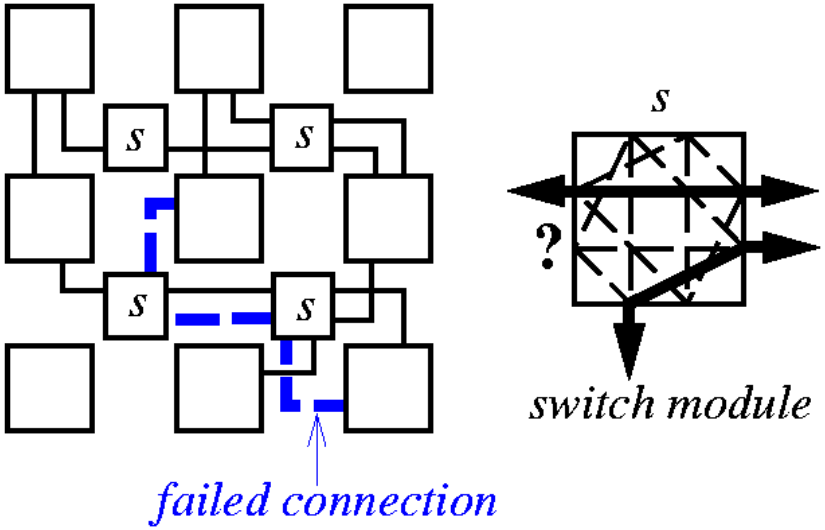
- Objective
  - **Guarantee 100% routability**.
- For high performance,
  - Minimize the maximum wire length.
  - Minimize the maximum path length.



Each channel has a capacity of 2 tracks.

Global Routing in FPGA

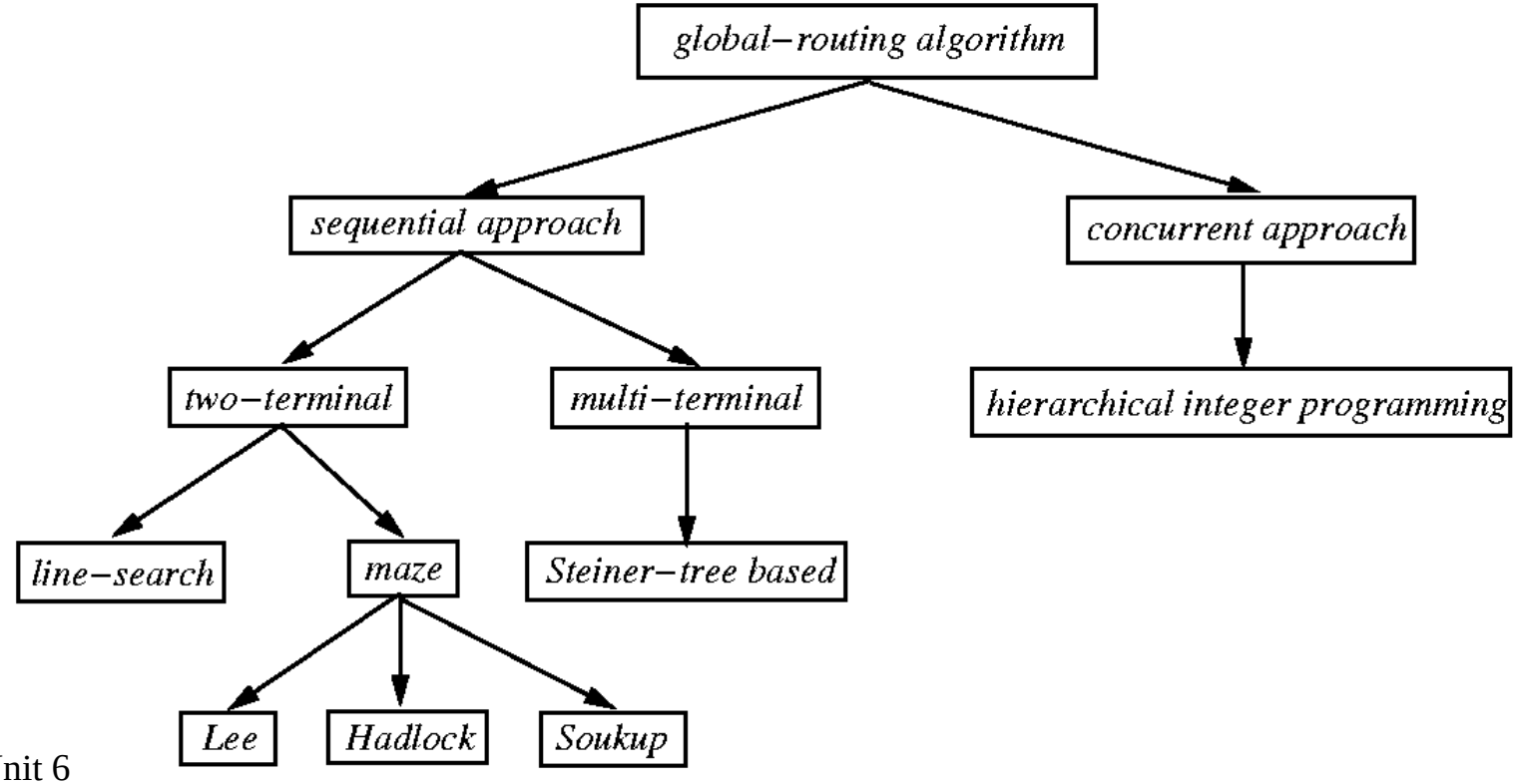
- Objective
  - Guarantee 100% routability.
  - Consider **switch-module architectural constraints**.
- For performance-driven routing,
  - **Minimize # of switches used**.
  - Minimize the maximum wire length.
  - Minimize the maximum path length.



Each channel has a capacity of 2 tracks.

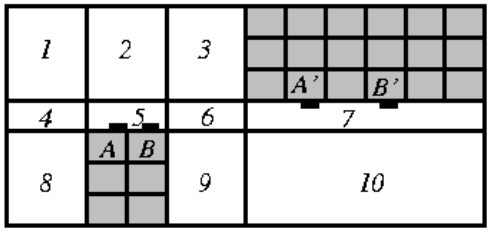
Classification of Global Routing Algorithm

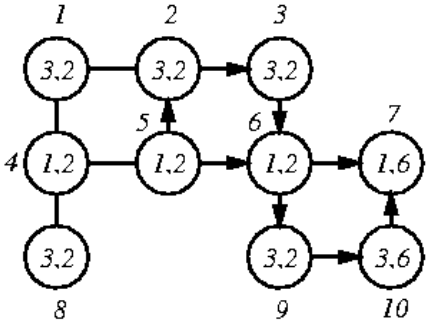
- **Sequential approach**: Assigns priority to nets; routes one net at a time based on its priority (net ordering?).
- **Concurrent approach**: All nets are considered at the same time (complexity?)

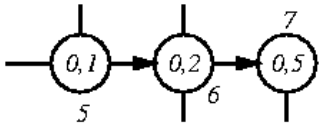


Global Routing: Maze Routing

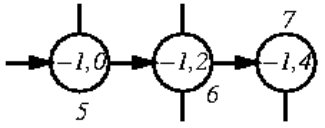
- Routing channels may be modeled by a weighted undirected graph called **channel connectivity graph**.
- Node $\leftrightarrow$ channel; edge $\leftrightarrow$ two adjacent channels; capacity: (*width*, *length*)



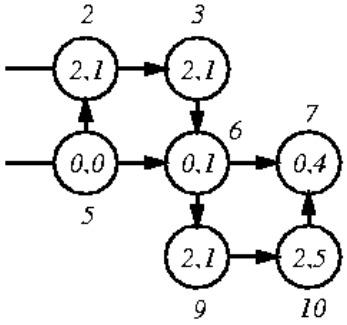




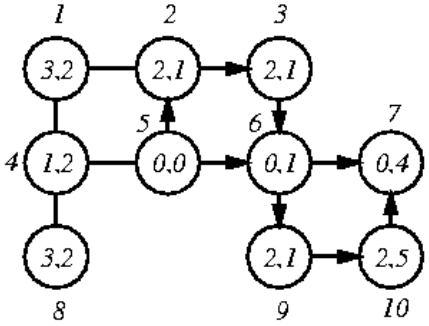
route A–A' via 5–6–7



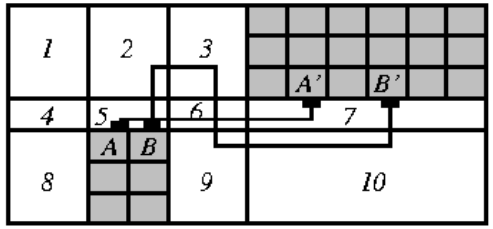
route B–B' via 5–6–7



route B–B' via 5–2–3–6–9–10–7



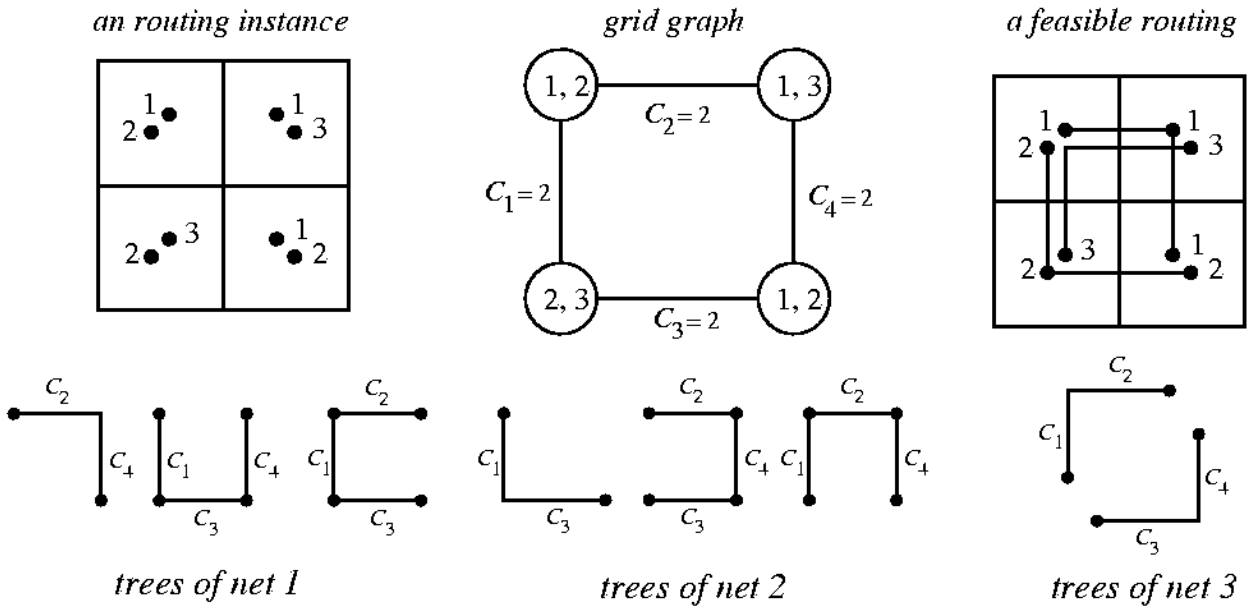
updated channel graph



maze routing for nets A and B

Global Routing by Integer Programming

- Suppose that for each net *i*, there are n_i possible trees $t_1^i, t_2^i, \ldots, t_{n_i}^i$ 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

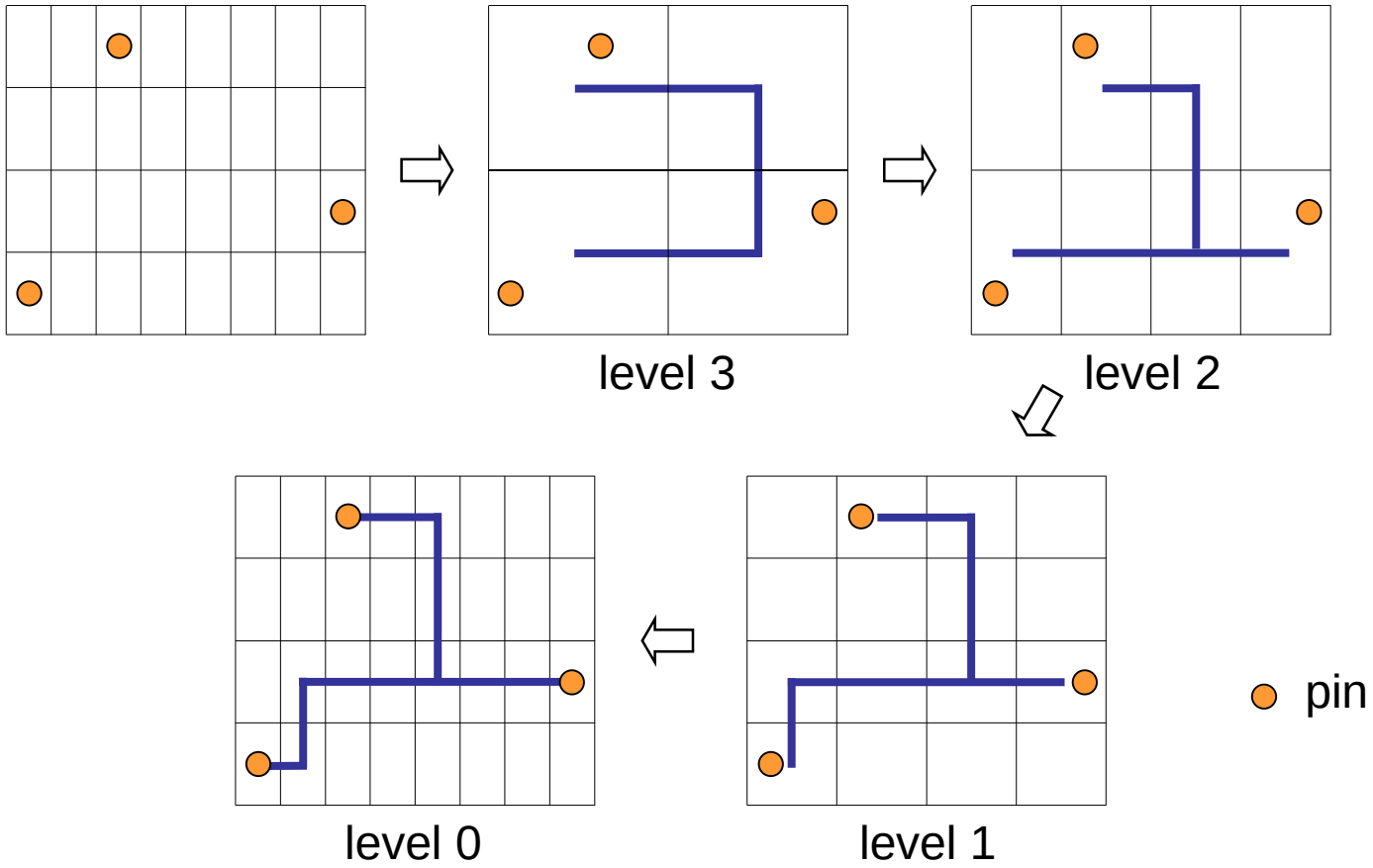
- $g_{i,j}$: cost of tree t_j^i ⇒ $g_{1,1} = 2$, $g_{1,2} = 3$, $g_{1,3} = 3$, $g_{2,1} = 2$, $g_{2,2} = 3$, $g_{2,3} = 3$, $g_{3,1} = 2$, $g_{3,2} = 2$.

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

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

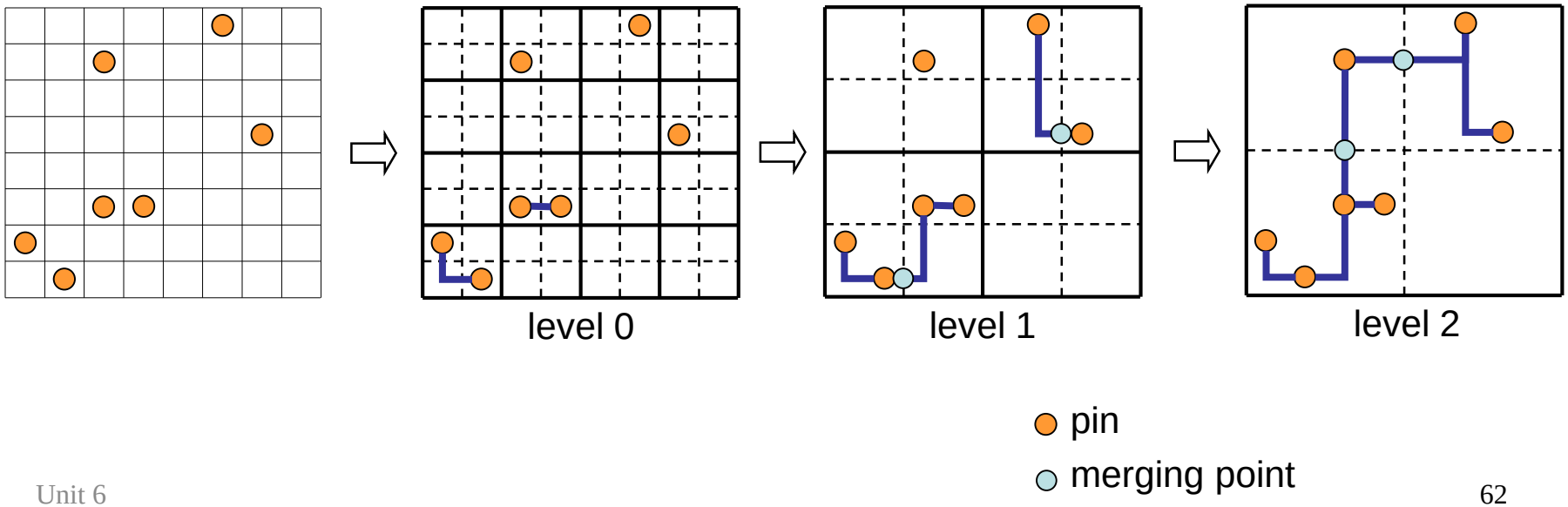
Top-down Hierarchical Global Routing

- Recursively divides routing regions into successively smaller **super cells**, and nets at each hierarchical level are routed sequentially or concurrently.



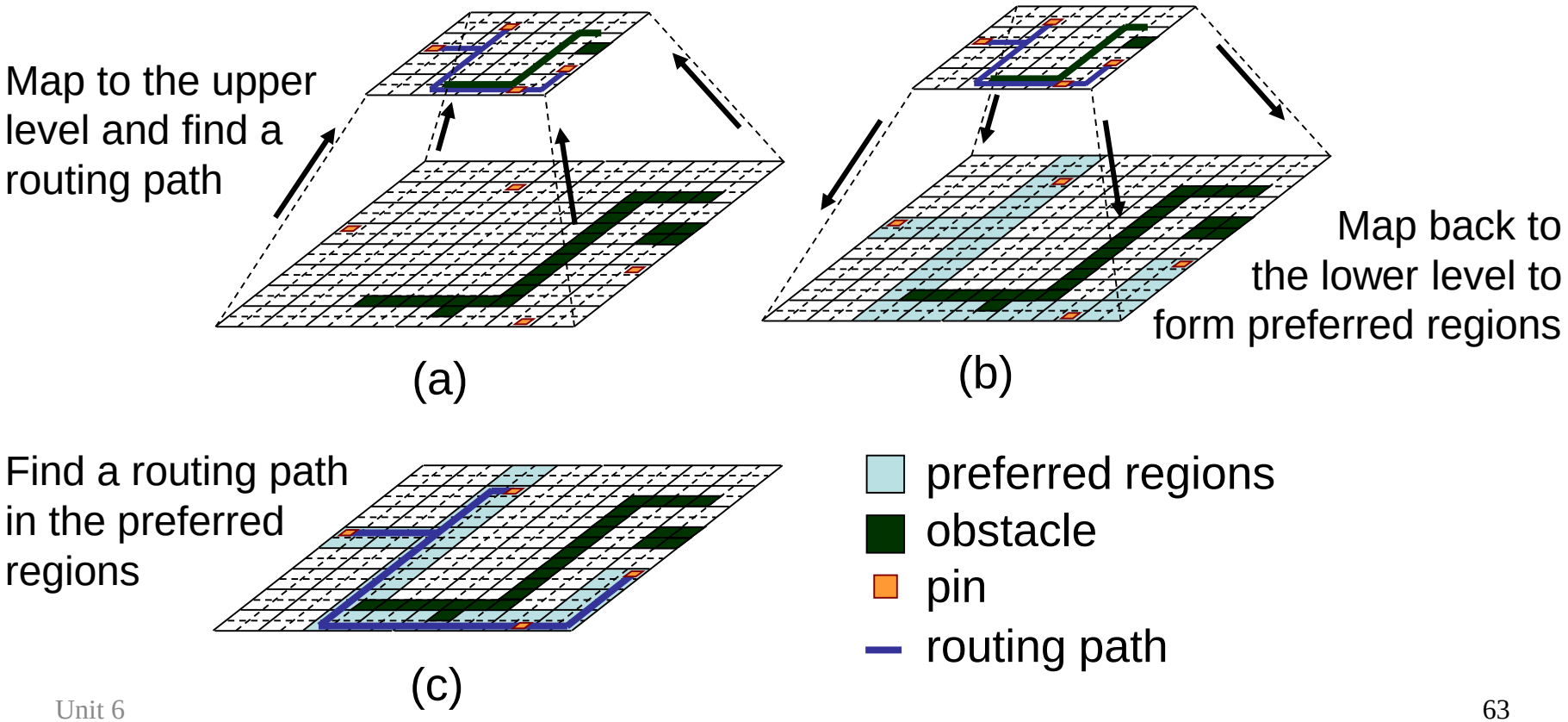
Bottom-up Hierarchical Global Routing

- At each hierarchical level, routing is restrained within each super cell individually.
- When the routing at the current level is finished, every four super cells are merged to form a new larger super cell at the next higher level.



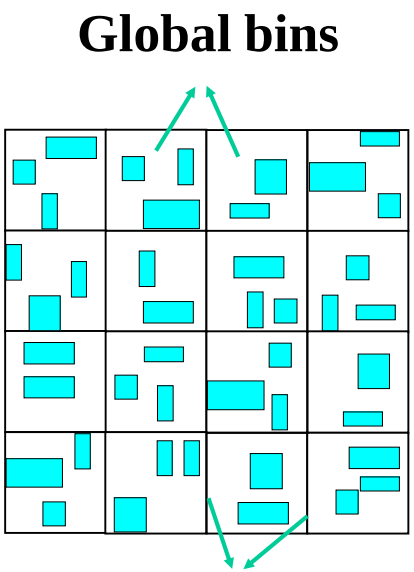
Hybrid Hierarchical Global Routing

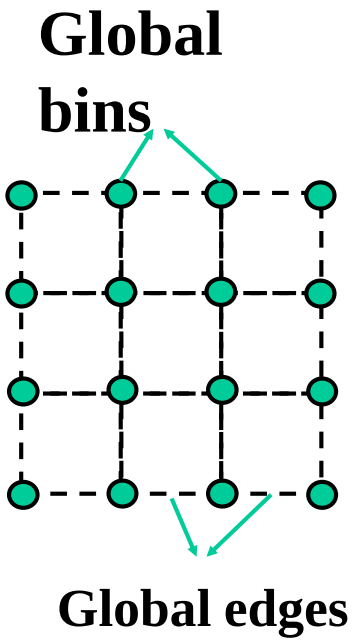
- (1) neighboring propagation, (2) preference partitioning, and (3) bounded routing.



Global Routing Problem Revisited

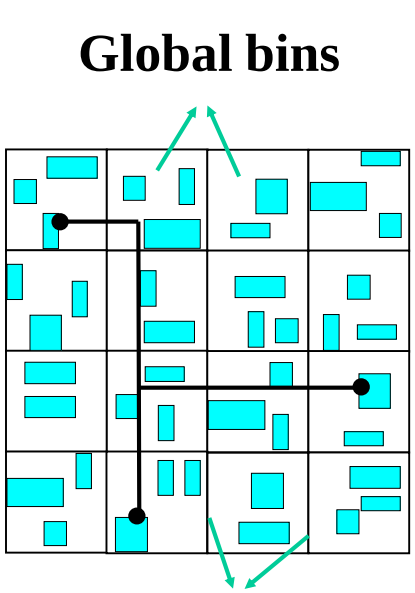
- supply $s(e)$: the number of available routing tracks passing through edge e
- demand $d(e)$: the number of wires that utilize edge e
- $overflow(e) = \begin{cases} d(e) - s(e) & if d(e) > s(e) \\ 0 & otherwise \end{cases}$

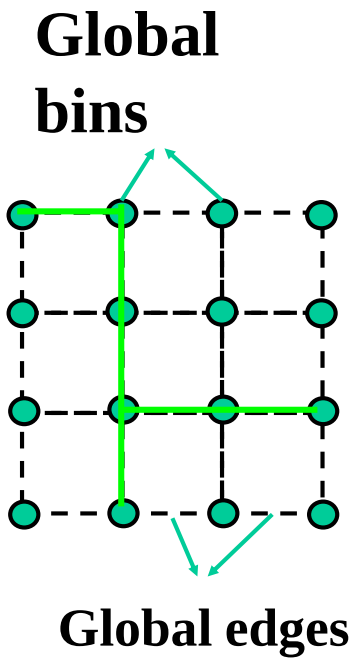




Global Routing Problem Revisited

- Input: a set of nets to be routed over a grid graph
- Output: Steiner tree topologies for all nets such that the total overflow, the maximum overflow and the total wirelength are minimized

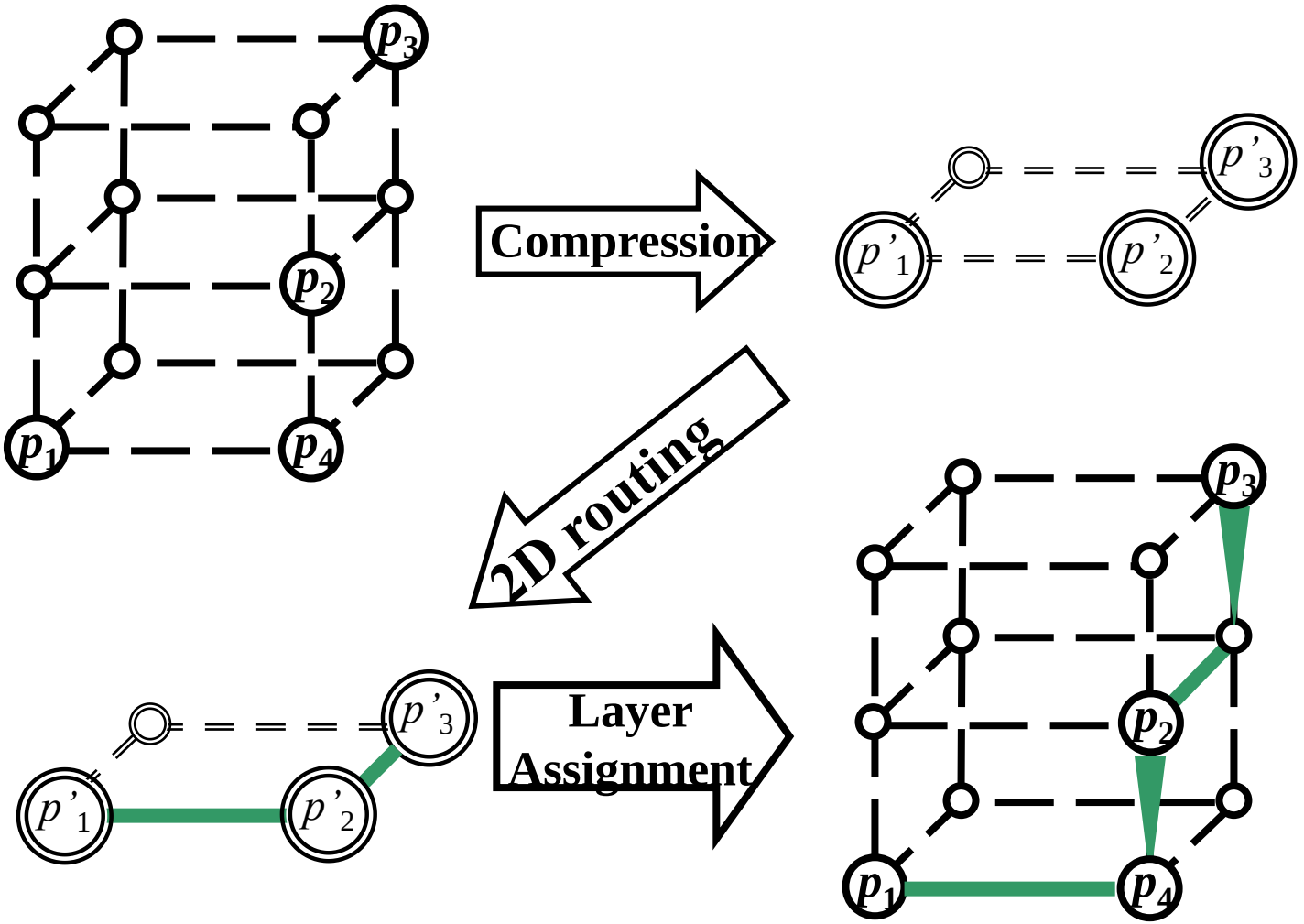




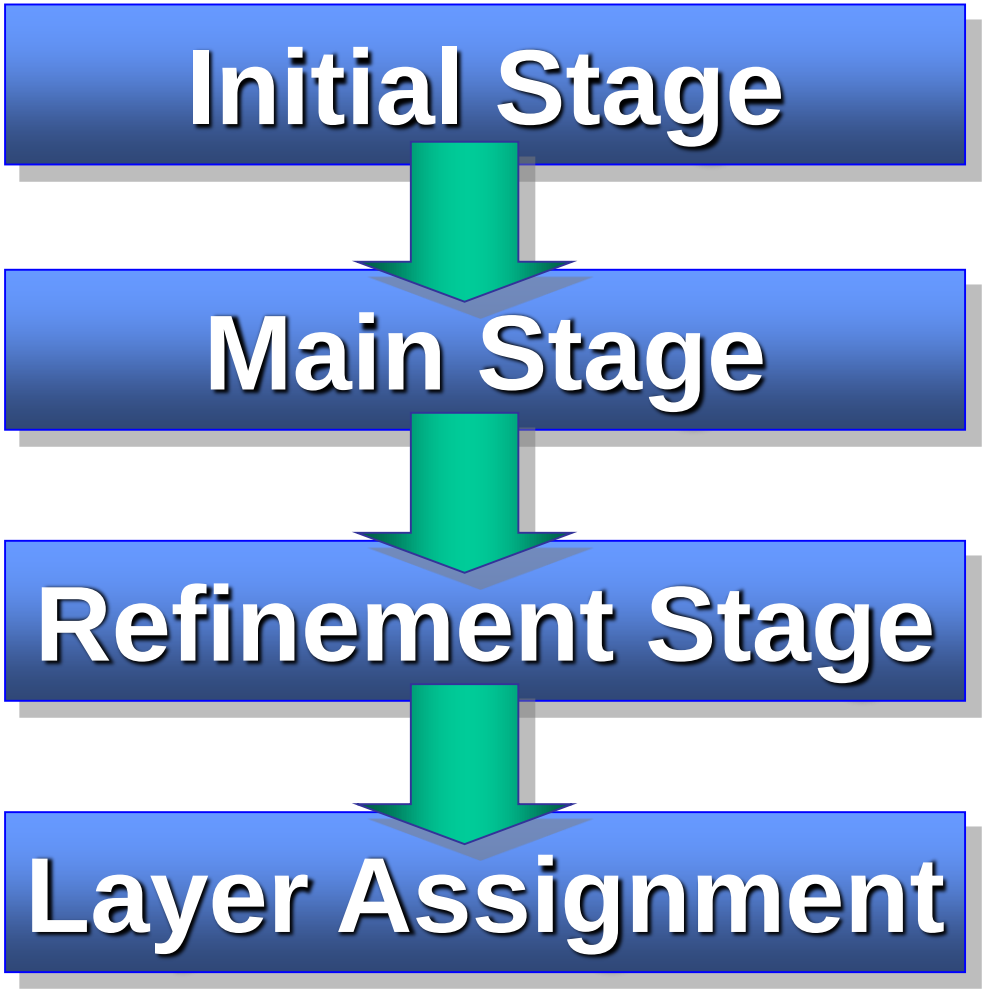
NTHU-Route 2.0

- Related publications:
  - Y.-J. Chang, Y.-T. Lee, J.-R. Gao, P.-C. Wu, and T.-C. Wang, “NTHU-Route 2.0: A Robust Global Router for Modern Designs,” IEEE TCAD 2010.
  - Y.-J. Chang, Y.-T. Lee and T.-C. Wang, “NTHU-Route 2.0: A Fast and Stable Global Router,” ICCAD 2008.
  - J.-R. Gao, P.-C. Wu and T.-C. Wang, “A New Global Router for Modern Designs,” ASP-DAC 2008. (NTHU-Route)
  - T.-H. Lee and T.-C. Wang, “Congestion-Constrained Layer Assignment for Via Minimization in Global Routing,” IEEE TCAD 2008.

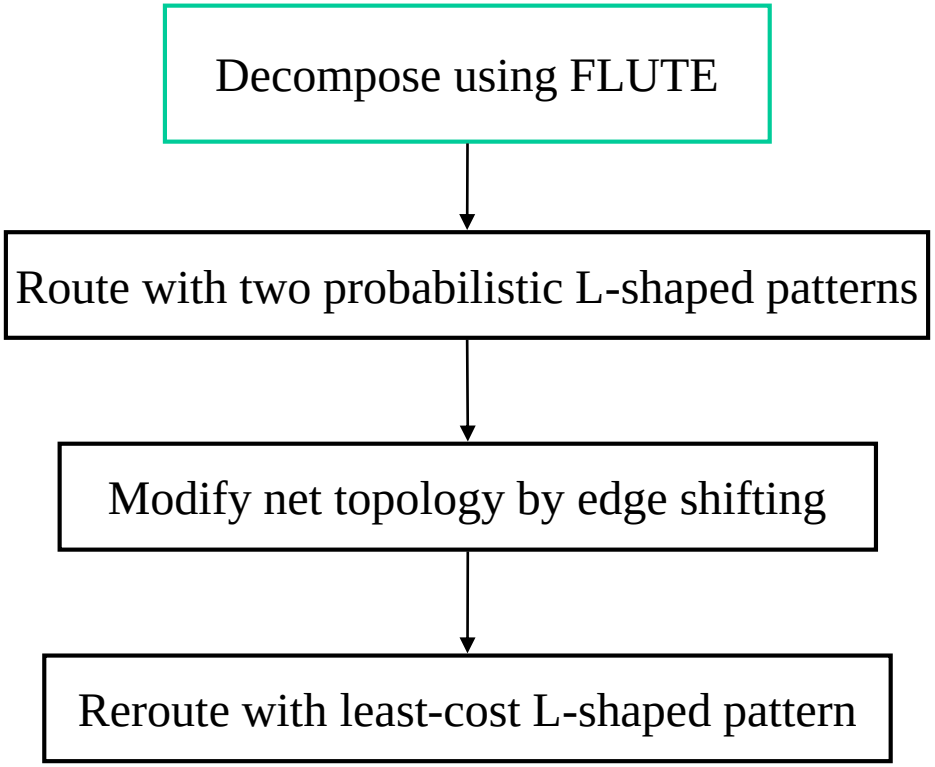
Multi-Layer Global Routing



Four Stages

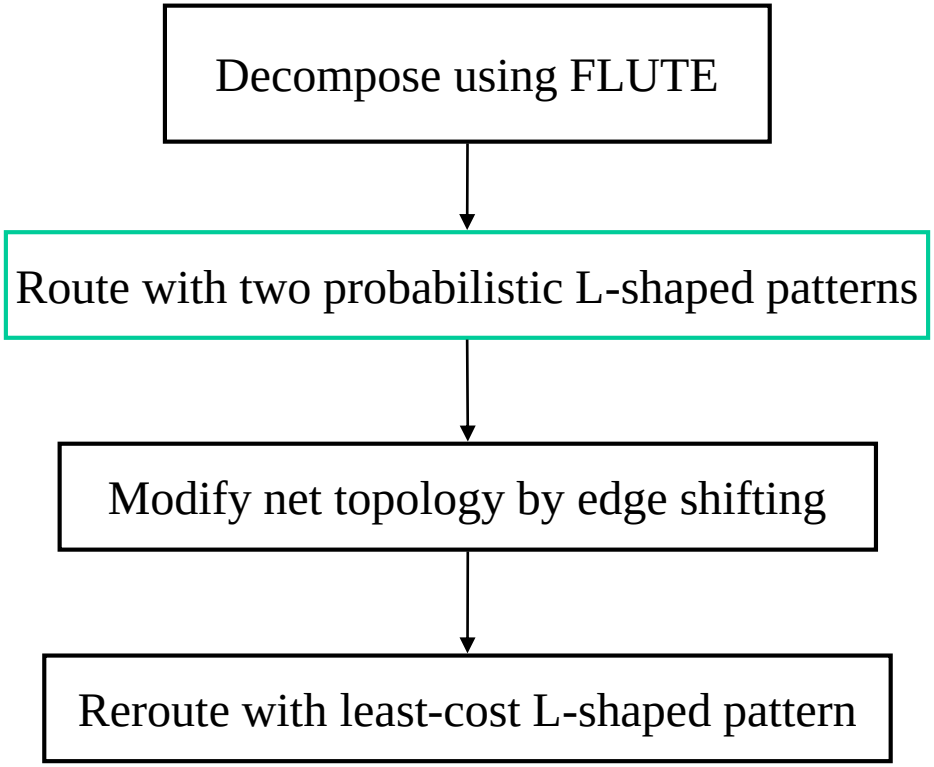

Initial Stage
Main Stage
Refinement Stage
Layer Assignment

Initial Stage

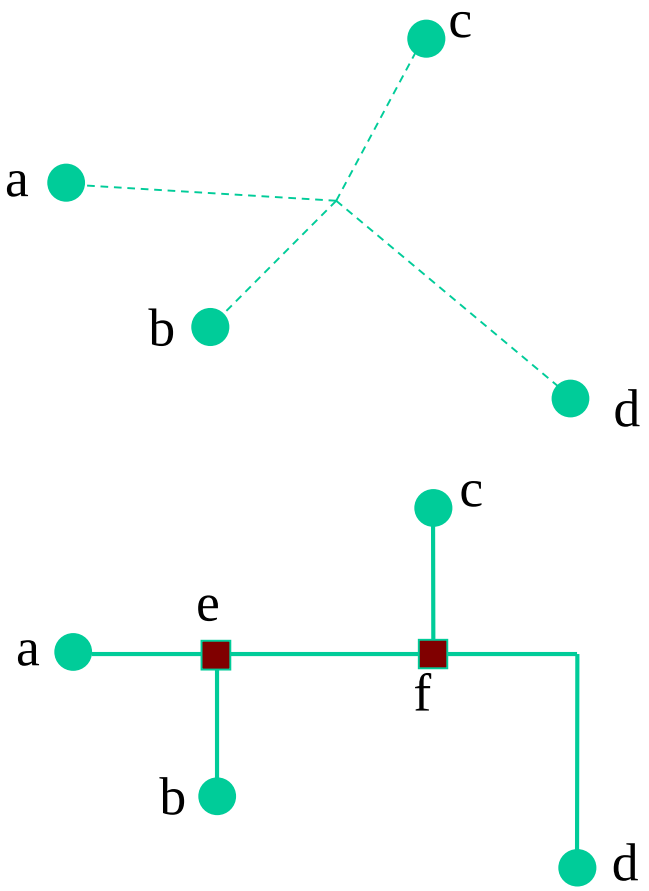


- First project the design onto a plane
- FLUTE [ISPD05]
  - Generation of a rectilinear Steiner minimal tree
  - A look-up table based method
  - Optimum for net with 9 or fewer pins

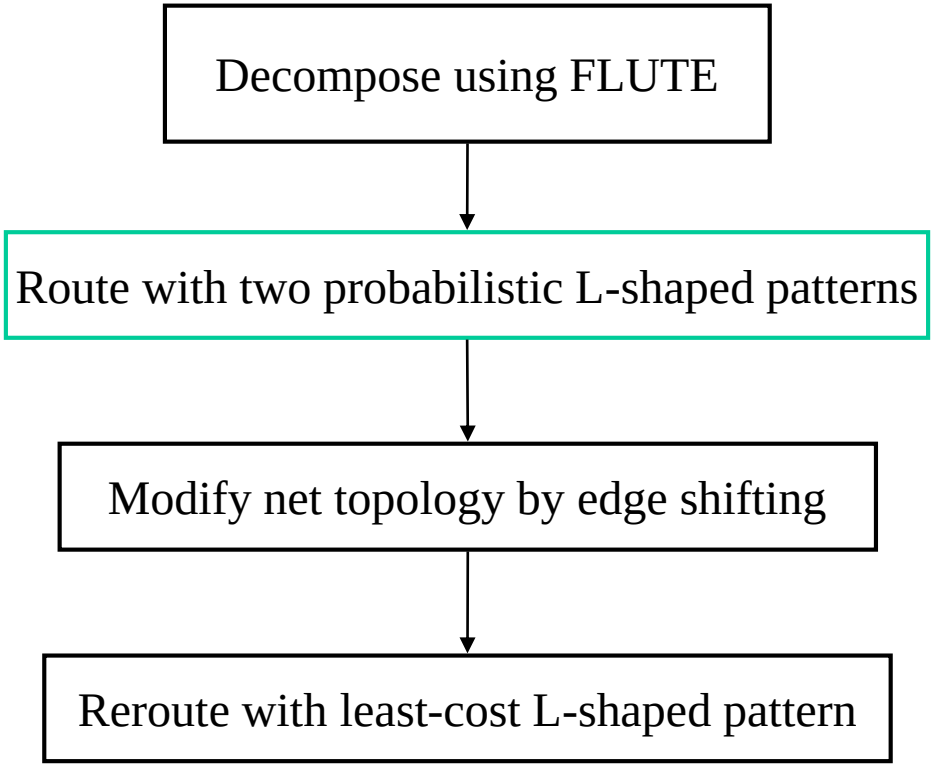
Initial Stage (cont'd)

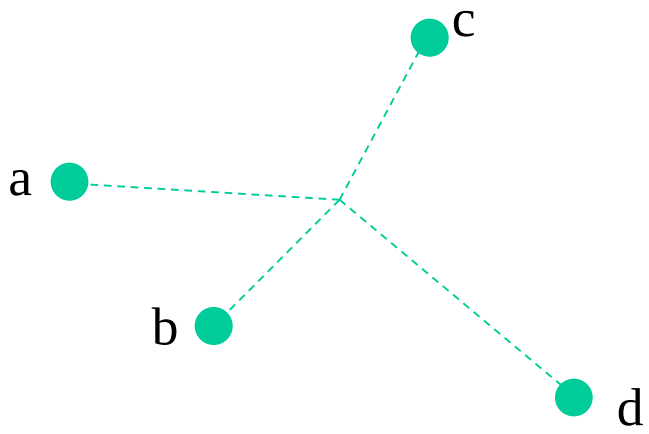


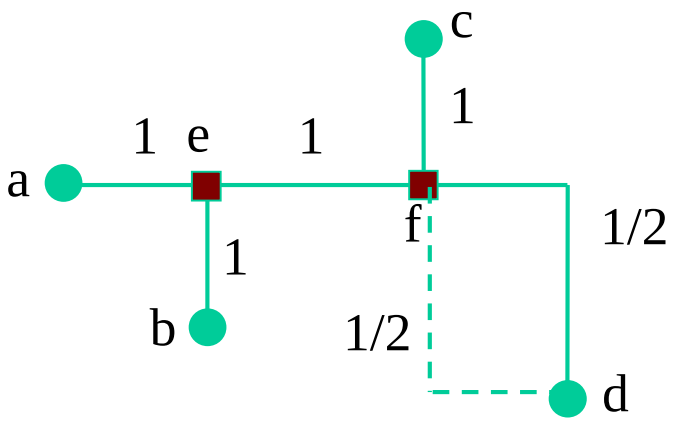
- Add ½ demand alone each edge of an L-shaped path



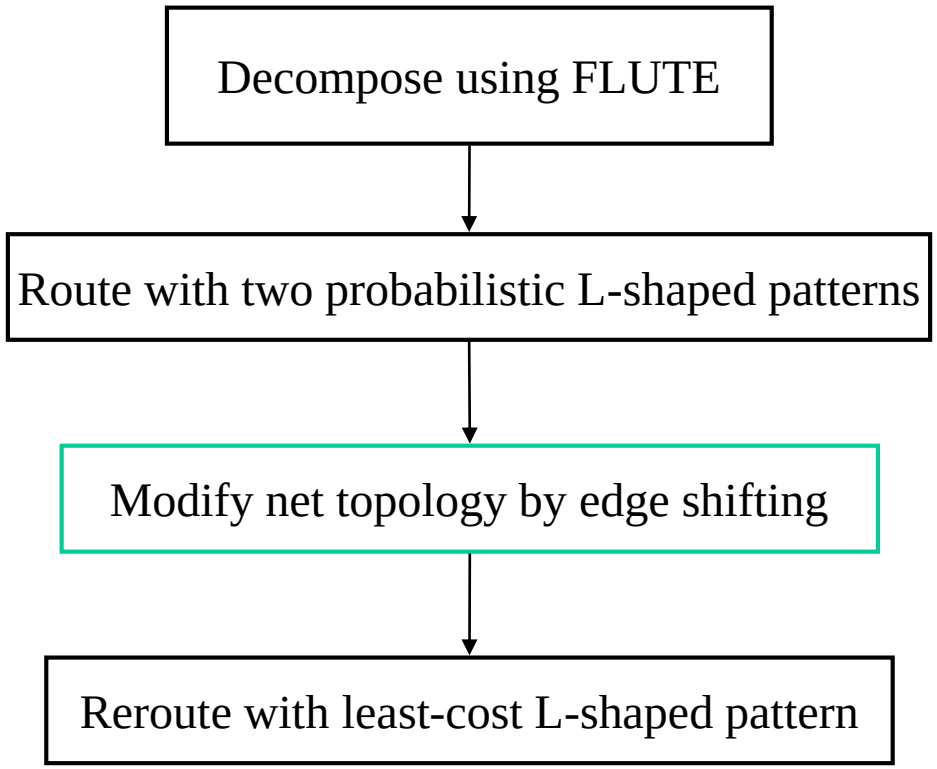
Initial Stage (cont'd)



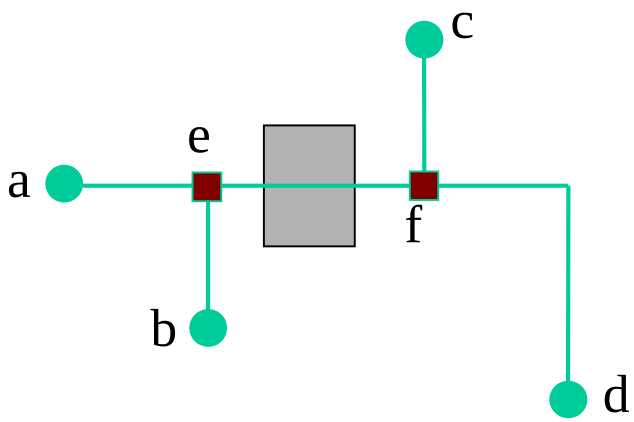




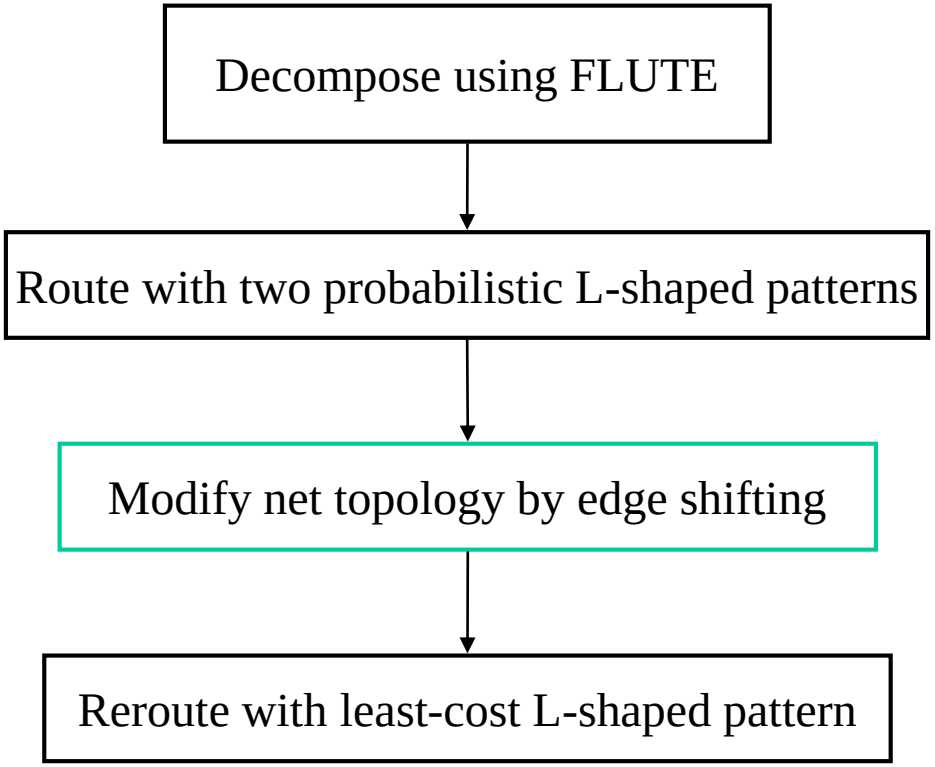
Initial Stage (cont'd)



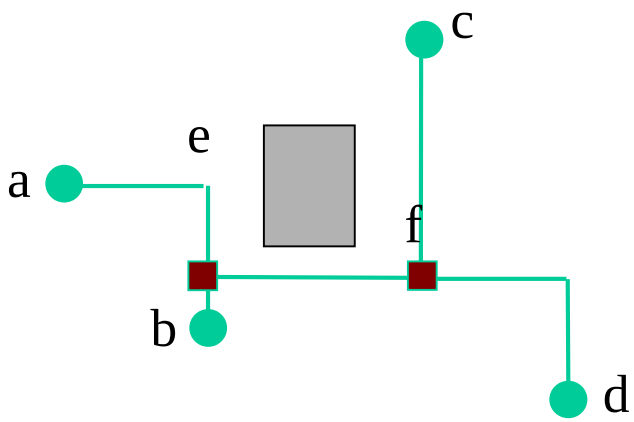
- Edge shifting [ICCAD06]
  - Move edges to less congested regions
  - Does not increase wirelength



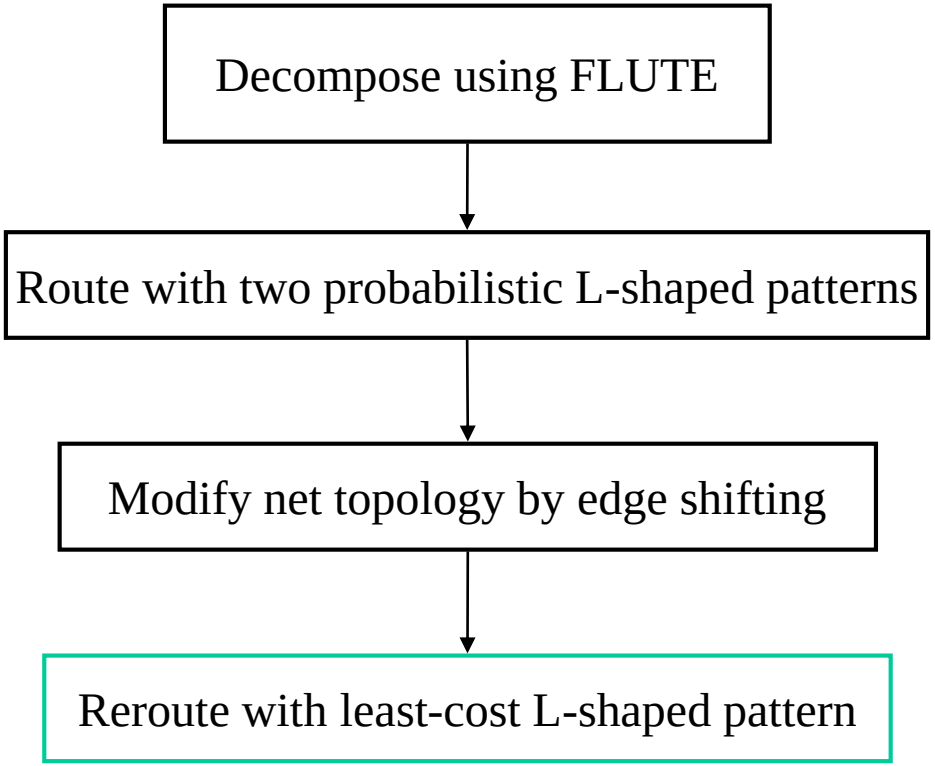
Initial Stage (cont'd)



- Edge Shifting [ICCAD06]
  - Move edges to less congested regions
  - Does not increase wirelength



Initial Stage (cont'd)



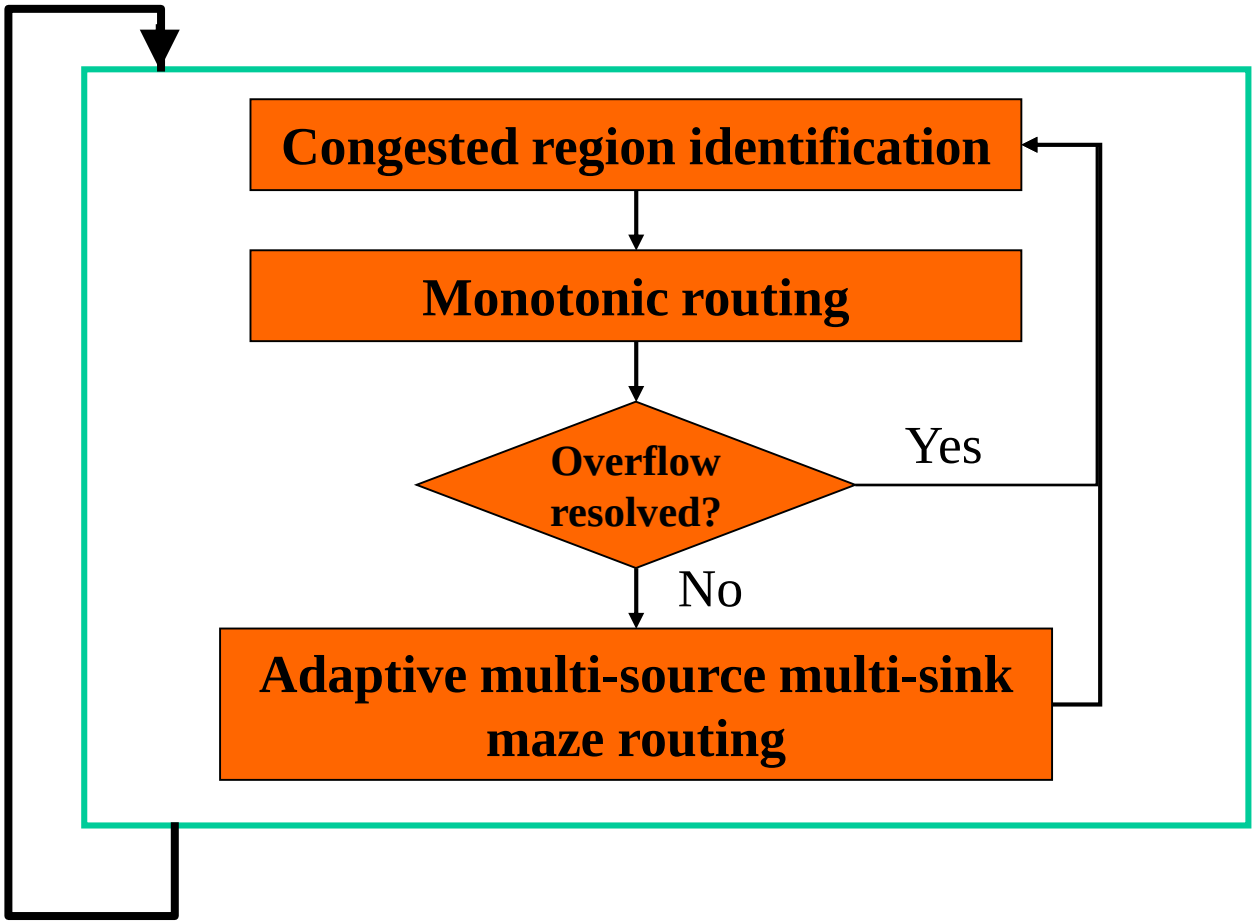
- Reroute all 2-pin nets in increasing order of the size of bounding box
- Add 1 demand to each edge along the chosen L-shaped path

$$cost_e = 1 + \frac{a}{1 + e^{-b*(d(e)-s(e))}}$$

[ICCAD06]

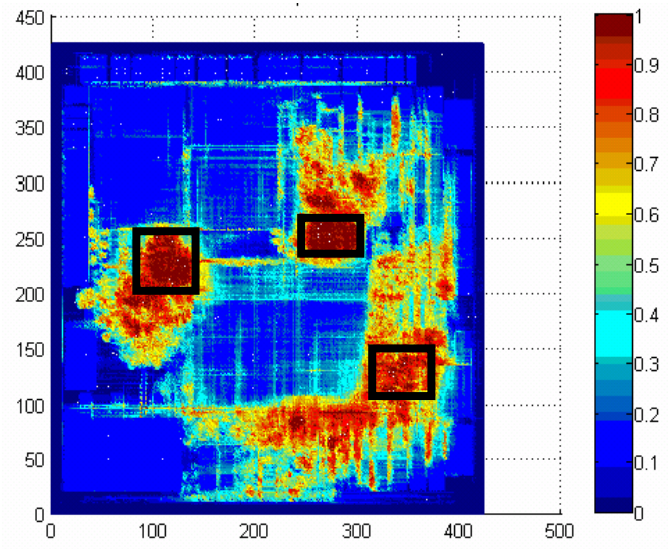
Main Stage

For a number of iterations



Rip-up and Reroute Based on Congested Region Identification

- The order of nets to be ripped up & rerouted affects the routing quality very much
- An algorithm is proposed to identify congested regions and rip-up & reroute two-pin nets with similar congestion at a time



(ISPD07 contest)

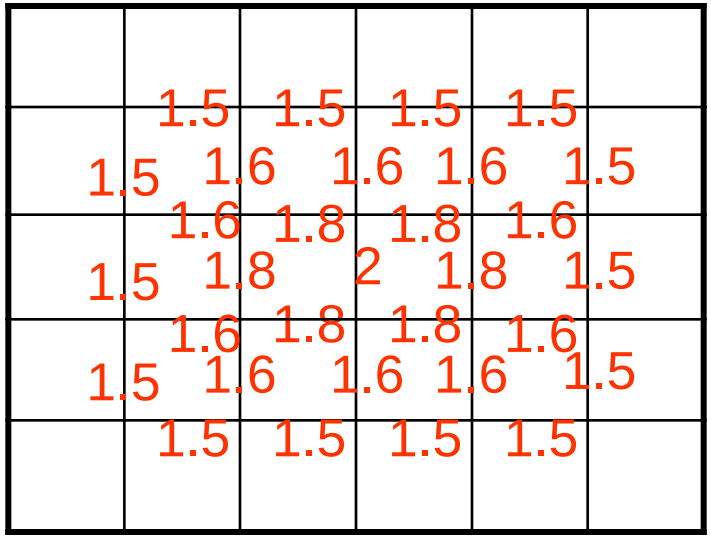
Congested Region Identification

- Partition the interval between the maximum congestion value and 1 into 10 sub-intervals
  - Each congested edge belongs to a sub-interval
- For each sub-interval I_k
  - Expand a rectangular region r_e from each edge e until $avg\_cong(r_e)$ is smaller than the lower bound of I_k

$$avg\_cong(r_e) = \frac{\sum d(e)}{\sum s(e)} \qquad e_i \text{ is an edge inside } r_e$$

  - Find two-pin nets within each region
  - For each 2-pin net in non-increasing order of bounding box size
    - Rip-up & reroute using monotonic routing
    - Adaptive multi-source multi-sink maze routing, if necessary

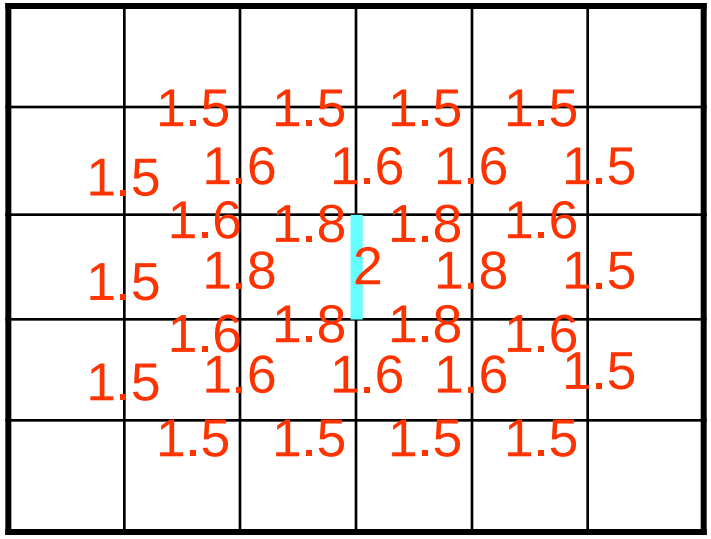
An Example of Region Identification



- Partition [2, 1]

  →{[1.1,1), [1.2, 1.1),..., [1.9, 1.8), [2, 1.9)}

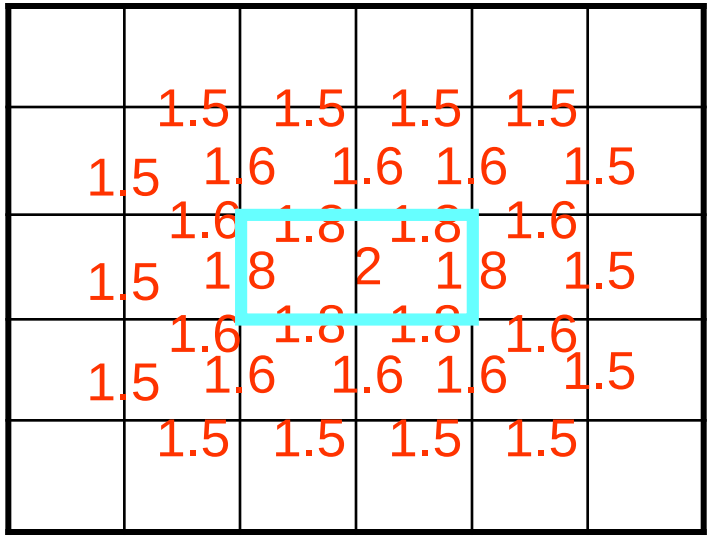
An Example of Region Identification



- Partition [2, 1]

  →{[1.1,1), [1.2, 1.1),..., [1.9, 1.8), [2, 1.9)}

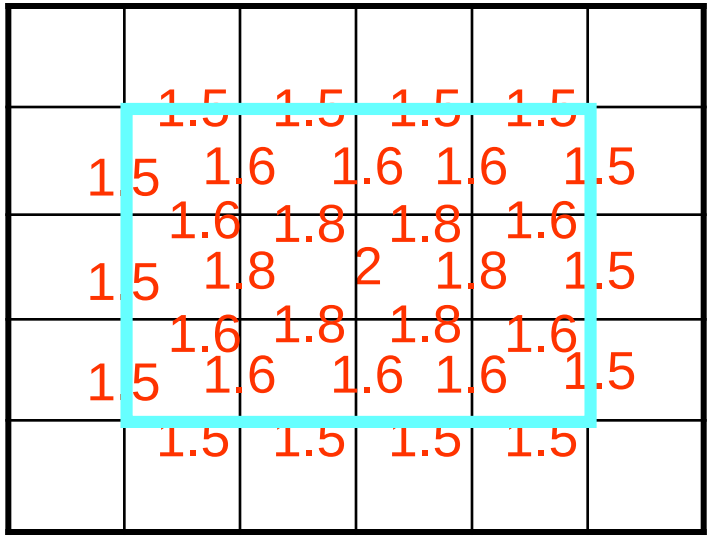
An Example of Region Identification



- Partition [2, 1]

  →{[1.1,1), [1.2, 1.1),..., [1.9, 1.8), [2, 1.9)}

An Example of Region Identification

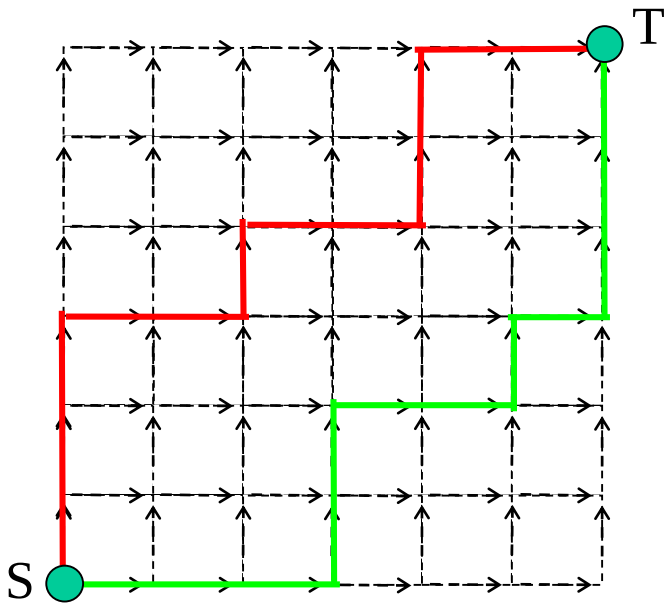


- Partition [2, 1]

  →{[1.1,1), [1.2, 1.1),..., [1.9, 1.8), [2, 1.9)}

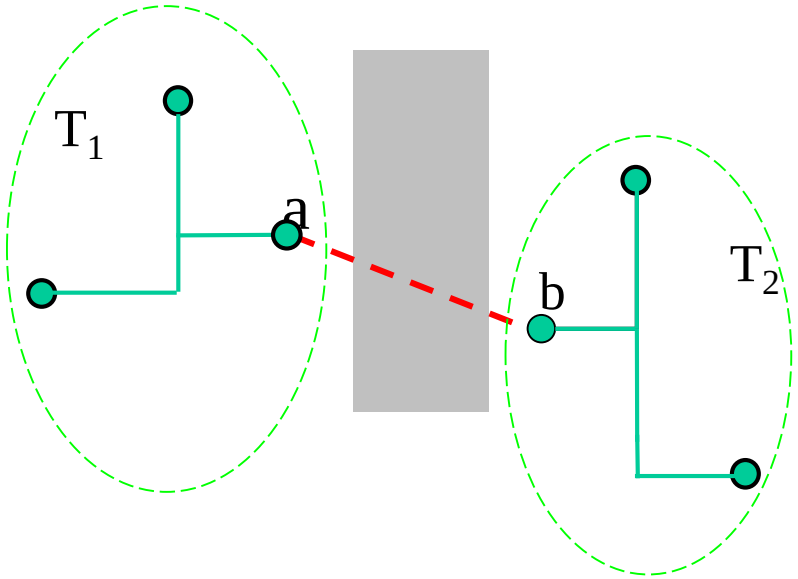
Monotonic Routing

- Proposed in [ASPDAC07]
- Finds the routing path in the direction toward *T* within the bounding box formed by *S* and *T*
- Can be done by dynamic programming with the same complexity as Z-shaped pattern routing

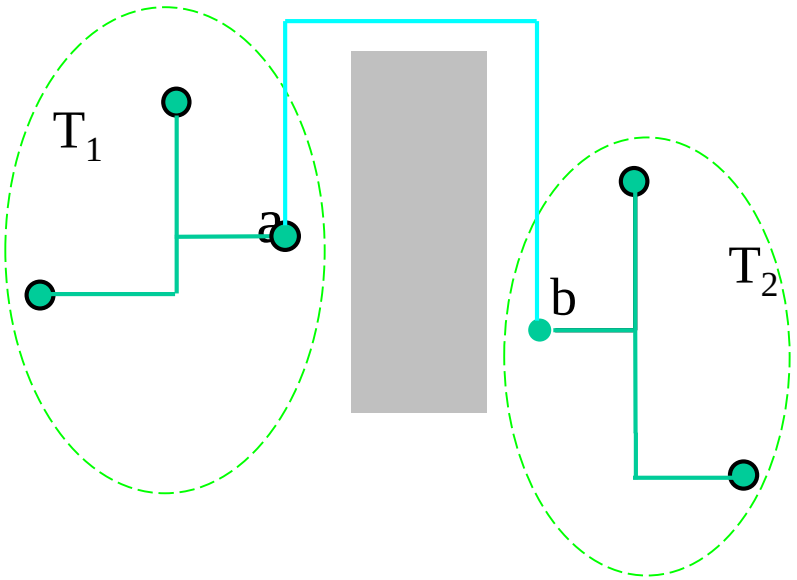


Adaptive Multi-Source Multi-Sink Maze Routing

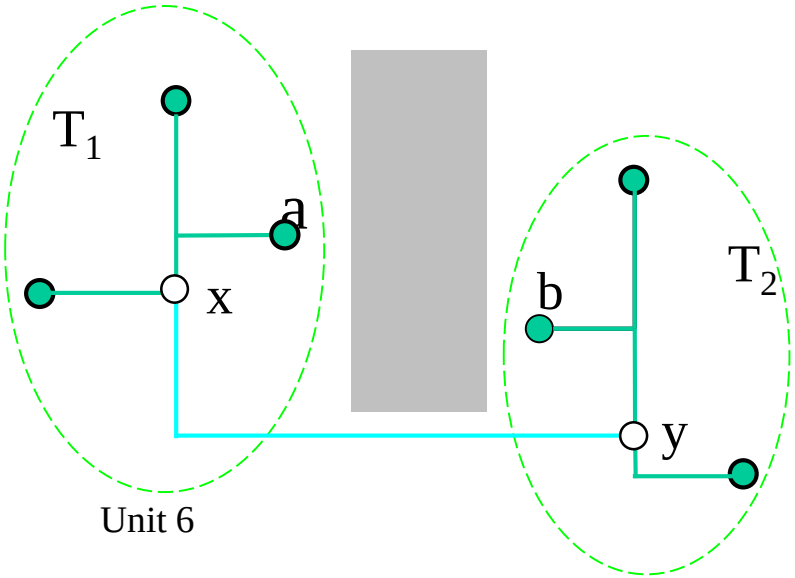
- General maze routing
  - One-source one-sink
- Multi-source multi-mink maze routing
  - Treat all grid points on one subtree as sources
  - Sinks are grid points on another subtree
- Adaptive multi-source multi-mink maze routing
  - Treat only pins or Steiner points as sources and sinks
  - More efficient



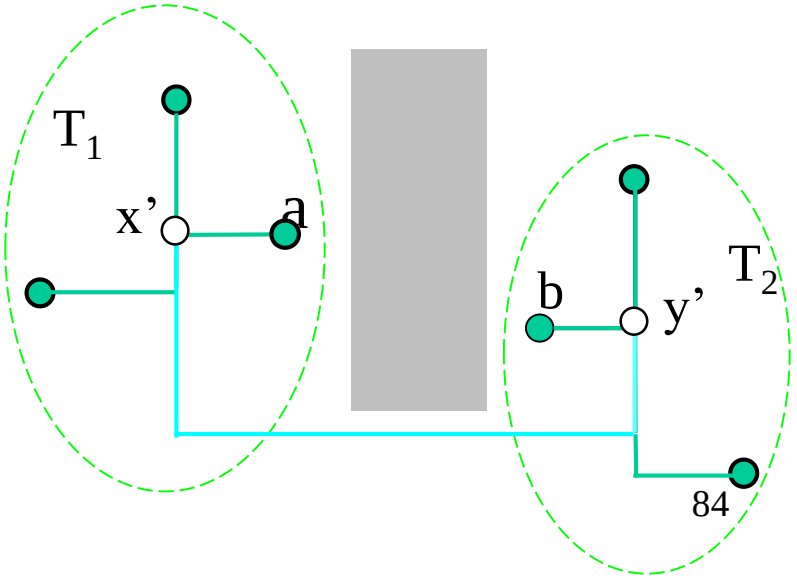
One-source one-sink



Multi-source multi-sink



Adaptive multi-source multi-sink



History Based Cost Function

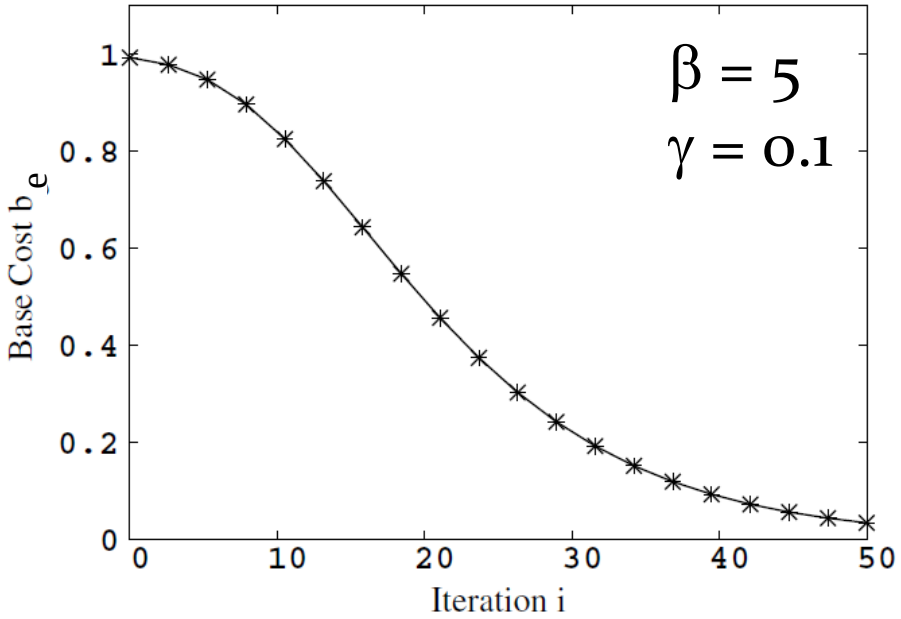
$$cost_e = b_e + h_e \times p_e + vc_e$$

- b_e: adaptive base cost function
- $h_e \times p_e$: congestion cost function with overflow prediction
- vc_e: via cost function for multi-layer design
- Sharing edges does not need additional cost

Adaptive Base Cost Function

- Provides a base cost between 1 to 0 adaptively

$$b_e = 1 - e^{-\beta e^{-\gamma i}}$$



Congestion Cost Function with Overflow Prediction

- $h_e \times p_e$: congestion cost
  - h_e: historical term

$$h_e^{i+1} = \begin{cases} h_e^i + 1 & \text{if } e \text{ has overflow} \\ h_e^i & \text{otherwise} \end{cases}$$

  - p_e: current congestion penalty

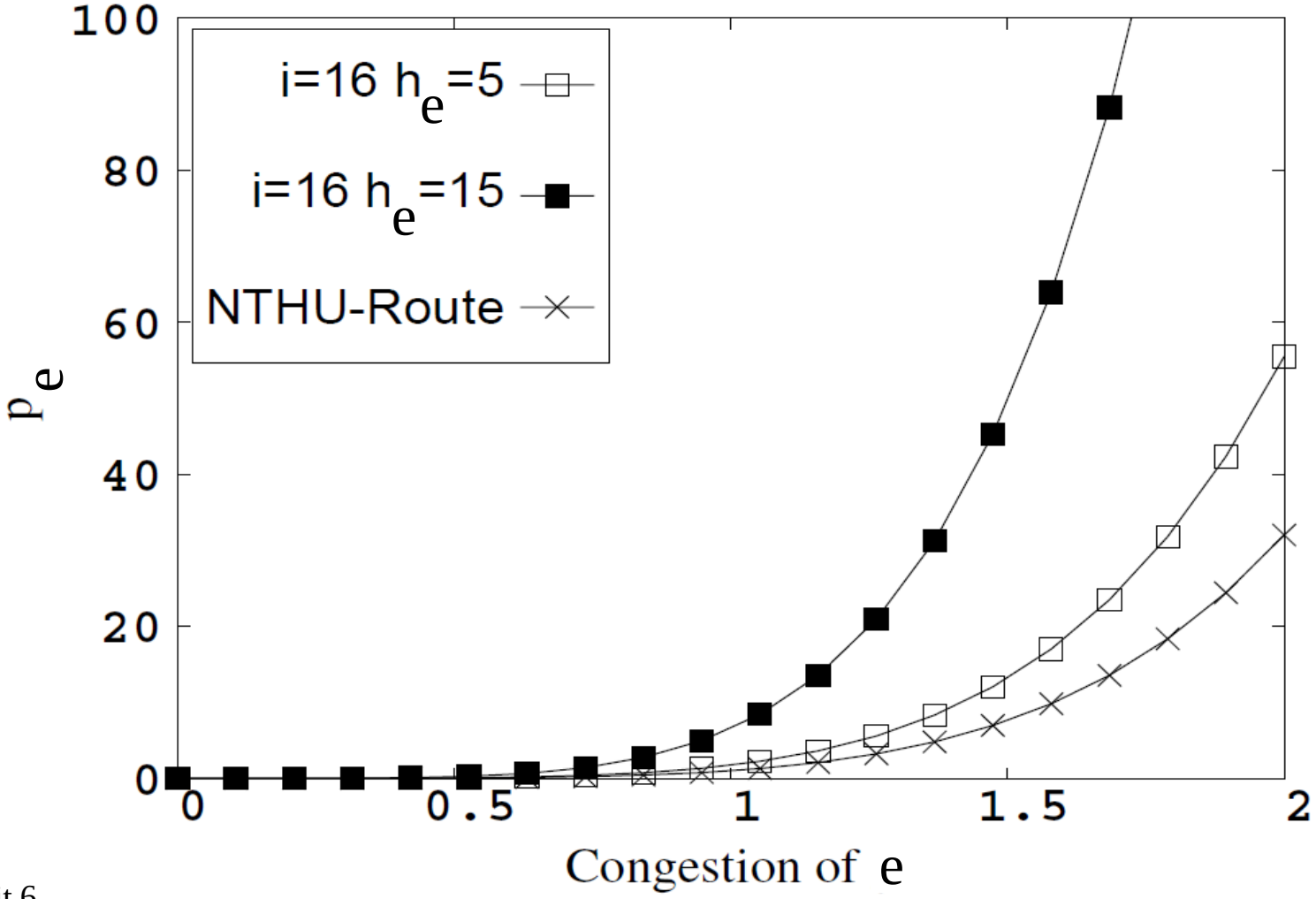
$$p_e = \left(\frac{d_e + 1}{c_e} \times f(h_e, i) \right)^{k1}$$

$$f(h_e, i) = \left(\frac{i \times (k_2 + adj(i))}{i \times (k_2 + adj(i)) - (h_e - 1)} \right)$$

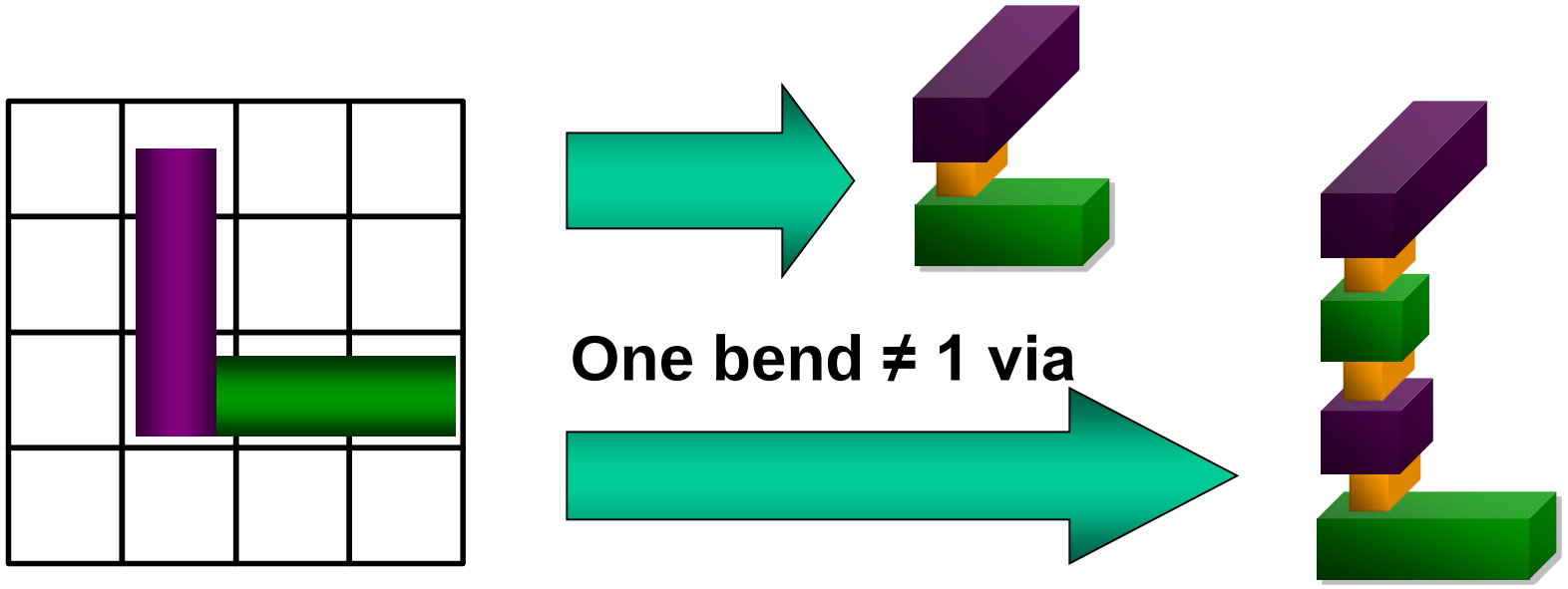
$$adj(i) = k_3 \times (1 - e^{-\beta e^{-\gamma i}})$$

    - $f(h_e, i)$: amplify congestion value if e has high probability to have overflow

Curves of Penalty Terms



Via Cost Function for Multi-layer Design



$$vc_e = \begin{cases} v_e \times c_e \times b_e & \text{if passing e makes a bend,} \\ 0 & \text{otherwise} \end{cases}$$

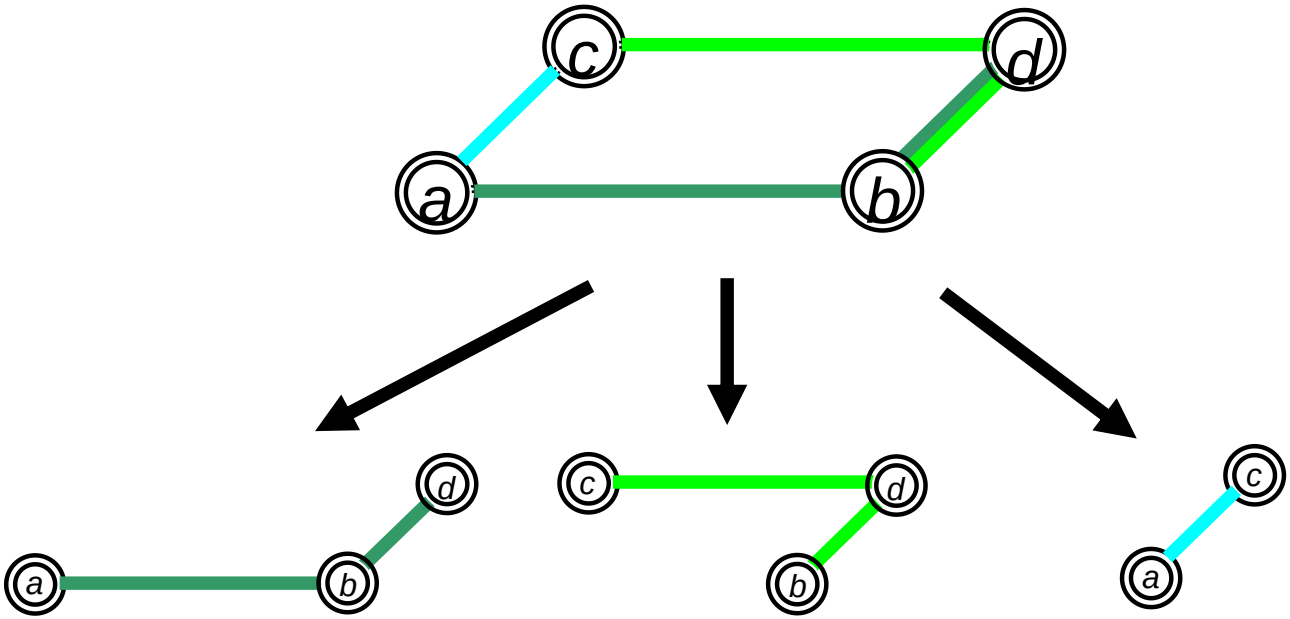
- v_e: the expected amount of vias for a bend
- c_e: the wirelength of a via
- b_e: adaptive base cost function

Refinement Stage

- Apply when iterative historical routing gets stuck
  - Recall that $cost_e = b_e + h_e \times p_e + vc_e$
  - $h_e \times p_e$ dominates edge cost when e tends to be congested
- Another evaluation way
  - If passing e induces overflow → $cost_e$=1
  - Otherwise → $cost_e$=0
- Rip-up & reroute 2-pin nets in decreasing order of total overflow
  - Monotonic routing
  - Adaptive multi-source multi-sink maze routing, if necessary

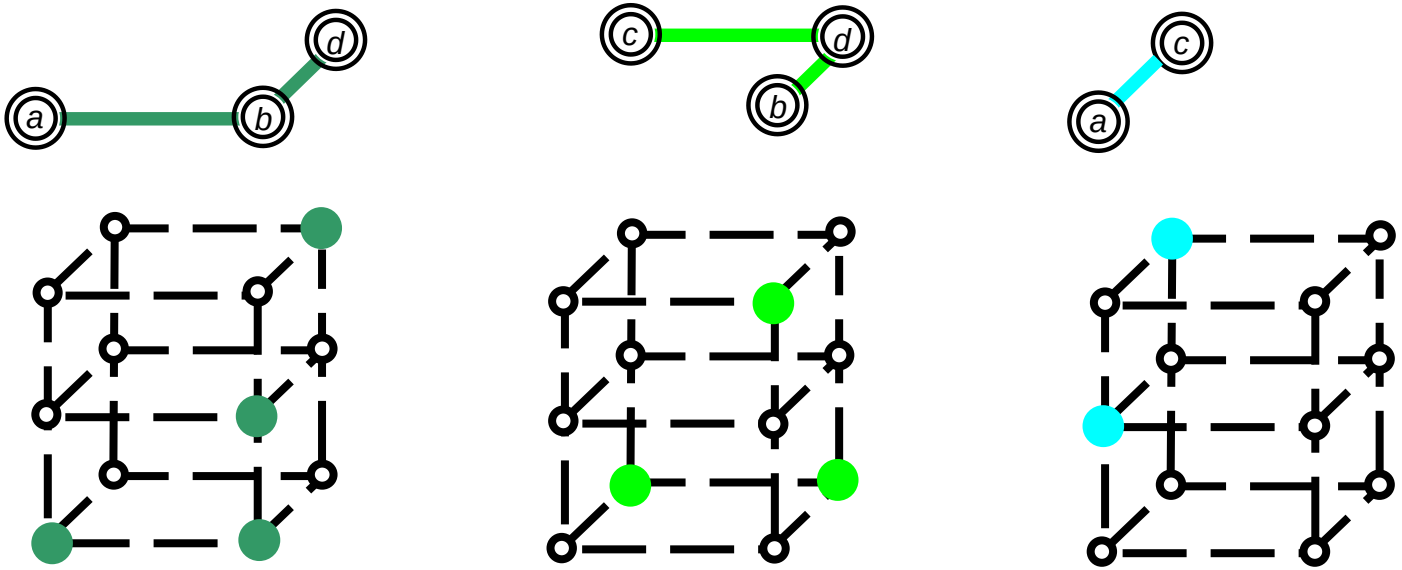
Layer Assignment Stage

- Input: 2D global routing solution $S(G^1,N^1)$



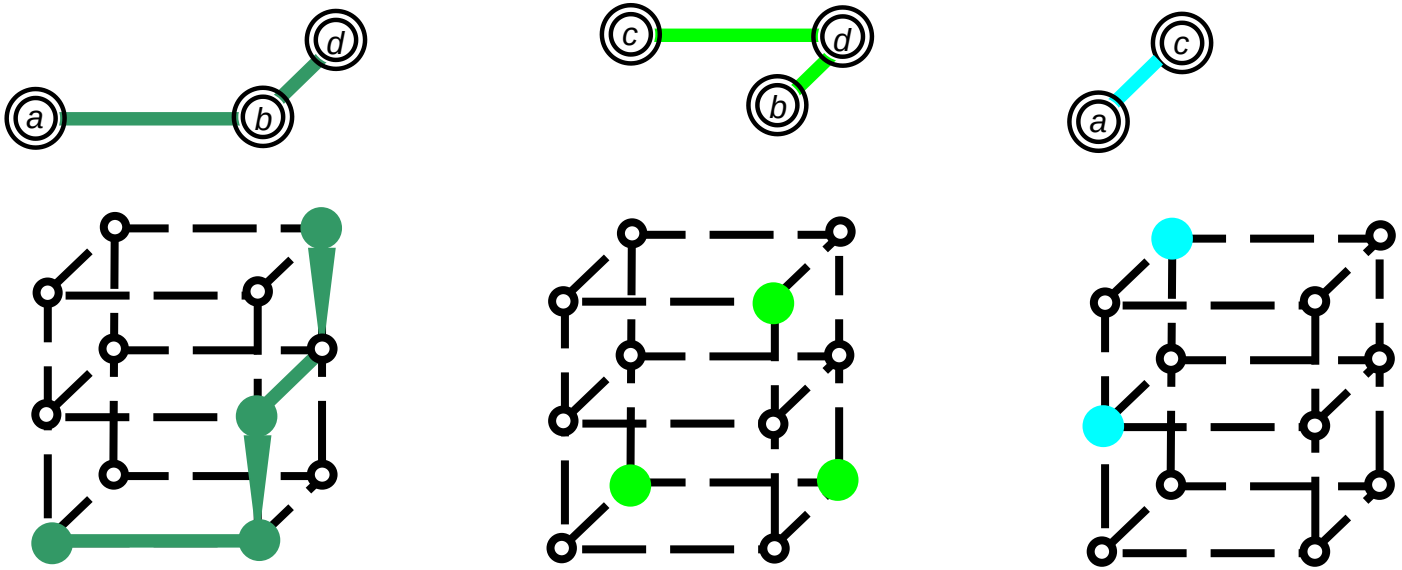
Layer Assignment Stage (cont'd)

- Output: 3D (k-layer) global routing solution $S(G^k, N^k)$



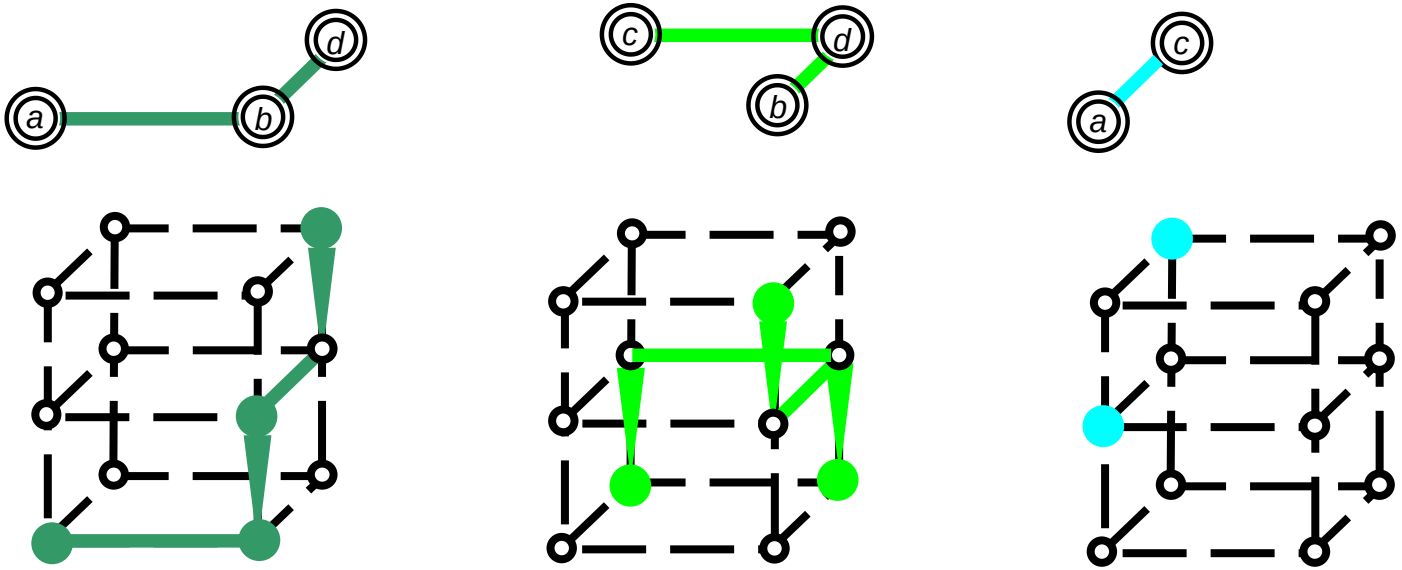
Layer Assignment Stage (cont'd)

- Output: 3D (k-layer) global routing solution $S(G^k, N^k)$



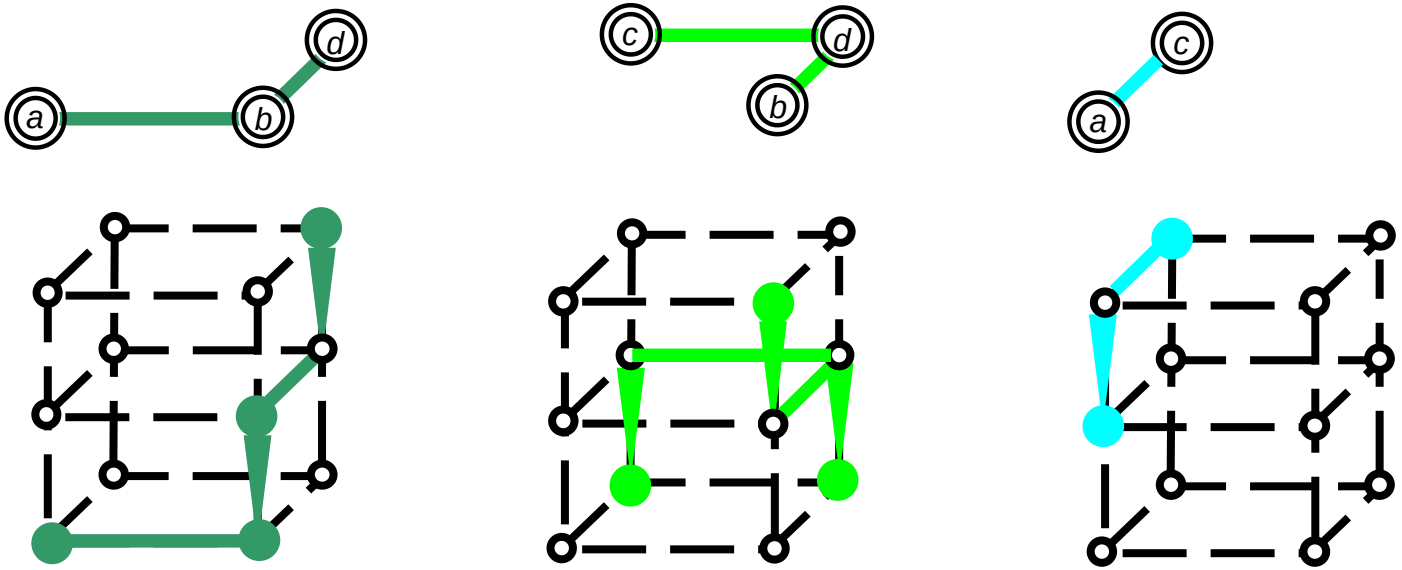
Layer Assignment Stage (cont'd)

- Output: 3D (k-layer) global routing solution $S(G^k, N^k)$



Layer Assignment Stage (cont'd)

- Output: 3D (k-layer) global routing solution $S(G^k, N^k)$



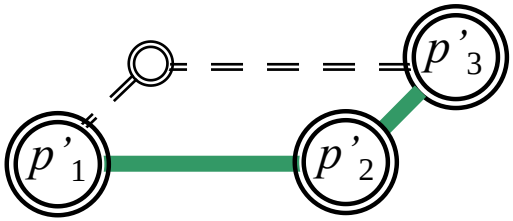
Layer Assignment Stage (cont'd)

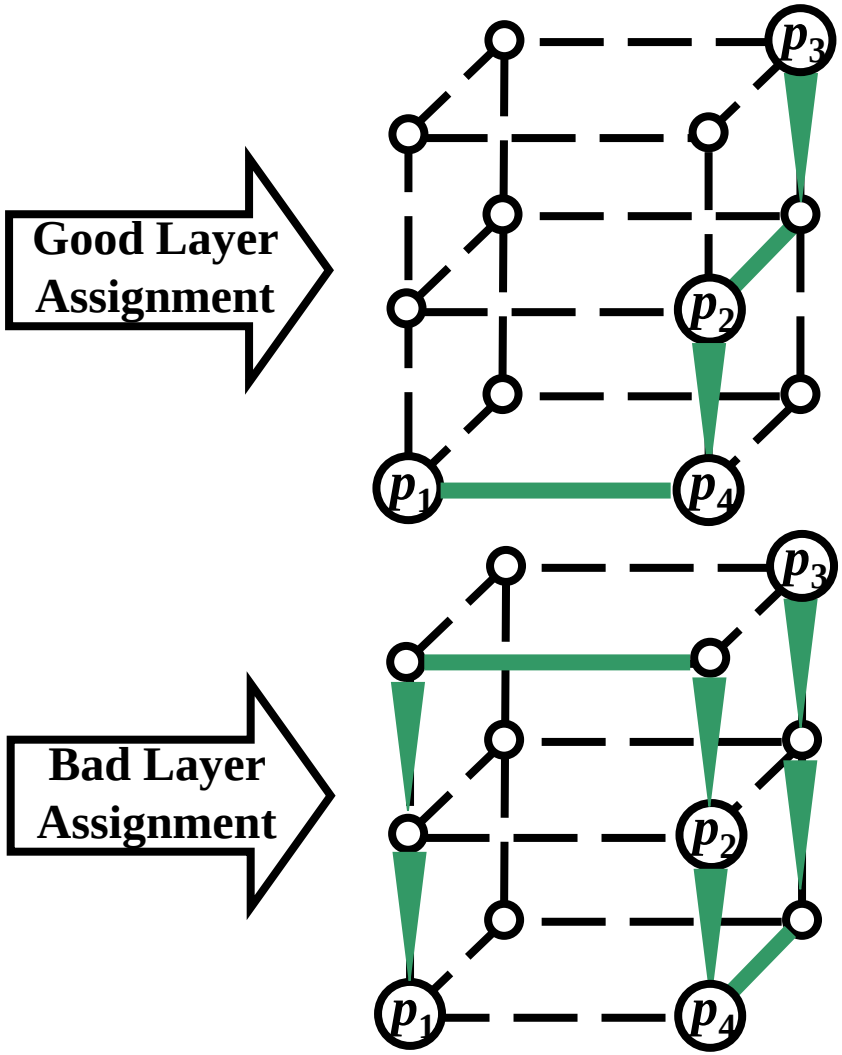
- Congestion constraints
  - Total overflow constraint
    $TO(S(G^k,N^k))=TO(S(G^1,N^1))$
  - Maximum overflow constraint
    $MO(S(G^k,N^k))= \lceil MO(S(G^1,N^1))/k \rceil$ (w/o preferred routing directions)
    or $= \lceil MO(S(G^1,N^1))/(k/2) \rceil$ (with preferred routing directions)
- Objective: via count minimization

Layer Assignment Algorithms

- 2-layer design
  - Allow horizontal wires on one layer and vertical wire on the other layer
  - Solution can be transformed by directly adding a via for each bend on the routing path
- *k*-layer design ($k > 2$)
  - A layer assignment method is adopted
    - Two components
      - Net order determination
      - Single-net layer assignment considering congestion constraints

Single-net Layer Assignment





Net Order Determination

- For each net T_i^1, compute $Score(T_i^1)$
- Sort nets according to scores

1 2 3

$$Score(T_i^1) = \alpha / Length(T_i^1) + \beta \times PinNum(T_i^1) + \gamma \times AvgDensity(T_i^1)$$

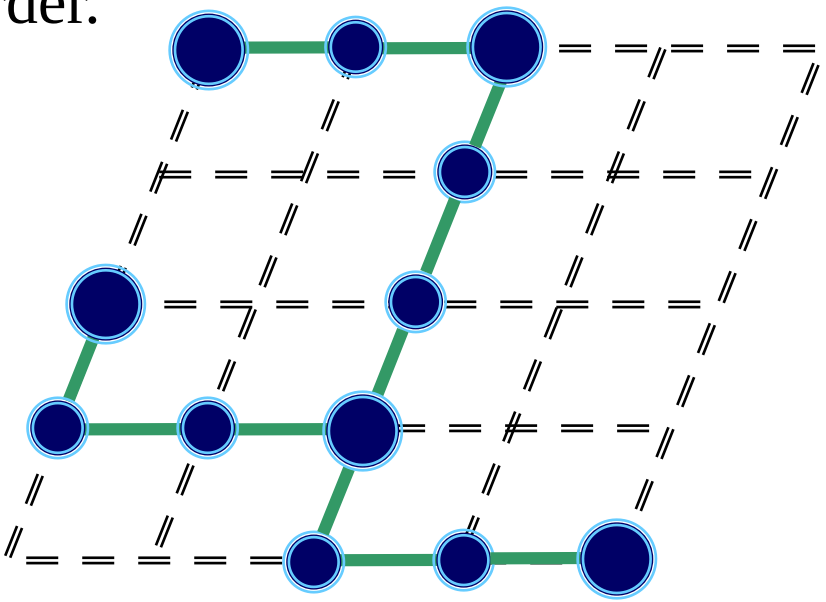
1 $Length(T_i^1)\uparrow \Rightarrow$ { cost of $T_i^k\uparrow$; # solutions $T_i^k\uparrow$ } $Score(T_i^1) \propto 1/Length(T_i^1)$

2 $PinNum(T_i^1)\uparrow \Rightarrow$ # solutions $T_i^k\downarrow$ $Score(T_i^1) \propto PinNum(T_i^1)$

3 $AvgDensity(T_i^1)\downarrow \Rightarrow$ Need of resource $\downarrow$ $Score(T_i^1) \propto AvgDensity(T_i^1)$

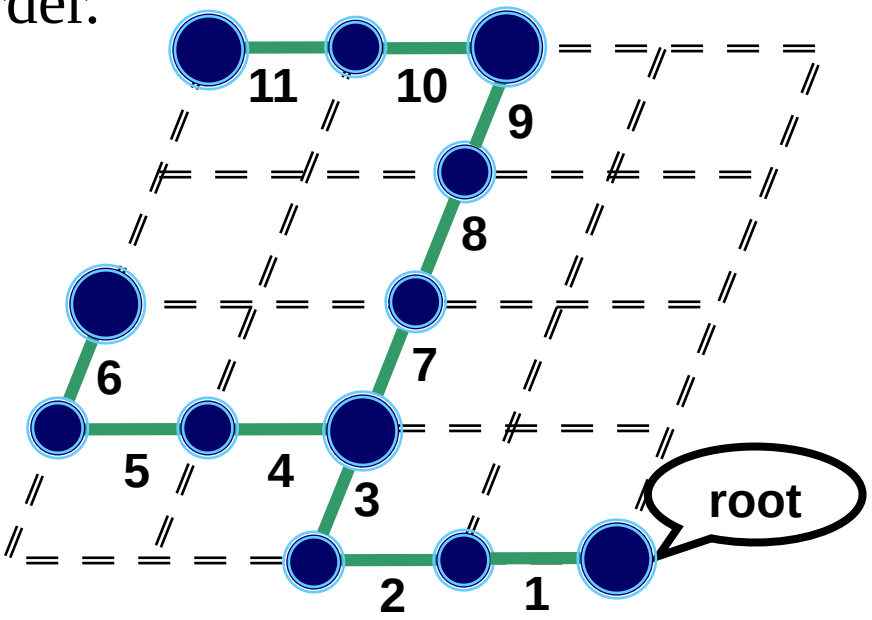
Single-Net Layer Assignment

- Optimal congestion-constrained single-net layer assignment for via count minimization
- For each net, layer assignment assigns each edge one at a time.
- The layer assignment order is the reverse order of DFS traversal order.



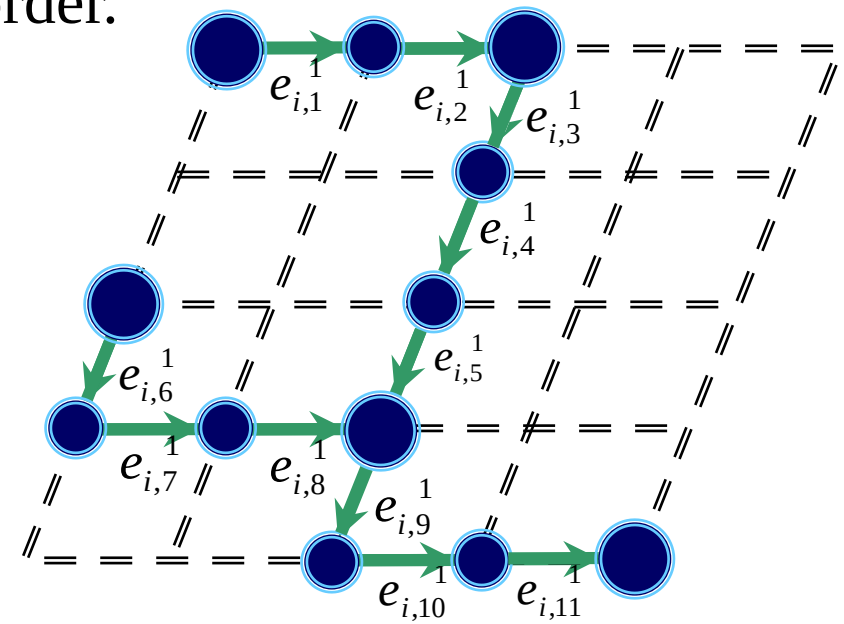
Single-Net Layer Assignment

- Optimal congestion-constrained single-net layer assignment for via count minimization
- For each net, layer assignment assigns each edge one at a time.
- The layer assignment order is the reverse order of DFS traversal order.



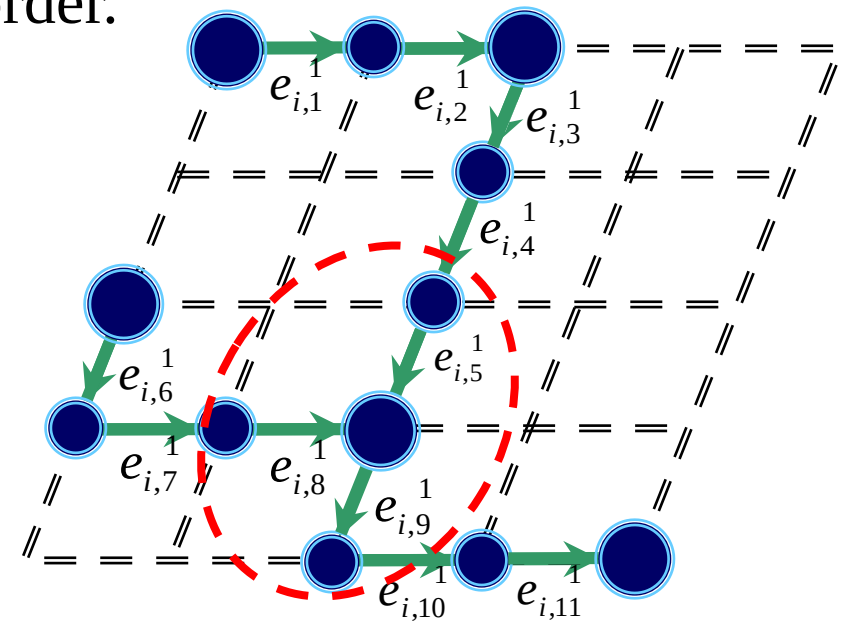
Single-Net Layer Assignment

- Optimal congestion-constrained single-net layer assignment for via count minimization
- For each net, layer assignment assigns each edge one at a time.
- The layer assignment order is the reverse order of DFS traversal order.



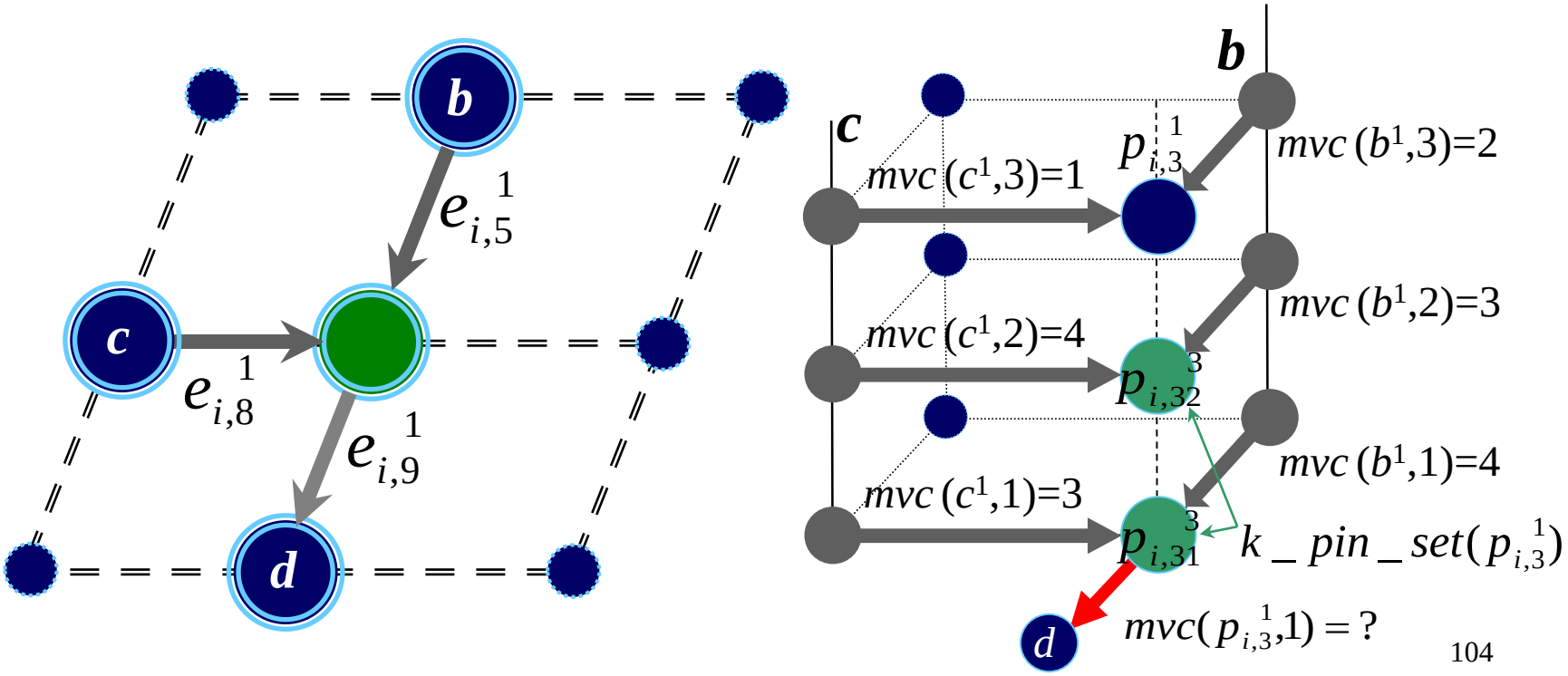
Single-Net Layer Assignment

- Optimal congestion-constrained single-net layer assignment for via count minimization
- For each net, layer assignment assigns each edge one at a time.
- The layer assignment order is the reverse order of DFS traversal order.



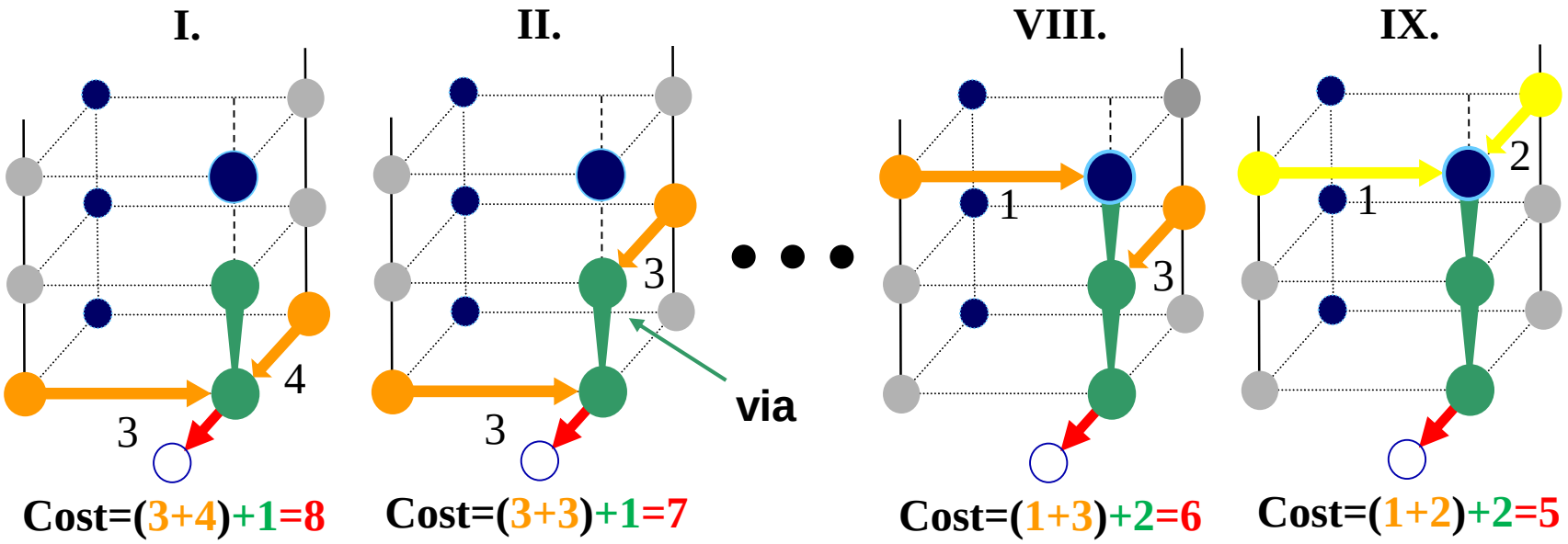
Single-Net Layer Assignment (cont'd)

- *mvc*(*v*,*l*): The minimum via count of a subtree rooted at *v* when the edge between *v* and its parent is assigned to layer *l*.



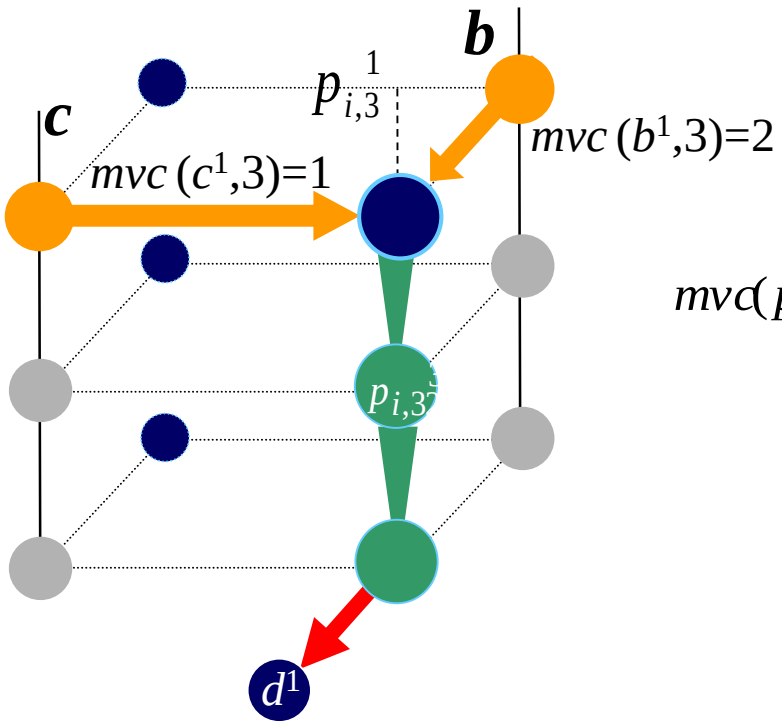
Single-Net Layer Assignment (cont'd)

- Enumerate all possible layer assignment results of children.



Single-Net Layer Assignment (cont'd)

- The minimum among all combinations is the answer



Cost from $ch\_v(p_{i,3}^{1})$

$$mvc(p_3^1,1) = (mvc(c^1,3) + mvc(b^1,3)) +$$

The via cost on $p_{i,3}^{1}$

=(1+2)+2

=5 vias

Single-Net Layer Assignment (cont'd)

- To take the maximum and total overflow constraints into account, an edge e^1 in G^1 can be assigned to an edge e in $k\_edge\_set(e^1)$ if after the assignment, the following two conditions are satisfied
  - $o(e) \leq \lceil MO(S(G^1, N^1))/k \rceil$ (or $o(e) \leq \lceil MO(S(G^1, N^1))/(k/2) \rceil$)
  - $o(e^1) \geq \sum o(e^k)$, where e^k is in $k\_edge\_set(e^1)$
- If assigning the edge between v and its parent to layer l violates either of the two conditions, then $mvc(v,l)$ is set to ∞.

Extensions of NTHU-Route 2.0

- **Temperature-aware global routing**
  - Y.-T. Lee, Y.-J. Chang, and T.-C. Wang, "A Temperature-Aware Global Router," VLSI-DAT 2010.
- **Antenna-aware layer assignment**
  - T.-H. Lee and T.-C. Wang, "Simultaneous Antenna Avoidance and Via Optimization in Layer Assignment of Multi-layer Global Routing," ICCAD 2010.
- **Layer-directives-aware global routing**
  - Y.-J. Chang, T.-H. Lee and T.-C. Wang, "GLADE: A Modern Global Router Considering Layer Directives," ICCAD 2010.
  - T.-H. Lee, Y.-J. Chang and T.-C. Wang, "An Enhanced Global Router with Consideration of General Layer Directives," ISPD 2011.

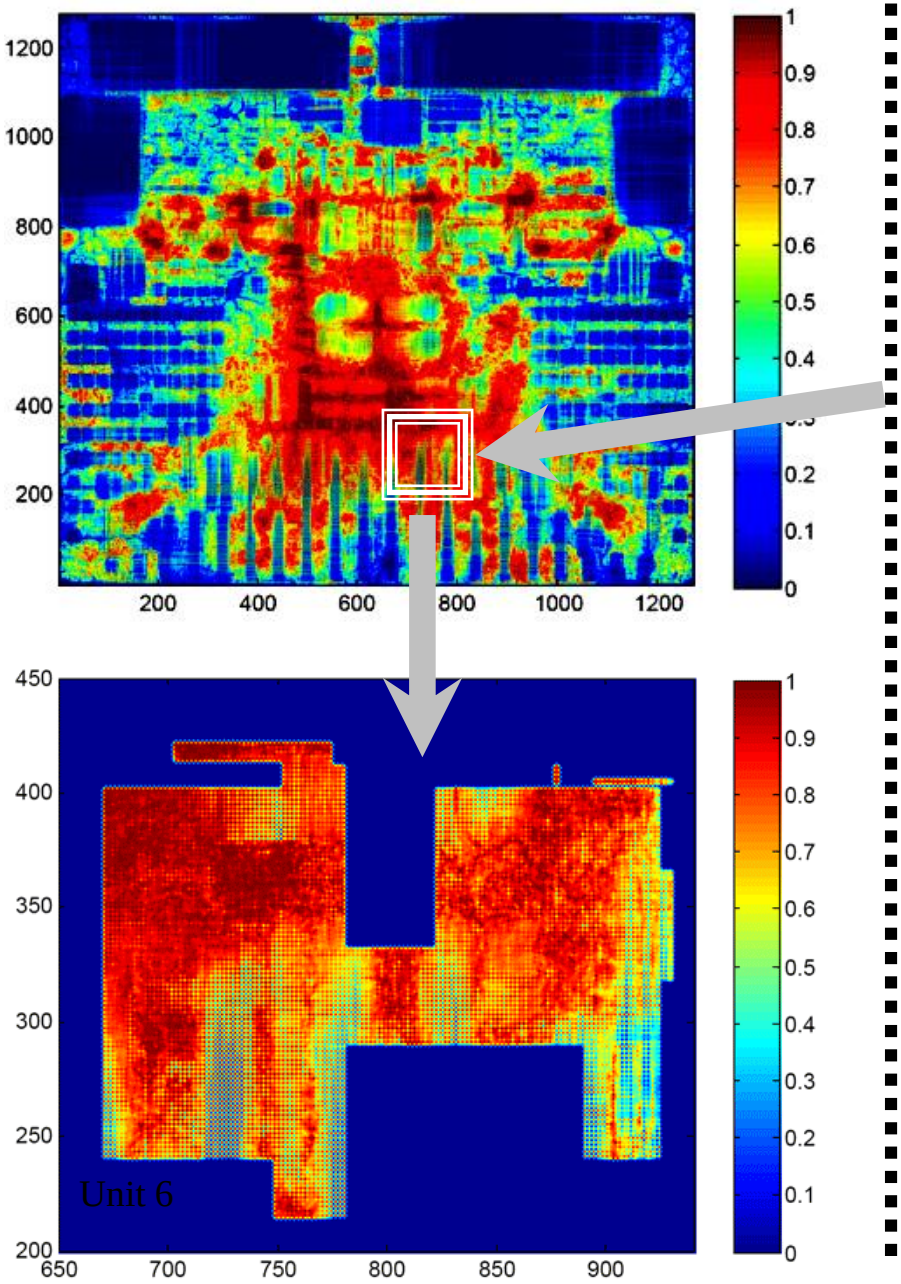
Extensions of NTHU-Route 2.0

- Thermal becomes an even more important problem in modern VLSI designs than before
  - J. Black proposed a model to formulate Mean Time to Failure (MTF) of an interconnect due to electromigration

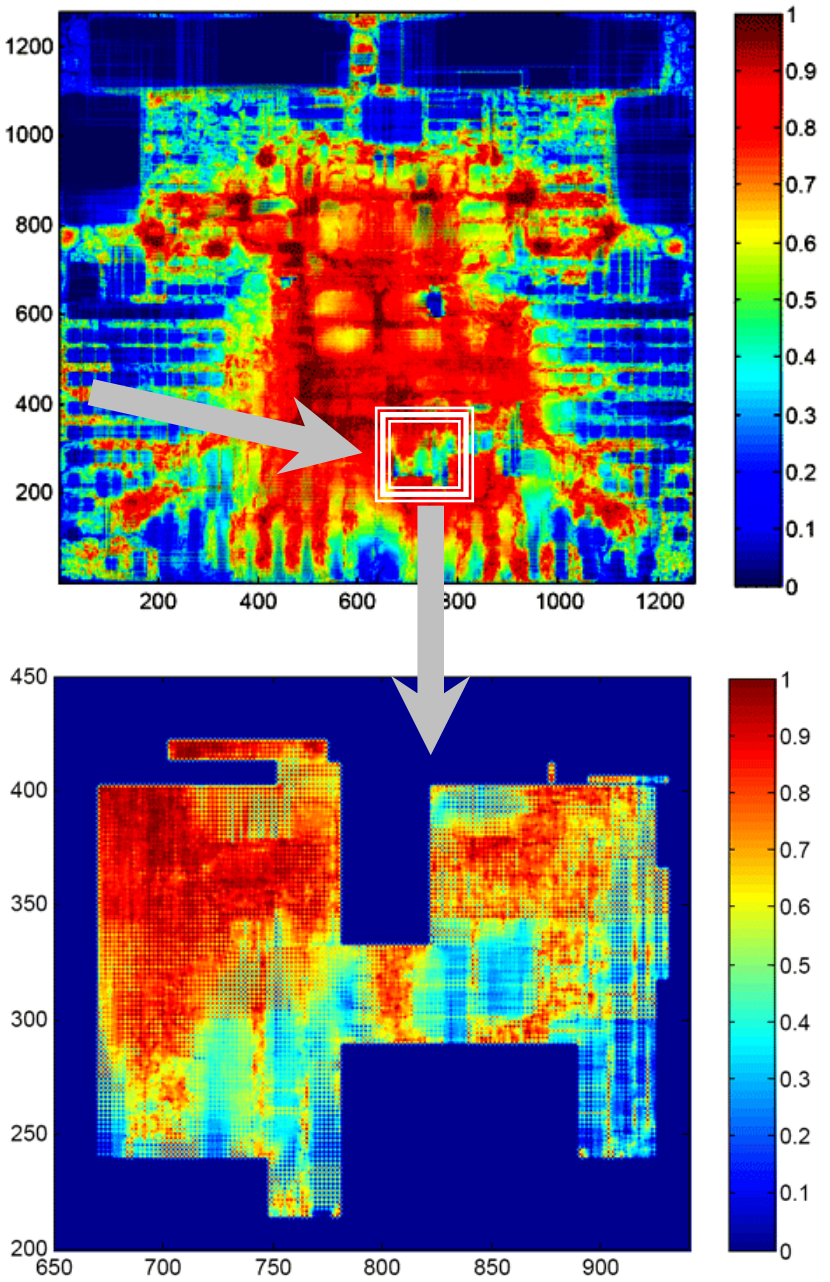
$$\mathrm{MTF} = M\, j^{-2} \mathrm{e}^{\frac{Q}{kT}}$$

- **Temperature-aware global routing** [VLSI-DAT 2010]
  - Also minimize the amount of routed net segments located in hot spots
  - Enhancements
    - Temperature-aware L-shaped pattern routing in 2D routing
    - Two temperature-aware cost functions in 2D routing

Congestion map of NTHU-Route 2.0

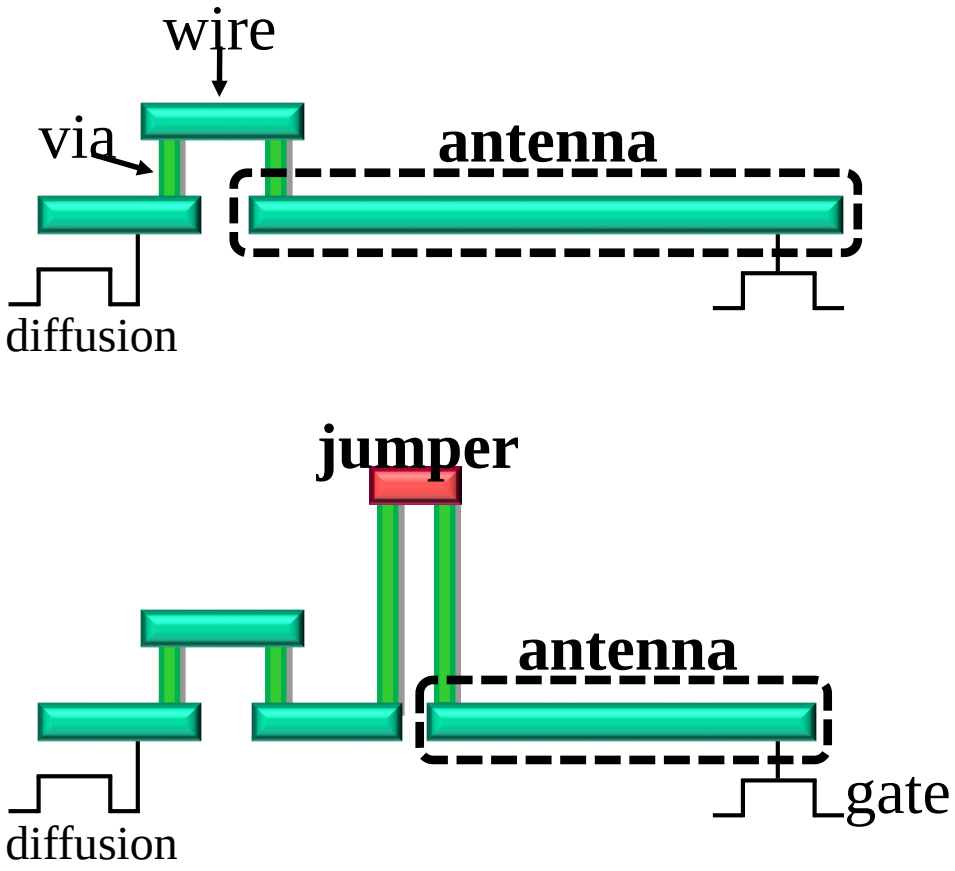


Congestion map of enhanced router



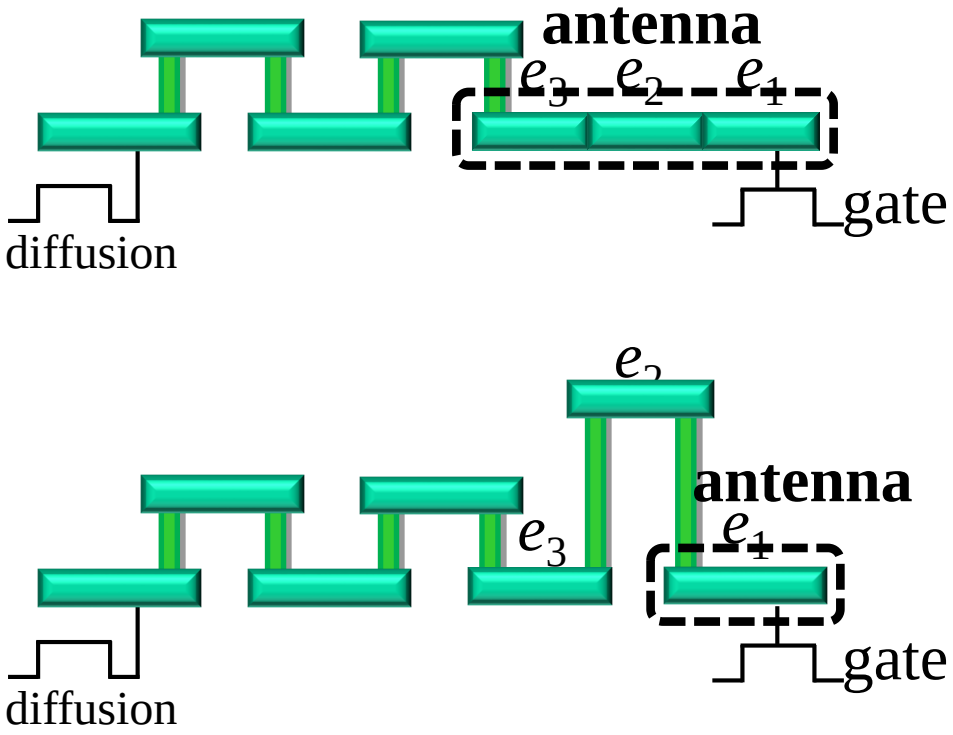
Extensions of NTHU-Route 2.0 (cont'd)

- Antenna effect



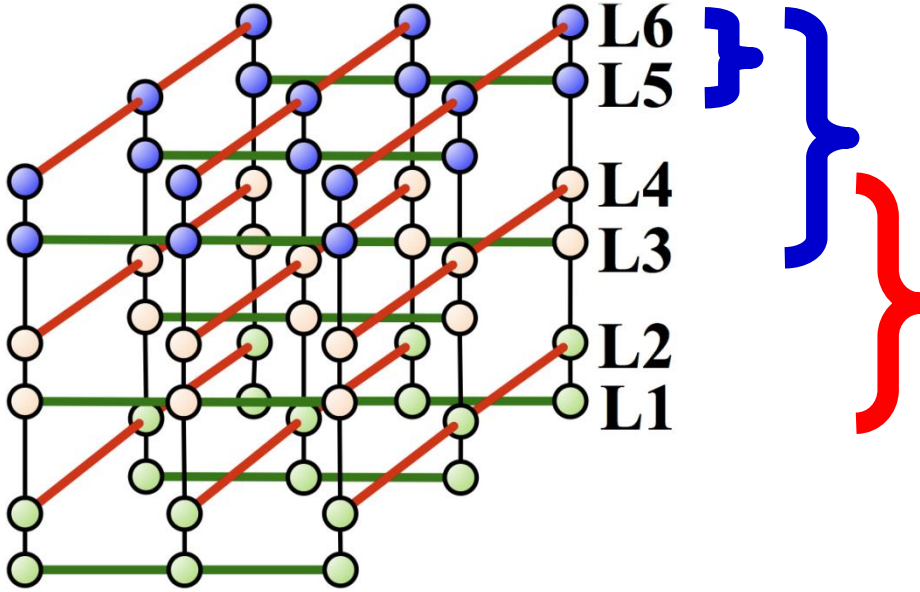
Extensions of NTHU-Route 2.0 (cont'd)

- **Antenna-aware layer assignment** [ICCAD 2010]
  - Also minimize the amount of antenna violations
  - Enhancements
    - Single-net layer assignment algorithm is modified to take both antenna avoidance and via count minimization into account for each net
    - A mini-cost max-flow based refinement procedure is proposed to further reduce the via count but without increasing the amount of antenna violations

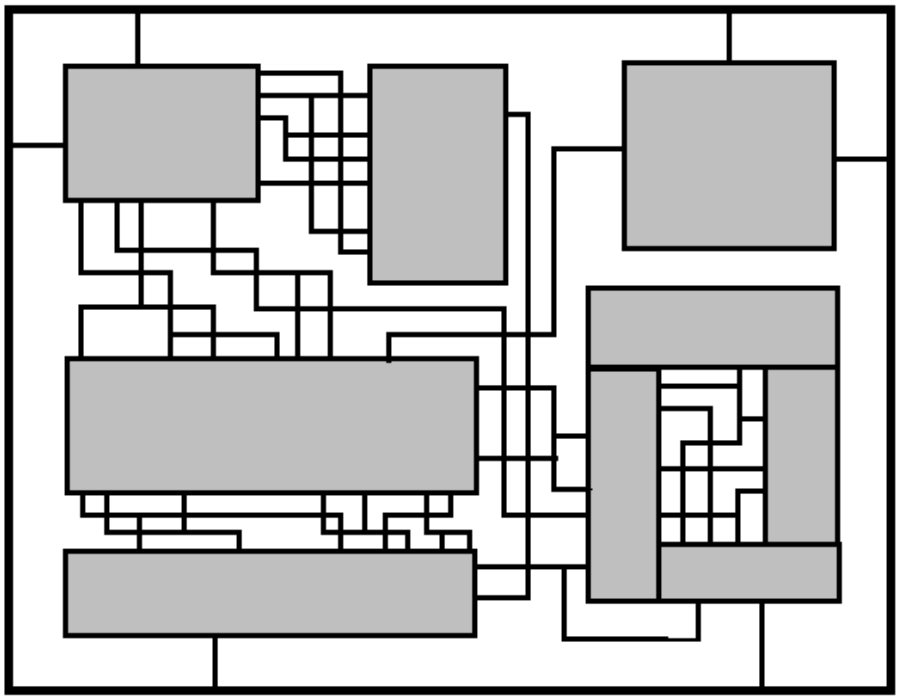


Extensions of NTHU-Route 2.0 (cont'd)

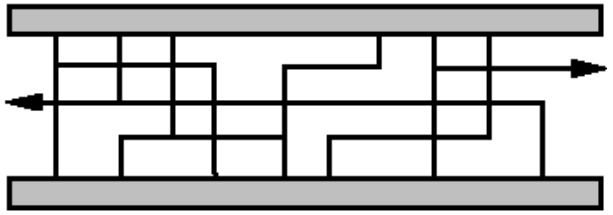
- Layer directives
  - Specify target layer ranges often for timing-critical nets to meet the performance target in a modern physical design flow
- **Layer-directives-aware global routing** [ICCAD 2010, ISPD 2011]
  - Also minimizes the amount of layer directive violations
  - Enhancements
    - Pseudo layer assignment during 2D routing
    - Layer–directives-aware layer assignment



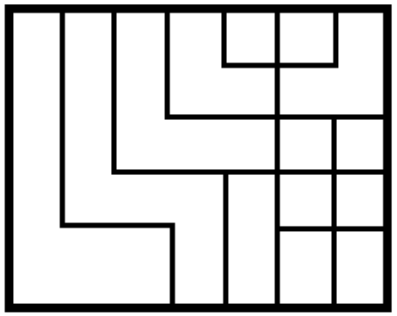
Detailed Routing



Detailed routing



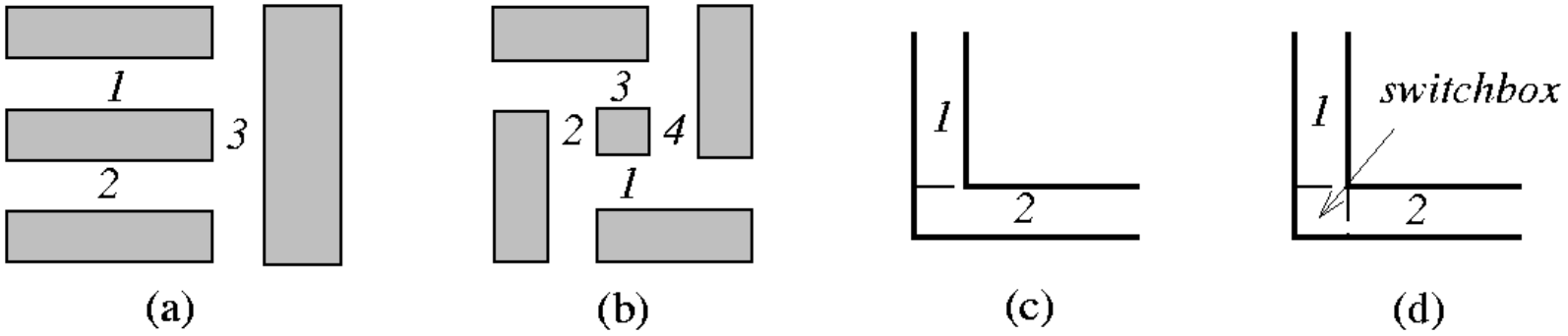
channel routing



switchbox routing

Order of Routing Regions

- (a) No conflicts in case of routing in the order of 1,2, and 3.
- (b) No ordering is possible to avoid conflicts.
- (c) The situation of (b) can be resolved by using L-channels.
- (d) An L-channel can be decomposed into two channels and a switchbox.

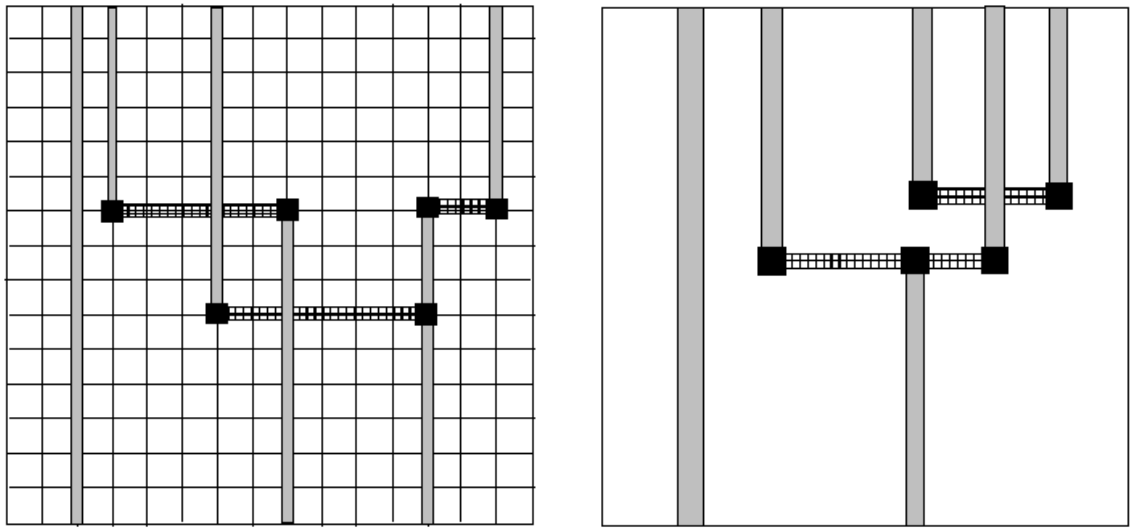


Routing Considerations

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

Routing Models

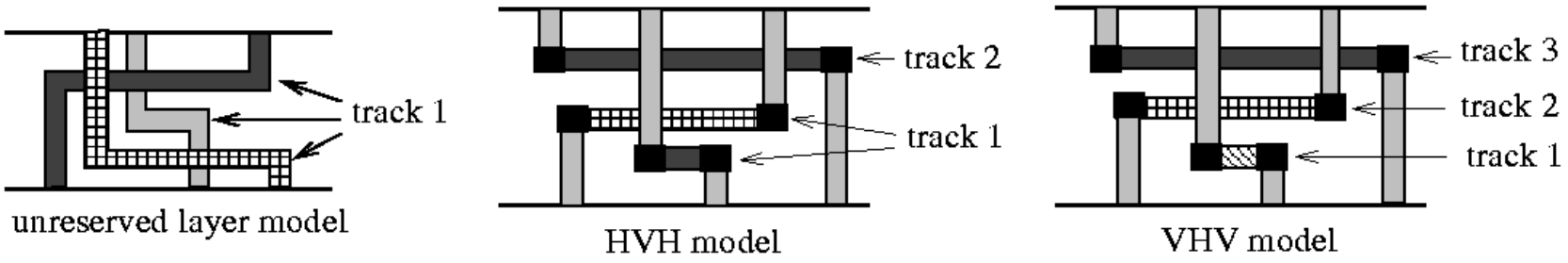
- **Grid-based model:**
  - A grid is super-imposed on the routing region.
  - Wires follow paths along the grid lines.
- **Gridless model:**



grid–based gridless

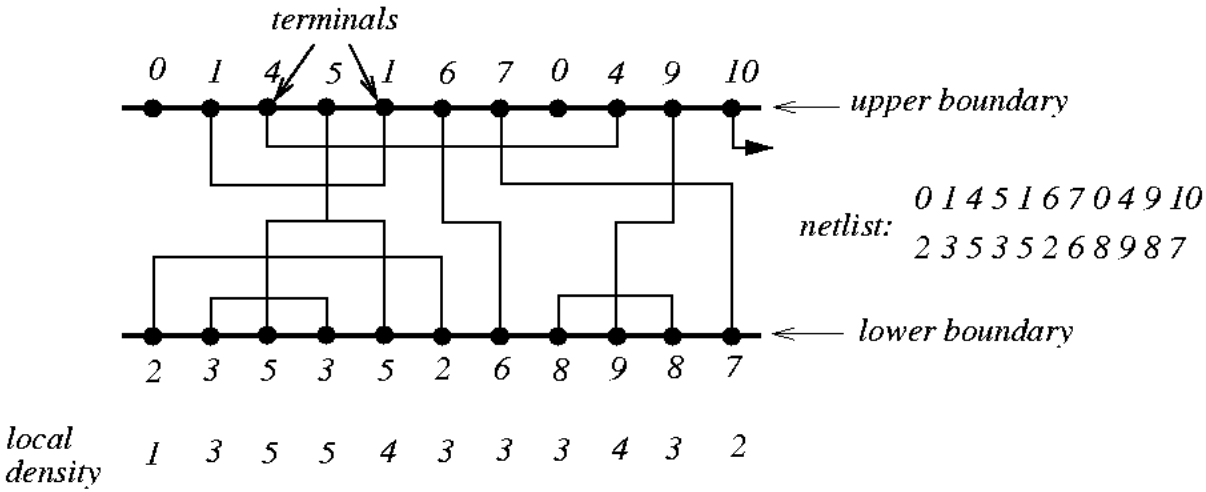
Models for Multi-Layer Routing

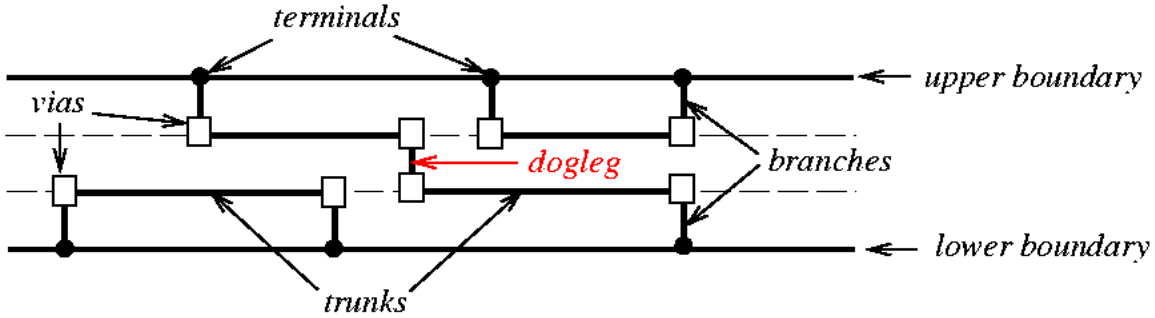
- **Unreserved layer model:** Any net segment is allowed to be placed in any layer.
- **Reserved layer model:** Certain type of segments are restricted to particular layer(s).
  - Two-layer: HV (horizontal-vertical), VH
  - Three-layer: HVH, VHV



3 types of 3–layer models

Terminology for Channel Routing





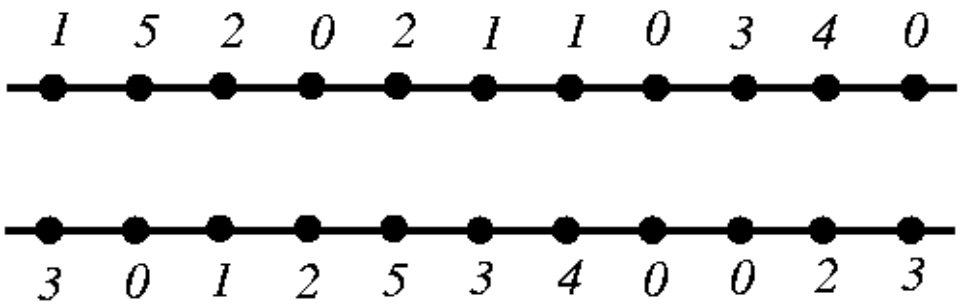
- Local density at column *i*: total # of nets that crosses column *i*.
- Channel density: maximum local density; # of horizontal tracks required $\geqq$ channel density.

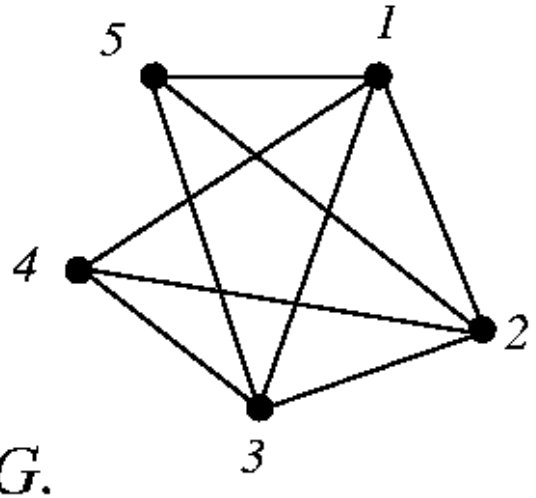
Channel Routing

- **Assignments of horizontal segments of nets to tracks.**
- **Assignments of vertical segments to connect**
  - horizontal segments of the same net in different tracks, and
  - the terminals of the net to horizontal segments of the net.
- **Horizontal and vertical constraints must not be violated.**
  - Horizontal constraints between two nets: The horizontal span of two nets overlaps each other.
  - Vertical constraints between two nets: There exists a column such that the terminal on top of the column belongs to one net and the terminal on bottom of the column belongs to the other net.
- **Objective: Channel height is minimized** (i.e., channel area is minimized).

Horizontal Constraint Graph (HCG)

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

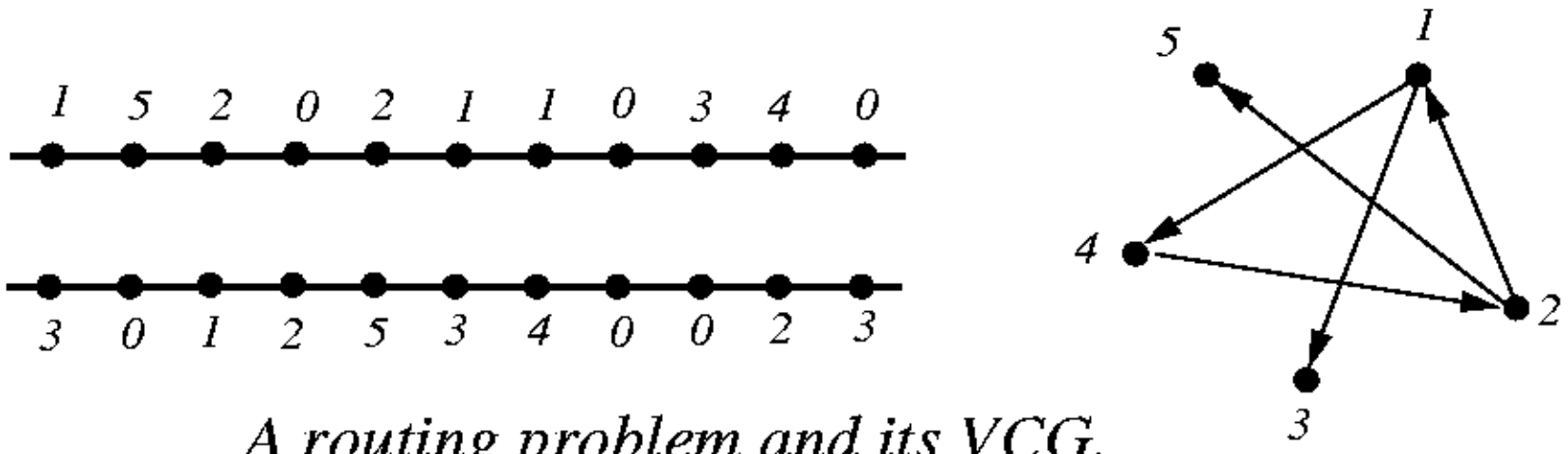




A routing problem and its HCG.

Vertical Constraint Graph (VCG)

- VCG $G = (V, E)$ is a **directed** graph where
  - $V = \{v_i | v_i$ represents a net $n_i\}$
  - $E = \{(v_i, v_j) |$ a vertical constraint exists between n_i and $n_j\}$.
- For graph G: vertices $\Leftrightarrow$ nets; edge $i \rightarrow j \Leftrightarrow$ net i must be above net j.



A routing problem and its VCG.

2-L Channel Routing: Basic Left-Edge Algorithm

- Hashimoto & Stevens, “Wire routing by optimizing channel assignment within large apertures,” DAC, 1971.
- **No vertical constraint.**
- HV-layer model is used.
- **Doglegs are not allowed.**
- Treat each net as an interval.
- Intervals are sorted according to their left-end x-coordinates.
- Intervals (nets) are routed one-by-one according to the order.
- For a net, tracks are scanned from top to bottom, and the first track that can accommodate the net is assigned to the net.
- Optimality: produces a routing solution with the minimum # of tracks (if no vertical constraint).

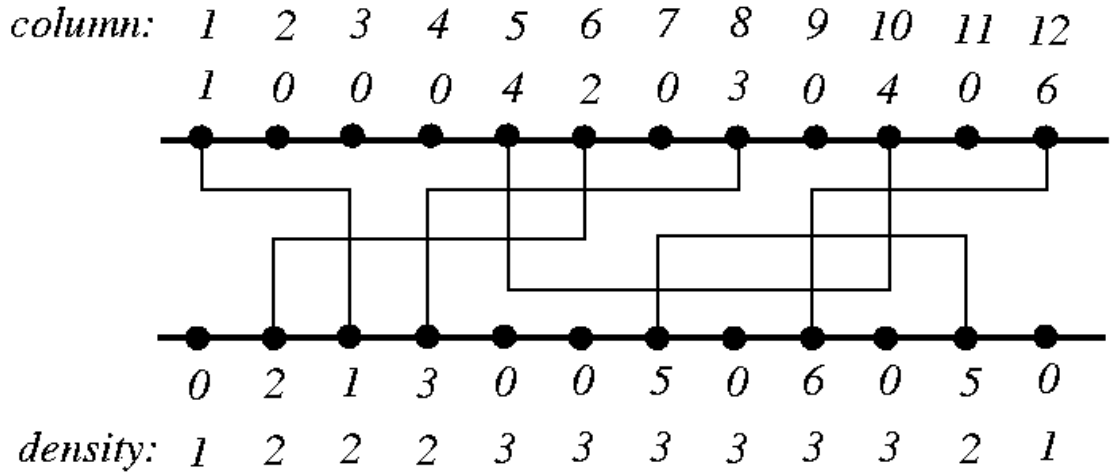
Basic Left-Edge Algorithm

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

```
1  begin
2  U ← {I_1, I_2, ..., I_n};
3  t ← 0;
4  while (U ≠ ∅) do
5      t ← t + 1;
6      watermark ← 0;
7      while (there is an I_j ∈ U s.t. s_j > watermark) do
8          Pick the interval I_j ∈ U with s_j > watermark,
           nearest watermark;
9          track[j] ← t;
10         watermark ← e_j;
11         U ← U − {I_j};
12 end
```

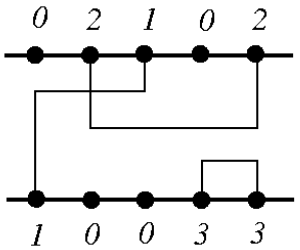
Example

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

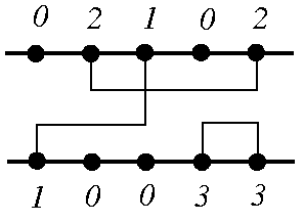


Basic Left-Edge Algorithm

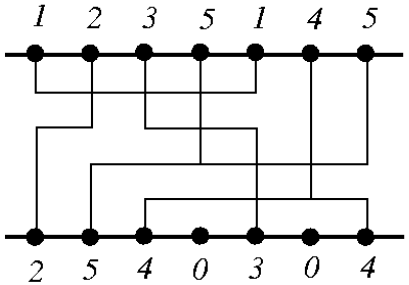
- If there is no vertical constraint, the basic left-edge algorithm is optimal.
- If there is any vertical constraint, the algorithm no longer guarantees optimal solution.

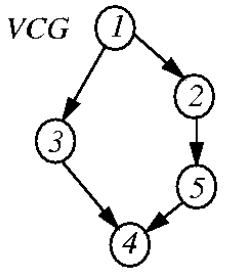


result from basic left–edge algorithm 3 tracks



optimal routing: 2 tracks





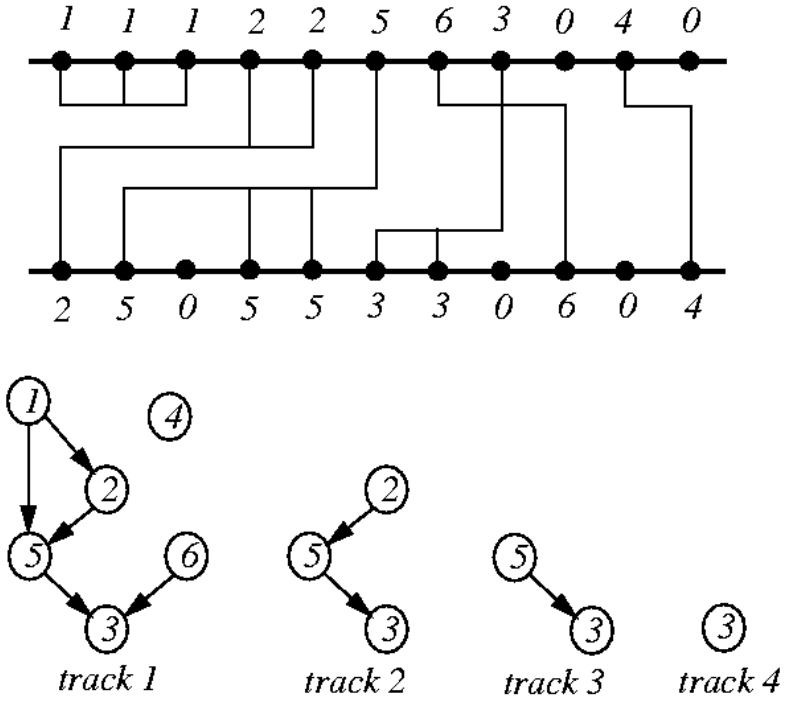
Constrained Left-Edge Algorithm

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

```
1  begin
2  U ← {I_1, I_2, ..., I_n};
3  t ← 0;
4  while (U ≠ ∅) do
5      t ← t + 1;
6      watermark ← 0;
7      while (there is an unconstrained I_j ∈ U s.t. s_j > watermark) do
8          Pick the interval I_j ∈ U that is unconstrained,
           with s_j > watermark, nearest watermark;
9          track[j] ← t;
10         watermark ← e_j;
11         U ← U − {I_j};
12 end
```

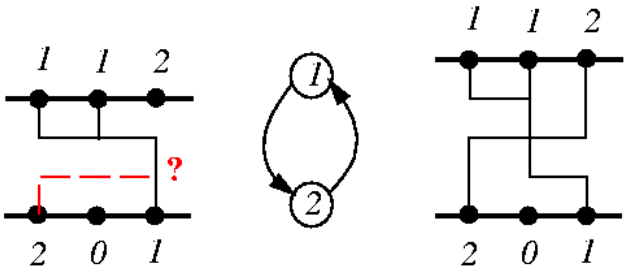
Constrained Left-Edge Example

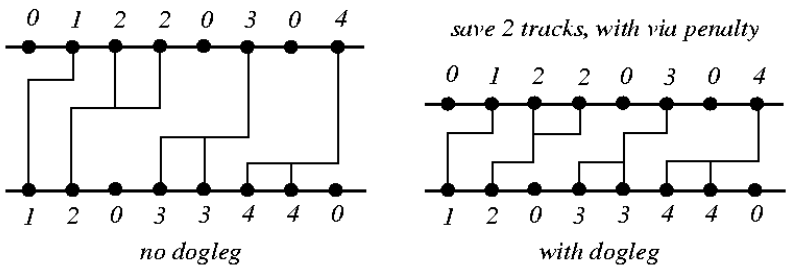
- I_1 = [1,3], I_2 = [1,5], I_3 =[6,8], I_4 = [10,11], I_5 = [2,6], I_6 = [7,9].
- Track 1: Route I_1 (cannot route I_3); Route I_6; Route I_4.
- Track 2: Route I_2; cannot route I_3.
- Track 3: Route I_5.
- Track 4: Route I_3.



Dogleg Channel Router

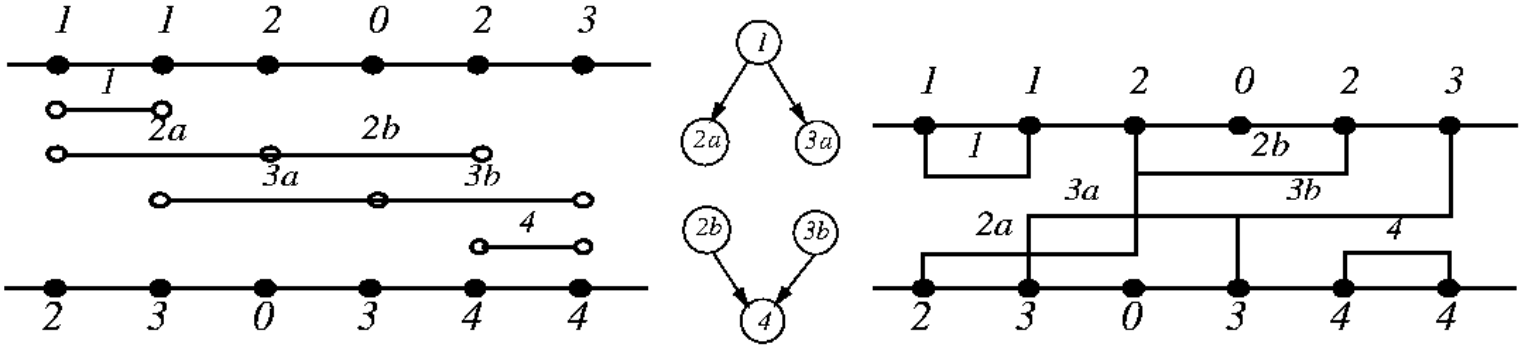
- Deutsch, “A dogleg channel router,” DAC, 1976.
- **Drawback of Left-Edge: cannot handle the cases with constraint cycles.**
  - **Doglegs** are used to resolve constraint cycle.
- **Drawback of Left-Edge: the entire net is on a single track.**
  - **Doglegs** are used to place parts of a net on different tracks to minimize channel height.
  - Might incur penalty for additional vias.





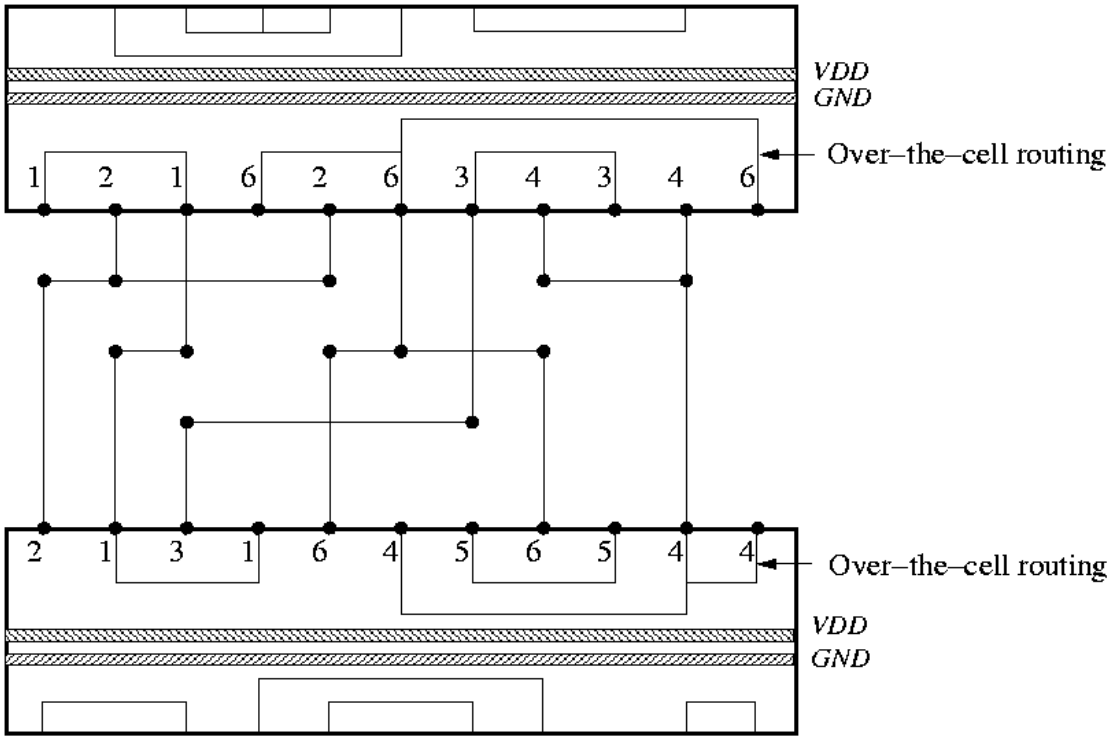
Dogleg Channel Router

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



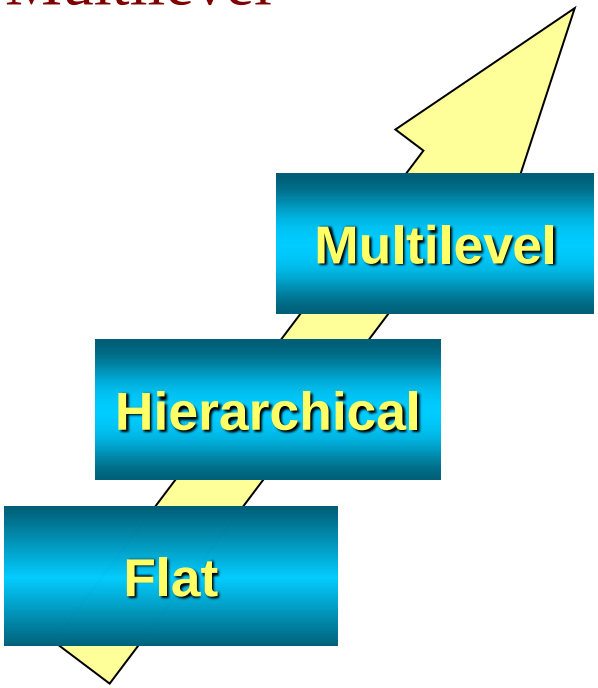
Over-the-Cell Routing

- Routing over the cell rows is possible due to the limited use of the 2nd (M2) mental layers within the cells.
- Divide the over-the-cell routing problem into 3 steps: (1) routing over the cell, (2) choosing the net segments, and (3) routing within the channel.



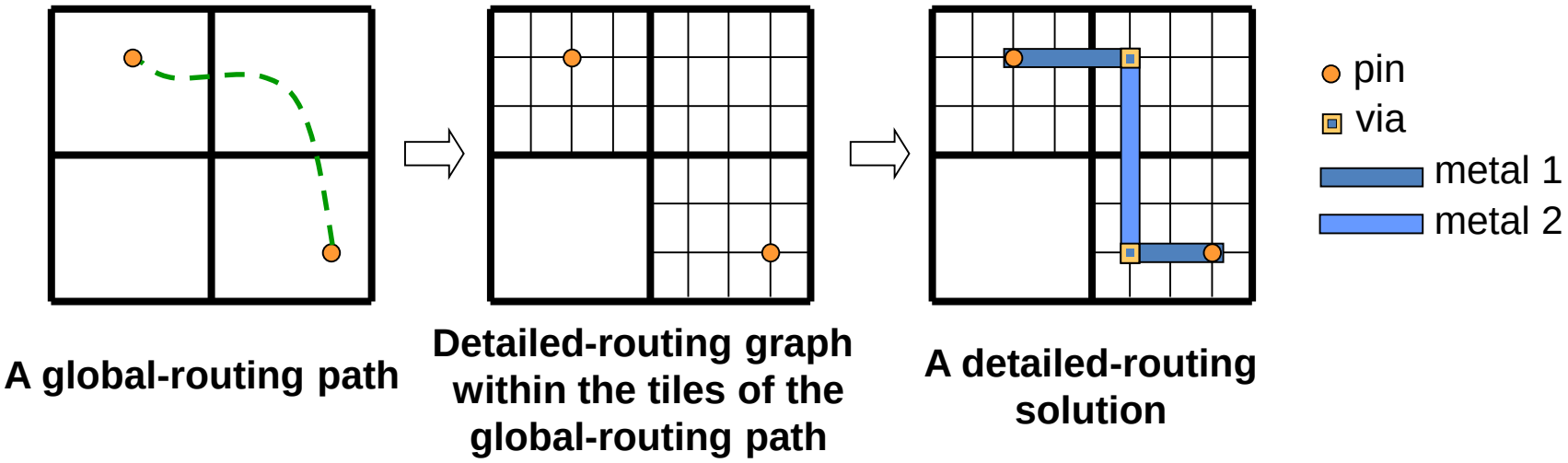
Routing Framework Evolution

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



Flat Routing Framework

- Global routing followed by detailed routing.
  - Maze searching, line searching, and/or A*-searching



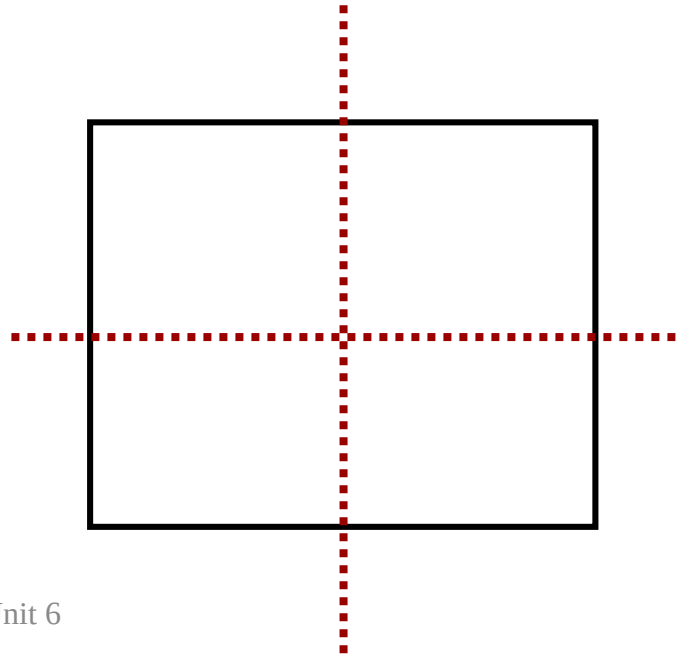
- Drawback: hard to handle larger problems

Hierarchical Routing Framework

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

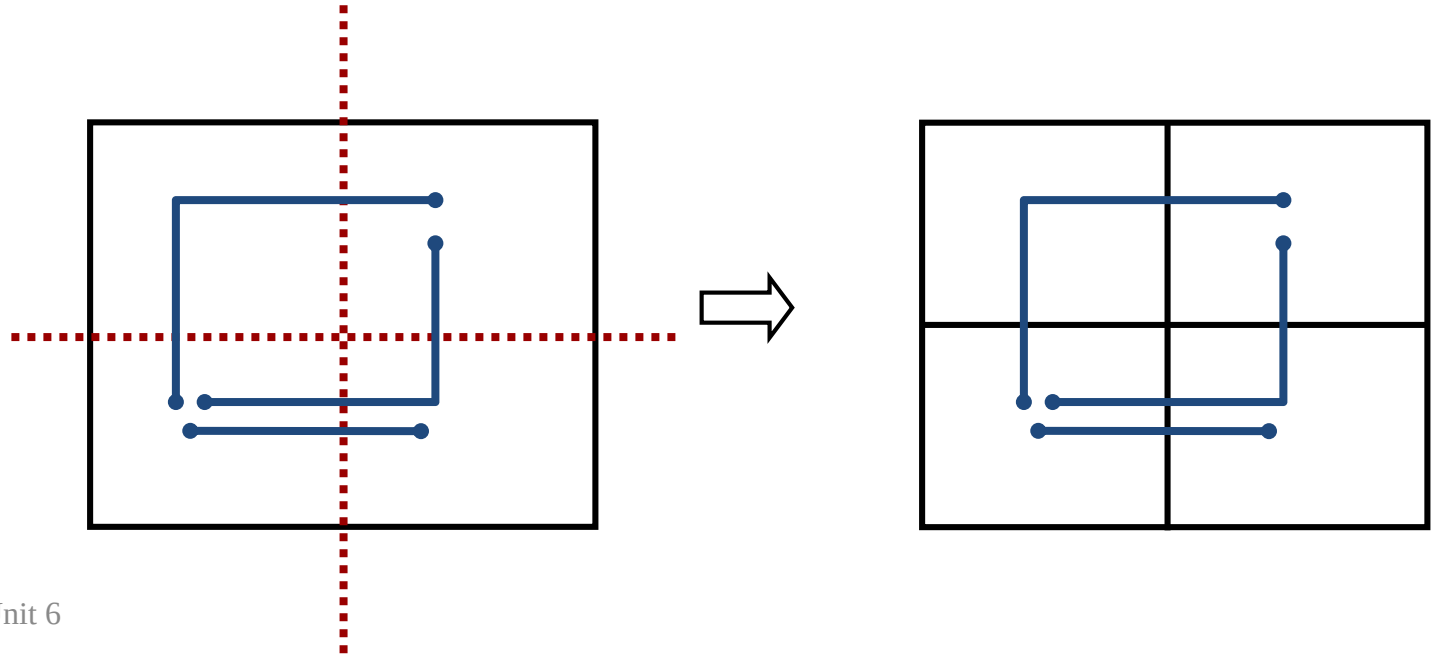
Hierarchical Routing Framework

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



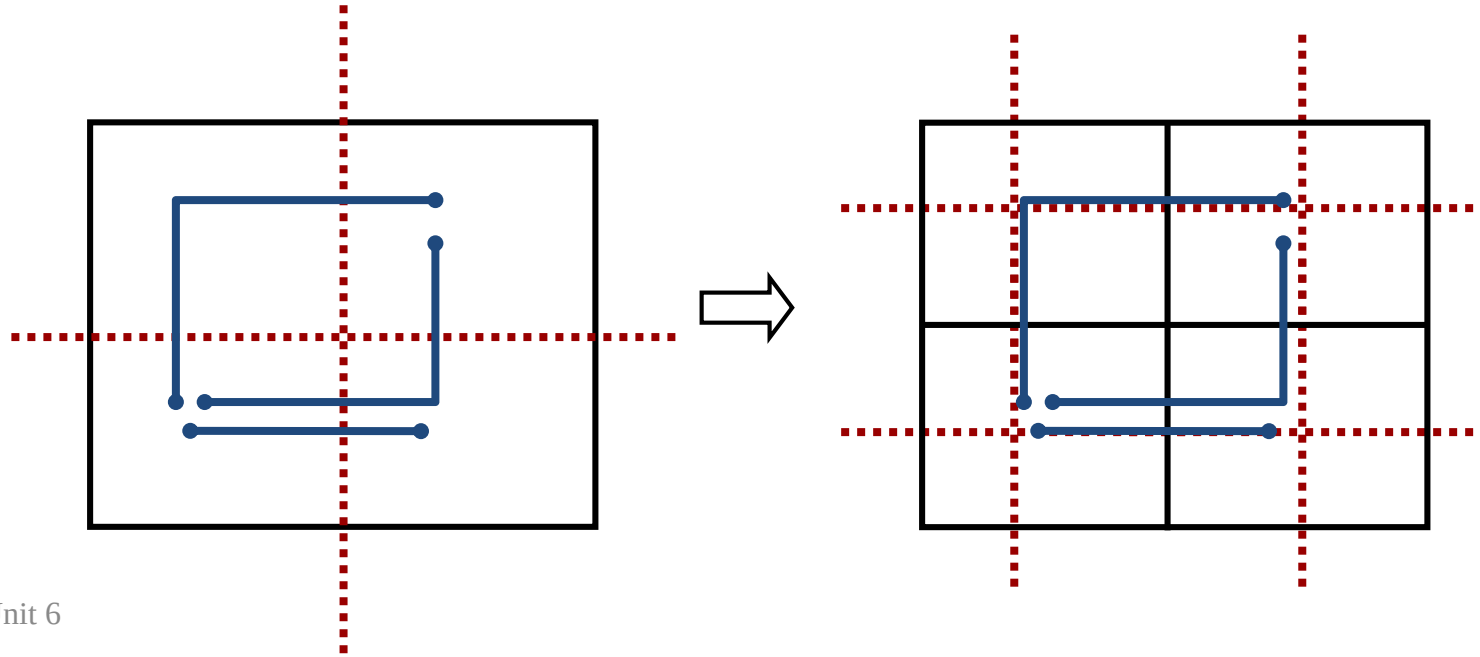
Hierarchical Routing Framework

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



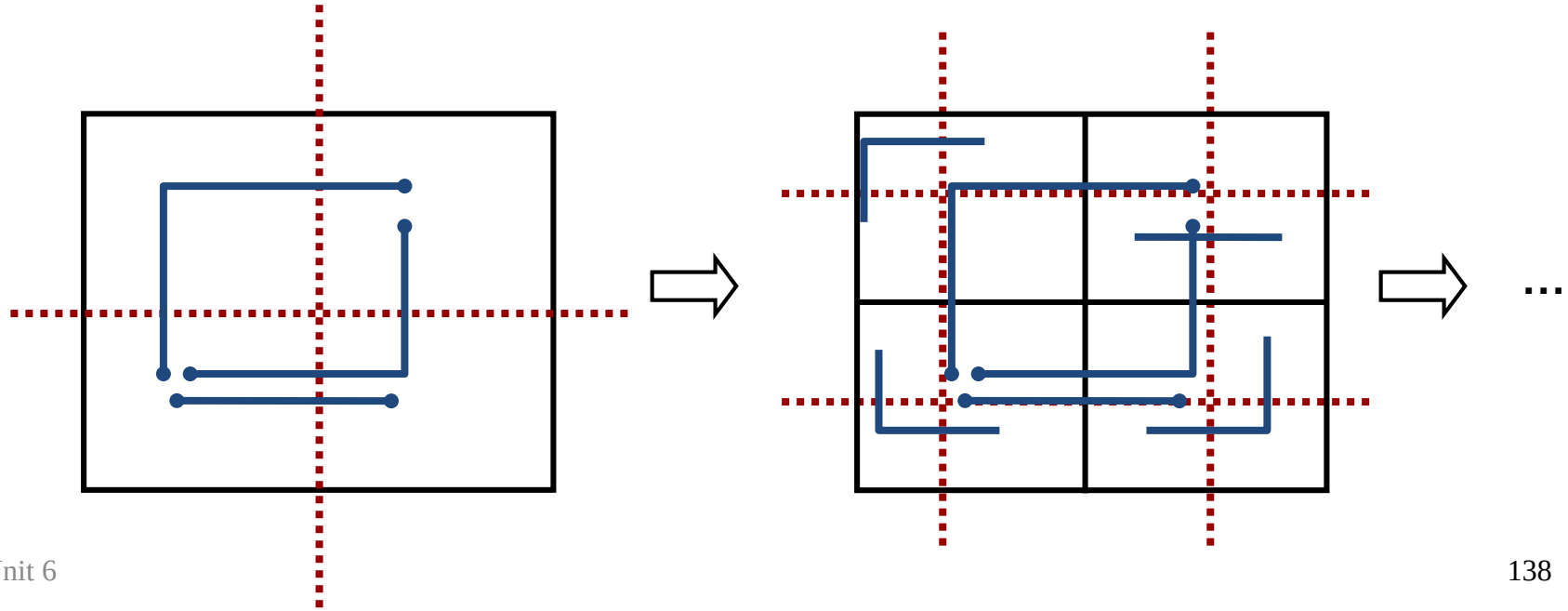
Hierarchical Routing Framework

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



Hierarchical Routing Framework

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



Multilevel Full-Chip Routing Framework

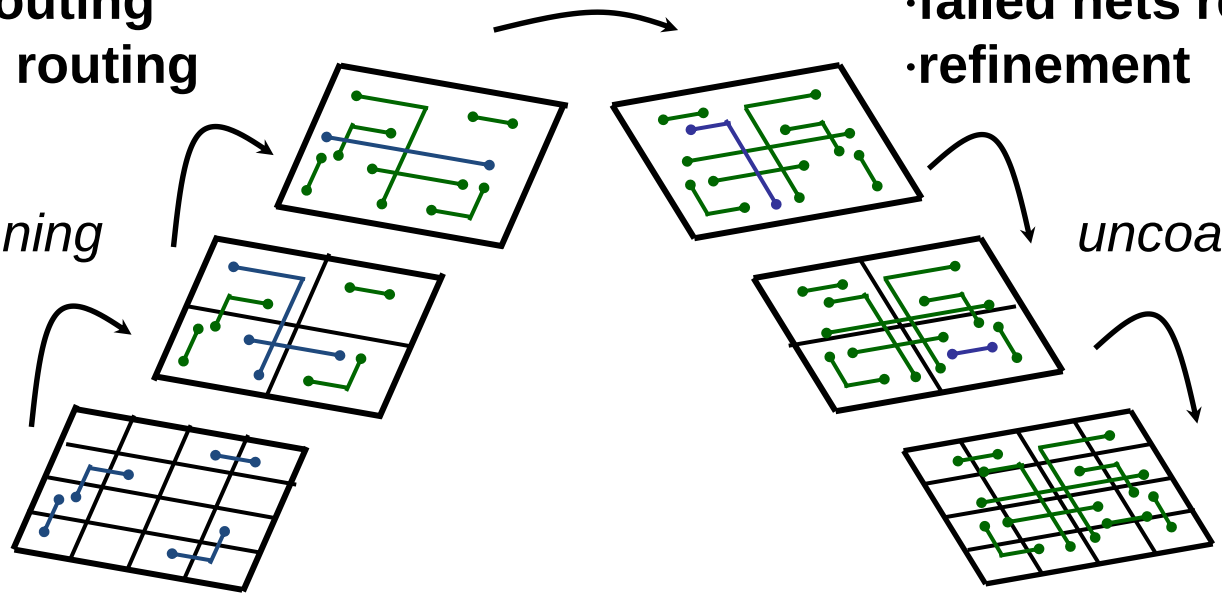
- Lin and Chang, "A novel framework for multilevel routing considering routability and performance," ICCAD-02 (TCAD-03).
- Multilevel framework: coarsening followed by uncoarsening.
- Coarsening (bottom-up) stage:
  - Constructs the net topology based on the minimum spanning tree.
  - Processes routing tiles one by one at each level, and only local nets (connections) are routed.
  - Applies two-stage routing of global routing followed by detailed routing.
  - Uses the L-shaped & Z-shaped pattern routing.
  - Performs resource estimation after detailed routing to guide the routing at the next level.
- Uncoarsening (top-down) stage
  - Completes the failed nets (connections) from the coarsening stage.
  - Uses a global and a detailed maze routers to refine the solution.

Λ-Shaped Multilevel Routing Framework

- - - - - To-be-routed net ——— Already-routed net

·**global routing**
·**detailed routing**

·**failed nets rerouting**
·**refinement**



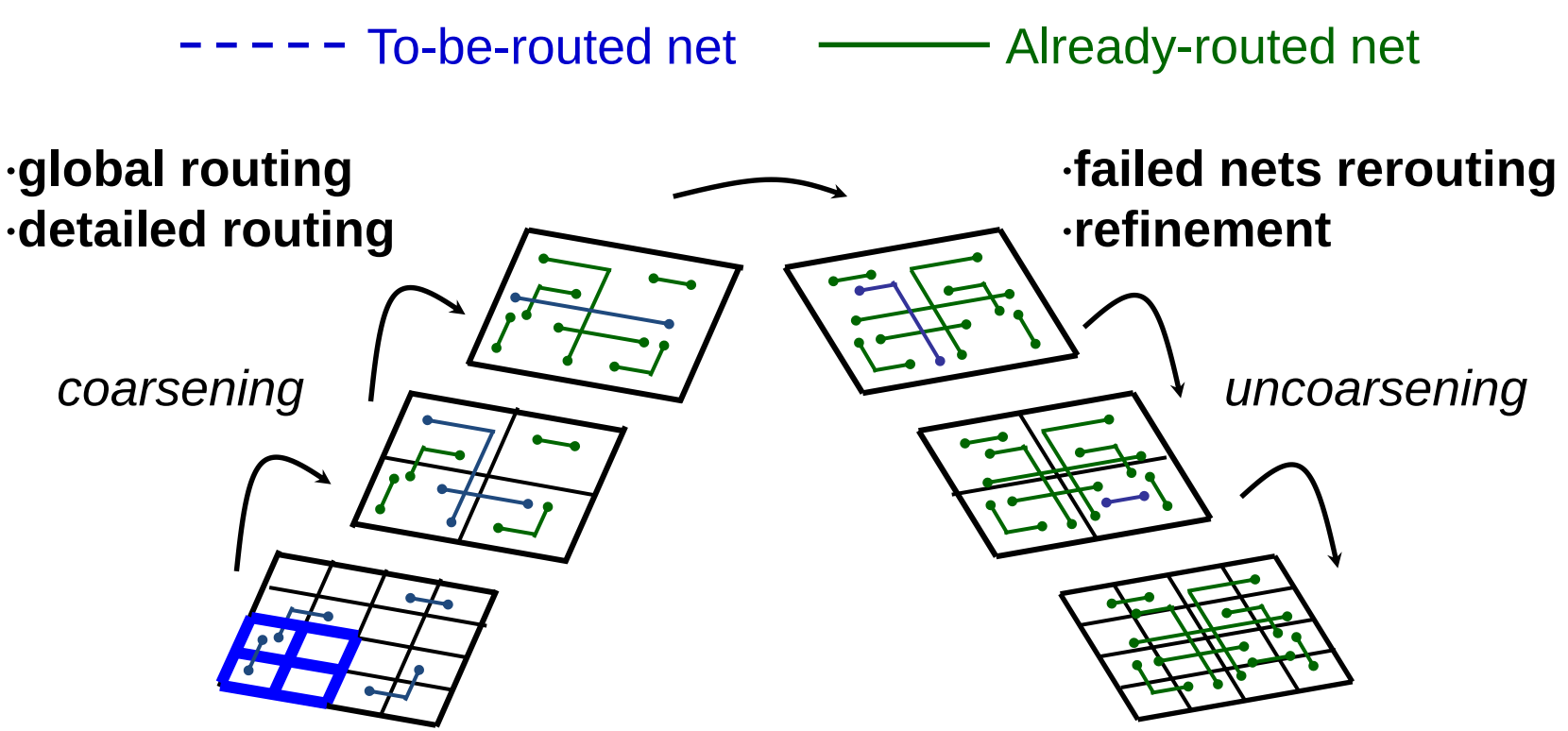
Perform global pattern & detailed routing for local connections and then estimate routing congestion for the next level

Cost: congestion

Use maze routing to reroute failed connections and refine the solution

Cost: congestion + net length

Λ-Shaped Multilevel Routing Framework



Perform global pattern & detailed routing for local connections and then estimate routing congestion for the next level

Cost: congestion

Use maze routing to reroute failed connections and refine the solution

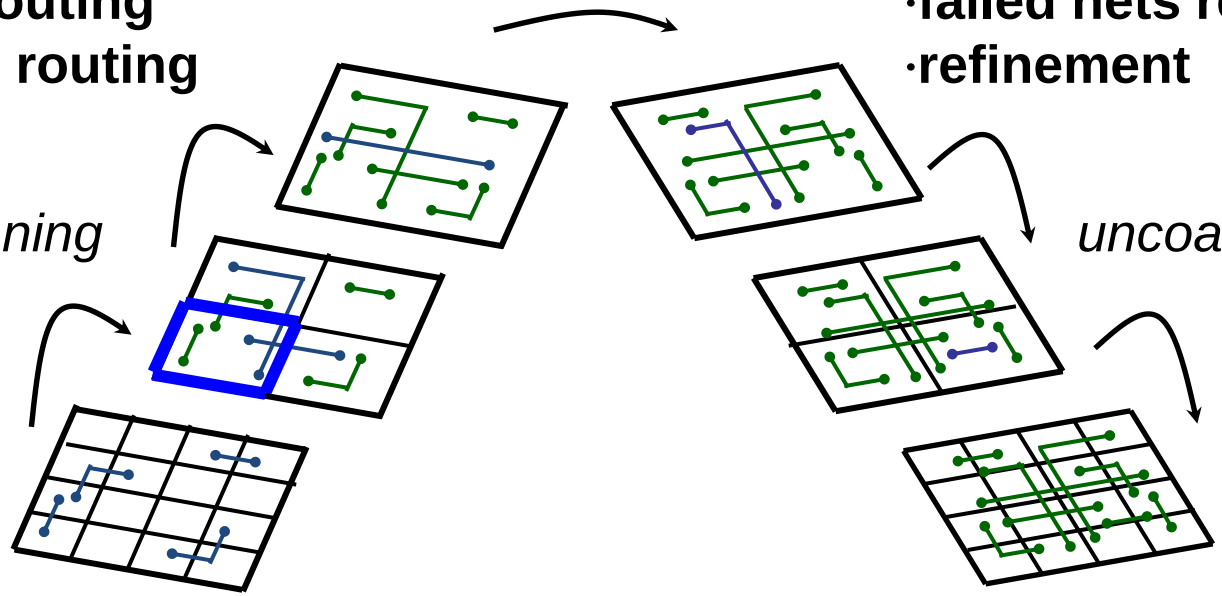
Cost: congestion + net length

Λ-Shaped Multilevel Routing Framework

- - - - - To-be-routed net ——— Already-routed net

·**global routing**
·**detailed routing**

·**failed nets rerouting**
·**refinement**



Perform global pattern & detailed routing for local connections and then estimate routing congestion for the next level

Cost: congestion

Use maze routing to reroute failed connections and refine the solution

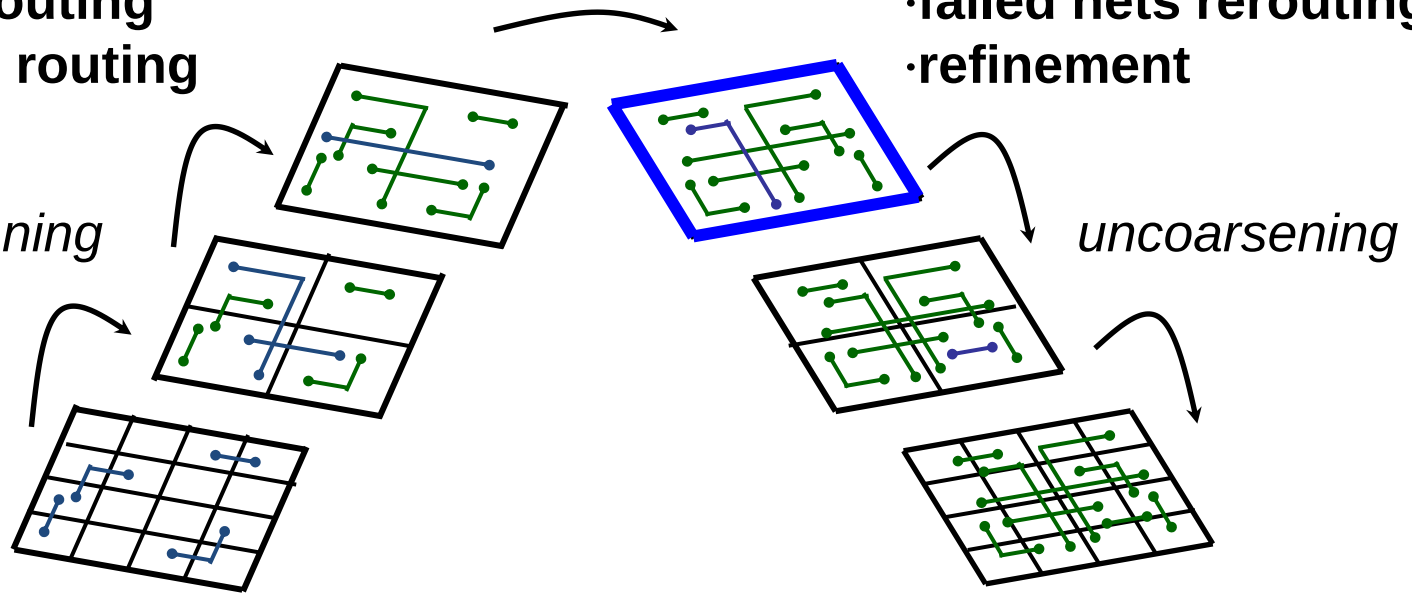
Cost: congestion + net length

Λ-Shaped Multilevel Routing Framework

- - - - - To-be-routed net ——— Already-routed net

·**global routing**
·**detailed routing**

·**failed nets rerouting**
·**refinement**



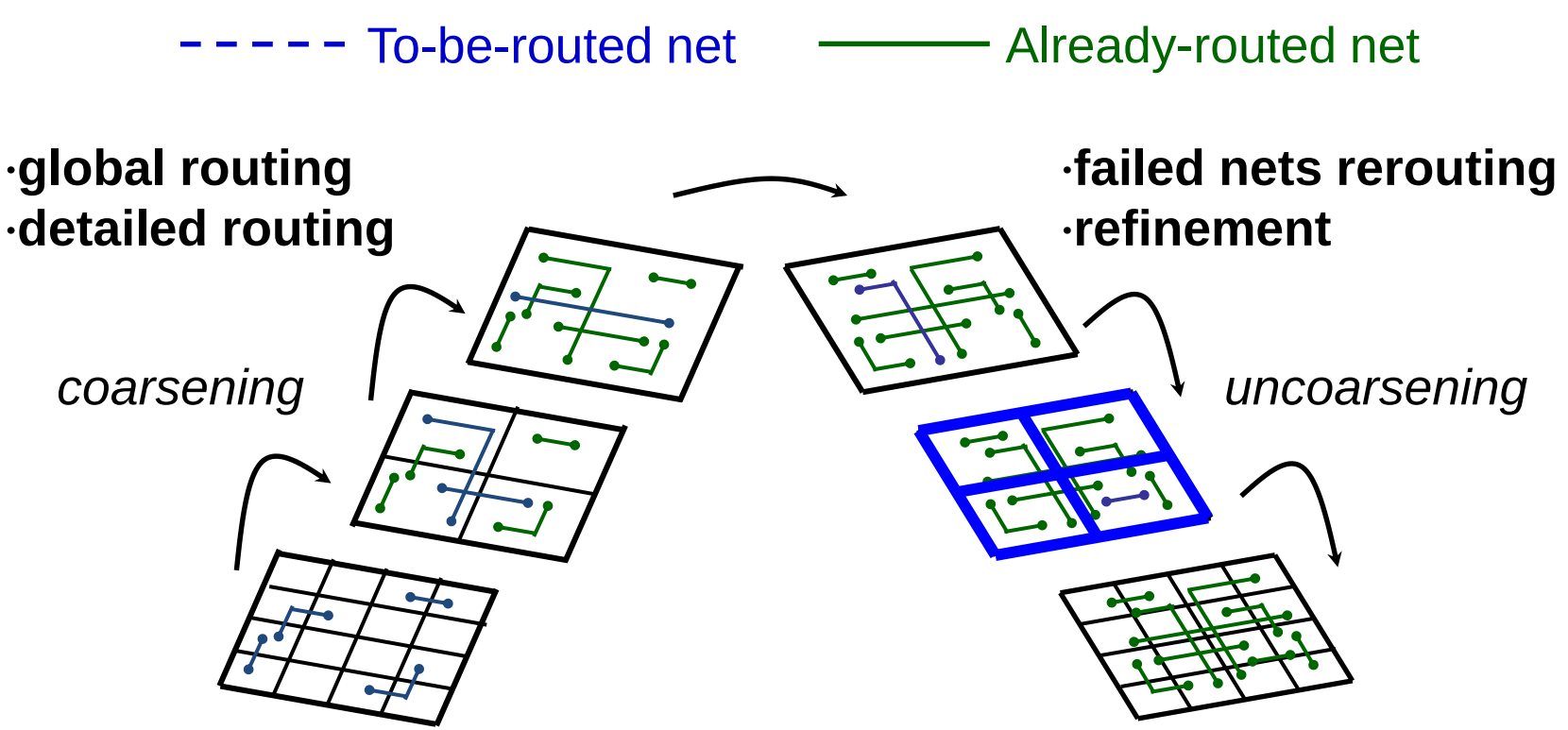
Perform global pattern & detailed routing for local connections and then estimate routing congestion for the next level

Cost: congestion

Use maze routing to reroute failed connections and refine the solution

Cost: congestion + net length

Λ-Shaped Multilevel Routing Framework



Perform global pattern & detailed routing for local connections and then estimate routing congestion for the next level

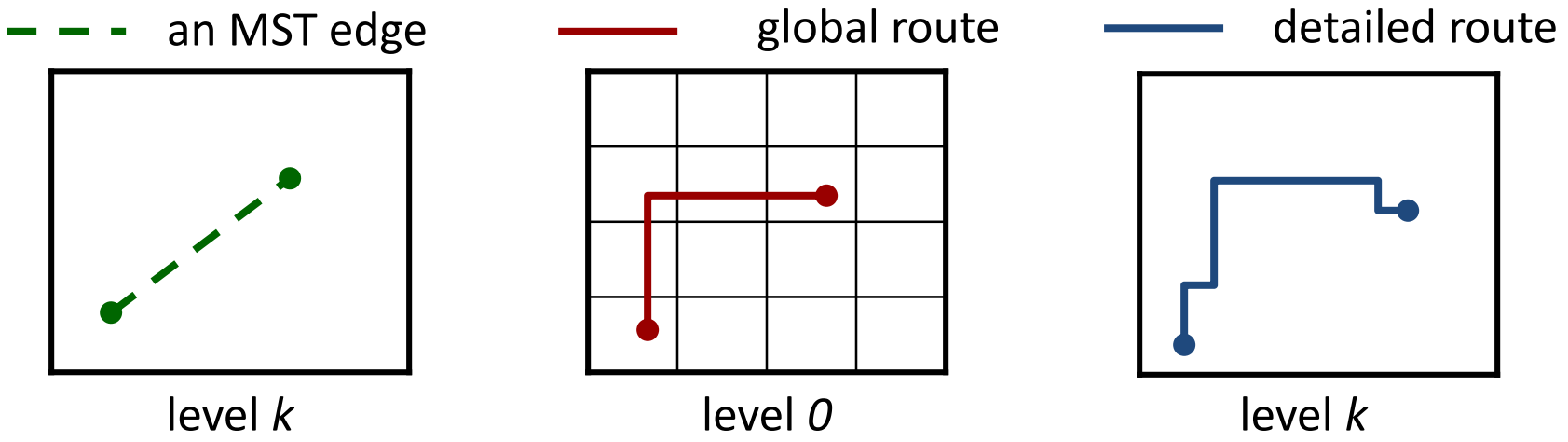
Cost: congestion

Use maze routing to reroute failed connections and refine the solution

Cost: congestion + net length

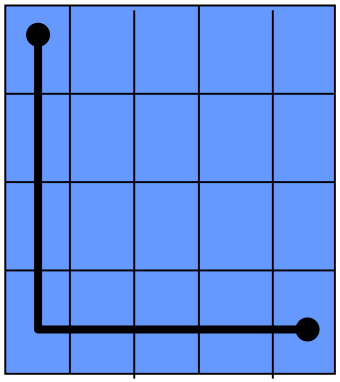
Coarsening Stage

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

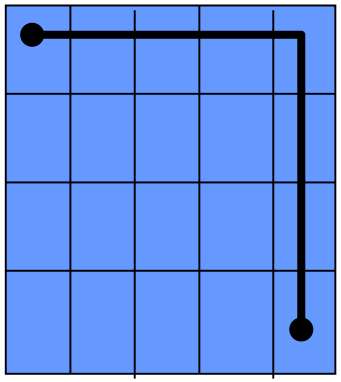


Global Routing

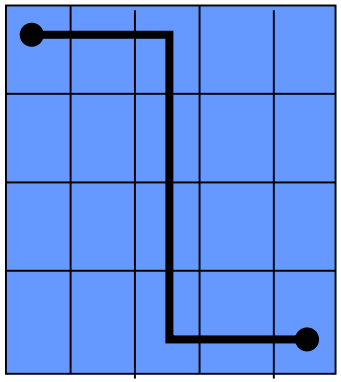
- Apply pattern routing for global routing
  - Use L-shaped and Z-shaped connections to route nets.
  - Has lower time complexity than maze routing.



Lower L-Shaped connection



Upper L-Shaped connection



Z-Shaped connection

Cost Function in Global Pattern Routing

$G_i = (V_i , E_i)$: multilevel routing graph at level i.

Define $R_e = \{e \in E_i \mid e$ is the edge chosen to route$\}$

Then

$$\text{cost}(R_e) = \sum_{e \in E} C_e,$$

where C_e is the congestion of edge e

and

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

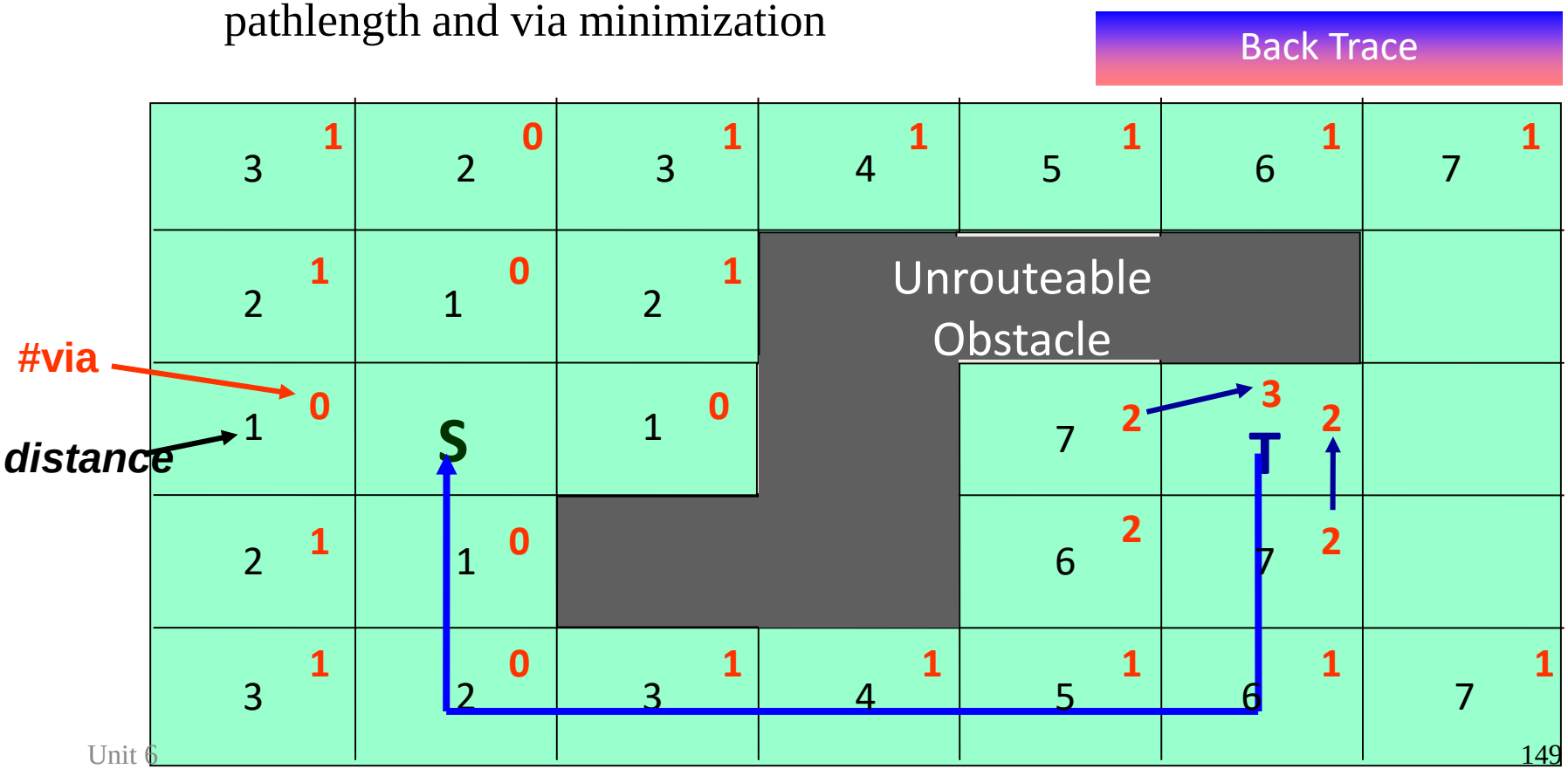
where p_e and d_e are the capacity and density associated with e, respectively.

Detailed Routing

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

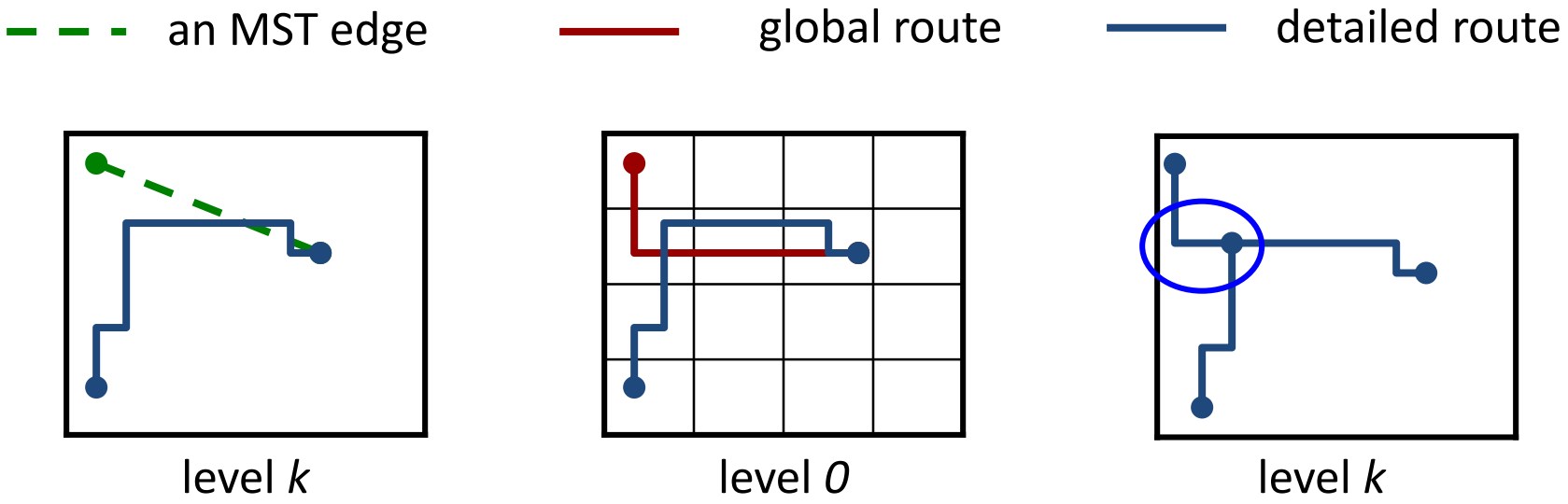
Via Minimization

- Simultaneous pathlength and via minimization (SPVM)
  - perform modified maze routing that simultaneously considers the pathlength and via minimization



Local Refinement

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



Resource Estimation

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

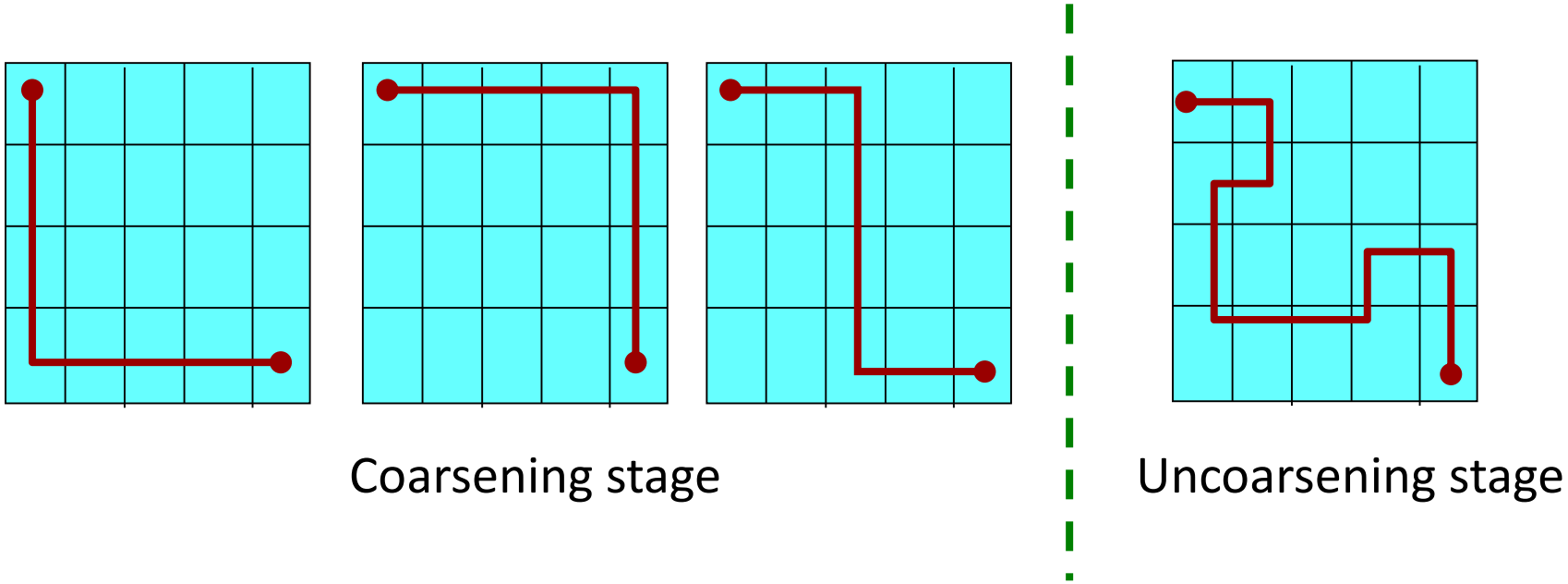
$$C_e = \frac{1}{2^{(p_e - d_e)}},$$

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

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

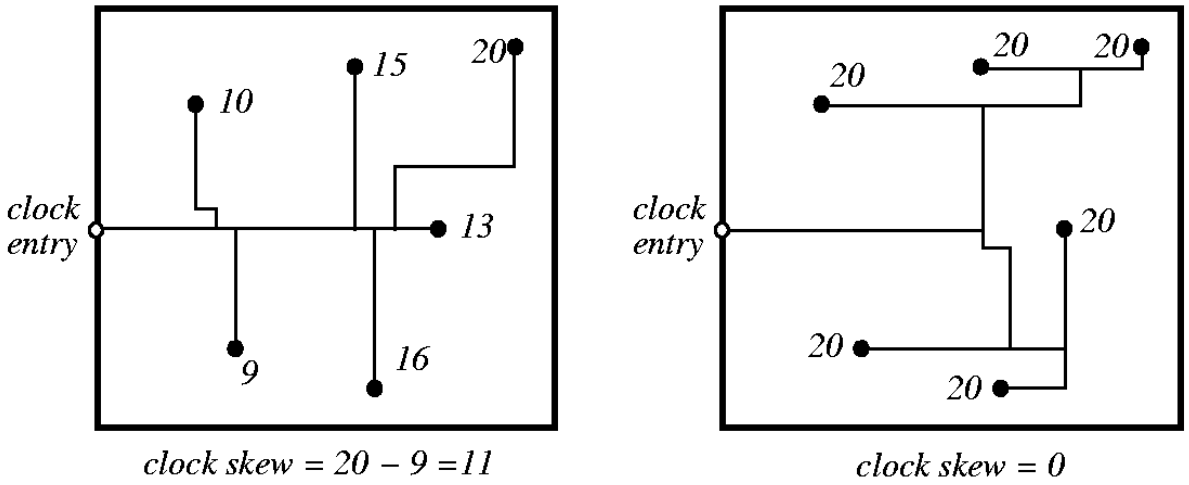
Uncoarsening Global Routing

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



Clock Routing

- **Clock skew** is defined as the difference in the minimum and the maximum arrival time of the clock.



- Route clock nets such that
  - Clock signals arrive simultaneously
  - Clock delay is minimized

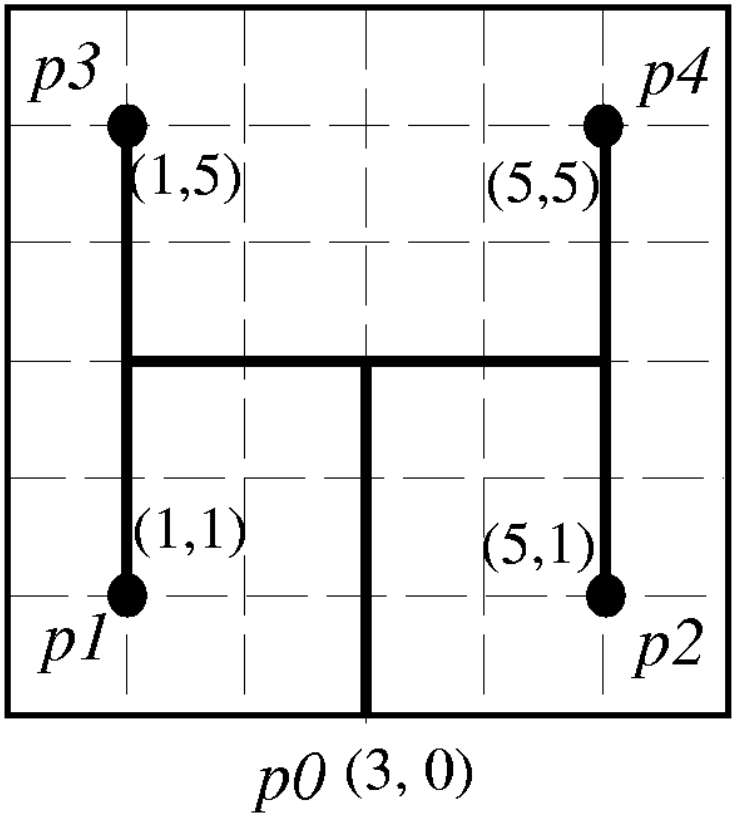
    * Other issues: total wirelength, power consumption.

Clock Routing Problem

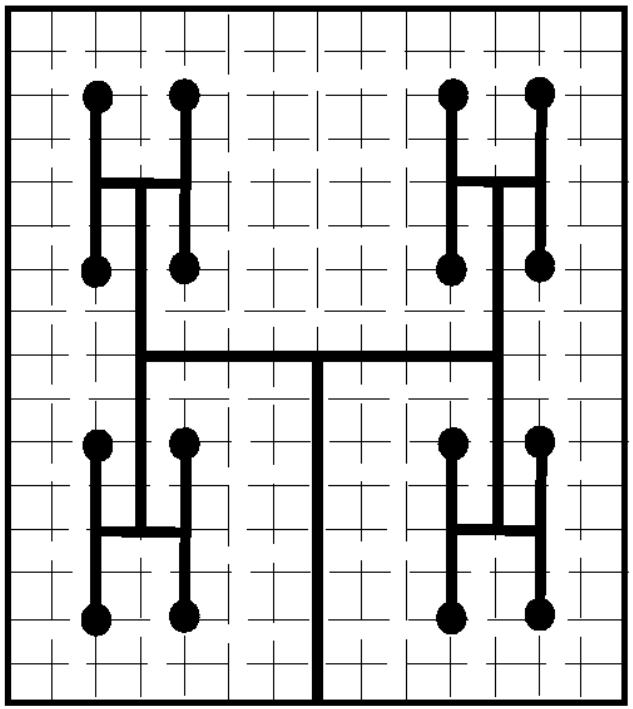
- Given the routing plane and a set of points $P=\{p_1,p_2,\ldots,p_n\}$ within the plane and clock entry point p_0 on the boundary of the plane, the **Clock Routing Problem (CRP)** is to interconnect each $p_i \in P$ such that $\max_{i,j\in P}|t(0,i)-t(0,j)|$ (skew) and $\max_{i\in P} t(0,i)$ (delay) are both minimized.
- Pathlength-based approaches
  - H-tree: Dhar, Franklin & Wang, *ICCD*, 1984.
  - Methods of means & medians (MMM): Jackson, Srinivasan & Kuh, *DAC*, 1990.
  - Geometric matching: Kahng, Cong & Robins, *IEEE TCAD*, 1993.
- RC-delay based approaches:
  - Exact zero skew: Tasy, *ICCAD*, 1991.
  - Deferred merge embedding (DME) algorithm: Boese & Kahng, ASICON-92; Chao, Hsu & Ho, DAC-92; Edahiro, NEC R&D, 1991.

H-Tree Based Algorithm

- H-tree: Dhar, Franklin, Wang, "Reduction of clock delays in VLSI structure," *ICCD*, 1984.



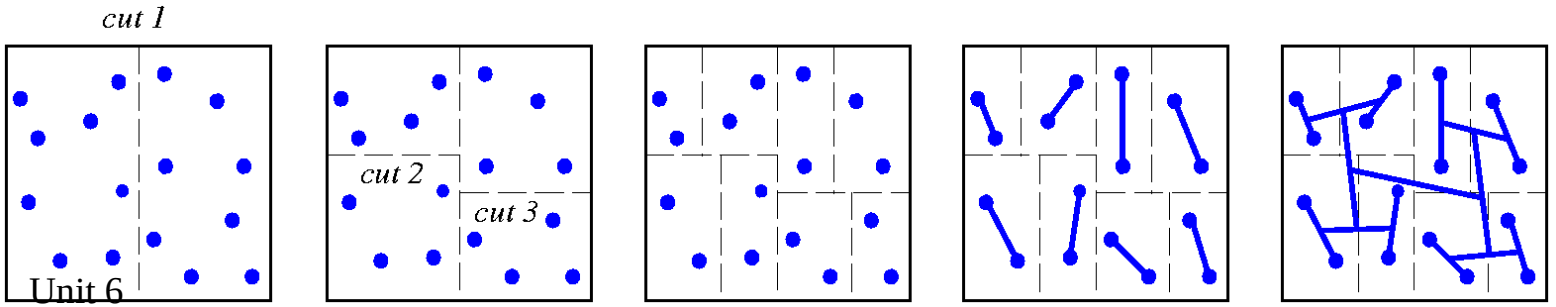
H−tree over 4 points



H−tree over 16 points

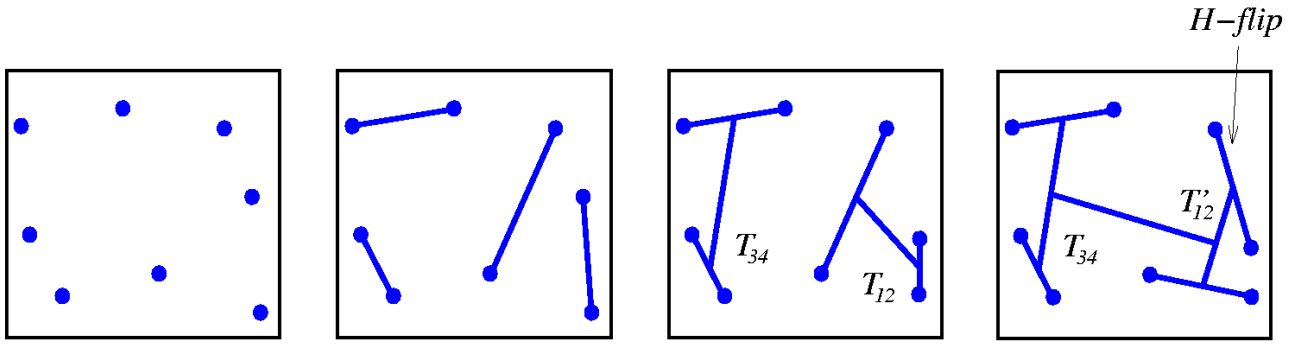
The MMM Algorithm

- Jackson, Srinivasan, Kuh, "Clock routing for high-performance ICs," *DAC*, 1990.
- Each clock pin is represented as a point in the region, S.
- The region is partitioned into two subregions, S_L and S_R.
- The center of mass is computed for each subregion.
- The center of mass of the region S is connected to each of the centers of mass of subregion S_L and S_R.
- The subregions S_L and S_R are then recursively split in Y-direction.
- The above steps are repeated with alternate splitting in X- and Y-direction.
- Time complexity: $O(n \log n)$.



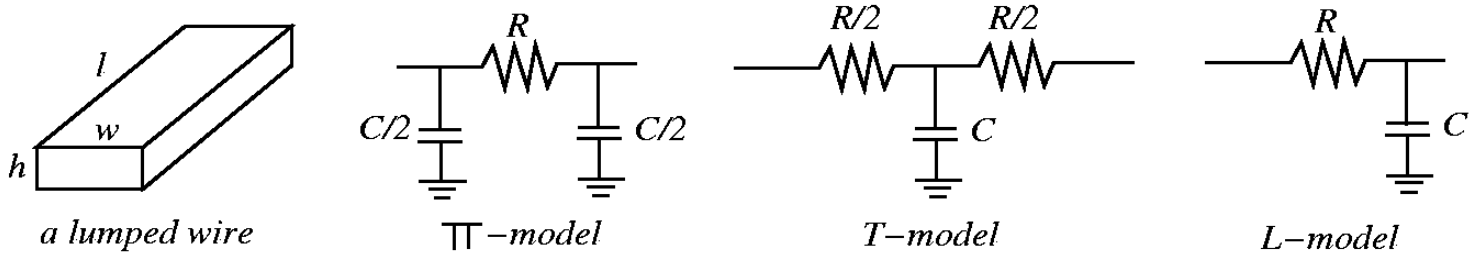
The Geometric Matching Algorithm

- Kahng, Cong, Robins, "Matching based models for high-performance clock routing," *IEEE TCAD*, 1993.
- Clock pins are represented as n nodes in the clock tree ($n=2^k$).
- Each node is a tree itself with clock entry point being node itself.
- The minimum cost matching on n points yields $n/2$ segments.
- The clock entry point in each sebtree of two nodes is the point on the segment such that length of both sides is the same.
- The above steps are repeated for each segment.
- Apply H-flipping to further reduce clock skew (and to handle edges intersection).

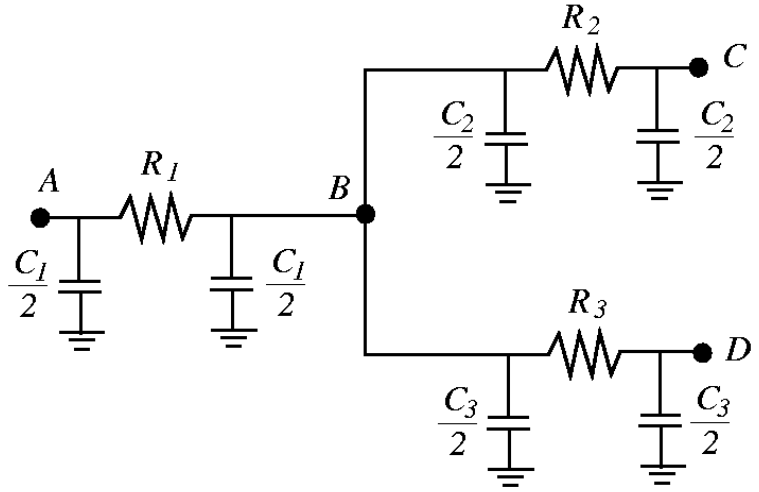


Delay Calculation

- Need to consider a more accurate delay model for general circuits.
- **RC Delay:** The delay caused by wires is due to their capacitance and resistance. $(R \propto \frac{l}{wh};\ C \propto wl)$
- Lumped circuit approximations for distributed RC lines: π-model, *T*-model, *L*-model.

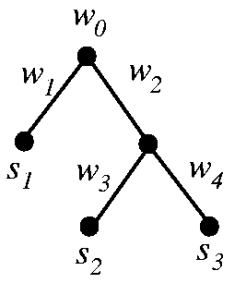


- RC delay: $D_{AB} = R_1(\frac{C_1}{2} + C_2 + C_3);\ B\ to\ C:\ D_{BC} = R_2(\frac{C_2}{2});\ B\ to\ D:\ D_{BD} = R_3(\frac{C_3}{2})$

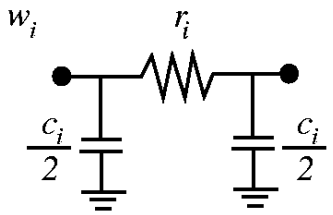


Delay Calculation for a Clock Tree

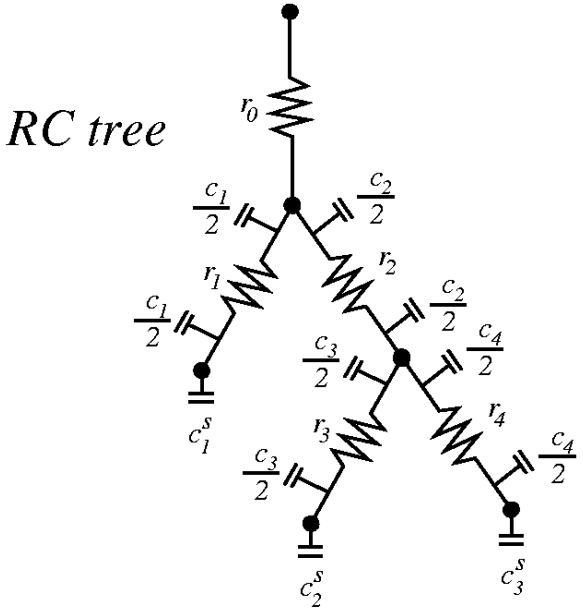
$$t_{03} = r_0(c_1 + c_2 + c_3 + c_4 + c_1^s + c_2^s + c_3^s) + r_2(\frac{c_2}{2} + c_3 + c_4 + c_2^s + c_3^s) + r_4(\frac{c_4}{2} + c_3^s).$$



clock tree



delay model



Exact Zero Skew Algorithm

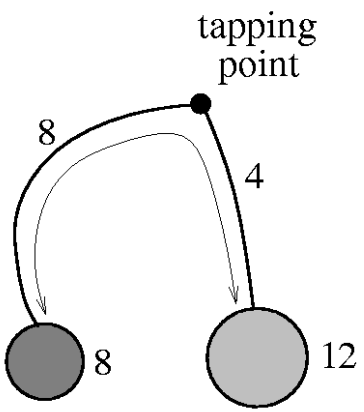
- Tsay, "Exact zero skew algorithm," *ICCAD*, 1991.
- To ensure the delay from the **tapping point** to leaf nodes of subtrees T_1 and T_2 being equal, it requires that

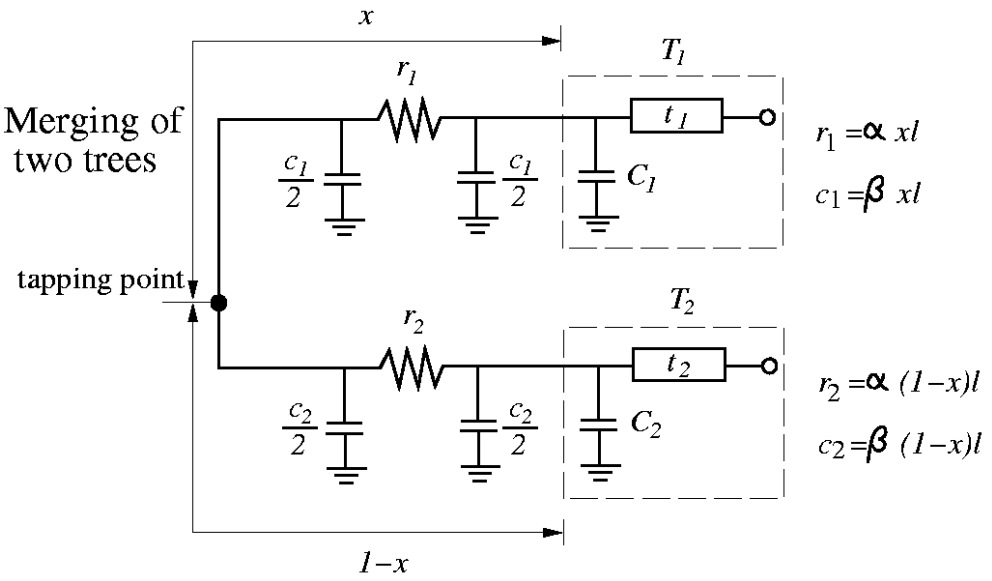
$$r_1(\frac{c_1}{2}+C_1)+t_1 = r_2(\frac{c_2}{2}+C_2)+t_2$$

- Solving the above equation, we have

$$x = \frac{(t_2-t_1)+\alpha l(C_2+\frac{\beta l}{2})}{\alpha l(\beta l+C_1+C_2)}$$

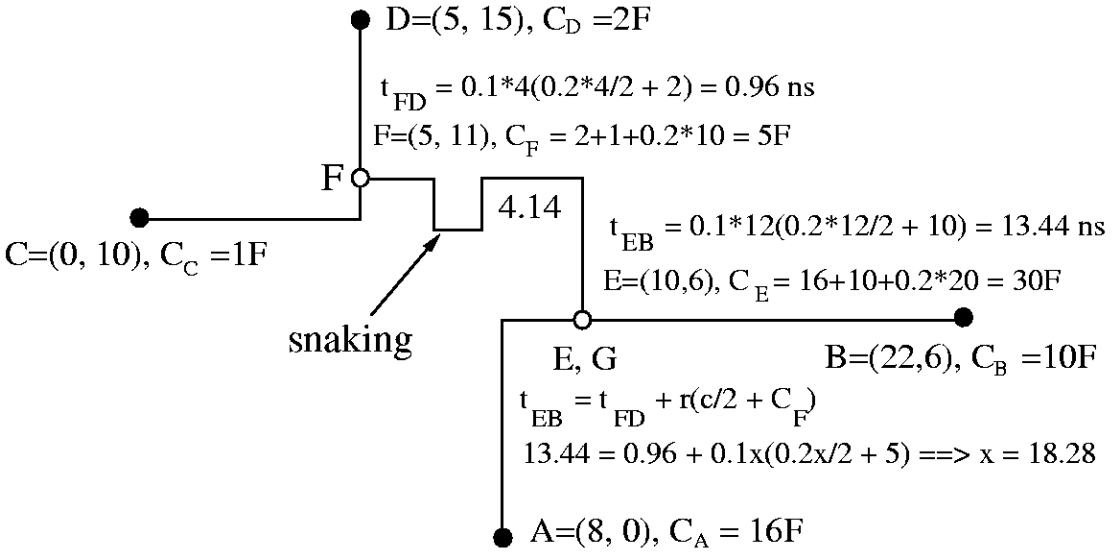
- α and β are the per unit values of resistance and capacitance, l the length of the interconnecting wire, $r_1=\alpha x l$, $c_1=\beta x l$, $r_2=\alpha(1-x)l$, $c_2=\beta(1-x)l$.





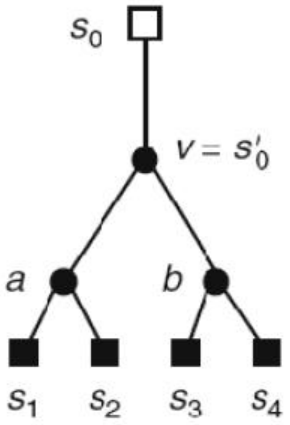
Zero-Skew Computation

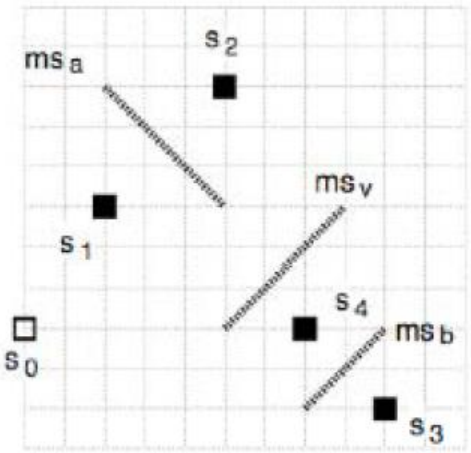
- **Balance delays:** $r_1(\frac{c_1}{2}+C_1)+t_1 = r_2(\frac{c_2}{2}+C_2)+t_2.$
- **Compute tapping points:** $x = \frac{(t_2-t_1)+\alpha l(C_2+\frac{\beta l}{2})}{\alpha l(\beta l + C_1 + C_2)}$, α(β): per unit values of resistance (capacitance); *l*: length of the wire; $r_1=\alpha x l$, $c_1=\beta x l$; $r_2 = \alpha(1-x)l$, $c_2=\beta(1-x)l$.
- If $x \notin [0,1]$, we need **snaking** to find the tapping point.
- Exp: α=0.1Ω/*unit*, β=0.2*F*/*unit*. (Find tapping points *E* for *A* and *B*, *F* for *C* and *D*, and *G* for *E* and *F*.)

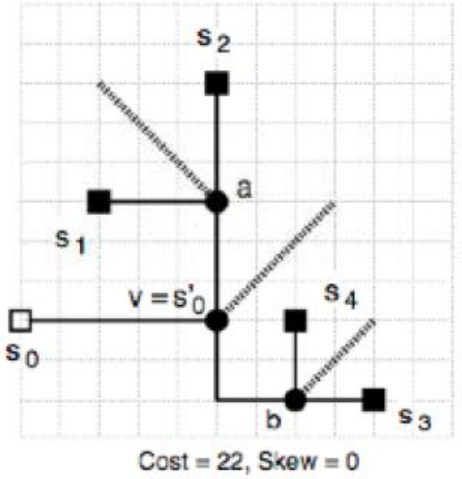


Deferred Merge Embedding (DME)

- Boese & Kahng, ASICON-92; Chao, Hsu, & Ho, DAC-92; Edahiro, NEC R&D, 1991
- Two stages
  - Bottom up: build a **segment** for potential taping points
  - Top down: determine exact tapping points

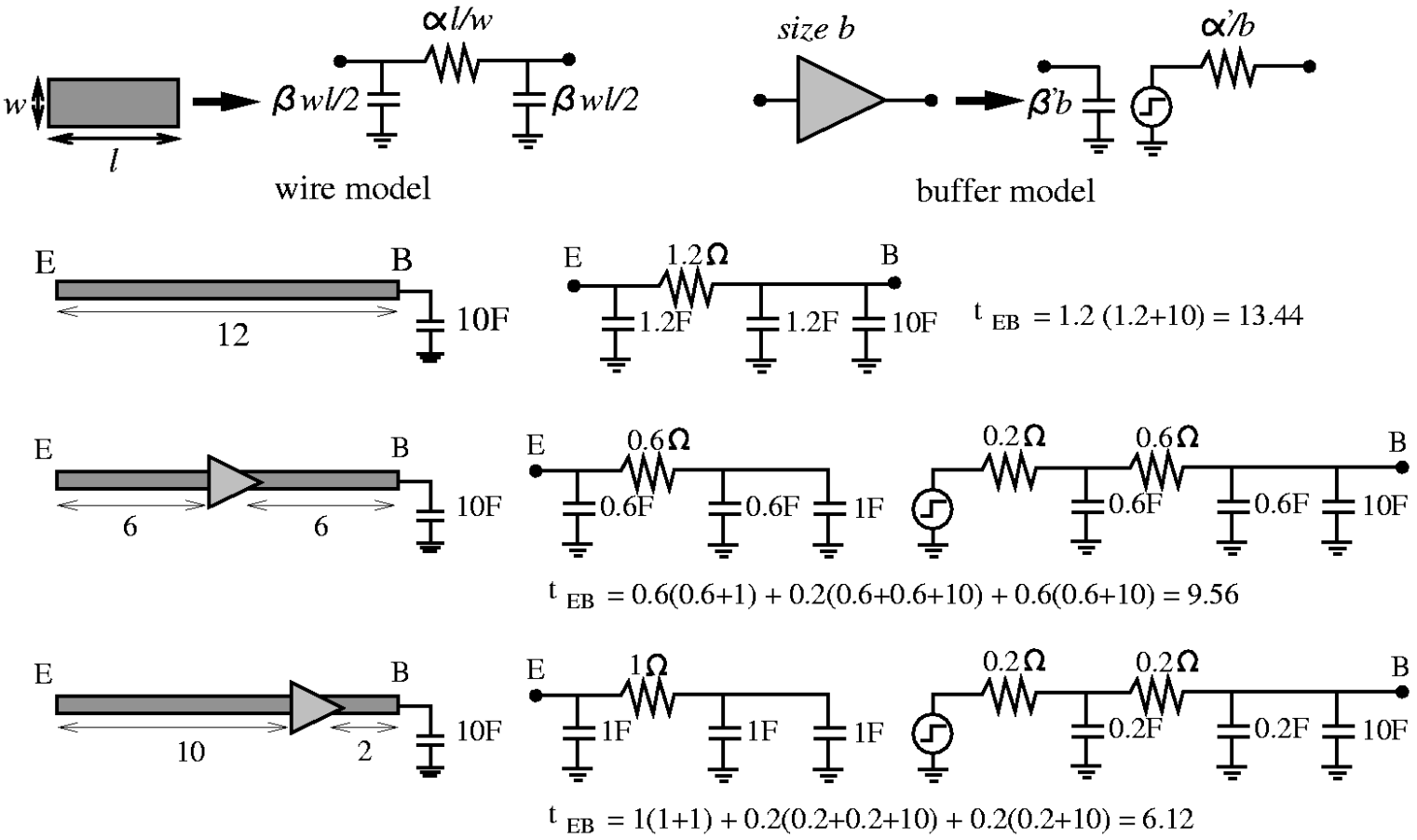






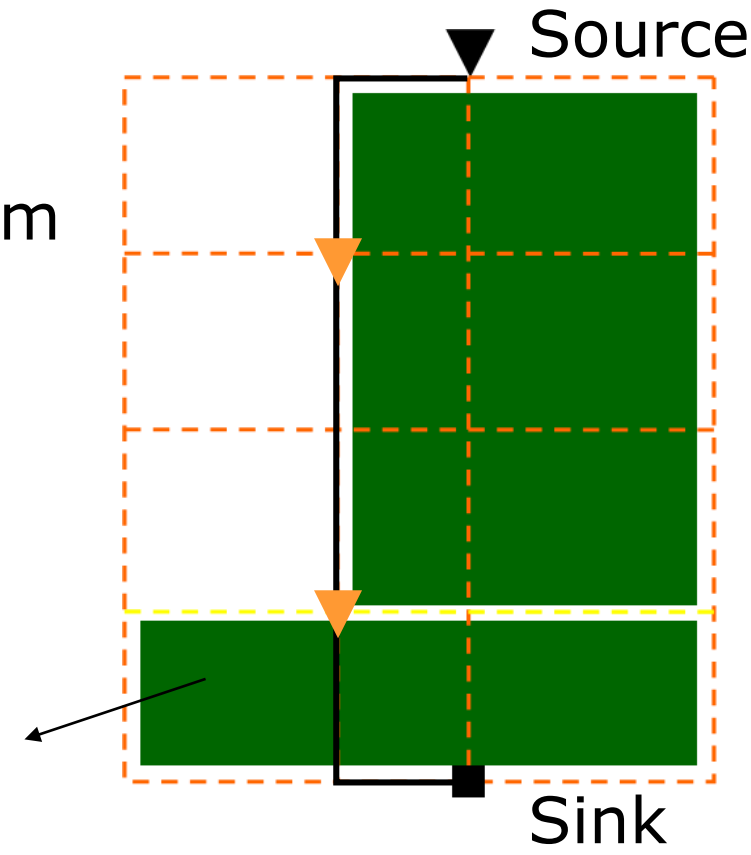
Delay Computation for Buffered Wires

- Wire: $\alpha=0.1\Omega$ /*unit* size, $\beta=0.2F$/*unit* size; buffer: $\alpha'=0.2\Omega$/*unit* size, $\beta'=1F$/*unit* size; unit-sized wire and buffer.



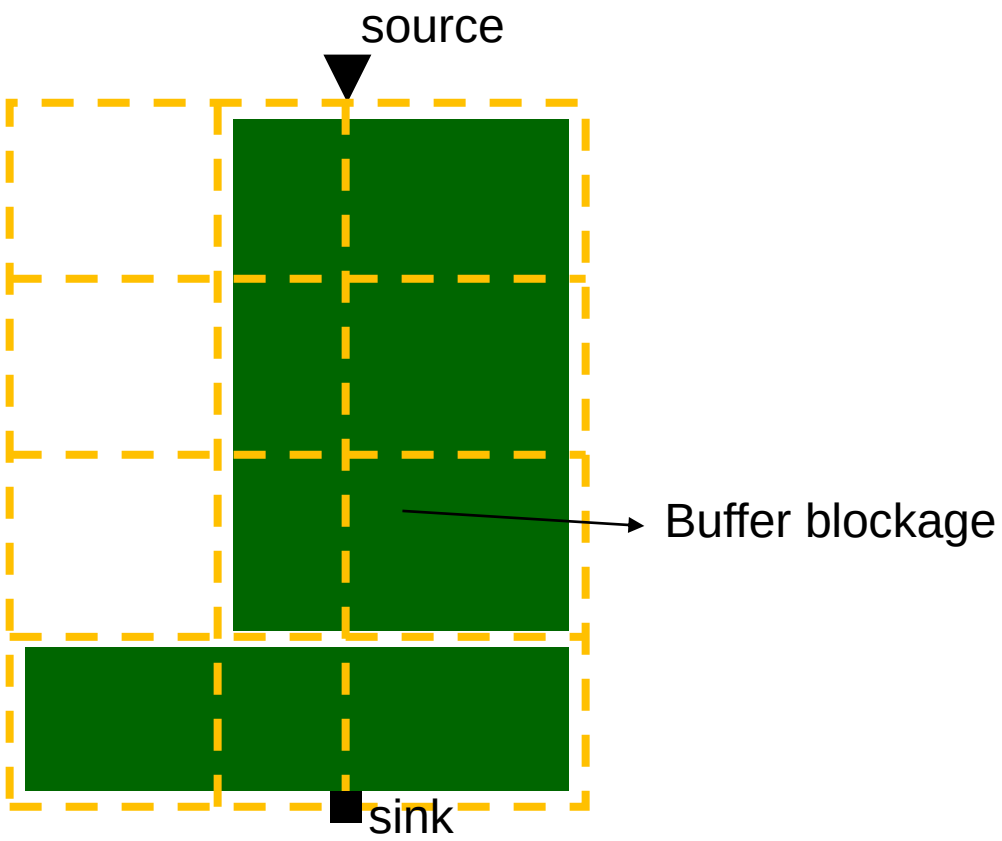
Buffered Path Construction

Goal: Find a minimum-delay buffered path from source to sink with blockage avoidance

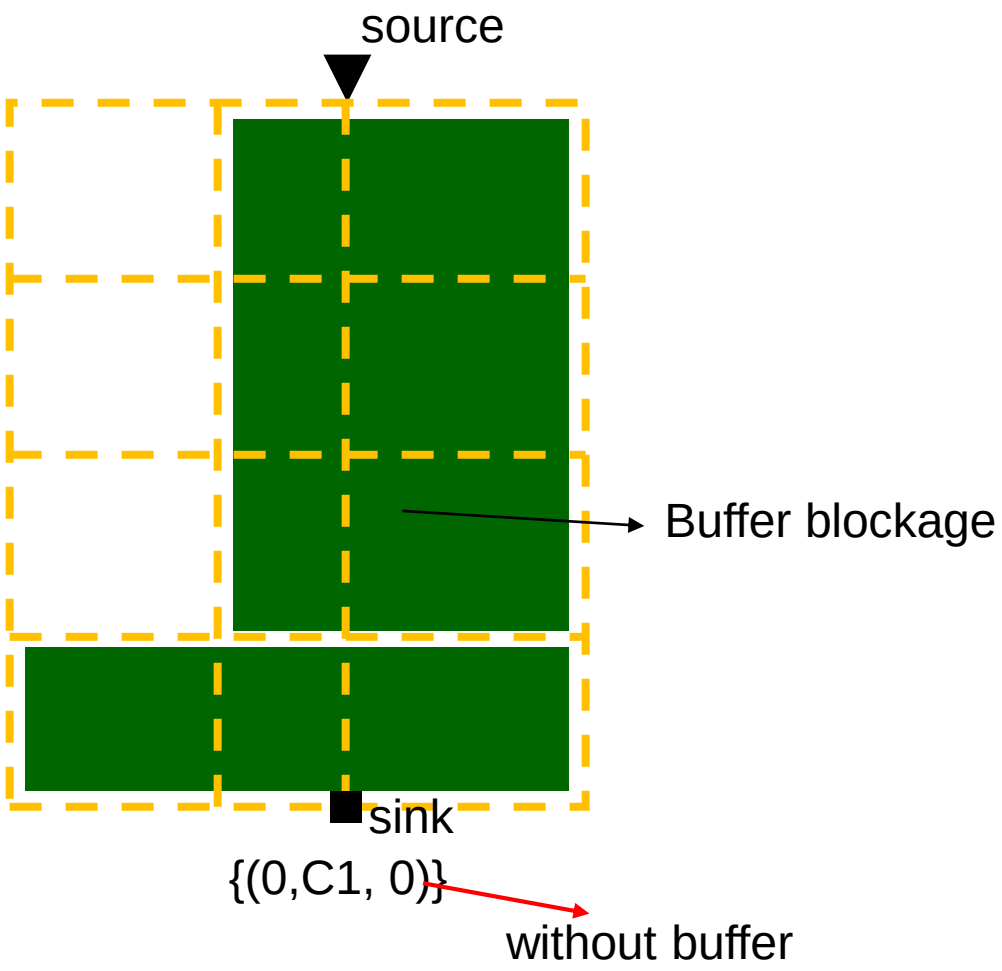


Fast Path Algorithm

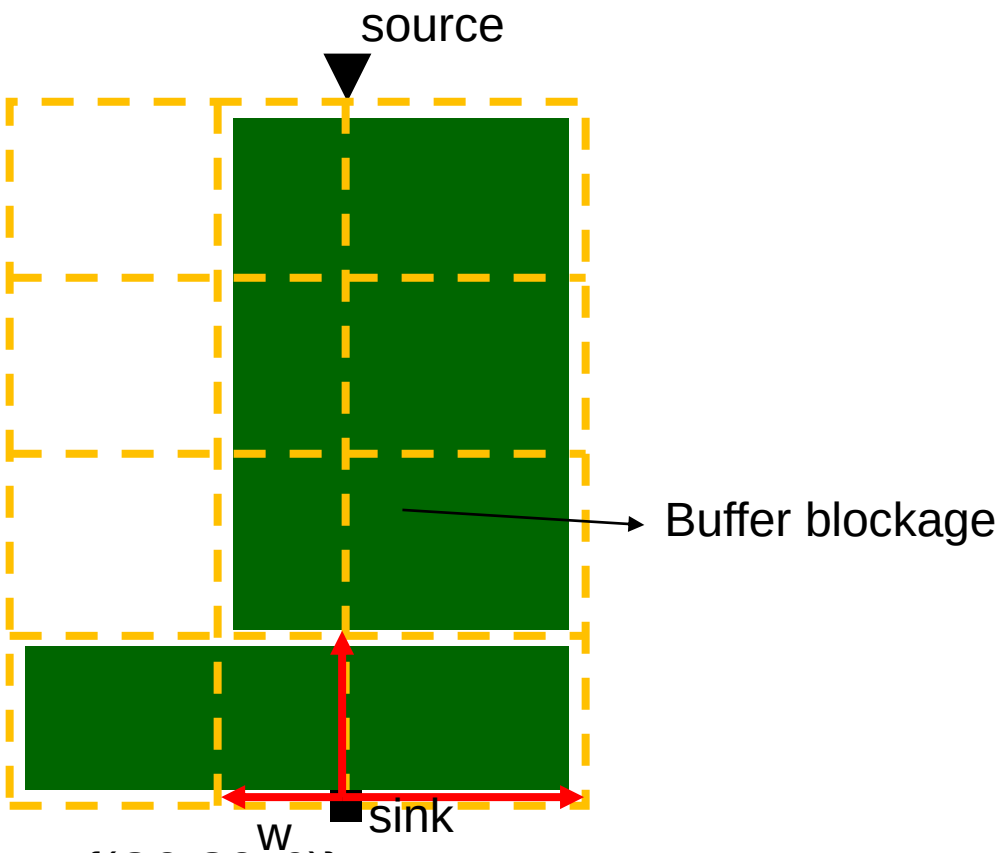
Proposed by Zhou *et al.*, DAC 1999



Fast Path Algorithm



Fast Path Algorithm



Q2= Q1+R (C/2 + C1)

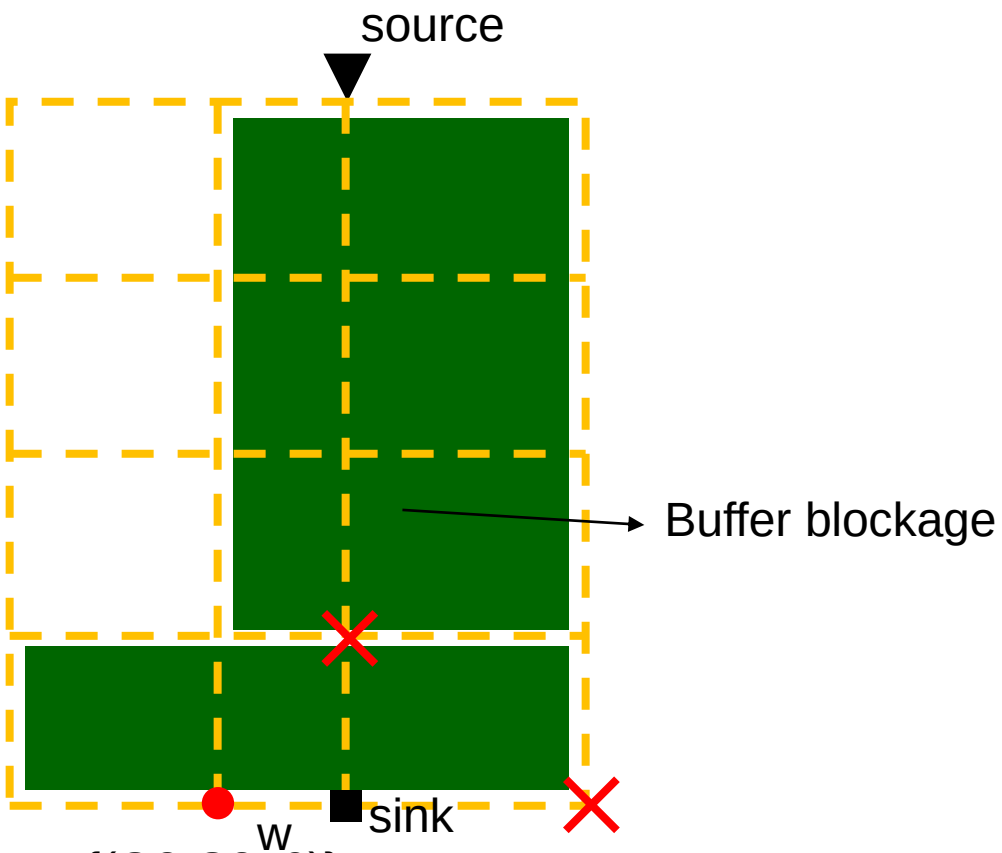
C2= C+C1

C is capacitance of wire w,

R is resistance of w

{(Q2,C2, 0)}

Fast Path Algorithm



Q2= Q1+R (C/2 + C1)

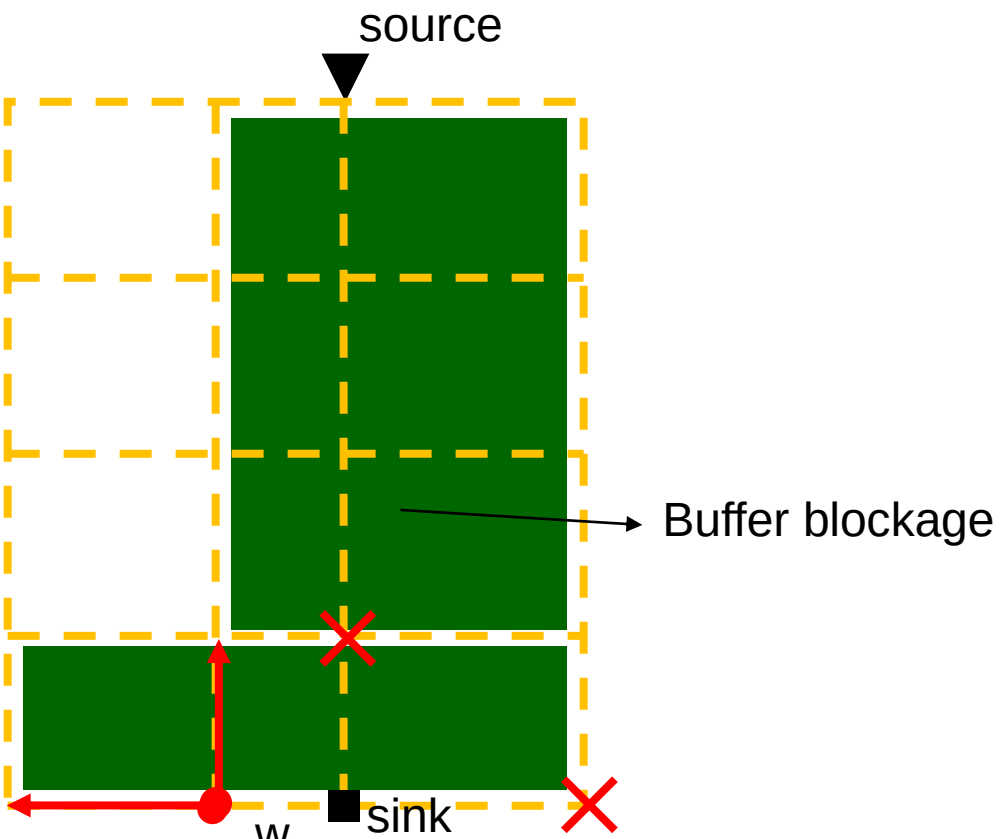
C2= C+C1

C is capacitance of wire w,

R is resistance of w

{(Q2,C2, 0)}

Fast Path Algorithm



Q2= Q1+R (C/2 + C1)

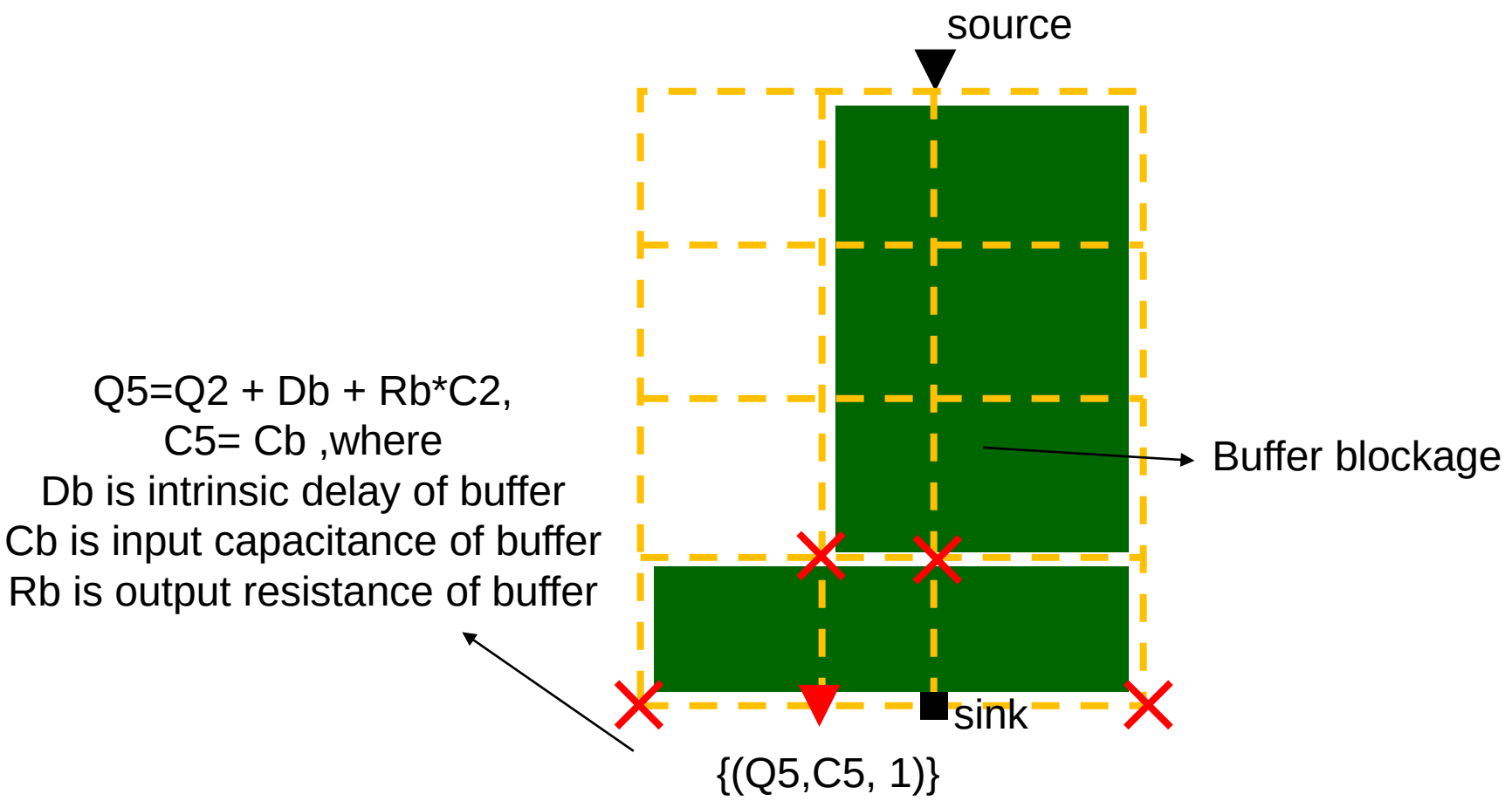
C2= C+C1

C is capacitance of wire w,

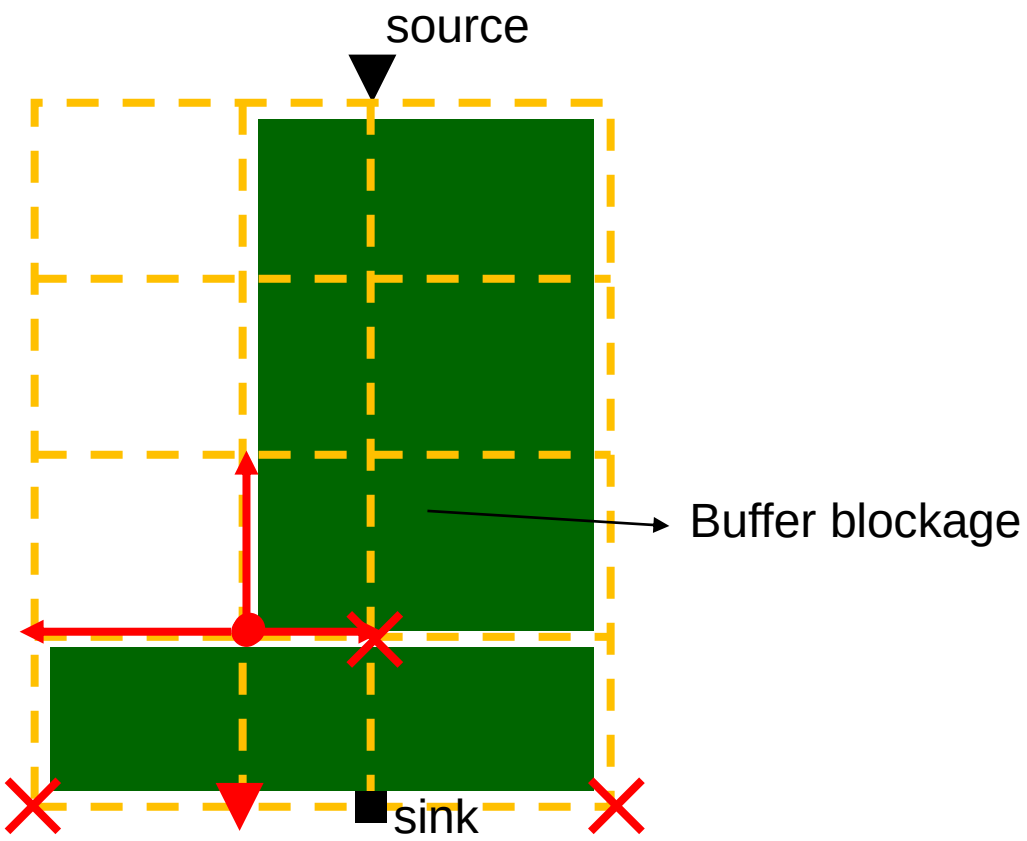
R is resistance of w

{(Q2,C2, 0)}

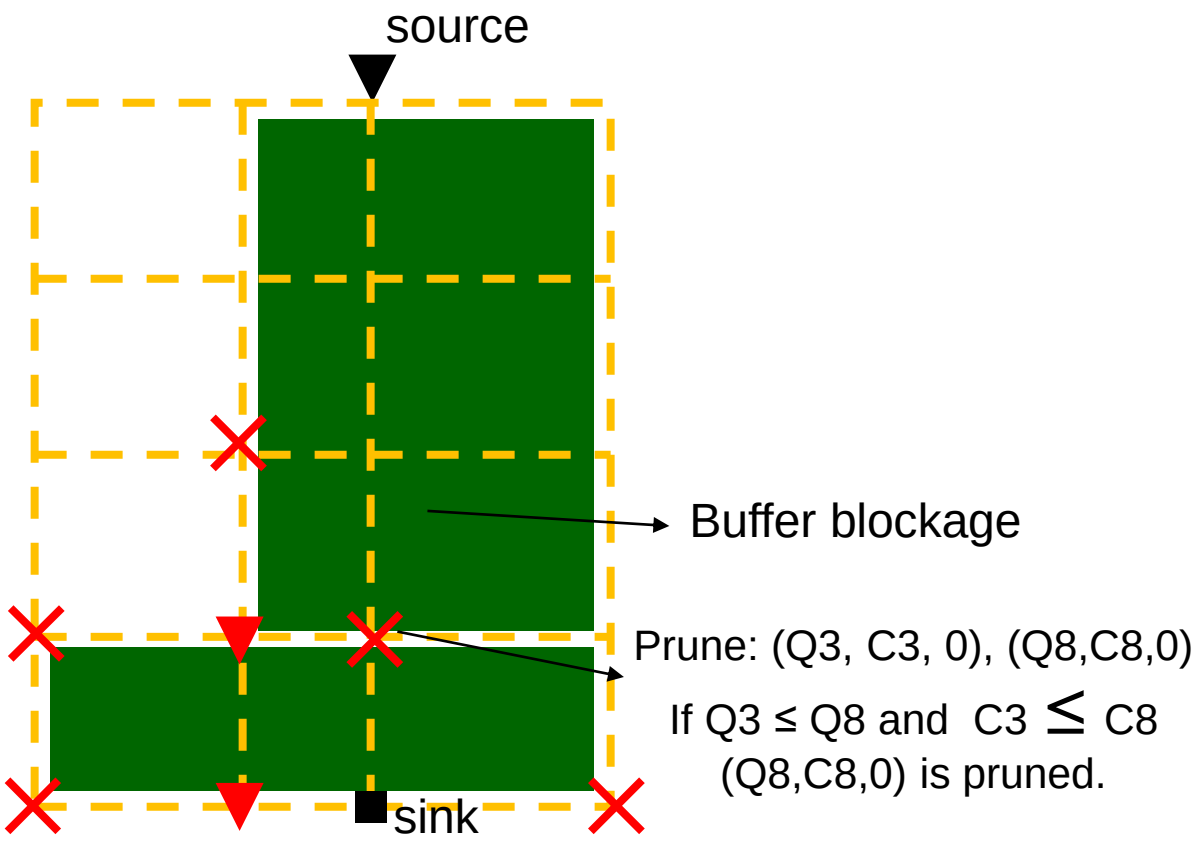
Fast Path Algorithm



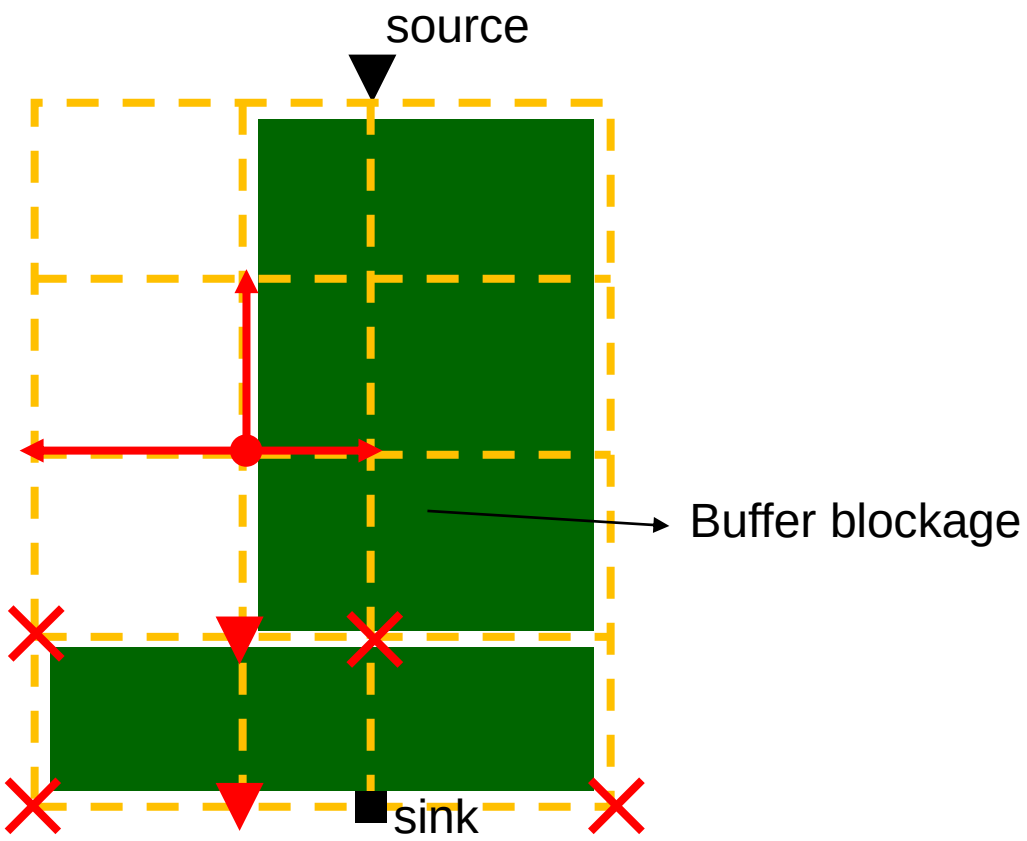
Fast Path Algorithm



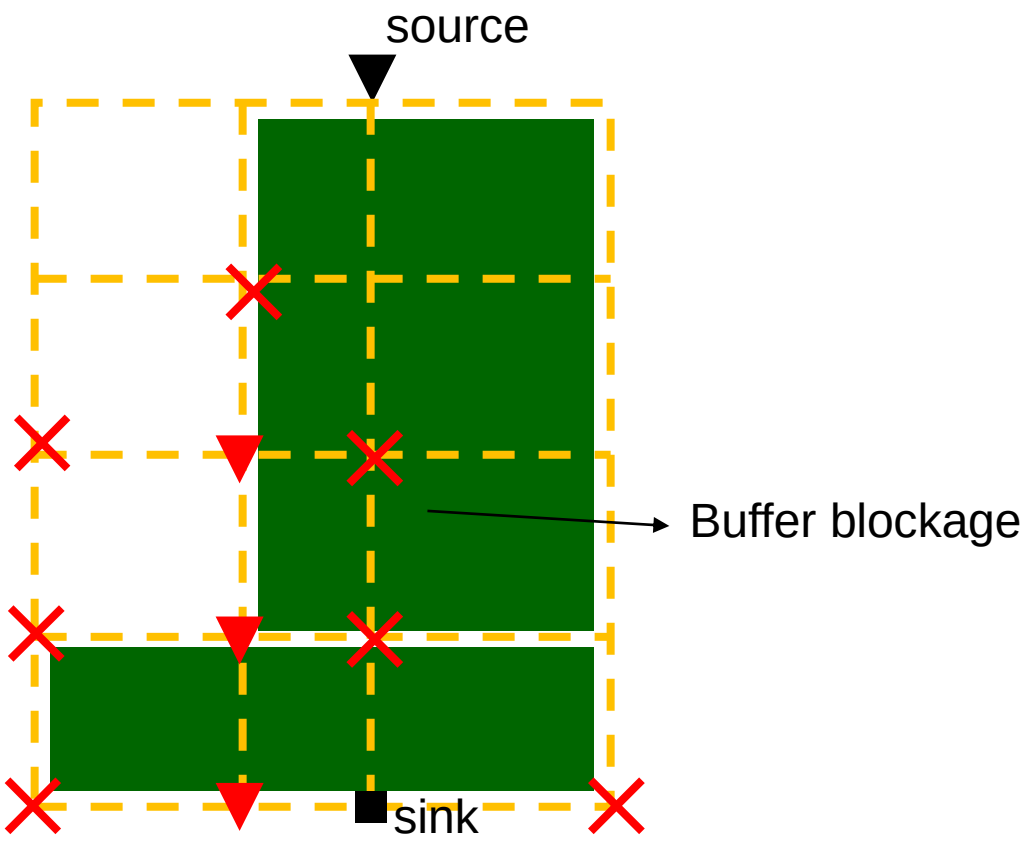
Fast Path Algorithm



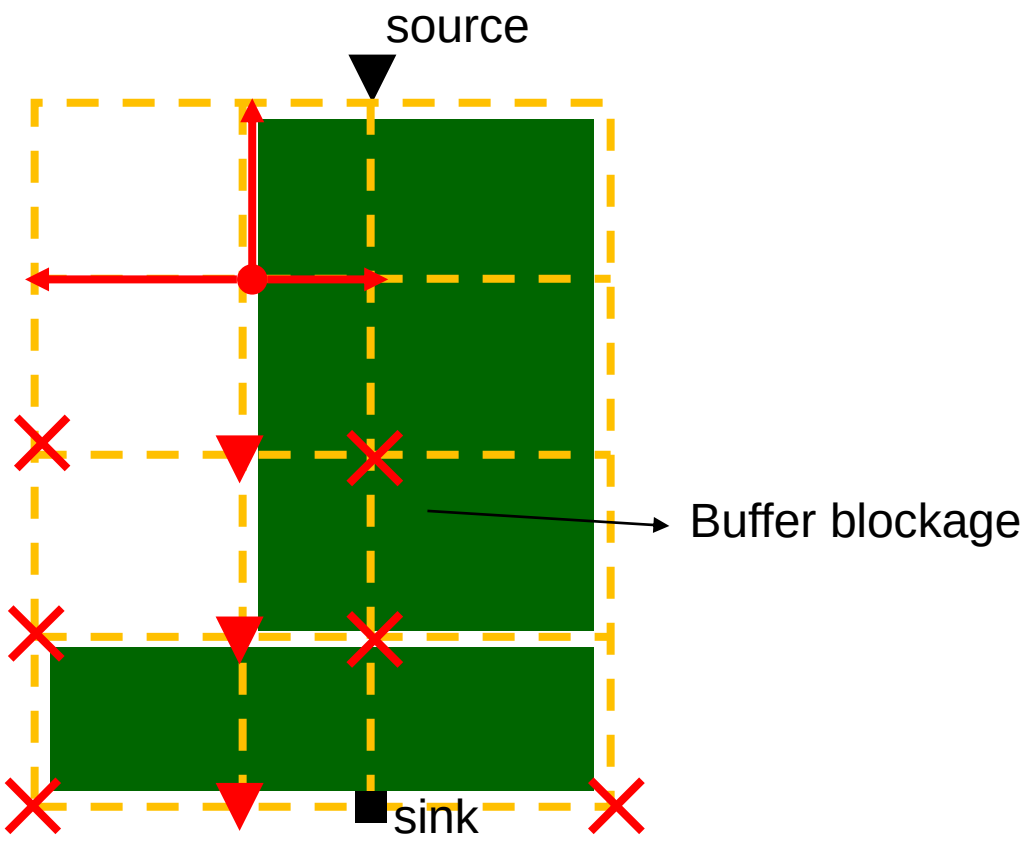
Fast Path Algorithm



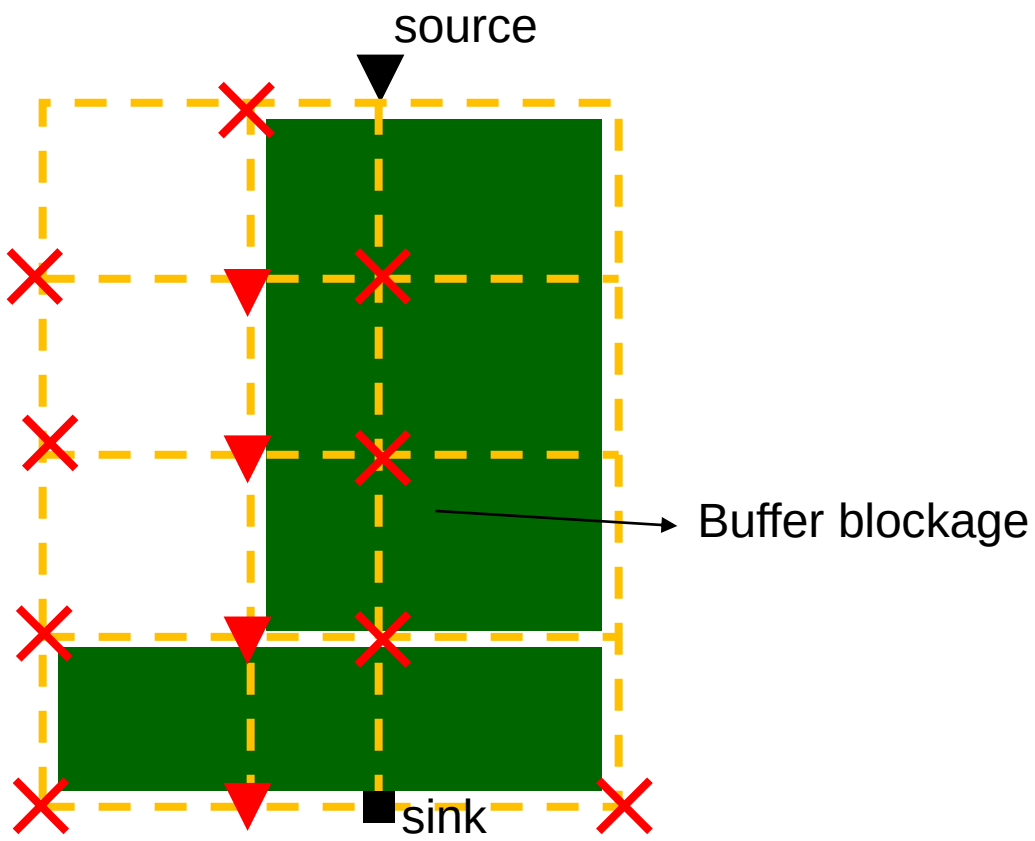
Fast Path Algorithm



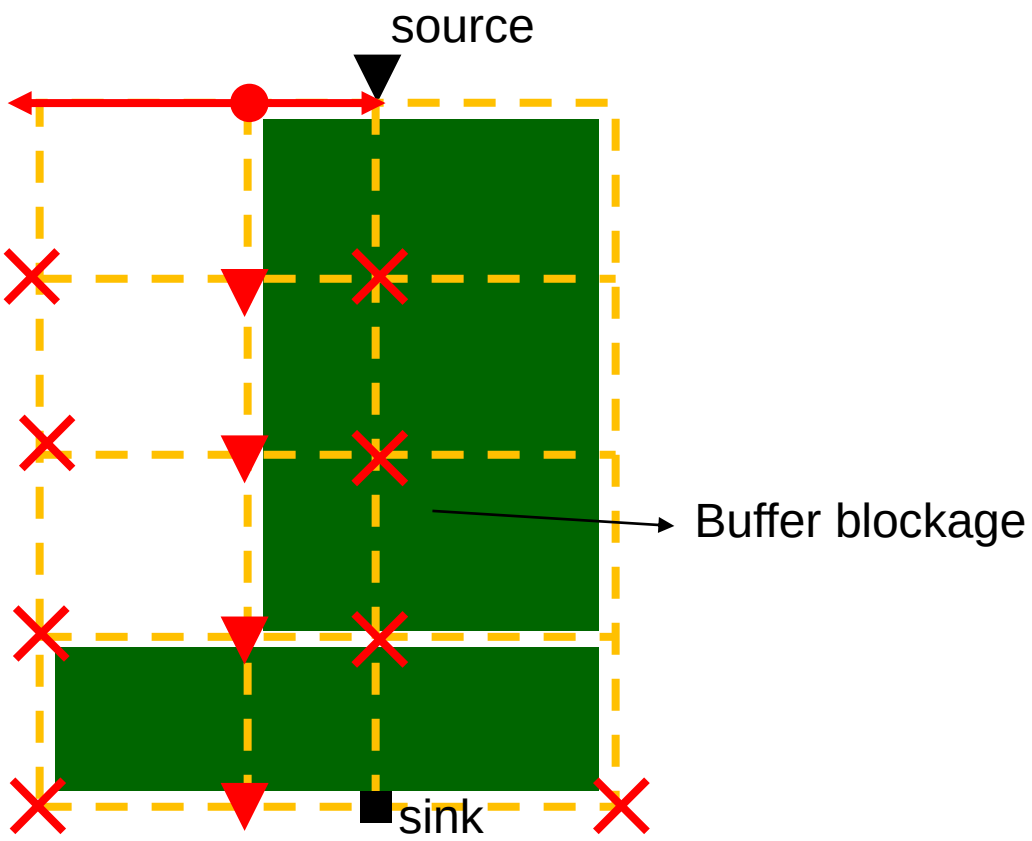
Fast Path Algorithm



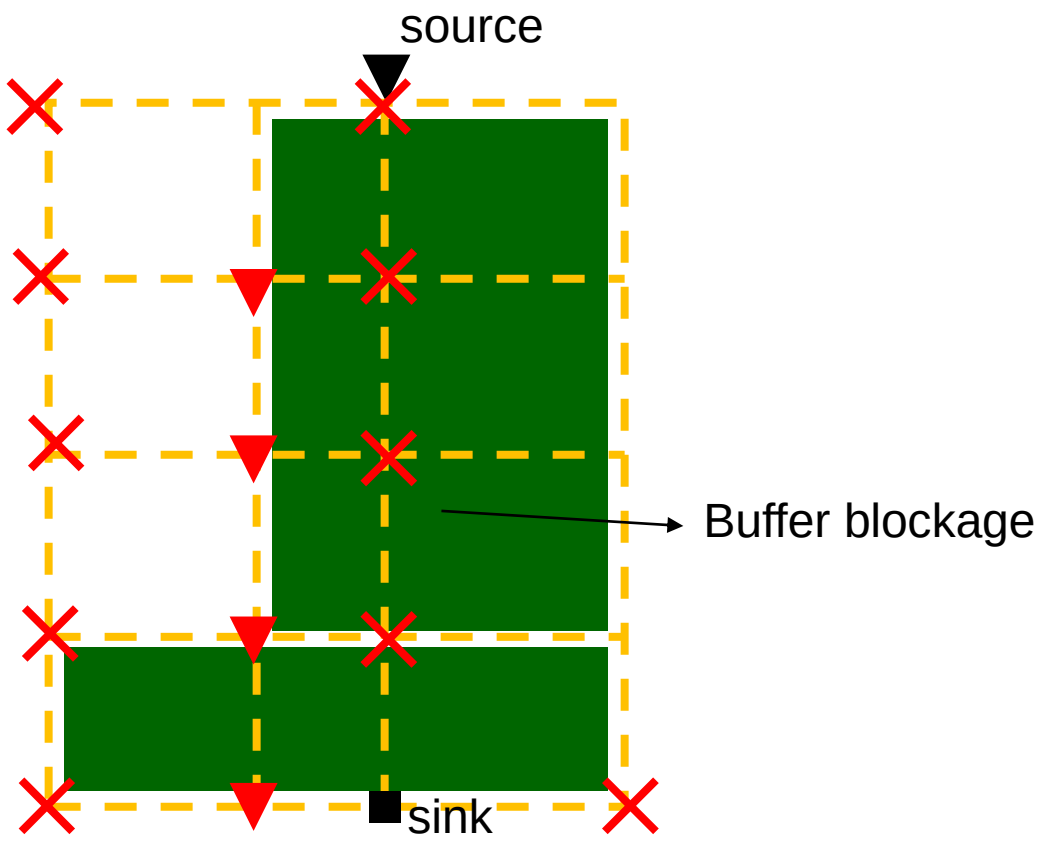
Fast Path Algorithm



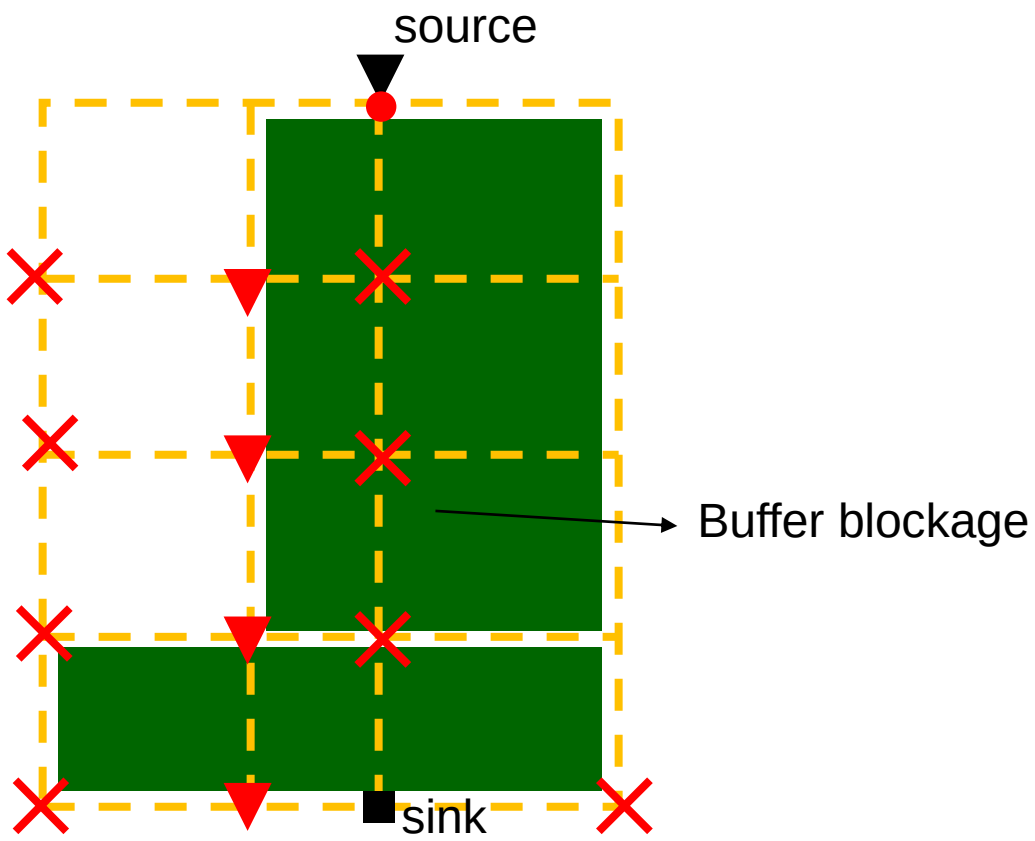
Fast Path Algorithm



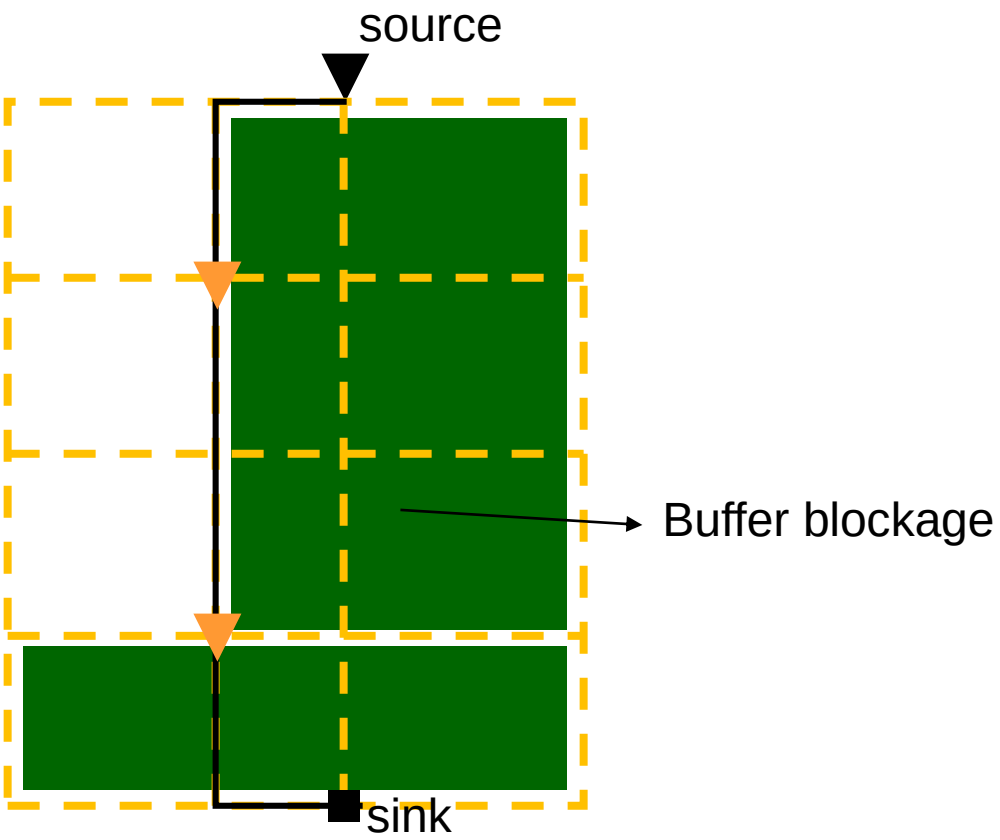
Fast Path Algorithm



Fast Path Algorithm

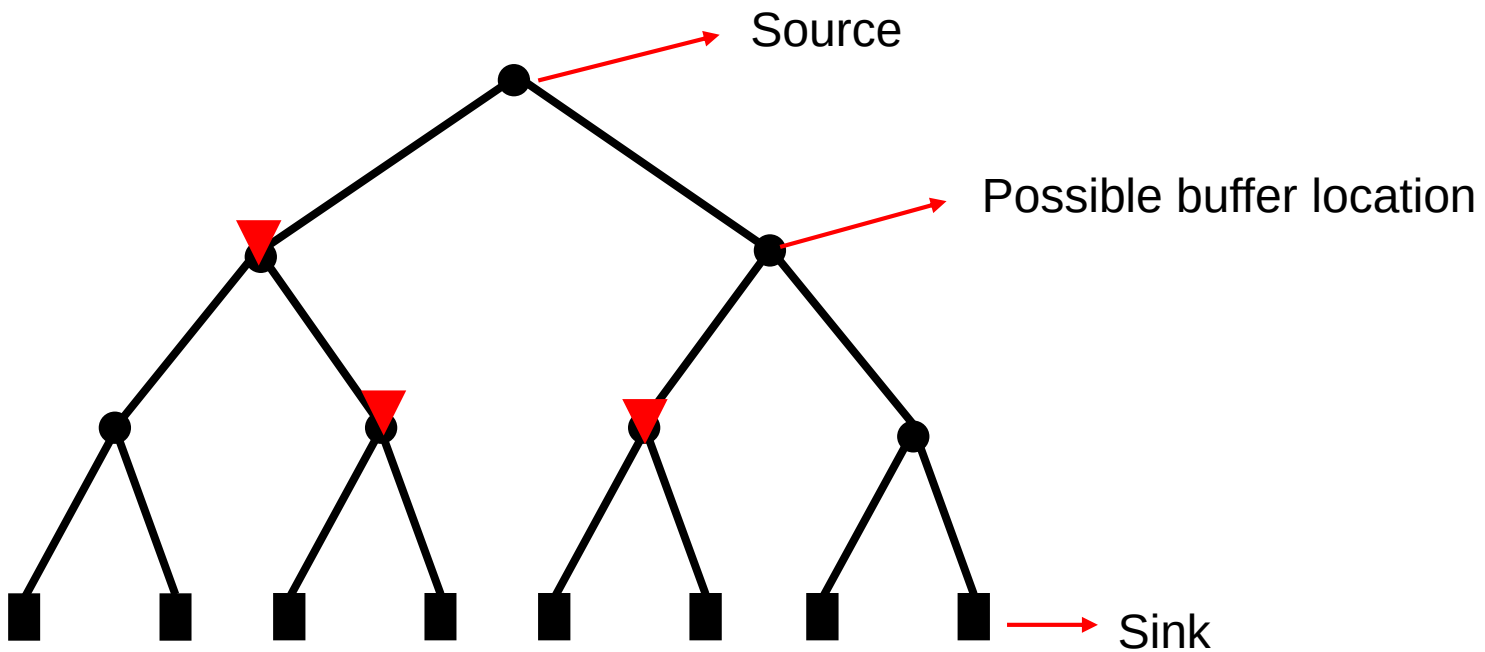


Fast Path Algorithm



Buffered Tree Construction

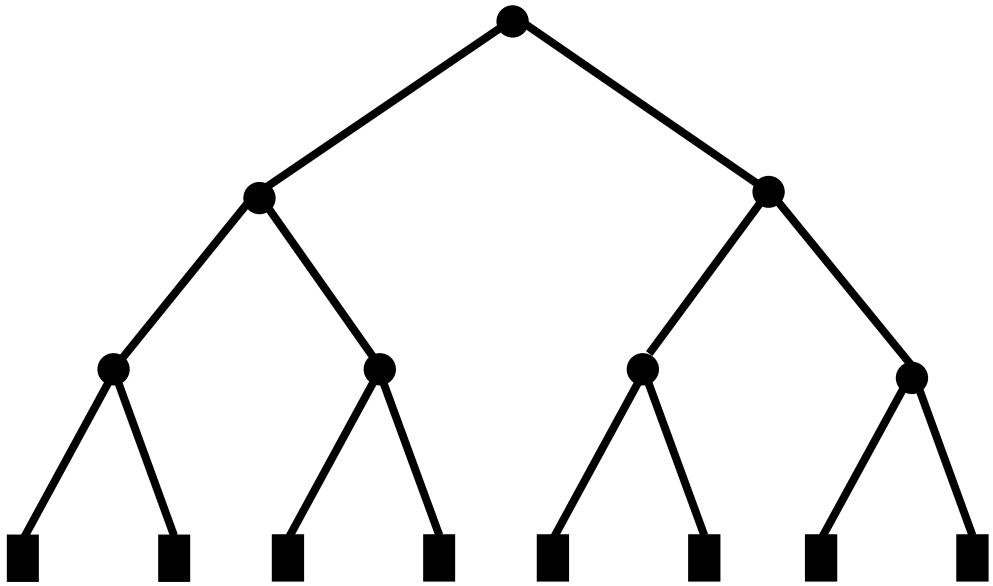
- Two-stage approach
  - Constructing a timing-driven tree
  - Considering buffer insertions at candidate locations



Routing Tree

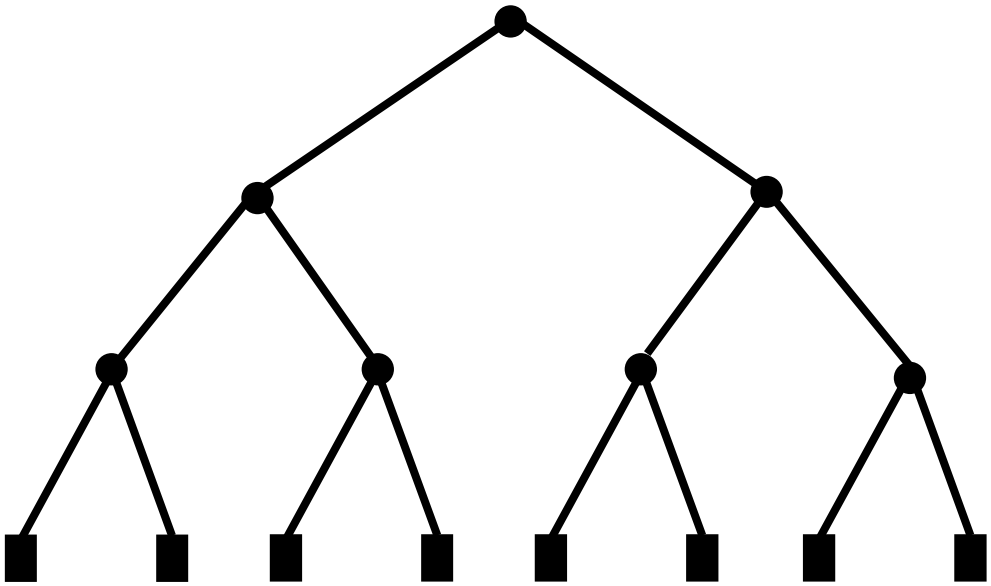
Classic Algorithm

Proposed by van Ginneken, ISCAS 1990



Routing Tree

Classic Algorithm

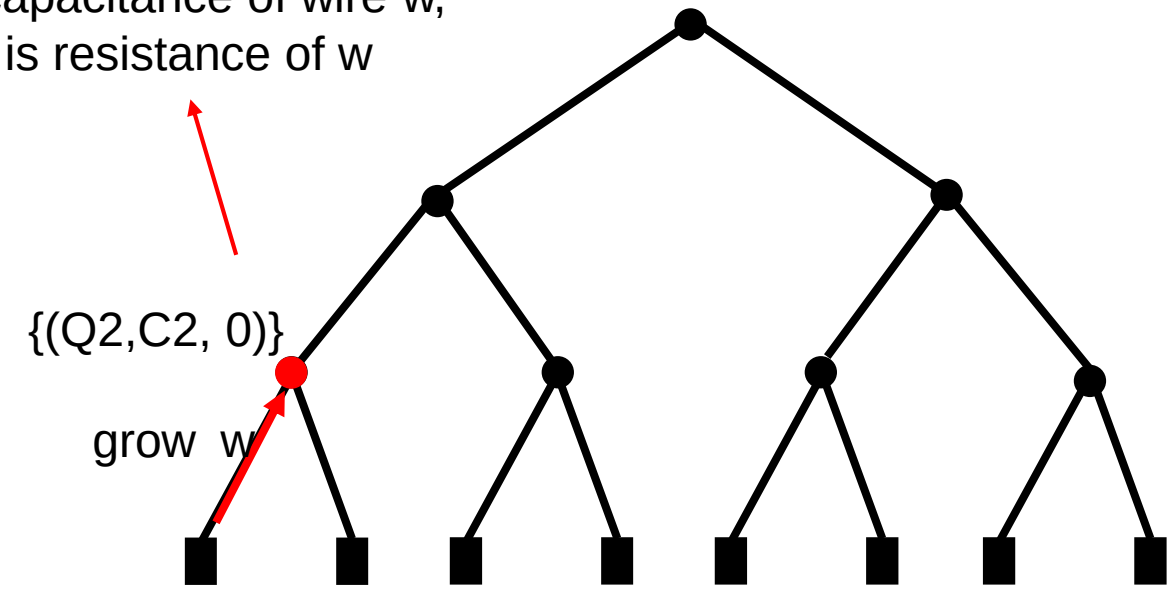


Q1,C1

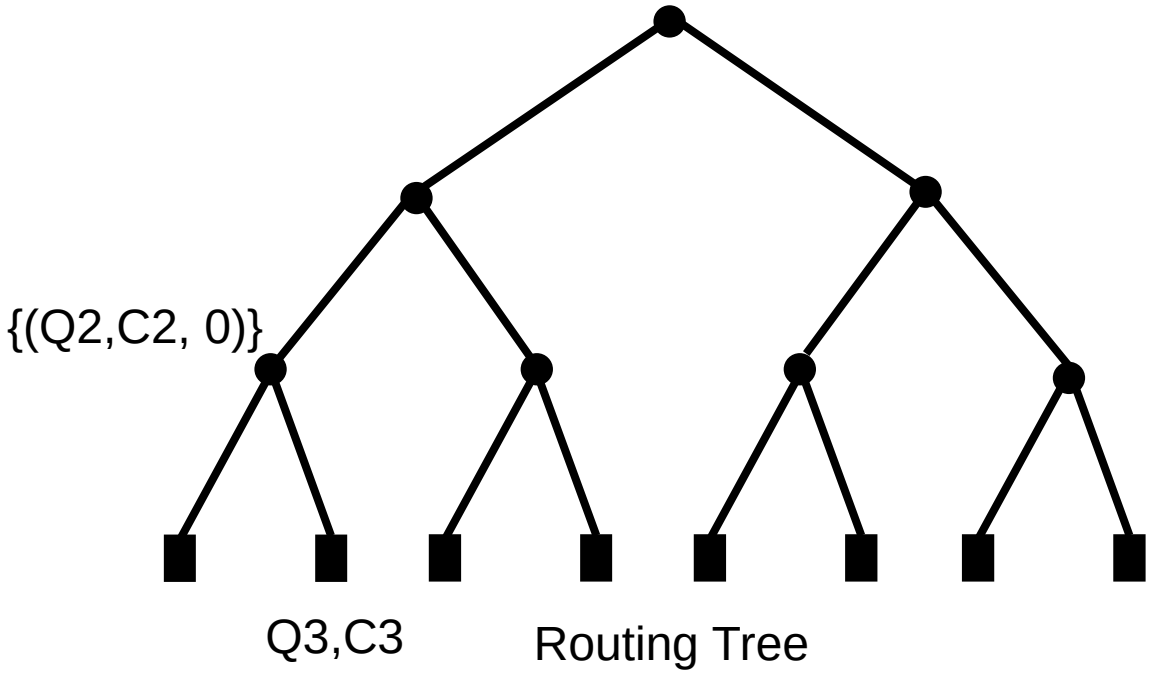
Routing Tree

Classic Algorithm

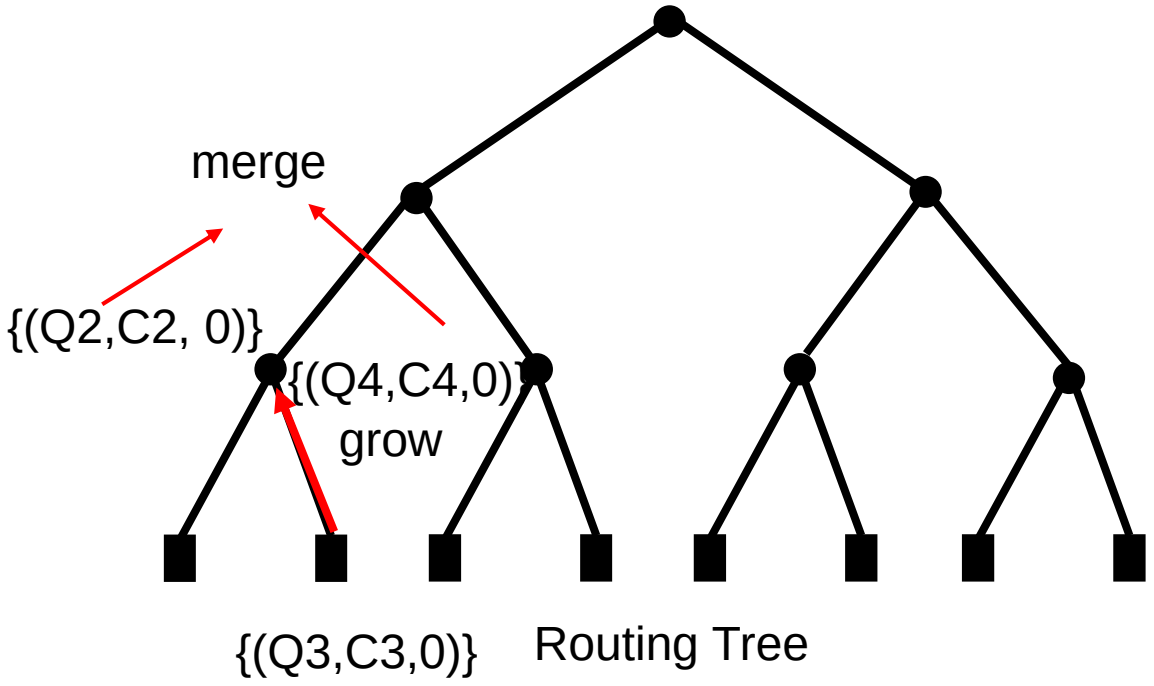
Q2= Q1- R (C/2 + C1)
C2= C+C1
C is capacitance of wire w,
R is resistance of w



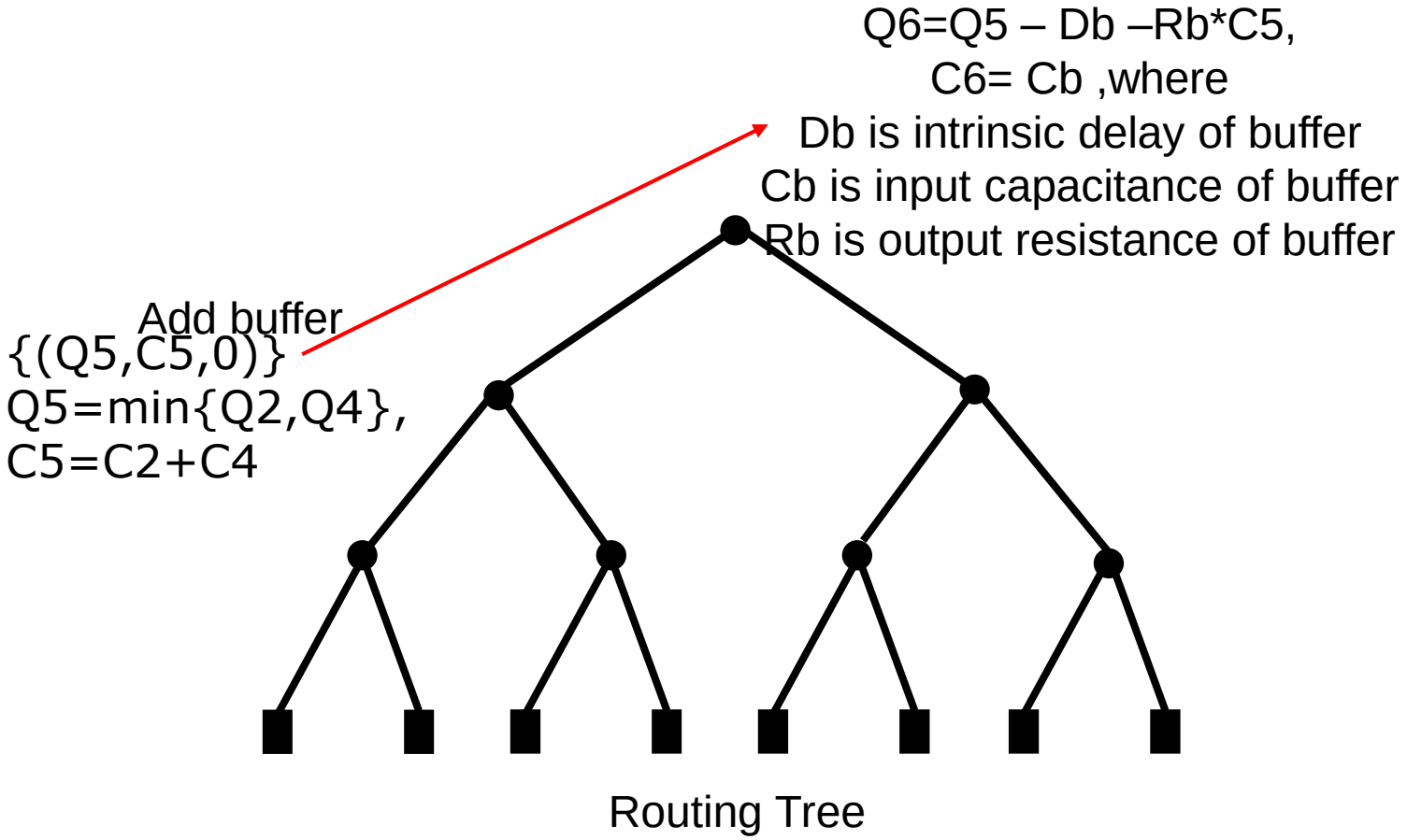
Classic Algorithm



Classic Algorithm

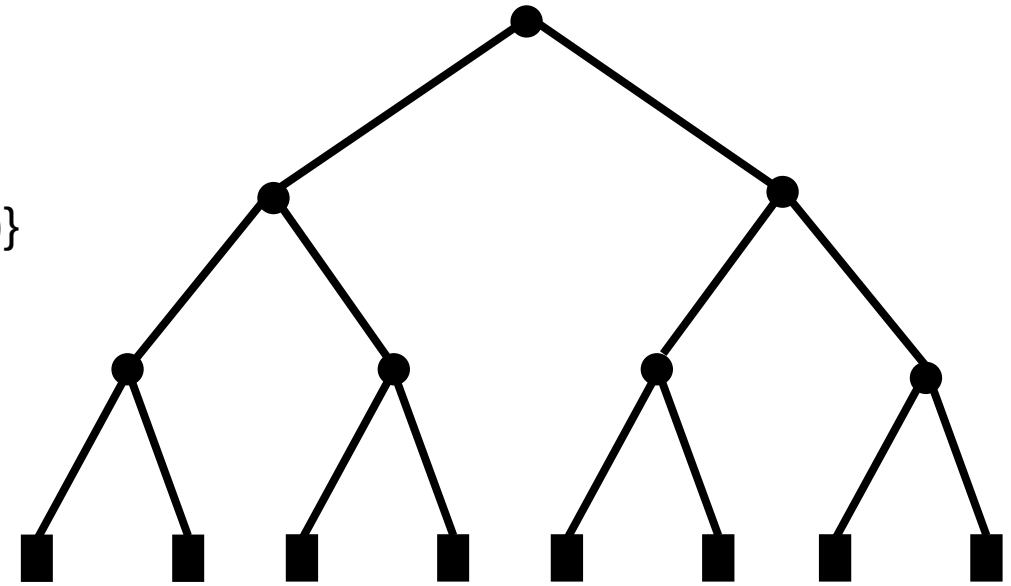


Classic Algorithm



Classic Algorithm

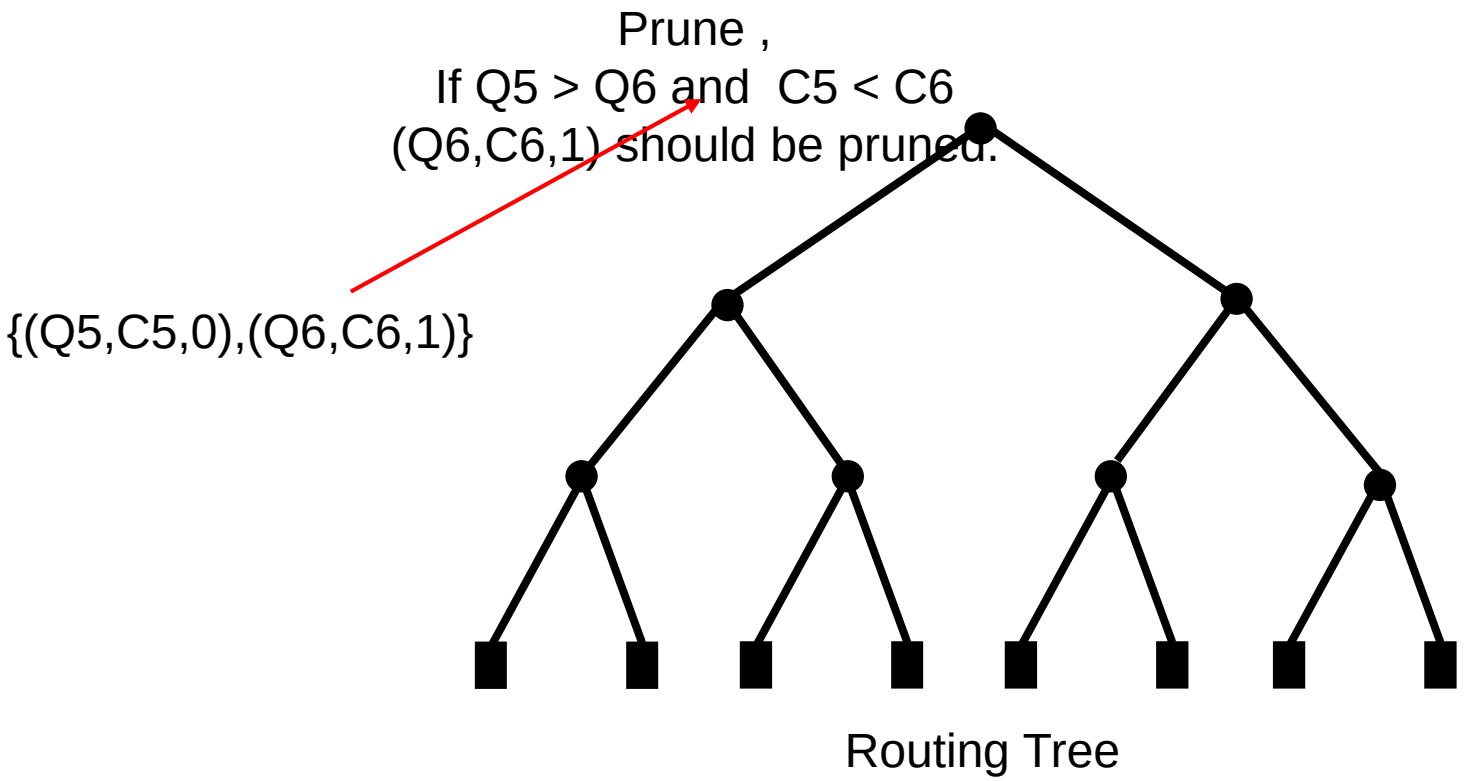
{(Q5,C5,0),(Q6,C6,1)}



Routing Tree

with buffer

Classic Algorithm

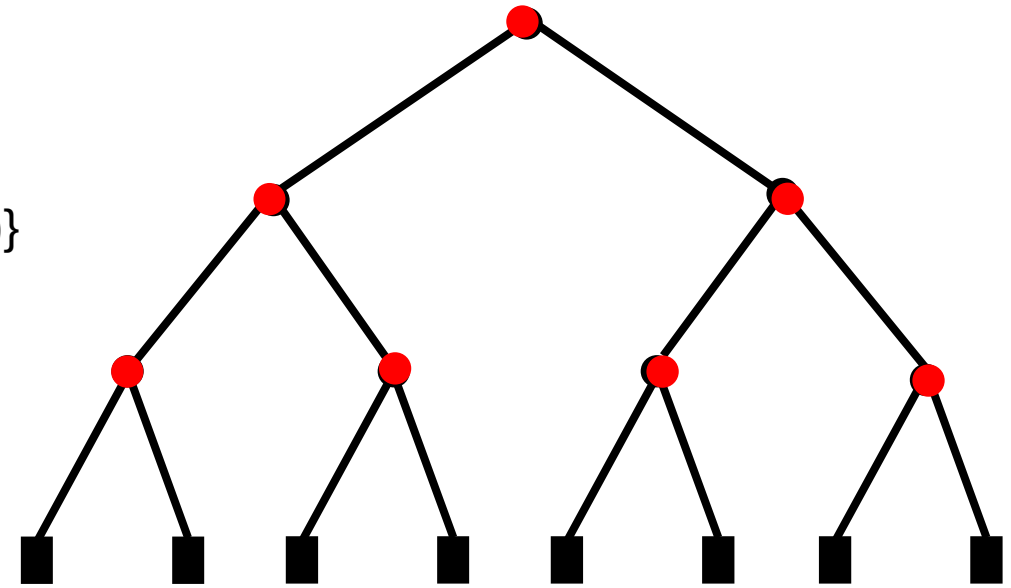


Classic Algorithm

$\{(Q_{r1},C_{r1},B_{r1}),\ldots,(Q_{rm},C_{rm},B_{rm})\}$.

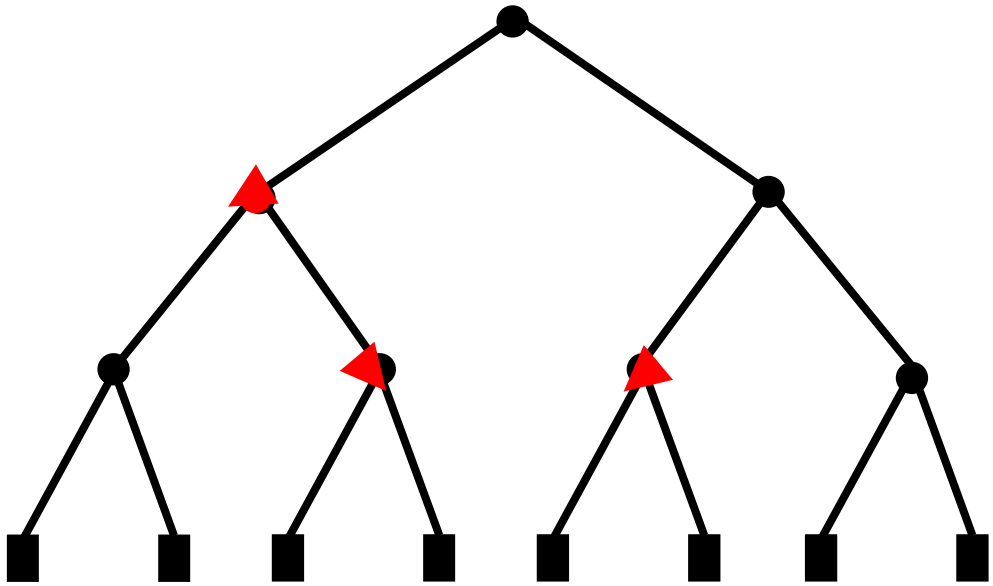
One with max required time will be picked up

{(Q5,C5,0),(Q6,C6,1)}



Routing Tree

Classic Algorithm



Buffered routing tree with maximal required time

Extensions of Classic Algorithm

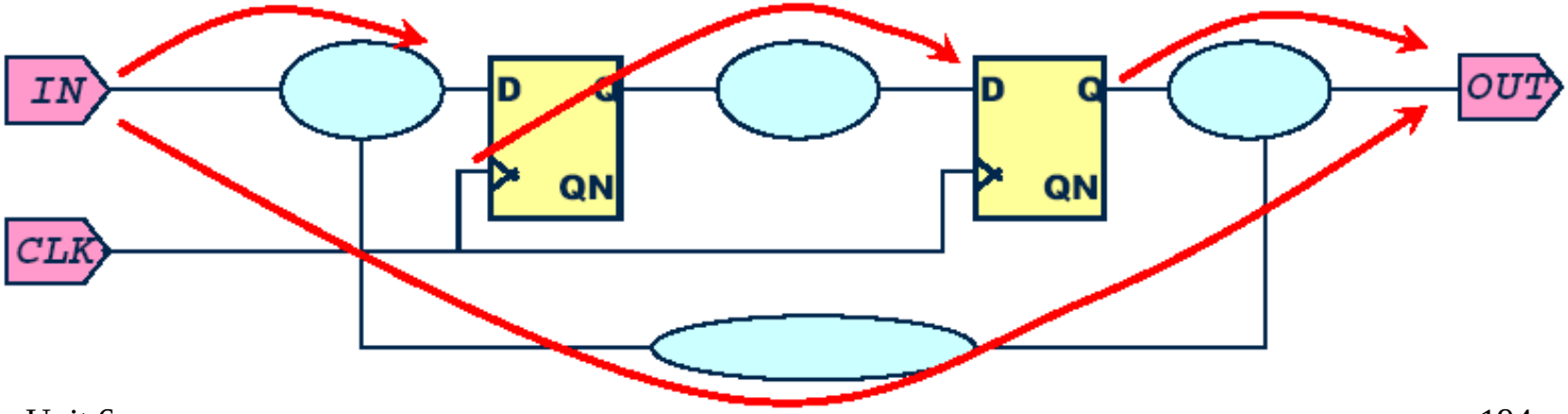
- Multiple buffer types
- Inverters
- Polarity constraints
- Controlling buffer resources
- Capacitance constraints
- Wire sizing
- Blockage recognition

Dynamic Timing Analysis

- So-called Simulation
  - Input vectors are applied during the simulation time
  - Simulator calculates the logic value and delay
- Disadvantages of DTA
  - Virtually impossible to do exhaustive analysis
    - Vector creation takes too long
    - Incomplete timing coverage
  - Hard to discern the cause of failure because the function and timing are analyzed at the same time
  - Requires more memory and CPU resources over STA
    - Long simulation time
    - Capacity limited

Static Timing Analysis

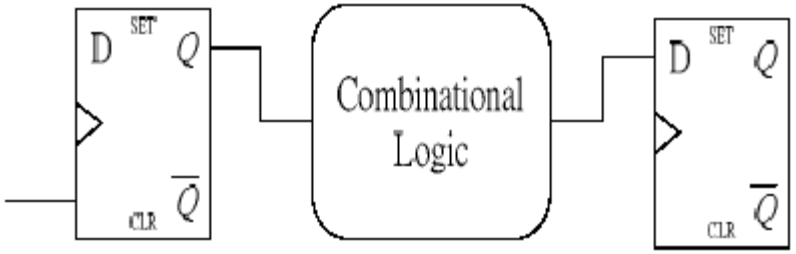
- A method for determining if a circuit meets timing constraints without having to simulate clock cycles
- Three main steps
  - Break the design into sets of timing paths
  - Calculate the delay of each path (create timing graph)
  - Check all path delays to see if the given timing constraints are met



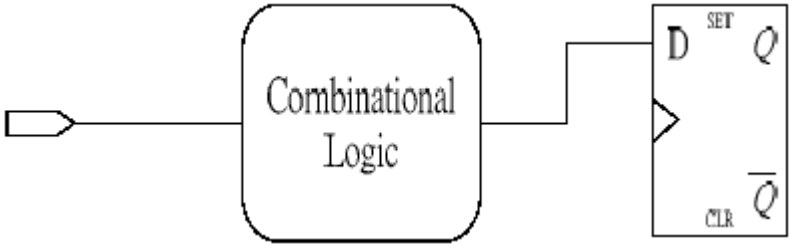
Advantages of STA

- Exhaustive timing coverage
- Does not require input vectors
- More efficient than DTA in memory and CPU resources
  - Faster operation
  - Capacity for millions of gates

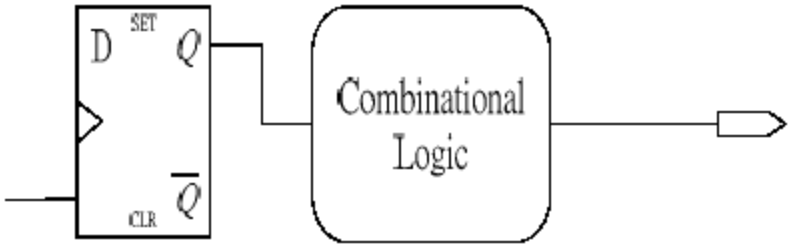
Four Types of Timing Paths



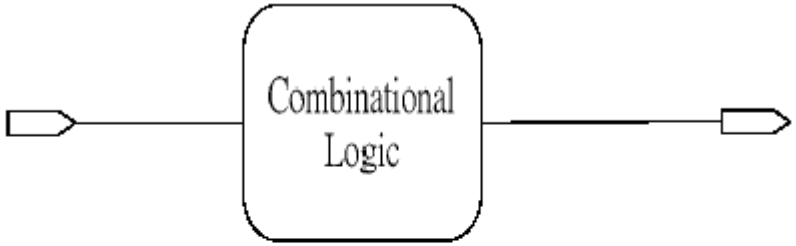
Start Point:
clock pin of sequential device
End Point:
data input pin of sequential devices



Start Point:
primary input port
End Point:
data input pin of sequential devices



Start Point:
clock pin of sequential device
End Point:
primary output port

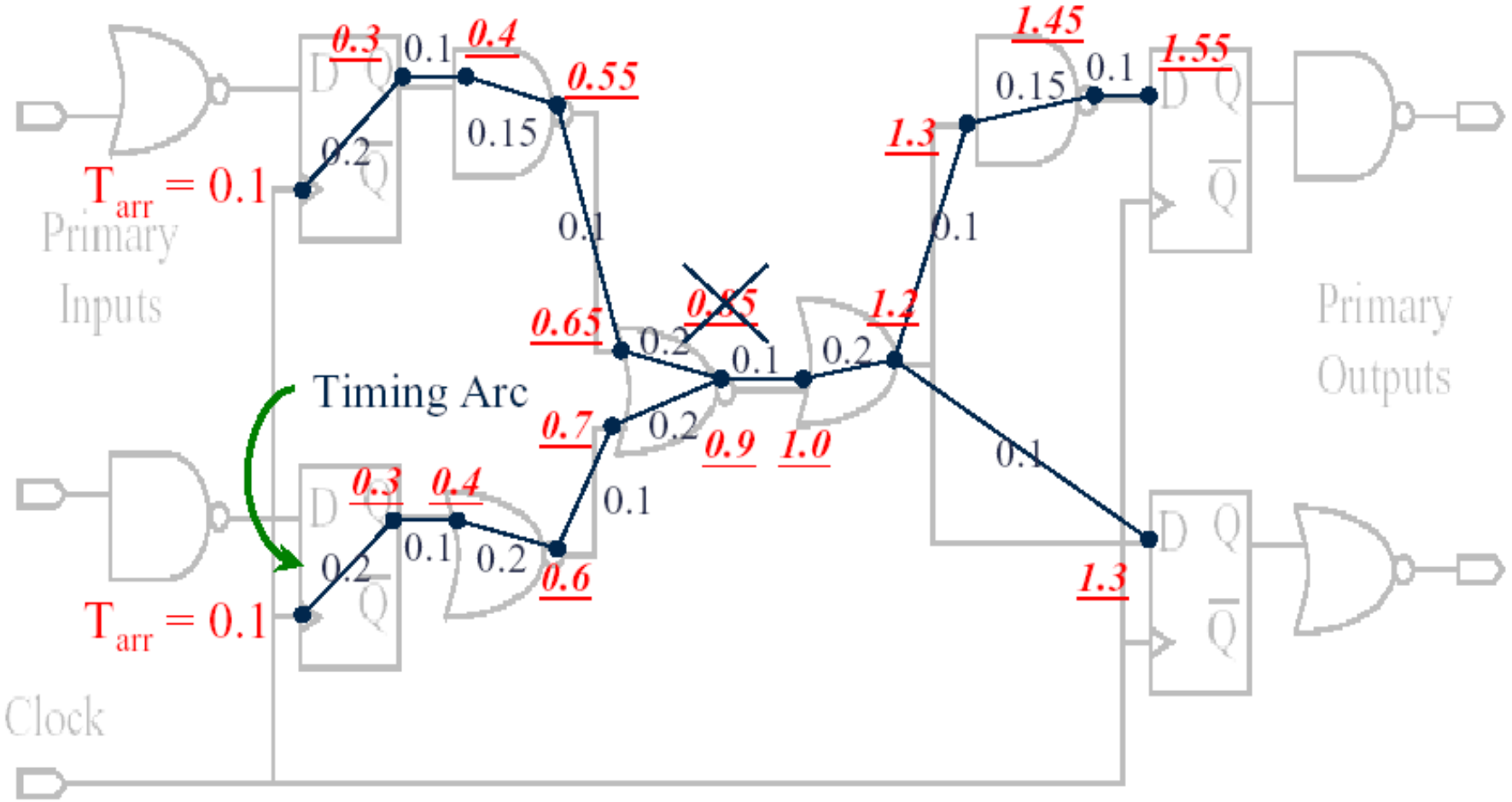


Start Point:
primary input port
End Point:
primary output port

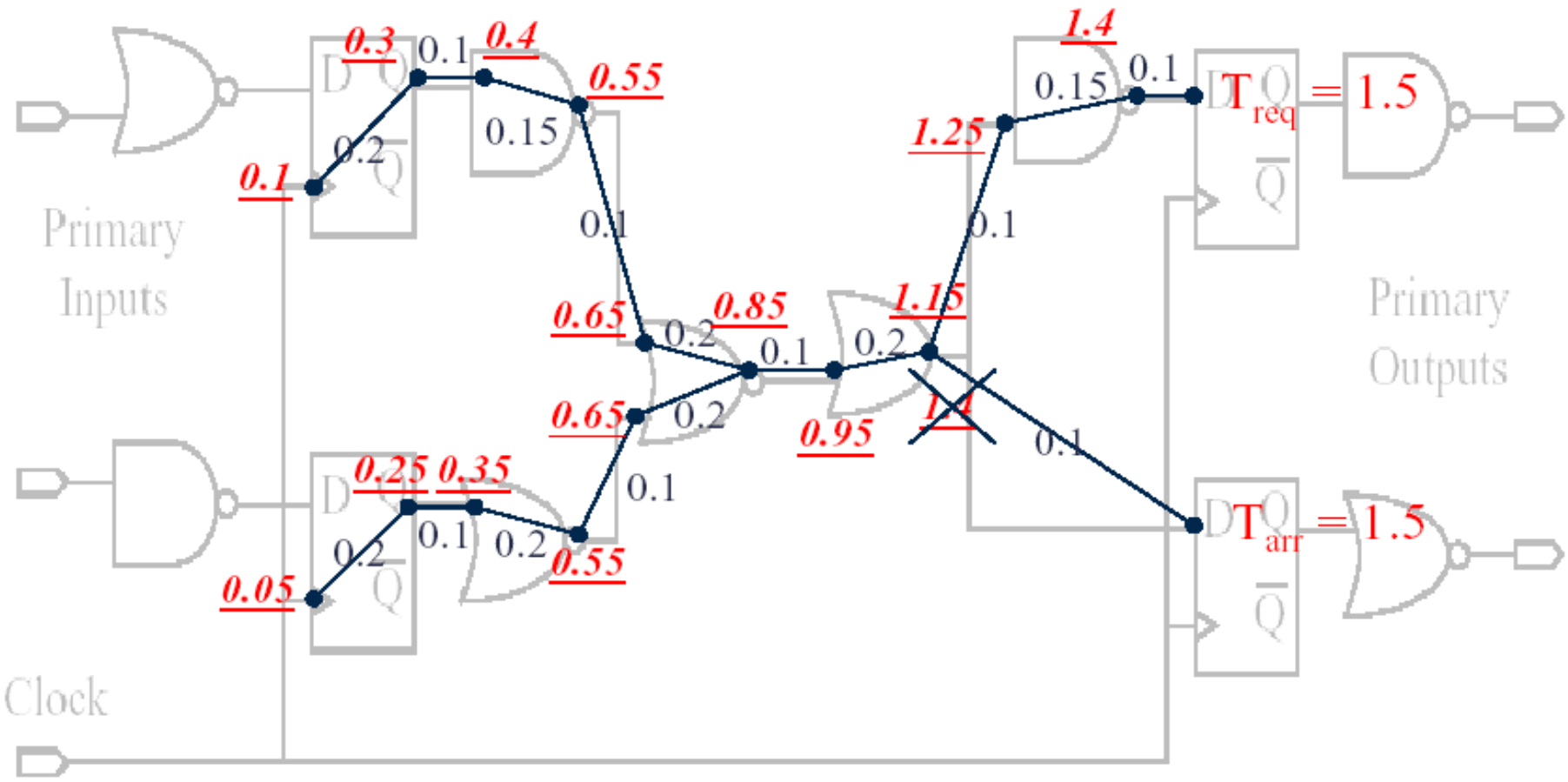
Create Timing Graph

- Netlist is represented as directed acyclic graph (DAG)
- Delay values associated with links (cells & nets) are calculated
- Create the timing graph of arrival time (AT)
- Create the timing graph of required time (RT)
- Create the slack graph
  - Timing is met when slack is greater than or equal to zero (RT should always be less than or equal to AT)

Timing Graph – Arrival Time

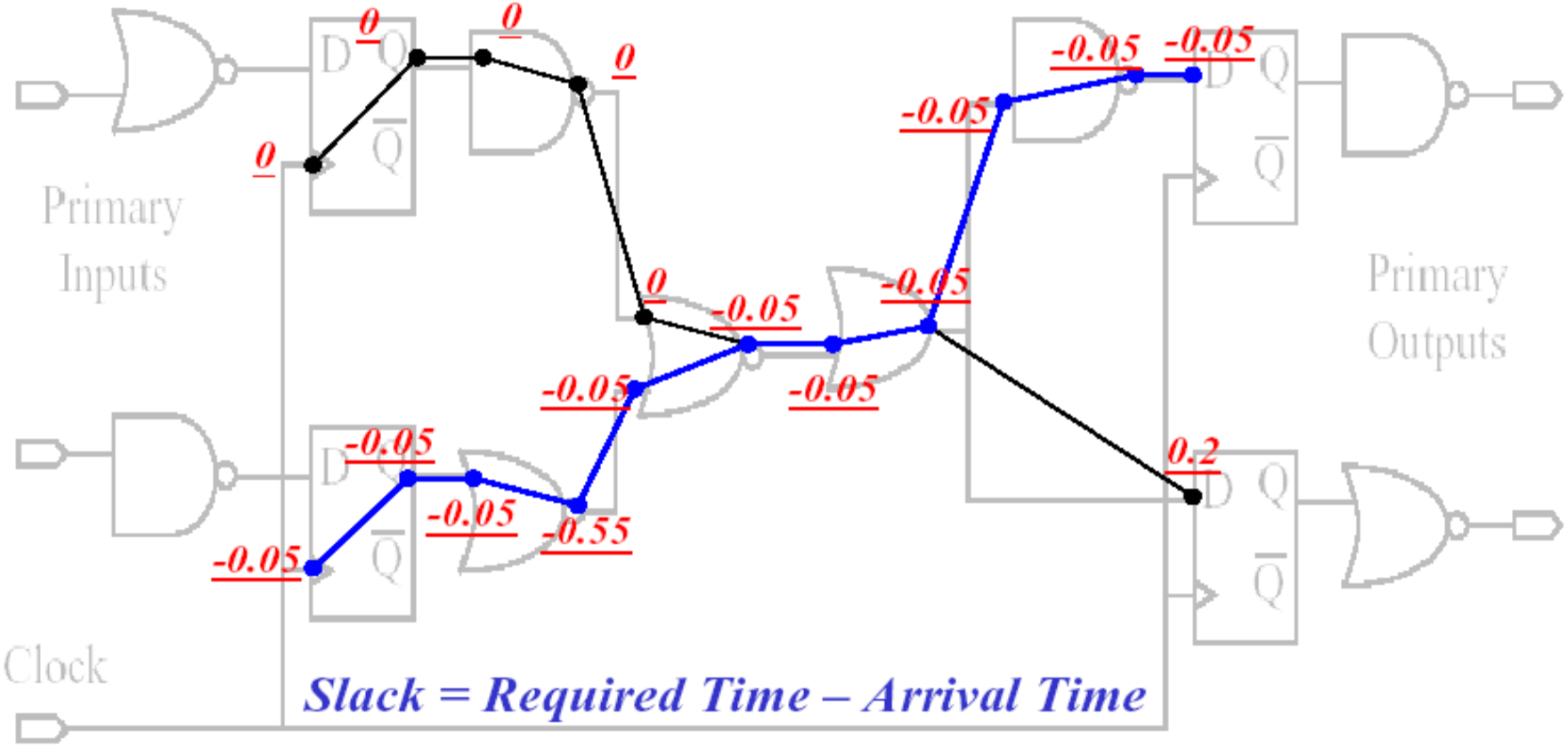


Timing Graph – Required Time



setup time case

Slack Graph

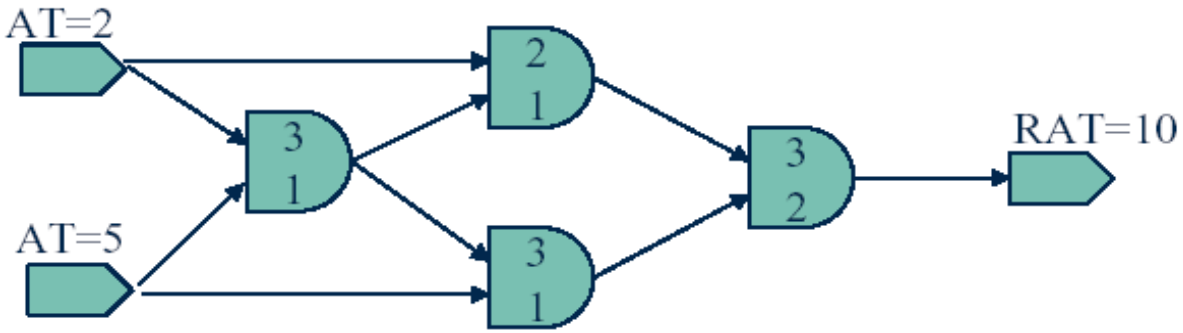


setup time case

Block-based vs. Path-based STA (1/2)

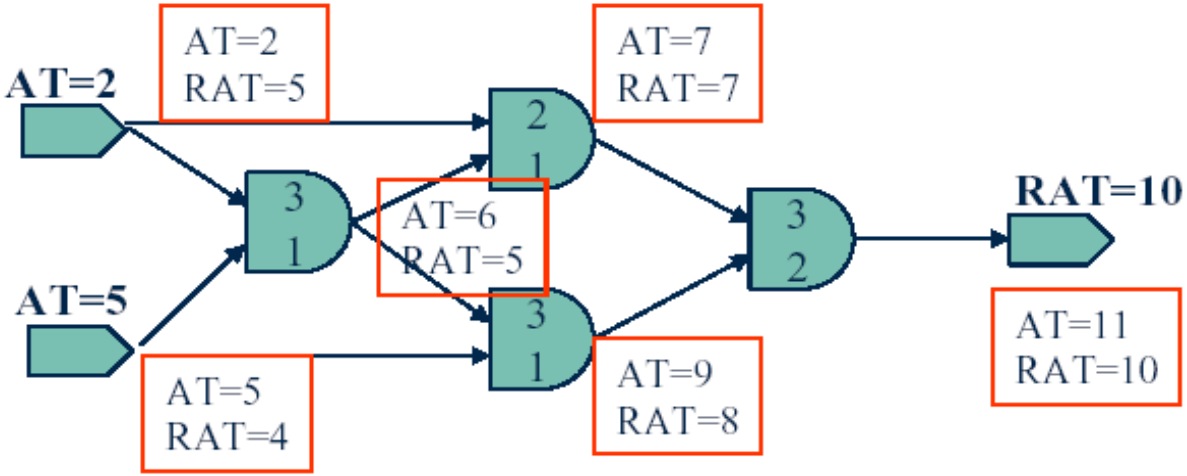
- Block-based: timing information is associated with discrete design elements (ports, pins, gates)
  - Slack is calculated on every design element
- Path-based: timing information is associated with topological paths (collections of design elements)

Block-based vs. Path-based STA (2/2)



Path-based:

2+2+3 = 7 (OK)
2+3+1+3 = 9 (OK)
2+3+3+2 = 10 (OK)
5+1+1+3 = 10 (OK)
5+1+3+2 = 11 (Fail)
5+1+2 = 8 (OK)



Block-based:

Critical path is determined as collection of gates with the same, negative slack: In our case, we see one critical path with slack = -1