Chapter 5 – Global Routing

VLSI Physical Design: From Graph Partitioning to Timing Closure

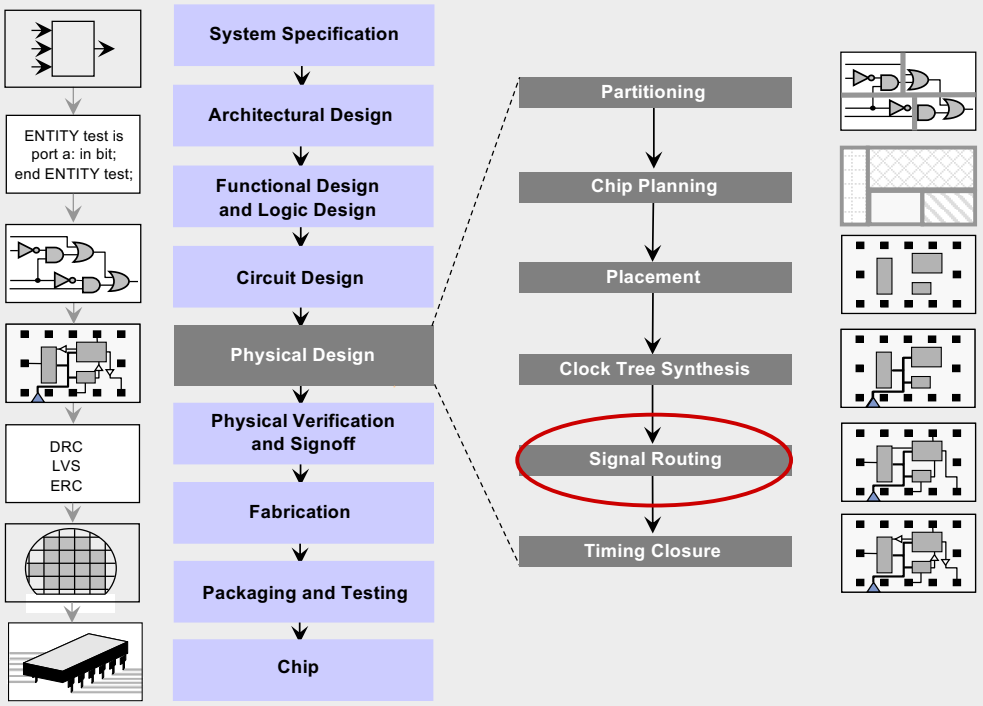
Original Authors:

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

Chapter 5 – Global Routing

© 2011 Springer Verlag

5.1 Introduction



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

Terminology:

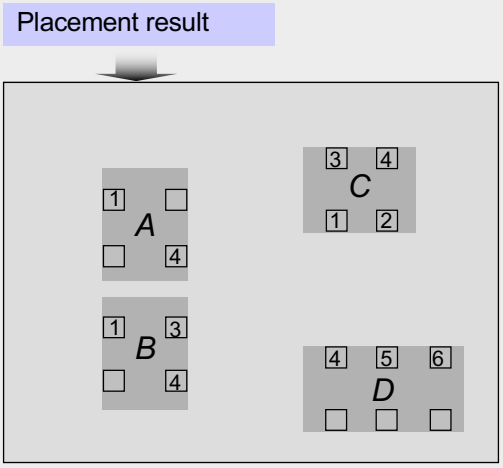
- Net: Set of two or more pins that have the same electric potential
- Netlist: Set of all nets.
- Congestion: Where the shortest routes of several nets are incompatible because they traverse the same tracks.
- Fixed-die routing: Chip outline and routing resources are fixed.
- Variable-die routing: New routing tracks can be added as needed.

5.1 Introduction: General Routing Problem

Netlist:

$N_1 = \{C_4, D_6, B_3\}$
$N_2 = \{D_4, B_4, C_1, A_4\}$
$N_3 = \{C_2, D_5\}$
$N_4 = \{B_1, A_1, C_3\}$

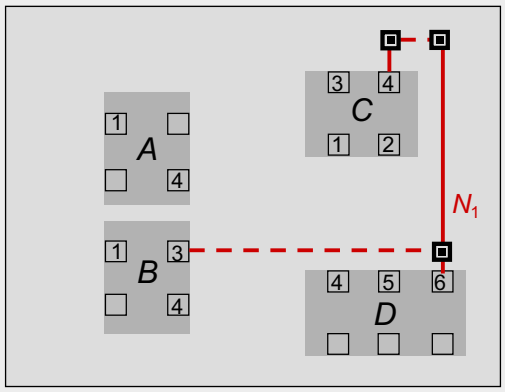
Technology Information (Design Rules)



Netlist:

$N_1 = \{C_4, D_6, B_3\}$
$N_2 = \{D_4, B_4, C_1, A_4\}$
$N_3 = \{C_2, D_5\}$
$N_4 = \{B_1, A_1, C_3\}$

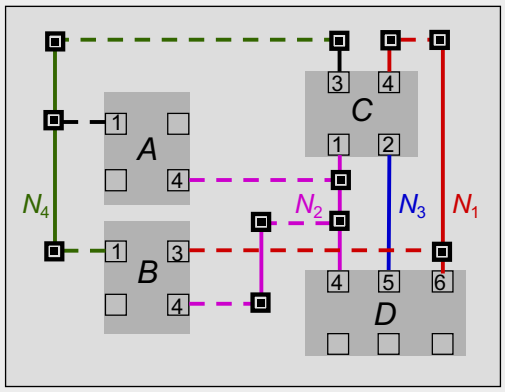
Technology Information (Design Rules)



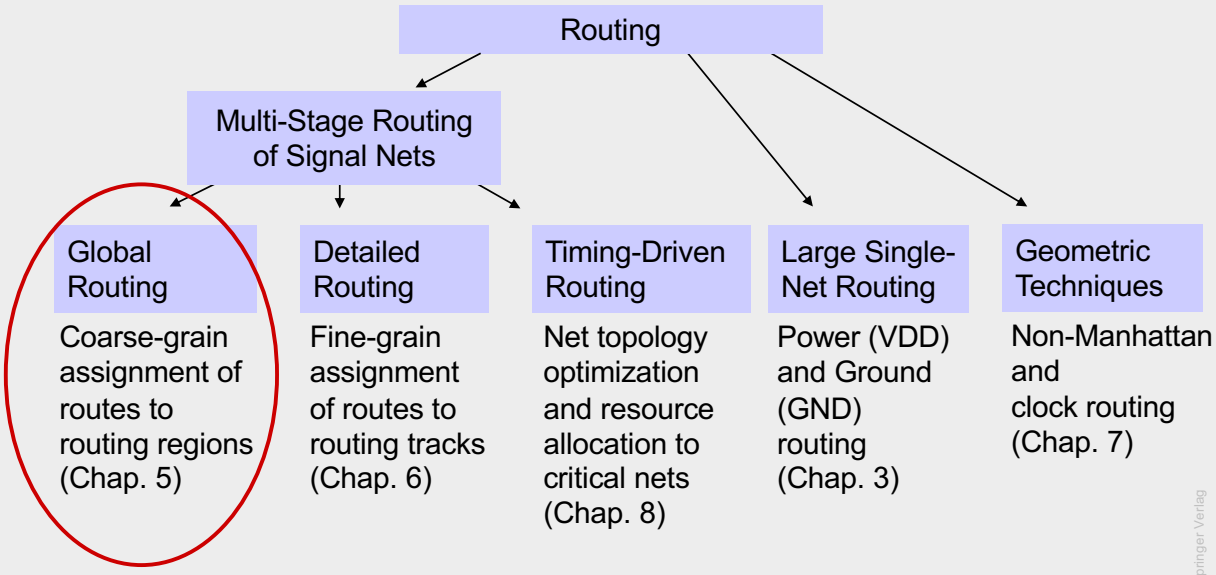
Netlist:

$N_1 = \{C_4, D_6, B_3\}$
$N_2 = \{D_4, B_4, C_1, A_4\}$
$N_3 = \{C_2, D_5\}$
$N_4 = \{B_1, A_1, C_3\}$

Technology Information (Design Rules)



5.1 Introduction

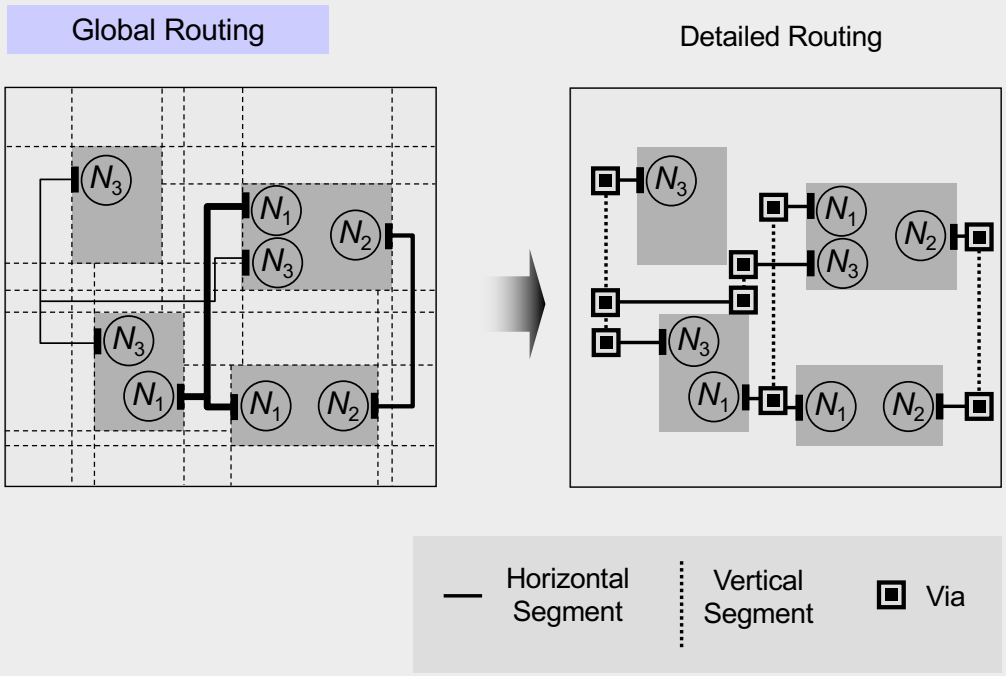


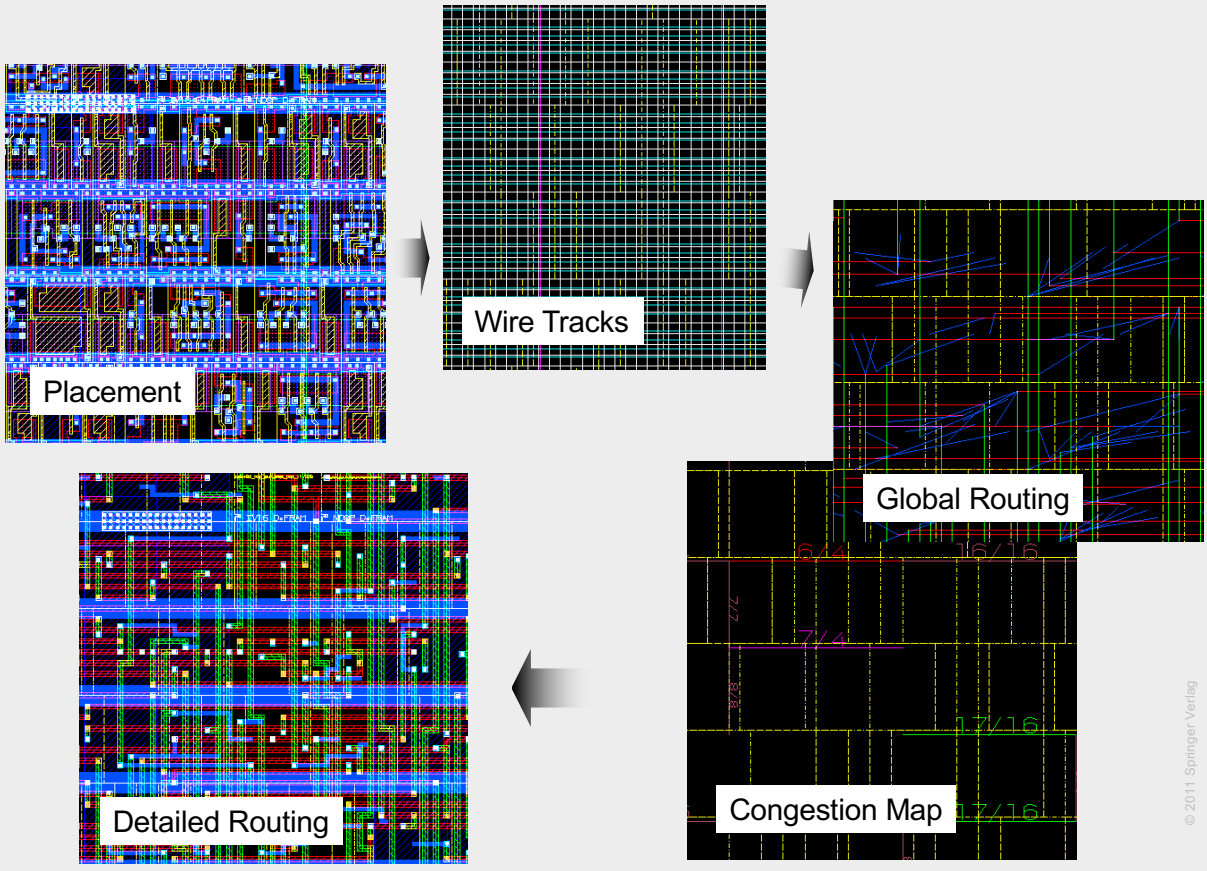
5.1 Introduction

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





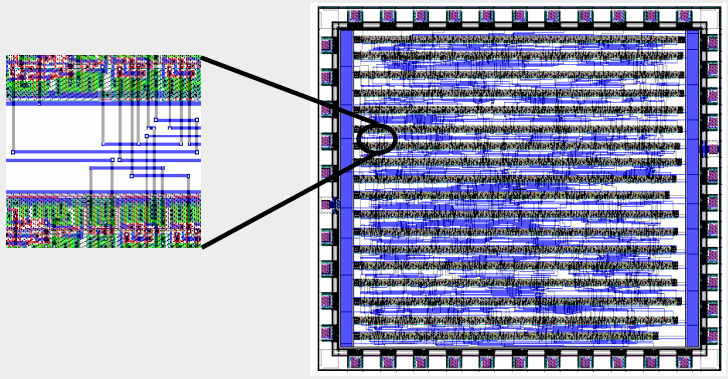
5.2 Terminology and Definitions

- Routing Track: Horizontal wiring path
- Routing Column: Vertical wiring path
- Routing Region: Region that contains routing tracks or columns
- Uniform Routing Region: Evenly spaced horizontal/vertical grid
- Non-uniform Routing Region: Horizontal and vertical boundaries that are aligned to external pin connections or macro-cell boundaries resulting in routing regions that have differing sizes

5.2 Terminology and Definitions

Channel

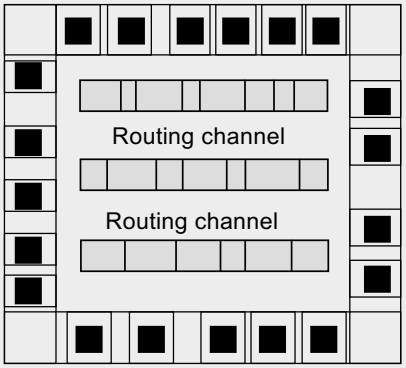
Rectangular routing region with pins on two opposite sides

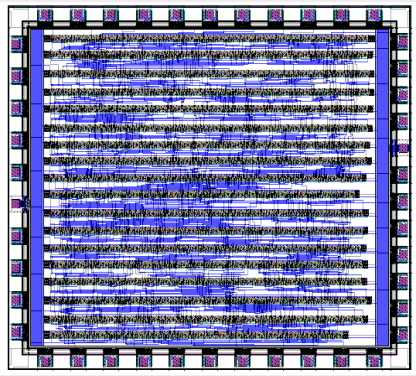


Standard cell layout (Two-layer routing)

Channel

Rectangular routing region with pins on two opposite sides

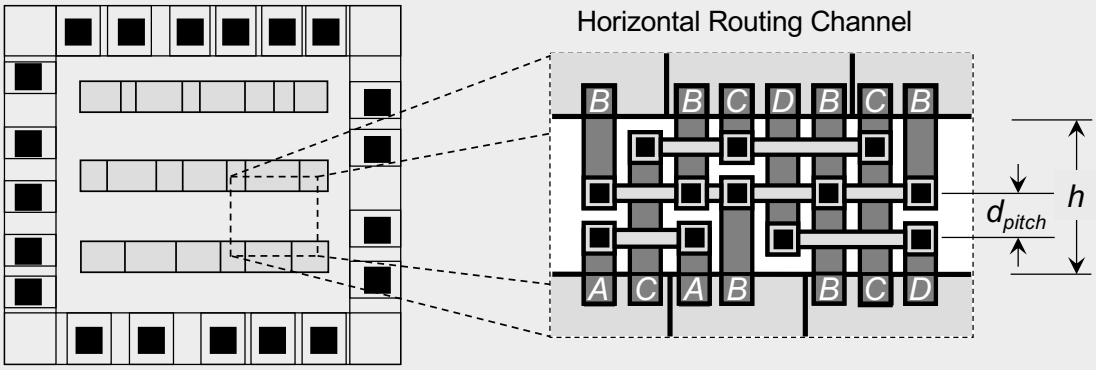




Standard cell layout (Two-layer routing)

Capacity

Number of available routing tracks or columns



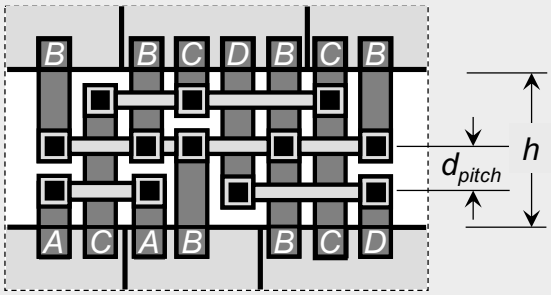
Capacity

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.

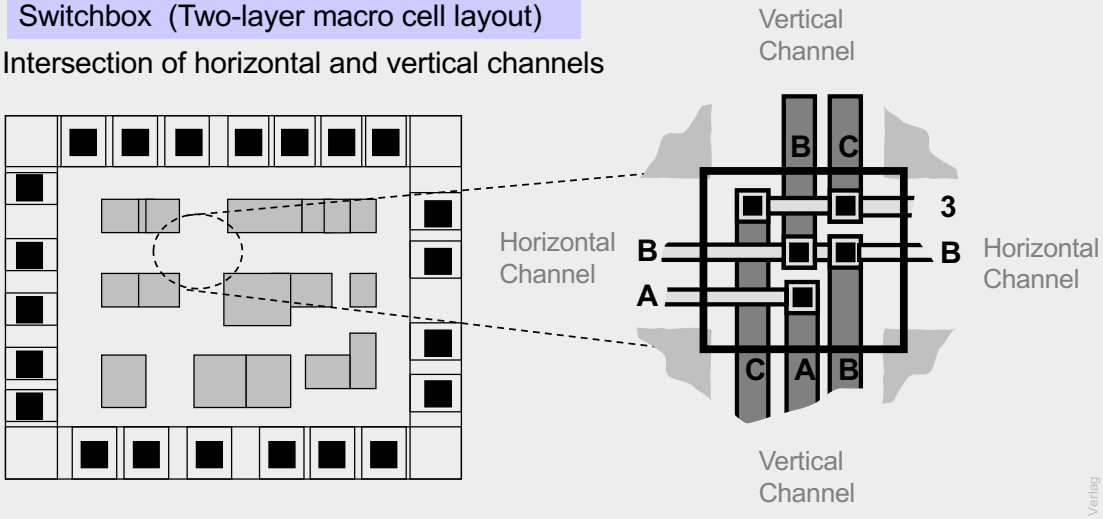
$$\sigma(Layers) = \sum_{layer \in Layers} \left\lfloor \frac{h}{d_{pitch}(layer)} \right\rfloor$$

Horizontal Routing Channel



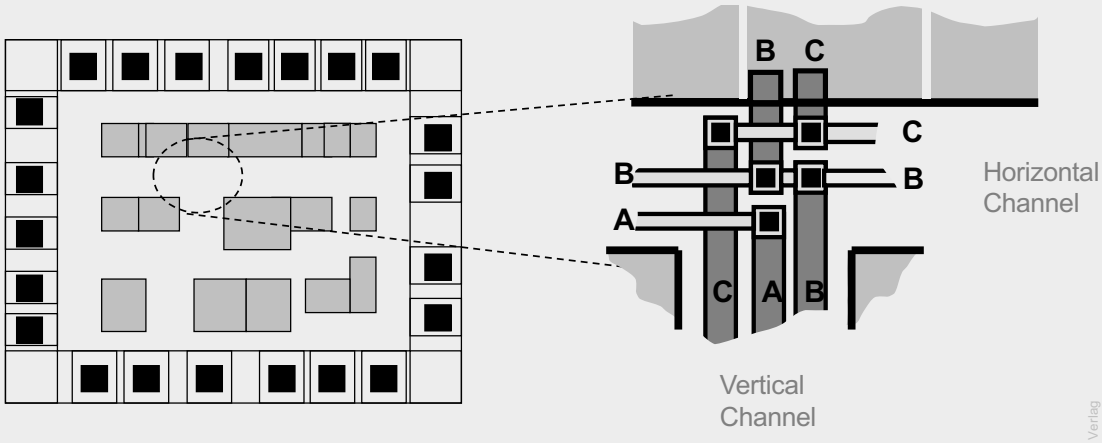
Switchbox (Two-layer macro cell layout)

Intersection of horizontal and vertical channels

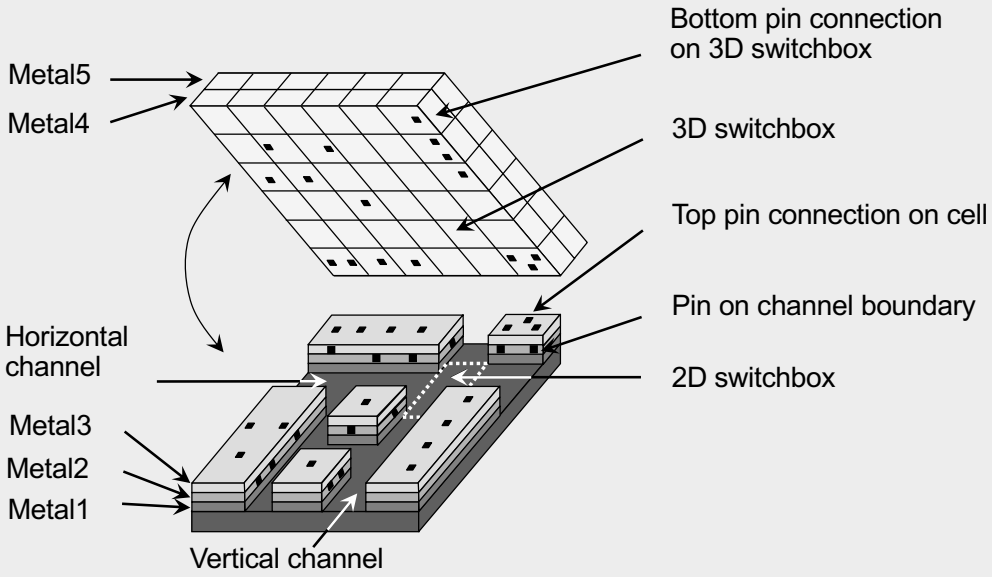


Horizontal channel is routed after vertical channel is routed

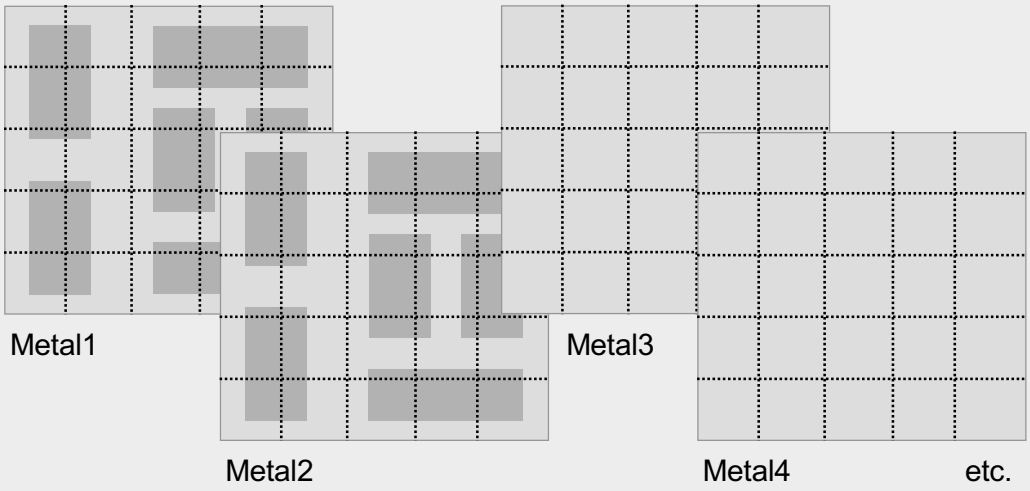
T-junction (Two-layer macro cell layout)



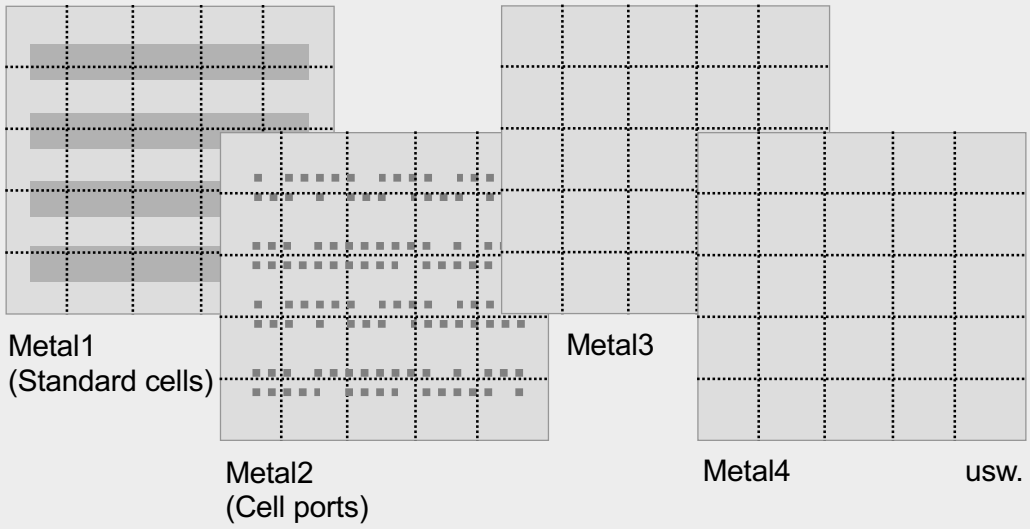
2D and 3D Switchboxes



Gcells (Tiles) with macro cell layout

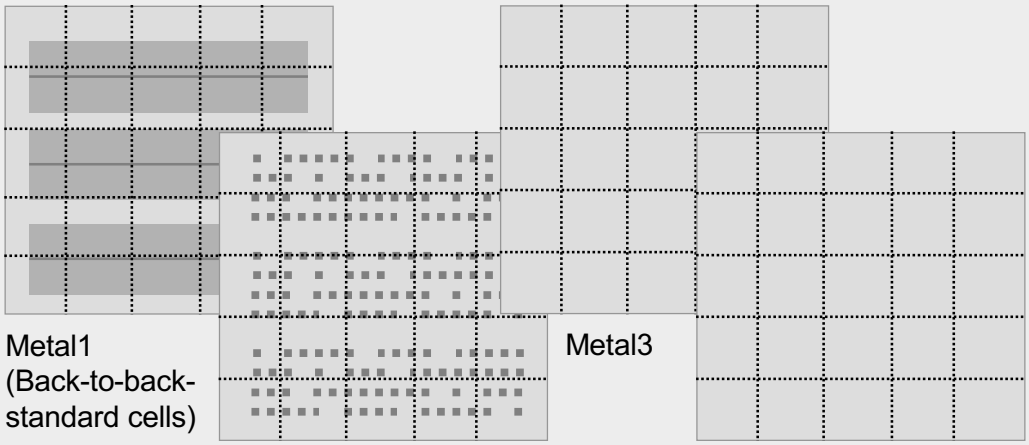


Gcells (Tiles) with standard cells



5.2 Terminology and Definitions

Gcells (Tiles) with standard cells (back-to-back)

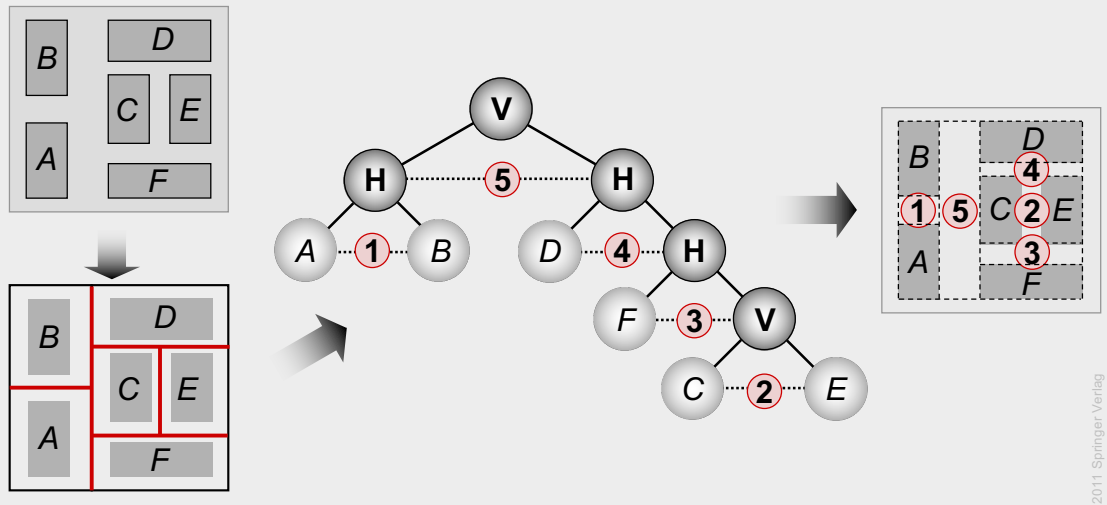


5.3 Optimization Goals

- 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

Full-custom design

Layout is dominated by macro cells and routing regions are non-uniform

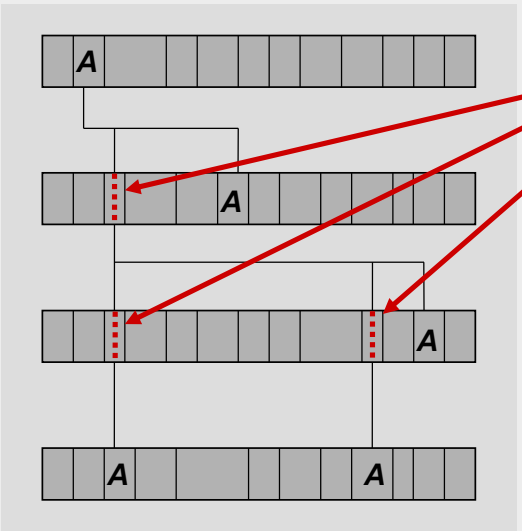


(1) Types of channels

(2) Channel ordering

Standard-cell design

If number of metal layers is limited, feedthrough cells must be used to route across multiple cell rows

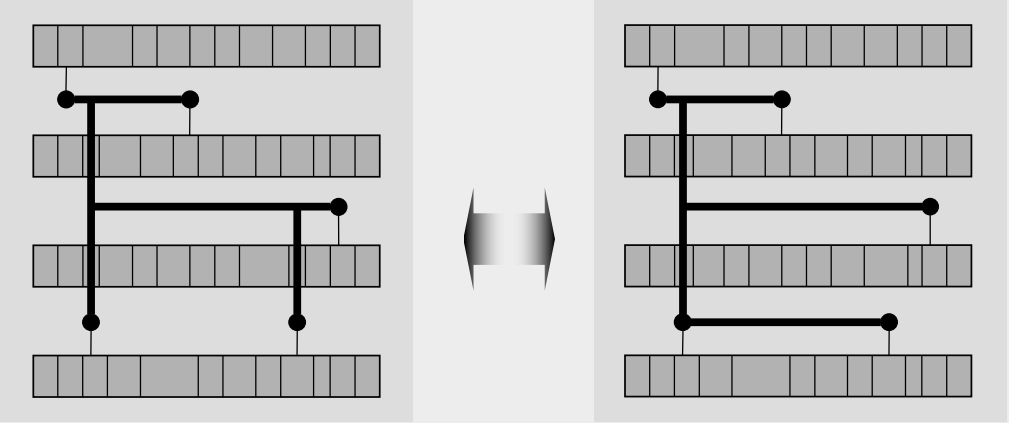


Feedthrough cells

Variable-die, standard cell design:

Total height = ΣCell row heights + All channel heights

Standard-cell design

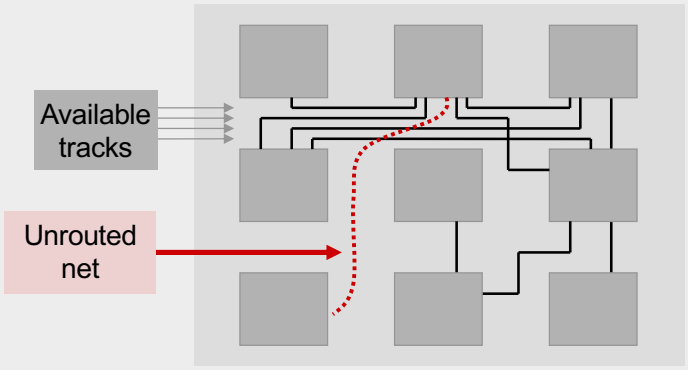


Steiner tree solution with minimal wirelength

Steiner tree solution with fewest feedthrough cells

Gate-array design

Cell sizes and sizes of routing regions between cells are fixed



Key Tasks:

Determine routability

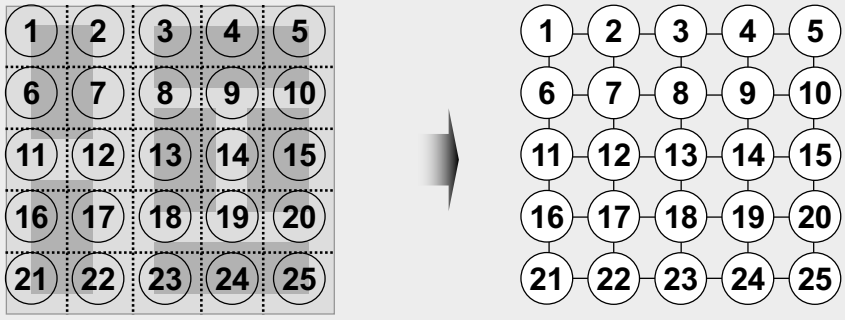
Find a feasible solution

5.4 Representations of Routing Regions

5.4 Representations of Routing Regions

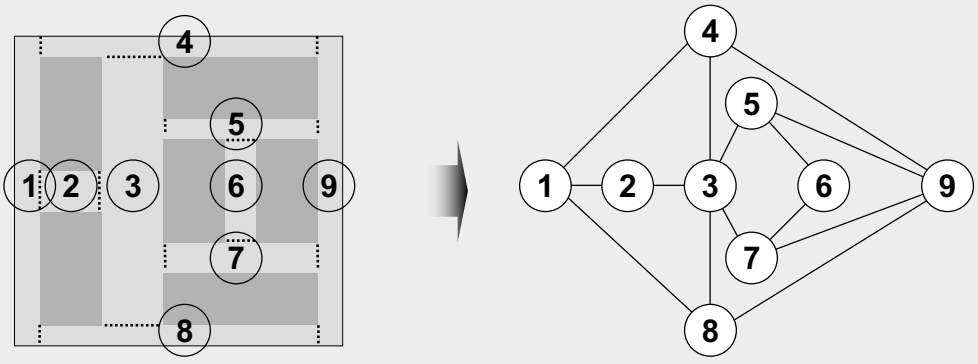
- Routing regions are represented using efficient data structures
- Routing context is captured using a graph, where
  - nodes represent routing regions and
  - edges represent adjoining regions
- Capacities are associated with both edges and nodes to represent available routing resources

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)

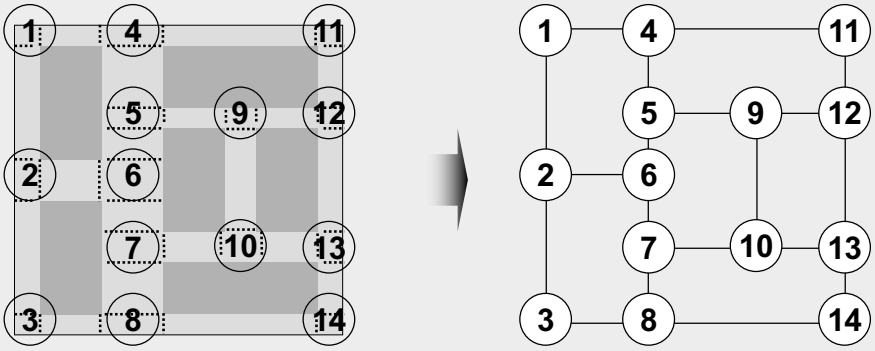
Channel connectivity graph



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



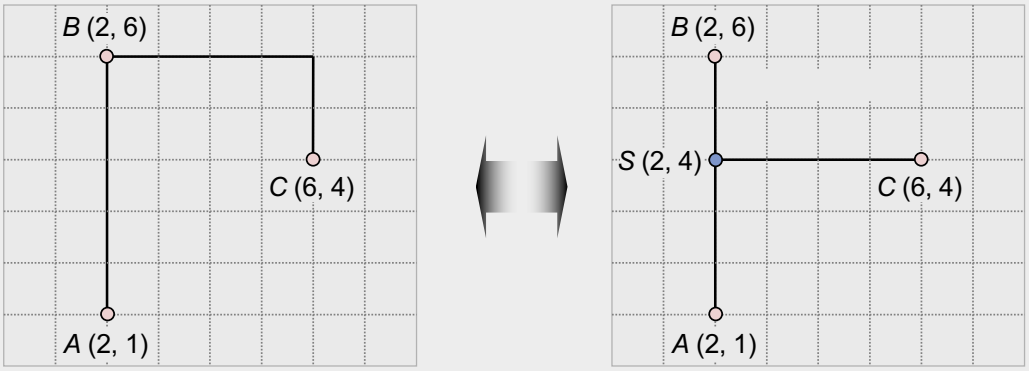
$G = (V, E)$, where the nodes $v \in V$ represent switchboxes
and an edge exists between two nodes if the corresponding switchboxes are on opposite sides of the same channel

5.5 The Global Routing Flow – General Idea

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
3. Assigning crosspoints along the edges of the routing regions (Midway routing)
   - Routes are assigned to fixed locations or crosspoints along the edges of the routing regions
   - Enables scaling of global and detailed routing

5.6 Single-Net Routing

5.6.1 Rectilinear Routing



Rectilinear minimum spanning tree (RMST)

Rectilinear Steiner minimum tree (RSMT)

5.6.1 Rectilinear Routing

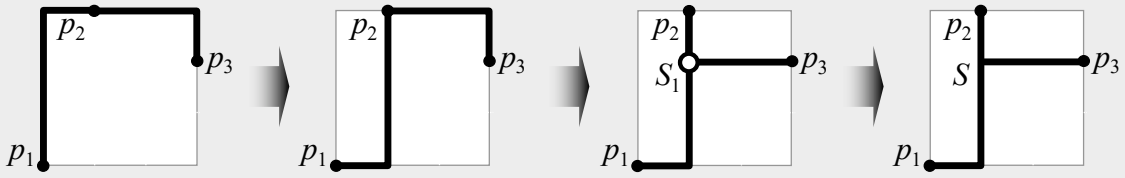
- An RMST can be computed in $O(p^2)$ time, where p is the number of terminals in the net using methods such as Prim's Algorithm
- Prim's Algorithm builds an MST by starting with a single terminal and greedily adding least-cost edges to the partially-constructed tree
- Advanced computational-geometric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

5.6.1 Rectilinear Routing

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



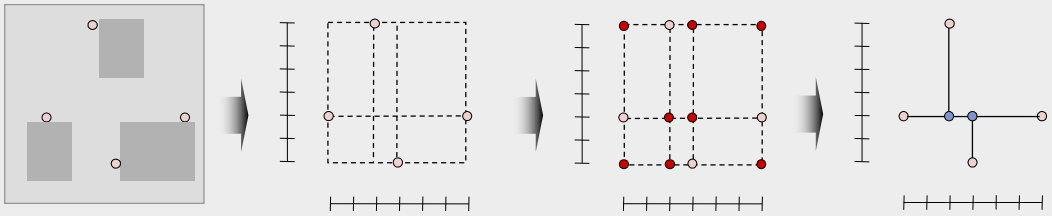
Construct *L*-shapes between points with (most) overlap of net segments

Final tree (RSMT)

Hanan grid

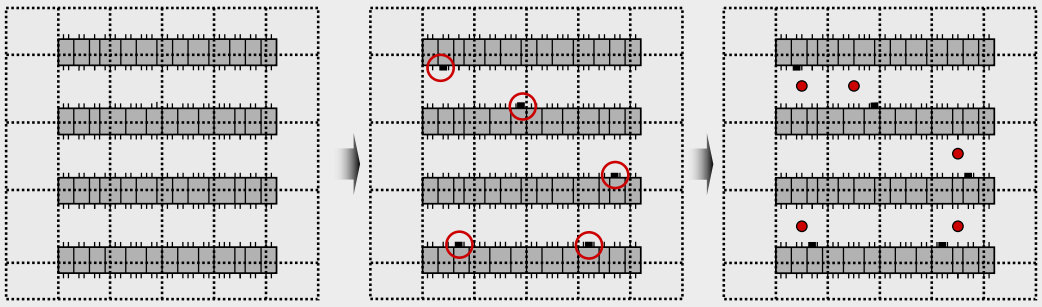
- Adding Steiner points to an RMST can significantly reduce the wirelength
- Maurice Hanan proved that for finding Steiner points, it suffices to consider only points located at the intersections of vertical and horizontal lines that pass through terminal pins
- The Hanan grid consists of the lines $x = x_p$, $y = y_p$ 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

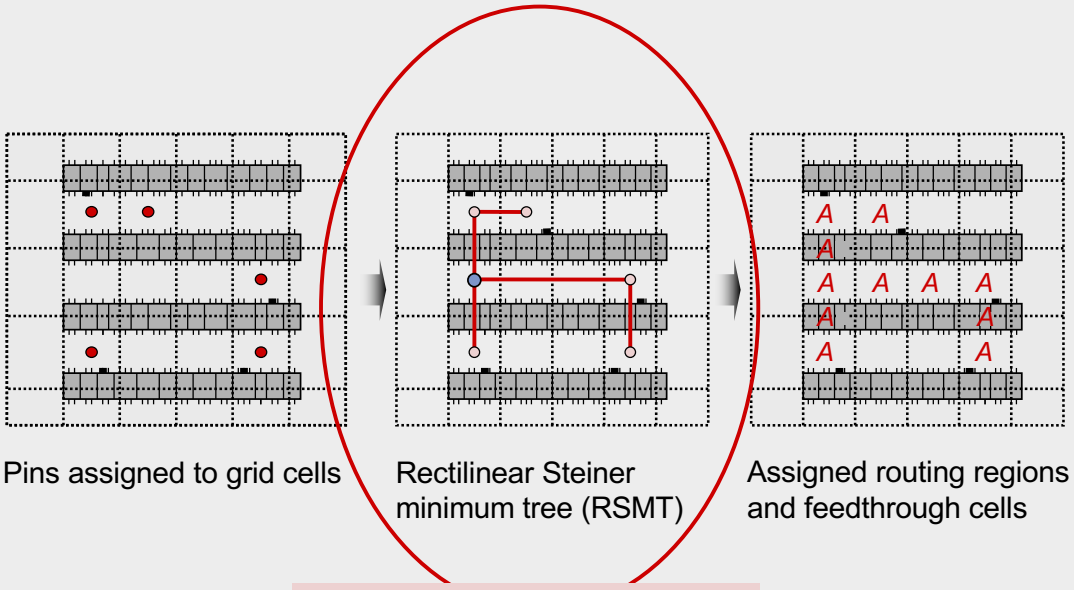


Terminal pins Intersection lines Hanan points (•) RSMT

5.6.1 Rectilinear Routing



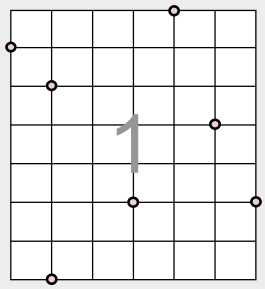
Definining routing regions Pin connections Pins assigned to grid cells



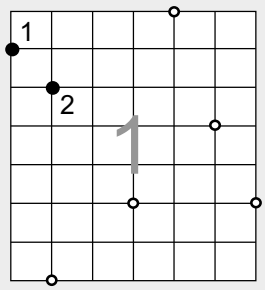
A Sequential Steiner Tree Heuristic

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

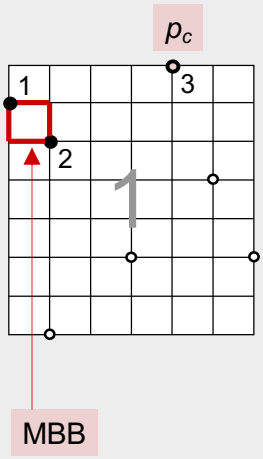
5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic

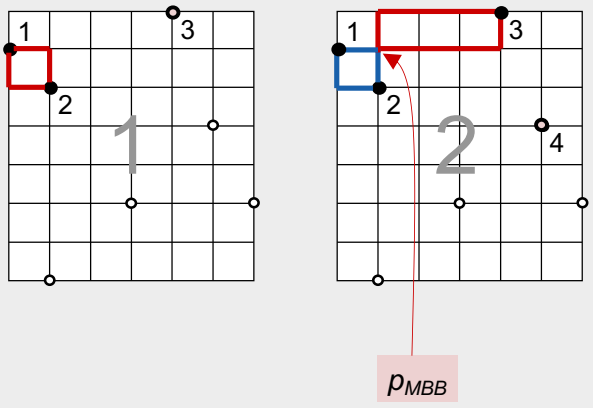


5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



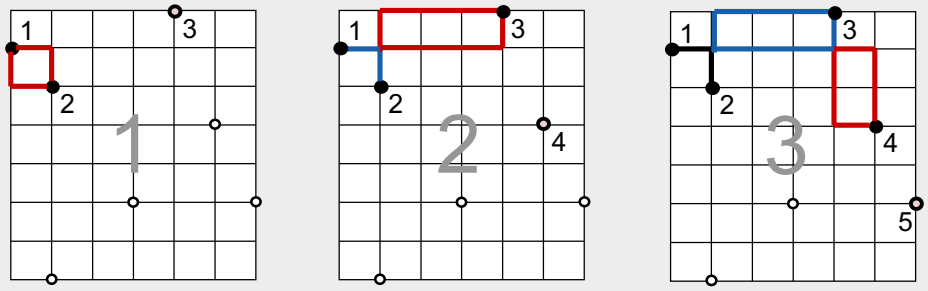
Find the closest (in terms of rectilinear distance) pin pair, construct their minimum bounding box (MBB)

5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



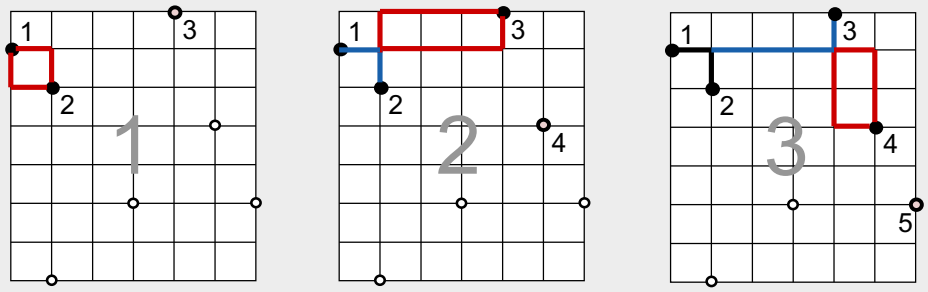
Find the closest point pair (p_{MBB},p_C) between any point p_{MBB} on the MBB and p_C from the set of pins to consider

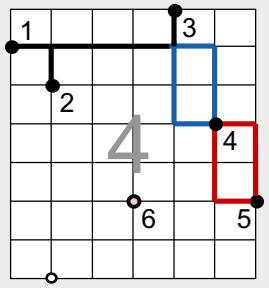
5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



Construct the MBB of p_{MBB} and p_C

5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic

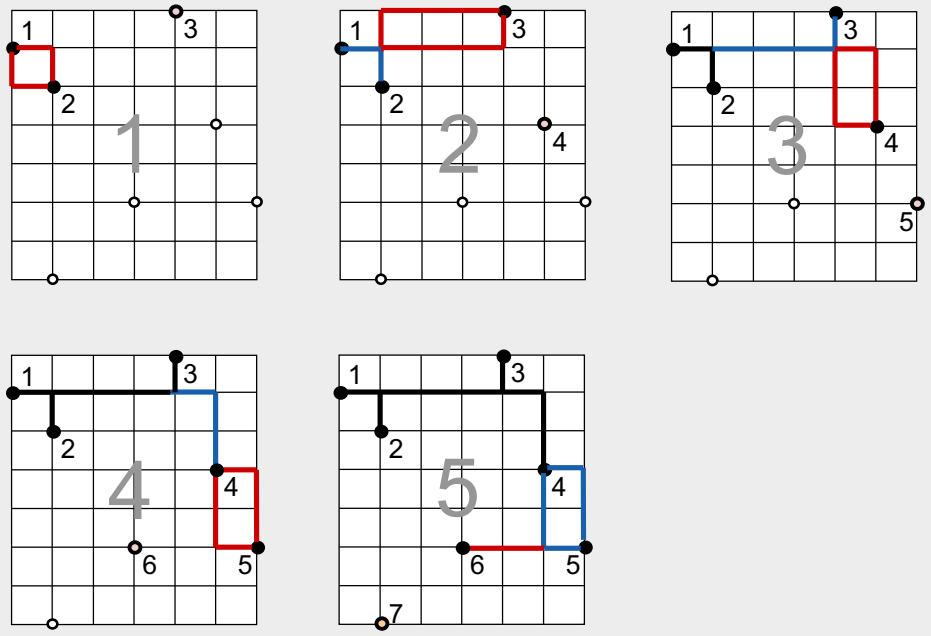




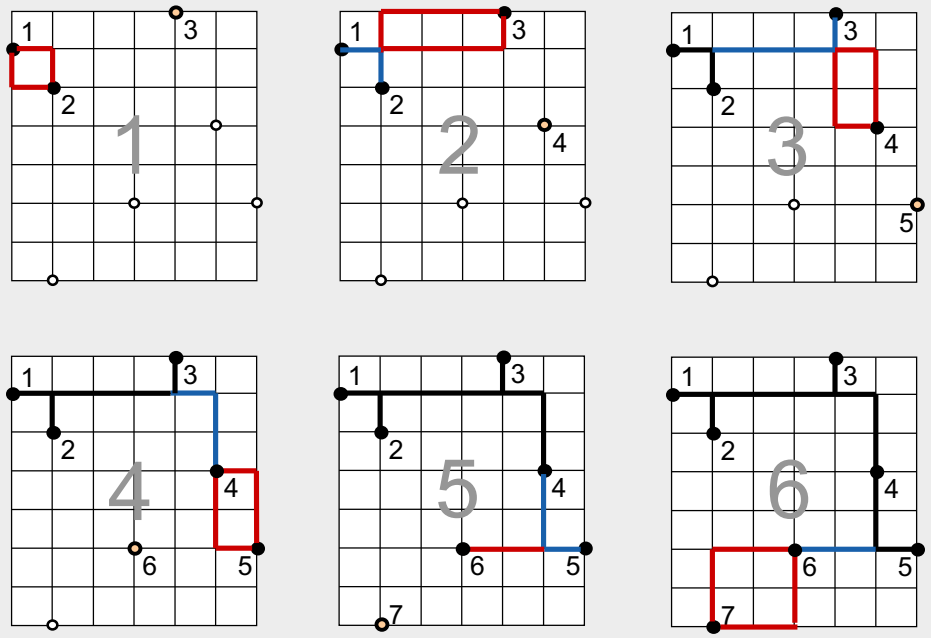
Add the *L*-shape that p_{MBB} lies on to *T* (deleting the other *L*-shape).
If p_{MBB} is a pin, then add any *L*-shape of the MBB to *T*.

Repeat Procedure

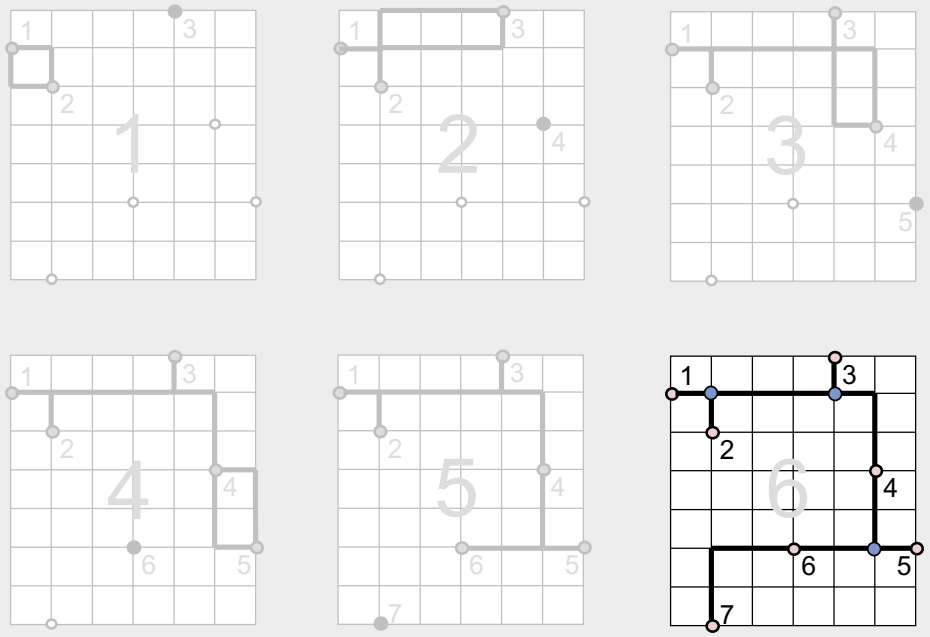
5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic



5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic

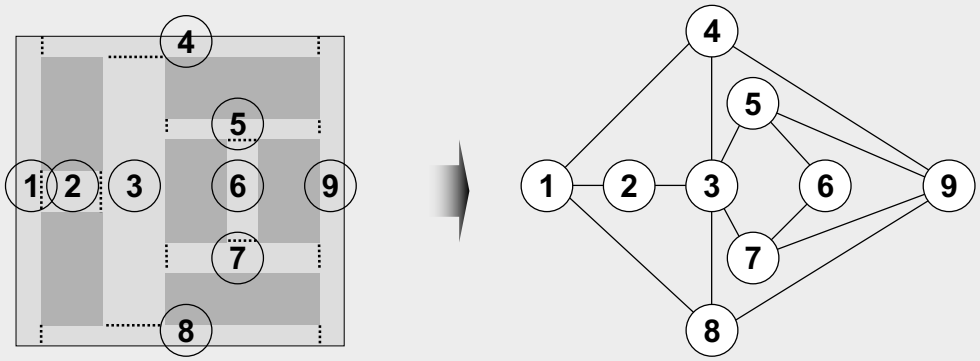


5.6.1 Rectilinear Routing: Example Sequential Steiner Tree Heuristic

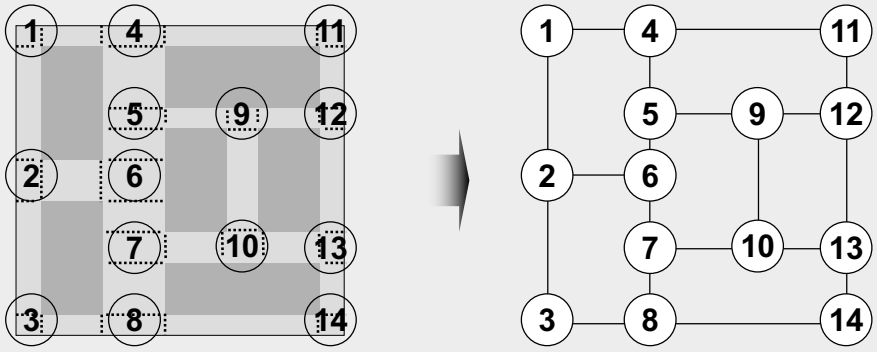


5.6.2 Global Routing in a Connectivity Graph

Channel connectivity graph



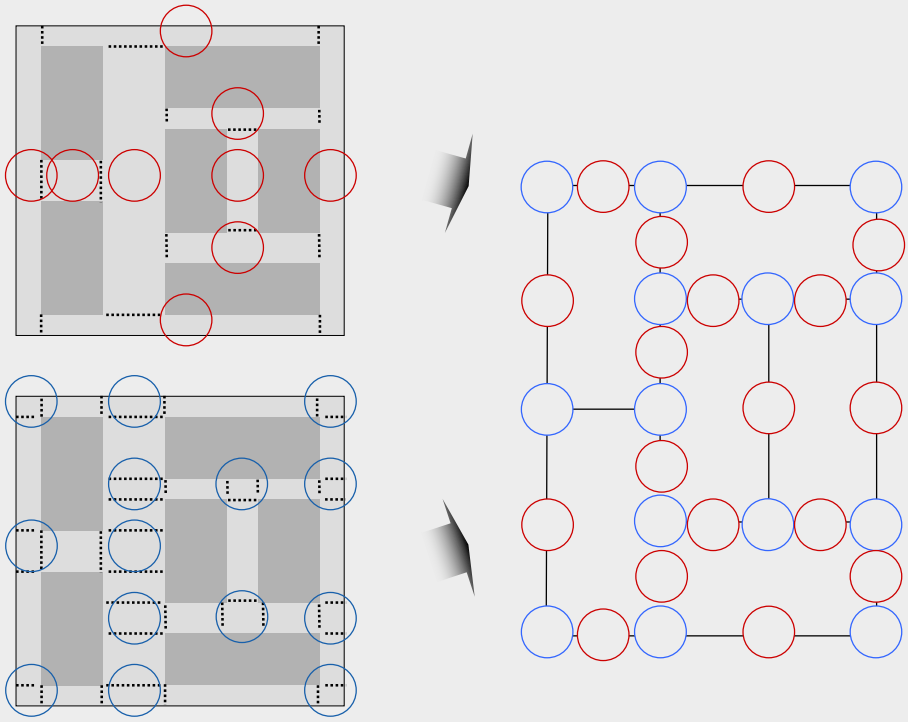
Switchbox connectivity graph



5.6.2 Global Routing in a Connectivity Graph

Channel connectivity graph

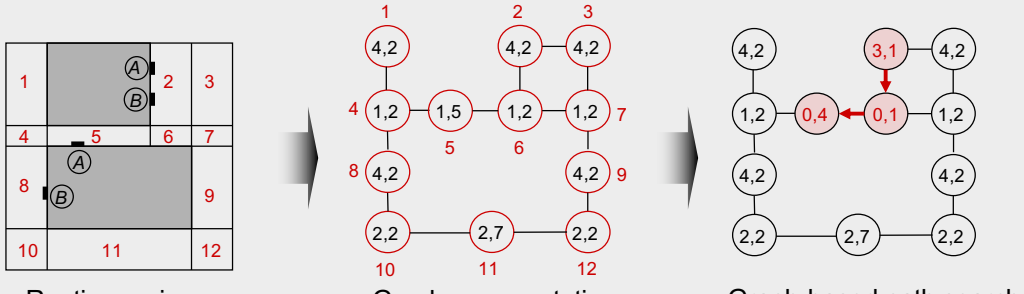
Switchbox connectivity graph



5.6.2 Global Routing in a Connectivity Graph

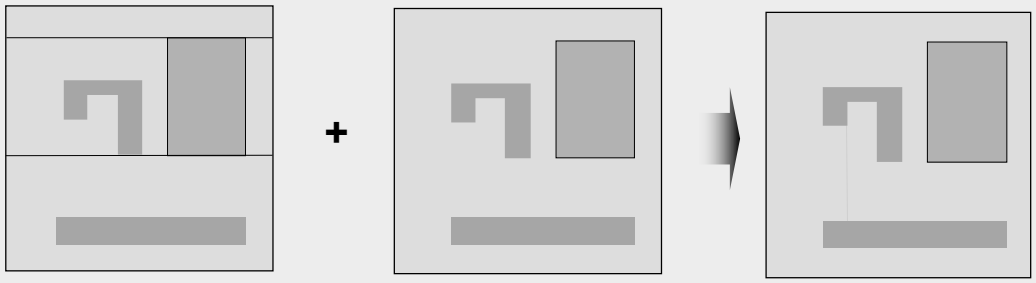
- Combines switchboxes and channels, handles non-rectangular block shapes
- Suitable for full-custom design and multi-chip modules

Overview:



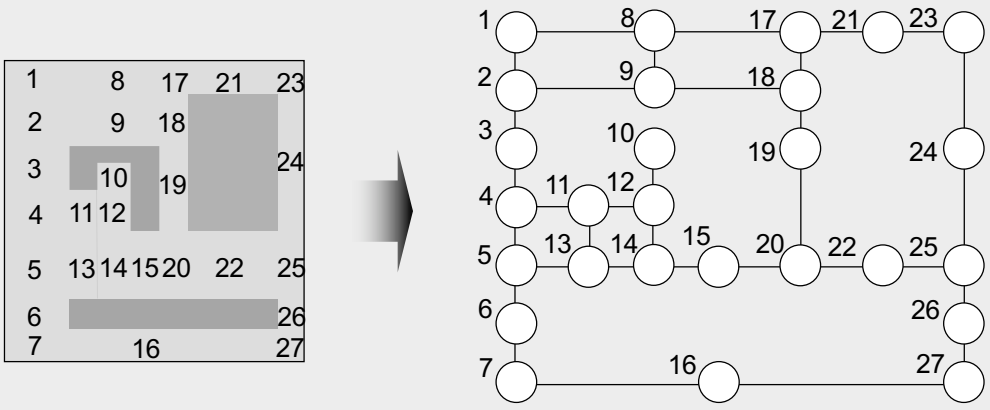
Routing regions Graph representation Graph-based path search

Defining the routing regions

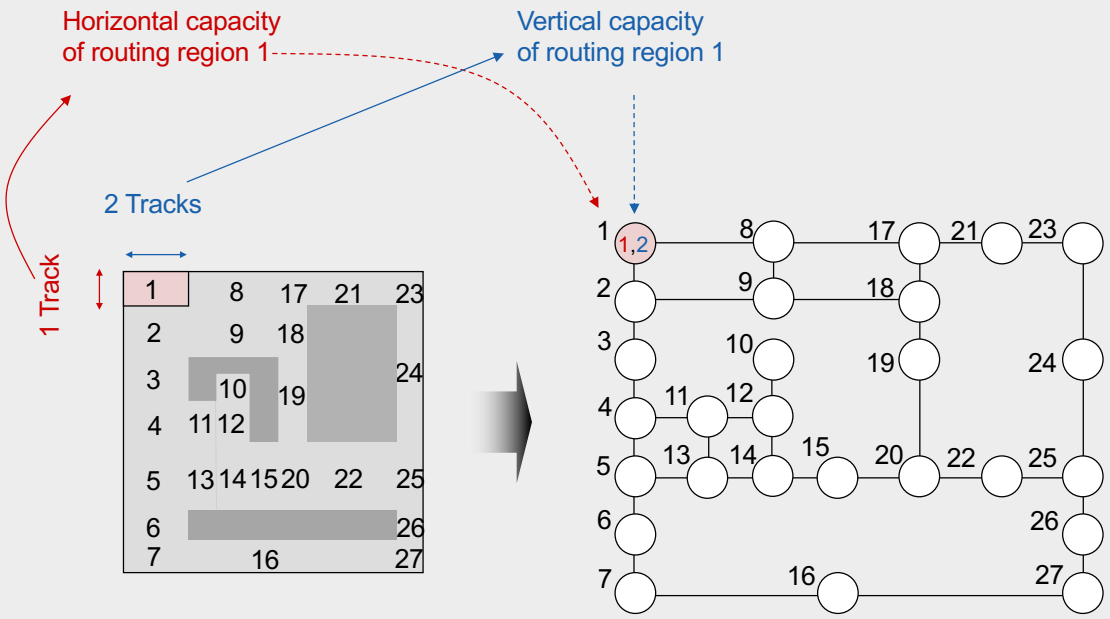


Horizontal macro-cell edges Vertical macro-cell edges

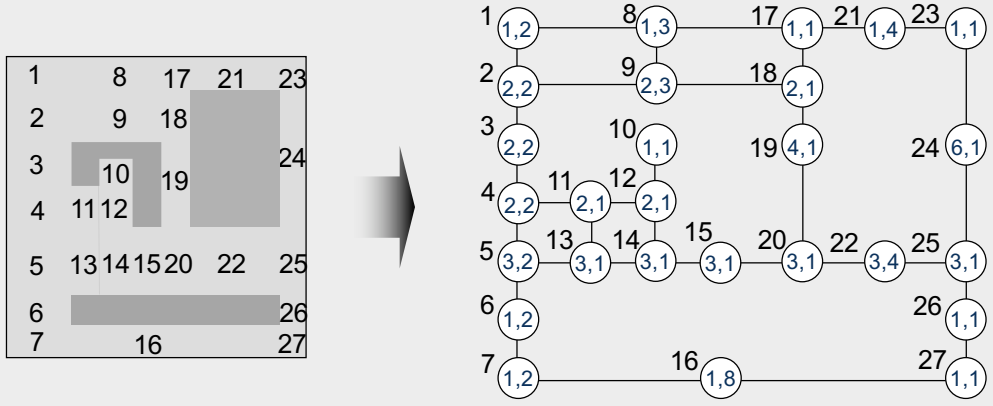
Defining the connectivity graph



5.6.2 Global Routing in a Connectivity Graph



5.6.2 Global Routing in a Connectivity Graph

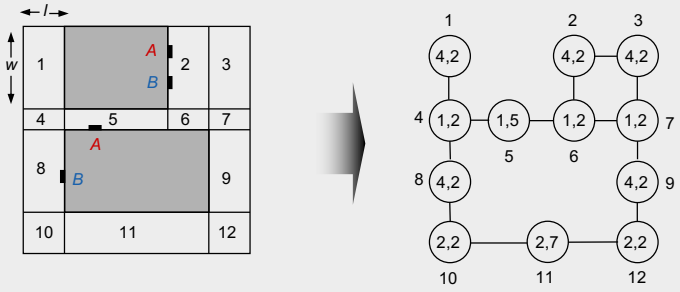


Algorithm Overview

1. Define routing regions
2. Define connectivity graph
3. Determine net ordering
4. Assign tracks for all pin connections in Netlist
5. Consider each net
   a) Free corresponding tracks for net's pins
   b) Decompose net into two-pin subnets
   c) Find shortest path for subnet connectivity graph
   d) If no shortest path exists, do not route, otherwise, assign subnet to the nodes of shortest path and update routing capacities
6. If there are unrouted nets, goto Step 5, otherwise END

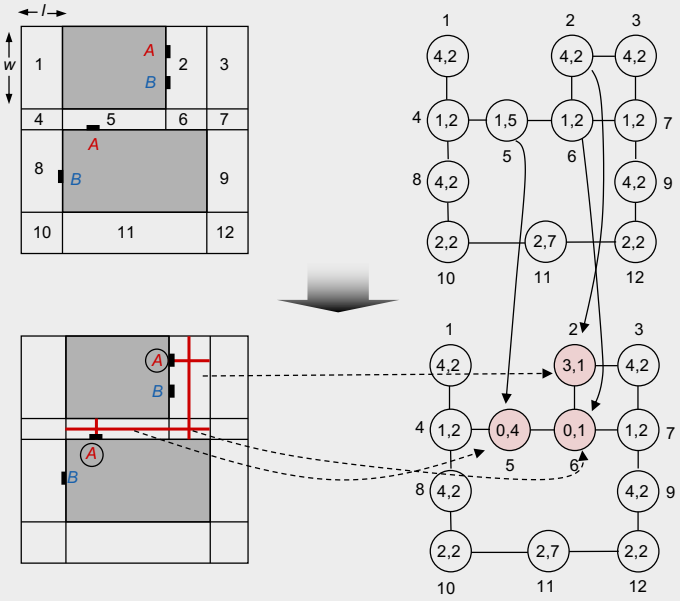
Example

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



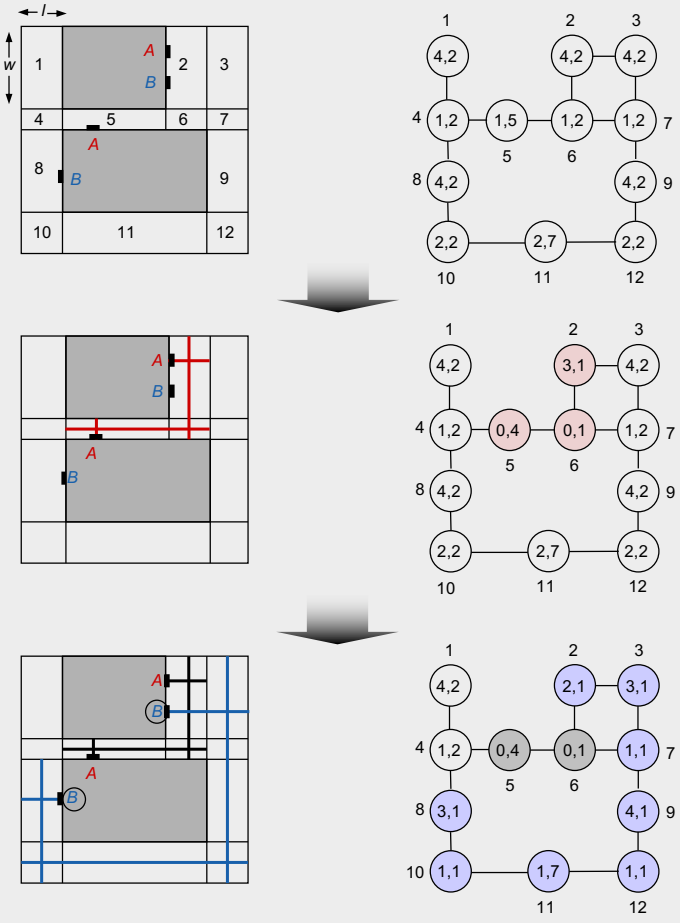
Example

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



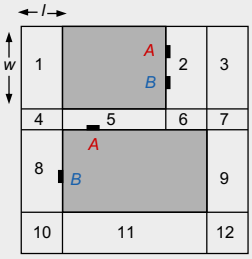
Example

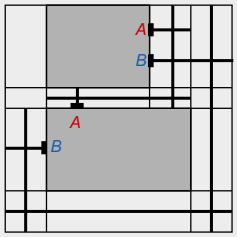
Global routing
of the nets A-A and B-B



Example

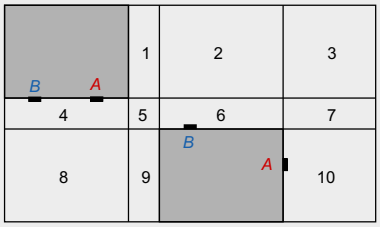
Global routing
of the nets A-A and B-B

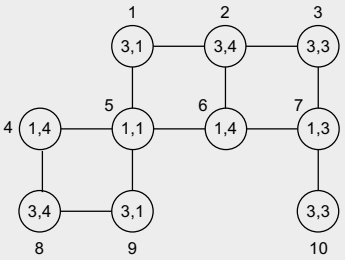


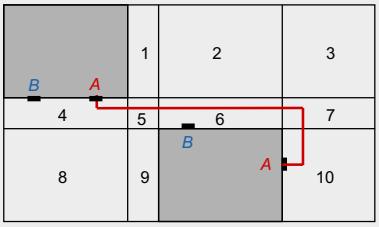


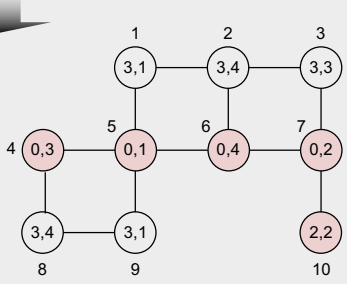
Example

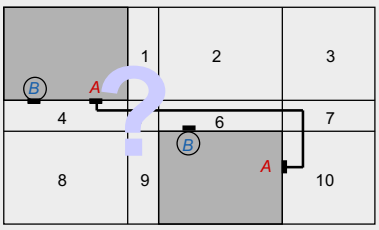
Determine
routability
of a placement

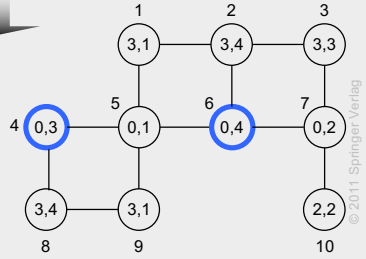






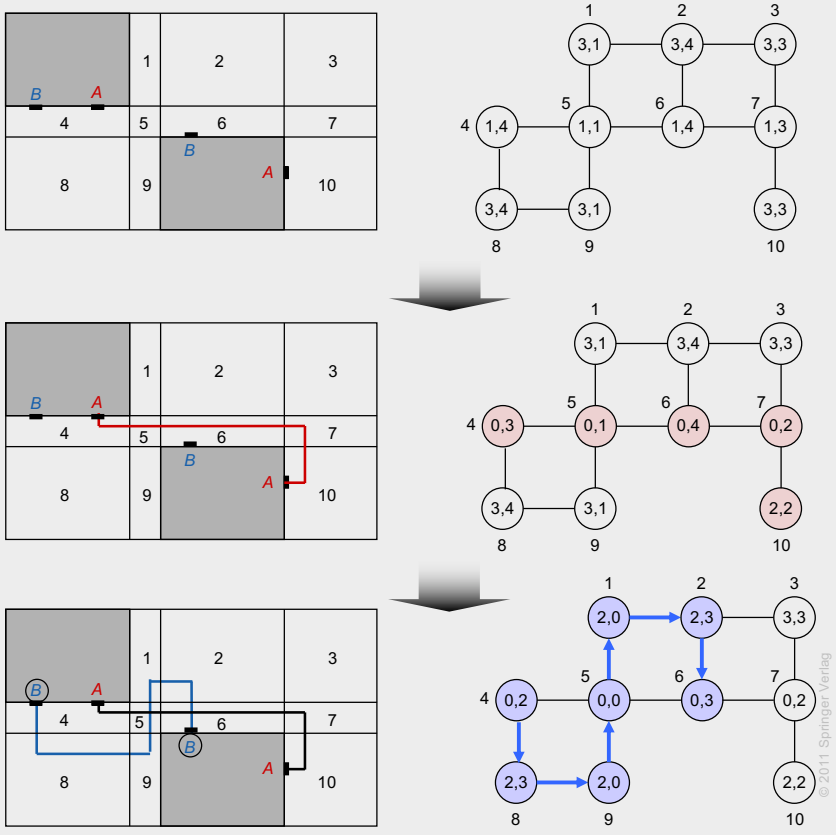






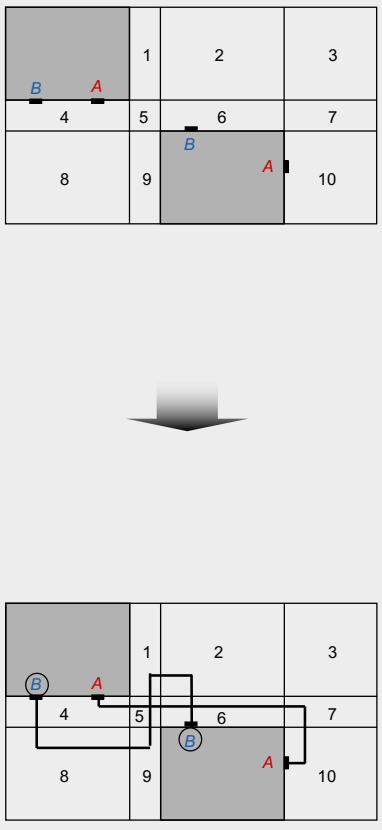
Example

Determine routability of a placement



Example

Determine routability of a placement

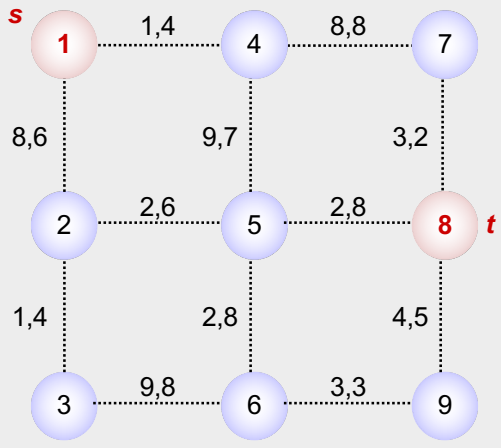


5.6.3 Finding Shortest Paths with Dijkstra's Algorithm

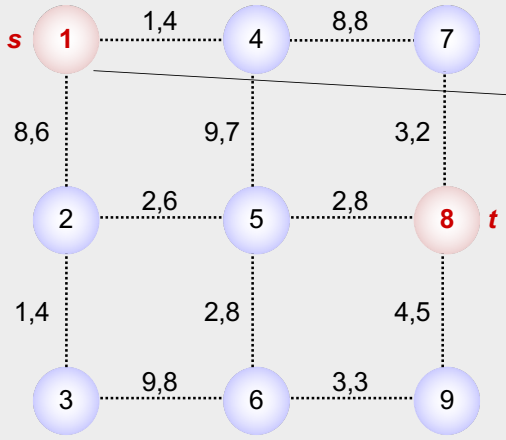
- Finds a shortest path between two specific nodes in the routing graph
- Input
  - graph $G(V,E)$ with non-negative *edge weights* W,
  - *source* (starting) node s, and
  - *target* (ending) node t
- Maintains three groups of nodes
  - Group 1 – contains the nodes that have not yet been visited
  - 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

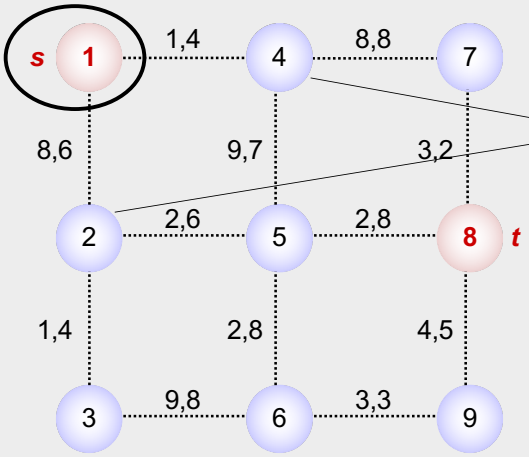


Find the shortest path from source s to target t where the path cost $\sum w_1 + \sum w_2$ is minimal



Group 2	Group 3
	(1)

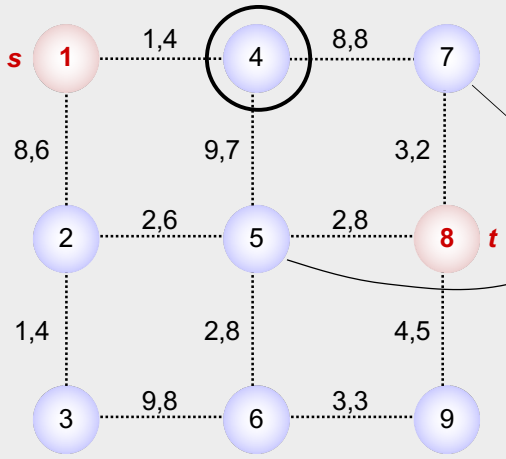
Current node: 1



Group 2	Group 3
N [2] 8,6 *W* [4] 1,4	[1]
	W [4] 1,4

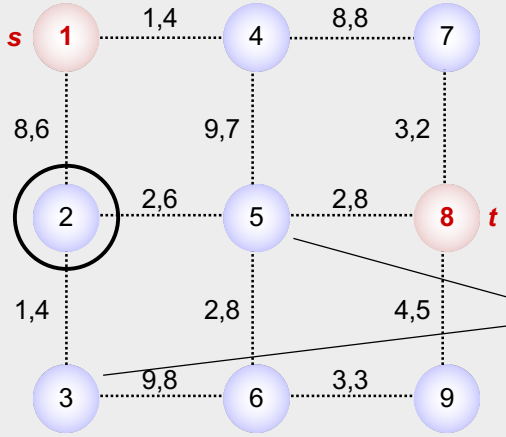
parent of node [node name] $\sum w_1(s,node), \sum w_2(s,node)$

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



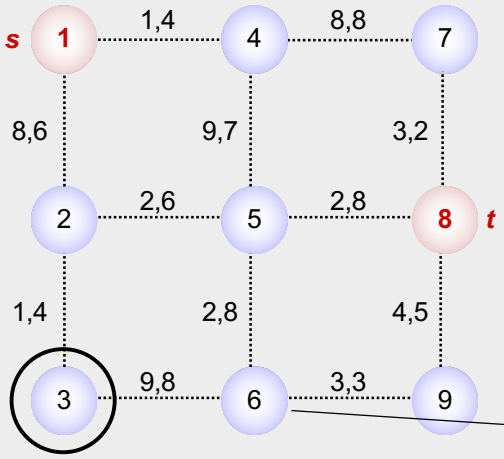
Group 2	Group 3
N [2] 8,6 *W* [4] 1,4	[1]
N [5] 10,11 *W* [7] 9,12	*W* [4] 1,4
	N [2] 8,6
parent of node [node name] $\sum w_1(s,node), \sum w_2(s,node)$	

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



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
	N [3] 9,10
parent of node [node name] $\sum w_1(s,node), \sum w_2(s,node)$	

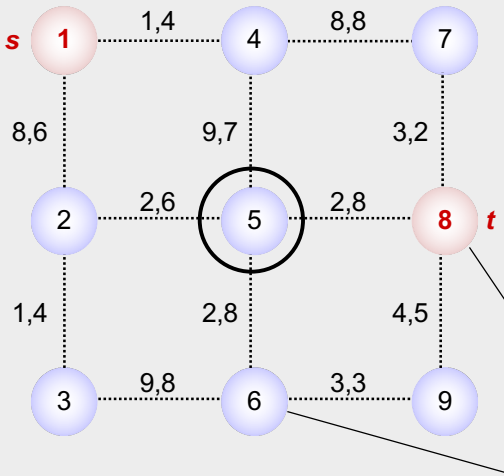
Current node: 2
Neighboring nodes: ~~1~~, 3, 5
Minimum cost in group 2: node 3



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 [5] 10,11

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

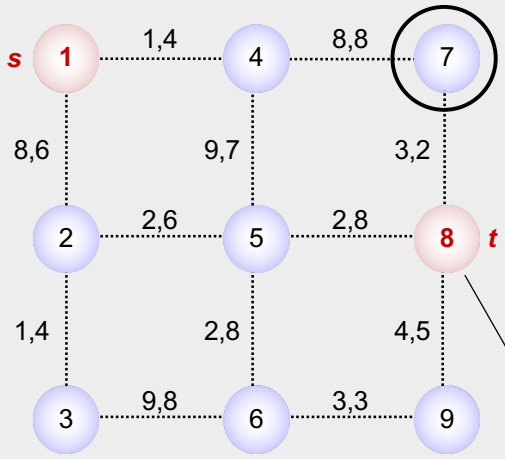
Current node: 3
Neighboring nodes: ~~2~~, 6
Minimum cost in group 2: node 5



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] $\sum w_1(s,node), \sum w_2(s,node)$

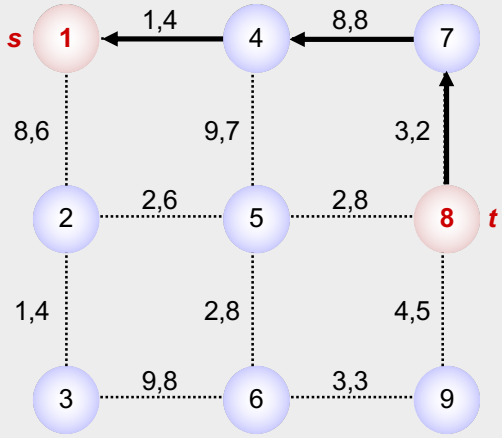
Current node: 5
Neighboring nodes: ~~2~~, ~~4~~, 6, 8
Minimum cost in group 2: node 7



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

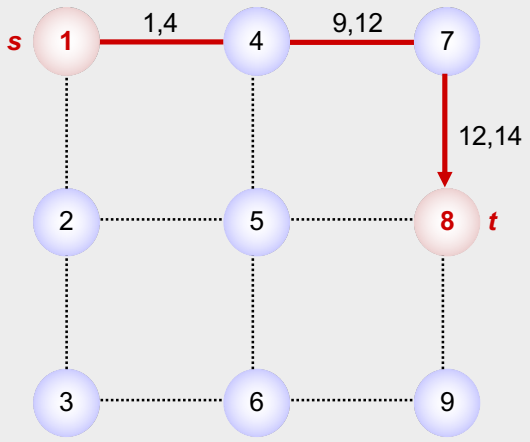
Group 2	Group 3
N (2) 8,6 *W* (4) 1,4	(1)
N (5) 10,11 *W* (7) 9,12	*W* (4) 1,4
N (3) 9,10 ~~*W* (5) 10,12~~	*N* (2) 8,6
W (6) 18,18	*N* (3) 9,10
N (6) 12,19 ~~*W* (8) 12,19~~	*N* (5) 10,11
N (8) 12,14	*W* (7) 9,12
	***N* (8) 12,14**

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



Retrace from *t* to *s*

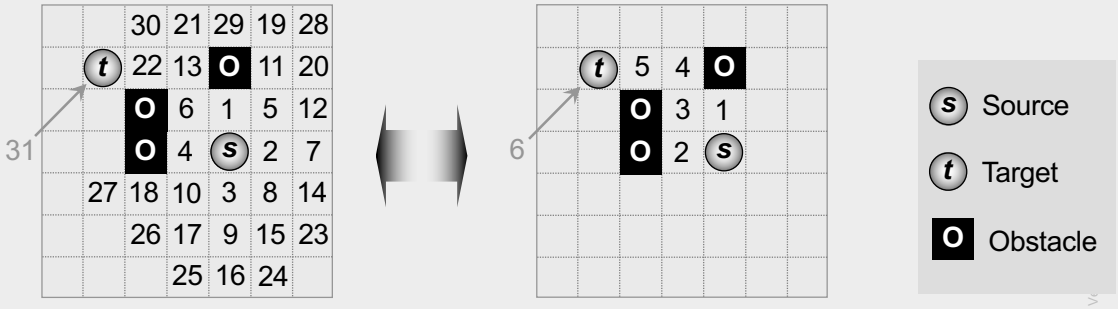
Group 2	Group 3
N (2) 8,6 *W* (4) 1,4	(1)
N (5) 10,11 *W* (7) 9,12	*W* (4) 1,4
N (3) 9,10 ~~*W* (5) 10,12~~	*N* (2) 8,6
W (6) 18,18	*N* (3) 9,10
N (6) 12,19 ~~*W* (8) 12,19~~	*N* (5) 10,11
N (8) 12,14	*W* (7) 9,12
	***N* (8) 12,14**



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

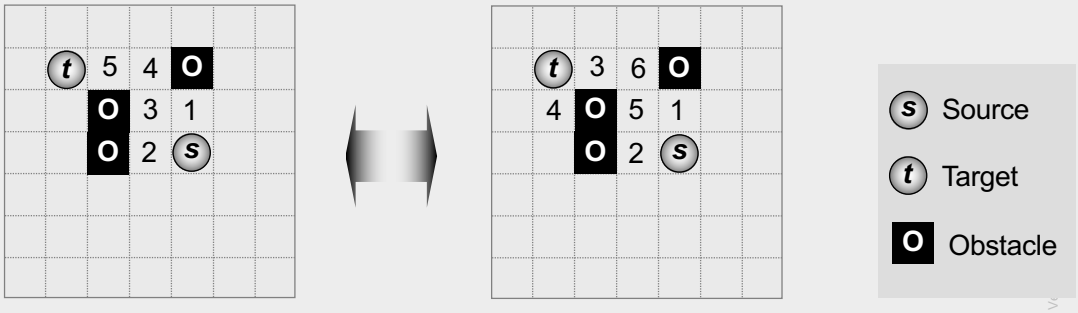


Dijkstra's algorithm (exploring 31 nodes)

A* search (exploring 6 nodes)

5.6.4 Finding Shortest Paths with A* Search

- Bidirectional A* search: nodes are expanded from both the source and target until the two expansion regions intersect
- Number of nodes considered can be reduced



Unidirectional A* search

Bidirectional A* search

5.7 Full-Netlist 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

5.7.1 Routing by Integer Linear Programming

- A linear program (LP) consists
  - of a set of constraints and
  - an optional objective function
- Objective function is maximized or minimized
- Both the constraints and the objective function must be linear
  - Constraints form a system of linear equations and inequalities
- Integer linear program (ILP): linear program where every variable can only assume integer values
  - Typically takes much longer to solve
  - In many cases, variables are only allowed values 0 and 1
- Several ways to formulate the global routing problem as an ILP, one of which is presented next

5.7.1 Routing by Integer Linear Programming

- Three inputs
  - $W \times H$ routing grid G,
  - Routing edge capacities, and
  - Netlist
- Two sets of variables
  - k Boolean variables $x_{net1}, x_{net2}, \ldots, x_{netk}$, each of which serves as an indicator for one of k specific paths or route options, for each net $net \in$ Netlist
  - k real variables $w_{net1}, w_{net2}, \ldots, w_{netk}$, each of which represents a net weight for a specific route option for $net \in$ Netlist
- Two types of constraints
  - Each net must select a single route (mutual exclusion)
  - Number of routes assigned to each edge (total usage) cannot exceed its capacity

5.7.1 Routing by Integer Linear Programming

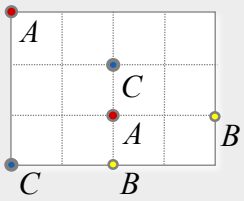
- Inputs
  - W,H: width W and height H of routing grid G
  - $G(i,j)$: grid cell at location (i,j) in routing grid G
  - $\sigma(G(i,j)\sim G(i+1,j))$: capacity of horizontal edge $G(i,j) \sim G(i+1,j)$
  - $\sigma(G(i,j)\sim G(i,j+1))$: capacity of vertical edge $G(i,j) \sim G(i,j+1)$
  - *Netlist*: netlist
- Variables
  - $x_{net1}, \ldots, x_{netk}$: k Boolean path variables for each net $net \in$ *Netlist*
  - $w_{net1}, \ldots, w_{netk}$: k net weights, one for each path of net $net \in$ *Netlist*
- Maximize $\sum_{net \in Netlist} w_{net_1} \cdot x_{net_1} + \ldots + w_{net_k} \cdot x_{net_k}$
- Subject to
  - Variable ranges
  - Net constraints
  - Capacity constraints

5.7.1 Routing by Integer Linear Programming – Example

Global Routing Using Integer Linear Programming

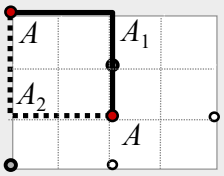
- Given
  - Nets *A*, *B*
  - $W = 5 \times H = 4$ routing grid *G*
  - $\sigma(e) = 1$ for all $e \in G$
  - *L*-shapes have weight 1.00 and *Z*-shapes have weight 0.99
  - The lower-left corner is (0,0).

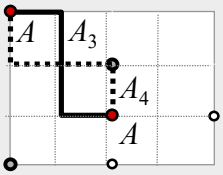
- Task
  - Write the ILP to route the nets in the graph below



5.7.1 Routing by Integer Linear Programming – Example

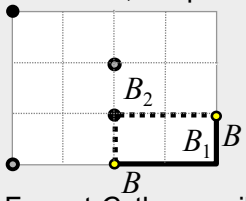
- Solution
  - For net *A*, the possible routes are two *L*-shapes (A_1,A_2) and two *Z*-shapes (A_3,A_4)

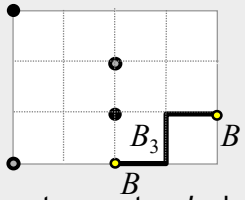




Net Constraints:
$x_{A1} + x_{A2} + x_{A3} + x_{A4} \le 1$
Variable Constraints:
$0 \le x_{A1} \le 1, 0 \le x_{A2} \le 1,$
$0 \le x_{A3} \le 1, 0 \le x_{A4} \le 1$

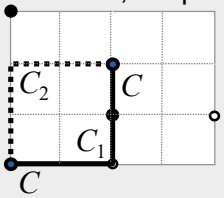
  - For net *B*, the possible routes are two *L*-shapes (B_1,B_2) and one *Z*-shape (B_3)

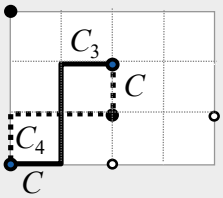




Net Constraints:
$x_{B1} + x_{B2} + x_{B3} \le 1$
Variable Constraints:
$0 \le x_{B1} \le 1, 0 \le x_{B2} \le 1,$
$0 \le x_{B3} \le 1$

  - For net *C*, the possible routes are two *L*-shapes (C_1,C_2) and two *Z*-shapes (C_3,C_4)





Net Constraints:
$x_{C1} + x_{C2} + x_{C3} + x_{C4} \le 1$
Variable Constraints:
$0 \le x_{C1} \le 1, 0 \le x_{C2} \le 1,$
$0 \le x_{C3} \le 1, 0 \le x_{C4} \le 1$

5.7.1 Routing by Integer Linear Programming – Example

Horizontal Edge Capacity Constraints:

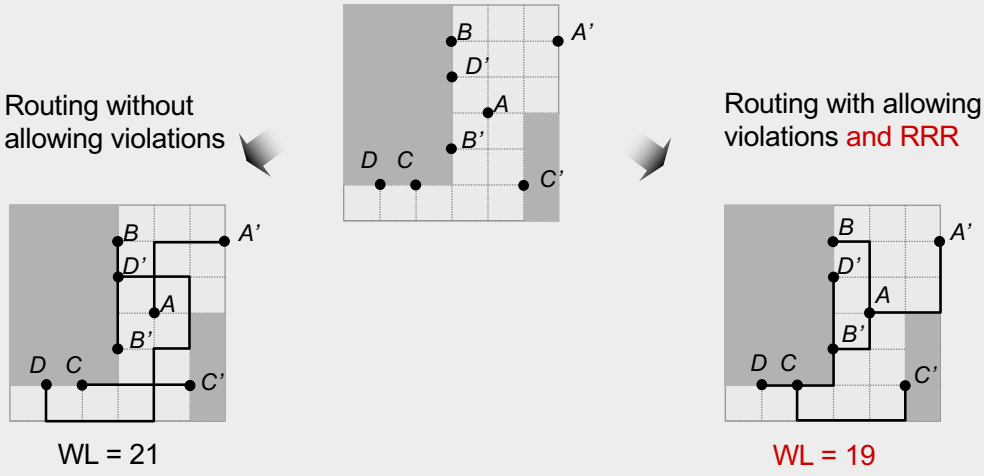
$G(0,0) \sim G(1,0)$:	$x_{C1} + x_{C3}$	$\leq$	$\sigma(G(0,0) \sim G(1,0)) = 1$
$G(1,0) \sim G(2,0)$:	x_{C1}	$\leq$	$\sigma(G(1,0) \sim G(2,0)) = 1$
$G(2,0) \sim G(3,0)$:	$x_{B1} + x_{B3}$	$\leq$	$\sigma(G(2,0) \sim G(3,0)) = 1$
$G(3,0) \sim G(4,0)$:	x_{B1}	$\leq$	$\sigma(G(3,0) \sim G(4,0)) = 1$
$G(0,1) \sim G(1,1)$:	$x_{A2} + x_{C4}$	$\leq$	$\sigma(G(0,1) \sim G(1,1)) = 1$
$G(1,1) \sim G(2,1)$:	$x_{A2} + x_{A3} + x_{C4}$	$\leq$	$\sigma(G(1,1) \sim G(2,1)) = 1$
$G(2,1) \sim G(3,1)$:	x_{B2}	$\leq$	$\sigma(G(2,1) \sim G(3,1)) = 1$
$G(3,1) \sim G(4,1)$:	$x_{B2} + x_{B3}$	$\leq$	$\sigma(G(3,1) \sim G(4,1)) = 1$
$G(0,2) \sim G(1,2)$:	$x_{A4} + x_{C2}$	$\leq$	$\sigma(G(0,2) \sim G(1,2)) = 1$
$G(1,2) \sim G(2,2)$:	$x_{A4} + x_{C2} + x_{C3}$	$\leq$	$\sigma(G(1,2) \sim G(2,2)) = 1$
$G(0,3) \sim G(1,3)$:	$x_{A1} + x_{A3}$	$\leq$	$\sigma(G(0,3) \sim G(1,3)) = 1$
$G(1,3) \sim G(2,3)$:	x_{A1}	$\leq$	$\sigma(G(1,3) \sim G(2,3)) = 1$

Vertical Edge Capacity Constraints:

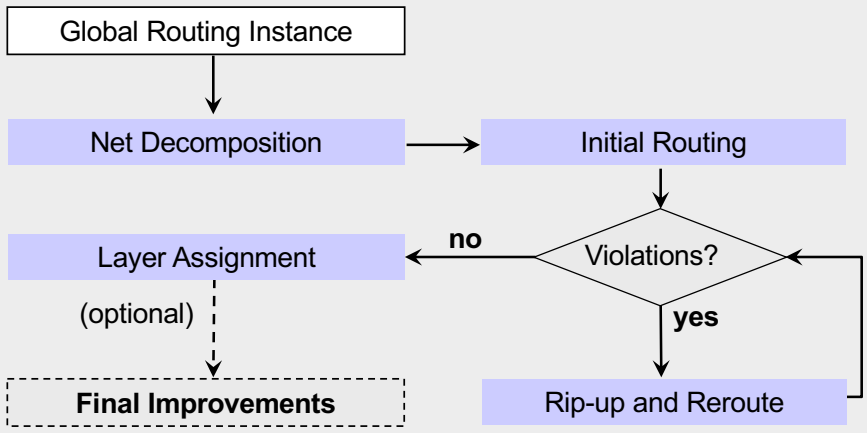
$G(0,0) \sim G(0,1)$:	$x_{C2} + x_{C4}$	$\leq$	$\sigma(G(0,0) \sim G(0,1)) = 1$
$G(1,0) \sim G(1,1)$:	x_{C3}	$\leq$	$\sigma(G(1,0) \sim G(1,1)) = 1$
$G(2,0) \sim G(2,1)$:	$x_{B2} + x_{C1}$	$\leq$	$\sigma(G(2,0) \sim G(2,1)) = 1$
$G(3,0) \sim G(3,1)$:	x_{B3}	$\leq$	$\sigma(G(3,0) \sim G(3,1)) = 1$
$G(4,0) \sim G(4,1)$:	x_{B1}	$\leq$	$\sigma(G(4,0) \sim G(4,1)) = 1$
$G(0,1) \sim G(0,2)$:	$x_{A2} + x_{C2}$	$\leq$	$\sigma(G(0,1) \sim G(0,2)) = 1$
$G(1,1) \sim G(1,2)$:	$x_{A3} + x_{C3}$	$\leq$	$\sigma(G(1,1) \sim G(1,2)) = 1$
$G(2,1) \sim G(2,2)$:	$x_{A1} + x_{A4} + x_{C1} + x_{C4}$	$\leq$	$\sigma(G(2,1) \sim G(2,2)) = 1$
$G(0,2) \sim G(0,3)$:	$x_{A2} + x_{A4}$	$\leq$	$\sigma(G(0,2) \sim G(0,3)) = 1$
$G(1,2) \sim G(1,3)$:	x_{A3}	$\leq$	$\sigma(G(1,2) \sim G(1,3)) = 1$
$G(2,2) \sim G(2,3)$:	x_{A1}	$\leq$	$\sigma(G(2,2) \sim G(2,3)) = 1$

5.7.2 Rip-Up and Reroute (RRR)

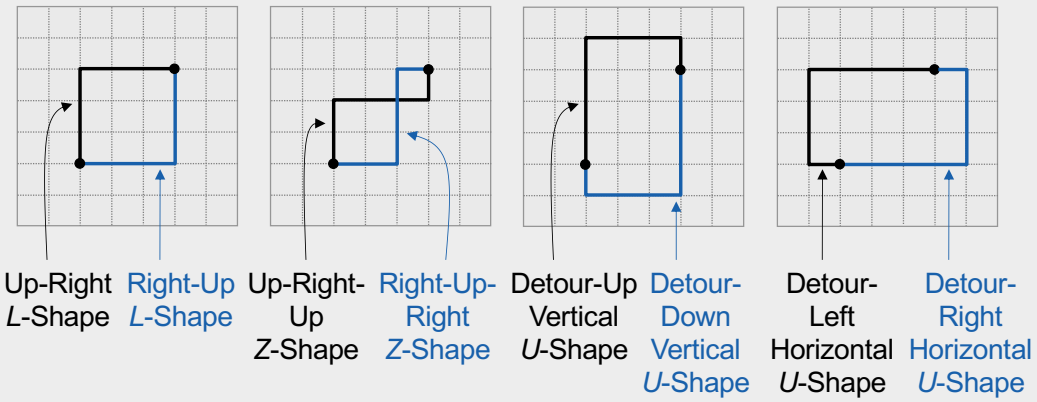
- Rip-up and reroute (RRR) framework: focuses on hard-to-route nets
- Idea: allow temporary violations, so that all nets are routed, but then iteratively remove some nets (rip-up), and route them differently (reroute)



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



- 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



- 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* φ(*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

Summary of Chapter 5 – Types of Routing

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

Summary of Chapter 5 – Types of Routing

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

Summary of Chapter 5 – Types of Routing

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