

CS612

Algorithms for Electronic Design Automation

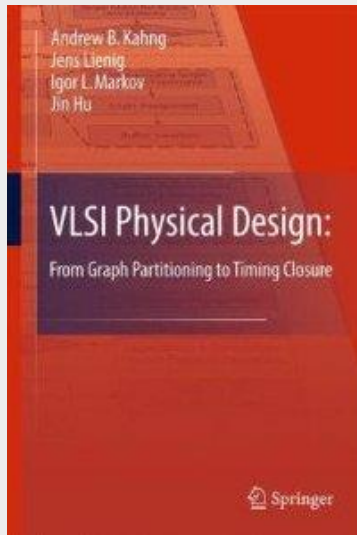


Global Routing

Mustafa Ozdal

***MOST SLIDES ARE FROM THE BOOK:
VLSI Physical Design: From Graph Partitioning to Timing Closure
MODIFICATIONS WERE MADE ON THE ORIGINAL SLIDES***

Chapter 2 – Netlist and System Partitioning

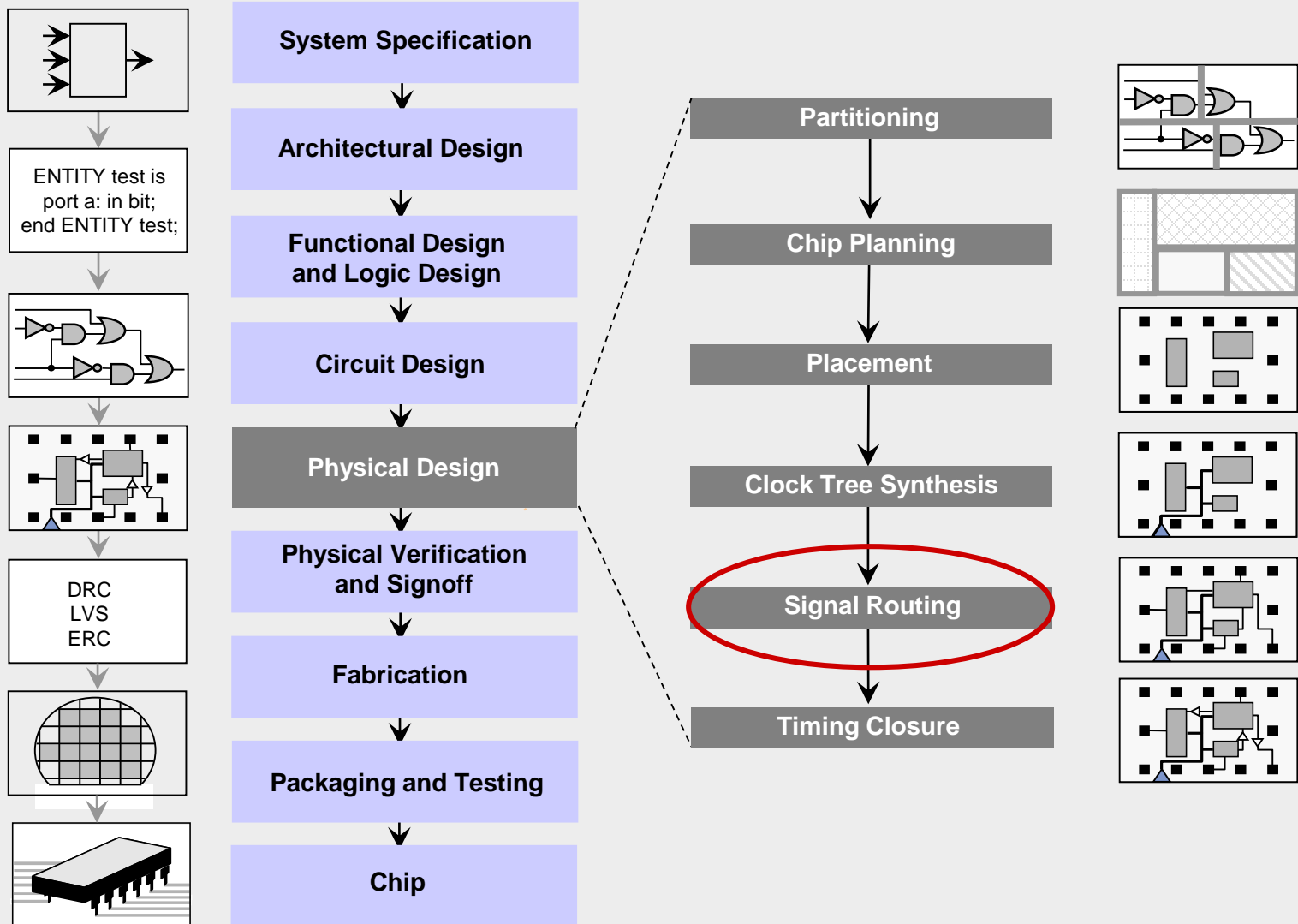


Original Authors:

Andrew B. Kahng, Jens Lienig, Igor L. Markov, Jin Hu

- 5.1 Introduction
- 5.2 Terminology and Definitions
- 5.3 Optimization Goals
- 5.4 Representations of Routing Regions
- 5.5 The Global Routing Flow
- 5.6 Single-Net Routing
 - 5.6.1 Rectilinear Routing
 - 5.6.2 Global Routing in a Connectivity Graph
 - 5.6.3 Finding Shortest Paths with Dijkstra's Algorithm
 - 5.6.4 Finding Shortest Paths with A* Search
- 5.7 Full-Netlist Routing
 - 5.7.1 Routing by Integer Linear Programming
 - 5.7.2 Rip-Up and Reroute (RRR)
- 5.8 Modern Global Routing
 - 5.8.1 Pattern Routing
 - 5.8.2 Negotiated-Congestion Routing

5.1 Introduction



Given a placement, a netlist and technology information,

- determine the necessary wiring, e.g., net topologies and specific routing segments, to connect these cells
- while respecting constraints, e.g., design rules and routing resource capacities, and
- optimizing routing objectives, e.g., minimizing total wirelength and maximizing timing slack.

5.1 Introduction: General Routing Problem

Netlist:

$$N_1 = \{C_4, D_6, B_3\}$$

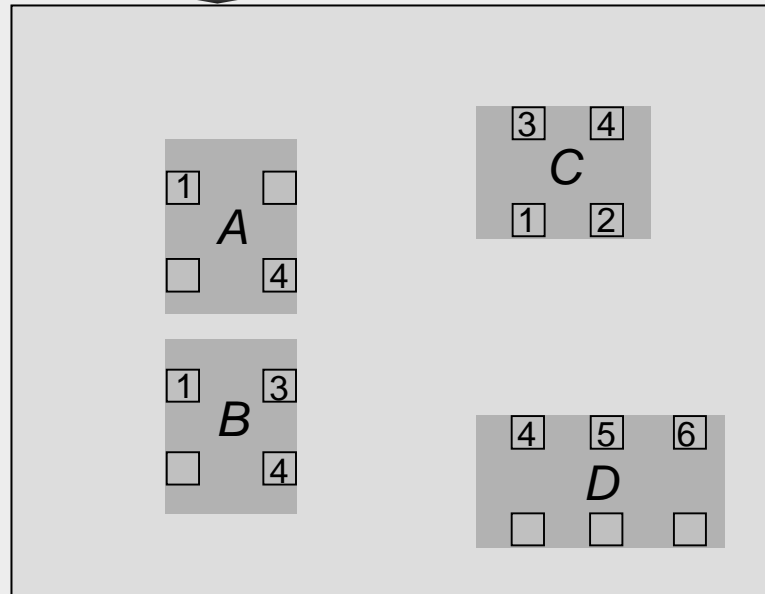
$$N_2 = \{D_4, B_4, C_1, A_4\}$$

$$N_3 = \{C_2, D_5\}$$

$$N_4 = \{B_1, A_1, C_3\}$$

Technology Information
(Design Rules)

Placement result



5.1 Introduction: General Routing Problem

Netlist:

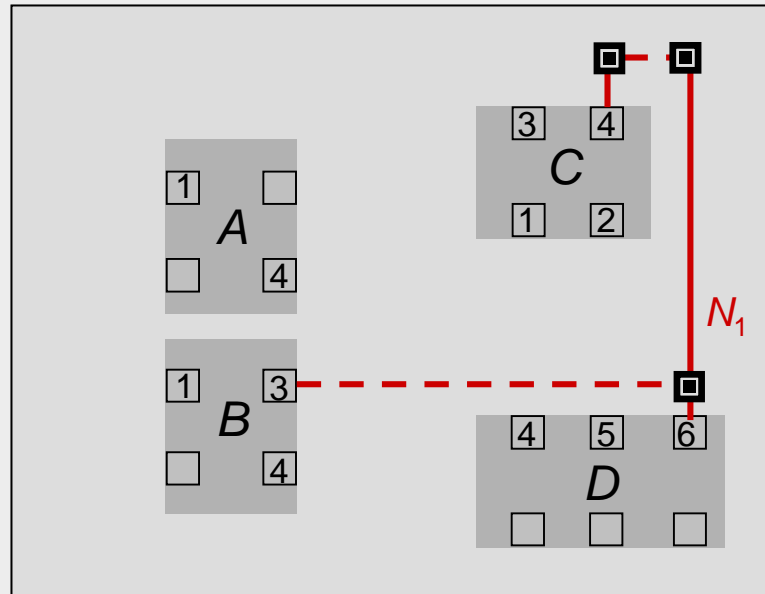
$$N_1 = \{C_4, D_6, B_3\}$$

$$N_2 = \{D_4, B_4, C_1, A_4\}$$

$$N_3 = \{C_2, D_5\}$$

$$N_4 = \{B_1, A_1, C_3\}$$

Technology Information
(Design Rules)



5.1 Introduction: General Routing Problem

Netlist:

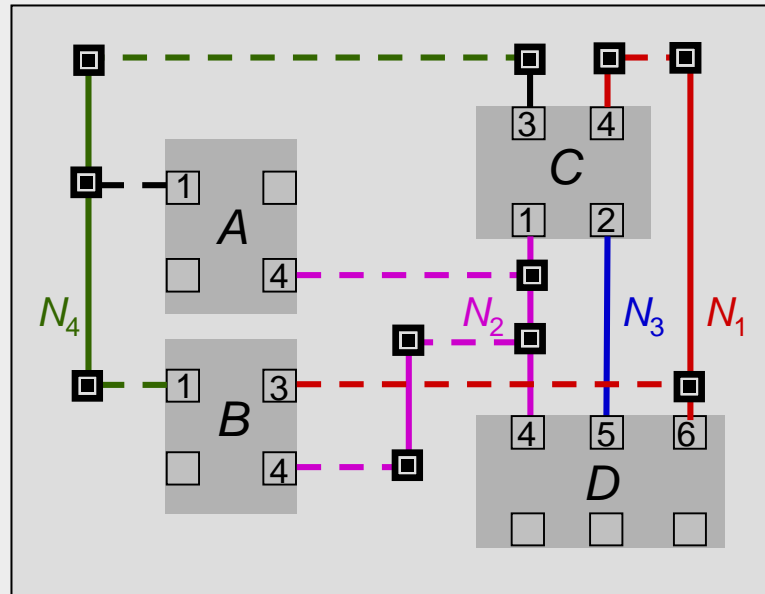
$$N_1 = \{C_4, D_6, B_3\}$$

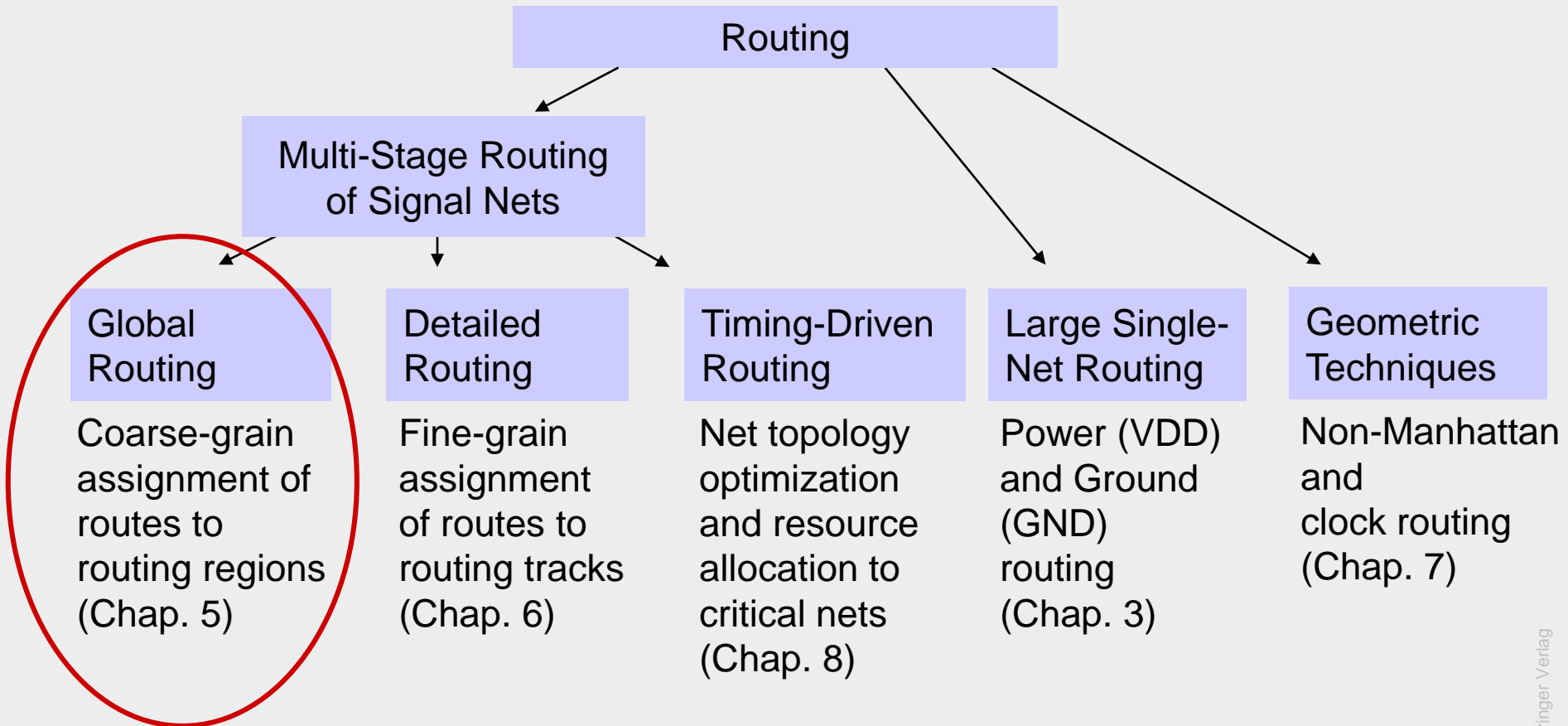
$$N_2 = \{D_4, B_4, C_1, A_4\}$$

$$N_3 = \{C_2, D_5\}$$

$$N_4 = \{B_1, A_1, C_3\}$$

Technology Information
(Design Rules)

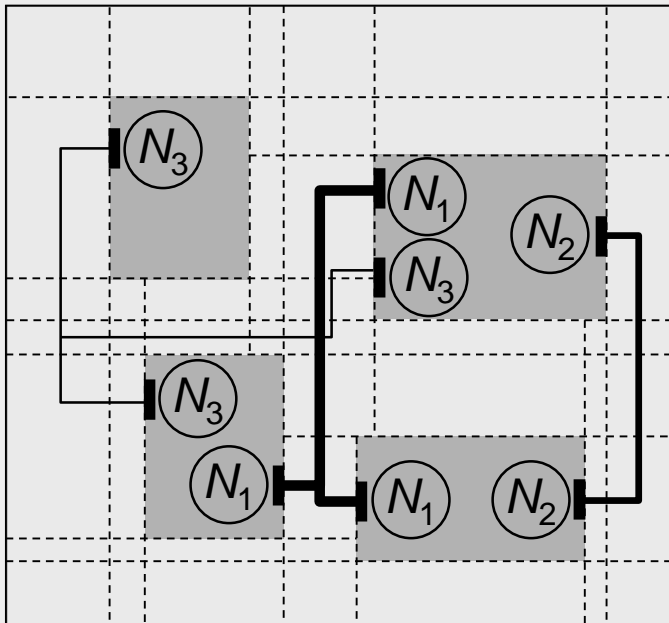




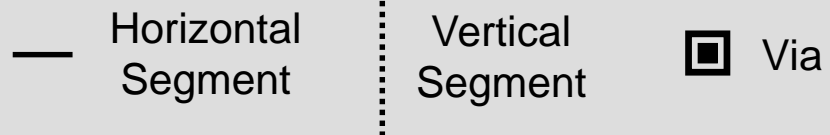
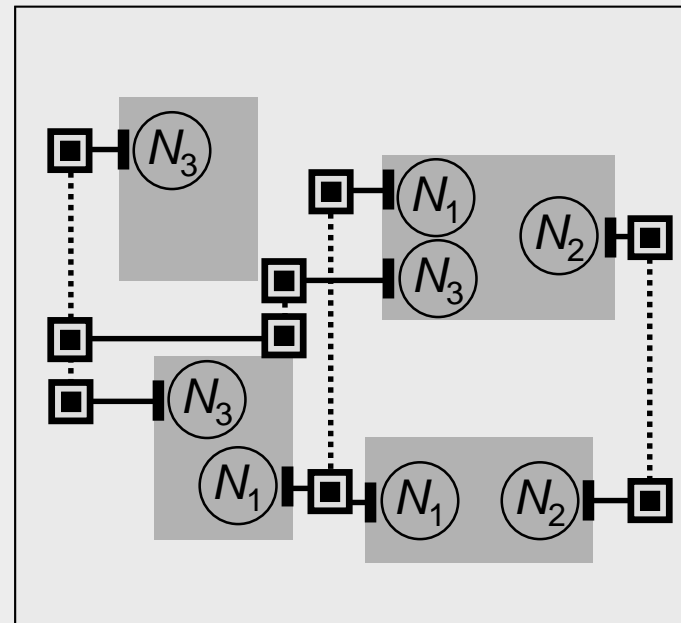
Global Routing

- Wire segments are tentatively assigned (embedded) within the chip layout
 - Chip area is represented by a coarse routing grid
 - Available routing resources are represented by edges with capacities in a grid graph
- ⇒ Nets are assigned to these routing resources

Global Routing



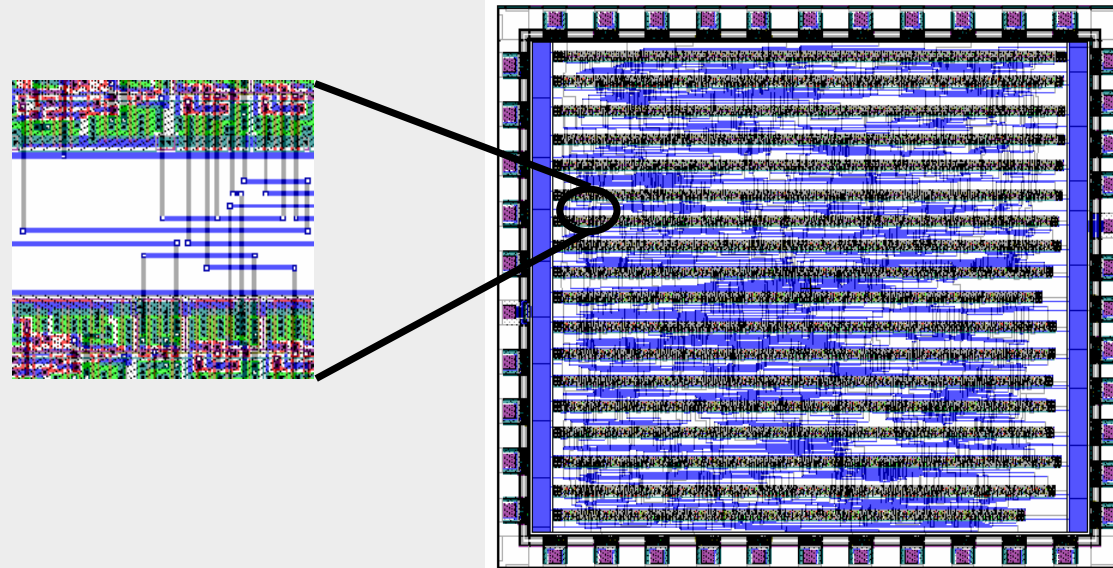
Detailed Routing



5.2 Terminology and Definitions

Channel

Rectangular routing region with pins on two opposite sides

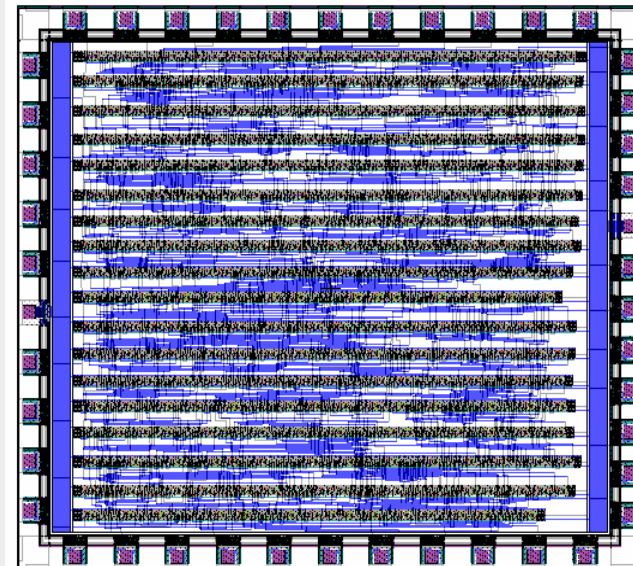
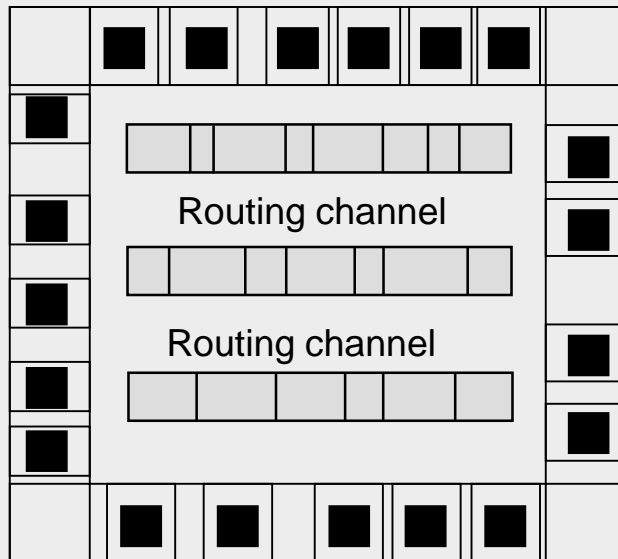


Standard cell layout (Two-layer routing)

5.2 Terminology and Definitions

Channel

Rectangular routing region with pins on two opposite sides

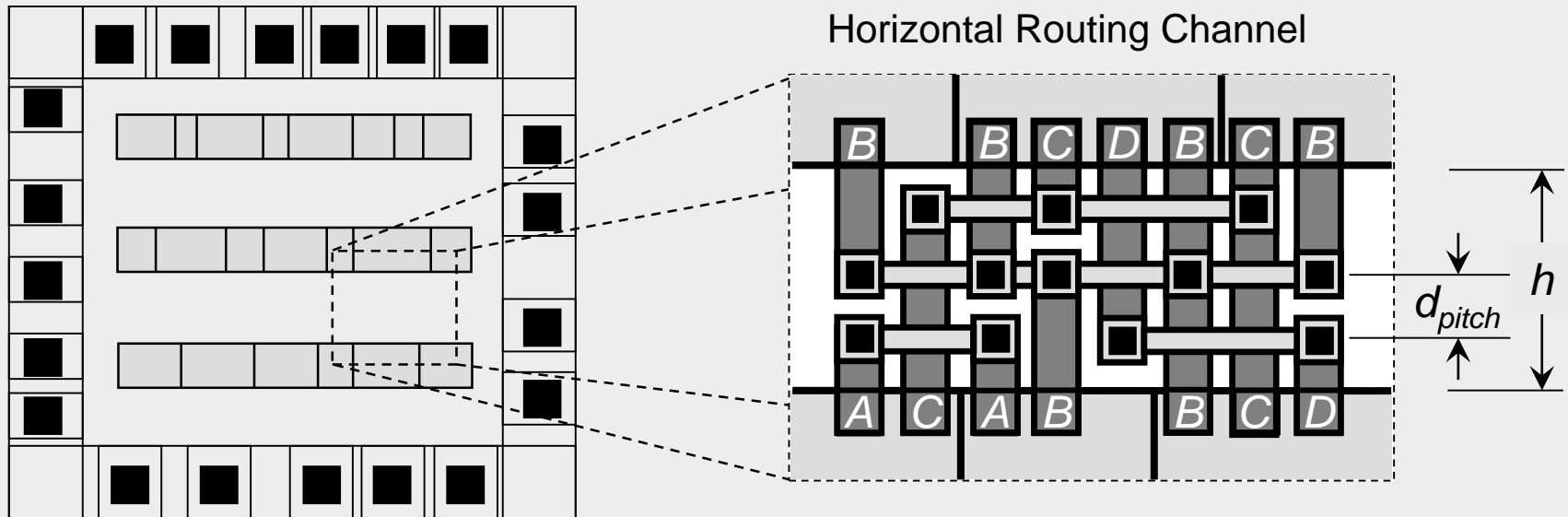


Standard cell layout (Two-layer routing)

5.2 Terminology and Definitions

Capacity

Number of available routing tracks or columns



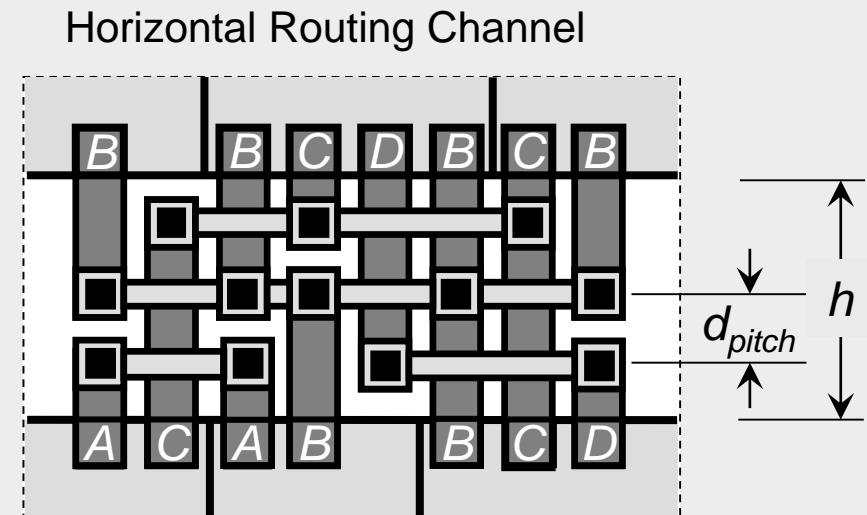
5.2 Terminology and Definitions

Capacity

Number of available routing tracks or columns

- For single-layer routing, the capacity is the height h of the channel divided by the pitch d_{pitch}
- For multilayer routing, the capacity σ is the sum of the capacities of all layers.

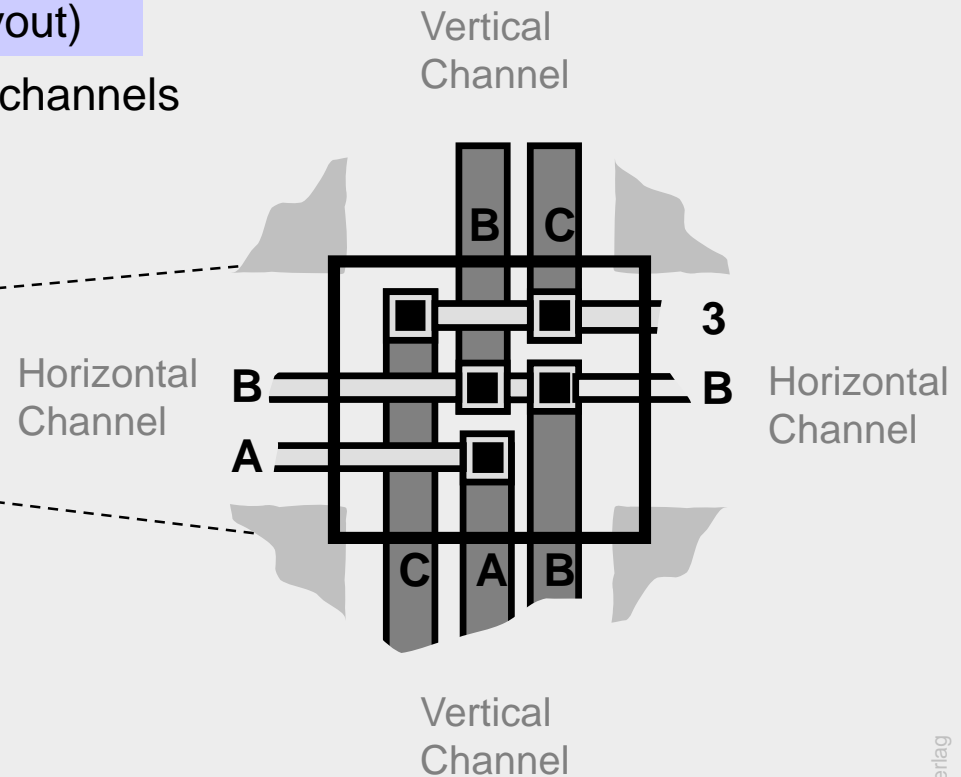
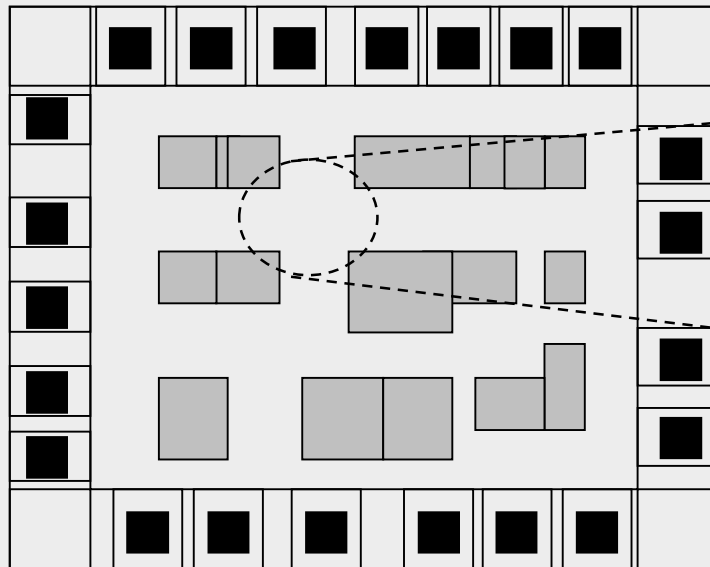
$$\sigma(Layers) = \sum_{layer \in Layers} \left\lfloor \frac{h}{d_{pitch}(layer)} \right\rfloor$$



5.2 Terminology and Definitions

Switchbox (Two-layer macro cell layout)

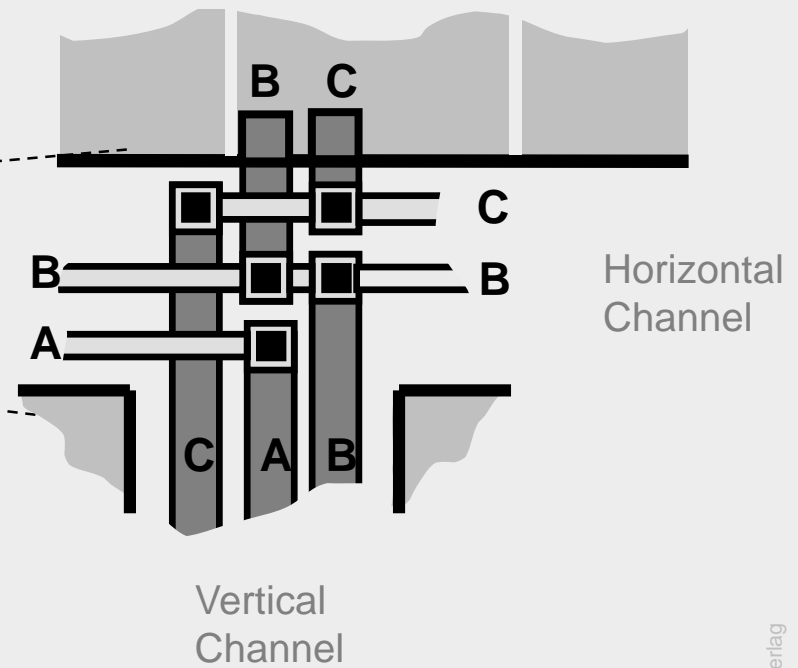
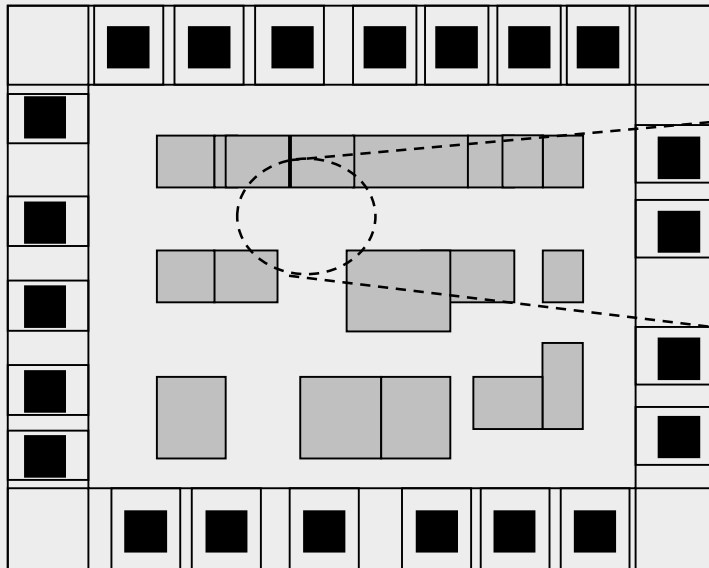
Intersection of horizontal and vertical channels



Horizontal channel is routed after vertical channel is routed

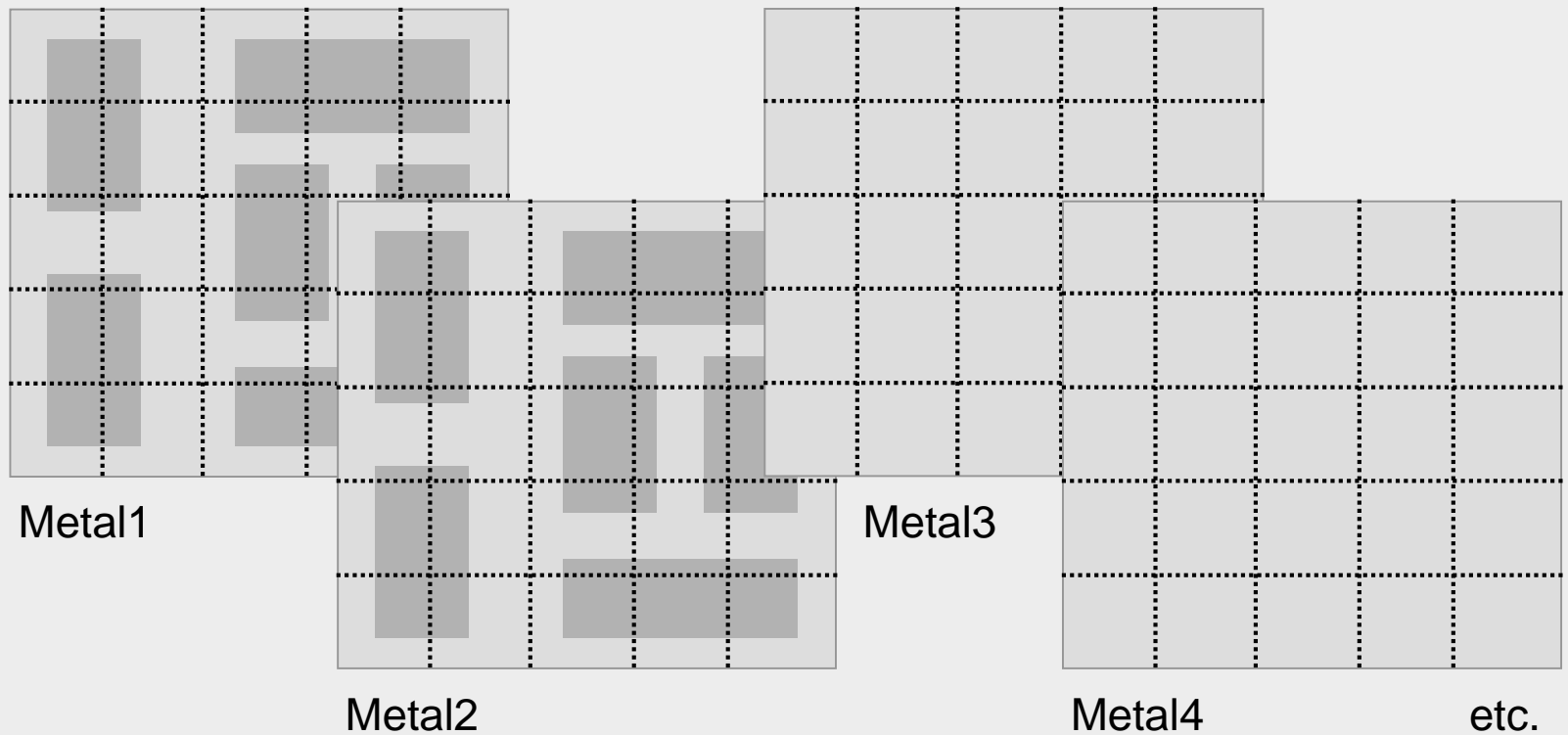
5.2 Terminology and Definitions

T-junction (Two-layer macro cell layout)



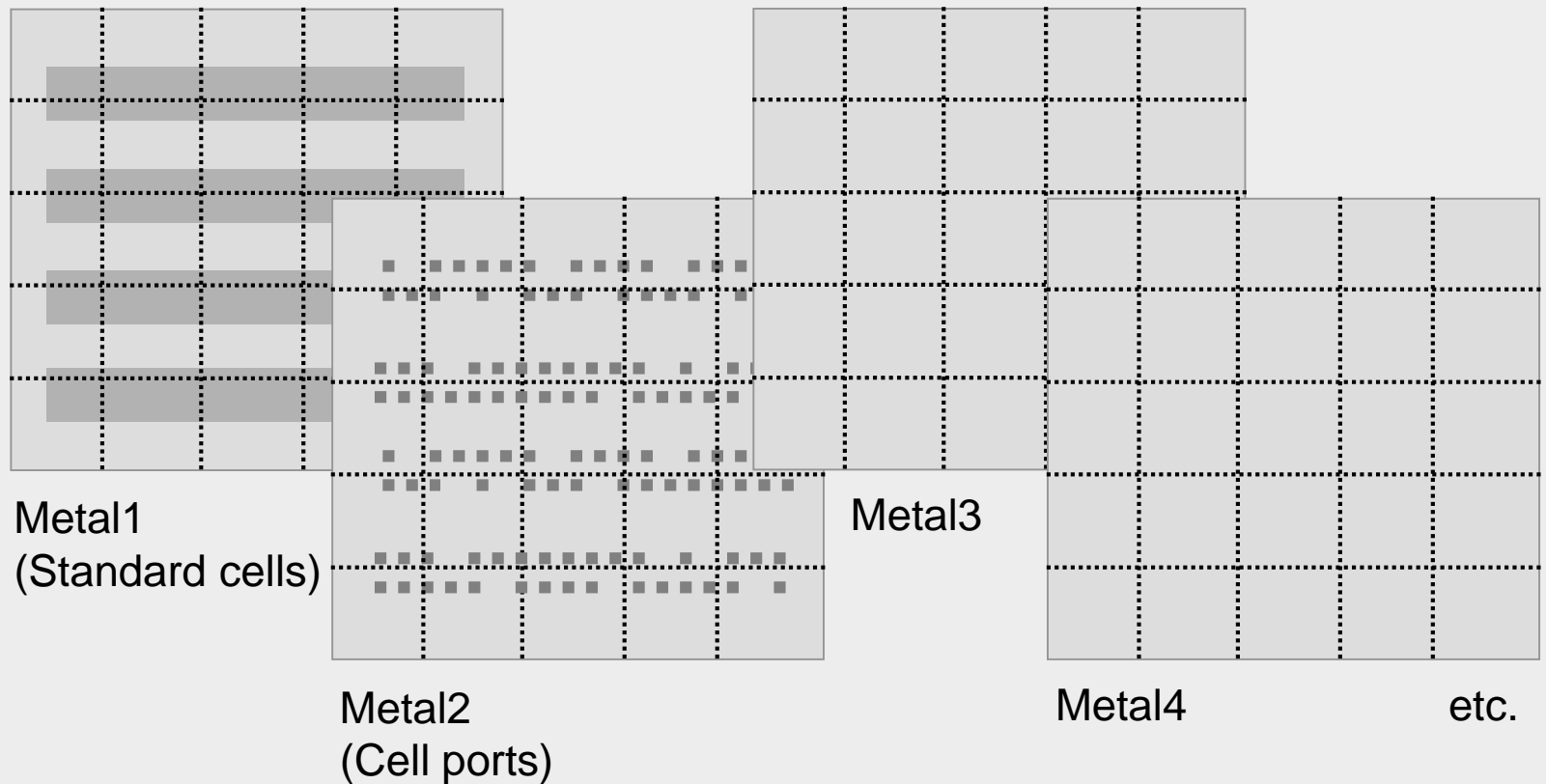
5.2 Terminology and Definitions

Gcells (Tiles) with macro cell layout



5.2 Terminology and Definitions

Gcells (Tiles) with standard cells



5.3 Optimization Goals

- Global routing seeks to
 - determine whether a given placement is routable, and
 - determine a coarse routing for all nets within available routing regions
- Considers goals such as
 - minimizing total wirelength, and
 - reducing signal delays on critical nets

5.4 Representations of Routing Regions

5.1 Introduction

5.2 Terminology and Definitions

5.3 Optimization Goals

 **5.4 Representations of Routing Regions**

5.5 The Global Routing Flow

5.6 Single-Net Routing

5.6.1 Rectilinear Routing

5.6.2 Global Routing in a Connectivity Graph

5.6.3 Finding Shortest Paths with Dijkstra's Algorithm

5.6.4 Finding Shortest Paths with A* Search

5.7 Full-Netlist Routing

5.7.1 Routing by Integer Linear Programming

5.7.2 Rip-Up and Reroute (RRR)

5.8 Modern Global Routing

5.8.1 Pattern Routing

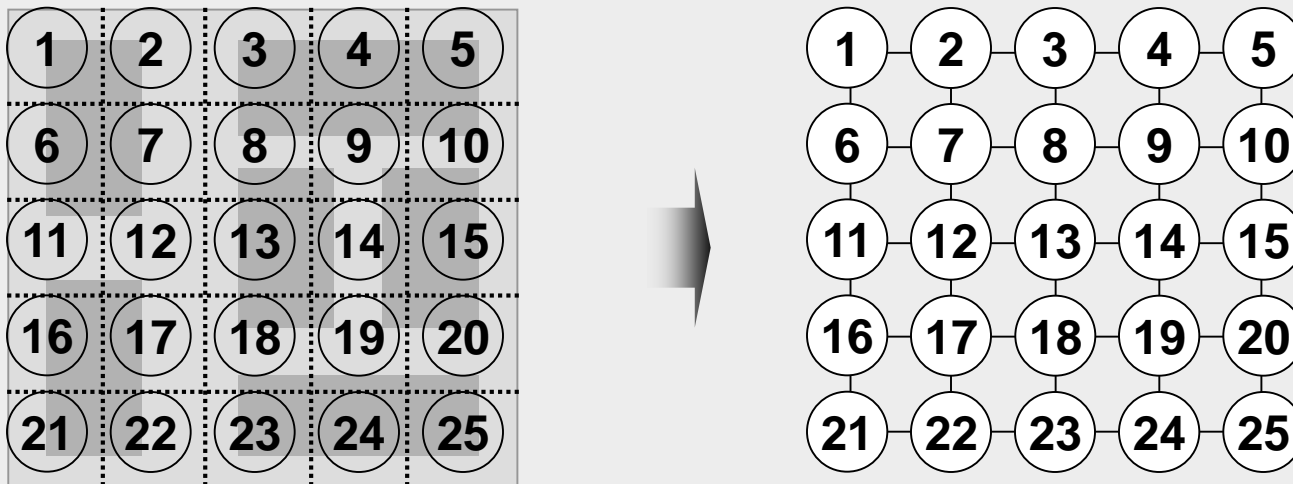
5.8.2 Negotiated-Congestion Routing

5.4 Representations of Routing Regions

- Routing regions are represented using efficient data structures
- Routing context is captured using a graph, where
 - nodes represent routing regions and
 - edges represent adjoining regions
- Capacities are associated with both edges and nodes to represent available routing resources

5.4 Representations of Routing Regions

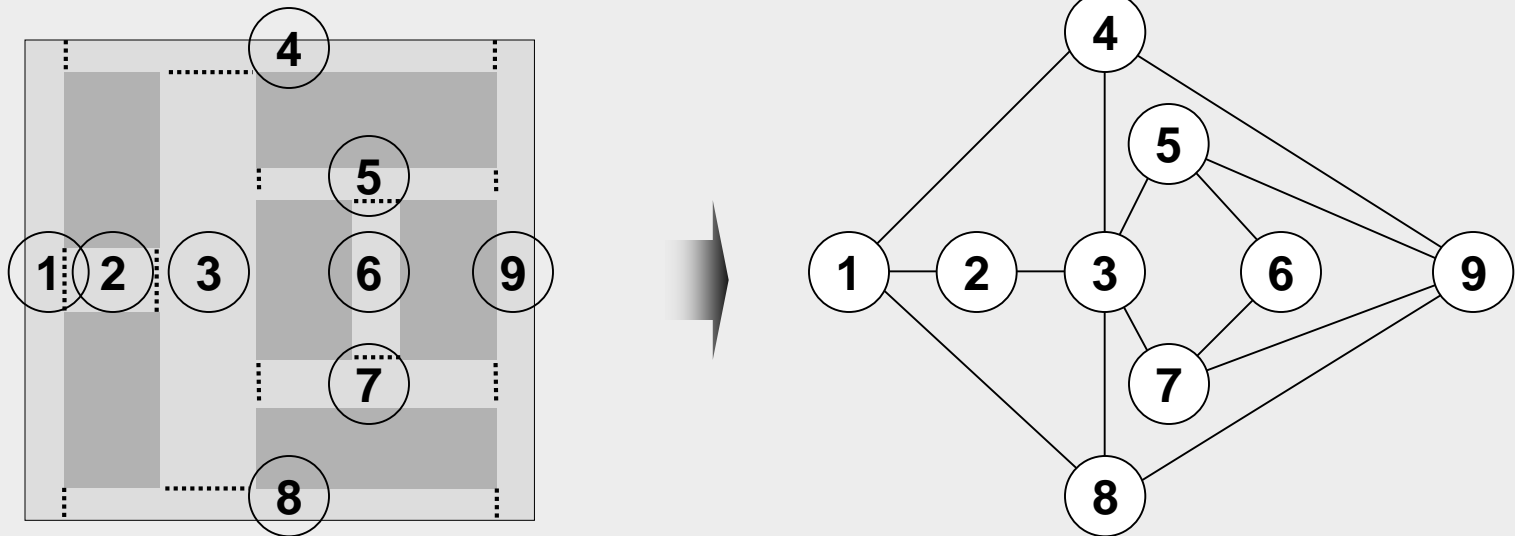
Grid graph model



$ggrid = (V, E)$, where the nodes $v \in V$ represent the **routing grid cells (*gcells*)** and the edges represent connections of grid cell pairs (v_i, v_j)

5.4 Representations of Routing Regions

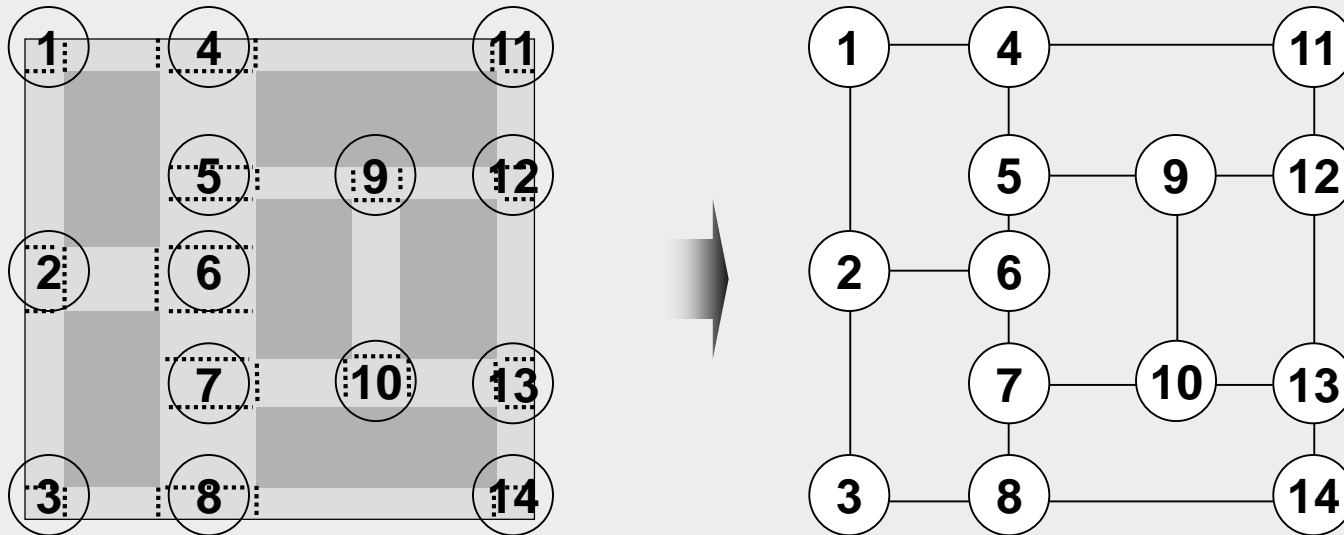
Channel connectivity graph



$G = (V, E)$, where the nodes $v \in V$ represent **channels**, and the edges E represent adjacencies of the channels


5.4 Representations of Routing Regions

Switchbox connectivity graph

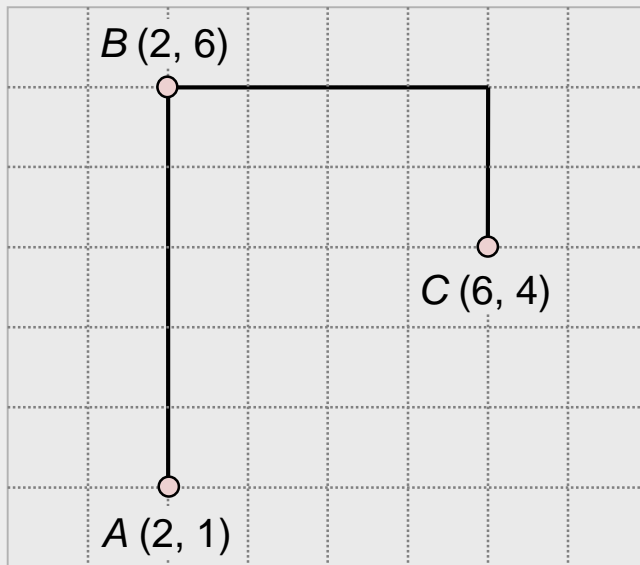


$G = (V, E)$, where the nodes $v \in V$ represent **switchboxes** and an edge exists between two nodes if the corresponding switchboxes are on opposite sides of the same channel

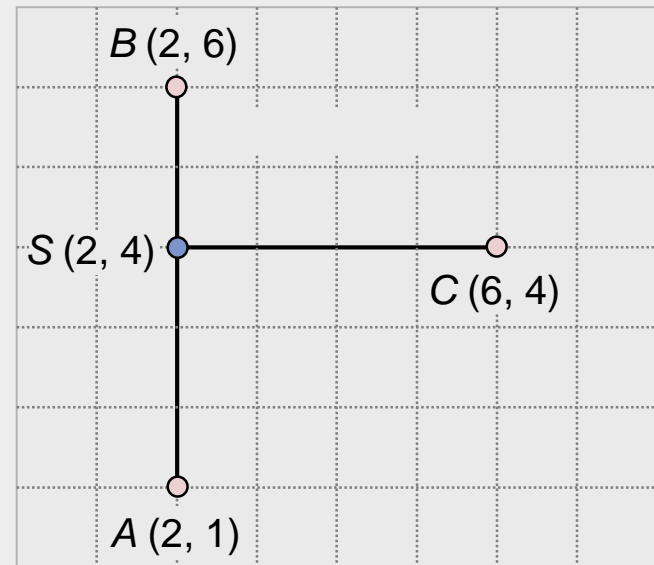
1. Defining the routing regions (Region definition)
 - Layout area is divided into routing regions
 - Nets can also be routed over standard cells
 - Regions, capacities and connections are represented by a graph
2. Mapping nets to the routing regions (Region assignment)
 - Each net of the design is assigned to one or several routing regions to connect all of its pins
 - Routing capacity, timing and congestion affect mapping

- 5.1 Introduction
- 5.2 Terminology and Definitions
- 5.3 Optimization Goals
- 5.4 Representations of Routing Regions
- 5.5 The Global Routing Flow
-  **5.6 Single-Net Routing**
 - 5.6.1 Rectilinear Routing
 - 5.6.2 Global Routing in a Connectivity Graph
 - 5.6.3 Finding Shortest Paths with Dijkstra's Algorithm
 - 5.6.4 Finding Shortest Paths with A* Search
- 5.7 Full-Netlist Routing
 - 5.7.1 Routing by Integer Linear Programming
 - 5.7.2 Rip-Up and Reroute (RRR)
- 5.8 Modern Global Routing
 - 5.8.1 Pattern Routing
 - 5.8.2 Negotiated-Congestion Routing

5.6.1 Rectilinear Routing



Rectilinear minimum spanning tree (RMST)



Rectilinear Steiner minimum tree (RSMT)

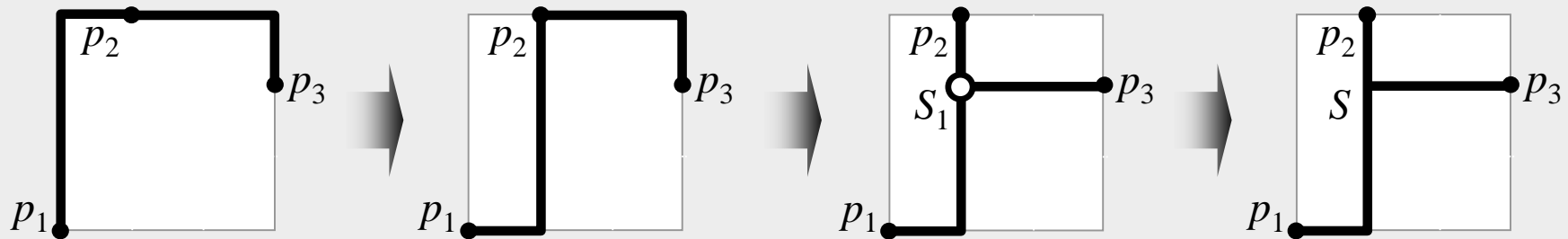
5.6.1 Rectilinear Routing

- An RMST can be computed in $O(p^2 \lg p)$ time, where p is the number of terminals in the net using methods such as Prim's Algorithm
- Prim's Algorithm builds an MST by starting with a single terminal and greedily adding least-cost edges to the partially-constructed tree
- Advanced computational-geometric techniques reduce the runtime to $O(p \log p)$

Characteristics of an RSMT

- An RSMT for a p -pin net has between 0 and $p - 2$ (inclusive) Steiner points
- The degree of any terminal pin is 1, 2, 3, or 4
The degree of a Steiner point is either 3 or 4
- A RSMT is always enclosed in the minimum bounding box (MBB) of the net
- The total edge length L_{RSMT} of the RSMT is at least half the perimeter of the minimum bounding box of the net: $L_{RSMT} \geq L_{MBB} / 2$

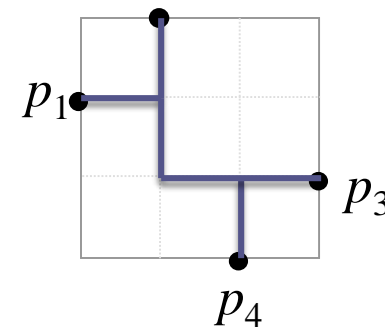
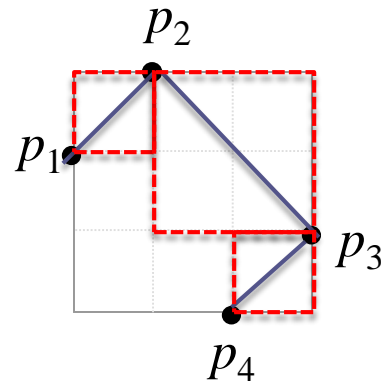
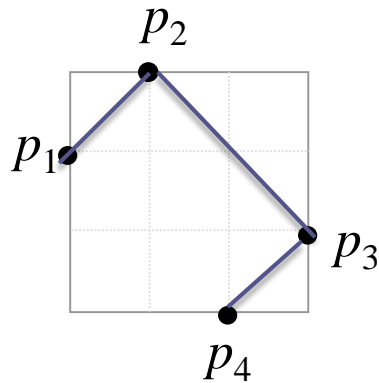
Transforming an initial RMST into a low-cost RSMT



Construct *L*-shapes between points with (most) overlap of net segments

Final tree (RSMT)

Converting an MST to RSMT

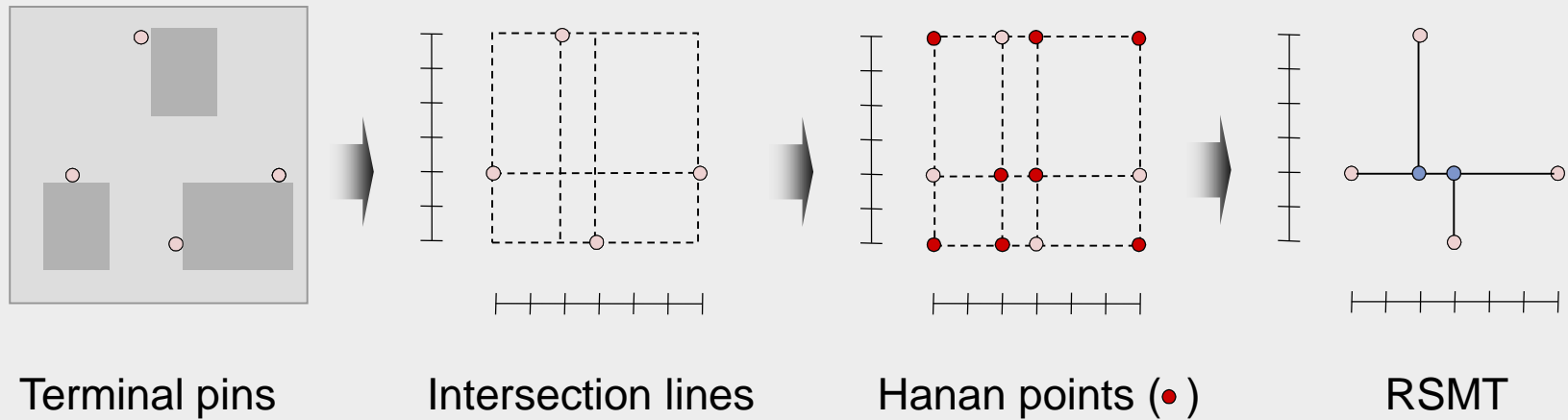


- Start with a minimum spanning tree (MST) with flylines
- Consider the bounding boxes with largest overlaps
- Choose the L-routes leading to the max segment sharing

Hanan grid

- Adding Steiner points to an RMST can significantly reduce the wirelength
- Maurice Hanan proved that for finding Steiner points, it suffices to consider only points located at the intersections of vertical and horizontal lines that pass through terminal pins
- The **Hanan grid** consists of the lines $x = x_p$, $y = y_p$ that pass through the location (x_p, y_p) of each terminal pin p
- The Hanan grid contains at most $(n^2 - n)$ candidate Steiner points ($n =$ number of pins), thereby greatly reducing the solution space for finding an RSMT

5.6.1 Rectilinear Routing

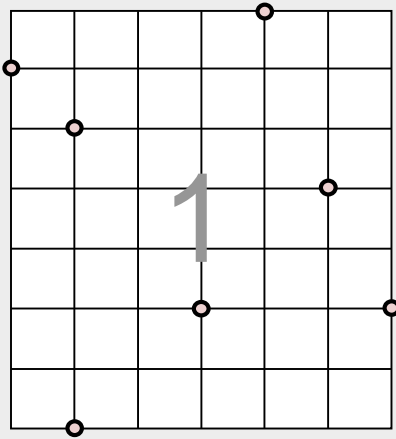


A Sequential Steiner Tree Heuristic

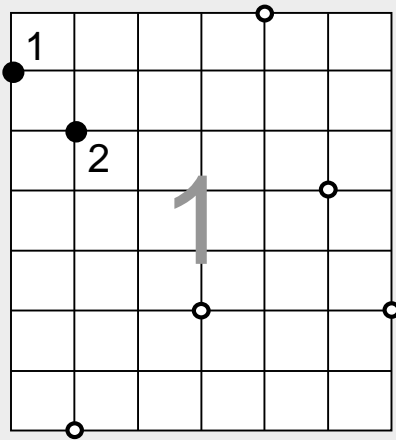
1. Find the closest (in terms of rectilinear distance) pin pair, construct their minimum bounding box (MBB)
2. Find the closest point pair (p_{MBB}, p_C) between any point p_{MBB} on the MBB and p_C from the set of pins to consider
3. Construct the MBB of p_{MBB} and p_C
4. Add the L -shape that p_{MBB} lies on to T (deleting the other L -shape). If p_{MBB} is a pin, then add any L -shape of the MBB to T .
5. Goto step 2 until the set of pins to consider is empty



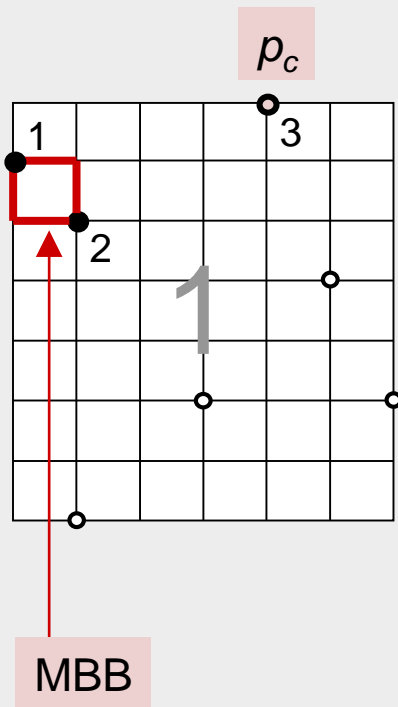
5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



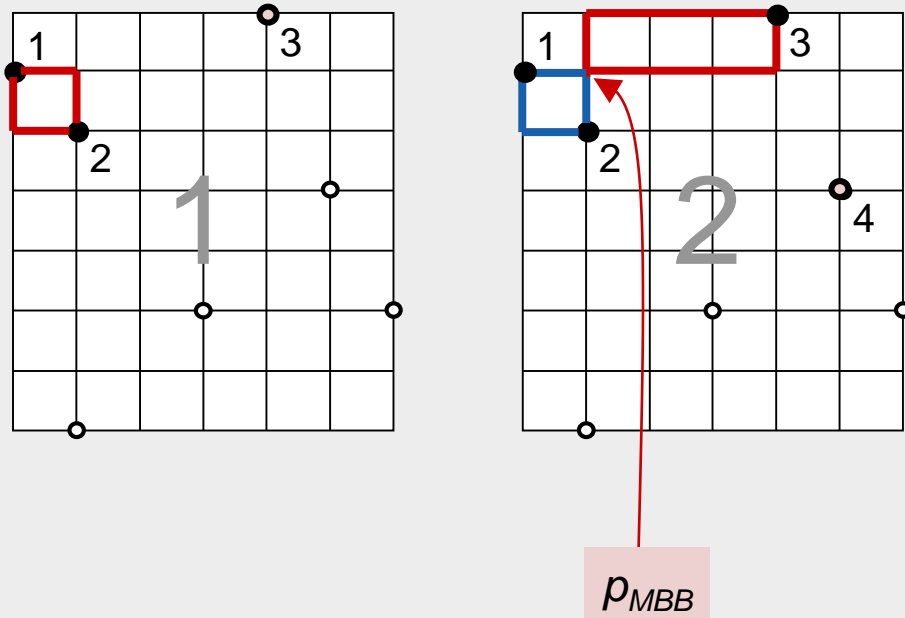
5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



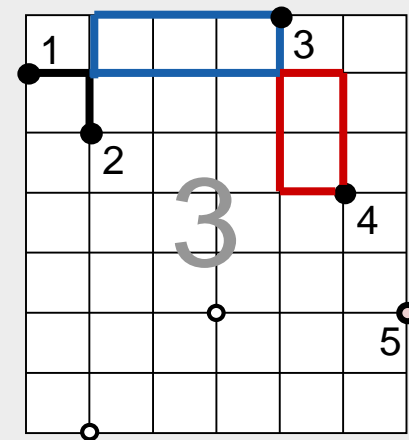
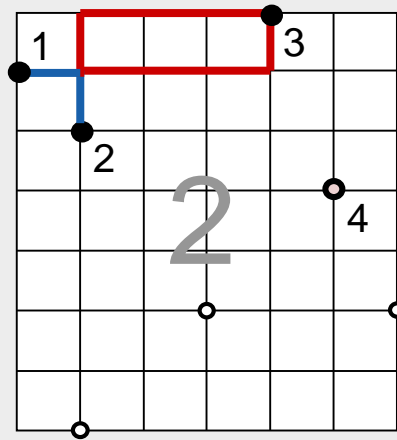
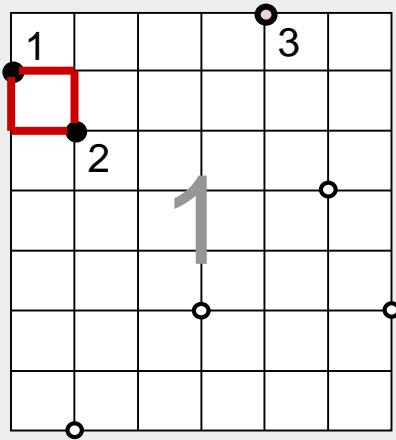
5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



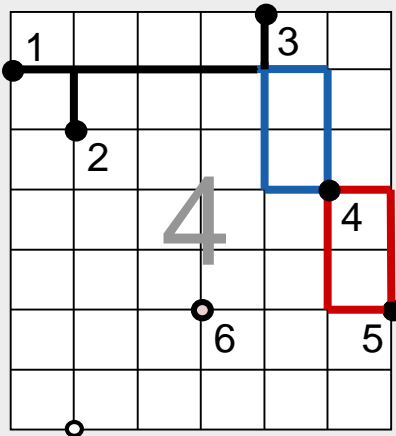
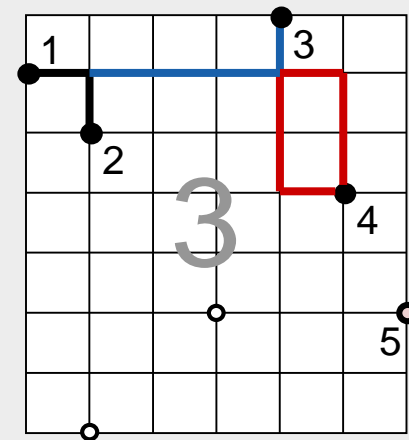
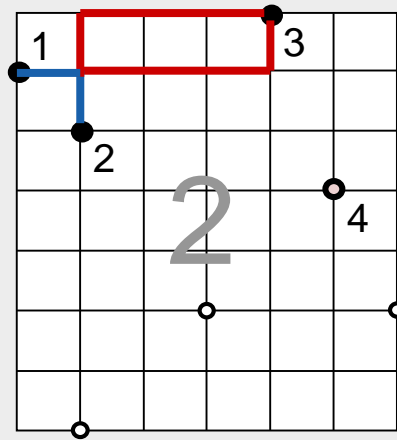
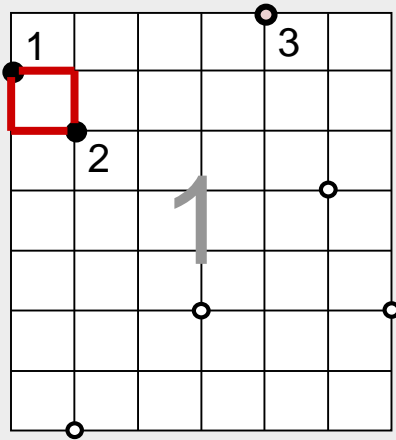
5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



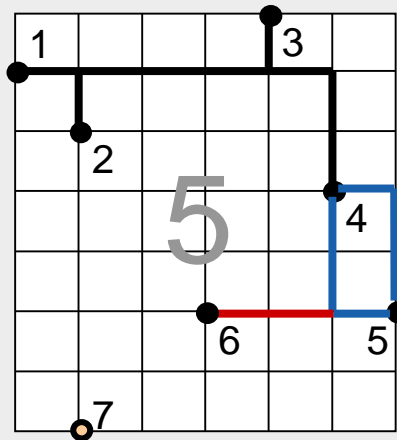
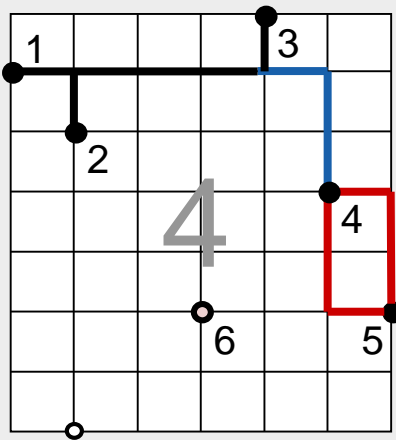
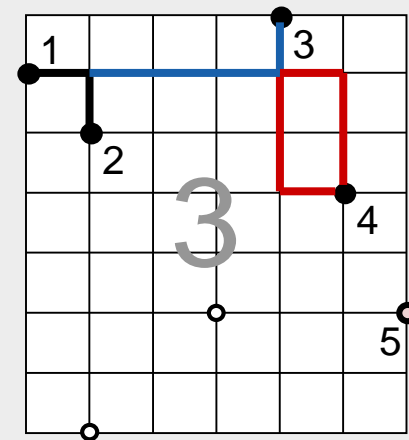
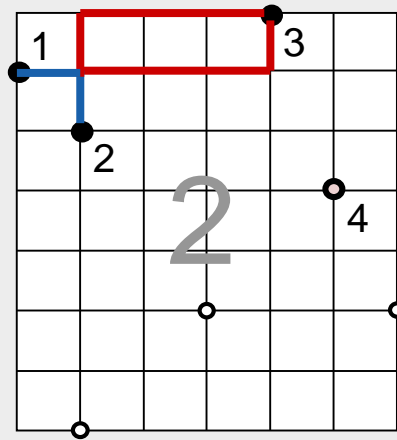
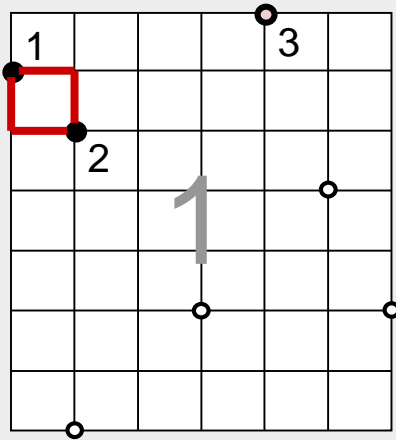
5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



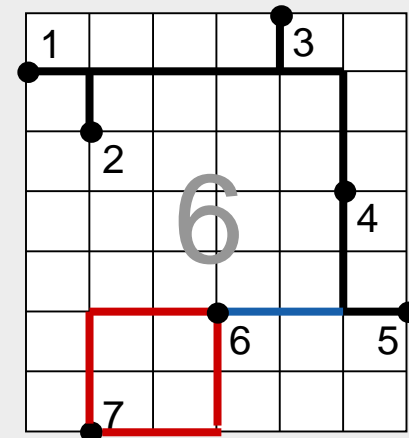
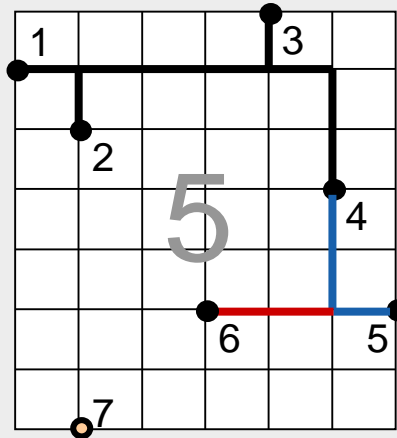
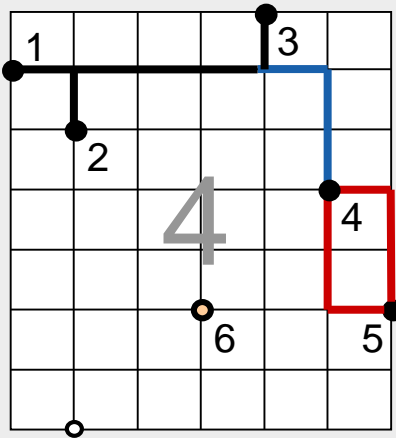
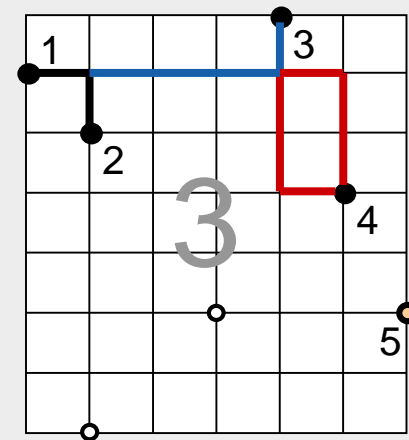
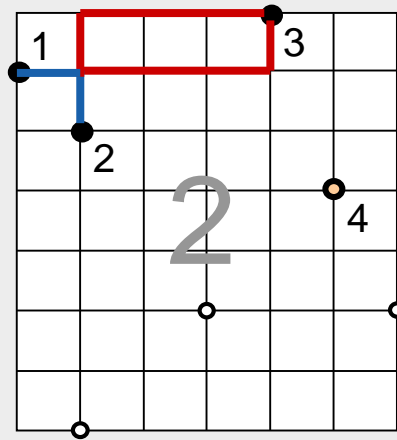
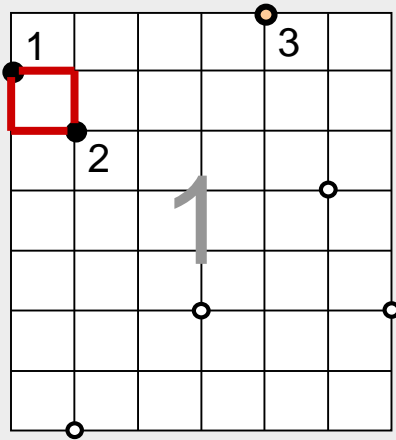
5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



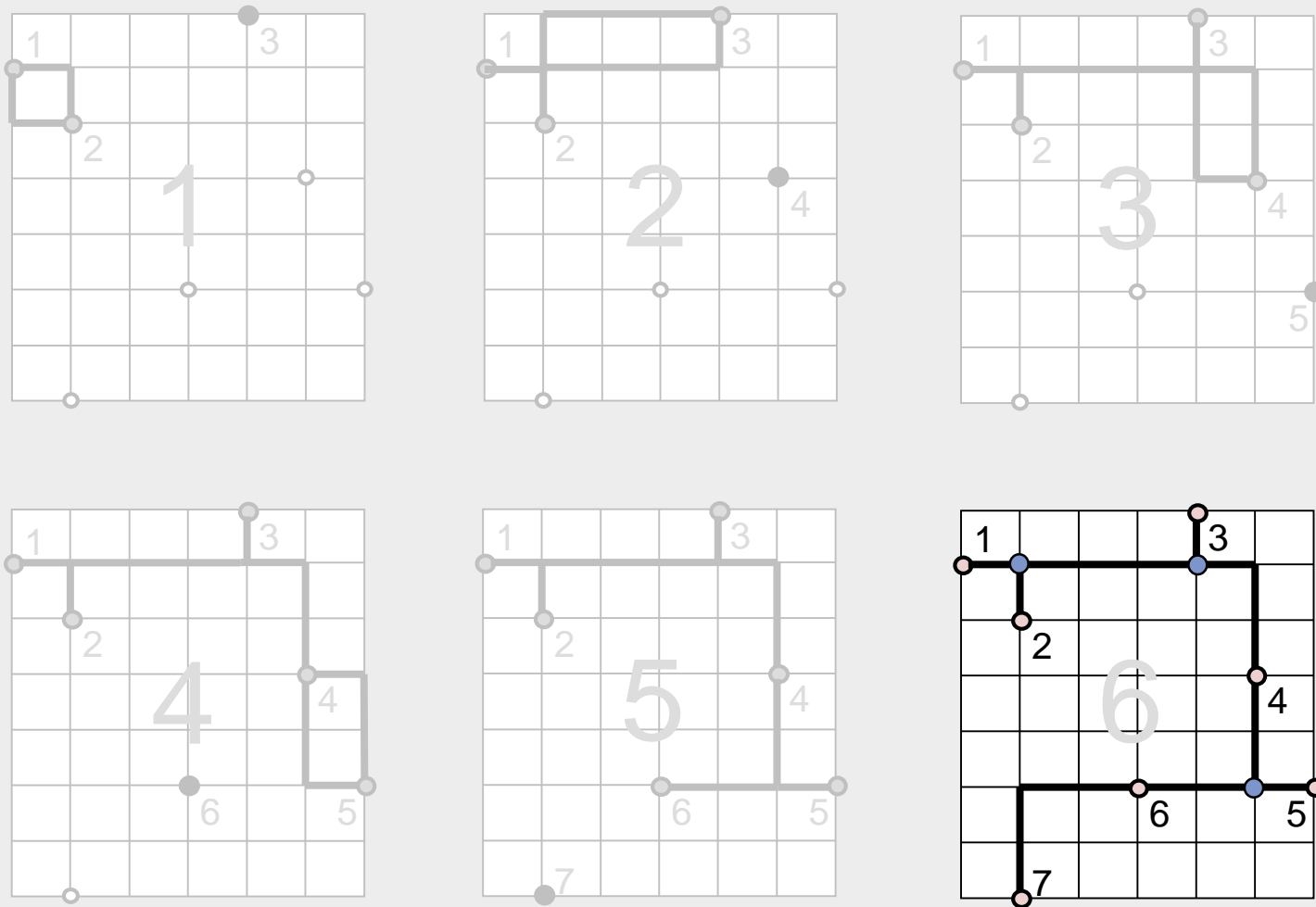
5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



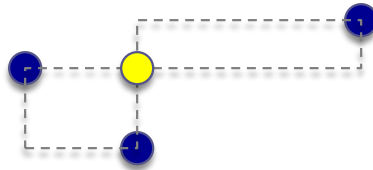
5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



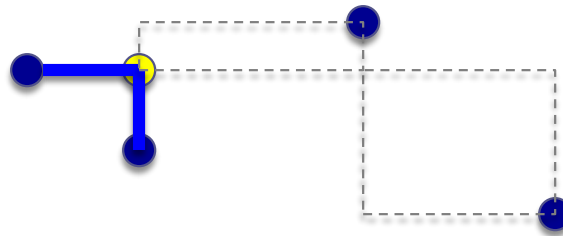
5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



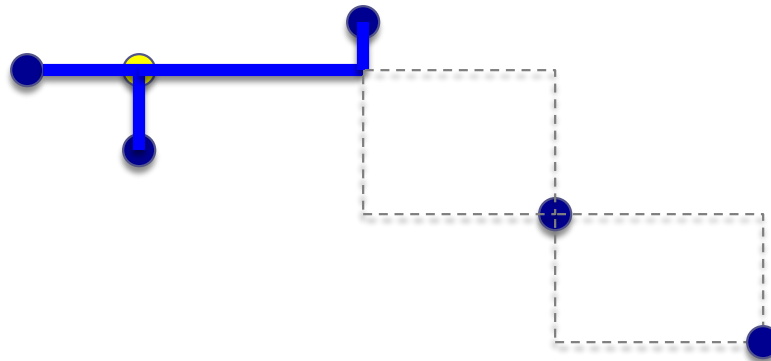
What is wrong with this heuristic?



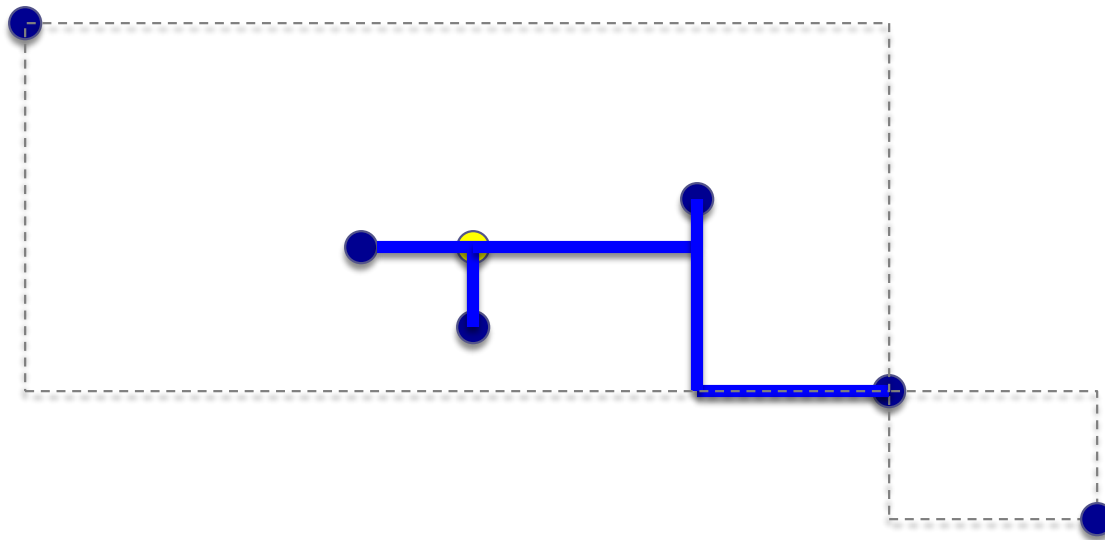
What is wrong with this heuristic?



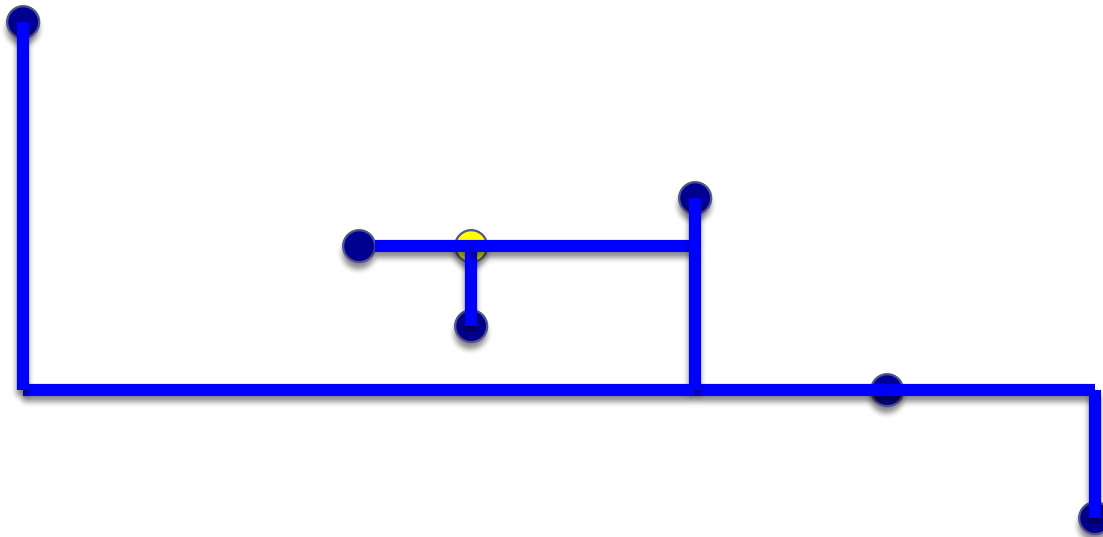
What is wrong with this heuristic?



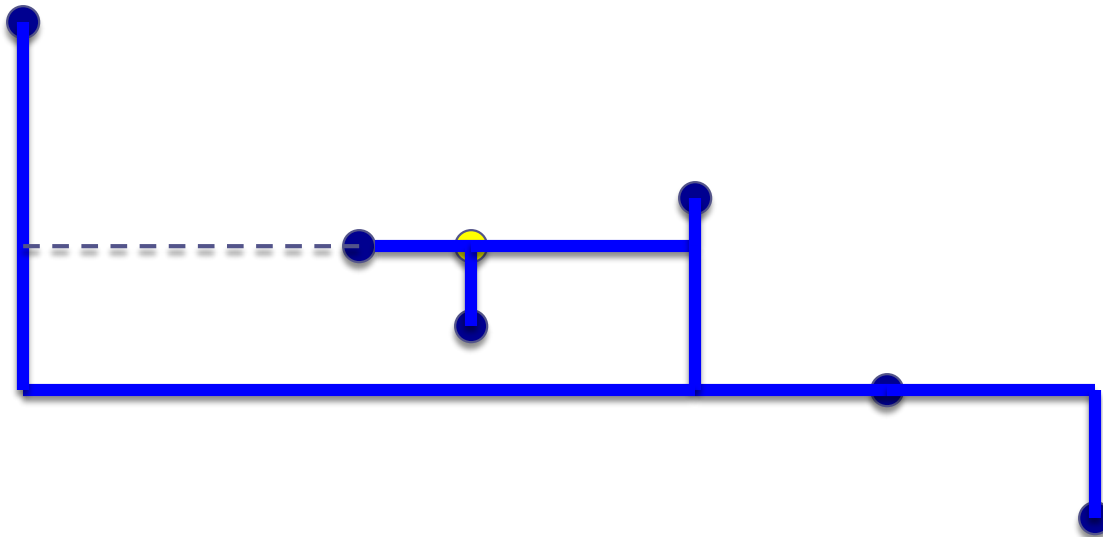
What is wrong with this heuristic?



What is wrong with this heuristic?



What is wrong with this heuristic?



Sequential processing of the pins leads to suboptimality.
Using the dashed segments would decrease the total wirelength.

Iterated 1-Steiner Approach

Notation:

$$\Delta\text{MST}(A, x) = \text{cost}(\text{MST}(A)) - \text{cost}(\text{MST}(A \cup x))$$

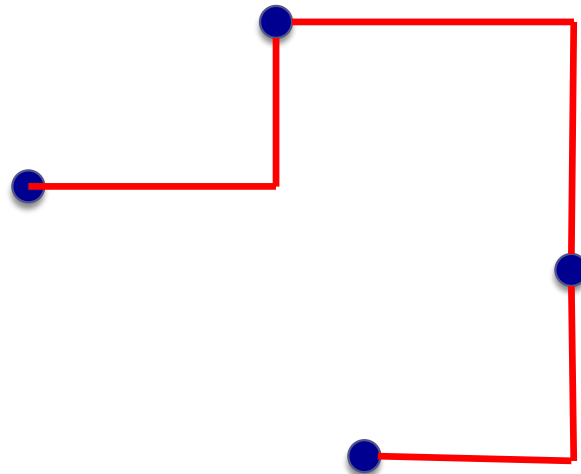
i.e. the change in the MST cost after we add the extra point x

Iterated 1-Steiner Heuristic:

1. Start with the original point set P
2. Find the Steiner point x such that $\Delta\text{MST}(P, x)$ is maximum
3. If $\Delta\text{MST}(P, x) > 0$ then add x to P
4. Remove the Steiner points in P that have degree ≤ 2
5. Repeat steps 2-4 until $\Delta\text{MST}(P, x) < 0$

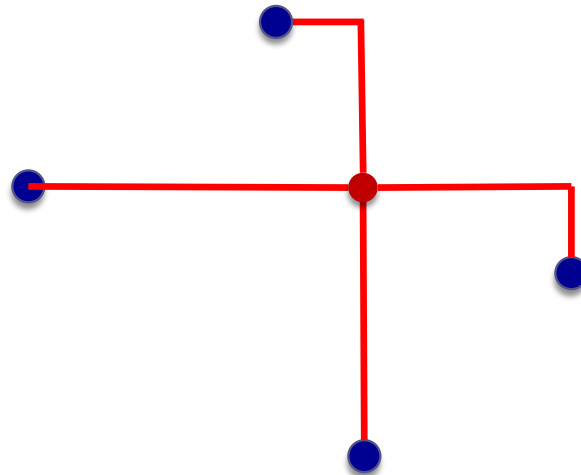
Example: Iterated 1-Steiner

Construct initial MST



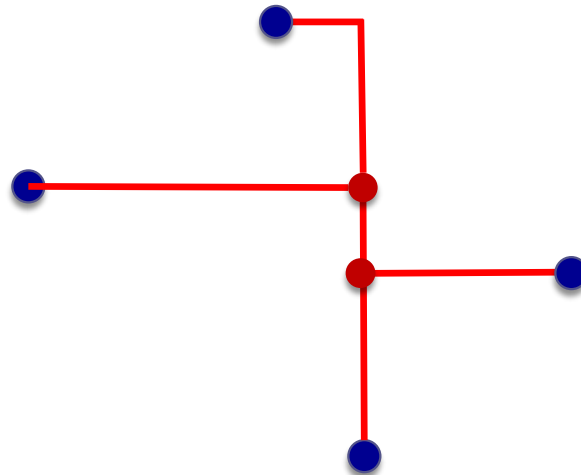
Example: Iterated 1-Steiner

Determine the new Steiner point that will reduce the wirelength most



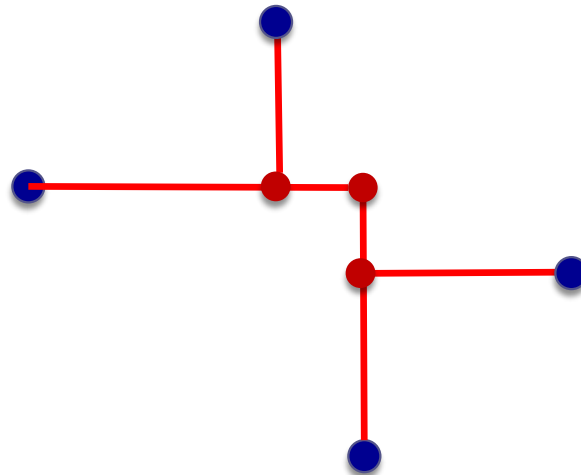
Example: Iterated 1-Steiner

Determine the new Steiner point that will reduce the wirelength most



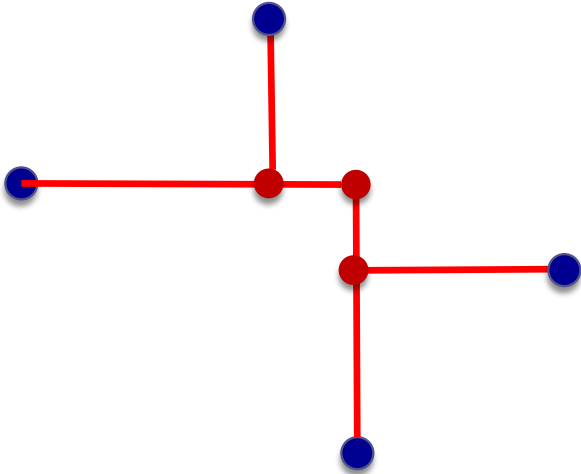
Example: Iterated 1-Steiner

Determine the new Steiner point that will reduce the wirelength most



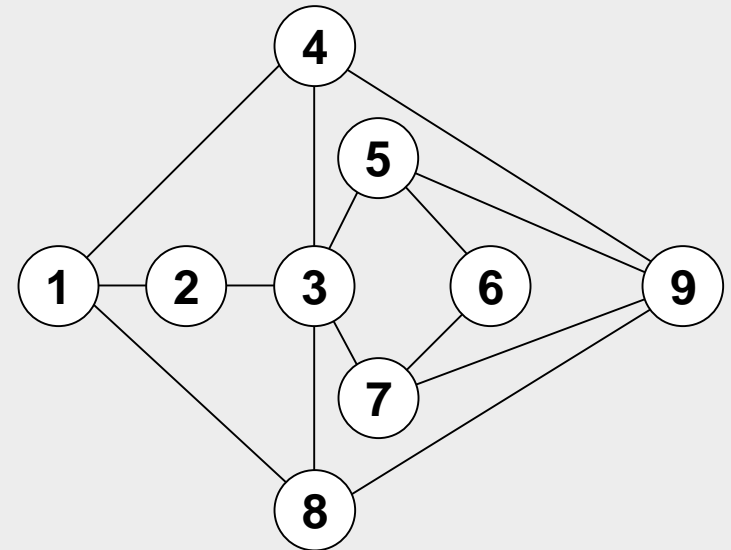
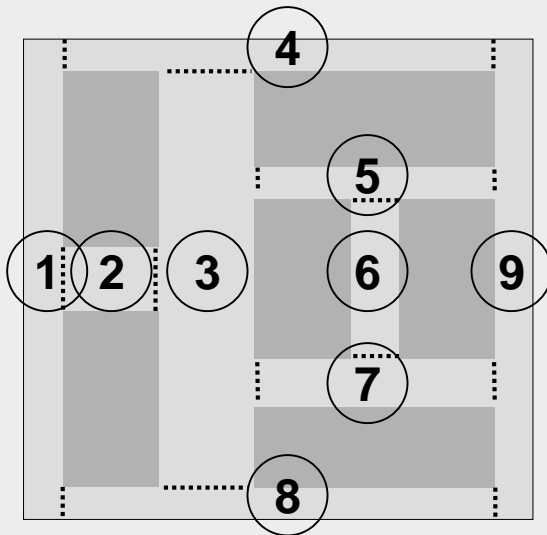
Example: Iterated 1-Steiner

Final Steiner tree

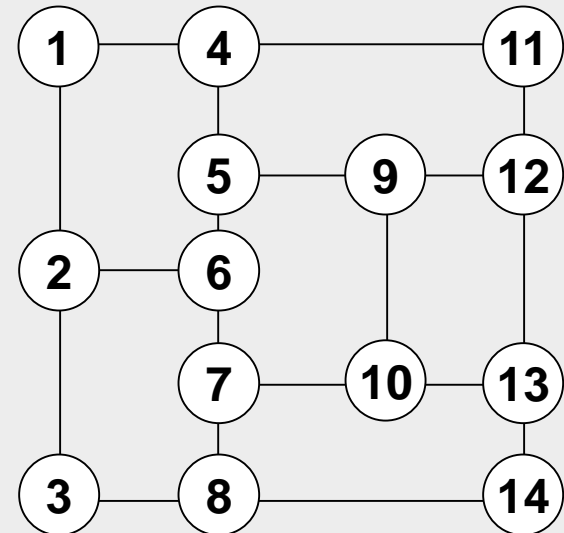
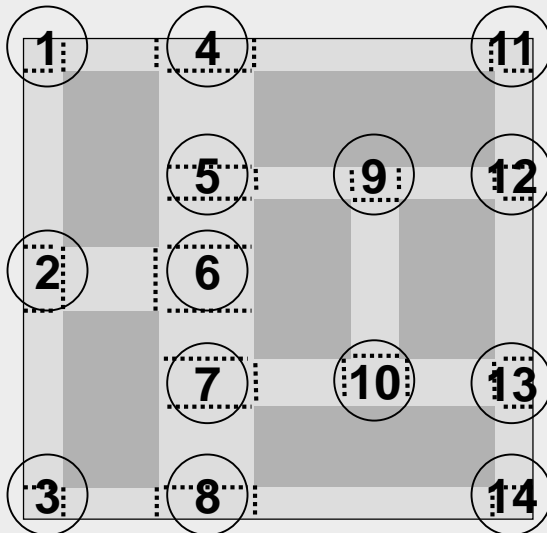


5.6.2 Global Routing in a Connectivity Graph

Channel connectivity graph

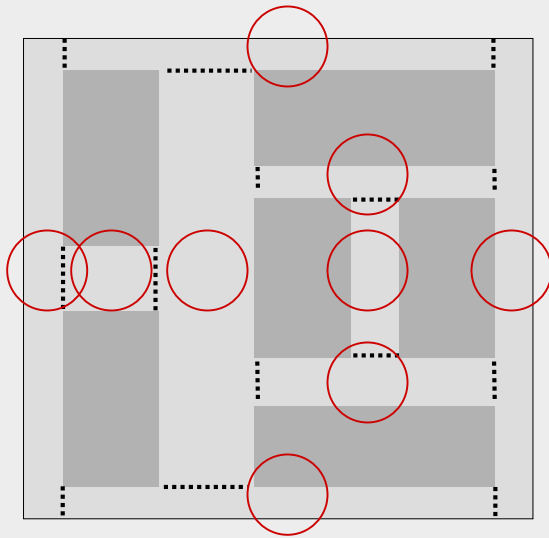


Switchbox connectivity graph

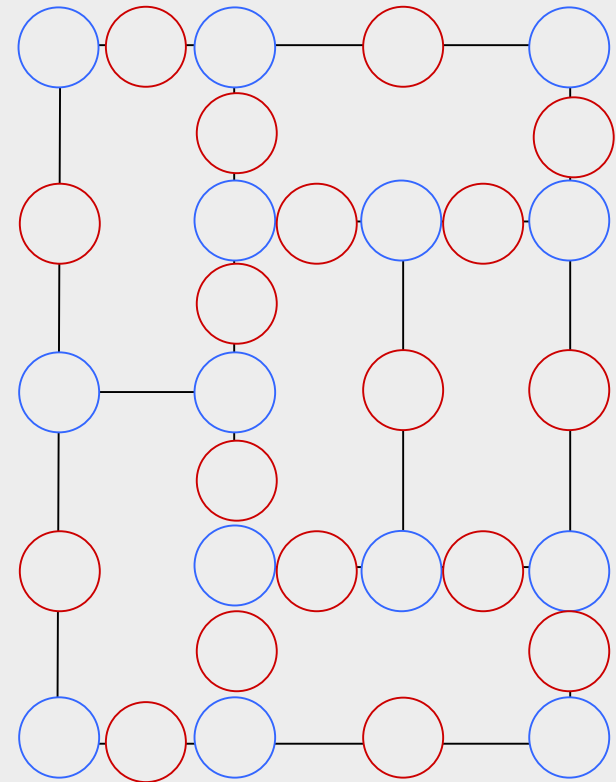
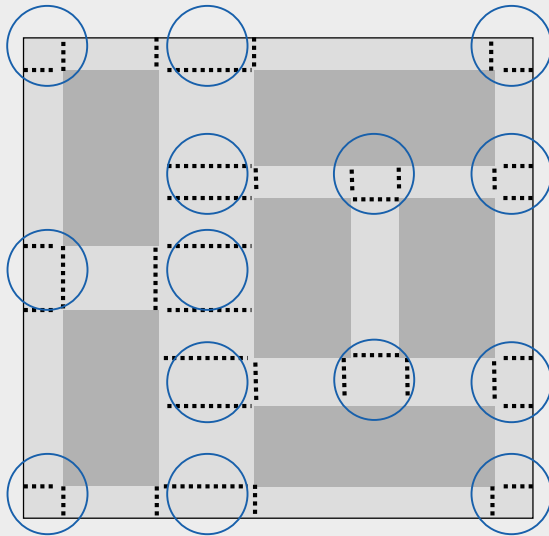


5.6.2 Global Routing in a Connectivity Graph

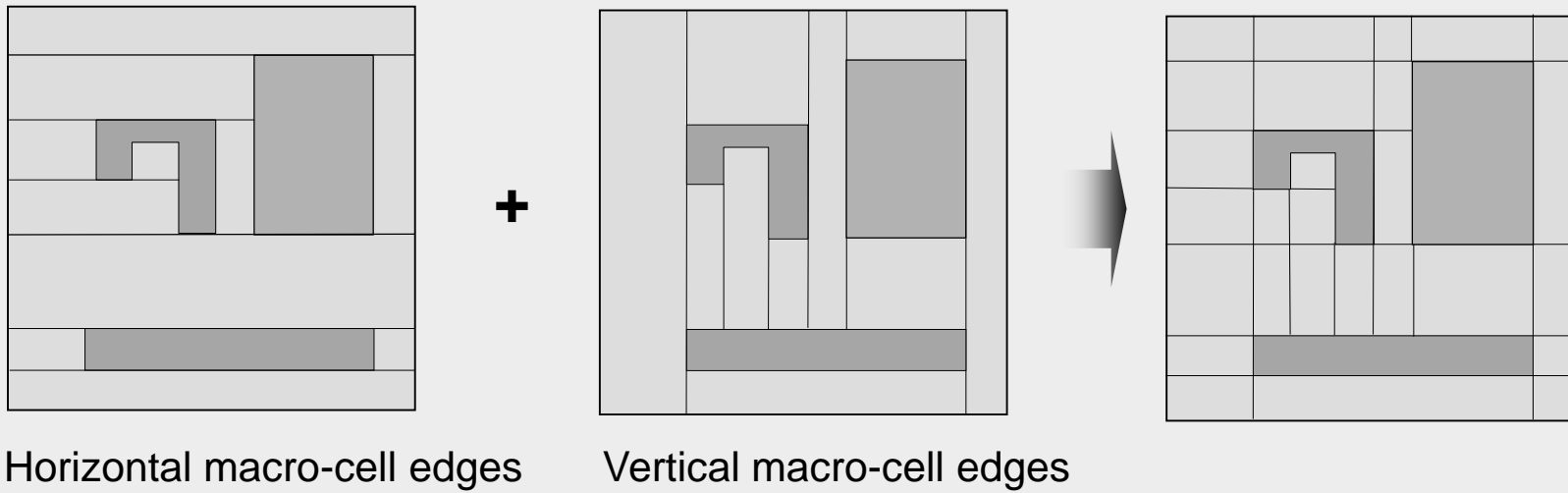
Channel connectivity graph



Switchbox connectivity graph



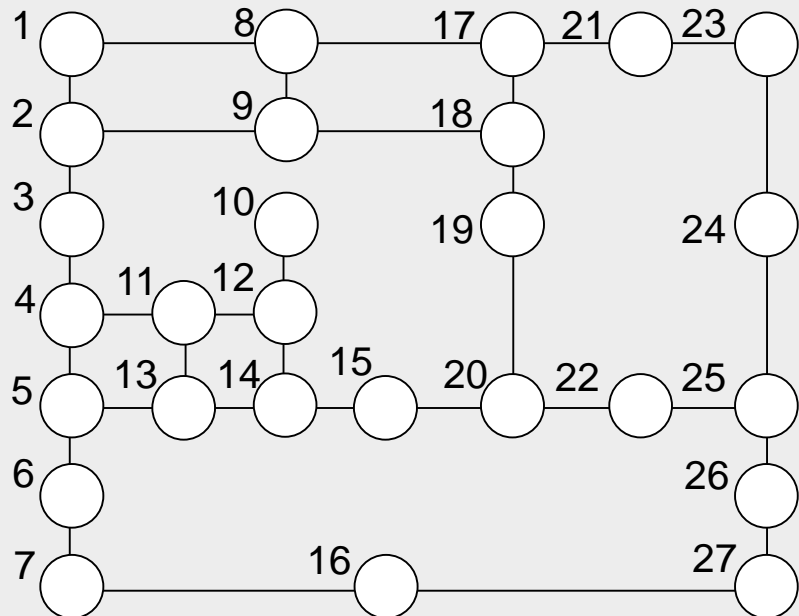
Defining the routing regions



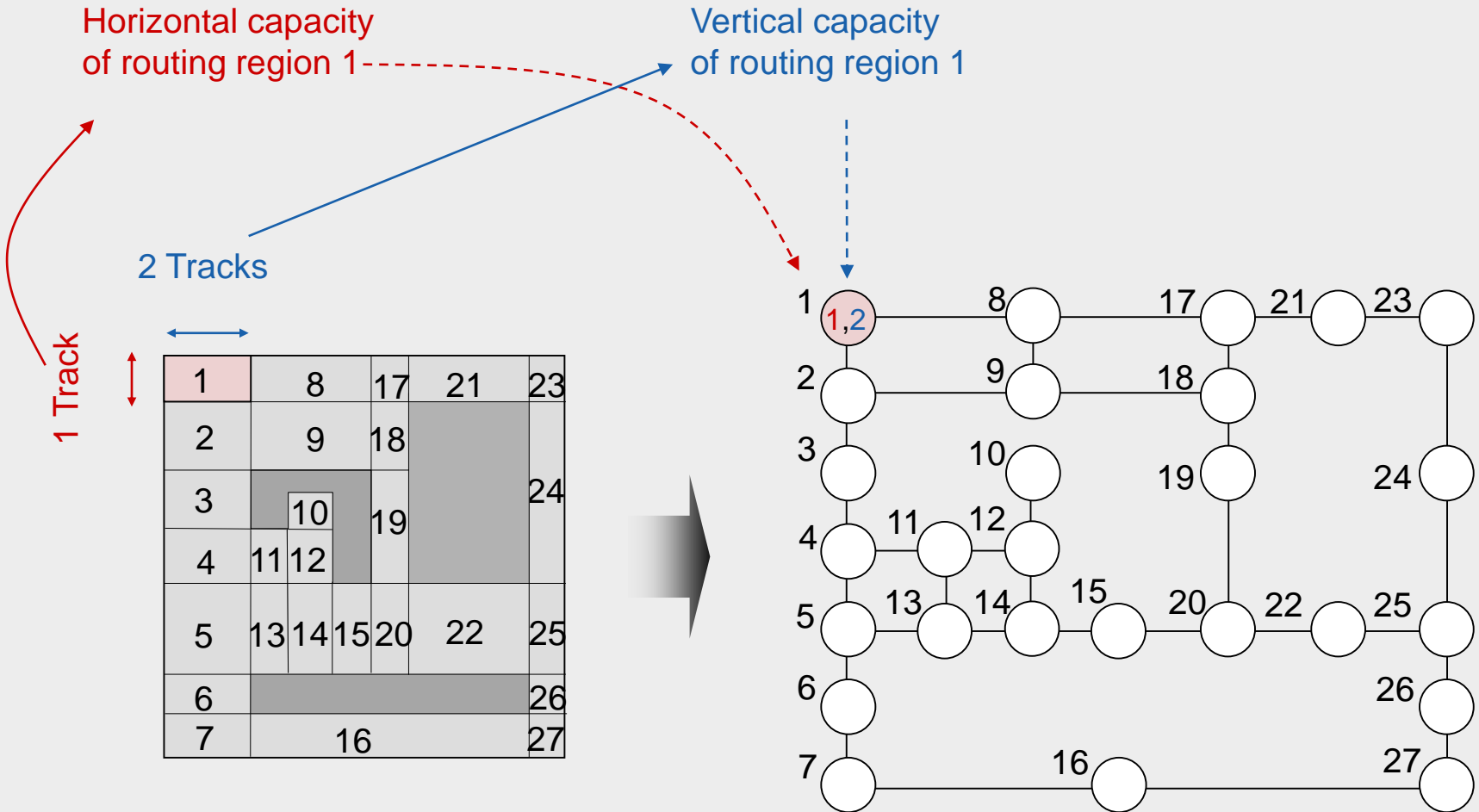
5.6.2 Global Routing in a Connectivity Graph

Defining the connectivity graph

1	8	17	21	23		
2	9	18				
3	10	19		24		
4	11	12				
5	13	14	15	20	22	25
6						26
7	16					27

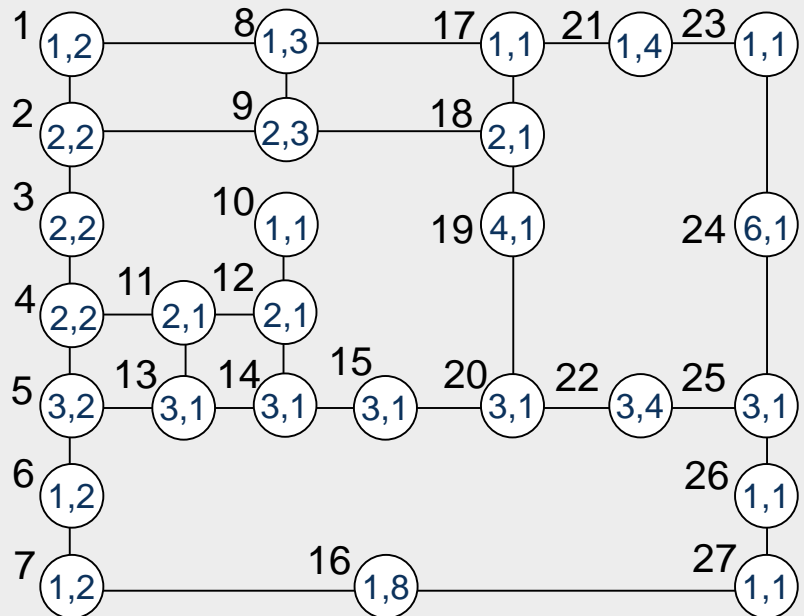


5.6.2 Global Routing in a Connectivity Graph



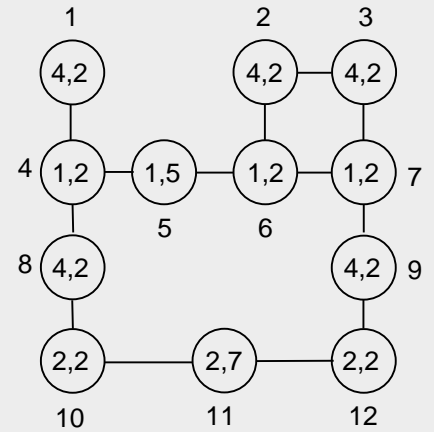
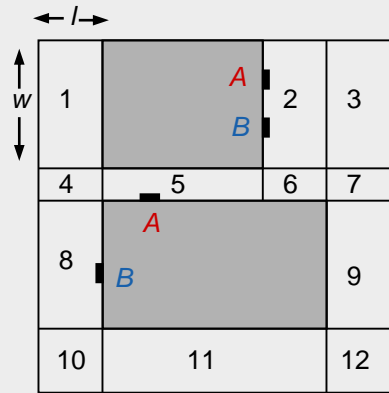
5.6.2 Global Routing in a Connectivity Graph

1	8	17	21	23
2	9	18		
3				24
4	11	12		
5	13	14	15	20
6				
7		16		



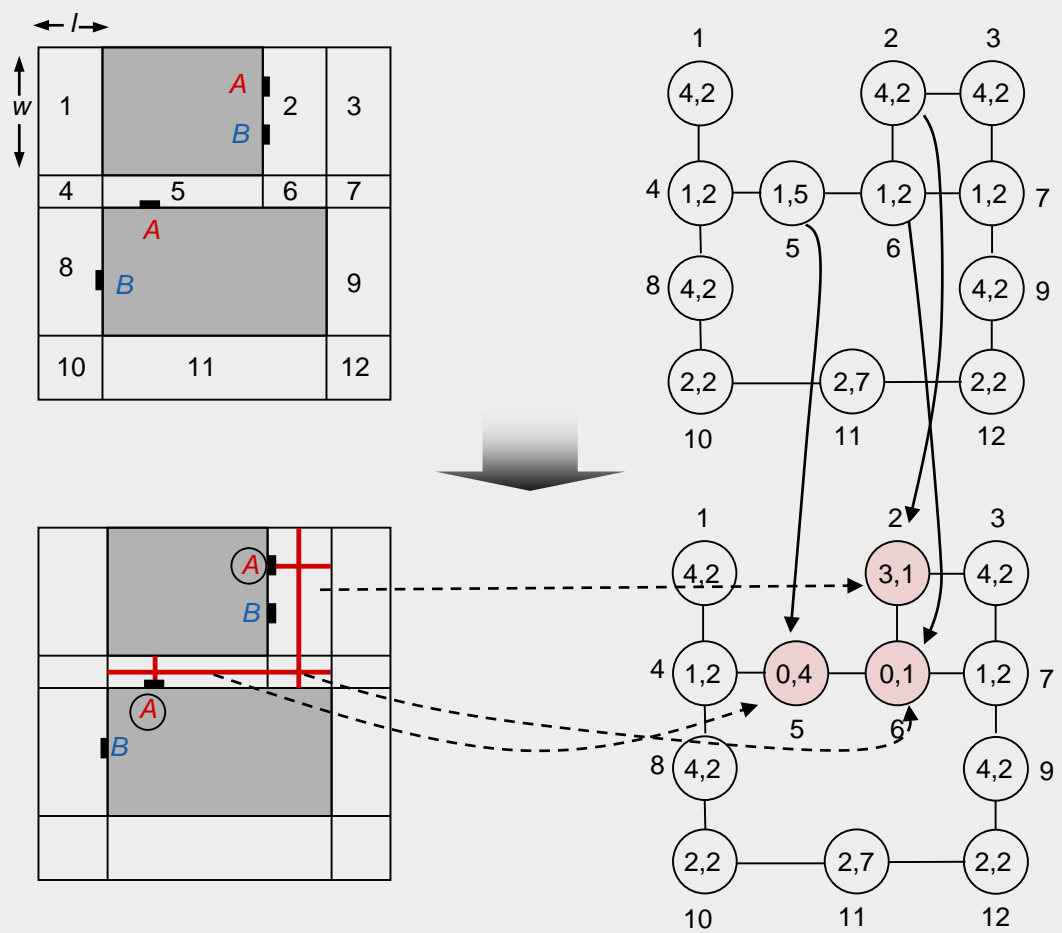
Example

Global routing of the nets **A-A** and **B-B**



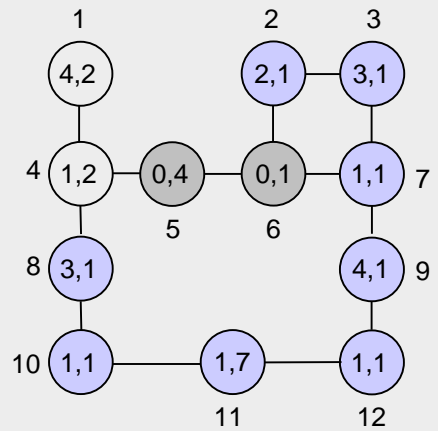
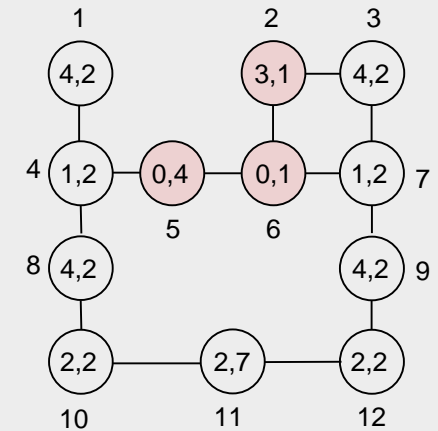
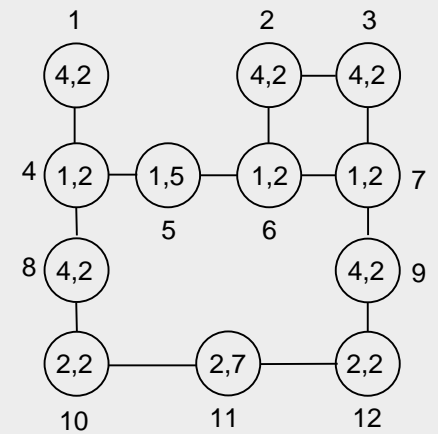
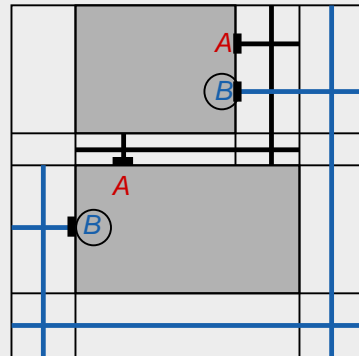
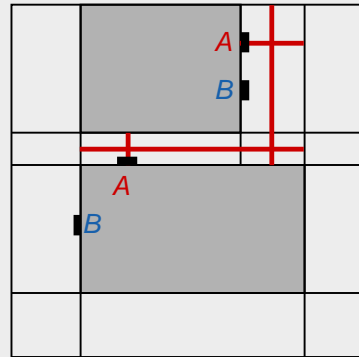
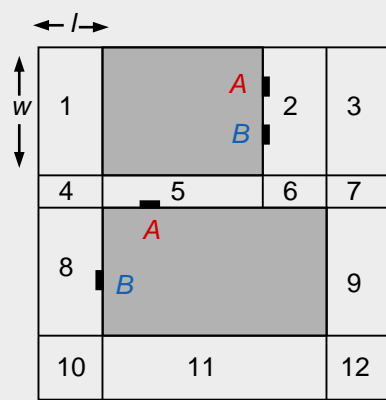
Example

Global routing of the nets **A-A** and **B-B**



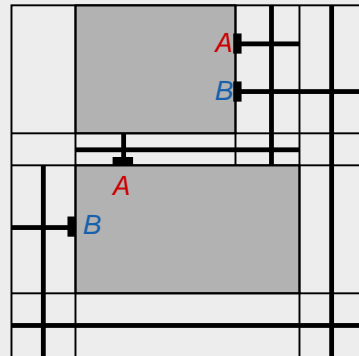
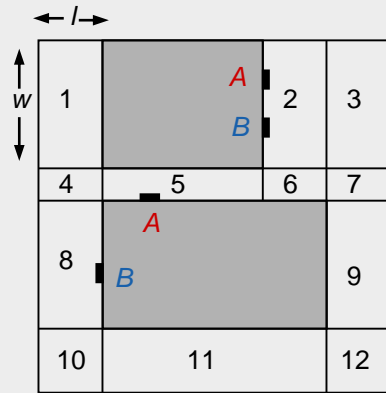
Example

Global routing of the nets **A-A** and **B-B**

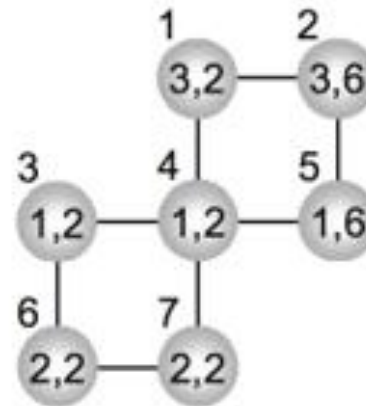
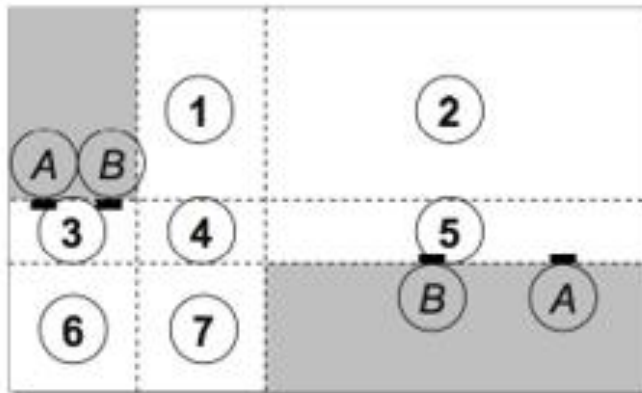


Example

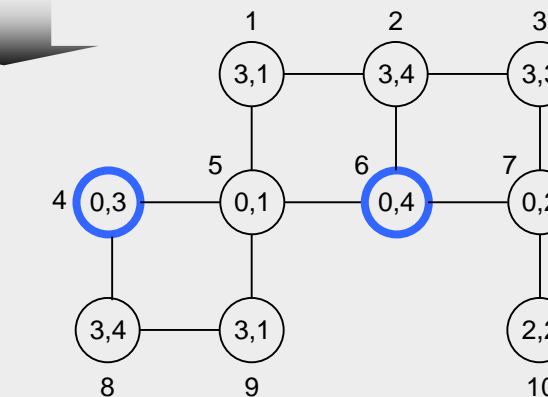
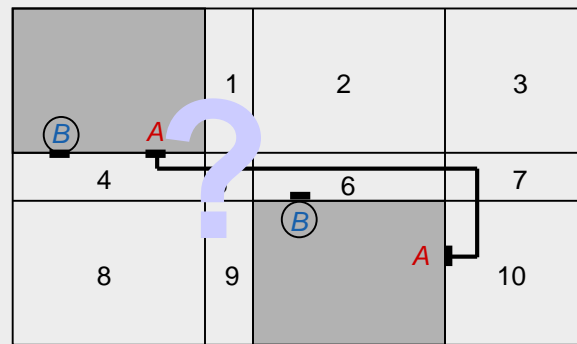
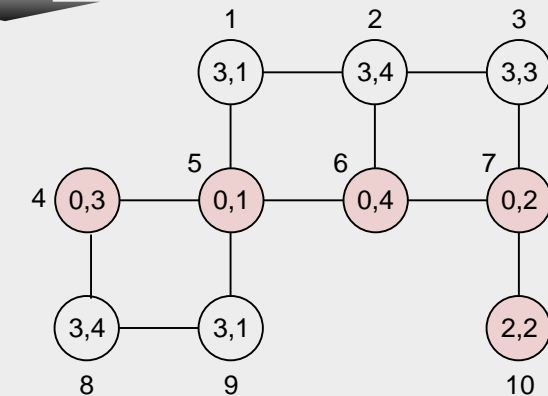
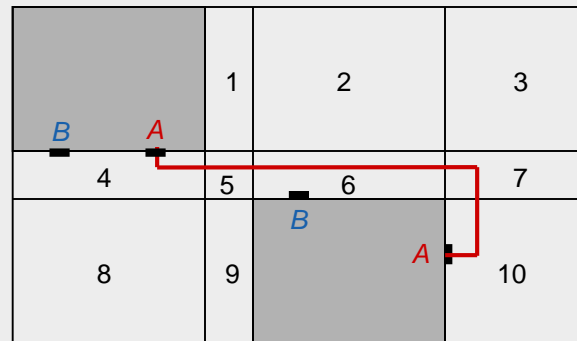
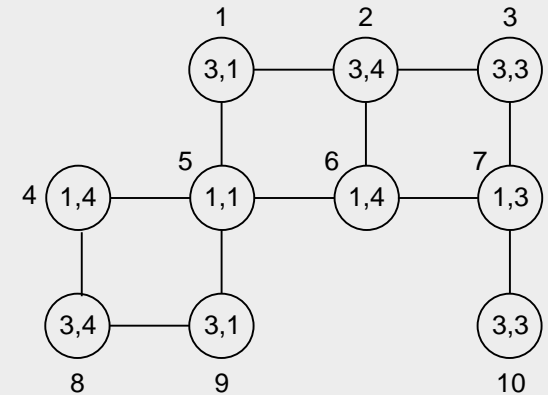
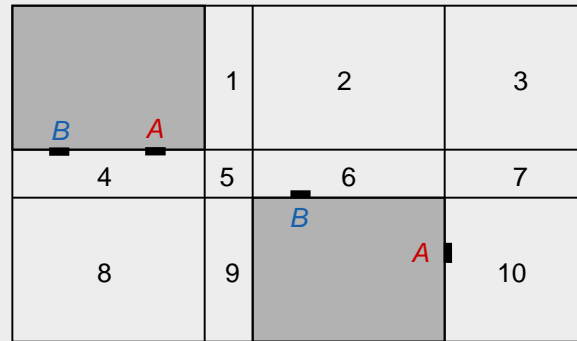
Global routing
of the nets **A-A** and **B-B**



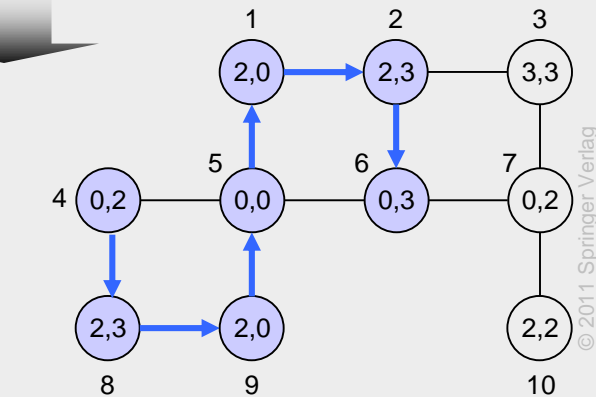
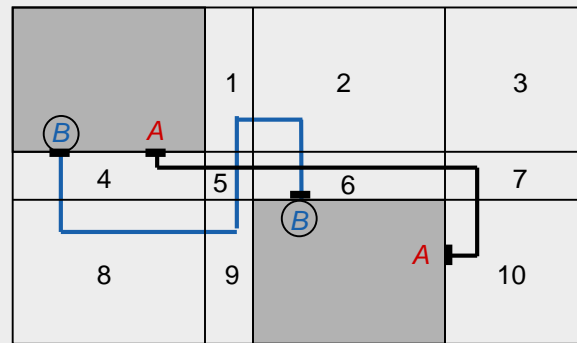
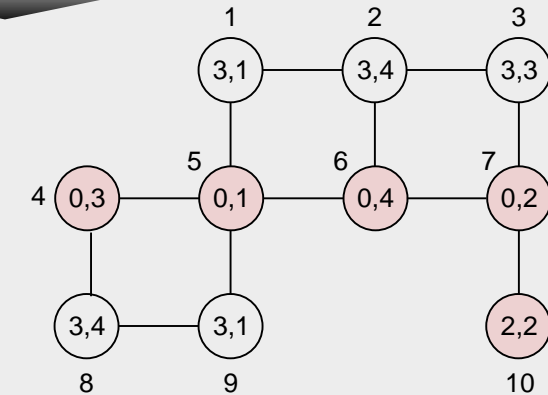
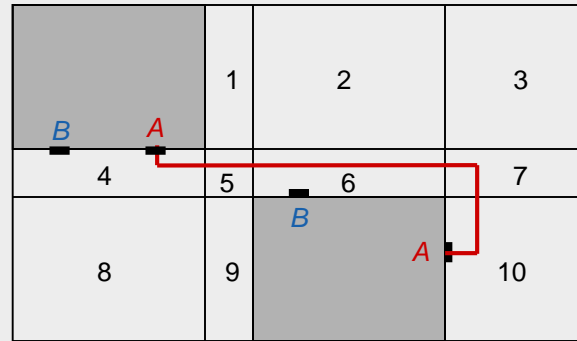
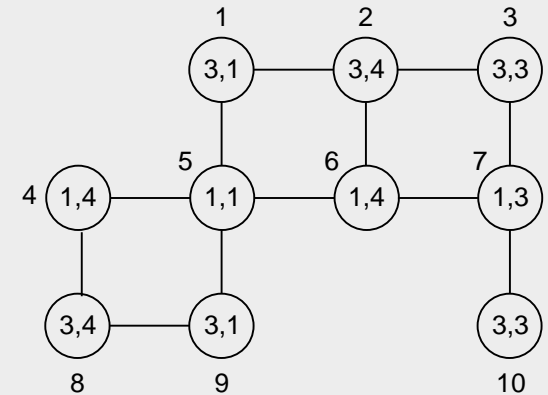
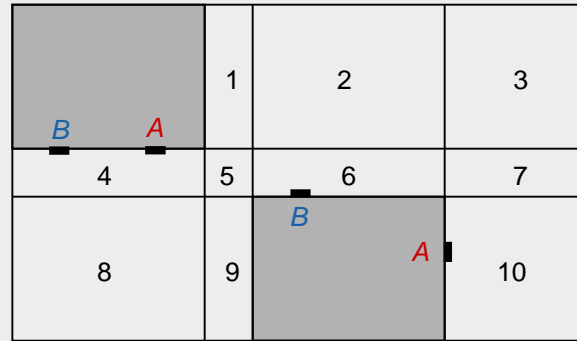
Quiz 10



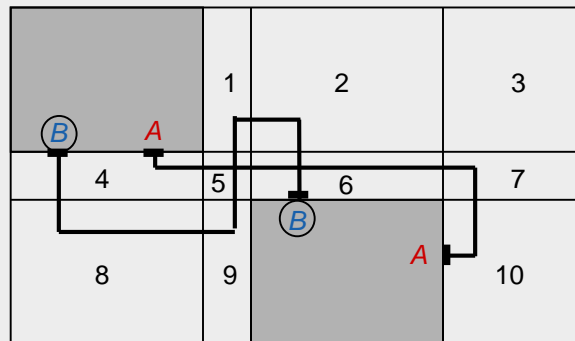
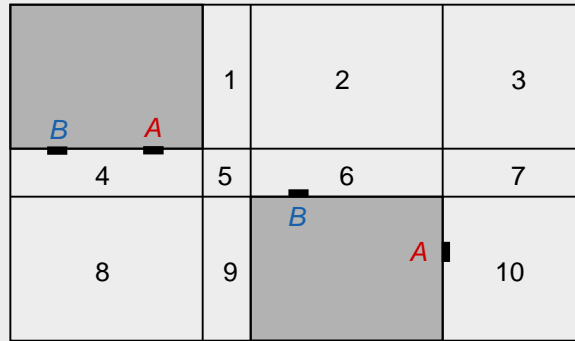
Example
Determine
routability
of a placement



Example
Determine
routability
of a placement














Example
 Determine
 routability
 of a placement



Single Net Routing Algorithms

- Lee's maze routing algorithm
- Maze routing enhancements
- Line search algorithms
- Routing nets with multiple terminals
- Dijkstra's algorithm
- A* search

Lee's Maze Routing Algorithm

9	8	7	6	7	8	9	10	11	12	13	14		
8	7	6	5	6	7	8	9	10	11	12	13	14	
7	6	5	4	5	6	7	8	9	10	11	12	13	14
6	5	4	3	4	5	6	7	8	9	10	11	12	13
5	4	3	2	3	4	5	6	7	8	9	10	11	12
4	3	2	1	2	3	4	5	6	7	8	9	10	11
3	2	1	S	1	2	3	4	5	6	7	8	9	10
4	3	2	1	2	3	4	5	6	7	8	9	10	11
5	4	3	2	3	4	5	6	7	8	9	10	11	12
6	5	4	3	4	5	6	7	8	9				13
7	6	5	4	5								15	14
8	7	6	5	6		12	13	14	15	T			15
9	8	7	6	7		11	12	13	14	15			
10	9	8	7	8	9	10	11	12	13	14	15		
11	10	9	8	9	10	11	12	13	14	15			
12	11	10	9	10	11	12	13	14	15				
13	12	11	10	11	12	13	14	15					
14	13	12	11	12	13	14	15						

Assumption:

Each grid cell has equal cost

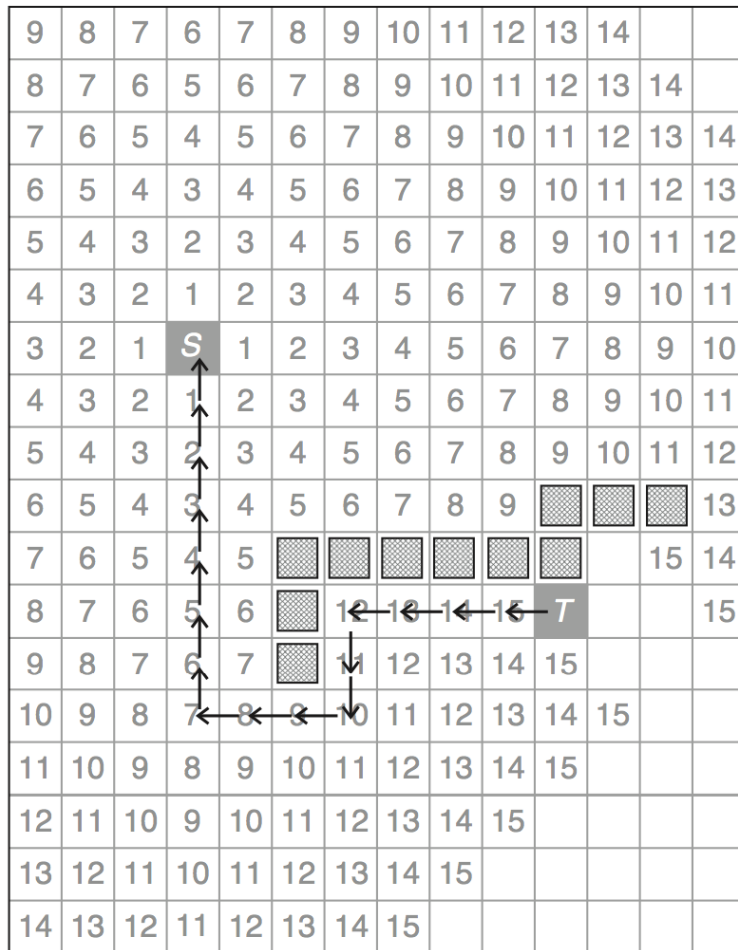
Similar to breadth-first search

Two steps:

1. Expand a wavefront
2. Backtrace

Finds an optimal path

Lee's Maze Routing Algorithm



Assumption:

Each grid cell has equal cost

Similar to breadth-first search

Two steps:

1. Expand a wavefront
2. Backtrace

Finds an optimal path

Hadlock's Min Detour Algorithm

			2	2	2	2	2	2	2			
		2	1	1	1	1	1	1	1	2		
2	1	S	0	0	0	0	0	0	0	1	2	
2	1	0	0	0	0	0	0	0	0	1	2	
2	1	0	0	0	0	0	0	0	0	1	2	
2	1	0	0	0	0	0	0	0	0	█	█	█
2	1	0	0	█	█	█	█	█	█			
2	1	0	0	█	2	2	2	2	T			
		2	1	1	█	2	2	2	2			
			2	2	2	2	2	2	2			

Observation: Shortest-path is the same as the path with min detour value.

Uses the detour number as cell label.

Cells with smaller labels expanded before others.

Finds an optimal path.

Wavefront Comparison

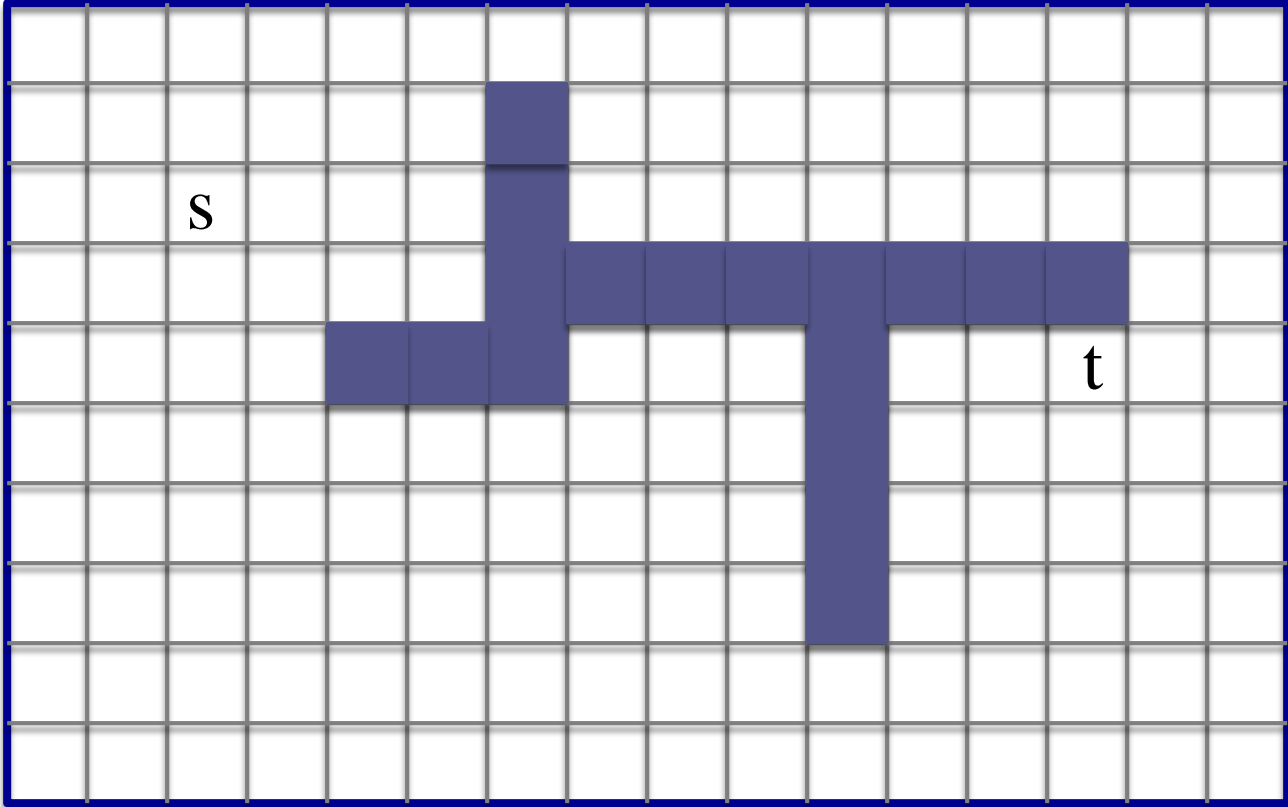
Lee's Maze Router

9	8	7	6	7	8	9	10	11	12	13	14		
8	7	6	5	6	7	8	9	10	11	12	13	14	
7	6	5	4	5	6	7	8	9	10	11	12	13	14
6	5	4	3	4	5	6	7	8	9	10	11	12	13
5	4	3	2	3	4	5	6	7	8	9	10	11	12
4	3	2	1	2	3	4	5	6	7	8	9	10	11
3	2	1	S	1	2	3	4	5	6	7	8	9	10
4	3	2	1	2	3	4	5	6	7	8	9	10	11
5	4	3	2	3	4	5	6	7	8	9	10	11	12
6	5	4	3	4	5	6	7	8	9	█	█	█	13
7	6	5	4	5	█	█	█	█	█	█		15	14
8	7	6	5	6	█	12	13	14	15	T			15
9	8	7	6	7	█	11	12	13	14	15			
10	9	8	7	8	9	10	11	12	13	14	15		
11	10	9	8	9	10	11	12	13	14	15			
12	11	10	9	10	11	12	13	14	15				
13	12	11	10	11	12	13	14	15					
14	13	12	11	12	13	14	15						

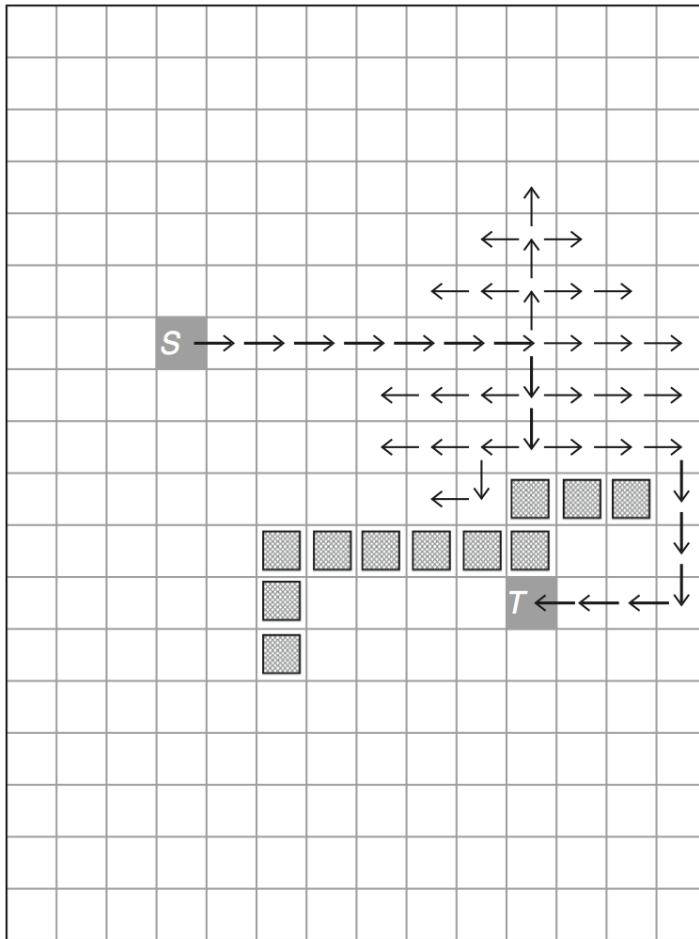
Hadlock's Min Detour Router

						2	2	2	2	2	2	2	2				
						2	1	1	1	1	1	1	1	2			
						2	1	S	0	0	0	0	0	1	2		
						2	1	0	0	0	0	0	0	0	1	2	
						2	1	0	0	0	0	0	0	0	1	2	
						2	1	0	0	0	0	0	0	0	█	█	█
						2	1	0	0	█	█	█	█	█	█		
						2	1	0	0	█	2	2	2	2	T		
										2	2	2	2	2			
										2	2	2	2	2			

Exercise



Soukup's Fast Maze Routing Algorithm








Iteratively conducted in 2 phases:

1. Expand towards target without changing direction until an obstacle is encountered.
2. Expand all directions as in the original maze routing algorithm. When a cell in the direction toward target is found, switch back to phase 1.

Not guaranteed to find optimal path

Maze Routing for Arbitrary Unit Costs






1	1	1	1	1	1
1	1	1	1	1	1
1	1		1	1	1
1	1		1	1	1
1	1		<i>T</i>	1	1
1	<i>S</i>			1	1
2	2	2	2	2	2

Previously assumed: All grid cells have equal costs.

What if different cells have different costs?

Will the original maze routing algorithm work?

Maze Routing for Arbitrary Unit Costs






6	5	6	7		
5	4	5	6	7	
4	3		7		
3	2			11	
2	1		11	10	11
1	0			9	10
2	2	4	6	8	10

Consider Lee's original maze routing algorithm.

This example illustrates the stage when the wavefront from the source reaches the target the first time.

Is this the optimal path?

Maze Routing for Arbitrary Unit Costs

6	5	6	7	8	9
5	4	5	6	7	8
4	3		7	8	9
3	2		8	9	12
2	1		9	10	11
1	0			9	10
2	2	4	6	8	10

Continue expanding after reaching the target.

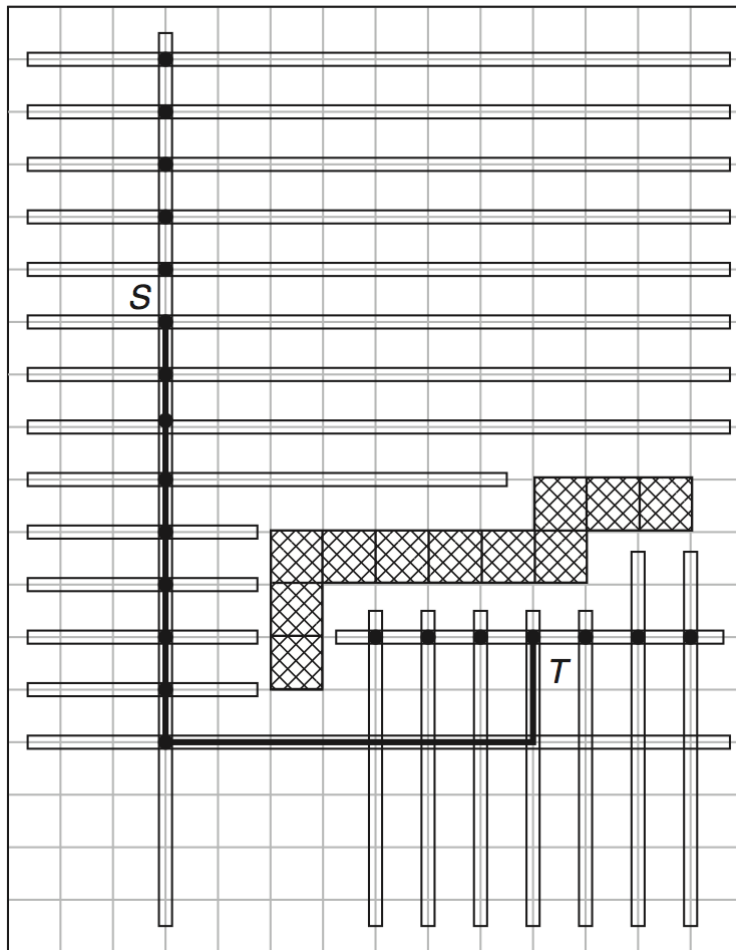
A *longer* path may turn out to have *smaller* cost.

Need to continue expanding after reaching the target.

The issue: We are using BFS on a graph with weighted edges.

→ Use Dijkstra's algorithm instead

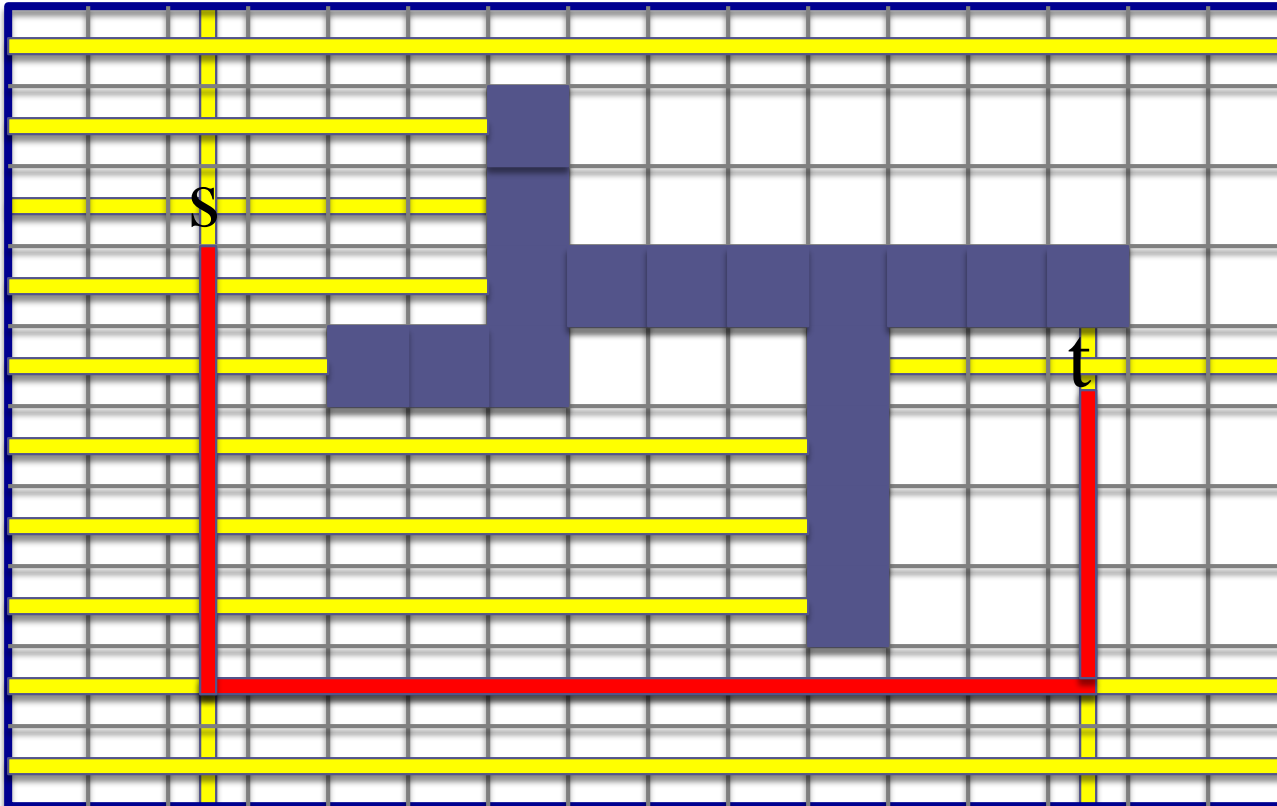
Mikami-Tabuchi's Line Search Algorithm



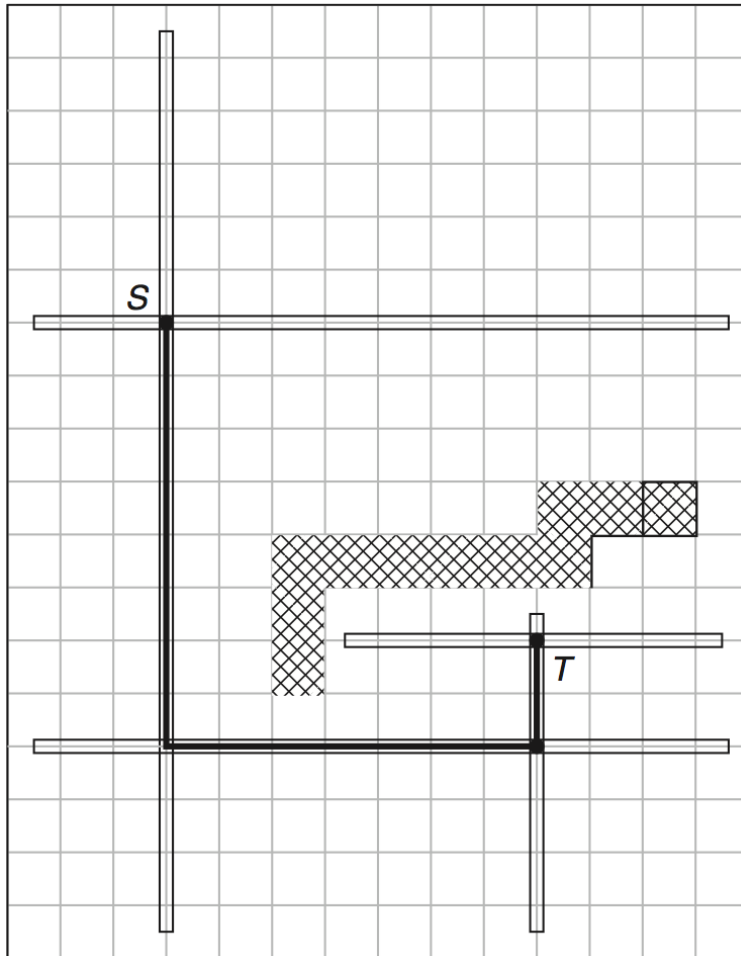
1. Expand a horizontal and vertical line from source and target.
2. In every iteration, expand from the last expanded line.
3. Continue until a line from the source intersects another line from the target.
4. Backtrace from the intersection.

Guaranteed to find min-bend path

Exercise



Hightower's Line Search Algorithm



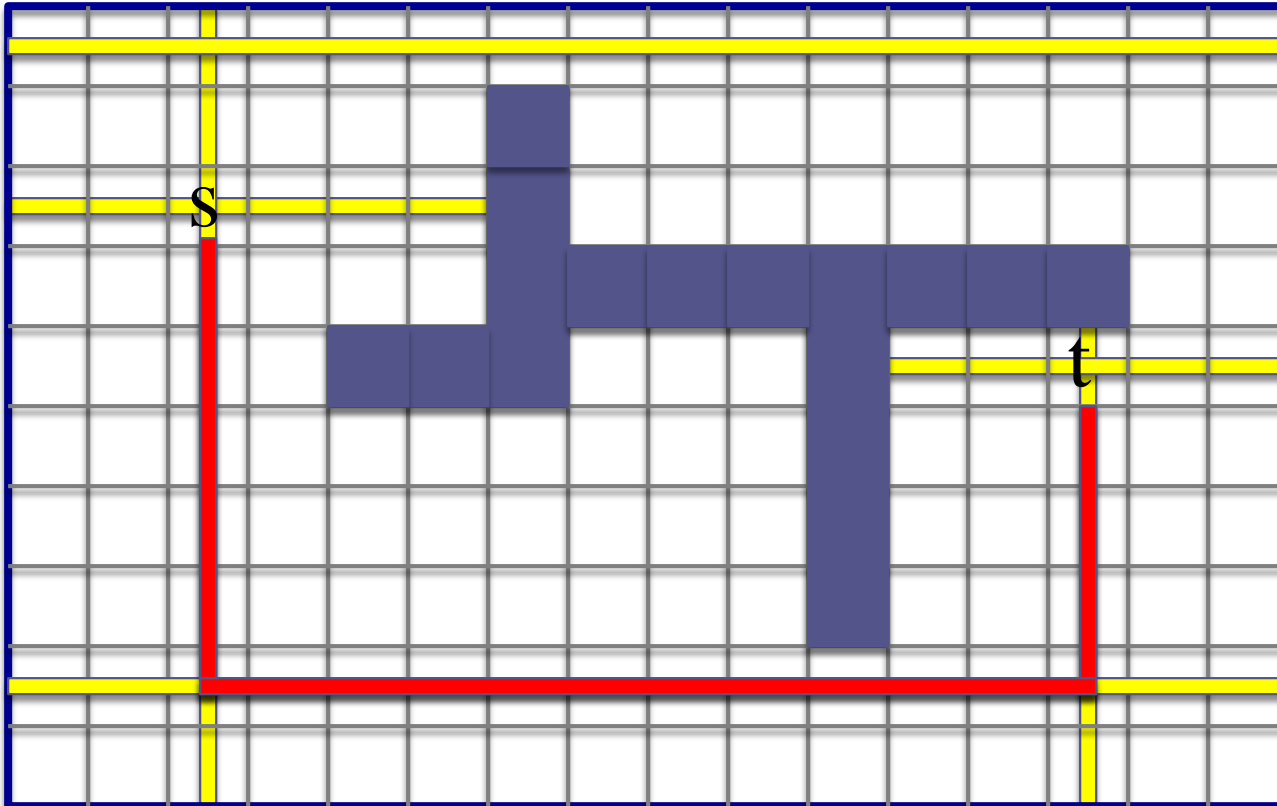
Does not expand from every point.

Identifies “escape lines” based on the positions of the obstacles that blocked the previous line.

Reduces the # of expansions significantly.

Not guaranteed to find a valid path even if one exists.

Exercise



Maze Routing and Line Search Algorithms

Summary

□ Maze routing:

- A variation of breadth-first search (BFS)
- Worst case complexity when all costs are uniform: $O(N \times M)$
where $N \times M$ is the grid size
 - *this complexity not guaranteed for arbitrary weights*
- Usually better to use Dijkstra's algorithm for arbitrary weights

□ Line search:

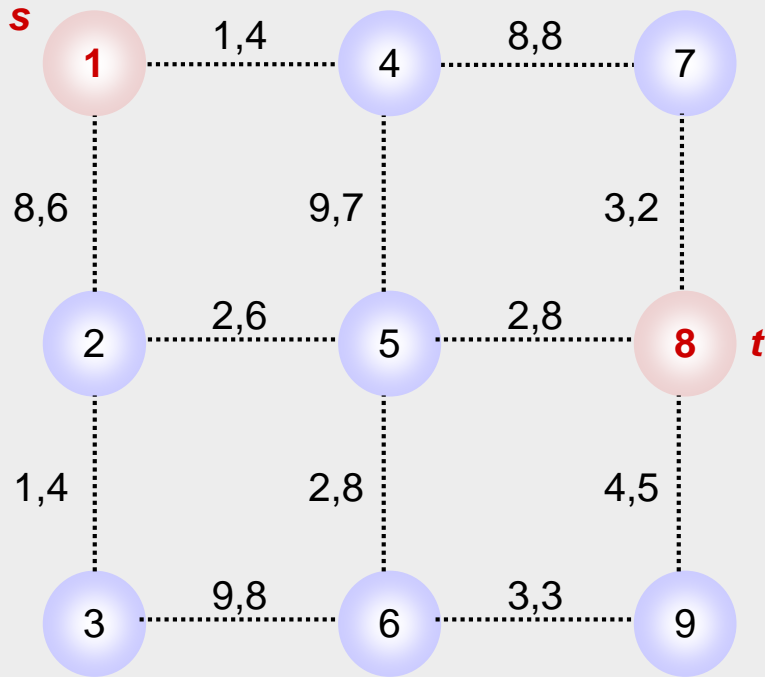
- Doesn't have to visit all grid points
- The runtime complexity depends on the # of bends
- Good when the number of bends in the solution is small
- Good when there are not many blockages in the design

5.6.3 Finding Shortest Paths with Dijkstra's Algorithm

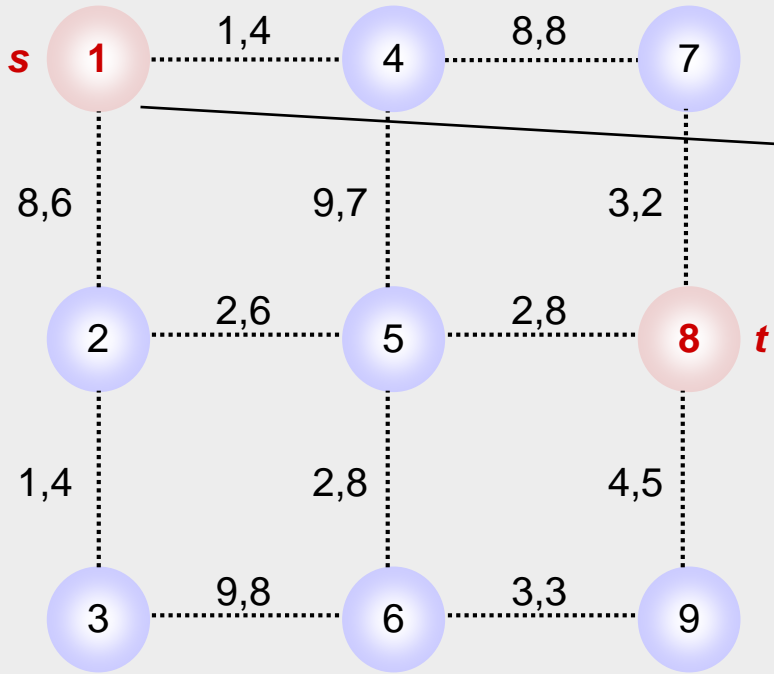
- Finds a shortest path between two specific nodes in the routing graph
- Input
 - graph $G(V,E)$ with non-negative *edge weights* W ,
 - *source* (starting) node s , and
 - *target* (ending) node t
- Maintains three groups of nodes
 - **Group 1** – contains the nodes that have not yet been visited
 - **Group 2** – contains the nodes that have been visited but for which the shortest-path cost from the starting node has not yet been found
 - **Group 3** – contains the nodes that have been visited and for which the shortest path cost from the starting node has been found
- Once t is in Group 3, the algorithm finds the shortest path by backtracing

5.6.3 Finding Shortest Paths with Dijkstra's Algorithm

Example

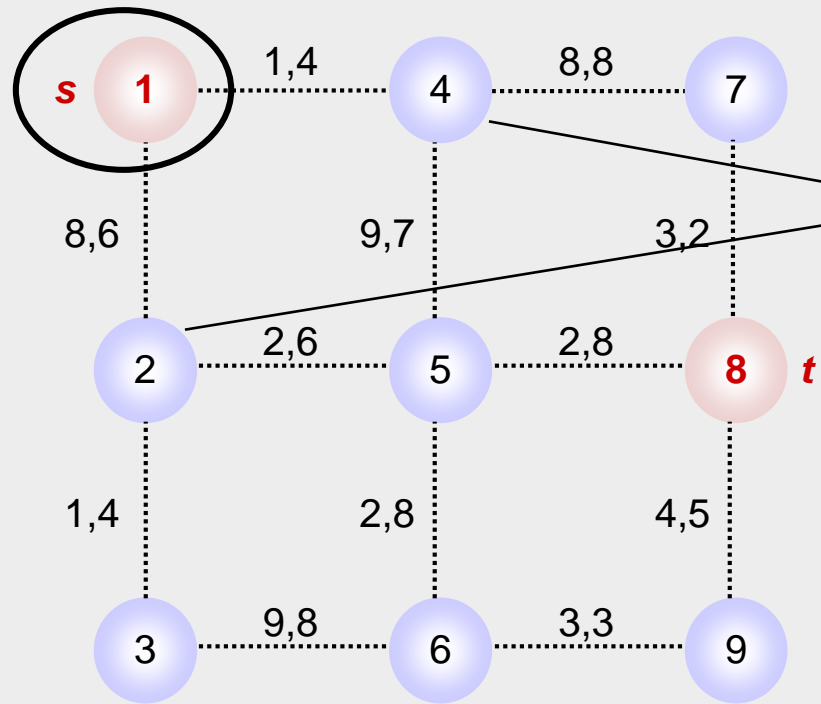


Find the shortest path from source s to target t where the path cost $\sum w_1 + \sum w_2$ is minimal

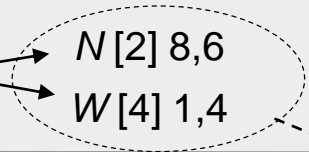


(1)

Current node: 1



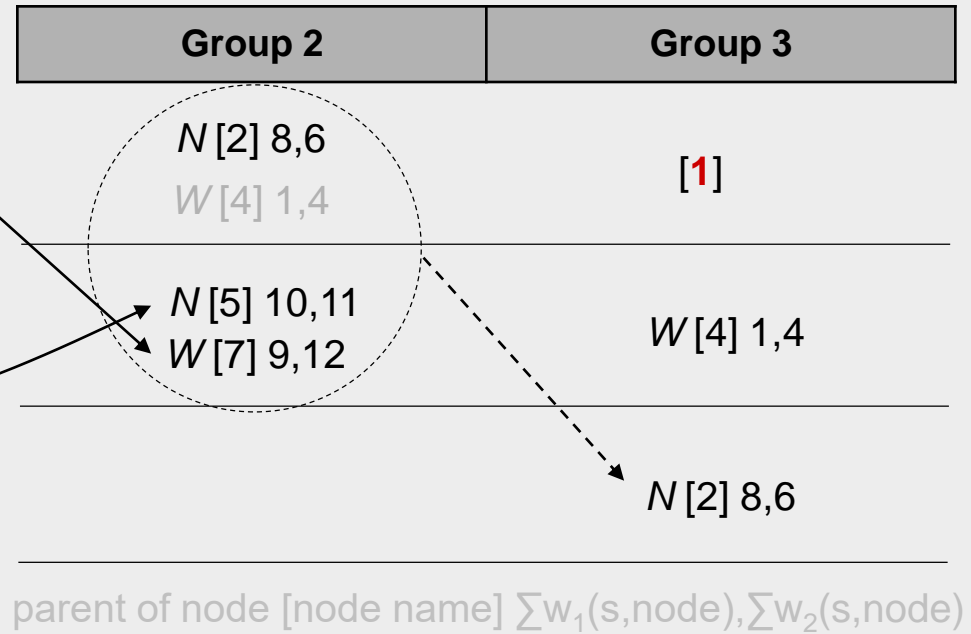
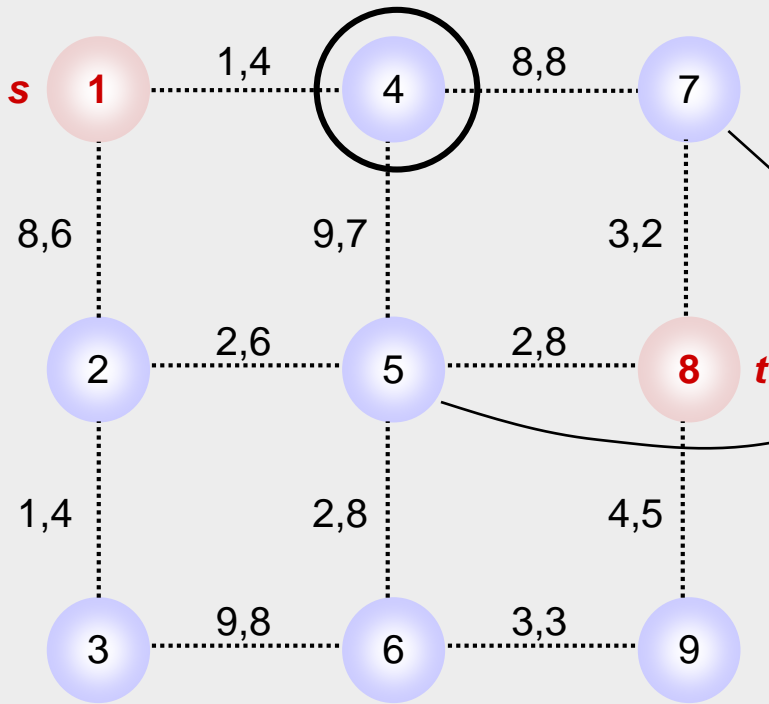
Group 2	Group 3
---------	---------



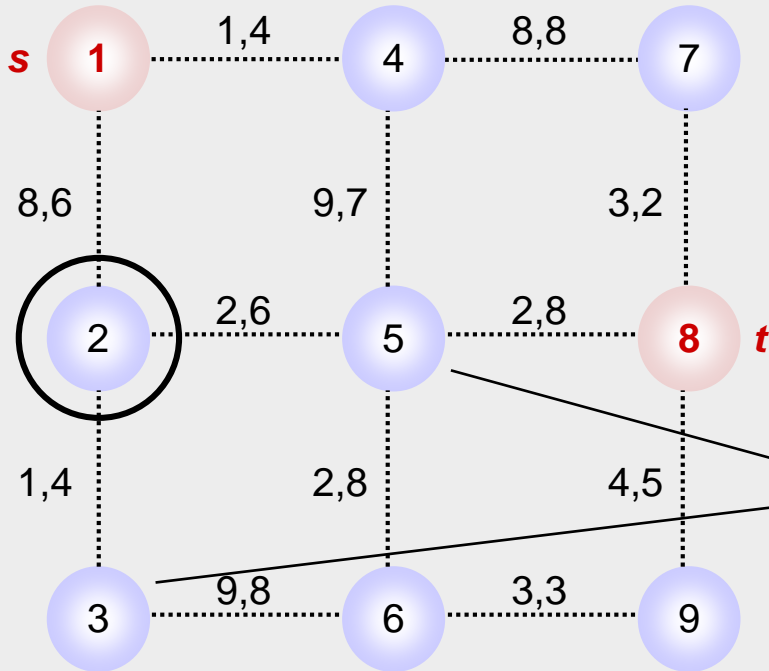
	[1]
	W[4] 1,4

parent of node [node name] $\sum w_1(s, \text{node}), \sum w_2(s, \text{node})$

Current node: 1
 Neighboring nodes: 2, 4
 Minimum cost in group 2: node 4



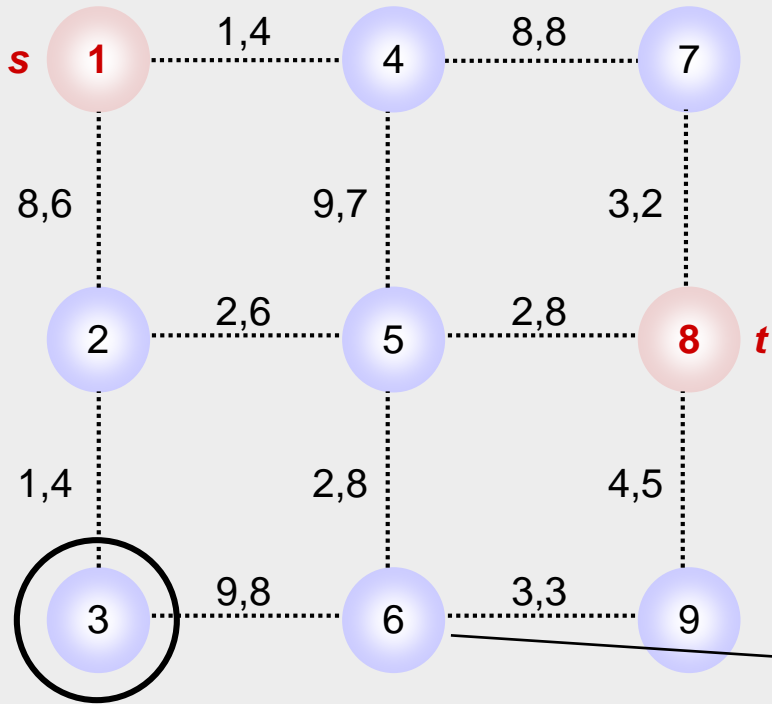
Current node: 4
 Neighboring nodes: 1, 5, 7
 Minimum cost in group 2: node 2



Group 2	Group 3
<i>N</i> [2] 8,6 <i>W</i> [4] 1,4	[1]
<i>N</i> [5] 10,11 <i>W</i> [7] 9,12	<i>W</i> [4] 1,4
<i>N</i> [3] 9,10 <i>W</i> [5] 10,12	<i>N</i> [2] 8,6
	<i>N</i> [3] 9,10

parent of node [node name] $\sum w_1(s,node), \sum w_2(s,node)$

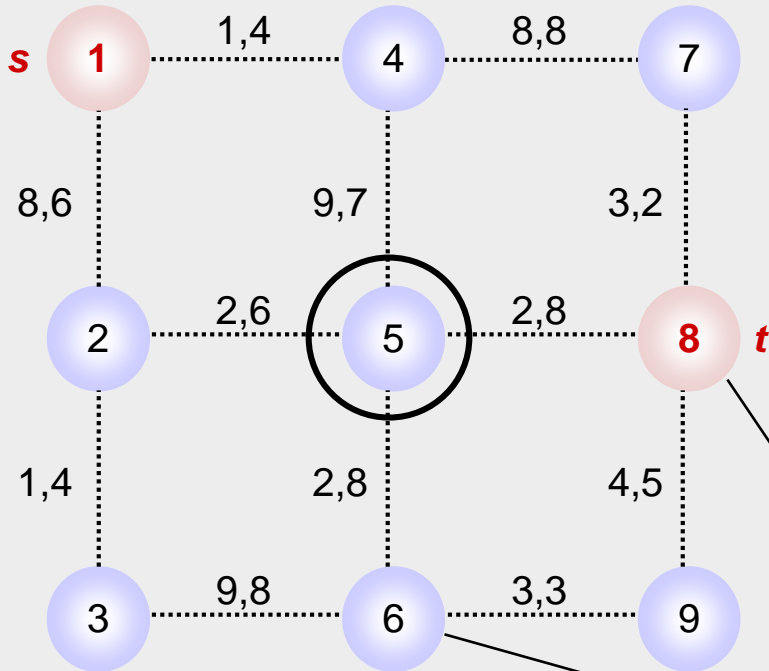
Current node: 2
 Neighboring nodes: 1, 3, 5
 Minimum cost in group 2: node 3



Current node: 3
 Neighboring nodes: 2, 6
 Minimum cost in group 2: node 5

Group 2	Group 3
<i>N</i> [2] 8,6 <i>W</i> [4] 1,4	[1]
<i>N</i> [5] 10,11 <i>W</i> [7] 9,12	<i>W</i> [4] 1,4
<i>N</i> [3] 9,10 <i>W</i>[5] 10,12	<i>N</i> [2] 8,6
<i>W</i> [6] 18,18	<i>N</i> [3] 9,10
	<i>N</i> [5] 10,11

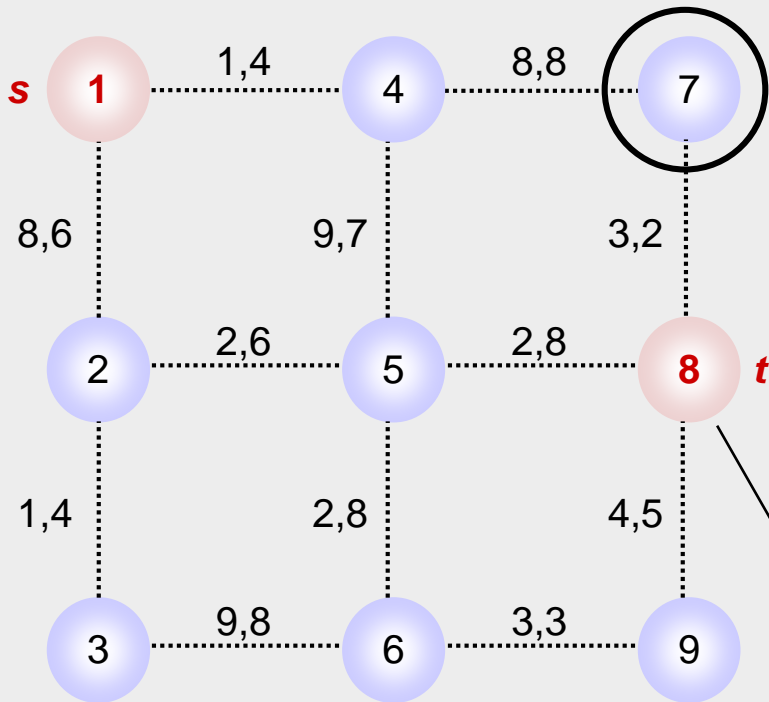
parent of node [node name] $\sum w_1(s, \text{node}), \sum w_2(s, \text{node})$



Current node: 5
 Neighboring nodes: 2, 4, 6, 8
 Minimum cost in group 2: node 7

Group 2	Group 3
$N[2] 8,6$ $W[4] 1,4$	[1]
$N[5] 10,11$ $W[7] 9,12$	$W[4] 1,4$
$N[3] 9,10$ $W[5] 10,12$	$N[2] 8,6$
$W[6] 18,18$	$N[3] 9,10$
$N[6] 12,19$ $W[8] 12,19$	$N[5] 10,11$
	$W[7] 9,12$

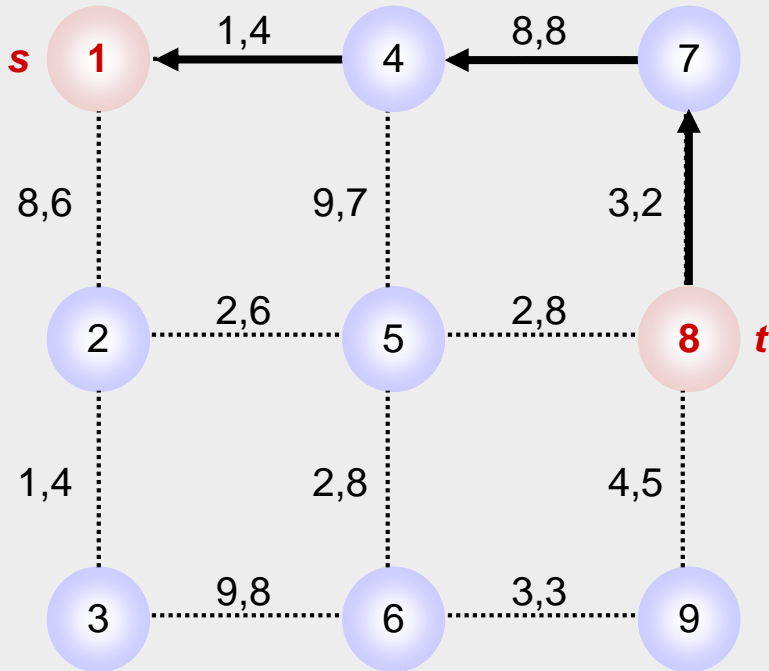
parent of node [node name] $\sum w_1(s,node), \sum w_2(s,node)$



Current node: 7
 Neighboring nodes: 4, 8
 Minimum cost in group 2: node 8

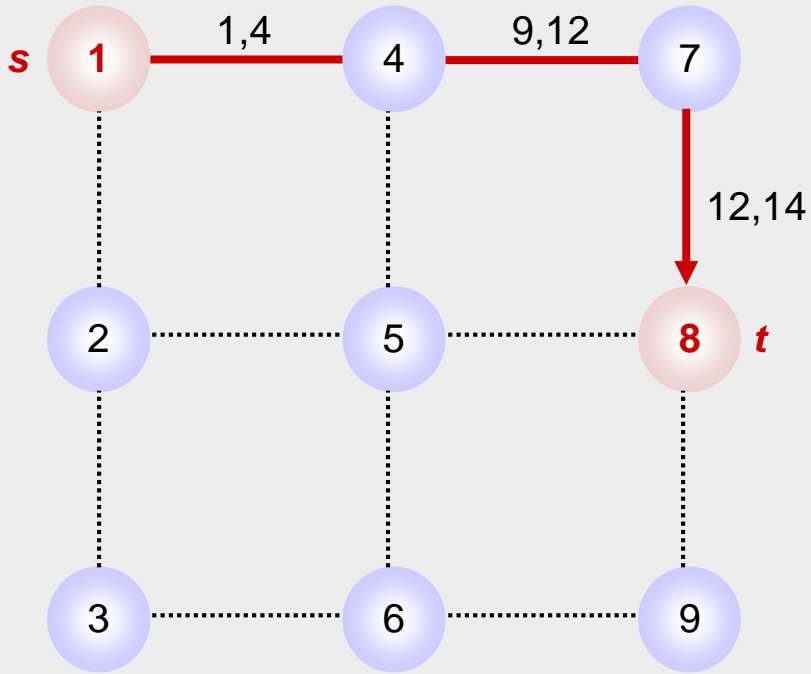
Group 2	Group 3
<i>N</i> (2) 8,6 <i>W</i> (4) 1,4	(1)
<i>N</i> (5) 10,11 <i>W</i> (7) 9,12	<i>W</i> (4) 1,4
<i>N</i> (3) 9,10 <i>W</i>(5) 10,12	<i>N</i> (2) 8,6
<i>W</i> (6) 18,18	<i>N</i> (3) 9,10
<i>N</i> (6) 12,19 <i>W</i>(8) 12,19	<i>N</i> (5) 10,11
<i>N</i>(8) 12,14	<i>W</i> (7) 9,12
	<i>N</i>(8) 12,14

parent of node [node name] $\sum w_1(s, \text{node}), \sum w_2(s, \text{node})$



Retrace from *t* to *s*

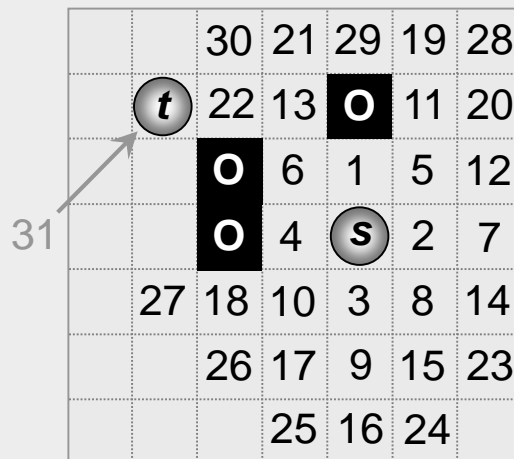
Group 2	Group 3
<i>N</i> (2) 8,6 <i>W</i> (4) 1,4	(1)
<i>N</i> (5) 10,11 <i>W</i> (7) 9,12	<i>W</i> (4) 1,4
<i>N</i> (3) 9,10 <i>W</i>(5) 10,12	<i>N</i> (2) 8,6
<i>W</i> (6) 18,18	<i>N</i> (3) 9,10
<i>N</i> (6) 12,19 <i>W</i>(8) 12,19	<i>N</i> (5) 10,11
<i>N</i> (8) 12,14	<i>W</i> (7) 9,12
	<i>N</i> (8) 12,14



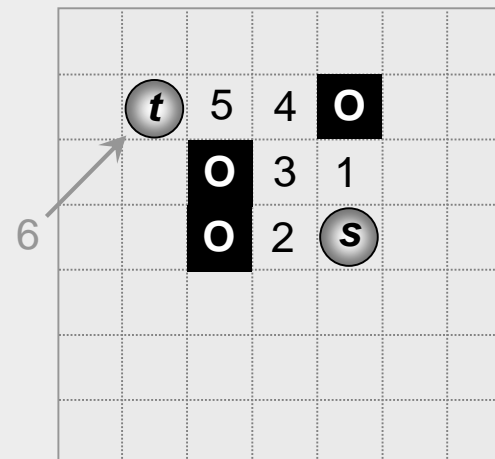
Optimal path 1-4-7-8 from s to t
with accumulated cost (12,14)

5.6.4 Finding Shortest Paths with A* Search

- **A* search** operates similarly to Dijkstra's algorithm, but extends the cost function to include an estimated distance from the current node to the target
- Expands only the most promising nodes; its best-first search strategy eliminates a large portion of the solution space



Dijkstra's algorithm
(exploring 31 nodes)



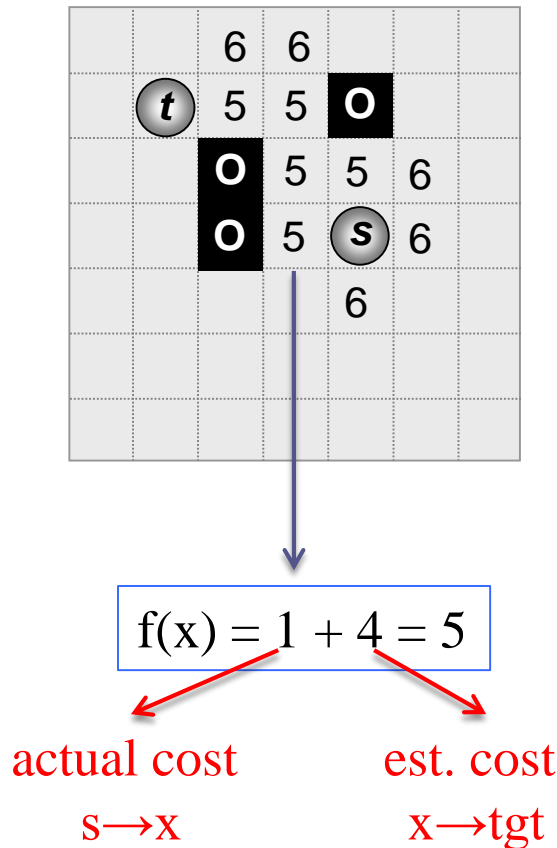
A* search
(exploring 6 nodes)



A* Search

- “*Best-first*” search
- Cost function: $f(x) = g(x) + h(x)$
 - $g(x)$: the cost of the partial path $\text{src} \rightarrow x$
 - $h(x)$: the estimated cost of the remaining path $x \rightarrow \text{tgt}$
 - Note: If $h(x) = 0 \rightarrow$ same as Dijkstra’s algorithm*
- Optimality: If $h(x)$ is admissible then the path computed is guaranteed to be optimal
 - $h(x)$ is admissible if it does not overestimate the cost

A* Search: Example



Assume each grid cell has $\text{cost} = 1$

An easy choice for $h(x)$: **the Manhattan distance between x and t gt**

→ Guaranteed to be a lower bound

Another practical heuristic:

For identical $f(x)$ values, break ties based on $h(x)$

Optimality of A* Search

- $h(x)$ is admissible: $h(x)$ does not overestimate the cost to target
- Optimality proof:

When A* terminates its search at node t :

What is the cost of the path found?

$$f(t) = g(t)$$

Can there be another node y that has $f(y) < f(t)$?

No, because we expand from the min cost node

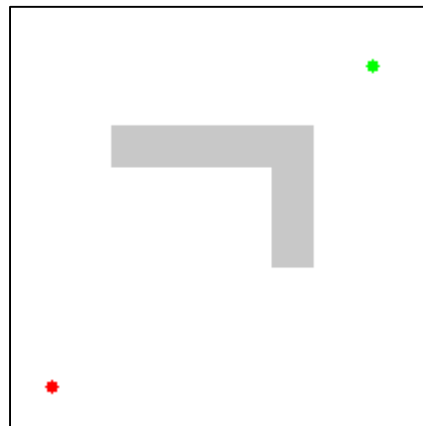
Can there be another node y that can lead to a shorter path?

No, because $f(y) = g(y) + h(y)$, and

$h(y)$ underestimates the path length to target.

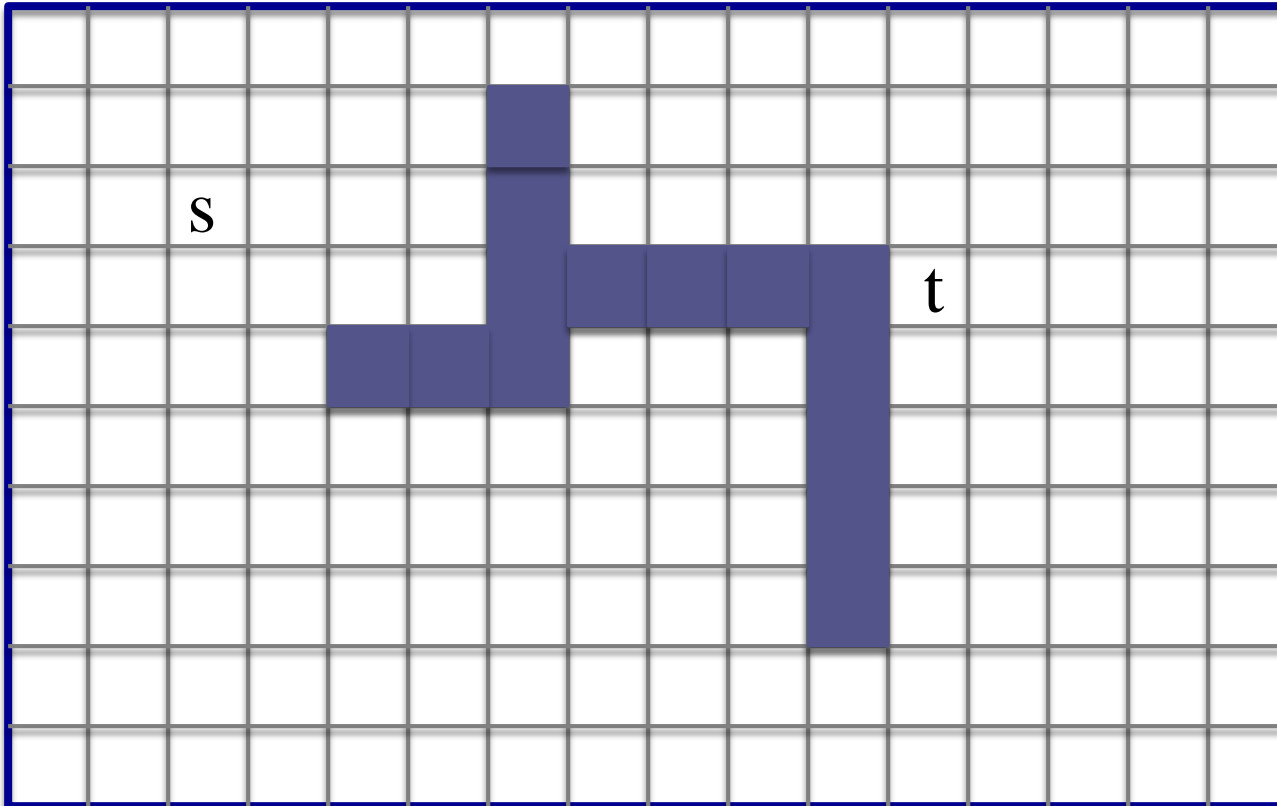
Inadmissible Heuristics

- We may want to use inadmissible heuristics when:
 - ▣ We cannot find an effective admissible heuristics
 - e.g. When the edge weights differ a lot. What if there is a 0-cost edge?
 - ▣ We don't want to explore all nodes with equal $f(x)$ values
- Possible to tradeoff solution quality vs. runtime using inadmissible heuristics




"Weighted A star with eps 5" by Subh83 - Own work. Licensed under CC BY 3.0 via Commons - https://commons.wikimedia.org/wiki/File:Weighted_A_star_with_eps_5.gif#/media/File:Weighted_A_star_with_eps_5.gif

Exercise



5.7 Full-Netlist Routing

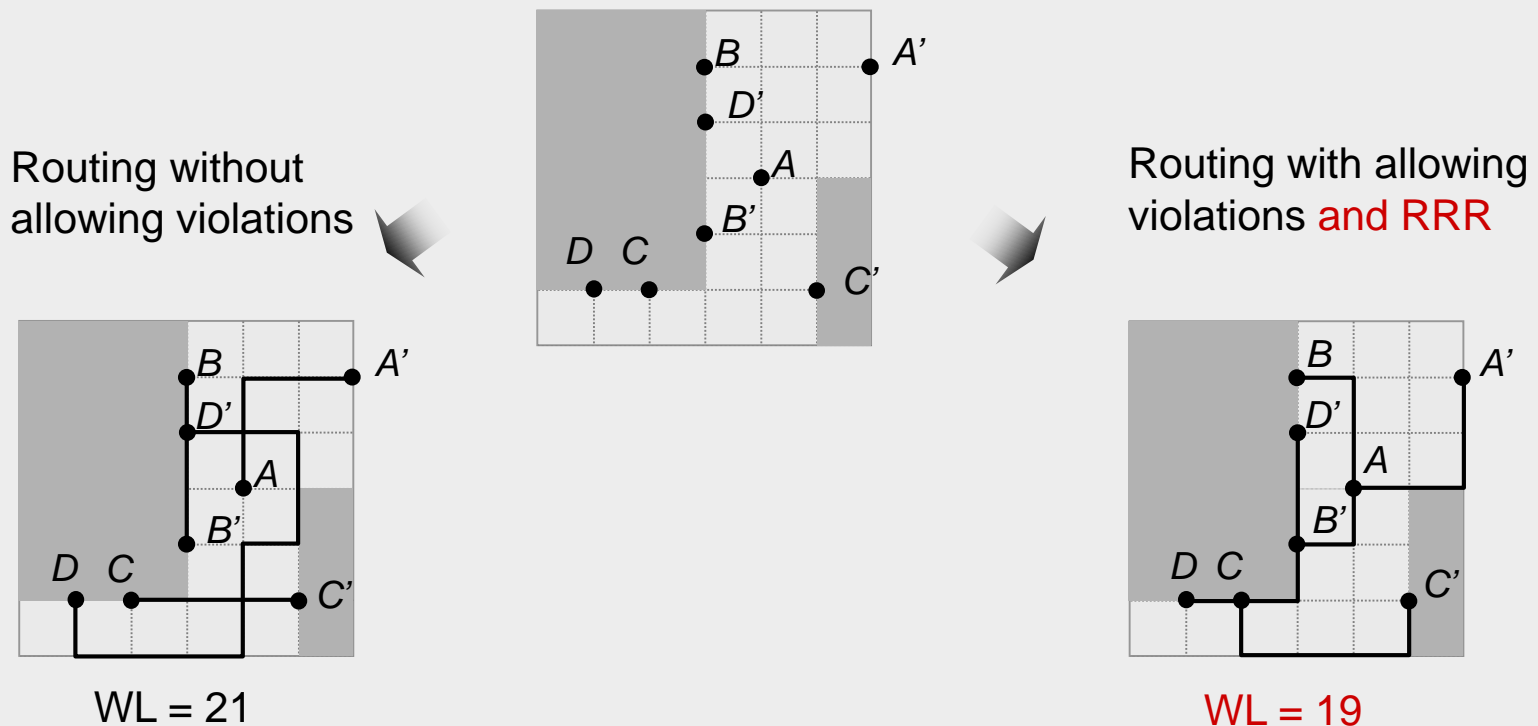
- 5.1 Introduction
- 5.2 Terminology and Definitions
- 5.3 Optimization Goals
- 5.4 Representations of Routing Regions
- 5.5 The Global Routing Flow
- 5.6 Single-Net Routing
 - 5.6.1 Rectilinear Routing
 - 5.6.2 Global Routing in a Connectivity Graph
 - 5.6.3 Finding Shortest Paths with Dijkstra's Algorithm
 - 5.6.4 Finding Shortest Paths with A* Search
-  **5.7 Full-Netlist Routing**
 - 5.7.1 Routing by Integer Linear Programming**
 - 5.7.2 Rip-Up and Reroute (RRR)**
- 5.8 Modern Global Routing
 - 5.8.1 Pattern Routing
 - 5.8.2 Negotiated-Congestion Routing

5.7 Full-Netlist Routing


- Global routers must properly match nets with routing resources, without oversubscribing resources in any part of the chip
- Signal nets are either routed
 - simultaneously, e.g., by **integer linear programming**, or
 - sequentially, e.g., one net at a time
- When certain nets cause resource contention or overflow for routing edges, sequential routing requires multiple iterations: **rip-up and reroute**

5.7.2 Rip-Up and Reroute (RRR)

- **Rip-up and reroute** (RRR) framework: focuses on hard-to-route nets
- Idea: allow temporary violations, so that all nets are routed, but then iteratively remove some nets (**rip-up**), and route them differently (**reroute**)

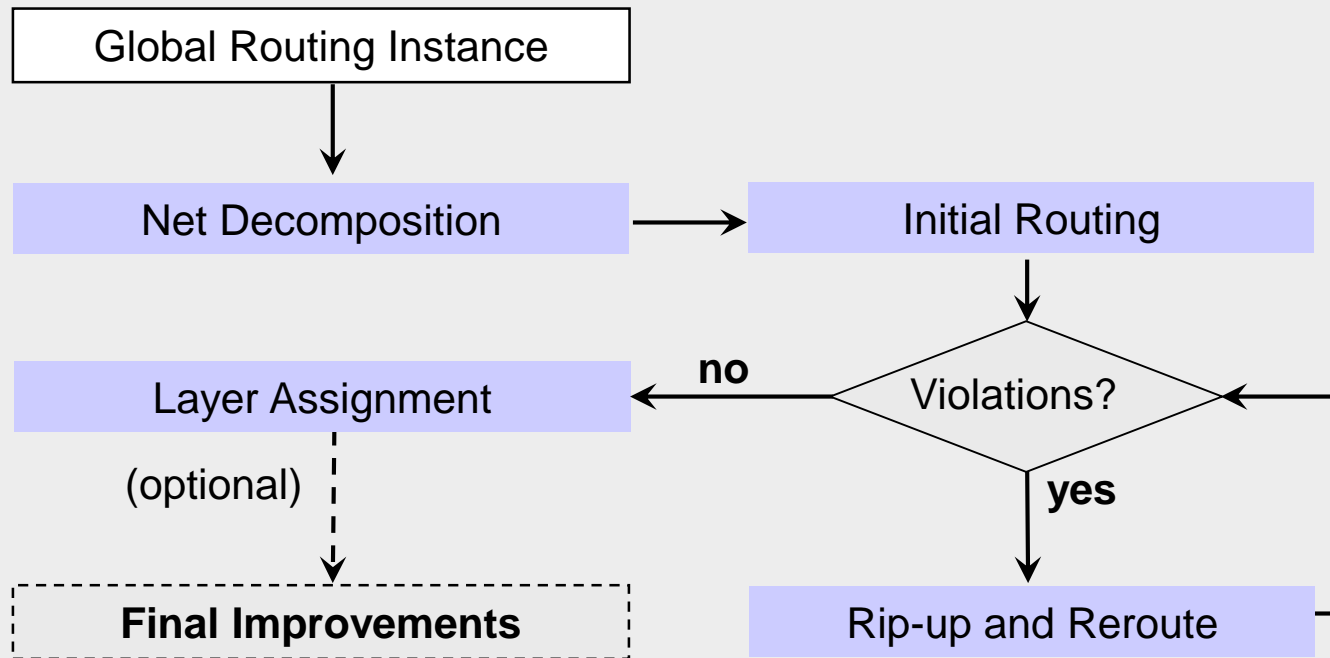


5.8 Modern Global Routing

- 5.1 Introduction
- 5.2 Terminology and Definitions
- 5.3 Optimization Goals
- 5.4 Representations of Routing Regions
- 5.5 The Global Routing Flow
- 5.6 Single-Net Routing
 - 5.6.1 Rectilinear Routing
 - 5.6.2 Global Routing in a Connectivity Graph
 - 5.6.3 Finding Shortest Paths with Dijkstra's Algorithm
 - 5.6.4 Finding Shortest Paths with A* Search
- 5.7 Full-Netlist Routing
 - 5.7.1 Routing by Integer Linear Programming
 - 5.7.2 Rip-Up and Reroute (RRR)
-  **5.8 Modern Global Routing**
 - 5.8.1 Pattern Routing**
 - 5.8.2 Negotiated-Congestion Routing**

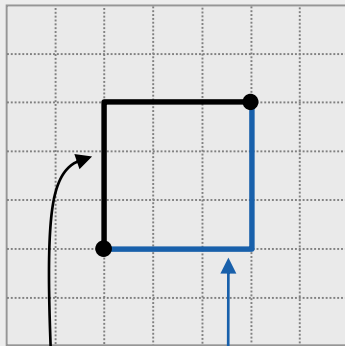
5.8 Modern Global Routing

- General flow for modern global routers, where each router uses a unique set of optimizations:



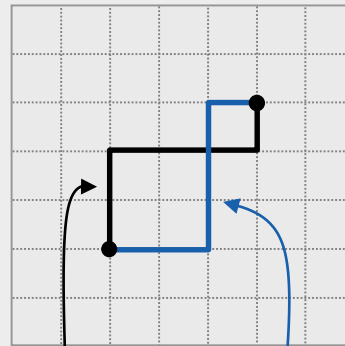
5.8 Modern Global Routing

- **Pattern Routing**
 - Searches through a small number of route patterns to improve runtime
 - Topologies commonly used in pattern routing: *L*-shapes, *Z*-shapes, *U*-shapes



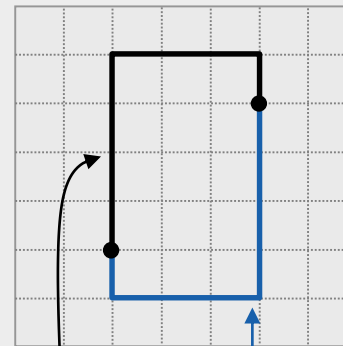
Up-Right
L-Shape

Right-Up
L-Shape



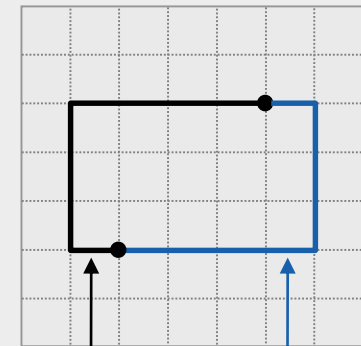
Up-Right-
Up
Z-Shape

Right-Up-
Right
Z-Shape



Detour-Up
Vertical
U-Shape

Detour-
Down
Vertical
U-Shape



Detour-
Left
Horizontal
U-Shape

Detour-
Right
Horizontal
U-Shape

- **Negotiated-Congestion Routing**

- Each edge e is assigned a cost value $cost(e)$ that reflects the demand for edge e
- A segment from net net that is routed through e pays a cost of $cost(e)$
- Total cost of net is the sum of $cost(e)$ values taken over all edges used by net .

$$cost(net) = \sum_{e \in net} cost(e)$$

- The edge cost $cost(e)$ is increased according to the *edge congestion* $\varphi(e)$, defined as the total number of nets passing through e divided by the capacity of e :

$$\varphi(e) = \frac{\eta(e)}{\sigma(e)}$$

- A higher $cost(e)$ value discourages nets from using e and implicitly encourages nets to seek out other, less used edges
- ⇒ Iterative routing approaches (Dijkstra's algorithm, A* search, etc.) find routes with minimum cost while respecting edge capacities

Global Routing

- Input: netlist, placement, obstacles + (usually) routing grid
- Partitions the routing region (chip or block) into global routing cells (gcells)
- Considers the locations of cells within a region as identical
- Plans routes as sequences of gcells
- Minimizes total length of routes and, possibly, routed congestion
- May fail if routing resources are insufficient
 - Variable-die can expand the routing area, so can't usually fail
 - Fixed-die is more common today (cannot resize a block in a larger chip)
- Interpreting failures in global routing
 - Failure with many violations => must restructure the netlist and/or redo global placement
 - Failure with few violations => detailed routing may be able to fix the problems

Detailed Routing

- Input: netlist, placement, obstacles, global routes (on a routing grid), routing tracks, design rules
- Seeks to implement each global route as a sequence of track segments
- Includes layer assignment (unless that is performed during global routing)
- Minimizes total length of routes, subject to design rules

Timing-Driven routing

- Minimizes circuit delay by optimizing timing-critical nets
- Usually needs to trade off route length and congestion against timing
- Both global and detailed routing can be timing-driven

Large-Net Routing

- Nets with many pins can be so complex that routing a single net warrants dedicated algorithms
- Steiner tree construction
 - Minimum wirelength, extensions for obstacle-avoidance
 - Nonuniform routing costs to model congestion
- Large signal nets are routed as part of global routing and then split into smaller segments processed during detailed routing

Clock Tree Routing / Power Routing

- Performed before global routing to avoid competition for resources occupied by signal nets

Summary of Chapter 5 – Routing Single Nets

- Usually ~50% of the nets are two-pin nets, ~25% have three pins, ~12.5% have four, etc.
 - Two-pin nets can be routed as L-shapes or using maze search (in a connectivity graph of the routing regions)
 - Three-pin nets usually have 0 or 1 branching point
 - Larger nets are more difficult to handle
- Pattern routing
 - For each net, considers only a small number of shapes (L, Z, U, T, E)
 - Very fast, but misses many opportunities
 - Good for initial routing, sometimes is sufficient
- Routing pin-to-pin connections
 - Breadth-first-search (when costs are uniform)
 - Dijkstra's algorithm (non-uniform costs)
 - A*-search (non-uniform costs and/or using additional distance information)

Summary of Chapter 5 – Routing Single Nets

- Minimum Spanning Trees and Steiner Minimal Trees in the rectilinear topology (RMSTs and RSMTs)
 - RMSTs can be constructed in near-linear time
 - Constructing RSMTs is NP-hard, but feasible in practice
- Each edge of an RMST or RSMT can be considered a pin-to-pin connection and routed accordingly
- Routing congestion introduces non-uniform costs, complicates the construction of minimal trees (which is why A*-search still must be used)
- For nets with <10 pins, RSMTs can be found using look-up tables (FLUTE) very quickly

Summary of Chapter 5 – Full Netlist Routing

- Routing by Integer Linear Programming (ILP)
 - Capture the route of each net by 0-1 variables, form equations constraining those variables
 - The objective function can represent total route length
 - Solve the equations while minimizing the objective function (ILP software)
 - Usually a convenient but slow technique, may not scale to largest netlists (can be extended by area partitioning)
- Rip-up and Re-route (RRR)
 - Processes one net at a time, usually by A*-search and Steiner-tree heuristics
 - Allows temporary overlaps between nets
 - When every net is routed (with overlaps), it removes (rips up) those with overlaps and routes them again with penalty for overlaps
 - This process may not finish, but often does, else use a time-out
- Both ILP-based routing and RRR can be applied in global and detailed routing
 - ILP-based routing is usually preferable for small, difficult-to-route regions
 - RRR is much faster when routing is easy

Summary of Chapter 5 – Modern Global Routing

- Initial routes are constructed quickly by pattern routing and the FLUTE package for Steiner tree construction - very fast
- Several iterations based on modified pattern routing to avoid congestion - also very fast
 - Sometimes completes all routes without violations
 - If violations remain, they are limited to a few congested spots
- The main part of the router is based on a variant of RRR called Negotiated-Congestion Routing (NCR)
 - Several proposed alternatives are not competitive
- NCR maintains "history" in terms of which regions attracted too many nets
- NCR increases routing cost according to the historical popularity of the regions
 - The nets with alternative routes are forced to take those routes
 - The nets that do not have good alternatives remain unchanged
 - Speed of increase controls tradeoff between runtime and route quality