Global Routing

Yih-Lang Li (李毅郎)

CS NYCU

Reference:

1. Sadiq M. Sait and Habib Youssef, "VLSI Physical Design Automation", 2ed, 1999

2. technical papers in ACM/IEEE conferences (DAC/ICCAD/ISPD/ASP-DAC) and journals (TCAD/TVLSI/TODAES)

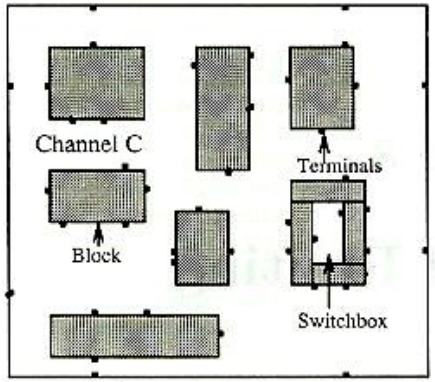
Thanks to groups of Profs Chris Chu and David Pan for sharing their presentation slides

Outline

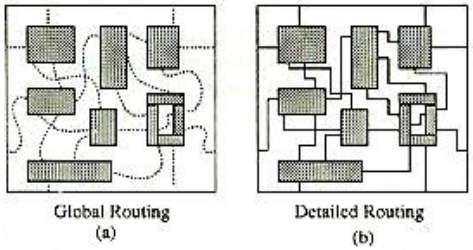
- Introduction
- Problem Formulation
- Grid-Based Algorithms
- Integer Programming Based Algorithm
- Recent Advances in Global Routing
  - ✓ FastRoute
  - ✓ NCTU-GR

Introduction

- Global Routing – find the passing regions for each net
  - Input – Netlist, Timing budget for critical nets, Placement information (block location & pin location), RC delay for each type of metal layer and via
  - Output – Routing regions and pin locations for each net on all the region boundaries it crosses
- Detailed Routing – find the actual geometry layout for each net within the assigned routing regions



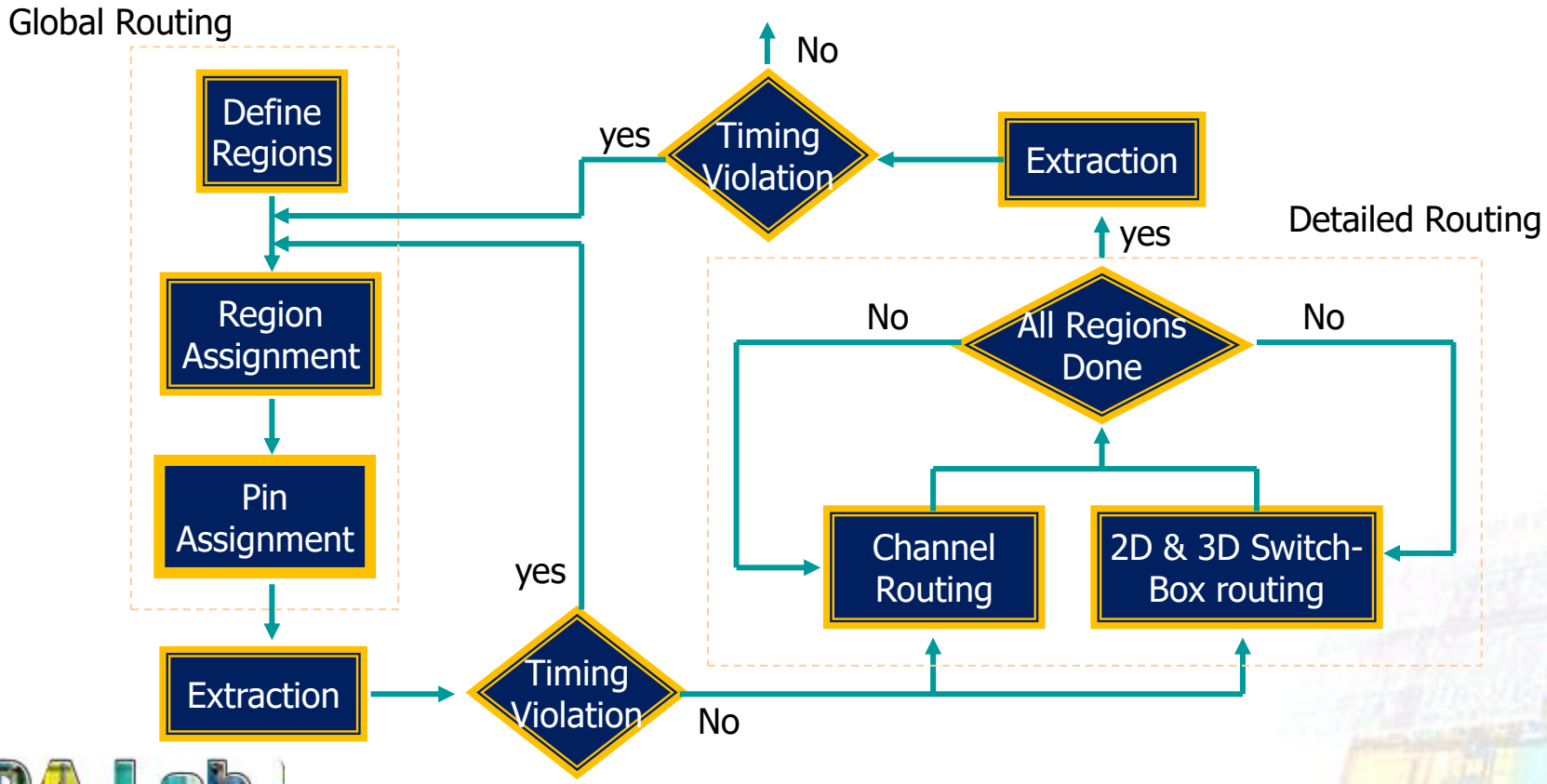
Layout of circuit. Blocks and pins after placement



Two phases of routing

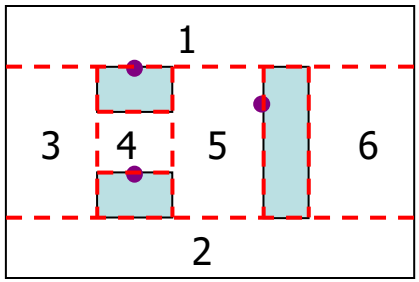
Introduction

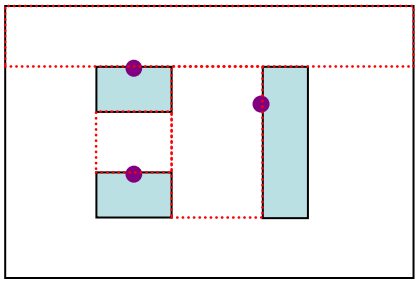
Two-phase routing flow

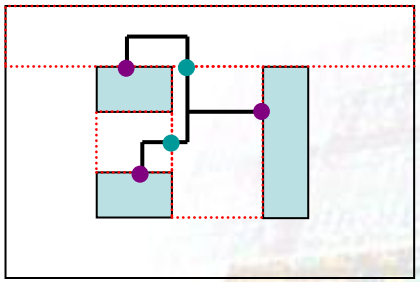


Introduction

- Three Steps of Global Routing
  - Region Definition – partition the entire routing space into routing regions
  - Region Assignment – identify the sequences of regions through which a net will be routed
    - Channel Capacity – $\frac{l \times h}{w + s}$ l: layer number, h: channel height, w: wire width, s: wire spacing
  - Pin Assignment – each net is assigned a pin on region boundaries

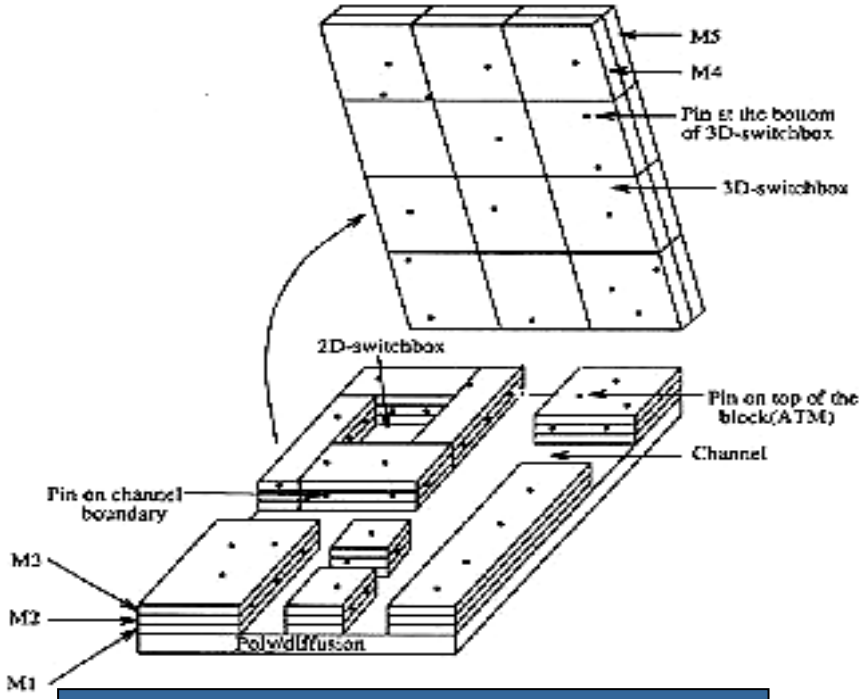




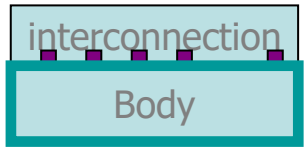


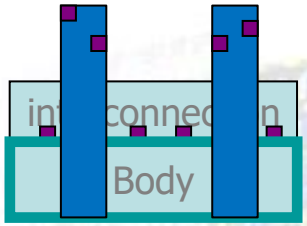
Introduction

- Custom-Based Design Layout Structure



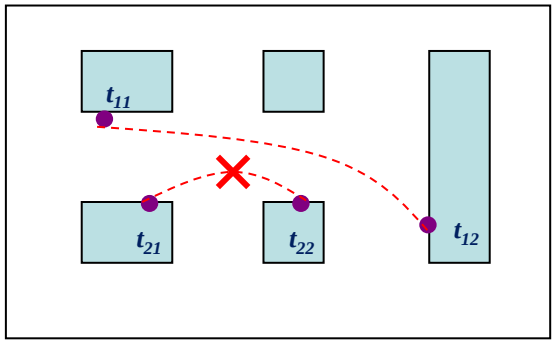
Three different routing regions

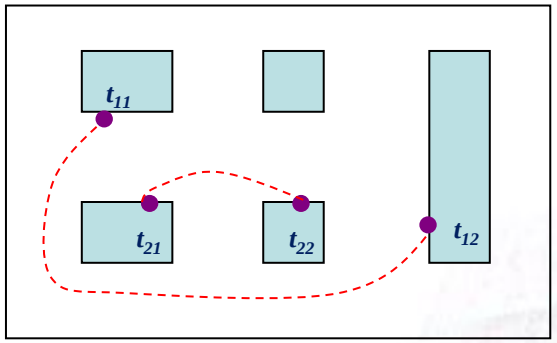




Introduction

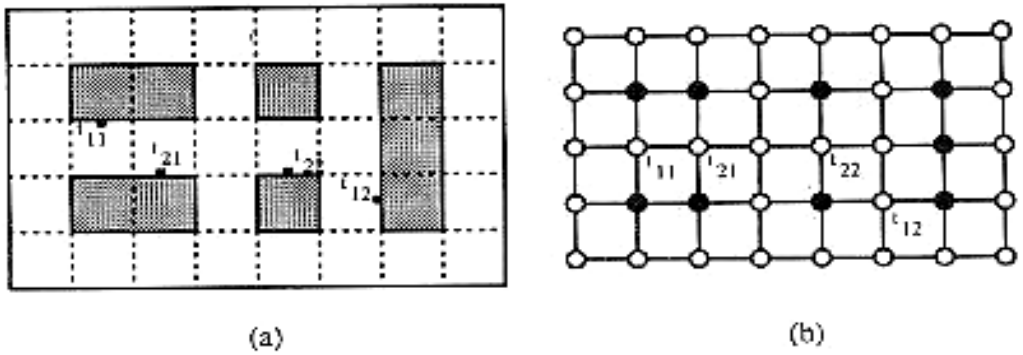
- Example for the trade-off between net routability and minimization of objective functions
  - Each channel has unit capacity
  - $N_1 = \{t_{11}, t_{12}\}, N_2 = \{t_{21}, t_{22}\}$





Problem Formulation

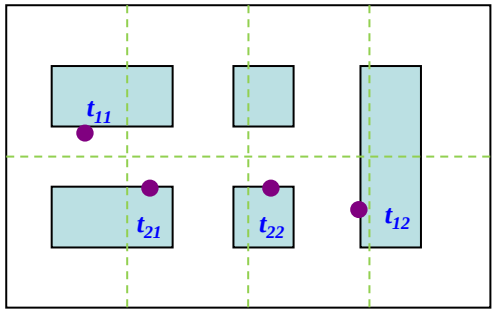
- Routing regions and their relationships and capacities are modeled as graphs
  - Grid Graph Model

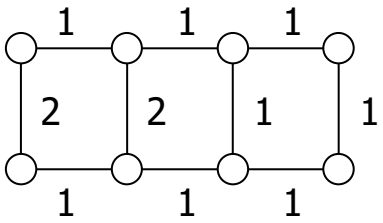


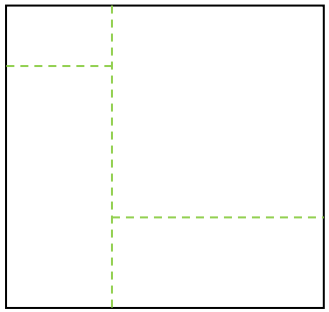
Grid Graph Model

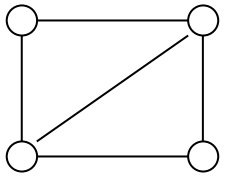
Problem Formulation

Checker Board Model

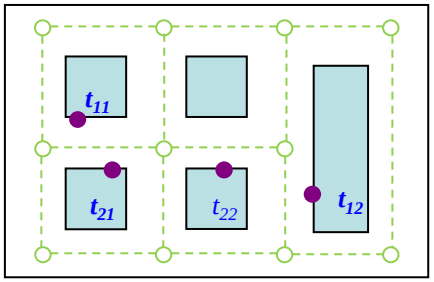


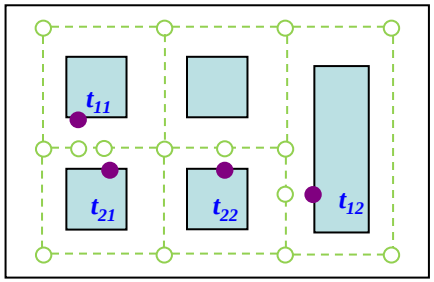


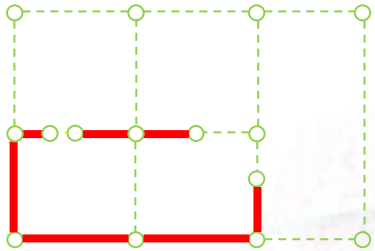




Channel Intersection Graph Model







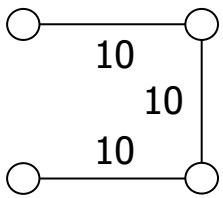
Extended Channel Intersection Graph

Problem Formulation

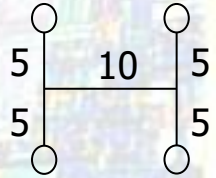
- Global routing – find a steiner tree for each net in the routing graph
  - Constraint – $w(e_i) \leq c(e_i)$ for each edge e_i, where $w(e_i)$ is the number of wires passing through the channel corresponding to e_i , and $c(e_i)$ is the channel capacity
  - Objective function – minimize *O, W, and vias*
  - For high-performance chip – minimize maximum of
    - Wire length
    - Diameter of a steiner tree – the maximum length of a path between any two vertices in the tree

$$overflow(e) = \begin{cases} d(e) - c(e), if\ d(e) > c(e) \\ 0 \qquad\quad , otherwise \end{cases}$$

$$TOF = \sum_{e \in E} overflow(e) \quad MOF = \max_{e \in E}(overflow(e))$$

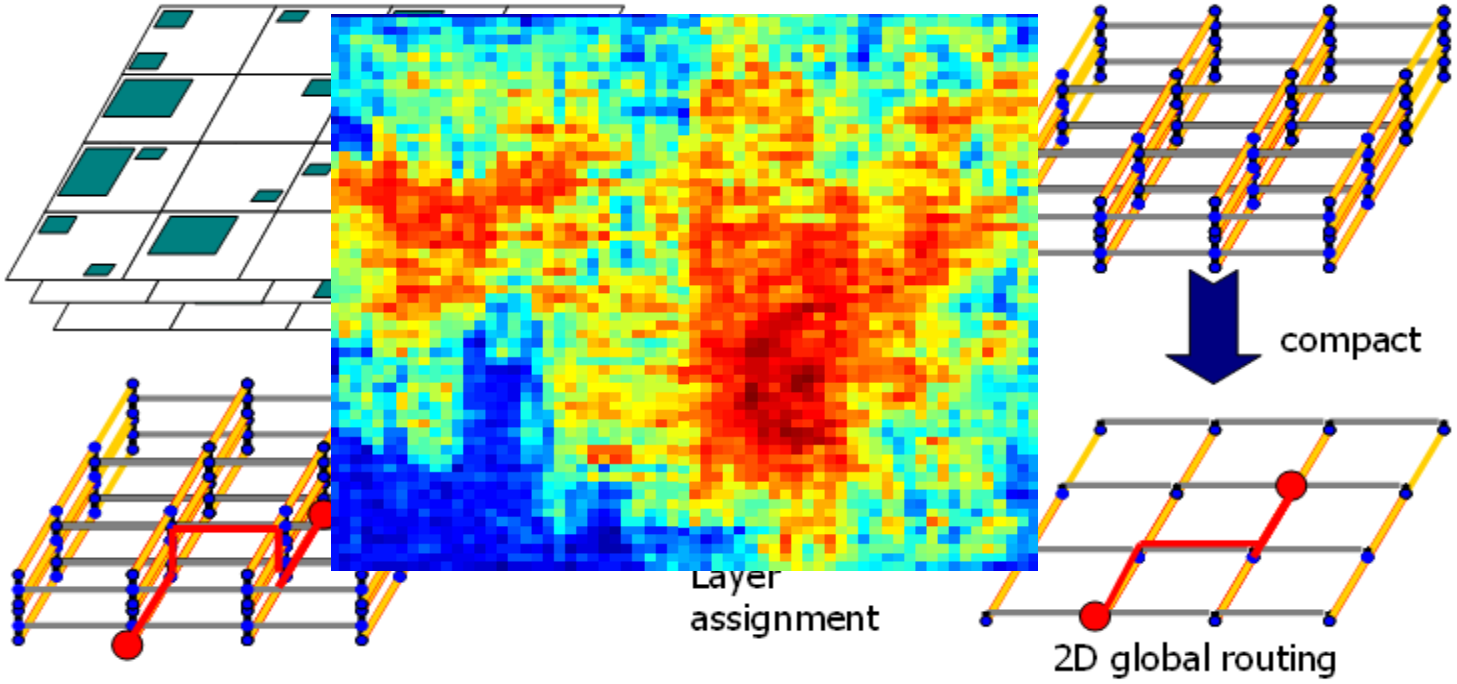


D1 = 30 > D2 = 20



Problem Formulation

- Modern global routing problem



Research Trends

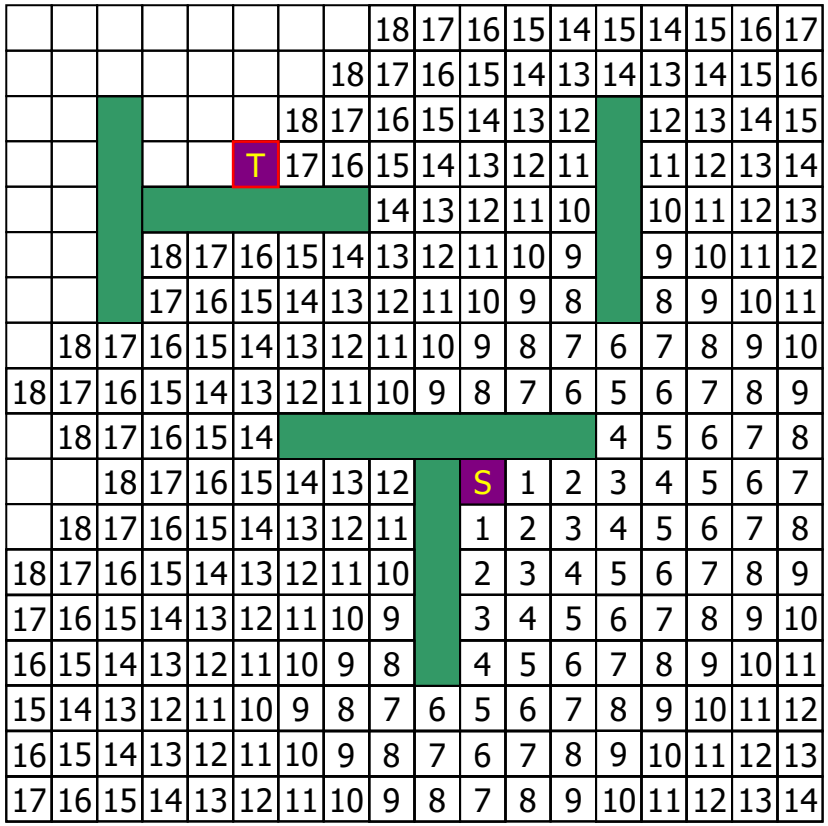
- **Recent publications in global routing**

Global router	Year	Main features
Fast-Route	ICCAD 06,08, ASPDAC 07,09	Logistic cost function, Monotonic routing, Virtual capacity
Maize-Router	ASPDAC 08	Extreme edge shifting
Box-Router	DAC 06, ICCAD 07	ILP-based box expansion
FGR	ICCAD 07	Discrete-Lagrange based routing
Archer	ICCAD 07	History-based cost function
NTHU-Route	ICCAD 08	New history based cost function
NTUgr	ASPDAC 09	Multiple forbidden-region expansion
NCTU GR	ASPDAC 09 (TVLSI 12)	Pseudo Random Ordering & Bi-Cost functions
GRIP	DAC 09	Cluster-computing-based global router
	DAC 10	A parallel IP approach to global routing
NCTU GR 2.0	DAC 10	Multi-threaded bounded length maze router
NCTU GR 3.0	ICCAD 11	Multiple supple voltage aware global router
MGR	ICCAD 11	MGR: Multi-Level Global Router
	ASP-DAC 11	Negotiation-Based LA for Via Count and Via Overflow Opt
NCTU GR 4.0 (Grace)	ICCAD 12	Hybrid Unilateral Monotonic Routing
	ISPD 12	Optimizing the antenna area and separators in LA
NCTU-IBM	DAC 13	Routing Congestion Estimation with Real Design Constraints

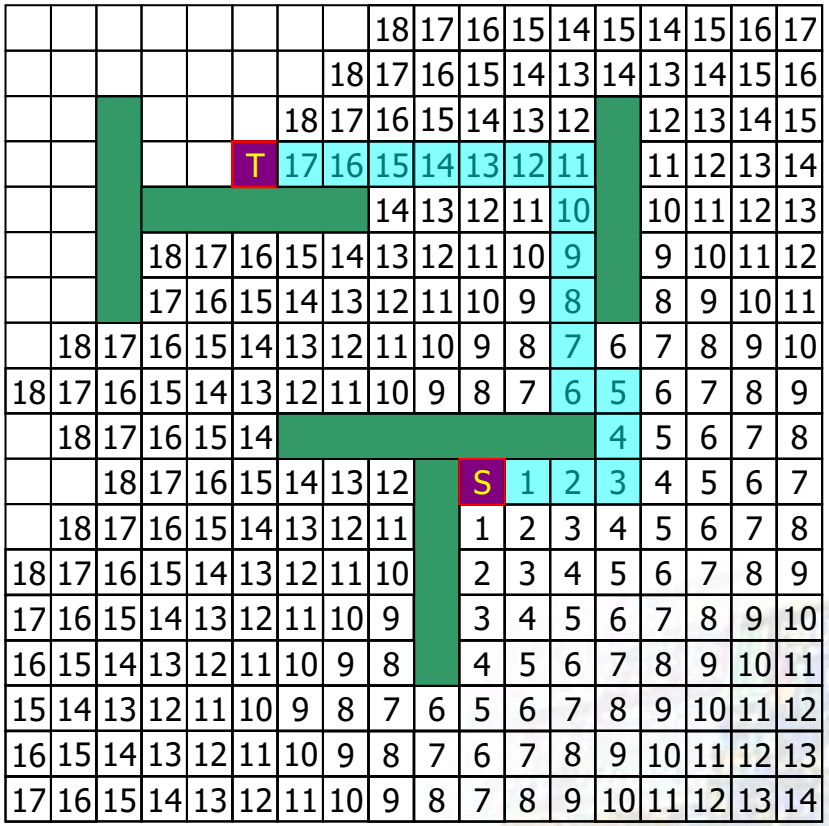
Grid-Based Algorithms

- Maze Routing Algorithms
  - Lee Algorithm – find a shortest path between two points (from one to the other) using breadth-first search
    - Three Steps
      - Wave propagation – label the un-labeled neighbors of cell i as cell $i+1$
      - Backtracing – select the cell that does not alter the direction
      - Label clearance – clear the label of all the cells except those cells on the backtracing path
    - Terminated conditions
      - The target point is found
      - All cells have been traversed
      - The length is greater than the upper bound on the path length
    - Timing complexity
      - wave propagation: $O(L^2)$, where L is the path length
      - backtracing: $O(L)$
    - Space complexity - $O(M \times N)$, where M is the row number and N is the column number

Grid-Based Algorithms



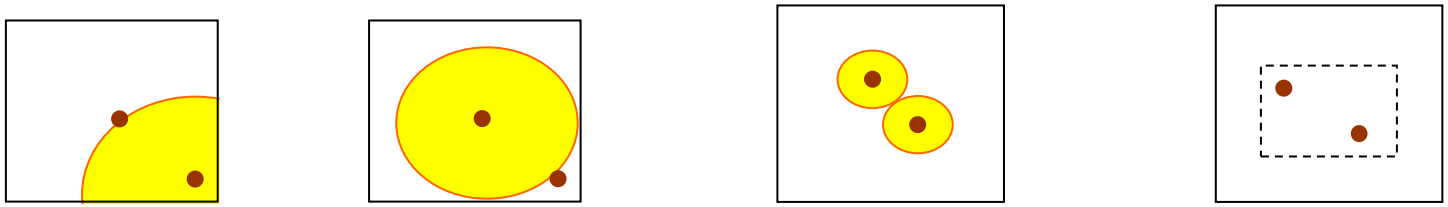
Wave propagation



Backtracing

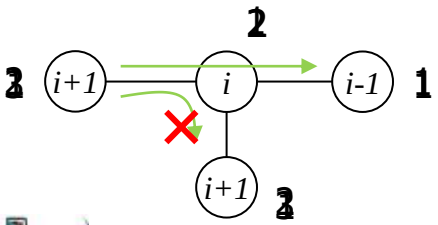
Grid-Based Algorithms

- Runtime Reduction

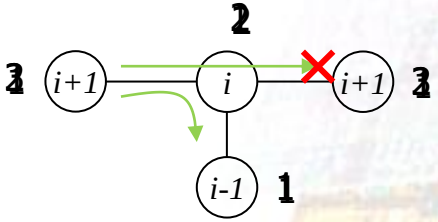


Start point selection | Double fanout | Framing

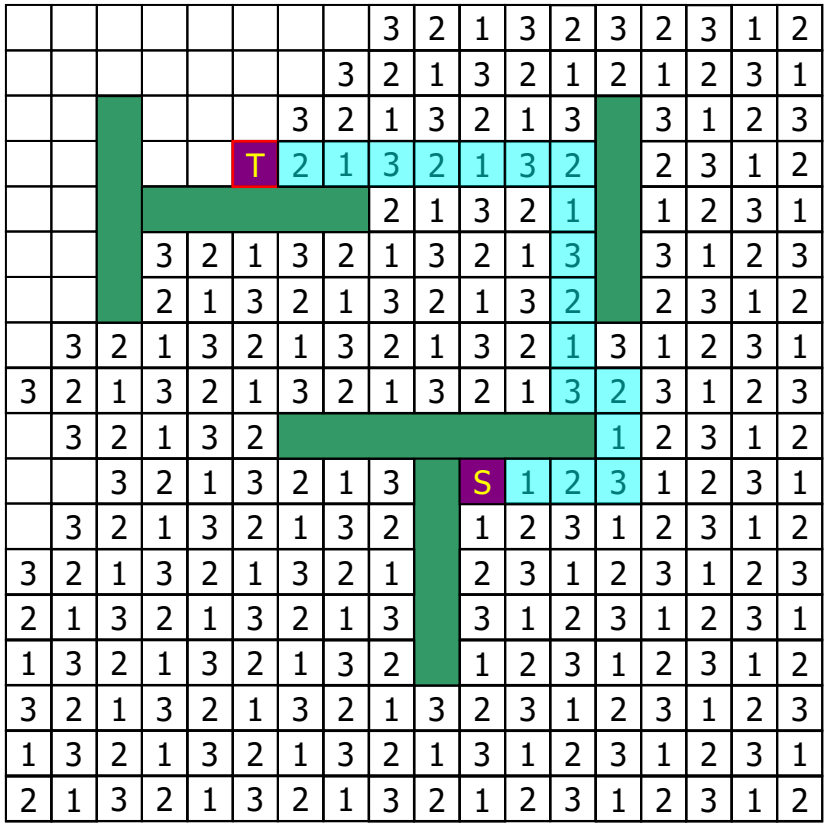
- Memory Reduction – cell states : {blocked, empty} + the states to distinguish predecessor and successor
  - Wave propagate by the sequence 1, 2, 3, 1, 2, 3, … - 5 states
  - Wave propagate by the sequence 1,1,2,2,1,1,2,2, … - 4 states



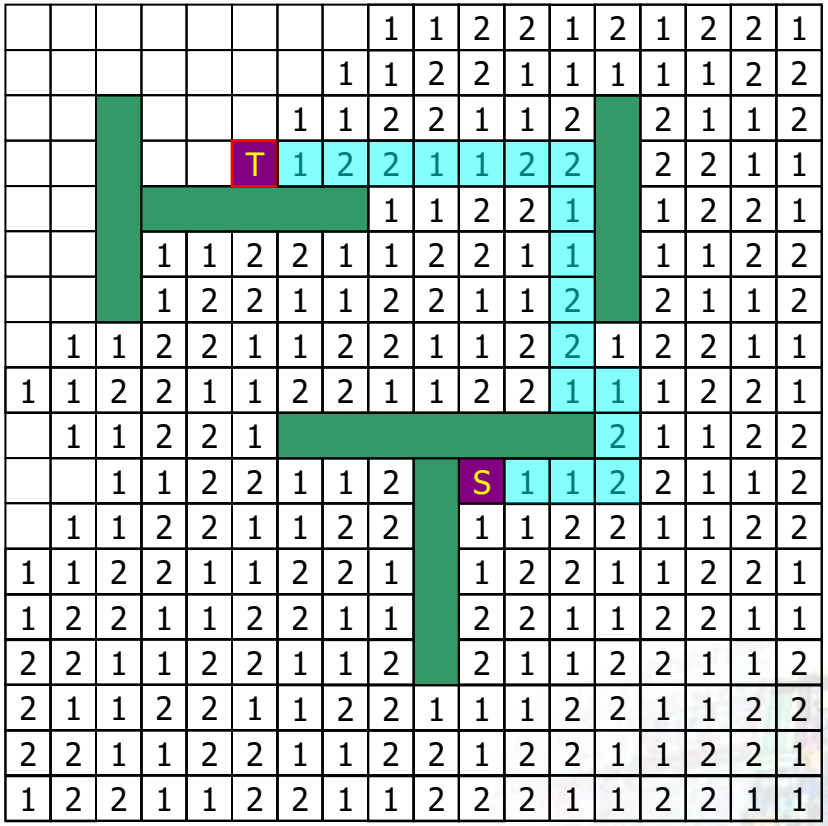
How to distinguish these two cases



Grid-Based Algorithms



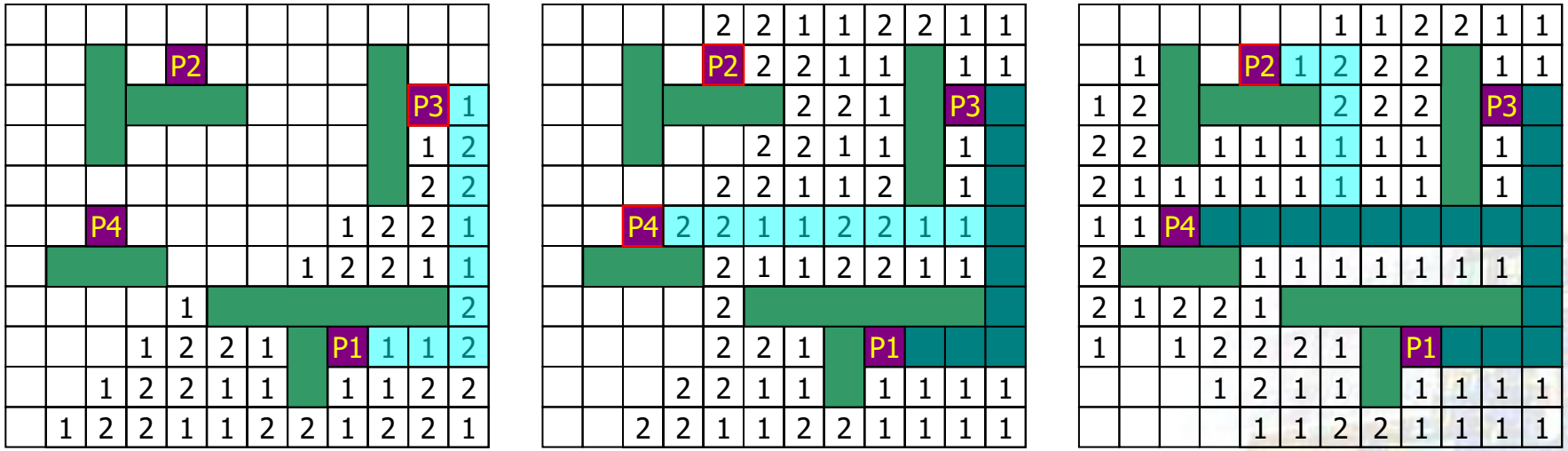
Wave propagation by the sequence 1,2,3,1,2,3,…



Wave propagation by the sequence 1,1,2,2,1,1,2,2…

Grid-Based Algorithms

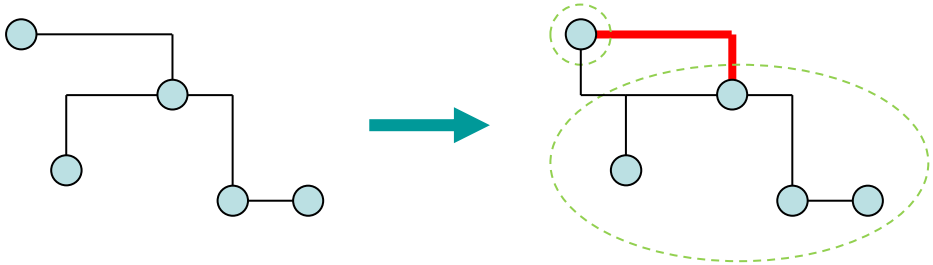
- Multi-terminal Net Routing – optimal steiner tree
  - Fast algorithm – find a steiner tree for n pins
    - Select an arbitrary pin as start and the other are targets
    - Use Lee algorithm to connect source to any one target
    - Select the connected component containing the initial start pin as start and repeat previous step
    - Repeat previous step until all pins are in one connected component



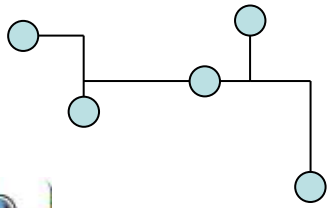
An example for multi-terminal net maze routing

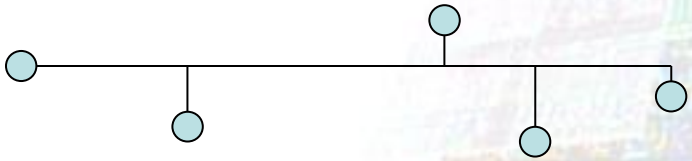
Grid-Based Algorithms

- Steiner tree improvement
  - Remove the segment connecting any two points, and re-route these two sub-trees



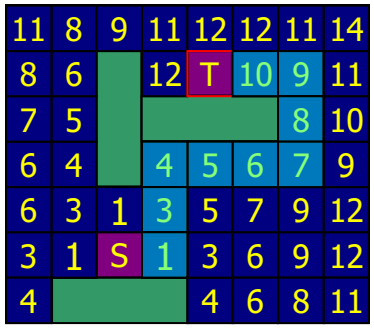
- Alternative Tree Topology
  - The closer pins are connected first
  - The further pins are connected first

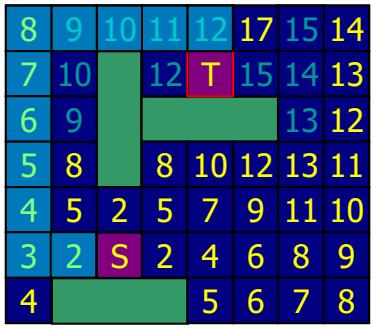


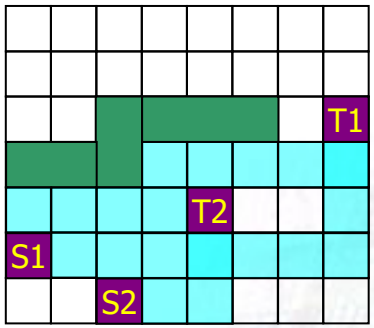


Grid-Based Algorithms

- Weighted Maze Routing – find a minimum-weight path rather than a shortest path
  - Main difference – the blocked cell will be mazed out again if later coming cost is less than the previous one
  - Application – maze routing for multiple nets







Grid-Based Algorithms

- Soukup's Algorithm
  - Combine depth-first search and breadth-first search
  - better speed than Lee algorithm by 10~50 times
  - not guarantee to find a shortest path
  - Flow
    - Use depth-first search to maze towards the target until reaching the target or obstacles or the move is away from the target
    - Use breadth-first search to escape from the obstacles to a location that can move towards the target

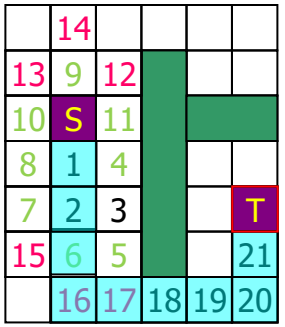
Grid-Based Algorithms

Algorithm Soukup_Algorithm(B,s,t,P)
begin
  $plist = s$; $nlist = \varnothing$; $order = 1$; $find\_path$ = **FALSE**;
  while $plist \neq \varnothing$
    for each vertex v_i in $plist$ **do**
      for each vertex v_j neighboring v_i **do**
        if $v_j = t$ { $L[v_j] = order$; $find\_path$ = **TRUE**; exit while;}
        if $B[v_j]$ = **UNBLOCKED then**
          if $DIR(v_i,v_j)$ = To_Target **then**
            $L[v_j] = order$; $order = order + 1$; $INSERT(v_j,plist)$;
            while $B[NGHBR\_IN\_DIR(v_i,v_j)]$ = UNBLOCKED && $DIR(v_i,v_j)$ = To_Target
              $v_k = v_j$; $v_j = NGHBR\_IN\_DIR(v_i,v_j)$; $v_i = v_k$;$L[v_j] = order$;
              $order = order + 1$; $INSERT(v_j,plist)$;
          else {$L[v_j] = order$; $order = order + 1$; $INSERT(v_j,nlist)$;}
    $plist = nlist$; $nlist = \varnothing$;
  if $find\_path$ = TRUE **then** { $RETRACE(L,P)$; } **else** path does not exist;
end

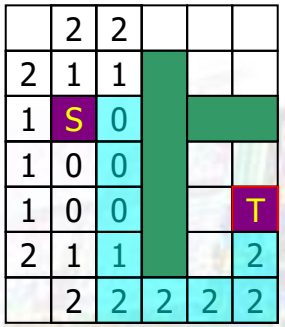
Grid-Based Algorithms

- Hadlock's Algorithm – label by detour number
  - Detour number of a path – the number of times that the path has turned away from the target

Algorithm Hadlock_Algorithm(B,s,t,P)
begin
  $plist = s$; $nlist = \varnothing$; $detour = 0$; $find\_path$ = **FALSE**;
  while $plist \neq \varnothing$
    for each vertex v_i in $plist$ **do**
      for each vertex v_j neighboring v_i **do**
        if $B[v_j]$ = **UNBLOCKED then**
          $D[v_j] = DETOUR\_NUMBER(v_j)$; $INSERT(v_j,nlist)$;
          if $v_j = t$ **then** { $find\_path$ = **TRUE**; exit while; }
    if $nlist = \varnothing$ **then** { $find\_path$ = **FALSE**; exit while ; }
    $detour = MINIMUM\_DETOUR(nlist)$;
    for each vertex v_k in $nlist$ **do**
      if $D[v_k] = detour$ **then** { $INSERT(v_k,plist)$; remove v_k from $nlist$; }
  if $find\_path$ = TRUE **then** { $RETRACE(L,P)$; } **else** path does not exist;
end



Soukup Algorithm

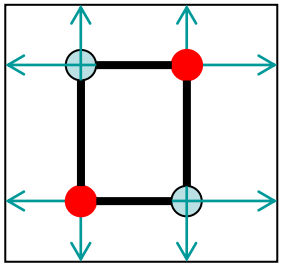


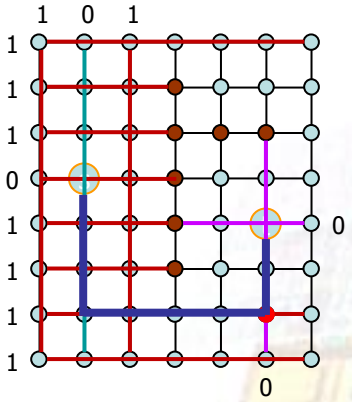
Hadlock Algorithm

n: forward expansion, n: the first set of escape expansion,
n: the second set of escape expansion

Grid-Based Algorithms

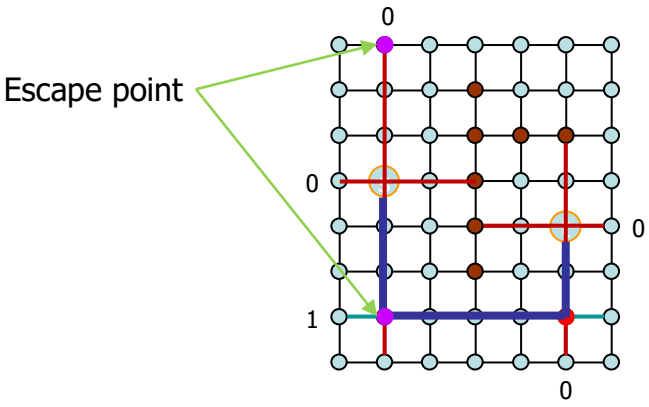
- Line-Probe/Search Algorithms
  - Mikami-Tabuchi Algorithm
    - Original Concept – without obstacles
      - Extend two perpendicular lines at both start and target points to get the intersection
    - How to work with existing obstacles
      - Repeatedly extend perpendicular extension lines at the base points on the existing extension lines until the extension lines from the start and target intersects





Grid-Based Algorithms

- Hightower's Algorithm
  - Escape point – the nearest point, that is not covered by the blockage, to the start terminal on the extension line
  - The next perpendicular extension line is only inserted on the escape point
- Comparison of different algorithms
  - Len: the path length, L: the number of extension lines



	Algorithms				
	Maze Routing			Line Probe	
	Lee	Soukup	Hadlock	Mikami	Hightower
Time Comp.	Len^2	Len^2	Len^2	L	L
Space Comp.	$h \times w$	$h \times w$	$h \times w$	L	L
Always find a path?	Yes	Yes	Yes	Yes	No
Shortest path?	Yes	No	Yes	Yes	No

Integer Programming Based Algorithm

- N: the number of nets, n_i: for net i, there are n_i possible trees $t_1^i, t_2^i, ..., t_{n_i}^i$ to route the net
- Steiner Tree Selection
  - *For each net i,*

$$x_{ij} = \begin{cases} 1 & \text{if net } i \text{ is routed according to tree } t_j^i \\ 0 & \text{otherwise} \end{cases}$$

$$\sum_{j=1}^{n_i} x_{ij} = 1$$

- Edge Capacity Constraint
  - M: total number of edges, T: total number of possible trees for all nets, $A_{M \times T} = [a_{i,p}]$: 0-1 matrix

Integer Programming Based Algorithm

- $T = \sum_{i=1}^{N} n_i, \quad p = \sum_{m=1}^{v-1} n_m + u$

Net1	net2	net3
1,2,3	1,2	1,2,3,4
1,2,3	4,5	6,7,8,9

$$a_{i,p} = \begin{cases} 1 & \text{if edge } i \text{ belongs to tree } t_u^v \\ 0 & \text{otherwise} \end{cases}$$

$$For\ each\ edge\ i, 1 \le i \le M$$

$$\sum_{v=1}^{N} \sum_{u=1}^{n_v} a_{i,p} \times x_{vu} \le c_i$$

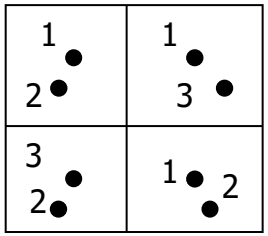
- Objective Function
  - g_{ij}: the cost of tree t_j^i, $F = \sum_{i=1}^{N} \sum_{j=1}^{n_i} g_{ij} \times x_{ij}$

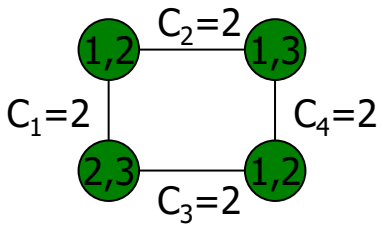
Integer Programming Based Algorithm

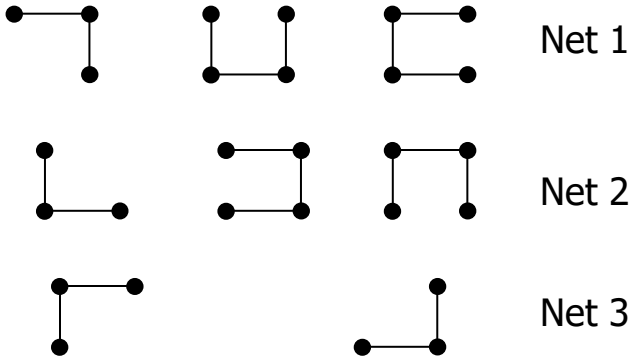
- 0-1 integer programming formulation

$$\begin{cases} \min \quad \sum_{i=1}^{N}\sum_{j=1}^{n_i} g_{ij} \times x_{ij}, \quad subject\ to \\ \sum_{j=1}^{n_i} x_{ij} = 1 & 1 \le i \le N \\ \sum_{v=1}^{N}\sum_{u=1}^{n_v} a_{i,p} \times x_{vu} \le c_i & 1 \le i \le M \\ x_{ij} = 0, 1 & 1 \le i \le N, \text{ and } 1 \le j \le n_k \end{cases}$$

Example







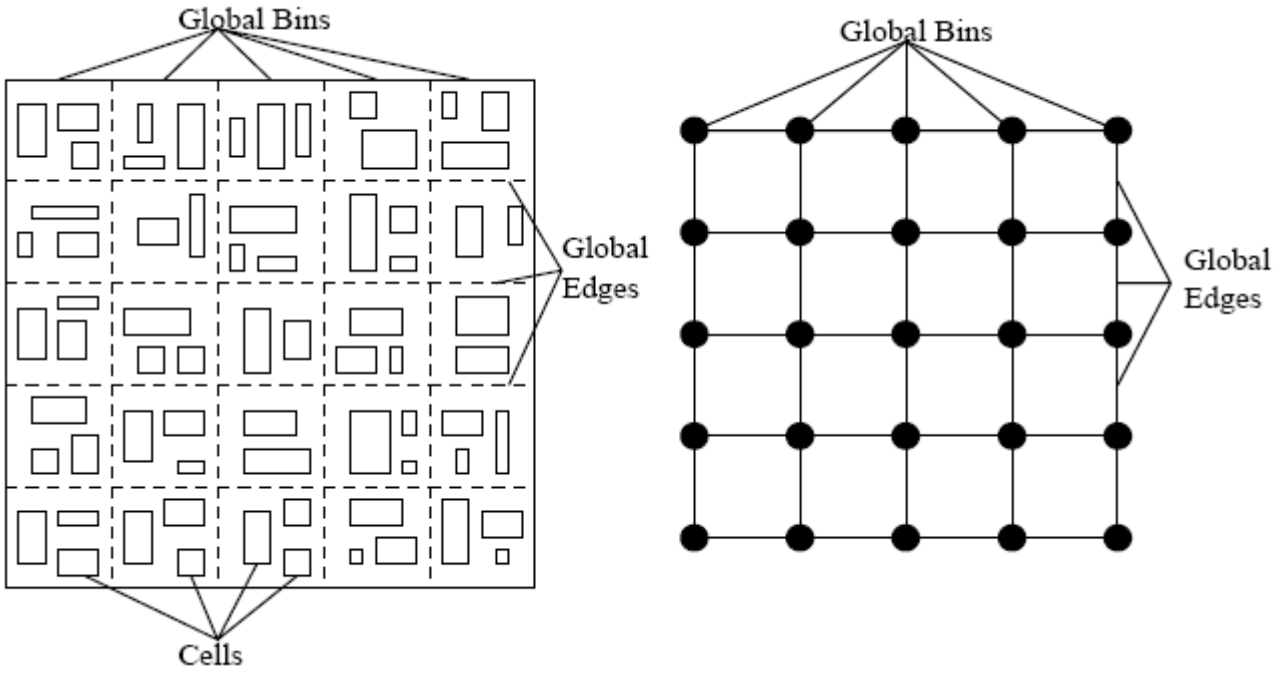
	t_1^1	t_1^2	t_1^3	t_2^1	t_2^2	t_2^3	t_3^1	t_3^2
1	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**0**
2	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
3	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**1**
4	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**

$g_{1,1}$=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

$$
\begin{cases}
F = 2x_{1,1} + 3x_{1,2} + 3x_{1,3} + 2x_{2,1} + 3x_{2,2} + 3x_{2,3} \\
\quad + 2x_{3,1} + 2x_{3,2} \leftarrow minimize \\
Subject\ to: \\
x_{1,1} + x_{1,2} + x_{1,3} = 1 \\
x_{2,1} + x_{2,2} + x_{2,3} = 1 \\
x_{3,1} + x_{3,2} = 1 \\
x_{1,2} + x_{1,3} + x_{2,1} + x_{2,3} + x_{3,1} \le 2 \\
x_{1,1} + x_{1,3} + x_{2,2} + x_{2,3} + x_{3,1} \le 2 \\
x_{1,2} + x_{1,3} + x_{2,1} + x_{2,2} + x_{3,2} \le 2 \\
x_{1,1} + x_{1,2} + x_{2,2} + x_{2,3} + x_{3,2} \le 2 \\
x_{i,j} = 0,1 \quad 1 \le i \le 3,\ 1 \le j \le 3
\end{cases}
$$

Predictable Routing

- Routing model – grid graph



R. Kastner, E. Bozogzadeh, and M. Sarrafzadeh. “Predictable routing.” In Proc. IEEE/ACM Intl. Conf. Computer-Aided Design, 2000.

Predictable Routing

- Congestion definition –

$$overflow_e = \begin{cases} d_e - c_e - t & \text{if } d_e > c_e \\ 0 & \text{otherwise} \end{cases}$$

- Overflow –

$$overflow = \sum_{e=1}^{n} overflow_e$$

- Global maze routing cost

$$overflow_{route} = \sum_{e \in RouteEdges} overflow_e$$

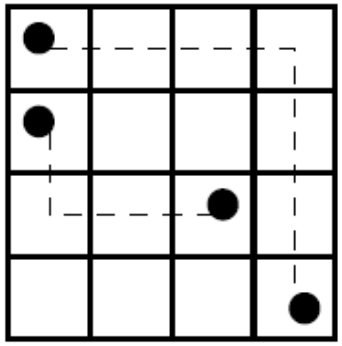
$$length_{route} = \mid RouteEdges \mid$$

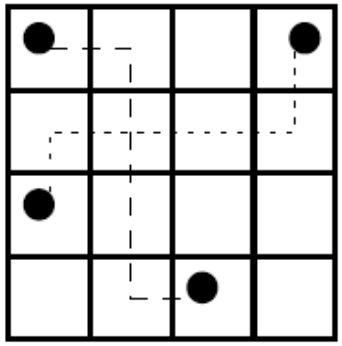
$$cost_{route} = \alpha \times overflow_{route} + length_{route}$$

$$cost_{total} = \sum_{allnets} cost_{route}$$

Predictable Routing

- Pattern Routing





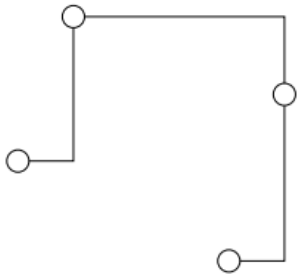
- Complexity Analysis

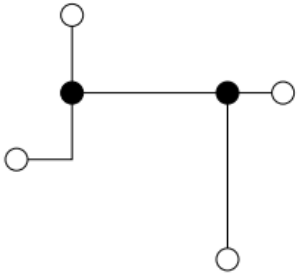
1. Given a net $n = \{(x_1, y_1), (x_2, y_2)\}$ and a grid graph G(V,E).
2. Let A be the edges on and within the bounding box of n. A $\subseteq$ E. $|A| = 2 \cdot |x_1 - x_2| \cdot |y_1 - y_2| + |x_1 - x_2| + |y_1 - y_2|$
3. Let P be the edges on the bounding box of n. P $\subseteq$ A. $|P| = 2 \cdot (|x_1 - x_2| + |y_1 - y_2|)$
4. Maze routing - $O(|E|)$
5. L-shaped pattern routing - $O(|P|)$
6. Z-shaped pattern routing - $O(|A|)$

- Theorem. $|P| \leq |A| \leq |E|$

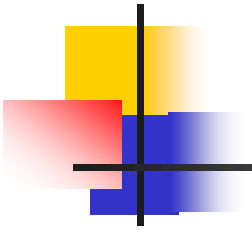
Rectilinear Steiner Minimum Tree

- Minimum Spanning Tree (MST) and Steiner Minimum Tree (SMT)





- Rectilinear Steiner Minimum Tree (RSMT) – one kind of SMT with rectilinear wire segments



FastRoute Algorithm

Phase 1: Congestion map generation

1) RSMT construction – FLUTE
2) 50% probability L-shaped pattern routing

Phase 2: Congestion-driven Steiner tree construction

1) Congestion-driven Steiner tree topology generation
2) Edge shifting

Phase 3: Pattern routing and Maze routing

1) Z-shaped pattern routing
2) Maze routing based on logistic cost function

Min Pan and Chris Chu, "FastRoute: A Step to Integrate Global Routing into Placement." IEEE/ACM International Conference on Computer-Aided Design, 2006

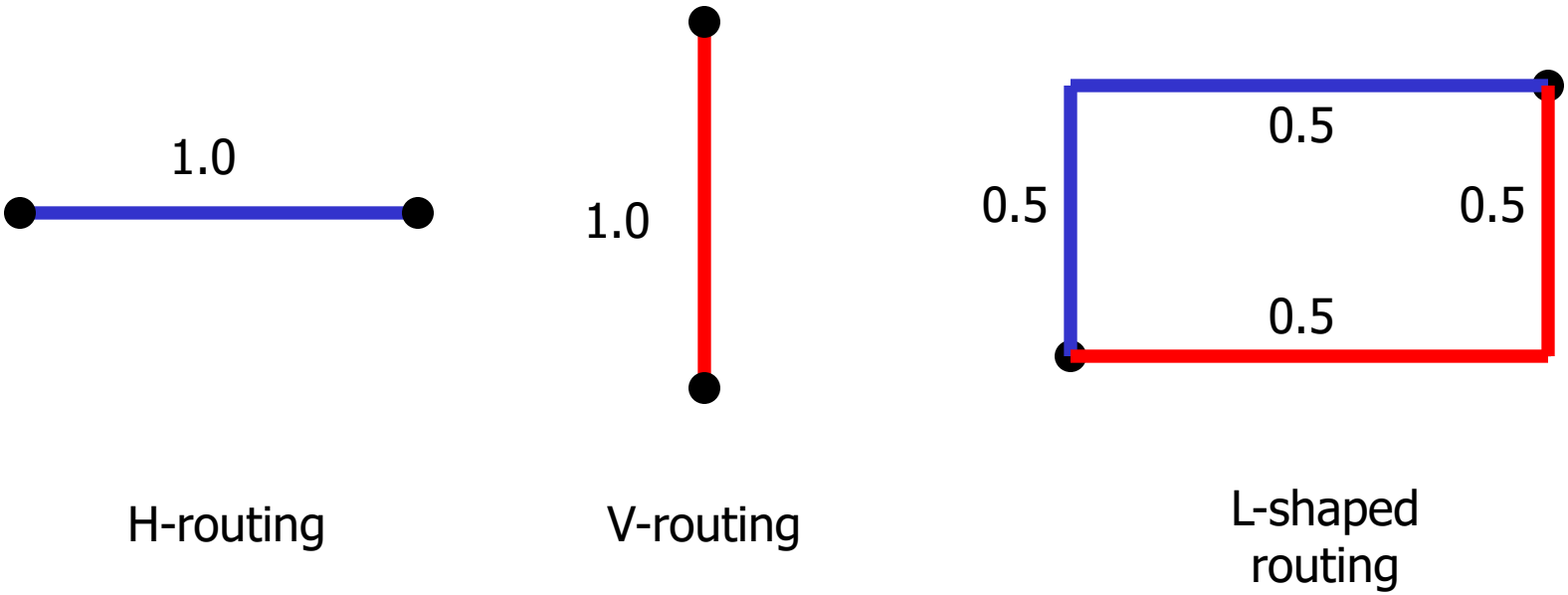
The following slides are Pan's presentation slides in ICCAD 2006

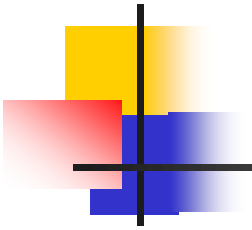
Congestion Map Generation (1)

- FLUTE (Fast LookUp Table Estimation) [ICCAD 04, ISPD 05]
  - An extremely fast and accurate Steiner Tree algorithm
- Very suitable for VLSI applications:
  - Optimal up to degree 9
  - Very accurate up to degree 100
  - 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

Congestion Map Generation (2)

- Break RSMT (Rectilinear Steiner Minimum Tree) generated by FLUTE into 2-pin nets
- Route all 2-pin nets with H-routing, V-routing and 50% probability L-shaped routing





FastRoute Algorithm

Phase 1: Congestion map generation

1) RSMT construction – FLUTE
2) 50% probability L-routing

Phase 2: Congestion-driven Steiner tree construction

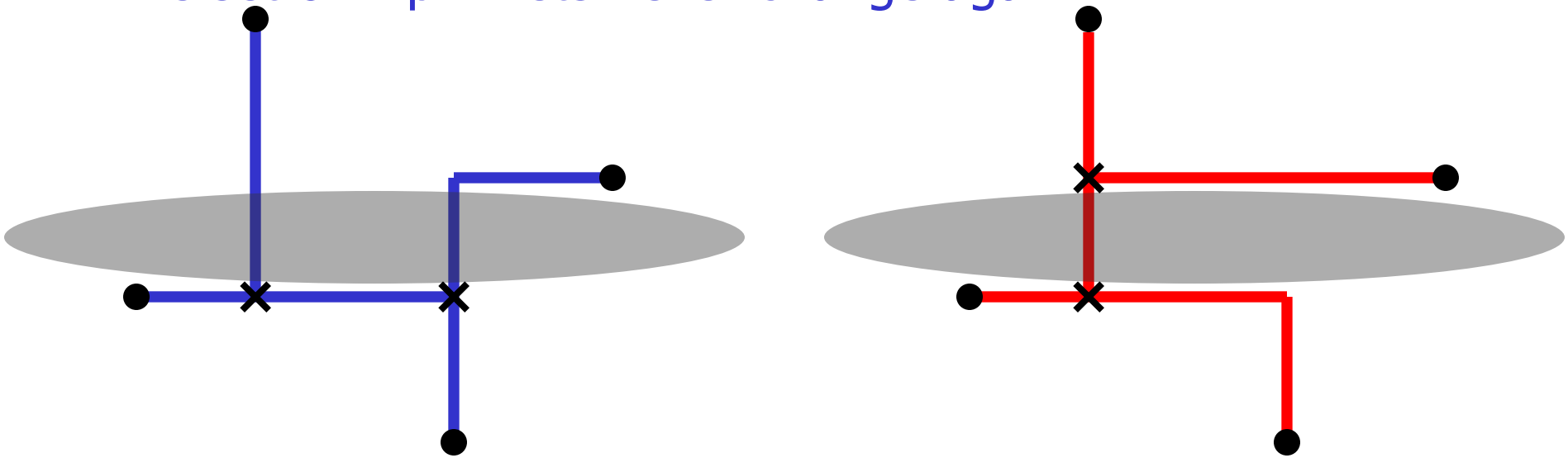
1) Congestion-driven Steiner tree topology generation
2) Edge shifting

Phase 3: Pattern routing and Maze routing

1) Z-routing
2) Maze routing based on logistic cost function

Limitation of Traditional Global Routing

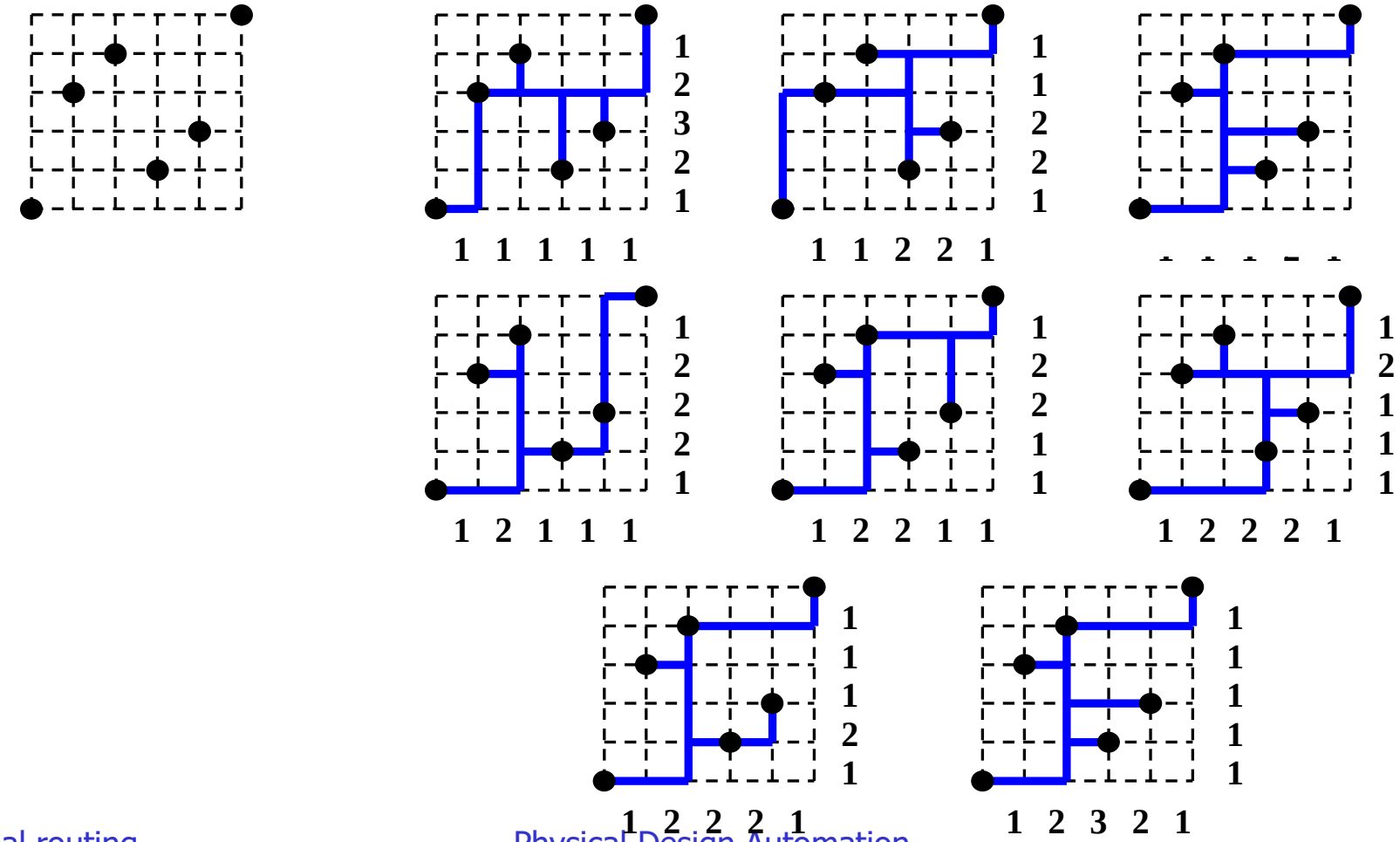
- Routing trees are constructed by RMST (Rectilinear Minimum Spanning Tree) or RSMT
  - Wirelength is used as criterion
  - No congestion consideration
- Trees broken down into 2-pin nets
- The set of 2-pin nets never change again



- Observation: Tree structures have great impact on congestion

Congestion-Driven Topology Generation (1)

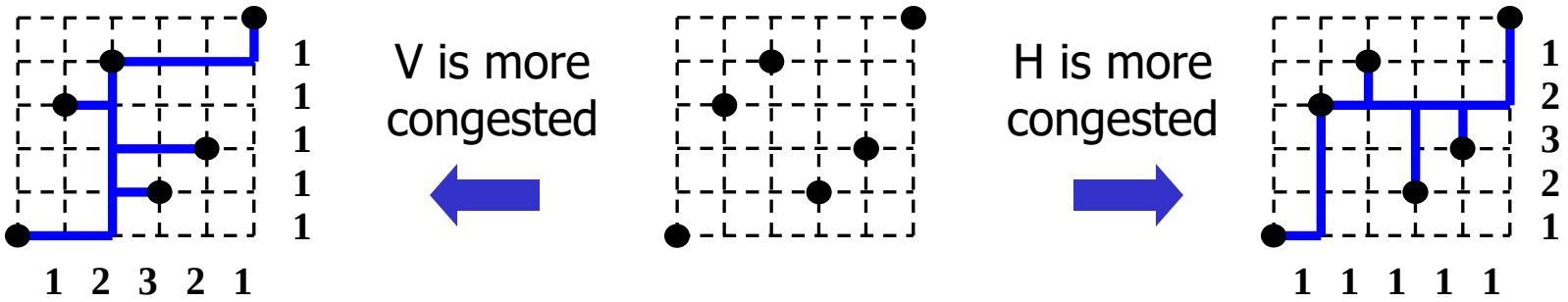
- Different topologies have very different routing demand distribution



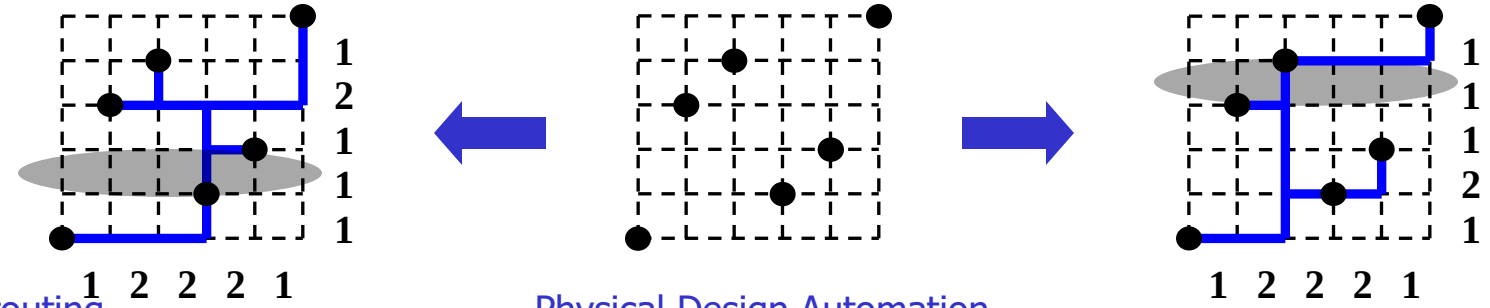
Congestion-Driven Topology Generation (2)

Change of topology is very powerful in resolving congestions

1. Balance the routing demand in H and V direction

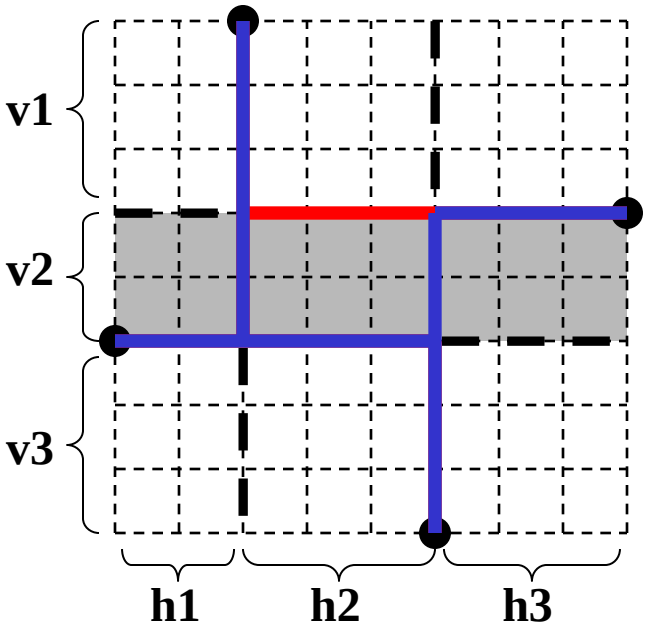


2. Shift routing demand away from the congested region in the same direction

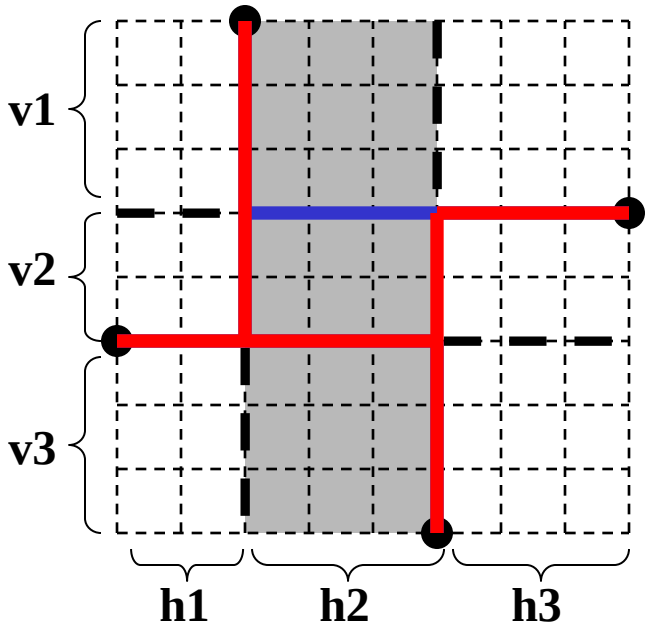


Congestion-Driven Topology Generation (3)

- Main idea:
  - Encourage to use topologies with less routing demand in congested row or column regions
  - Weight the segment length according to congestion and apply *FLUTE* to find the best topology



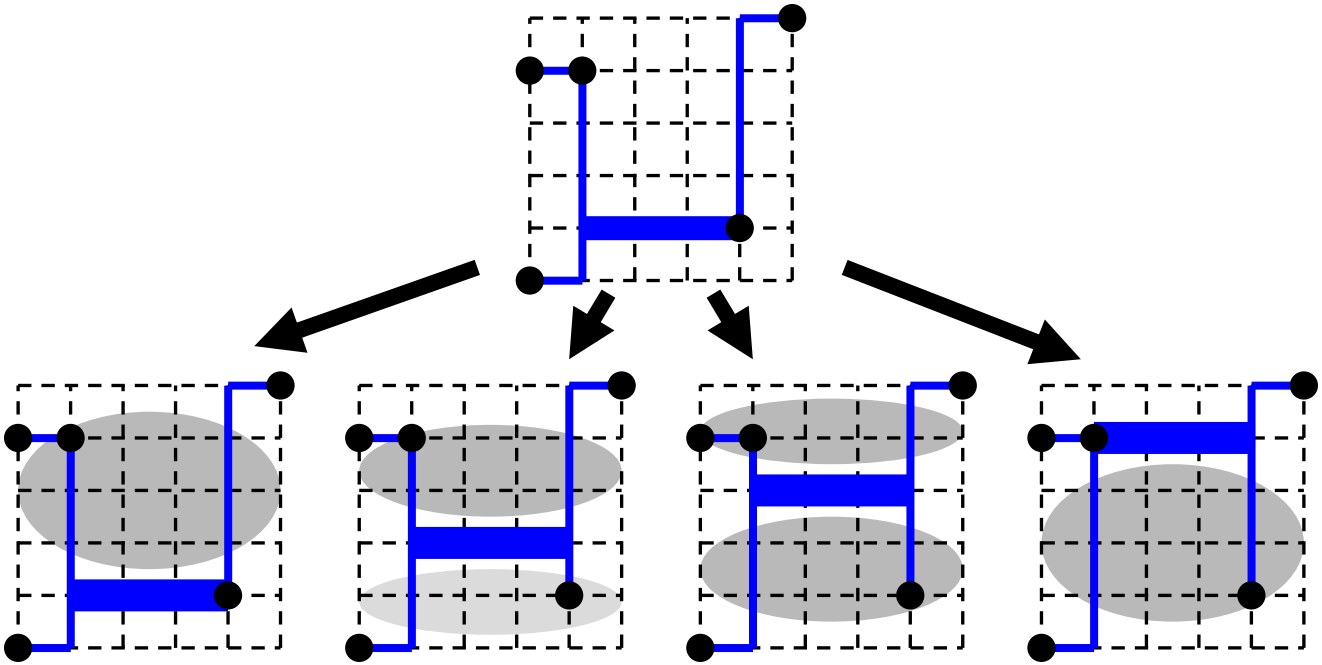
Row region corresponding to v2



Column region corresponding to h2

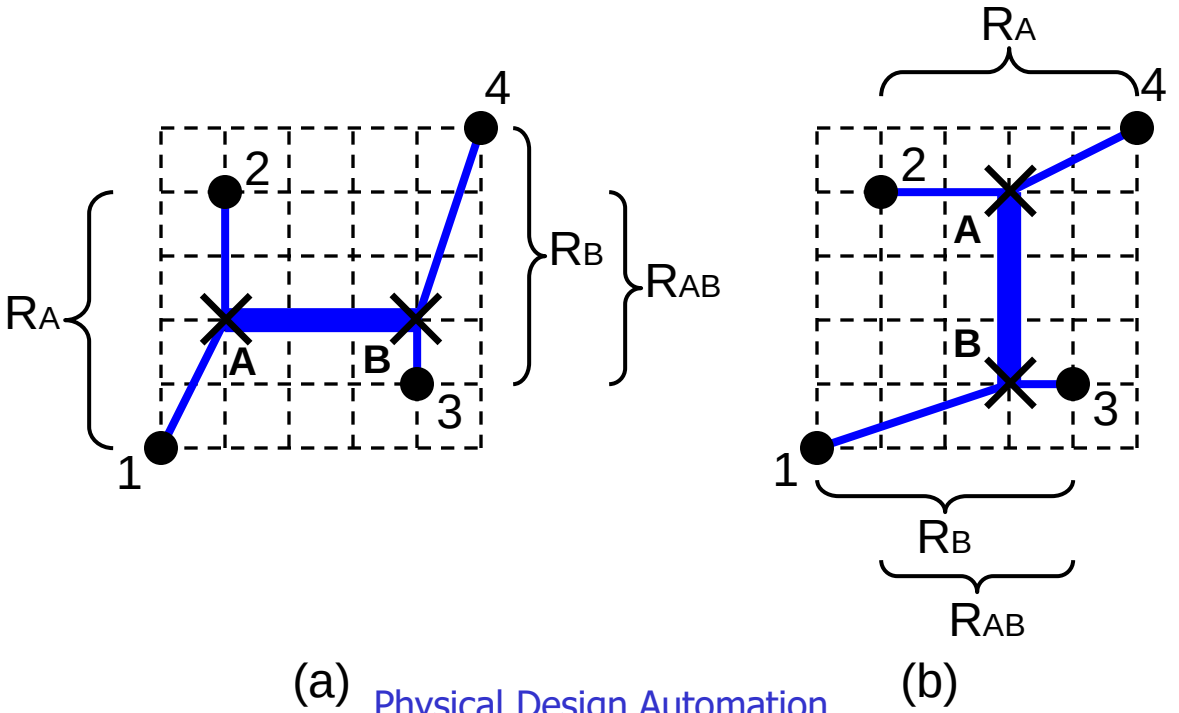
Edge Shifting (1)

- After fixing Steiner tree topology, there is still room to improve congestion
- Shifting some H-edges or V-edges in the tree can improve congestion without changing wirelength



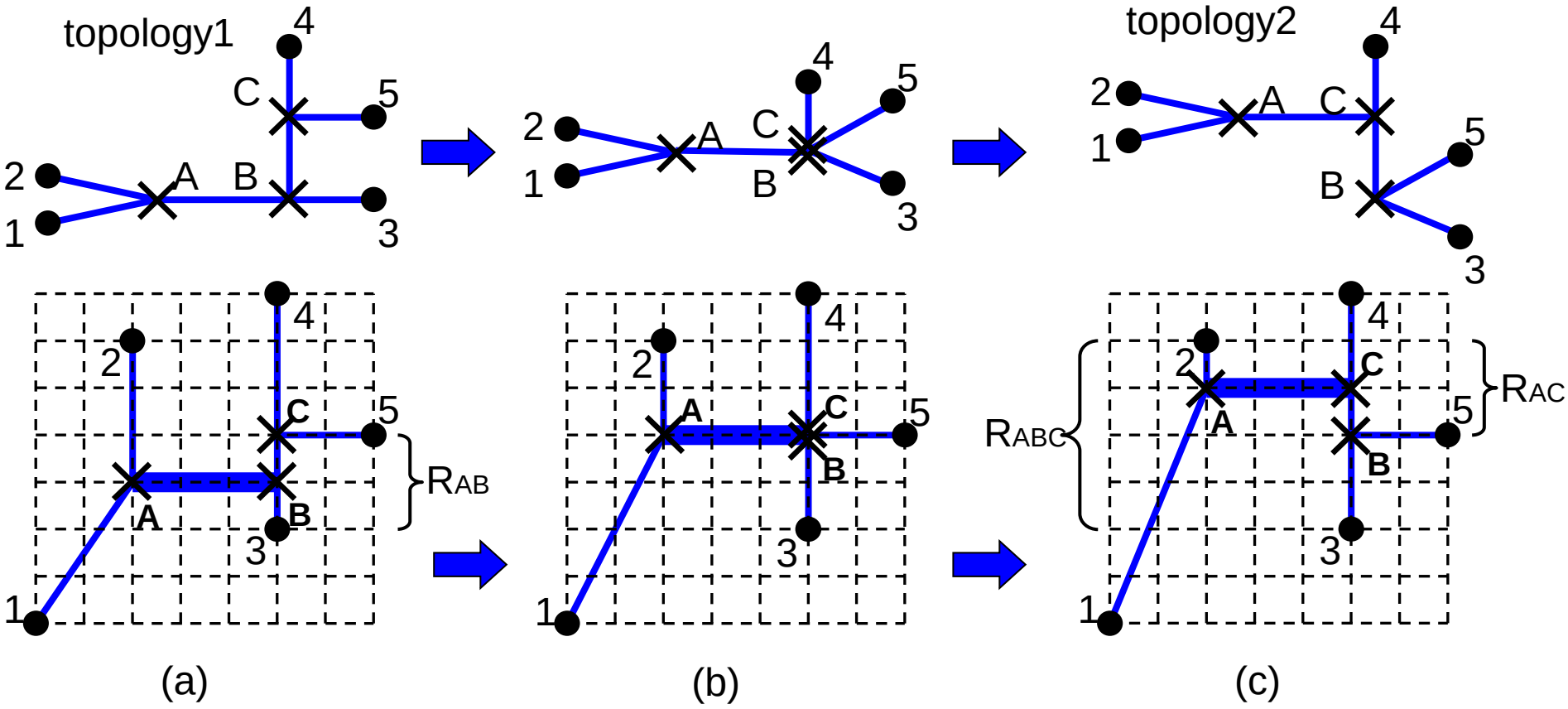
Edge Shifting (2)

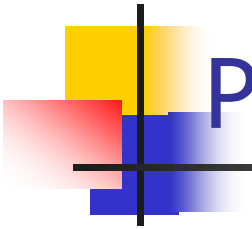
- Safe range: the range to shift a tree edge without changing the wirelength of the tree
- Used to find the best position of a tree edge
- Is the maximum common sliding range for the pair of Steiner nodes without changing wirelength



Edge Shifting (3)

- May need to exchange the Steiner nodes in original topology to explore the full safe range





Phase 2 Flow

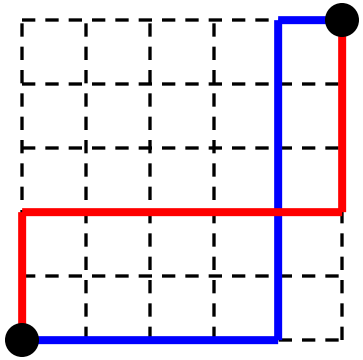
Phase 2: Congestion-driven Steiner tree construction

For each net *n*

1. Remove the routing demand caused by *n*
2. Reconstruct the topology of *n* considering the current congestion map
3. Apply *Edge-shifting* to further improve congestion
4. Route each 2-pin net in the Steiner tree for *n* by L-routing
5. Add the routing demand of *n* to congestion map

Phase 3: Pattern Routing and Maze Routing

- Pattern routing (L-shape, Z-shape)
  - Very fast
    - L-shape: O(m+n) (m, n are the grids a net spans)
    - Z-shape: O(mn)
  - May not find the optimal path
- Maze routing
  - Much slower O(mn•log(mn))
  - Able to find the optimal path
- Phase 3 flow:
  - Round 1:
    - Z-shaped pattern routing
  - Round 2:
    - Long nets pass congested region: Maze routing
    - Other nets: Z-shaped pattern routing

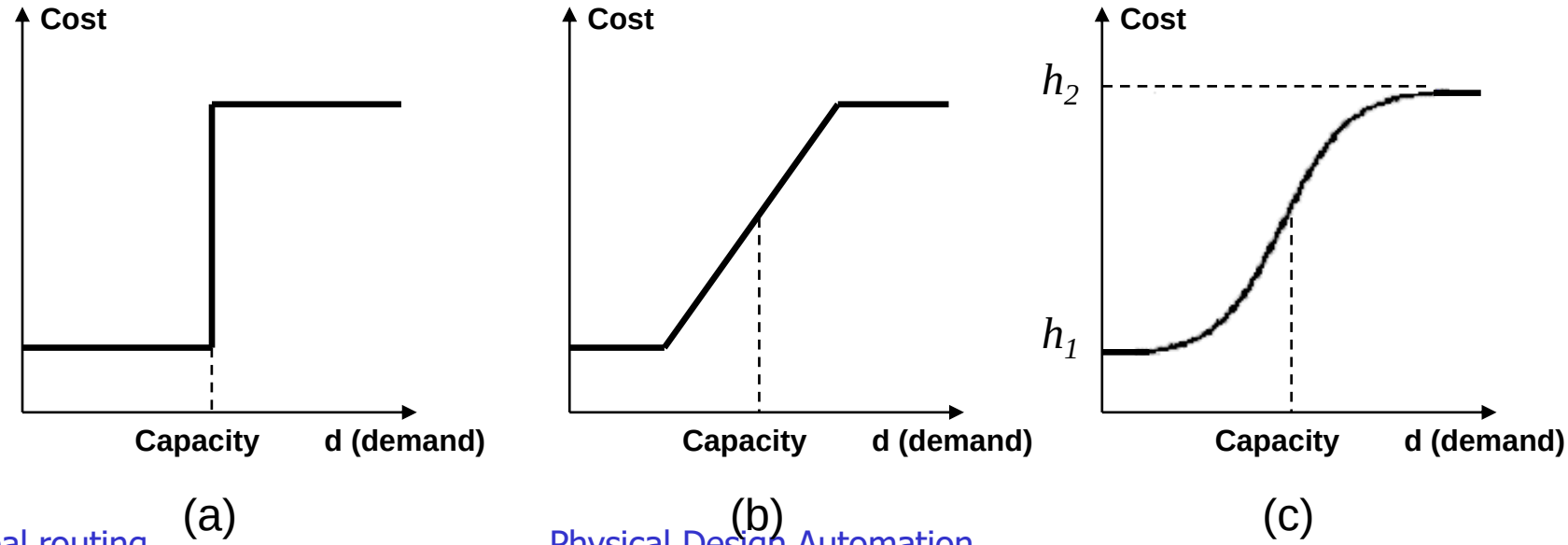


Z-shaped pattern routing

Cost Function

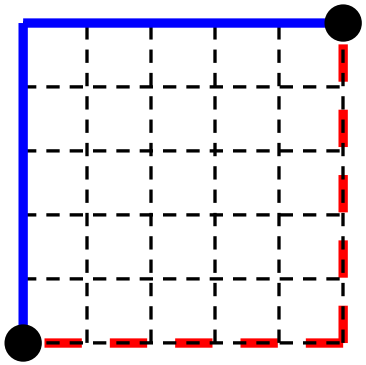
- Cost function is used to capture the cost to use a specific global edge in the grid graph
  - Abrupt function (Labyrinth)
  - Piece-wise linear function (Chi Dispersion Router)
  - Logistic function (FastRoute)

$$Cost(d) = h_1 + \frac{h_2}{1 + e^{-k(d - Capacity)}}$$

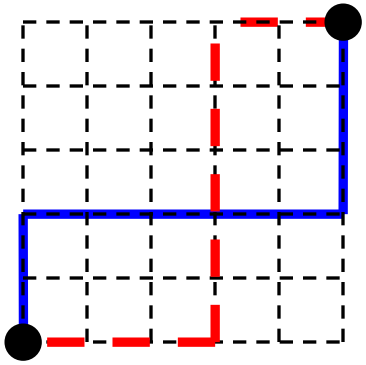


Pattern Routing (FastRoute 2.0)

- Use predefined patterns to route 2-pin nets



L-Shaped



Z-Shaped

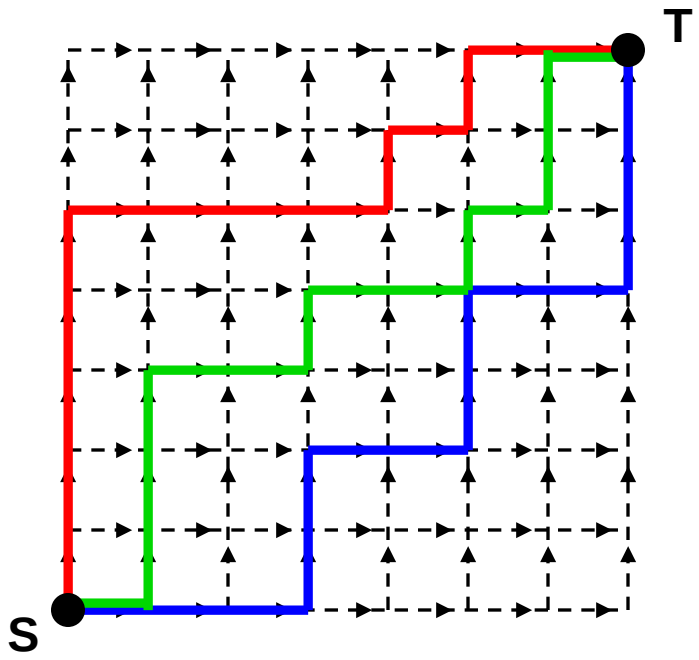
- Pros
  - Speed up the routing procedure
  - Constant # bends
- Cons
  - Very limited # paths being searched
  - Solution quality could be much worse than maze routing

Min Pan and Chris Chu, FastRoute 2.0: A High-quality and Efficient Global Router. Asian and South Pacific Design Automation Conference, pages 250-255, 2007. The following slides are Pan's presentation slides in ASP-DAC 07

Monotonic Routing (1)

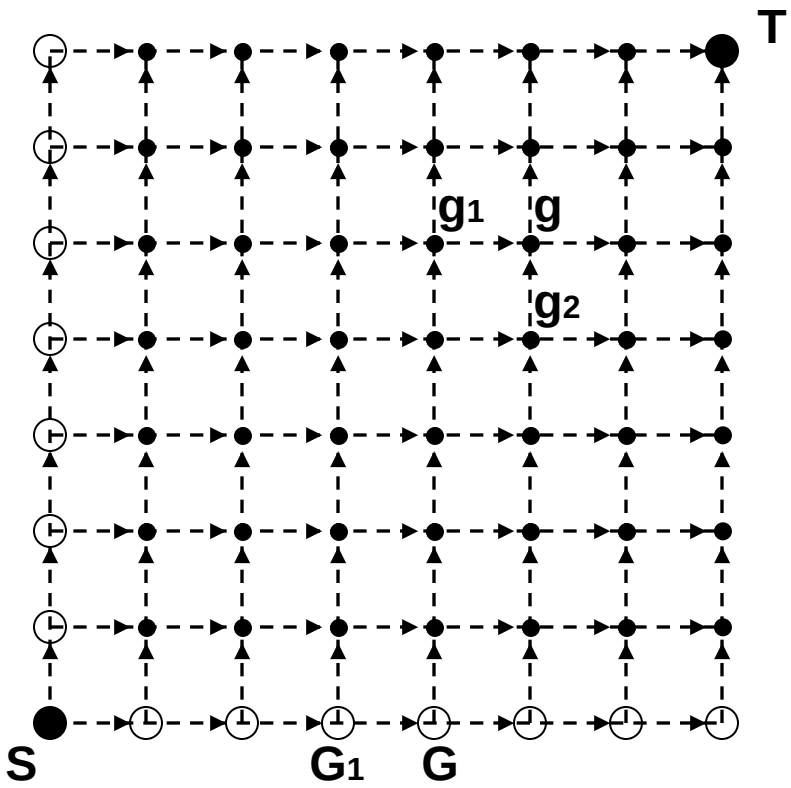
- Routing 2-pin nets (S to T)
- The routing path is monotonic from S to T
- # paths (m×n grids)
  - L-pattern: 2
  - Z-pattern: m+n-2
  - Monotonic

$$\binom{m+n-2}{m-1} = \frac{(m+n-2)!}{(m-1)!(n-1)!}$$



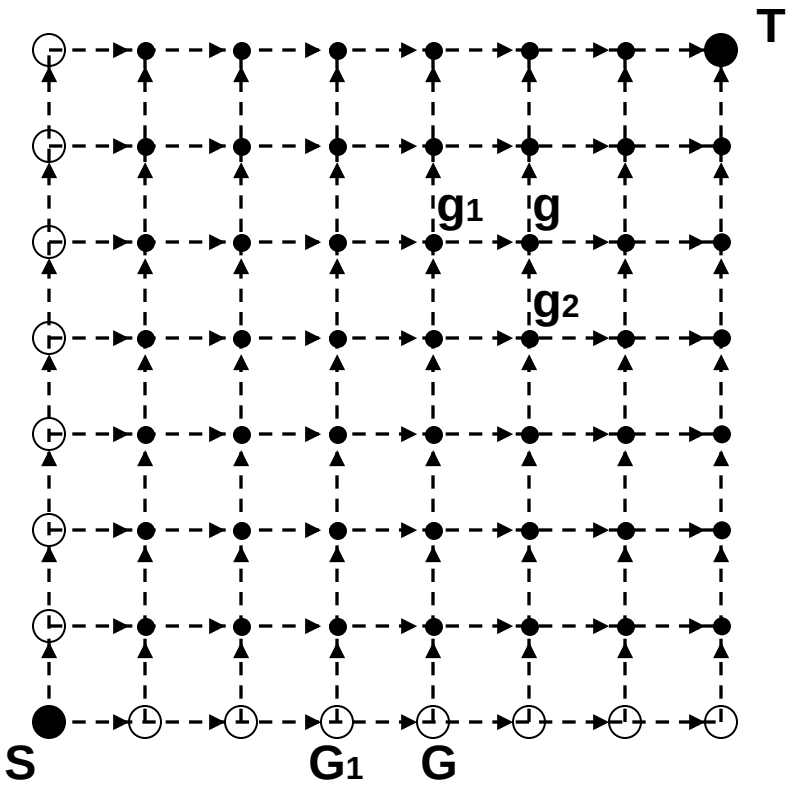
Monotonic Routing (2)

- Least cost monotonic routing path from S to T
  - For any grid points, the least cost monotonic routing path can be found easily
  - Dynamic programming to find the least cost monotonic path
  - Complexity: O(mn)
    same as Z-pattern routing

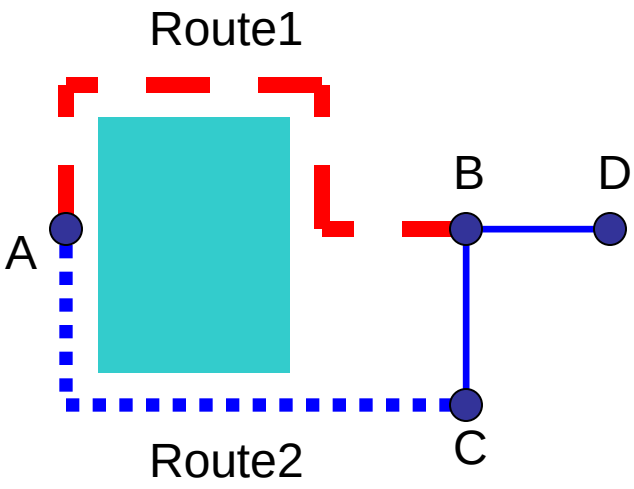


Monotonic Routing Algorithm

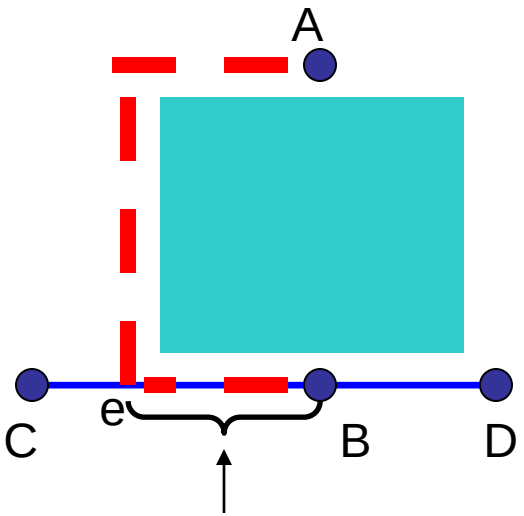
1. d(S) = 0
2. **for** x = 1 to m
3. G = (x, 0), G_1 = (x-1, 0)
4. d(G) = d(G_1) + cost(G, G_1), π(G) = G_1
5. **for** y = 1 to n
6. G = (0, y), G_1 = (0, y-1)
7. d(G) = d(G_1) + cost(G, G_1), π(G) = G_1
8. **for** x = 1 to m
9. **for** y = 1 to n
10. g = (x, y)
11. g_1 = (x-1, y), g_2 = (x, y-1)
12. **if** d(g_1)+cost(g, g_1) < d(g_2)+cost(g, g_2)
13. d(g) = d(g_1)+cost(g, g_1), π(g) = g_1
14. **else**
15. d(g) = d(g_2)+cost(g, g_2), π(g) = g_2
16. Trace back from T using π to find the least cost monotonic path



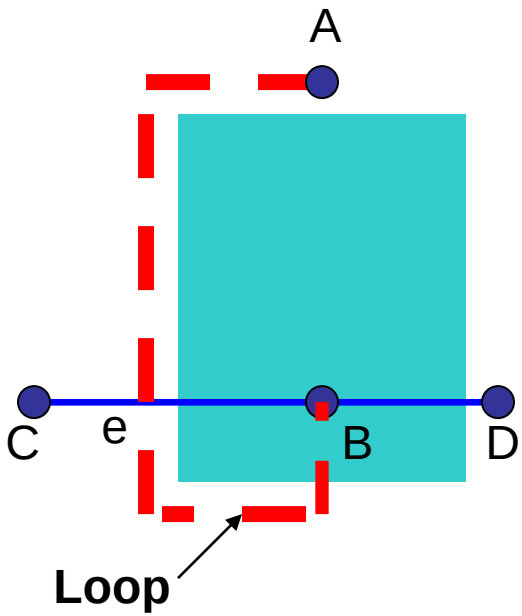
Problem with Edge-by-Edge Maze Routing



Unnecessary Detour



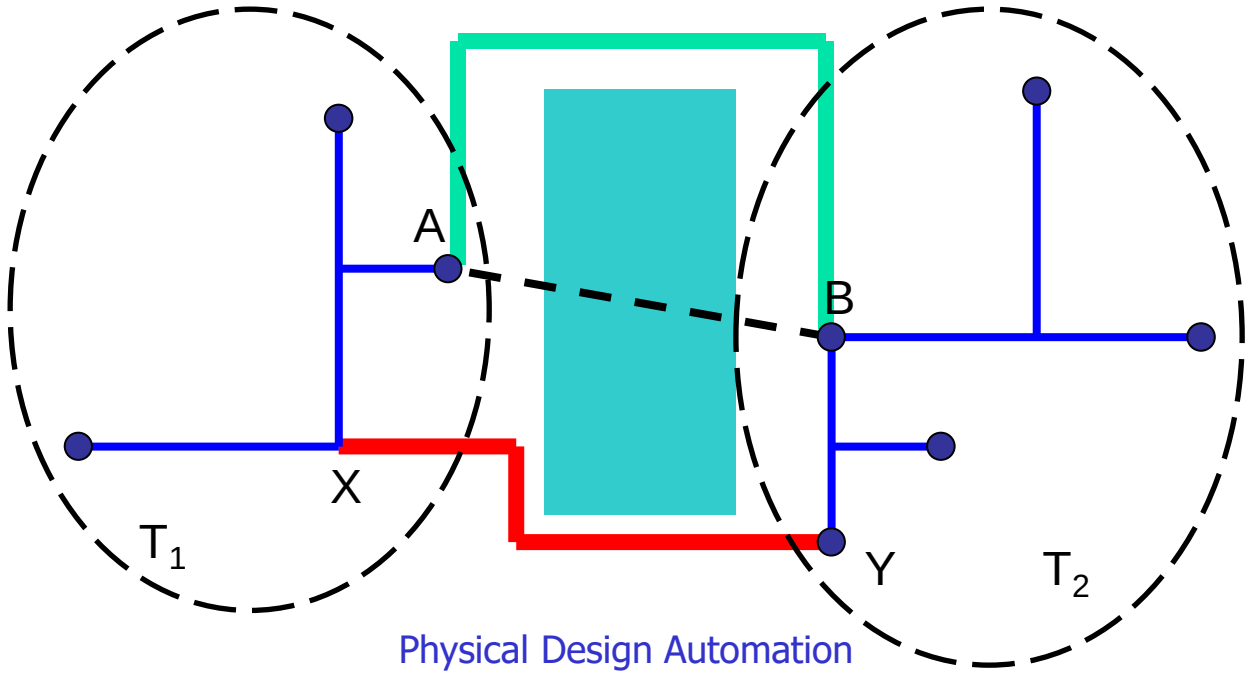
Redundant Routing



Loop

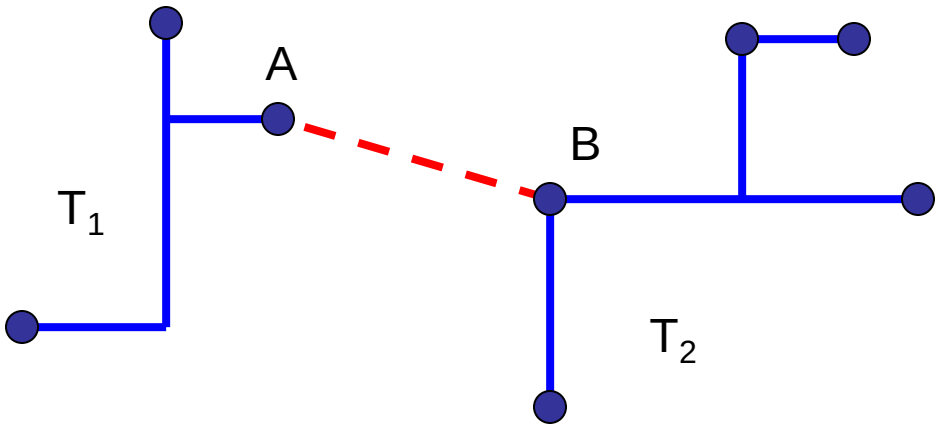
Multi-source Multi-sink Maze Routing

- Maze routing
  - Edge-by-Edge Single-source Single-sink maze routing
    - Route each tree edge one by one, from one endpoint to the other
  - Multi-source Multi-sink maze routing
    - Route between two subtrees



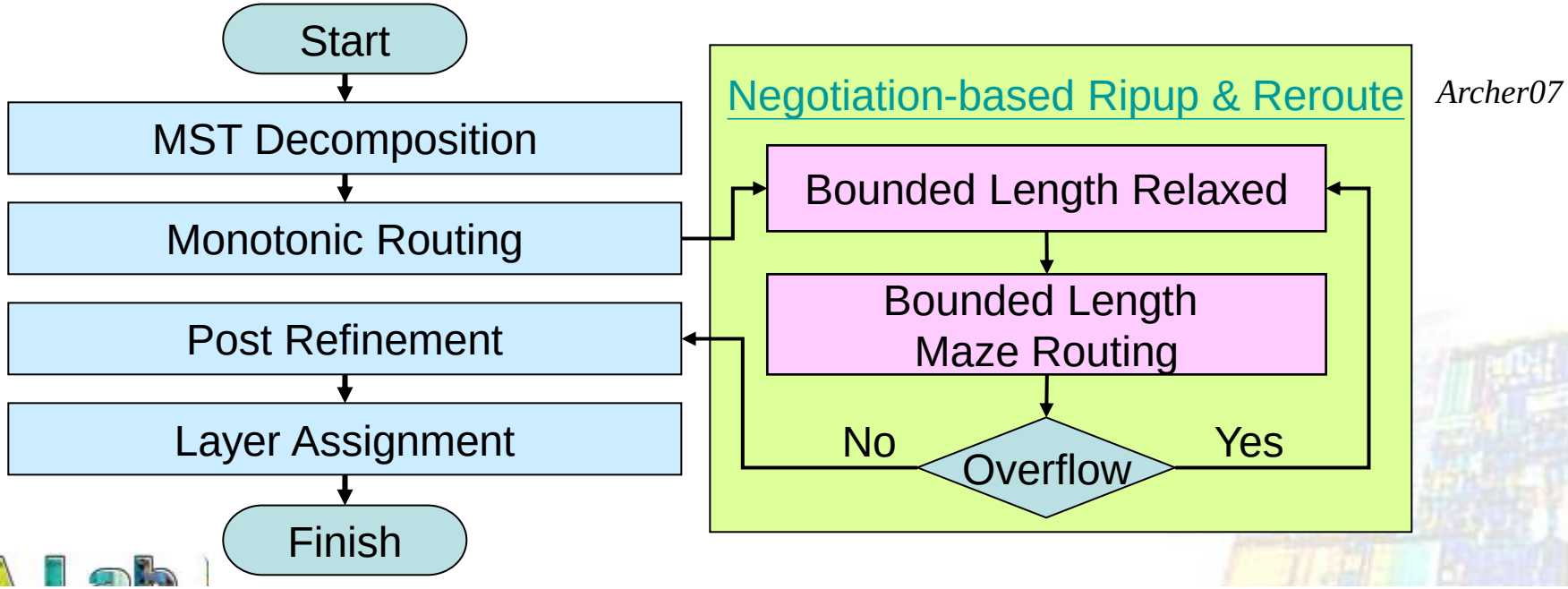
Multi-source Multi-sink Maze Routing Algorithm

- Break the tree edge (A, B) to be routed and get two subtrees T_1 (contains A) and T_2 (contains B)
- First put all points on T_1 into priority queue Q
- Loop similar to Dijkstra's Algorithm to update shortest path from T_1 to the grid points
- Stop when any of the points on T_2 is extracted from Q
- Runtime complexity O(*V*lg*V*) (*V* is the # grids)



NCTU GR 2.0

- Main feature
  - ✓ Bounded length maze routing, a simple but very effective technique
  - ✓ The first multi-threaded GR on the multi-core platform
- Bounding box vs. bounded length

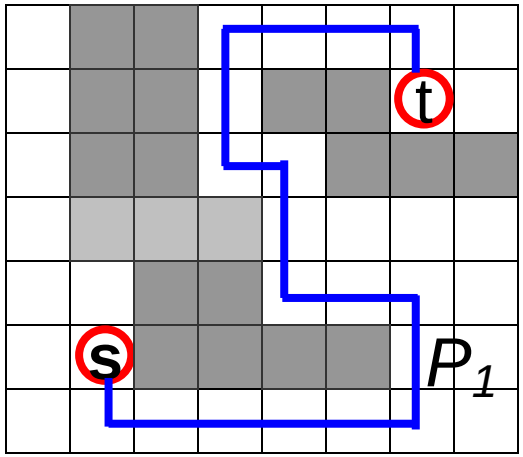


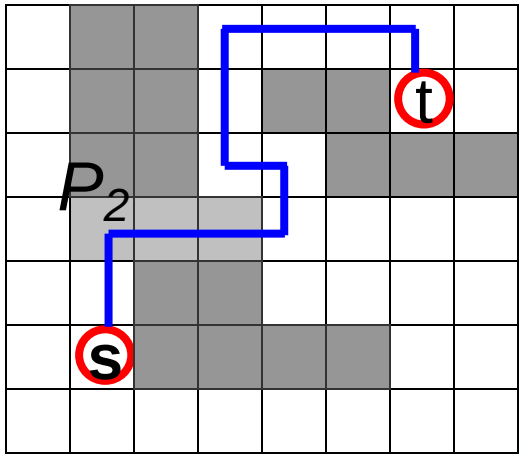
Wen-Hao Liu, Wei-Chun Kao, Yih-Lang Li, and Kai-Yuan Chao, "NCTU-GR 2.0: Multi-Threaded Collision-Aware Global Routing with Bounded-Length Maze Routing", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 32, No. 5, pp. 709 – 722, May 2013

Bounded Length Maze Routing (BLMR)

- Given 4-tuple(*s*, *t*, *G*, *BL*)
- The objective of the BLMR problem is to identify a minimal-cost path from *s* to *t* on graph *G*, and the wirelengh of the path cannot exceed *BL*.

***BL*=16**

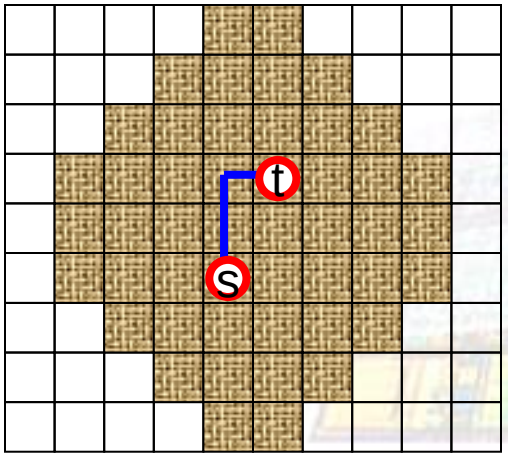




Bounded Length Maze Routing (BLMR)

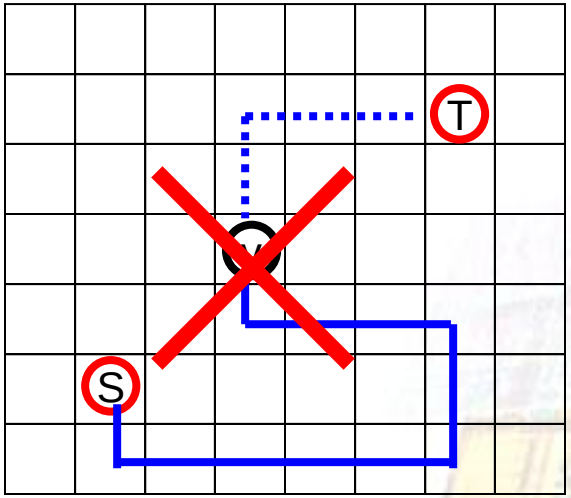
- BLMR not only restricts the searching region to speed up maze routing, but also effectively utilizes routing resources by avoiding producing lengthy wire.

- The proposed BLMR algorithms
  - ✓ Optimal-BLMR algorithm obtains a minimum-cost routing solution under the bounded-length constraint.
  - ✓ Heuristic-BLMR is faster than optimal-BLMR, but can not guarantee to get optimal solution.



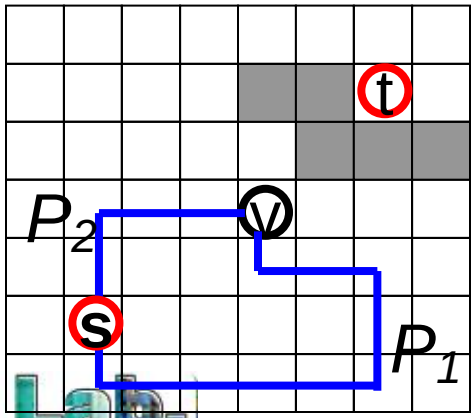
Optimal-BLMR

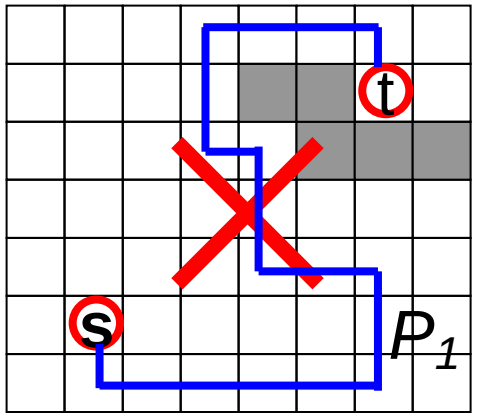
- Optimal-BLMR adopts two different policies from traditional maze routing.
  - ✓ Prunes violated path
  - ✓ Reserves path candidates
- Prune violated path: Path $P_i(s, v)$ is discarded if the sum of $wl(P_i)$ and $Manh(v, t)$ is greater than BL.
- For example
  - ✓ $wl(P_i)$ =12, $Manh(v, t)$=5, BL=14

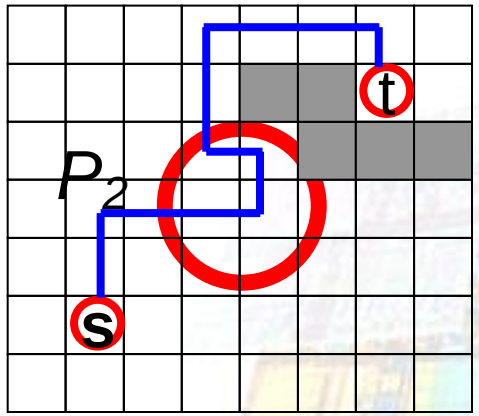


Optimal-BLMR

- Reserves path candidates: assuming that there are two or more paths from *s* to *v*
  - ✓ Traditional maze routing only reserves the minimal-cost path.
  - ✓ Optimal-BLMR reserves more than one path because the length slack of the minimal-cost path may be less than the wirelength required to detour around congested regions.
- Ex: PathCost(P_1)<PathCost(P_2), BL=16







Heuristic-BLMR

- Heuristic-BLMR is much faster than optimal-BLMR.
  - ✓ heuristic-BLMR reserves only one path from the source to the current node, and the other paths are discarded.
- The major problem of heuristic-BLMR is determining which path candidate should be reserved.
  - ✓ Heuristic-BLMR reserves the minimal-cost path candidate with enough length slack to bypass the congested regions.
  - ✓ If no path candidates have enough length slack, the shortest path candidate is reserved because the shortest path has a greater chance to bypass the congested regions.

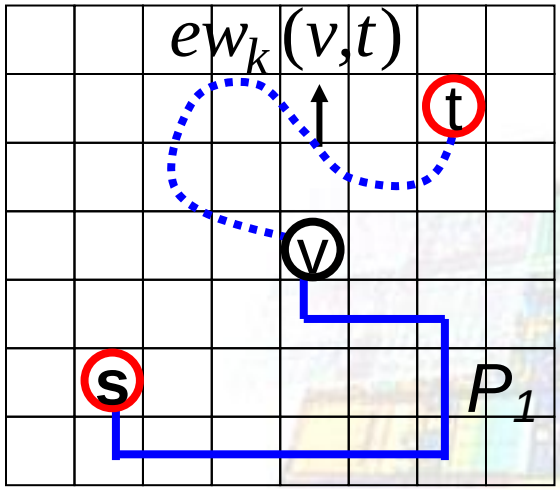
Heuristic-BLMR

- The *history-based estimated wirelength* of path from *v* to *t* is estimated as follows,

$$ew_k(v,t) = Manh(v, t) \times \frac{L_{k-1}(s, t)}{Manh(s, t)}$$

- Heuristic-BLMR predicts that $P_i(s,v)$ has sufficient length slack to bypass the congested regions from *v* to *t* if the following equation holds,

$$wl(P_i) + ew_k(v, t) \le BL$$



Bounded-Length Relaxation

- Bounded-length relaxation encourages heuristic-BLMR to obtain routing results with less overflow at the cost of using longer wirelength.

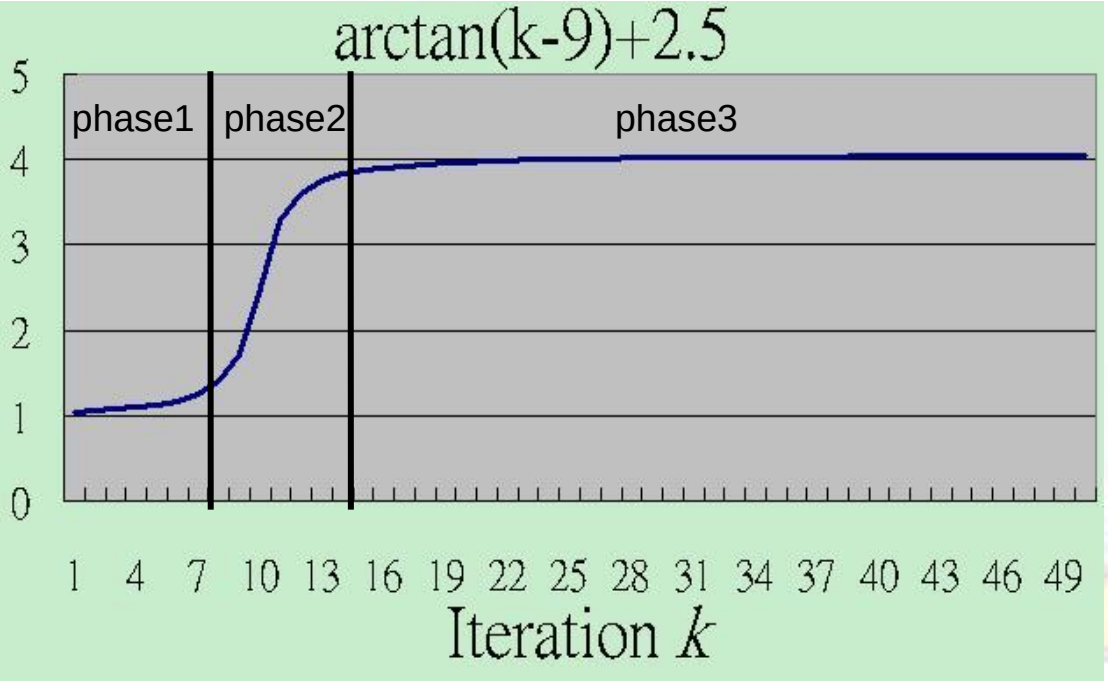
- The bounded-length of net *n* at iteration *k* is formulated as follows:

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

- The variables α and β are user-defined constants, and α is set to 9 and β is set to 2.5 in this study

Bounded-Length Relaxation

- Phase 1 (*1* to *8)*: solves most overflows, only increases a little total wirelength.
- Phase 2 (9 to 15*)*: most benchmarks are finished in this phase.
- Phase 3 (iteration>*16*): avoid producing an overly long wirelength and enlarging the search region too much.



Experimental Environment

- Implemented in C/C++ language
- Linux machine with a quad-core 3.0 GHz Intel Xeon-based CPUs and 32GB memory.
- Two sets of benchmarks : ISPD'07 and ISPD'08 global routing contest benchmarks
- All the other global routers for comparison are also performed on the same platform.

Comparison on Overflow-Free Cases

- Comparison between the proposed sequential router and the other routers on overflow-free cases.

TABLE COMPARISON BETWEEN THE PROPOSED SEQUENTIAL ROUTER AND THE OTHER ROUTERS ON OVERFLOW-FREE CASES.

Benchmark	SGR		NTHU-Route 2.0 [9]		FastRoute 4.1 [12]		NTUgr [10]		NCTUgr [13]	
	WL	CPU(m)	WL	CPU(m)	WL	CPU(m)	WL	CPU(m)	WL	CPU(m)
adaptec1	53.04	2.52	53.49	4.86	53.73	3.31	56.05	4.41	53.50	3.90
adaptec2	51.51	0.65	52.31	1.42	52.17	0.95	53.34	1.32	51.69	1.45
adaptec3	129.24	3.40	131.11	6.16	130.82	3.69	133.58	5.49	130.35	4.88
adaptec4	120.53	1.23	121.73	2.08	121.24	1.25	123.26	2.82	120.67	2.28
adaptec5	154.01	6.03	155.55	11.95	155.81	6.70	159.29	13.61	154.70	9.07
newblue1	45.76	2.37	46.53	4.07	46.33	12.01	X	X	45.99	3.63
newblue2	74.51	0.64	75.85	1.17	75.12	0.85	77.41	0.62	74.88	0.90
newblue5	228.68	5.26	231.73	10.88	230.94	9.82	238.98	25.77	230.31	15.03
newblue6	175.49	4.44	177.01	10.34	177.87	8.78	185.38	14.19	176.87	9.67
bigblue1	56.29	4.09	56.35	6.93	56.64	4.22	57.81	10.83	56.56	6.35
bigblue2	88.66	4.21	90.59	6.47	91.18	12.12	92.14	758.82	89.40	11.18
bigblue3	129.35	1.91	130.76	3.91	130.04	2.06	134.25	6.05	129.66	4.38
Ratio	1	1	1.012	1.900	1.012	1.768	1.039	18.659	1.006	1.920

Comparison on Hard-to-Route Cases

- Comparison between the proposed sequential router and the other routers on hard-to-route cases.

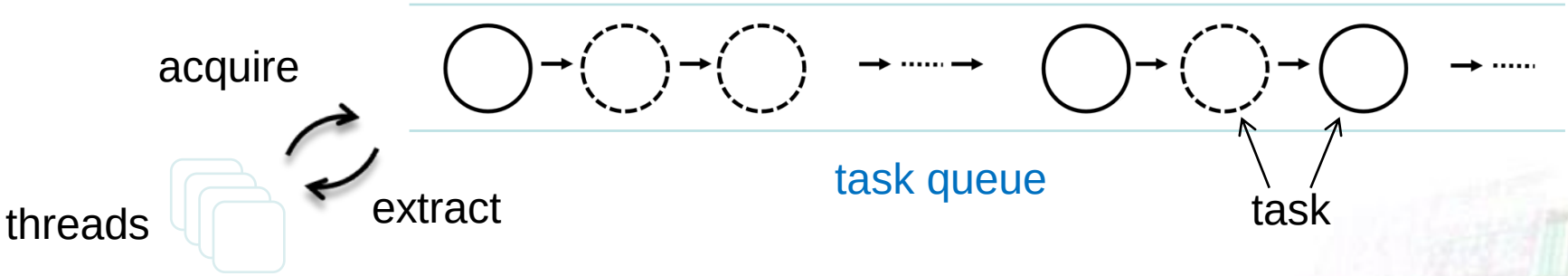
- SGR achieves much better performance for maximum overflow, wirelength and runtime.

TABLE COMPARISON BETWEEN THE PROPOSED SEQUENTIAL ROUTER AND THE OTHER ROUTERS ON HARD-TO-ROUTE BENCHMARKS.

	SGR			NTHU-Route 2.0 [9]			FastRoute 4.1 [12]			NTUgr [10]			NCTUgr [13]		
	MO/TO	WL	CPU(m)	MO/TO	WL	CPU(m)	TO	WL	CPU(m)	MO/TO	WL	CPU(m)	MO/TO	WL	CPU(m)
newblue3	194/ 31710	105.36	143.34	204 /31454	106.49	64.97	736 /31276	108.40	15.99	> 1 day	> 1 day	> 1 day	198/ 31808	104.28	131.43
newblue4	2 / 144	127.27	17.33	4/138	130.46	52.01	2/136	130.46	65.23	2/146	136.42	1405.38	2 / 134	126.79	40.92
newblue7	2 / 58	342.90	85.67	2/62	353.35	50.28	4/54	353.38	868.74	4/308	365.35	1402.52	2 / 114	338.63	71.52
bigblue4	2 / 194	225.00	60.23	2/162	231.04	52.63	2/130	230.24	93.25	6/174	240.48	1429.85	2 / 164	223.99	65.37
ratio		1	1		1.023	1.229		1.027	3.891		1.069	40.405		0.992	1.300

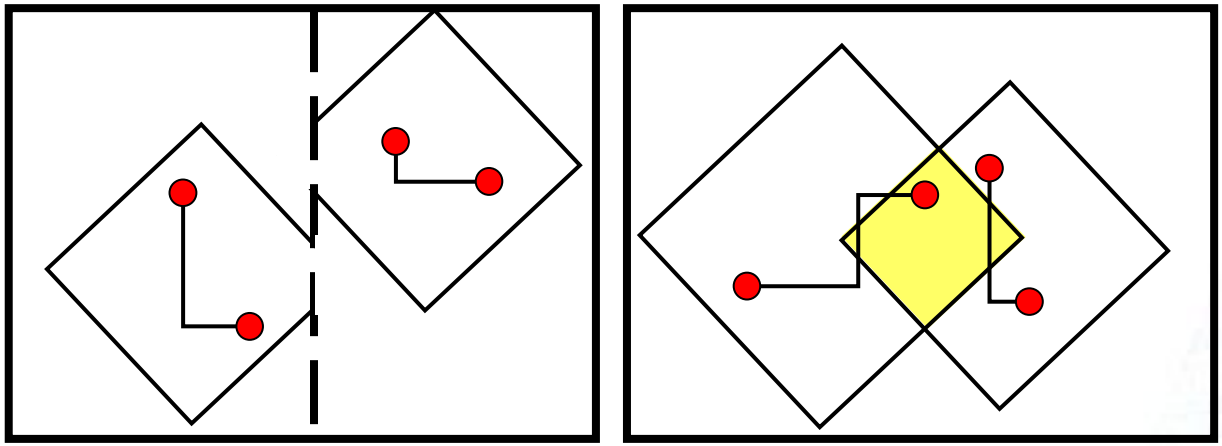
Task-based Concurrency Strategy (TSC)

- Concept:
  - Define a two pin net routing as a **task.**
  - Maintain a **task queue** to contain all tasks.
  - Each thread repeatedly acquires a task from task queue when the thread completes its current task.



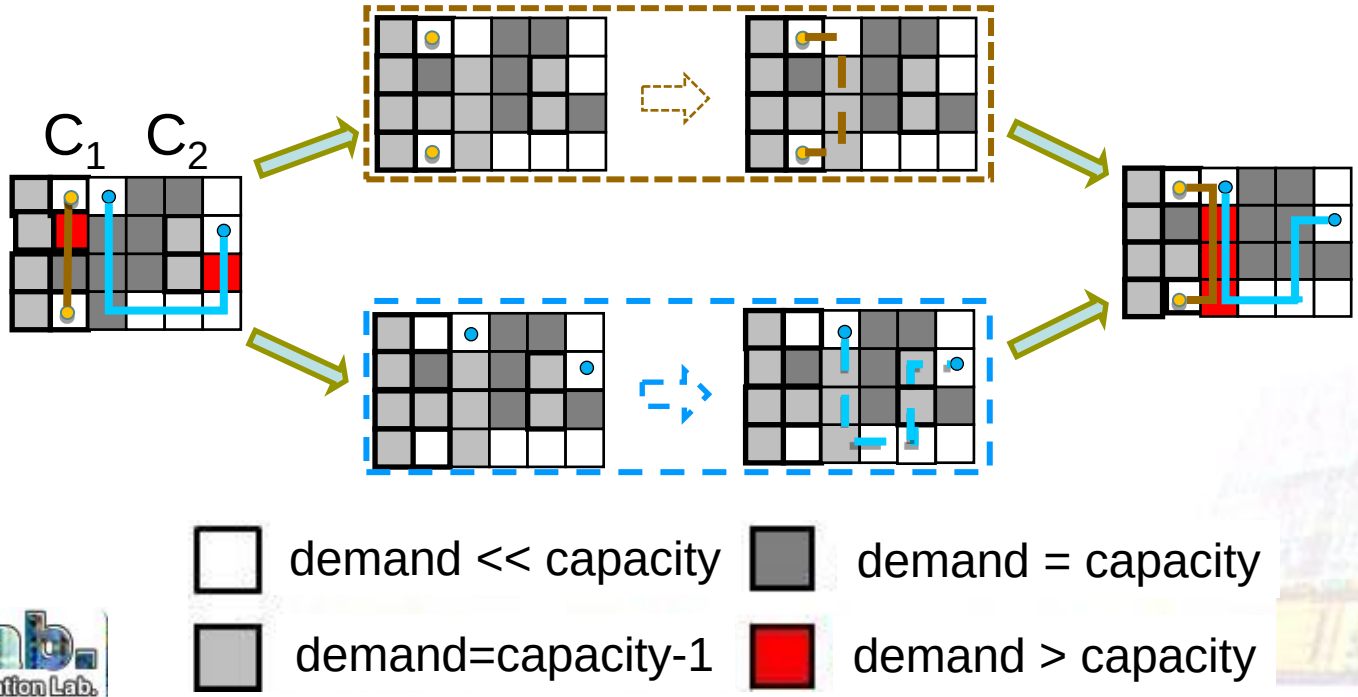
Comparison between TSC and PSC

- The problems of partitioning-based concurrency strategy (PSC)
  - ✓ Load unbalance
  - ✓ The nets which cross more than one subregion
  - ✓ Boundary restriction
- TSC can avoid those problems, but the race condition of routing resources becomes a critical challenge.



Collision

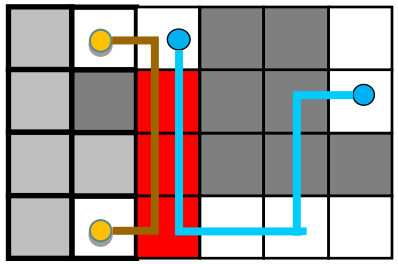
- A collision is said to occur if two or more threads demand the same routing resource simultaneously, which may result in overflow.



Collision Prevention Approach

- Collision often occurs when several nets are close to each other in a congested region.

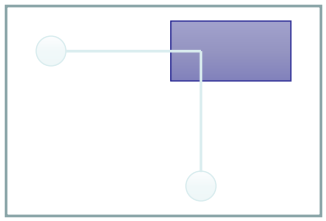
Ex:

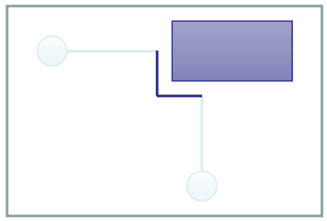


- Major challenges of collision prevention
  - Threads must distinguish whether a grid edge would be used by other connections' routing paths.
  - The communication between each thread is time consuming : losing efficiency.

Collision Prevention Approach

- Analysis shows that each new routing path reuses about 80% grid edges of the original routing path.

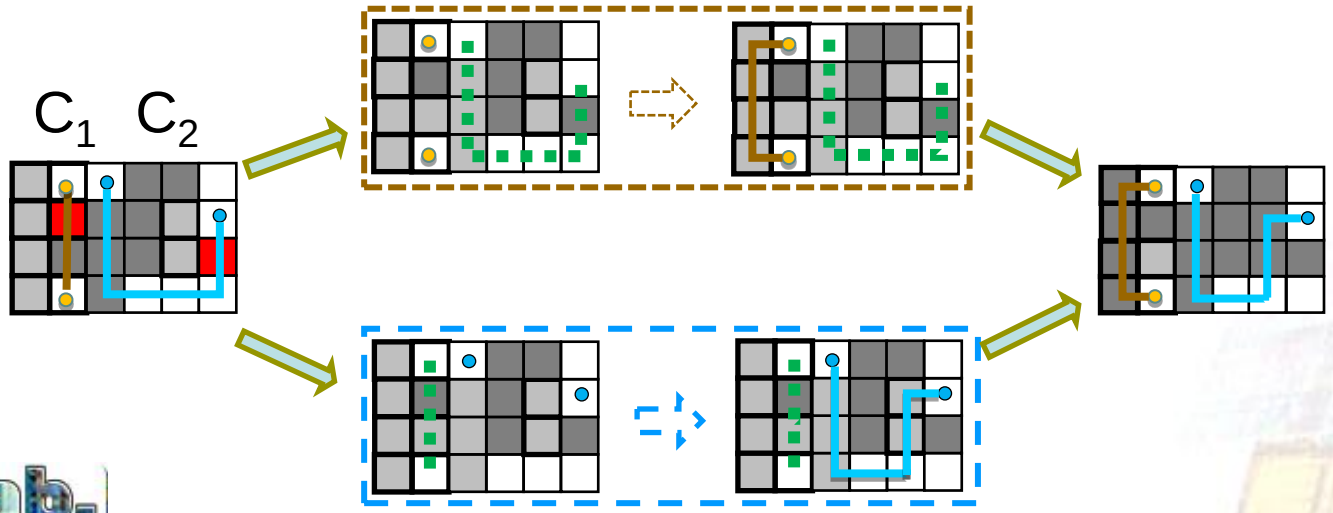




- The proposed collision-prevention approach utilizes this property to estimate the demand of each grid edge by other currently routed nets.
- As a result, each thread can estimate the risk of using each grid edge.

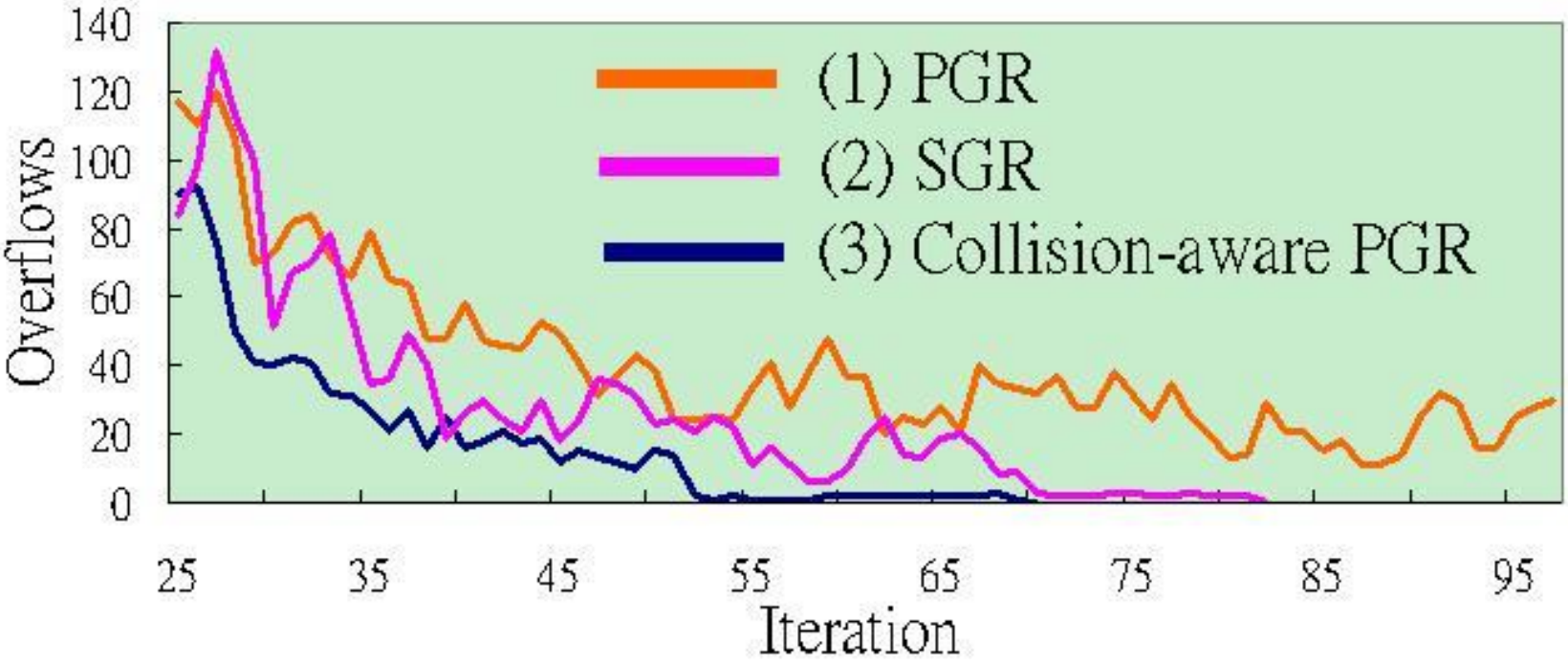
Collision Prevention Approach

- Each thread marks the grid edges of the original routing path on the congestion map.
- Then each thread can distinguish whether the grid edge was used in previous iteration.



The Effect of Collision Prevention Approach

- Relationship between the routing iteration number and overflows in newblue1 for PGR, SGR and Collision-aware PGR.



Comparison on Overflow-Free Cases

- Comparison between the proposed sequential router and parallel routers on overflow-free cases.
- Collision-aware PGR is performed 10 times to display the worst, average, and best results.

TABLE COMPARISON BETWEEN THE PROPOSED SEQUENTIAL ROUTER AND PARALLEL ROUTERS ON OVERFLOW-FREE CASES.

	SGR	Collision-aware PGR (quad cores)											
	R&R time (m)	Wirelength			R&R time (m)			WL inc %			R&R time improv. rate		
		worst	average	best	worst	average	best	worst	average	best	worst	average	best
adaptec1	1.78	53.28	53.27	53.25	0.53	0.51	0.50	0.45	0.43	0.40	3.36	3.48	3.54
adaptec2	0.20	51.54	51.53	51.53	0.10	0.09	0.08	0.06	0.05	0.04	2.06	2.32	2.57
adaptec3	1.77	129.31	129.30	129.29	0.56	0.54	0.53	0.06	0.05	0.04	3.19	3.28	3.32
adaptec4	0.10	120.55	120.54	120.54	0.05	0.05	0.05	0.01	0.01	0.01	1.92	2.06	2.21
adaptec5	4.06	154.50	154.48	154.45	1.20	1.15	1.13	0.32	0.31	0.29	3.39	3.54	3.60
newblue1	2.00	45.74	45.74	45.73	0.86	0.74	0.66	-0.03	-0.04	-0.05	2.32	2.69	3.03
newblue2	0.04	74.52	74.51	74.51	0.03	0.02	0.02	0.01	0.01	0.00	1.42	1.55	1.72
newblue5	2.92	228.85	228.82	228.80	1.05	1.01	0.95	0.07	0.06	0.06	2.78	2.88	3.06
newblue6	2.52	175.80	175.73	175.60	0.90	0.87	0.84	0.18	0.14	0.06	2.80	2.90	2.99
bigblue1	3.25	56.76	56.73	56.72	1.11	1.05	0.99	0.83	0.78	0.75	2.93	3.10	3.29
bigblue2	3.54	88.82	88.79	88.77	1.76	1.52	1.20	0.18	0.14	0.12	2.02	2.34	2.95
bigblue3	0.61	129.44	129.43	129.42	0.27	0.26	0.24	0.07	0.06	0.05	2.28	2.39	2.56
Average								0.18	0.17	0.15	2.54	2.71	2.90

Comparison on Hard-to-Route Cases

- Comparison between the proposed sequential router and parallel router on hard-to-route cases.

- Collision-aware PGR runs 2.96 times, 3.12 times, and 3.3 times faster than SGR in the worst, average, and best cases, respectively.

TABLE COMPARISON BETWEEN THE PROPOSED SEQUENTIAL ROUTER AND PARALLEL ROUTERS ON HARD-TO-ROUTE BENCHMARKS.

	SGR	Collision-aware PGR (quad cores)														
	R&R	OF			Wirelength			R&R time (m)			WL inc %			R&R time improv. rate		
	time (m)	worst	average	best	worst	average	best	worst	average	best	worst	average	best	worst	average	best
newblue3	142.70	31810	31776	31710	105.38	105.33	105.30	66.97	63.48	61.12	0.01	-0.03	-0.06	2.13	2.25	2.33
newblue4	15.76	150	144	138	127.33	127.29	127.24	6.30	6.12	5.78	0.04	0.01	-0.03	2.50	2.58	2.73
newblue7	84.19	74	66	60	342.56	342.51	342.45	27.14	26.32	25.14	-0.10	-0.11	-0.13	3.10	3.20	3.35
bigblue4	59.48	200	195	190	224.95	224.93	224.90	14.45	13.30	12.42	-0.02	-0.03	-0.04	4.12	4.47	4.79
Average											-0.02	-0.04	-0.07	2.96	3.12	3.30

NCTU GR 3.0 – Motivation to MDSV GR

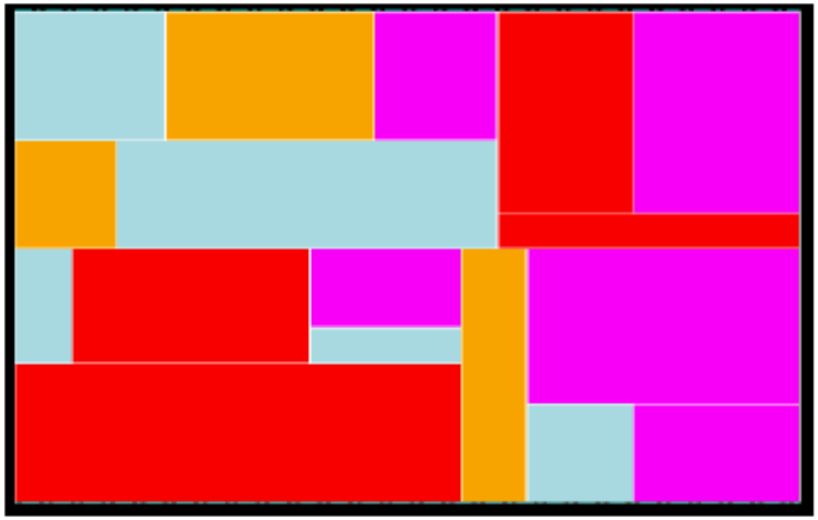
- High power consumption shortens the battery life of handheld devices and causes thermal and reliability problems.
- In modern circuit design, most of the total power consumption is dynamic power consumption, which is proportional to the square of supply voltage *Vdd*.
  - ✓ $P_d = kcV^2f$
- Multiple dynamic supply voltage (MDSV) effectively reduces dynamic power with a sophisticated control on different function units' voltages.

Wen-Hao Liu, Yih-Lang Li and Kai-Yuan Chao, "High-Quality Global Routing for Multiple Dynamic Supply Voltage Designs," Proceedings of International Conference on Computer-Aided Design (ICCAD 2011), San Jose, California, 2011.

NCTU GR 3.0 – Motivation to MDSV GR

- In region-based MDSV designs, circuits are partitioned into several power domains, each of which occupies a contiguous physical space and operates at a main supply voltage

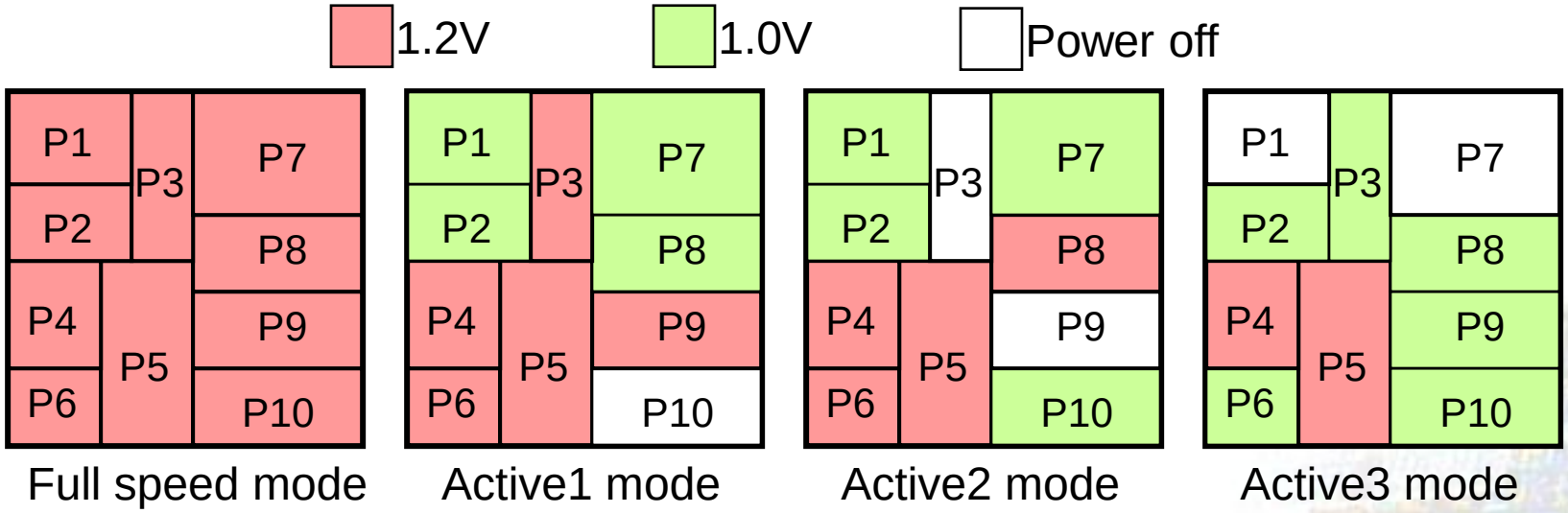
Block-based structure



Non-block-based structure

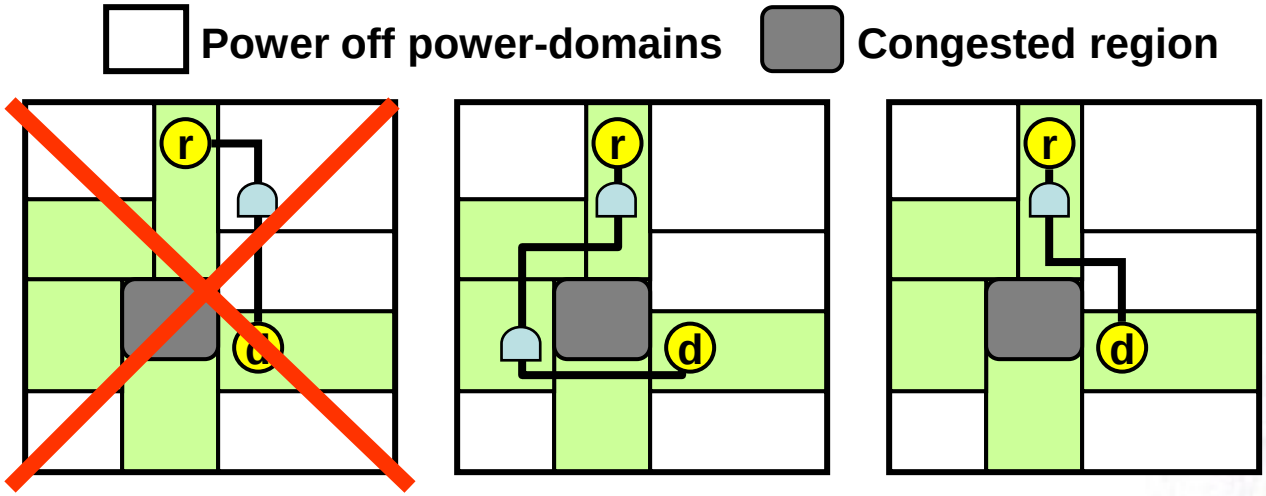
NCTU GR 3.0 – Motivation to MDSV GR

- In MDSV designs, the supply voltage of each power domain dynamically changes according to the power mode. To save power, power-domains are even shutdown in some power modes such as waiting mode and sleeping mode.



The Routing Limitation of MDSV Designs

- If $WL(P) > V_{thr}$, inserting a repeater is required.
- As the net is active, the repeaters on the net have to be placed in the non-shutdown power-domains.



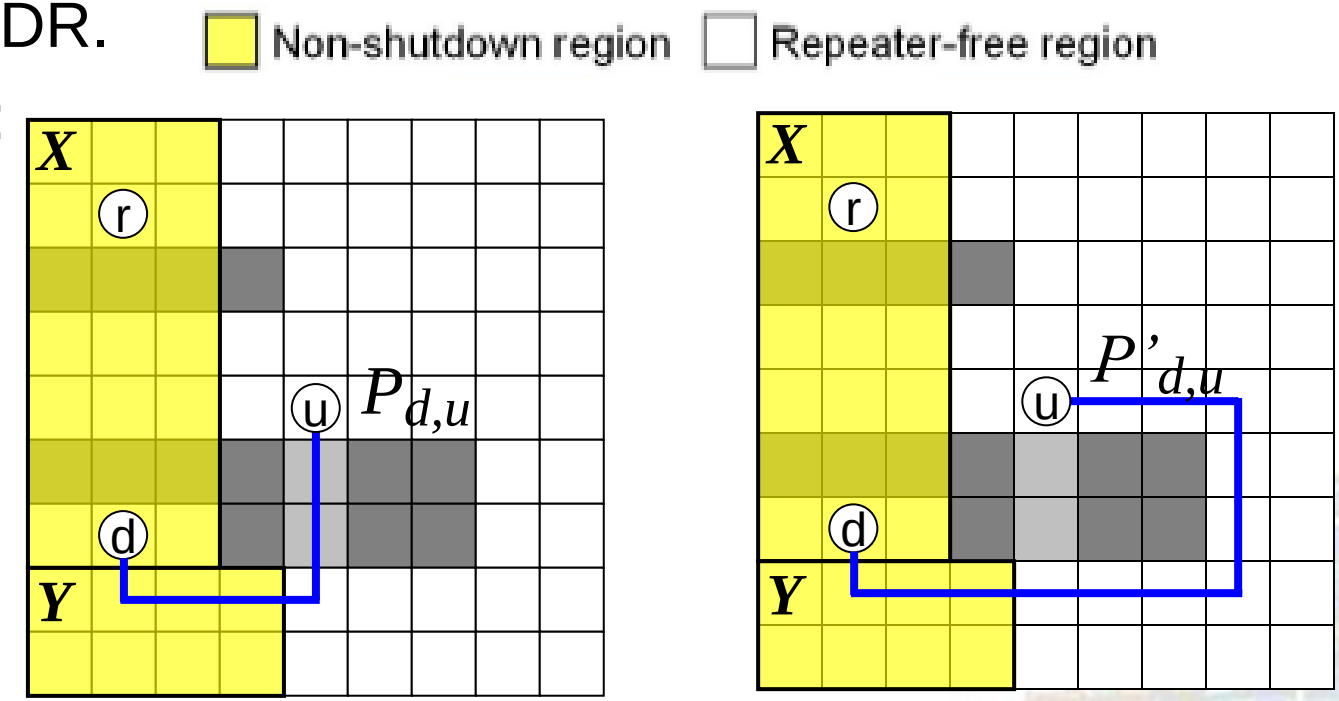
The Routing Limitation of MDSV designs

- Definition 1. ***Repeater-free region***
  - ✓ In any power mode of an MDSV design, if a net n_i is active when a power domain p is shut down, p is the ***repeater-free region*** associated with net n_i.

- Definition 2. ***Driving length constraint***
  - ✓ If the path of the driver of net n_i is in power domain X and n_i enters its repeater-free regions from a non-shutdown region Y, the continuous wire length in the repeater-free regions cannot exceed $\boldsymbol{L_x(Y)}$.

P2P Power Domain Aware Routing (PPPDR)

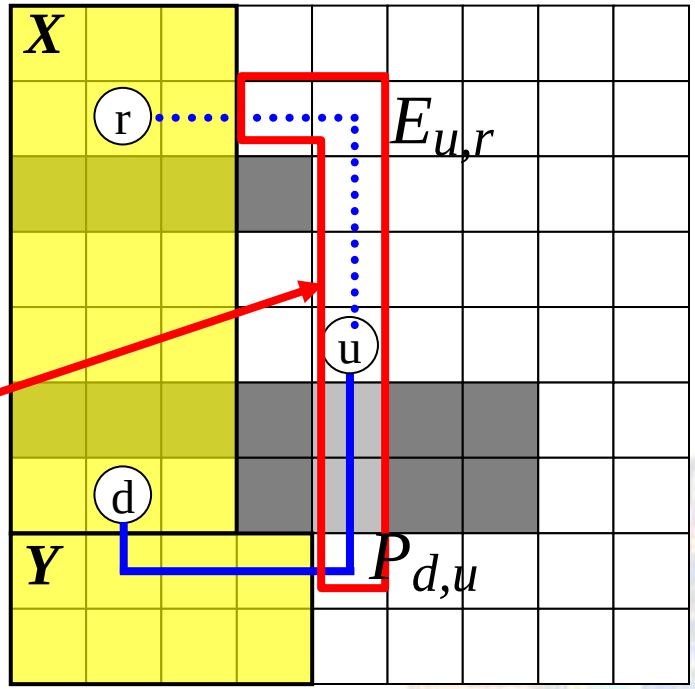
- PPPDR propagates from the driver to the receiver, and stores all current paths in a heap.
- How to reserve a better path in the heap is the main issue of PPPDR.
- For example:



Look-Ahead Path Selection

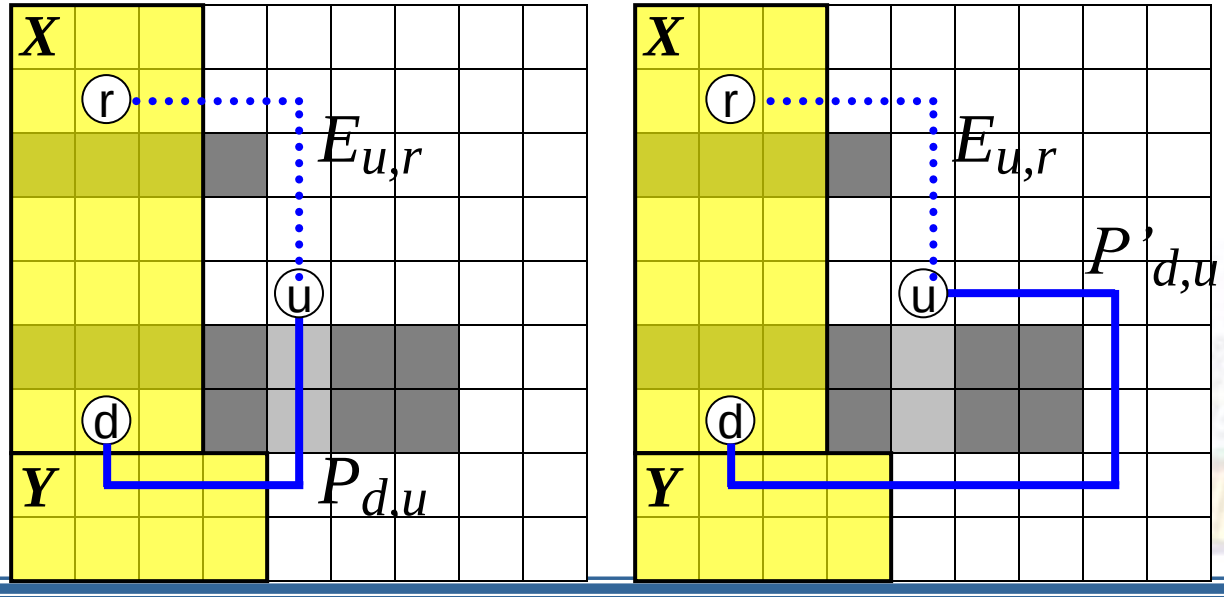
- This work proposes a look-ahead path selection method for reserving a legal routing path with minimal routing cost in the heap.
- A path, say $P_{d,u}$, is considered *feasible* if the following equation is true:

$$CW(P_{d,u}) + CW(E_{u,r}) \leq L_x(Y)$$



Look-Ahead Path Selection

- The proposed look-ahead path selection uses the following scheme to choose between $P_{d,u}$ and $P'_{d,u}$.
  - ✓ If only one path is feasible, the feasible path is reserved.
  - ✓ If both paths are feasible, the least-cost path is reserved.
  - ✓ If neither path is feasible, the path with the shorter continued wirelength (CW) in the repeater-free region is reserved

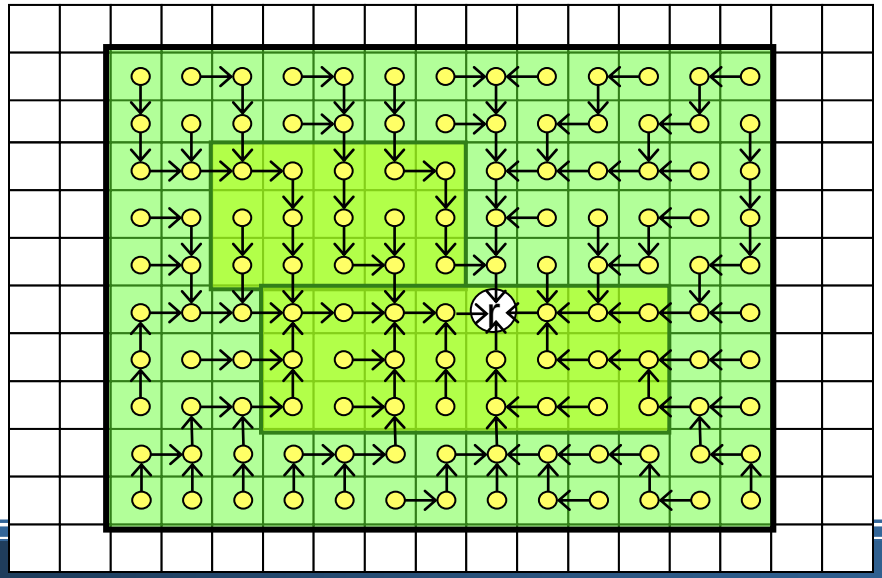


Accelerating Path Selection by Look-Up Table

- Look-ahead path selection invoking monotonic routing to identify $E_{u,r}$ requires the time complexity of $O(|V|)$.
- If look-ahead path selection can access a table to get CW of $E_{u,r}$, the time complexity of path selection can be reduced from $O(|V|)$ to $O(|1|)$.
- Therefore, before starting PPPDR, we attempt to build a lookup table to pre-store the CW of $E_{u,r}$, where u is any node that can be explored during PPPDR.

Accelerating Path Selection by Look-Up Table

- Obtaining the least-cost MPs (monotonic paths) from the potential explored nodes to r by following steps:
  - ✓ Identify a maximum search region (SR_{max}).
  - ✓ Perform four monotonic routings, from r to the corners of SR_{max}.
  - ✓ Each node in the SR_{max} has a predecessor, each node can reach r along the predecessors. Namely, the least-cost MP from r to each node in the SR_{max} is obtained.



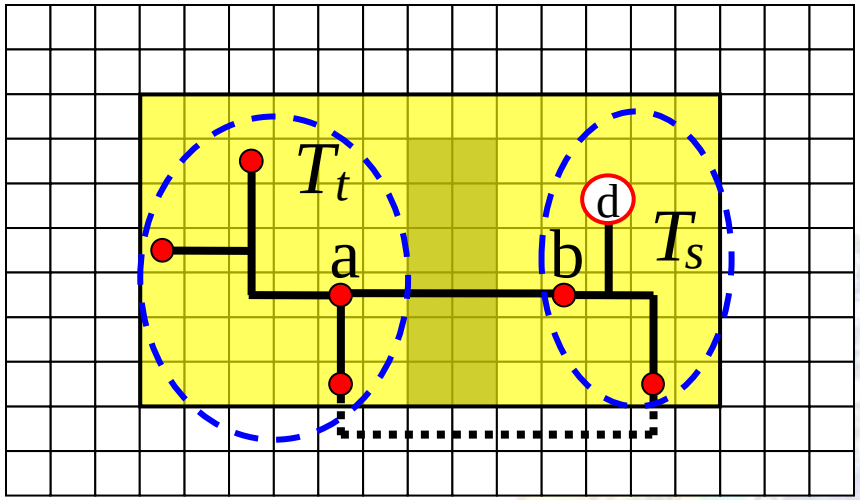
Multi-Source Multi-Target PDR (MMPDR)

- Given a multi-pin net, if a path $P_{a,b}$ passes through the overflowed G-cells, $P_{a,b}$ is ripped up and then two sub-trees T_s and T_t are obtained. (T_s contains the driver)

- MMPDR is a tree-to-tree routing algorithm which is used to route a new path from T_s to T_t.

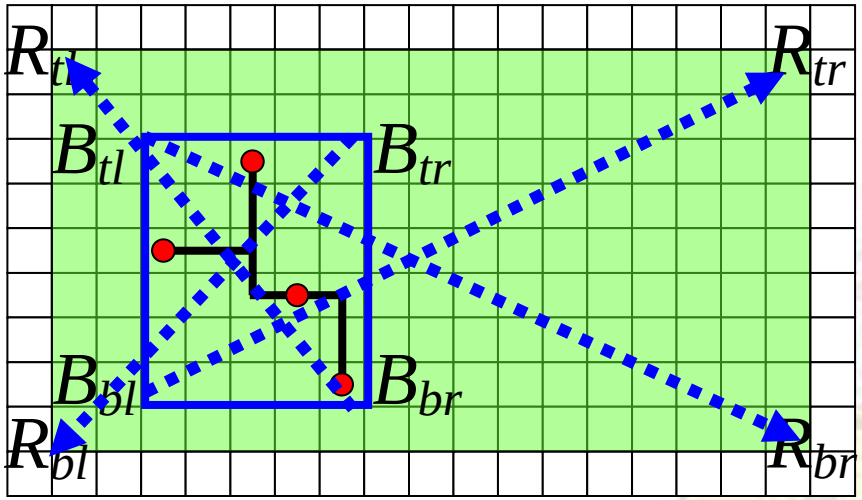
- During MMPDR, path $P_{d,u}$ is considered *feasible* if the following equation is true:

$$CW(P_{d,u}) + CW(E_{u,Tt}) \leq L_x(Y)$$



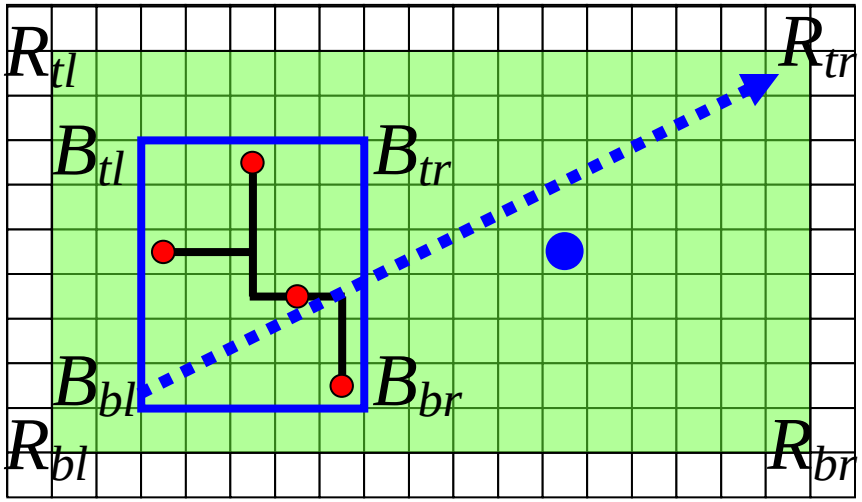
Multi-Source Multi-Target PDR (MMPDR)

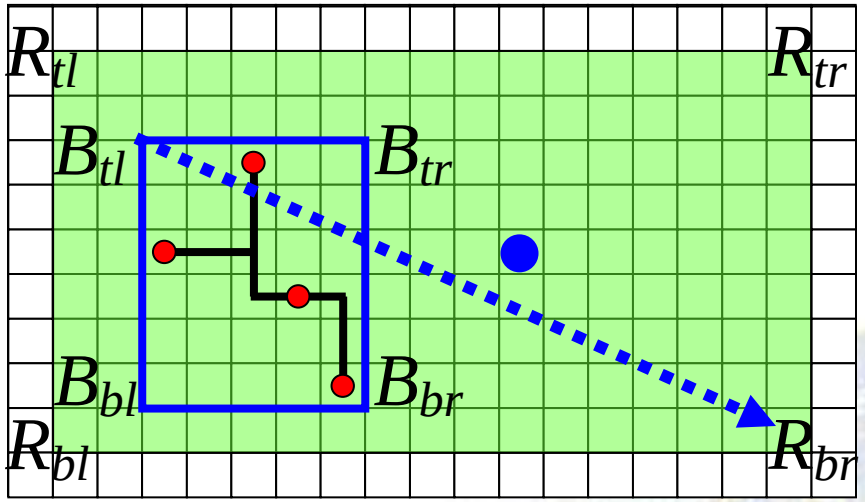
- Obtaining the least-cost MPs from the potential explored nodes to T_t by following steps:
  - ✓ Identify a maximum search region (SR_{max}) and a target box (B_{tar}) which is the minimum rectangle enclosing the target tree T_t.
  - ✓ Performed four tree-to-partial-nodes monotonic routings (TPNMR):
    - ➢ from B_{bl} to R_{tr},
    - ➢ from B_{br} to R_{tl},
    - ➢ from B_{tl} to R_{br}
    - ➢ from B_{tr} to R_{bl}.



Multi-Source Multi-Target PDR (MMPDR)

- Four TPNMRs identify for each node more than one MPs to T_t.
- The least-cost MP that connects a node to T_t is the one with the least cost from the results obtained using four TPNMRs.
- For example:

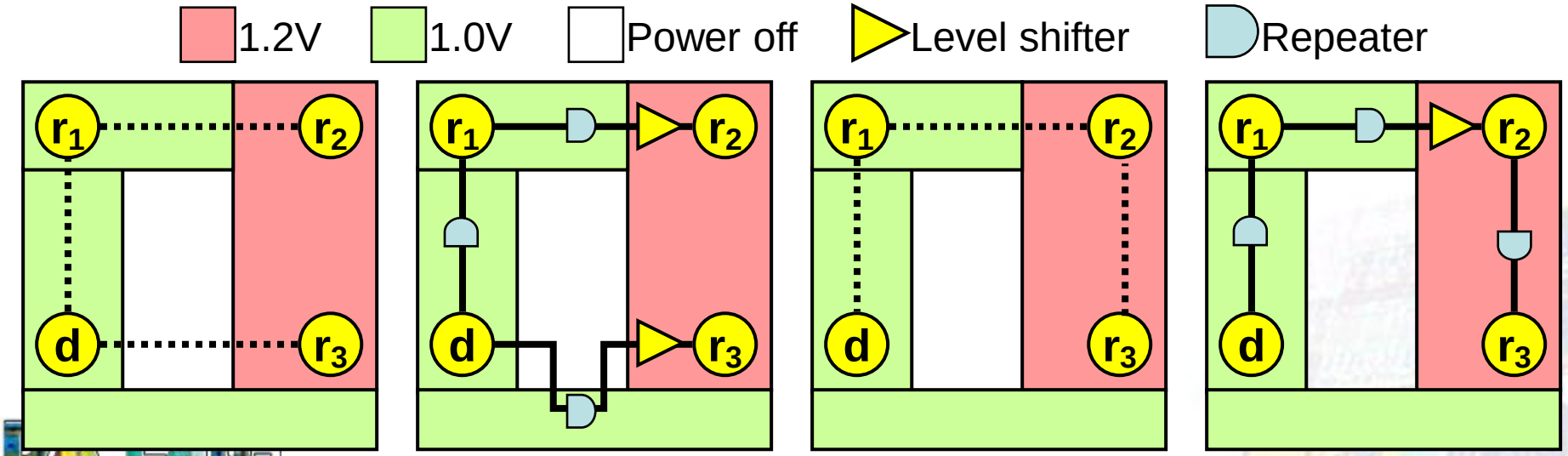




Power-Domain Aware Minimum Spanning Tree

- In MDSV designs, if a net connects a driver of lower voltage to a receiver of higher voltage, level shifters need to be inserted on the path at the boundaries of higher voltage power domain.

- The initial tree topology of each net affects the number of level shifters and the final routing wirelength.



Algorithm Flow

Algorithm MDSV-based global router
Input: grid graph G, power domains PDS, power modes M, forbidden regions F, nets N

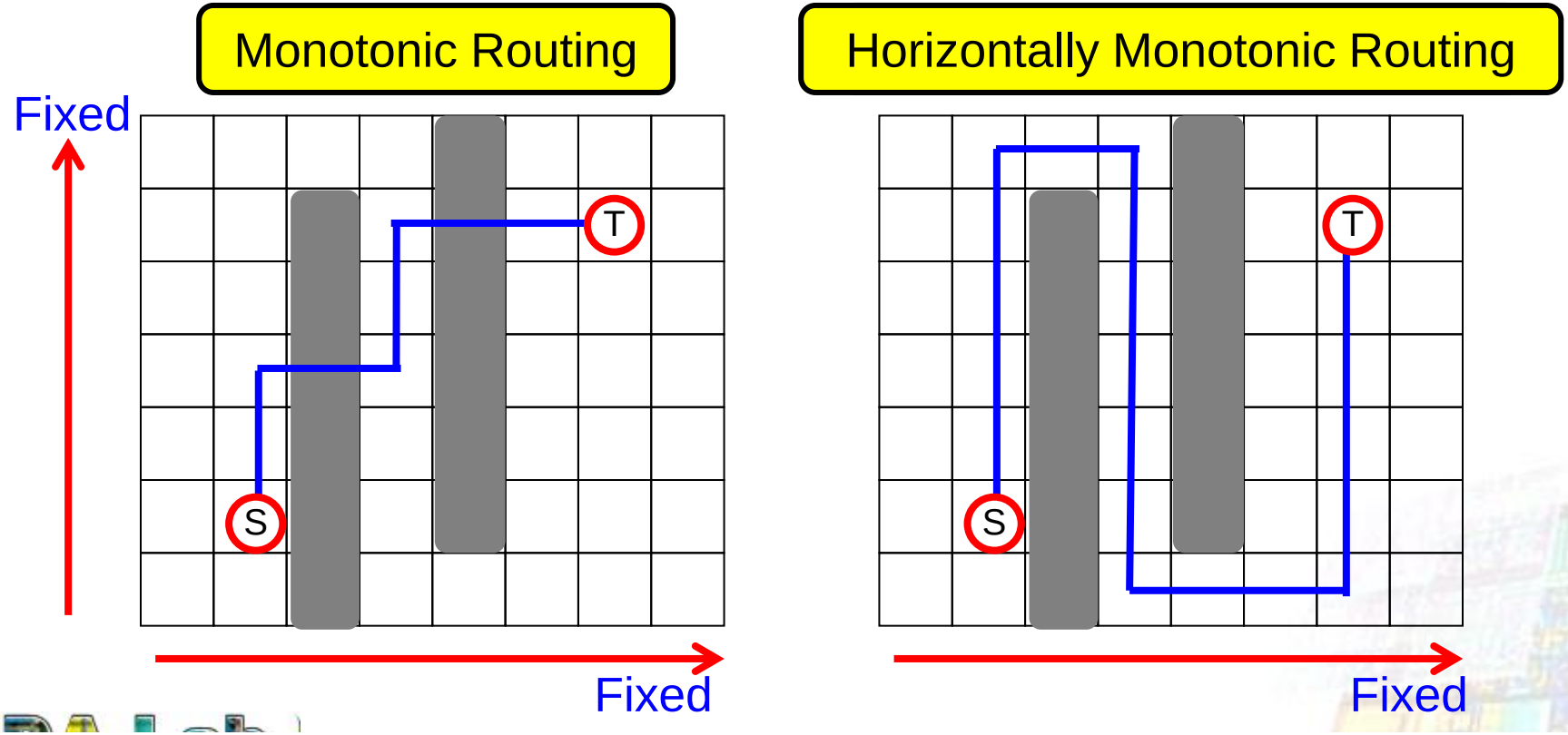
1. PDMST_Decomposition(N)
2. Monotonic_Routing(N, G)
3. **while (** G has overflows or driving length violation**) do**
4. **foreach** net n_i with overflows or driving length violation **do**
5. Rip_Up(n_i , G)
6. Maze_Routing(n_i , G, F)
7. **if** (n_i violates driving length constraint)
8. Rip_Up(n_i , G)
9. T←Building_Lookup_Table(n_i , PDS, M, N)
10. PPPDR(n_i , G, T, F) or MMPDR(n_i , G, T, F)
11. **end if**
12. **end foreach**
13. **end while**
14. **end**

obtaining congestion map

In the rip-up and rerouting stage, maze routing is first adopted to reroute a net. If the maze routing result of the net violates the driving length constraint, then the net will be ripped up and rerouted again using PDR.

Unilateral Monotonic Routing (NCTUGR 4.0)

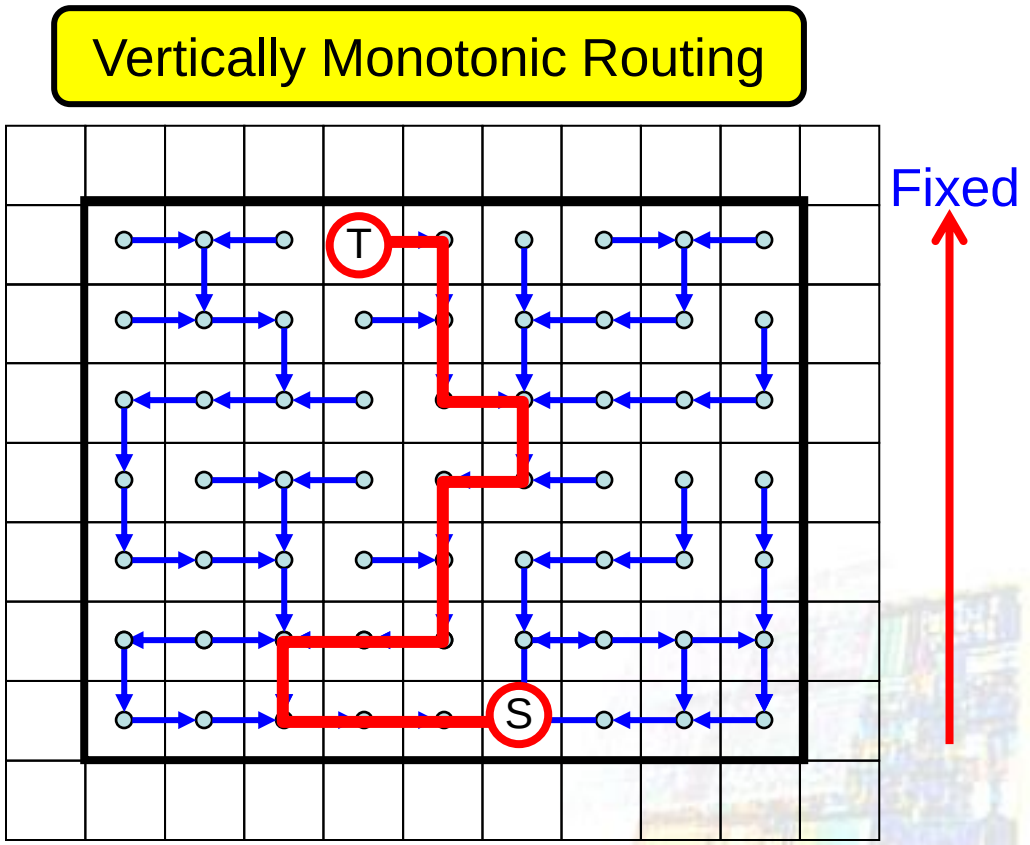
- **Unilateral monotonic routing** can seek a detoured path and running in the similar runtime as that of monotonic routing.



Wen-Hao Liu, Yih-Lang Li and Cheng-Kok Koh, "A Fast Maze-Free Routing Congestion Estimator with Hybrid Unilateral Monotonic Routing", Proceedings of International Conference on Computer-Aided Design (ICCAD 2012), San Jose, California, 2012.

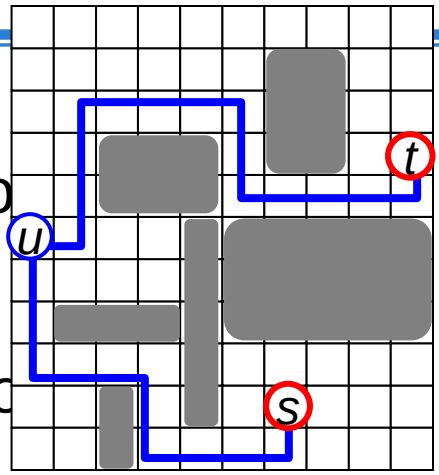
Unilateral Monotonic Routing

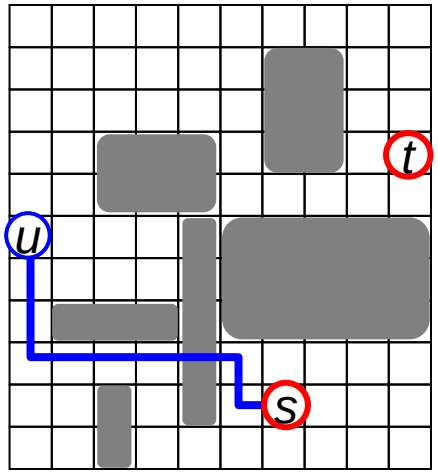
- Unilateral monotonic routing is based on the dynamic programming technique.
- Given a bounding box *B*, unilateral monotonic routing identifies a parent for each node within *B* in a bottom-up manner.
- The time complexity of unilateral monotonic routing is $O(|B|)$.



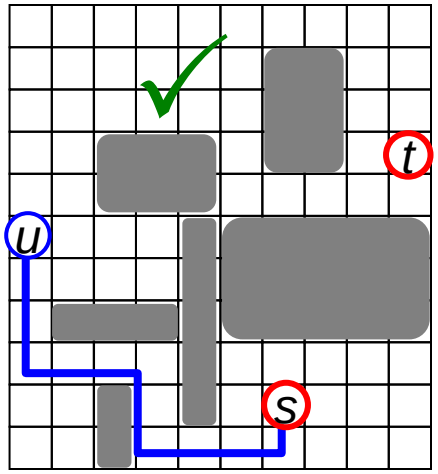
HUM Routing

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

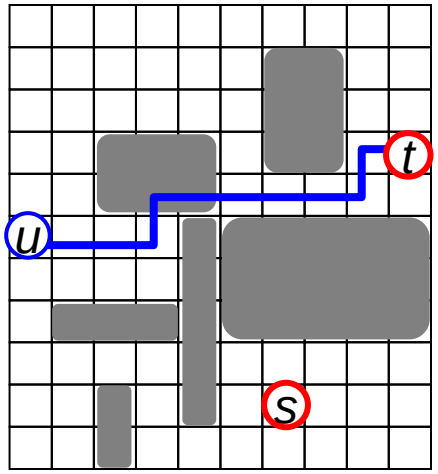




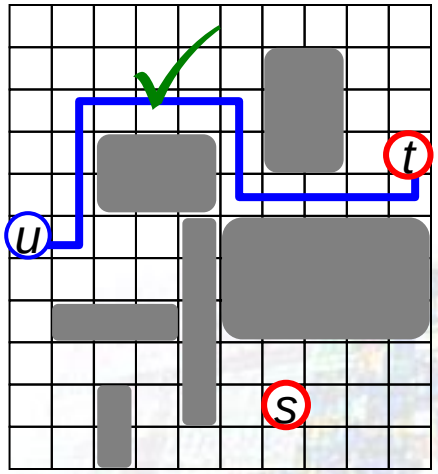
Vertically monotonic path from s to u



Horizontally monotonic path from s to u



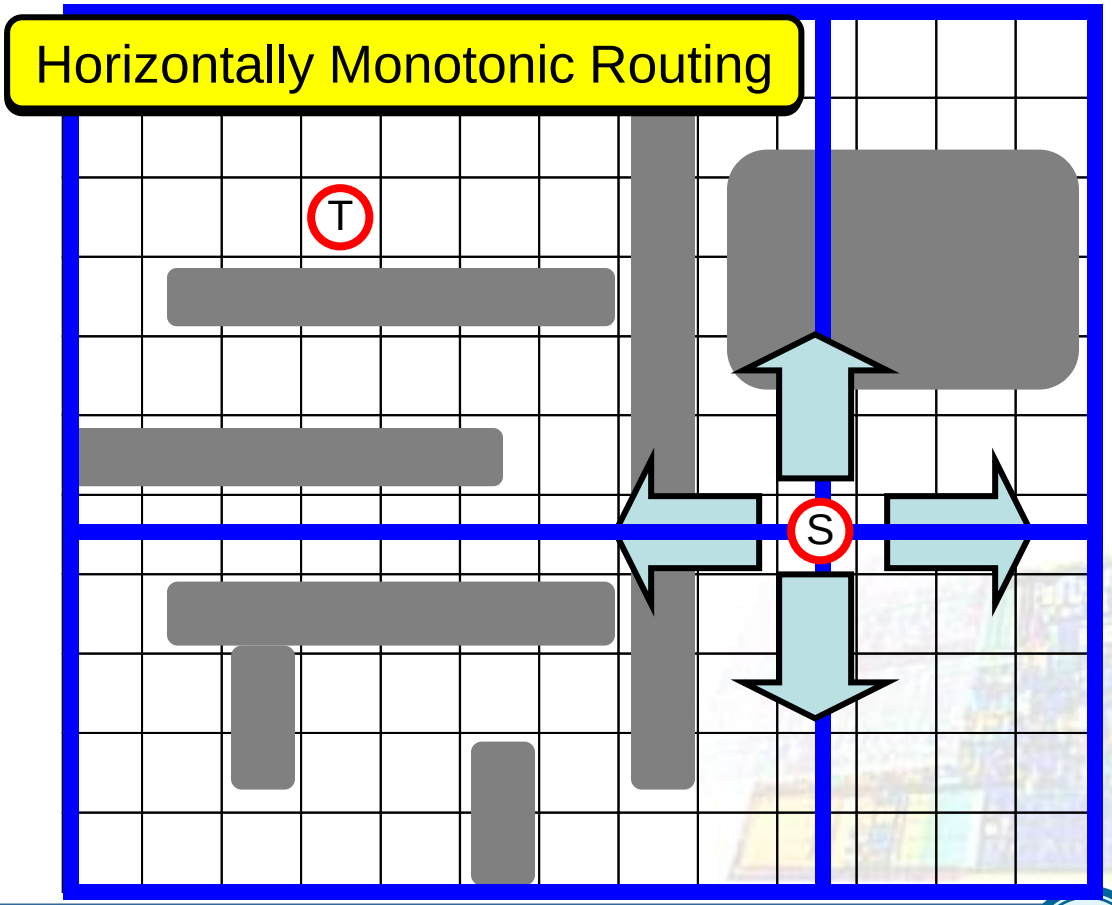
Vertically monotonic path from t to u



Horizontally monotonic path from t to u

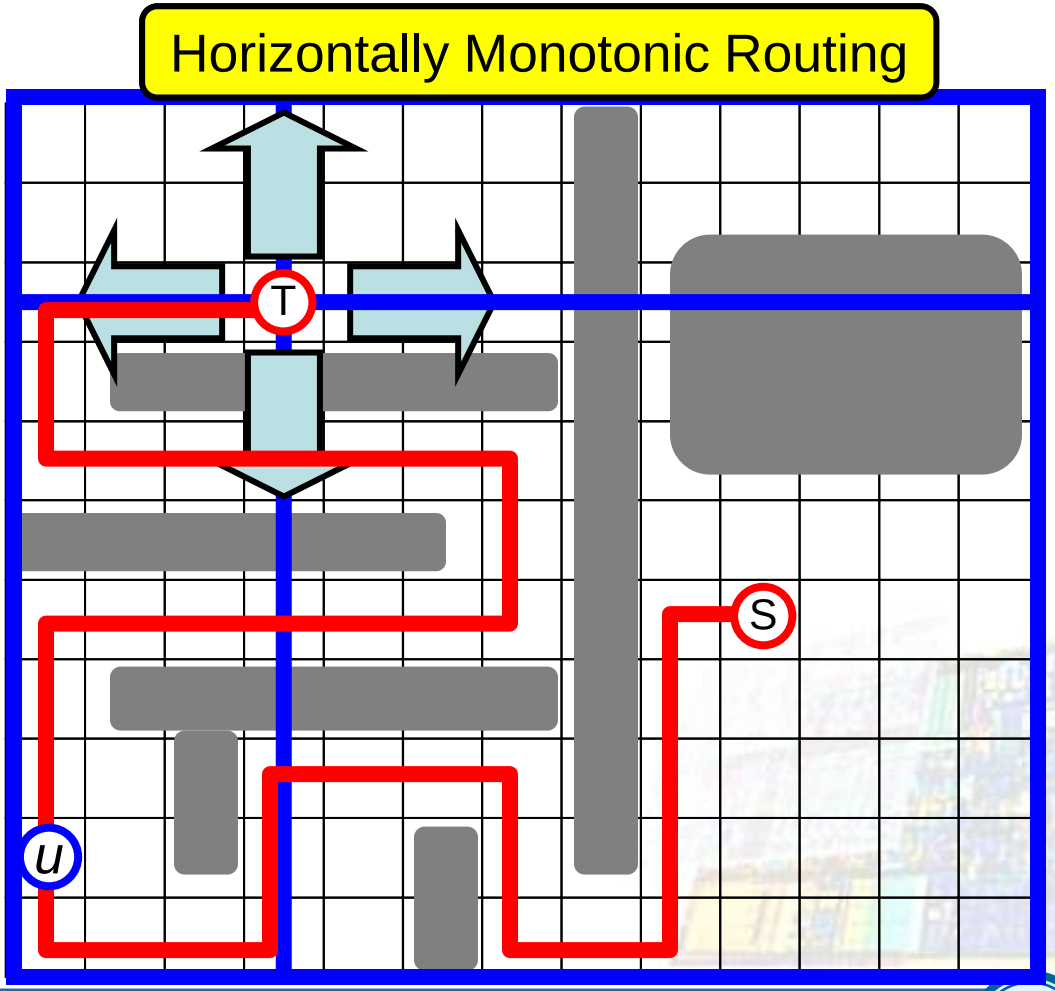
HUM Routing

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



HUM Routing

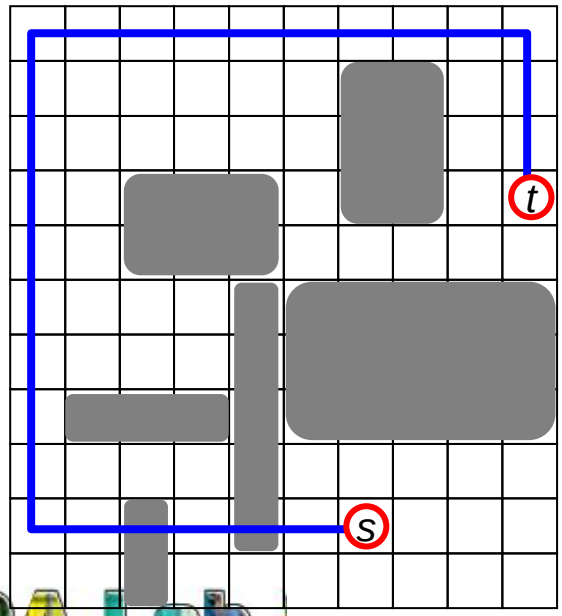
- Perform horizontally monotonic routings from *T* to the left and right boundaries.
- Perform vertically monotonic routings from *T* to the top and bottom boundaries.
- Select the best node *u* to minimize the routing cost of $P_{s,u}$ + $P_{t,u}$



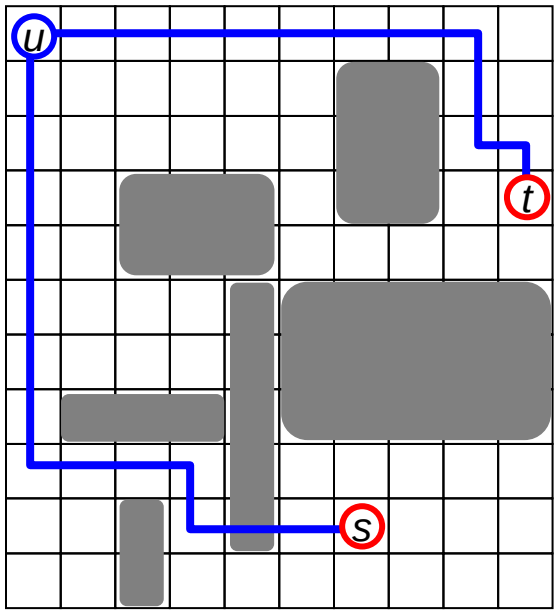
Comparison

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

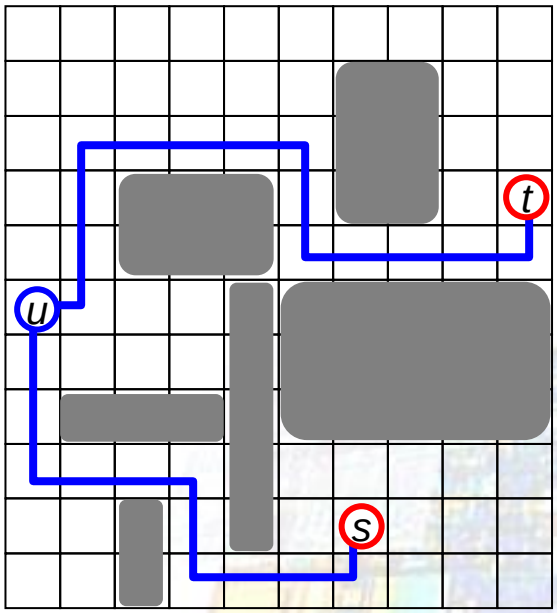
3-bend routing



Escape routing

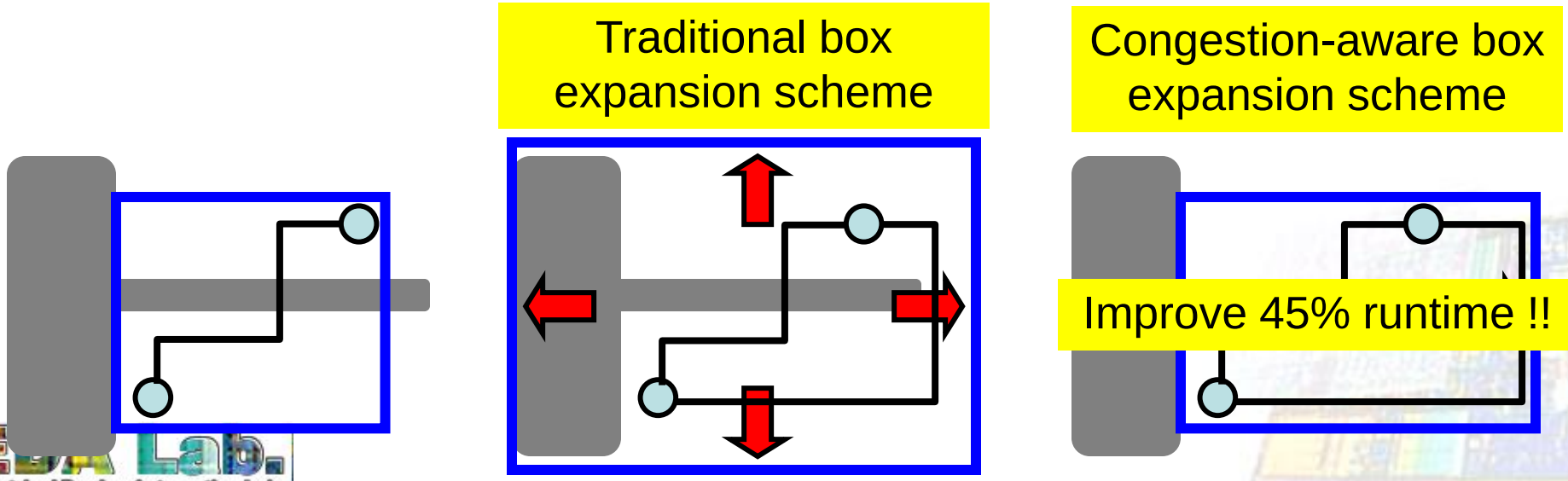


HUM routing

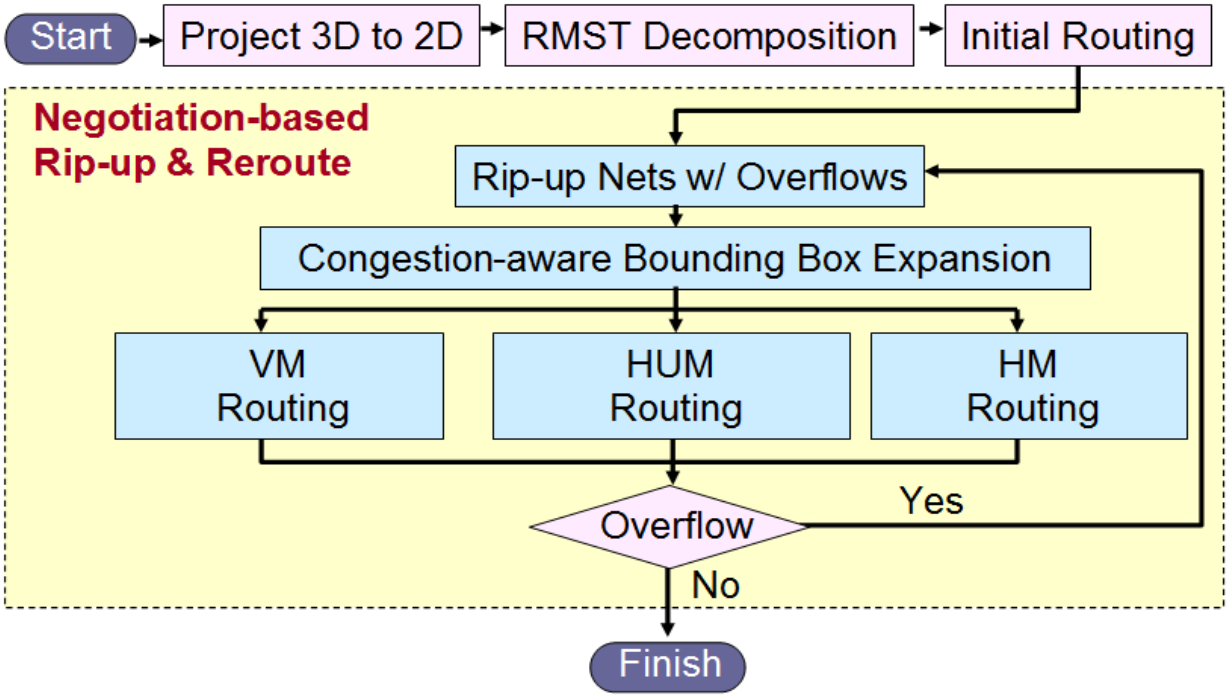


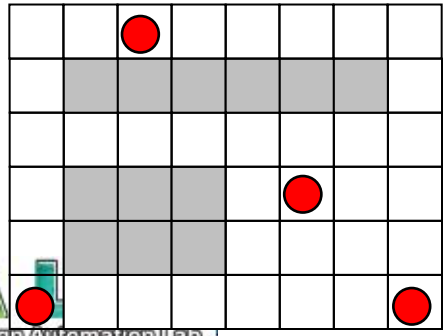
Congestion-aware Bounding Box Expansion

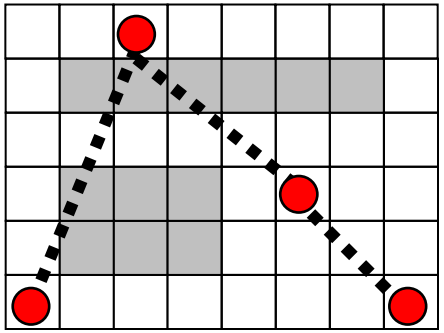
- If a net passes through a congested region, the bounding box of the net expands and then the net is rerouted.
- Over expanding the bounding box may increase the runtime, so we present a congestion-aware bounding box expansion scheme to avoid over expanding.

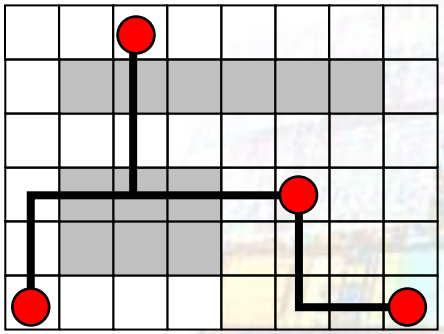


Design Flow of Grace

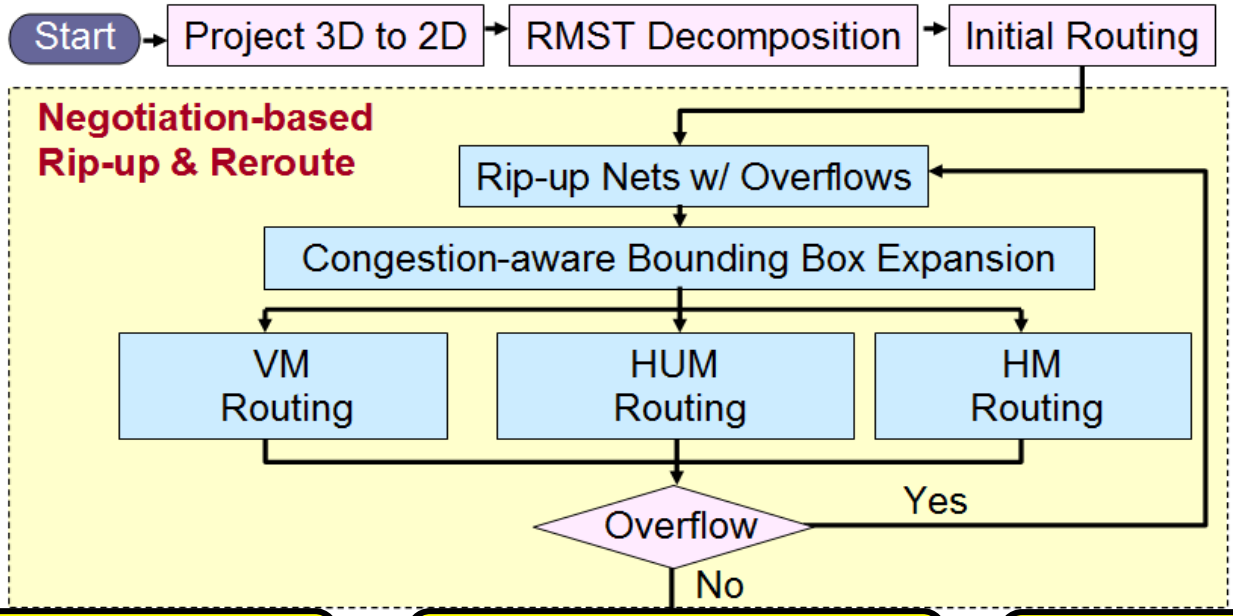




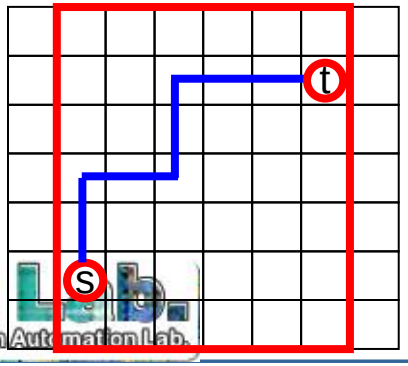




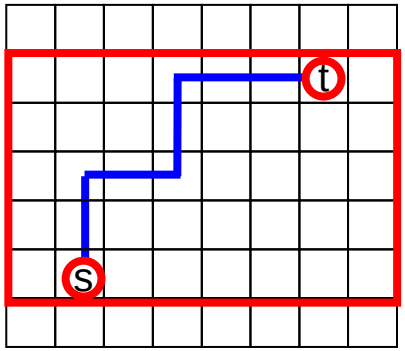
Design Flow of Grace



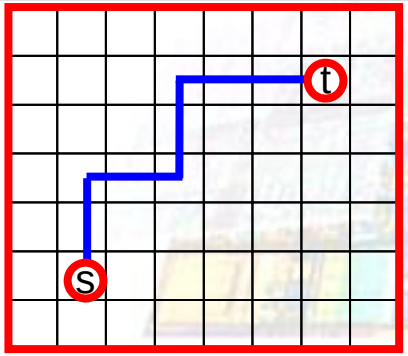
Horizontally Monotonic Routing



Vertically Monotonic Routing



HUM Routing



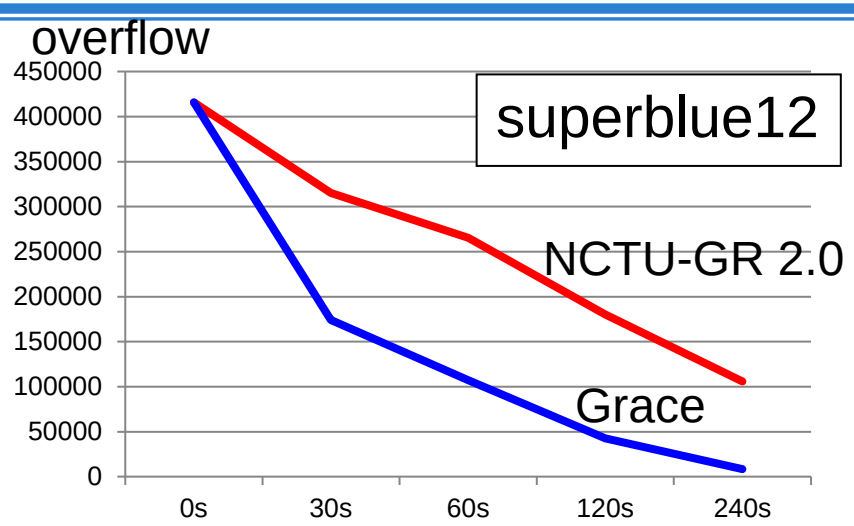
Experimental Result

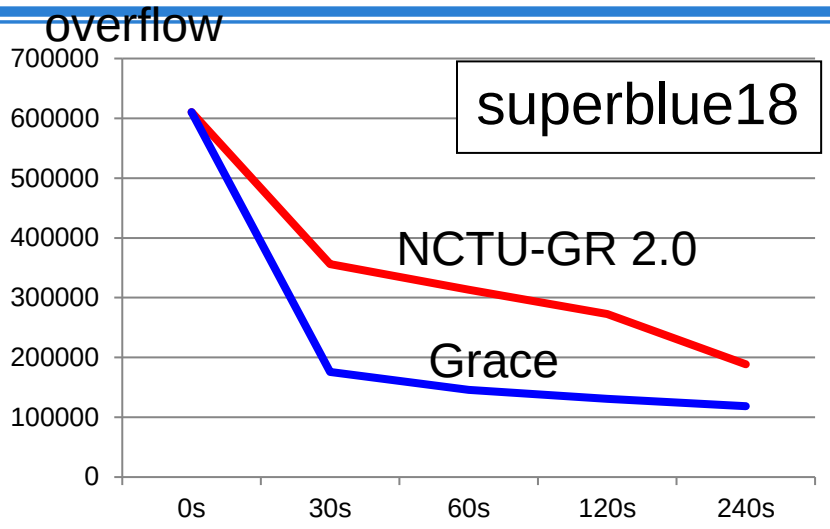
- Adopt the placement solutions of Ripple in ISPD11 placement contest to be the benchmarks

Bench-marks	NCTU-GR 2.0 [1]			Grace			
	Overflow	WL (10^5)	Time (s)	Overflow	WL (10^5)	Time (s)	speedup
superblue1	0	160.84	31	0	161.63	4	7.61
superblue2	1442	350.75	3623	1430	358.04	203	17.81
superblue4	224	122.80	536	236	124.67	30	17.75
superblue4	0	198.44	573	0	202.02	86	6.67
superblue10	39222	321.59	26509	40300	336.61	1227	21.61
superblue12	0	260.86	1333	0	279.49	484	2.75
superblue15	0	205.07	651	0	208.81	60	10.8
superblue18	109514	159.1	5174	125988	162.25	550	9.4

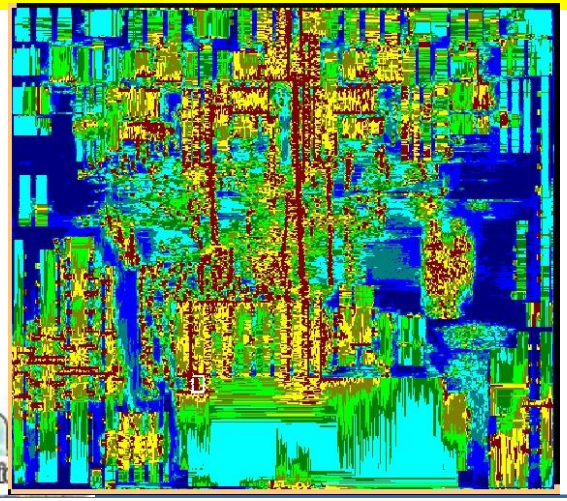
[1] W.-H. Liu, W.-C. Kao, Y.-L. Li and K.-Y. Chao, "Multi-Threaded Collision-Aware Global Routing with Bounded-Length Maze Routing" (DAC10).

Experimental Results

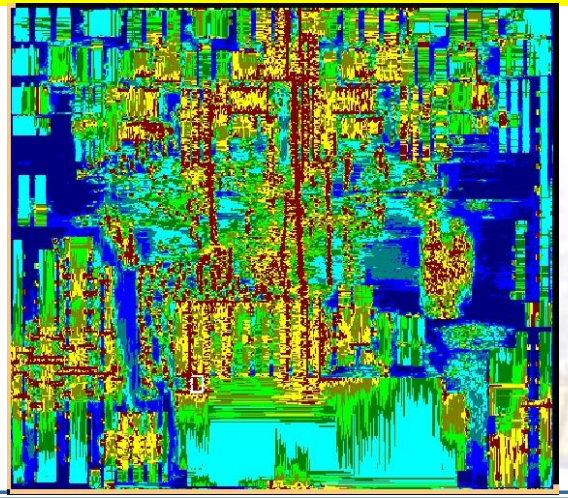




NCTU-GR 2.0 : superblue4



Grace : superblue4



Remarks: ISPD Global Routing Contests

- 2007 GR Contests
  - FGR, MaizeRouter and BoxRouter were top three
- 2008 GR Contests
  - NTHU-Route 2.0, NTUgr and FastRoute 3.0 took the top three places