《芯片设计自动化与智能优化》
Routing

The slides are partly based on Prof. David Z. Pan’s lecture notes at UT Austin.

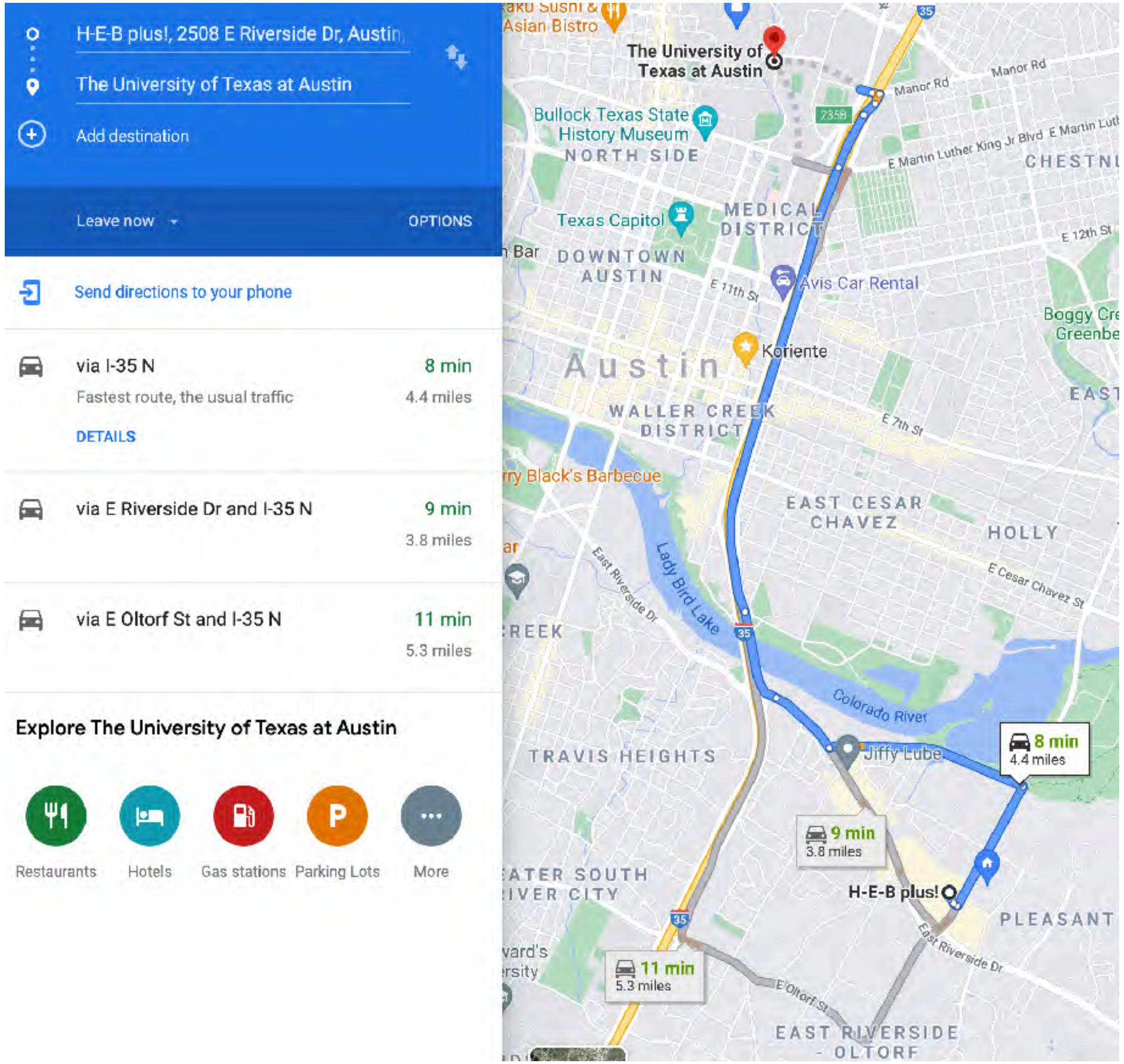
Yibo Lin

Peking University

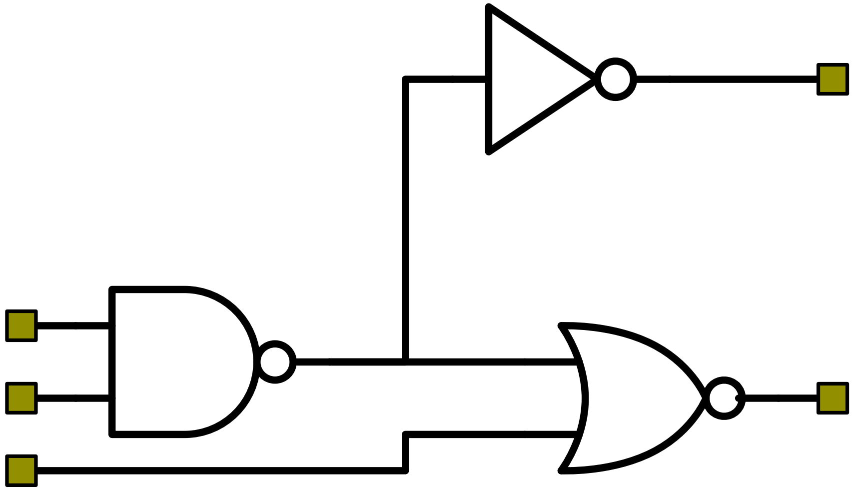
Outline

- What is routing
- Tree generation
  - Minimum Steiner tree
  - FLUTE
  - SALT
- Routing
  - Maze routing
  - Speedup maze routing
  - Global routing and detailed routing
  - Sequential routing
  - Concurrent routing

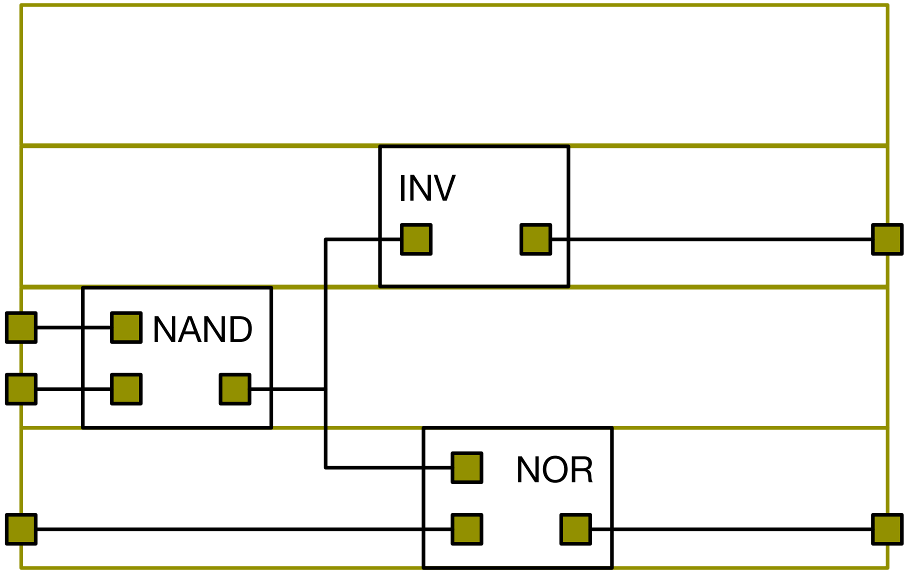
What is Routing



What is Routing

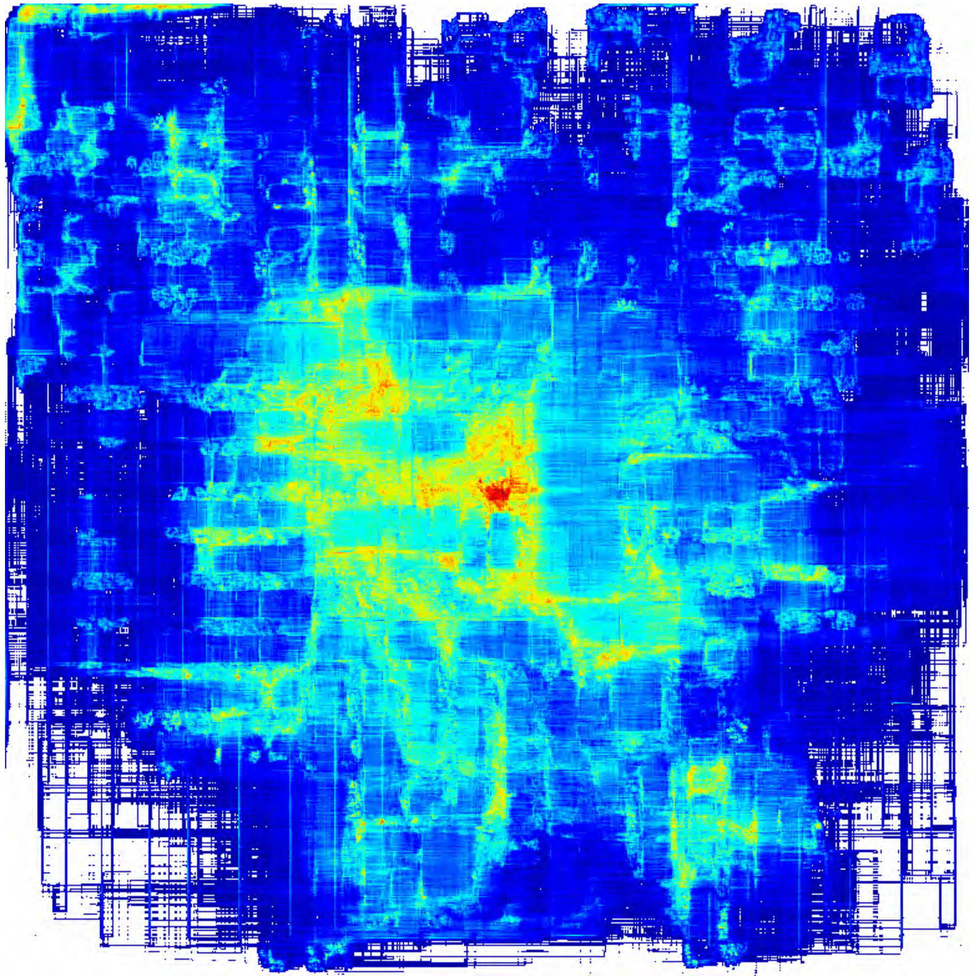


Netlist

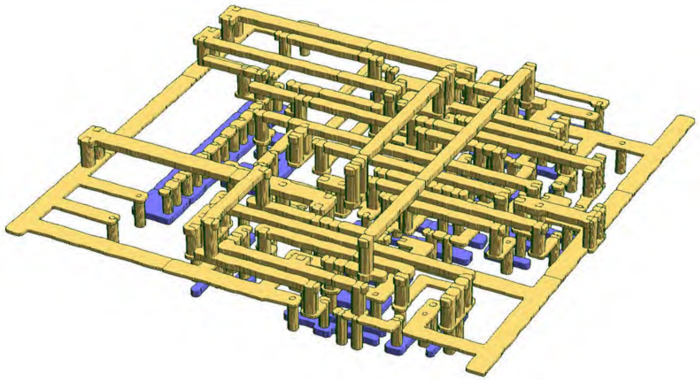


Placement and Routing

What is Routing



[Curtesy Umich]



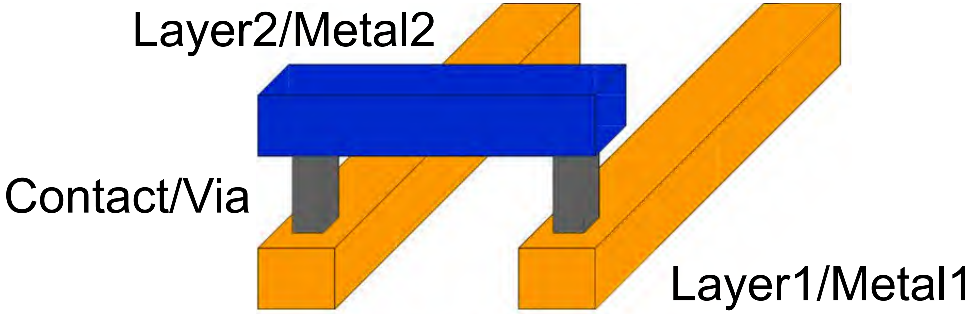
A zoom-in 3D view
[curtesy samyzaf]

Challenging problem

- 10+ metal layers
- Millions of nets
- May be highly congested
- Minimize wirelength

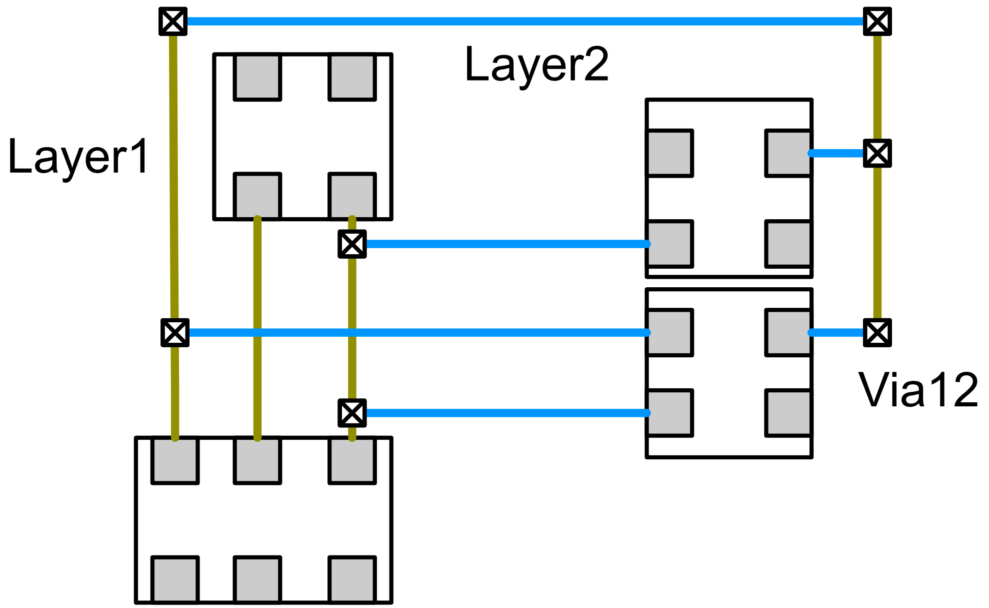
Routing Problem Formulation

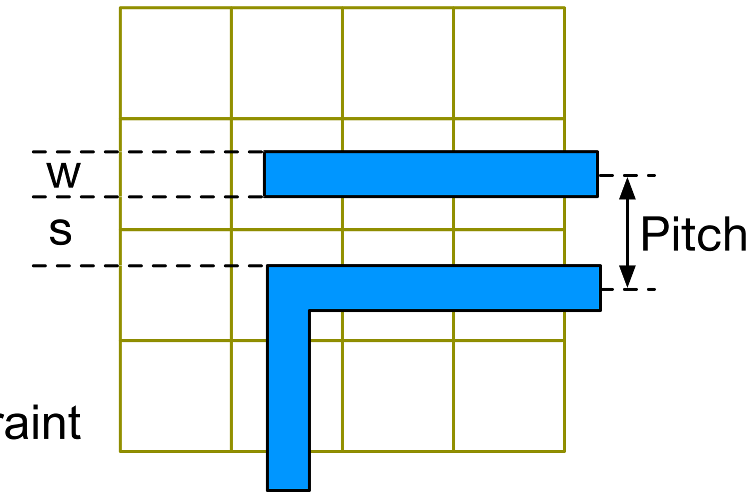
- Apply it after floorplanning/placement
- Input
  - Netlist
  - Timing budget for, typically, critical nets
  - Locations of blocks and locations of pins
- Output
  - Geometric layouts of all nets
- Objective
  - Minimize the total wire length, the number of vias, or just completing all connections without increasing the chip area.
  - Each net meets its timing budget



The Routing Constraints

- Placement constraint
- Number of routing layers
- Delay constraint
- Meet all geometrical constraints (design rules)
- Physical/Electrical/Manufacturing constraints:
  - Crosstalk
  - Process variations, yield, or lithography issues?





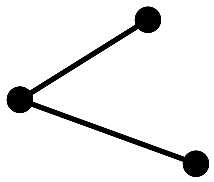
Geometrical constraint

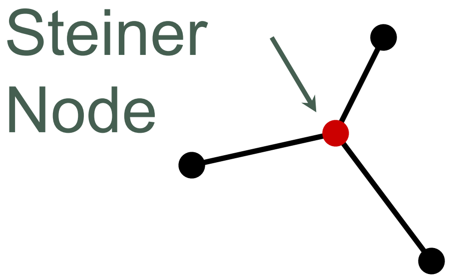
Steiner Tree

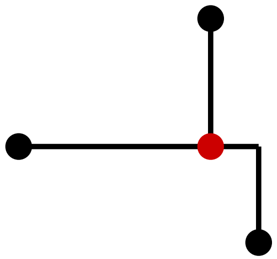
- For a multi-pin net, we can construct a spanning tree to connect all the pins together.
- But the wire length will be large.
- Better use Steiner Tree:

  A tree connecting all pins and some additional nodes (Steiner nodes).
- Rectilinear Steiner Tree:

  Steiner tree in which all the edges run horizontally and vertically.







Routing Problem is Very Hard

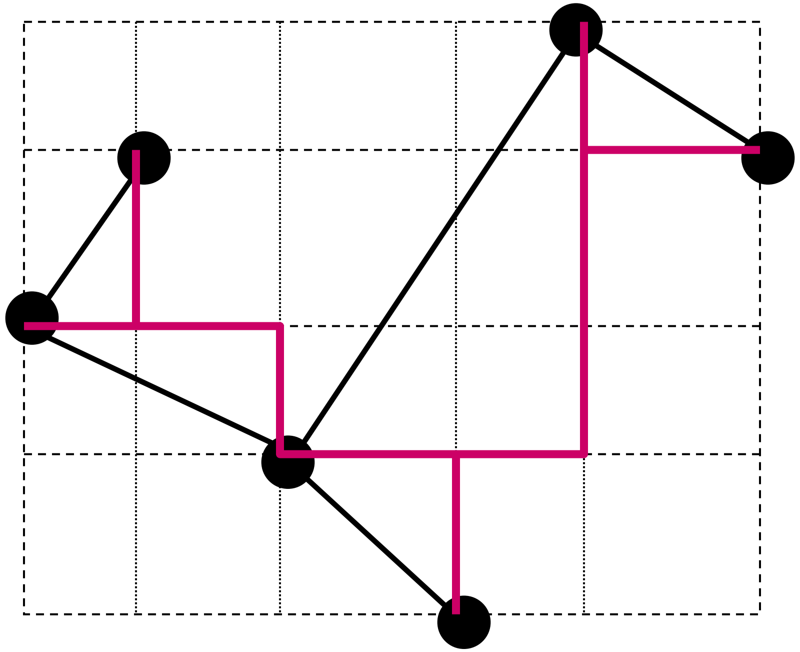
- Minimum Steiner Tree Problem:
  - Given a net, find the Steiner tree with the minimum length.
  - This problem is NP-hard!
- May need to route tens of thousands of nets simultaneously without overlapping.
- Obstacles may exist in the routing region.

Outline

- What is routing
- **Tree generation**
  - Minimum Steiner tree
  - FLUTE
  - SALT
- Routing
  - Maze routing
  - Global routing and detailed routing
  - Sequential routing
  - Concurrent routing

Steiner Tree based Algorithms

- For multi-pin nets.
- Find Steiner tree instead of shortest path.
- Construct a Steiner tree from the minimum spanning trees (MST)



FLUTE Overview

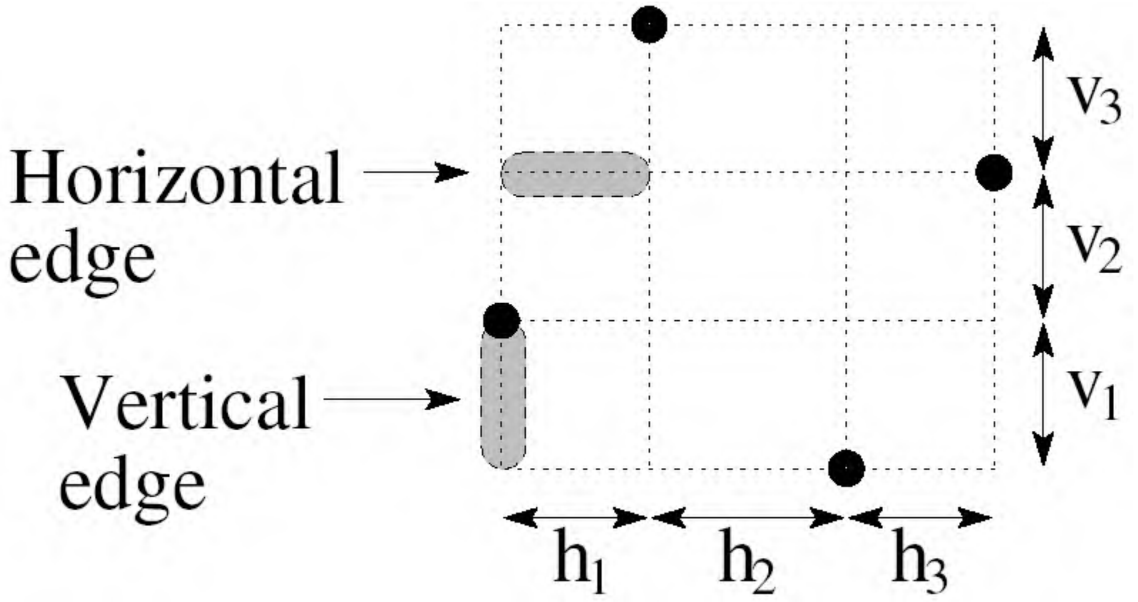
- Solve Rectilinear Steiner minimal tree (RSMT) problem:
  - Given pin positions, find a rectilinear Steiner tree with minimum WL
- Basic idea:
  - LUT to handle small nets
  - Net breaking technique to recursively break large nets
- Handling of small nets (with a few pins) is extremely well:
  - Optimal and extremely efficient for nets up to 7 pins
- So FLUTE is especially suitable for VLSI applications:
  - Over all 1.57 million nets in 18 IBM circuits [ISPD 98]
    - Average error is 0.72%
    - Runtime faster than minimum spanning tree algorithm

TCAD2007 Best Paper

Chu, Chris, and Yiu-Chung Wong. "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design." TCAD 2007.

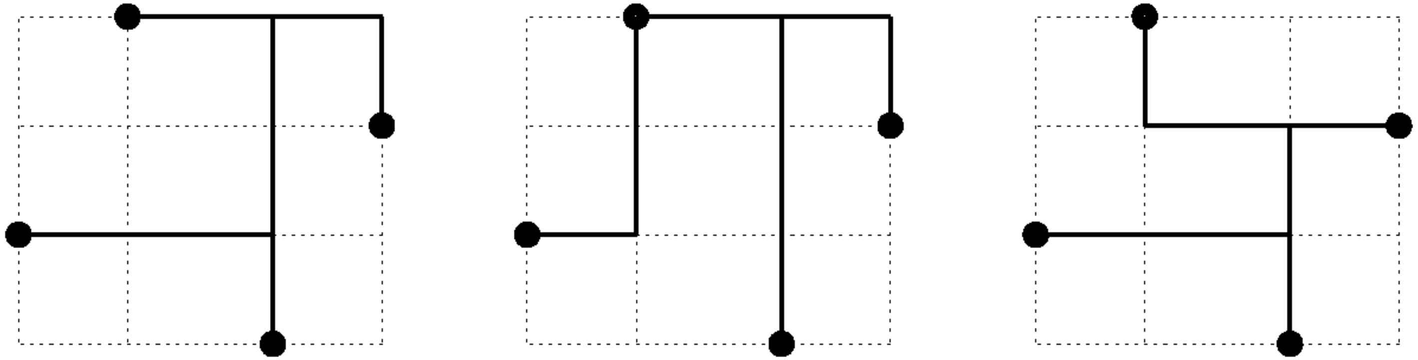
Preliminary

- A net is a set of n pins
- Degree of a net is the number of pins in it
- Consider routing along Hanan grid
- 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)

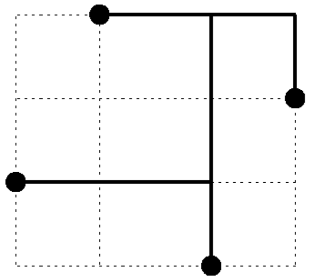
$$WL = h_1 + h_2 + h_3 + v_1 + 2v_2 + 3v_3$$
(1, 1, 1, 1, 2, 3)

$$WL = h_1 + 2h_2 + h_3 + v_1 + v_2 + v_3$$
(1, 2, 1, 1, 1, 1)

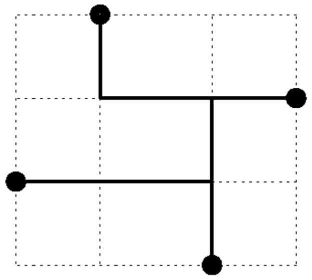
- WL can be expressed as a vector of the coefficients
- Called Wirelength Vector

Potentially Optimal WV (POWV)

- To find optimal wirelength, can enumerate all WVs
- However, most WVs can never produce optimal WL
  - (1, 2, 1, 1, 1, 2) is redundant as it always produces a larger WL than (1, 2, 1, 1, 1, 1)



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



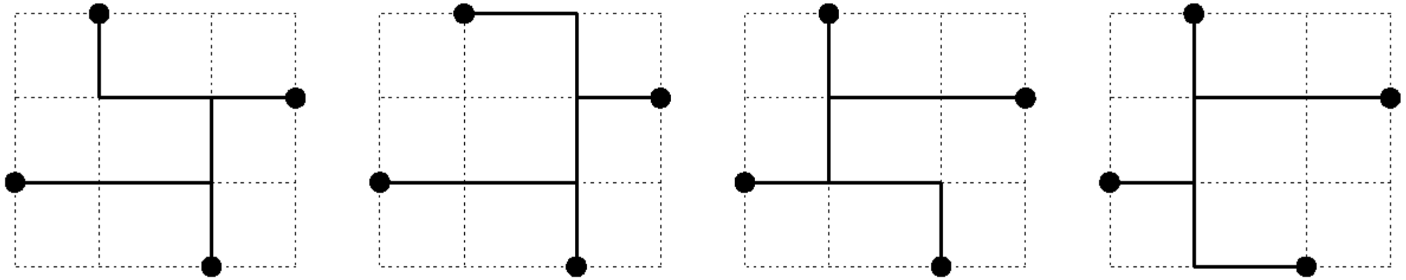
(1, 2, 1, 1, 1, 1)

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

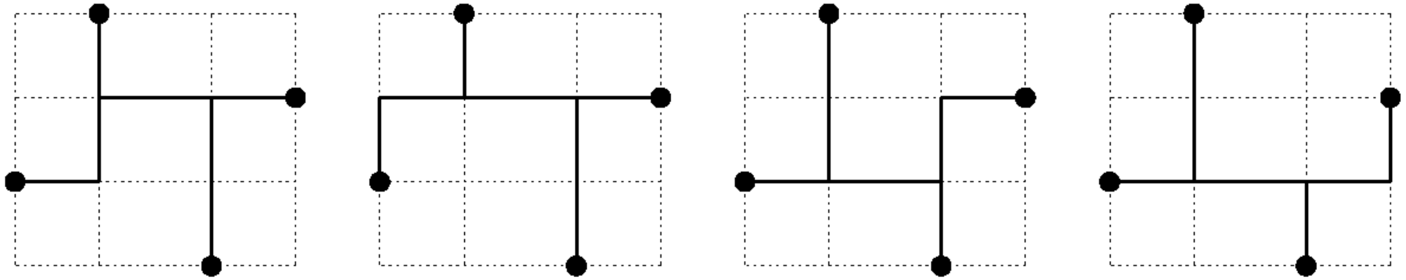
of POWVs is Very Small

- 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)



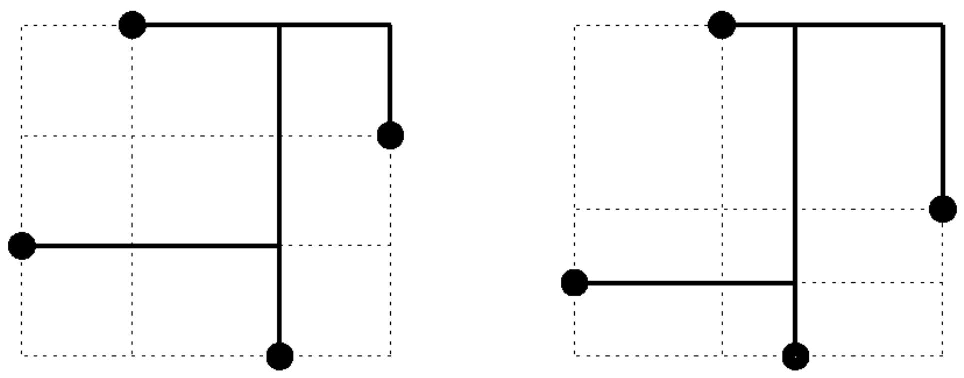
POWV
(1,1,1,1,2,1)



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

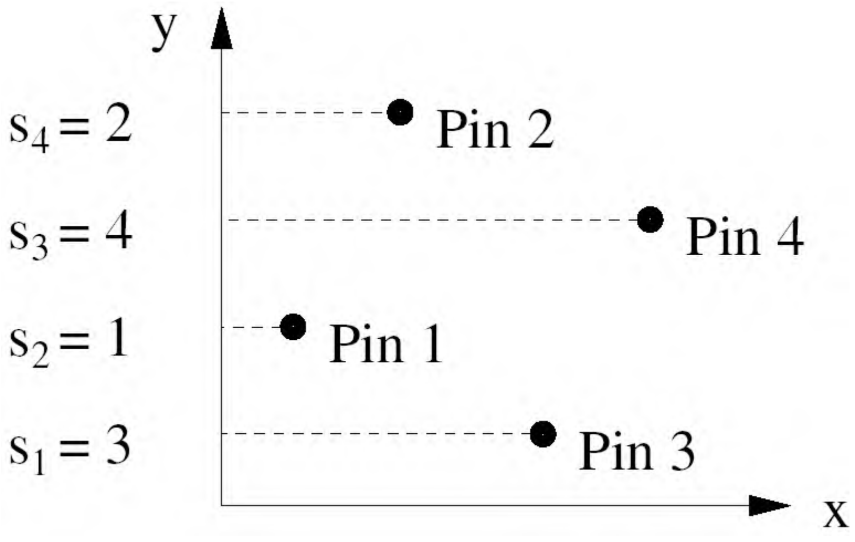
- We try to group together nets that can share the same set of POWVs
- For example, these two nets share the same set of POWVs:



Grouping by Vertical Sequence

- Define vertical sequence $s_1 s_2 \ldots s_n$ to be the list of pin indexes sorted in y-coordinate

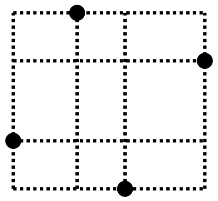
Vertical sequence = 3142



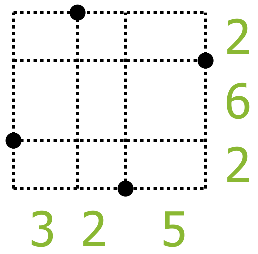
- Lemma: The set of all degree-n nets can be divided into $n!$ groups according to the vertical sequence such that all nets in each group share the same set of POWVs

Steps in FLUTE WL Estimation

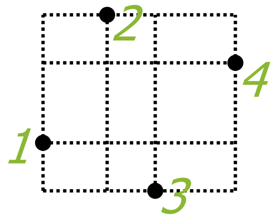
1. Input a net



2. Find h_i's and v_i's



3. Find vertical sequence



3142

4. Get POWVs from LUT

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

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

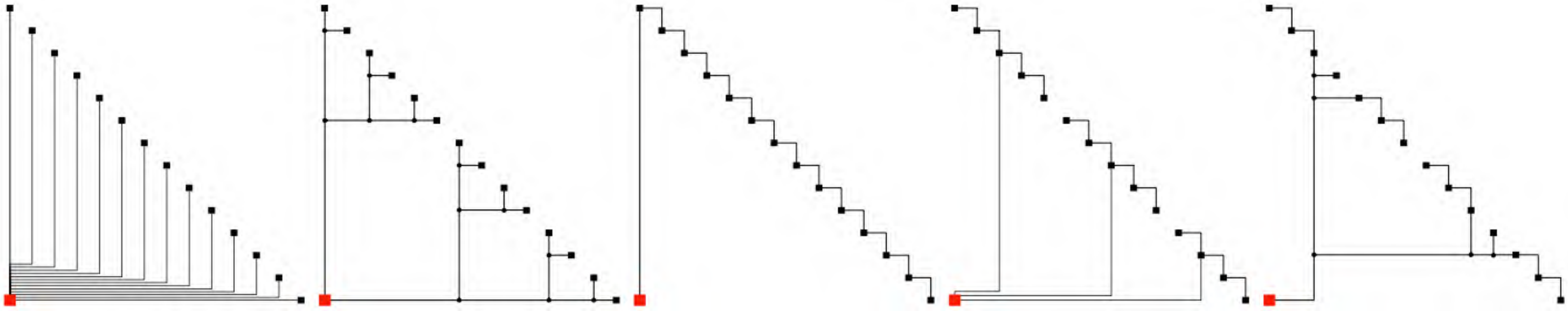
5. Find WL for each POWV and return the best

HPWL + h_2 = 22 ← return

HPWL + v_2 = 26

- *Remark:*
- One RSMT topology can also be pre-computed and stored for each POWV
- Impractical for high-degree nets (degree >= 9)
  - Other technique to break down high-degree nets

SALT – Steiner Shallow-Light Tree Algorithm



Spanning SPT (shortest path tree)

RSMA (rectilinear Steiner minimum arborescence)

RMST/RSMT

Spanning SLT (shallow-light tree)

Steiner SLT

Chen, Gengjie, and Evangeline FY Young. "SALT: Provably good routing topology by a novel Steiner shallow-light tree algorithm." TCAD 2019.

SALT – Steiner Shallow-Light Tree Algorithm

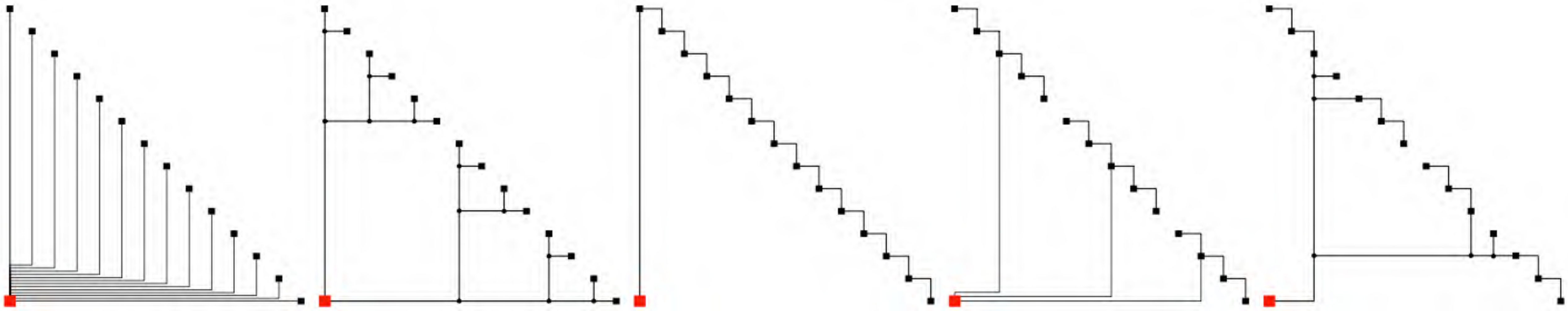
- Trade-offs between path length and tree weight
  - **Path length**: implies wire delay
  - **Tree weight**: implies routing resource usage (routability), power consumption, cell delay and wire delay
- Spanning/Steiner $(\bar{\alpha}, \bar{\beta})$-shallow-light tree (SLT) T
  - **Shallowness** $\alpha = \max\{\frac{d_T(r,v)}{d_G(r,v)} | v \in V \backslash \{r\}\} \leq \bar{\alpha}$
    - $d_G(r, v)$: distance from v to root r on graph/metric G
  - **Lightness** $\beta = \frac{w(T)}{w(MST(G))} \leq \bar{\beta}$

[ICCAD2017 Best Paper]

Chen, Gengjie, and Evangeline FY Young. "SALT: Provably good routing topology by a novel Steiner shallow-light tree algorithm." TCAD 2019.

SALT – Steiner Shallow-Light Tree Algorithm

	Shallowest	Lightest	Shallow-light
Spanning	Spanning SPT ($O(m+n\log n)$)	MST ($O(m+n\log n)$)	Spanning SLT
Steiner	Steiner SPT (NP-hard)	SMT (NP-hard)	Steiner SLT
Rectilinear Steiner	RSMA (NP-hard)	RSMT (NP-hard)	Rectilinear Steiner SLT



Spanning SPT

$\alpha = \frac{13}{13}, \beta = \frac{182}{39}$

RSMA

$\alpha = \frac{13}{13}, \beta = \frac{54}{39}$

RMST/RSMT

$\alpha = \frac{39}{13}, \beta = \frac{39}{39}$

Spanning SLT

$\alpha = \frac{17}{13}, \beta = \frac{61}{39}$

Steiner SLT

$\alpha = \frac{17}{13}, \beta = \frac{44}{39}$

Chen, Gengjie, and Evangeline FY Young. "SALT: Provably good routing topology by a novel Steiner shallow-light tree algorithm." TCAD 2019.

Previous Work

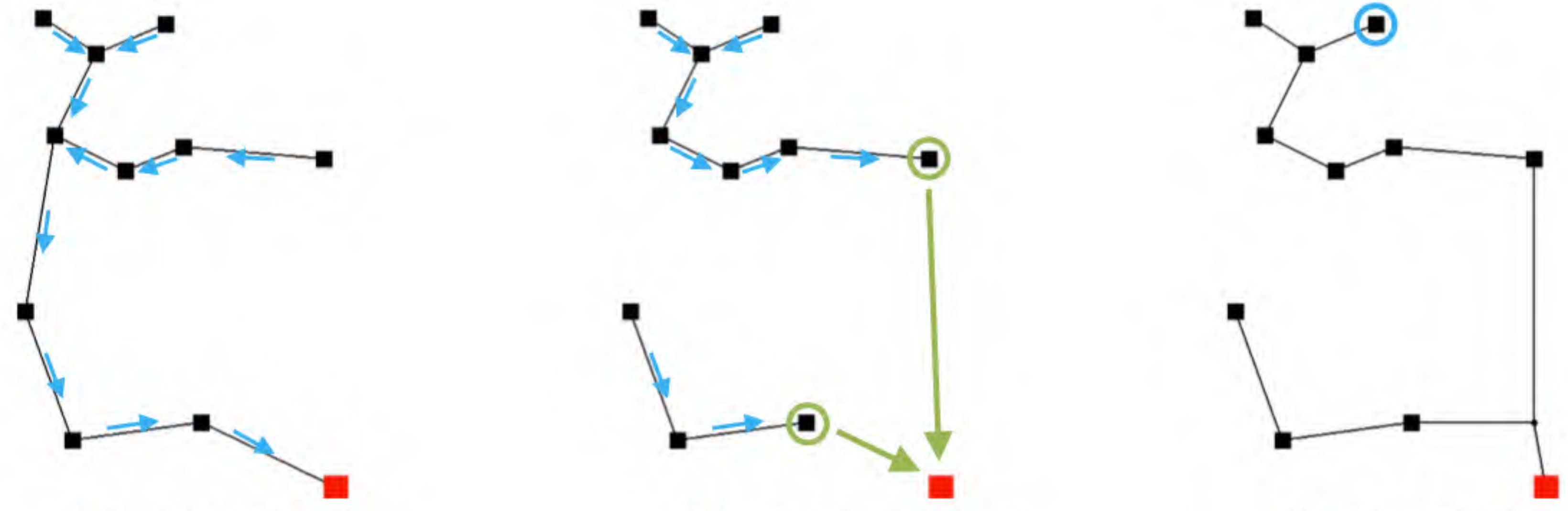
- Spanning $(1+\epsilon, O(\frac{1}{\epsilon}))$-SLT
  - ABP/BRBC $(1+2\epsilon, O(\frac{2}{\epsilon}))$ [Awerbuch, TR'91] [Cong, TCAD'92]
  - KRY $(1+\epsilon, O(\frac{2}{\epsilon}))$ [Khuller, SODA'93, Algorithmica'95]
- Steiner $(1+\epsilon, O(\log\frac{1}{\epsilon}))$-SLT
  - ES $(1+2\epsilon, 4+2\left\lceil\log\frac{1}{\epsilon}\right\rceil)$ [Elkin, FOCS'11, SICOMP'15]
  - PD combines SPT and MST [Alpert, TCAD'95]
  - Bonn trades off between cell and wire delay [Scheifele, ICCAD'16, Algorithmica'17]

SALT

- Steiner SLT on a general graph
  - $(1+\epsilon, 2+\left\lceil \log \frac{2}{\epsilon} \right\rceil)$-SLT
- Runtime complexity
  - $O(n^2) \rightarrow O(n \log n)$ in Manhattan space

SALT

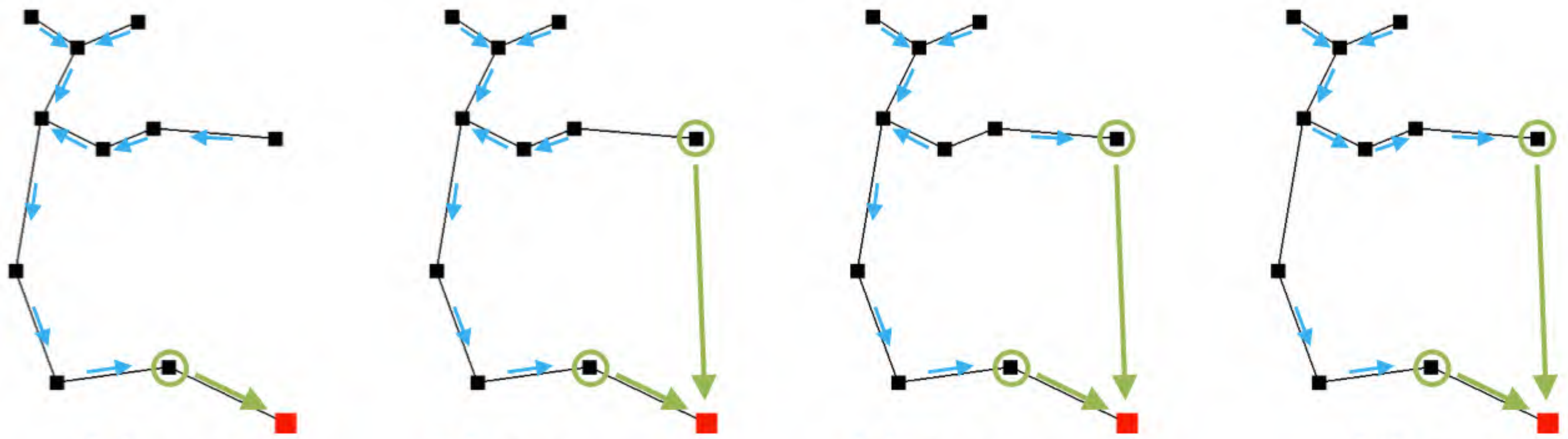
- Construct **MST** T_M
- Identify **breakpoints** B during **DFS** on T_M, which results to forest F
- Obtain **Steiner SPT** T_B on $G[B \cup \{r\}]$, and $T = F \cup T_B$ is the output



(a) MST T_M (b) Forest F (c) SALT T

SALT – DFS and Breakpoints

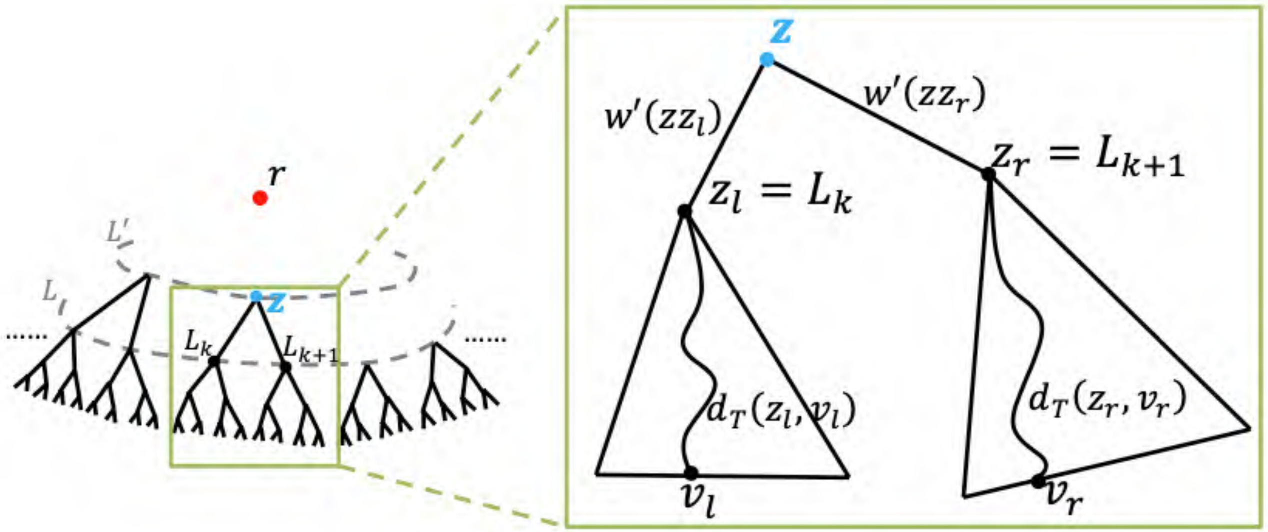
- Make sure $d_T(r,v) \le \bar{\alpha} \cdot d_G(r,v)$
  - Breakpoints will be connected to r by shortest paths
  - Other vertexes also benefit

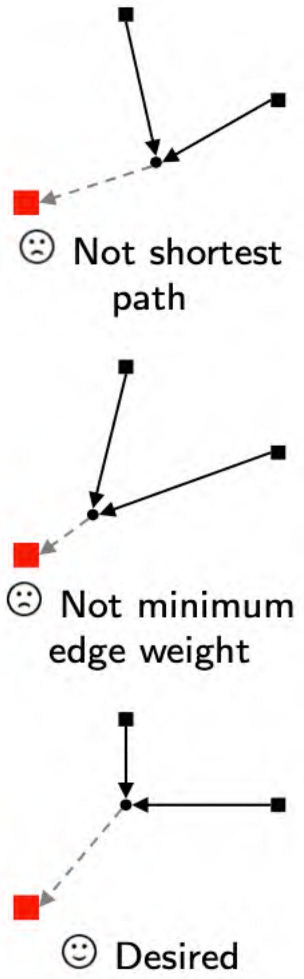


(a) Initial (b) Path length improved (c) Further improved (d) Final

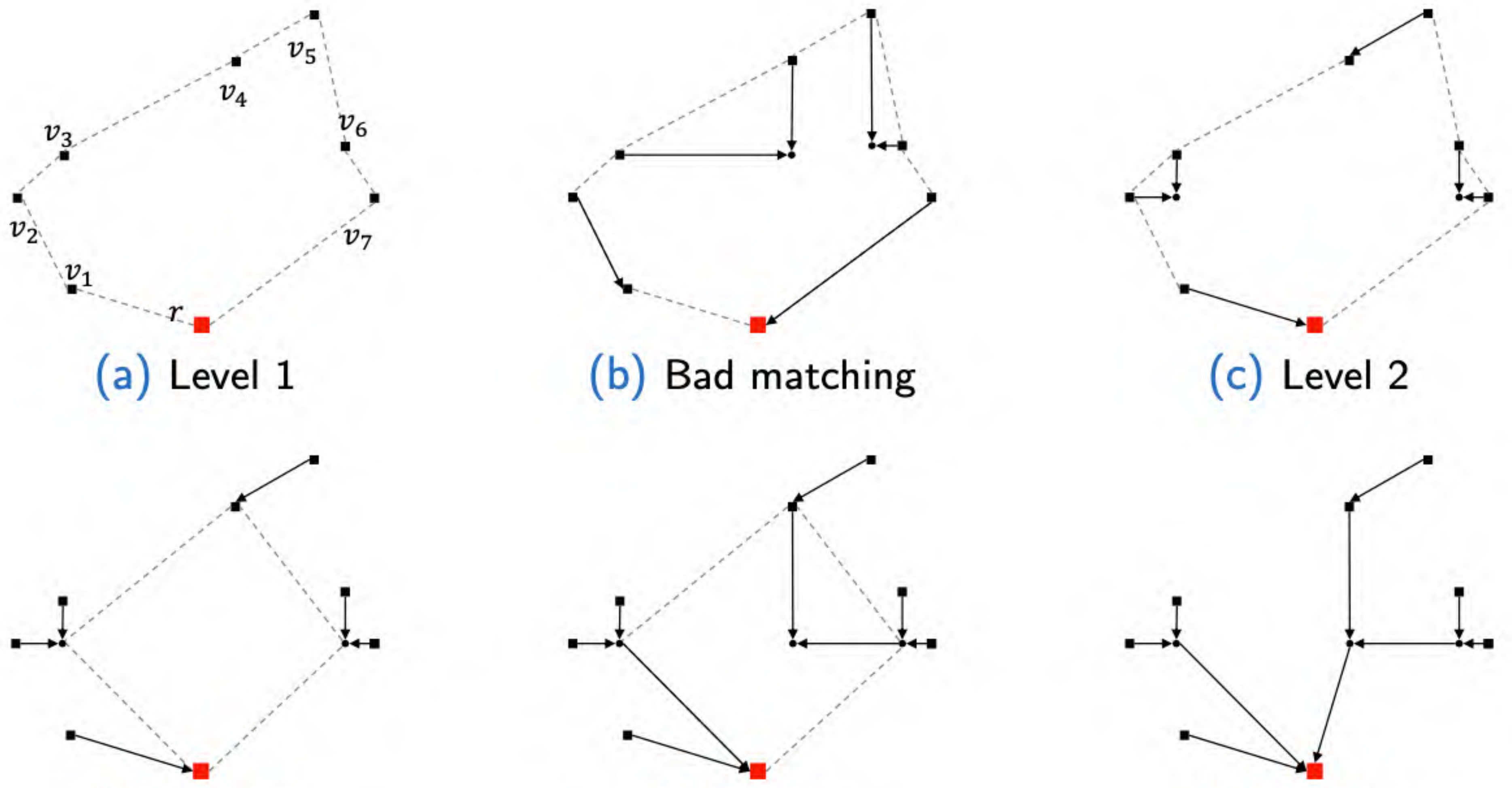
SALT – Light Steiner SPT T_B

- A full balanced binary tree
- Constructed level-by-level from bottom
- Merge neighboring vertexes pair by pair into Steiners in each level
  - Determine Steiner by minimizing edge weights while preserving shortest paths
  - Select a light matching for pairing up along (Hamiltonian) circle





SALT – Example on Manhattan Space

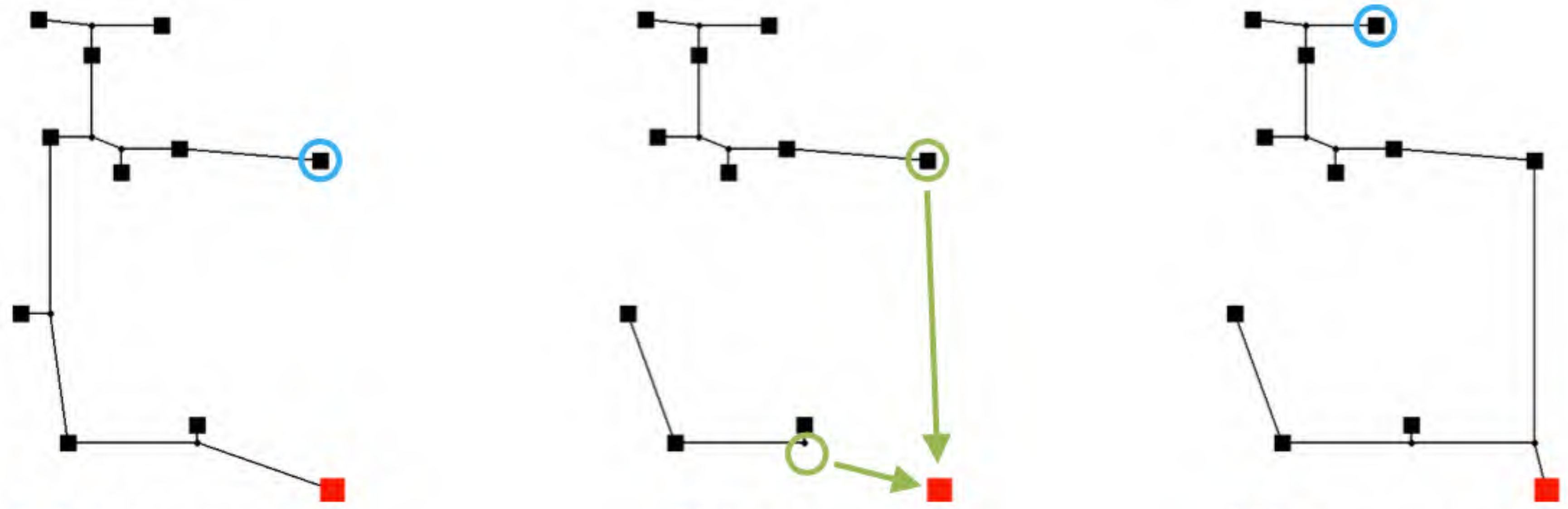


(a) Level 1 (b) Bad matching (c) Level 2

(d) Level 2 (e) Level 3 (f) Level 4

Rectilinear SALT

- Construct **RSMT** T_M by **FLUTE** [Chu, TCAD2008]
- Get breakpoints B and forest F
- Obtain **RSMA** T_B on $G[B \cup \{r\}]$ by **CL** [Cordova,TR1994], and $T = F \cup T_B$ is the output

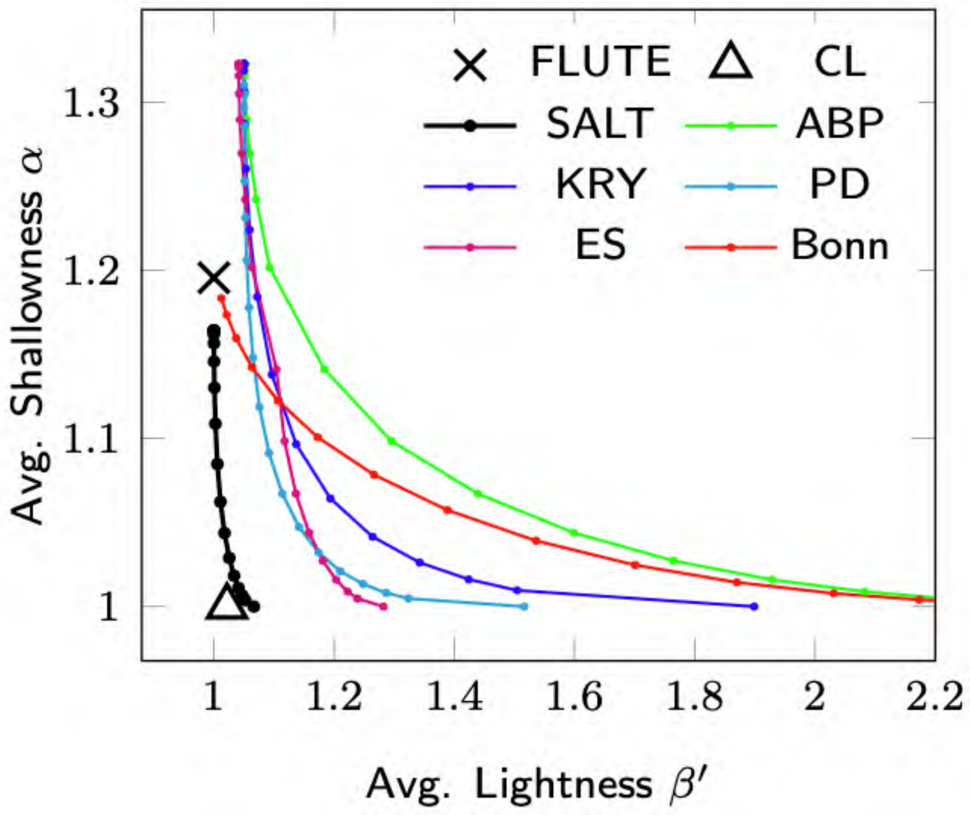


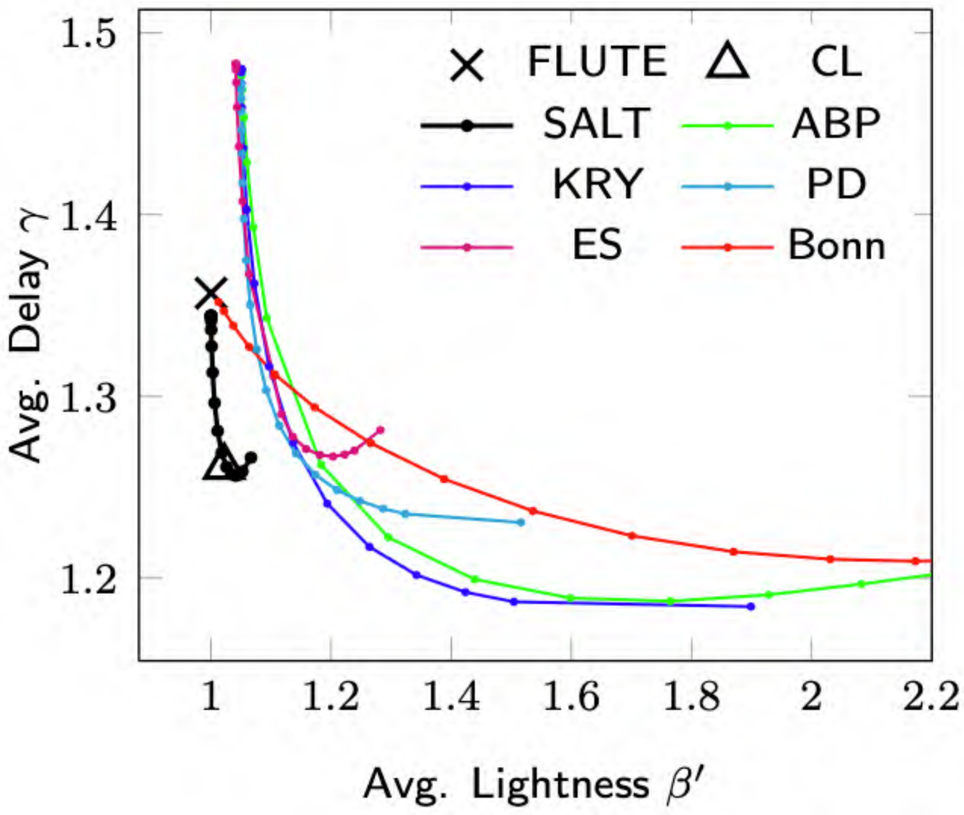
(a) RSMT T_M by FLUTE

(b) Forest F

(c) Rectilinear SALT T

Comparison between Different Algorithms





Chen, Gengjie, and Evangeline FY Young. "SALT: Provably good routing topology by a novel Steiner shallow-light tree algorithm." TCAD 2019.

Summary for Tree Generation

- Minimum Steiner tree
  - NP problem
- FLUTE
  - LUT-based method
  - Work well on nets with degrees <= 7
- SALT
  - Balance shallowness and lightness

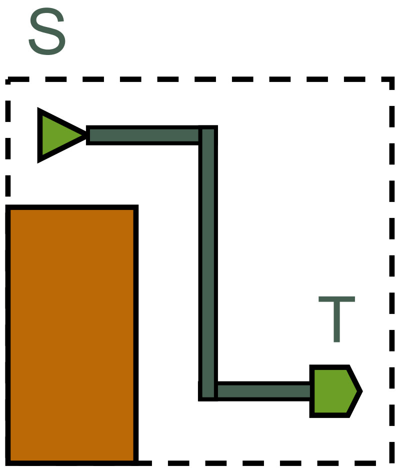
Outline

- What is routing
- Tree generation
  - Minimum Steiner tree
  - FLUTE
  - SALT
- **Routing**
  - Maze routing
  - Global routing and detailed routing
  - Sequential routing
  - Concurrent routing

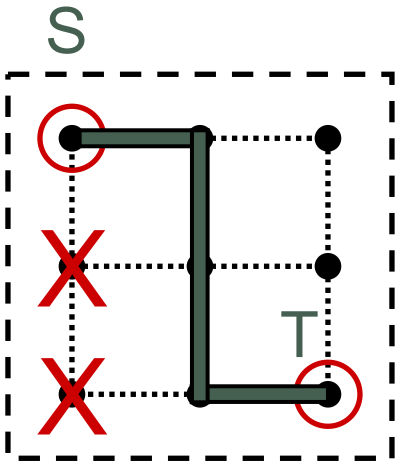
Maze Routing Problem

- Input
  - A planar rectangular grid graph.
  - Two points S and T on the graph.
  - Obstacles modeled as blocked vertices.
- Objective
  - Find the shortest path connecting S and T.
- This technique can be used in global or detailed routing problems.

Grid Graph



Area Routing

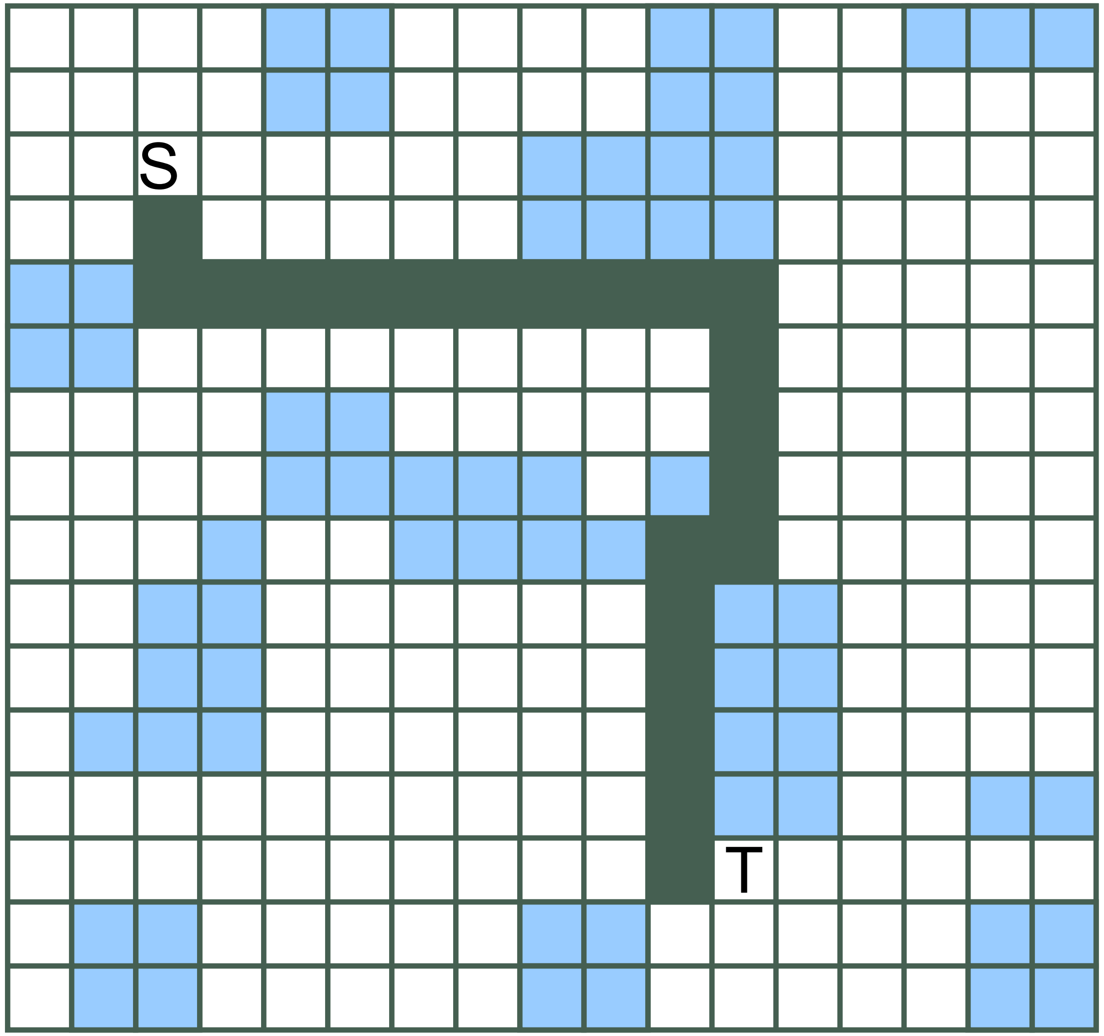


Grid Graph (Maze)

S	✓	
X	✓	
X	✓	T

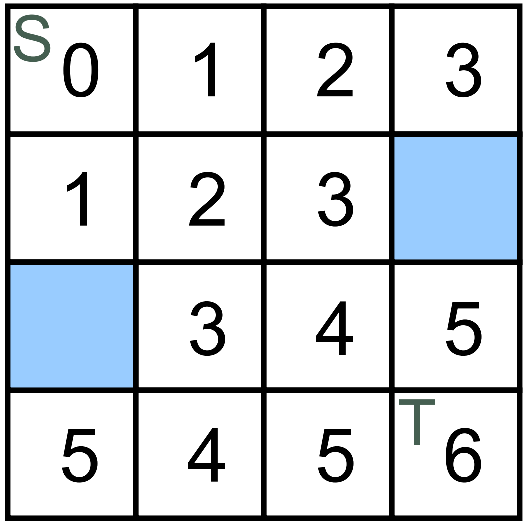
Simplified Representation

Maze Routing



Lee's Algorithm

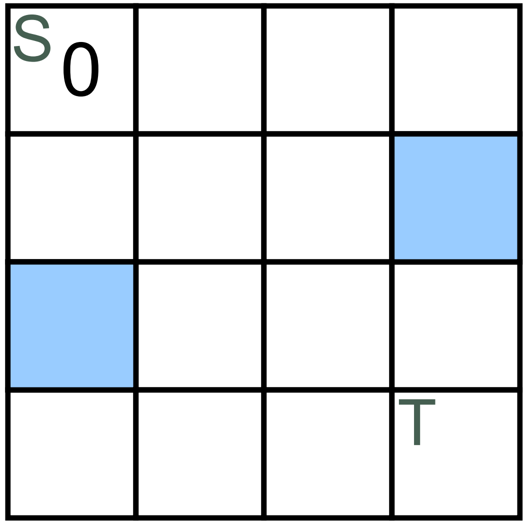
- Basic idea
- A Breadth-First Search (BFS) of the grid graph.
- Always find the shortest path possible.
- Consists of two phases:
  - Wave Propagation
  - Retrace



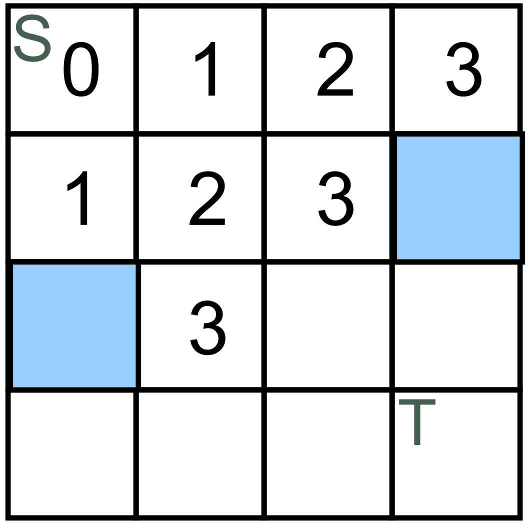
Lee, Chin Yang. "An algorithm for path connections and its applications." IRE transactions on electronic computers 3 (1961): 346-365.

Wave Propagation

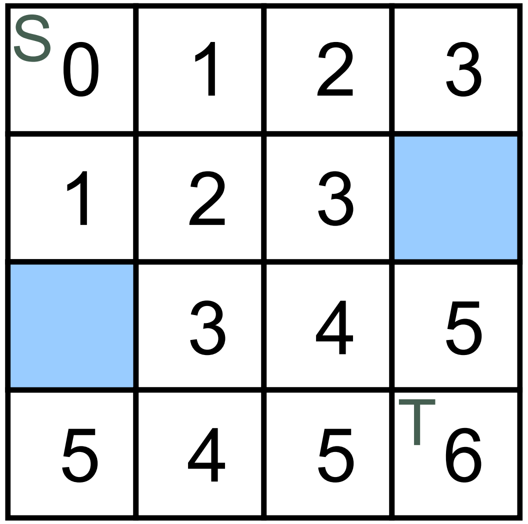
- At step k, all vertices at Manhattan-distance k from S are labeled with k.
- A Propagation List (FIFO) is used to keep track of the vertices to be considered next.



After Step 0



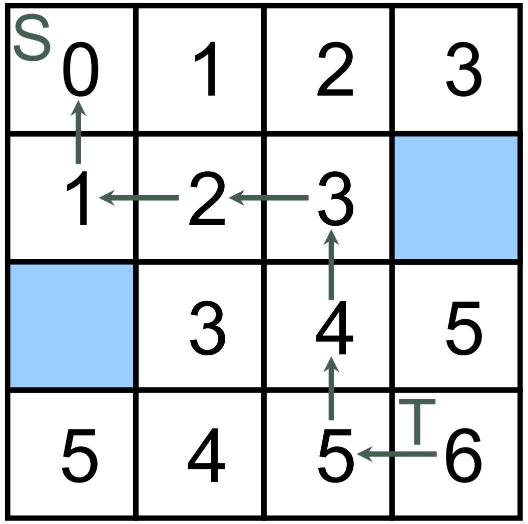
After Step 3



After Step 6

Retrace

- Trace back the actual route.
- Starting from *T*.
- At *vertex with k, go to any vertex with label k*-1.



Final labeling

How many grids visited using Lee's algorithm?

13	12	11	10			7	6	7	7			9	10			
12	11	10	9			6	5	6	7			8	9	10	11	12
11	10	9	8	7	6	5	4					7	8	9	10	11
10	9	8	7	6	5	4	3					6	7	8	9	10
		7	6	5	4	3	2	1	2	3	4	5	6	7	8	9
		6	5	4	3	2	1	S	1	2	3	4	5	6	7	8
9	8	7	6			3	2	1	2	3	4	5	6	7	8	9
10	9	8	7						3		5	6	7	8	9	10
11	10	9	8	9	10					7	6	7	8	9	10	11
12	11			10	11	12	11	10	9	8			9	10	11	12
13	12			11	12	13	12	11	10	9			10	11	12	13
				12	13		13	12	11	10			11	12	13	
				13				13	12	11			12	13		
									13	12	T		13			
										13						

Time and Space Complexity

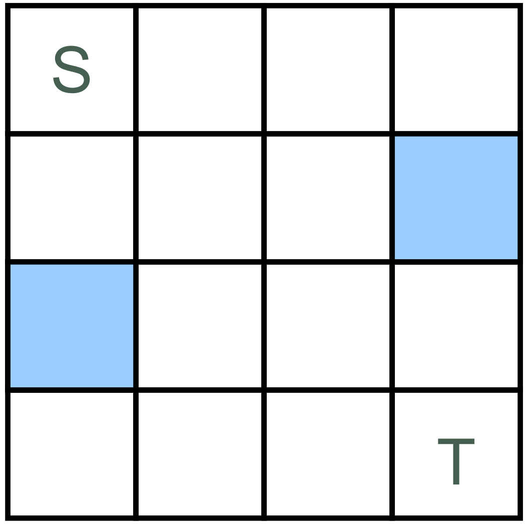
- For a grid structure of size $w \times h$:
  - Time per net = O(wh)
  - Space = O(wh log wh) (O(log wh) bits are needed to store each label.)
- For a 4000 $\times$ 4000 grid structure:
  - 24 bits per label
  - Total 48 Mbytes of memory!

Improvement to Lee’s Algorithm

- Improvement on memory:
  - Aker’s Coding Scheme
- Improvement on run time:
  - Starting point selection
  - Double fan-out
  - Framing
  - Hadlock’s Algorithm
  - Soukup’s Algorithm

Aker's Coding Scheme to Reduce Memory Usage

- For the Lee's algorithm, labels are needed during the retrace phase.
- But there are only two possible labels for neighbors of each vertex labeled i, which are, $i-1$ and $i+1$.
- So, is there any method to reduce the memory usage?
- **One bit** (independent of grid size) is enough to distinguish between the two labels.



Sequence:
...... (what sequence?)

(Note: In the sequence, the labels before and after each label must be different in order to tell the forward or the backward directions.)

Akers, Sheldon B. "A modification of Lee's path connection algorithm." *IEEE Transactions on Electronic Computers* 1 (1967): 97-98.

Aker's Coding Scheme to Reduce Memory Usage

- For the Lee's algorithm, labels are needed during the retrace phase.
- But there are only two possible labels for neighbors of each vertex labeled i, which are, $i-1$ and $i+1$.
- So, is there any method to reduce the memory usage?
- **One bit** (independent of grid size) is enough to distinguish between the two labels.

S	0	1	0
0	1	0	
	0	1	0
0	1	0	1T

Correct?

Akers, Sheldon B. "A modification of Lee's path connection algorithm." *IEEE Transactions on Electronic Computers* 1 (1967): 97-98.

Aker's Coding Scheme to Reduce Memory Usage

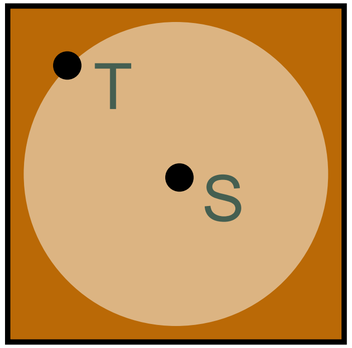
- For the Lee's algorithm, labels are needed during the retrace phase.
- But there are only two possible labels for neighbors of each vertex labeled i, which are, $i-1$ and $i+1$.
- So, is there any method to reduce the memory usage?
- **One bit** (independent of grid size) is enough to distinguish between the two labels.

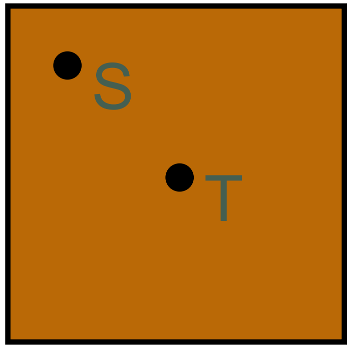
S	1	1	0
1	1	0	
	0	0	1
1	0	1	1T

Akers, Sheldon B. "A modification of Lee's path connection algorithm." *IEEE Transactions on Electronic Computers* 1 (1967): 97-98.

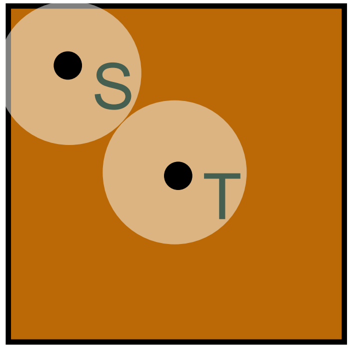
Schemes to Reduce Run Time

1. Starting Point Selection:

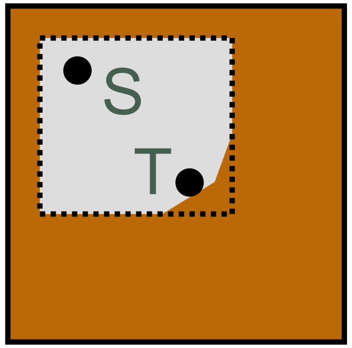




2. Double Fan-Out:



3. Framing:

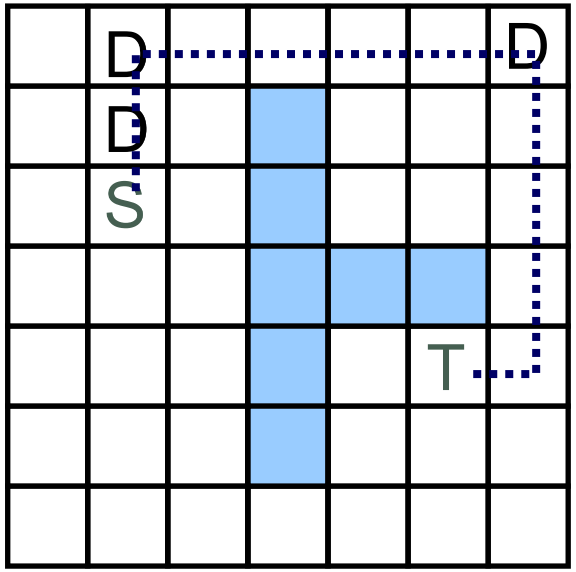


Hadlock's Algorithm to Reduce Run Time

- Detour number
- For a path P from S to T, let detour number $d(P)$ = # of grids directed away from T, then
- $$L(P) = MD(S,T) + 2d(P)$$

length

shortest Manhattan distance



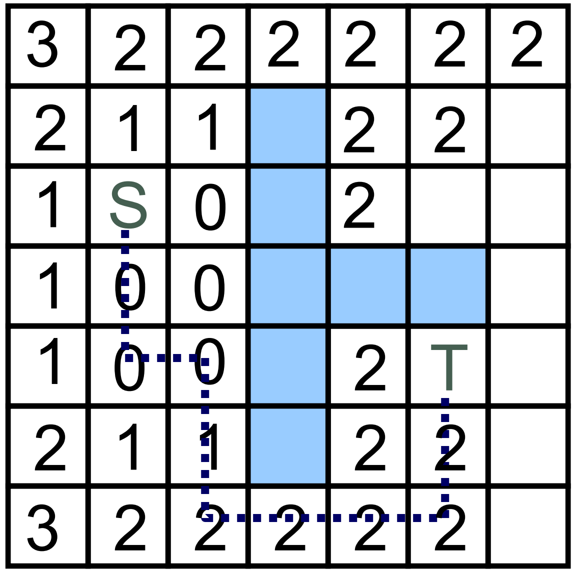
D: Detour

$d(P) = 3$
$MD(S,T) = 6$
$L(P) = 6 + 2\times3 = 12$

- So minimizing $L(P)$ and $d(P)$ are the same.

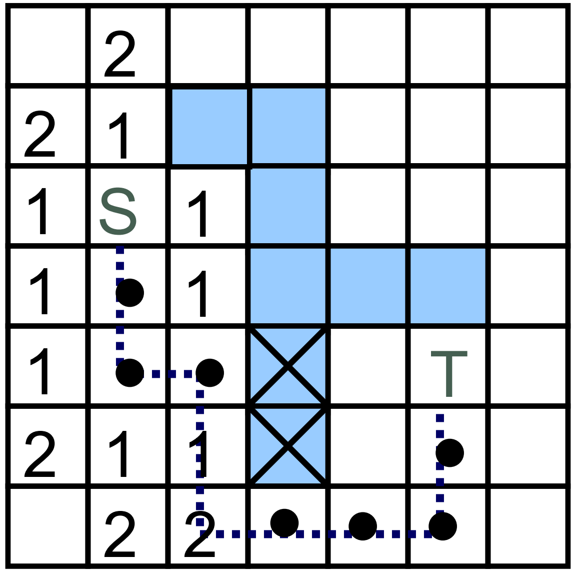
Hadlock's Algorithm

- Label vertices with detour numbers.
- Vertices with smaller detour number are expanded first.
- Therefore, favor paths without detour.

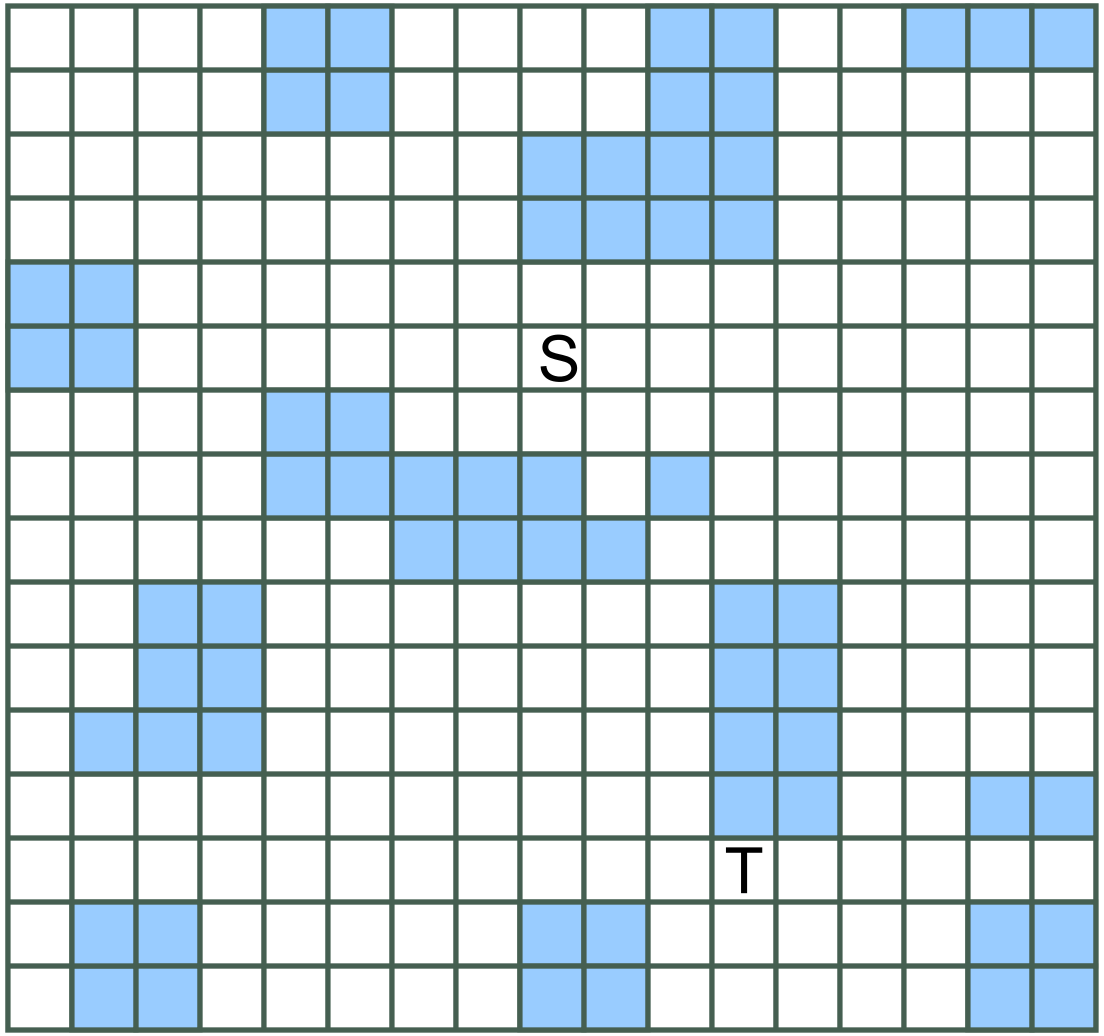


Soukup's Algorithm to Reduce Run Time

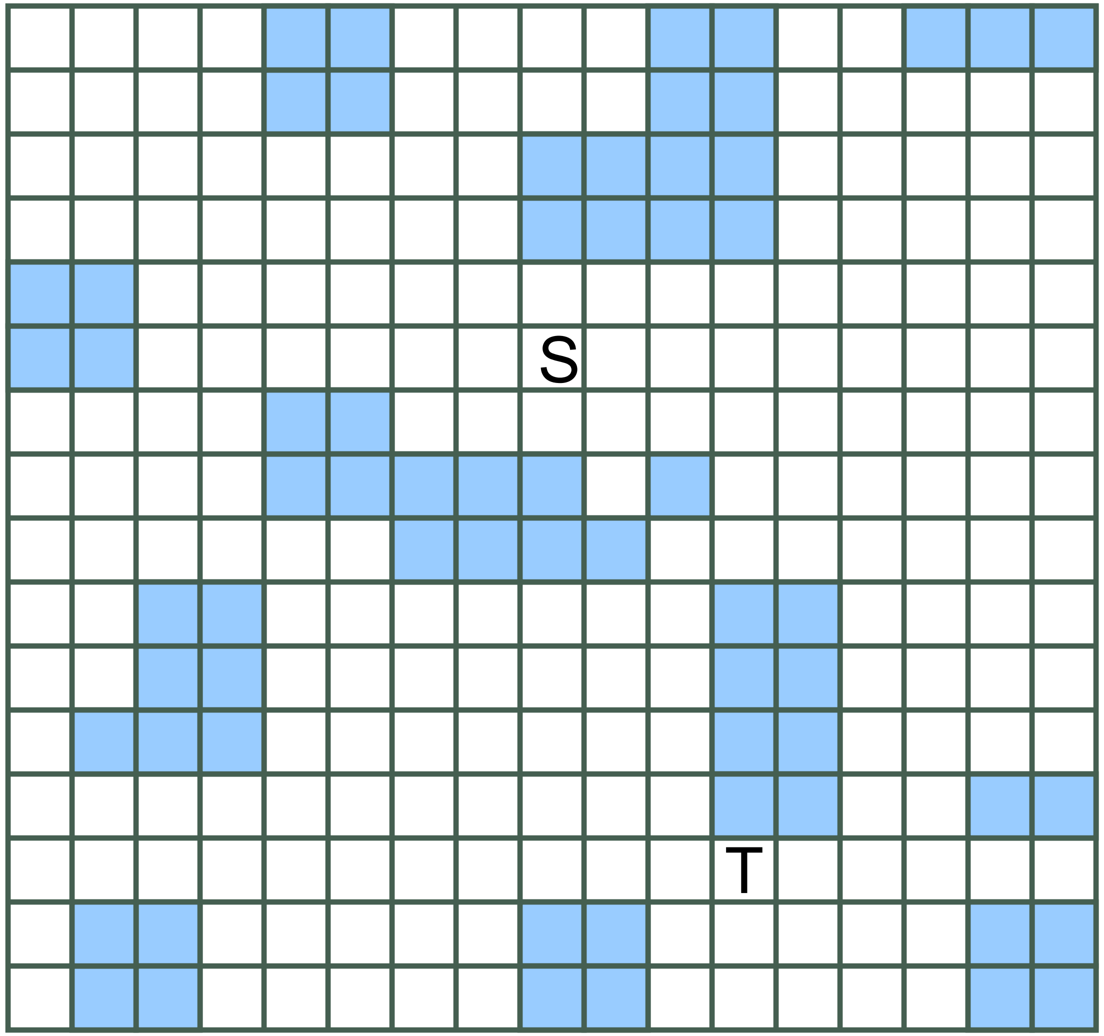
- Basic idea
- Soukup's Algorithm: BFS+DFS
  - Explore in the direction towards the target without changing direction. (DFS)
  - If obstacle is hit, search around the obstacle. (BFS)
- May get Sub-Optimal solution.



How many grids visited using Hadlock’s?



How many grids visited using Soukup's?



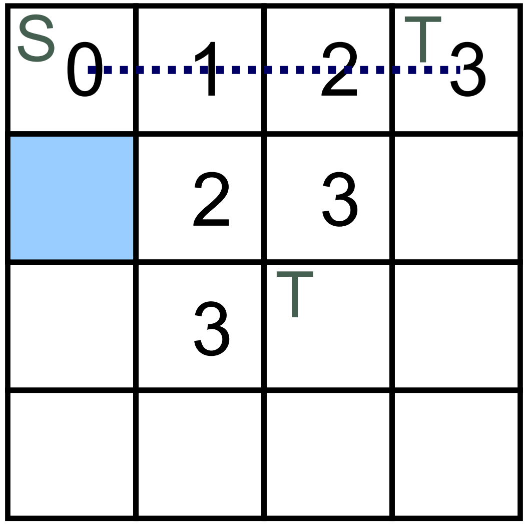
Multi-Pin Nets

- For a k-pin net, connect the k pins using a rectilinear Steiner tree with the shortest wire length on the maze.
- This problem is NP-Complete.
- Just want to find some good heuristics.

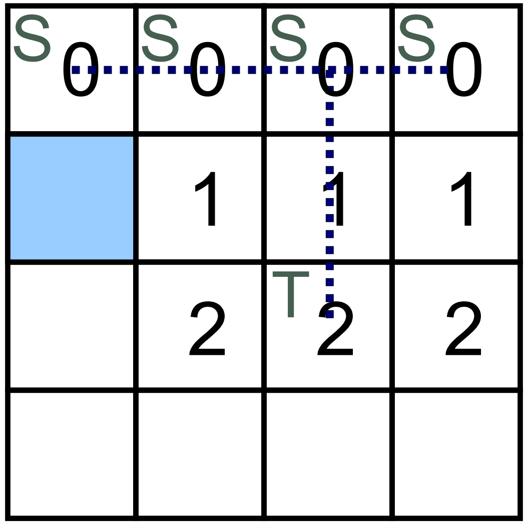
- This problem can be solved by extending the Lee's algorithm:
  - Connect one pin at a time, or
  - Search for several targets simultaneously, or
  - Propagate wave fronts from several different sources simultaneously.

Extension to Multi-Pin Nets

1st Iteration

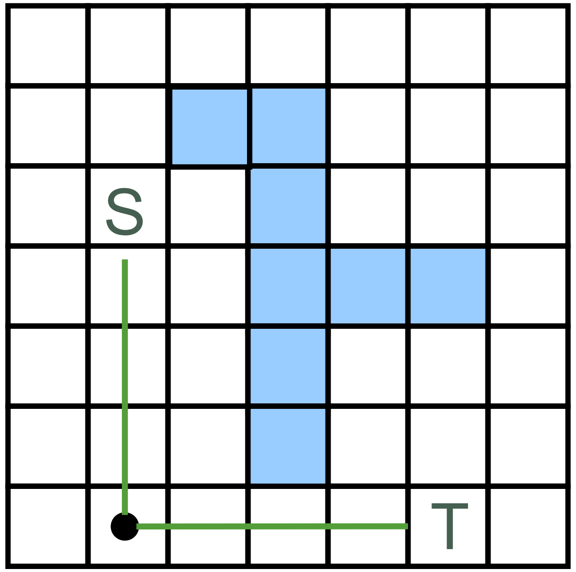


2nd Iteration

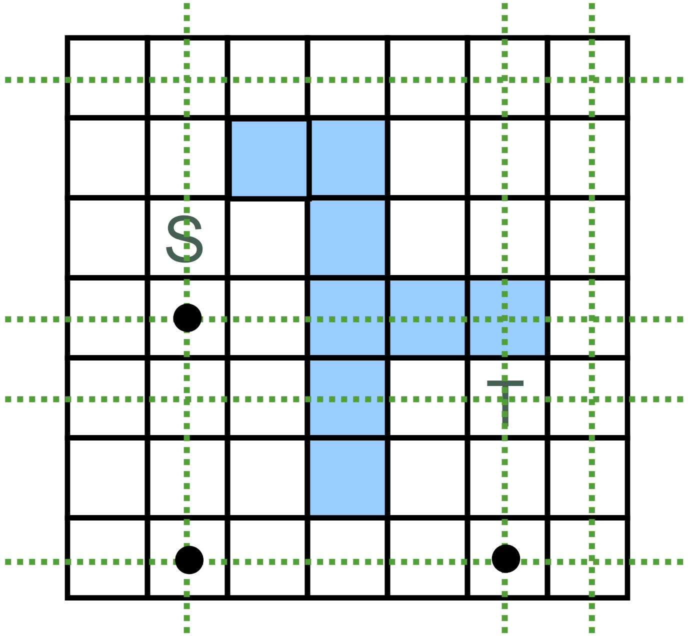


Speedup Maze Routing

- Pattern routing
  - Most nets are simple, e.g., L-shape
  - Can connect >80% nets
  - [NCTUgr, TCAD2013]

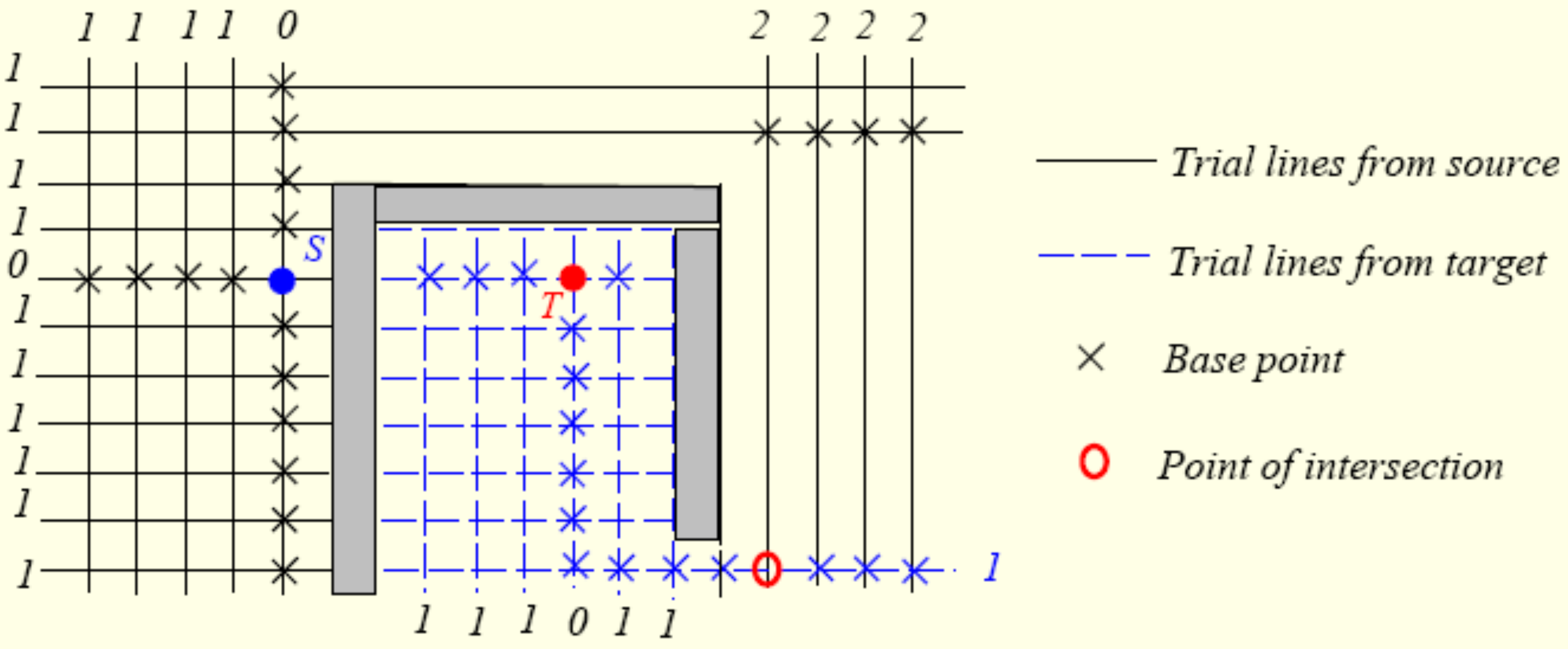


- Coarsening search steps
  - Only check grids that can make a turn
  - [Dr.CU, ICCAD2019]



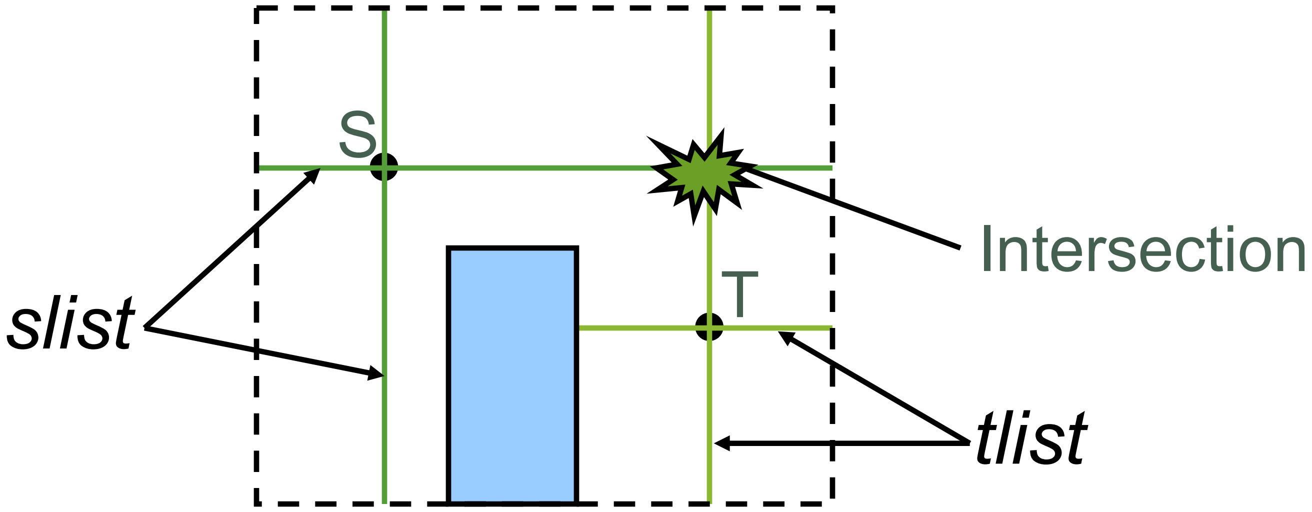
Mikami & Tabuchi's Algorithm

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



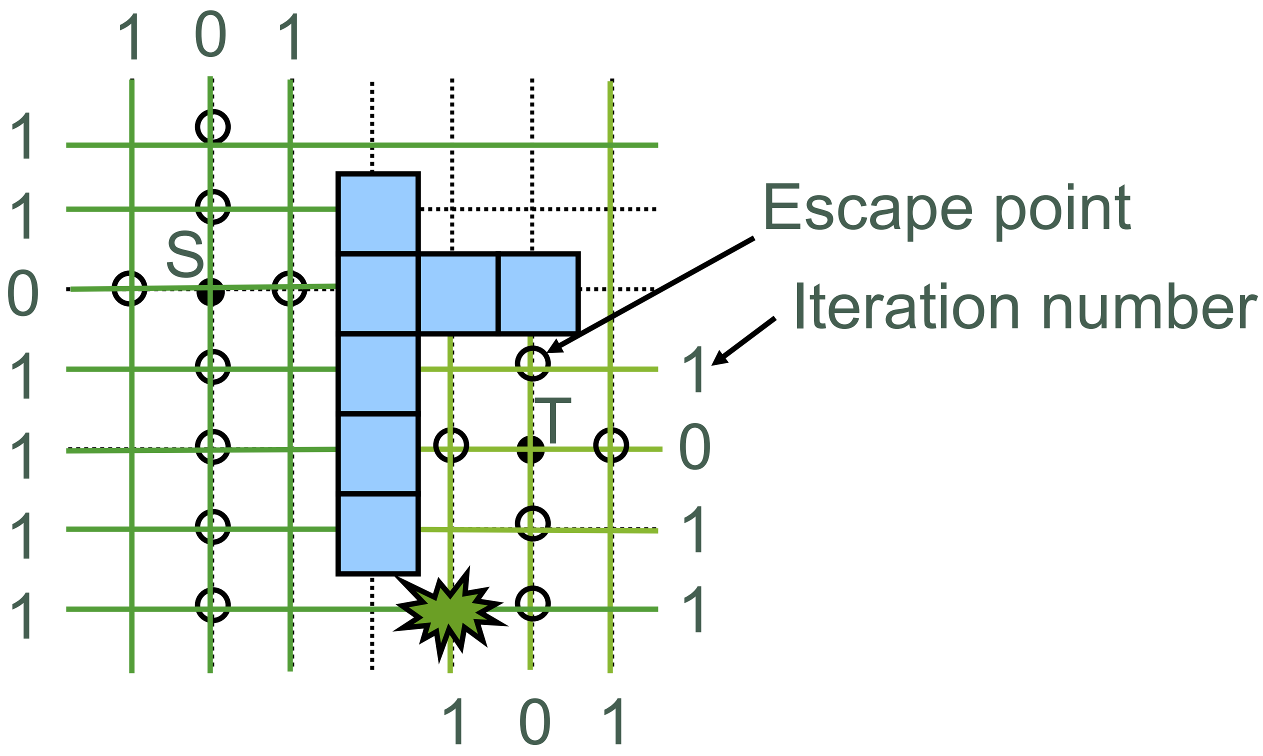
Line Probing

- Keep two lists of line segments, *slist* and *tlist*, for the source and the target respectively.
- If a line segment from *slist* intersects with one from *tlist*, a route is found; else, new line segments are generated from the escape points.



Line Probing

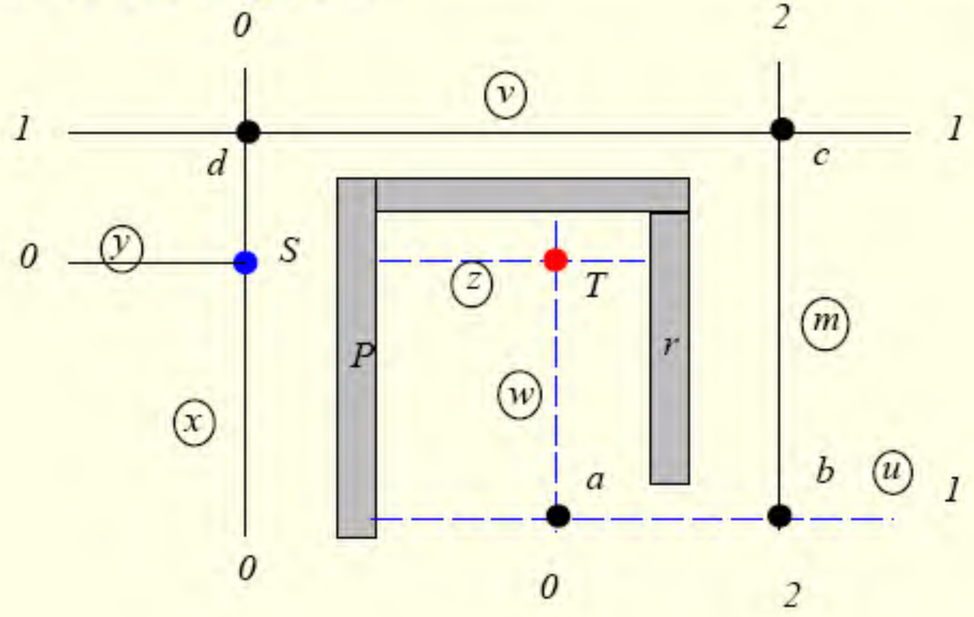
- We can use all the grid vertices on the line segments as escape points:



- Always find a path but may not be optimal.

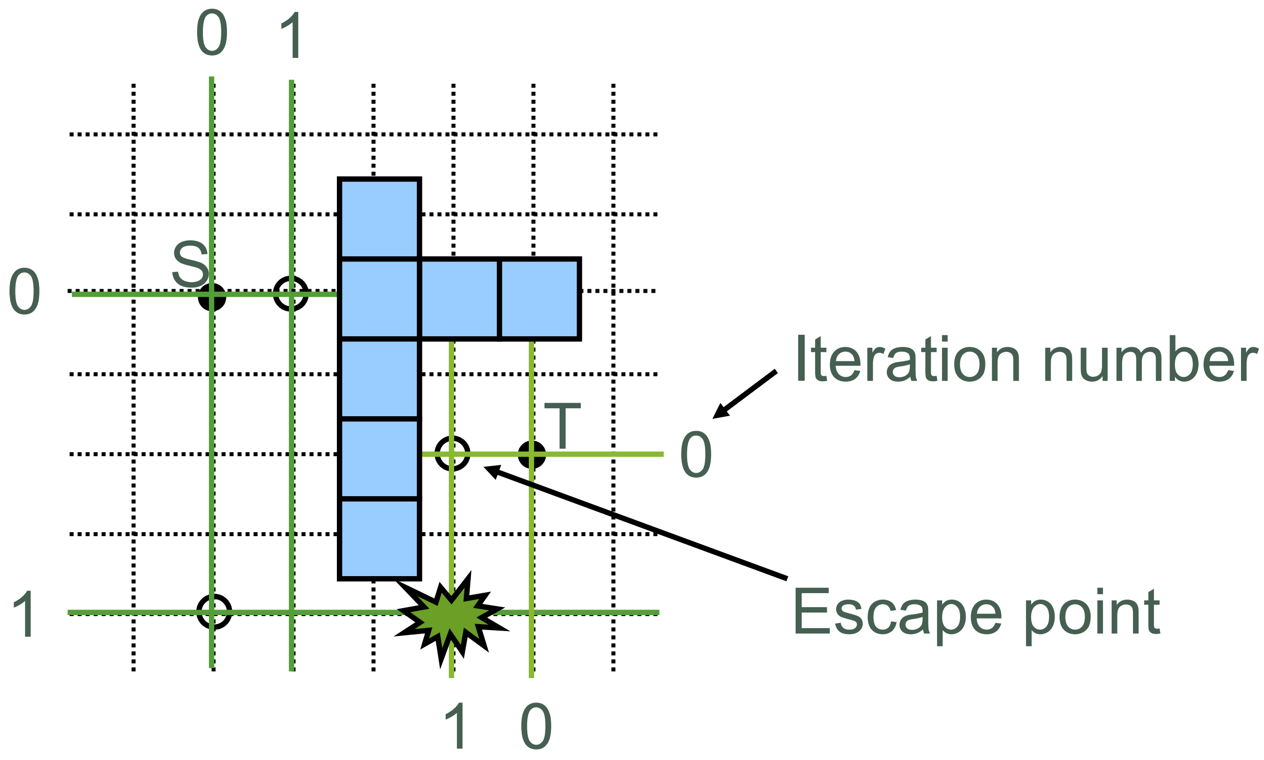
Hightower's Algorithm

- 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.



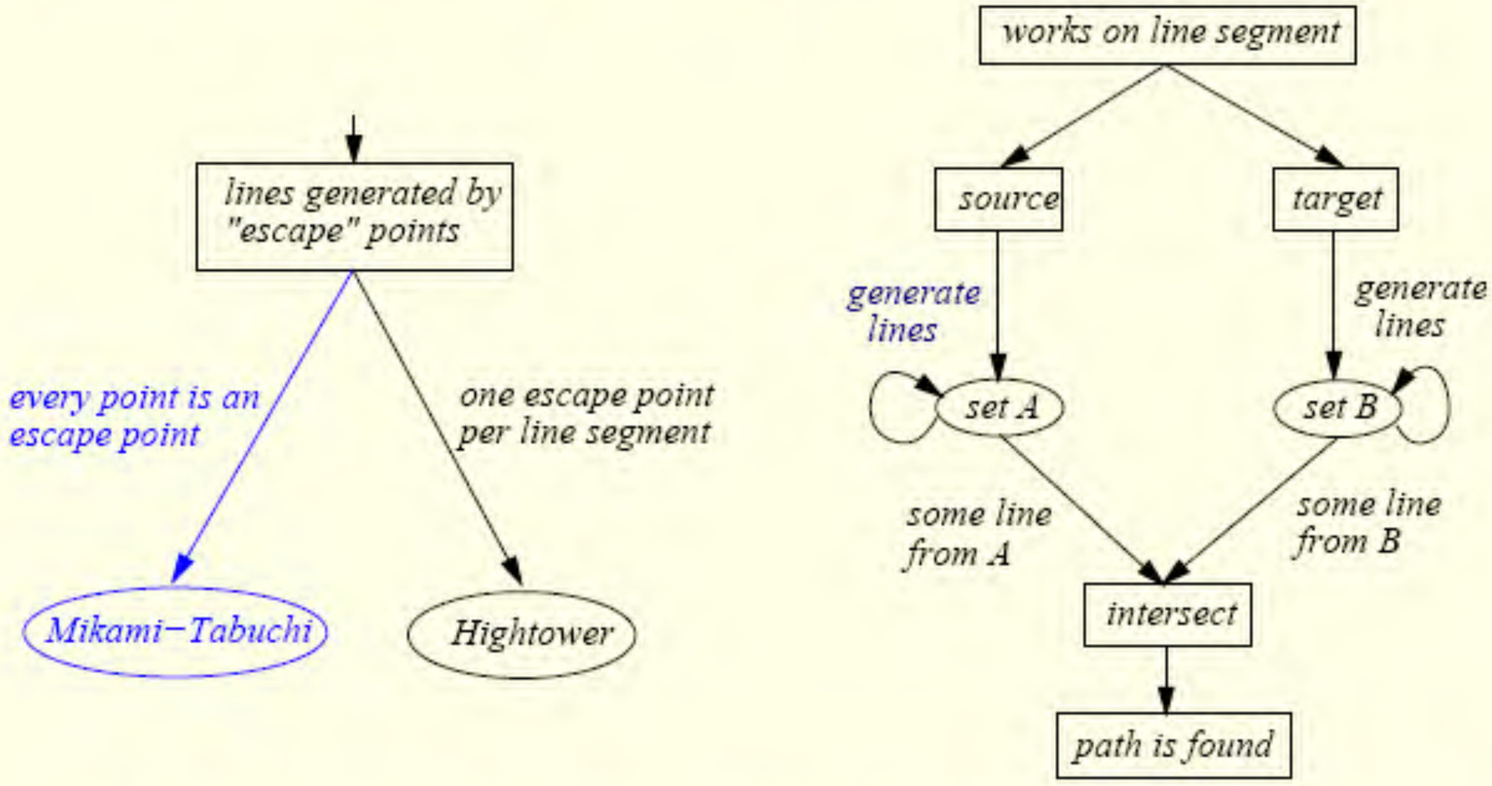
Line Probing

- We can pick just one escape point from each line segment.



- May fail to find a path even if one exists.

Features of Line-Search Algorithms

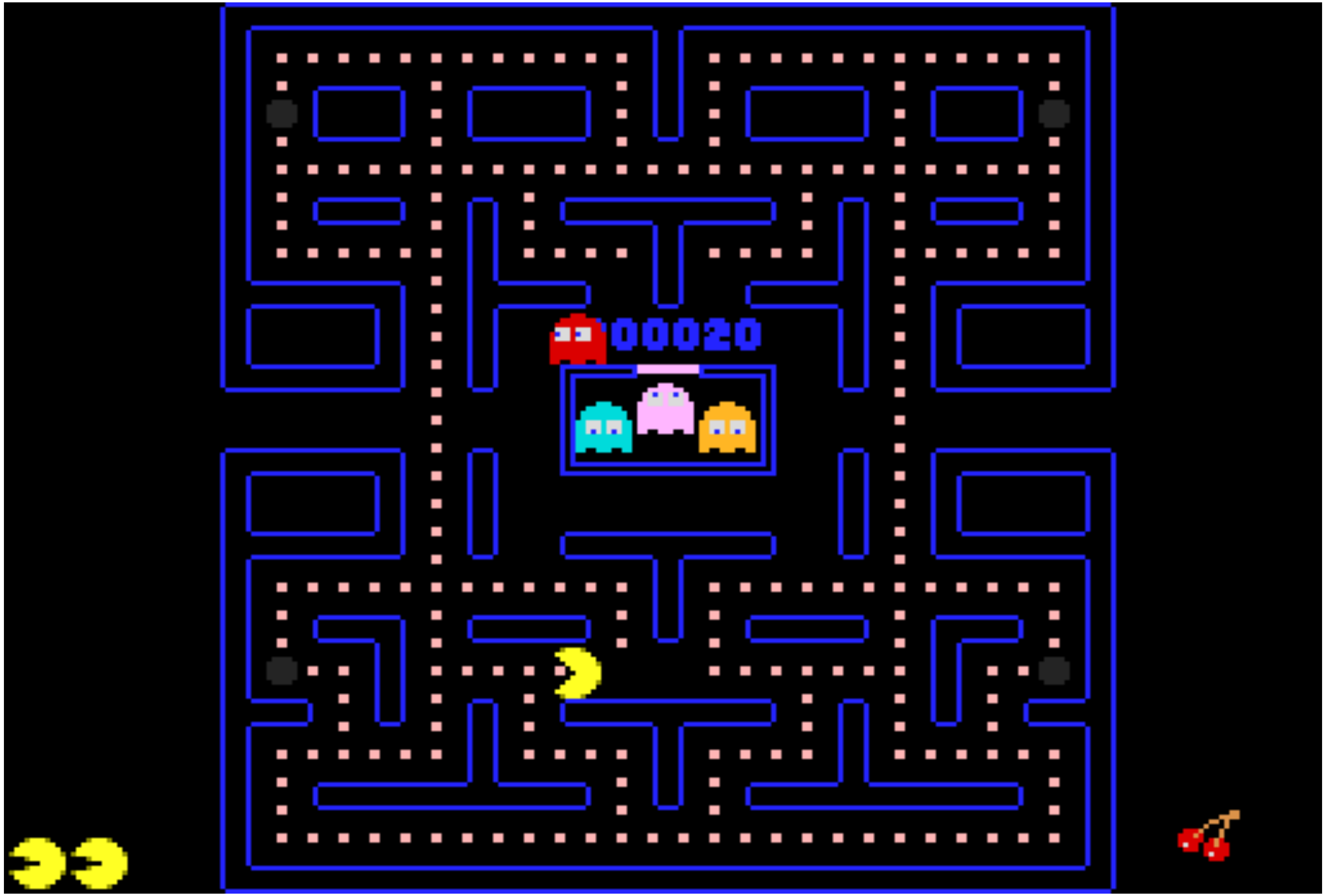


- Time and space complexities: $O(L)$, where L is the # of line segments generated.

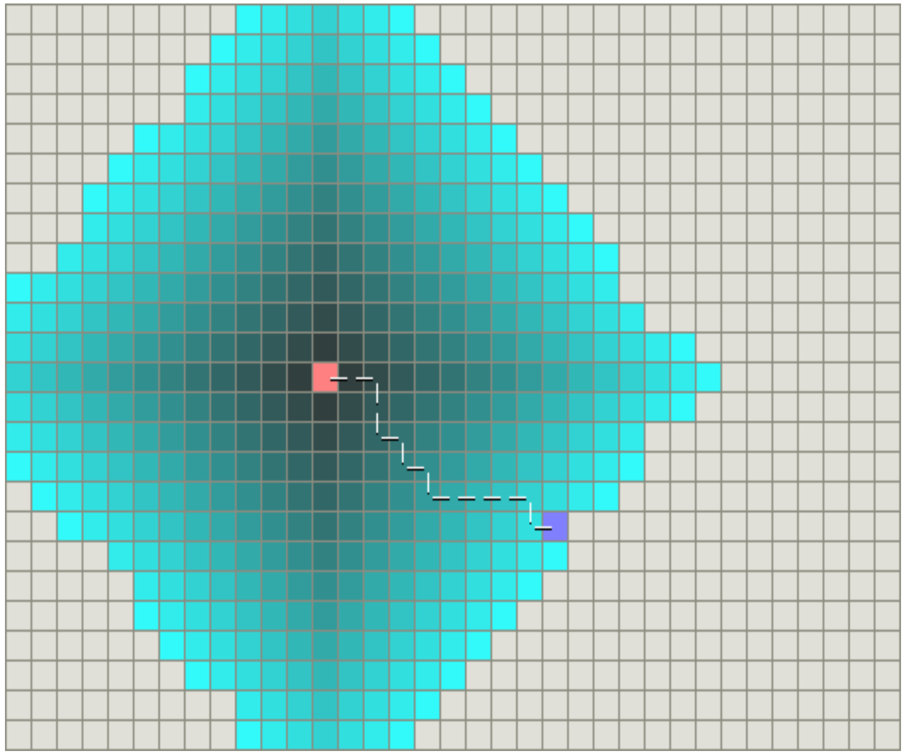
Comparison of Algorithms

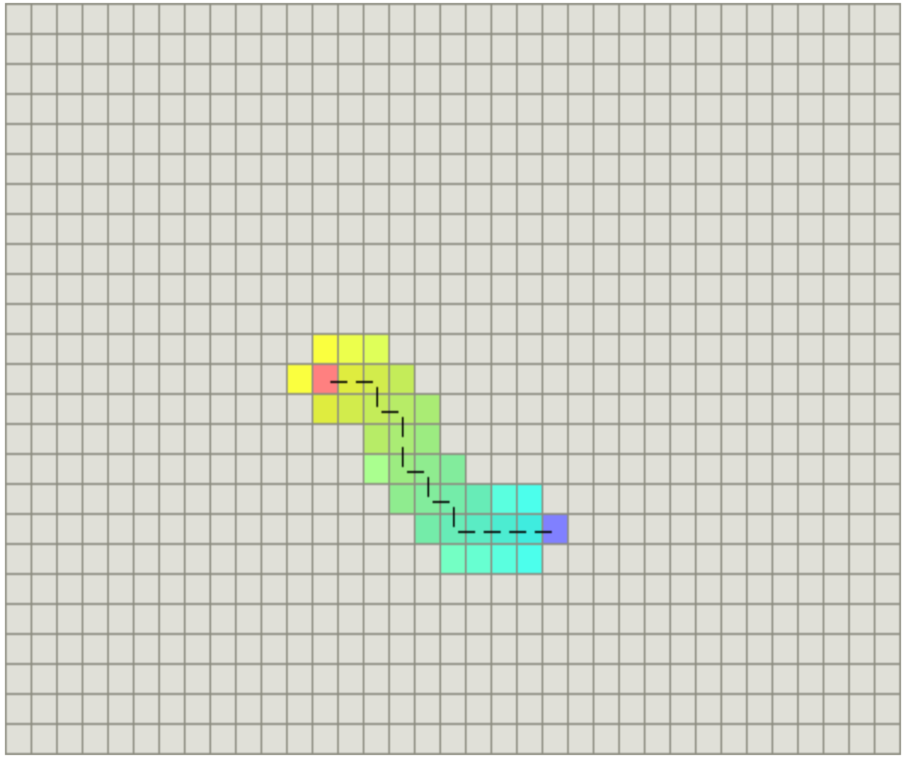
	Maze routing			Line search	
	Lee	Soukup	Hadlock	Mikami	Hightower
Time	$O(MN)$	$O(MN)$	$O(MN)$	$O(L)$	$O(L)$
Space	$O(MN)$	$O(MN)$	$O(MN)$	$O(L)$	$O(L)$
Finds path if one exists?	yes	yes	yes	yes	no
Is the path shortest?	yes	no	yes	no	no
Works on grids or lines?	grid	grid	grid	line	line

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

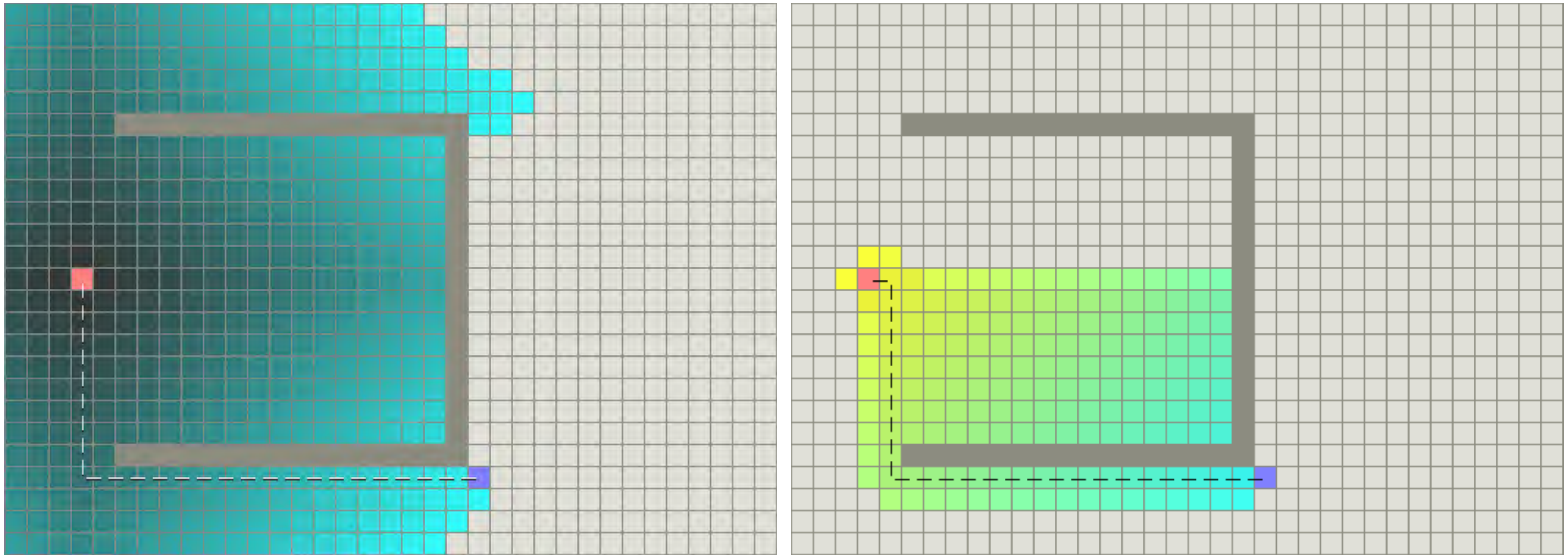

00020

Maze vs A* routing (I)





Maze vs A* routing (II)



Shortest Path Based Algorithms

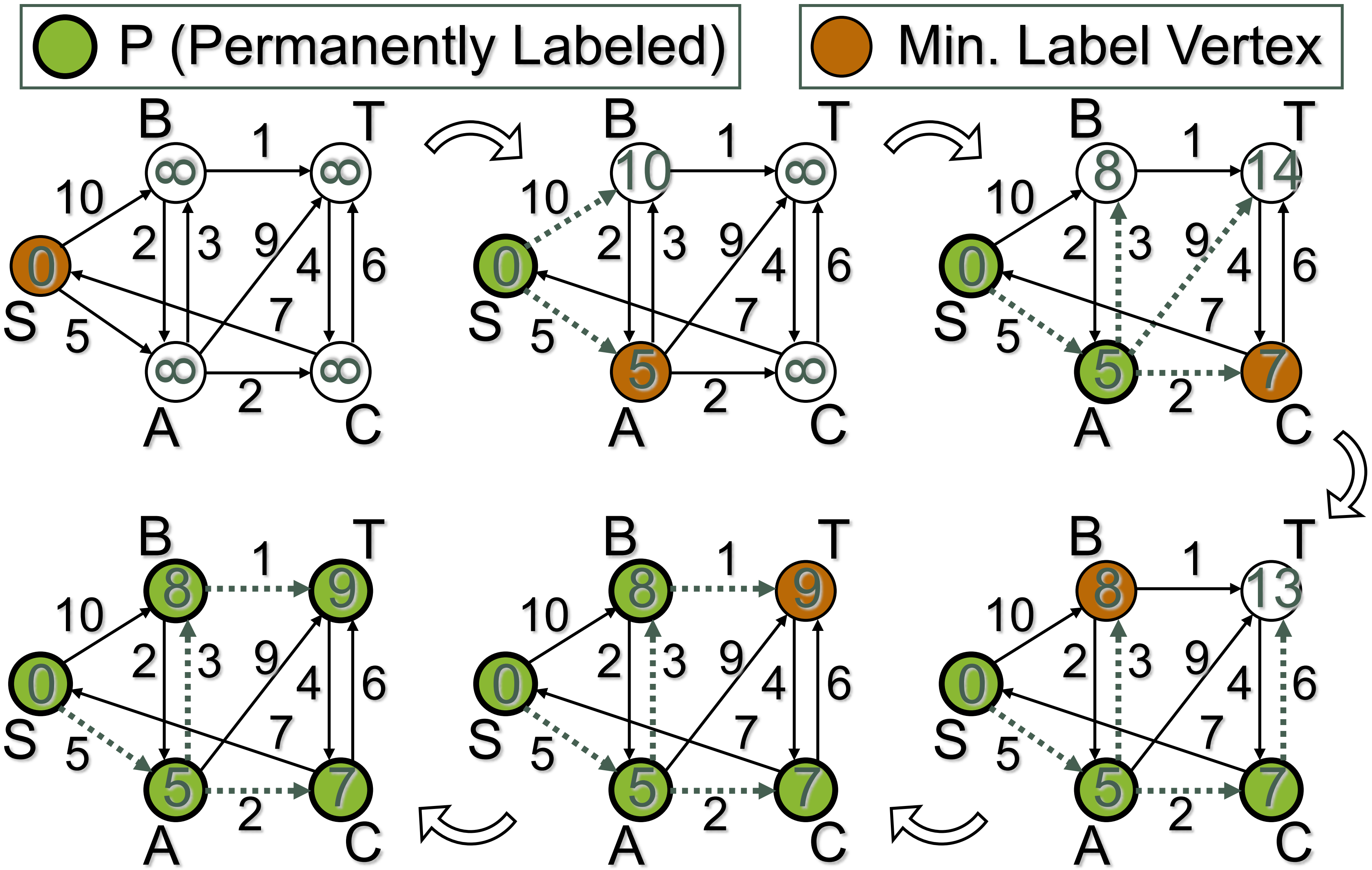
- For 2-terminal nets only.
- Use Dijkstra's algorithm to find the shortest path between the source *s* and the sink *t* of a net.
- Different from Maze Routing:
  - The graph need not be a rectangular grid.
  - The edges need not be of unit length.

Dijkstra’s Shortest Path Algorithm

- Label of vertices = Shortest distance from S.
- Let P be the set of permanently labeled vertices.

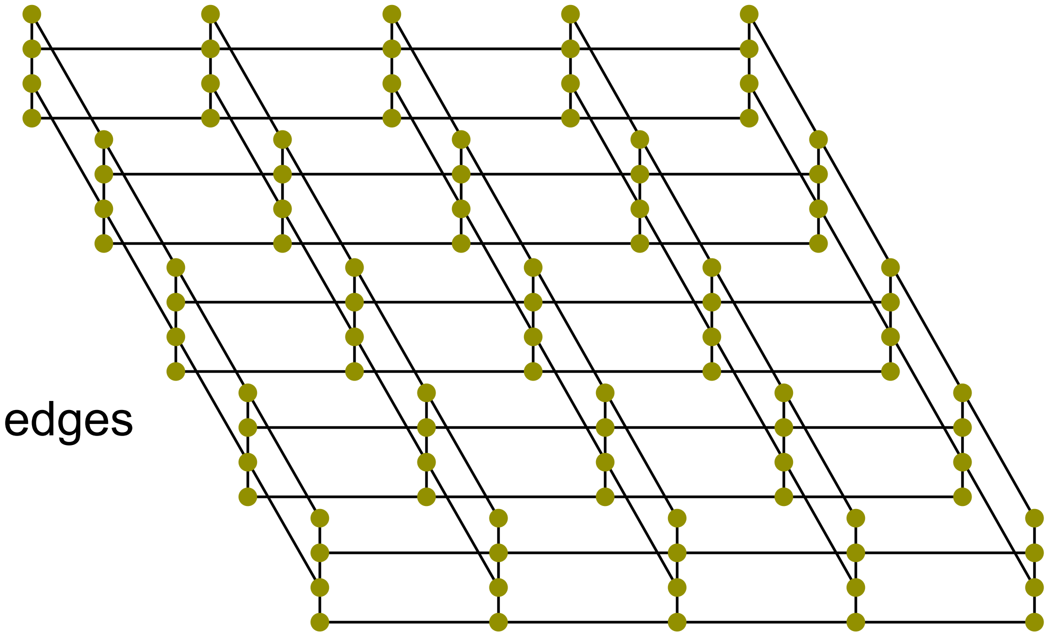
- Initially,
  - P = Empty Set.
  - Label of S = 0, Label of all other vertices = infinity.
- While (T is not in P) do
  - Pick the vertex v with the min. label among all vertices not in P.
  - Add v to P.
  - Update the label for all neighbours of v.

Dijkstra's Algorithm: Example



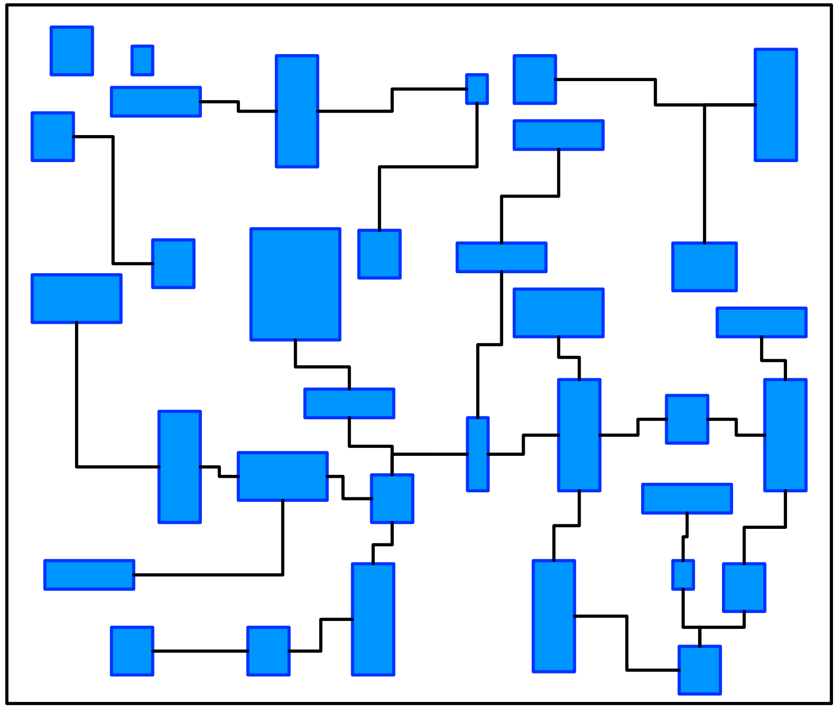
Typical Routing Flow – Divide and Conquer

Global routing

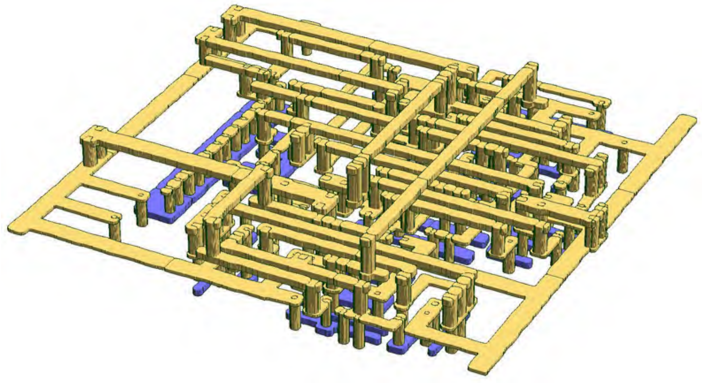


Coarse-grain routing

Typical Routing Flow – Divide and Conquer



Detailed routing



[curtesy samyzaf]

Fine-grain routing, larger solution space
Need to handle detailed design rules

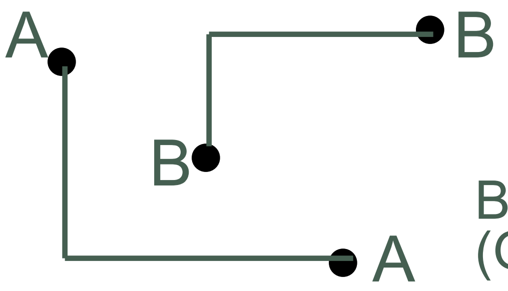
Sequential Routing

- Algorithm:
- 1. Graph modeling of the routing regions
- 2. For each net k:
  - 2.1 Find a route r for net k on the graph.
  - 2.2 For each edge e in r:
    - 2.2.1 capacity(e) = capacity(e) - 1
    - 2.2.2 if capacity(e) < 0 then cost(e) = $\alpha \times$ cost(e)

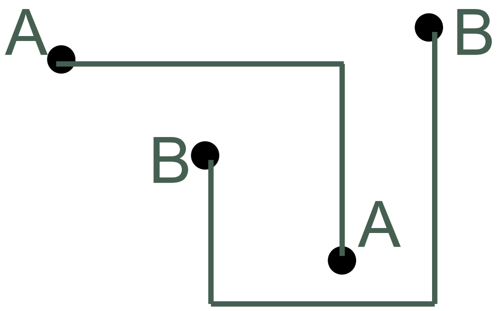
We can use different methods to do this.

Net Ordering

- In sequential approach, we need some net ordering.
- A bad net ordering will increase the total wire length, and may even prevent completion of routing for some circuits which are indeed routable.



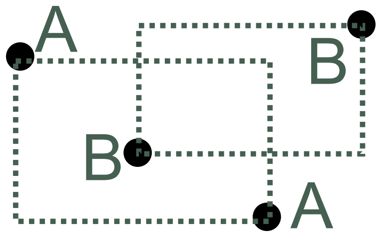
B first
(Good order)



A first
(Bad order)

Criteria for Net Ordering

- Criticality of net - critical nets first.
- Estimated wire length - short nets first since they are less flexible.
- Consider bounding rectangles (BR):

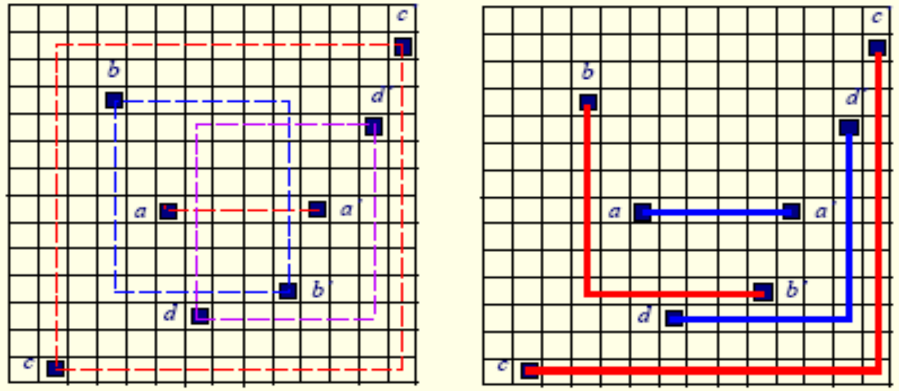


B is in A's BR

Which one should be routed first and why? (Note that this rule of thumb is not always applicable.)

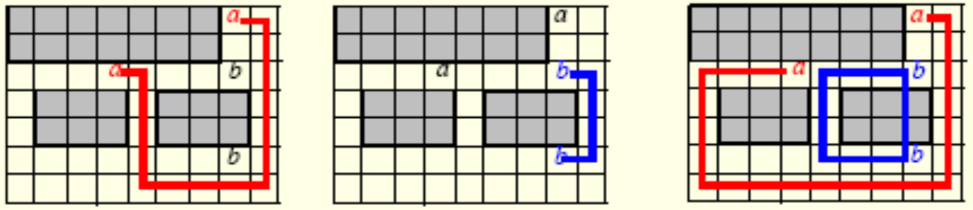
Net Ordering (cont'd)

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



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

- A mutually intervening case:



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

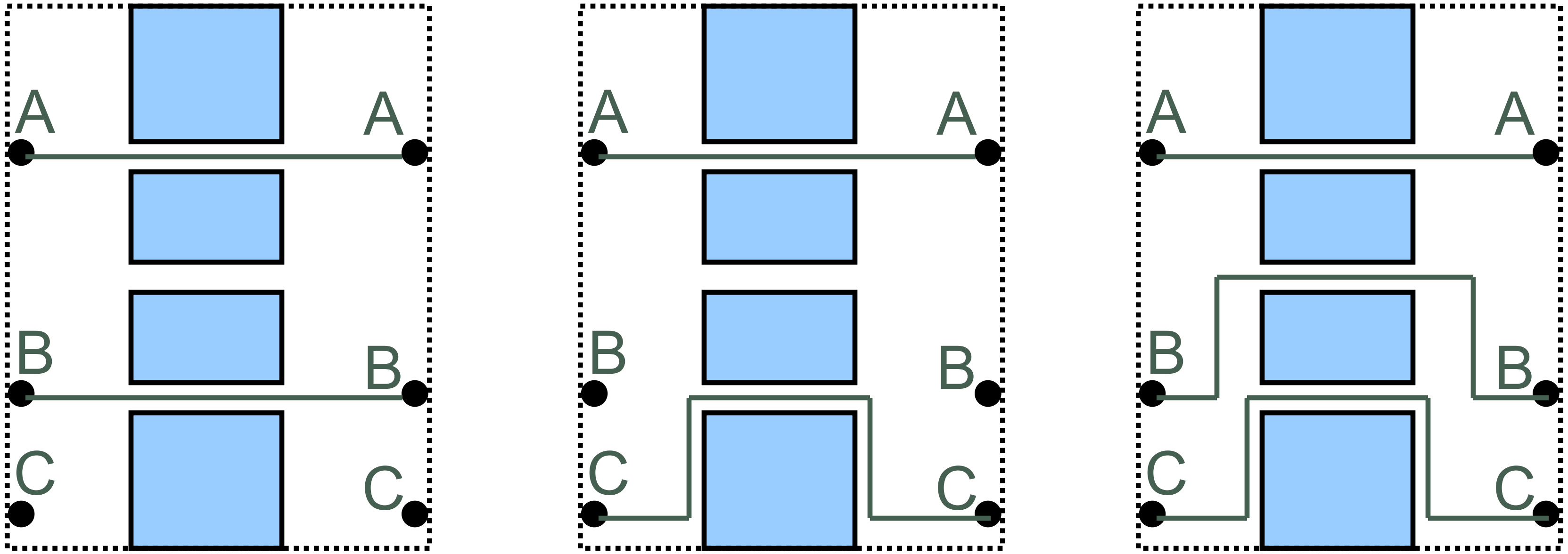
Rip-up and Re-route Scheme

- It is impossible to get the optimal net ordering.
- If some nets are failed to be routed, the rip-up and re-route technique can be applied:

Cannot route C

So rip-up B
and route C first.

Finally route B.



Recent Negotiation Congestion based Routers

- Liu, Wen-Hao, et al. "NCTU-GR 2.0: Multithreaded collision-aware global routing with bounded-length maze routing." *IEEE Transactions on computer-aided design of integrated circuits and systems* 32.5 (2013): 709-722.
- Li, Haocheng, et al. "Dr. CU 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction." *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019.
- Murray, Kevin E., et al. "Vtr 8: High-performance cad and customizable fpga architecture modelling." *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 13.2 (2020): 1-55.

Concurrent Approach

- Consider all the nets simultaneously.
- Formulate as an integer program.
- Given

Nets	*Set of possible routing trees*
net 1	$T_{11}, T_{12}, \ldots\ldots, T_{1k_1}$
⋮	⋮
net n	$T_{n1}, T_{n2}, \ldots, T_{nk_n}$

L_{ij} = Total wire length of T_{ij}

C_e = Capacity of edge e

- Determine variable x_{ij} s.t. $x_{ij} = 1$ if T_{ij} is used
  $x_{ij} = 0$ otherwise.

Integer Programming

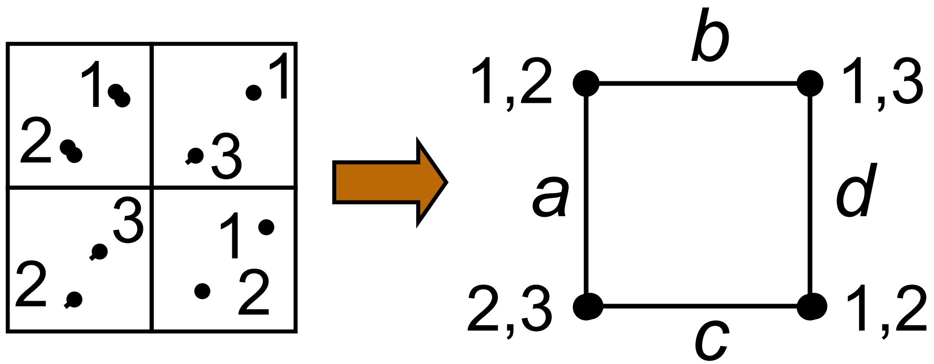
$$\text{Min.} \sum_{i=1}^{n} \sum_{j=1}^{k_i} L_{ij} \times x_{ij}$$

$$\text{s.t.} \quad \sum_{j=1}^{k_i} x_{ij} = 1 \quad \text{for all } i = 1, \ldots, n$$

$$\sum_{i,j \text{ s.t. } e \in T_{ij}} x_{ij} \le C_e \quad \text{for all edge } e$$

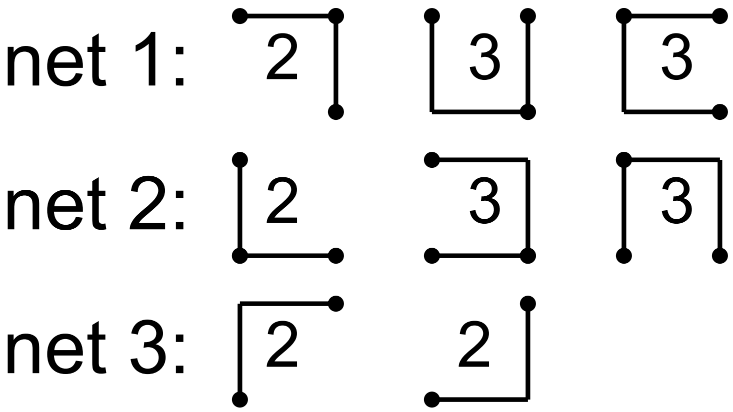
$$x_{ij} = 0 \text{ or } 1 \quad \forall i, j$$

Example

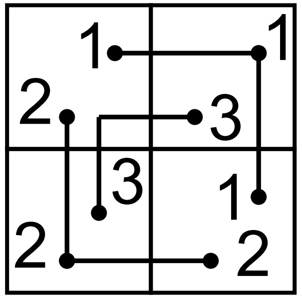


$C_a = C_b = C_c = C_d = 2$

Possible trees:



Solution



$$\text{Min. } 2x_{11} + 3x_{12} + 3x_{13} + 2x_{21} + 3x_{22} + 3x_{23} + 2x_{31} + 2x_{32}$$

$$\text{s.t.} \begin{cases} x_{11} + x_{12} + x_{13} = 1; \\ x_{21} + x_{22} + x_{23} = 1; \\ x_{31} + x_{32} = 1; \\ x_{ij} = 0 \text{ or } 1 \quad \forall i, j; \end{cases}$$

What are the constraints for edge capacity?

$$\boxed{x_{12} + x_{13} + x_{21} + x_{23} + x_{31} < C_a}$$

Integer Programming Approach

- Standard techniques to solve IP.
- No net ordering. Give global optimum.
- Can be extremely slow, especially for large problems.
- To make it faster, a fewer choices of routing trees for each net can be used. May make the problem infeasible or give a bad solution.
- Determining a good set of choices of routing trees is a hard problem by itself.

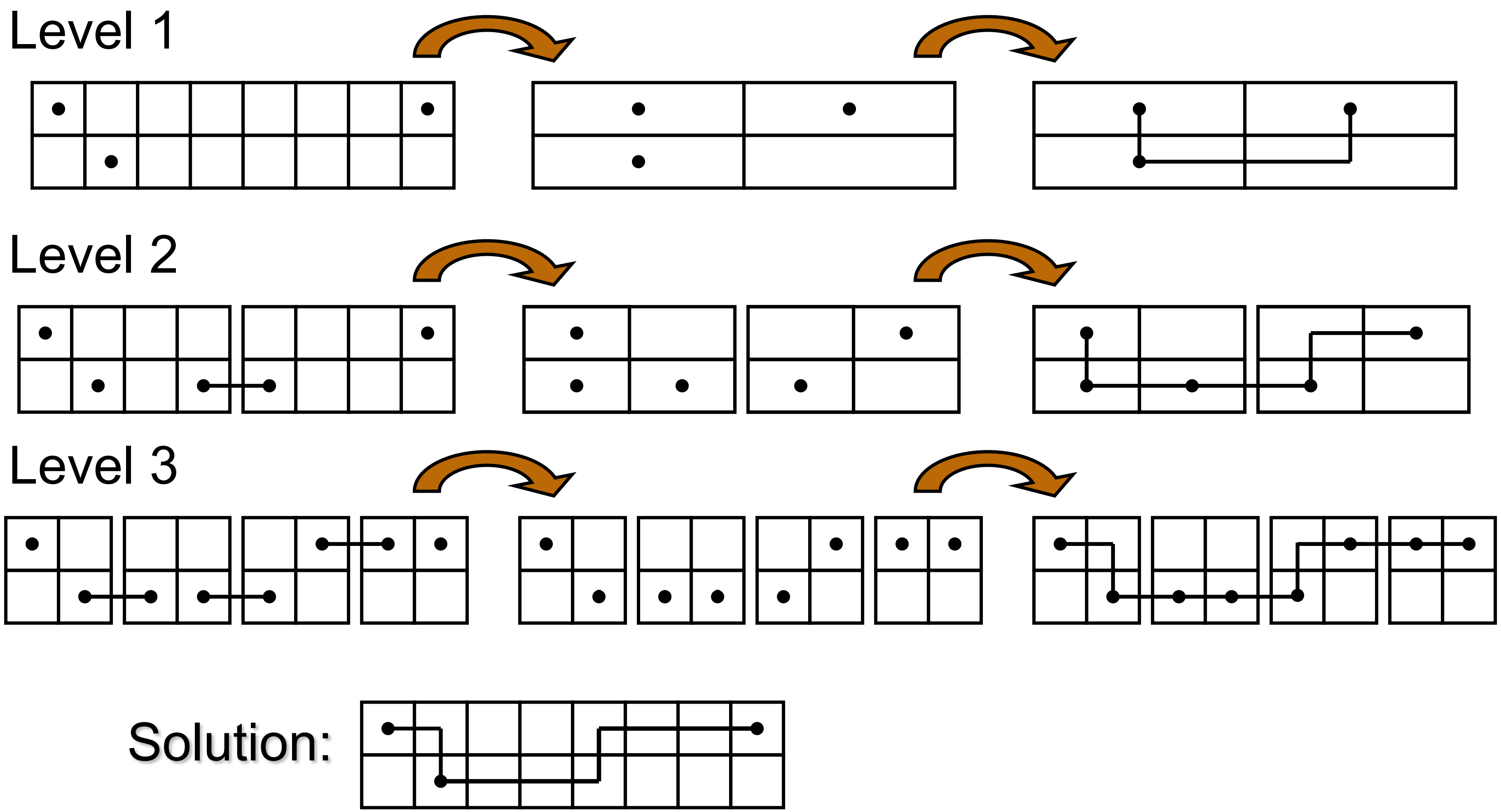
Hierarchical Approach to Speed Up IP

- Large Integer Programs are difficult to solve.
- Hierarchical Approach reduces global routing to routing problems on a 2x2 grid.
- Decompose recursively in a top-down fashion.
- Those 2x2 routing problems can be solved optimally by integer programming formulation.

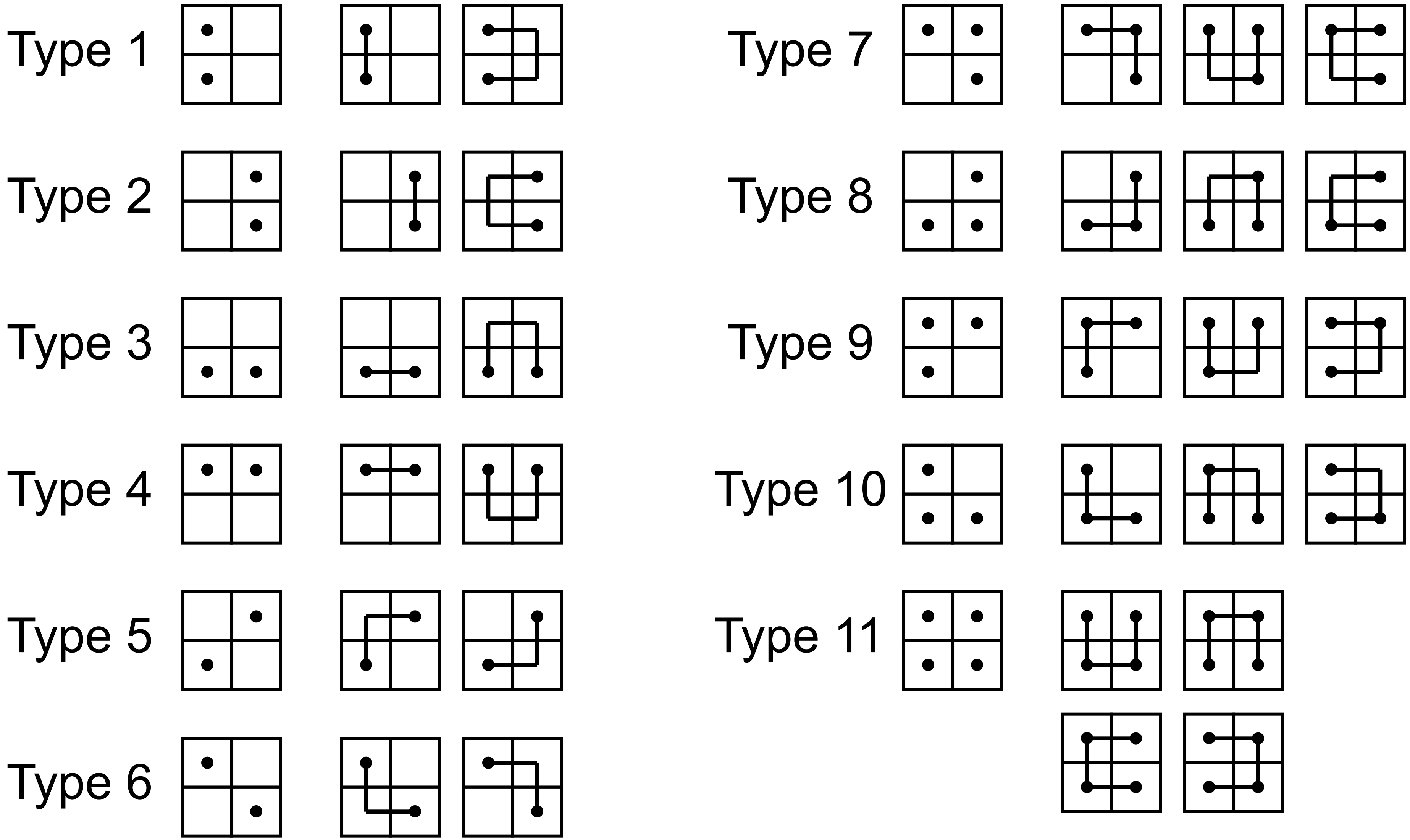
Burstein, Michael, and Richard Pelavin. "Hierarchical wire routing." TCAD 1983.

Example

- Solving a 2xn routing problem hierarchically.



Types of 2x2 Routing Problems



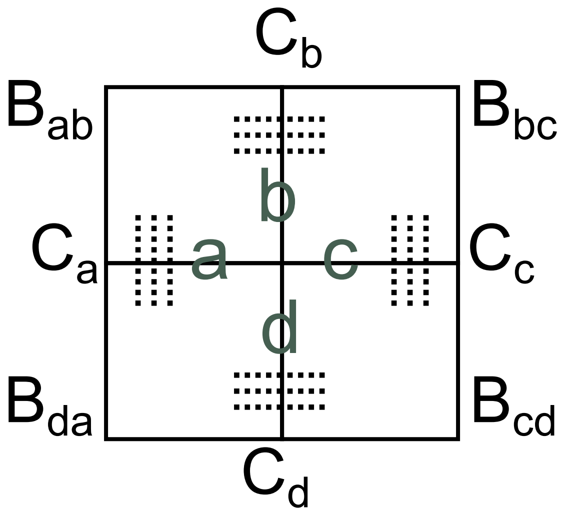
Objective Function of 2x2 Routing

- Possible Routing Trees:
- T11, T12, T21, T22,....., T11,1,..., T11,4
- # of nets of each type: n1, ..., n11
- Determine xij: # of type-i nets using Tij for routing.
- yi: # of type-i nets that fails to route.

$$y_i + \sum_j x_{ij} = n_i \quad i = 1, \ldots, 11$$

Want to minimize $\sum_{i=1}^{11} y_i$.

Constraints of 2x2 Routing



Constraints on Edge Capacity:

$$\sum_{i,j \text{ s.t. } a \in T_{ij}} x_{ij} \le C_a$$

$$\sum_{i,j \text{ s.t. } b \in T_{ij}} x_{ij} \le C_b$$

$$\sum_{i,j \text{ s.t. } c \in T_{ij}} x_{ij} \le C_c$$

$$\sum_{i,j \text{ s.t. } d \in T_{ij}} x_{ij} \le C_d$$

Constraints on # of Bends in a Region:

$$\sum_{i,j \text{ s.t. } T_{ij} \text{ has a bend in region } ab} x_{ij} \le B_{ab}$$

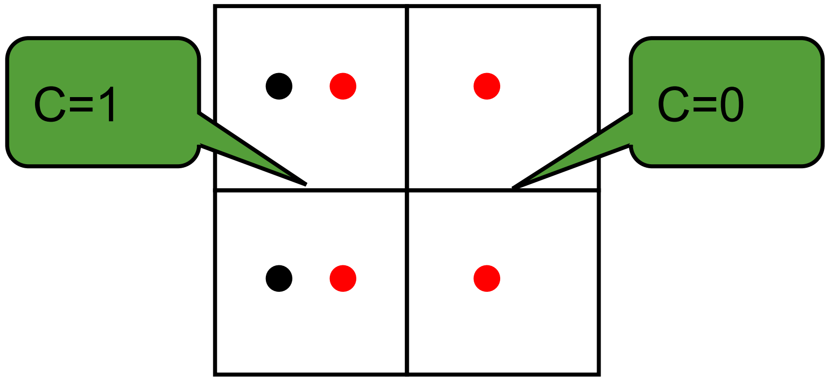
$$\sum_{i,j \text{ s.t. } T_{ij} \text{ has a bend in region } bc} x_{ij} \le B_{bc}$$

$$\sum_{i,j \text{ s.t. } T_{ij} \text{ has a bend in region } cd} x_{ij} \le B_{cd}$$

$$\sum_{i,j \text{ s.t. } T_{ij} \text{ has a bend in region } da} x_{ij} \le B_{da}$$

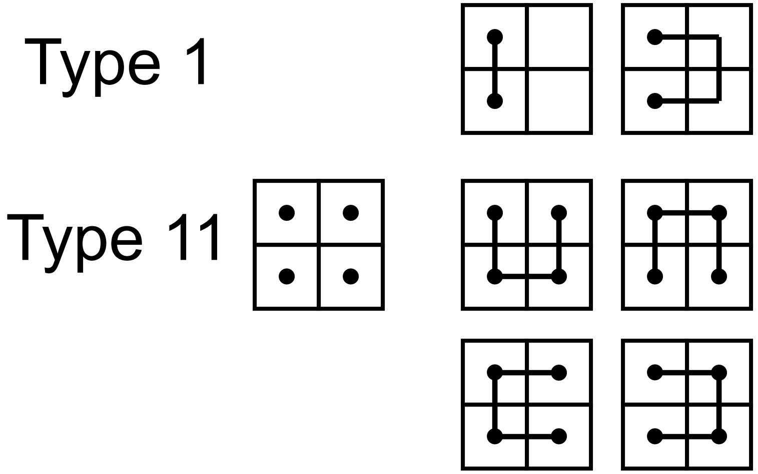
Pop Quiz

- If you two nets, one with 2 pins, the other with 4 pins with a zero capacity edge
  - What is going to be the result?



$$y_i + \sum_j x_{ij} = n_i \quad i = 1, \ldots, 11$$

Want to minimize $\sum_{i=1}^{11} y_i$.



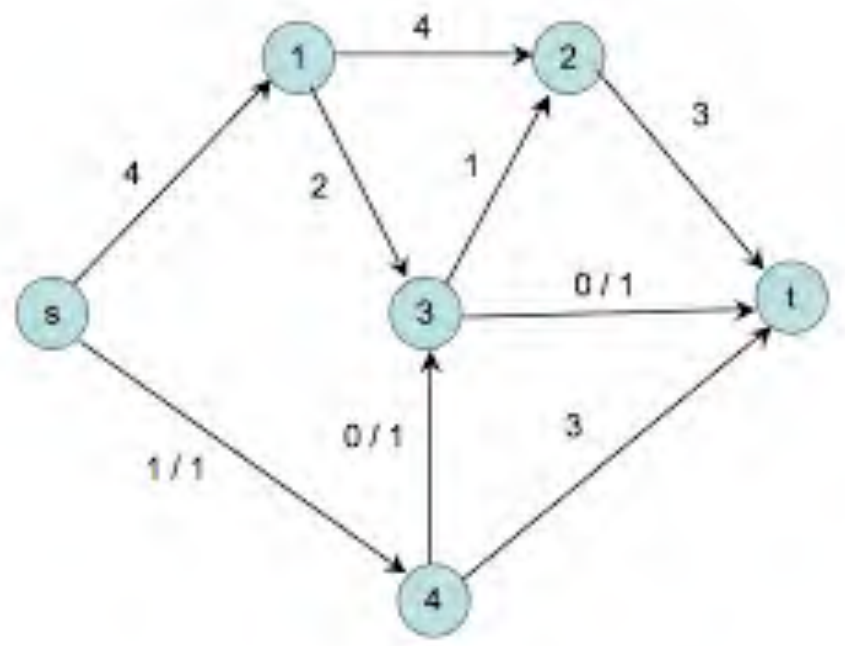
ILP Formulation of 2x2 Routing

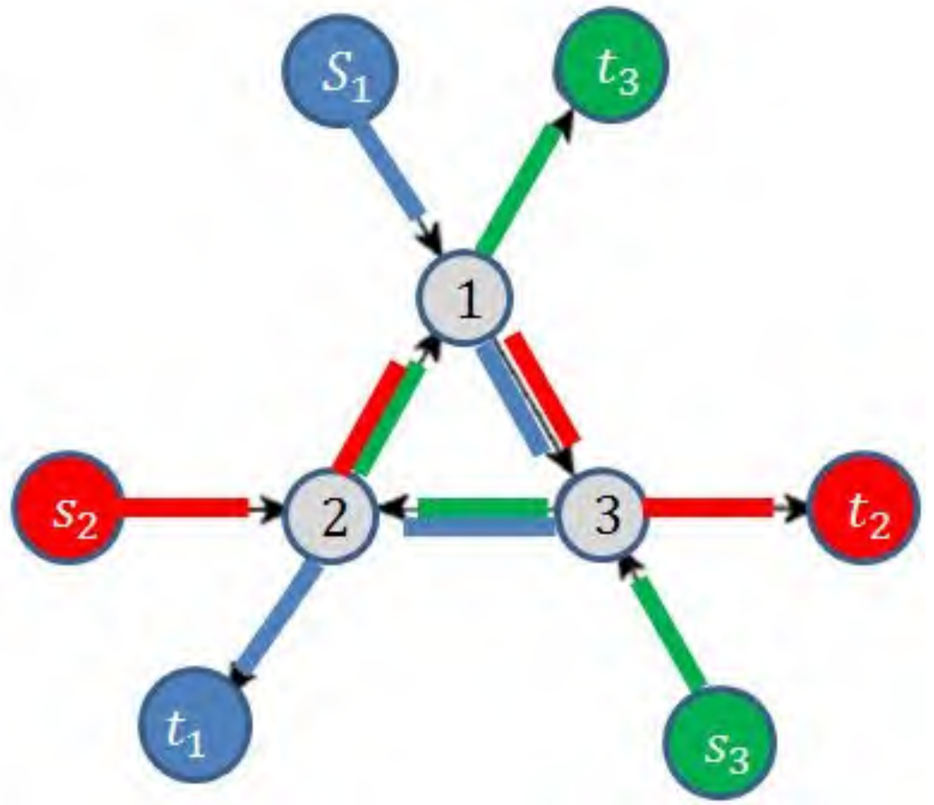
$$
\begin{aligned}
\text{Min.}\quad & \sum_{i=1}^{11} y_i \\
\text{s.t.}\quad & y_i + \sum_j x_{ij} = n_i \quad i = 1,\dots,11 \\
& x_{ij} \ge 0,\ y_i \ge 0 \quad \forall i, j \\
& \sum_{i,j \text{ s.t. } a\in T_{ij}} x_{ij} \le C_a \qquad \sum_{i,j \text{ s.t. } T_{ij} \text{ has a bend in region } ab} x_{ij} \le B_{ab} \\
& \sum_{i,j \text{ s.t. } b\in T_{ij}} x_{ij} \le C_b \qquad \sum_{i,j \text{ s.t. } T_{ij} \text{ has a bend in region } bc} x_{ij} \le B_{bc} \\
& \sum_{i,j \text{ s.t. } c\in T_{ij}} x_{ij} \le C_c \qquad \sum_{i,j \text{ s.t. } T_{ij} \text{ has a bend in region } cd} x_{ij} \le B_{cd} \\
& \sum_{i,j \text{ s.t. } d\in T_{ij}} x_{ij} \le C_d \qquad \sum_{i,j \text{ s.t. } T_{ij} \text{ has a bend in region } da} x_{ij} \le B_{da}
\end{aligned}
$$

- Only 39 variables (28 x_{ij} and 11 y_i) and 19 constraints (plus 38 non-negative constrains).
- Problems of this size are usually not too difficult to solve.

Multi-Commodity Flow

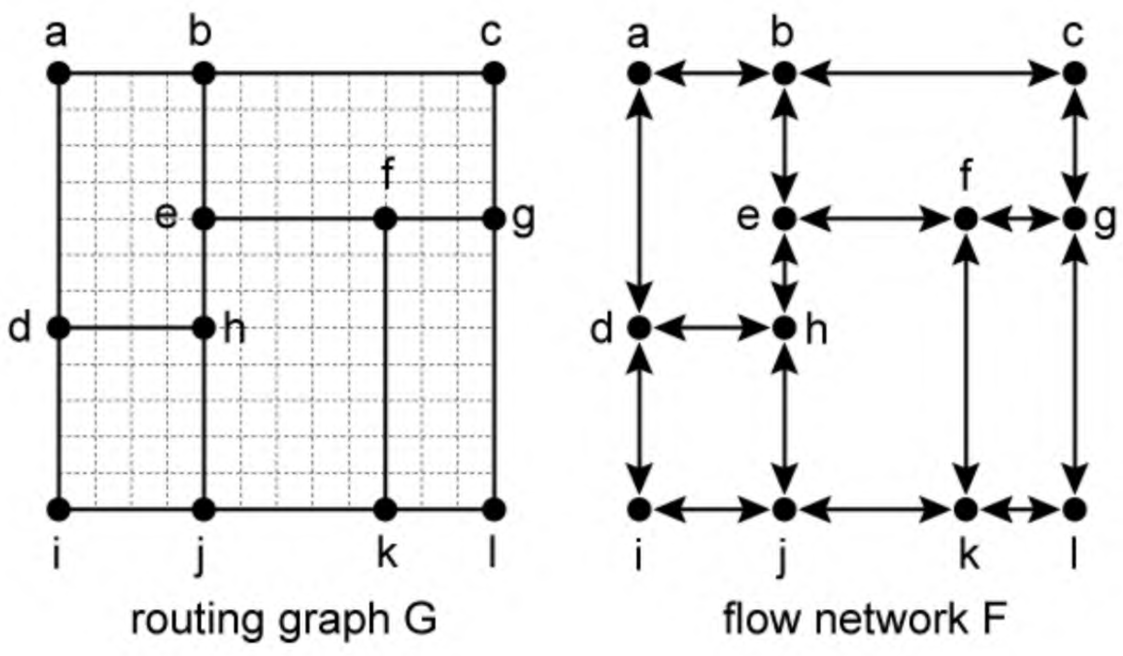
- Strongly NP-Complete for integer flows





Multi-Commodity Flow based Routing

- An example
  - Capacity of each edge in G is 2
  - Each edge in G becomes a pair of bi-directional arcs in F
  - $n_1 = \{a, l\}$
  - $n_2 = \{i, c\}$
  - $n_3 = \{d, f\}$
  - $n_4 = \{k, d\}$
  - $n_5 = \{g, h\}$
  - $n_6 = \{b, k\}$

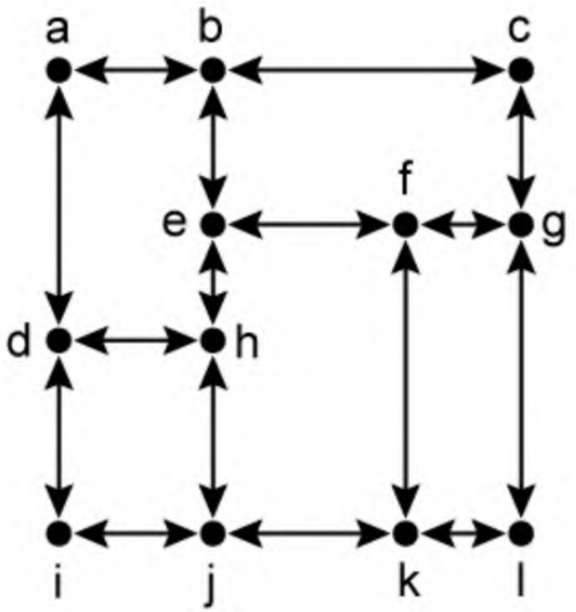


Sung-Kyu Lim, Practical Problems in VLSI Physical Design Automation (2008, Springer)
http://limsk.ece.gatech.edu/book/slides/pdf/multi-commodity.pdf

Flow Network

- Each arc has a cost based on its length
  - Let x_e^k denote a binary variable for arc e w.r.t. net k
  - x_e^k means net k uses arc e in its route
  - Total number of x-variables: $16 \times 2 \times 6 = 192$

arc	cost	arc	cost	arc	cost	arc	cost
(a, b)	4	(b, a)	4	(b, c)	8	(c, b)	8
(d, h)	4	(h, d)	4	(e, f)	5	(f, e)	5
(f, g)	3	(g, f)	3	(i, j)	4	(j, i)	4
(j, k)	5	(k, j)	5	(k, l)	3	(l, k)	3
(a, d)	7	(d, a)	7	(d, i)	5	(i, d)	5
(b, e)	4	(e, b)	4	(e, h)	3	(h, e)	3
(h, j)	5	(j, h)	5	(f, k)	8	(k, f)	8
(c, g)	4	(g, c)	4	(g, l)	8	(l, g)	8



ILP Objective Function

- Minimize

$$
\begin{aligned}
&4(x^1_{a,b} + \cdots + x^6_{a,b}) + 4(x^1_{b,a} + \cdots + x^6_{b,a}) + 8(x^1_{b,c} + \cdots + x^6_{b,c}) + \\
&8(x^1_{c,b} + \cdots + x^6_{c,b}) + 4(x^1_{d,h} + \cdots + x^6_{d,h}) + 4(x^1_{h,d} + \cdots + x^6_{h,d}) + \\
&5(x^1_{e,f} + \cdots + x^6_{e,f}) + 5(x^1_{f,e} + \cdots + x^6_{f,e}) + 3(x^1_{f,g} + \cdots + x^6_{f,g}) + \\
&3(x^1_{g,f} + \cdots + x^6_{g,f}) + 4(x^1_{i,j} + \cdots + x^6_{i,j}) + 4(x^1_{j,i} + \cdots + x^6_{j,i}) + \\
&5(x^1_{j,k} + \cdots + x^6_{j,k}) + 5(x^1_{k,j} + \cdots + x^6_{k,j}) + 3(x^1_{k,l} + \cdots + x^6_{k,l}) + \\
&3(x^1_{l,k} + \cdots + x^6_{l,k}) + 7(x^1_{a,d} + \cdots + x^6_{a,d}) + 7(x^1_{d,a} + \cdots + x^6_{d,a}) + \\
&5(x^1_{d,i} + \cdots + x^6_{d,i}) + 5(x^1_{i,d} + \cdots + x^6_{i,d}) + 4(x^1_{b,e} + \cdots + x^6_{b,e}) + \\
&4(x^1_{e,b} + \cdots + x^6_{e,b}) + 3(x^1_{e,h} + \cdots + x^6_{e,h}) + 3(x^1_{h,e} + \cdots + x^6_{h,e}) + \\
&5(x^1_{h,j} + \cdots + x^6_{h,j}) + 5(x^1_{j,h} + \cdots + x^6_{j,h}) + 8(x^1_{f,k} + \cdots + x^6_{f,k}) + \\
&8(x^1_{k,f} + \cdots + x^6_{k,f}) + 4(x^1_{c,g} + \cdots + x^6_{c,g}) + 4(x^1_{g,c} + \cdots + x^6_{g,c}) + \\
&8(x^1_{g,l} + \cdots + x^6_{g,l}) + 8(x^1_{l,g} + \cdots + x^6_{l,g})
\end{aligned}
$$

ILP Demand Constraint

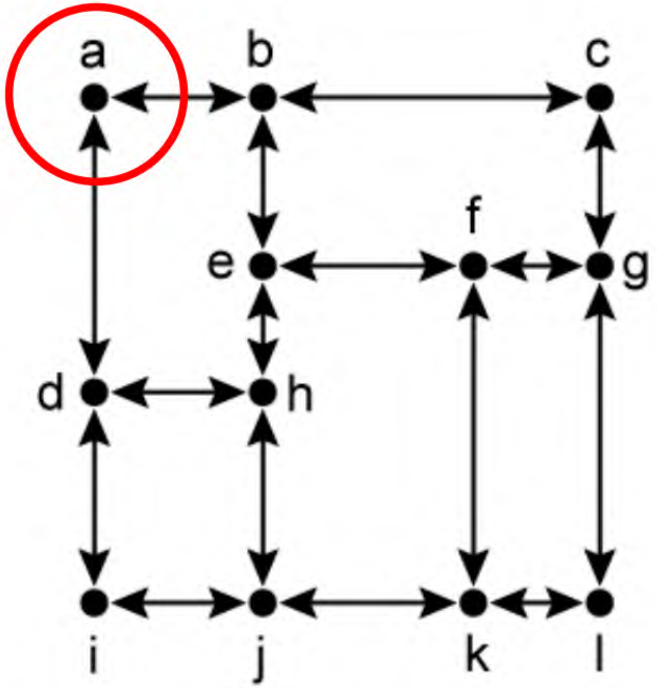
- Utilize demand constant
  - $z_v^k = 1$ means node v is the source of net k (= −1 if sink)
  - Total number of z-constants: $12 \times 6 = 72$

From net $n_1 = \{a, l\}$, we have $z_a^1 = 1$, $z_l^1 = -1$.
From net $n_2 = \{i, c\}$, we have $z_i^2 = 1$, $z_c^2 = -1$.
From net $n_3 = \{d, f\}$, we have $z_d^3 = 1$, $z_f^3 = -1$.
From net $n_4 = \{k, d\}$, we have $z_k^4 = 1$, $z_d^4 = -1$.
From net $n_5 = \{g, h\}$, we have $z_g^5 = 1$, $z_h^5 = -1$.
From net $n_6 = \{b, k\}$, we have $z_b^6 = 1$, $z_k^6 = -1$.

ILP Demand Constraint

- Node a: source of net n_1

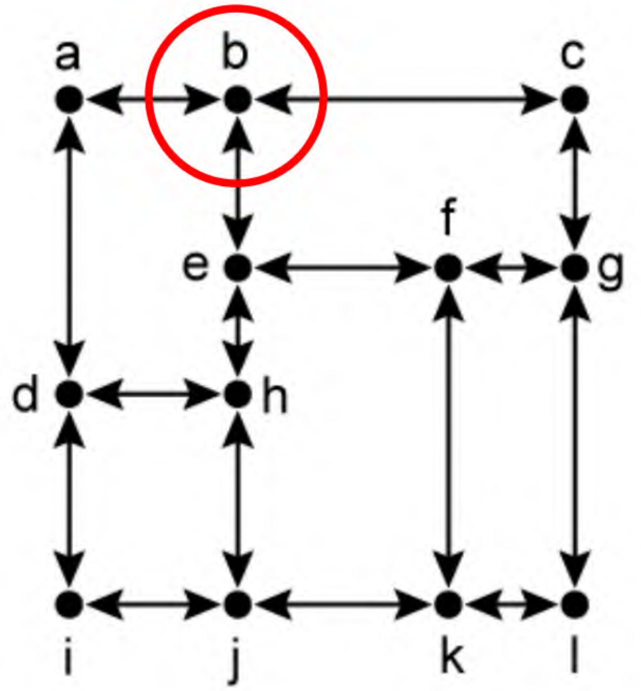
$$
\begin{aligned}
x^1_{a,b} + x^1_{a,d} - x^1_{b,a} - x^1_{d,a} &= 1 \\
x^2_{a,b} + x^2_{a,d} - x^2_{b,a} - x^2_{d,a} &= 0 \\
x^3_{a,b} + x^3_{a,d} - x^3_{b,a} - x^3_{d,a} &= 0 \\
x^4_{a,b} + x^4_{a,d} - x^4_{b,a} - x^4_{d,a} &= 0 \\
x^5_{a,b} + x^5_{a,d} - x^5_{b,a} - x^5_{d,a} &= 0 \\
x^6_{a,b} + x^6_{a,d} - x^6_{b,a} - x^6_{d,a} &= 0
\end{aligned}
$$



ILP Demand Constraint

- Node b: source of net n_6

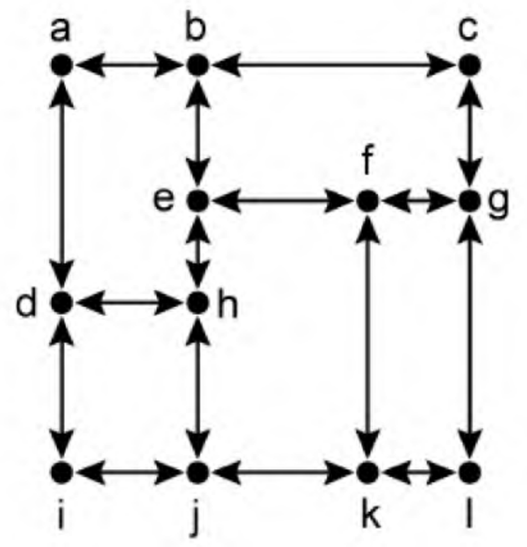
$$x^1_{b,a} + x^1_{b,e} + x^1_{b,c} - x^1_{a,b} - x^1_{e,b} - x^1_{c,b} = 0$$
$$x^2_{b,a} + x^2_{b,e} + x^2_{b,c} - x^2_{a,b} - x^2_{e,b} - x^2_{c,b} = 0$$
$$x^3_{b,a} + x^3_{b,e} + x^3_{b,c} - x^3_{a,b} - x^3_{e,b} - x^3_{c,b} = 0$$
$$x^4_{b,a} + x^4_{b,e} + x^4_{b,c} - x^4_{a,b} - x^4_{e,b} - x^4_{c,b} = 0$$
$$x^5_{b,a} + x^5_{b,e} + x^5_{b,c} - x^5_{a,b} - x^5_{e,b} - x^5_{c,b} = 0$$
$$x^6_{b,a} + x^6_{b,e} + x^6_{b,c} - x^6_{a,b} - x^6_{e,b} - x^6_{c,b} = 1$$



ILP Capacity Constraint

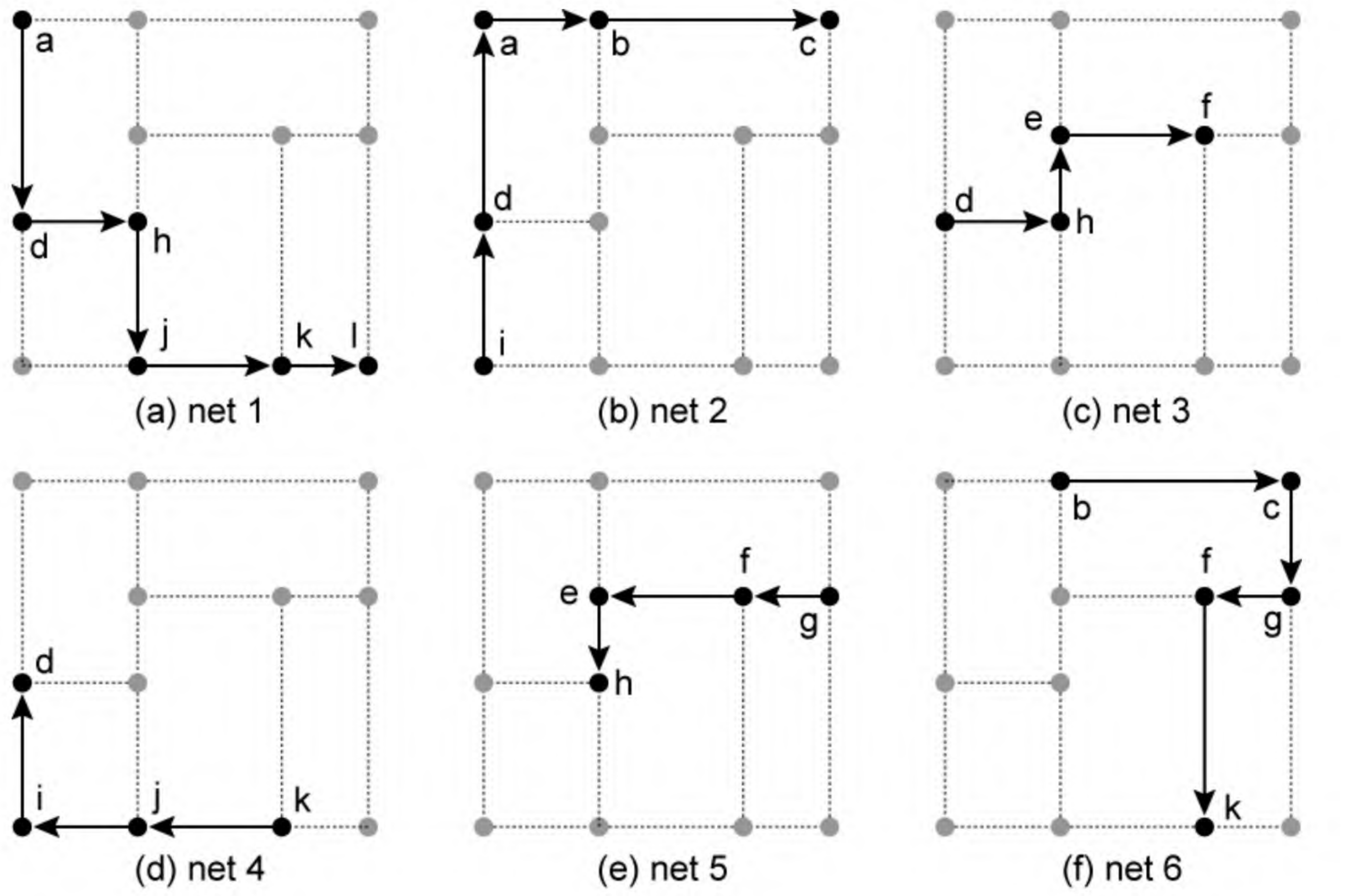
- Each edge in the routing graph allows 2 nets

$$
\begin{aligned}
&x^1_{a,b} + \cdots x^6_{a,b} + x^1_{b,a} + \cdots x^6_{b,a} \leq 2 \\
&x^1_{b,c} + \cdots + x^6_{b,c} + x^1_{c,b} + \cdots + x^6_{c,b} \leq 2 \\
&x^1_{d,h} + \cdots + x^6_{d,h} + x^1_{h,d} + \cdots + x^6_{h,d} \leq 2 \\
&x^1_{e,f} + \cdots + x^6_{e,f} + x^1_{f,e} + \cdots + x^6_{f,e} \leq 2 \\
&\cdots \\
&x^1_{h,j} + \cdots + x^6_{h,j} + x^1_{j,h} + \cdots + x^6_{j,h} \leq 2 \\
&x^1_{f,k} + \cdots + x^6_{f,k} + x^1_{k,f} + \cdots + x^6_{k,f} \leq 2 \\
&x^1_{c,g} + \cdots + x^6_{c,g} + x^1_{g,c} + \cdots + x^6_{g,c} \leq 2 \\
&x^1_{g,l} + \cdots + x^6_{g,l} + x^1_{l,g} + \cdots + x^6_{l,g} \leq 2
\end{aligned}
$$



ILP Solution

- Min-cost: 108 (= sum of WL), 22 non-zero variable



Detour

Summary of Routing

- Maze routing
  - Lee's algorithm
- Global routing and detailed routing
- Sequential routing
  - Rip-up and reroute
- Concurrent routing
  - Topology selection based ILP
  - Multi-commodity flow

Resources

- Survey FLUTE and SALT paper
- Survey NCTUgr 2.0, Dr.CU 2.0, and CU-GR
  - https://github.com/cuhk-eda/dr-cu
  - https://github.com/cuhk-eda/cu-gr