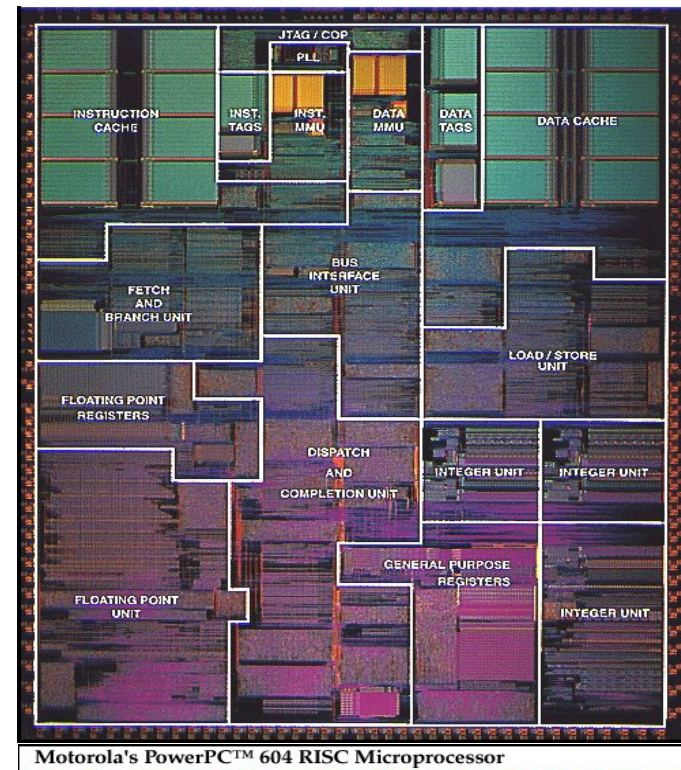


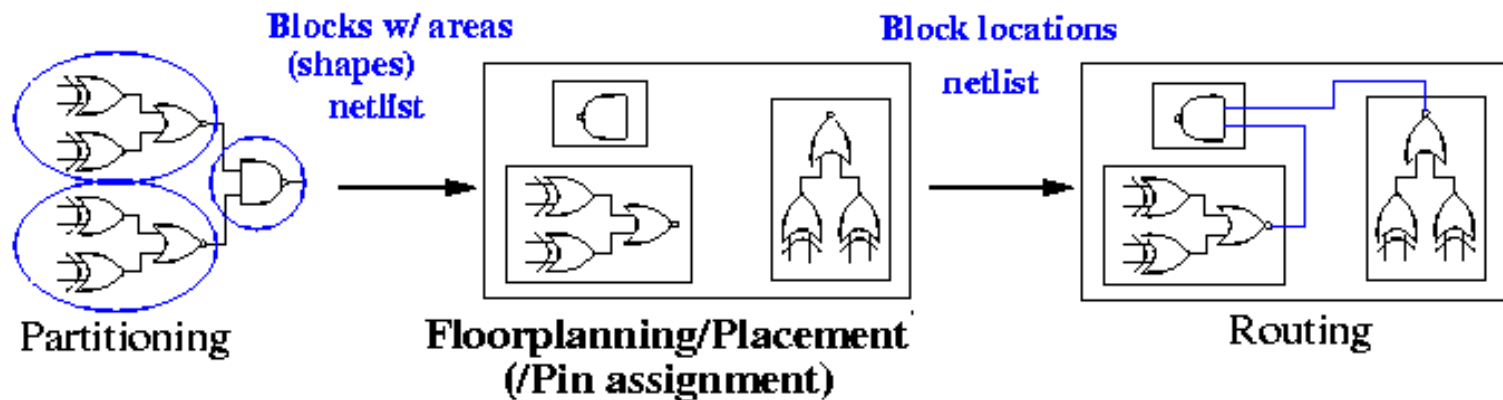
Floorplanning

- Course contents:
 - Normalized polish expression for **slicing** floorplans
 - Sequence pair for general (**non-slicing**) floorplans
 - Tree based non-slicing floorplans (B*-tree)
 - ILP for general floorplans
 - Modern floorplanning considerations



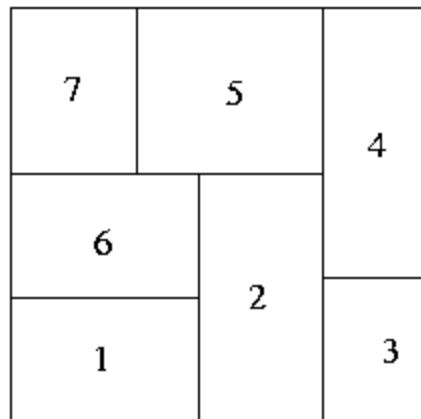
Floorplanning/Placement

- Partitioning leads to
 - Blocks with well-defined **areas and shapes** (**rigid/hard** blocks).
 - Blocks with approximated 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.

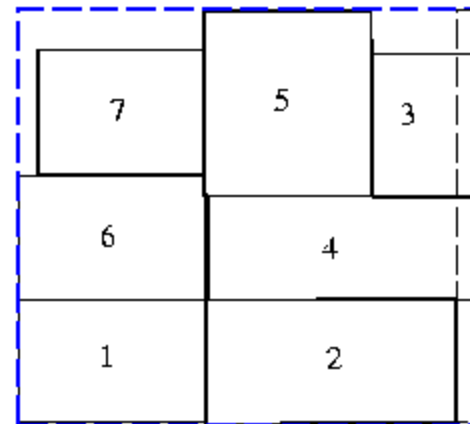


Floorplanning Problem

- 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

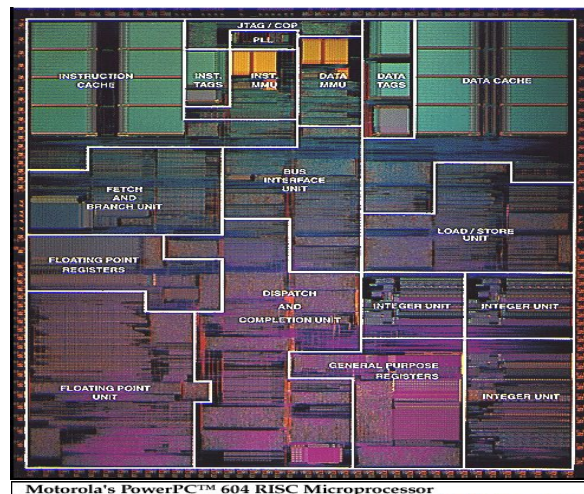
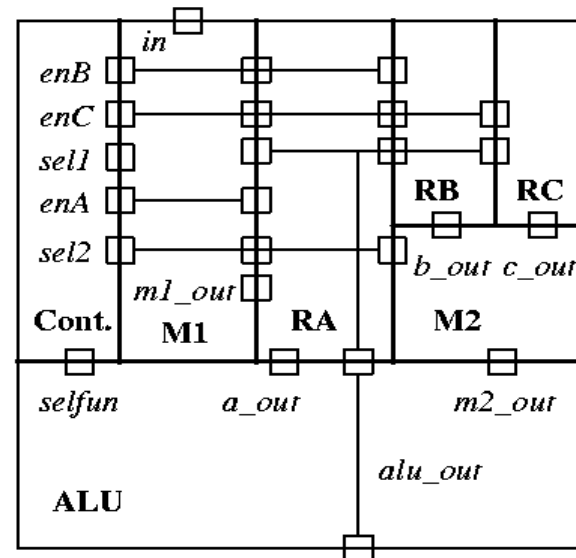
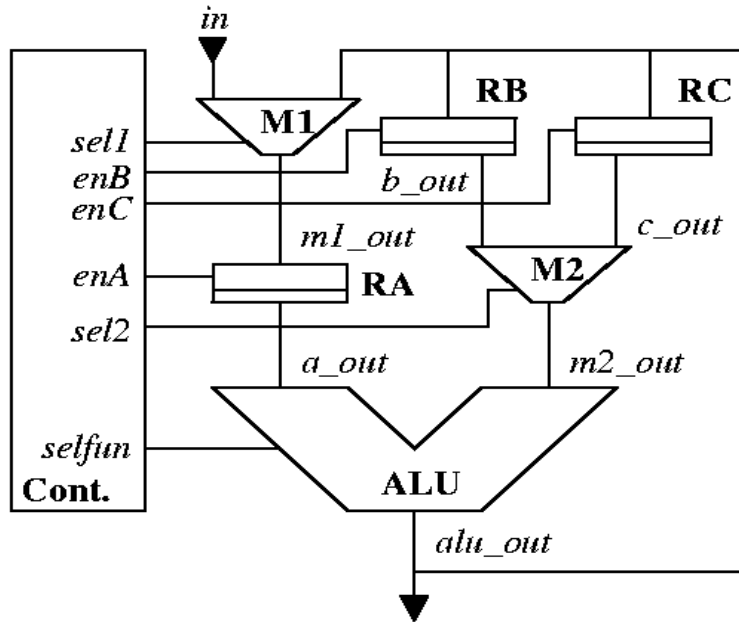


An optimal floorplan,
in terms of area

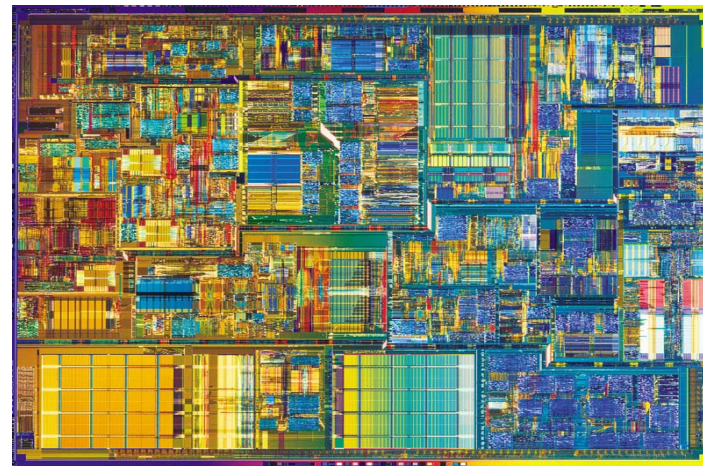


A non-optimal floorplan

Floorplan Examples



PowerPC 604

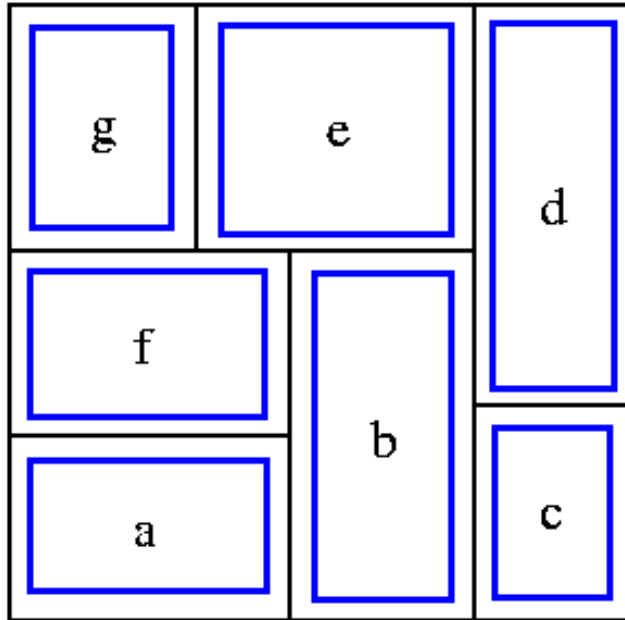





Pentium 4

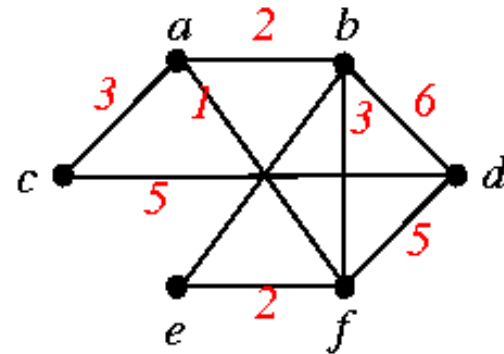
Early Layout Decision Methodology

- An IC is a 2-D medium; consider the dimensions of blocks in early stages of the design helps to improve the quality.
- Floorplanning gives early feedback
 - Suggests valuable architectural modifications
 - Estimates the whole chip area
 - Estimates delay and congestion due to wiring
- Floorplanning fits very well in a *top-down* design strategy; the *step-wise refinement* strategy also propagated in software design.
- Floorplanning considers the *flexibility* in the shapes and terminal locations of blocks.

Floorplan Design

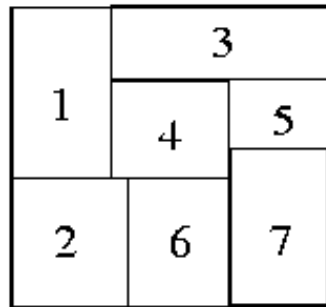


- *Modules:*  x y
- *Area:* $A=xy$
- *Aspect ratio:* $r \leq y/x \leq s$
- *Rotation:*  
- *Module connectivity*

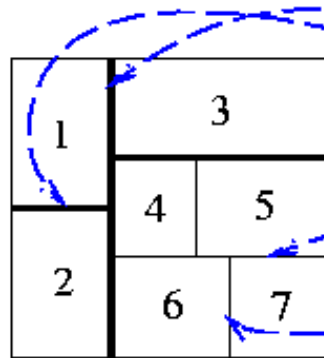


Slicing Floorplan Structure

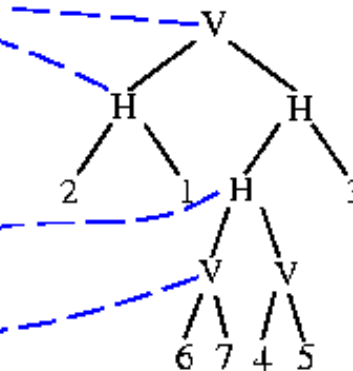
- **Rectangular dissection:** Subdivision of a given rectangle by a finite # of horizontal and vertical line segments into a finite # of non-overlapping rectangles.
- **Slicing structure:** a rectangular dissection that can be obtained by repetitively subdividing rectangles horizontally or vertically.
- **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.



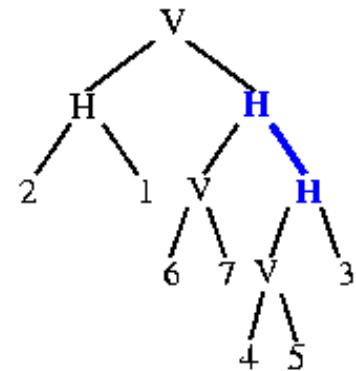
Non-slicing floorplan



Slicing floorplan



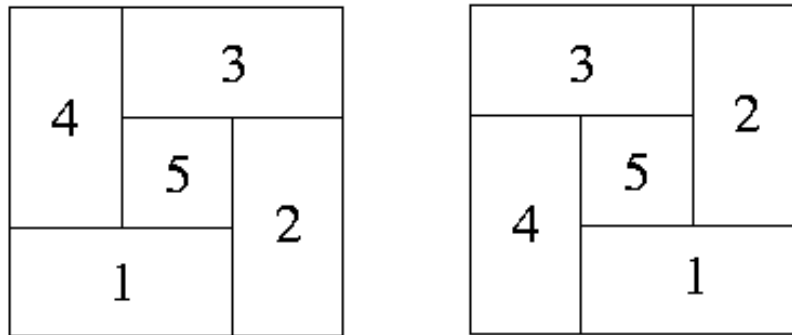
A slicing tree (skewed)



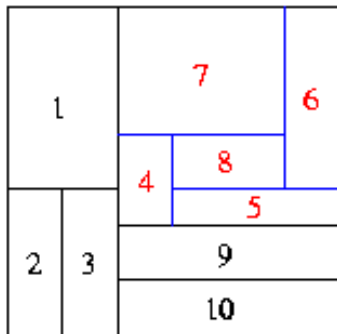
Another slicing tree (non-skewed)

Floorplan Order

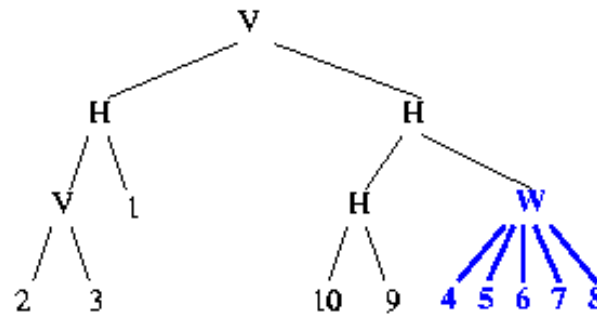
- **Wheel:** The smallest non-slicing floorplans (Wang and Wong, TCAD, Aug. 92).
- **Order of a floorplan:** a slicing floorplan is of order 2.
- **Floorplan tree:** A tree representing the hierarchy of partitioning.



The two possible wheels.



A floorplan of order 5



Corresponding floorplan tree

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.


- Ingredients

- solution space
- neighborhood structure
- cost function
- annealing schedule

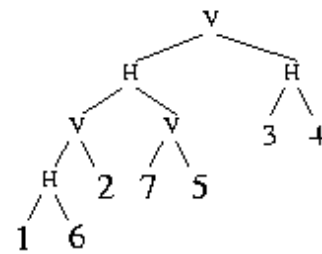
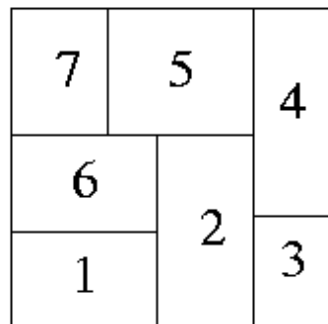
Solution Representation

- 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
 - every operand j , $1 \leq j \leq n$, appears exactly once in E ;
 - (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 .

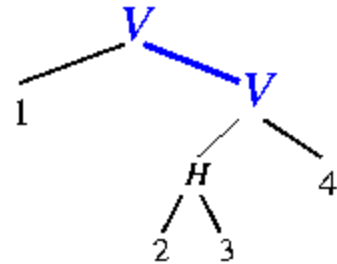
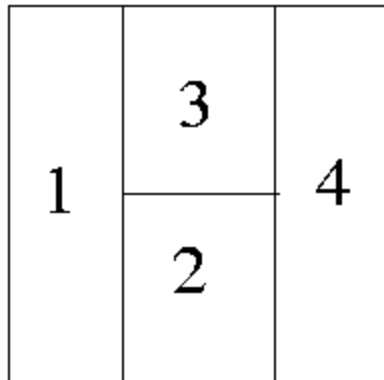


$E = 16H2V75VH34HV$

$E = 16 + 2 * 75 * + 34 + *$

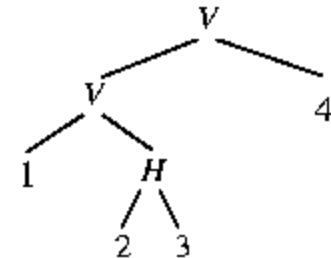
Postorder traversal of a tree!

Redundant Representation



$E = 123H4VV$

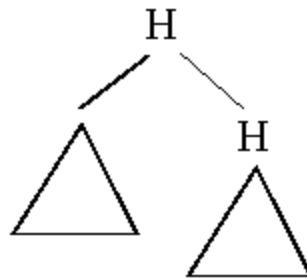
non-skewed!



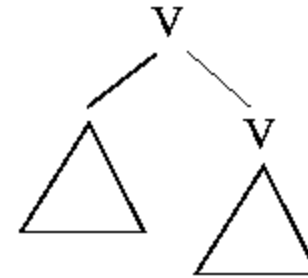
$E = 123HV4V$

skewed!

Non-skewed cases



..... HH

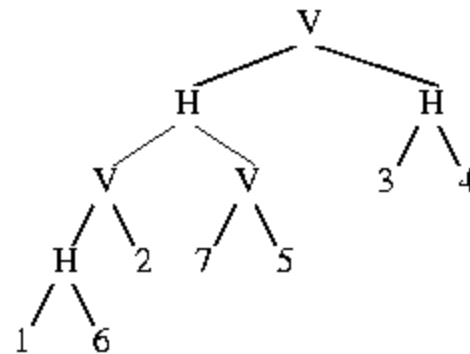
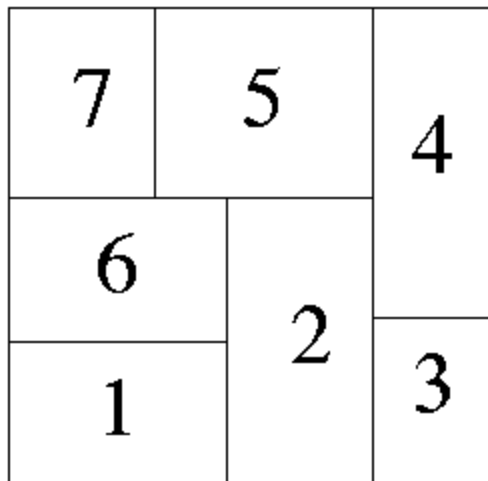


..... VV

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

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).
- Given a **normalized Polish expression**, we can construct a **unique** rectangular slicing structure.

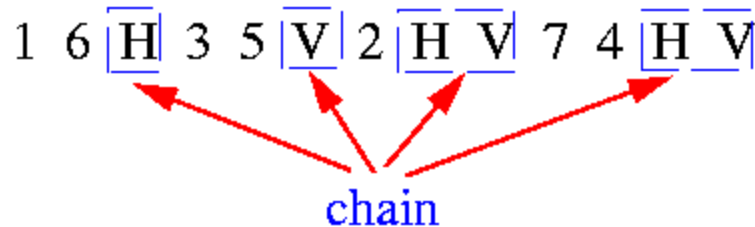


$E = 16H2V75VH34HV$

A normalized Polish expression

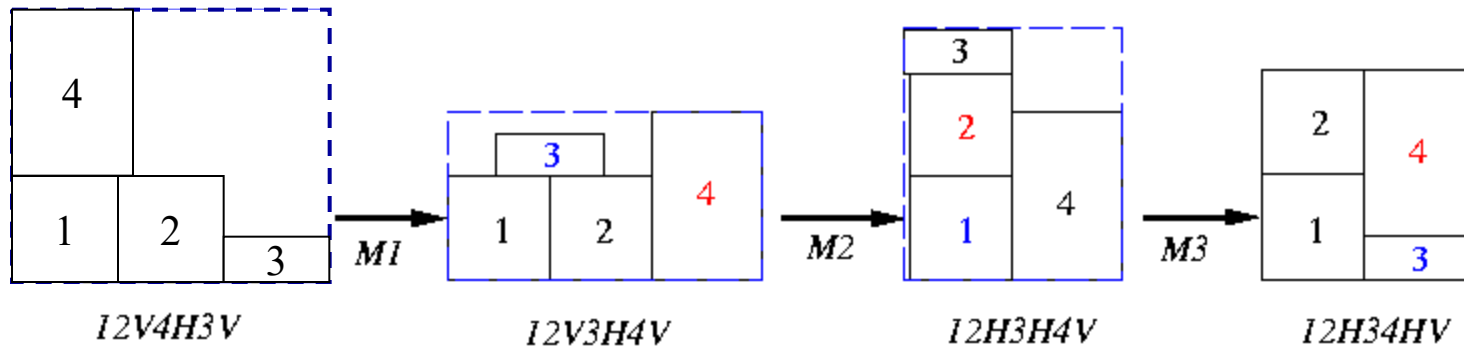
Neighborhood Structure

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



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

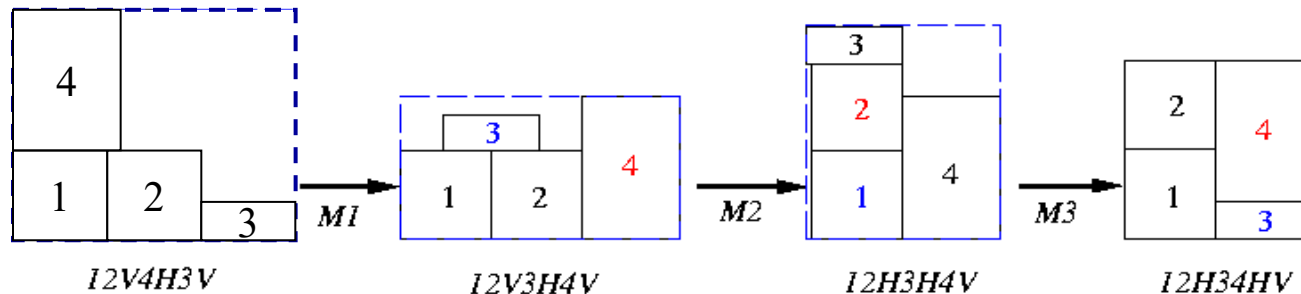
Effects of Perturbation



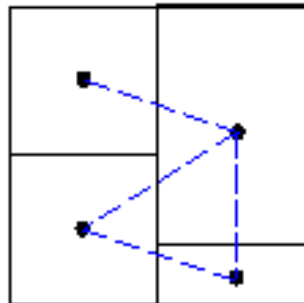
- **Question:** The balloting property holds during the moves?
 - $M1$ and $M2$ moves are OK.
 - **Check the $M3$ moves!** Reject “illegal” $M3$ moves.
- **Check $M3$ moves:** Assume that $M3$ swaps the operand e_i with the operator e_{i+1} , $1 \leq i \leq k-1$. Then, the swap will not violate the balloting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

Cost Function

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

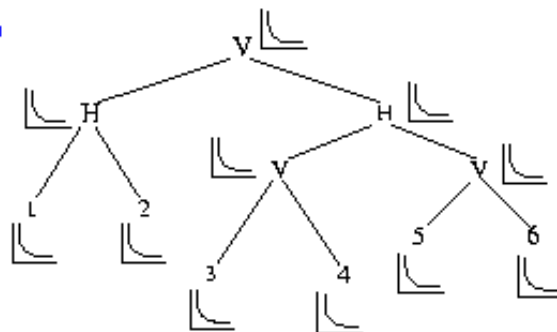
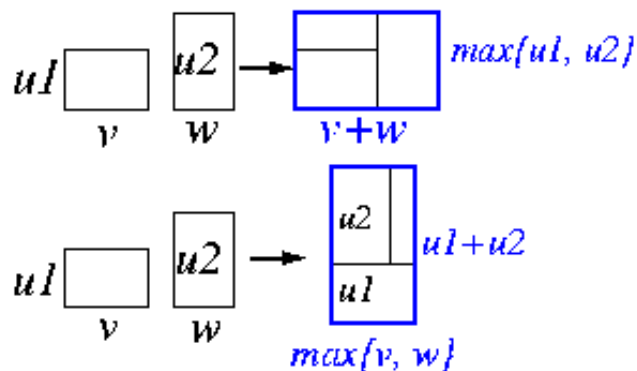
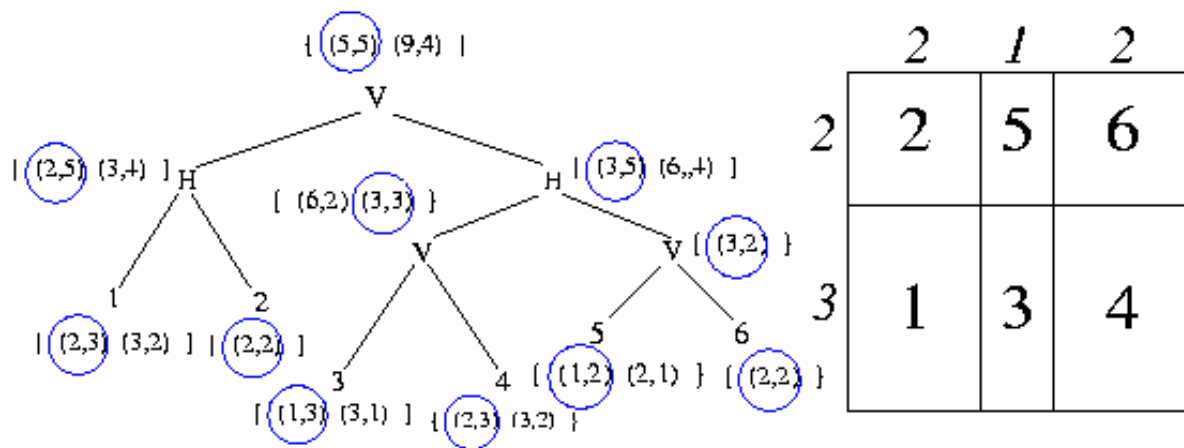


- $W = \sum_{ij} c_{ij} d_{ij}$.
 - c_{ij} : # of connections between blocks i and j .
 - d_{ij} : center-to-center distance between basic rectangles i and j .



Area Computation for Hard Blocks

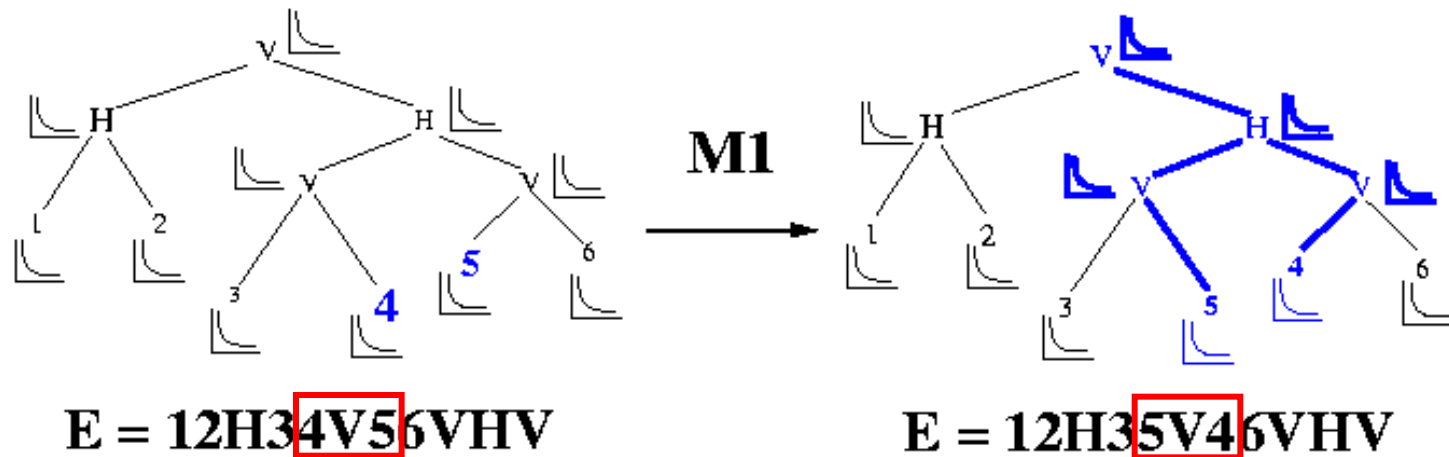
- Allow rotation



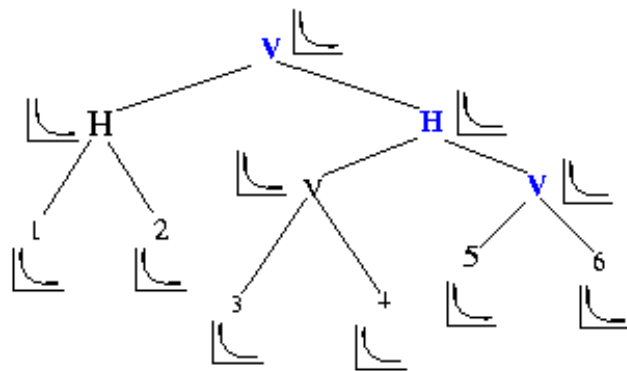
- Wiring cost?
 - Center-to-center interconnection length

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.

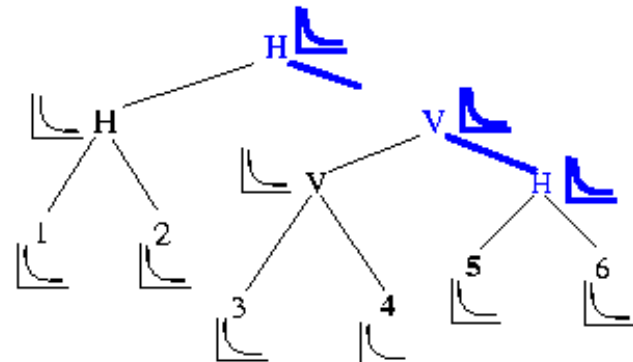


Incremental Computation of Cost Function (cont)

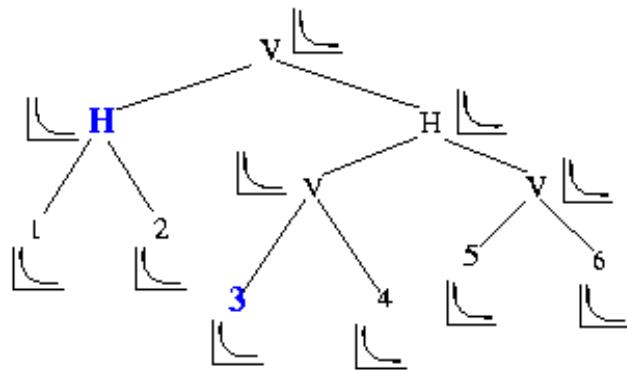


E = 12H34V56VHV

M2

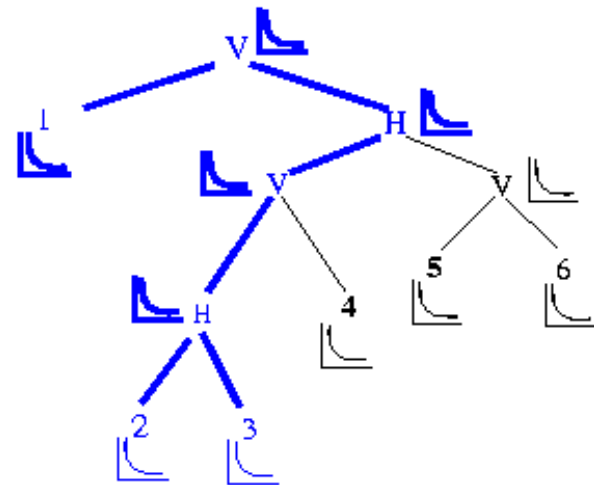


E = 12H34V56HVH



E = 12H34V56VHV

M3



E = 123H4V56VHV

Annealing Schedule

- Initial solution: 12V3V ... 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.

Algorithm: Wong-Liu (P, ε, r, k)

```

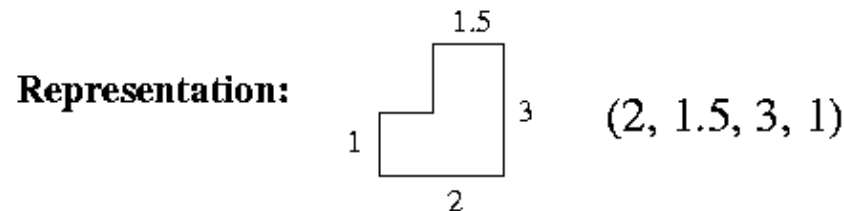
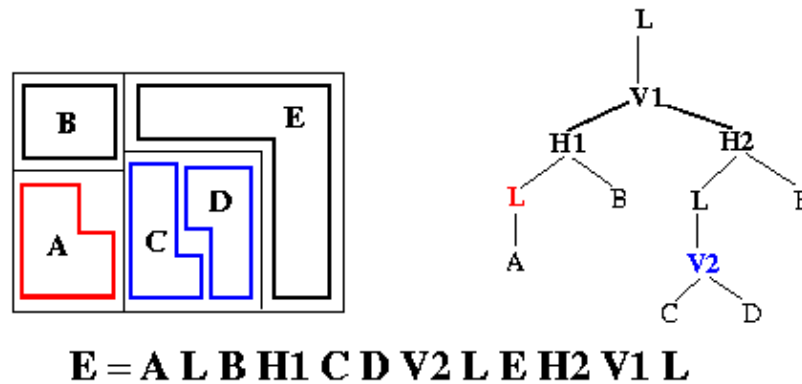
1 begin
2  $E \leftarrow 12V3V4V \dots nV$ ; /* initial solution */
3  $Best \leftarrow E$ ;  $T_0 \leftarrow \frac{\Delta_{avg}}{\ln(P)}$ ;  $M \leftarrow MT \leftarrow uphill \leftarrow 0$ ;  $N = kn$ ;
4 repeat
5    $MT \leftarrow uphill \leftarrow reject \leftarrow 0$ ;
6   repeat
7     SelectMove( $M$ );
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  $\leftarrow \text{TRUE}$ ;
15         $NE \leftarrow \text{Swap}(E, e_i, e_{i+1})$ ;
16         $MT \leftarrow MT+1$ ;  $\Delta cost \leftarrow cost(NE) - cost(E)$ ;
17        if ( $\Delta cost \leq 0$ ) or ( $\text{Random} < \frac{-\Delta cost}{e^{\frac{1}{T}}}$ )
18          then
19            if ( $\Delta cost > 0$ ) then  $uphill \leftarrow uphill + 1$ ;
20             $E \leftarrow NE$ ;
21            if  $cost(E) < cost(best)$  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

```

H.-M. Chen

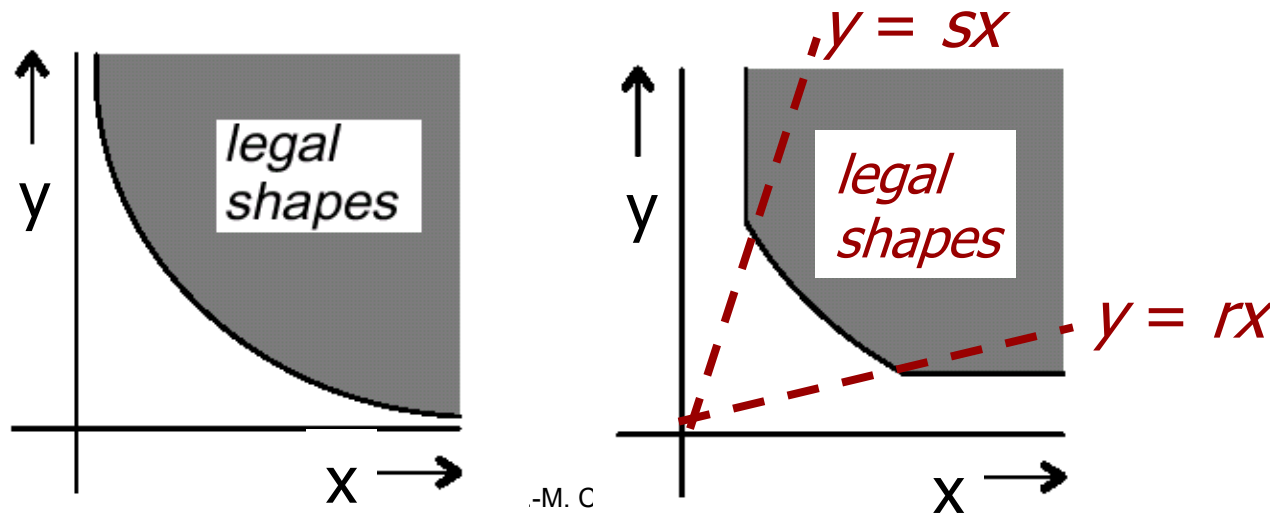
Extension to L-Shaped Modules

- Unary operator L : Change an L-shaped figure into a rectangle
- Binary operators V_1, V_2, H_1, H_2 : Combine 2 rectangles or L-shaped figures to form a rectangle or an L-shaped figure.
- Can generate non-slicing floorplans.



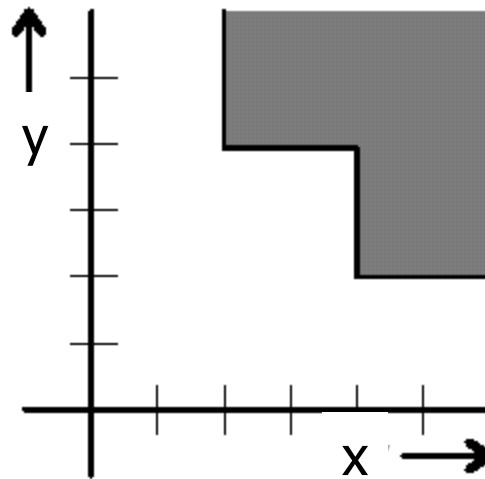
Shape Curve for Floorplan Sizing

- A soft (flexible) blocks b can have different aspect ratios, but is with a fixed area A .
- The shape function of b is a hyperbola: $xy = A$, or $y = A/x$, for width x and height y .
- Very thin blocks are often not interesting and feasible to design
 - Add two straight lines for the constraints on aspect ratios.
 - Aspect ratio: $r \leq y/x \leq s$.



Shape Curve

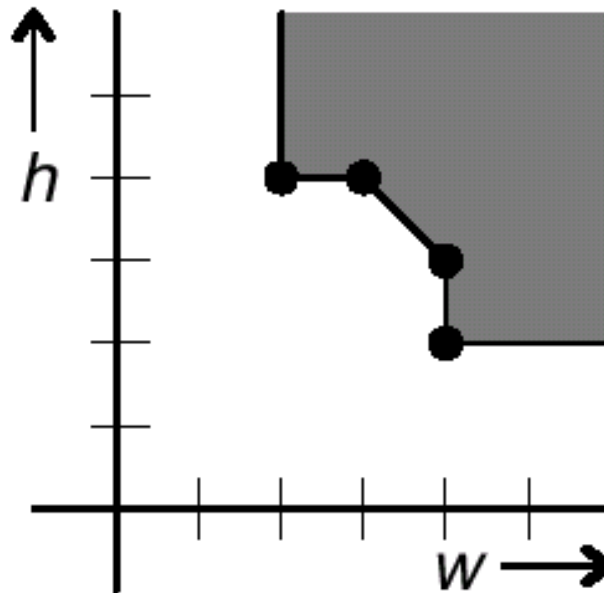
- Since a basic block is built from discrete transistors, it is not realistic to assume that the shape function follows the hyperbola continuously.
- In an extreme case, a block is rigid/hard: it can only be rotated and mirrored during floorplanning or placement.



The shape curve of a 2×4 hard block.

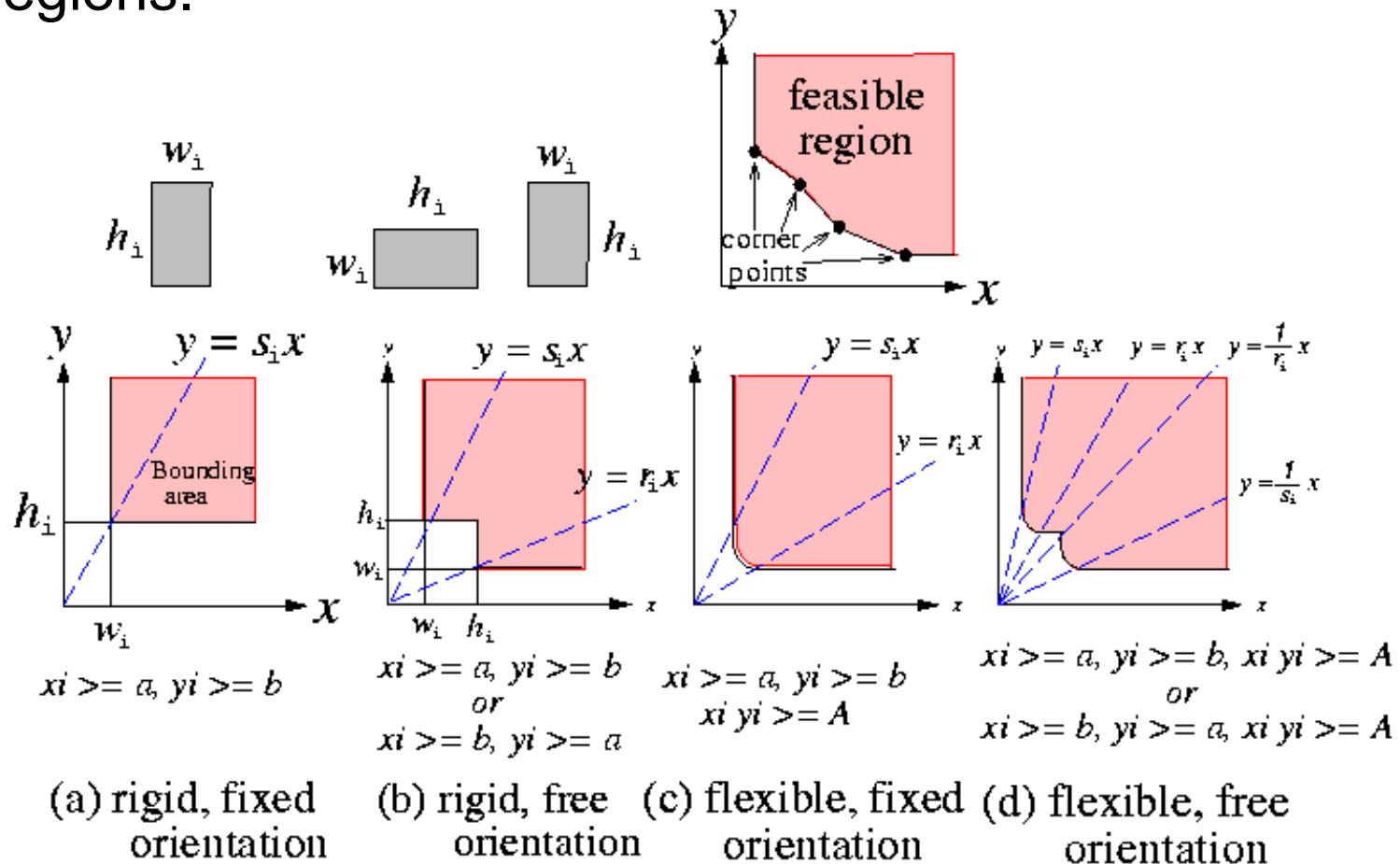
Shape Curve (cont)

- 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.



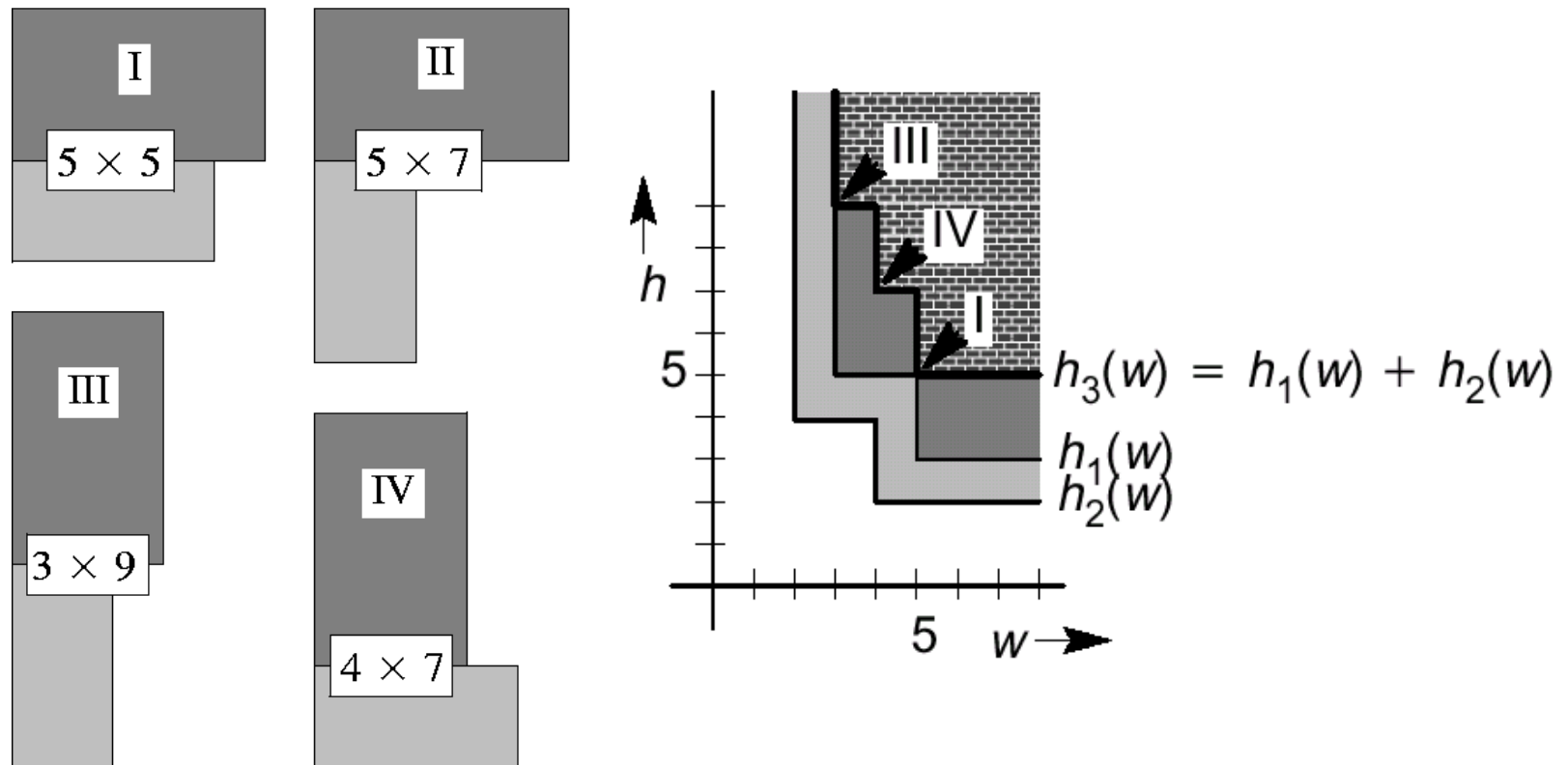
Feasible Implementations

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



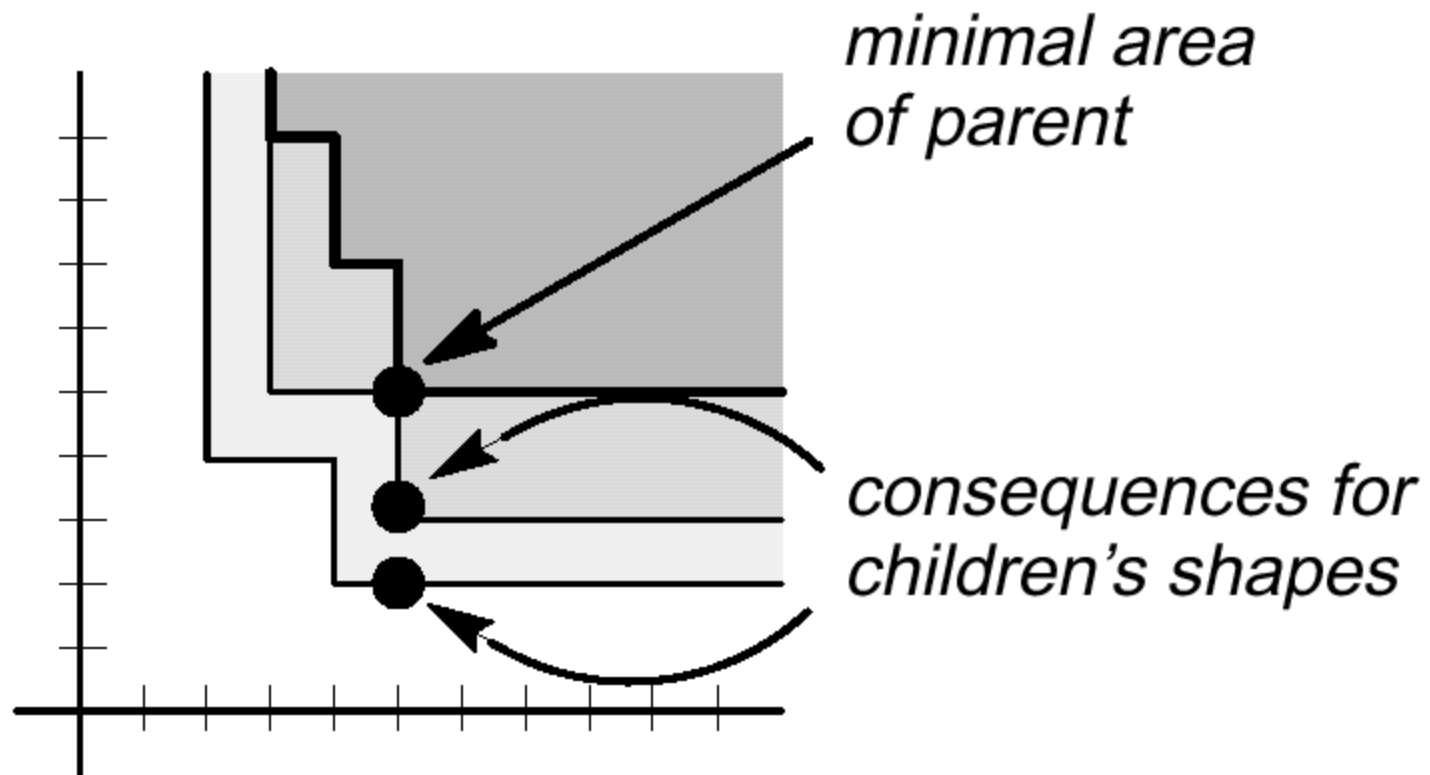
Vertical Abutment

- Composition by vertical abutment (horizontal cut) \Rightarrow the addition of shape functions.



Deriving Shapes of Children

- A choice for the minimal shape of a **composite block** fixes the shapes of its children blocks.



Slicing Floorplan Sizing

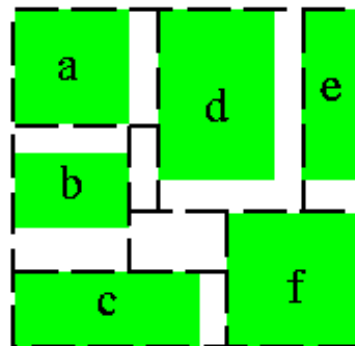
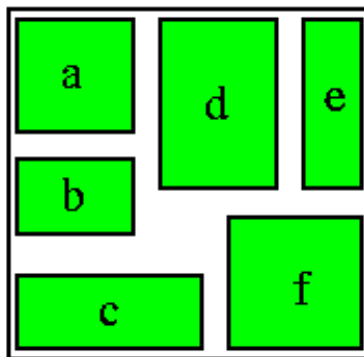
- The shape functions of all **leaf blocks** are given as piecewise linear functions.
- Traverse the slicing tree to compute the shape functions of all composite blocks (**bottom-up composition**).
- Choose the desired shape of the top-level block
 - Only the corner points of the function need to be evaluated for area minimization.
- Propagate the consequences of the choice down to the leaf blocks (**top-down propagation**).
- The sizing algorithm runs in polynomial time for slicing floorplans
 - NP-complete for non-slicing floorplans

P*-admissible Solution Space

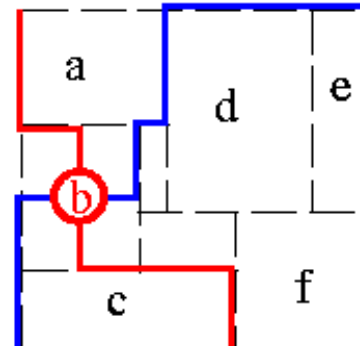
- **P-admissible** solution space for Problem P (Murata et al., ICCAD-95)
 1. the solution space is finite,
 2. every solution is feasible,
 3. evaluation for each configuration is possible in polynomial time and so is the implementation of the corresponding configuration (P), **and**
 4. the configuration corresponding to the best evaluated solution in the space coincides with an optimal solution of P. (admissible)
- **P*-admissible** solution space (Lin & Chang, DAC-2002)
 5. The relationship between any two blocks is defined in the representation (topological representation).
- Slicing floorplan is **not** P-admissible. Why?
- P*-admissible floorplan representations: **Sequence Pair, BSG, TCG, TCG-S.**

Sequence Pair (SP)

- Murata, Fujiyoshi, Nakatake, Kajitani, “Rectangle-Packing Based Module Placement,” ICCAD-95 (also in *The Best of ICCAD*)
- Represent a packing by a pair of module-name sequences (e.g., $(abdecf, cbfade)$).
 - Solution space: $(n!)^2$
- Correspond all pairs of the sequences to a P-admissible solution space.
- Search in the P-admissible solution space (by simulated annealing).
 - Swap two nodes only in a sequence
 - Swap two nodes in both sequences



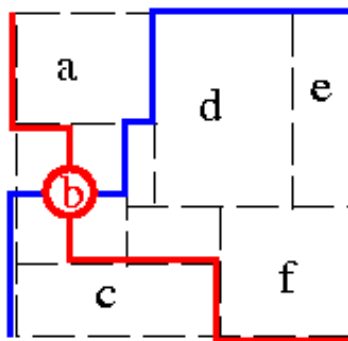
A floorplan



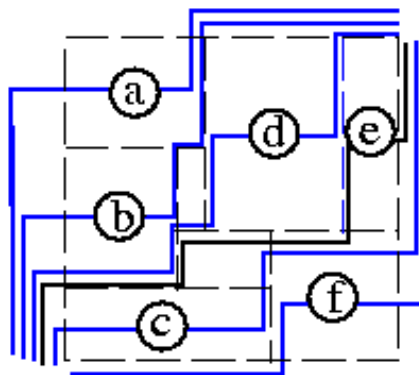
Loci of module b

Relative Module Positions

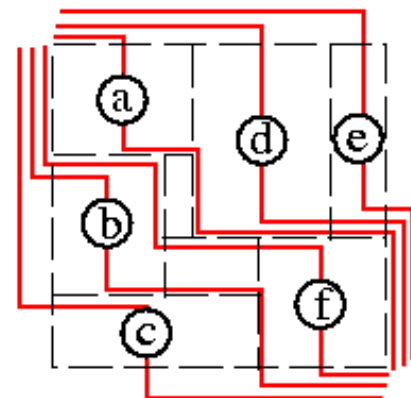
- A floorplan is a partition of a chip into **rooms**, each containing at most one block.
- **Locus** (right-up, left-down, up-left, down-right)
 1. Take a non-empty room.
 2. Start at the center of the room, walk in two alternating directions to hit the sides of rooms.
 3. Continue until to reach a corner of the chip.
- **Positive locus** Γ_+ : Union of right-up locus and left-down locus.
- **Negative locus** Γ_- : Union of up-left locus and down-right locus.



Loci of module b



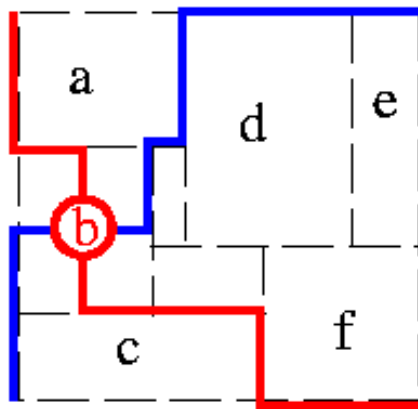
Positive loci: abdecf



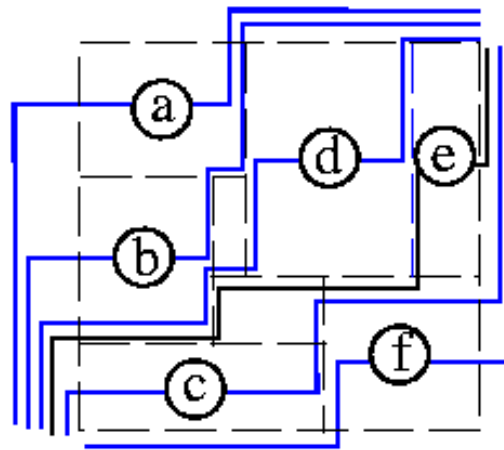
Negative loci: cbfade

Geometrical Information

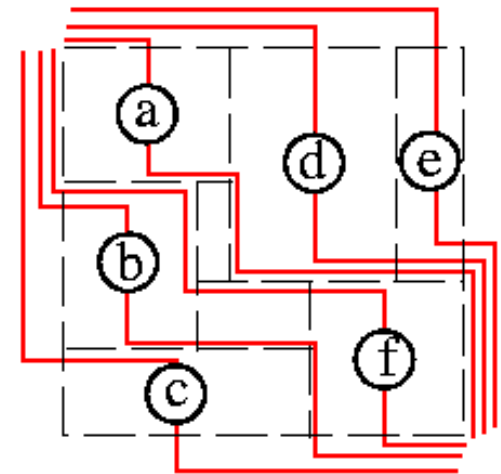
- No pair of positive (negative) loci cross each other, i.e., **loci are linearly ordered**.
- SP uses two sequences (Γ_+ , Γ_-) to represent a floorplan.
 - **H-constraint:** ($..a..b..$, $..a..b..$) iff a is on the left of b
 - **V-constraint:** ($..a..b..$, $..b..a..$) iff b is below a



Loci of module b



Positive loci: abdecf

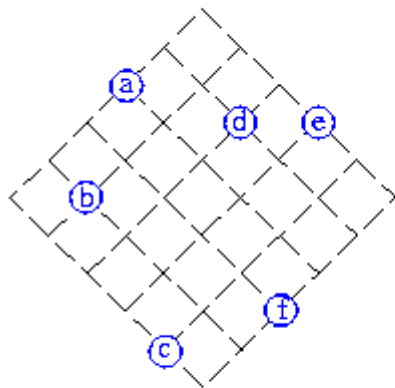


Negative loci: cbfade

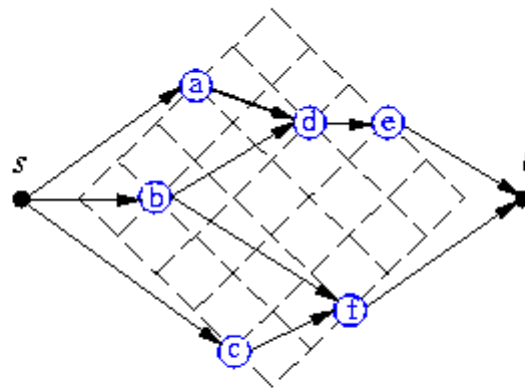
$$(\Gamma_+, \Gamma_-) = (abdecf, cbfade)$$

(\square_+, \square_-) -Packing

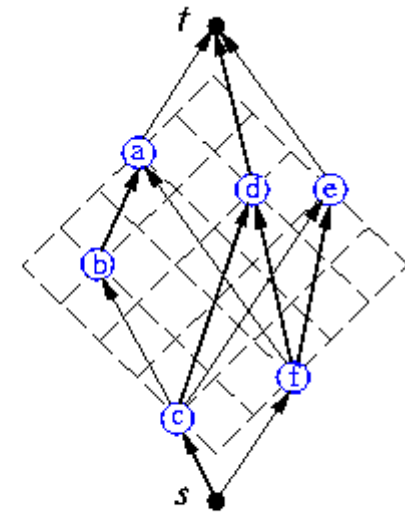
- For every SP (\square_+, \square_-) , there is a (\square_+, \square_-) packing.
- **Horizontal constraint graph** $G_H(V, E)$ (similarly for $G_V(V, E)$):
 - V : source s , sink t , n vertices for modules.
 - E : (s, x) and (x, t) for each module x , and (x, y) iff x must be left to y .
 - **Vertex weight**: 0 for s and t , **width** of module x for the other vertices.



Packing for sequence pair:
(*abdecf, cbfude*)



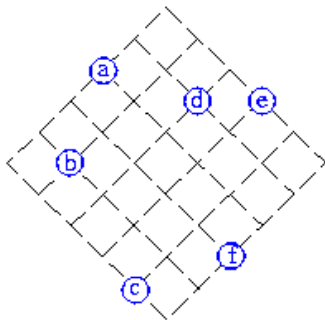
Horizontal constraint graph
(Transitive edges are not shown)



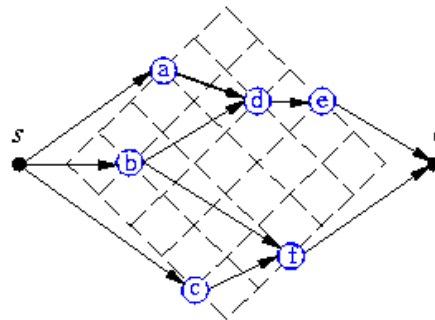
Vertical constraint graph
(Transitive edges are not shown)

Cost Evaluation for Sequence Pair (1/3)

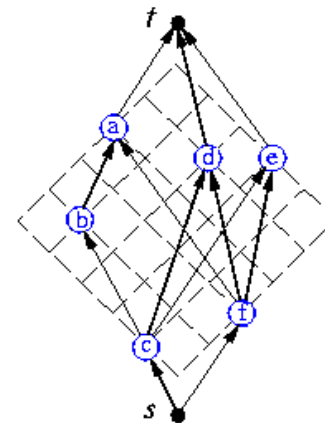
- Graph-based packing computation
 - **Optimal** (\square_+ , \square_-)-Packing can be obtained in $O(n^2)$ time by applying a longest path algorithm on a vertex-weighted directed acyclic graph. (Murata et.al., ICCAD-95)
 - G_H and G_V are independent.
 - The X and Y coordinates of each module are the minimum values of the longest path length between s and the corresponding vertex in G_H and G_V , respectively.



Packing for sequence pair:
(ubdefc, cbfude)



Horizontal constraint graph
(Transitive edges are not shown)



Vertical constraint graph
(Transitive edges are not shown)

Cost Evaluation for Sequence Pair (2/3)

- Graph-based packing computation (cont)
 - Building constraint graph (relative placement computation)
 - $O(n^2)$ time (Murata et.al., ICCAD-95)
 - $O(n \log n)$ time (Lin et.al., ISCAS-2000) : Direct view algorithm
 - Mapping (absolute placement computation)
 - $O(n^2) \rightarrow O(n \log n)$
 - Incremental packing computation
 - $O(\sqrt{n} \log \sqrt{n})$ (Lin et.al., ECCTD-2001)

Cost Evaluation for Sequence Pair (3/3)

- Non-graph-based packing computation
 - Maximum-weighted common subsequence (Tang & Wong, DATE-2000 and ASP-DAC-2001)
 - Compute block positions
 - Based on computing the longest common subsequence in a pair of weighted sequences
 - Cost evaluation can be done in $O(n \lg \lg n)$ time (ASP-DAC-2001)

Maximum-Weight Common Subsequence (MWCS) (1/6)

- A **weighted sequence** is a sequence on a given set S , and every element in S has a weight.

- **Example:**

A sequence (4 3 1 6 2 5)

weight: 4 3 3 2 4 6

- Given 2 weighted sequences X and Y , a sequence Z is a **common subsequence** of X and Y if Z is a subsequence of both X and Y .

- **Example:**

$X=(4\ 3\ 1\ 6\ 2\ 5)$ $Y=(6\ 3\ 5\ 4\ 1\ 2)$

$Z=(3\ 1\ 2)$ is a common subsequence

Maximum-Weight Common Subsequence (MWCS) (2/6)

- The **length** of a common subsequence

$$Z=(z_1 z_2 \dots z_n) \text{ is: } \sum_{i=1}^n w(z_i)$$

- **MWCS** is the common subsequence with the maximal length:

$$\max_Z \sum_i w(z_i)$$

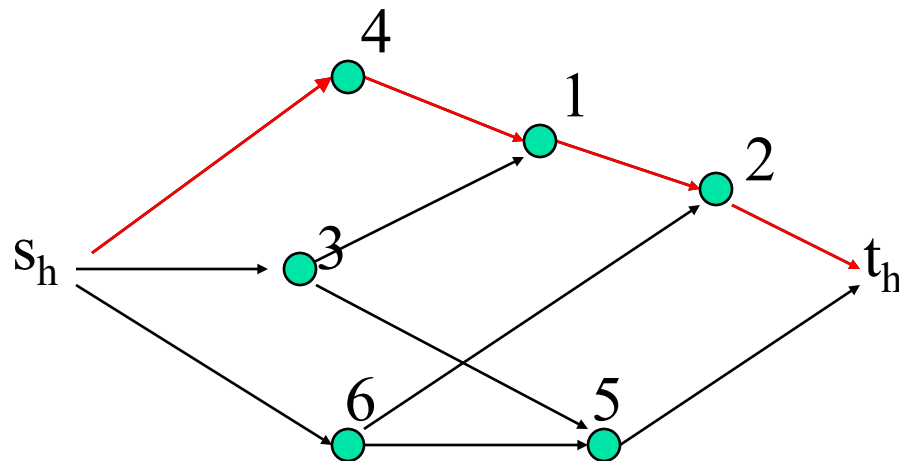
- **Example:**

$X=(4 \text{ } 3 \text{ } 1 \text{ } 6 \text{ } 2 \text{ } 5) \quad Y=(6 \text{ } 3 \text{ } 5 \text{ } 4 \text{ } 1 \text{ } 2)$

weight: $2 \text{ } 3 \text{ } 4 \text{ } 6 \text{ } 3 \text{ } 4 \quad 6 \text{ } 3 \text{ } 4 \text{ } 2 \text{ } 4 \text{ } 3$

$(3 \text{ } 1 \text{ } 2)$ is a MWCS. Its length is $10=3+4+3$.

Maximum-Weight Common Subsequence (MWCS) (3/6)



5 paths correspond to
5 comm. subseq. of (X, Y)

4 1 2

3 1 2

3 5

6 2

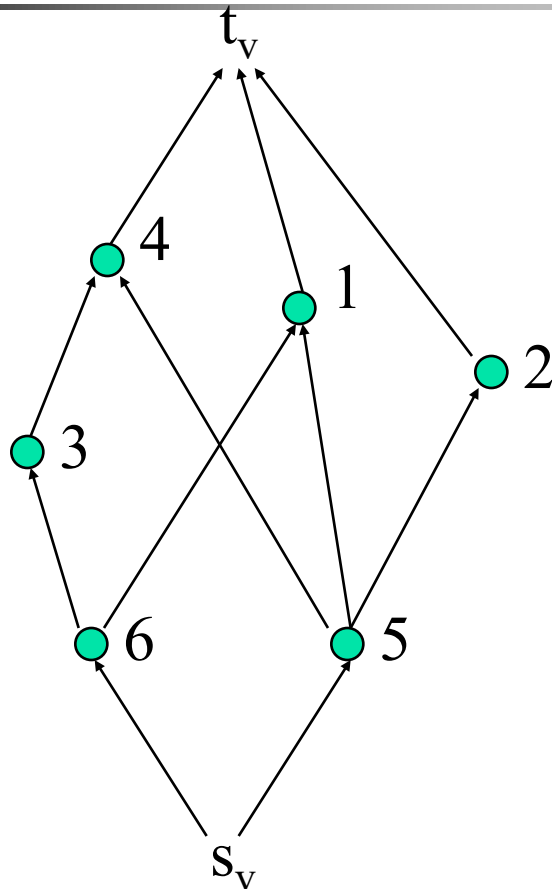
6 5

weight = width of block

Correspondence with constraint graph G_h

$(X, Y) = \langle 4 \ 3 \ 1 \ 6 \ 2 \ 5, 6 \ 3 \ 5 \ 4 \ 1 \ 2 \rangle$

Maximum-Weight Common Subsequence (MWCS) (4/6)



5 paths correspond to
5 comm. subseq. of (X^R, Y)

6 3 4

6 1

5 4

5 1

5 2

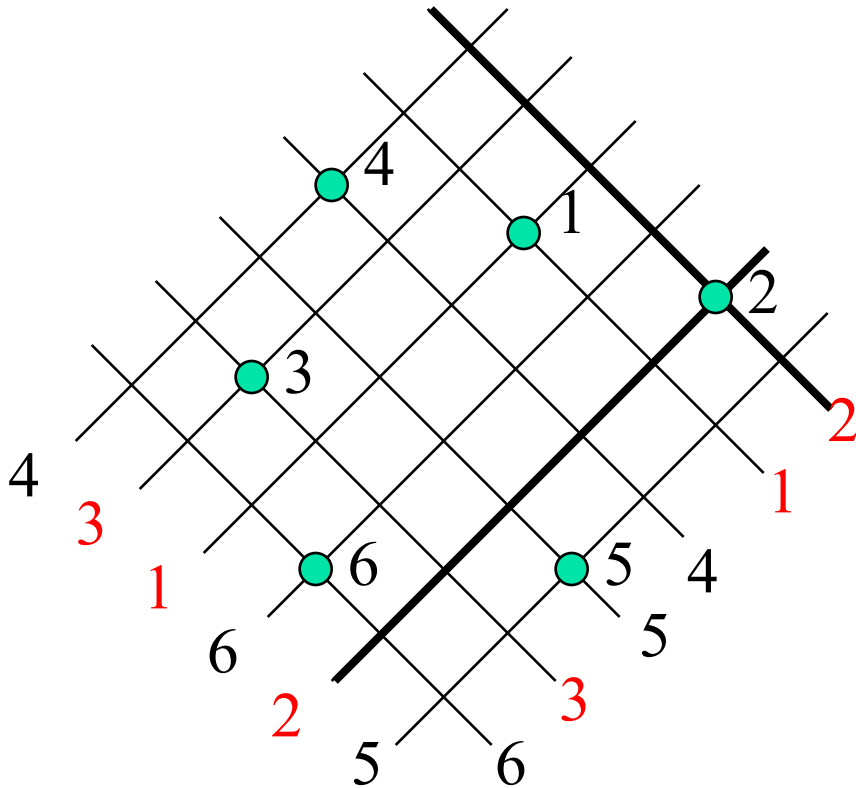
weight = height of block

Correspondence with constraint graph G_v

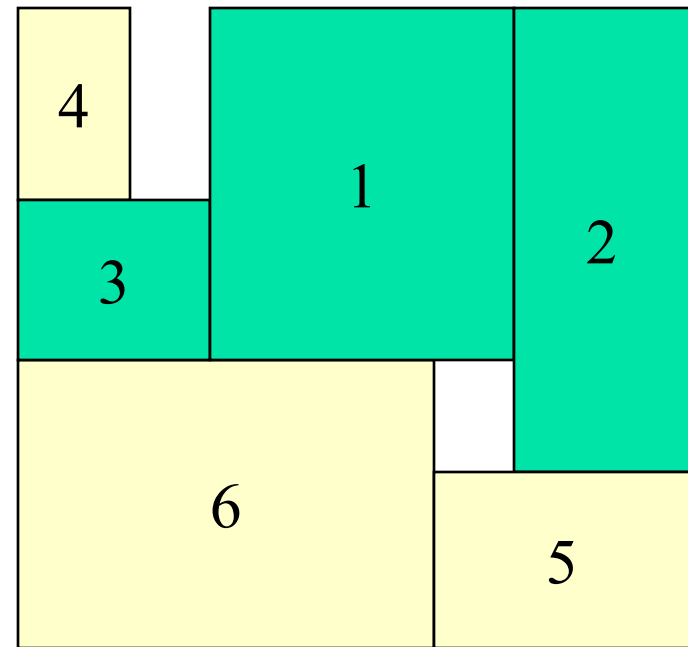
$(X^R, Y) = \langle 5\ 2\ 6\ 1\ 3\ 4, 6\ 3\ 5\ 4\ 1\ 2 \rangle$

X^R is the reverse of X

Maximum-Weight Common Subsequence (MWCS) (5/6)



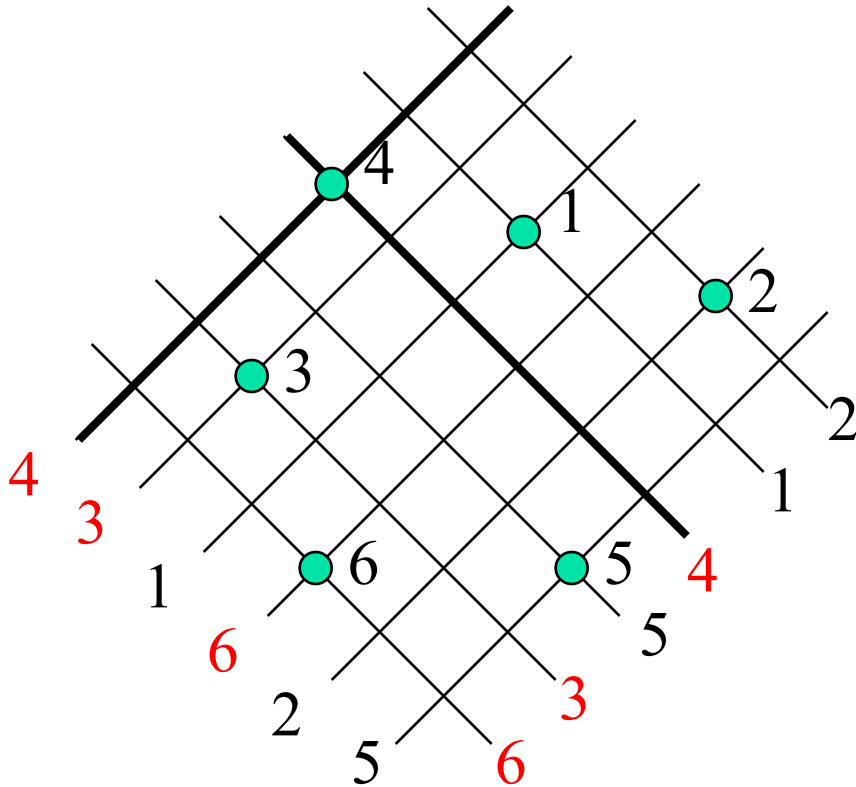
Oblique grid



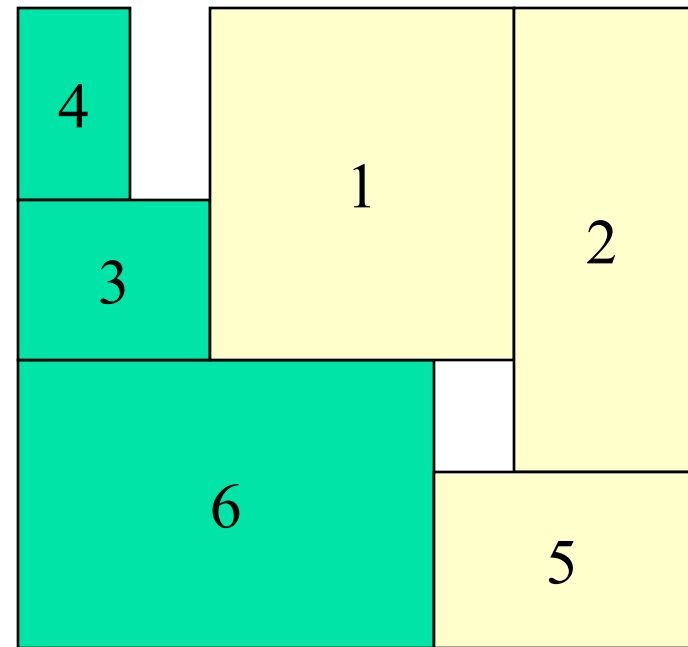
placement

Seq-Pair $(X, Y) = (4 \ 3 \ 1 \ 6 \ 2 \ 5, 6 \ 3 \ 5 \ 4 \ 1 \ 2)$, weight: blocks' width

Maximum-Weight Common Subsequence (MWCS) (6/6)



Oblique grid

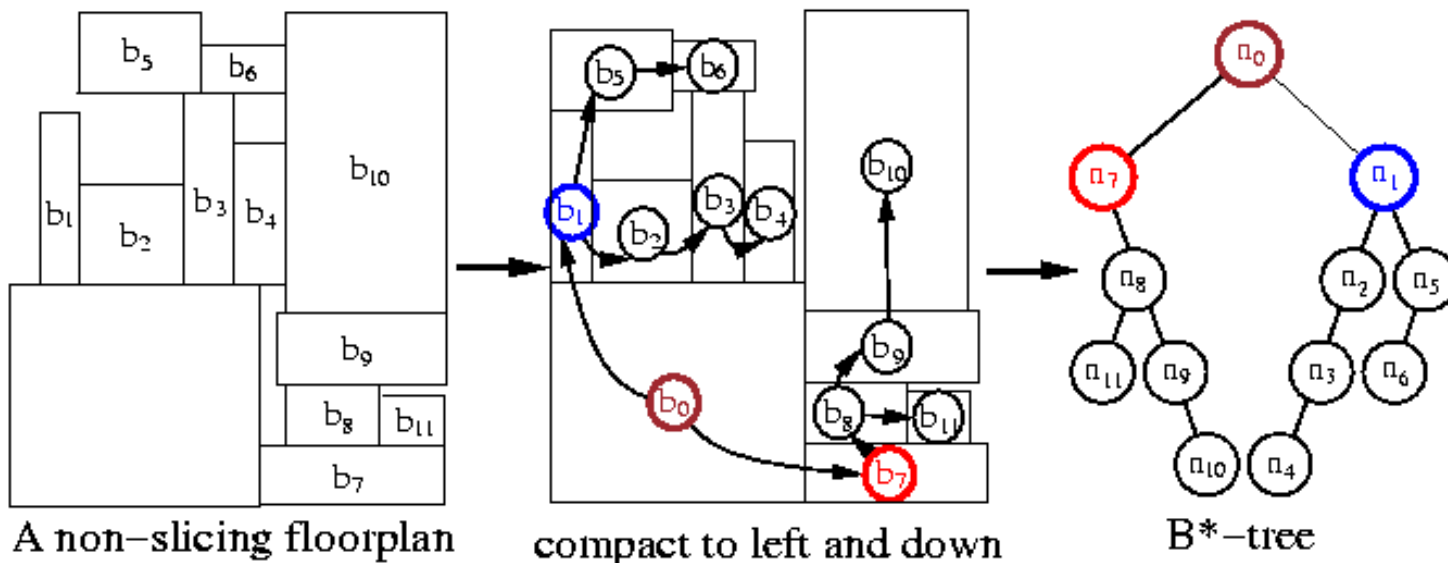


placement

Seq-Pair, $(X^R, Y) = (5 \ 2 \ 6 \ 1 \ 3 \ 4, 6 \ 3 \ 5 \ 4 \ 1 \ 2)$, weights: blocks' height

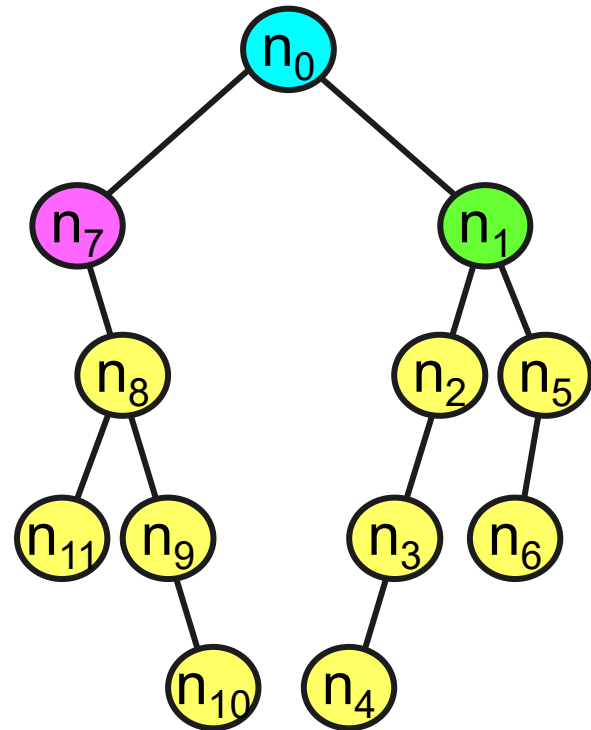
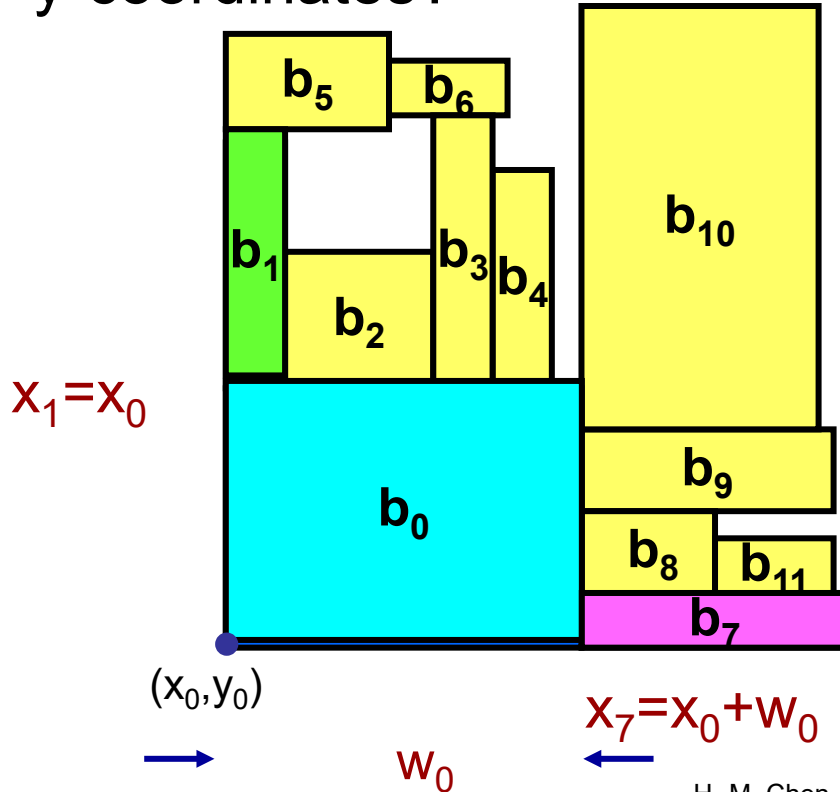
B*-Tree: Compacted Floorplan Representation

- Chang et. al., “B*-tree: A new representation for non-slicing floorplans,” DAC-2k.
 1. Compact modules to left and bottom.
 2. 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 ($x_j = x_i$).



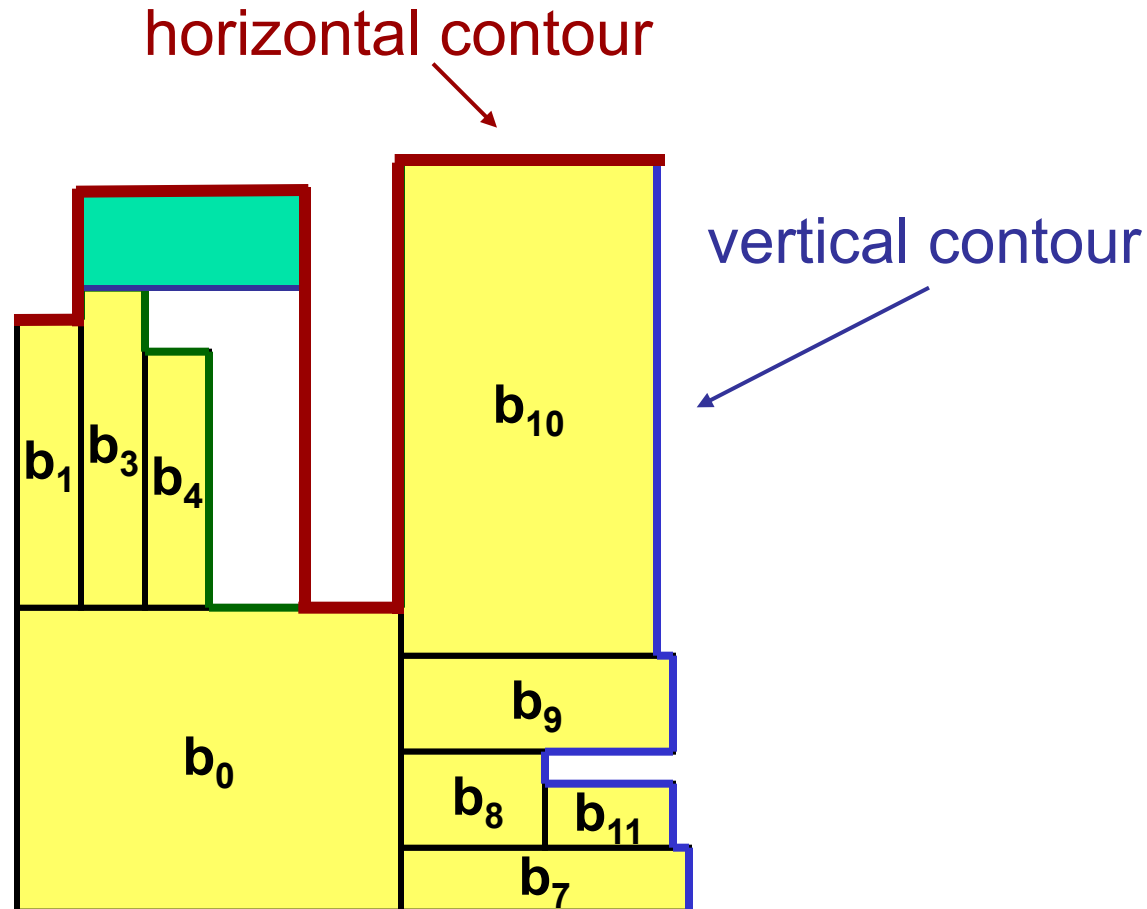
B*-tree Packing

- 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?

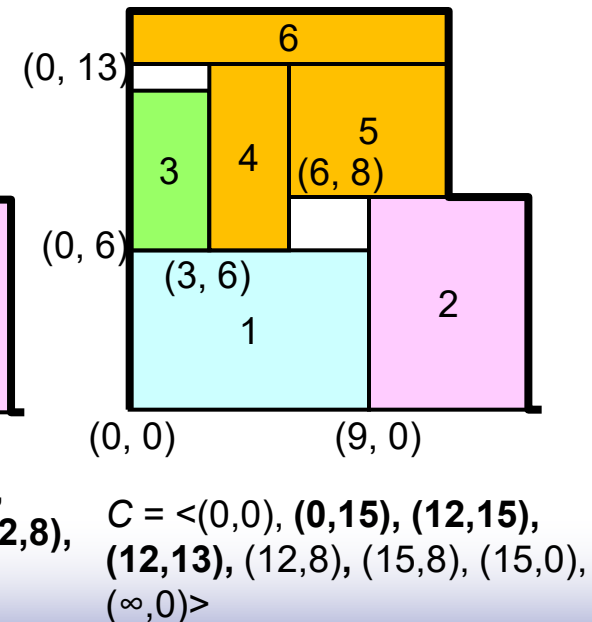
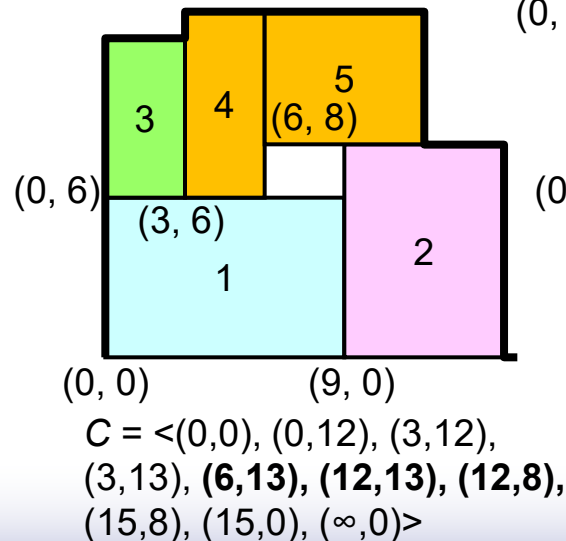
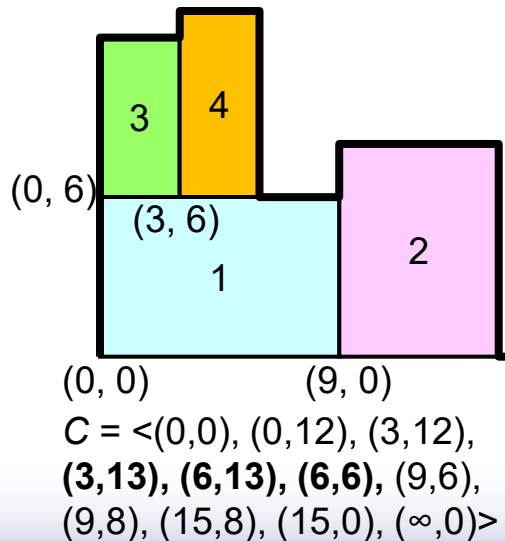
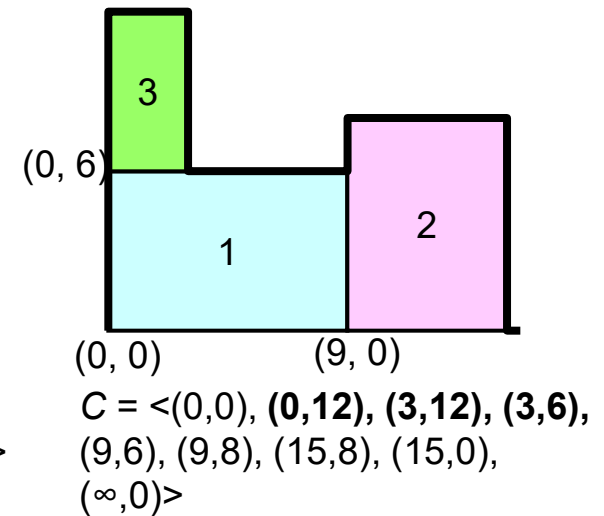
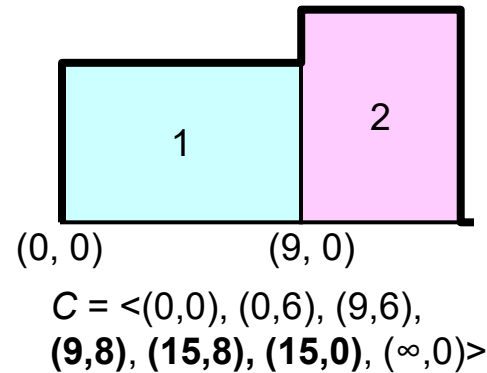
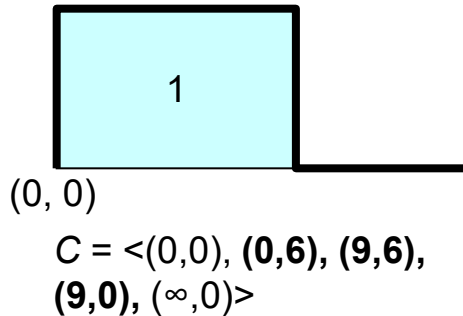


Computing y-coordinates

- Reduce the complexity of computing a y-coordinate to amortized $O(1)$ time. (same as in O-tree)

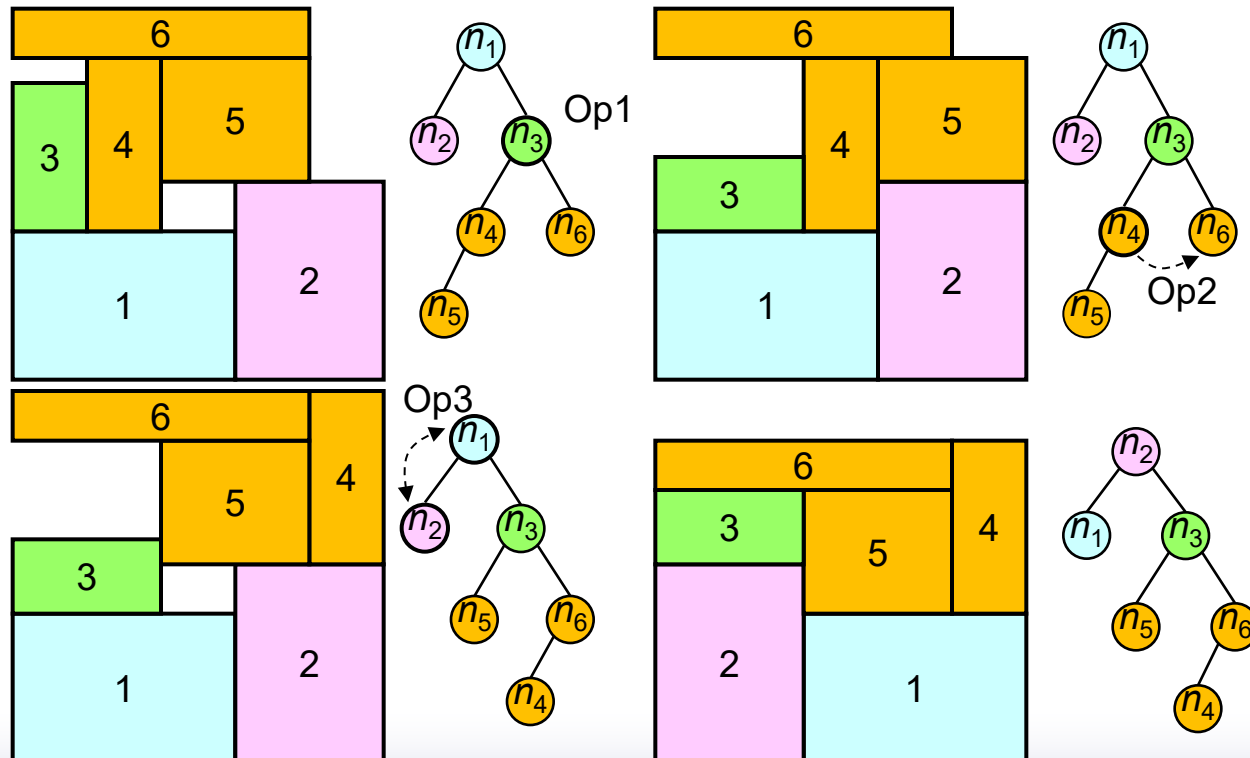


Contour Data Structure



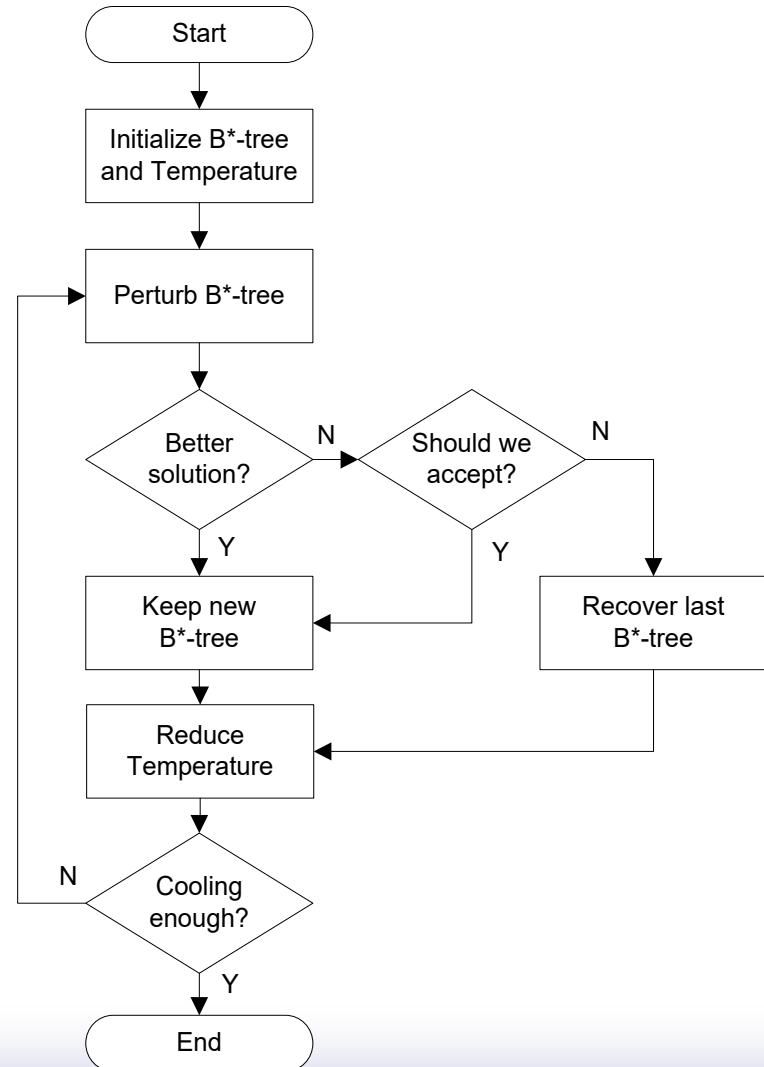
B-Tree Perturbation*

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



Simulated Annealing Using B^* -trees

- The cost function is based on problem requirements.



Pros and Cons

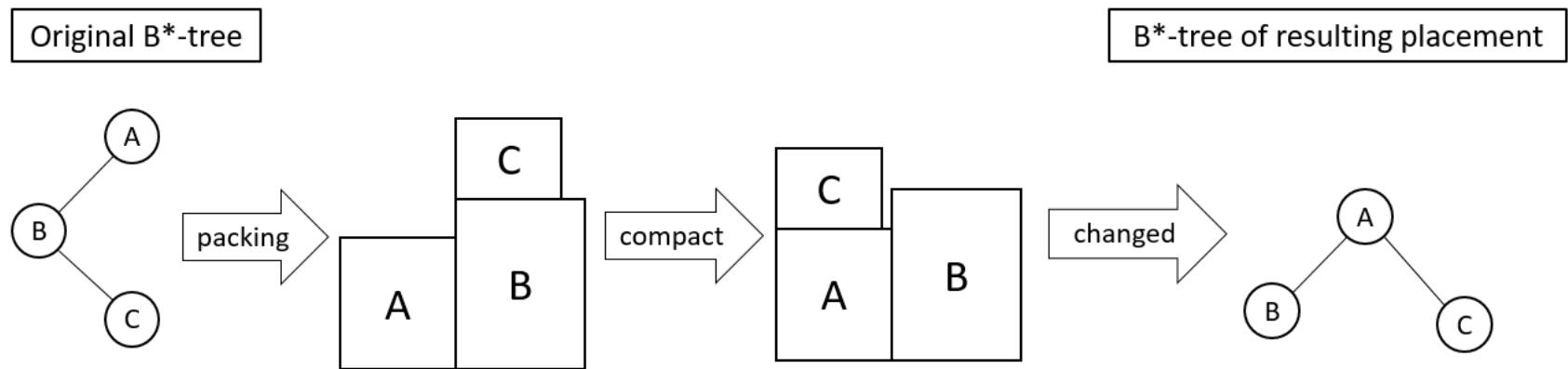
- Advantages

- Binary tree based, efficient and easy.
- Flexible to deal with hard, preplaced, soft, and rectilinear modules.
- Transformation between a tree and its placement takes only linear time (v.s. $O(n^2)$ or $O(n \lg \lg n)$ for sequence pair).
- Operate only on one B*-tree (v.s. 2 O-trees).
- Can evaluate area cost incrementally.
- Smaller solution space: only $O(n! 4^n / n^{1.5})$ combinations (v.s. $O((n!)^2)$ for sequence pair).
- Directly corresponds to multilevel framework for large-scale floorplan designs.

- Disadvantages

- Representation may change after packing.
- Less flexible than sequence pair in representation
 - Can represent only compacted placement.

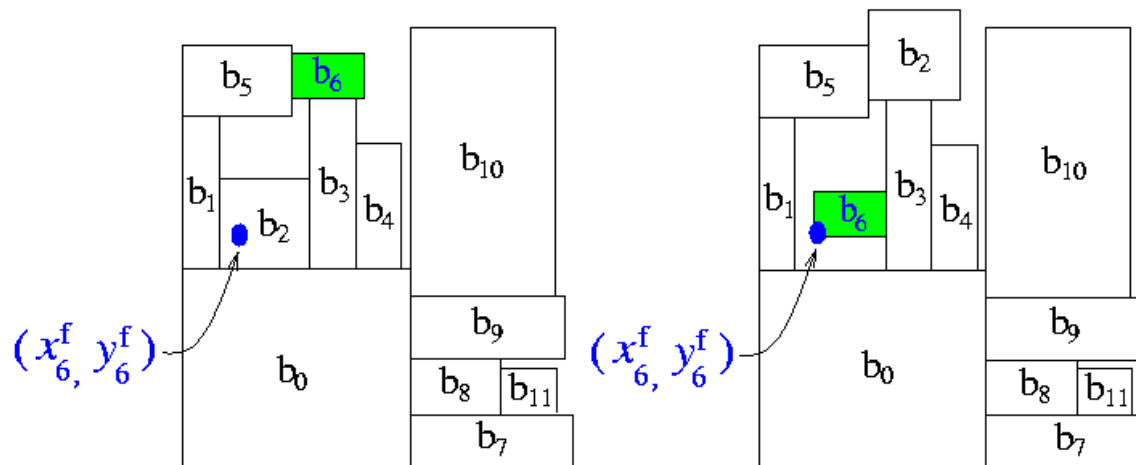
B*-Tree May Change after Packing



For compacted floorplan representations, the representation might change after packing.
The resulting placement might not correspond to the original B*-tree due to the compacting operation during packing.

Coping with Pre-placed Modules

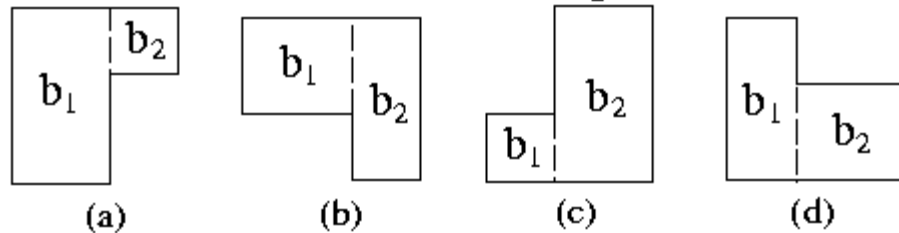
- If there are modules ahead or lower than b_i so that b_i cannot be placed at its fixed position (x_i^f, y_i^f) , exchange b_i with the module in $D_i = \{b_j \mid (x_j, y_j) \leq (x_i^f, y_i^f)\}$ that is closest to (x_i^f, y_i^f) .
- Incremental area cost update is possible.
 - E.g., the positions of $b_0, b_7, b_8, b_{11}, b_9, b_{10}$, and b_1 (before b_2 in the DFS order of T) remain unchanged after the exchange since they are in front of b_2 in the DFS order.



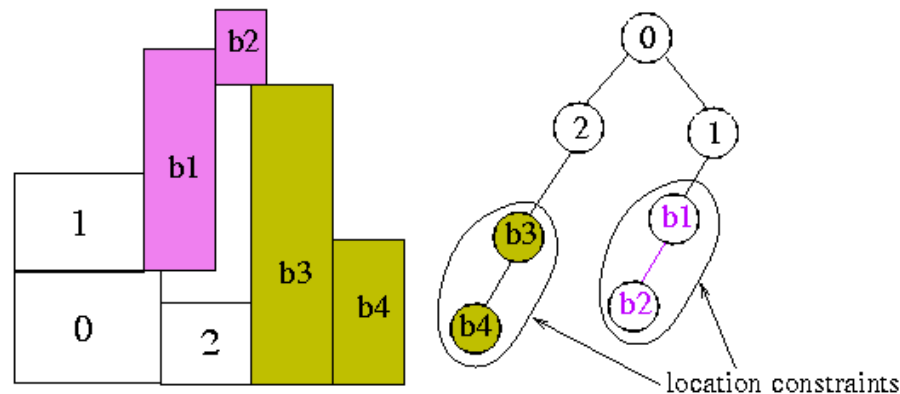
b_6 is a preplaced module

Coping with Rectilinear Modules

- Wu, Chang, Chang, “Rectilinear block placement using B*-trees,” ICCD-00
- Partition a rectilinear module into rectangular sub-modules.

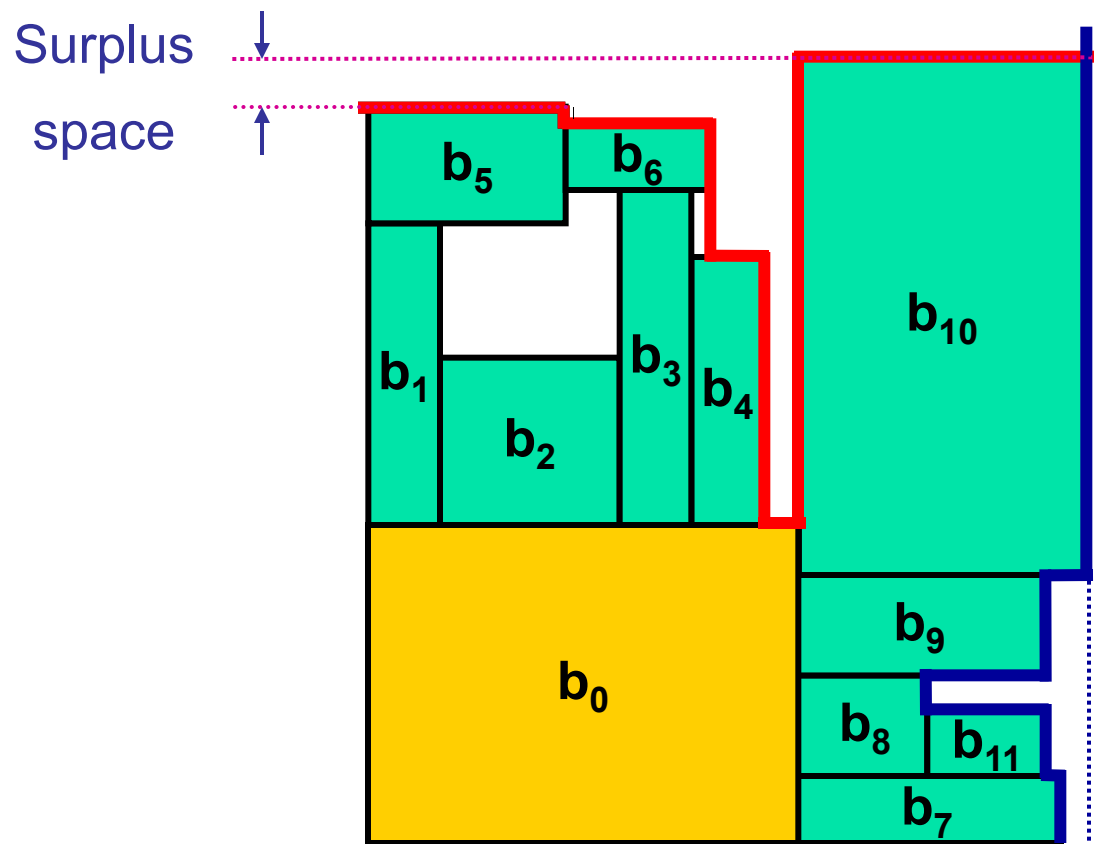


- Keep **location constraints** for the sub-modules.
 - E.g., Keep the right sub-module as the left child in the B*-tree.
- Align sub-modules, if necessary.
- Treat the sub-modules of a module as a whole during processing.



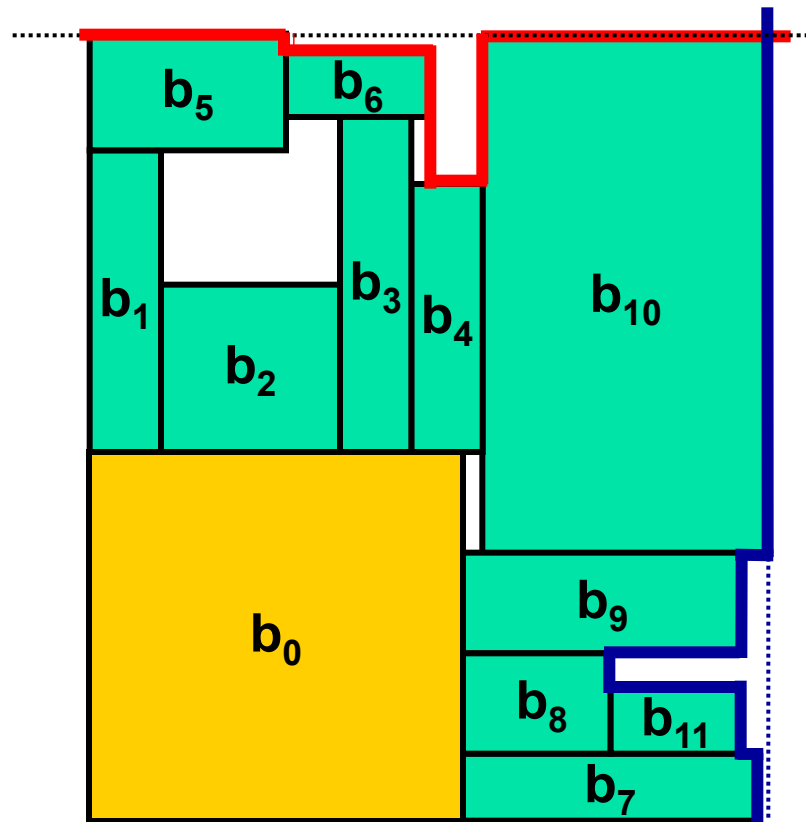
Coping with Soft Modules (1/2)

- Step1: Change the shape of the inserted soft module.
- Step2: Change the shapes of other soft modules.



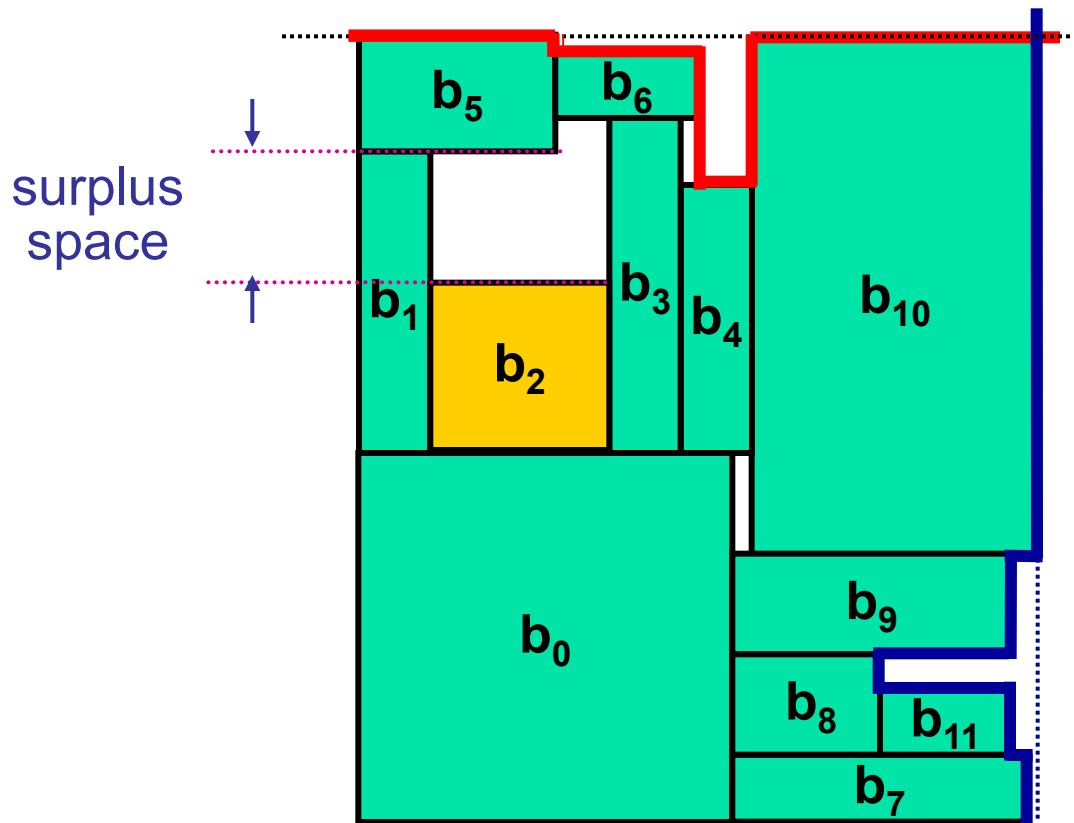
Coping with Soft Modules (1/2)

- Step1: Change the shape of the inserted soft module
- Step2: Change the shape of other soft modules



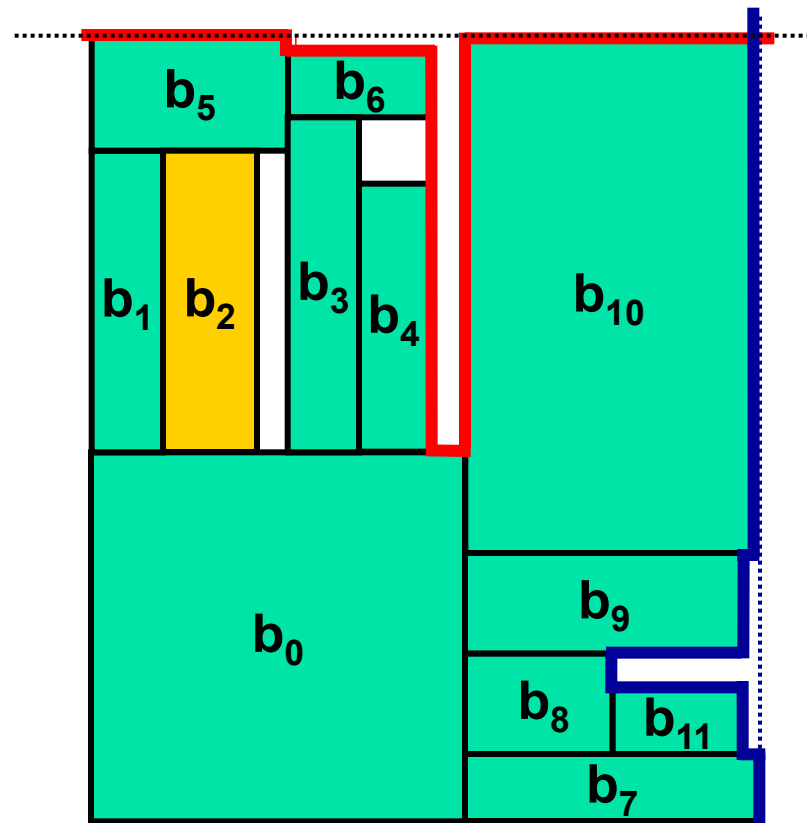
Coping with Soft Modules (2/2)

- Step1: Change the shape of the inserted soft module
- Step2: Change the shapes of other soft modules



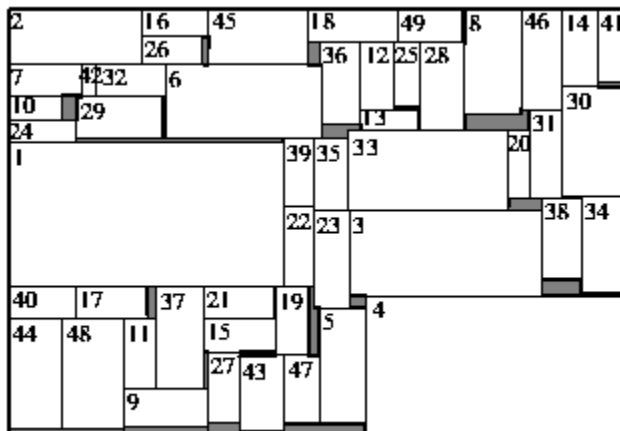
Coping with Soft Modules (2/2)

- Step1: Change the shape of the inserted soft module
- Step2: Change the shape of other soft modules

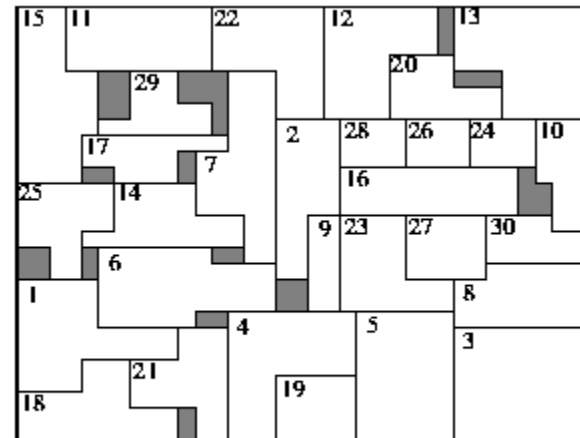


Perturbations & Solutions

- Perturbing B*-trees in simulated annealing
 - Op1: Rotate a module.
 - [Op2: Flip a module.]
 - Op3: Move a module to another place.
 - Op4: Swap two modules.
- ami49: Area = 36.74 mm^2 ; dead space = 3.53%; CPU time = 0.25 min on SUN Ultra 60 (optimum = 35.445 mm^2).



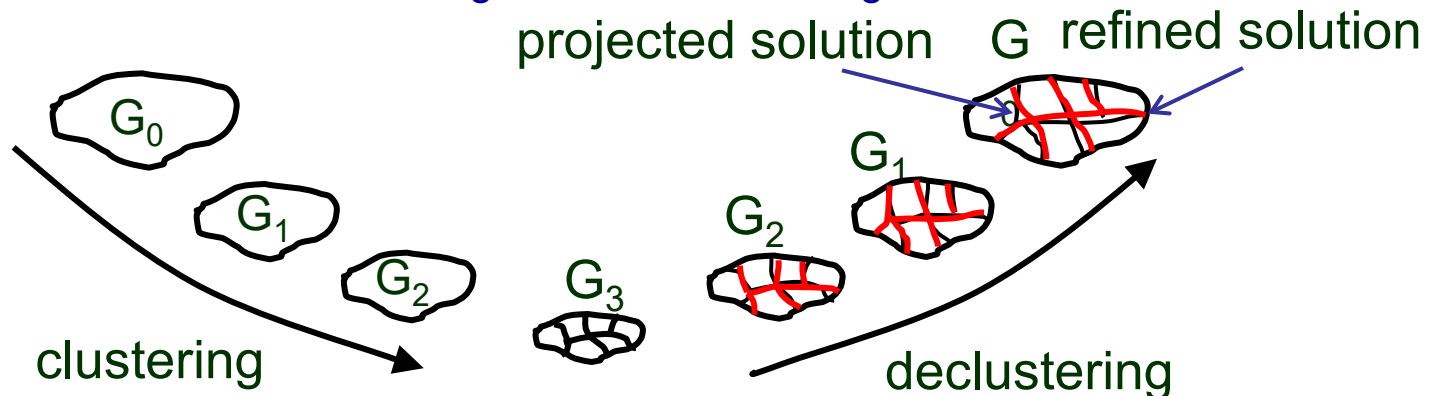
ami49



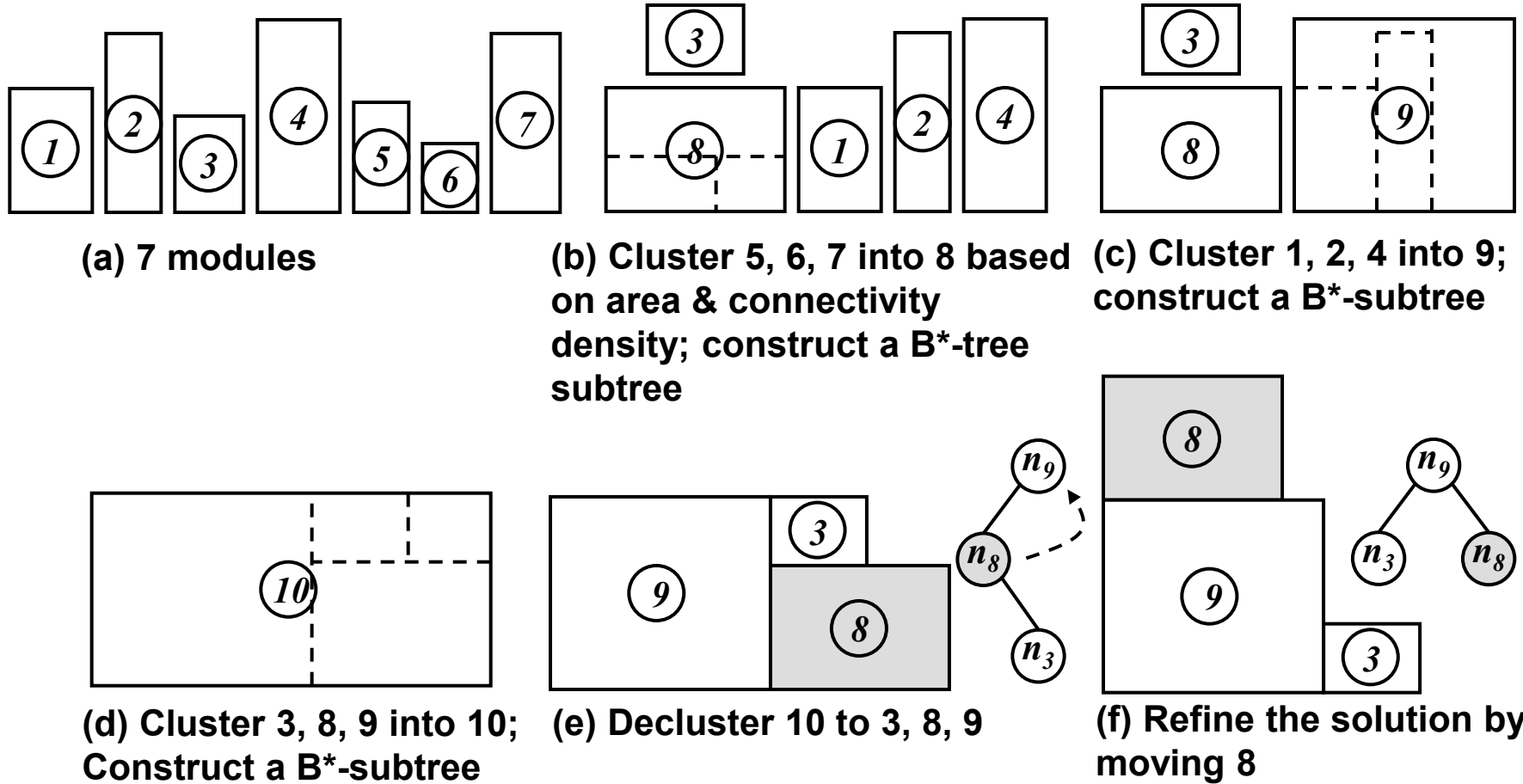
Rectangular, L-, and T-shaped modules

Multilevel B*-trees

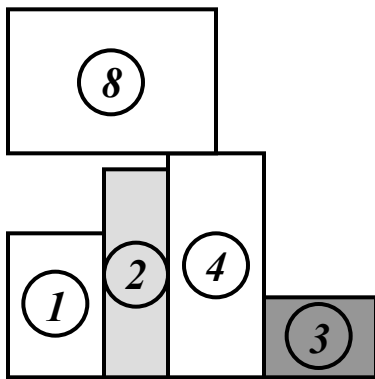
- Lee, Hsu, Chang, Yang, “Multilevel floorplanning/placement for large-scale modules using B*-trees,” DAC-2003.
- Two stages for MB*-tree: clustering followed by declustering.
- Clustering
 - Iteratively groups a set of modules based on area utilization and module connectivity.
 - Constructs a B*-tree to keep the geometric relations for the newly clustered modules.
- Declustering
 - Iteratively ungroups a set of the previously clustered modules (i.e., perform tree expansion)
 - Refines the solution using simulated annealing.



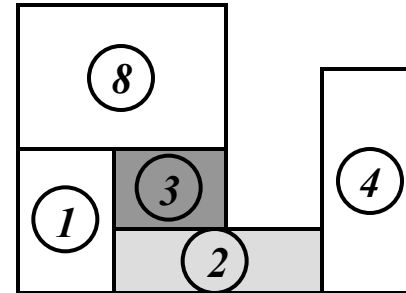
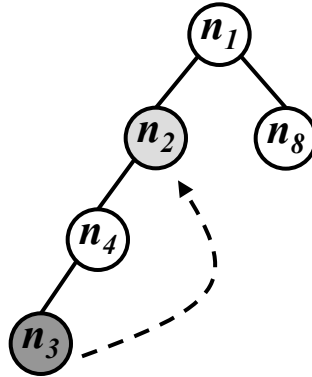
Multilevel B*-tree Example



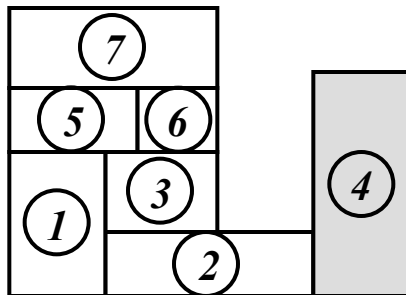
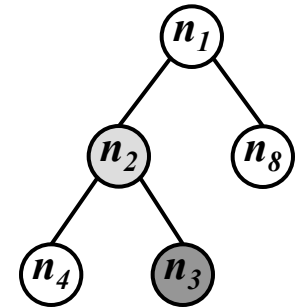
Multilevel B*-tree Example (cont'd)



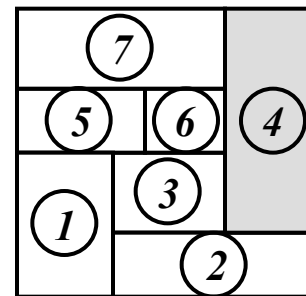
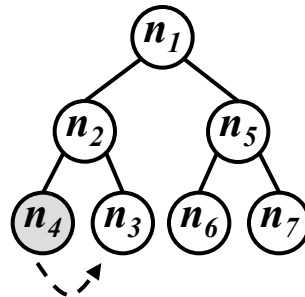
(g) Decluster 9 to 1, 2, 4



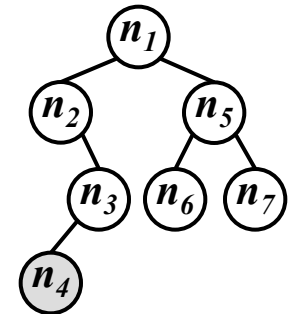
(h) Refine the solution by moving 2, 3



(i) Decluster 8 to 5, 6, 7

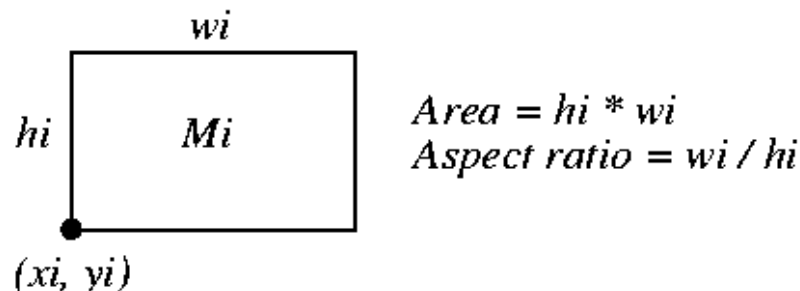


(j) Refine the solution by moving 4



Floorplanning by Mathematical Programming

- Sutanthavibul, Shragowitz, and Rosen, “An analytical approach to floorplan design and optimization,” 27th DAC, 1990.
- Notation:
 - w_i, h_i : width and height of module M_i .
 - (x_i, y_i) : coordinate of the lower left corner of module M_i .
 - $a_i \leq w_i/h_i \leq b_i$: aspect ratio w_i/h_i of module M_i . (Note: We defined aspect ratio as h_i/w_i before.)
- Goal: Find a mixed **integer linear programming (ILP)** formulation for the floorplan design.
 - **Linear** constraints? Objective function?



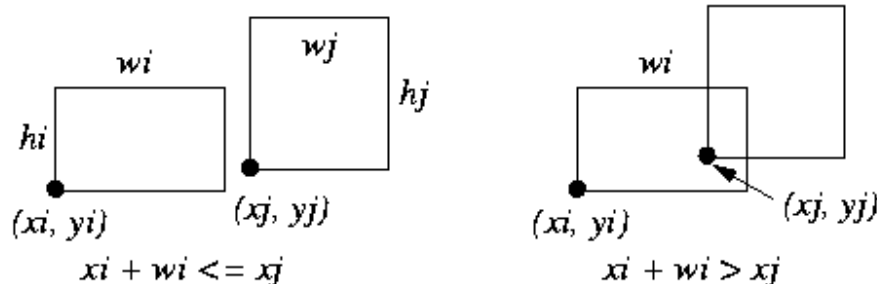
Nonoverlap Constraints

- Two modules M_i and M_j are nonoverlap, if at least one of the following linear constraints is satisfied (cases encoded by p_{ij} and q_{ij}):

		p_{ij}	q_{ij}
M_i to the left of M_j :	$x_i + w_i \leq x_j$	0	0
M_i below M_j :	$y_i + h_i \leq y_j$	0	1
M_i to the right of M_j :	$x_i - w_j \geq x_j$	1	0
M_i above M_j :	$y_i - h_j \geq y_j$	1	1

- Let W, H be upper bounds on the floorplan width and height, respectively.
- Introduce two 0, 1 variables p_{ij} and q_{ij} to denote that one of the above inequalities is enforced; e.g., $p_{ij} = 0, q_{ij} = 1 \Rightarrow y_i + h_i \leq y_j$ is satisfied

$$\begin{aligned}
 x_i + w_i &\leq x_j + W(p_{ij} + q_{ij}) \\
 y_i + h_i &\leq y_j + H(1 + p_{ij} - q_{ij}) \\
 x_i - w_j &\geq x_j - W(1 - p_{ij} + q_{ij}) \\
 y_i - h_j &\geq y_j - H(2 - p_{ij} - q_{ij})
 \end{aligned}$$



Cost Function & Constraints

- Minimize $Area = xy$, **nonlinear!** (x, y : width and height of the resulting floorplan)
- How to fix?
 - Fix the width W and minimize the height y !
- Four types of constraints:
 1. no two modules overlap ($\forall i, j: 1 \leq i < j \leq n$);
 2. each module is enclosed within a rectangle of width W and height H ($x_i + w_i \leq W, y_i + h_i \leq H, 1 \leq i \leq n$);
 3. $x_i \geq 0, y_i \geq 0, 1 \leq i \leq n$;
 4. $p_{ij}, q_{ij} \in \{0, 1\}$.
- w_i, h_i are known.

Mixed ILP for Floorplanning

Mixed ILP for the floorplanning problem with rigid, fixed modules.

$$\begin{array}{ll}
 \min & y \\
 \text{subject to} & \\
 & x_i + w_i \leq W, \quad 1 \leq i \leq n \quad (1) \\
 & y_i + h_i \leq y, \quad 1 \leq i \leq n \quad (2) \\
 & x_i + w_i \leq x_j + W(p_{ij} + q_{ij}), \quad 1 \leq i < j \leq n \quad (3) \\
 & y_i + h_i \leq y_j + H(1 + p_{ij} - q_{ij}), \quad 1 \leq i < j \leq n \quad (4) \\
 & x_i - w_j \geq x_j - W(1 - p_{ij} + q_{ij}), \quad 1 \leq i < j \leq n \quad (5) \\
 & y_i - h_j \geq y_j - H(2 - p_{ij} - q_{ij}), \quad 1 \leq i < j \leq n \quad (6) \\
 & x_i, y_i \geq 0, \quad 1 \leq i \leq n \quad (7) \\
 & p_{ij}, q_{ij} \in \{0, 1\}, \quad 1 \leq i < j \leq n \quad (8)
 \end{array}$$

- Size of the mixed ILP: for n modules,
 - # continuous variables: $O(n)$; # integer variables: $O(n^2)$; # linear constraints: $O(n^2)$.
 - Unacceptably huge program for a large n ! (How to cope with it?)
- Popular LP software: LINDO, Ip_solve, etc.

Mixed ILP for Floorplanning (cont)

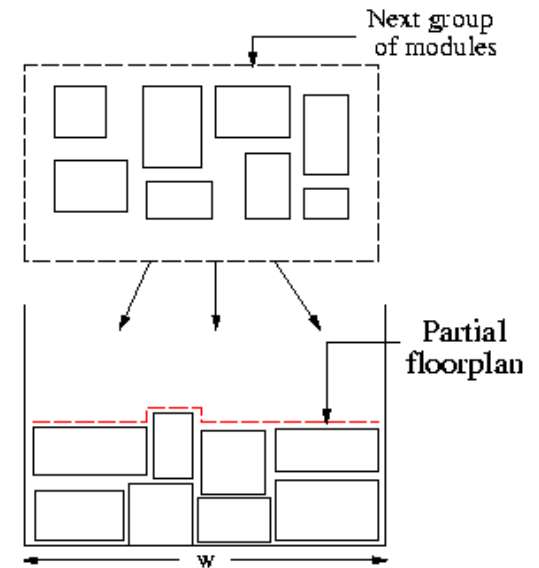
Mixed ILP for the floorplanning problem: rigid, freely oriented modules.

$$\begin{aligned}
 & \min \quad y \\
 & \text{subject to} \\
 & \quad x_i + r_i h_i + (1 - r_i) w_i \leq W, \quad 1 \leq i \leq n \quad (9) \\
 & \quad y_i + r_i w_i + (1 - r_i) h_i \leq y, \quad 1 \leq i \leq n \quad (10) \\
 & \quad x_i + r_i h_i + (1 - r_i) w_i \leq x_j + M(p_{ij} + q_{ij}), \quad 1 \leq i < j \leq n \quad (11) \\
 & \quad y_i + r_i w_i - (1 - r_i) h_i \leq y_j + M(1 + p_{ij} - q_{ij}), \quad 1 \leq i < j \leq n \quad (12) \\
 & \quad x_i - r_j h_j + (1 - r_j) w_j \geq x_j - M(1 - p_{ij} + q_{ij}), \quad 1 \leq i < j \leq n \quad (13) \\
 & \quad y_i - r_j w_j - (1 - r_j) h_j \geq y_j - M(2 - p_{ij} - q_{ij}), \quad 1 \leq i < j \leq n \quad (14) \\
 & \quad x_i, y_i \geq 0, \quad 1 \leq i \leq n \quad (15) \\
 & \quad p_{ij}, q_{ij} \in \{0, 1\}, \quad 1 \leq i < j \leq n \quad (16)
 \end{aligned}$$

- For each module i with free orientation, associate a 0-1 variable r_i :
 - $r_i = 0$: 0° rotation for module i .
 - $r_i = 1$: 90° rotation for module i .
- $M = \max\{W, H\}$.

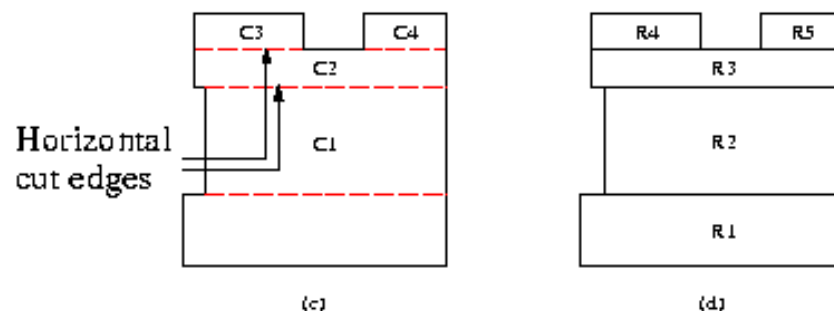
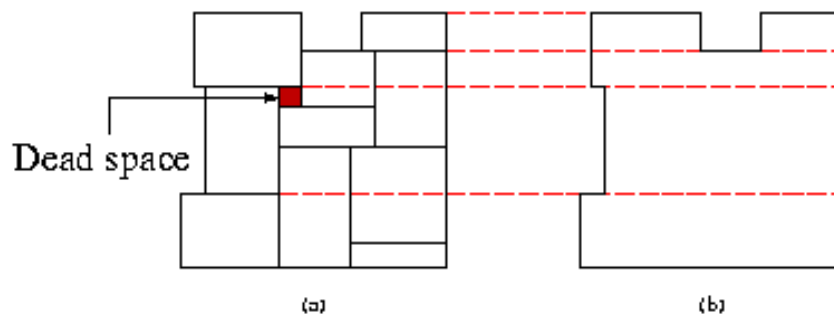
Reducing the Size of the Mixed ILP

- Time complexity of a mixed ILP: exponential!
- Recall the large size of the mixed ILP: # variables, # constraints: $O(n^2)$.
 - How to fix it?
- Key: Solve a partial problem at each step
 - successive augmentation
 - Classic cluster-growth greedy approach
 - Repeatedly select subsets of modules and formulate corresponding linear programs, along with additional constraints from previously selected modules
- Questions:
 - How to select next subgroup of modules? ☐ linear ordering based on connectivity. (cluster growth)
 - How to minimize the # of required variables?



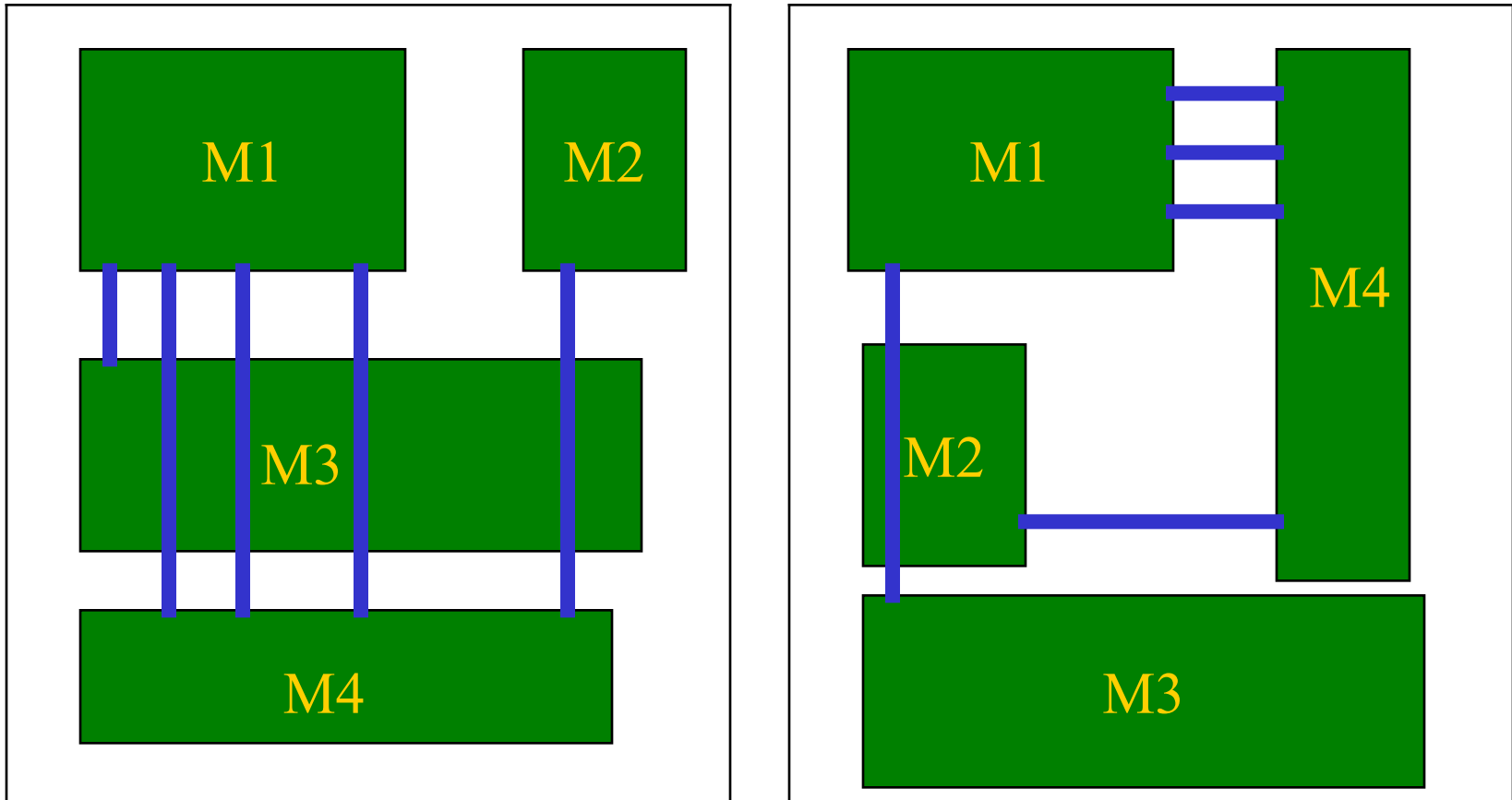
Reducing the Size of the Mixed ILP (cont)

- Size of each successive mixed ILP depends on (1) # of modules in the next group; (2) “size” of the partially constructed floorplan.
- Keys to deal with (2)
 - Minimize the problem size of the partial floorplan.
 - Replace the already placed modules by a set of covering rectangles.
 - # rectangles is usually much smaller than # placed modules.



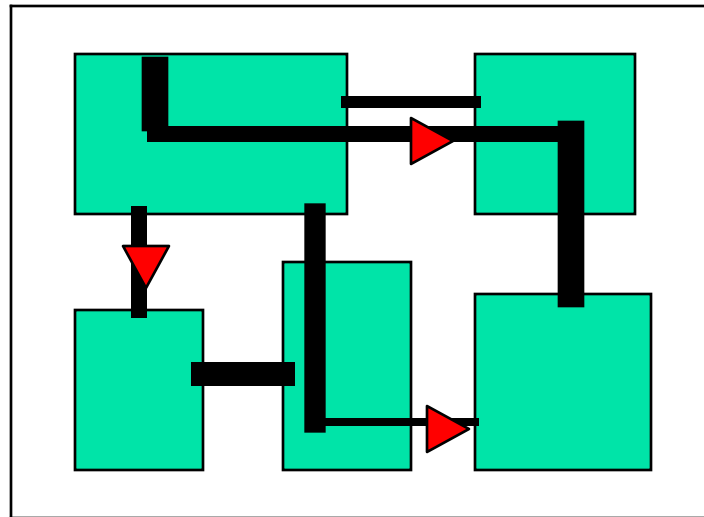
Interconnect-Centric Floorplanning

- Floorplanning greatly influences interconnect structure

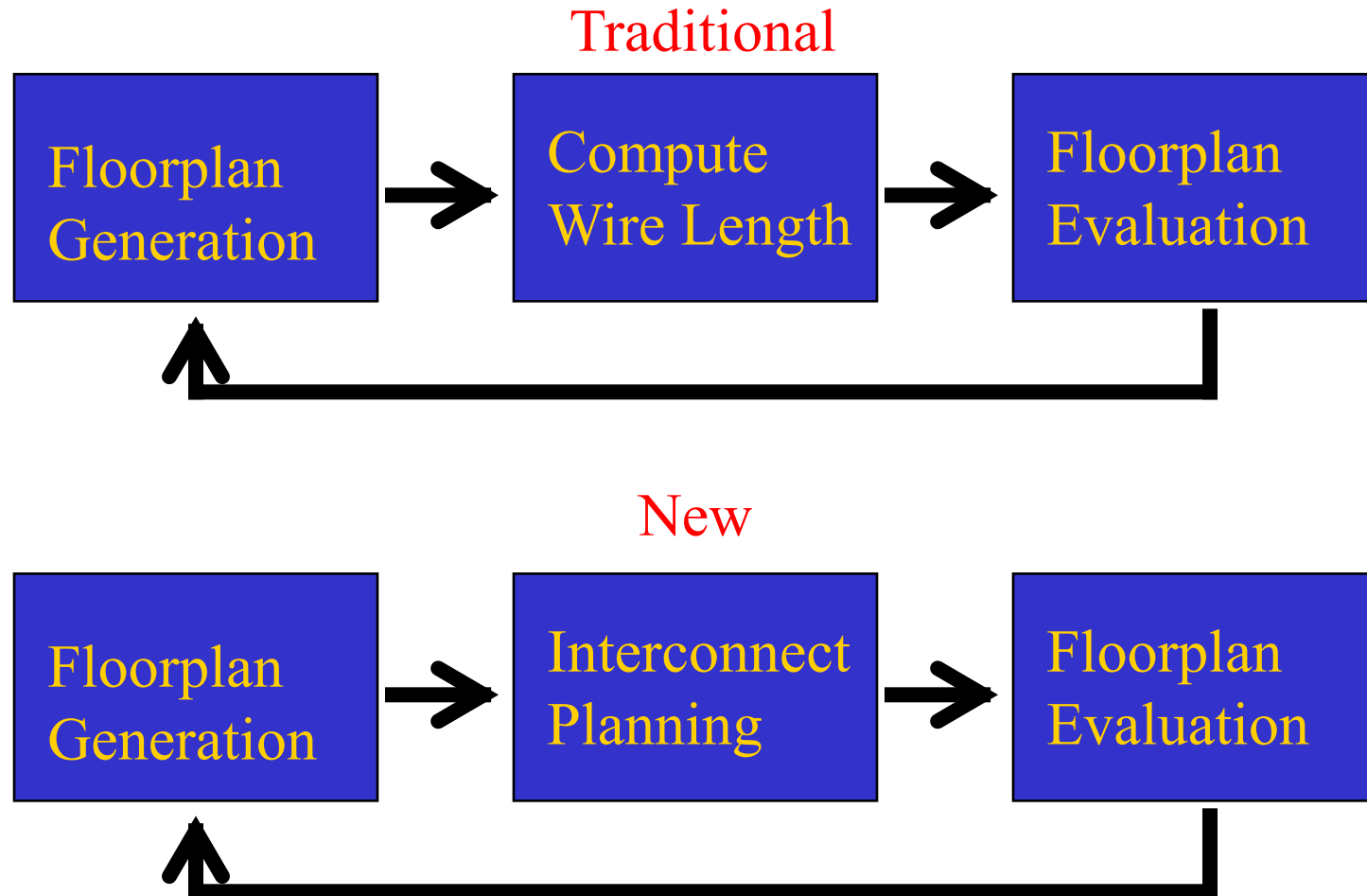


Interconnect Planning

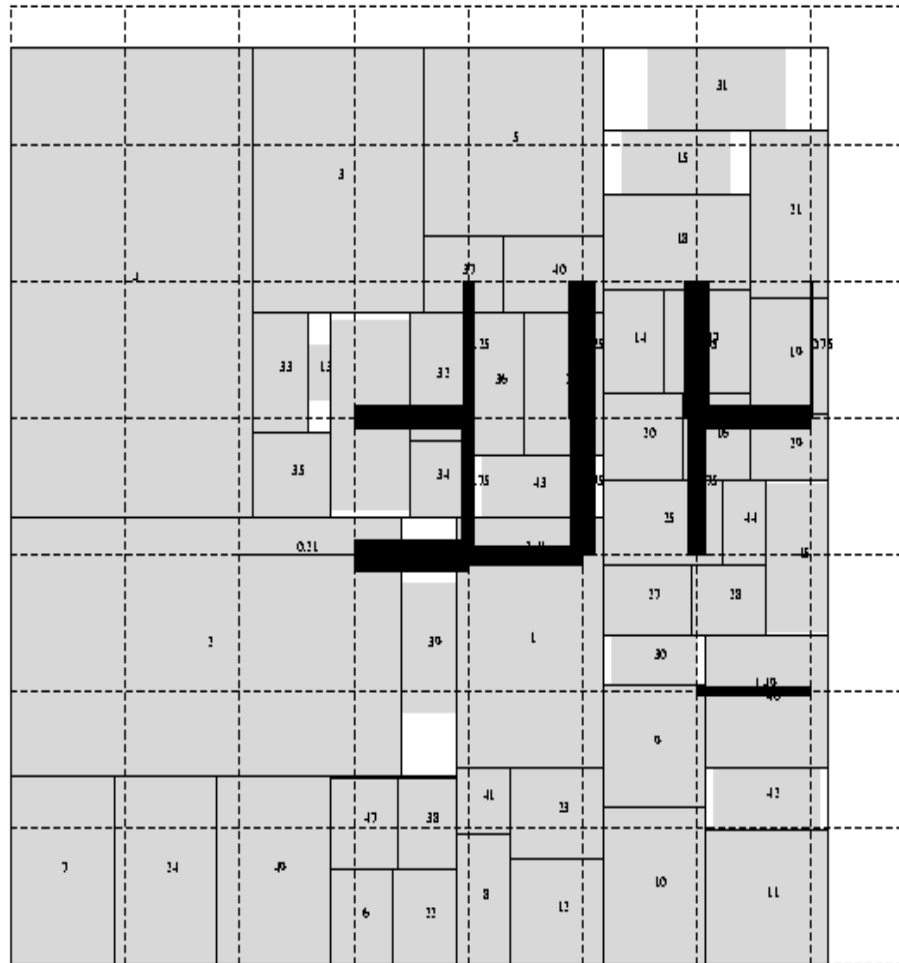
- Pin assignment and routing of global interconnects
- Buffer insertion and sizing
 - Buffer block planning
- Wire sizing



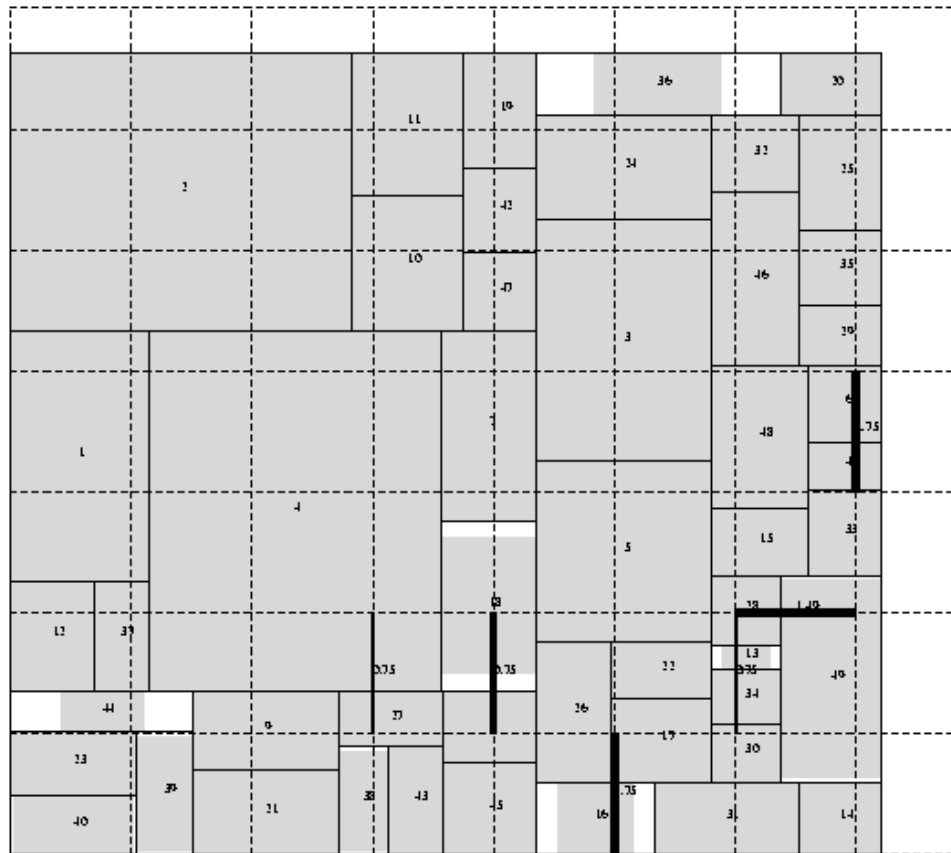
Floorplanning and Interconnect Planning



Interconnect-Centric Floorplanning: ami49 (1/2)

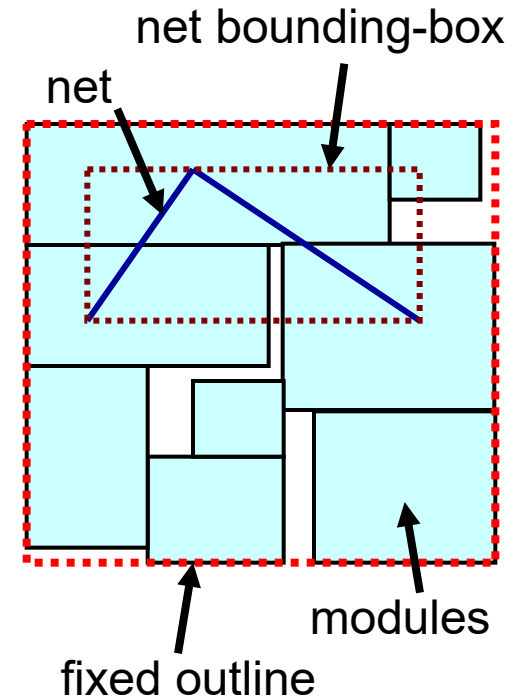


Interconnect-Centric Floorplanning: ami49 (2/2)



Fixed-Outline Floorplanning

- ❑ Input
 - Modules, netlist, **fixed outline**
- ❑ Output
 - Module positions, orientations
- ❑ Objectives
 - **Minimize the half-perimeter wirelength (HPWL)**
 - All modules are within the fixed die (fixed-outline constraint) and no overlaps occur between modules



Fixed-Outline Constraint

- ❑ Fixed-outline floorplanning is more prevailing in modern VLSI design
- ❑ Given the *maximum white-space fraction* Γ and *desired aspect ratio* R^* , the outline is defined by

$$H^* = \sqrt{(1+\Gamma)AR^*} \quad W^* = \sqrt{(1+\Gamma)A/R^*}$$

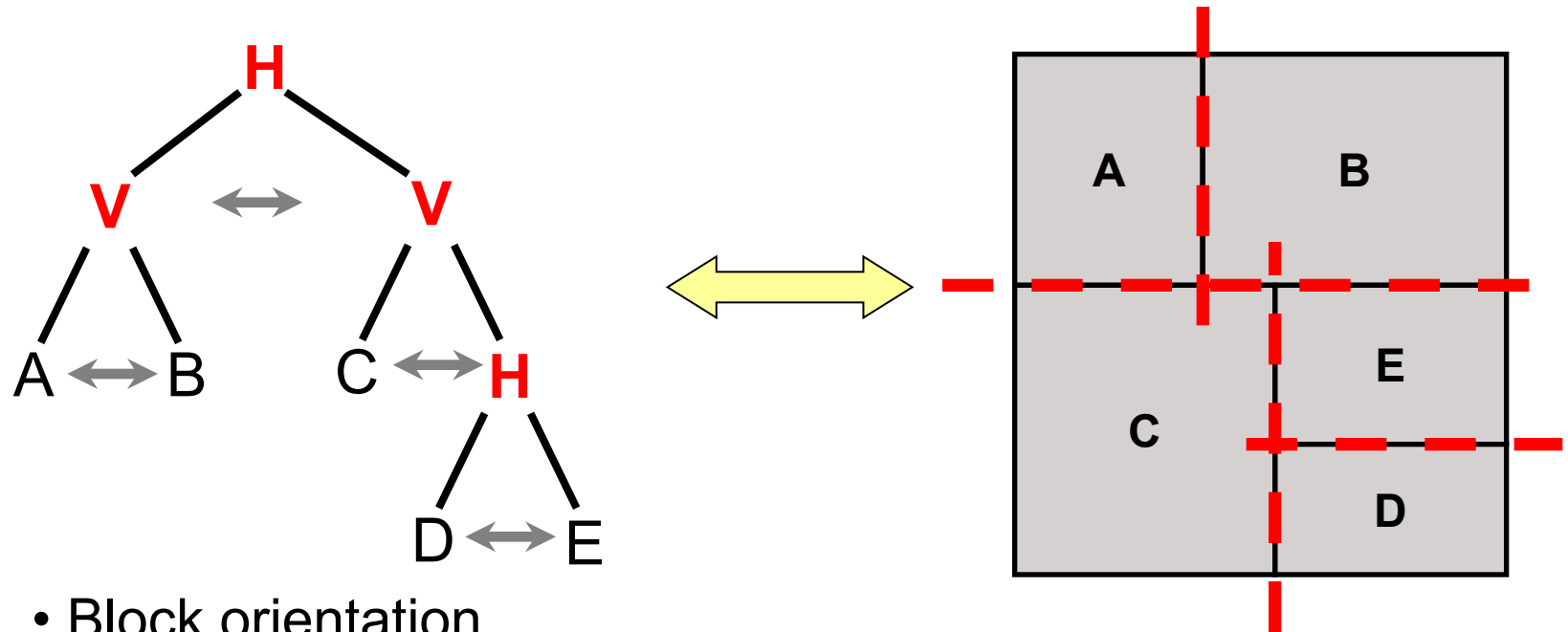
- $R^* = H^*/W^*, H^*W^* = (1+\Gamma)A$

- ❑ Cost for floorplan F

$$\Phi(F) = \alpha A + \beta L + (1 - \alpha - \beta)(R^* - R)^2$$

A	Block area
L	Wirelength
R^*	Fixed-outline aspect ratio
R	Current floorplan aspect ratio

Defer: Fixed-Outline Slicing Floorplan



- Block orientation
- Slice line direction (H/V)
- Left-right or top-bottom relative order

J. Z. Yan, and C. Chu, "DeFer: Deferred Decision Making Enabled Fixed-Outline Floorplanning Algorithm," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 367–381, 2010.

Optimal **S**lack-Driven Block **S**haping Algorithm in Fixed-Outline Floorplanning

□ Input

ISPD 2012 Best Paper Award
(by J. Z. Yan and C. Chu)

■ n Blocks

□ Area A_i for block i

□ Width bounds W_i^{\min} and W_i^{\max} for block i

□ Height bounds H_i^{\min} and H_i^{\max} for block i

■ Constraint graphs G_h and G_v

■ Fixed-outline region

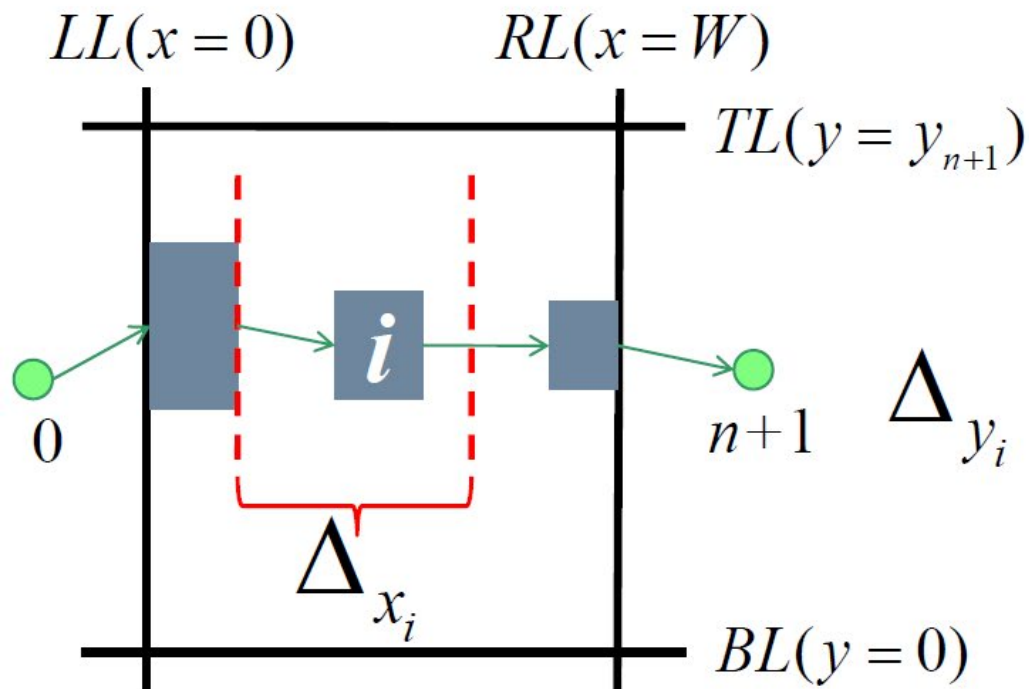
□ Output

■ Block coordinates (x_i, y_i) , width w_i and height h_i

□ All blocks inside fixed-outline region

□ All blocks without overlaps

Optimal **S**lack-Driven Block **S**haping Algorithm in Fixed-Outline Floorplanning



- G_h, G_v
- Shape of n blocks

horizontal slack

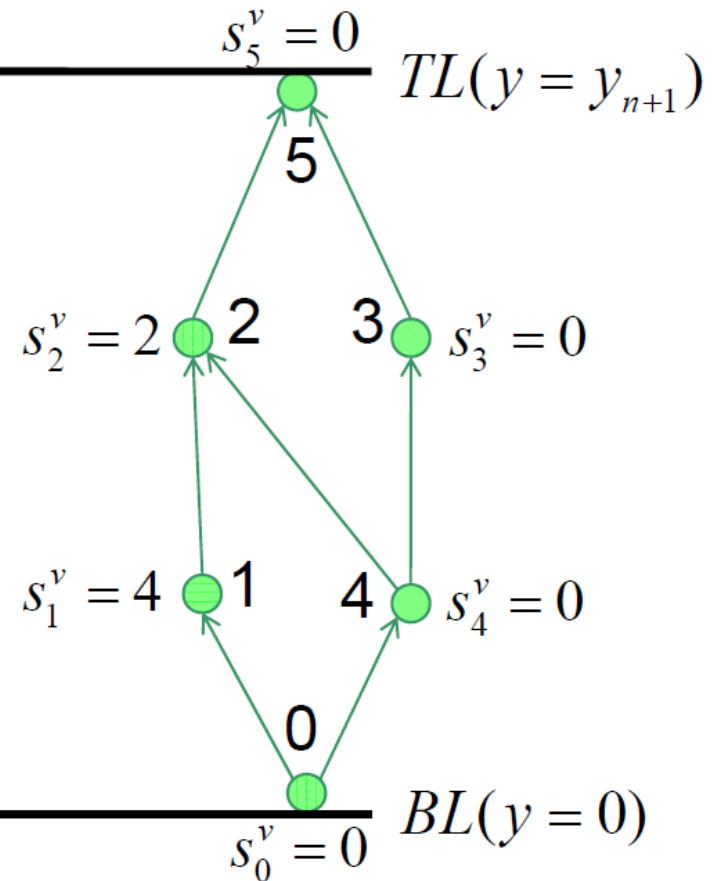
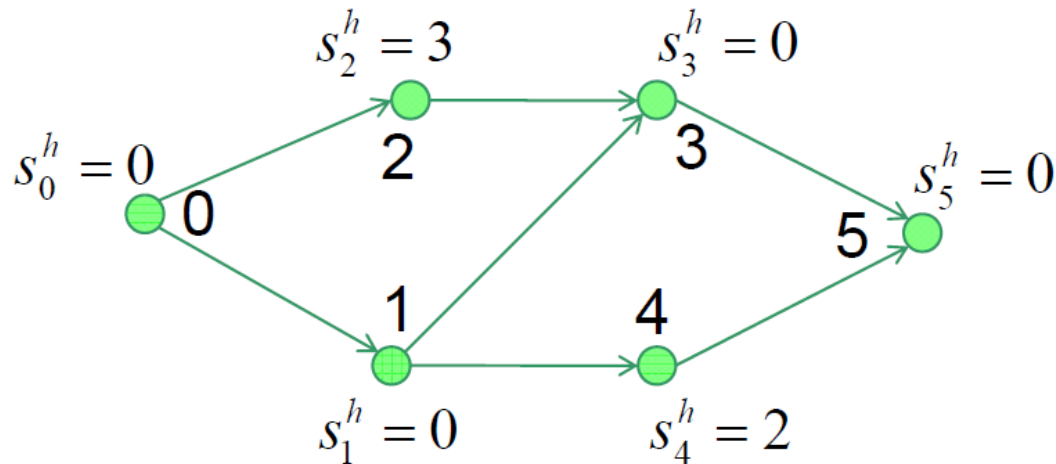
$$s_i^h = \max(0, \Delta_{x_i})$$

vertical slack

$$s_i^v = \max(0, \Delta_{y_i})$$

Optimal **S**lack-Driven Block **S**haping Algorithm in Fixed-Outline Floorplanning

- Horizontal Critical Path (**HCP**)
- Vertical Critical Path (**VCP**)



Length of VCP = Layout height y_{n+1}

Basic Slack-Driven Shaping

- ❑ Soft blocks are shaped *iteratively*.
- ❑ At each iteration, apply two operations:



- Globally distribute the total amount of slack to the individual soft block.
- ❑ Algorithm stops when there is no identified soft block to shape.
- ❑ Layout height is monotonically reducing, and layout width is bouncing, but always within the upper bound.

Summary: Floorplanning (1/3)

- Floorplanning objectives: (1) minimize area, (2) meet timing constraints, (3) maximize routability (minimize congestion), ((4) determine shapes of soft modules)
- Existing representations
 - **Slicing**: slicing tree (DAC-82), normalized Polished expression (DAC-86)
 - **Mosaic**: CBL (ICCAD-2k), Q-Sequence (AP-CAS-2k, DATE-02), Twin binary tree (ISPD-01)
 - **Compacted**: O-tree (DAC-99), B*-tree (DAC-2k), MB*-tree (DAC-03), CS (TVLSI, 2003)
 - **General**: SP (ICCAD-95), BSG (ICCAD-96), TCG (DAC-01), TCG-S (DAC-02).
- P*-admissible representations: all representations for general floorplans.
- P-admissible, non-P*-admissible representations (for area): all for compacted floorplans.
- What makes a good representation?
 - Easy, effective, efficient, flexible, stable

Summary: Floorplanning (2/3)

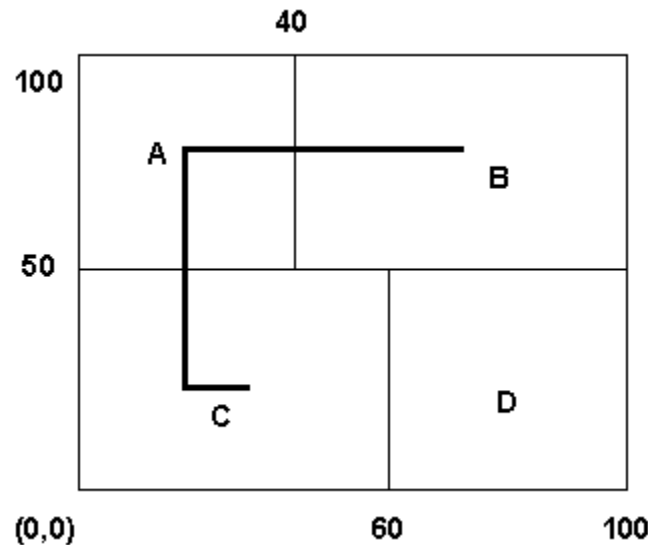
- Other issues
 - **Soft module:** shape curve (NPE, DAC-86), (Integer) linear programming (DAC-90, DAC-2k), stretching range (B*-tree, DAC-2k), Lagrangian relaxation (SP, ISPD-2k)
 - **Preplaced module:** ASPDAC-98 (BSG), ASPDAC-01 (SP), DAC-2K (B*-tree), ISCAS-01 (B*-tree), DAC-02 (TCG-S)
 - **Symmetry module:** DAC-99 (SP), ICCAD-02 (B*-tree)
 - **Rectilinear module:** TCAD-2K (SP), ICCAD-98 (SP), ISPD-98 (SP), ISPD-01 (SP), DATE-02 (TCG), TVLSI-02 (TCG), ICCD-2K (B*-tree), ACM TODAES-03 (B*-tree), ISPD-01 (O-tree)
 - **Range constraint:** ISPD-99 (NPE), ASPDAC-01 (SP), DAC-02 (TCG-S)
 - **Boundary constraint:** ASPDAC-01 (SP), DAC-02 (TCG-S), IEE Proc.-02 (B*-tree)
- Since each representation has its pros and cons, so maybe we can
 - Integrate two or more representations to get a better one (e.g., TCG-S, DAC-02)
 - Apply different representations at different stages
- Large-scale module floorplanning/placement (MB*-tree, DAC-03)

Summary: Floorplanning (3/3)

- Performance-driven floorplanning
 - Buffer planning (ICCAD-99, ISPD-2K, DAC-01, ASPDAC-03)
 - Wire planning (ICCAD-99)
 - Power supply planning (ASPDAC-01)
 - Power supply noise-aware floorplanning (ASPDAC-03)
- Fixed-outline floorplanning

2003 MOE IC/CAD Contest: Problem 1

- Chip Floorplanning with Hard/Soft Macros



Input files :

[problem1.mac]:

.chip_bbox (100,100)

.macro A 2000 0.6 1.5

.macro B 3000 0.8 1.2

.macro C 3000 0.8 1.5

.macro D 2000 0.8 0.8 // hard macro

+

[problem1.spc]:

.net N1 A B C

+

Output files :

[problem1.rpt]

.macro A (0, 50) (40, 100)

.macro B (40, 50) (100, 100)

.macro C (0, 0) (60, 50)

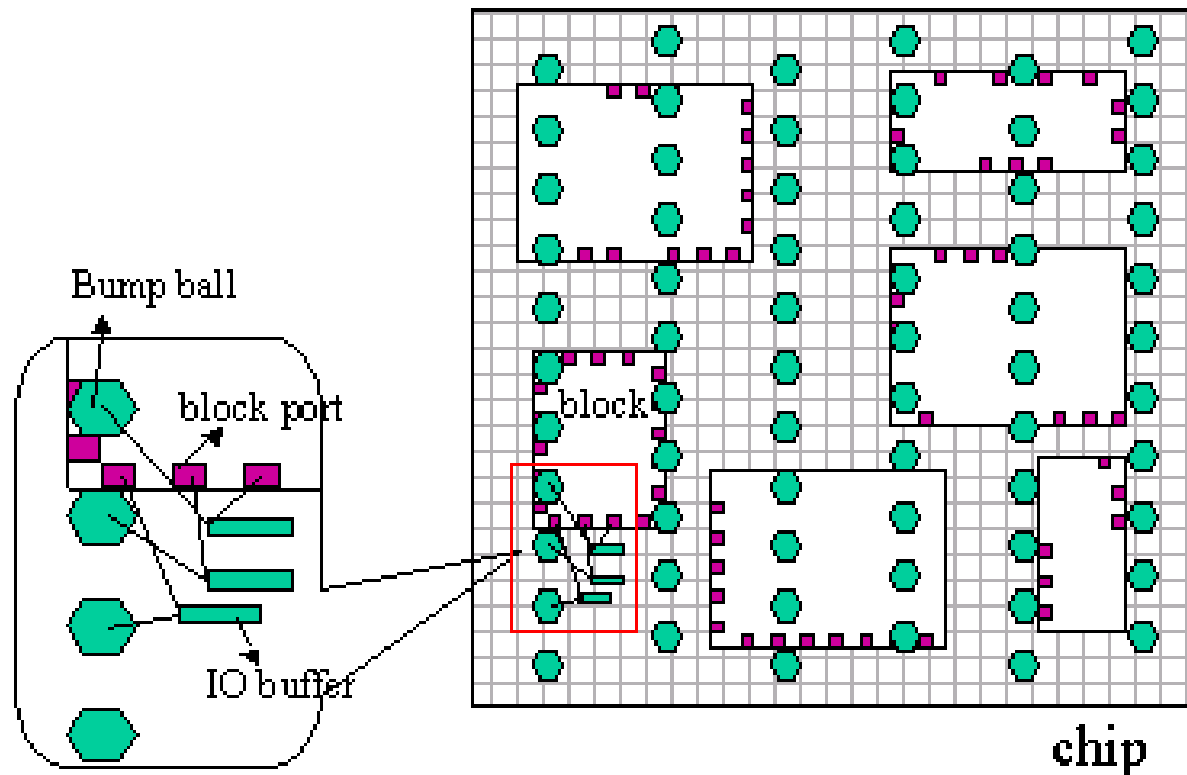
.macro D (60, 0) (100, 50)

.mst 110

.area 10000

2003 MOE IC/CAD Contest: Problem 3

- **Block and Input/Output Buffer Placement for Skew/Delay Minimization in Flip-chip Design**



2005 MOE IC/CAD Contest Problem 5

- Chip placement for MPW (Multiple Project Wafer)
 - Manufacturing cost minimization in shuttle mask sharing in getting certain amount of prototyping chips
- Needs to decide the floorplan of reticle(s) and cut lines for wafer(s) in order to get less cost
 - Also needs to consider manufacturing technology issue (#metal layers)
- Needs some algorithmic aspects and geometrical thinking
- References:
 - A.B. Kahng et.al., “Multi-Project Reticle Floorplanning and Wafer Dicing”, ISPD 2004
 - Report from previous generation problem (online soon)

Illustrations for MPW

