

Introduction to Electronic Design Automation

Jie-Hong Roland Jiang
江介宏

Department of Electrical Engineering
National Taiwan University



Spring 2012

1

Physical Design

High-level synthesis



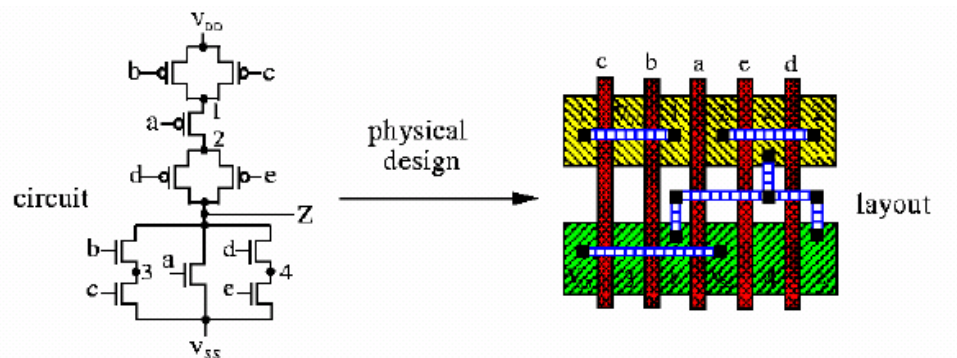
Logic synthesis



Physical design

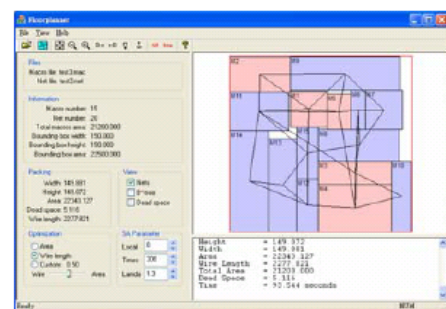
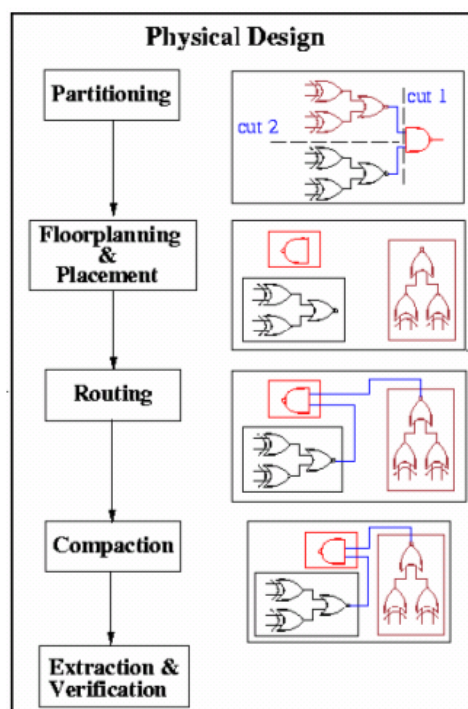
Physical Design

- ❑ Physical design converts a circuit description into a geometric description.
- ❑ The description is used to manufacture a chip.
- ❑ Physical design cycle:
 1. Logic partitioning
 2. Floorplanning and placement
 3. Routing
 4. Compaction
- ❑ Others: circuit extraction, timing verification and design rule checking



3

Physical Design Flow



B*-tree based floorplanning system



A routing system

4

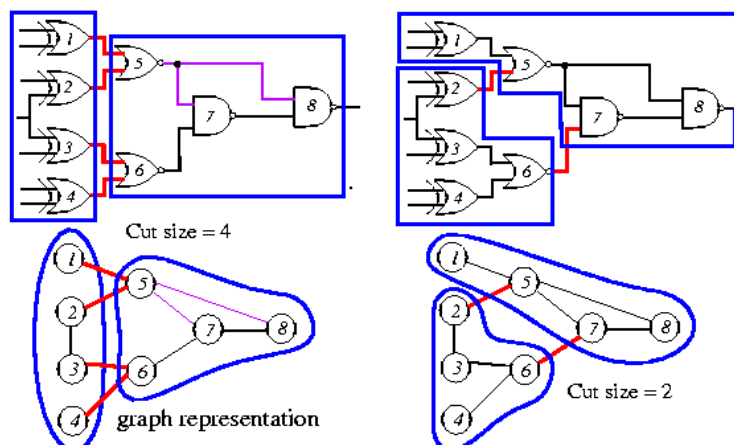
Outline

- Partitioning
- Floorplanning
- Placement
- Routing
- Compaction

5

Circuit Partitioning

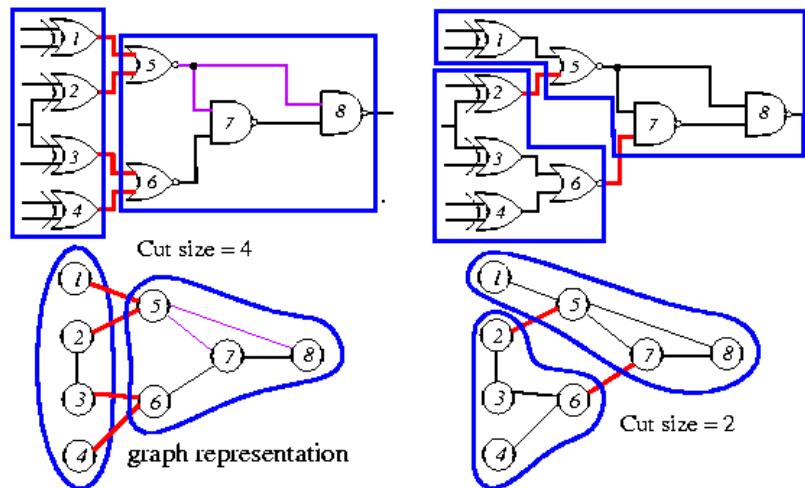
- Course contents:
 - Kernighan-Lin partitioning algorithm



6

Circuit Partitioning

- **Objective:** Partition a circuit into parts such that every component is within a prescribed range and the # of connections among the components is minimized.
 - More constraints are possible for some applications.
- Cutset? Cut size? Size of a component?



7

Problem Definition: Partitioning

- **k-way partitioning:** Given a graph $G(V, E)$, where each vertex $v \in V$ has a **size** $s(v)$ and each edge $e \in E$ has a **weight** $w(e)$, the problem is to divide the set V into k disjoint subsets $V_1, V_2, \dots, V_k$, such that an objective function is optimized, subject to certain constraints.
- **Bounded size constraint:** The size of the i -th subset is bounded by B_i (i.e., $\sum_{v \in V_i} s(v) \leq B_i$).
 - Is the partition balanced?
- **Min-cut cost between two subsets:** Minimize $\sum_{\forall e=(u,v) \wedge p(u) \neq p(v)} w(e)$, where $p(u)$ is the partition # of node u .
- The 2-way, balanced partitioning problem is NP-complete, even in its simple form with identical vertex sizes and unit edge weights.

8

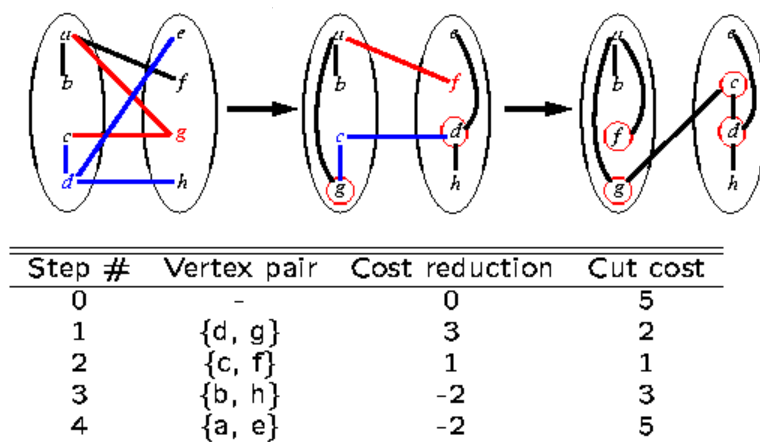
Kernighan-Lin Algorithm

- ❑ Kernighan and Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, Feb. 1970.
- ❑ An **iterative, 2-way, balanced** partitioning (bi-sectioning) heuristic.
- ❑ Till the cut size keeps decreasing
 - Vertex pairs which give the largest decrease **or the smallest increase** in cut size are exchanged.
 - These vertices are then **locked** (and thus are prohibited from participating in any further exchanges).
 - This process continues until all the vertices are locked.
 - Find the set with the largest partial sum for swapping.
 - Unlock all vertices.

9

K-L Algorithm: A Simple Example

- ❑ Each edge has a unit weight.



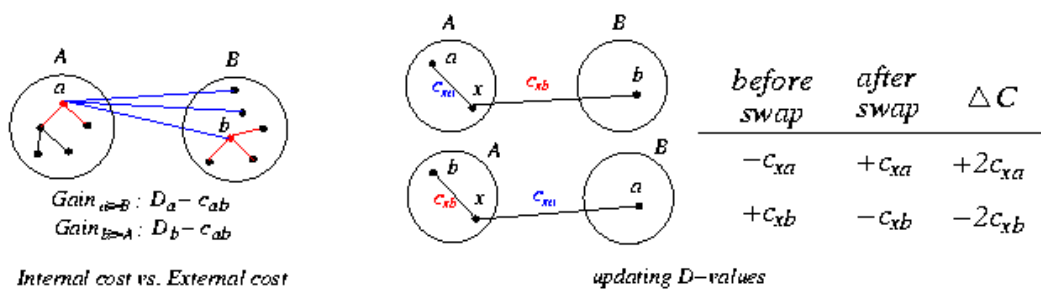
- ❑ Questions: How to compute cost reduction? What pairs to be swapped?
 - Consider the change of internal & external connections.

10

Properties

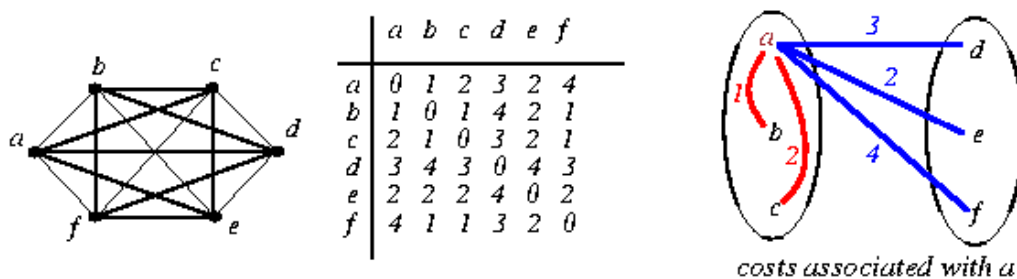
- Two sets A and B such that $|A| = n = |B|$ and $A \cap B = \emptyset$.
- External cost** of $a \in A$: $E_a = \sum_{v \in B} c_{av}$.
- Internal cost** of $a \in A$: $I_a = \sum_{v \in A} c_{av}$.
- D -value of a vertex a : $D_a = E_a - I_a$ (cost reduction for moving a).
- Cost reduction (gain) for swapping a and b : $g_{ab} = D_a + D_b - 2c_{ab}$.
- If $a \in A$ and $b \in B$ are interchanged, then the new D -values, D' , are given by

$$\begin{aligned} D'_x &= D_x + 2c_{xa} - 2c_{xb}, \forall x \in A - \{a\} \\ D'_y &= D_y + 2c_{yb} - 2c_{ya}, \forall y \in B - \{b\}. \end{aligned}$$



11

A Weighted Example



$$\text{Initial cut cost} = (3+2+4) + (4+2+1) + (3+2+1) = 22$$

Iteration 1

$I_a = 1 + 2 = 3;$	$E_a = 3 + 2 + 4 = 9;$	$D_a = E_a - I_a = 9 - 3 = 6$
$I_b = 1 + 1 = 2;$	$E_b = 4 + 2 + 1 = 7;$	$D_b = E_b - I_b = 7 - 2 = 5$
$I_c = 2 + 1 = 3;$	$E_c = 3 + 2 + 1 = 6;$	$D_c = E_c - I_c = 6 - 3 = 3$
$I_d = 4 + 3 = 7;$	$E_d = 3 + 4 + 3 = 10;$	$D_d = E_d - I_d = 10 - 7 = 3$
$I_e = 4 + 2 = 6;$	$E_e = 2 + 2 + 2 = 6;$	$D_e = E_e - I_e = 6 - 6 = 0$
$I_f = 3 + 2 = 5;$	$E_f = 4 + 1 + 1 = 6;$	$D_f = E_f - I_f = 6 - 5 = 1$

12

A Weighted Example (cont'd)

Iteration 1:

$$\begin{array}{lll}
 I_a = 1 + 2 = 3; & E_a = 3 + 2 + 4 = 9; & D_a = E_a - I_a = 9 - 3 = 6 \\
 I_b = 1 + 1 = 2; & E_b = 4 + 2 + 1 = 7; & D_b = E_b - I_b = 7 - 2 = 5 \\
 I_c = 2 + 1 = 3; & E_c = 3 + 2 + 1 = 6; & D_c = E_c - I_c = 6 - 3 = 3 \\
 I_d = 4 + 3 = 7; & E_d = 3 + 4 + 3 = 10; & D_d = E_d - I_d = 10 - 7 = 3 \\
 I_e = 4 + 2 = 6; & E_e = 2 + 2 + 2 = 6; & D_e = E_e - I_e = 6 - 6 = 0 \\
 I_f = 3 + 2 = 5; & E_f = 4 + 1 + 1 = 6; & D_f = E_f - I_f = 6 - 5 = 1
 \end{array}$$

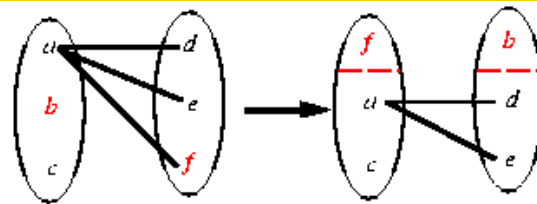
$g_{xy} = D_x + D_y - 2c_{xy}$

$$\begin{array}{ll}
 g_{ad} = D_a + D_d - 2c_{ad} = 6 + 3 - 2 \times 3 = 3 \\
 g_{ae} = 6 + 0 - 2 \times 2 = 2 \\
 g_{af} = 6 + 1 - 2 \times 4 = -1 \\
 g_{bd} = 5 + 3 - 2 \times 4 = 0 \\
 g_{be} = 5 + 0 - 2 \times 2 = 1 \\
 g_{bf} = 5 + 1 - 2 \times 1 = 4 \text{ (maximum)} \quad (\hat{g}_1 = 4) \\
 g_{cd} = 3 + 3 - 2 \times 3 = 0 \\
 g_{ce} = 3 + 0 - 2 \times 2 = -1 \\
 g_{cf} = 3 + 1 - 2 \times 1 = 2
 \end{array}$$

Swap b and f .

13

A Weighted Example (cont'd)



$D'_x = D_x + 2c_{xp} - 2c_{xq}, \forall x \in A - \{p\}$ (swap p and $q, p \in A, q \in B$)

$$\begin{array}{ll}
 D'_a = D_a + 2c_{ab} - 2c_{af} = 6 + 2 \times 1 - 2 \times 4 = 0 \\
 D'_c = D_c + 2c_{cb} - 2c_{cf} = 3 + 2 \times 1 - 2 \times 1 = 3 \\
 D'_d = D_d + 2c_{df} - 2c_{db} = 3 + 2 \times 3 - 2 \times 4 = 1 \\
 D'_e = D_e + 2c_{ef} - 2c_{eb} = 0 + 2 \times 2 - 2 \times 2 = 0
 \end{array}$$

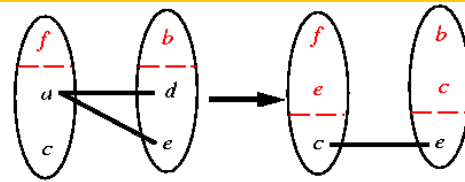
$g_{xy} = D'_x + D'_y - 2c_{xy}$

$$\begin{array}{ll}
 g_{ad} = D'_a + D'_d - 2c_{ad} = 0 + 1 - 2 \times 3 = -5 \\
 g_{ae} = D'_a + D'_e - 2c_{ae} = 0 + 0 - 2 \times 2 = -4 \\
 g_{cd} = D'_c + D'_d - 2c_{cd} = 3 + 1 - 2 \times 3 = -2 \\
 g_{ce} = D'_c + D'_e - 2c_{ce} = 3 + 0 - 2 \times 2 = -1 \text{ (maximum)} \quad (\hat{g}_2 = -1)
 \end{array}$$

Swap c and e .

14

A Weighted Example (cont'd)



$$\square D''_x = D'_x + 2c_{xp} - 2c_{xq}, \forall x \in A - \{p\}$$

$$D''_a = D'_a + 2c_{ac} - 2c_{ae} = 0 + 2 \times 2 - 2 \times 2 = 0$$

$$D''_d = D'_d + 2c_{de} - 2c_{dc} = 1 + 2 \times 4 - 2 \times 3 = 3$$

$$\square g_{xy} = D''_x + D''_y - 2c_{xy}$$

$$g_{ad} = D''_a + D''_d - 2c_{ad} = 0 + 3 - 2 \times 3 = -3 (\hat{g}_3 = -3)$$

■ Note that this step is redundant

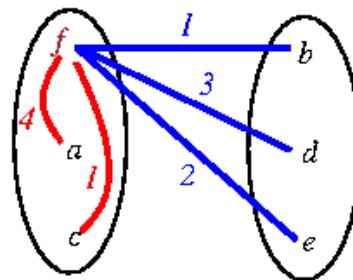
$$\square \text{Summary: } \hat{g}_1 = g_{bf} = 4, \hat{g}_2 = g_{ce} = -1, \hat{g}_3 = g_{ad} = -3. (\sum_{i=1}^n \hat{g}_i = 0).$$

$$\square \text{Largest partial sum } \max \sum_{i=1}^k \hat{g}_i = 4 \quad (k=1) \Rightarrow \text{Swap } b \text{ and } f.$$

15

A Weighted Example (cont'd)

	a	b	c	d	e	f
a	0	1	2	3	2	4
b	1	0	1	4	2	1
c	2	1	0	3	2	1
d	3	4	3	0	4	3
e	2	2	2	4	0	2
f	4	1	1	3	2	0



$$\text{Initial cut cost} = (1+3+2) + (1+3+2) + (1+3+2) = 18 \quad (22-4)$$

□ Iteration 2: Repeat what we did at Iteration 1
(Initial cost = $22-4=18$).

$$\square \text{Summary: } \hat{g}_1 = g_{ce} = -1, \hat{g}_2 = g_{ab} = -3, \hat{g}_3 = g_{fd} = 4.$$

$$\square \text{Largest partial sum} = \max \sum_{i=1}^k \hat{g}_i = 0 \quad (k=3) \Rightarrow \text{Stop!}$$

16

Kernighan-Lin Algorithm

Algorithm: Kernighan-Lin(G)

Input: $G = (V, E), |V| = 2n$.

Output: Balanced bi-partition A and B with “small” cut cost.

```
1 begin
2 Bipartition  $G$  into  $A$  and  $B$  such that  $|V_A| = |V_B|$ ,  $V_A \cap V_B = \emptyset$ ,
  and  $V_A \cup V_B = V$ .
3 repeat
4   Compute  $D_v, \forall v \in V$ .
5   for  $i = 1$  to  $n$  do
6     Find a pair of unlocked vertices  $v_{ai} \in V_A$  and  $v_{bi} \in V_B$  whose
      exchange makes the largest decrease or smallest increase in
      cut cost;
7     Mark  $v_{ai}$  and  $v_{bi}$  as locked, store the gain  $\hat{g}_i$ , and compute
      the new  $D_v$ , for all unlocked  $v \in V$ ;
8   Find  $k$ , such that  $G_k = \sum_{i=1}^k \hat{g}_i$  is maximized;
9   if  $G_k > 0$  then
10    Move  $v_{a1}, \dots, v_{ak}$  from  $V_A$  to  $V_B$  and  $v_{b1}, \dots, v_{bk}$  from  $V_B$  to  $V_A$ ;
11  Unlock  $v, \forall v \in V$ .
12 until  $G_k \leq 0$ ;
13 end
```

17

Time Complexity

- ❑ Line 4: Initial computation of D : $O(n^2)$
- ❑ Line 5: The **for**-loop: $O(n)$
- ❑ The body of the loop: $O(n^2)$.
 - Lines 6--7: Step i takes $(n - i + 1)^2$ time.
- ❑ Lines 4--11: Each pass of the repeat loop: $O(n^3)$.
- ❑ Suppose the repeat loop terminates after r passes.
- ❑ The total running time: $O(rn^3)$.
 - Polynomial-time algorithm?

18

Extensions of K-L Algorithm

□ Unequal sized subsets (assume $n_1 < n_2$)

1. Partition: $|A| = n_1$ and $|B| = n_2$.
2. Add $n_2 - n_1$ dummy vertices to set A. Dummy vertices have no connections to the original graph.
3. Apply the Kernighan-Lin algorithm.
4. Remove all dummy vertices.

□ Unequal sized “vertices”

1. Assume that the smallest “vertex” has unit size.
2. Replace each vertex of size s with s vertices which are fully connected with edges of infinite weight.
3. Apply the Kernighan-Lin algorithm.

□ k -way partition

1. Partition the graph into k equal-sized sets.
2. Apply the Kernighan-Lin algorithm for each pair of subsets.
3. Time complexity? Can be reduced by recursive bi-partition.

19

Outline

□ Partitioning

□ Floorplanning

□ Placement

□ Routing

□ Compaction

20

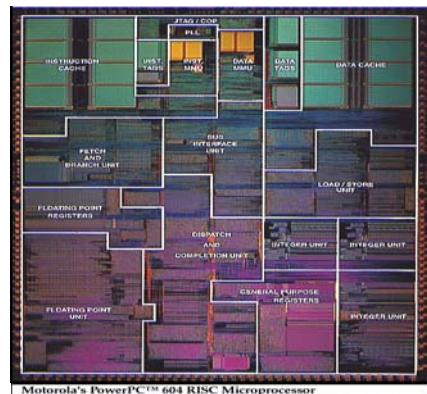
Floorplanning

□ Course contents

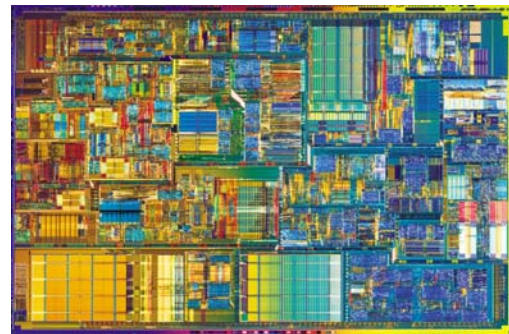
- Floorplan basics
- Normalized Polish expression for slicing floorplans
- B*-trees for non-slicing floorplans

□ Reading

- ## ■ Chapter 10



PowerPC 604



Pentium 4

21

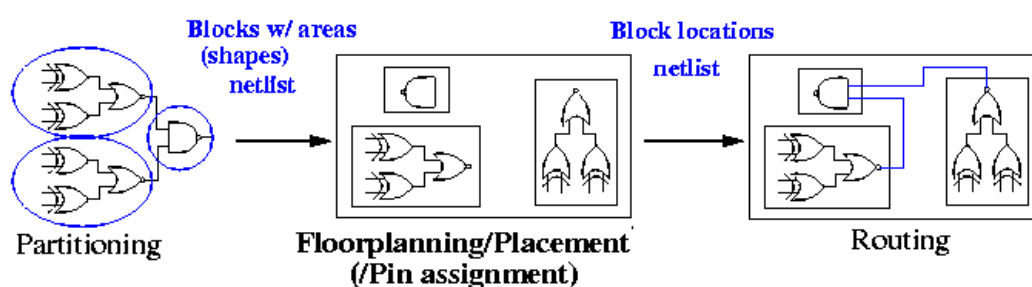
Floorplanning

- Partitioning leads to

- Blocks with well-defined **areas and shapes** (**rigid/hard** blocks).
- Blocks with approximate areas and no particular shapes (**flexible/soft** blocks).
- A **netlist** specifying connections between the blocks.

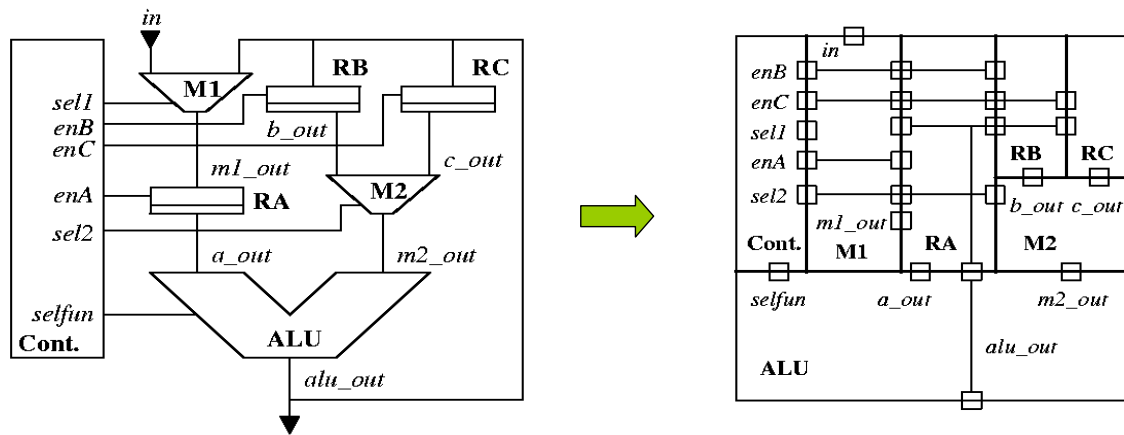
Objectives

- Find **locations** for all blocks.
- Consider shapes of soft block and pin locations of all the blocks.



22

Early Layout Decision Example



23

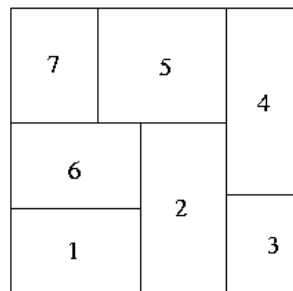
Early Layout Decision Methodology

- ❑ An integrated circuit is essentially a two-dimensional medium; taking this aspect into account in early stages of the design helps in creating designs of good quality.
- ❑ Floorplanning gives early feedback: thinking of layout at early stages may suggest valuable architectural modifications; floorplanning also aids in estimating delay due to wiring.
- ❑ Floorplanning fits very well in a *top-down* design strategy, the *step-wise refinement* strategy also propagated in software design.
- ❑ Floorplanning assumes, however, *flexibility* in layout design, the existence of cells that can adapt their shapes and terminal locations to the environment.

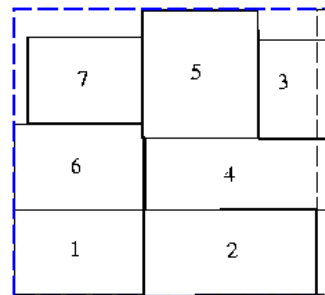
24

Floorplanning Problem

- Inputs to the floorplanning problem:
 - A set of blocks, hard or soft.
 - Pin locations of hard blocks.
 - A netlist.
- Objectives: minimize **area**, reduce **wirelength** for (critical) nets, maximize **routability** (minimize **congestion**), determine shapes of soft blocks, etc.



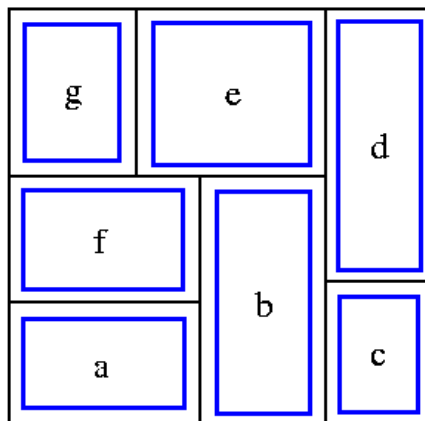
An optimal floorplan,
in terms of area

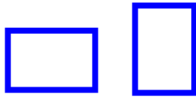


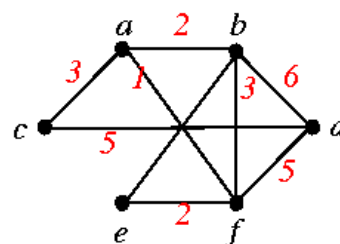
A non-optimal floorplan

25

Floorplan Design



- Modules: 
- Area: $A=xy$
- Aspect ratio: $r \leq y/x \leq s$
- Rotation: 
- Module connectivity

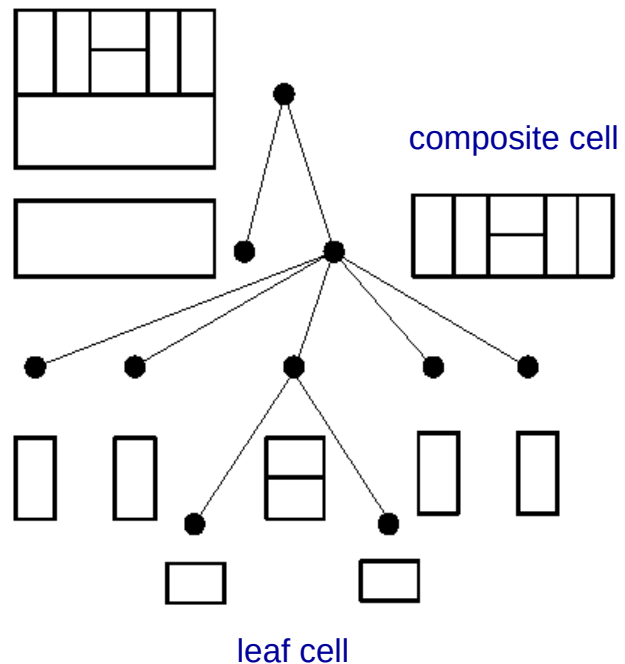


26

Floorplanning Concepts

- **Leaf cell (block / module):** a cell at the lowest level of the hierarchy; it does not contain any other cell.

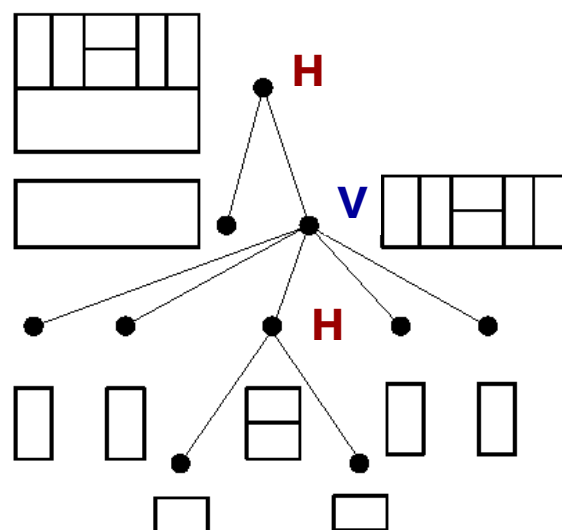
- **Composite cell (block / module):** a cell that is composed of either leaf cells or composite cells. The entire IC is the highest-level composite cell.



27

Slicing Floorplan + Slicing Tree

- A composite cell's subcells are obtained by a horizontal or vertical *bisection* of the composite cell.
- Slicing floorplans can be represented by a **slicing tree**.
- In a slicing tree, all cells (except for the top-level cell) have a *parent*, and all composite cells have *children*.
- A slicing floorplan is also called a floorplan of **order 2**.



H: horizontal cut

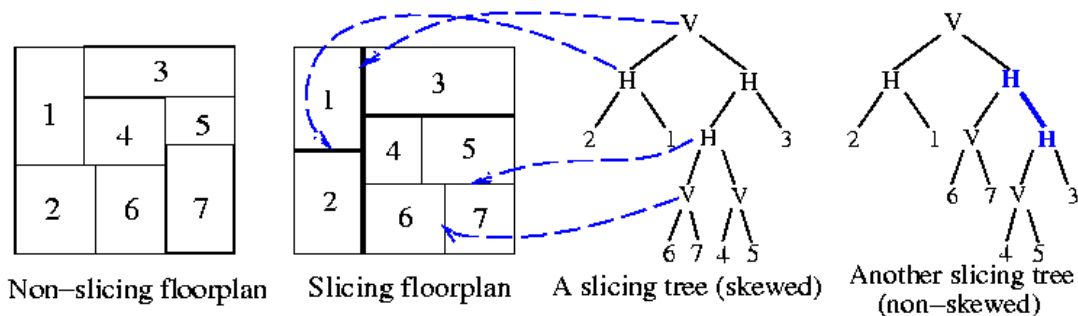
V: vertical cut

different from the definitions in the textbook!!

28

Skewed Slicing Tree

- ❑ **Rectangular dissection:** Subdivision of a given rectangle by a finite # of horizontal and vertical line segments into a finite # of non-overlapping rectangles.
- ❑ **Slicing structure:** a rectangular dissection that can be obtained by repetitively subdividing rectangles horizontally or vertically.
- ❑ **Slicing tree:** A binary tree, where each internal node represents a vertical cut line or horizontal cut line, and each leaf a basic rectangle.
- ❑ **Skewed slicing tree:** One in which no node and its **right** child are the same.



29

Slicing Floorplan Design by Simulated Annealing

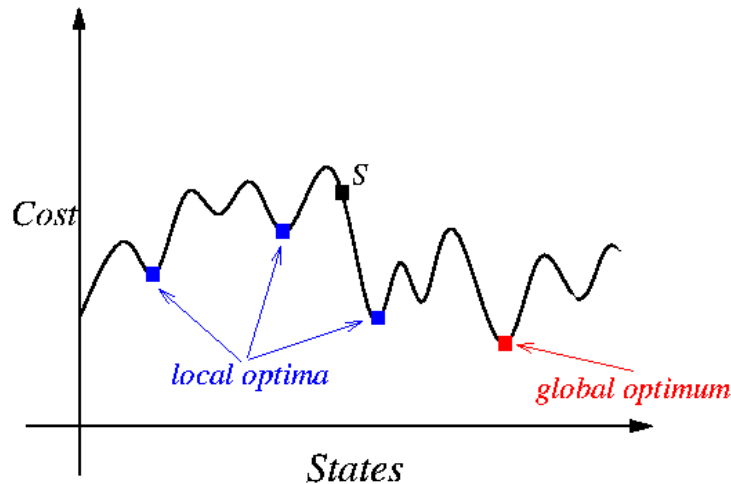
❑ Related work

- Wong & Liu, "A new algorithm for floorplan design," DAC-86.
 - ❑ Considers slicing floorplans.
- Wong & Liu, "Floorplan design for rectangular and L-shaped modules," ICCAD'87.
 - ❑ Also considers L-shaped modules.
- Wong, Leong, Liu, *Simulated Annealing for VLSI Design*, pp. 31--71, Kluwer Academic Publishers, 1988.

30

Simulated Annealing

- Kirkpatrick, Gelatt, and Vecchi, "Optimization by simulated annealing," *Science*, May 1983.
- Greene and Supowit, "Simulated annealing without rejected moves," ICCD-84.



31

Simulated Annealing Basics

- Non-zero probability for "up-hill" moves.
- Probability depends on
 1. magnitude of the "up-hill" movement
 2. total search time

$$Prob(S \rightarrow S') = \begin{cases} 1 & \text{if } \Delta C \leq 0 \quad /* \text{"down-hill" moves} */ \\ e^{-\frac{\Delta C}{T}} & \text{if } \Delta C > 0 \quad /* \text{"up-hill" moves} */ \end{cases}$$

- $\Delta C = cost(S') - Cost(S)$
- T : Control parameter (temperature)
- Annealing schedule: $T = T_0, T_1, T_2, \dots$, where $T_i = r^i T_0$ with $r < 1$.

32

Generic Simulated Annealing Algorithm

```
1 begin
2 Get an initial solution S;
3 Get an initial temperature  $T > 0$ ;
4 while not yet "frozen" do
5   for  $1 \leq i \leq P$  do
6     Pick a random neighbor  $S'$  of  $S$ ;
7      $\Delta \leftarrow \text{cost}(S') - \text{cost}(S)$ ;
8     /* downhill move */
9     if  $\Delta \leq 0$  then  $S \leftarrow S'$ 
10    /* uphill move */
11    if  $\Delta > 0$  then  $S \leftarrow S'$  with probability  $e^{-\frac{\Delta}{T}}$ ;
12   $T \leftarrow rT$ ; /* reduce temperature */
13 return S
14 end
```

33

Basic Ingredients for Simulated Annealing

□ Analogy:

Physical system	Optimization problem
state	configuration
energy	cost function
ground state	optimal solution
quenching	iterative improvement
careful annealing	simulated annealing

□ Basic Ingredients for Simulated Annealing:

- **Solution space**
- **Neighborhood structure**
- **Cost function**
- **Annealing schedule**

34

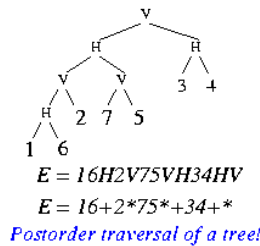
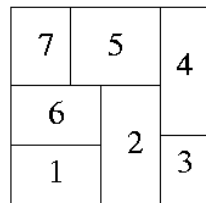
Solution Representation of Slicing Floorplan

- An expression $E = e_1 e_2 \dots e_{2n-1}$, where $e_i \in \{1, 2, \dots, n, H, V\}$, $1 \leq i \leq 2n-1$, is a **Polish expression** of length $2n-1$ iff
 1. every operand j , $1 \leq j \leq n$, appears exactly once in E ;
 2. (the **balloting property**) for every subexpression $E_i = e_1 \dots e_i$, $1 \leq i \leq 2n-1$, # operands $>$ # operators.

1 6 H 3 5 V 2 H V 7 4 H V

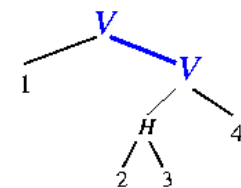
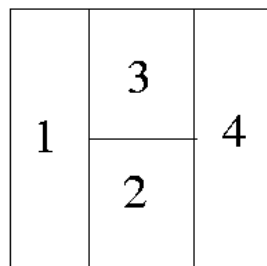
of operands = 4 = 7
of operators = 2 = 5

- Polish expression $\leftrightarrow$ Postorder traversal.
- ijH : rectangle i on bottom of j ; ijV : rectangle i on the left of j .

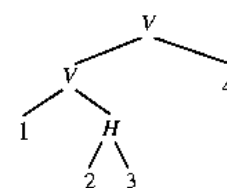


35

Redundant Representations

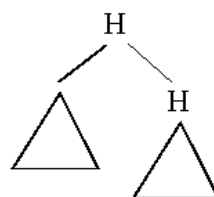


$E = 123H4VV$
non-skewed!

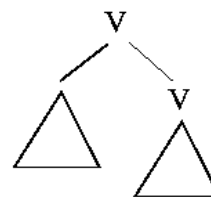


$E = 123HV4V$
skewed!

Non-skewed cases



..... HH



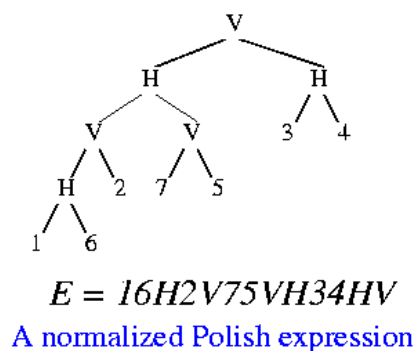
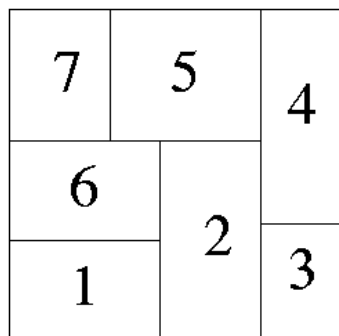
..... VV

- **Question:** How to eliminate ambiguous representation?

36

Normalized Polish Expression

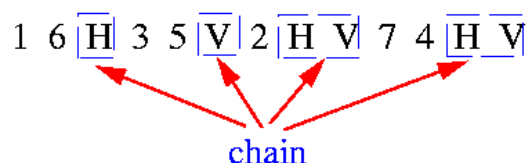
- A Polish expression $E = e_1 e_2 \dots e_{2n-1}$ is called **normalized** iff E has no consecutive operators of the same type (H or V), i.e. skewed.
- Given a **normalized Polish expression**, we can construct a **unique** rectangular slicing structure.



37

Neighborhood Structure

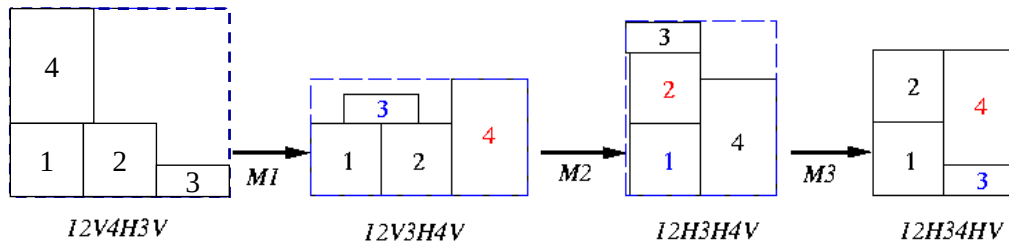
- **Chain:** $HVHVH \dots$ or $VHVHV \dots$



- **Adjacent:** 1 and 6 are adjacent operands; 2 and 7 are adjacent operands; 5 and V are adjacent operand and operator.
- 3 types of moves:
 - **M1 (Operand Swap):** Swap two adjacent operands.
 - **M2 (Chain Invert):** Complement some chain ($\overline{V} = H, \overline{H} = V$).
 - **M3 (Operator/Operand Swap):** Swap two adjacent operand and operator.

38

Effects of Perturbation

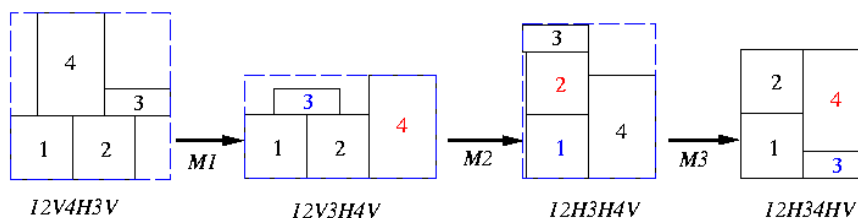


- **Question:** The balloting property holds during the moves?
 - M1 and M2 moves are OK.
 - **Check the M3 moves!** Reject “illegal” M3 moves.
- **Check M3 moves:** Assume that the M3 move swaps the operand e_i with the operator e_{i+1} , $1 \leq i \leq k-1$. Then, the swap will not violate the balloting property iff $2N_{i+1} < i$.
 - N_k : # of operators in the Polish expression $E = e_1 e_2 \dots e_k$, $1 \leq k \leq 2n-1$

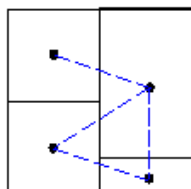
39

Cost Function

- $\phi = A + \lambda W$.
 - A: area of the smallest rectangle
 - W: overall wiring length
 - λ : user-specified parameter



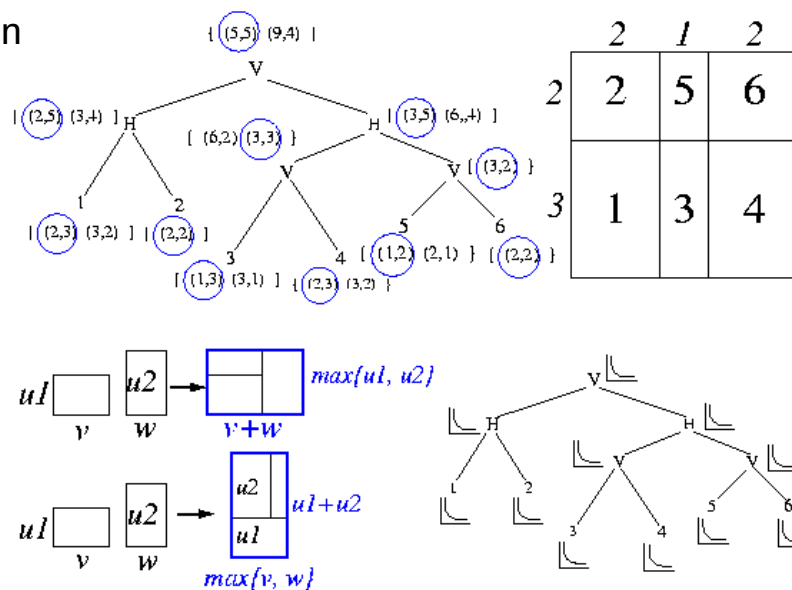
- $W = \sum_{ij} c_{ij} d_{ij}$.
 - c_{ij} : # of connections between blocks i and j .
 - d_{ij} : center-to-center distance between basic rectangles i and j .



40

Area Computation for Hard Blocks

Allow rotation



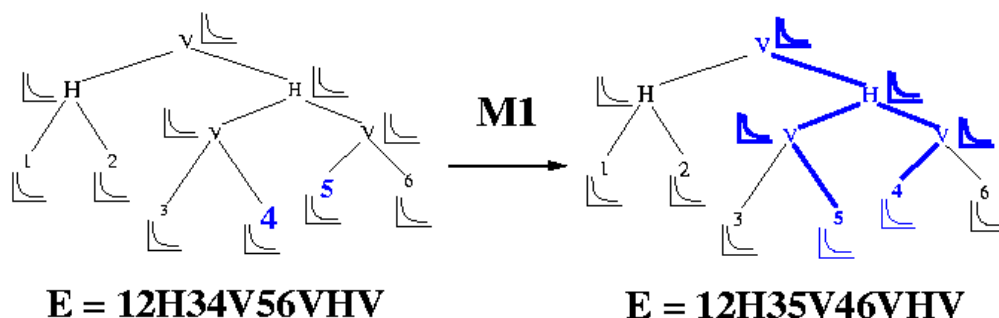
Wiring cost?

- Center-to-center interconnection length

41

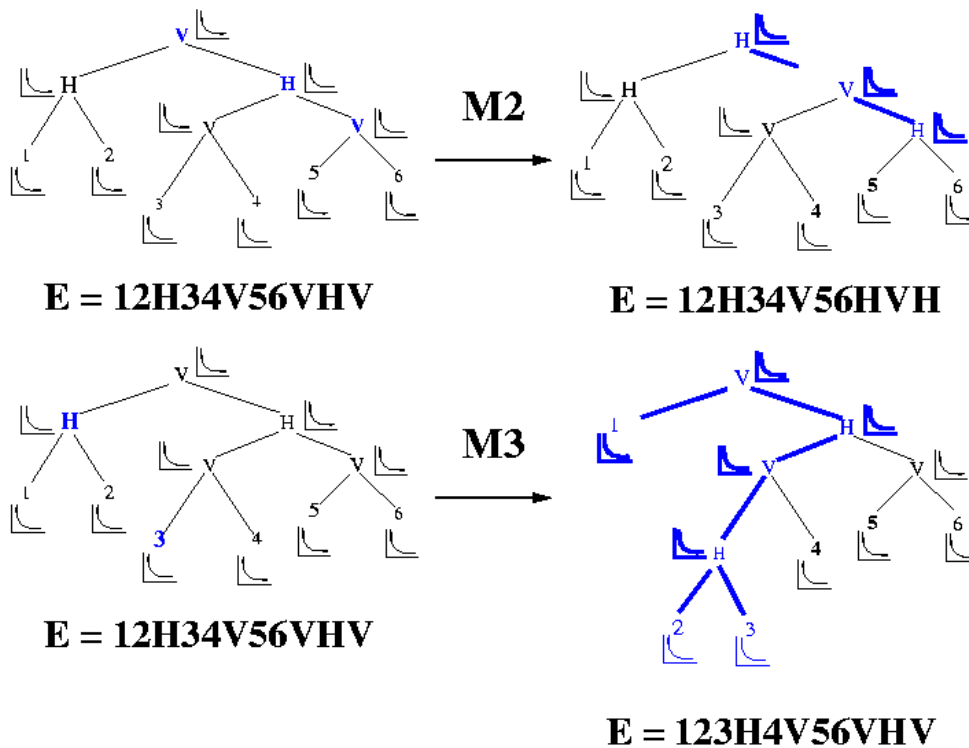
Incremental Computation of Cost Function

- Each move leads to only a minor modification of the Polish expression.
- At most **two paths** of the slicing tree need to be updated for each move.



42

Incremental Computation of Cost Function (cont'd)



43

Annealing Schedule

□ Initial solution: $12V3V \dots nV$.

1	2	3		n
---	---	---	--	---

□ $T_i = r^i T_0, i = 1, 2, 3, \dots; r = 0.85$.

□ At each temperature, try kn moves ($k = 5-10$).

□ Terminate the annealing process if

- # of accepted moves $< 5\%$,
- temperature is low enough, or
- run out of time.

44

Wong-Liu Algorithm

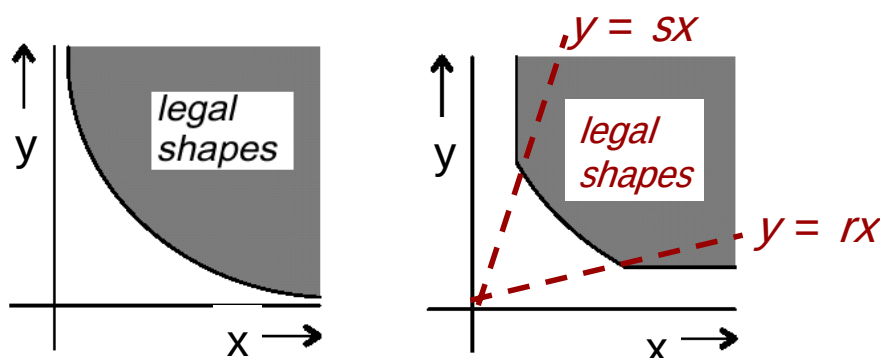
```

Input: (P,  $\varepsilon$ , r, k)
1 begin
2  $E \leftarrow 12V3V4V \dots nV$ ; /* initial solution */
3  $Best \leftarrow E$ ;  $T_0 \leftarrow \frac{\Delta_{avg}}{\ln(P)}$ ;  $M \leftarrow MT \leftarrow uphill \leftarrow 0$ ;  $N = kn$ ;
4 repeat
5    $MT \leftarrow uphill \leftarrow reject \leftarrow 0$ ;
6   repeat
7     SelectMove(M);
8     Case M of
9        $M_1$ : Select two adjacent operands  $e_i$  and  $e_j$ ;  $NE \leftarrow \text{Swap}(E, e_i, e_j)$ ;
10       $M_2$ : Select a nonzero length chain C;  $NE \leftarrow \text{Complement}(E, C)$ ;
11       $M_3$ : done  $\leftarrow \text{FALSE}$ ;
12      while not (done) do
13        Select two adjacent operand  $e_i$  and operator  $e_{i+1}$ ;
14        if ( $e_{i-1} \neq e_{i+1}$ ) and ( $2 N_{i+1} < i$ ) then done  $\leftarrow \text{TRUE}$ ;
13'       Select two adjacent operator  $e_i$  and operand  $e_{i+1}$ ;
14'       if ( $e_i \neq e_{i+2}$ ) then done  $\leftarrow \text{TRUE}$ ;
15        $NE \leftarrow \text{Swap}(E, e_i, e_{i+1})$ ;
16        $MT \leftarrow MT+1$ ;  $\Delta cost \leftarrow \text{cost}(NE) - \text{cost}(E)$ ;
17       if ( $\Delta cost \leq 0$ ) or ( $\text{Random} < e^{\frac{-\Delta cost}{T}}$ )
18         then
19           if ( $\Delta cost > 0$ ) then uphill  $\leftarrow uphill + 1$ ;
20            $E \leftarrow NE$ ;
21           if  $\text{cost}(E) < \text{cost}(best)$  then  $best \leftarrow E$ ;
22           else reject  $\leftarrow reject + 1$ ;
23       until (uphill > N) or ( $MT > 2N$ );
24        $T \leftarrow rT$ ; /* reduce temperature */
25 until (reject/MT > 0.95) or ( $T < \varepsilon$ ) or OutOfTime;
26 end
  
```

45

Shape Curve

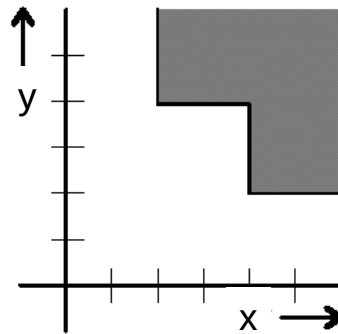
- Flexible cells imply that cells can have different aspect ratios.
- The relation between the width x and the height y is: $xy = A$, or $y = A/x$. The shape function is a hyperbola.
- Very thin cells are not interesting and often not feasible to design. The shape function is a combination of a hyperbola and two straight lines.
 - Aspect ratio: $r \leq y/x \leq s$.



46

Shape Curve (cont'd)

- ❑ Leaf cells are built from discrete transistors: it is not realistic to assume that the shape function follows the hyperbola continuously.
- ❑ In an extreme case, a cell is rigid: it can only be rotated and mirrored during floorplanning or placement.

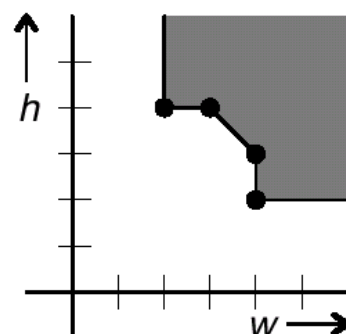


The shape function of a 2×4 inset cell.

47

Shape Curve (cont'd)

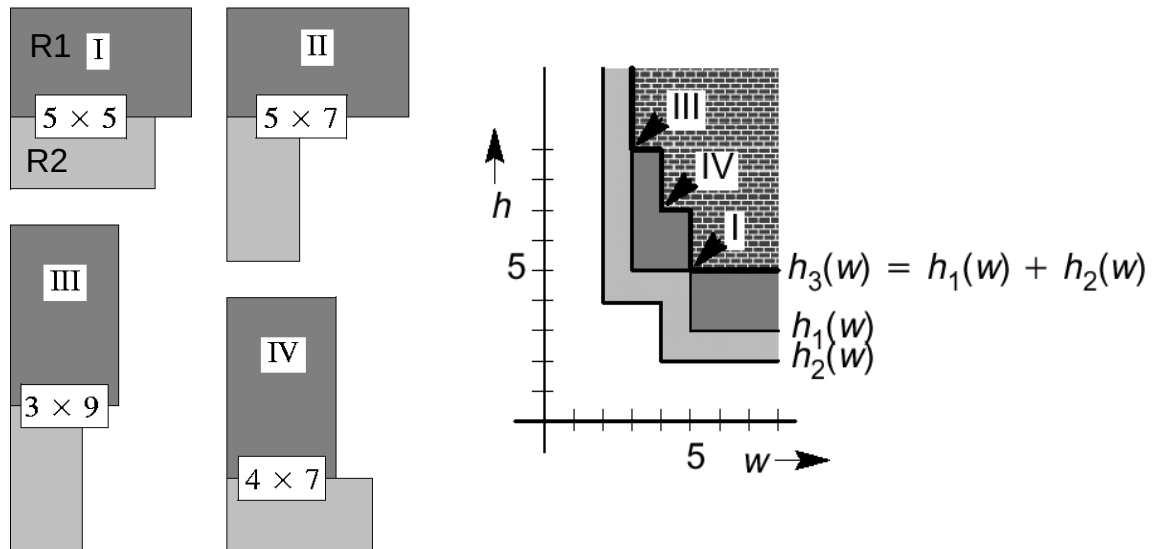
- ❑ In general, a *piecewise linear* function can be used to approximate any shape function.
- ❑ The points where the function changes its direction, are called the *corner (break) points* of the piecewise linear function.



48

Addition for Vertical Abutment

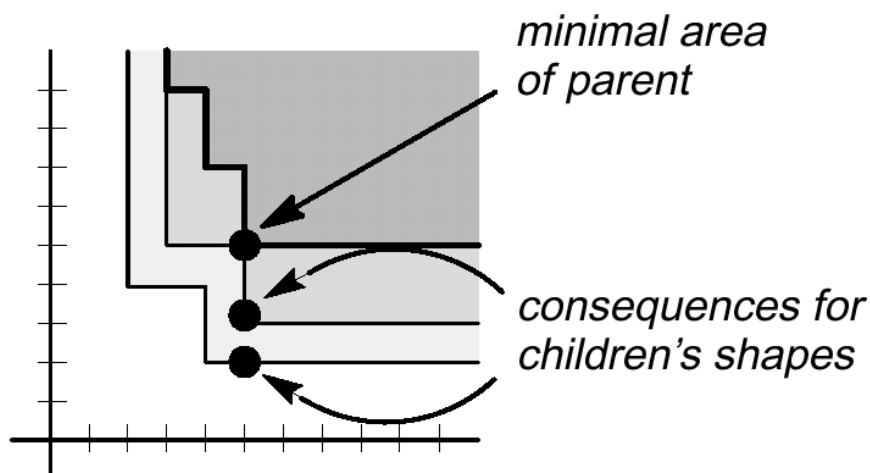
- Composition by vertical abutment $\Rightarrow$ the addition of shape functions.



49

Deriving Shapes of Children

- A choice for the minimal shape of composite cell fixes the shapes of the shapes of its children cells.



50

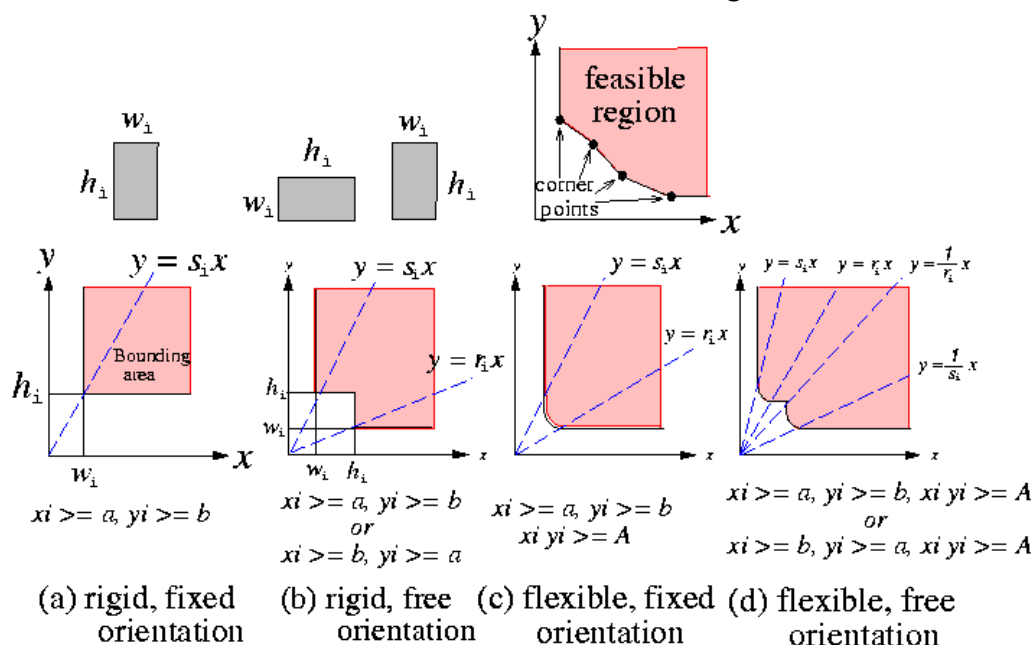
Sizing Algorithm for Slicing Floorplans

- The shape functions of all leaf cells are given as piecewise linear functions.
- Traverse the slicing tree in order to compute the shape functions of all composite cells (bottom-up composition).
- Choose the desired shape of the top-level cell; as the shape function is piecewise linear, only the break points of the function need to be evaluated, when looking for the minimal area.
- Propagate the consequences of the choice down to the leaf cells (top-down propagation).
- The sizing algorithm runs in polynomial time for slicing floorplans
 - NP-complete for non-slicing floorplans

51

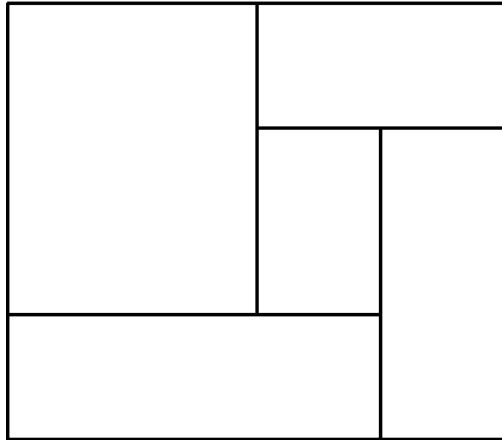
Feasible Implementations

- Shape curves correspond to different kinds of constraints where the shaded areas are feasible regions.



52

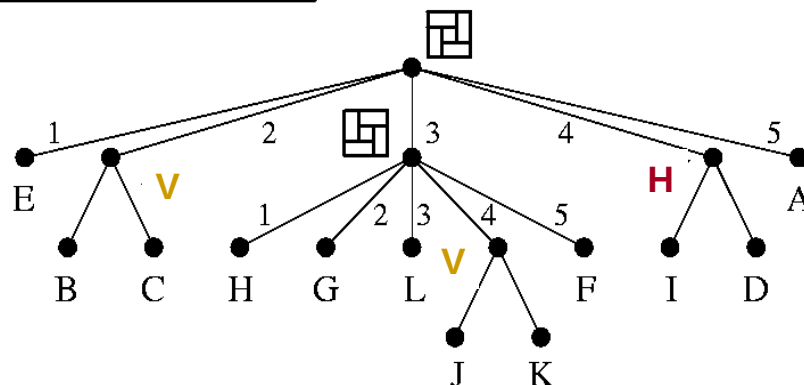
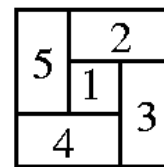
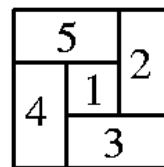
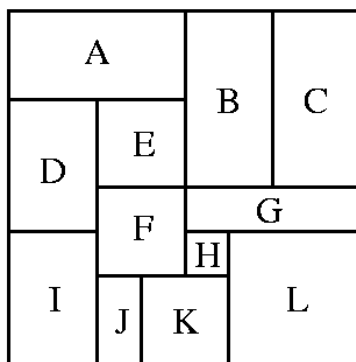
Wheel or Spiral Floorplan



- This floorplan is not slicing!
- **Wheel** is the smallest non-slicing floorplans.
- Limiting floorplans to those that have the slicing property is reasonable: it certainly facilitates floorplanning algorithms.
- Taking the shape of a wheel floorplan and its mirror image as the basis of operators leads to hierarchical descriptions of *order 5*.

53

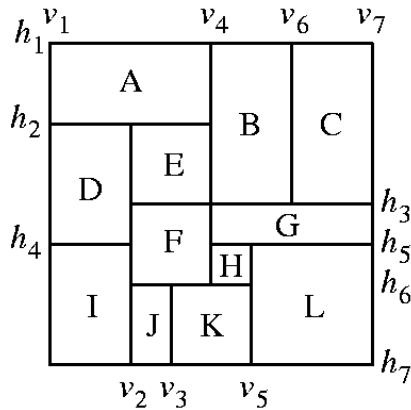
Order-5 Floorplan Examples



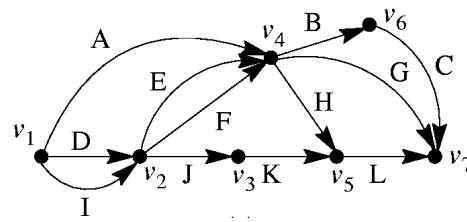
54

General Floorplan Representation: Polar Graphs

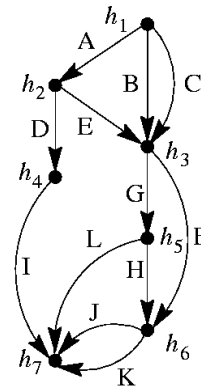
- vertex: channel segment
- edge: cell/block/module



horizontal polar graph



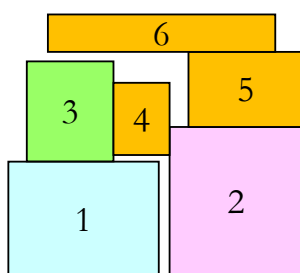
vertical polar graph



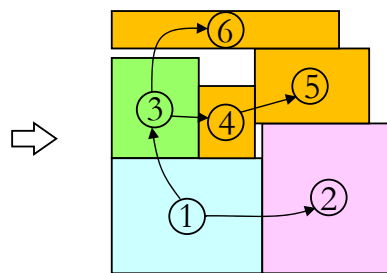
55

B*-Tree: Compacted Floorplan Representation

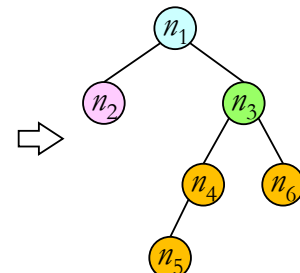
- Chang et al., "B*-tree: A new representation for non-slicing floorplans," DAC 2000.
 - Compact modules to left and bottom
 - Construct an ordered binary tree (B*-tree)
 - Left child: the lowest, adjacent block on the right ($x_j = x_i + w_i$)
 - Right child: the first block above, with the same x-coordinate ($x_j = x_i$)



A non-slicing floorplan



Compact to left and down

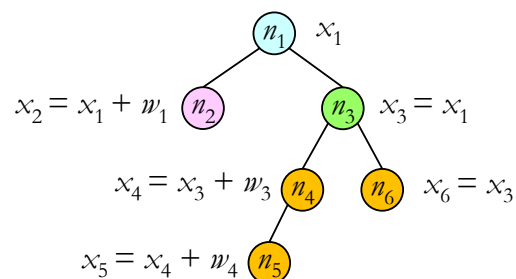
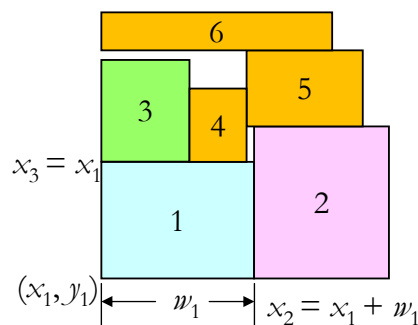


B*-tree

56

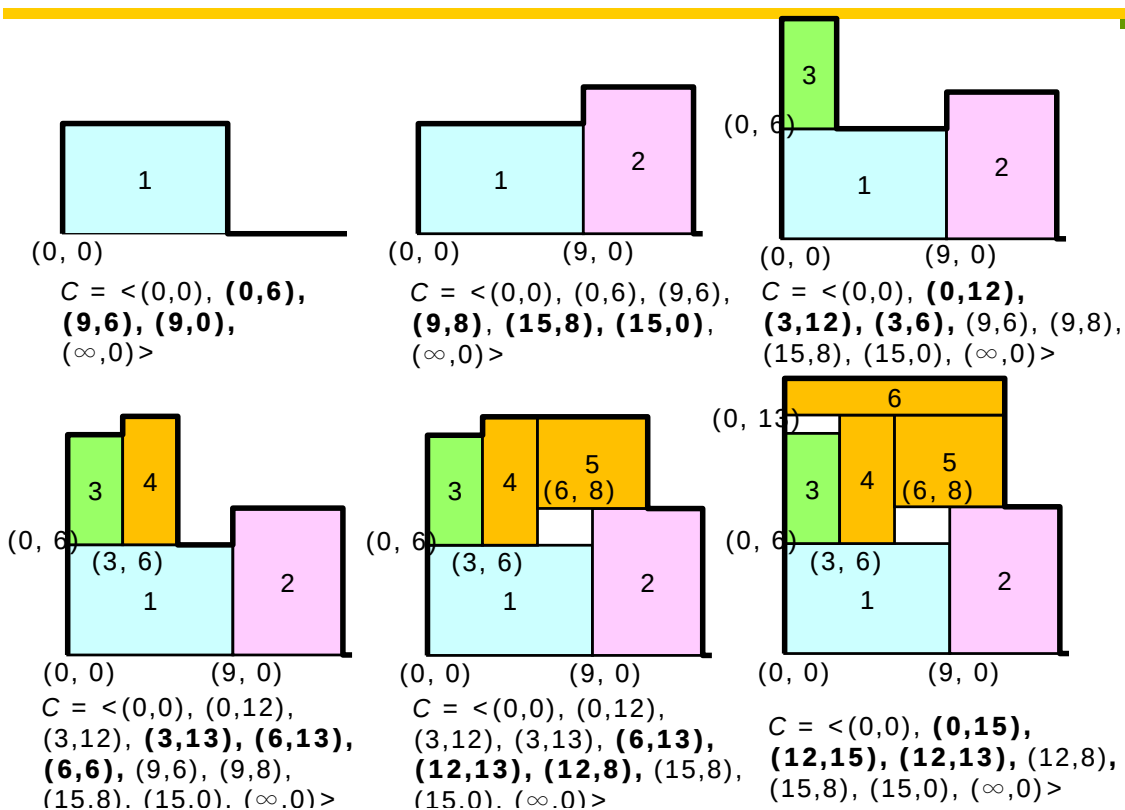
B*-tree Packing

- x-coordinates can be determined by the tree structure
 - Left child: the lowest, adjacent block on the right ($x_j = x_i + w_i$)
 - Right child: the first block above, with the same x-coordinate ($x_j = x_i$)
- Y-coordinates?
 - Horizontal contour: Use a doubly linked list to record the current maximum y-coordinate for each x-range
 - Reduce the complexity of computing a y-coordinate to amortized $O(1)$ time



57

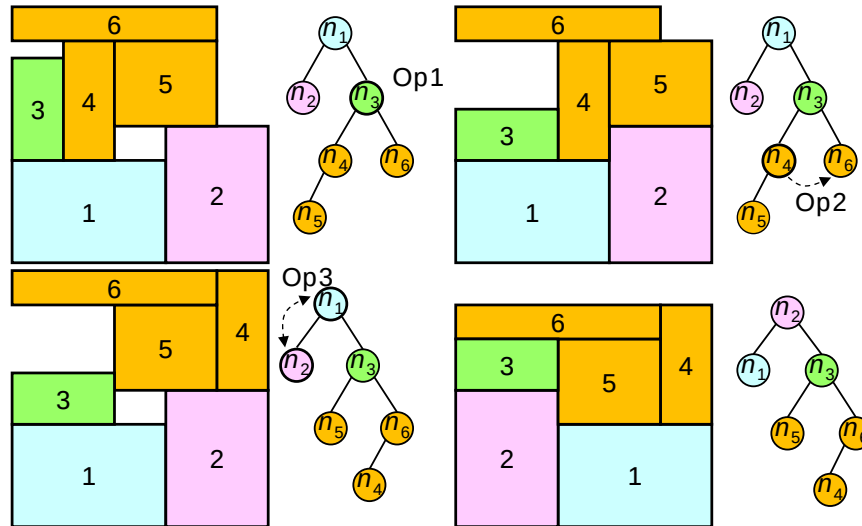
Contour Data Structure



58

B*-tree Perturbation

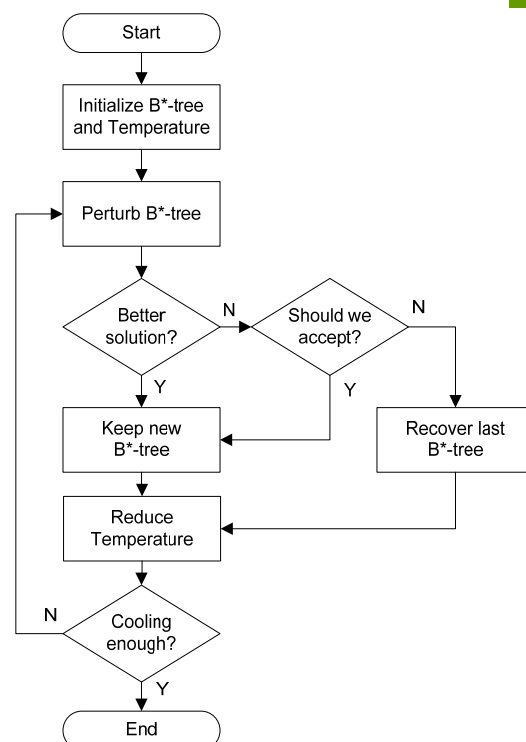
- Op1: rotate a macro
- Op2: move a node to another place
- Op3: swap two nodes



59

Simulated Annealing Using B*-tree

- The cost function is based on problem requirements



60

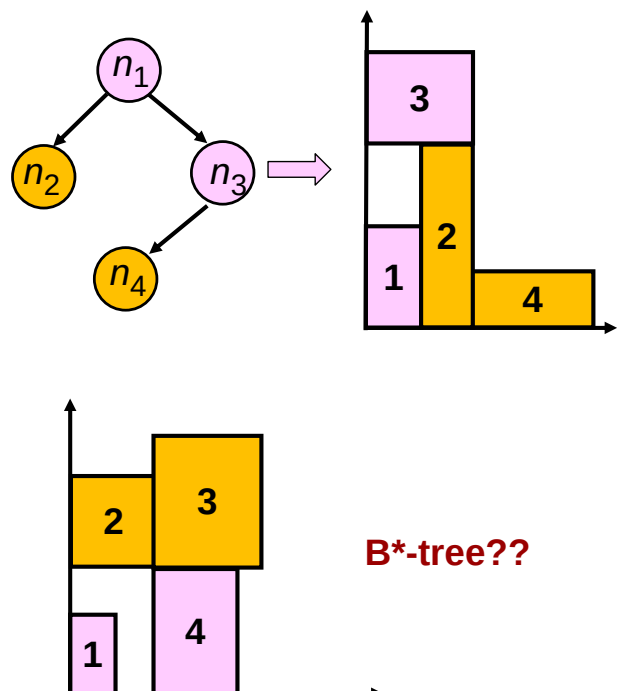
Strengths of B*-tree

- ❑ Binary tree based, efficient and easy
- ❑ Flexible to deal with various placement constraints by augmenting the B*-tree data structure (e.g., preplaced, symmetry, alignment, bus position) and rectilinear modules
- ❑ Transformation between a tree and its placement takes only linear time
- ❑ Operate on only one B*-tree (vs. two O-trees)
- ❑ Can evaluate area cost incrementally
- ❑ Smaller solution space: only $O(n! 4^n/n^{1.5})$ combinations
- ❑ Directly corresponds to hierarchical and multilevel frameworks for large-scale floorplan designs
- ❑ Can be extended to 3D floorplanning & related applications

61

Weaknesses of B*-tree

- ❑ Representation may change after packing
- ❑ Only a partially topological representation; less flexible than a fully topological representation
 - B*-tree can represent only compacted placement



62

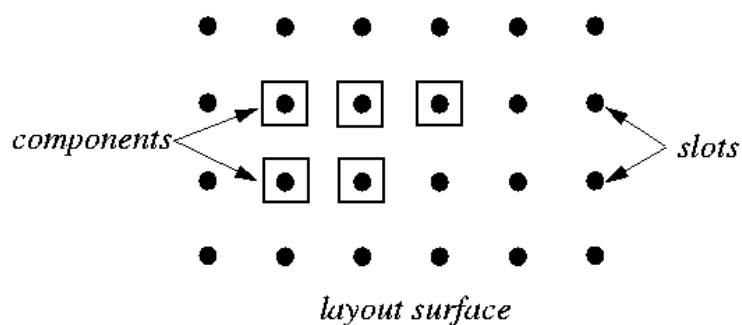
Outline

- Partitioning
- Floorplanning
- Placement
- Routing
- Compaction

63

Placement

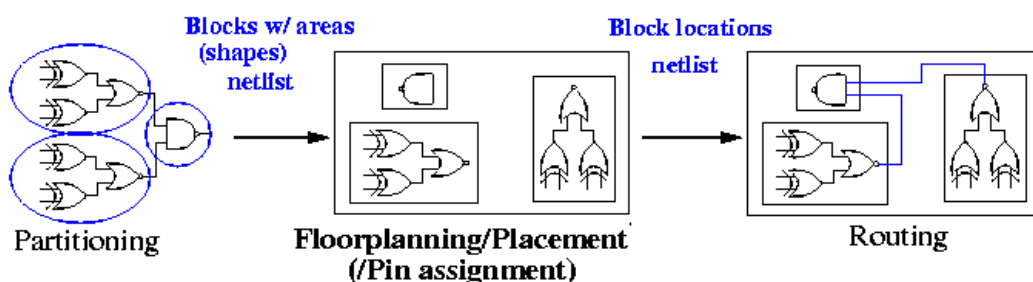
- Course contents:
 - Placement metrics
 - Constructive placement: cluster growth, min cut
 - Iterative placement: force-directed method, simulated annealing
- Reading
 - Chapter 11



64

Placement

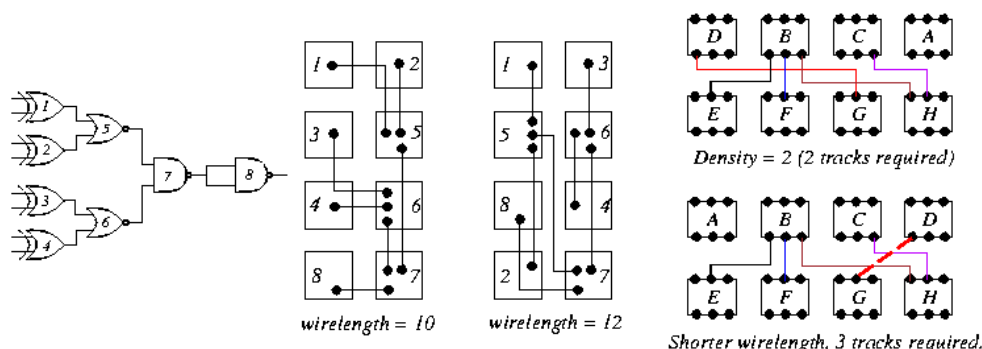
- **Placement** is the problem of automatically assigning correct positions on the chip to predesigned cells, such that some cost function is optimized.
- Inputs: A set of **fixed** cells/modules, a netlist.
- Goal: Find the best position for each cell/module on the chip according to appropriate cost functions.
 - Considerations: **routerability/channel density, wirelength**, cut size, performance, thermal issues, I/O pads.



65

Placement Objectives and Constraints

- What does a placement algorithm try to optimize?
 - total area
 - total wire length
 - number of horizontal/vertical wire segments crossing a line
- Constraints:
 - placement should be routable (no cell overlaps; no density overflow).
 - timing constraints are met (some wires should always be shorter than a given length).

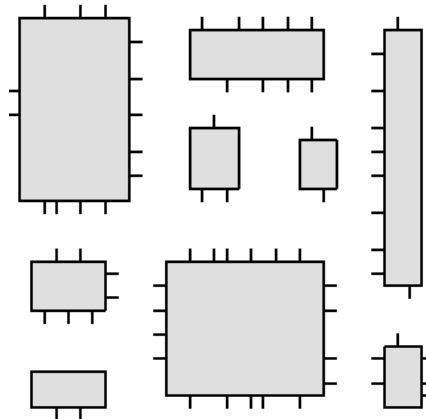


66

VLSI Placement: Building Blocks

- Different design styles create different placement problems.
 - E.g., building-block, standard-cell, gate-array placement
 - Building block: The cells to be placed have arbitrary shapes.

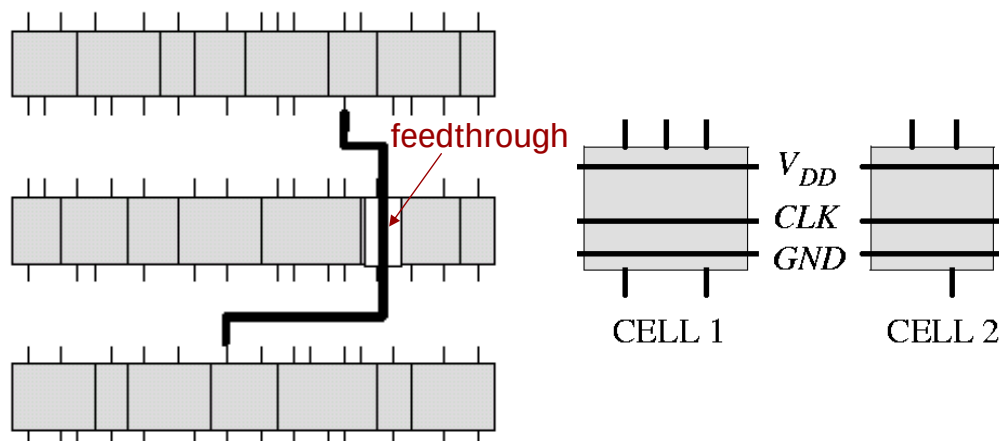
building block example



67

VLSI Placement: Standard Cells

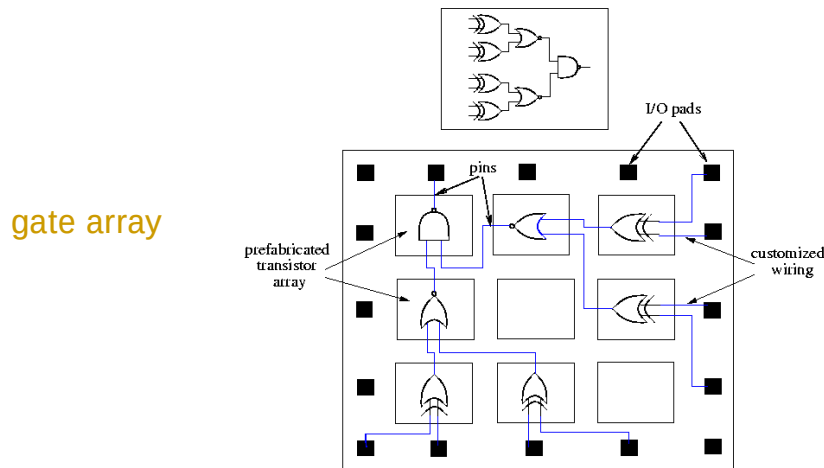
- Standard cells are designed in such a way that power and clock connections run horizontally through the cell and other I/O leaves the cell from the top or bottom sides.
- The cells are placed in rows.
- Sometimes **feedthrough** cells are added to ease wiring.



68

Consequences of Fabrication Method

- ❑ Full-custom fabrication (building block):
 - Free selection of aspect ratio (quotient of height and width).
 - Height of wiring channels can be adapted to necessity.
- ❑ Semi-custom fabrication (gate array, standard cell):
 - Placement has to deal with fixed carrier dimensions.
 - Placement should be able to deal with fixed channel capacities.



69

Relation with Routing

- ❑ Ideally, placement and routing should be performed simultaneously as they depend on each other's results. This is, however, too complicated.
 - P&R: placement and routing
- ❑ In practice placement is done prior to routing. The placement algorithm estimates the wire length of a net using some *metric*.

70

Wirelength Estimation

- **Semi-perimeter method:** Half the perimeter of the bounding rectangle that encloses all the pins of the net to be connected. Most widely used approximation!
- **Steiner-tree approximation:** Computationally expensive.
- **Minimum spanning tree:** Good approximation to Steiner trees.
- **Squared Euclidean distance:** Squares of all pairwise terminal distances in a net using a quadratic cost function

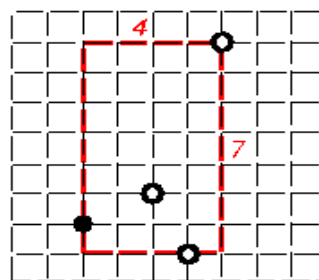
$$\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \gamma_{ij} [(x_i - x_j)^2 + (y_i - y_j)^2]$$

- **Complete graph:** Since #edges in a complete graph is $\left(\frac{n(n-1)}{2}\right)$,

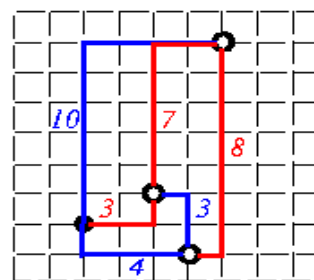
$$\text{wirelength} \approx \frac{2}{n} \sum_{(i,j) \in \text{net}} \text{dist}(i, j).$$

71

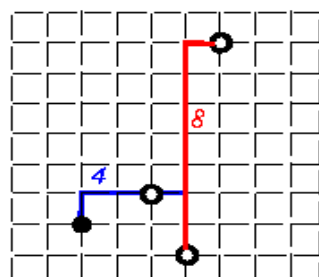
Wirelength Estimation (cont'd)



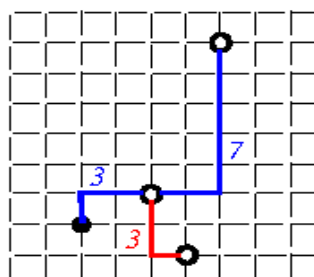
semi-perimeter len = 11



complete graph len * 2/n = 17.5



Steiner tree len = 12



Spanning tree len = 13

72

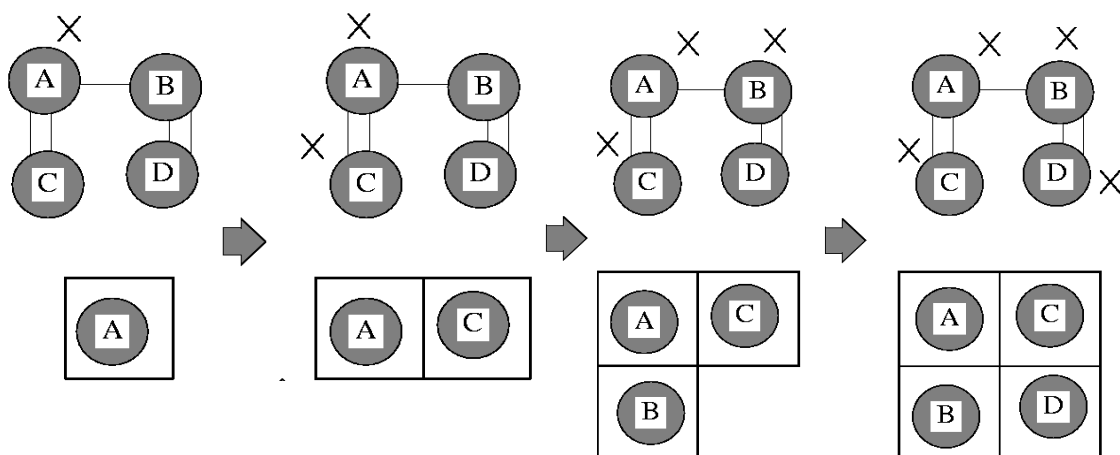
Placement Algorithms

- ❑ The placement problem is NP-complete
- ❑ Popular placement algorithms:
 - **Constructive algorithms:** once the position of a cell is fixed, it is not modified anymore.
 - ❑ Cluster growth, min cut, etc.
 - **Iterative algorithms:** intermediate placements are modified in an attempt to improve the cost function.
 - ❑ Force-directed method, etc
 - **Nondeterministic approaches:** simulated annealing, genetic algorithm, etc.
- ❑ Most approaches combine multiple elements:
 - Constructive algorithms are used to obtain an **initial placement**.
 - The initial placement is followed by an **iterative improvement** phase.
 - The results can further be improved by **simulated annealing**.

73

Bottom-Up Placement: Clustering

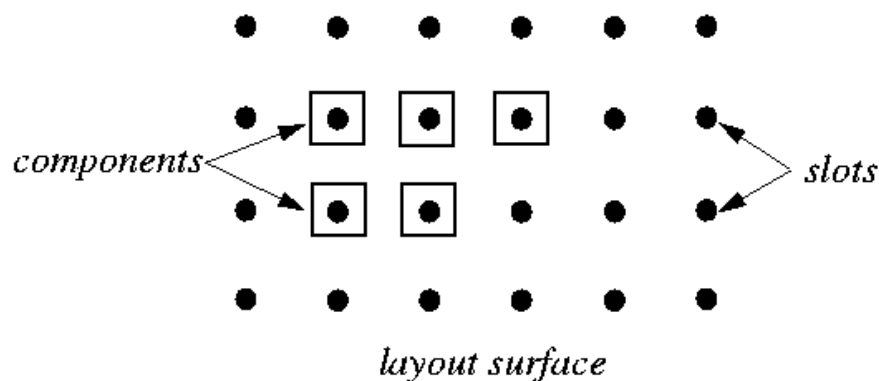
- ❑ Starts with a single cell and finds more cells that share nets with it.



74

Placement by Cluster Growth

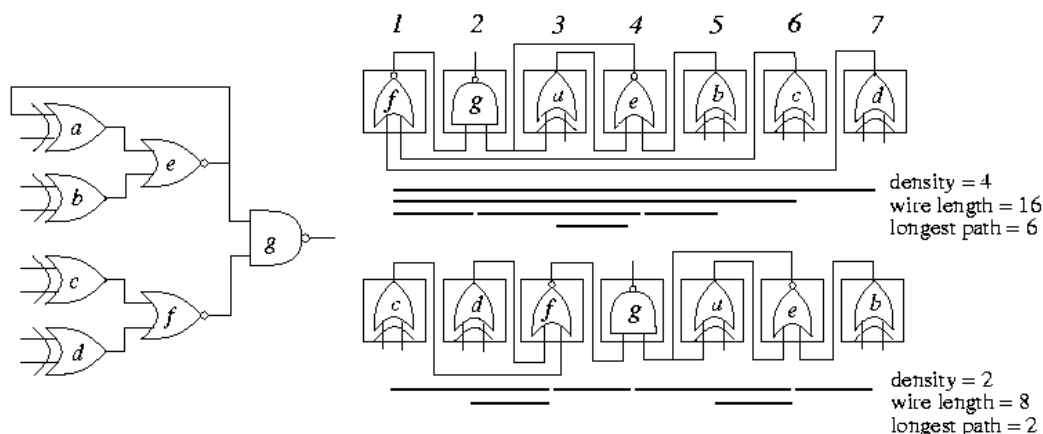
- Greedy method: Selects unplaced components and places them in available slots.
 - SELECT: Choose the unplaced component that is most strongly connected to all of the placed components (or most strongly connected to any single placed component).
 - PLACE: Place the selected component at a slot such that a certain "cost" of the partial placement is minimized.



75

Cluster Growth Example

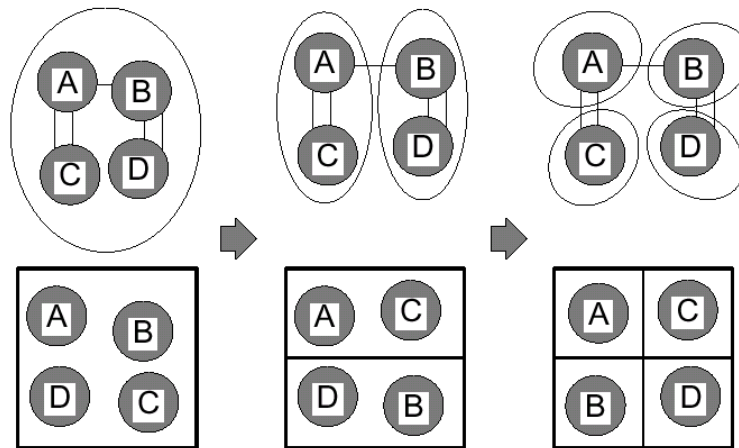
- # of other terminals connected: $c_a=3$, $c_b=1$, $c_c=1$, $c_d=1$, $c_e=4$, $c_f=3$, and $c_g=3 \Rightarrow e$ has the most connectivity.
- Place e in the center, slot 4. a , b , g are connected to e , and $\Rightarrow$ Place a next to e (say, slot 3). Continue until all cells are placed.
- Further improve the placement by swapping the gates.



76

Top-down Placement: Min Cut

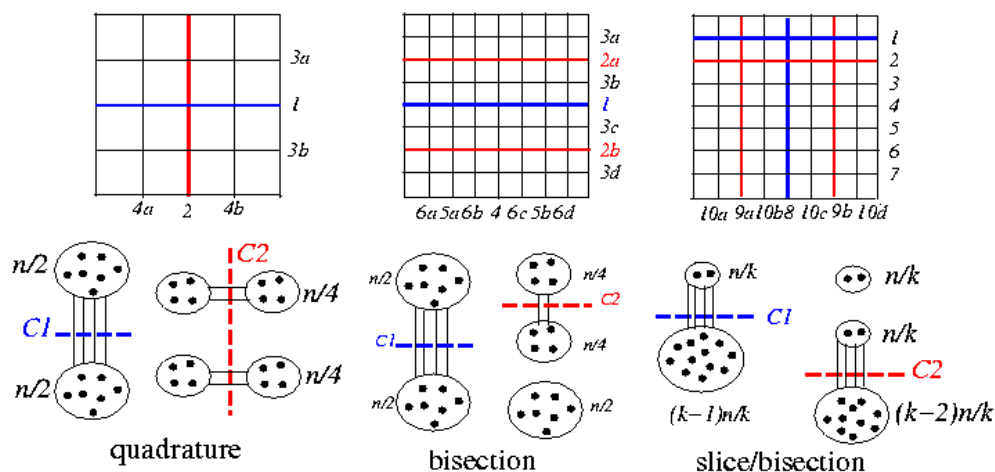
- Starts with the whole circuit and ends with small circuits.
- Recursive bipartitioning of a circuit (e.g., K&L) leads to a min-cut placement.



77

Min-Cut Placement

- Breuer, "A class of min-cut placement algorithms," DAC, 1977.
- Quadrature:** suitable for circuits with high density in the center.
- Bisection:** good for standard-cell placement.
- Slice / Bisection:** good for cells with high interconnection on the periphery.



78

Algorithm for Min-Cut Placement

Algorithm: Min_Cut_Placement(N, n, C)

/* N : the layout surface */

/* n : # of cells to be placed */

/* n_0 : # of cells in a slot */

/* C : the connectivity matrix */

1 **begin**

2 **if** ($n \leq n_0$) **then** PlaceCells(N, n, C)

3 **else**

4 (N_1, N_2) $\leftarrow$ CutSurface(N);

5 (n_1, C_1), (n_2, C_2) $\leftarrow$ Partition(n, C);

6 **Call** Min_Cut_Placement(N_1, n_1, C_1);

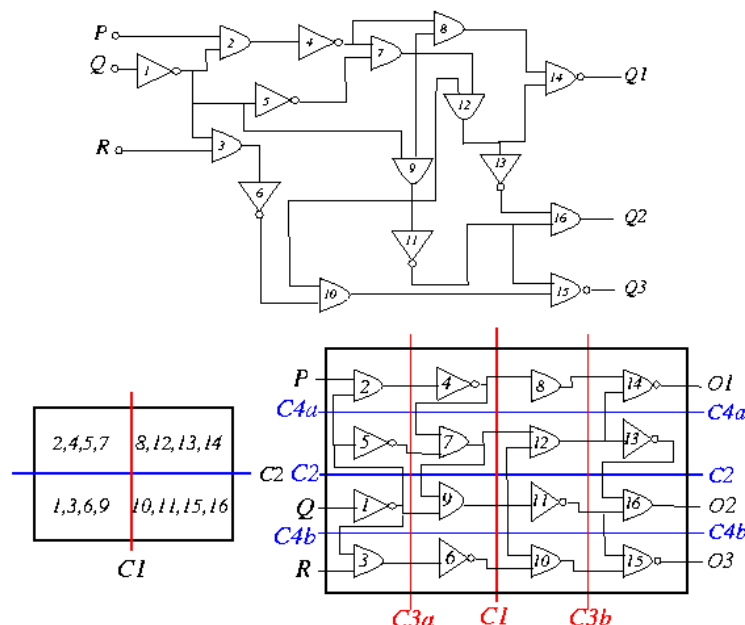
7 **Call** Min_Cut_Placement(N_2, n_2, C_2);

8 **end**

79

Quadrature Placement Example

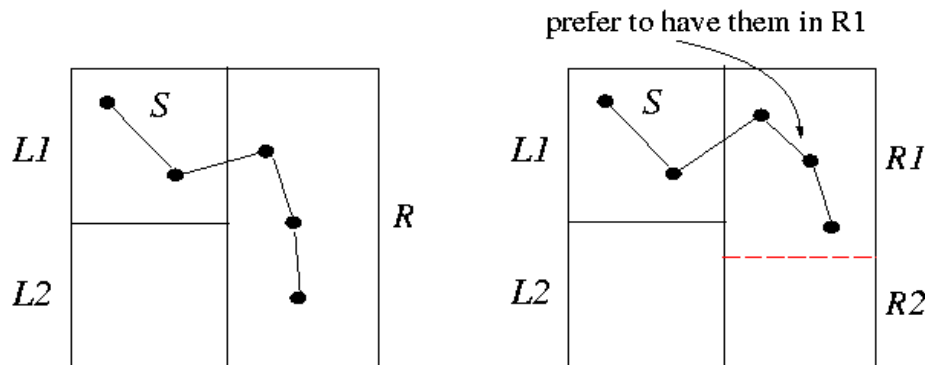
- Apply the K-L heuristic to partition + Quadrature Placement: Cost $C_1 = 4$, $C_{2L} = C_{2R} = 2$, etc.



80

Min-Cut Placement with Terminal Propagation

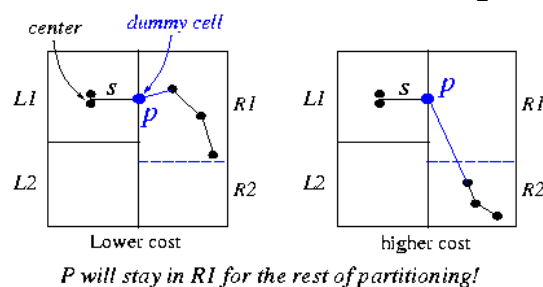
- Dunlop & Kernighan, "A procedure for placement of standard-cell VLSI circuits," *IEEE TCAD*, Jan. 1985.
- Drawback of the original min-cut placement: Does not consider the positions of terminal pins that enter a region.
 - What happens if we swap {1, 3, 6, 9} and {2, 4, 5, 7} in the previous example?



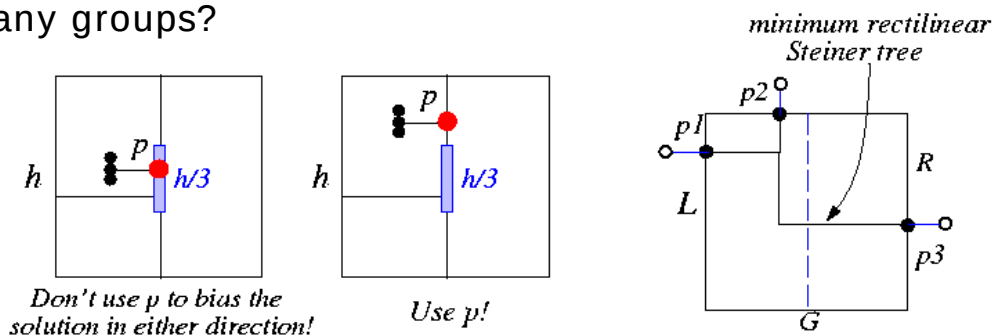
81

Terminal Propagation

- We should use the fact that s is in L_1 !



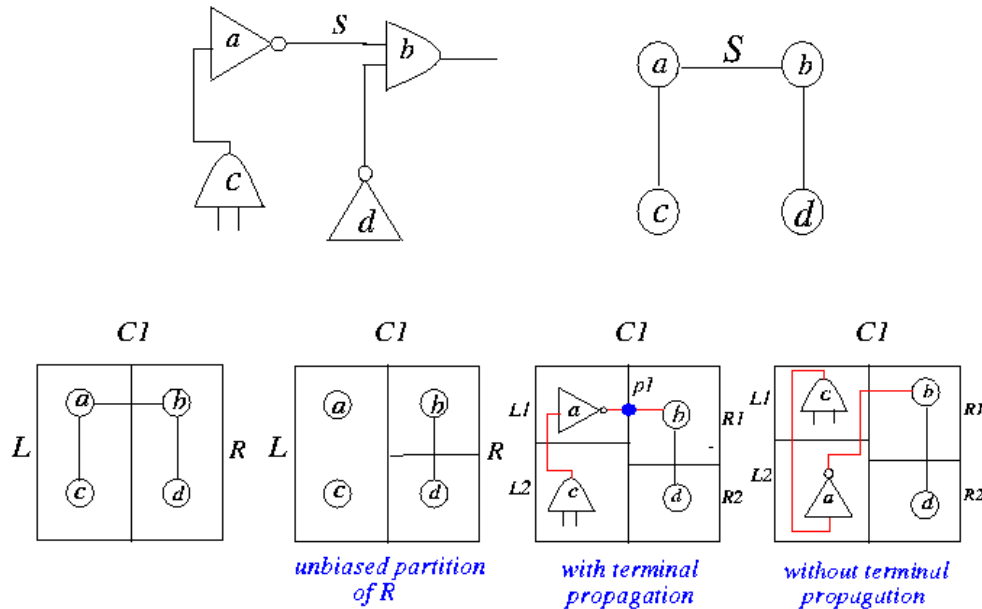
- When not to use p to bias partitioning? Net s has cells in many groups?



82

Terminal Propagation Example

- Partitioning must be done breadth-first, not depth-first.



83

General Procedure for Iterative Improvement

Algorithm: Iterative_Improvement()

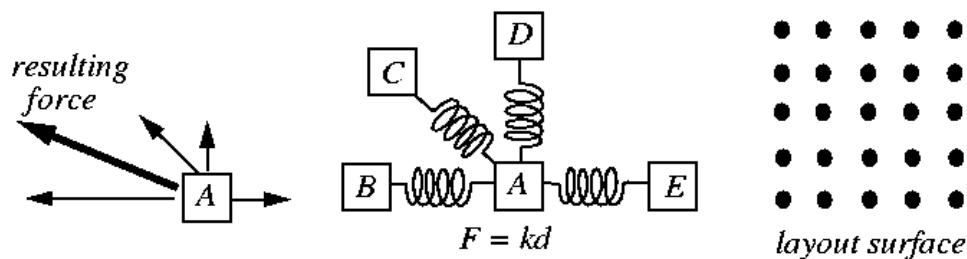
```

1  begin
2   $s \leftarrow \text{initial\_configuration}();$ 
3   $c \leftarrow \text{cost}(s);$ 
4  while (not stop()) do
5       $s' \leftarrow \text{perturb}(s);$ 
6       $c' \leftarrow \text{cost}(s');$ 
7      if (accept( $c$ ,  $c'$ ))
8          then  $s \leftarrow s';$ 
9  end
    
```

84

Placement by the Force-Directed Method

- Hanan & Kurtzberg, "Placement techniques," in *Design Automation of Digital Systems*, Breuer, Ed, 1972.
- Quinn, Jr. & Breuer, "A force directed component placement procedure for printed circuit boards," *IEEE Trans. Circuits and Systems*, June 1979.
- Reduce the placement problem to solving a set of simultaneous linear equations to determine equilibrium locations for cells.
- Analogy to Hooke's law: $F = kd$, F : force, k : spring constant, d : distance.
- Goal: Map cells to the layout surface.



85

Finding the Zero-Force Target Location

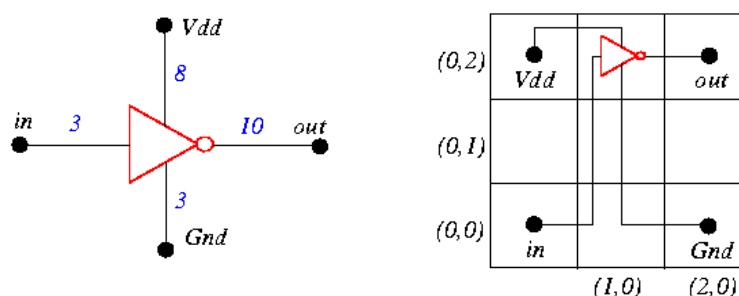
- Cell i connects to several cells j 's at distances d_{ij} 's by wires of weights w_{ij} 's. Total force: $F_i = \sum_j w_{ij} d_{ij}$
- The zero-force target location $(\hat{x}_i, \hat{y}_i)$ can be determined by equating the x - and y -components of the forces to zero:

□ In the example,

$$\sum_j w_{ij} \cdot (x_j - \hat{x}_i) = 0 \Rightarrow \hat{x}_i = \frac{\sum_j w_{ij} x_j}{\sum_j w_{ij}} \quad \text{and} \quad \hat{y}_i = 1.50.$$

$$\sum_j w_{ij} \cdot (y_j - \hat{y}_i) = 0 \Rightarrow \hat{y}_i = \frac{\sum_j w_{ij} y_j}{\sum_j w_{ij}}$$

$$\hat{x}_i = \frac{8 \times 0 + 10 \times 2 + 3 \times 0 + 3 \times 2}{8 + 10 + 3 + 3} = 1.083$$



86

Force-Directed Placement

- Can be constructive or iterative:
 - Start with an initial placement.
 - Select a “most profitable” cell p (e.g., maximum F , critical cells) and place it in its zero-force location.
 - “Fix” placement if the zero-location has been occupied by another cell q .
 - Popular options to fix:
 - **Ripple move:** place p in the occupied location, compute a new zero-force location for q , ...
 - **Chain move:** place p in the occupied location, move q to an adjacent location, ...
 - Move p to a free location close to q .

87

Force-Directed Placement

Algorithm: Force-Directed_Placement

```
1 begin
2 Compute the connectivity for each cell;
3 Sort the cells in decreasing order of their connectivities into list  $L$ ;
4 while ( $IterationCount < IterationLimit$ ) do
5   Seed  $\leftarrow$  next module from  $L$ ;
6   Declare the position of the seed vacant;
7   while ( $EndRipple = FALSE$ ) do
8     Compute target location of the seed;
9     case the target location
10    VACANT:
11      Move seed to the target location and lock;
12       $EndRipple \leftarrow TRUE$ ;  $AbortCount \leftarrow 0$ ;
13    SAME AS PRESENT LOCATION:
14       $EndRipple \leftarrow TRUE$ ;  $AbortCount \leftarrow 0$ ;
15    LOCKED:
16      Move selected cell to the nearest vacant location;
17       $EndRipple \leftarrow TRUE$ ;  $AbortCount \leftarrow AbortCount + 1$ ;
18      if ( $AbortCount > AbortLimit$ ) then
19        Unlock all cell locations;
20         $IterationCount \leftarrow IterationCount + 1$ ;
21    OCCUPIED AND NOT LOCKED:
22      Select cell as the target location for next move;
23      Move seed cell to target location and lock the target location;
24       $EndRipple \leftarrow FALSE$ ;  $AbortCount \leftarrow 0$ ;
26 end
```

88

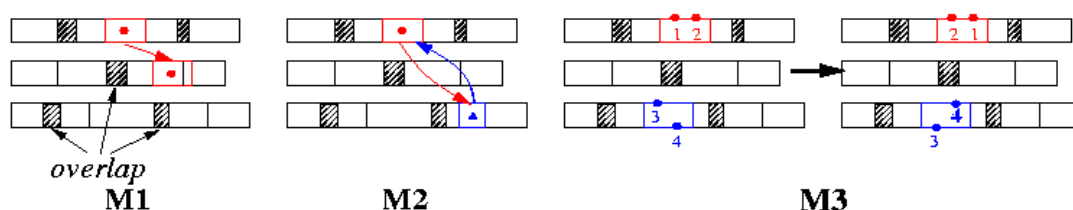
Placement by Simulated Annealing

- ❑ Sechen and Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE J. Solid-State Circuits*, Feb. 1985; "TimberWolf 3.2: A new standard cell placement and global routing package," DAC-86.
- ❑ TimberWolf: Stage 1
 - Modules are moved between different rows as well as within the same row.
 - Module overlaps are allowed.
 - When the temperature is reached below a certain value, stage 2 begins.
- ❑ TimberWolf: Stage 2
 - Remove overlaps.
 - Annealing process continues, but only interchanges adjacent modules within the same row.

89

Solution Space & Neighborhood Structure

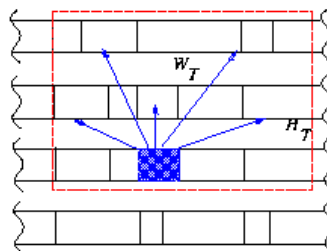
- ❑ **Solution Space:** All possible arrangements of the modules into rows, possibly with overlaps.
- ❑ **Neighborhood Structure:** 3 types of moves
 - M_1 : Displace a module to a new location.
 - M_2 : Interchange two modules.
 - M_3 : Change the orientation of a module.



90

Neighborhood Structure

- TimberWolf first tries to select a move between M_1 and M_2 :
 $Prob(M_1) = 0.8, Prob(M_2) = 0.2$.
- If a move of type M_1 is chosen and it is rejected, then a move of type M_3 for the same module will be chosen with probability 0.1.
- Restrictions: (1) what row for a module can be displaced? (2) what pairs of modules can be interchanged?
- **Key: Range Limiter**
 - At the beginning, (W_T, H_T) is big enough to contain the whole chip.
 - Window size shrinks as temperature decreases. Height & width $\propto \log(T)$.
 - Stage 2 begins when window size is so small that no inter-row module interchanges are possible.



91

Cost Function

- Cost function: $C = C_1 + C_2 + C_3$.
- C_1 : total estimated wirelength.
 - $C_1 = \sum_{i \in \text{Nets}} (\alpha_i w_i + \beta_i h_i)$
 - α_i, β_i are horizontal and vertical weights, respectively. ($\alpha_i=1, \beta_i=1 \Rightarrow$ half perimeter of the bounding box of Net i .)
 - Critical nets: Increase both α_i and β_i .
 - If vertical wirings are “cheaper” than horizontal wirings, use smaller vertical weights: $\beta_i < \alpha_i$.
- C_2 : penalty function for module overlaps.
 - $C_2 = \gamma \sum_{i \neq j} O_{ij}^2$, γ : penalty weight.
 - O_{ij} : amount of overlaps in the x-dimension between modules i and j .
- C_3 : penalty function that controls the row length.
 - $C_2 = \delta \sum_{r \in \text{Rows}} |L_r - D_r|$, δ : penalty weight.
 - D_r : desired row length.
 - L_r : sum of the widths of the modules in row r .

92

Annealing Schedule

- $T_k = r_k T_{k-1}$, $k = 1, 2, 3, \dots$
- r_k increases from 0.8 to max value 0.94 and then decreases to 0.8.
- At each temperature, a total # of nP attempts is made.
- n : # of modules; P : user specified constant.
- Termination: $T < 0.1$.

93

Outline

- Partitioning
- Floorplanning
- Placement
- Routing
 - Global routing
 - Detailed routing
- Compaction

94

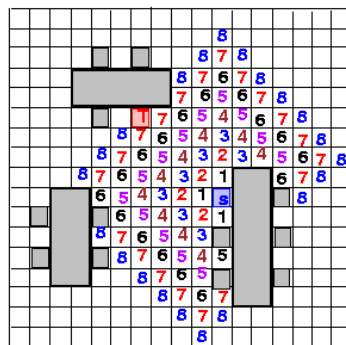
Routing

□ Course contents:

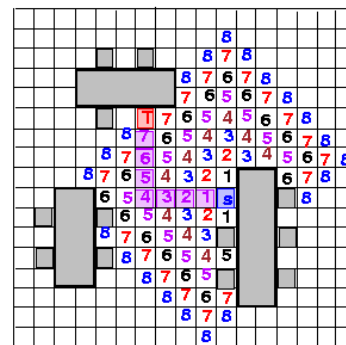
- Global routing
- Detail routing

□ Reading

- Chapter 12



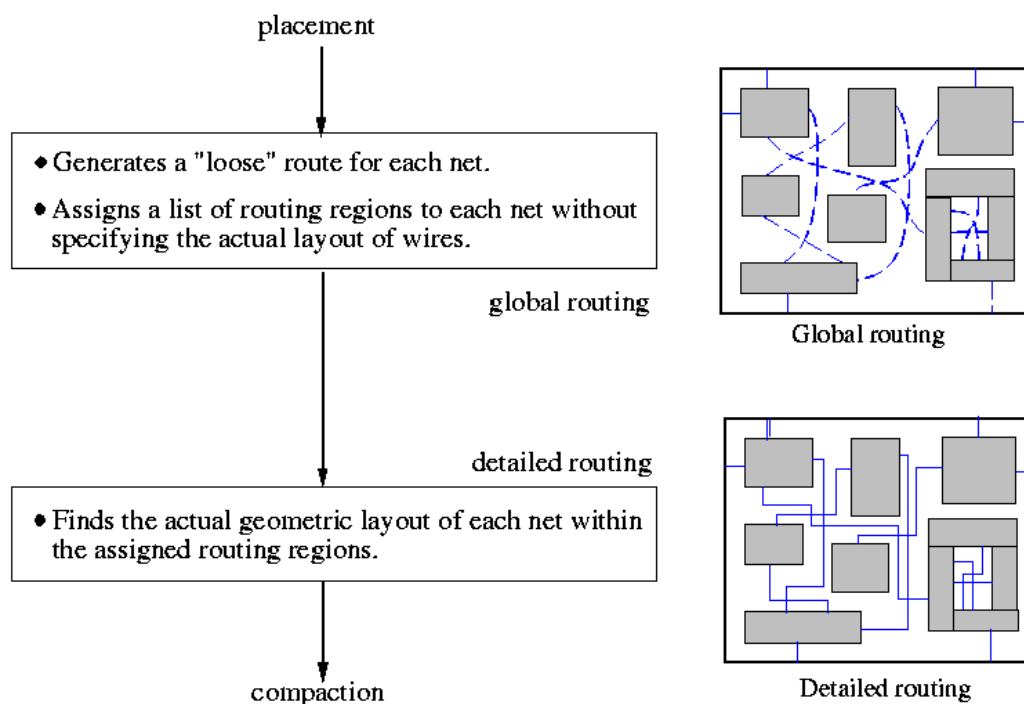
Filling



Retrace

95

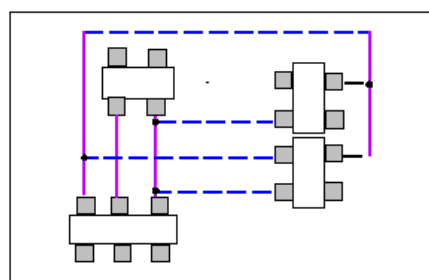
Routing



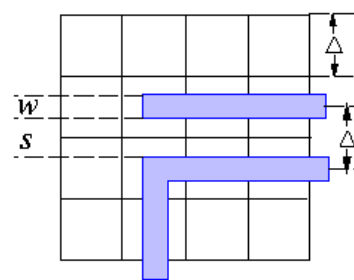
96

Routing Constraints

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



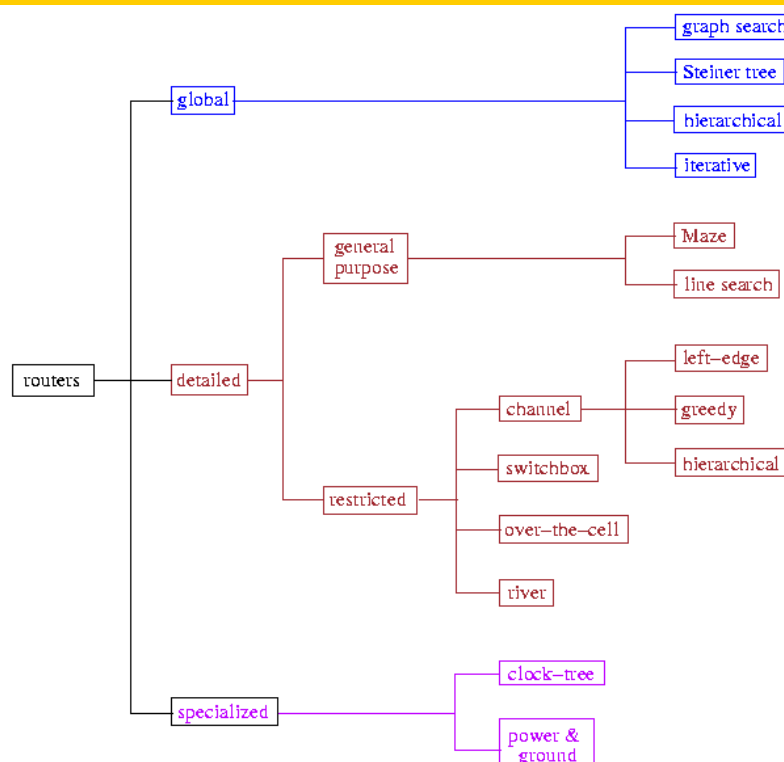
Two-layer routing



Geometrical constraint

97

Classification of Routing



98

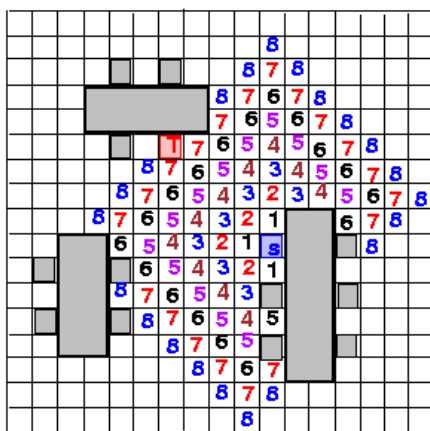
Maze Router: Lee Algorithm

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

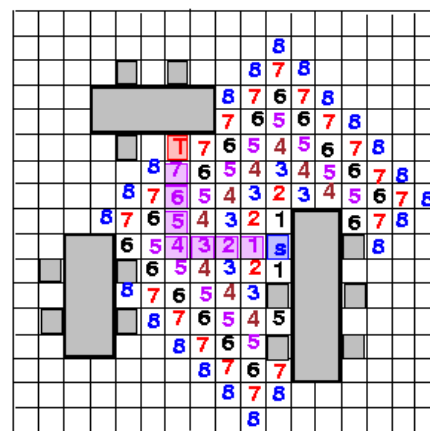
99

Lee Algorithm

- ❑ Find a path from *S* to *T* by “wave propagation”.



Filling



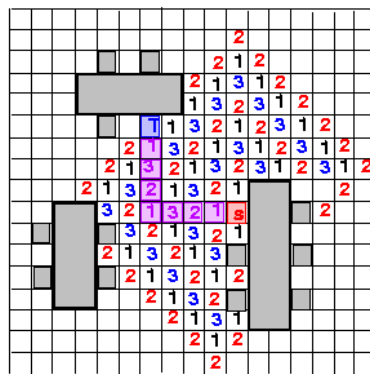
Retrace

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

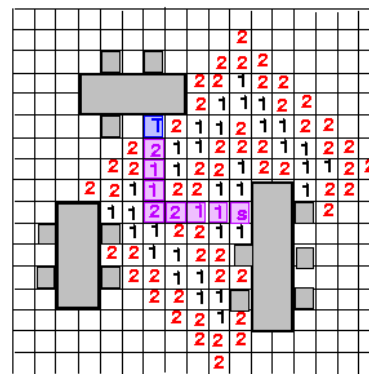
100

Reducing Memory Requirement

- Akers's Observations (1967)
 - Adjacent labels for k are either $k-1$ or $k+1$.
 - Want a labeling scheme such that each label has its preceding label different from its succeeding label.
- Way 1: coding sequence 1, 2, 3, 1, 2, 3, ...; states: 1, 2, 3, *empty*, *blocked* (3 bits required)
- Way 2: coding sequence 1, 1, 2, 2, 1, 1, 2, 2, ...; states: 1, 2, *empty*, *blocked* (need only 2 bits)



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



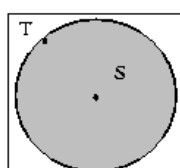
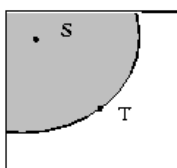
Sequence: 1, 1, 2, 2, 1, 1, 2, 2, ...

101

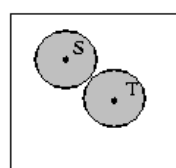
Reducing Running Time

- Starting point selection: Choose the point farthest from the center of the grid as the starting point.
- Double fan-out: Propagate waves from both the source and the target cells.
- Framing: Search inside a rectangle area 10--20% larger than the bounding box containing the source and target.
 - Need to enlarge the rectangle and redo if the search fails.

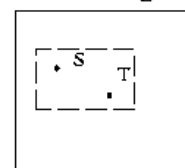
starting point selection



double fan-out



framing



102

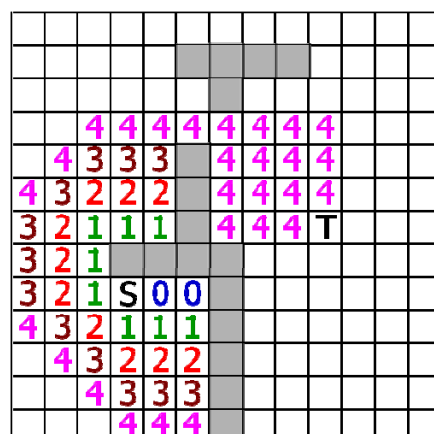
Hadlock's Algorithm

- Hadlock, "A shortest path algorithm for grid graphs," *Networks*, 1977.
- Uses detour number (instead of labeling wavefront in Lee's router)
 - Detour number, $d(P)$: # of grid cells directed **away from** its target on path P .
 - $MD(S, T)$: the Manhattan distance between S and T .
 - Path length of P , $I(P)$: $I(P) = MD(S, T) + 2 d(P)$.
 - $MD(S, T)$ fixed! $\Rightarrow$ Minimize $d(P)$ to find the shortest path.
 - For any cell labeled i , label its adjacent unblocked cells **away from** T $i+1$; label i otherwise.
- Time and space complexities: $O(MN)$, but substantially reduces the # of searched cells.
- Finds the shortest path between S and T .

103

Hadlock's Algorithm (cont'd)

- $d(P)$: # of grid cells directed **away from** its target on path P .
- $MD(S, T)$: the Manhattan distance between S and T .
- Path length of P , $I(P)$: $I(P) = MD(S, T) + 2d(P)$.
- $MD(S, T)$ fixed! $\Rightarrow$ Minimize $d(P)$ to find the shortest path.
- For any cell labeled i , label its adjacent unblocked cells **away from** T $i+1$; label i otherwise.



■ obstacle

104

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



- Each cell is represented by a vertex.
- Two vertices are joined by an edge if the corresponding cells are adjacent to each other.



Global-Routing Problem

- Given a netlist $N = \{N_1, N_2, \dots, N_n\}$, a routing graph $G = (V, E)$, find a Steiner tree T_i for each net N_i , $1 \leq i \leq n$, such that $U(e_j) \leq c(e_j)$, $\forall e_j \in E$ and $\sum_i L(T_i)$ is minimized, where
 - $c(e_j)$: capacity of edge e_j
 - $x_{ij} = 1$ if e_j is in T_i ; $x_{ij} = 0$ otherwise
 - $U(e_j) = \sum_i x_{ij}$: # of wires that pass through the channel corresponding to edge e_j
 - $L(T_i)$: total wirelength of Steiner tree T_i
- For high performance, the maximum wirelength $\max_i L(T_i)$ is minimized (or the longest path between two points in T_i is minimized).

109

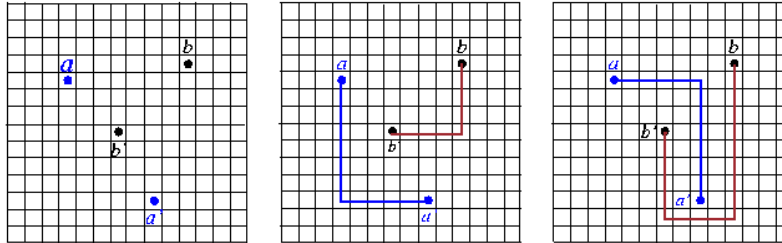
Classification of Global-Routing Algorithms

- Sequential approach:
 - Select a net order and route nets sequentially in the order
 - Earlier routed nets might block the routing of subsequent nets
 - Routing quality heavily depends on net ordering
 - Strategy: Heuristic net ordering + rip-up and rerouting
- Concurrent approach:
 - All nets are considered simultaneously
 - E.g., 0-1 integer linear programming (0-1 ILP)

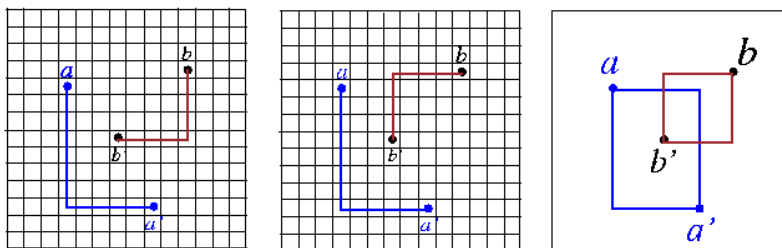
110

Net Ordering

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



*route net *a* before net *b**



*route net *b* before net *a**

111

Net Ordering (cont'd)

- Order the nets in the ascending order of the # of pins within their bounding boxes.
- Order the nets in the ascending (descending) order of their lengths if routability (timing) is the most critical metric.
- Order the nets based on their timing criticality.

112

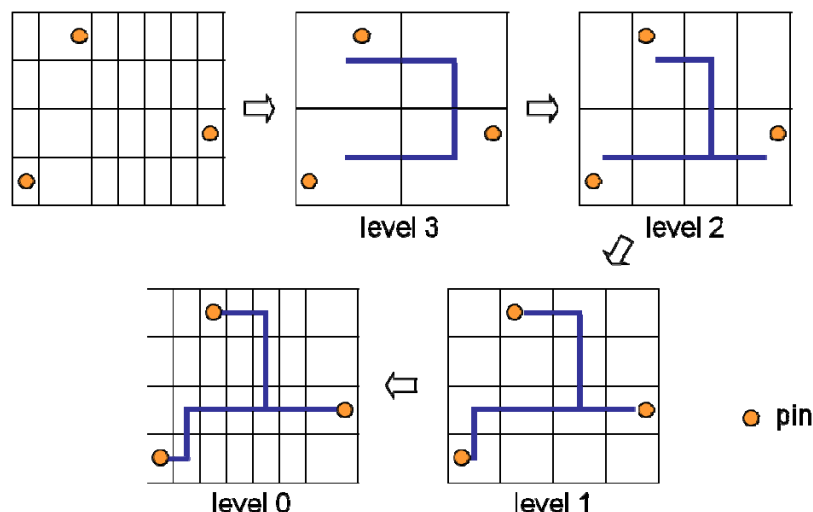
Rip-Up and Re-routing

- ❑ Rip-up and re-routing is required if a global or detailed router fails in routing all nets.
- ❑ Approaches: the manual approach? the automatic procedure?
- ❑ Two steps in rip-up and re-routing
 1. Identify bottleneck regions, rip off some already routed nets.
 2. Route the blocked connections, and re-route the ripped-up connections.
- ❑ Repeat the above steps until all connections are routed or a time limit is exceeded.

113

Top-down Hierarchical Global Routing

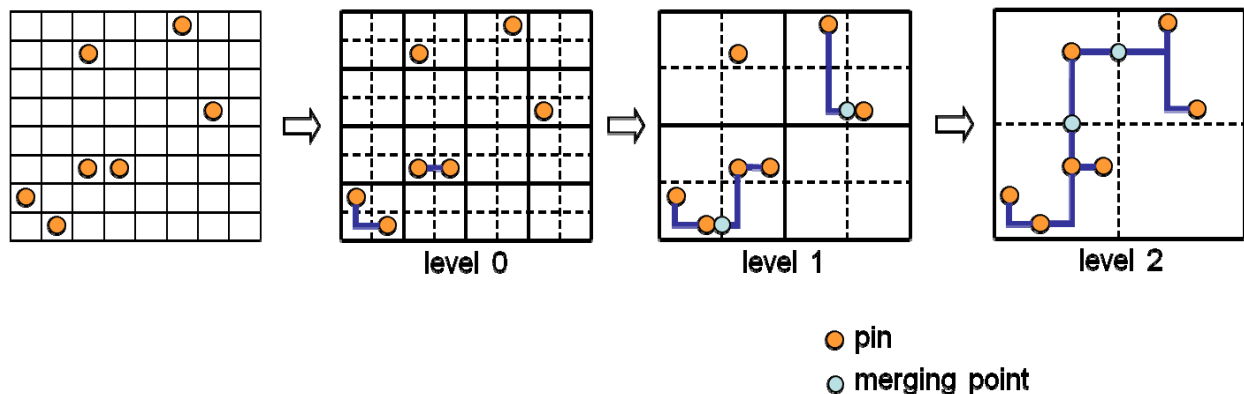
- ❑ Recursively divides routing regions into successively smaller **super cells**, and nets at each hierarchical level are routed sequentially or concurrently.



114

Bottom-up Hierarchical Global Routing

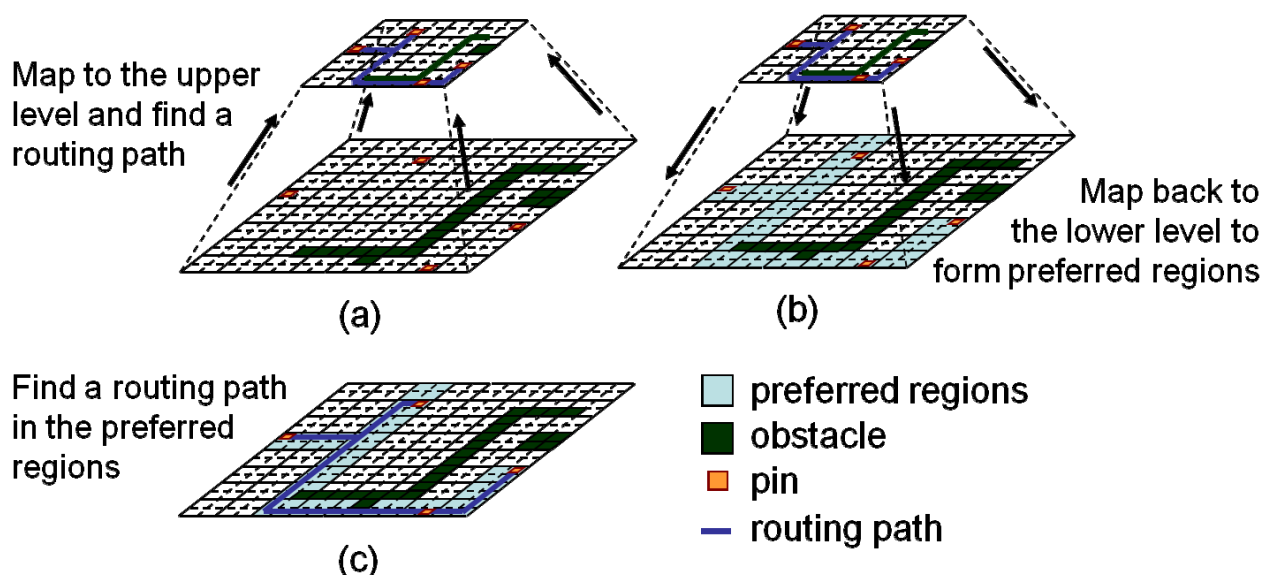
- At each hierarchical level, routing is restrained within each super cell individually.
- When the routing at the current level is finished, every four super cells are merged to form a new larger super cell at the next higher level.



115

Hybrid Hierarchical Global Routing

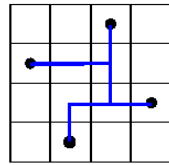
- (1) neighboring propagation, (2) preference partitioning, and (3) bounded routing



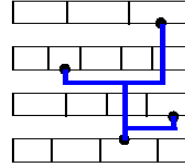
116

The Routing-Tree Problem

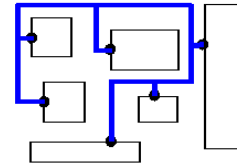
- **Problem:** Given a set of pins of a net, interconnect the pins by a "routing tree."



gate array

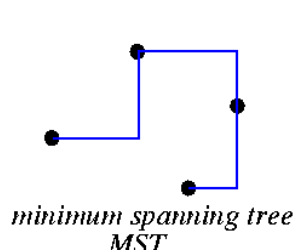


standard cell

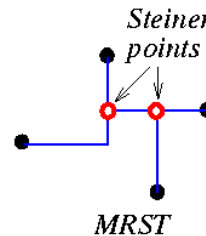


building block

- **Minimum Rectilinear Steiner Tree (MRST) Problem:** Given n points in the plane, find a minimum-length tree of rectilinear edges which connects the points.
- $MRST(P) = MST(P \cup S)$, where P and S are the sets of original points and Steiner points, respectively.



minimum spanning tree
MST

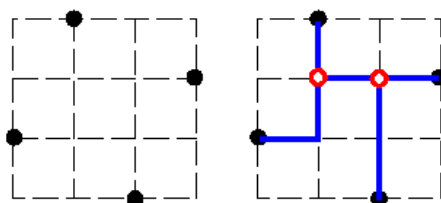


MRST

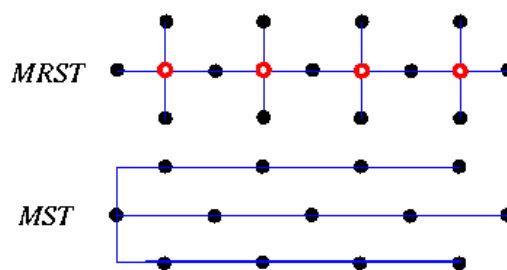
117

Theoretical Results for the MRST Problem

- **Hanan's Thm:** There exists an MRST with all Steiner points (set S) chosen from the intersection points of horizontal and vertical lines drawn points of P .
 - Hanan, "On Steiner's problem with rectilinear distance," *SIAM J. Applied Math.*, 1966.
- **Hwang's Theorem:** For any point set P , $\frac{Cost(MST(P))}{Cost(MRST(P))} \leq \frac{3}{2}$.
 - Hwang, "On Steiner minimal tree with rectilinear distance," *SIAM J. Applied Math.*, 1976.
- Best existing approximation algorithm: Performance bound $61/48$ by Foessmeier *et al.*



Hanan grid

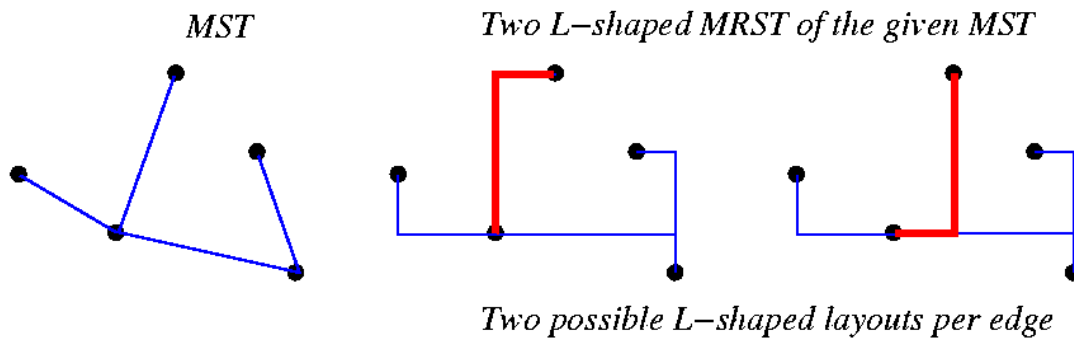


$$Cost(MST)/Cost(MRST) \rightarrow 3/2$$

118

Coping with the MRST Problem

- Ho, Vijayan, Wong, “New algorithms for the rectilinear Steiner problem,”
 1. Construct an MRST from an MST.
 2. Each edge is straight or L-shaped.
 3. Maximize overlaps by dynamic programming.
- About 8% smaller than $\text{Cost}(\text{MST})$.



119

Iterated 1-Steiner Heuristic for MRST

- Kahng & Robins, “A new class of Steiner tree heuristics with good performance: the iterated 1-Steiner approach,” ICCAD-90.

Algorithm: Iterated_1-Steiner(P)

P : set of n points.

1 **begin**

2 $S \leftarrow \emptyset$;

/* $H(P \cup S)$: set of Hanan points */

/* $\Delta \text{MST}(A, B) = \text{Cost}(\text{MST}(A)) - \text{Cost}(\text{MST}(A \cup B))$ */

3 **while** ($\text{Cand} \leftarrow \{x \in H(P \cup S) \mid \Delta \text{MST}(P \cup S, \{x\}) > 0\} \neq \emptyset$) **do**

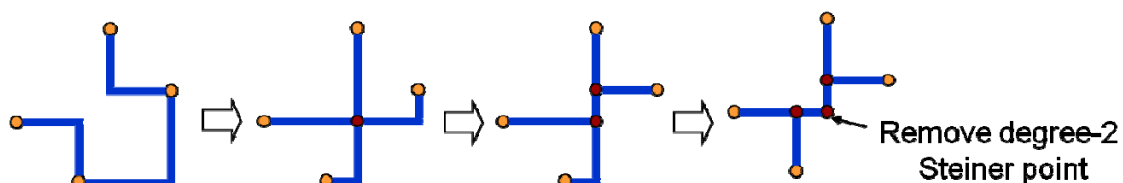
4 Find $x \in \text{Cand}$ and which maximizes $\Delta \text{MST}(P \cup S, \{x\})$;

5 $S \leftarrow S \cup \{x\}$;

6 Remove points in S which have degree ≤ 2 in $\text{MST}(P \cup S)$;

7 **return** $\text{MST}(P \cup S)$;

8 **end**



120

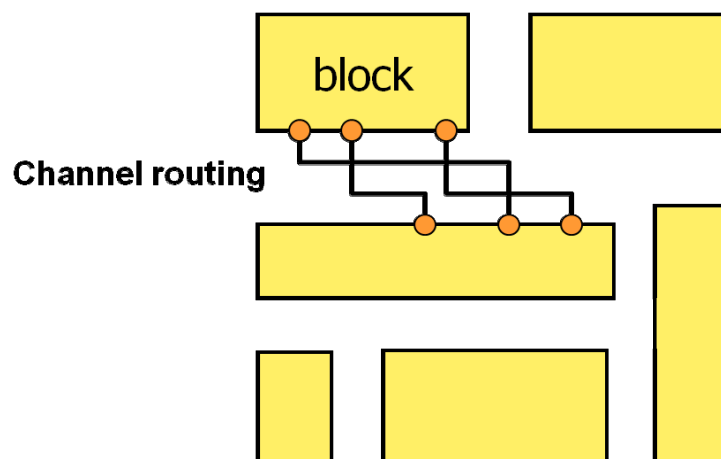
Outline

- Partitioning
- Floorplanning
- Placement
- Routing
 - Global routing
 - Detailed routing
- Compaction

121

Channel Routing

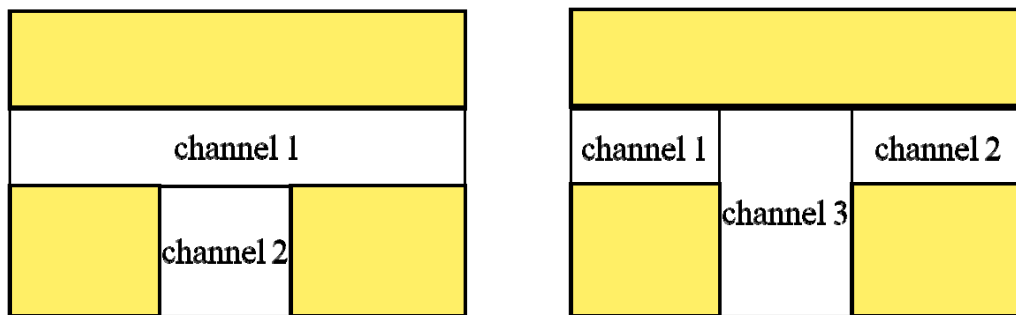
- In earlier process technologies, channel routing was pervasively used since most wires were routed in the free space (*i.e.*, routing channel) between a pair of logic blocks (cell rows)



122

Routing Region Decomposition

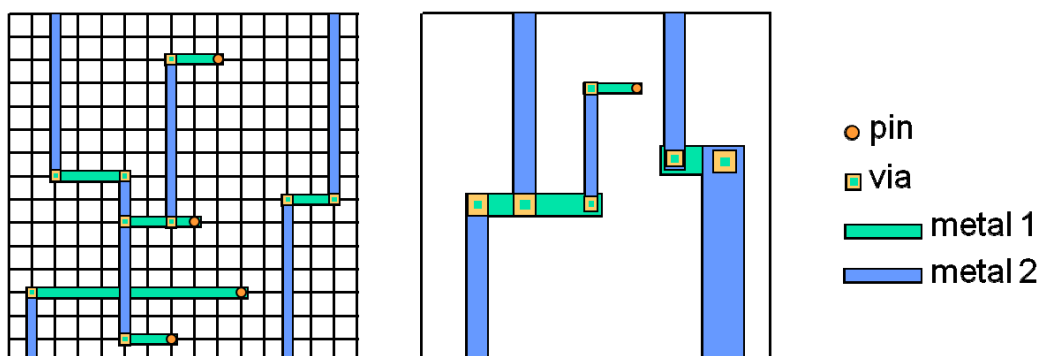
- There are often various ways to decompose a routing region.
- The order of routing regions significantly affects the channel-routing process.



123

Routing Models

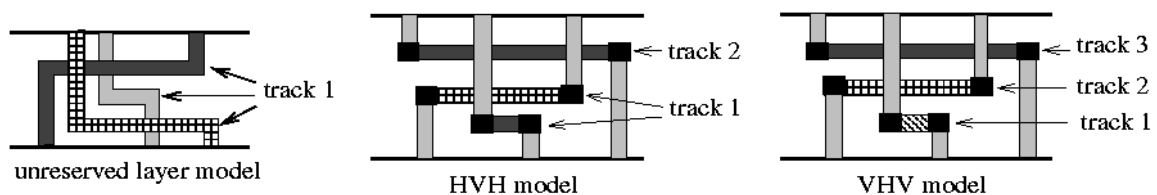
- **Grid-based model:**
 - A grid is super-imposed on the routing region.
 - Wires follow paths along the grid lines.
 - **Pitch:** distance between two gridded lines
- **Gridless model:**
 - Any model that does not follow this “gridded” approach.



124

Models for Multi-Layer Routing

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

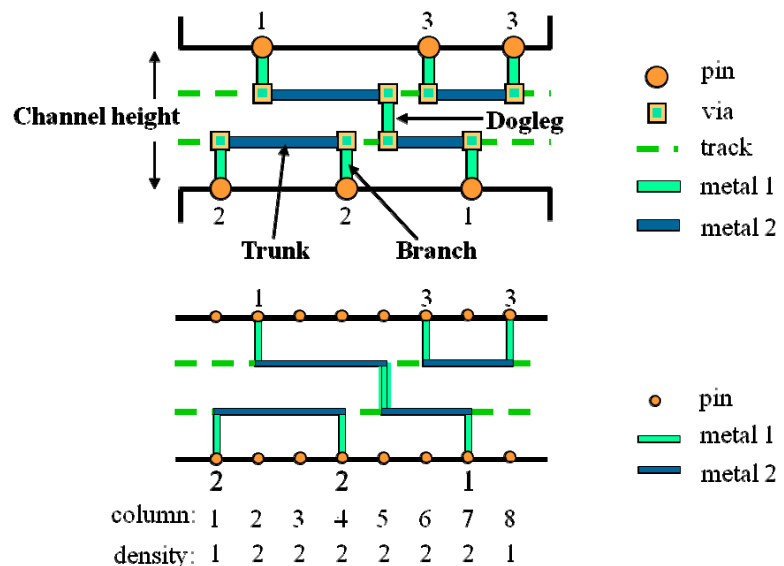


3 types of 3-layer models

125

Terminology for Channel Routing

- **Local density at column i , $d(i)$:** total # of nets that crosses column i .
- **Channel density:** maximum local density
 - # of horizontal tracks required $\geq$ channel density.



126

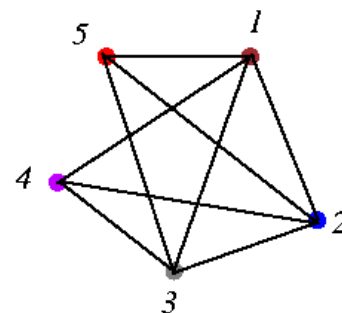
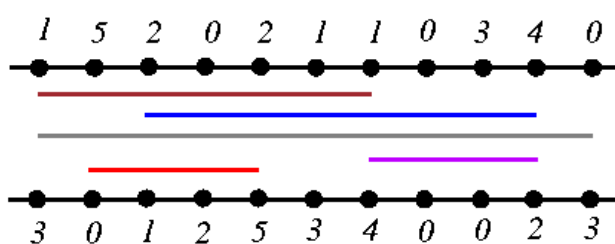
Channel Routing Problem

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

127

Horizontal Constraint Graph (HCG)

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

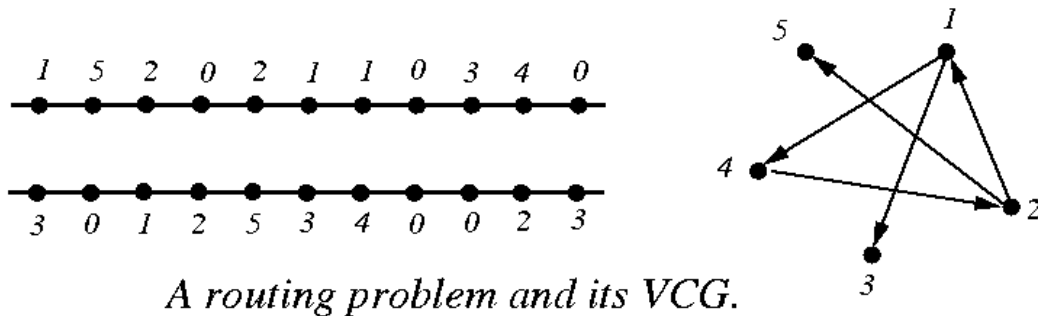


A routing problem and its HCG.

128

Vertical Constraint Graph (VCG)

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



129

2-Layer Channel Routing: Basic Left-Edge Algorithm

- Hashimoto & Stevens, "Wire routing by optimizing channel assignment within large apertures," DAC-71.
- **No vertical constraint.**
- HV-layer model is used.
- **Doglegs are not allowed.**
- Treat each net as an interval.
- Intervals are sorted according to their left-end x-coordinates.
- Intervals (nets) are routed one-by-one according to the order.
- For a net, tracks are scanned from top to bottom, and the first track that can accommodate the net is assigned to the net.
- **Optimality:** produces a routing solution with the minimum # of tracks (if no vertical constraint).

130

Basic Left-Edge Algorithm

Algorithm: Basic_Left-Edge(U , $track[j]$)

U : set of unassigned intervals (nets) $I_1, \dots, I_n$;

$I_j = [s_j, e_j]$: interval j with left-end x-coordinate s_j and right-end e_j ;

$track[j]$: track to which net j is assigned.

```

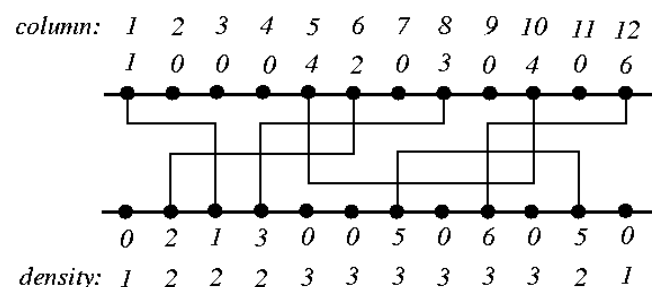
1 begin
2  $U \leftarrow \{I_1, I_2, \dots, I_n\}$ ;
3  $t \leftarrow 0$ ;
4 while ( $U \neq \emptyset$ ) do
5    $t \leftarrow t + 1$ ;
6    $watermark \leftarrow 0$ ;
7   while (there is an  $I_j \in U$  s.t.  $s_j > watermark$ ) do
8     Pick the interval  $I_j \in U$  with  $s_j > watermark$ ,
       nearest  $watermark$ ;
9      $track[j] \leftarrow t$ ;
10     $watermark \leftarrow e_j$ ;
11     $U \leftarrow U - \{I_j\}$ ;
12 end

```

131

Basic Left-Edge Example

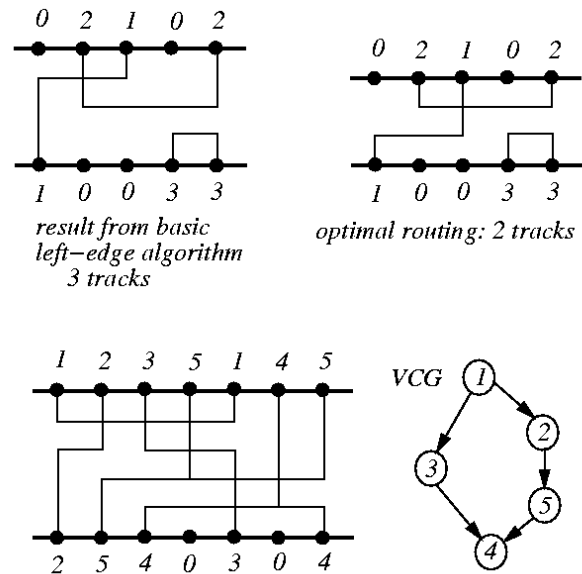
- $U = \{I_1, I_2, \dots, I_6\}$; $I_1 = [1, 3]$, $I_2 = [2, 6]$, $I_3 = [4, 8]$, $I_4 = [5, 10]$, $I_5 = [7, 11]$, $I_6 = [9, 12]$.
- $t = 1$:
 - Route I_1 : $watermark = 3$;
 - Route I_3 : $watermark = 8$;
 - Route I_6 : $watermark = 12$;
- $t = 2$:
 - Route I_2 : $watermark = 6$;
 - Route I_5 : $watermark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



132

Basic Left-Edge Algorithm

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



133

Constrained Left-Edge Algorithm

Algorithm: Constrained_Left-Edge(U , $track[j]$)

U : set of unassigned intervals (nets) $I_1, \dots, I_n$;

$I_j = [s_j, e_j]$: interval j with left-end x -coordinate s_j and right-end e_j ;

$track[j]$: track to which net j is assigned.

```

1 begin
2  $U \leftarrow \{ I_1, I_2, \dots, I_n \}$ ;
3  $t \leftarrow 0$ ;
4 while ( $U \neq \emptyset$ ) do
5    $t \leftarrow t + 1$ ;
6    $watermark \leftarrow 0$ ;
7   while (there is an unconstrained  $I_j \in U$  s.t.  $s_j > watermark$ ) do
8     Pick the interval  $I_j \in U$  that is unconstrained,
       with  $s_j > watermark$ , nearest  $watermark$ ;
9      $track[j] \leftarrow t$ ;
10     $watermark \leftarrow e_j$ ;
11     $U \leftarrow U - \{ I_j \}$ ;
12 end
```

134

Constrained Left-Edge Example

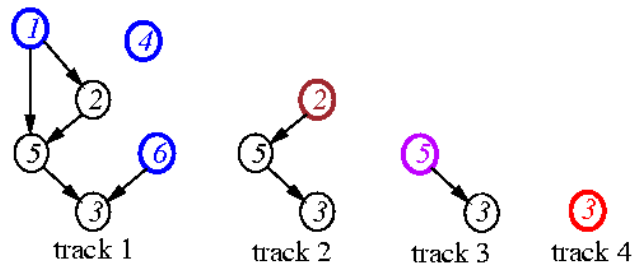
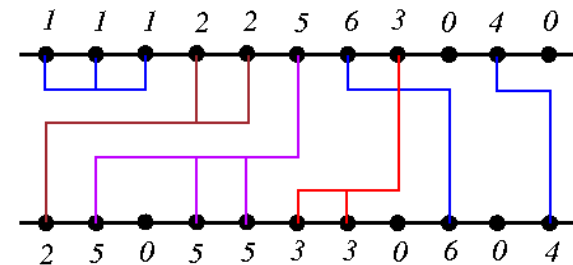
□ $I_1 = [1, 3]$, $I_2 = [1, 5]$, $I_3 = [6, 8]$, $I_4 = [10, 11]$, $I_5 = [2, 6]$, $I_6 = [7, 9]$.

□ Track 1: Route I_1 (cannot route I_3); Route I_6 ; Route I_4 .

□ Track 2: Route I_2 ;

□ Track 3: Route I_5 .

□ Track 4: Route I_3 .

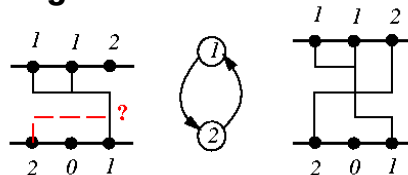


135

Dogleg Channel Router

□ Deutch, "A dogleg channel router," 13rd DAC, 1976.

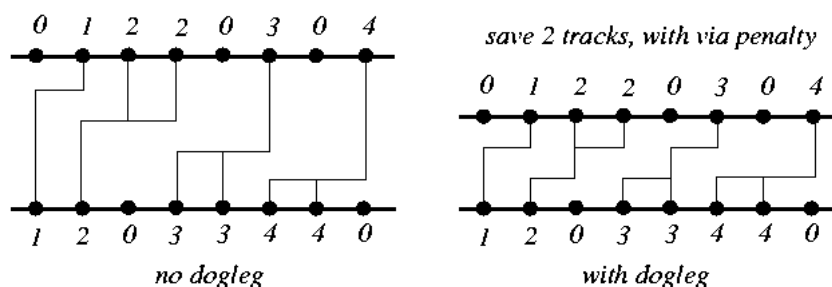
□ **Drawback of Left-Edge: cannot handle the cases with constraint cycles.**



□ **Drawback of Left-Edge: the entire net is on a single track.**

■ **Doglegs** are used to place parts of a net on different tracks to minimize channel height.

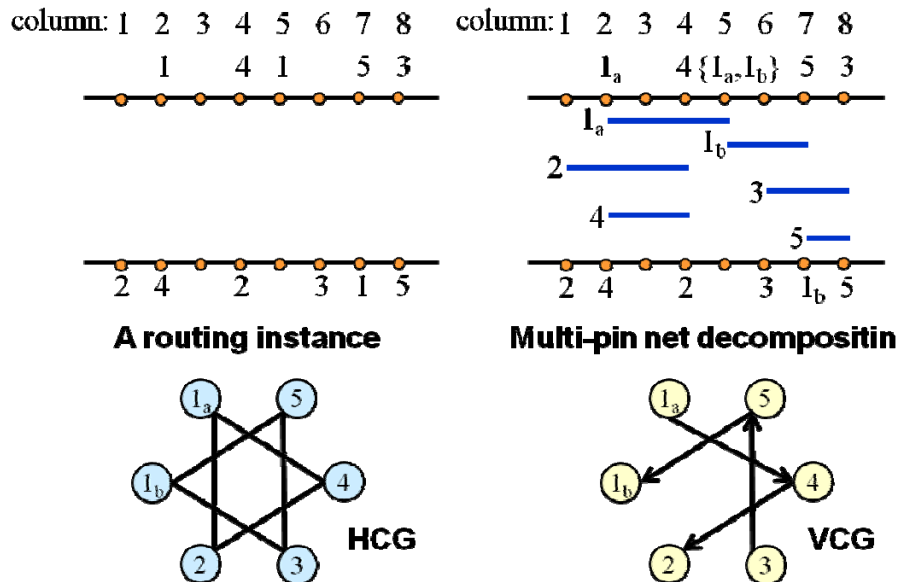
■ Might incur penalty for additional vias.



136

Dogleg Channel Router

- Each multi-pin net is broken into a set of 2-pin nets.
- Modified Left-Edge Algorithm is applied to each subnet.

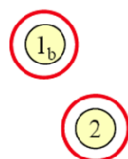


137

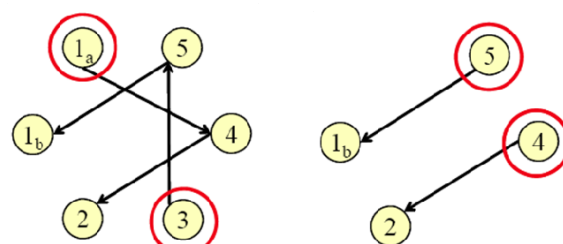
Dogleg Channel Routing Example

Net	Range
2	[1,4]
1_a	[2,5]
4	[2,4]
1_b	[5,7]
3	[6,8]
5	[7,8]

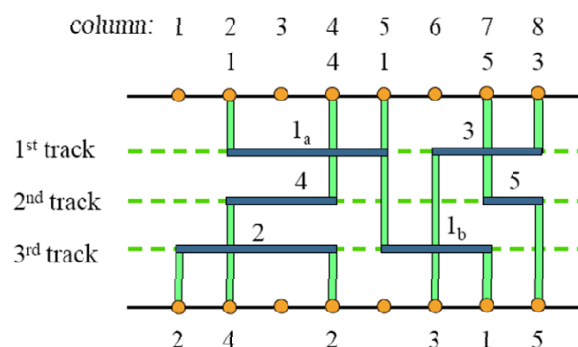
(a) Nets ordered by left-end coordinates



(d) 1_b and 2 are assigned to the 3rd track



(b) 1_a and 3 are assigned to the 1st track (c) 4 and 5 are assigned to the 2nd track



(e) The final routing result with doglegs

138

Modern Routing Considerations

- ❑ Signal/power Integrity
 - Capacitive crosstalk
 - Inductive crosstalk
 - IR drop
- ❑ Manufacturability
 - Process variation
 - Optical proximity correction (OPC)
 - Chemical mechanical polishing (CMP)
 - Phase-Shift Mask (PSM)
- ❑ Reliability
 - Double via insertion
 - Process antenna effect
 - Electromigration (EM)
 - Electrostatic discharge (ESD)

139

Outline

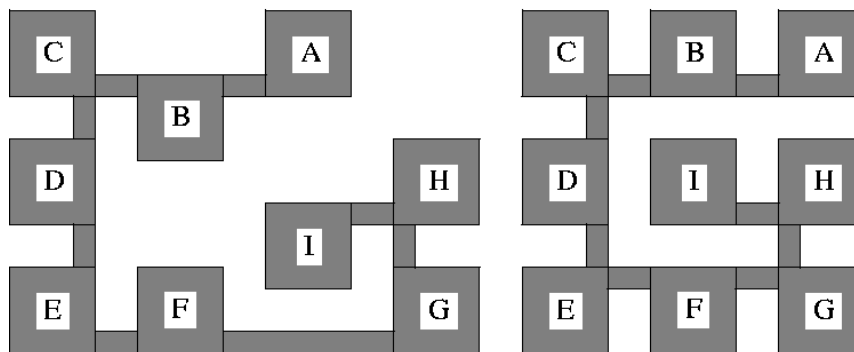
- ❑ Partitioning
- ❑ Floorplanning
- ❑ Placement
- ❑ Routing
- ❑ Compaction

140

Layout Compaction

□ Course contents

- Design rules
- Symbolic layout
- Constraint-graph compaction



141

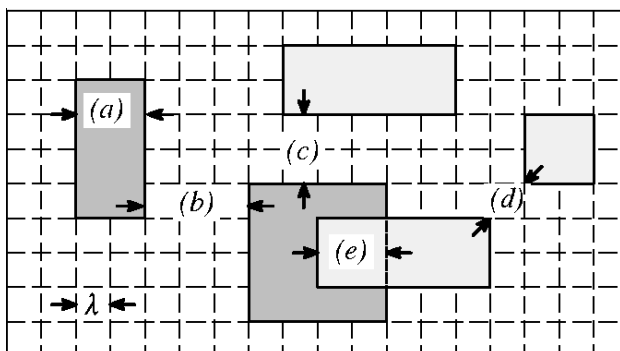
Design Rules

- **Design rules:** restrictions on the mask patterns to increase the probability of successful fabrication.

- Patterns and design rules are often expressed in λ rules.

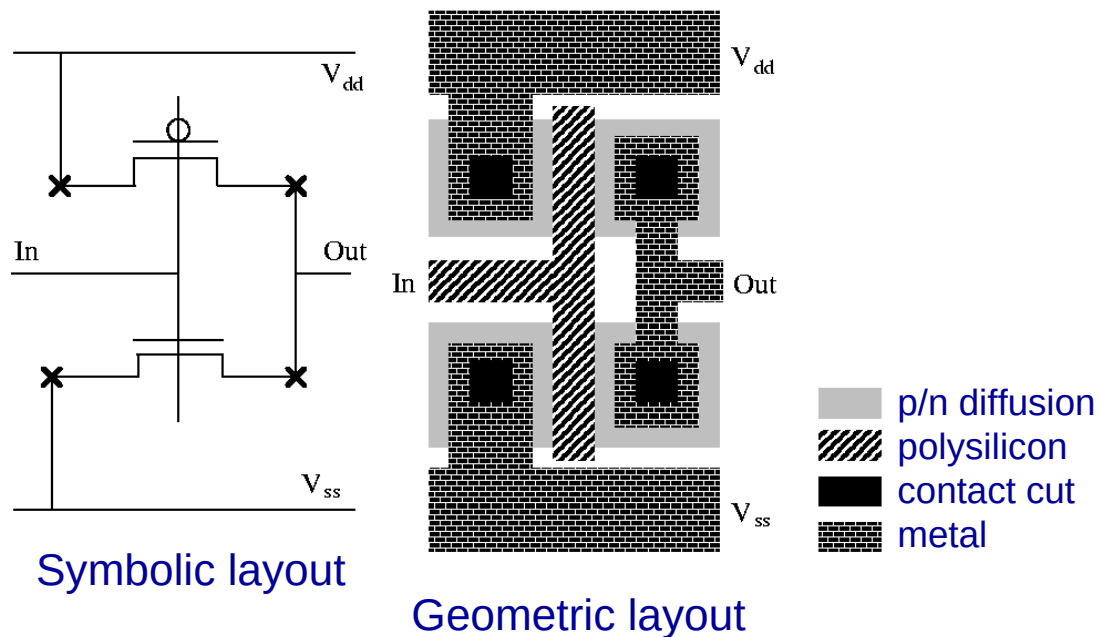
- Most common design rules:

- minimum-width rules (valid for a mask pattern of a specific layer): (a).
- minimum-separation rules (between mask patterns of the same layer or different layers): (b), (c), (d).
- minimum-overlap rules (mask patterns in different layers): (e).



142

CMOS Inverter Layout Example



143

Symbolic Layout

- ❑ **Geometric (mask) layout:** coordinates of the layout patterns (rectangles) are absolute (or in multiples of λ).
- ❑ **Symbolic (topological) layout:** only relations between layout elements (below, left to, etc) are known.
 - Symbols are used to represent elements located in several layers, e.g. transistors, contact cuts.
 - The *length*, *width* or *layer* of a wire or other layout element might be left unspecified.
 - Mask layers not directly related to the functionality of the circuit do not need to be specified, e.g. n-well, p-well.
- ❑ The symbolic layout can work with a technology file that contains all design rule information for the target technology to produce the geometric layout.

144

Compaction and Its Applications

- ❑ A **compaction program** or **compactor** generates layout at the mask level. It attempts to make the layout as dense as possible.
- ❑ Applications of compaction:
 - **Area minimization**: remove redundant space in layout at the mask level.
 - **Layout compilation**: generate mask-level layout from symbolic layout.
 - **Redesign**: automatically remove design-rule violations.
 - **Rescaling**: convert mask-level layout from one technology to another.

145

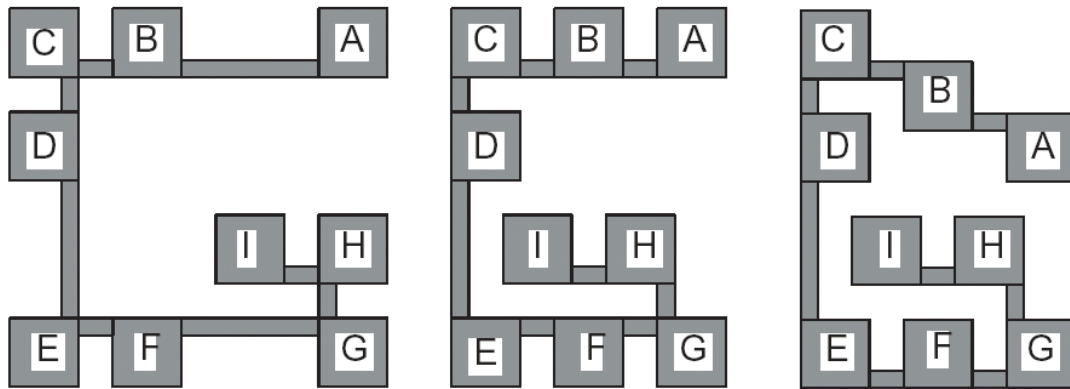
Aspects of Compaction

- ❑ Dimension:
 - 1-dimensional (1D) compaction: layout elements only are moved or shrunk in one dimension (x or y direction).
 - ❑ Is often performed first in the x-dimension and then in the y-dimension (or vice versa).
 - 2-dimensional (2D) compaction: layout elements are moved and shrunk simultaneously in two dimensions.
- ❑ Complexity:
 - 1D compaction can be done in polynomial time.
 - 2D compaction is NP-hard.

146

1D Compaction: X Followed By Y

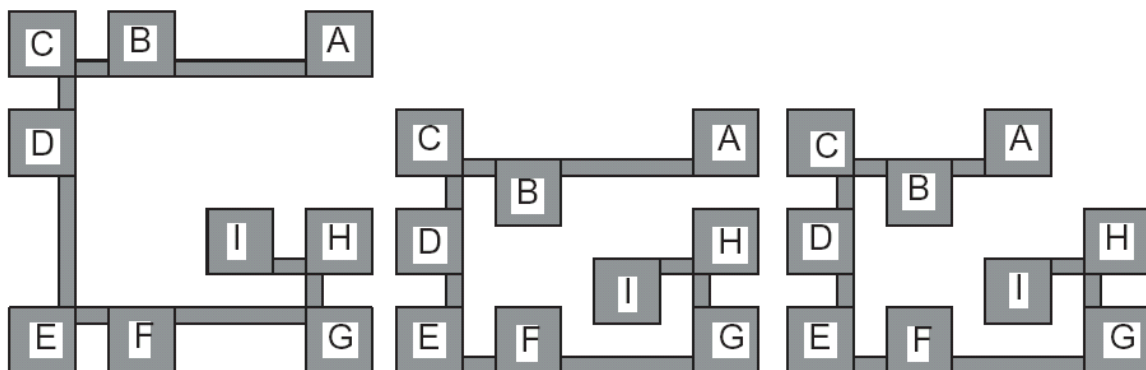
- Each square is $2\lambda * 2\lambda$, minimum separation is 1λ .
- Initially, the layout is $11\lambda * 11\lambda$.
- After compacting along the **x** direction, then the **y** direction, we have the layout size of $8\lambda * 11\lambda$.



147

1D Compaction: Y Followed By X

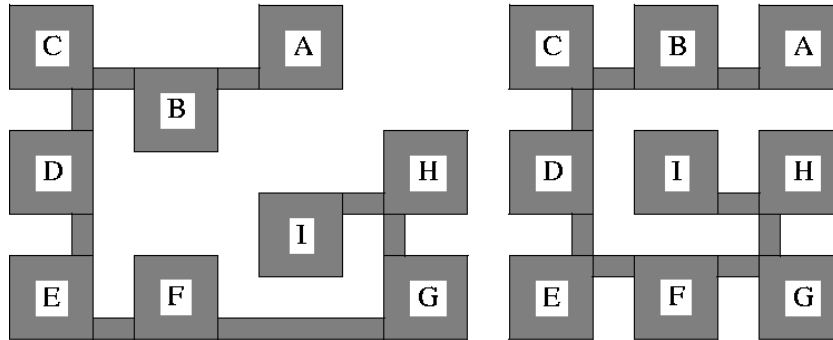
- Each square is $2\lambda * 2\lambda$, minimum separation is 1λ .
- Initially, the layout is $11\lambda * 11\lambda$.
- After compacting along the **y** direction, then the **x** direction, we have the layout size of $11\lambda * 8\lambda$.



148

2D Compaction

- Each square is $2\lambda * 2\lambda$, minimum separation is 1λ .
- Initially, the layout is $11\lambda * 11\lambda$.
- After **2D compaction**, the layout size is only $8\lambda * 8\lambda$.



- Since 2D compaction is NP-complete, most compactors are based on repeated 1D compaction.

149

Inequalities for Distance Constraints

- Minimum-distance design rules can be expressed as inequalities.

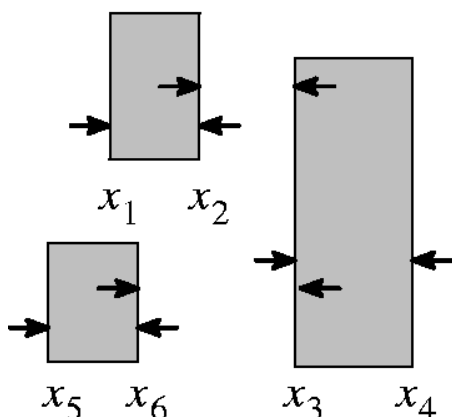
$$x_j - x_i \geq d_{ij}.$$

- For example, if the minimum width is a and the minimum separation is b , then

$$x_2 - x_1 \geq a$$

$$x_3 - x_2 \geq b$$

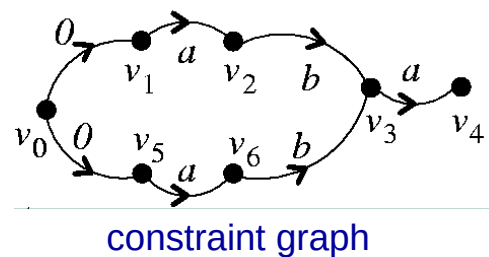
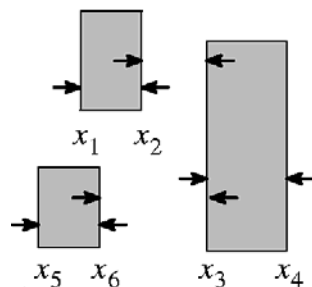
$$x_3 - x_6 \geq b$$



150

The Constraint Graph

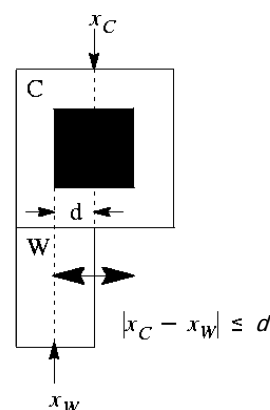
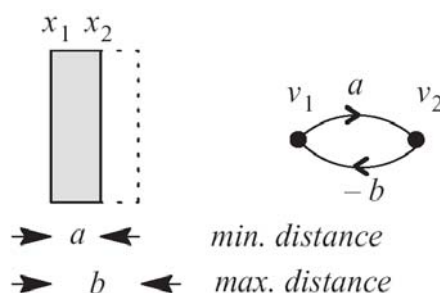
- The inequalities can be used to construct a constraint graph $G(V, E)$:
 - There is a vertex v_i for each variable x_i .
 - For each inequality $x_j - x_i \geq d_{ij}$ there is an edge (v_i, v_j) with weight d_{ij} .
 - There is an extra source vertex, v_0 ; it is located at $x = 0$; all other vertices are at its right.
- If all the inequalities express minimum-distance constraints, the graph is **acyclic** (DAG).
- The longest path in a constraint graph determines the layout dimension.



151

Maximum-Distance Constraints

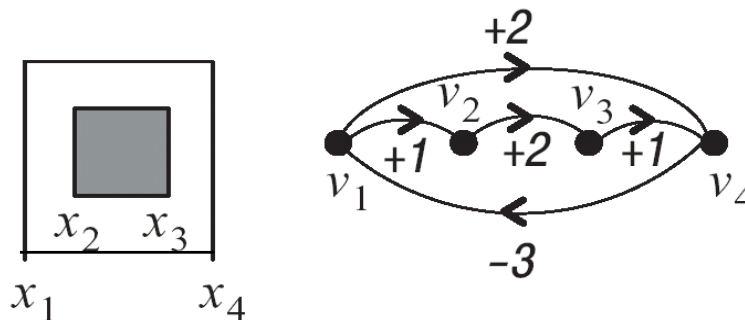
- Sometimes the distance of layout elements is bounded by a maximum, e.g., when the user wants a maximum wire width, maintains a wire connecting to a via, etc.
 - A maximum distance constraint gives an inequality of the form: $x_j - x_i \leq c_{ij}$ or $x_i - x_j \geq -c_{ij}$
 - Consequence for the constraint graph: **backward edge**
 - (v_j, v_i) with weight $d_{ji} = -c_{ij}$; the graph is not acyclic anymore.
- The longest path in a constraint graph determines the layout dimension.



152

Longest-Paths in Cyclic Graphs

- Constraint-graph compaction with maximum-distance constraints requires solving the longest-path problem in cyclic graphs.
- Two cases are distinguished:
 - **There are positive cycles:** No bounded solution for longest paths. (The inequality constraints are conflicting.) We shall detect the cycles.
 - **All cycles are negative:** Polynomial-time algorithms exist.



153

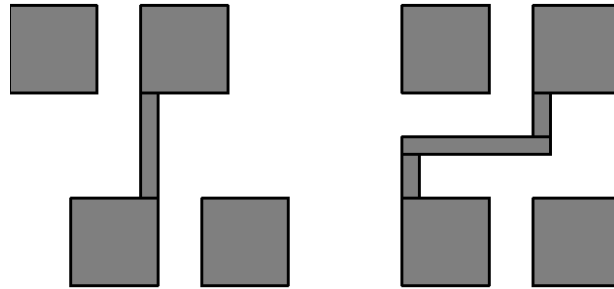
Longest and Shortest Paths

- Longest paths become shortest paths and vice versa when edge weights are multiplied by -1 .
- Situation in DAGs: both the longest and shortest path problems can be solved in **linear** time.
- Situation in cyclic directed graphs:
 - **All weights are positive:** shortest-path problem in P (Dijkstra), no feasible solution for the longest-path problem.
 - All weights are negative: longest-path problem in P (Dijkstra), no feasible solution for the shortest-path problem.
 - No positive cycles: longest-path problem is in P.
 - No negative cycles: shortest-path problem is in P.

154

Remarks on Constraint-Graph Compaction

- **Noncritical layout elements:** Every element outside the **critical paths** has freedom on its best position => may use this freedom to optimize some cost function.
- **Automatic jog insertion:** The quality of the layout can further be improved by automatic **jog insertion**.

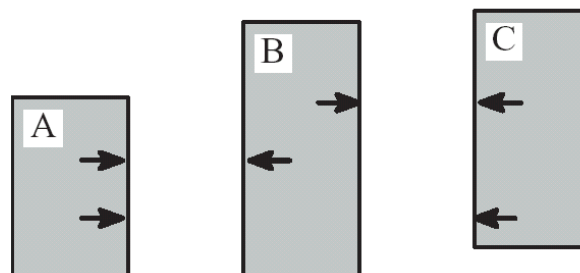


- **Hierarchy:** A method to reduce complexity is hierarchical compaction, e.g., consider cells only.

155

Constraint Generation

- The set of constraints should be irredundant and generated efficiently.
- An edge (v_i, v_j) is **redundant** if edges (v_i, v_k) and (v_k, v_j) exist and $w((v_i, v_j)) \leq w((v_i, v_k)) + w((v_k, v_j))$.
 - The minimum-distance constraints for (A, B) and (B, C) make that for (A, C) redundant.



- Doenhardt and Lengauer have proposed a method for irredundant constraint generation with complexity $O(n \log n)$.

156