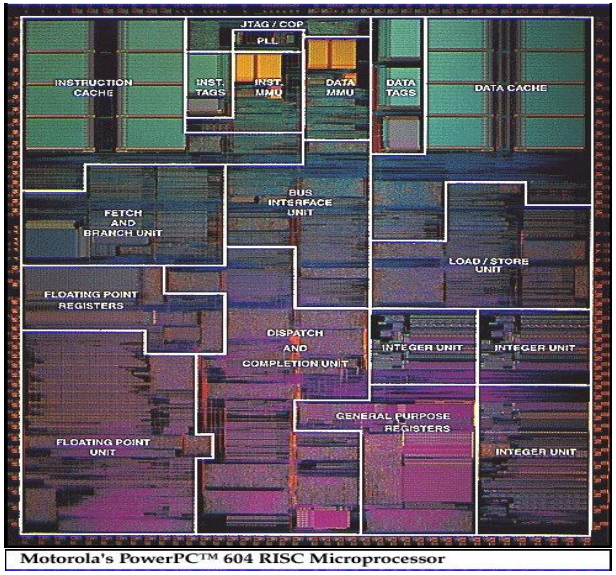
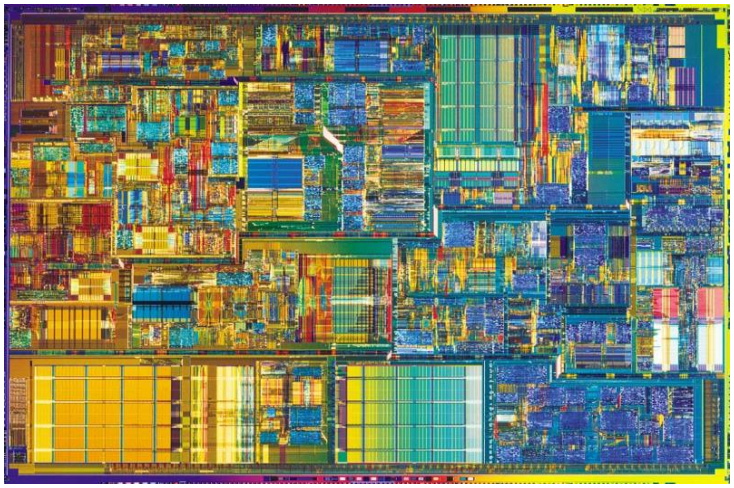
Unit 5B: Floorplanning

- Course contents
  - Floorplan basics
  - Normalized Polish expression for **slicing** floorplans
  - B*-trees for **non-slicing** floorplans
- Readings
  - Chapters 8 and 5.6



PowerPC 604

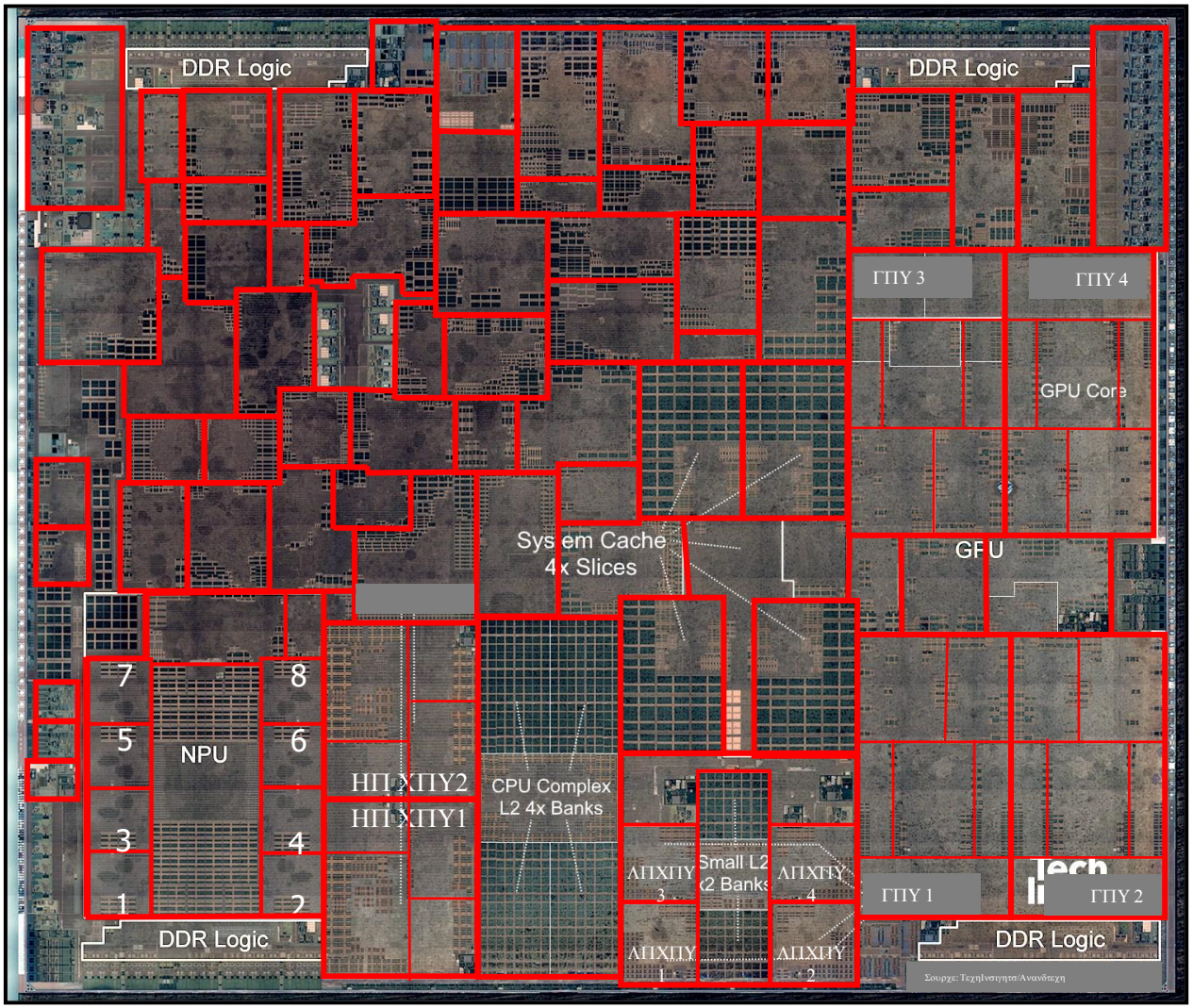


Pentium 4

Apple A12
iPhone Xs (2018)

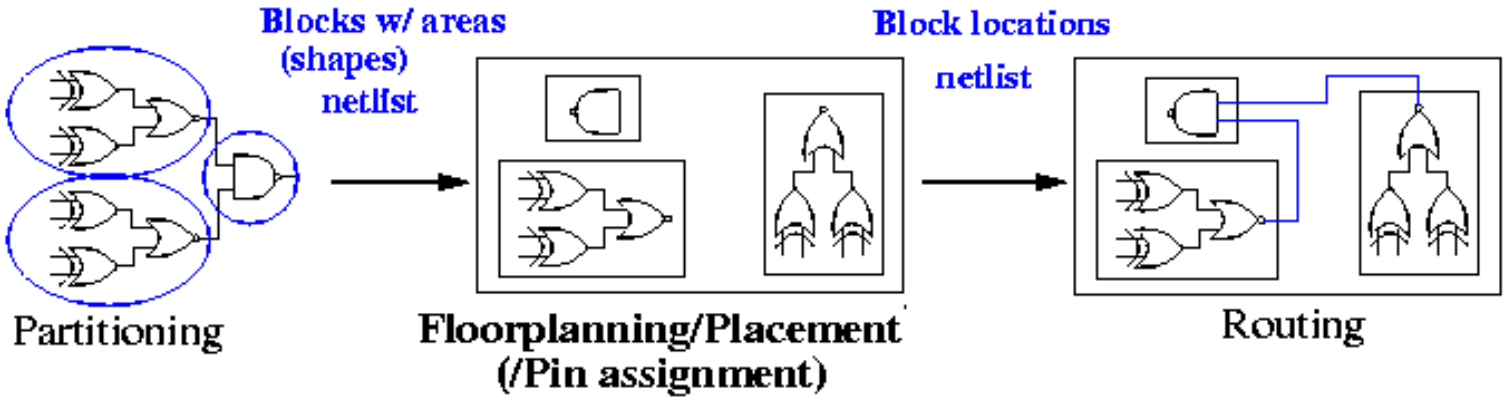
EDA plays a critical role!

- 7nm TSMC FinFet
- 9.89 x 8.42 = 83.27 mm2
- 6.9 Billion transistors
- 4 **GPU** cores (~18%): 9 Blocks
- 6 **CPU** cores (~14%): 13 Blocks
  - 4 'Tempest' Low Power CPU cores
  - 2 'Vortex' High Performance CPU cores
  - L2 & L3 caches
- 8 **NPU/TPU** cores (~7%) 4 Blocks
- DDR (~3%) 1 block
- Misc (~57%): 50 Unique Blocks
- Total: ~75 unique blocks



Floorplanning

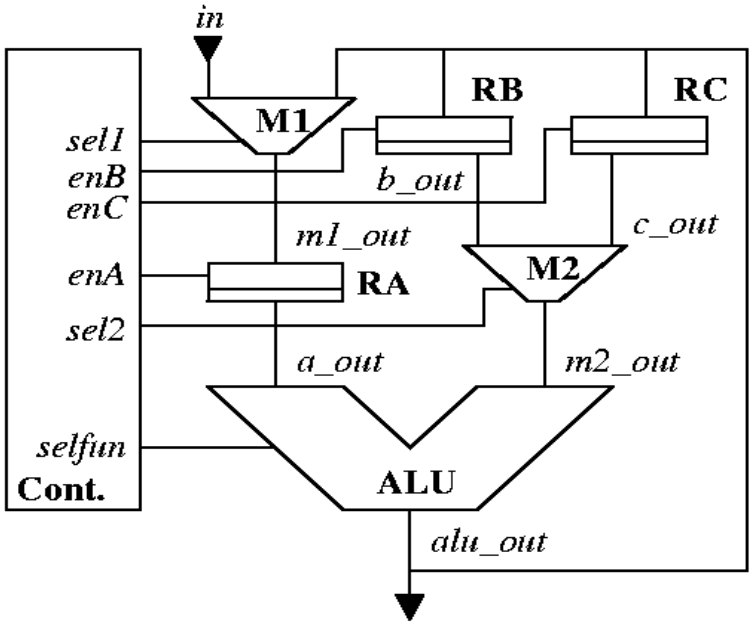
- Partitioning leads to
  - Blocks with well-defined **areas and shapes** (rigid/hard blocks).
  - Blocks with approximate areas and no particular shapes (flexible/soft blocks).
  - A **netlist** specifying connections between the blocks.
- Objectives
  - Find **locations** for all blocks.
  - Consider shapes of soft block and pin locations of all the blocks.

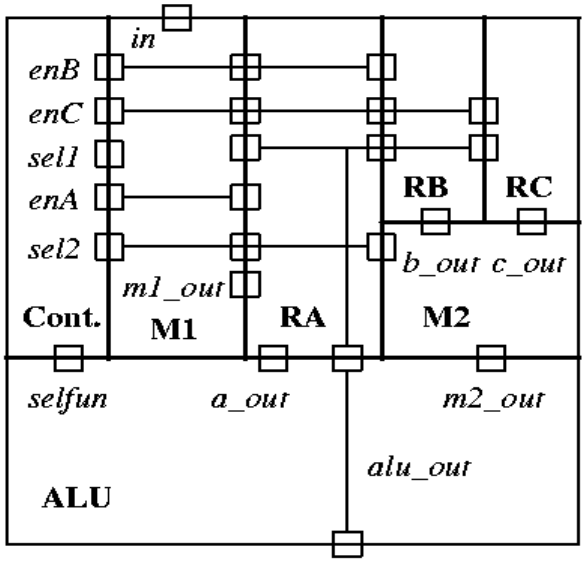


Very Often

- Floorplanning & placement are interchangeable
- If really want to distinguish them, floorplanning is more about blocks/macros (including soft ones)
- Placement means to deal with millions of standard cells
  - Global placement, detailed placement

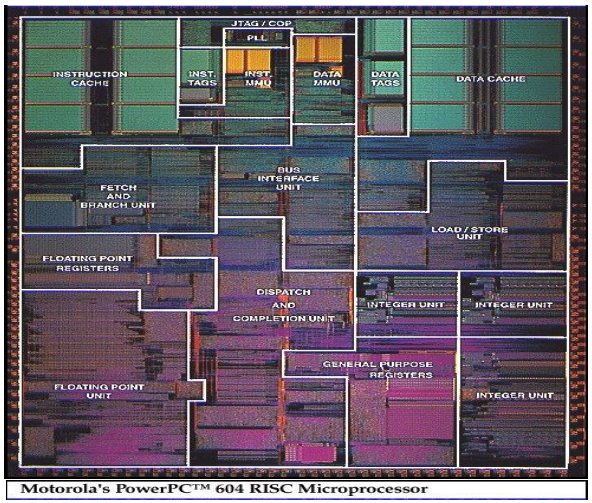
Some Early Layout Decision Examples



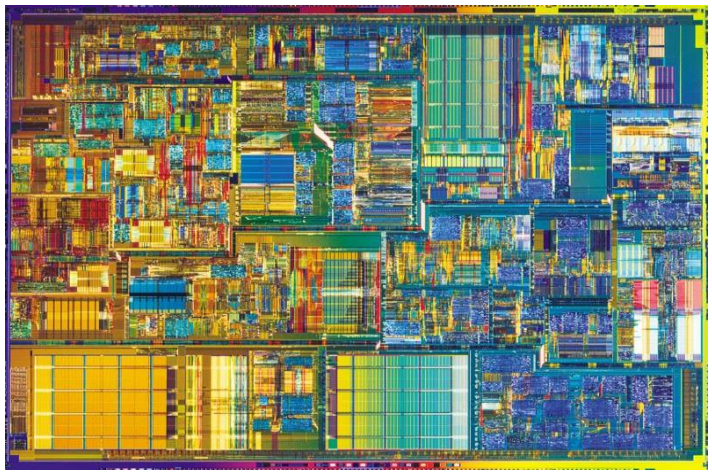


Floorplanning follows the schematics (or block diagram)

Samsung DRAM memory is using this methodology



PowerPC 604



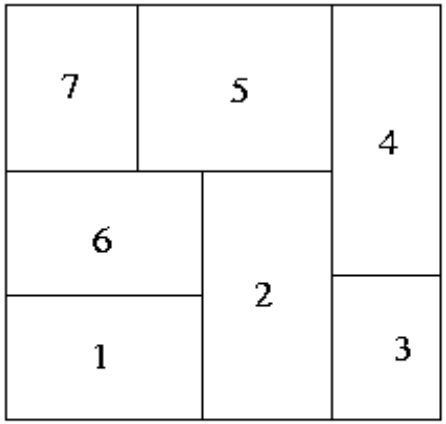
Pentium 4

Early Layout Decision Methodology

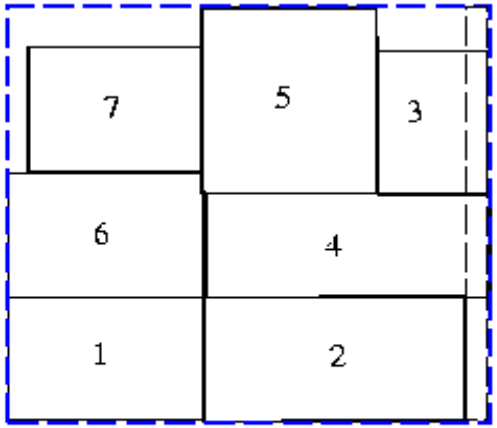
- An integrated circuit is essentially a two-dimensional medium; taking this aspect into account in early stages of the design helps in creating designs of good quality.
- Floorplanning gives early feedback: thinking of layout at early stages may suggest valuable architectural modifications; floorplanning 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.

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.



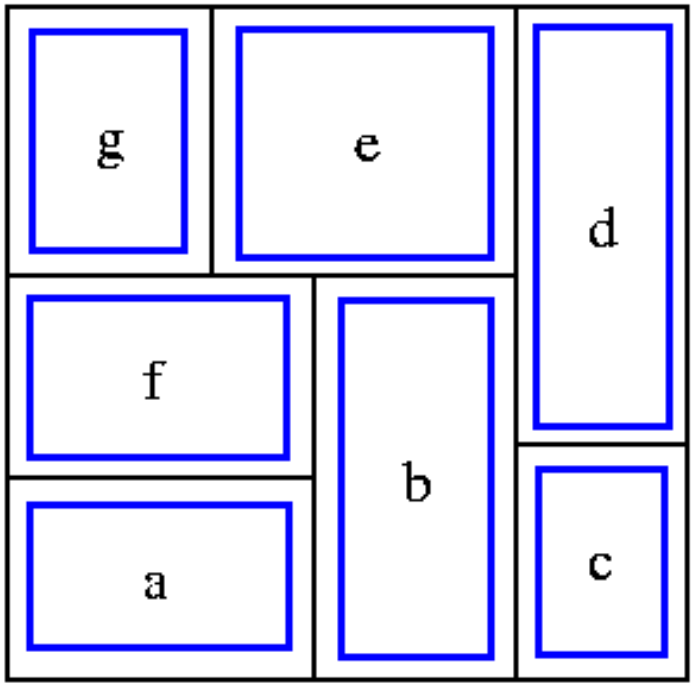
An optimal floorplan, in terms of area



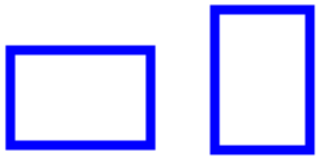
A non–optimal floorplan

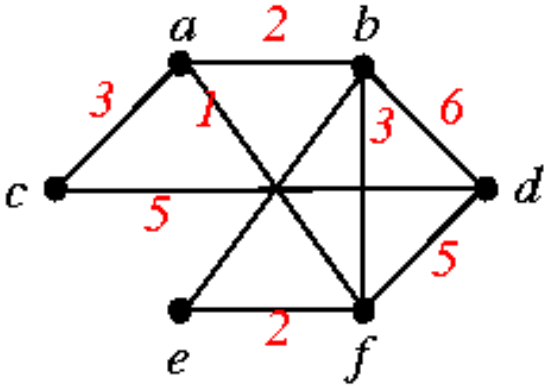
White space?
Re-orientation?
Estimating wire length?
How about congestion?

Floorplan Design



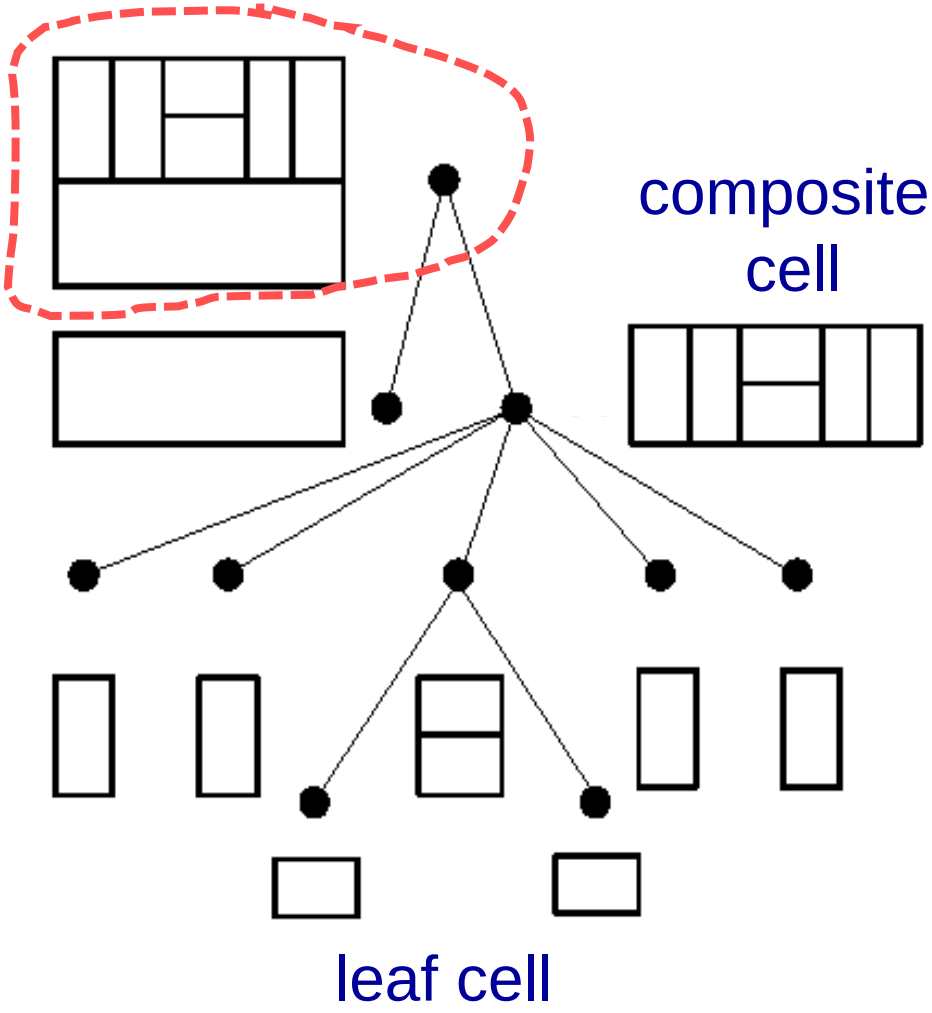
- *Modules:* 

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



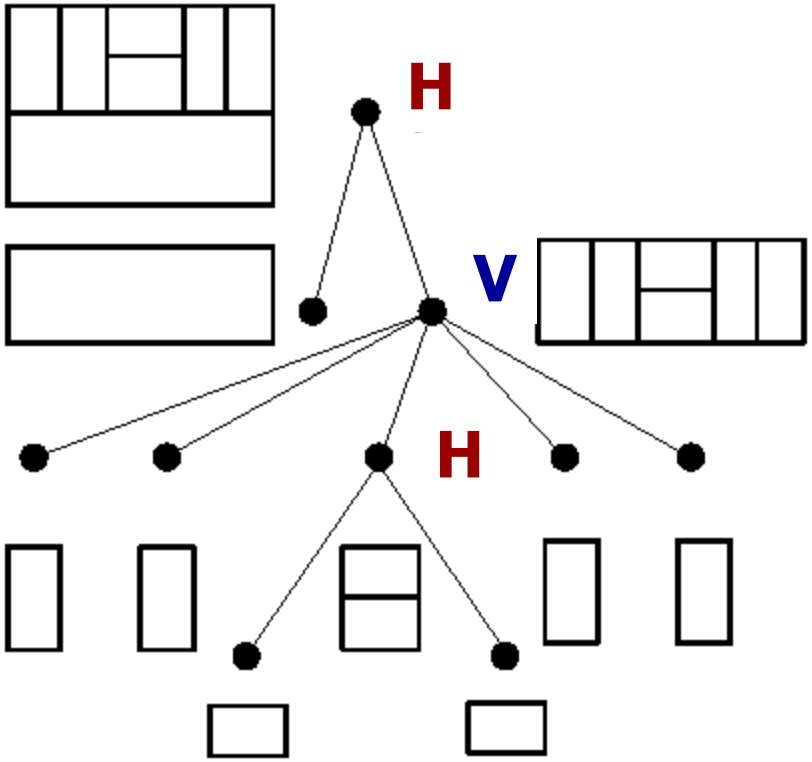
Floorplanning Concepts

- **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.



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.



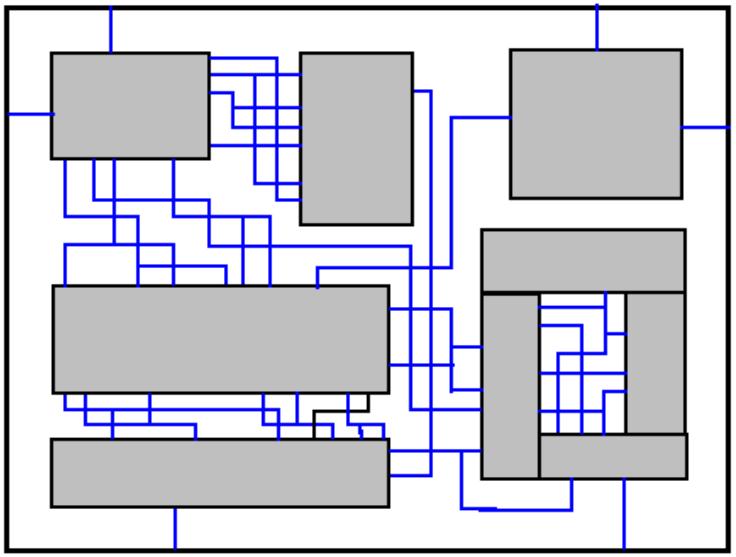
H: horizontal cut

V: vertical cut

different from the definitions in the text!!

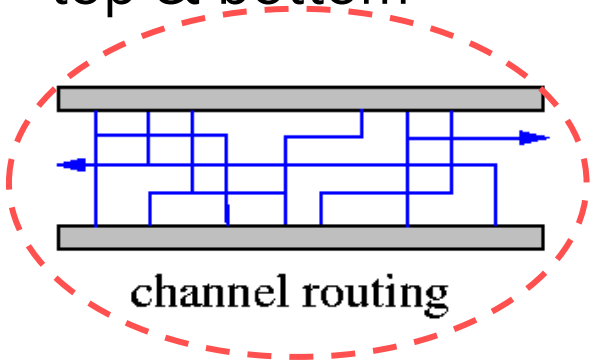
Unit 5E: Channel, Clock, and Power/Ground Routing

- Course contents
  - Channel routing
  - Clock routing
  - Power/ground routing
  - (Bus routing)
- Readings
  - Chapters 9.3 and 9.4

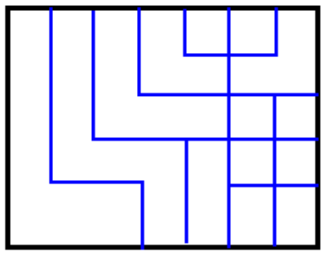


Detailed routing

Pins are fixed at top & bottom



channel routing

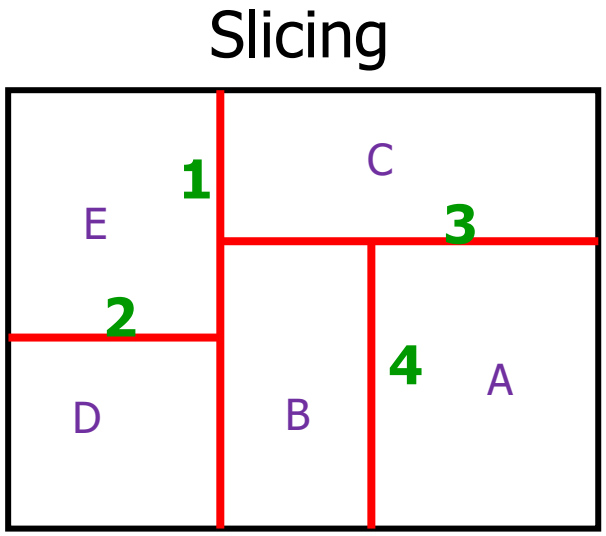


switchbox routing

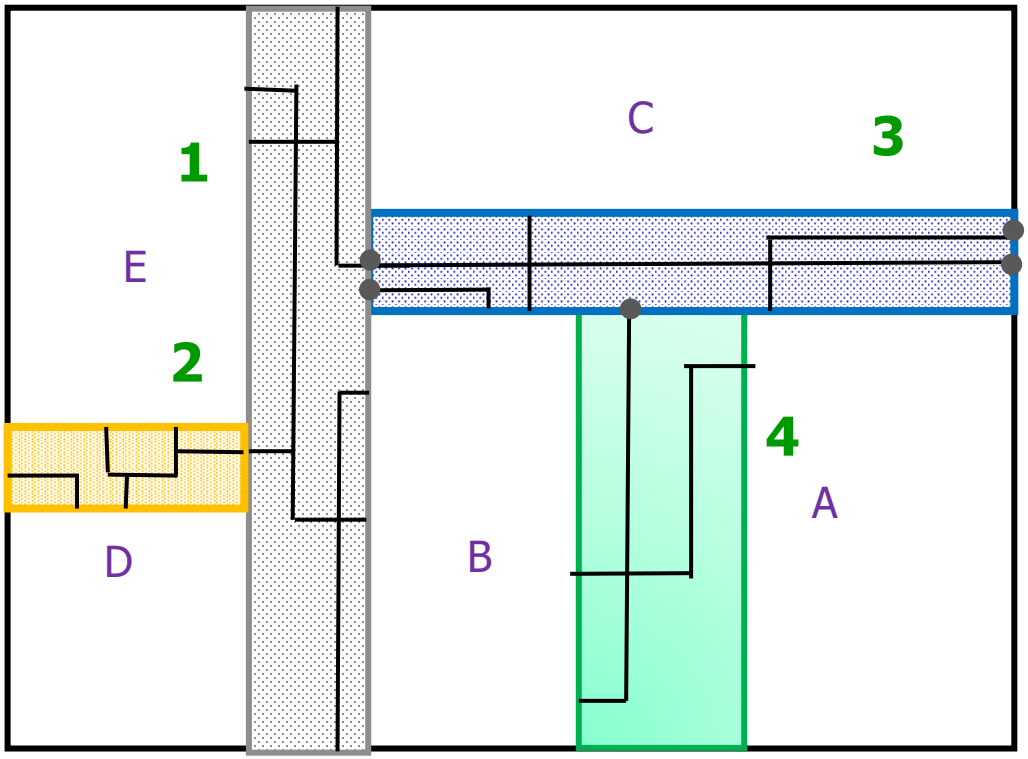
Router decides the **exit** locations at right & left sides

Pins are fixed at >=3 sides

A Good Merit Of Slicing Structure

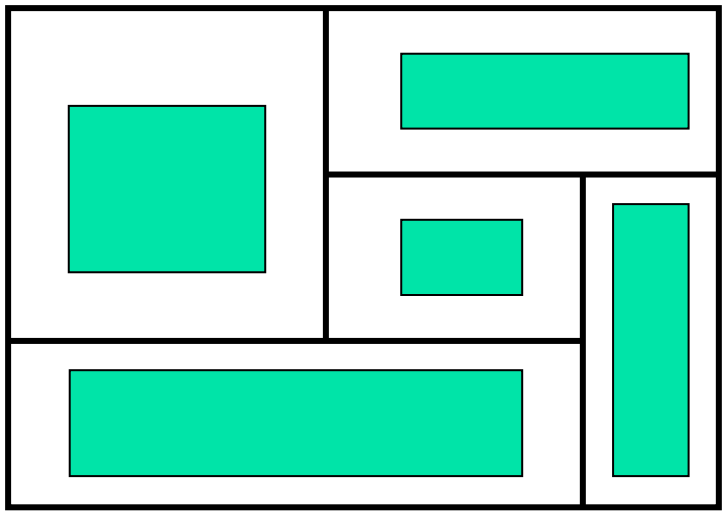


If the channel-4 is routed, then area (A, B) can be packed. Then, we route channel-3, area (C, (A+B)) can be decided. Then, we do channel-2 to get area (D, E). Lastly we route channel-1



Channel routing order is **4, 3, 2, 1**

Any Example Of Non-Slicing Structure



Slicing cut line does not go all the way to divide

This will create a switch box routing, i.e., pins are fixed at switch box boundary → switch box routing is hard and may leave some nets open

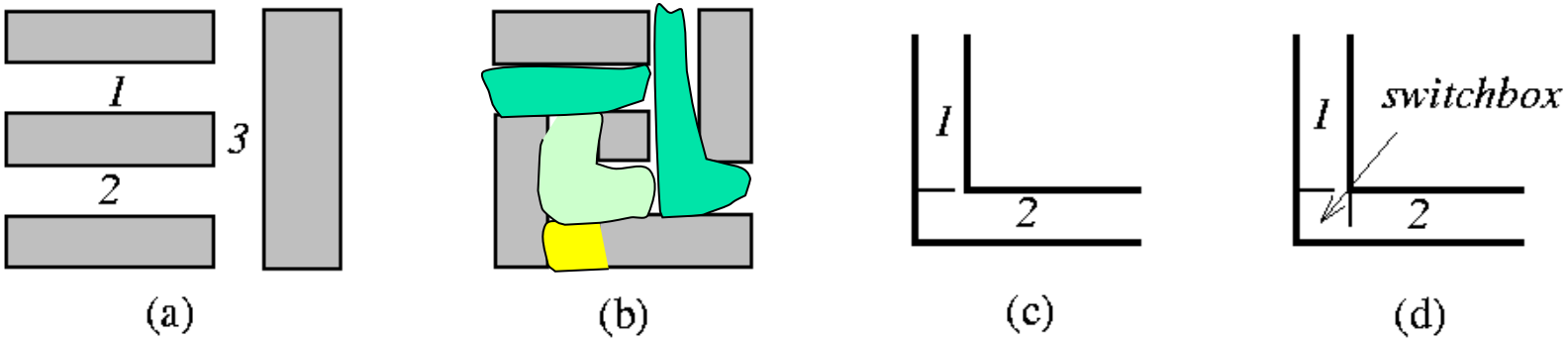
Order of Routing Regions and L-Channels

(a) No conflicts in case of routing in the order of 1, 2, and 3.

(b) No ordering is possible to avoid conflicts.

(c) The situation of (b) can be resolved by using L-channels.

(d) An L-channel can be decomposed into a channel and a switchbox.



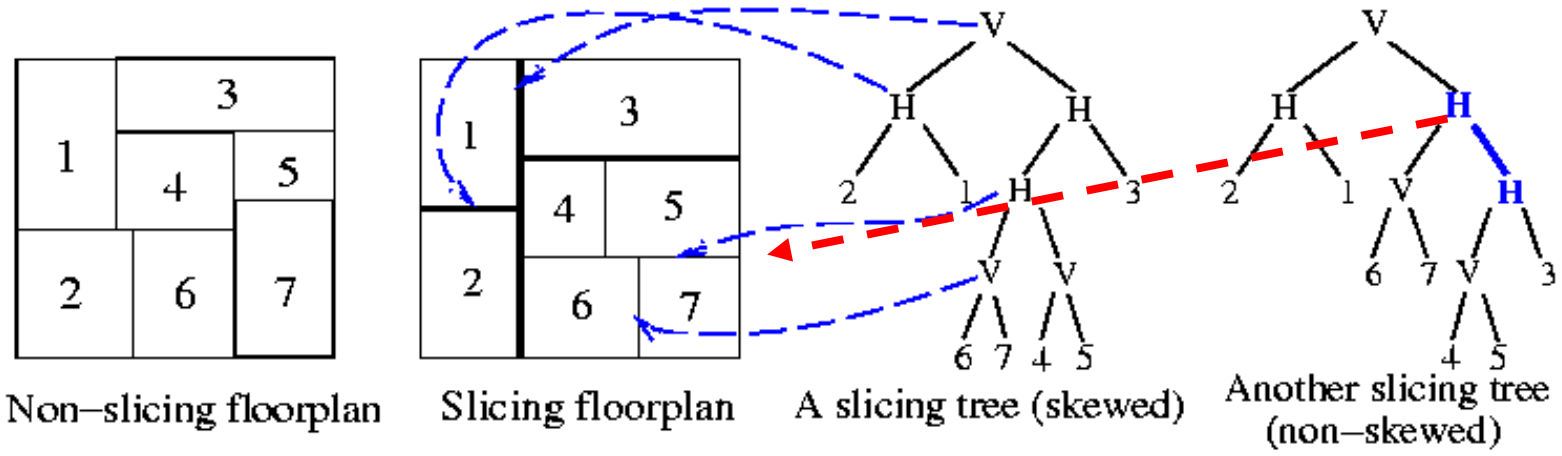
Why Slicing Is Attractive?
Easier Channel Routing Problem

- Variable size: chip layout size can be changed
  - In the past, designers will follow slicing structure, whose channel heights can be adjusted. But not all the chips can follow slicing structure.
  - And how to handle incremental routing?

- Fixed size: chip size is fixed – nowadays it is the majority
  - Two connection styles:
    - Every net is connected; then, do rip-up-reroute to fix DRC violations
    - Route as many nets as possible; if violations, leave nets open
  - Will you, as a project leader, wonder if all the routing can be done within such a fixed size area? Any suggestions?

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. (it is obtained by making consecutive vertical cuts from right to left, and making consecutive horizontal cuts from top to bottom.)



One slicing floorplan can have multiple slicing tress

Solution Representation

Please see Chap-10 (by TungChieh Chen, YW Chang). **First, explain bottom expression**

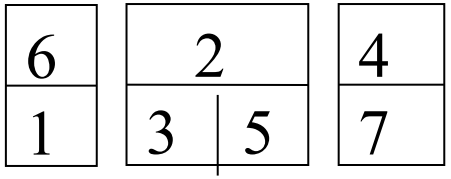
- An expression $E = e_1\ e_2 \ldots\ e_{2n-1}$, where $e_i \in \{1, 2, \ldots, n, H, V\}$, $1 \le i \le 2n\text{-}1$, is a **Polish expression** of length $2n$-1 iff
  1. every operand j, $1 \le j \le n$, appears exactly once in E;
  2. (**the balloting property**) for every subexpression $E_i = e_1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

Lenth is 13

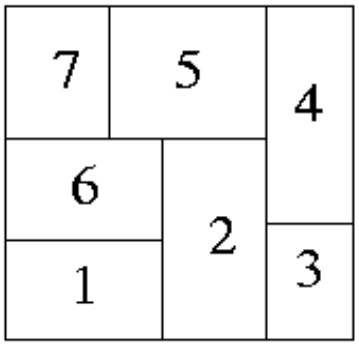
1 6 H 3 5 V 2 H V 7 4 H V

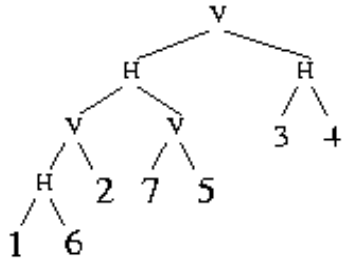
# of operands = 4 ..... = 7
# of operators = 2 ..... = 5

Welcome to draw its floorplan



- Polish expression ↔ Post-order traversal.
- *ijH*: rectangle *i* on bottom of *j*; *ijV*: rectangle *i* on the left of *j*.





E = 16H2V75VH34HV

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

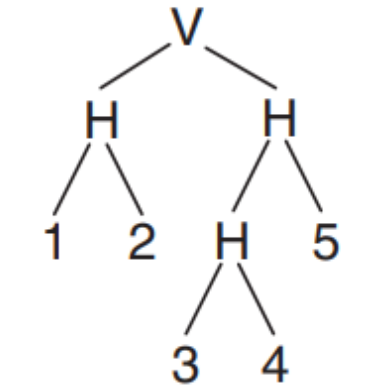
Postorder traversal of a tree!

Example 10.5 Given a binary tree shown in Figure 10.8, we can construct a Polish expression $E =$ 12H34H5HV based on the post-order traversal.

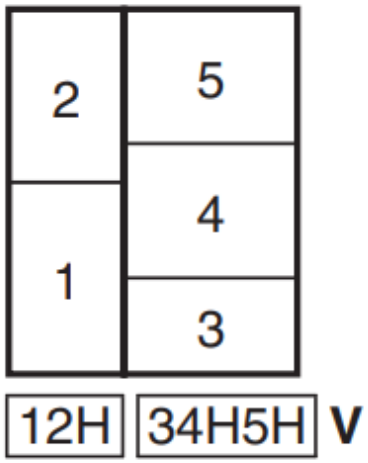
The balloting property is verified in the following table. For each column, the number of operands is always larger than that of operators. Further, $E =$ 12H34H5HV is a normalized Polish expression because there are no consecutive operators of the same type.

	1	*2*	*H*	*3*	*4*	*H*	*5*	*H*	*V*
No. of operands	1	2	2	3	4	4	5	5	5
No. of operators	0	0	1	1	1	2	2	3	4

Lenth is 9



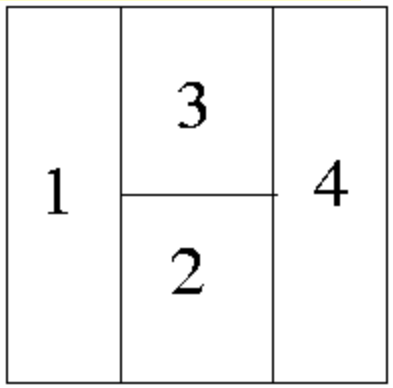
$E =$ 12H34H5HV

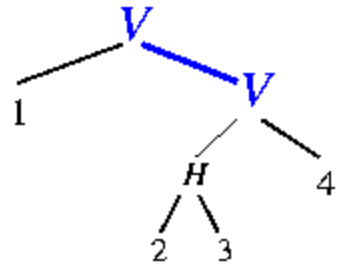


(d)

Redundant Representations

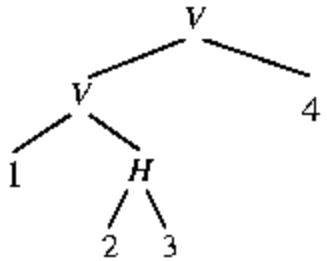
Has two representations





$E = 123H4VV$

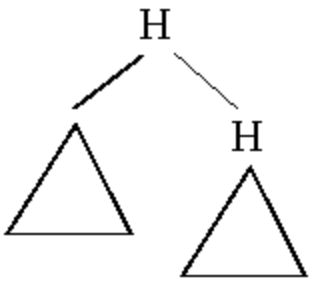
non–skewed!



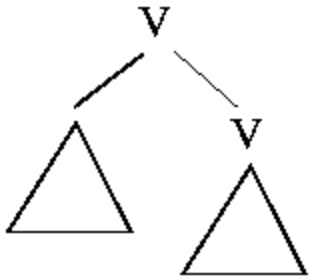
$E = 123HV4V$

skewed!

Non–skewed cases



..... HH .....



..... VV .....

(Non-skewed slicing tree, because there is a node H as the right child of anther node H, as shown on the left.)

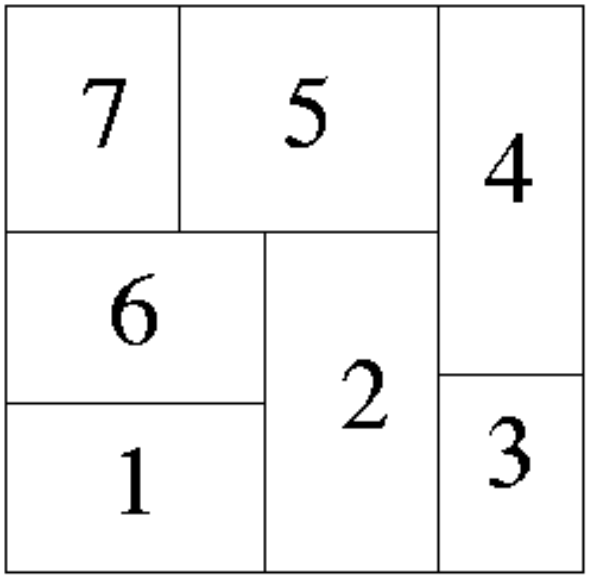
- **Question:** How to eliminate ambiguous representation?

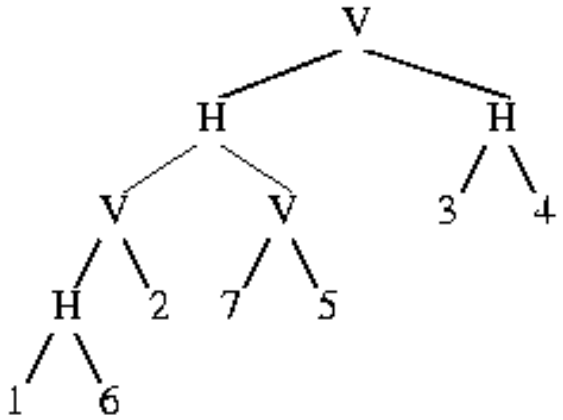
Normalized Polish Expression

- A Polish expression $E = e_1\ e_2\ \ldots\ e_{2n-1}$ is called **normalized** iff E has no consecutive operators of the same type (H or V).

Hence, skewed slicing tree

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



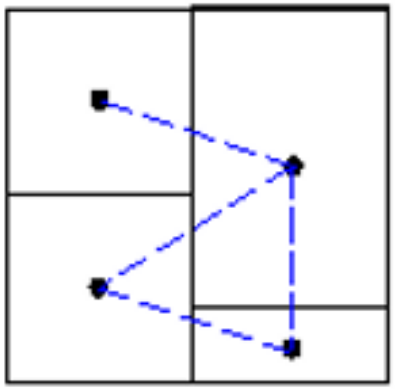


$E = 16H2V75VH34HV$

A normalized Polish expression

How Do We Decide If A Slicing Structure Is Good

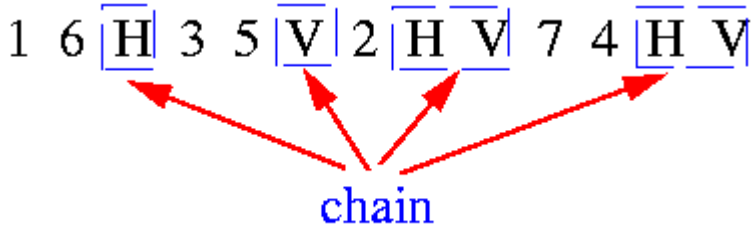
- Besides area, can we quickly estimate the total wire length?

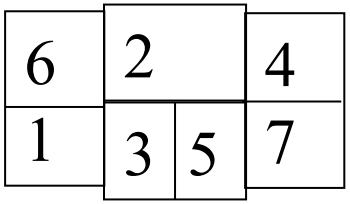


- How about congestions?
- Aspect ratio

Neighborhood Structure (For Simulated Annealing)

- **Chain:** *HVHVH* … or *VHVHV* …



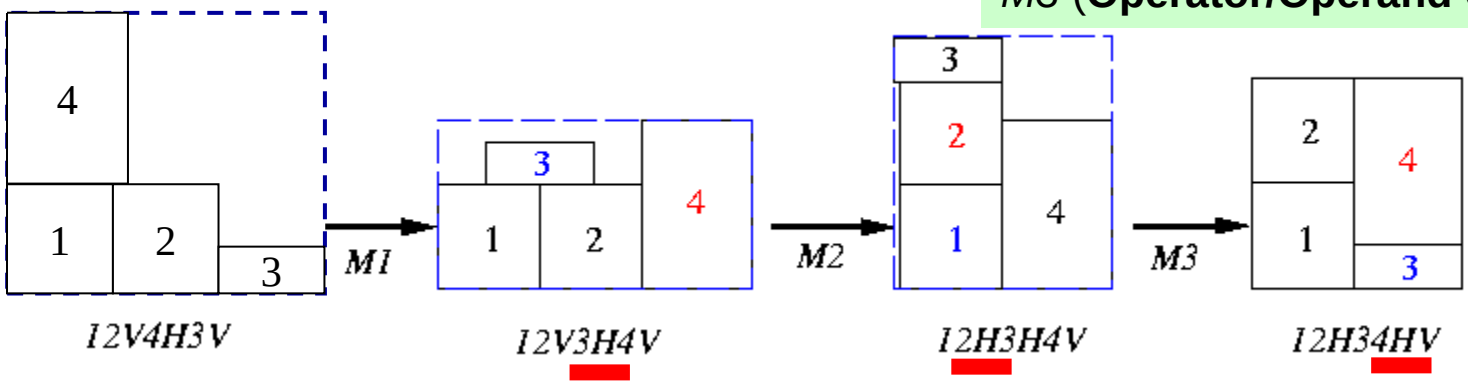


- **Adjacent:** 1 and 6 are adjacent operands; 2 and 7 are adjacent operands; 5 and *V* are adjacent operand and operator.

- 3 types of moves:
  - *M1* (**Operand Swap**): Swap two adjacent operands.
  - *M2* (**Chain Invert**): Complement some chain (*V* = *H*, *H* = V).
  - *M3* (**Operator/Operand Swap**): Swap two adjacent operand and operator.

Effects of Perturbation

M1 (**Operand Swap**):
M2 (**Chain Invert**):
M3 (**Operator/Operand Swap**):



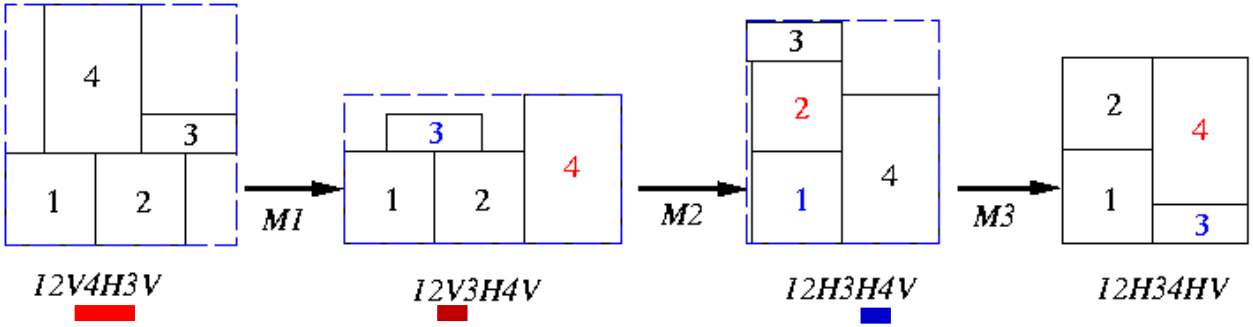
- **Question:** The balloting property holds during the moves?
  - *M1* and *M2* moves are OK.
  - **Check the *M3* moves!** Reject "illegal" *M3* moves.

- **Check *M3* moves:** Assume that the *M3* move swaps the operand e_i with the operator e_{i+1}, $1 \le i \le k\text{-}1$. Then, the swap will not violate the balloting property iff $2N_{i+1} < i$.
  - N_k: # of operators in the Polish expression $E = e_1\ e_2\ \ldots\ e_k$, $1 \le k \le 2n\text{-}1$

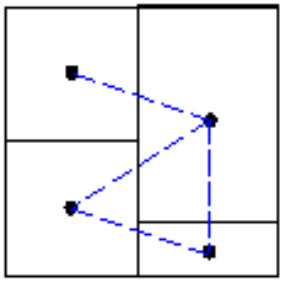
Cost Function

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

M1 (**Operand Swap**):
M2 (**Chain Invert**):
M3 (**Operator/Operand Swap**):

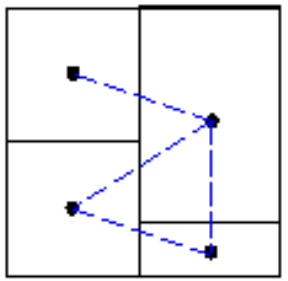


- $W = \sum_{ij} c_{ij}\, d_{ij}$.
  - c_{ij}: # of connections between blocks *i* and *j*.
  - d_{ij}: center-to-center distance between basic rectangles *i* and *j*.



Slicing tree for a floorplan → Polish expression → Perturb expression → Evaluate cost functions

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



Can we rotate?

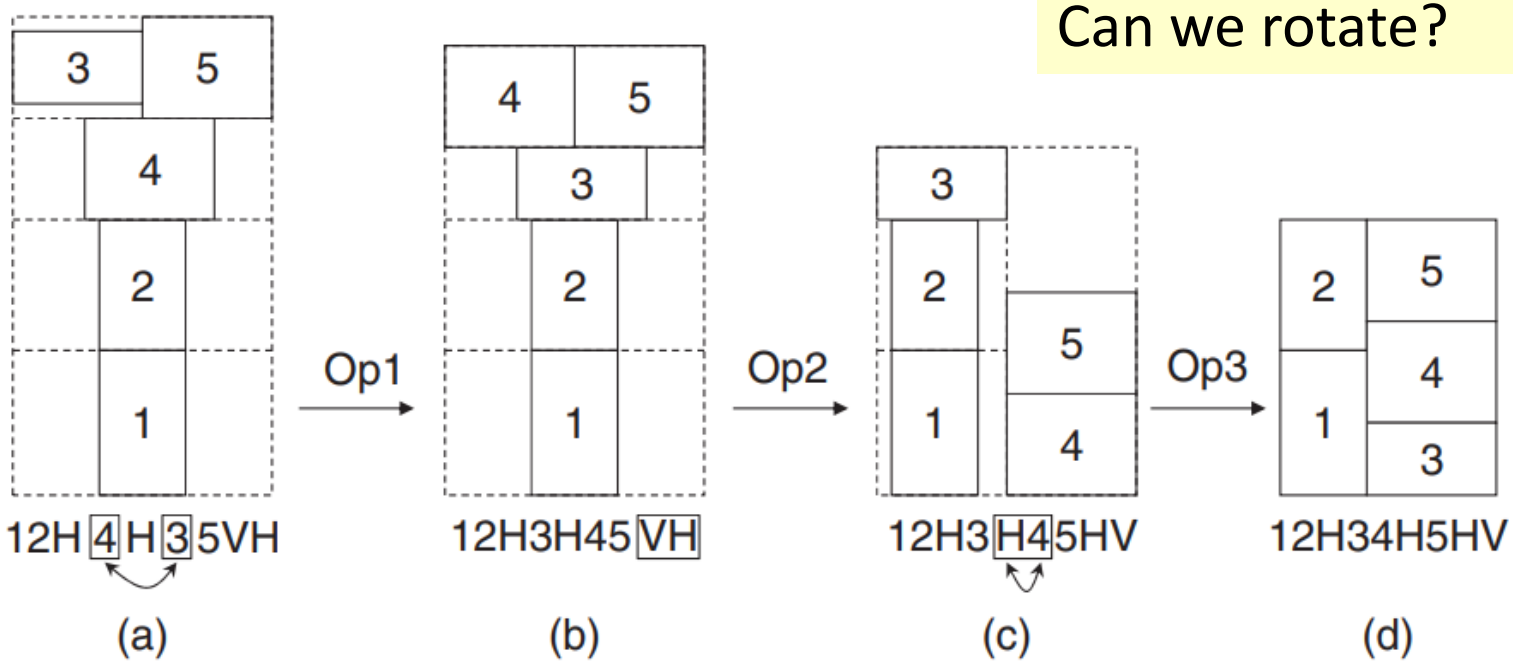
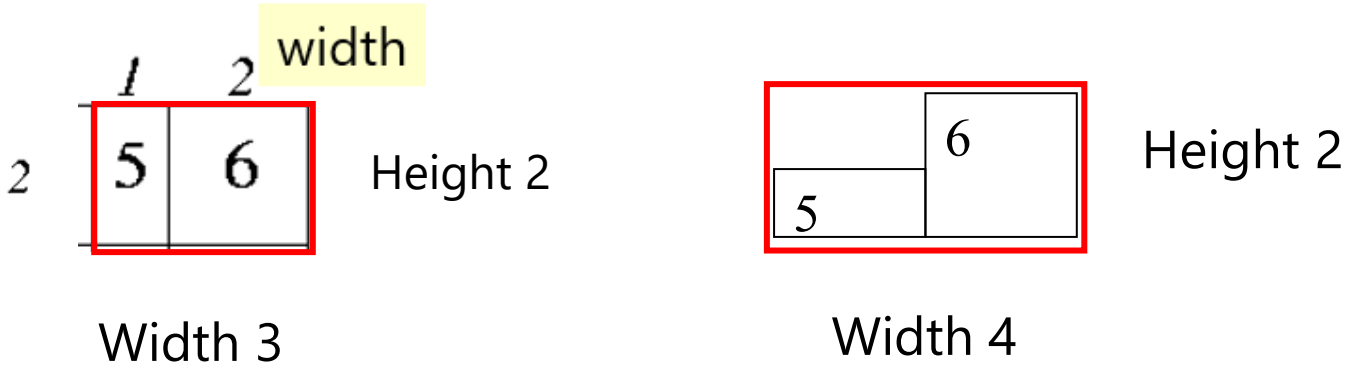


FIGURE 10.11

Illustrations of the perturbations in Example 10.7.

I believe several debugging utilities were developed:
- given a slicing structure, generate an expression
- given an expression, create the corresponding slicing structure

If macro is rotated, we need to compute the bounding box area

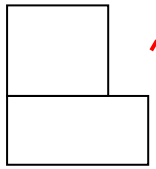


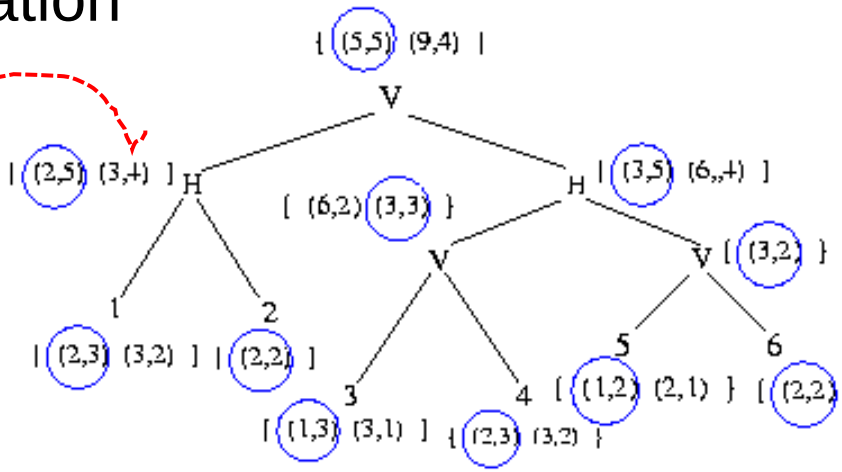
Width 3

Width 4

Area Computation for Hard Blocks

- Allow rotation

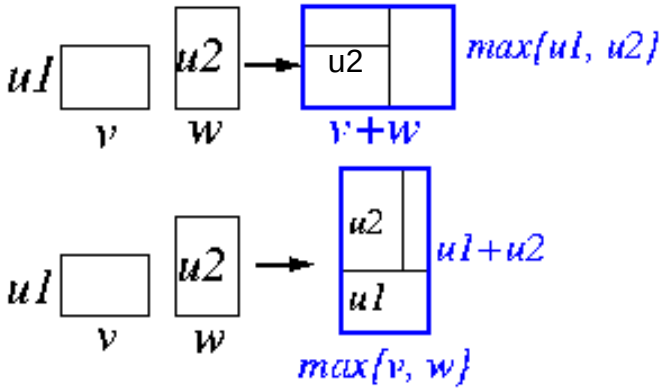


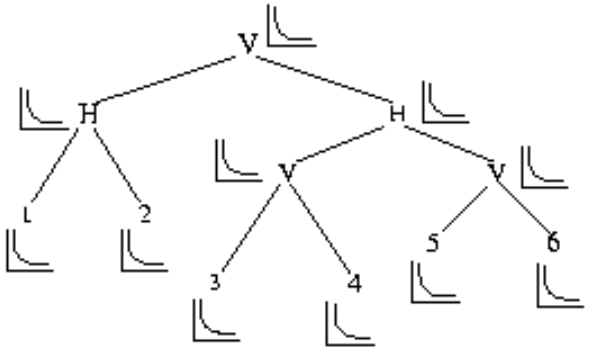


width

	2	1	2
2	2	5	6
3	1	3	4

Inst-1 (w=2, h=3),
If rotated, (3,2)



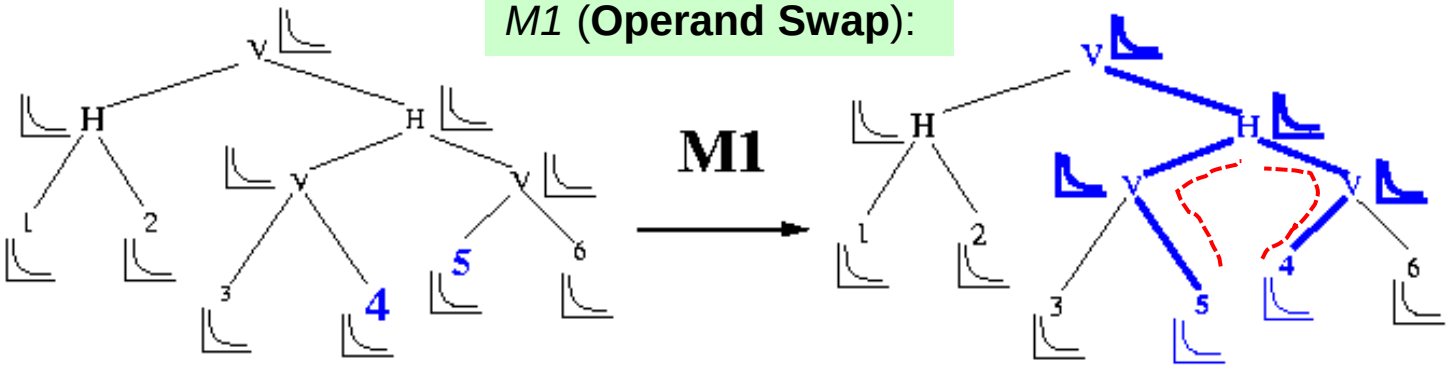


- Wiring cost?
  - Center-to-center interconnection length

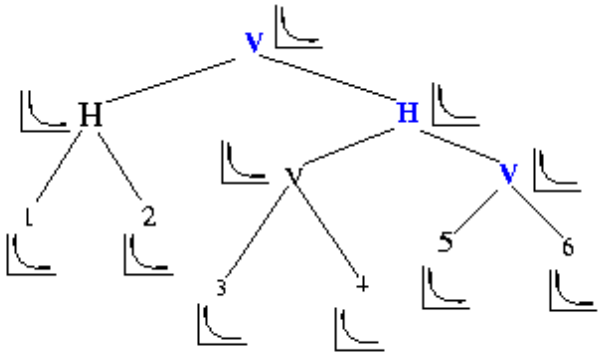
Incremental Computation of Cost Function

- 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.

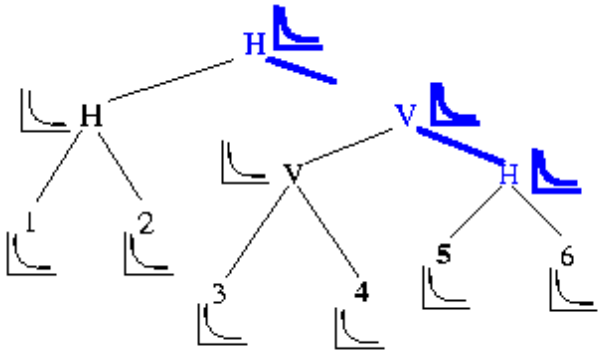
M1 (**Operand Swap**):



Incremental Computation of Cost Function (cont'd)

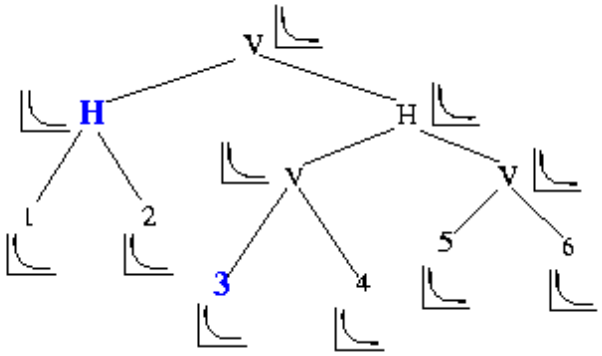


M2

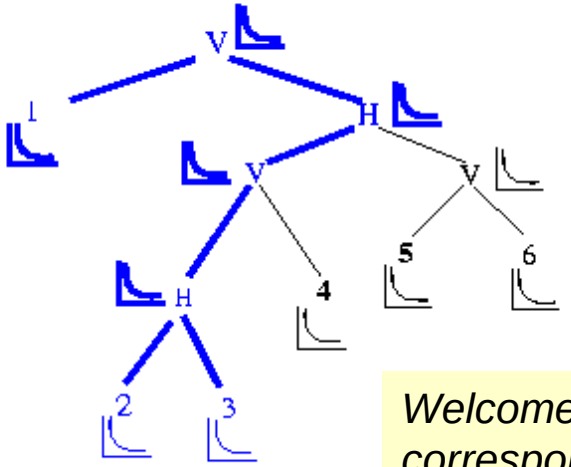


E = 12H34V56VHV

E = 12H34V56HVH



M3



E = 12H34V56VHV

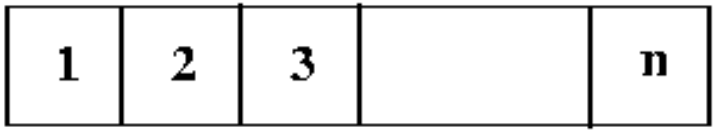
E = 123H4V56VHV

Welcome to draw the corresponding floorplans

M2 (**Chain Invert**):
M3 (**Operator/Operand Swap**):

Annealing Schedule

- Initial solution: 12*V*3*V* … *nV*.



- $T_i = r^i T_0$, i = 1, 2, 3, …; 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 Floorplanning (P, ε, r, k)

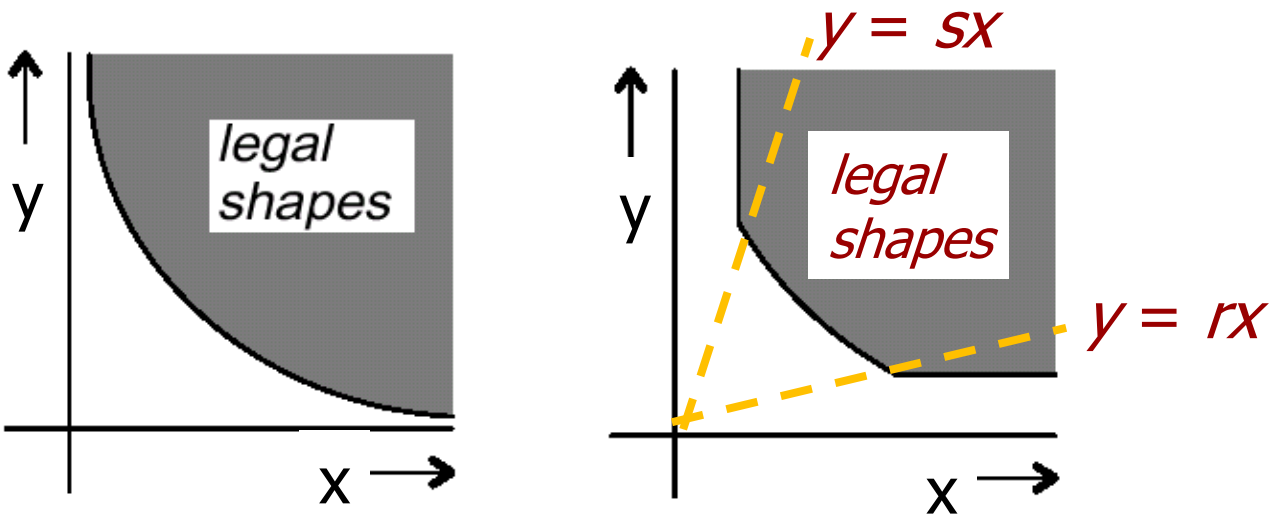
```
1 begin
2 E ← 12V3V4V … nV; /* initial solution */
3 Best ← E; T0 ← Δ_avg / ln(P) ; M ← MT ← uphill ← 0; N = kn;
4 repeat
5   MT ← uphill ← reject ← 0;
6   repeat
7     SelectMove(M);
8     Case M of
9     M1: Select two adjacent operands e_i and e_j; NE ← Swap(E, e_i, e_j);
10    M2: Select a nonzero length chain C; NE ← Complement(E, C);
11    M3: done ← FALSE;
12       while not (done) do
13          Select two adjacent operand e_i and operator e_{i+1};
14          if (e_{i-1} ≠ e_{i+1}) and (2 N_{i+1} < i) then done ← TRUE;
13'         Select two adjacent operator e_i and operand e_{i+1};
14'         if (e_i ≠ e_{i+2}) then done ← TRUE;
15       NE ← Swap(E, e_i, e_{i+1});
16    MT ← MT+1; Δcost ← cost(NE) - cost(E);
17    if (Δcost ≤ 0) or (Random < e^(-Δcost/T) )
18     then
19        if (Δcost > 0) then uphill ← uphill + 1;
20        E ← NE;
21        if cost(E) < cost(best) then best ← E;
22     else reject ← reject + 1;
23   until (uphill > N) or (MT > 2N);
24   T ← rT; /* reduce temperature */
25 until (reject/MT > 0.95) or (T < ε) or OutOfTime;
26 end
```

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 to simulated annealing
  - solution space?
  - neighborhood structure?
  - cost function?
  - annealing schedule?

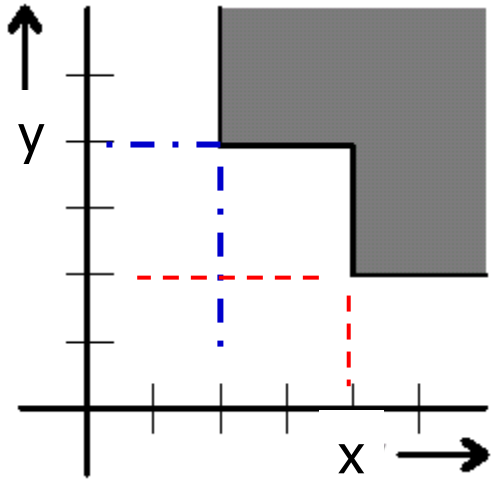
Shape Curve (For Soft Macros)

- 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.
  - Often we could say that the realization needs an area A (i.e., xy >= A)
- 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: $r <= y/x <= s$.



Shape Curve (cont'd)

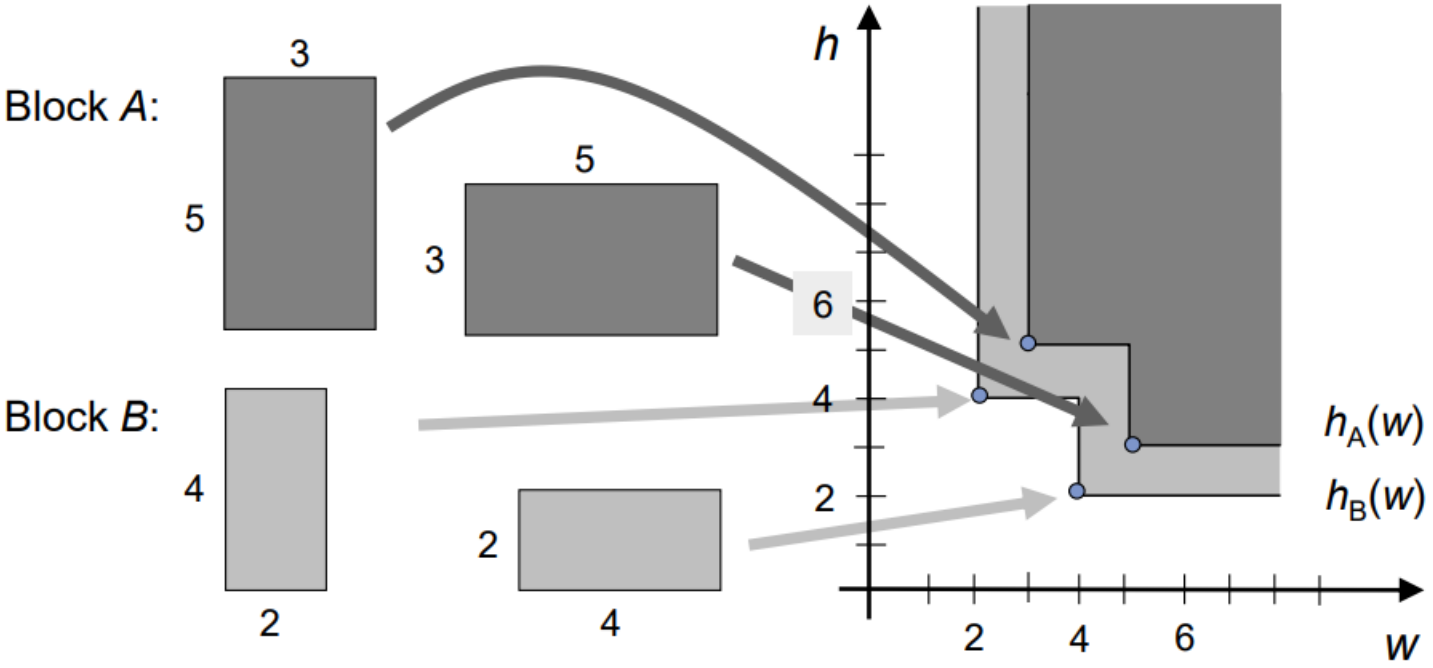
- Leaf cells are built from discrete transistors: it is not realistic to assume that the shape function follows the hyperbola continuously.
- In an extreme case, a cell is rigid: it can only be rotated and mirrored during floorplanning or placement.



The shape function of a 2 × 4 inset cell.

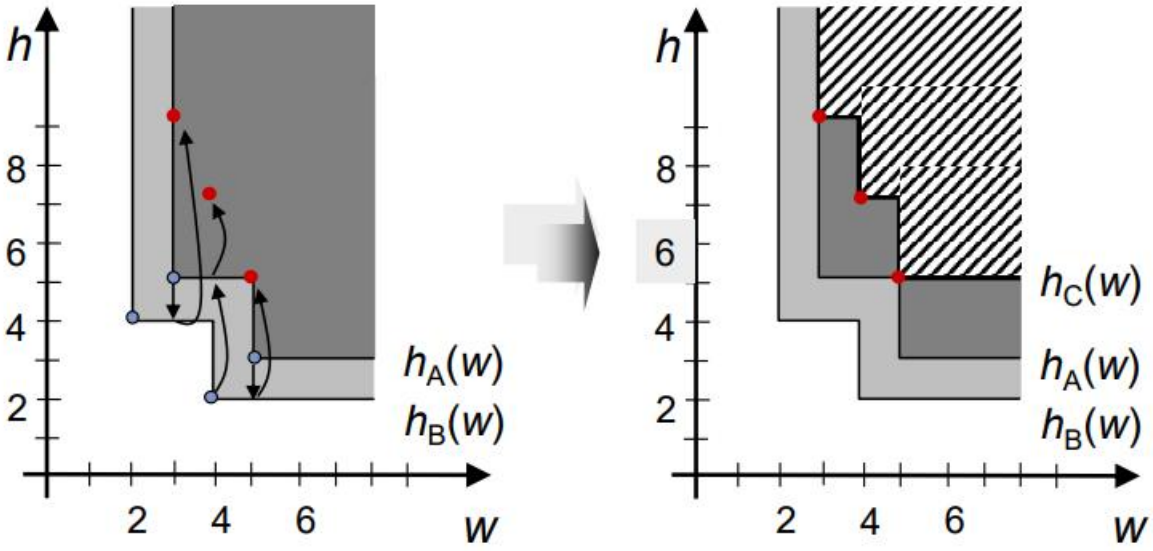
Floorplan Sizing – Example

Step 1: Construct the shape functions of the blocks



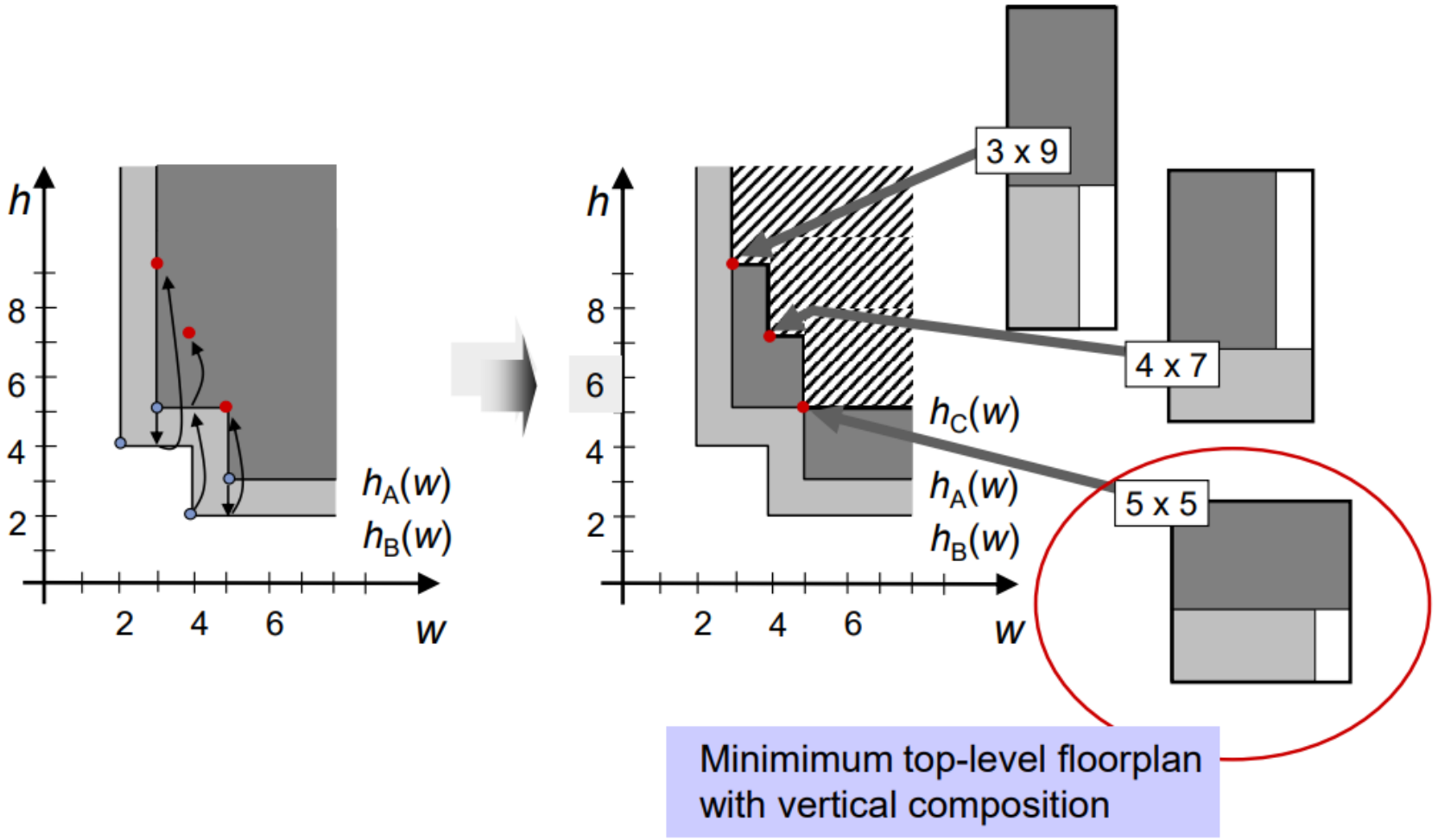
Floorplan Sizing – Example

Step 2: Determine the shape function of the top-level floorplan (vertical)



Floorplan Sizing – Example

Step 2: Determine the shape function of the top-level floorplan (vertical)



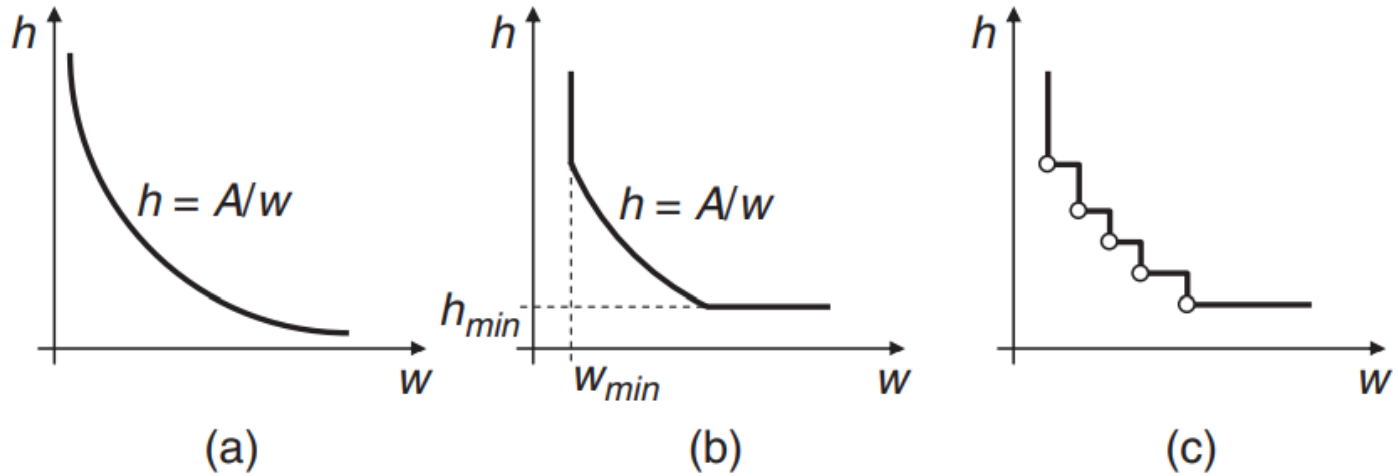
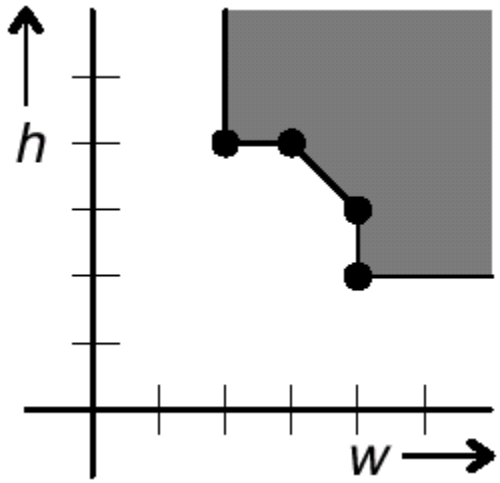


FIGURE 10.30

The shape curve of a module: (a) The shape curve in a hyperbola function. (b) The shape curve with the minimum width/height constraint. (c) The piecewise linear shape curve.

Shape Curve (cont'd)

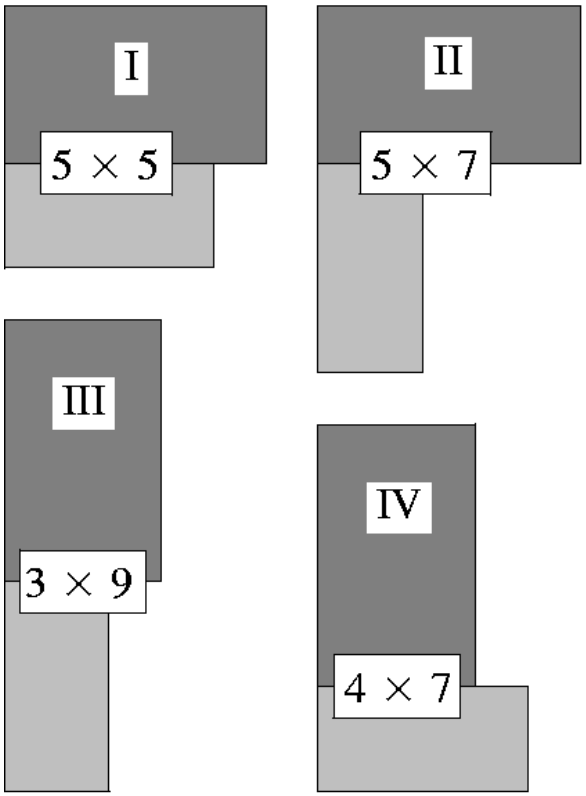
- In general, a *piecewise linear* function can be used to approximate any shape function.
- The points where the function changes its direction, are called the corner (*break) points* of the piecewise linear function.

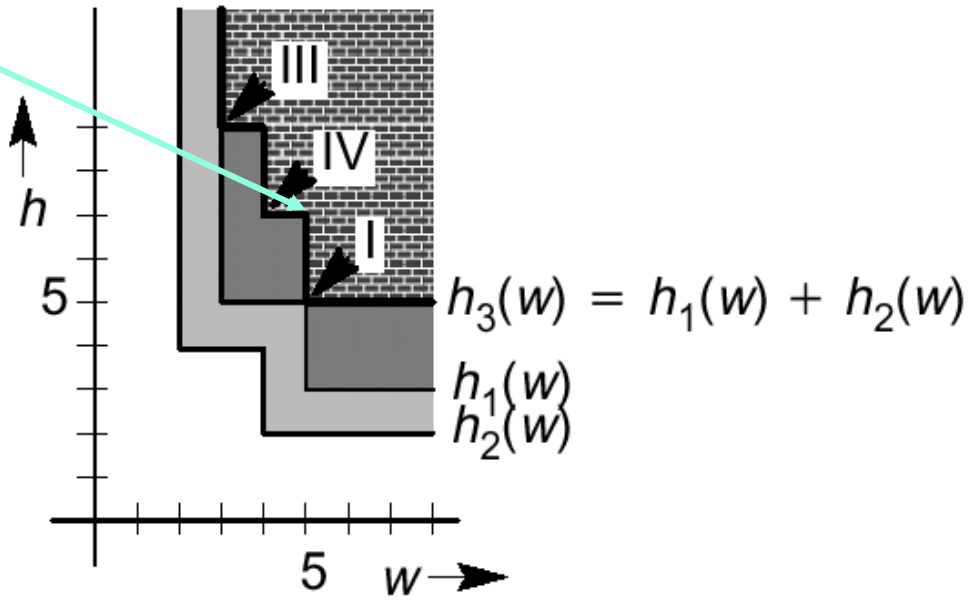


Addition for Vertical Abutment

- Composition by vertical abutment ⇒ the additon of shape functions.

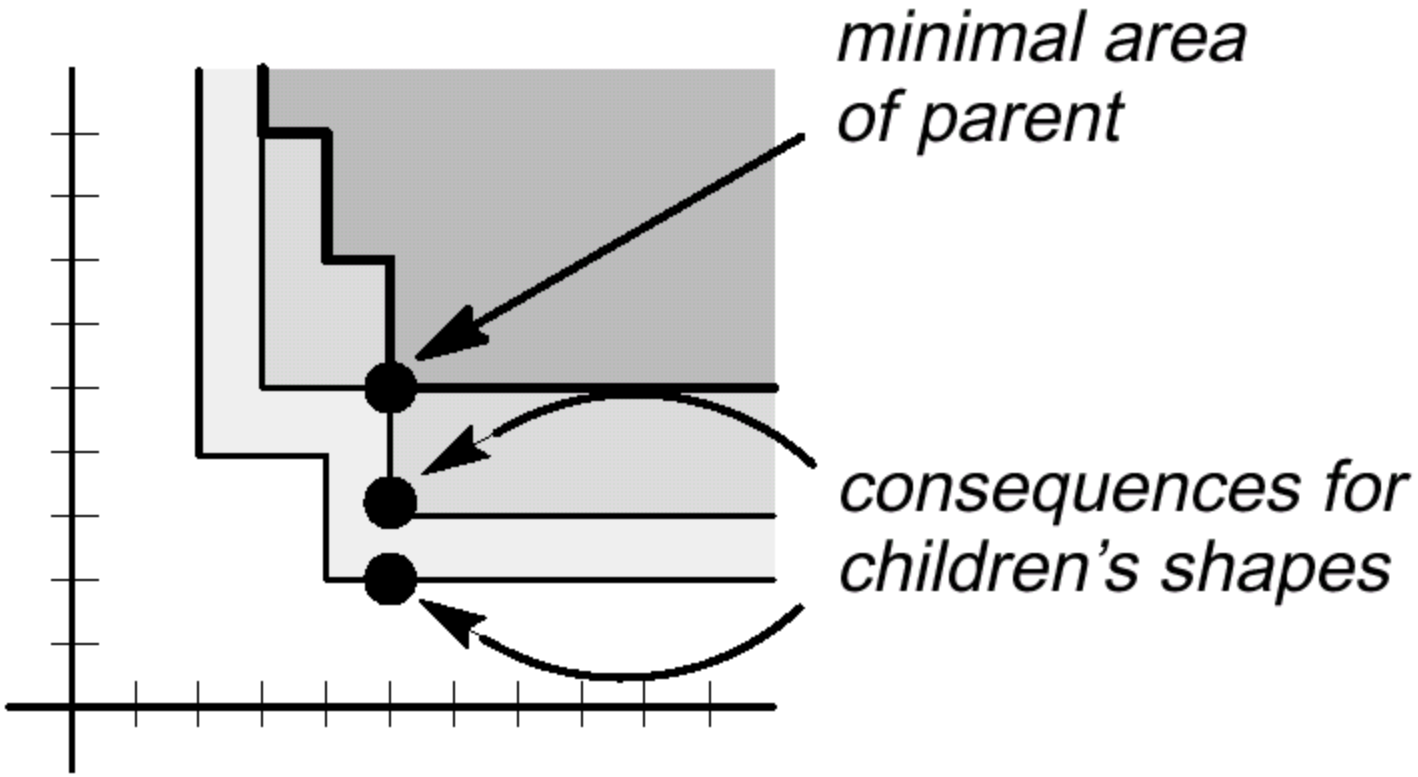
We are doing these for cost functions



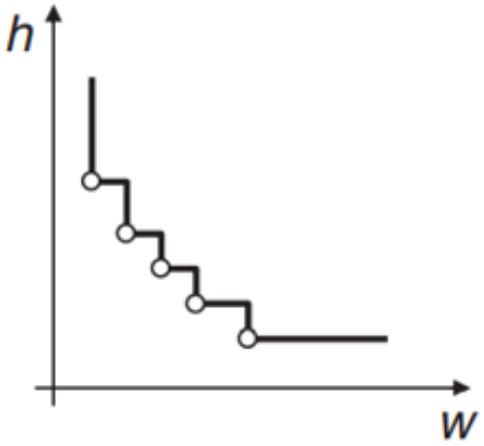


Deriving Shapes of Children

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



Floorplan sizing. The availability of flexible cells implies the possibility of having different shapes for the same hardware unit. It is therefore possible to choose a suitable shape for each leaf cell such that the resulting floorplan is optimal in some sense (e.g. minimal area).



(c)

Sizing Algorithm for Slicing Floorplans

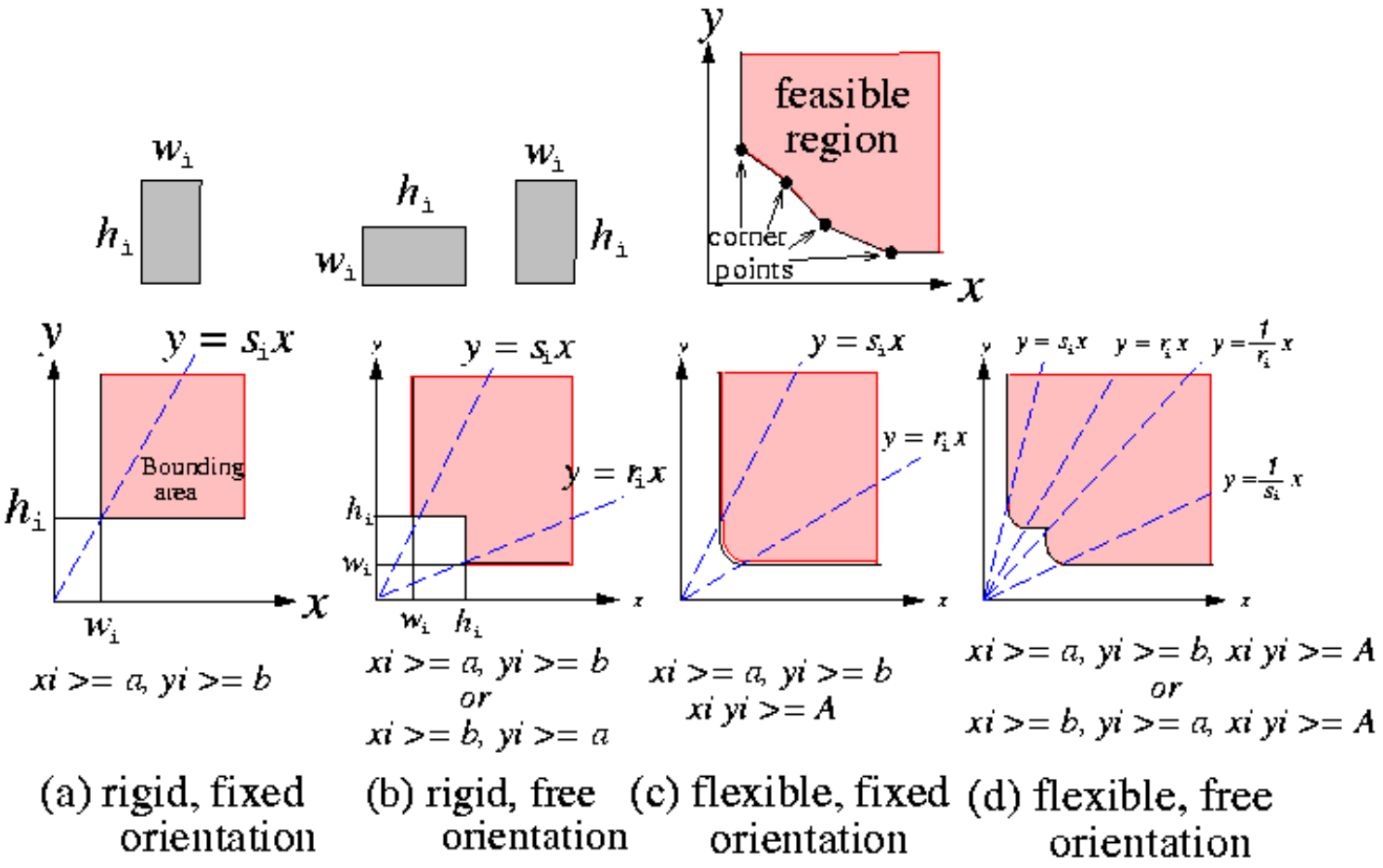
- The shape functions of all leaf cells are given as piecewise linear functions.
- Traverse the slicing tree in order to compute the shape functions of all composite cells (bottom-up composition).

- 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

Feasible Implementations

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

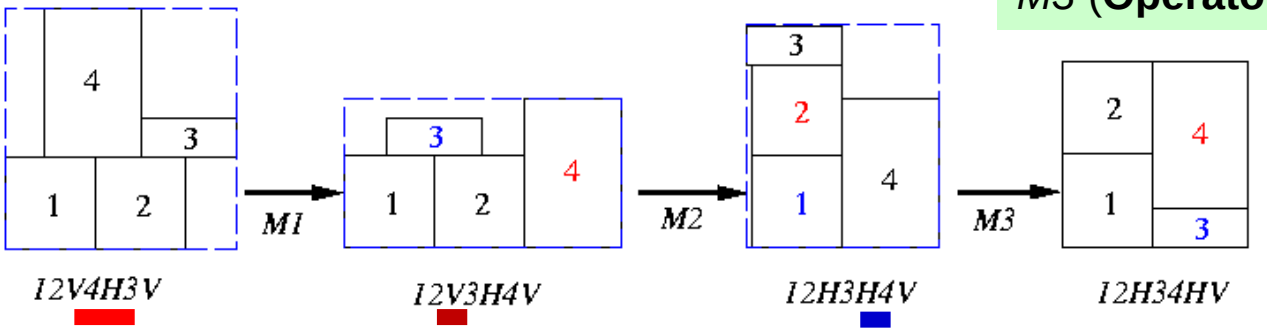


Cost Function

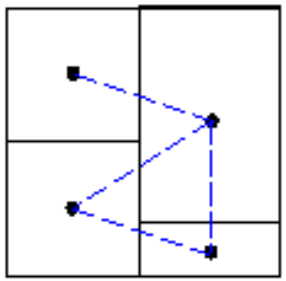
Can we consider congestion in cost function? How?

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

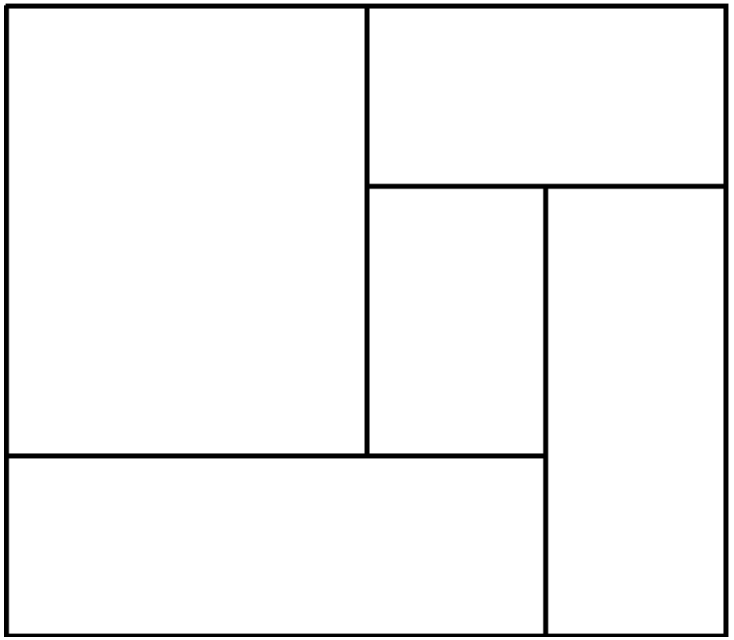
M1 (**Operand Swap**):
M2 (**Chain Invert**):
M3 (**Operator/Operand Swap**):



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



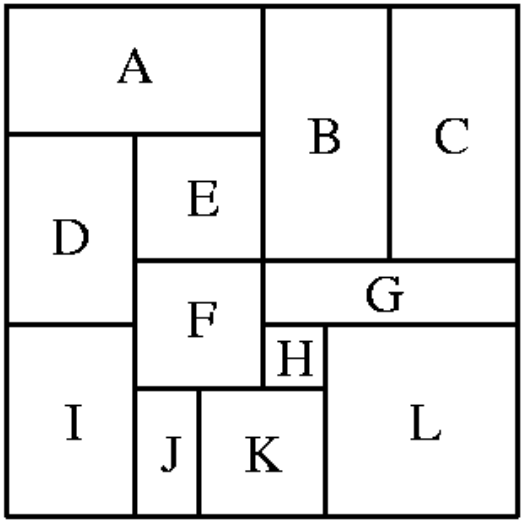
Wheel or Spiral Floorplan

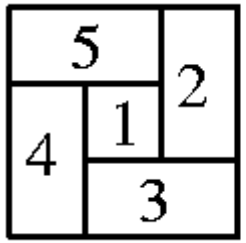


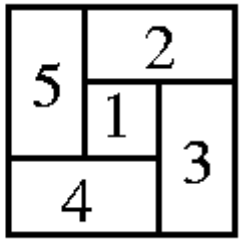
- This floorplan is not slicing!
- **Wheel** is the smallest non-slicing floorplans.

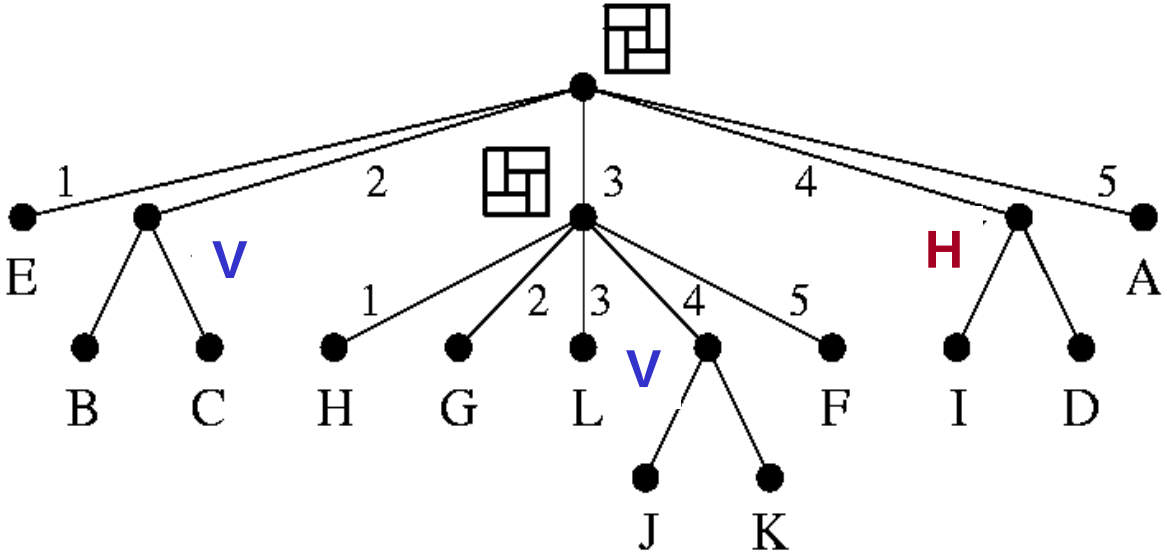
- Limiting floorplans to those that have the slicing property is reasonable: it certainly facilitates floorplanning algorithms.
- Taking the shape of a wheel floorplan and its mirror image as the basis of operators leads to hierarchical descriptions of *order 5*.

Order-5 Floorplan Examples

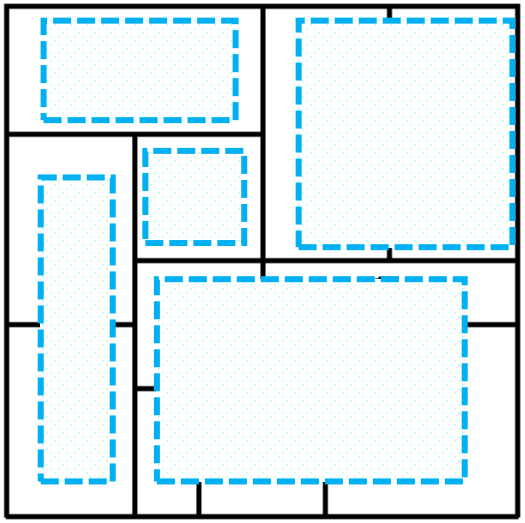


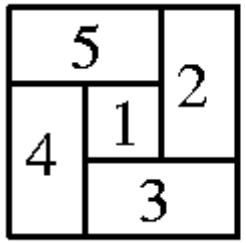


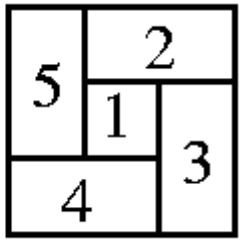


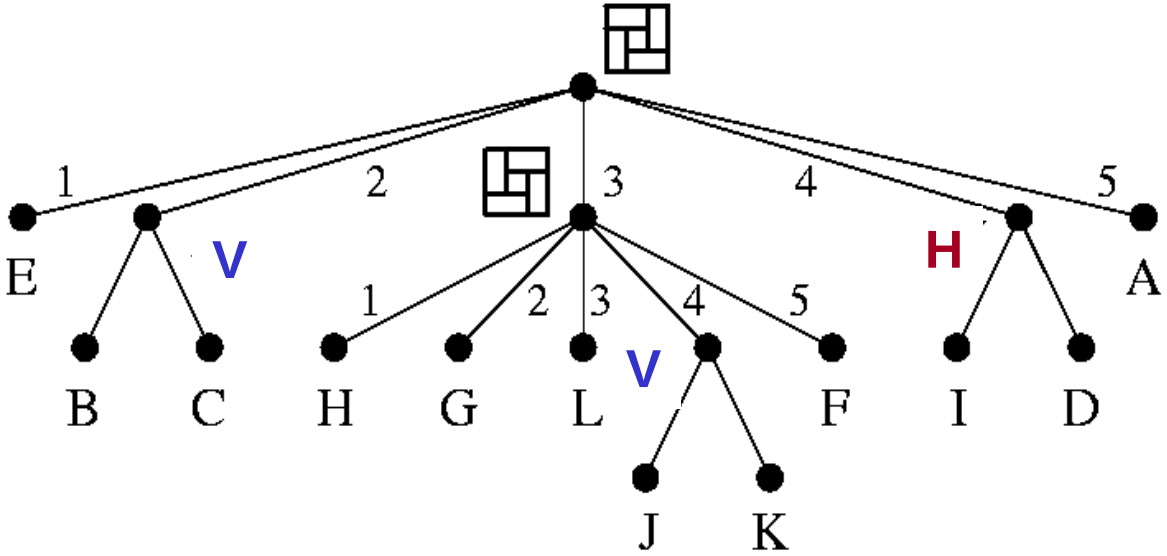


Order-5 Floorplan Examples



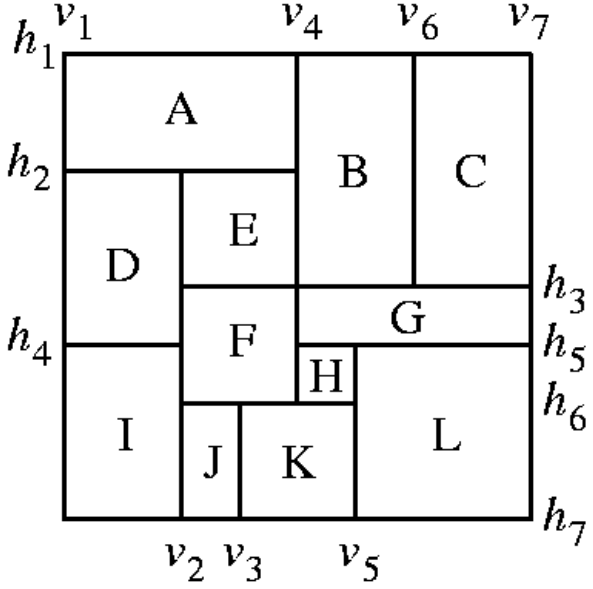


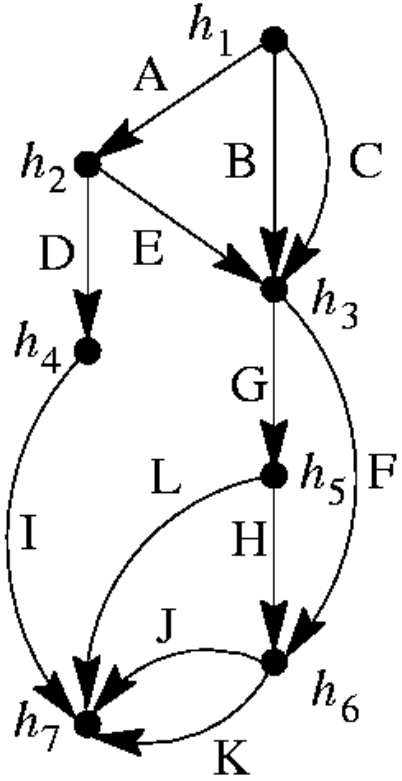




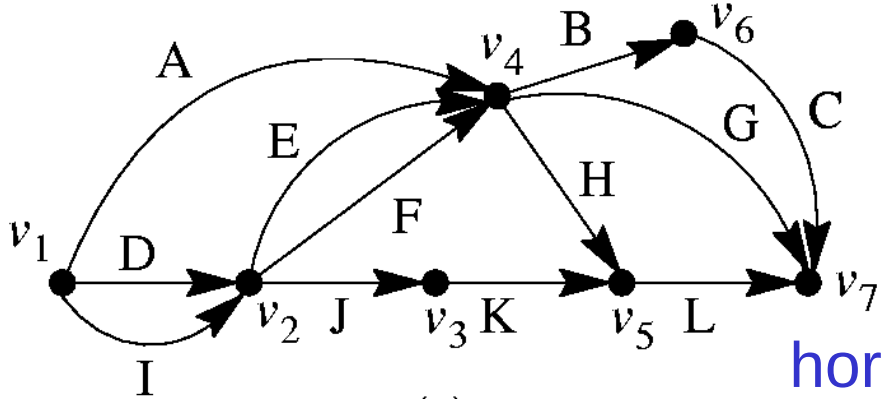
General Floorplan Representation: Polar Graphs (Non Slicing)

- vertex：channel segment
- edge (weight): cell/block/module (dimension).





vertical polar graph



horizontal polar graph

About Sizing Of Floorplans

The corresponding area can then be obtained by computing the height and width of the floorplan, which amounts to the computation of the longest paths in the vertical and horizontal polar graphs respectively. This can be done in O(n) time using the algorithm of Section 6.4.1, because the polar graphs have exactly n edges each.

6.4 Algorithms for Constraint-graph Compaction

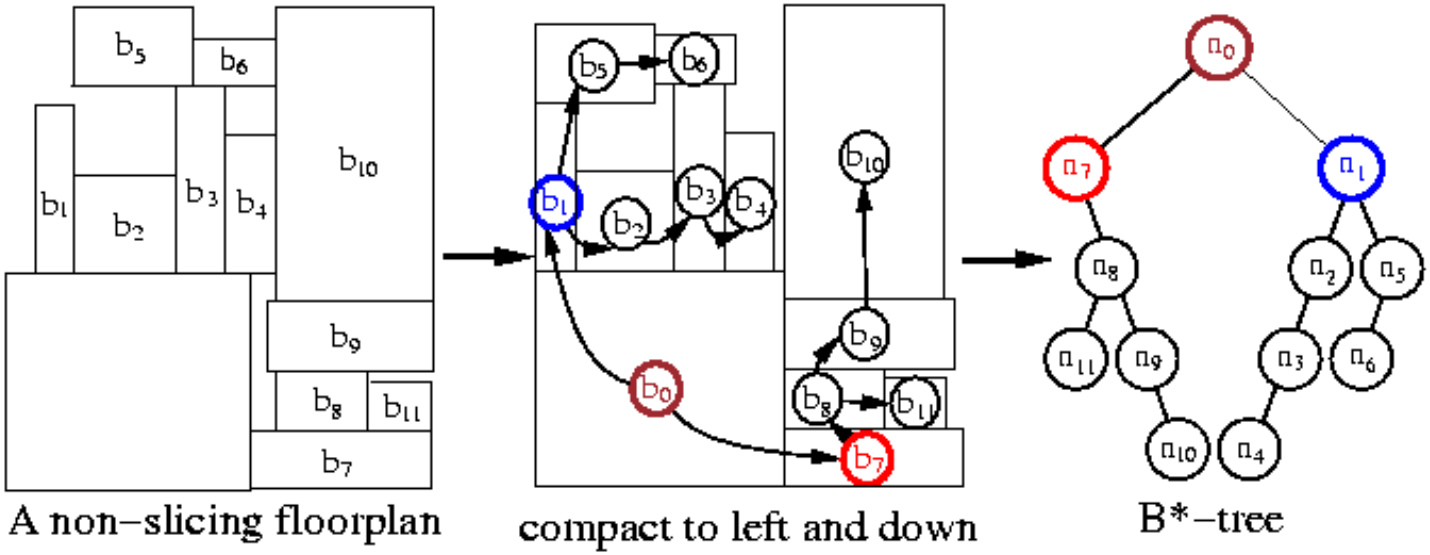
In this section, algorithms for constraint-graph compaction are presented. This type of compaction amounts to finding the longest path in a directed graph as was discussed above. First an algorithm for the simpler case of the longest path in a DAG is given. Then two different algorithms for graphs with cycles are explained.

6.4.1 A Longest-path Algorithm for DAGs

B*-Tree for Compacted Non-Slicing Floorplans

- Chang et. al., "B*-tree: A new representation for non-slicing floorplans," DAC-2000 (B*-tree is based on ordered binary tree to model compacted floorplans)
  1. Compact modules to left and bottom (O-tree).
  2. Construct an ordered binary tree (B*-tree) (see the paper for more rigid rules!!)
  - **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 ($x_j = x_i$).
- 1-to-1 correspondence between a compacted non-slicing floorplan and its induced B*-tree.

Admissible placement means that it is compacted and can neither move down nor move left.



https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.36.750&rep=rep1&type=pdf

Sequence pair/ BSG	a1. can handle non-slicing structure a2. very flexible in representation d1. time-consuming d2. the solution space is large d3. sequence encoding cost is high d4. harder to transform between a sequence pair and a placement d5. sequence pair cannot handle soft modules directly d6. BSG incurs redundancies
O-tree	a1. can handle non-slicing structure a2. the solution space is smaller a3. transformation between representation and placement takes only linear time a4. encoded by fewer bits than sequence pair and BSG d1. less flexible than BSG/sequence pair in representation d2. tree structure is irregular, harder for implementation d3. need to encode and operate on module sequence d4. need to transform between the tree and its placement during processing d5. inserting positions are limited, might deviate from the optimal during solution perturbation
B*-tree	a1. can handle non-slicing structure a2. binary-tree based, efficient a3. flexible to deal with hard, pre-placed, soft, and rectilinear modules, etc a4. smaller encoding cost a5. except for handling soft modules, only need to transform from a tree to its placement during processing, which takes only linear time a6. can evaluate area cost incrementally a7. the solution space is smaller d1. less flexible than BSG/sequence pair in representation

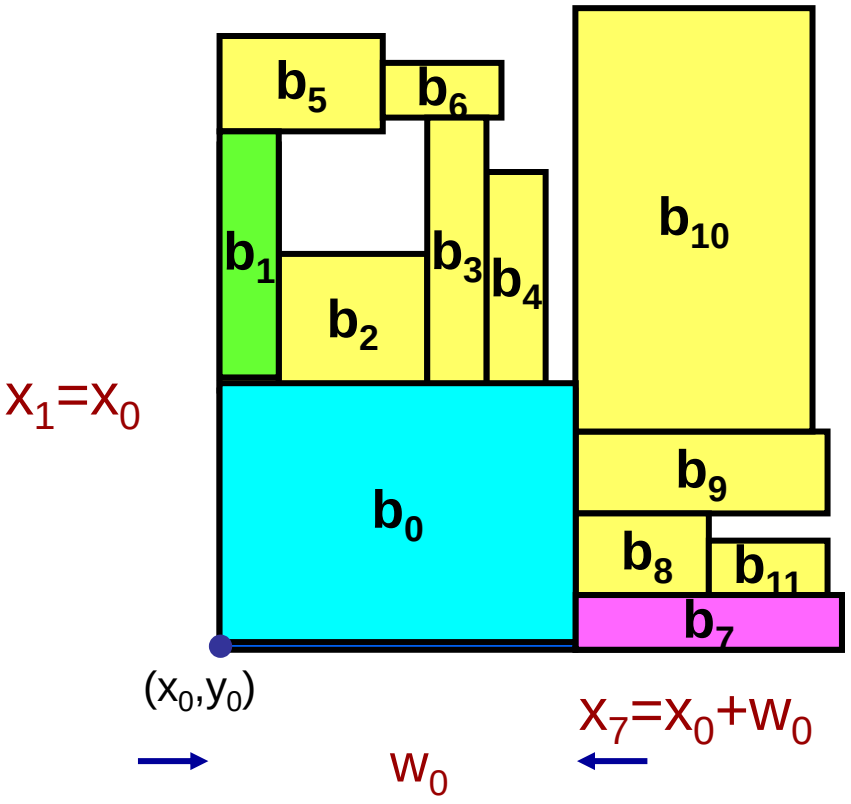
bounded-sliceline grid (BSG) structure

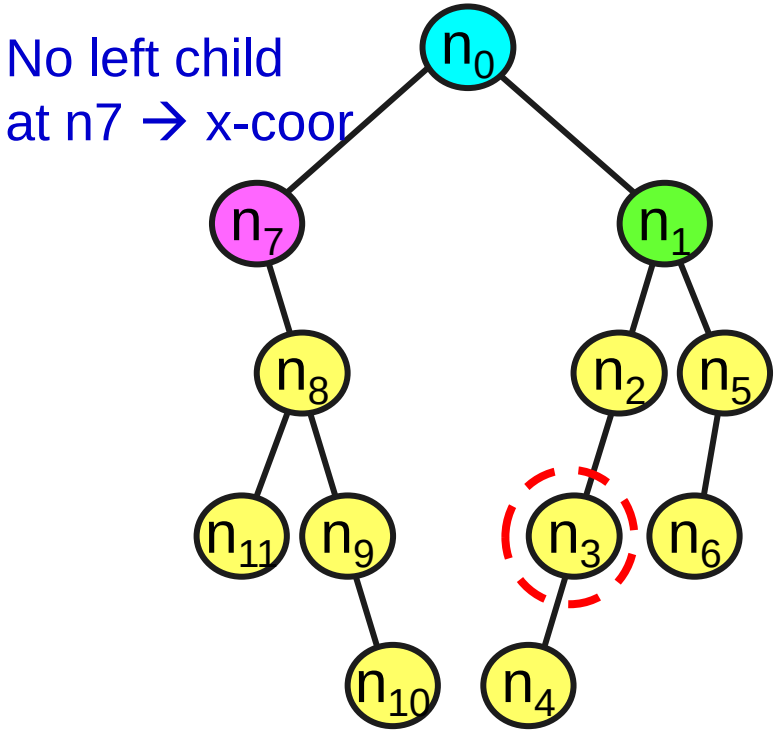
Table 10.6 Comparison among Floorplan Representations

Representation	Solution Space	Packing Time	Flexibility
Normalized Polish Expression	$O(n!2^{3n}/n^{1.5})$	$O(n)$	Slicing
Corner Block List	$O(n!2^{3n})$	$O(n)$	Mosaic
Twin Binary Sequence	$O(n!2^{3n}/n^{1.5})$	$O(n)$	Mosaic
O-tree	$O(n!2^{2n}/n^{1.5})$	$O(n)$	Compacted
B*-tree	$O(n!2^{2n}/n^{1.5})$	$O(n)$	Compacted
Corner Sequence	$\leq (n!)^2$	$O(n)$	Compacted
Sequence Pair	$(n!)^2$	$O(n^2)$	General
BSG	$O(n!C(n^2, n))$	$O(n^2)$	General
Transitive Closure Graph	$(n!)^2$	$O(n^2)$	General
TCG-S	$(n!)^2$	$O(n \lg n)$	General
Adjacent Constraint Graph	$O((n!)^2)$	$O(n^2)$	General

B*-tree Packing

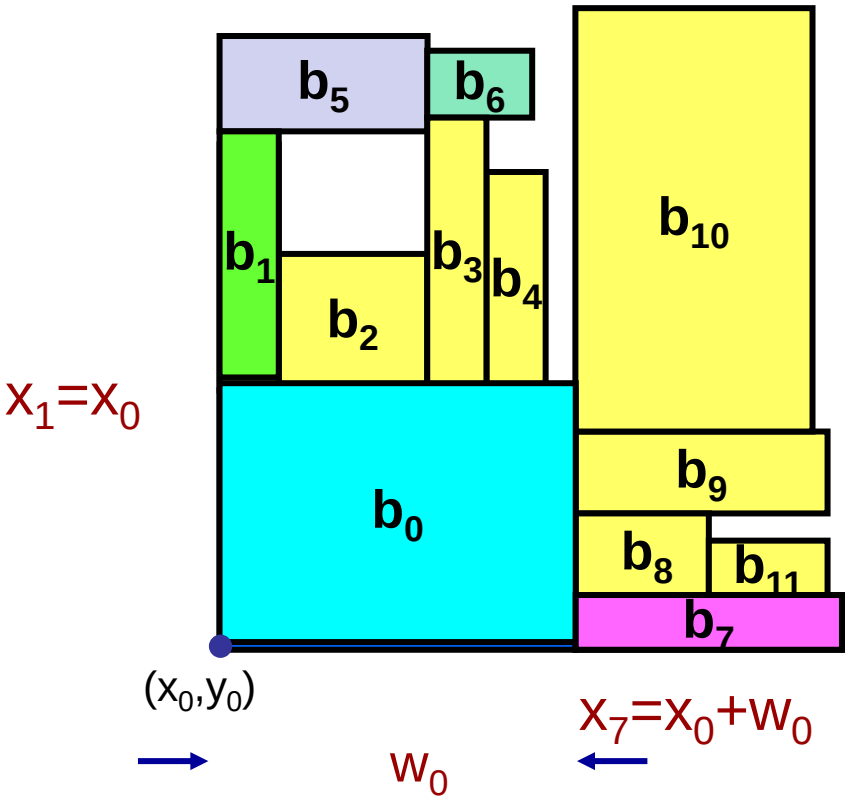
- x-coordinates can be determined by the tree structure.
  - Left child: the lowest, adjacent block on the right ($x_j = x_i + w_i$).
  - Right child: the first block above, with the same *x*-coordinate ($x_j = x_i$).
- y-coordinates?

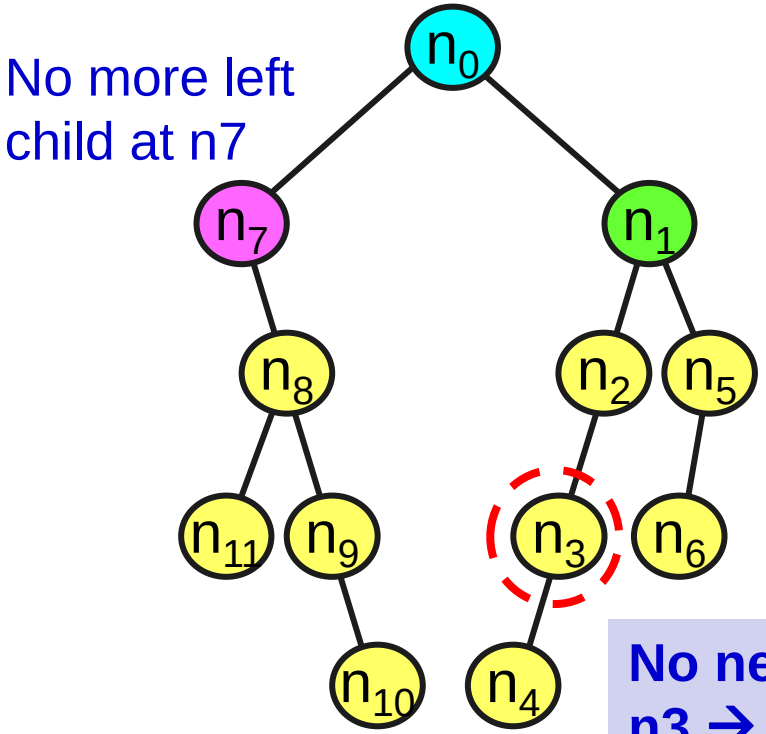




B*-tree Packing

- 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?





No need for n3 → n6 (even x3=x6)

To Improve → Perturbations

- 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.

- There are APIs to update B*-tree quickly for these operations

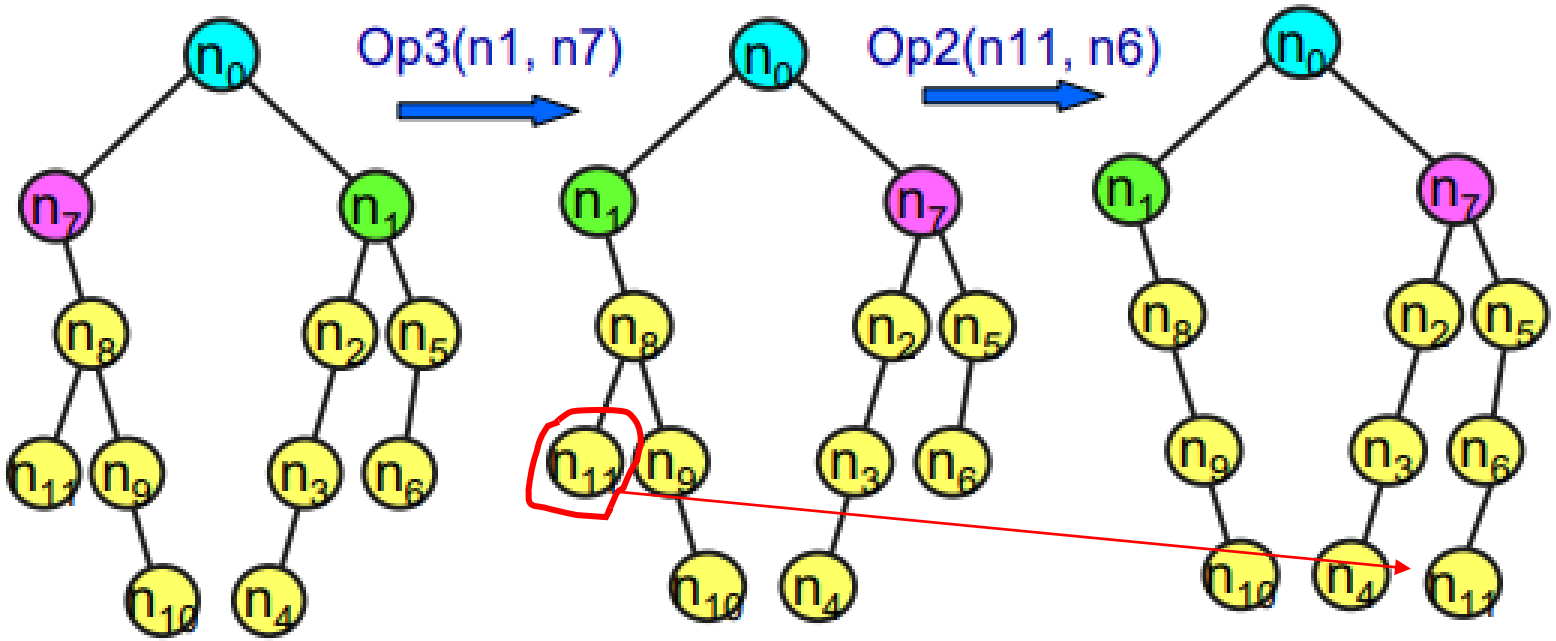
- Also how to handle pre-placed macros? Soft macros?

http://ntur.lib.ntu.edu.tw/bitstream/246246/200611150121534/1/1467.pdf

http://cc.ee.ntu.edu.tw/~ywchang/Papers/boundary-btree.pdf

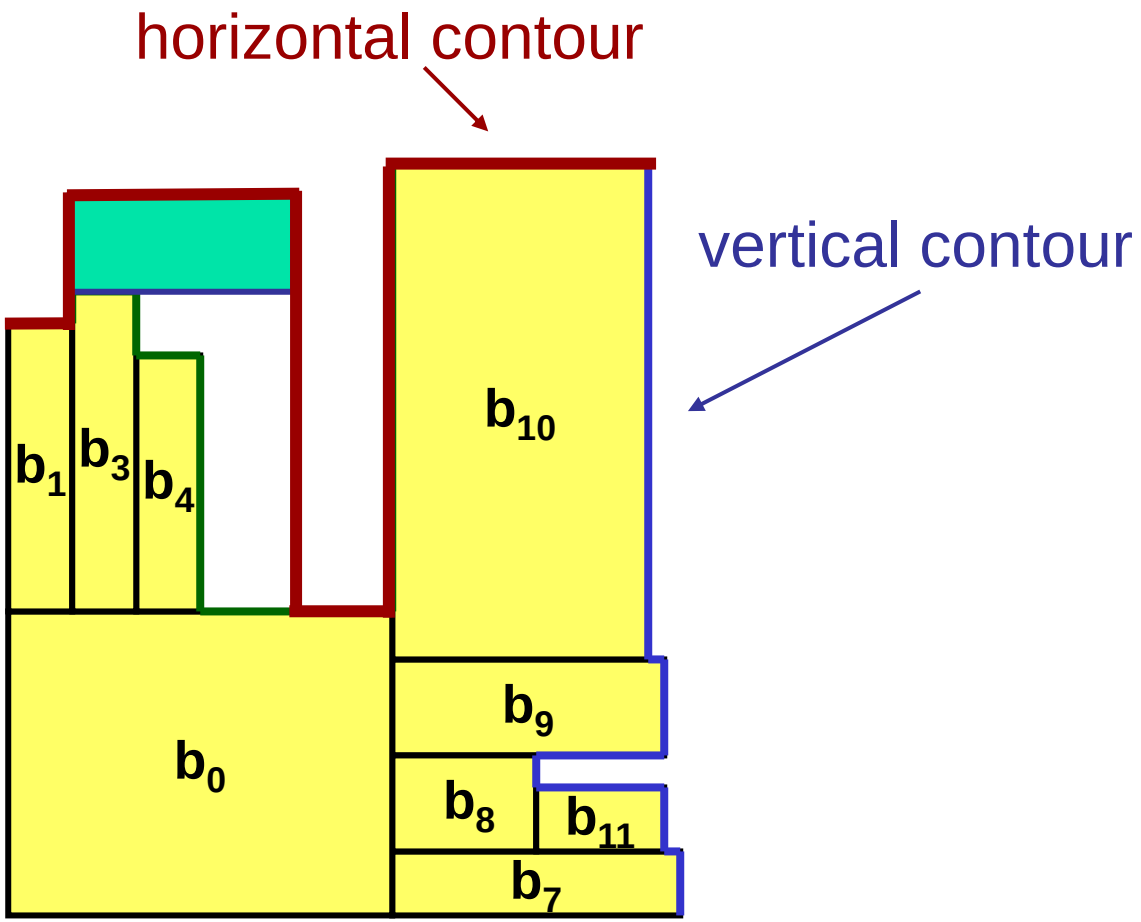
B*-Tree Perturbation

- Op1: rotate a macro
- Op2: delete & insert
- Op3: swap 2 nodes
- Op4: resize a soft macro



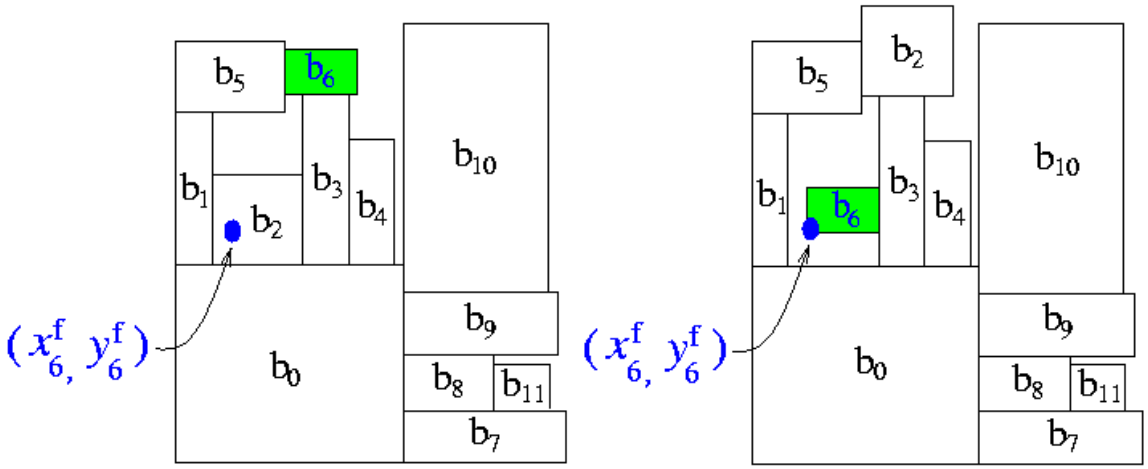
Computing y-coordinates

- the running time for determining the y-coordinate of a newly inserted module would be linear to the number of modules
- Reduce the complexity of computing a y-coordinate to amortized O(1) time.



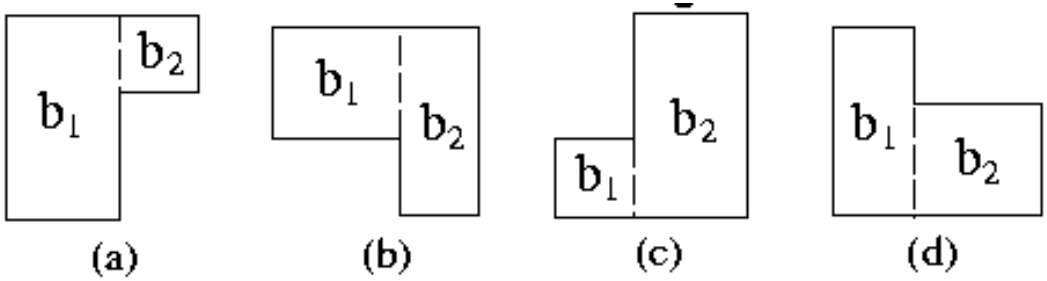
Coping with Pre-placed Modules

- If there are modules ahead or lower than b_i so that b_i cannot be placed at its fixed position (x^f_i, y^f_i), exchange b_i with the module in $D_i = \{b_j \mid (x_j, y_j) \mathbf{<=} (x^f_i, y^f_i)$ that is most close to $(x^f_i, y^f_i)\}$.
- 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.

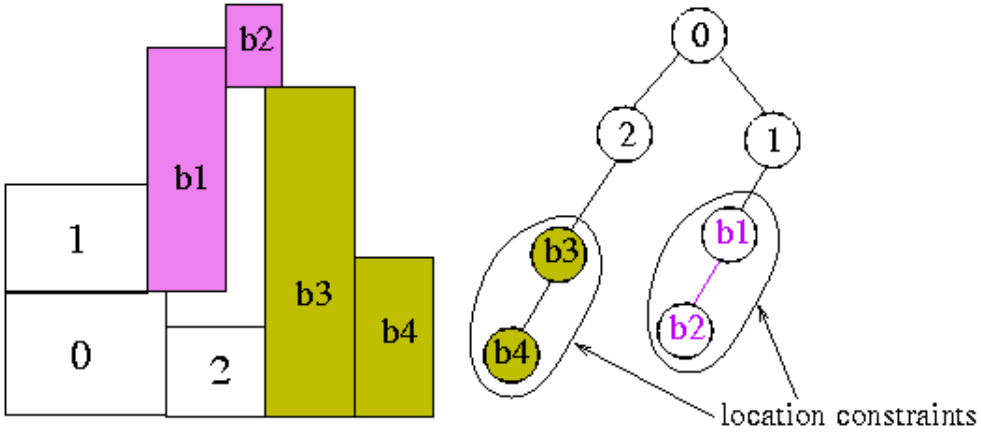


Coping with Rectilinear Modules

- Partition a rectilinear module into rectangular sub-modules.

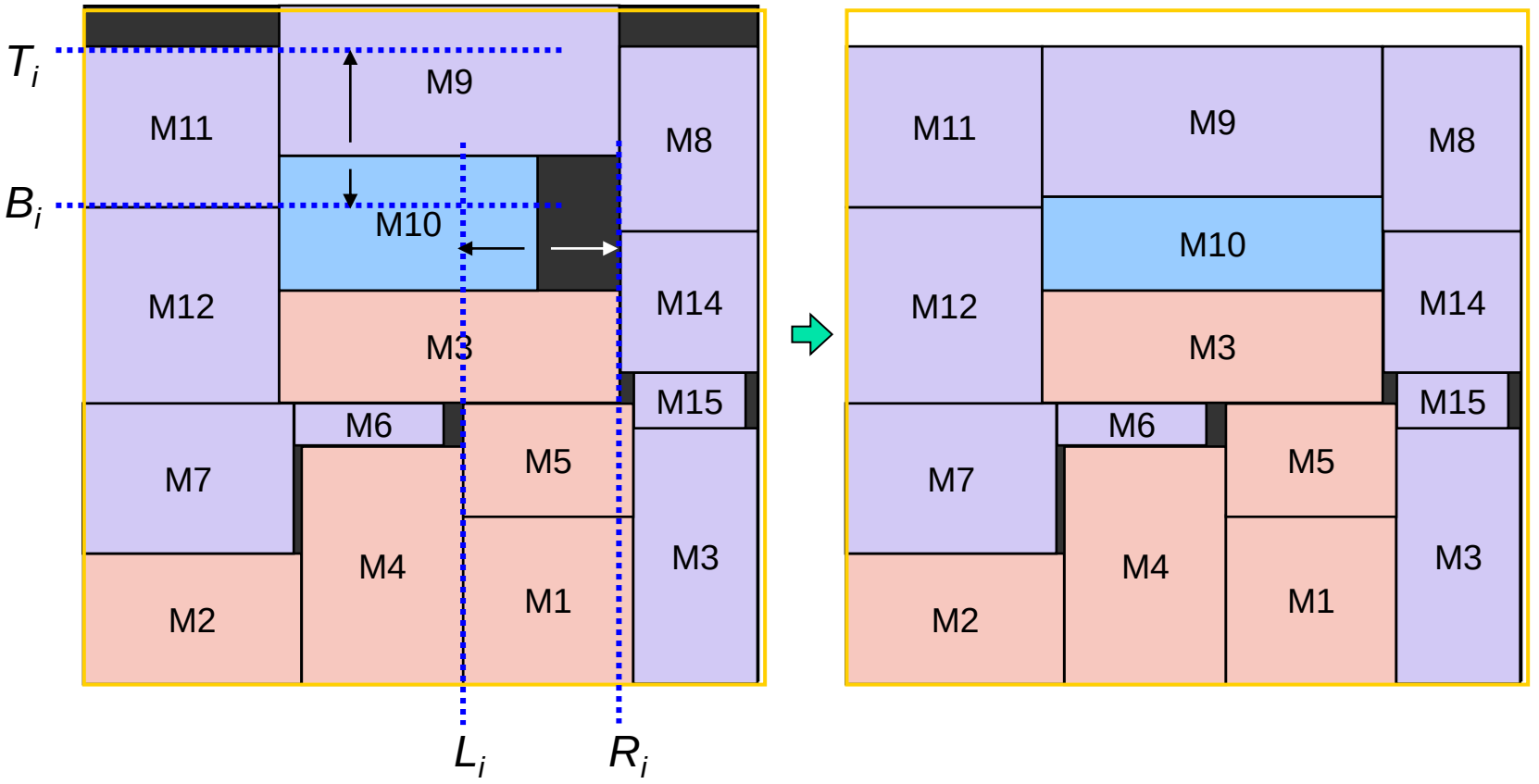


- Keep **location constraints** for the sub-modules.
  - e.g., Keep the right sub-module as the left child in the B*-tree.
- Align sub-modules, if necessary.
- Treat the sub-modules of a module as a whole during processing



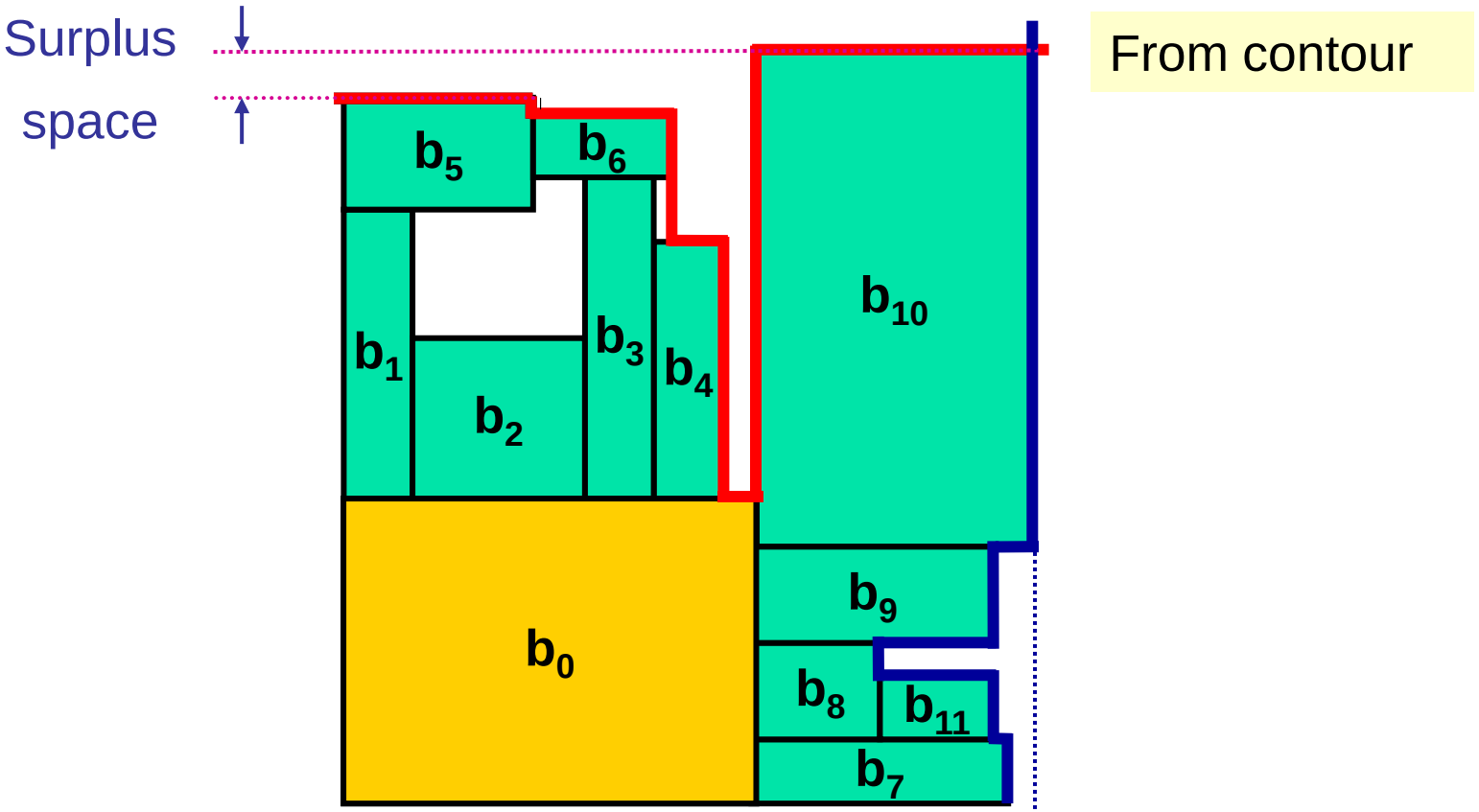
An Easy Way to Cope with Soft Modules

- Change the shape of a soft module to align with adjacent modules.
- Process soft modules one by one.



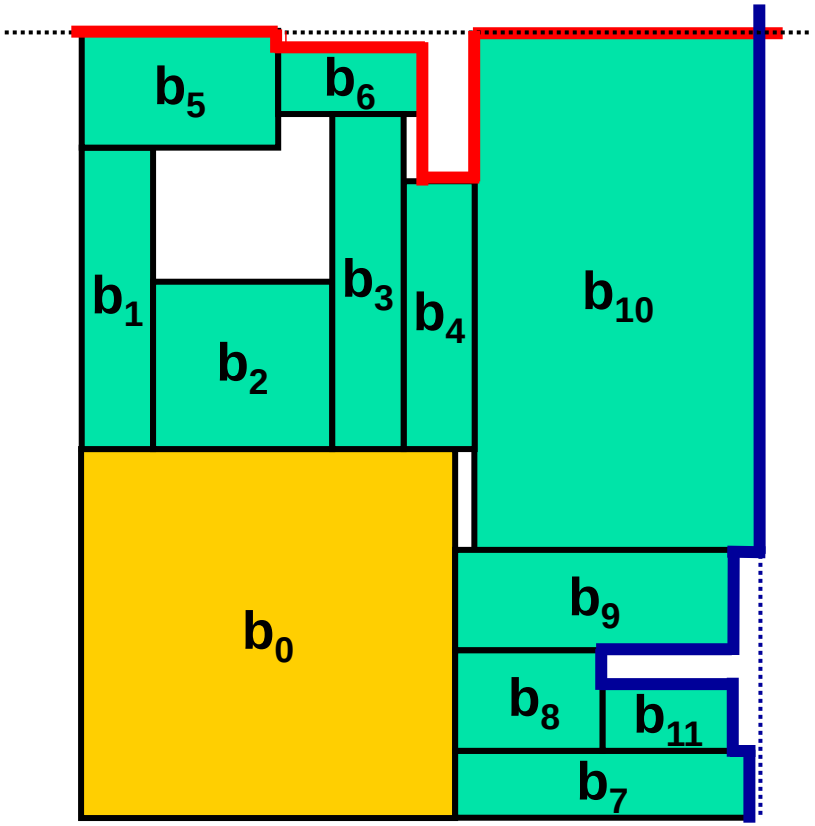
More on Soft Modules

- Step1: Change the shape of the inserted soft module (say, b0).
- Step2: Change the shapes of other soft modules.



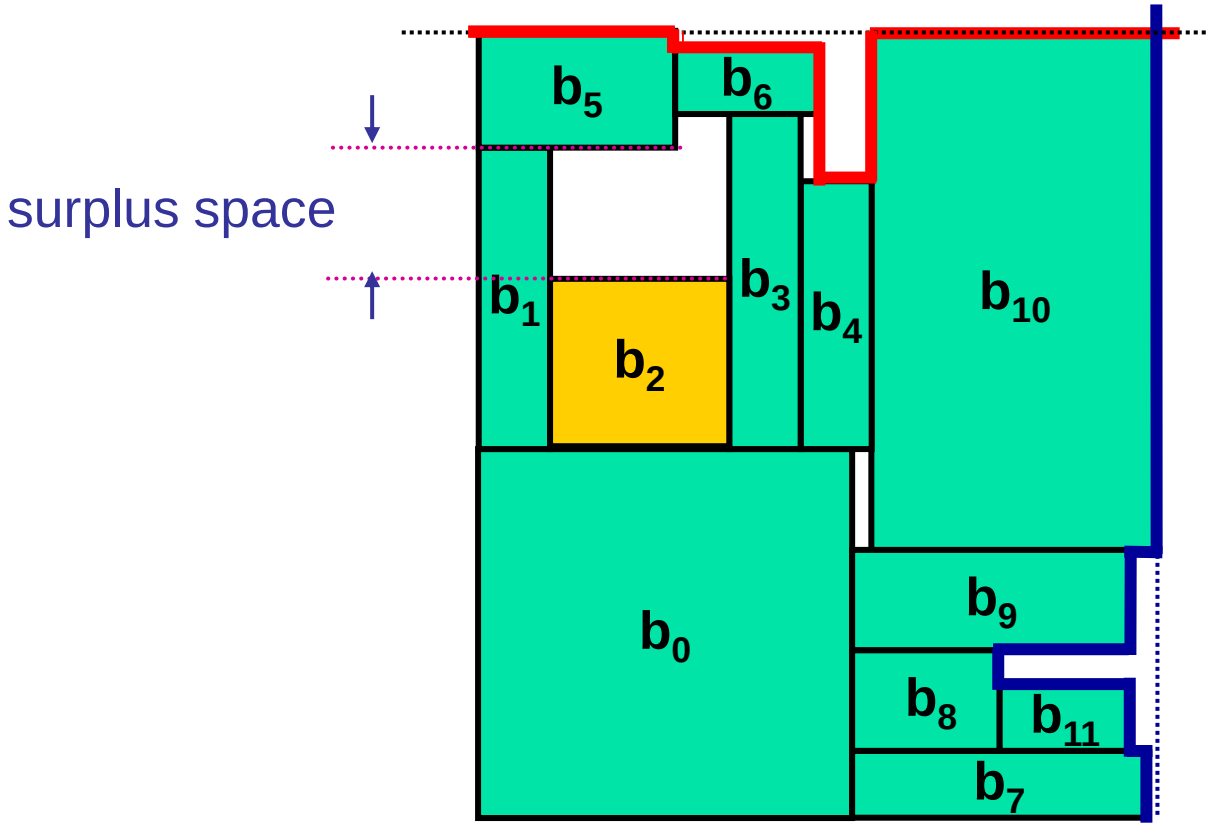
More on Soft Modules

- Step1: Change the shape of the inserted soft module (say, b0)
- Step2: Change the shape of other soft modules



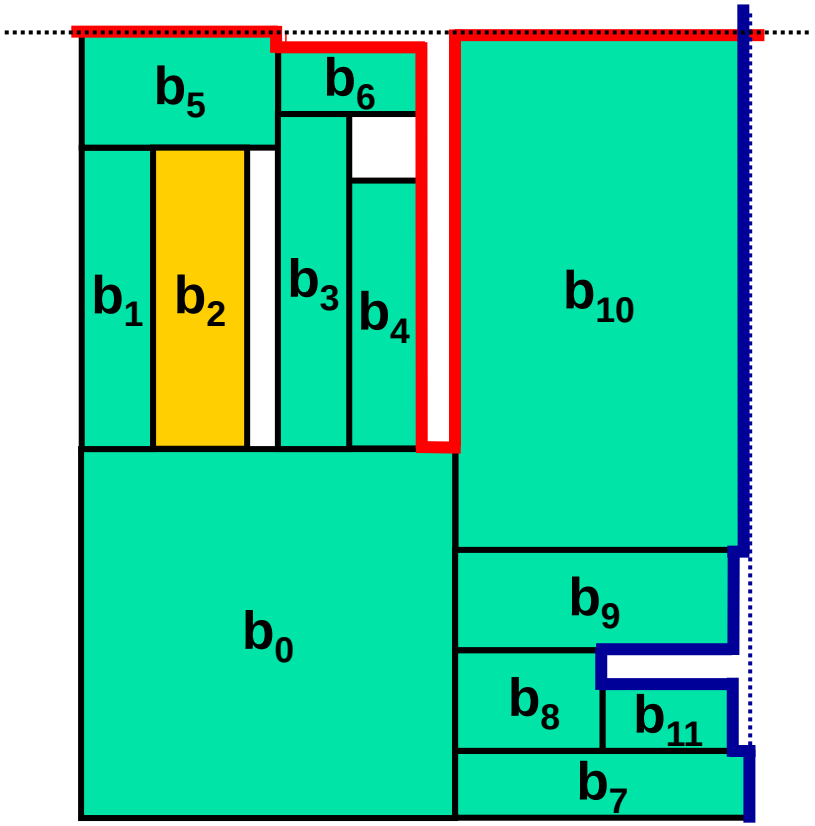
More on Soft Modules

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



More on Soft Modules

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



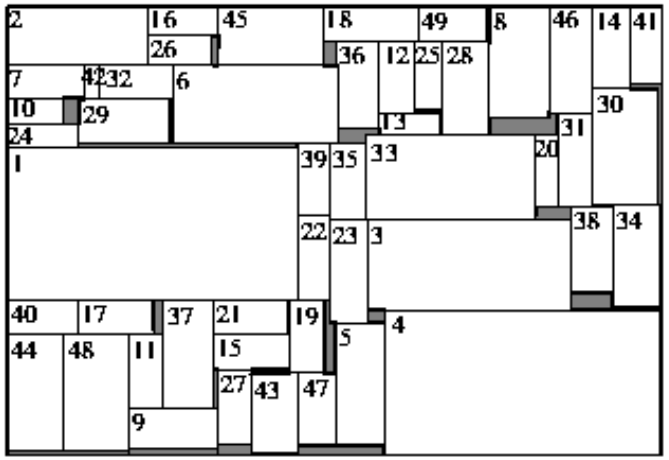
Perturbations & Solutions

- 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.

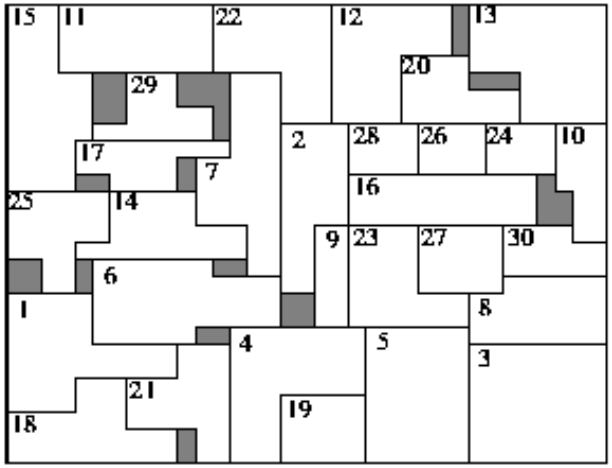
$$\text{Cost} = \alpha \frac{A}{A_{\text{norm}}} + (1-\alpha)\frac{W}{W_{\text{norm}}} \qquad (3)$$

A_{norm} = average area

- ami49: Area = 36.74 *mm*^{2}; dead space = 3.53%; CPU time = 0.25 min on SUN Ultra 60 (optimum = 35.445 *mm*^{2}).



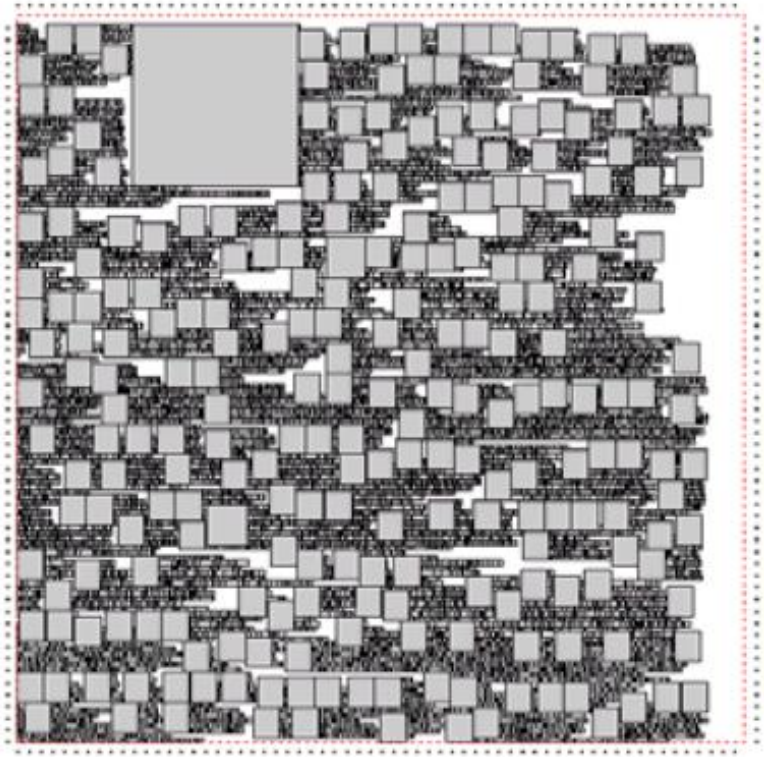
ami49

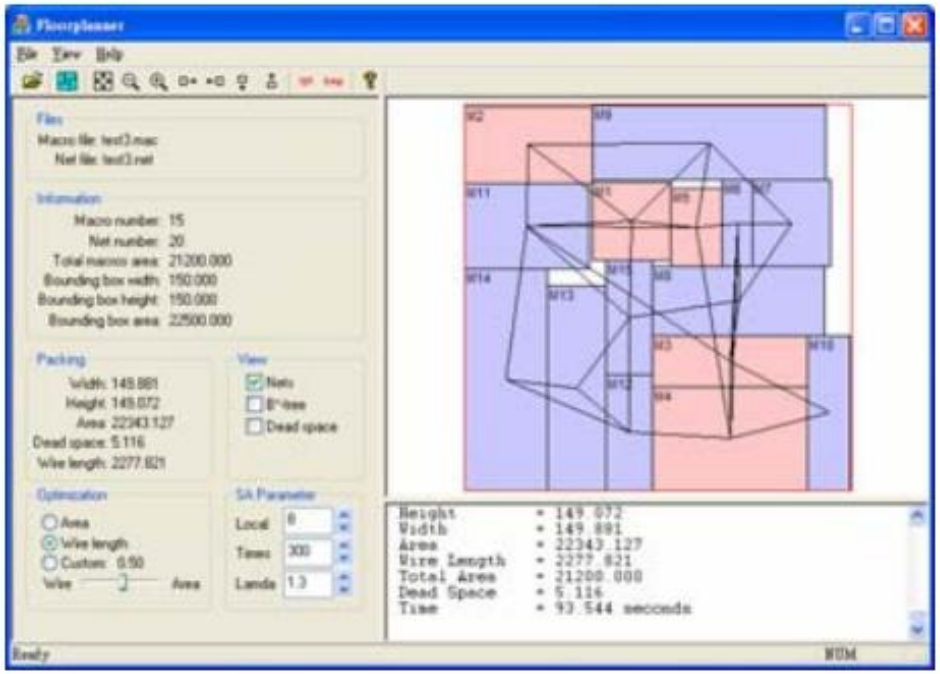


Rectangular, L–, and T–shaped modules

B*-tree Based Floorplanner

- Benchmark: ISPD98 ibm01 with 12,752 cells, 247 macros
  - A_{max}/A_{min} = 8416





Courtesy of Tung-Chieh Chen

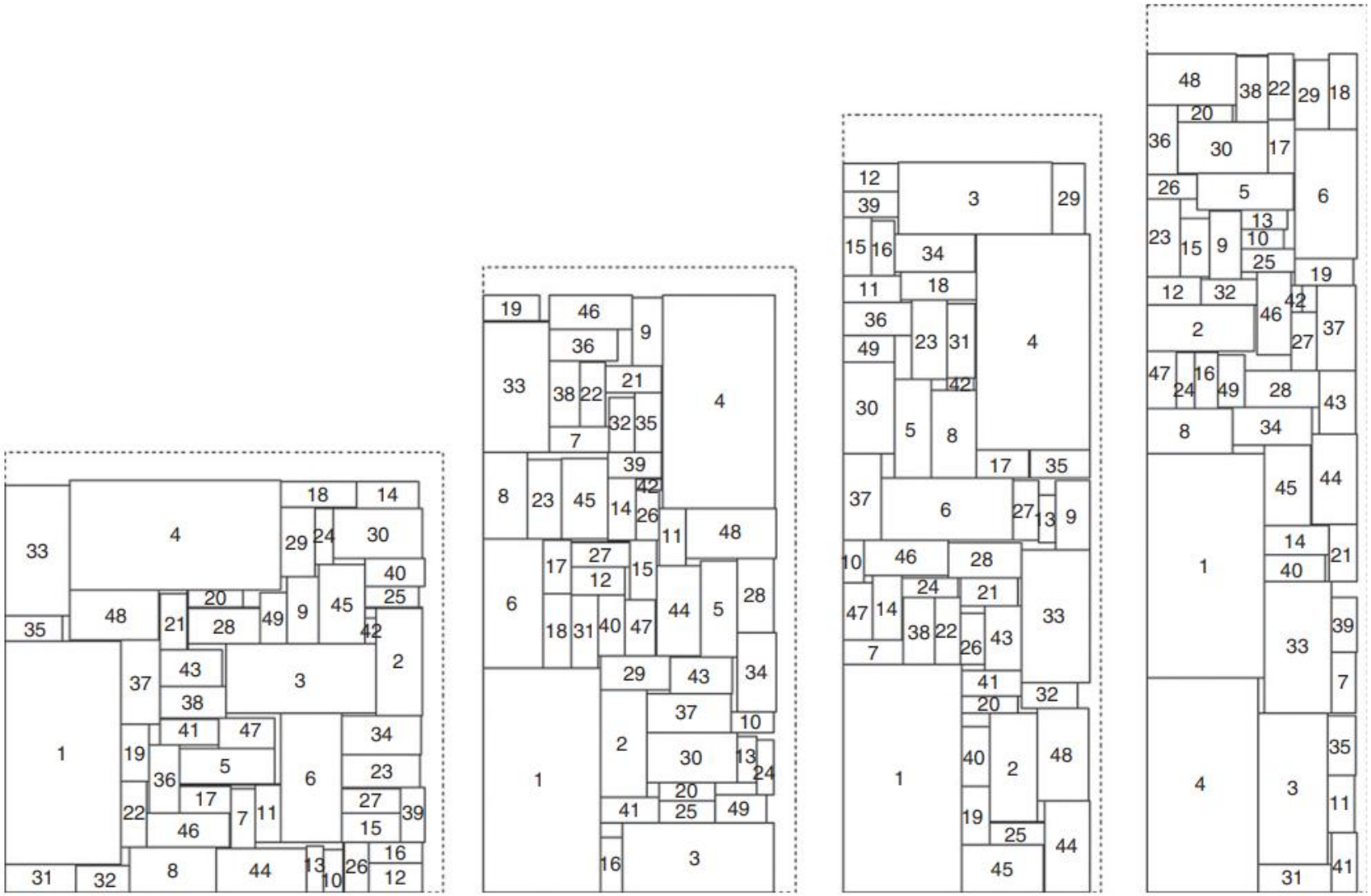


FIGURE 10.35

The floorplans of the MCNC circuit ami49 with fixed-outline ratios 1, 2, 3, and 4.

How About Fixed-Outline Constraints?

For a collection of blocks with the total area A and the given maximum percent of dead space Γ, we construct a fixed outline with the aspect ratio R∗, i.e., height/width. The height H∗ and width W∗ of the outline are defined by

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

We define the cost function Φ for a floorplan solution F by

$$\Phi(F) = \alpha A + \beta W + (1-\alpha-\beta)(R-R^*)^2 \qquad (5)$$

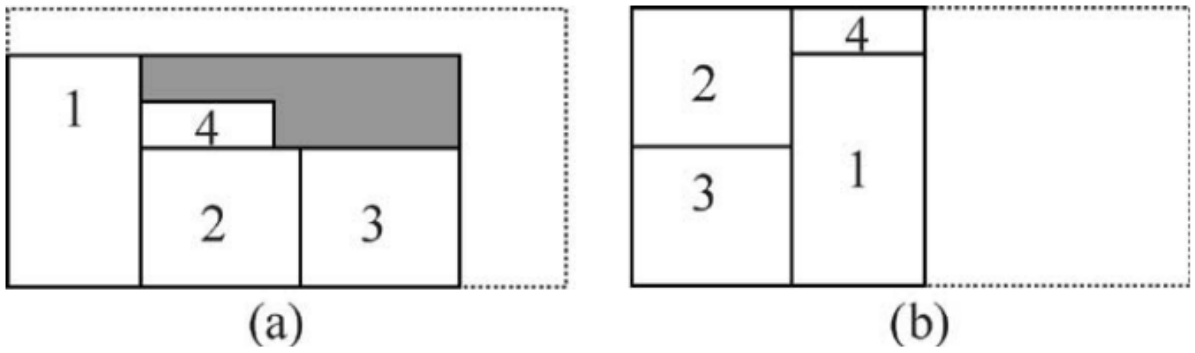
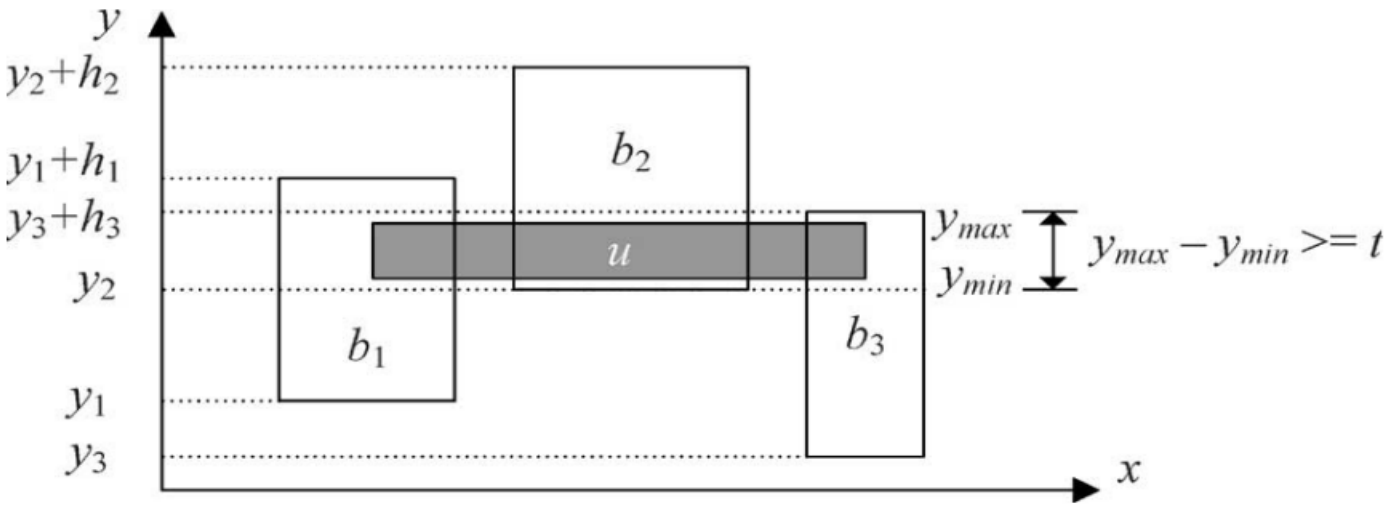


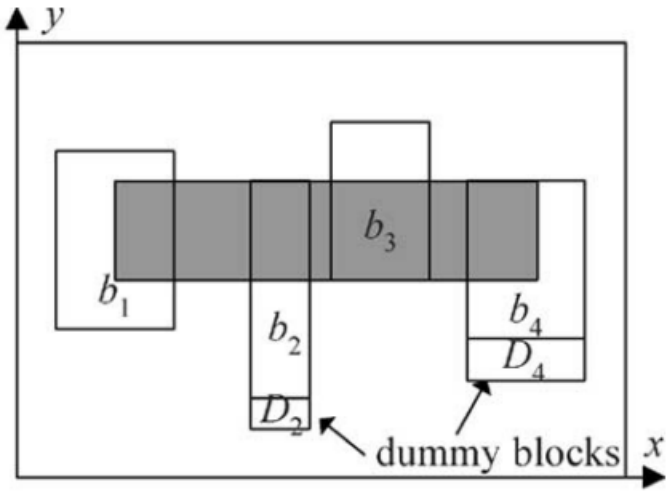
Fig. 3. Two feasible floorplans. Dotted line is fixed outline, and fixed-outline aspect ratio R^* is 0.5. Floorplan aspect ratio R is (a) 0.5 and (b) 1.

How About Buses?

*B∗-tree with the bus constraints (*buses are assigned on the top two layers)



(b) inserting dummy blocks, bus H, t, {b1, b2, b3, b4} is satisfied



(b)

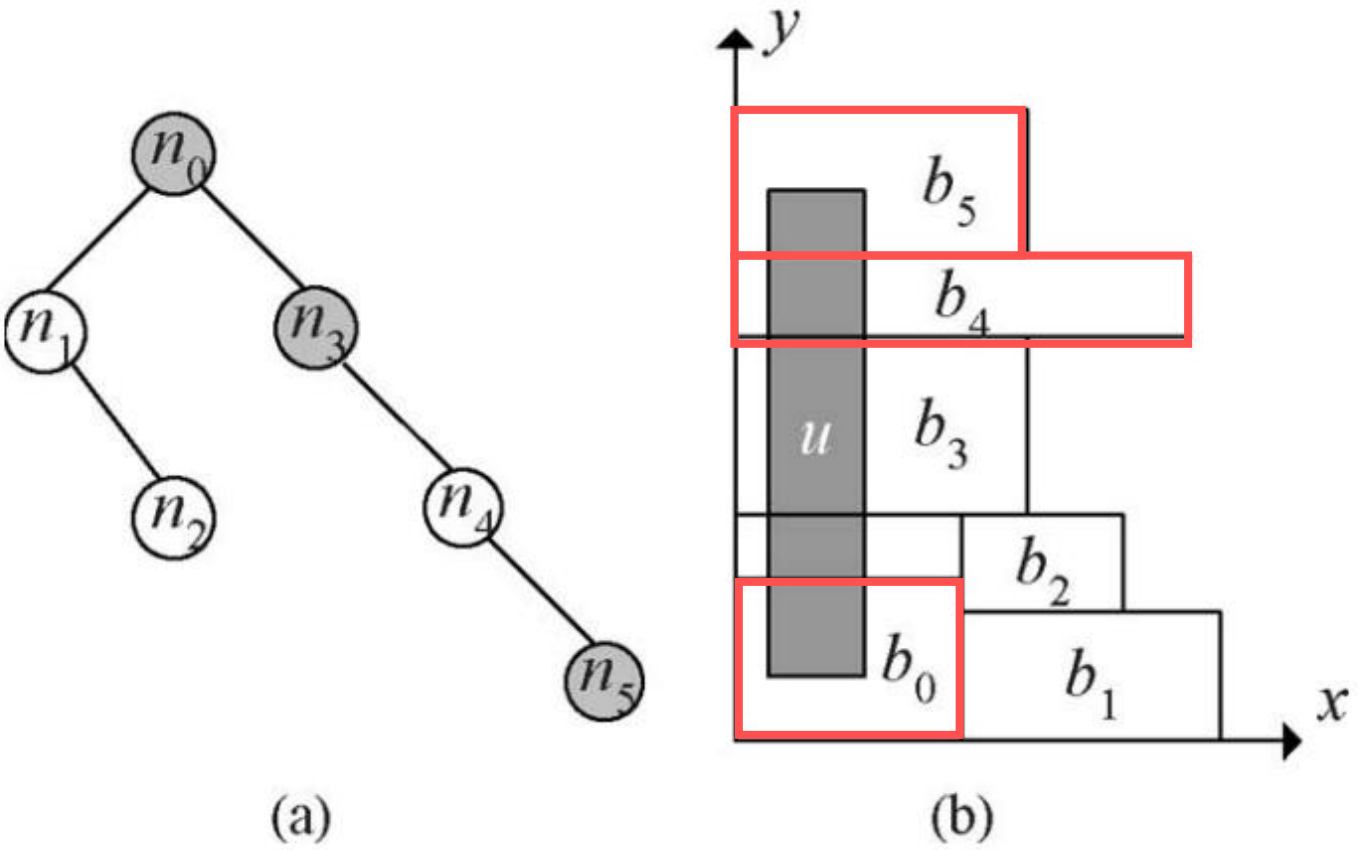


Fig. 8. (a) Right-skewed subtree and (b) corresponding feasible vertical bus $u = \langle V, t, \{b_0, b_4, b_5\} \rangle$.

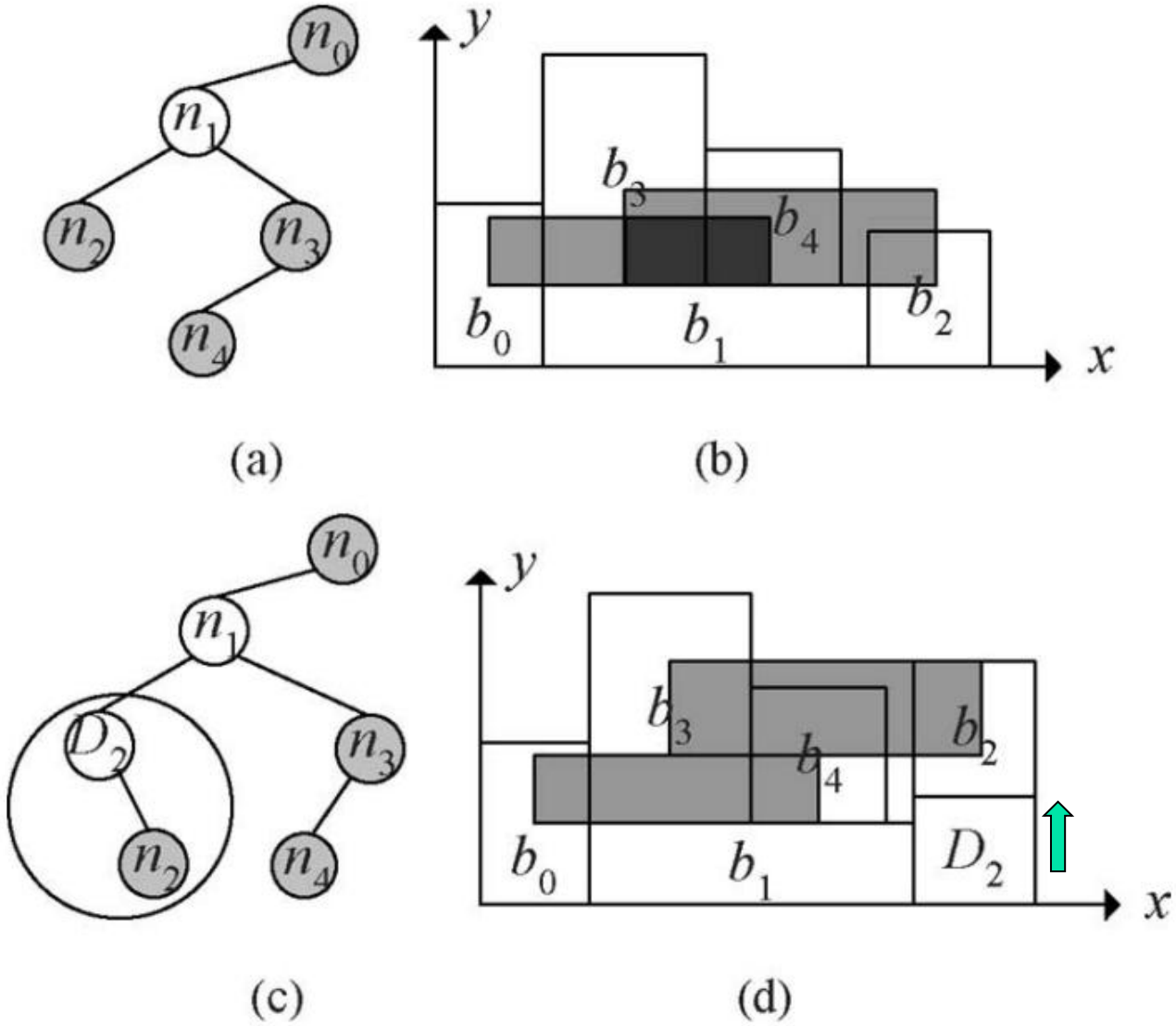
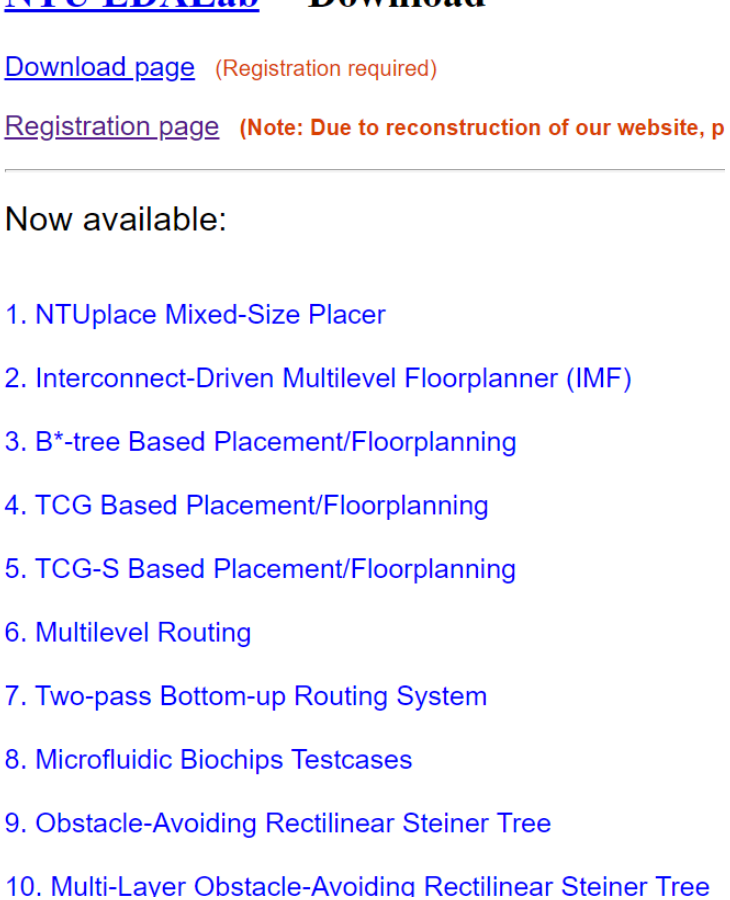


Fig. 10. Two horizontal buses $u = \{b_0, b_4\}$ and $v = \{b_2, b_3\}$. (a) Two buses overlap. (b) By inserting dummy block, we can get feasible floorplan without bus overlapping.

Summary So Far

- Talking about floorplan representations
- Slicing structures
- Representations
- B* tree
  - https://eda.ee.ntu.edu.tw/w04/research.htm

- How about any other Algorithms?



NTU EDALab -- Download

Download page (Registration required)

Registration page **(Note: Due to reconstruction of our website, p**

Now available:

1. NTUplace Mixed-Size Placer
2. Interconnect-Driven Multilevel Floorplanner (IMF)
3. B*-tree Based Placement/Floorplanning
4. TCG Based Placement/Floorplanning
5. TCG-S Based Placement/Floorplanning
6. Multilevel Routing
7. Two-pass Bottom-up Routing System
8. Microfluidic Biochips Testcases
9. Obstacle-Avoiding Rectilinear Steiner Tree
10. Multi-Layer Obstacle-Avoiding Rectilinear Steiner Tree

Analytical Approach

http://cc.ee.ntu.edu.tw/~ywchang/Courses/PD_Source/EDA_floorplanning.pdf

Analytical approach is a mathematical programming formulation that includes an objective function and a set of constraints. For the floorplanning problem, we need to consider two sets of basic constraints:

(1) the module nonoverlapping constraint and

(2) the dimension constraint

Our objective is to minimize the floorplan area, xy

Mixed-Size Placement

- (by Jason Cong et al) Global placement (likely each cell size is fixed or a point, using analytical method to get one),
  - then, remove overlapping,
  - greedy approach to legalize standard cells;
  - then, sliding-window-based cell swapping to reduce wire length
- (by Adya et al) 1st rely on an arbitrary black-box standard-cell placer to obtain an initial placement; then remove overlaps using a fixed-outline floor planner; then, focus on standard cells; (or using force-directed to get initial placement)
- (by Chen et al) multi-packing tree macro placer; an automatic floorplanning algorithm by using dataflow information and design exploration techniques to obtain high quality mixed macro and cell placement
- (by Chang/MediaTek et al) using reinforcement learning for chip placement
  - https://arxiv.org/pdf/2204.06407.pdf

- (paper reading) you can choose floorplanning/placement using ML papers and study/present

https://www.youtube.com/watch?v=zR9IusOpEzk

Google's Chip Designing AI

Google's Paper

https://www.nature.com/articles/s41586-021-03544-w.epdf?sharing_token=tYaxh2mR5EozfsSL0WHZLdRgN0jAjWel9jnR3ZoTv0PW0K0NmVrRsFPaMa9Y5We9O4Hqf_liatg-IvhiVcYpHL_YQpqkurA31sxqtmA-E1yNUWVMMVSBxWSp7ZFFIWawYQYnEXoBE4esRDSWqubhDFWUPyI5wK_5B_YIO-D_kS8%3D
From Google

Article

A graph placement methodology for fast chip design

https://doi.org/10.1038/s41586-021-03544-w

Received: 3 November 2020

Accepted: 13 April 2021

Published online: 9 June 2021

Check for updates

Azalia Mirhoseini[1,4]✉, Anna Goldie[1,3,4]✉, Mustafa Yazgan[2], Joe Wenjie Jiang[1], Ebrahim Songhori[1], Shen Wang[1], Young-Joon Lee[2], Eric Johnson[1], Omkar Pathak[2], Azade Nazi[1], Jiwoo Pak[2], Andy Tong[2], Kavya Srinivasa[2], William Hang[3], Emre Tuncer[2], Quoc V. Le[1], James Laudon[1], Richard Ho[2], Roger Carpenter[2] & Jeff Dean[1]

Chip floorplanning is the engineering task of designing the physical layouts of a computer chip. Despite five decades of research[1], chip floorplanning has defied automation, requiring months of intense effort by physical design engineers to

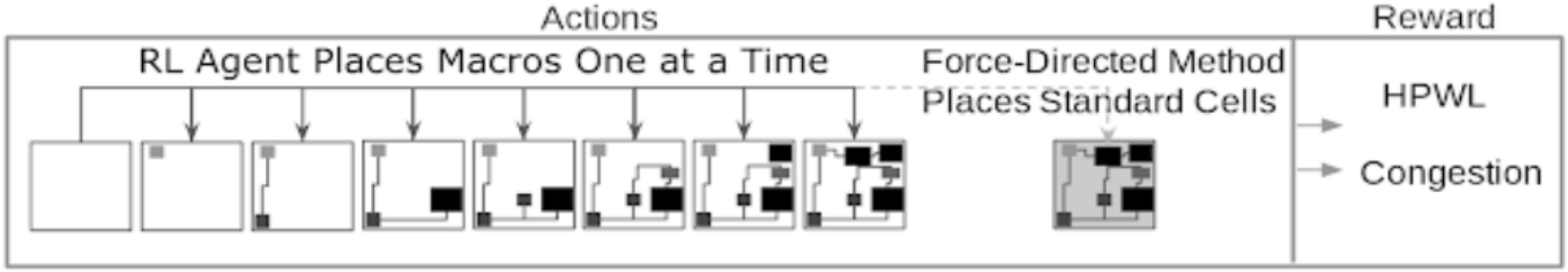
(For TsingHua University only)

Key Takeaways

- Deep reinforcement learning method that outperforms/matches human expert performance on chip floorplanning
- Generates placements in under 6 hours, whereas human-expert baselines take weeks or months at a high operation and opportunity cost
- Superhuman chip floorplans generated by this method were used in Google's latest AI accelerator (TPU)!

So far it is to place macros, not standard cells.
Macros tend to be placed around the chip boundary.

https://ai.googleblog.com/2020/04/chip-design-with-deep-reinforcement.html



During each training iteration, the macros are placed by the policy one at a time and the standard cell clusters are placed by a force-directed method. The reward is calculated from the weighted combination of approximate wirelength and congestion.

In each iteration of training, the macros are sequentially placed by the RL agent, after which the standard cell clusters are placed by a force-directed method, which models the circuit as a system of springs to minimize wirelength. RL training is guided by a fast-but-approximate reward signal calculated for each of the agent's chip placements using the weighted average of approximate wirelength (i.e., the half-perimeter wirelength, HPWL) and approximate congestion (the fraction of routing resources consumed by the placed netlist).

…

For example, the pre-trained policy organically identifies an arrangement that **places the macros near the edges** of the chip with a convex space in the center in which to place the standard cells. This results in lower wirelength between the macros and standard cells without introducing excessive routing congestion.

The Semiconductor Design Space

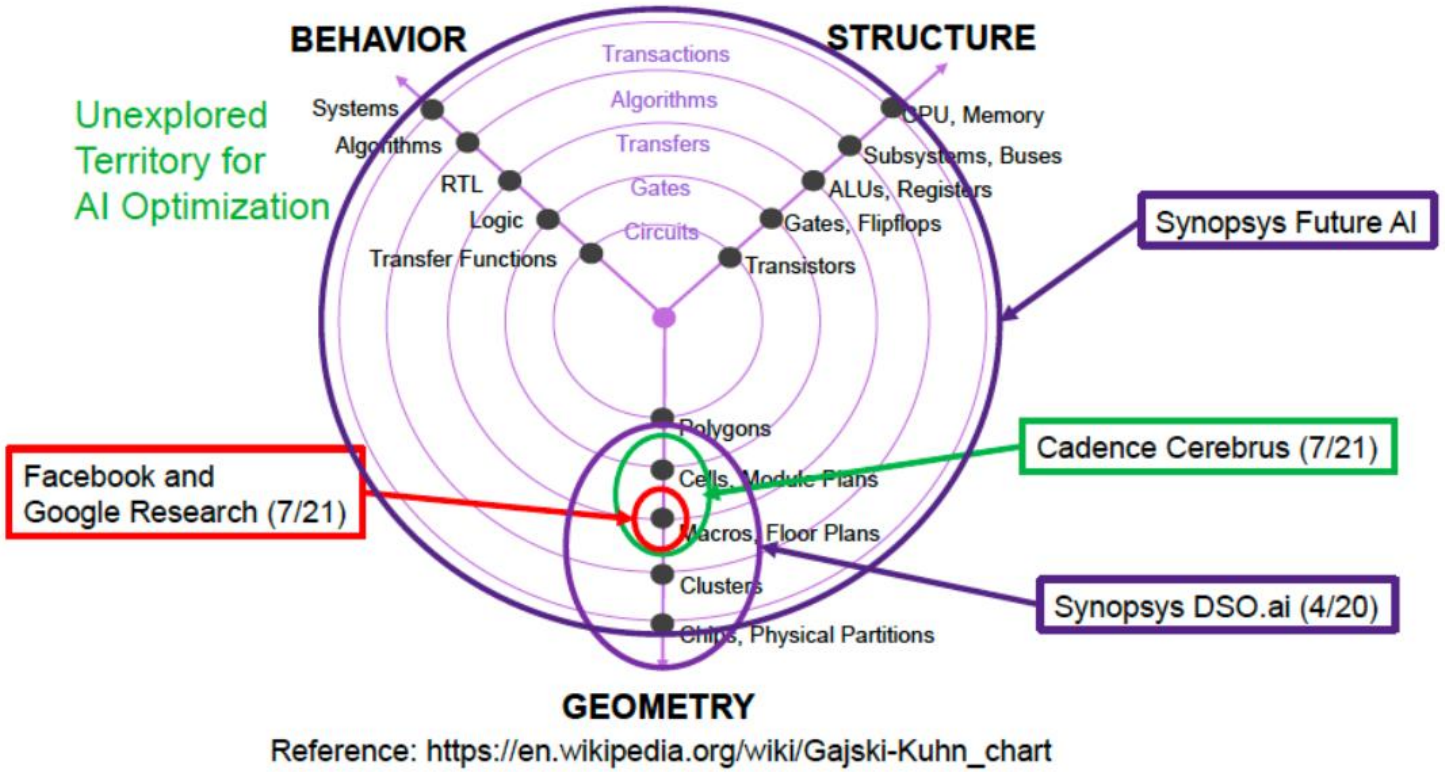
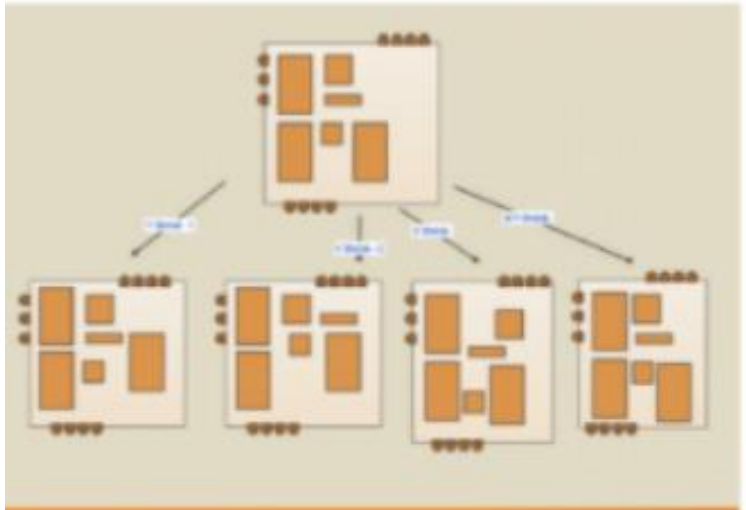


Figure 1: The chip design process aligns along three axes: Behavior (what the chip should do), Structure (how the chip will do it), and Geometry (how the chip is instantiated in masks for production).

Cerebrus: ML for Automated Floorplan Exploration



- Floorplan can be **automatically resized in any direction**
- **Innovus™** mixed placer used to find **optimal macro location in resized floorplan**

Design – CPU Core	
Process	12nm
Performance	2GHz

Customer wanted to achieve 2GHz on latest CPU implementation

Cerebrus optimized floorplan and implementation flow concurrently

Cerebrus Improvements vs Baseline

Parameter	Improvement
Performance	+200MHz
Total failing timing	83%
Leakage power	17%

cadence

We can guess CDNS is using multi-processing to explore several floorplans of various sizes at the same time

https://blogs.synopsys.com/from-silicon-to-software/2022/06/07/chip-floorplan-design-automation/ (June 7, 2022)

GUC Boosts Productivity with Synopsys IC Compiler II Automated FreeForm Macro Placement Technology. It achieved a 14% reduction in switching power and a 19% reduction in wirelength.

The connectivity-driven FreeForm Macro Placement technology is congestion-aware and simultaneously places standard cells and macros for better quality-of-results (QoR). Its coarse placement engine concurrently optimizes wirelength, timing, and power for macros and standard cells.

New automated and machine learning-driven technologies in Synopsys IC Compiler II and Synopsys Fusion Compiler digital implementation solutions streamline floorplan design for better results and productivity.

Ordering Of The Major Topics

- Digital design flow
- CMOS logic gates and Boolean equations
- Algorithms and complexity
- Basic logic synthesis, technology mapping
- Compaction
- Partitioning
- Floorplanning
- Placement (next)
- Routing, P/G wiring, clock tree routing, …
- High level synthesis