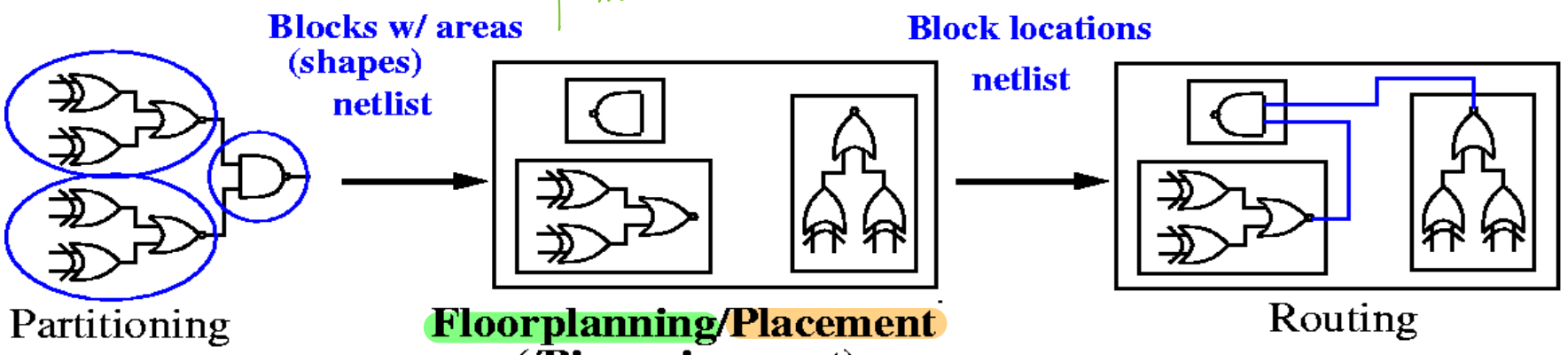
partition 後，"電路被分成"很多個 group"。
(Module, blocks)

Floorplanning

- Partitioning leads to
  - Modules (or called blocks) with well-defined areas and shapes (*hard modules*). ↳事先定義好。Ex: macro
  - Modules with approximated areas and no particular shapes (*soft modules*).
  - A netlist specifying connections between the modules. ↳包含一些 gate，可以算出面積，但形狀還沒確定。
- Objectives ↳在晶片的位置，or擺放的位置
  - Find **locations** for all modules, as well as **orientations (if allowable)** for hard modules. 決定 hard modules 的方位
  - Find shapes (and pin locations) of the soft modules.

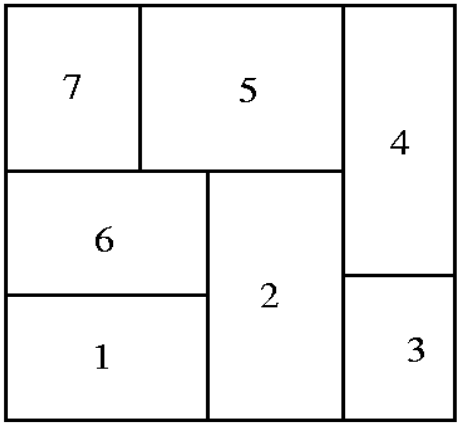


若是 soft module，則可以在 Floorplanning 決定好的形狀、位置內，再去決定 gate 的位置

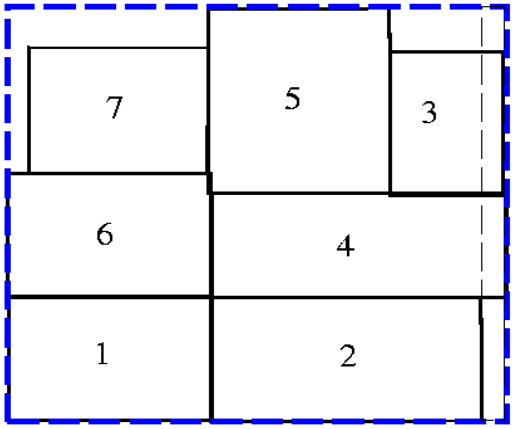
Floorplanning Problem

- Inputs:
  - A set of modules, hard or soft.
  - Pin locations of hard modules.
  - A netlist. ⇒ "net" 連到哪些 modules
- Objectives: Minimize area, reduce wirelength for (critical) nets, maximize routability (minimize congestion), determine shapes of soft modules.

減少"module之間連線的長度"。



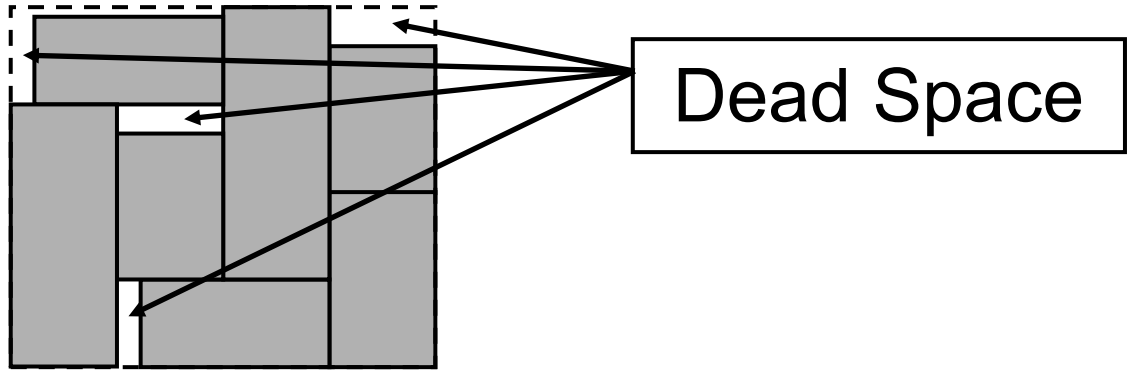
An optimal floorplan, in terms of area



A non-optimal floorplan

Dead Space (white space)

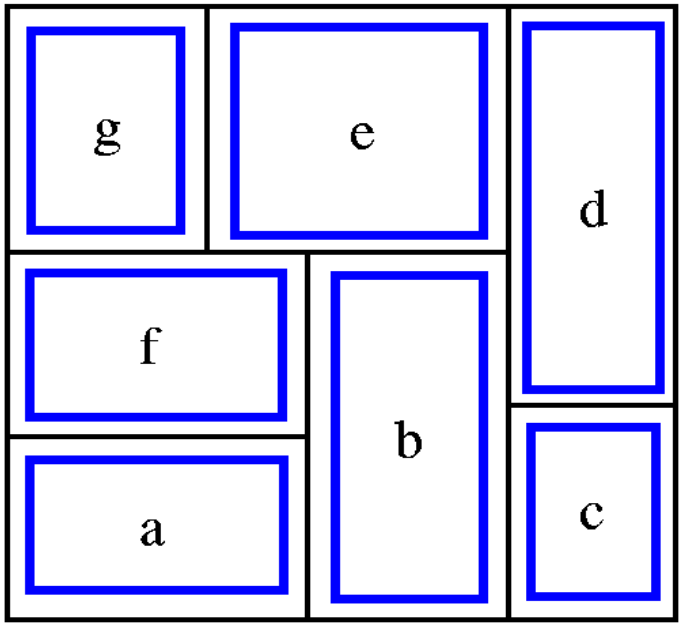
- The space that is wasted.



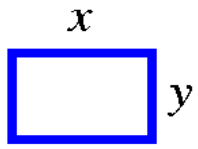
- Minimizing area is the same as minimizing dead space.
- Percentage of dead space

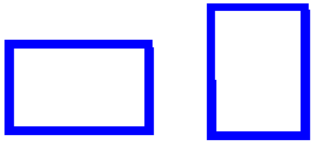
  = ((Area of resulting rectangle / Total area of all modules) – 1) x 100%.

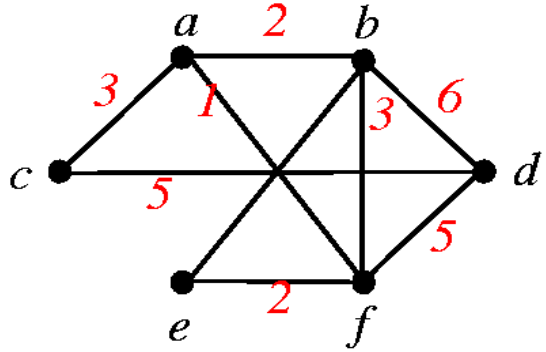
Floorplan Design



- 目標。
  - 找到一個"大矩形"能圍住全部的 module
  - "大矩形"內，會有很多個"小矩形"。
  - "module"要能放在"小矩形內"。

- *Modules:* 

  ⇒ 簡化問題，假設 module 都是"矩形"。
- *Area: A=xy*
  - soft module 才要考慮，因為"寬高會改變"。
- *Aspect ratio: r <= y/x <= s*
  - 問題的 Input
- *Rotation:* 
  ⇒ hard module 要考慮"方位"。
- *Module connectivity*
  - "每個點"是一個 module



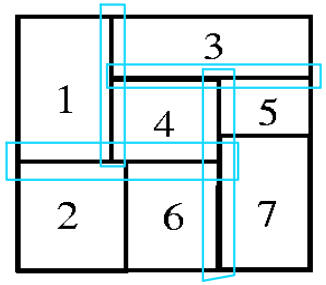
Bounds on Aspect Ratios

- If there is no bounds on aspect ratios, we can always pack modules completely tight (i.e., no dead space).

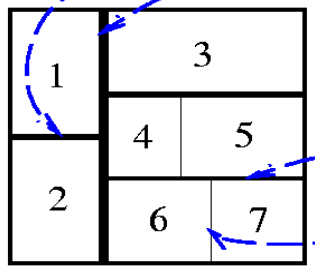
- We do not want to layout a module as a long strip.

Terminology

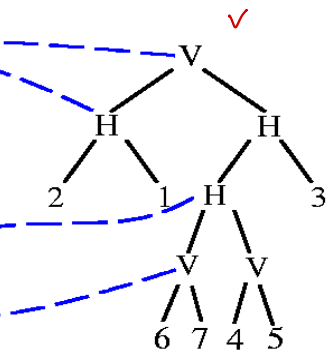
- **Rectangular dissection**: Subdivision of a given rectangle by a finite # of horizontal and vertical line segments into a finite # of non-overlapping rectangles.
- **Slicing structure**: a rectangular dissection that can be obtained by repetitively subdividing rectangles horizontally or vertically.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



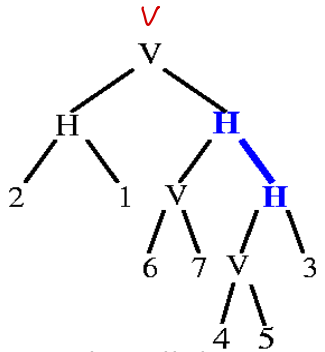
Non–slicing floorplan



Slicing floorplan



A slicing tree (skewed)

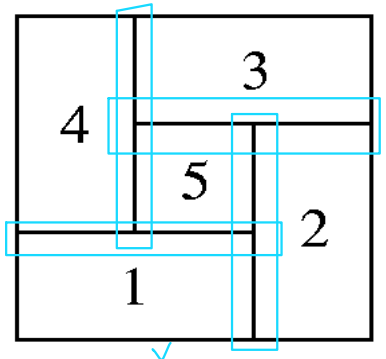


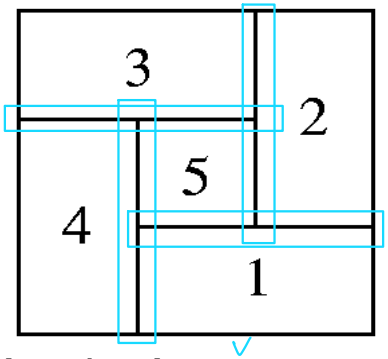
Another slicing tree (non–skewed)

Terminology (cont'd)

↳ 有2種可能的畫法(對稱)

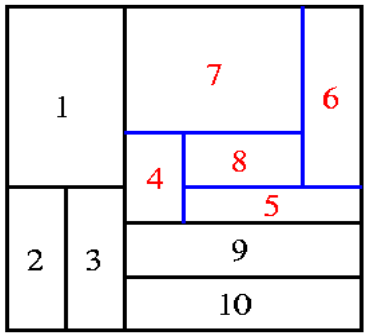
- **Wheel**: The smallest non-slicing floorplans (Wang & Wong, IEEE TCAD'92).
- **Floorplan of order 5.** ⇒ 切割"大矩形"時,同時使用"wheel"和"slicing structure"。
- **Floorplan tree**: A tree representing the hierarchy of partitioning.





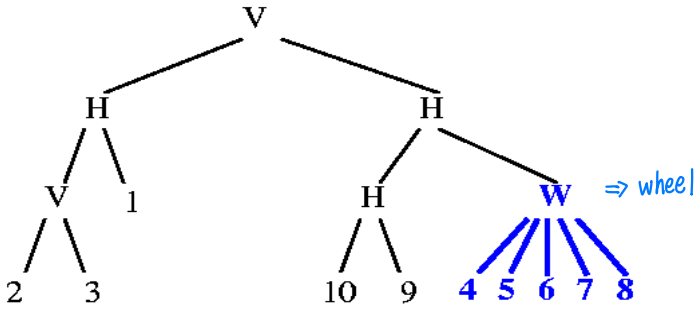
⇒ 用4條線,將"大矩形"分成"5個小矩形"。
(不是用 slicing structure)

The two possible wheels.



A floorplan of order 5

⇒



⇒ wheel

Corresponding floorplan tree

Slicing Floorplan Design by Simulated Annealing

- Related works
  - Wong & Liu, "A new algorithm for floorplan design," DAC'86.
  - Wong, Leong & Liu, *Simulated Annealing of VLSI Design*, pp. 31-51, Kluwer Academic Publishers, 1988.
- Ingredients: solution space, neighborhood structure, cost function, annealing schedule.

用什麼方式"表示答案"。 給了一個"答案",要用什麼機制,產生另一個答案

如何評估"答案的quality"。 ①起始溫度 ②每一個溫度,要產生多少不同的答案 ③如何讓溫度下降 ④結束條件

Solution Representation

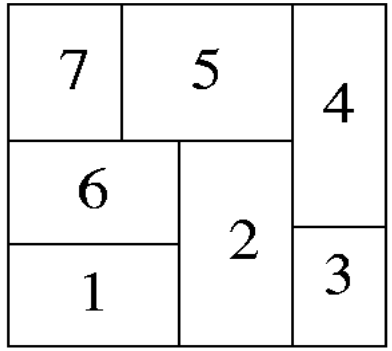
- An expression $E = e_1 e_2 \ldots e_{2n-1}$, where $e_i \in \{1, 2, \ldots, n, H, V\}$, $1 \le i \le 2n-1$, is a **Polish expression** of length $2n-1$ iff

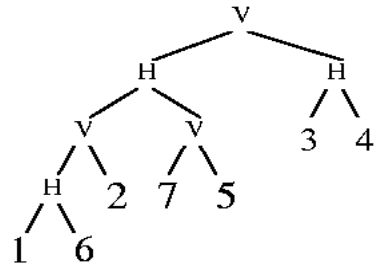
  1. every operand j, $1 \le j \le n$, appears exactly once in E;
  2. (**balloting property**) for every sub-expression $E_i = e_1 \ldots e_i$, $1 \le i \le 2n-1$, #operands > #operators.

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

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

- Polish expression ↔ Postorder traversal.
- *ijH*: *i* below *j*; *ijV*: *i* on the left of *j*.



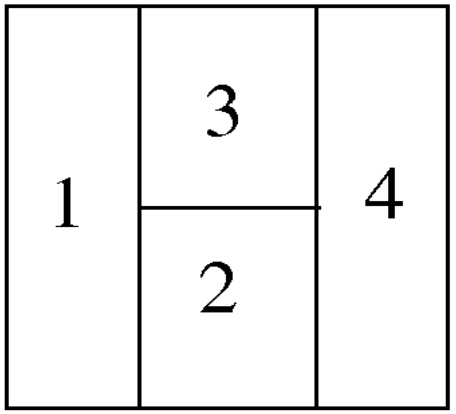


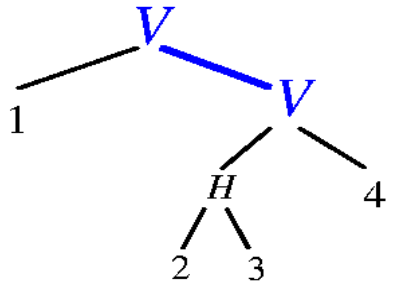
E = 16H2V75VH34HV

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

Postorder traversal of a tree!

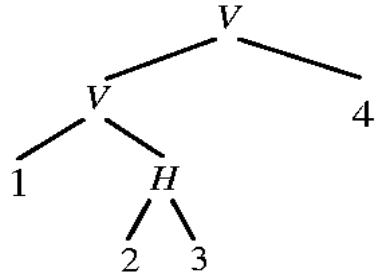
Solution Representation (cont'd)





$E = 123H4VV$

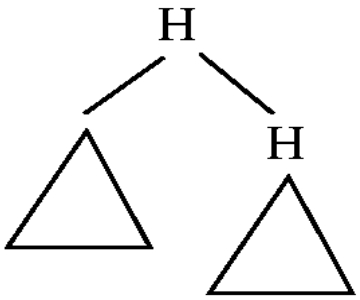
non−skewed!



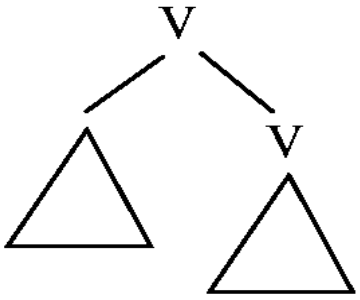
$E = 123HV4V$

skewed!

Non−skewed cases



..... HH .....



..... VV .....

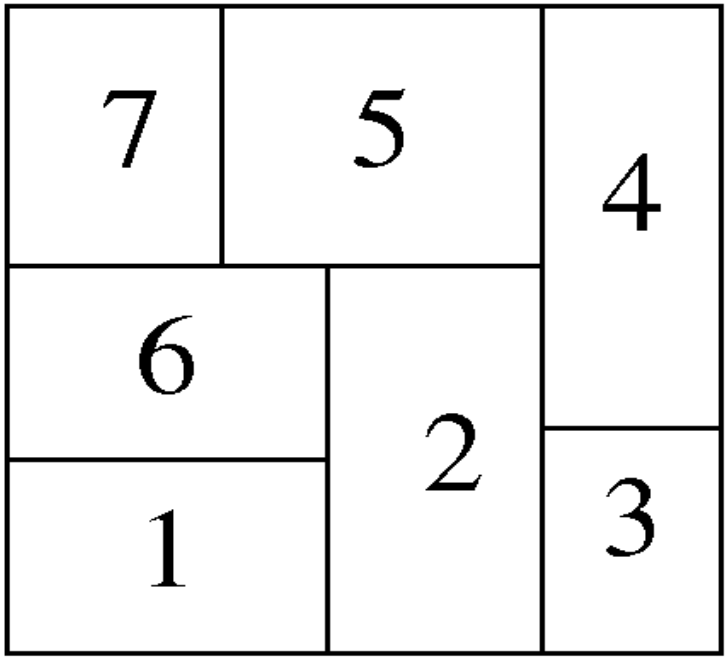
- **Question**: how to eliminate redundant representations?

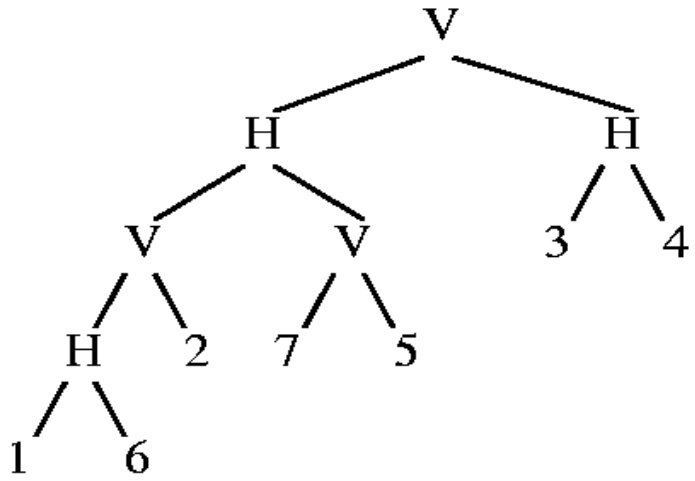
"不想要"一個 pattern，有多個答案"。

不同的 Normalized Polish Expression，可以組成 solution space

Normalized Polish Expression (NPE)

- A Polish expression $E = e_1e_2\ldots e_{2n-1}$ 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. ⇒ 一個 slicing structure，對應一個 Normalized Polish Expression





$E = 16H2V75VH34HV$

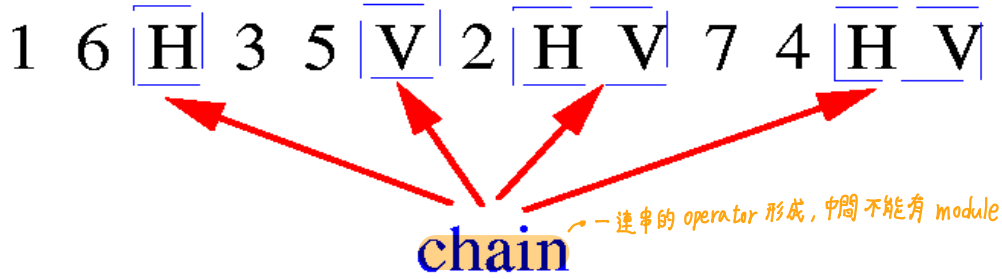
A normalized Polish expression

此部份，只知道"不同 module 的空間分佈"，還沒考慮"module 的寬高"。

給一個 NPE，產生另一個 NPE

Neighborhood Structure

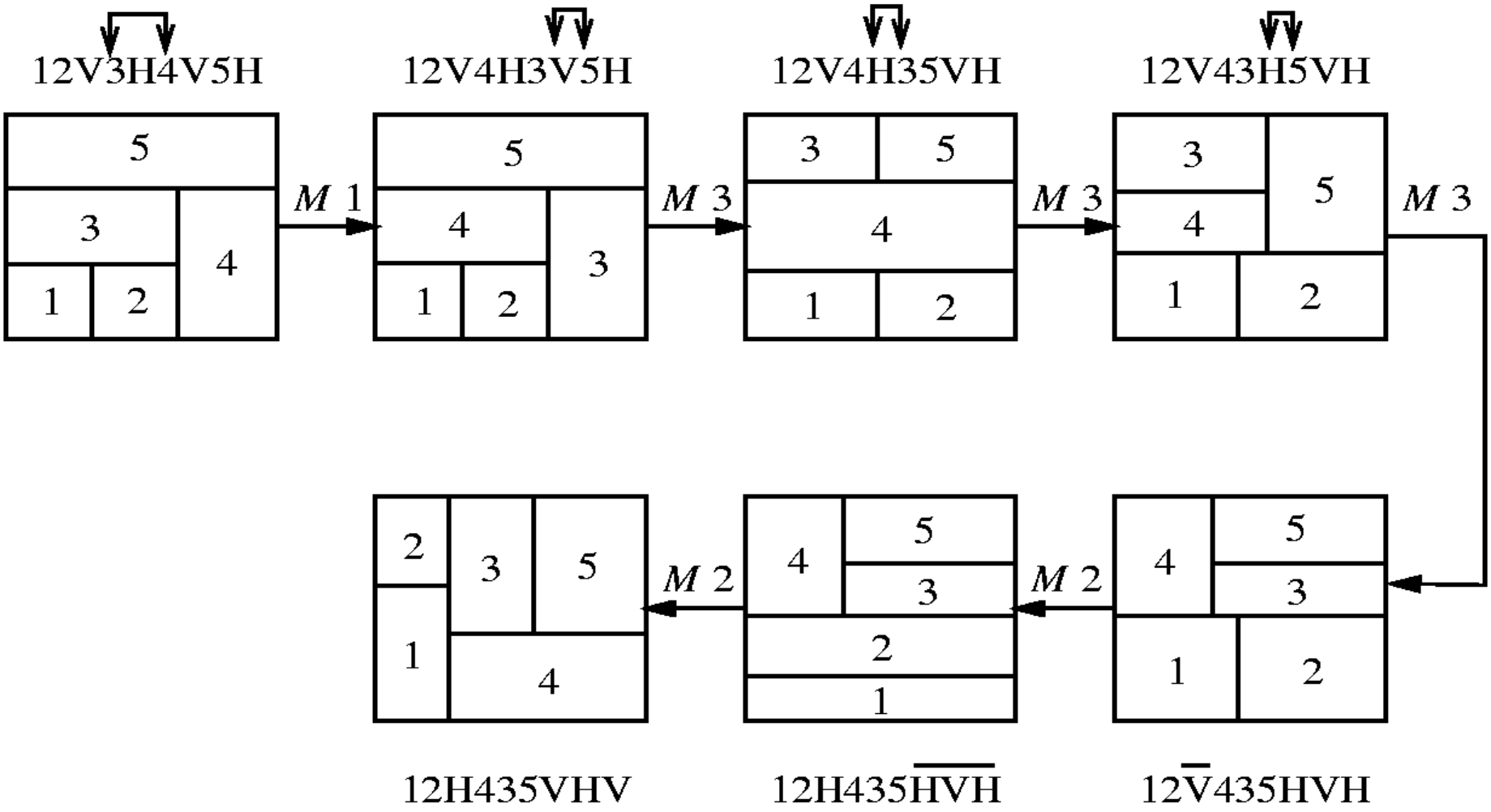
- **Chain**: *HVHVH*…or *VHVHV*…



- **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: ⇒ 3種方式，從一個 NPE，產生另一個 NPE
  - M_1 (**Operand Swap**): swap two adjacent operands.
  - M_2 (**Chain Invert**): Complement a chain. $(\overline{V} = H, \overline{H} = V)$
  - M_3 (**Operator/Operand Swap**): Swap two adjacent operand and operator. 產生的 Polish Expression，不一定是 NPE，要檢查。
- It can be proved that each normalized Polish expression can be obtained from any other one through a finite set of moves of the above three types.

有機會找到"最佳的 NPE"。

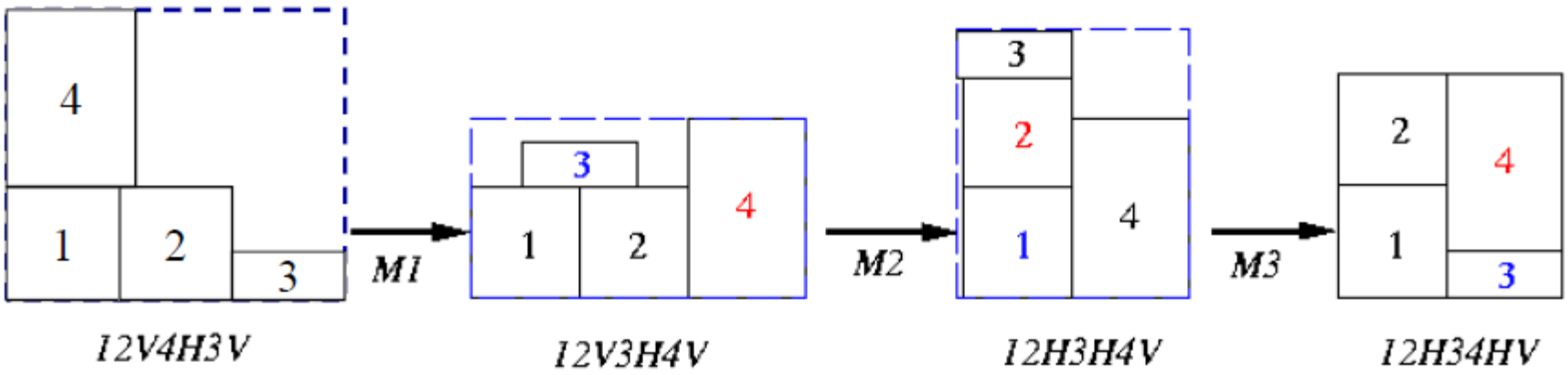
Solution Perturbation



此部份，有考慮"不同 module 的空間分佈"，且有考慮"module 的寬高"。
(不同 module 的大小都固定了)

透過 M_1, M_2, M_3，從一個 NPE，產生另一個 NPE

Effects of Perturbation



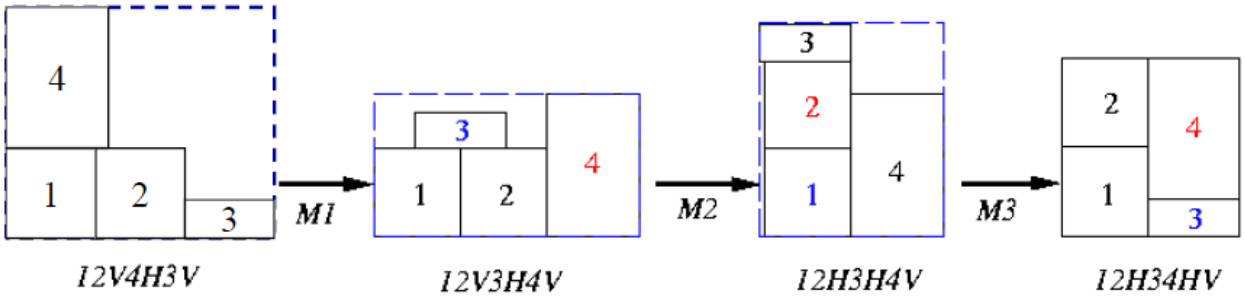
- **Question**: Does the balloting property hold during the moves?
  - M_1 and M_2 moves are OK.
  - Check the M_3 moves! Reject "illegal" M_3 moves.
- **Check M_3 moves**: Assume that the M_3 move swaps the operand e_i with the operator e_{i+1}, $1 \le i \le 2n-2$. Then, the swap will not violate the balloting property iff $2N_{i+1} < i$.
  - N_k: # of operators in the Polish expression $E = e_1 e_2 \ldots e_k$, $1 \le k \le 2n-1$.

Cost Function

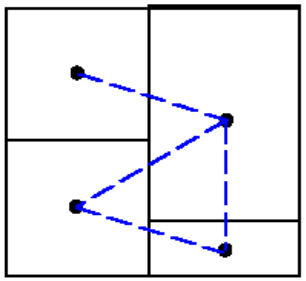
(評估"得到的NPE", 品質好or不好)

- $\Phi = A + \lambda W$
  - A: area of the smallest rectangle
  - W: overall wiring length
  - λ: user-specified parameter ⇒用來選擇比較在意"A" or "W"。

給一個 NPE，會對應到一個 slicing structure，
但是可能會對應到很多不同的 floorplain，要找"面積最小的 floorplain"。
↳"小 rectangle 放的位置，可能會再上下偏移，而且"module 放的方位也可以不同"。
↳可以知道"小 rectangle 的長寬，位置"。
↳然後就能算"W"。



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



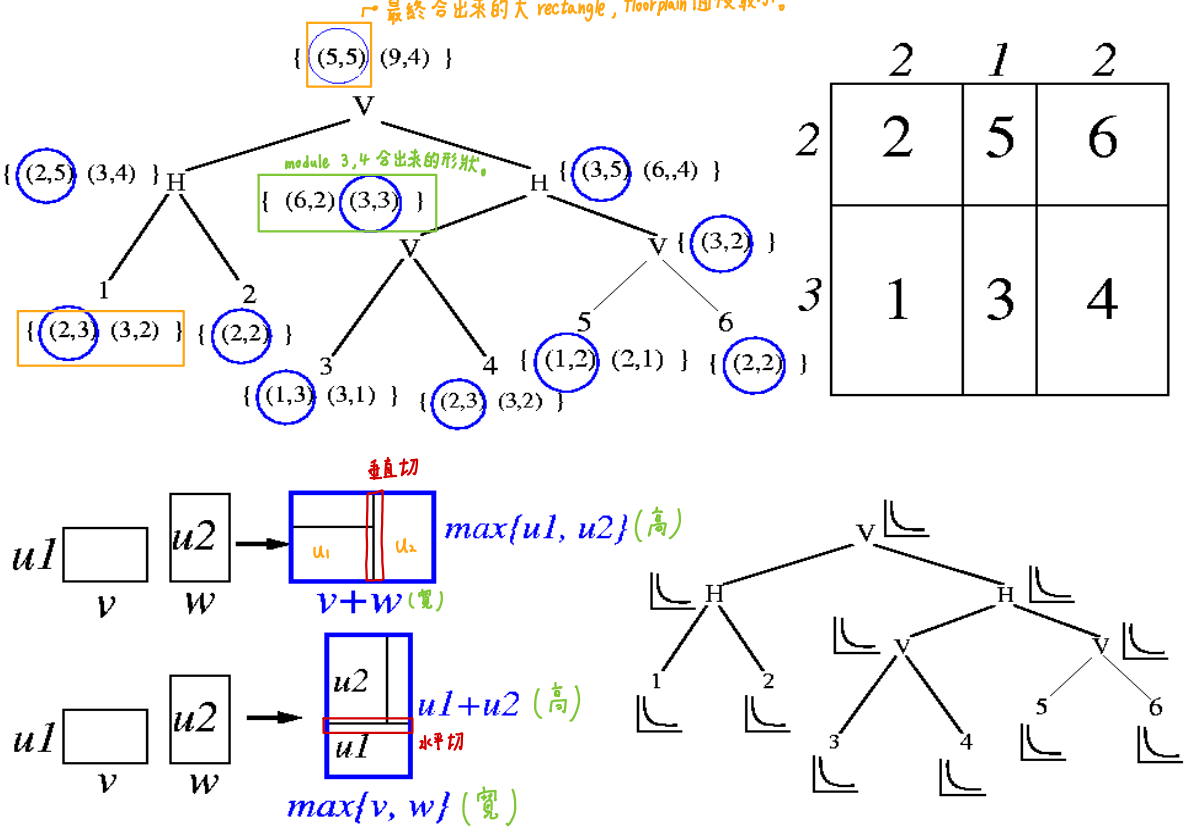
Area Computation for Hard Blocks

計算"面積最小的 floorplan"。

(此部份是 Hard modules) ⇒ shape 只有2種(轉 90°)
↳ 若是 soft module，也能用此算法。⇒ soft module 面積固定，寬高不固定。

- Stockmeyer, "Optimal orientations of cells in slicing floorplan designs," *Information and Control*, 1983.
  ↳ 若是 non-slicing structure ⇒ NP-Hard
  shape 會有無限多種。⇓ 取一定的數量，盡量逼近。
- Time complexity: $O(knd)$, where n is # modules, d is the depth of tree, and each module has $O(k)$ possible shapes.
  ↳ polynomial time

目標：給一個 NPE，要決定如何將 module，擺在"小 retangle"，才能讓 floorplain 的面積更小。

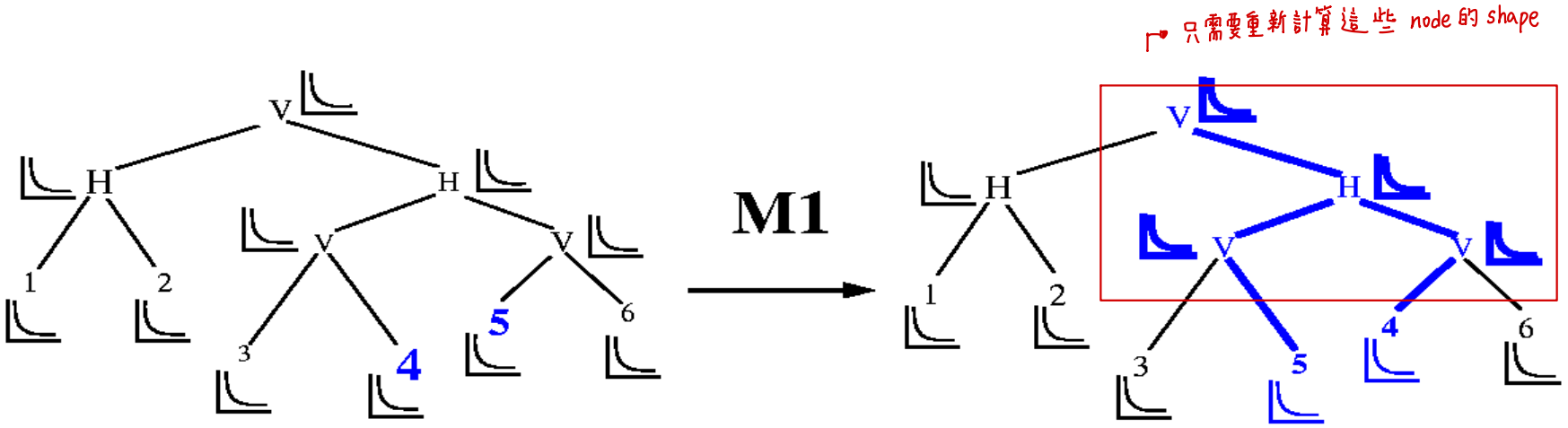


Slicing Floorplan Sizing

- The shape function of each leaf block is given as a staircase (or piecewise linear) function.
- 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).

Incremental Area Computation

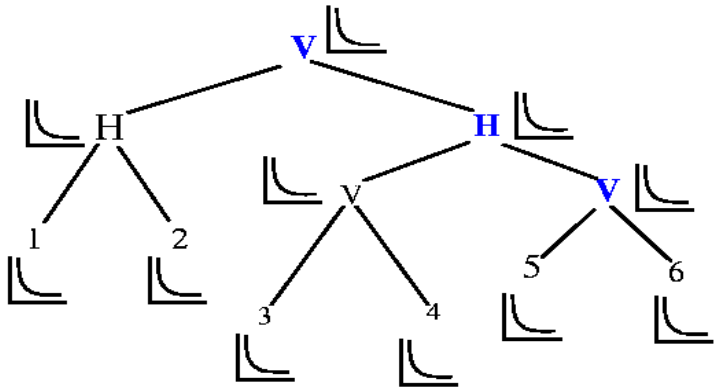
- Each move leads to only a minor modification of the Polish expression. ⇒ 減少計算時間
- At most **two paths** of the slicing tree need to be updated for each move.



E = 12H34V56VHV

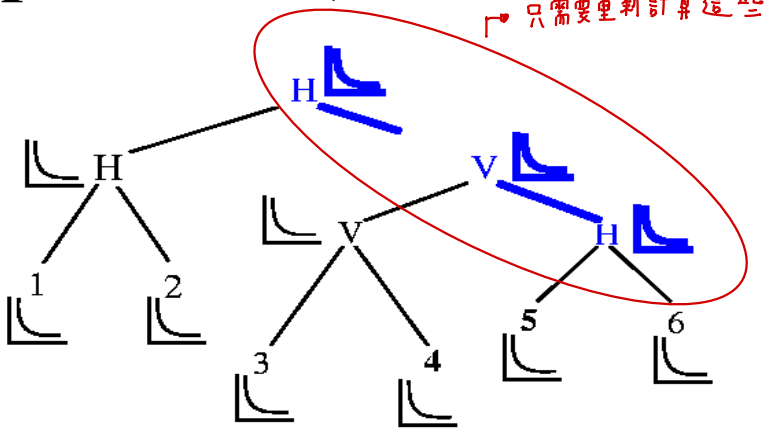
E = 12H35V46VHV

Incremental Area Computation (cont'd)

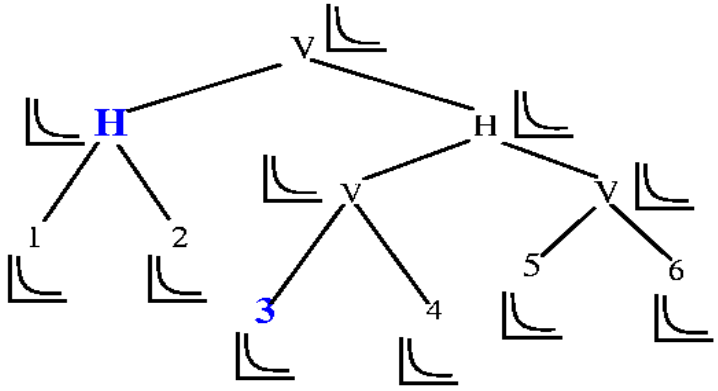


E = 12H34V56VHV

M2

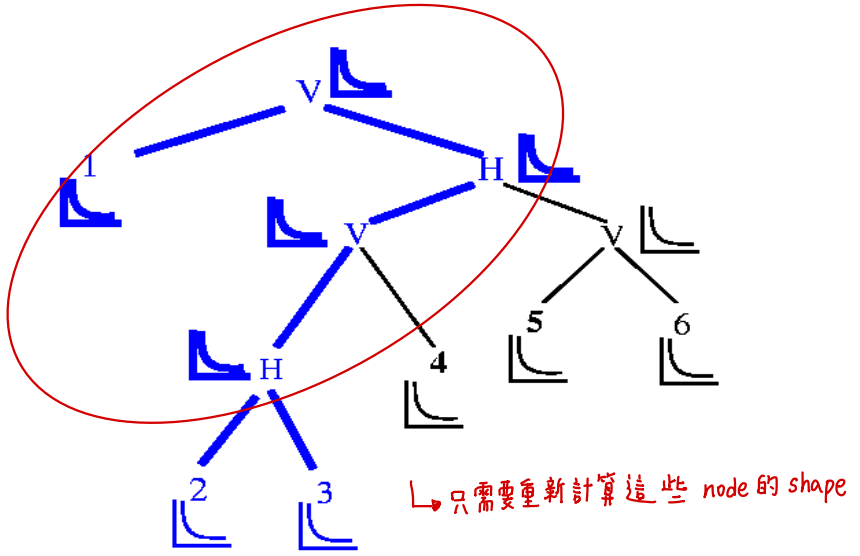


E = 12H34V56HVH



E = 12H34V56VHV

M3



E = 123H4V56VHV

Annealing Schedule

- Initial solution: $12V3V\ldots nV$

- $T_i = r^i T_0$, $i = 1, 2, 3, \ldots$; $r = 0.85$
- 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.

"隨機性很高，同一個 testcase，要跑很多次，取平均。

Algorithm: Simulated_Annealing_Floorplanning(P, ϵ, r, k)

```
1  begin
2  E ← 12V3V4V...nV; /* initial solution */
3  Best ← E; T0 ← -Δavg / 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 ei and ej; NE ← Swap(E, ei, ej);
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 ei and operator ei+1;
14                 if (ei−1 ≠ ei+1) and (2Ni+1 < i) then done ← TRUE;
15             NE ← Swap(E, ei, ei+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
```

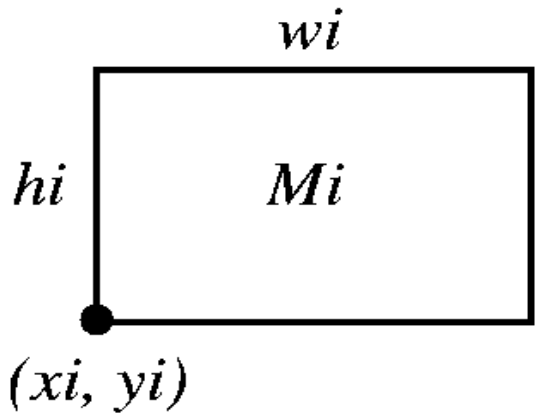
起始溫度

"P"是"一開始接受不好答案的機率"。⇒自己設

數學規劃

Non-slicing Floorplan Design by Mathematical Programming

- Sutanthavibul, Shragowitz, and Rosen, "An analytical approach to floorplan design and optimization," DAC'90.
- Notation:
  - w_i, h_i: width and height of module M_i.
  - (x_i, y_i): coordinates of the lower left corner of module M_i.
  - $a_i \le w_i/h_i \le b_i$: 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 (MILP)** formulation for the floorplan design.
  - **Linear** constraints? Objective function?



$$Area = hi * wi$$
$$Aspect\ ratio = wi / hi$$

Non-overlap Constraints

至少有一個條件要成立。

- Two modules M_i and M_j do not overlap, if at least one of the following linear constraints is satisfied (cases encoded by p_{ij} and q_{ij}):

h, w ⇒ 是已知的常數。⇒假設都是 hard module。
x, y ⇒ 是變數，做完 floorplain 才知道

∵在 ILP，必須讓4個條件同時成立 ∴要改寫4個不等式

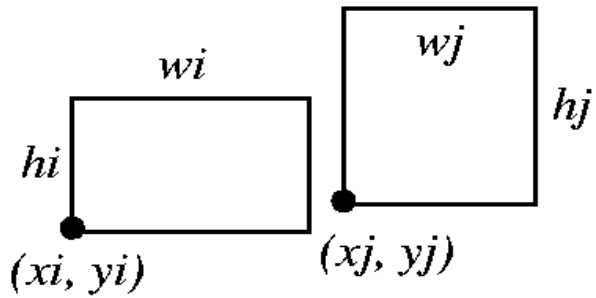
$$
\begin{array}{ll}
M_i \text{ to the left of } M_j: & x_i + w_i \le x_j \\
M_i \text{ below } M_j: & y_i + h_i \le y_j \\
M_i \text{ to the right of } M_j: & x_i - w_j \ge x_j \\
M_i \text{ above } M_j: & y_i - h_j \ge y_j
\end{array}
$$

p_{ij}	q_{ij}	在改寫後的4個不等式
0	0	⇒讓第1個條件成立
0	1	⇒讓第2個條件成立
1	0	⇒讓第3個條件成立
1	1	⇒讓第4個條件成立

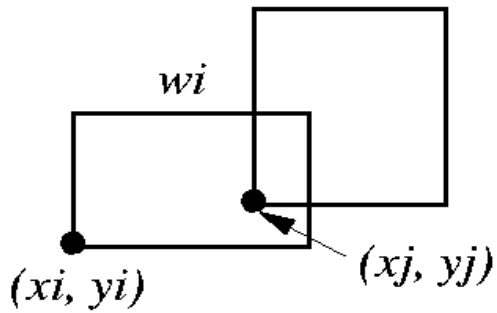
- Let W, H be upper bounds on the floorplan width and height, respectively. ⇒期望 floorplain 後，最外圍矩形的寬高
- Introduce two 0, 1 variables p_{ij} and q_{ij} to denote that one of the above inequalities is enforced; e. g., $p_{ij} = 0$, $q_{ij} = 1 \Rightarrow y_i + h_i \le y_j$ is satisfied.

改寫上面的4個不等式

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



xi + wi <= xj



xi + wi > xj

Cost Function & Constraints

- Minimize Area = xy, nonlinear! (x, y: width and height of the resulting floorplan) ⇒如果是 nonlinear，就不能用 MILP 解決。
- How to fix? ↝最後 floorplain 的結果，寬度不要超過 W，最小化高度。⇒ linear

  - Fix the width W and minimize the height y!
- Four types of constraints: ↝要滿足的條件。
  1. No two modules overlap ($\forall i, j$: $1 \le i < j \le n$);
  2. Each module is enclosed within a rectangle of width W and height y ($x_i + w_i \le W$, $y_i + h_i \le y$, $1 \le i \le n$);
  3. $x_i \ge 0$, $y_i \ge 0$, $1 \le i \le n$;
  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.

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

- Size of the mixed ILP: for n modules,
  - # continuous variables: $O(n)$; # integer variable: $O(n^2)$; # linear constraints: $O(n^2)$.
  - Unacceptably huge program for a large n!
- Popular LP software: LINDO, lp_solve, CPLEX, GUROBI, etc.

Mixed ILP for Floorplanning (cont'd)

∵多一個 r_i 變數 ∴要修改不等式。⇒仍要是 Linear

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

$r_i \in \{0, 1\}$, $1 \le i \le n$

- For each module i with free orientation, associate a 0-1 variable r_i; ⇒此部份允許 hard modules 可以旋轉。需多一個變數: r_i。⇒多一個變數，solver 解 ILP 問題，所需的時間會變長。
- $r_i = 0$: 0° rotation for module i.
- $r_i = 1$: 90 ° rotation for module i.
- $M = \max\{W, H\}$.

Soft Modules

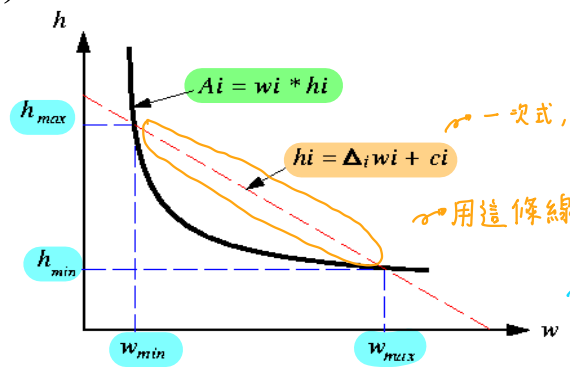
- Assumptions: w_i, h_i are unknown; area lower bound: A_i.
- Module size constraints: $w_i h_i \geq A_i$; $a_i \leq \frac{w_i}{h_i} \leq b_i$.
- Hence, $w_{min} = \sqrt{A_i a_i}$, $w_{max} = \sqrt{A_i b_i}$, $h_{min} = \sqrt{\frac{A_i}{b_i}}$, $h_{max} = \sqrt{\frac{A_i}{a_i}}$
- $w_i h_i \geq A_i$ nonlinear! How to fix? (The following fixing scheme is different from the one given in the paper.)
  - Can apply a first-order approximation of the equation: a line passing through (w_{min}, h_{max}) and (w_{max}, h_{min}).

$$h_i = \Delta_i w_i + c_i \qquad /* \; y = mx + c \; */$$

$$\Delta_i = \frac{h_{max} - h_{min}}{w_{min} - w_{max}} \qquad /* \; slope \; */$$

$$c_i = h_{max} - \Delta_i w_{min} \qquad /* \; c = y_0 - mx_0 \; */$$

  - Substitute $\Delta_i w_i + c_i$ for h_i to form linear constraints (x_i, y_i, w_i are unknown; Δ_i , c_i, h_i can be computed as above).



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?

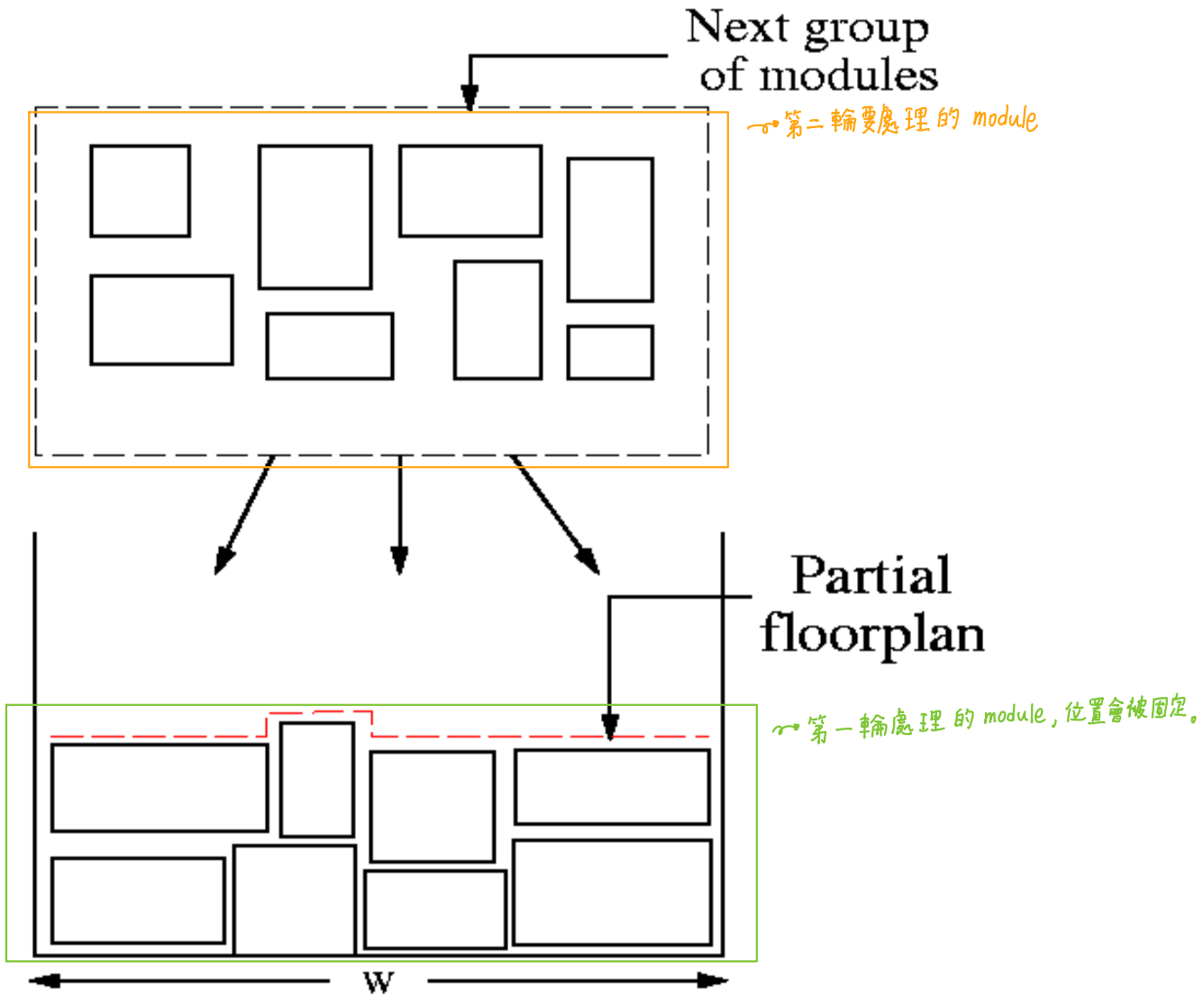
module 的數量很大時，ILP 的 solver 需要花很多時間。

- Key: solve a partial problem at each step (successive augmentation)

解法：假設現在有"100個 module"，ILP solver 要花很多時間才能找到"最佳解"。
∴將"100個 module"分成 10次解決，ILP solver 一次處理"10個 module"。
⇒ 可能無法得到最佳解

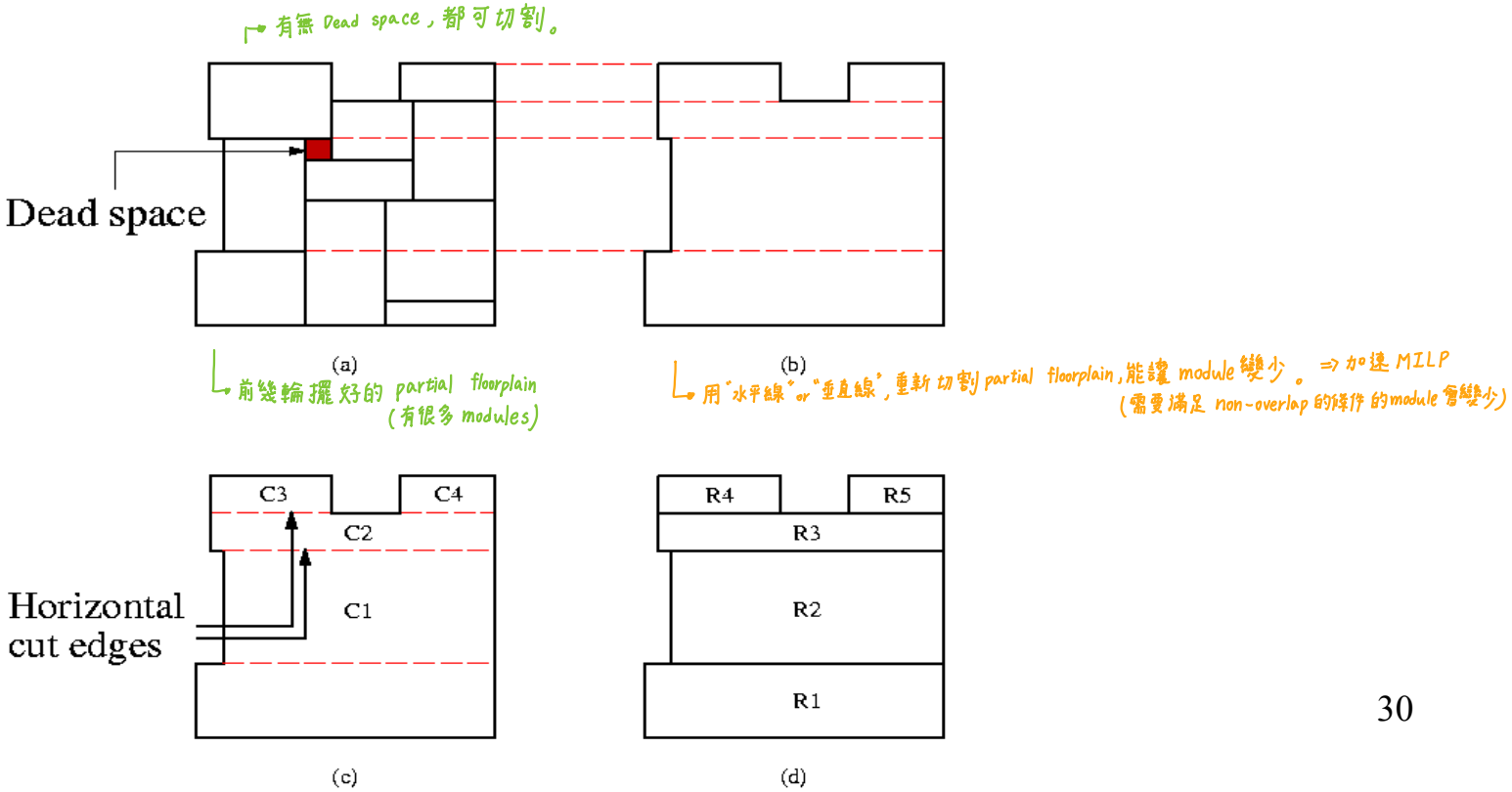
- Questions:
  - How to select next subgroup of modules? ⇒linear ordering based on connectivity.
  - How to minimize # of required variables?

將"100個 module"分成 10次解決，ILP solver 一次處理"10個 module"。



Reducing the Size of the Mixed ILP (cont'd)

- 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.



P-admissible Solution Space

- **P-admissible** solution space for Problem *P*:
  1. the solution space is finite,
  2. every solution is feasible, ⇒合理的答案
  3. evaluation for each configuration is possible in polynomial time and so is the implementation of the corresponding configuration, **and**
  4. the configuration corresponding to the best evaluated solution in the space coincides with an optimal solution of *P*.
- Slicing floorplan is **not** P-admissible. Why?
- A P-admissible floorplan representation:

  Sequence-Pair.

  ↳ Ex: 前面 slicing floorplain 提到的 NPE ⇒ 不能用在 non-slicing floorplain

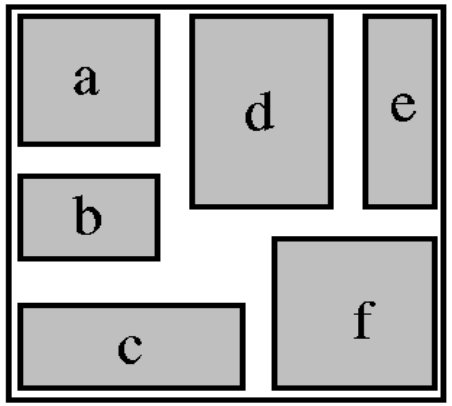
  ↳ 可以用來描述 slicing floorplain, non-slicing floorplain

2個 sequence 形成一個 pair，sequence 是"module 的 permutation"。
若有 n個 module，則可能的排列組合有"n!"個。

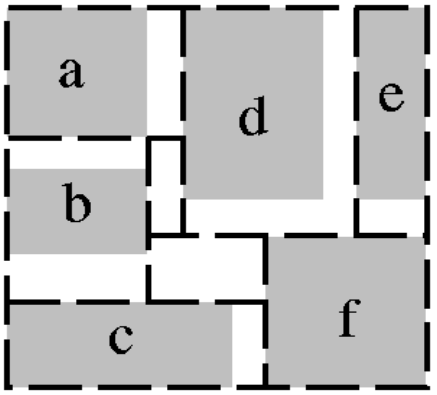
Sequence Pair (SP)

- Murata, Fujiyoshi, Nakatake and Kajitani, "Rectangle-packing-based module placement," ICCAD'95.
- Represent a *packing* by a pair of module permutations called sequence-pair (e.g., (*abdecf*, *cbfade*)).
- The set of all sequence-pairs is a P-admissible solution space whose size is $(n!)^2$.
- Search in the P-admissible solution space by simulated annealing. 加速找答案
  - Swap two modules in the first sequence.
  - Swap two modules in both sequences.
  - Rotate a module.

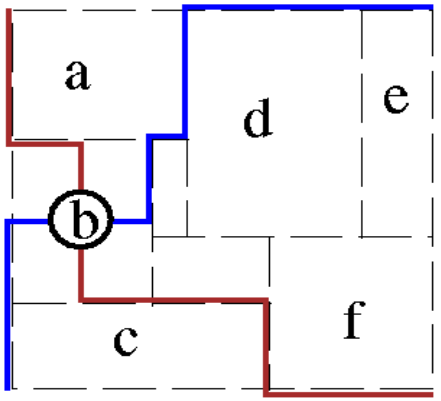
從一個 sequence pair 產生新的 sequence pair 的方法。



擺放的結果



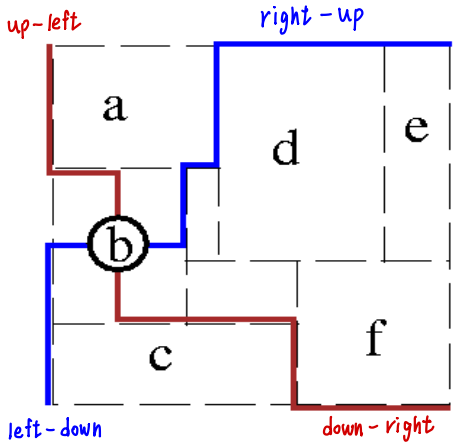
A floorplan



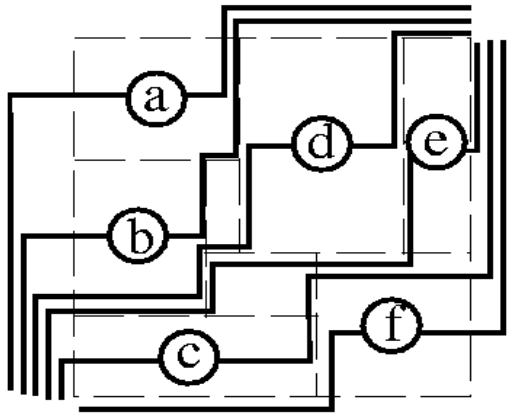
Loci of module b

Relative Module Positions

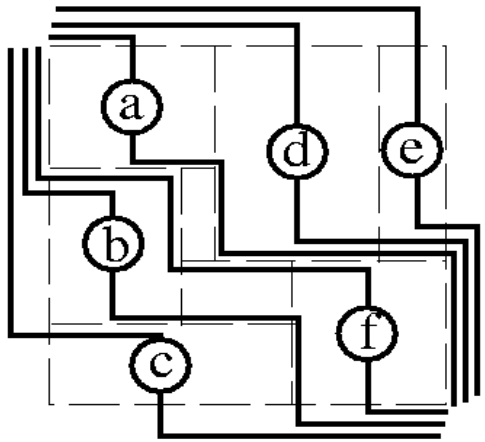
- A floorplan is a partition of a chip into rooms, each containing at most one module.
- **Locus** (right-up, left-down, up-left, down-right)
  1. Take a non-empty room.
  2. Start at the center of the room, walk in two alternating directions to hit the sides of rooms.
  3. Continue until to reach a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



Unit 4 *Loci of module b*



Positive loci: abdecf

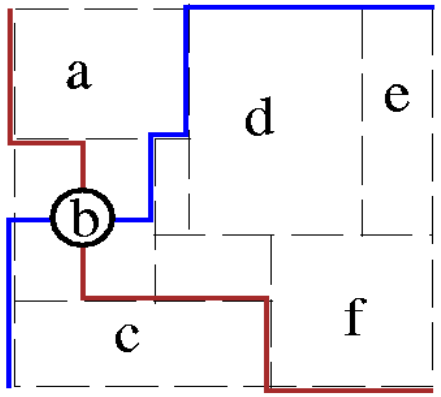


Negative loci: cbfade

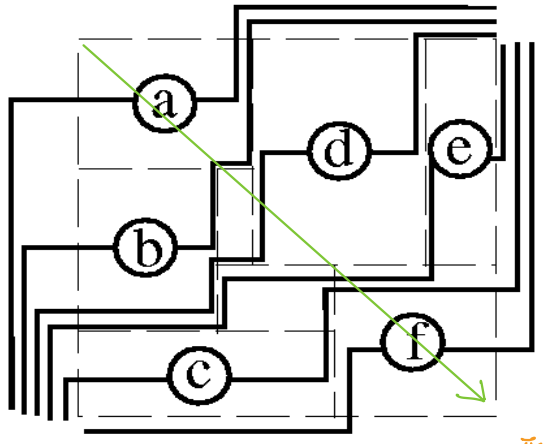
Geometrical Information

- No pair of positive (negative) loci cross each other, i.e., **loci are linearly ordered**. => 每一個 module 形成的"positive loci 或 negative loci"，彼此不會 overlap。如圖 2, 3。
- **Sequence-Pair** (Γ_+, Γ_-): Γ_+ (Γ_-) is the module permutation representing the order of positive (negative) loci.
  E.g., (Γ_+, Γ_-) = (*abdecf*, *cbfade*). sequence pair；一個 sequence pair，會對應到很多不同的 floorplain，因為還沒考慮到 module 的形狀。
  在 sequence pair 出現的關係，會對應到 floorplain 中"不同 module 之間的空間關係"。
- *x*' is **after (before)** *x* in both Γ_+ and Γ_- $\Rightarrow$ *x*' is **right (left)** to *x*.
- *x*' is **after (before)** *x* in Γ_+ and before **(after)** *x* in Γ_- $\Rightarrow$ *x*'is **below (above)** *x*. sequce pair 中，只能找到一種空間關係（上下 or 左右），Ex: "a"和"f"。

取 sequence 的方向



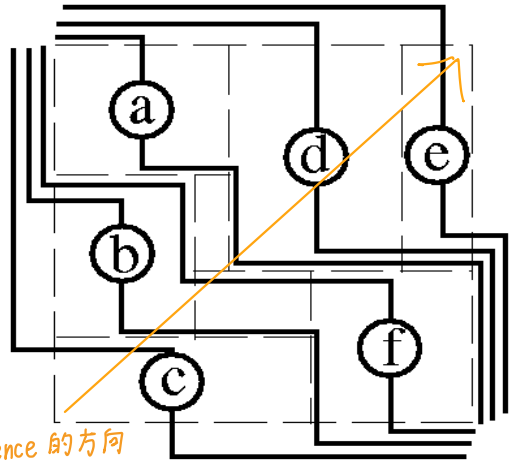
Loci of module b



Positive loci: abdecf

sequence pair 的"第一個 sequence"。

取 sequence 的方向



Negative loci: cbfade

sequence pair 的"第二個 sequence"。

∴ 每一個 floorplain 的結果，都能找到一個 sequence pair 作對應。

給一個 sequence pair，希望找到 面積最小的 placement ⇒ 面積最小的 floorplain

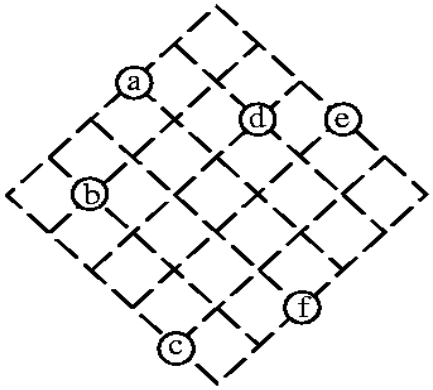
想成是 placement，任2個 module 不能重疊

Optimal (Γ_+, Γ_-)-Packing

① 在 Horizontal constraint graph 中，找"s到t的longest path"，即為 面積最小的 placement 的"寬度"。
② 在 Vertical constraint graph 中，找"s到t的longest path"，即為 面積最小的 placement 的"高度"。

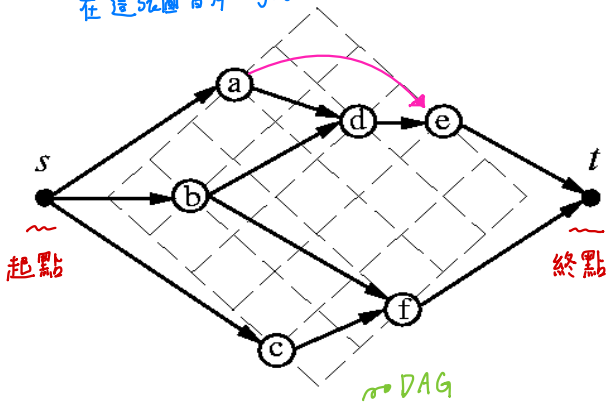
- For every sequence-pair (Γ_+, Γ_-), there is an optimal (Γ_+, Γ_-)-packing.
- **Horizontal constraint graph** $G_H(V, E)$ (similarly for $G_V(V, E)$):
  - V: source s, sink t, n vertices for modules.
  - E: (s, x) and (x, t) for each module x, and (x, x') iff x must be left-to x'.
  - **Vertex weight**: 0 for s and t, **width** of module x for the other vertices.

如果是 vertical constraint graph，就會改成"height"。



Packing for sequence pair: (abdecf, cbfade)

"其他 vertex" 對應一個 module。
在 sequence pair 有"左右關係"，則
在這張圖會有 edge。



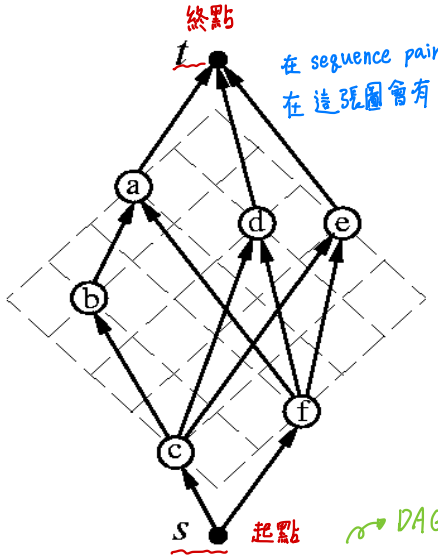
DAG

Horizontal constraint graph (Transitive edges are not shown)

利用 sequence pair 得知的"上下、左右關係"，可以得到這2張圖

終點

在 sequence pair 有"上下關係"，則
在這張圖會有 edge。

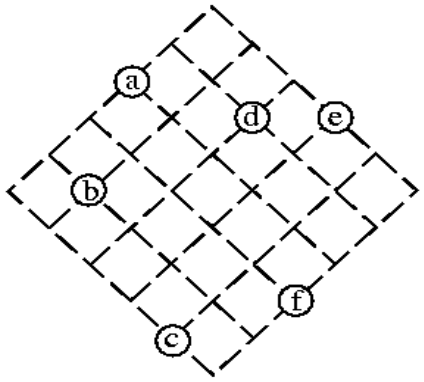


DAG

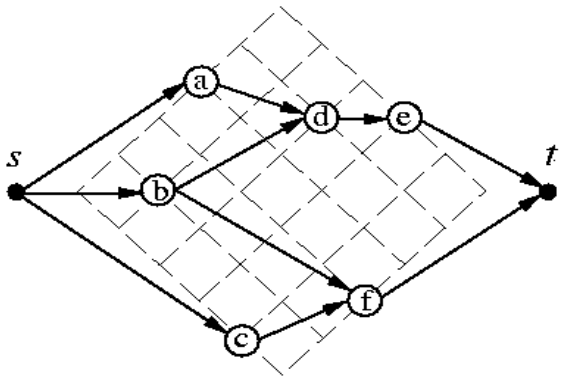
Vertical constraint graph (Transitive edges are not shown)

Optimal (Γ_+, Γ_-)-Packing (cont'd)

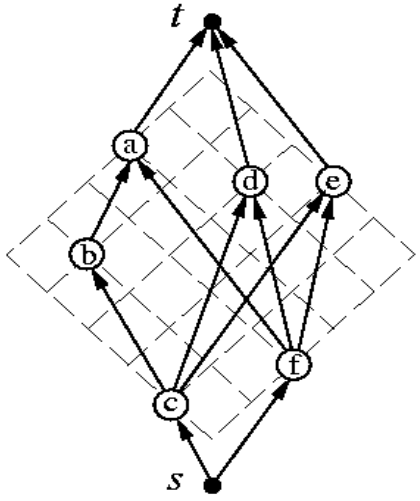
- **Optimal (Γ_+, Γ_-)-packing** can be obtained in $O(n^2)$ time by applying a longest path algorithm on a vertex-weighted directed acyclic graph.
  - G_H and G_V are independent.
  - The x and y coordinates of each module are determined by assigning the longest path length between s and the vertex of the module in G_H and G_V, respectively.
- More efficient algorithms for obtaining optimal (Γ_+, Γ_-)-packing: $O(n\log n)$ by Takahashi, IEICE'96; $O(n\log n)$ by Tang, Tian & Wong, DATE'00; $O(n\log\log n)$ by Tang & Wong, ASP-DAC'01.



Packing for sequence pair: (abdecf, cbfade)



Horizontal constraint graph (Transitive edges are not shown)



Vertical constraint graph (Transitive edges are not shown)

O-Tree

- Guo, Cheng, and Yoshimura, "An o-tree representation of non-slicing floorplan and its applications," DAC'99.
- Definitions:

  - A placement is **L-compact (B-compact)** if and only if no module can be moved left (down) from its original position with other modules' positions fixed.

  - A placement is **LB-compact** (or **admissible**) if and only if it is both L-compact and B-compact.

- Given any placement P_1, a corresponding admissible placement P_2 can be obtained by a sequence of x-direction and y-direction compactions. The overall area of P_2 is no larger than the overall area of P_1.

若有 n 個 module，則可以 construct 含有 "n+1 個 node" 的 tree。⇒ tree 有 ordered

O-Tree Encoding

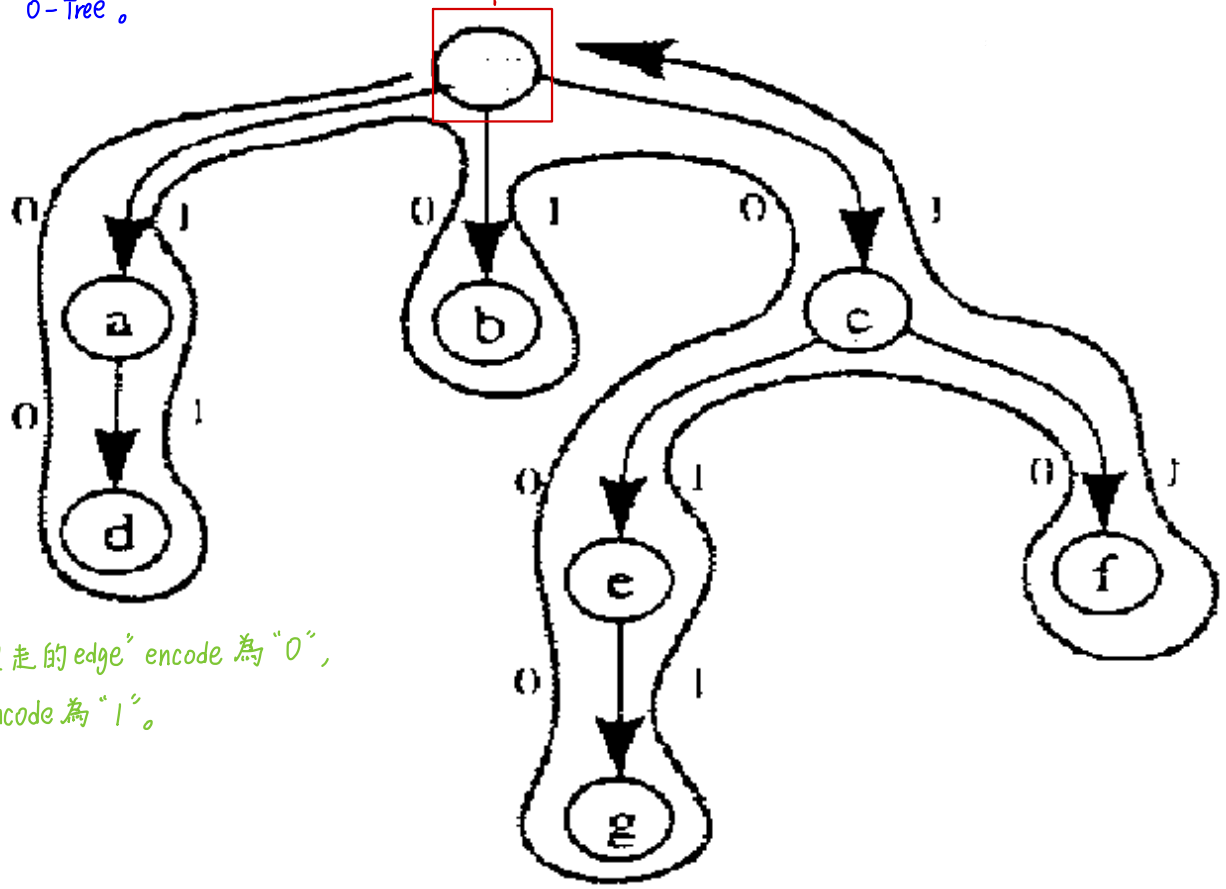
對 O-Tree 作 DFS 的順序

利用 DFS，對 O-Tree 作 Encoding

(00110100011011, *adbcegf*) with Depth-First-Search

用來表示 "O-Tree"。

root 不代表任何 module



"第一次走的 edge" encode 為 "0"，
否則 encode 為 "1"。

O-Tree Encoding (cont'd)

- **Space needed to store (T,π):** Given a tree with n nodes in addition to its root, the label of each node can be encoded into a $[\lg n]$ bit string, and hence $n(2+[\lg n])$ bits are needed to store (T,π) where $2n$ bits for T, and $n[\lg n]$ bits for π.

- **Number of possible (T,π)'s:** $O(n!2^{2n-2}/n^{1.5})$

利用"O-Tree"得到對應的"module placement"。

O-Tree and Placement

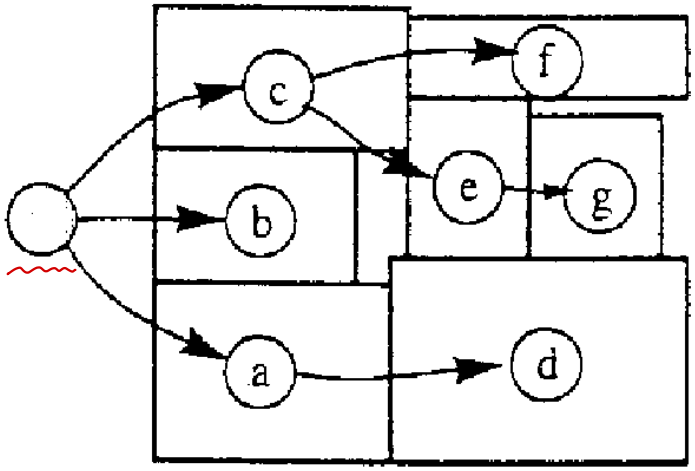
- **Horizontal O-Tree:** => B-compact placement，可以長出 Vertical O-Tree

  - Suppose i is the parent of j, $\Psi(j)$ is the set of blocks each of which appears before j in π and overlaps with j in the x-coordinate projections.

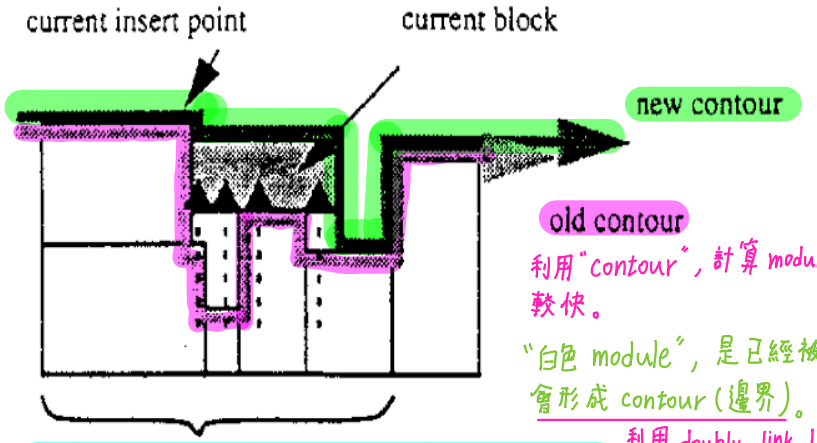
依照 O-Tree 的 order，決定每個 module 的"x、y座標"。

(00110100011011,*adbcegf*)

$$\begin{cases} x_j = x_i + w_i & \Rightarrow \text{先決定每個 module 的 x座標。} \\ y_j = 0 \mid (\max_{k \in \psi(j)} y_k + h_k) & \Rightarrow \text{module 能放多低就多低} \end{cases}$$



座標：(0,)
w=0, h=0



利用"contour"，計算 module 的 y座標較快。

"白色 module"，是已經被擺好，會形成 contour (邊界)。

利用 doubly link list 紀錄

↳ 可以順便建出"vertical constraint graph"。

- **Vertical O-Tree:** can be defined similarly => L-compact placement，可以長出 Horizontal O-Tree

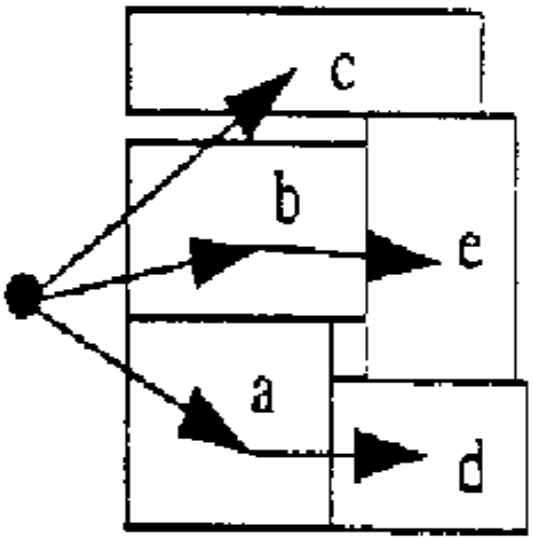
∴透過不斷迭代，可以得到 LB-compact placement

↳先決定每個 module 的 y座標。

O-Tree and Placement (cont'd)

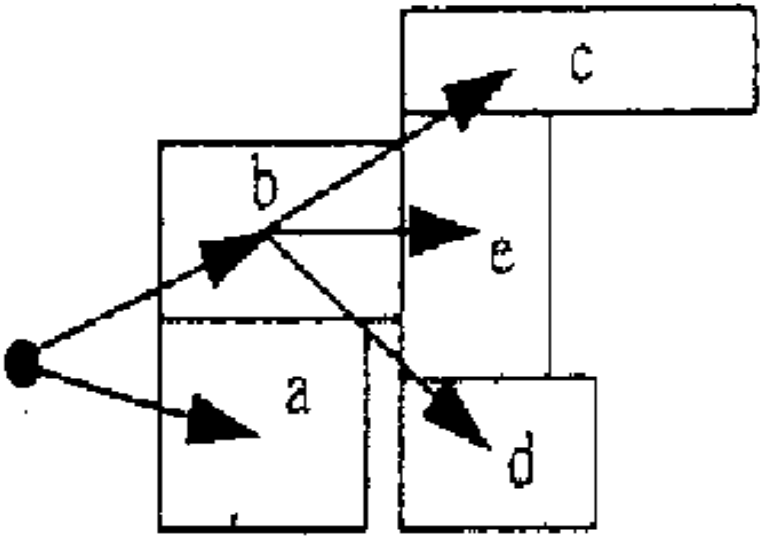
An O-tree is **admissible** if its corresponding placement is admissible.

(0011001101,adbec)



admissible

(0100101011,abdec)



not admissible

∵module還可以往左移

得到 admissible O-Tree 的方法

Admissible O-Tree Transformation (AOT)

- Given a horizontal O-tree T, we can first get a vertical constraint graph G_y by applying "OT2OCG" to T in linear time, and then get a vertical O-tree T_y by applying "CG2OT" to G_y in linear time. After applying the same procedures OT2OCG and CG2OT again, we can get another horizontal O-tree. The OT2OCG and CG2OT are iterated until an admissible O-tree is found.

$$\text{H-O-tree} \xrightarrow{\text{OT2OCG}} G_v\text{(B-compact)} \xrightarrow{\text{CG2OT}} \text{V-O-tree} \xrightarrow{\text{OT2OCG}} G_h\text{(L-compact)} \xrightarrow{\text{CG2OT}} \text{H-O-tree...}$$

- All compactions are **monotone** because modules are either moved down or left. Therefore, convergence of the above iteration is assured and we can get an admissible O-tree.

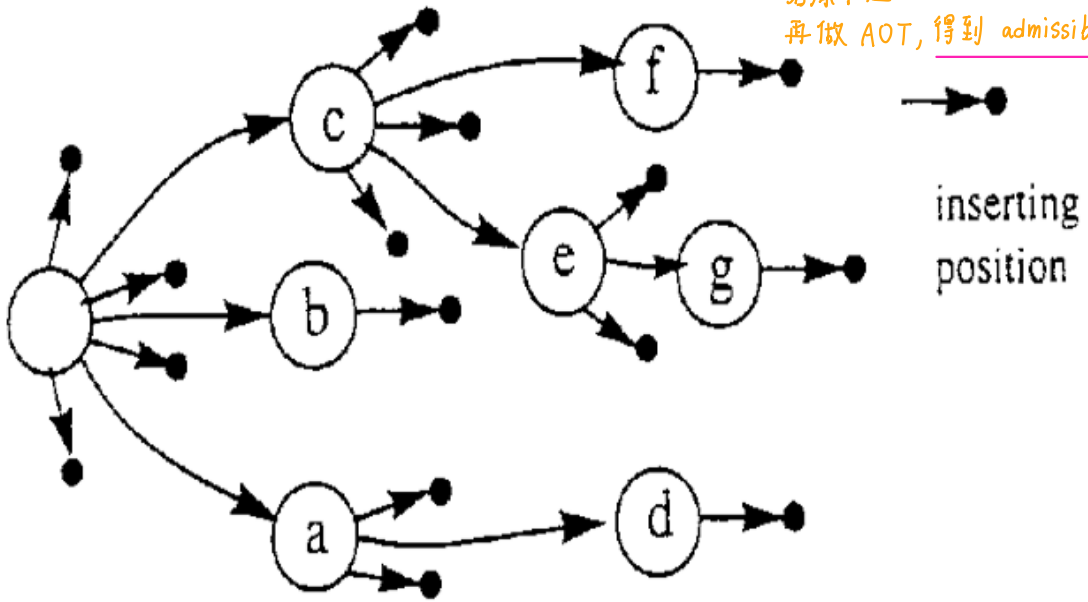
Solution Perturbation

a Select a module B_i in the O-tree (T,π).

b Delete B_i from the O-tree (T,π).

c Insert B_i in the position with the best cost value among all possible inserting positions in (T,π) as an external node.

d Perform a-c on it orthogonal O-tree.



•Given any O-tree with n nodes, the number of possible inserting position as external nodes is $2n$-1.

A Deterministic Placement Algorithm

- Perturb O-trees in sequence.
  - Select nodes (module) in sequence and find the best perturb position for each of them.
  - A perturbed O-tree can be made admissible using AOT.
- Implementation is straightforward.

一種 floorplan 的表示法 ⇒ 可以表示 non-slicing floorplan

B*-Tree

Binary tree

- Chang, Chang, Wu, and Wu, "B*-tree: a new representation for non-slicing floorplans," DAC'00.
- Ideas:

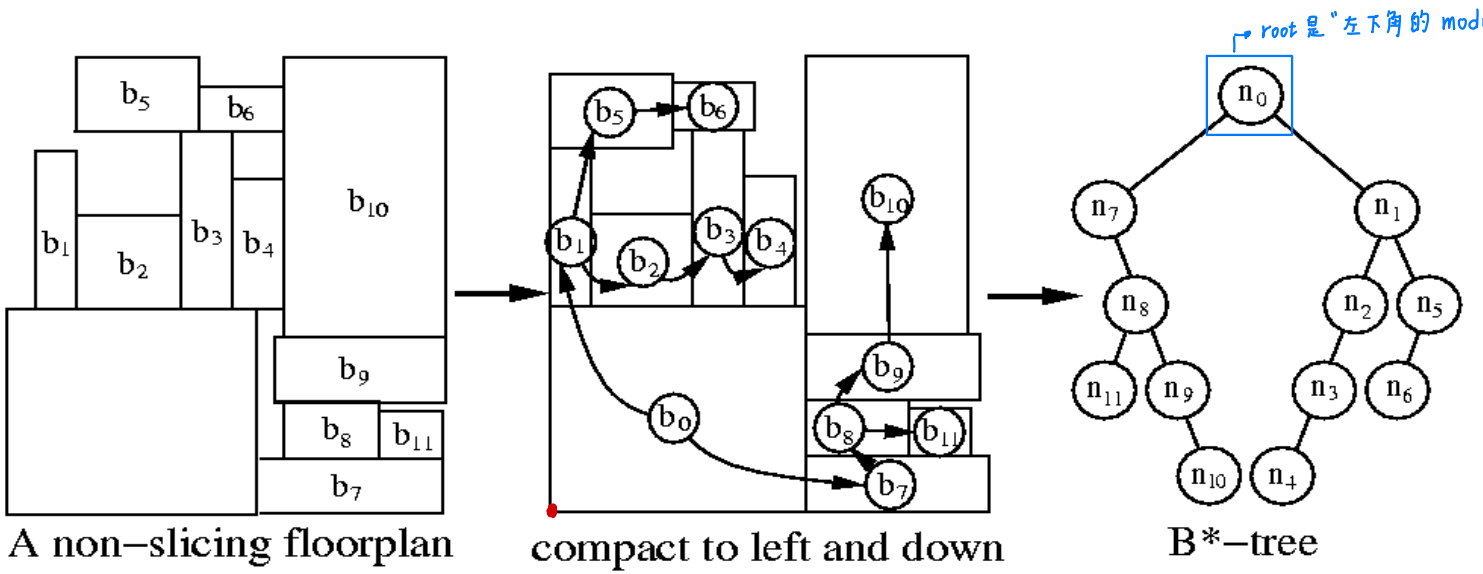
若有一個 admissible placement，則可以 construct 一個 B* tree

  - From an admissible placement to a B*-tree:
    Left child: the lowest module on the right.

在 parent 的右邊界，y座標最小的 module

    Right child: the module above, with the same left-side coordinate.

在 parent 的上邊界，"x座標"和 parent 一樣的 module
(以 module 的左下角為基準)



root 是"左下角的 module"。

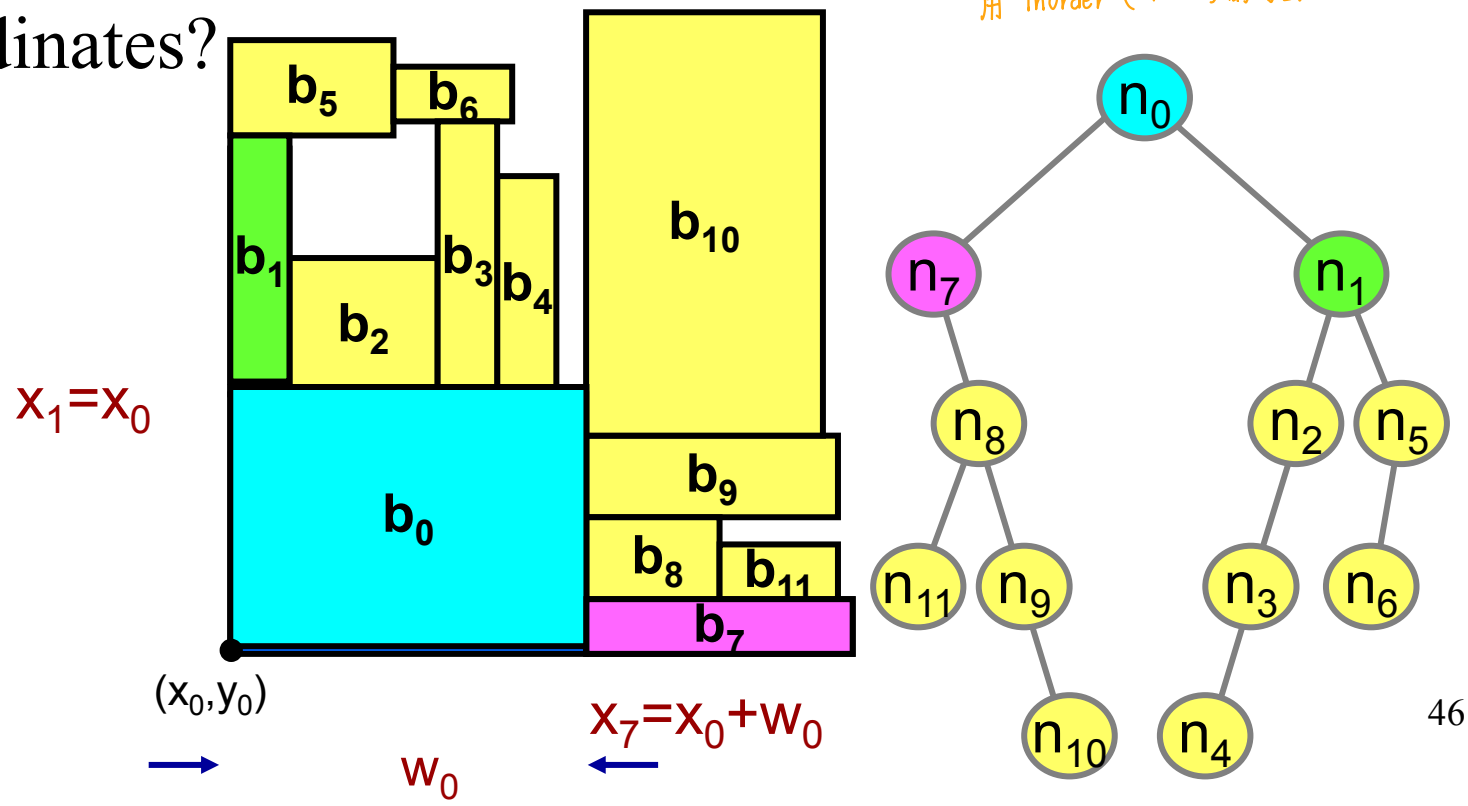
先將 floorplan 轉成 admissible placement

從 B* tree 得到對應的 admissible placement

B*-tree Packing

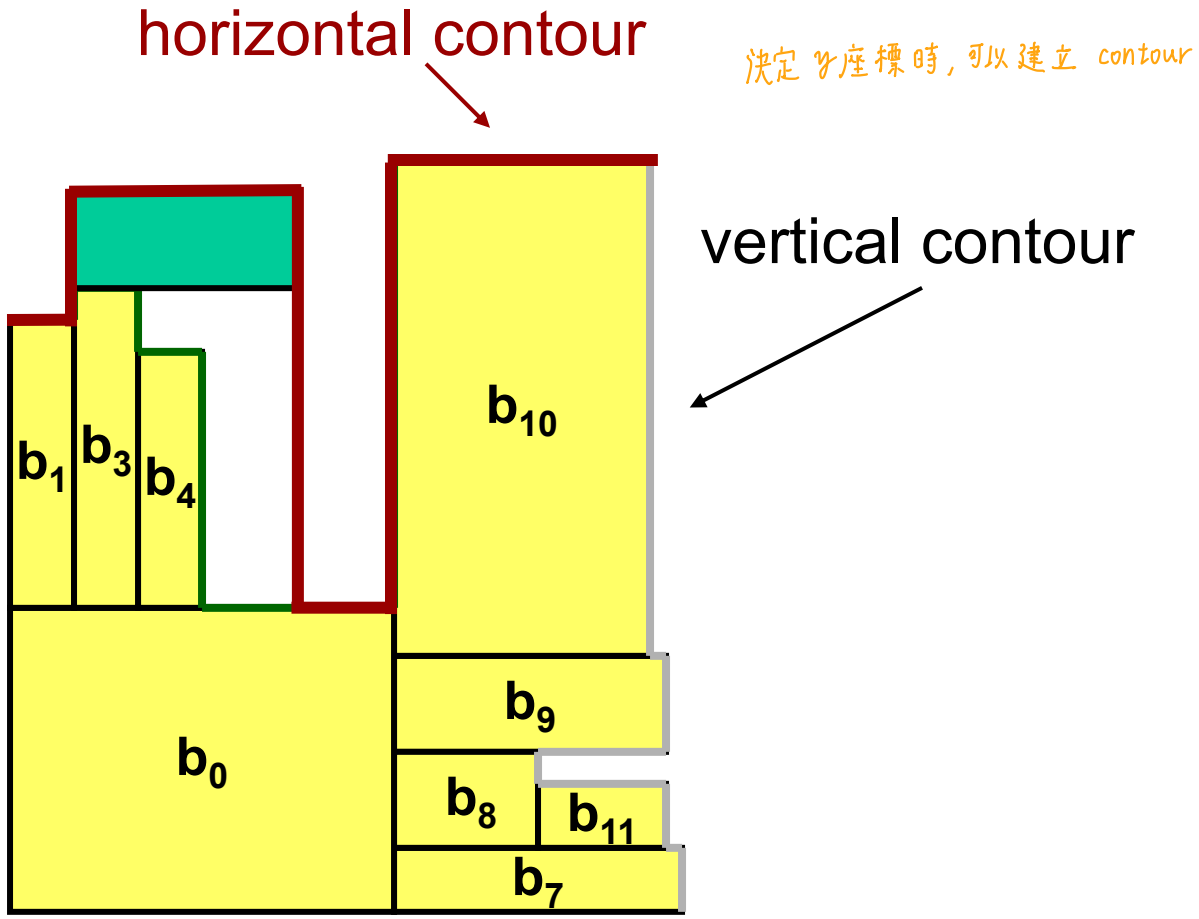
- x-coordinates can be determined by the tree structure. B* tree 得到的 placement，可能是：① admissible placement ② B-compact ⇒ 與"左、右小孩"的設定有關
  - Left child: the lowest, adjacent block on the right ($x_j = x_i + w_i$).
  - Right child: the first block above, with the same x-coordinate ($x_j = x_i$).
- y-coordinates?

用 inorder（中左右）的方式 track node。



Computing y-coordinates

- Reduce the complexity of computing a y-coordinate to amortized O(1) time.



Perturbations & Solutions

- Perturbing B*-trees in simulated annealing
  - Op1: Rotate a module. ⇒ 旋轉 90°, 180°
  - Op2: Flip a module. ⇒ 若是 hard module，則 module 可以沿著"y軸 or x軸"作翻轉，翻轉後"pin 的位置會變"。wirelength 會變
  - Op3: Move a module to another place.
  - Op4: Swap two modules

產生新的 B*tree 的方法

B* tree 的概念和 O-Tree 很像

B* tree: binary tree

O-Tree: tree

* ① B* tree，O-Tree 沒辦法準確知道"任2個 module 擺放位置的相對關係"。
  ② sequence pair，slicing tree，NPE 可以準確知道"任2個 module 擺放位置的相對關係"。

Fixed-Outline Floorplanning

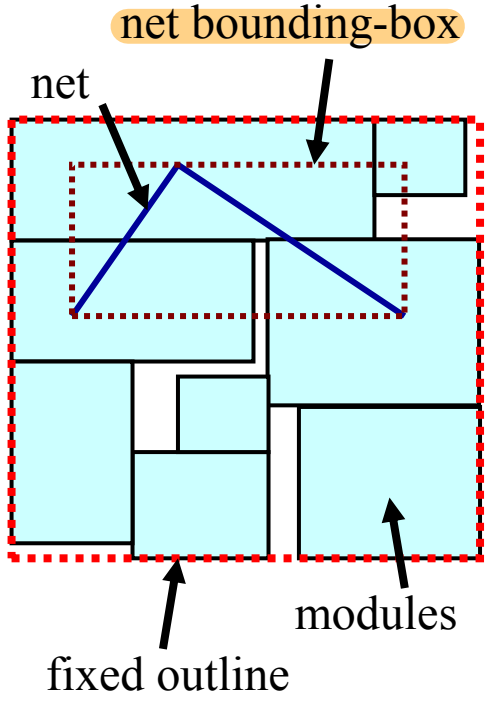
此篇 paper 是使用 "B* tree" 搭配 SA，所以做出來的結果可能是 non-slicing floorplan

- Chen and Chang, "Modern floorplanning based on fast simulated annealing," ISPD'05 & TCAD'06
- 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

"net bounding box" 的周長的一半。

module 要放在 "固定的區域 (fixed-outline)" 中。
寬高會指定。

包含"一條 net 上的所有 pin"，所框出來的"最小矩形"。



∴ floorplan 的總面積 = A(1+Γ)

Fixed-Outline Constraint

所有 module 的總面積。　　Dead space 的百分比

- Given the total area A of modules, the percentage Γ of dead space, and the desired aspect ratio R^*, the outline is defined by
  - Fixed-outline 的 aspect ratio

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

⇒ Fixed-outline 的寬高。

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

此部份希望 "R 和 R*" 要接近，cost 會較小。

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

  - A: floorplan area ~ 透過 B* tree，得到的 floorplan 的面積。
  - L: wirelength ~ 加總 "每條 net 的 HPWL"。
  - R^*: fixed-outline aspect ratio
  - R: floorplan aspect ratio ~ 透過 B* tree，得到的 floorplan 的 aspect ratio ⇒ R = H/W

α, β 是 "user define"。

Simulated Annealing Schedules

- Classical simulated annealing (SA) ⇒傳統的 SA
  - Non-zero probability for up-hill move:
    接受較差答案的機率 ↳ "新cost" − "目前的cost"。 ↳ solution 的 cost 較高，相較於"目前的cost"。 ↳ 產生的答案較差。

$$p = e^{-\Delta C/T}$$

    ↳ p是"一開始接受不好答案的機率"。⇒"剛開始"會設很高
  - Initial temperature: $T = |\Delta_{avg} / \ln p|$, p is the initial acceptance rate (typically, close to 1.0), Δ_{avg} is the average cost of the up-hill moves
    ↳ 產生一堆答案，且答案是 up-hill move，算出它們的"Δc的平均值"。
  - Classical temperature updating function: λ is set to a fixed value (e.g., 0.85) ⇒下降溫度的方式

$$T_{new} = \lambda T_{old}, \qquad 0 < \lambda < 1$$

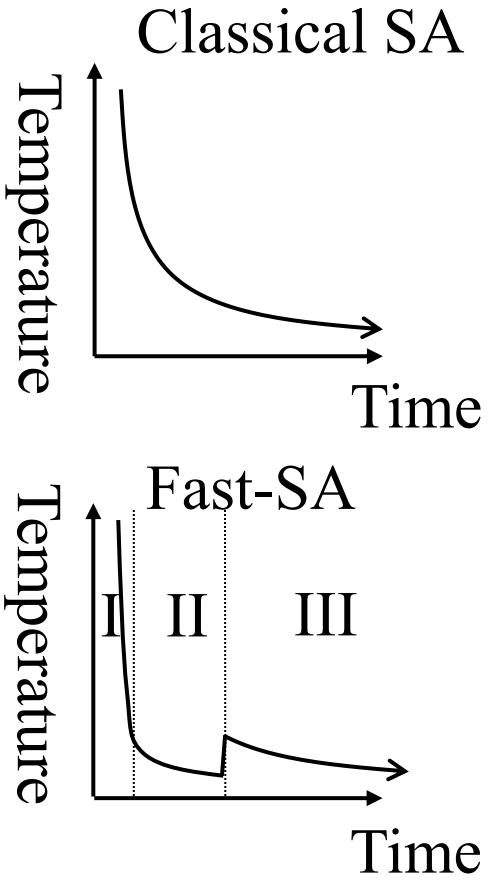
- TimberWolf annealing schedule (Sechen and Sangiovanni-Vincentelli, DAC'86) ↳一種軟體
  - Increase λ gradually from its lowest value (0.8) to its highest value (approximately 0.95) and then gradually decreases λ back to its lowest value. ⇒"溫度下降的方式"不是常數。

此篇 paper 是使用 "Fast-SA"。

Fast Simulated Annealing

- Fast Simulated Annealing (Fast-SA) consists of 3 stages
  - High-temperature random search (temperature T -> a very large value) ⇒剛開始"溫度逼近無限大"。⇒"接受不好的答案"的機率很大。
  - Pseudo-greedy local search (T -> 0) ⇒"T 接近於 0"。⇒"接受不好的答案"的機率很小。
  - Hill-climbing search (increase T to simulate regular SA)

    拉高溫度，再用"傳統 SA"的溫度下降方式。



Fast Simulated Annealing (cont'd)

- Temperature update (T_1: initial temperature)

代表"3階段"。 "r"代表"第幾個 iteration"。

$$T_r = \begin{cases} \dfrac{\Delta avg}{\ln P} & r = 1 \quad ① \\ \dfrac{T_1 \langle \Delta_{cost} \rangle}{rc} & 2 \le r \le k \quad ② \\ \dfrac{T_1 \langle \Delta_{cost} \rangle}{r} & r > k \quad ③ \end{cases}$$

Δavg	Average uphill cost
P	Initial acceptance rate
$\langle \Delta_{cost} \rangle$	Average cost change for current temperature (在某個溫度下，產生的答案的"平均改變的cost"。 ↳ Δ_{cost} 會被 normalize ∴ 數值 < 1)
r	Number of iterations
c, k	User-specified parameters (e.g., c = 100, k = 7) 控制溫度要下降多快。

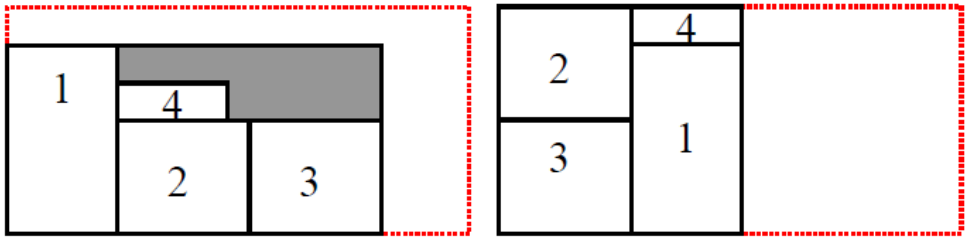
- If $\langle \Delta_{cost} \rangle$ is larger, temperature decreases slowly.
- If $\langle \Delta_{cost} \rangle$ is smaller, temperature decreases quickly.

"程式執行的過程中",會適時更改"α、β的值"。

Adaptive Fast-SA

- The aspect ratio of the best floorplan area in the fixed outline is not the same as that of the outline.
- Decrease the weight of aspect ratio penalty (1 - α - β) to concentrate more on the floorplan wirelength/area optimization (i.e., increase α and β).
  - Adopt an adaptive method to control the weights in the cost function based on *n* most recent floorplans.
  - The more feasible floorplans, the less aspect ratio penalty.

若產生的 floorplan,能擺進 Fixed-Outline ,但是 aspect ratio 沒有接近 R*,則可以增加"α,β",降低懲罰。

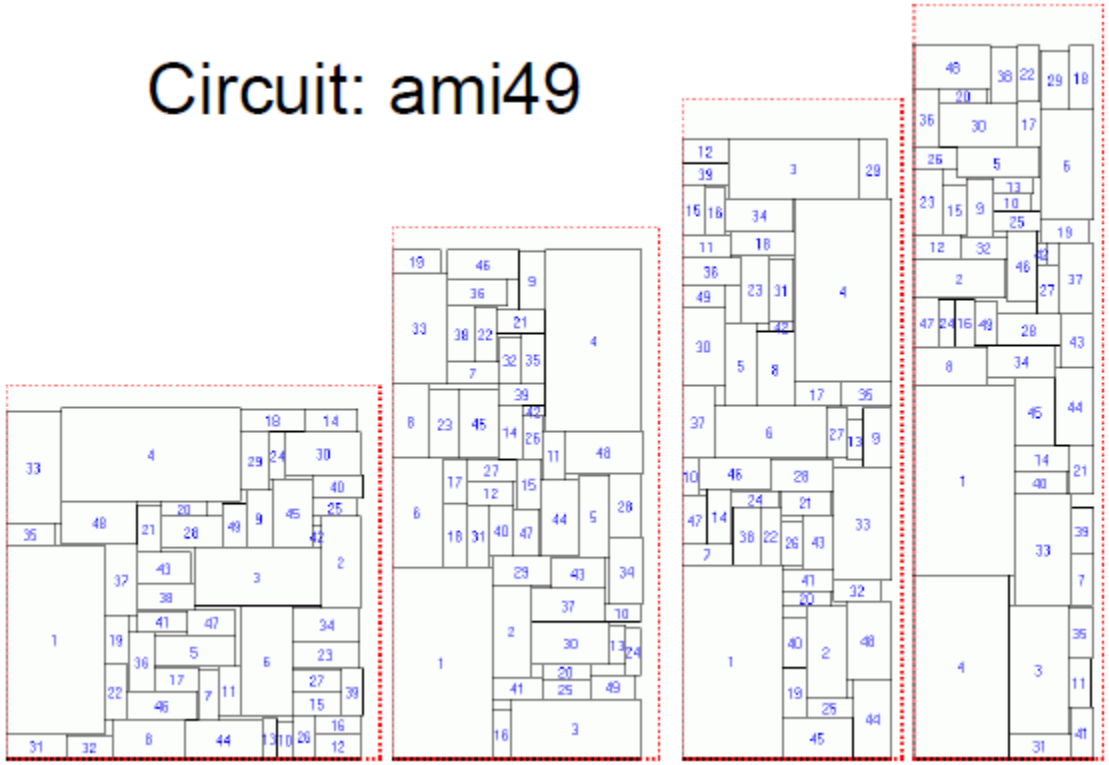


R = 0.5 R = 1

fixed-outline aspect ratio R* = 0.5

B*-tree Fixed-Outline Floorplanning Results

- B*-tree representation
- Fixed-outline floorplans with 10% dead space and aspect ratios 1, 2,3, and 4.



Floorplanning for Large-Scale Circuits

做 floorplan 時，要處理的 module 數量非常多。

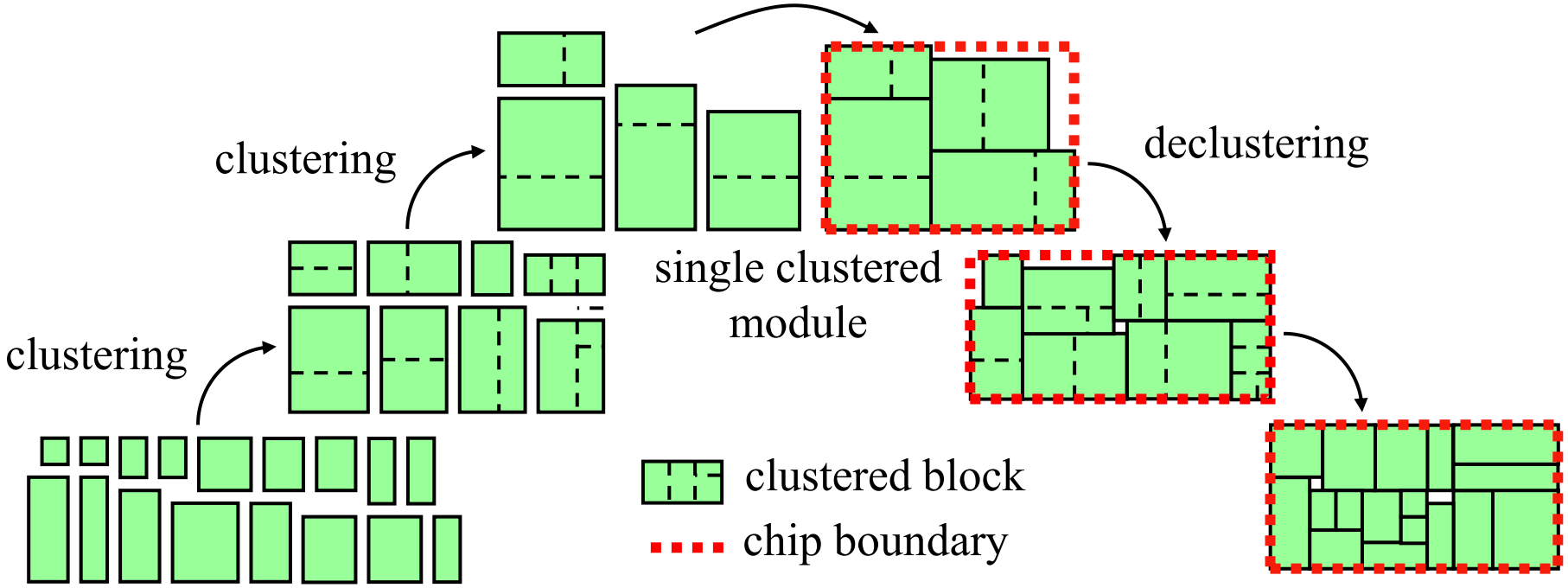
- Lee, Hsu, Chang, Yang, "Multilevel floorplanning/placement for large-scale modules using B*-trees," DAC'03 & TCAD'07
- Clustering (bottom-up coarsening) + declustering (top-down uncoarsening
  - Clustering
    - Iteratively groups a set of modules based on area utilization and module connectivity

      Dead Space
    - Constructs a B*-tree to keep the geometric relations for the newly clustered modules ⇒ "每一個 cluster" 都要建一個 B* tree
  - Declustering
    - Iteratively ungroups a set of the previously clustered modules (i.e., perform tree expansion)
    - Refines the solution using simulated annealing

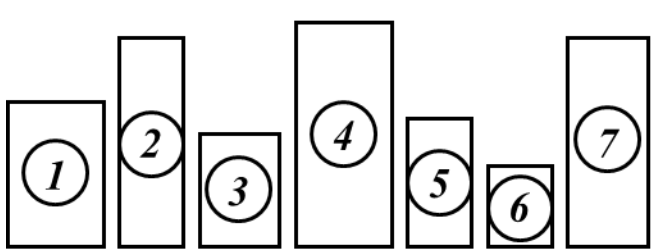
Λ-Shaped Multilevel Floorplanning

Cluster the modules based on area and local connectivity and create clustered modules for the next level.

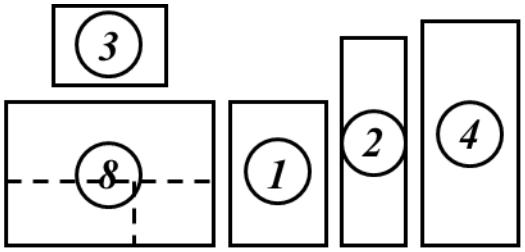
Recursively decluster the clusters and use simulated annealing to refine the floorplan.



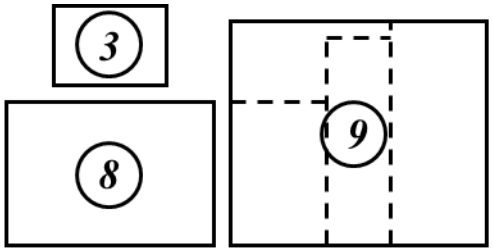
MB*-tree: Multilevel B*-tree



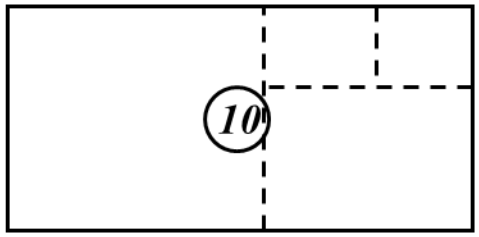
(a) 7 modules



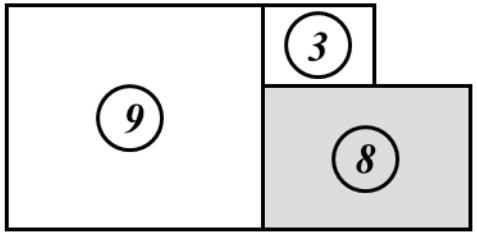
(b) Cluster 5, 6, 7 into 8 based on area & connectivity density; construct a B*-tree subtree



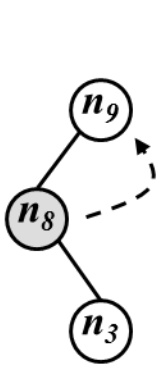
(c) Cluster 1, 2, 4 into 9; construct a B*-subtree

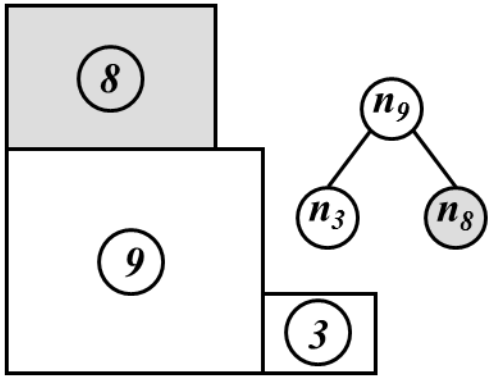


(d) Cluster 3, 8, 9 into 10; Construct a B*-subtree



(e) Decluster 10 to 3, 8, 9



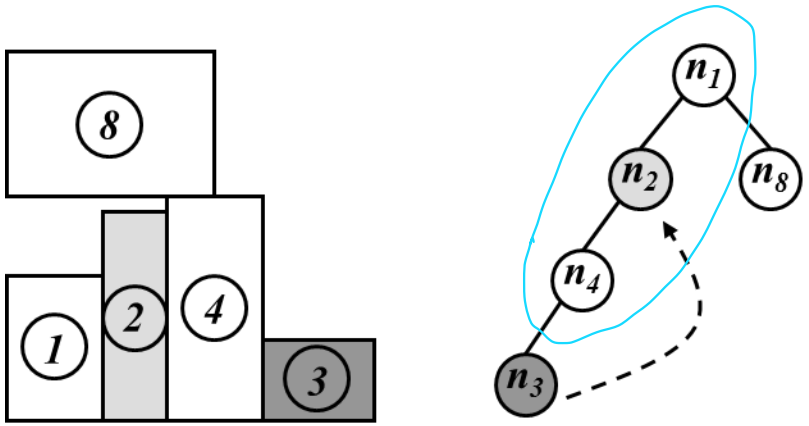


(f) Refine the solution by moving 8

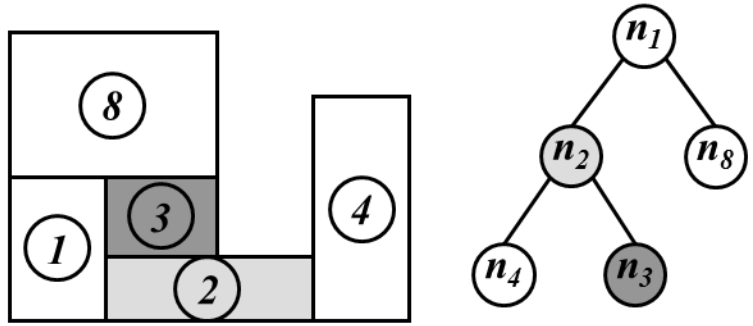
利用 SA 產生新的 B* tree

↳ 這步有點多餘

MB*-tree: Multilevel B*-tree (cont'd)

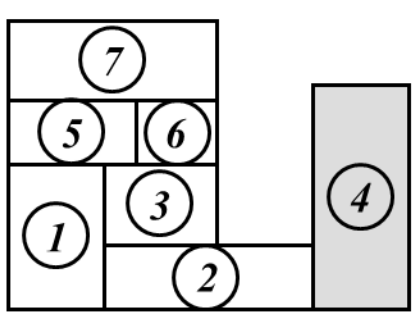


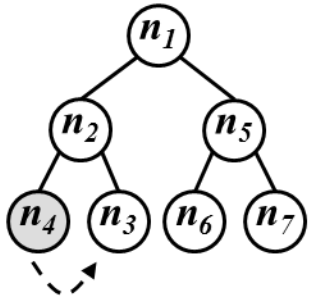
(g) Decluster 9 to 1, 2, 4



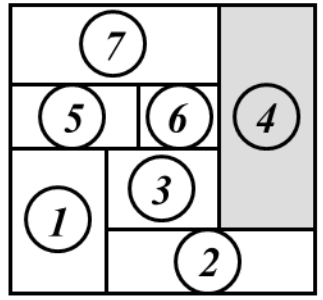
(h) Refine the solution by moving 2, 3

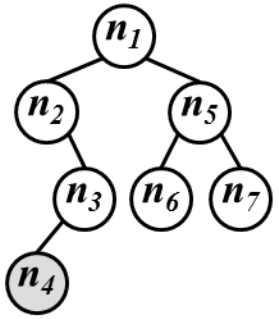
利用 SA 產生新的 B* tree





(i) Decluster 8 to 5, 6, 7





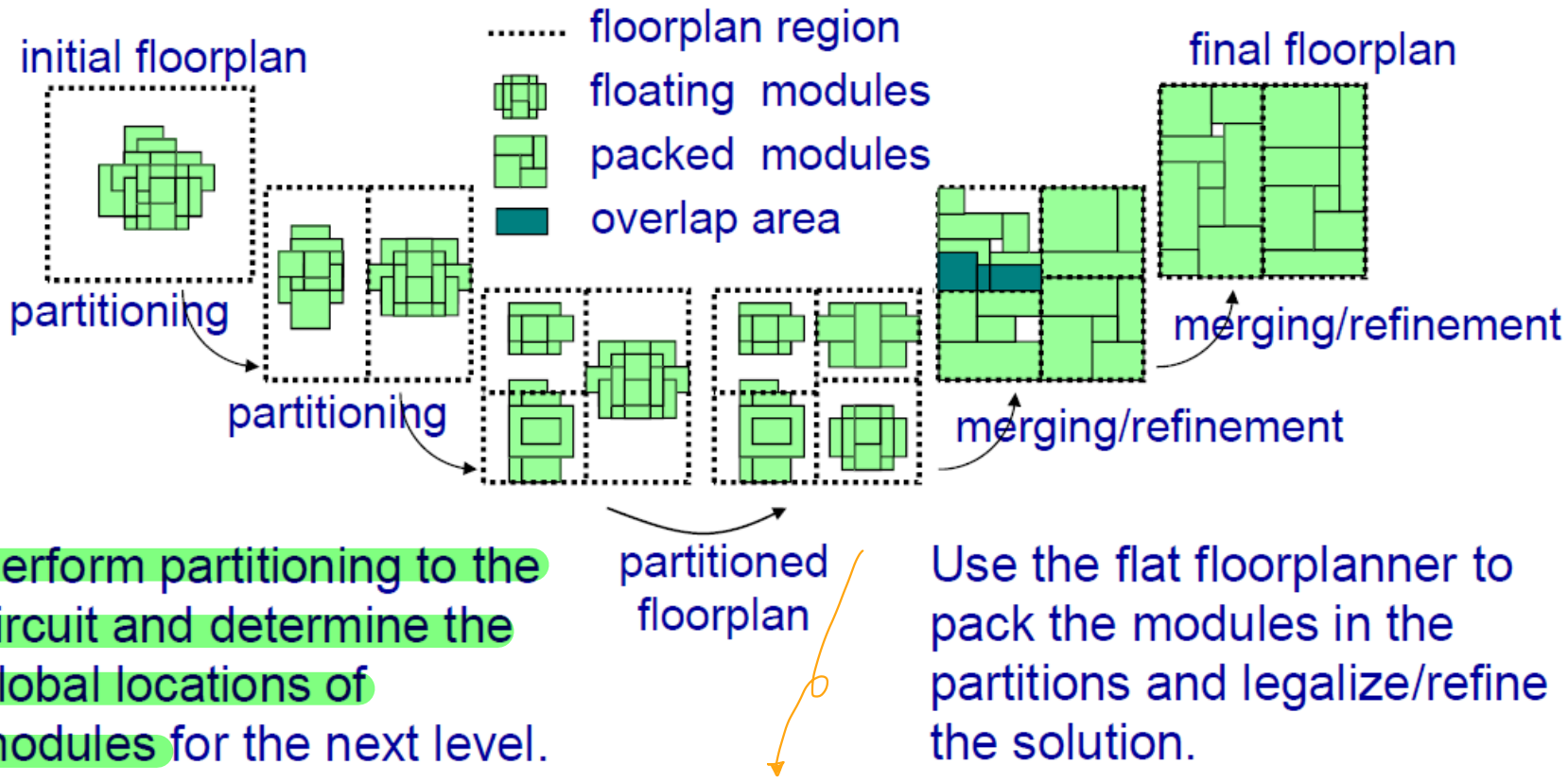
(j) Refine the solution by moving 4

利用 SA 產生新的 B* tree

IMF: V-Shaped Multilevel Floorplanning

做 floorplan 時，要處理的 module 數量非常多。

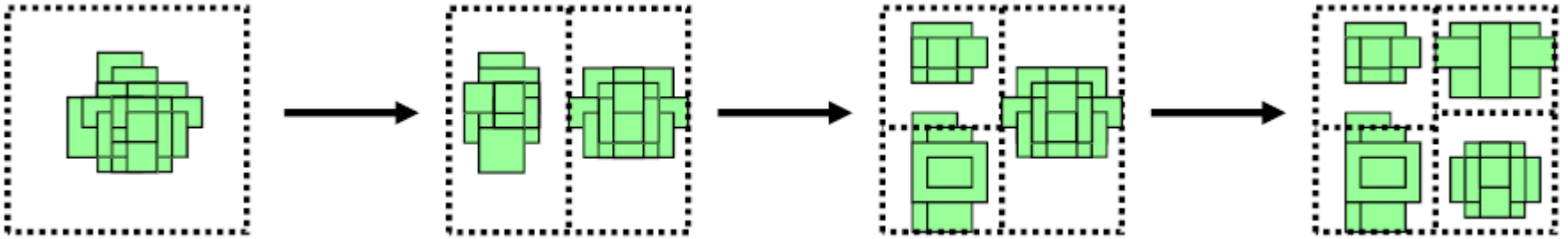
- Chen, Chang, Lin, "IMF: interconnect-driven floorplanning for large-scale building-module designs," ICCAD'05



對每一個 region 的 modules，做 Fixed-outline floorplan
（可能有很多個 module）

Stage 1: Partitioning Stage

- All modules are set to the center of the chip region initially
- Partition the circuit recursively to minimize the interconnect and assign the regions of the modules
- The partitioning stage continues until the number of modules in each partition is smaller than a threshold, and the partitioned floorplan is obtained.



Stage 2: Merging Stage

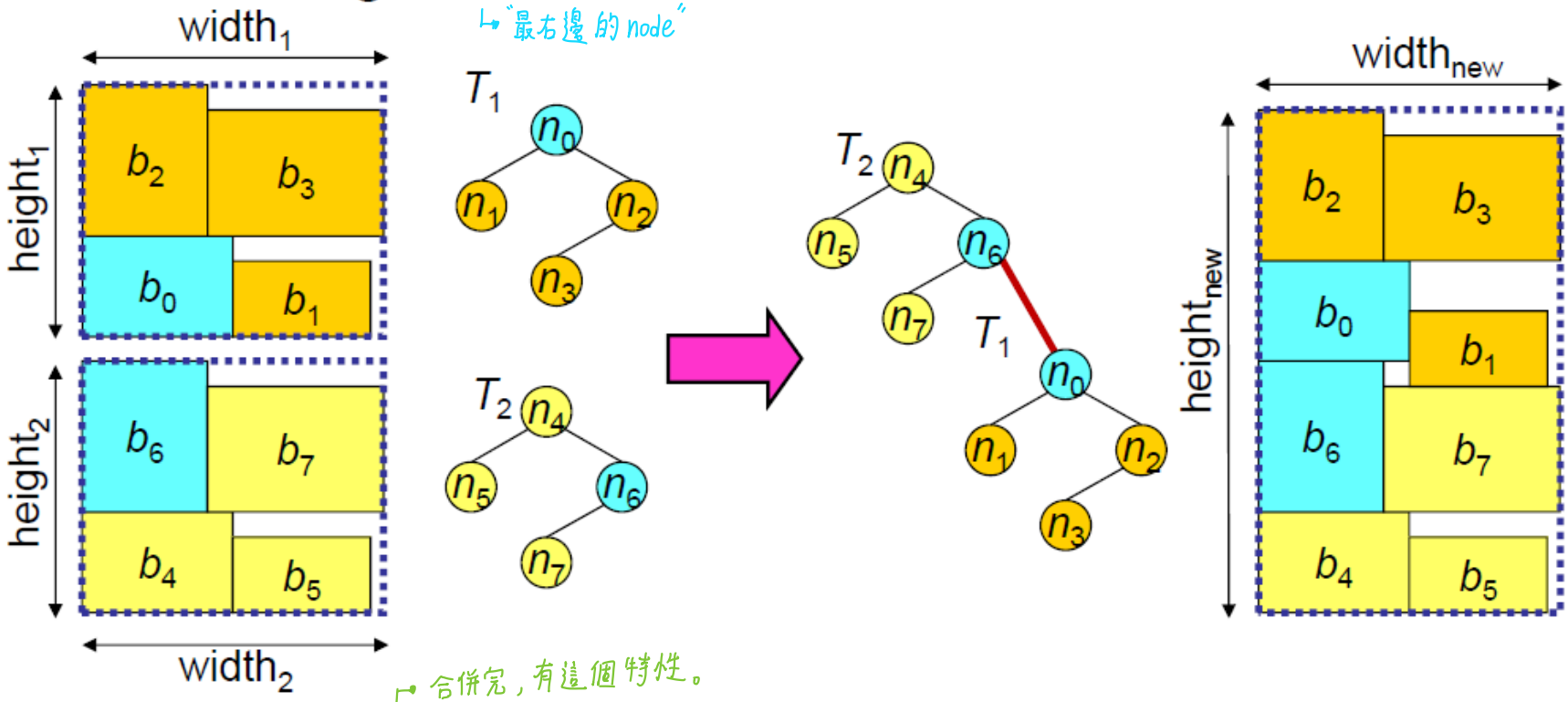
- Construct a B*-tree and find the sub-floorplan for each sub-region (fixed-outline floorplanning)
- Cost function for the simulated annealing: area, wirelength, and aspect ratio penalty
- Merge two B*-trees (sub-floorplans) to form a new B*-tree (floorplan) recursively
- Refine the merged sub-floorplan using fixed-outline floorplanning again

Vertical Merging

↳ 被"水平的 cut"切開，則"垂直合併"。

Make the root of the top B*-tree as the right child of the right-most node of the bottom B*-tree.

↳ "最右邊的 node"

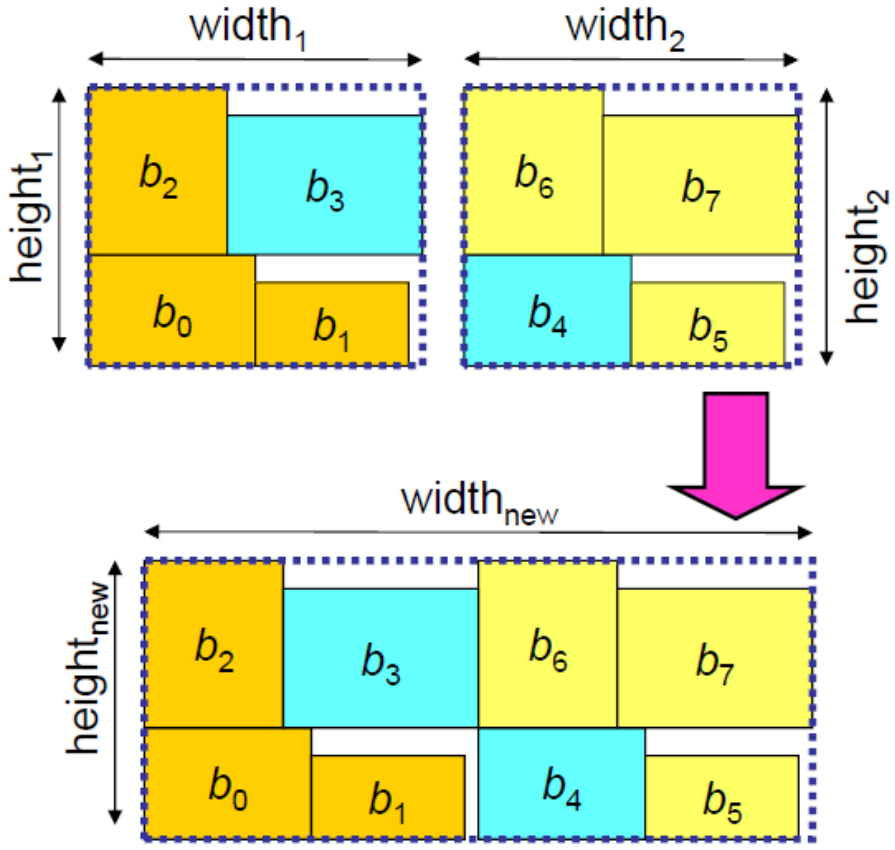


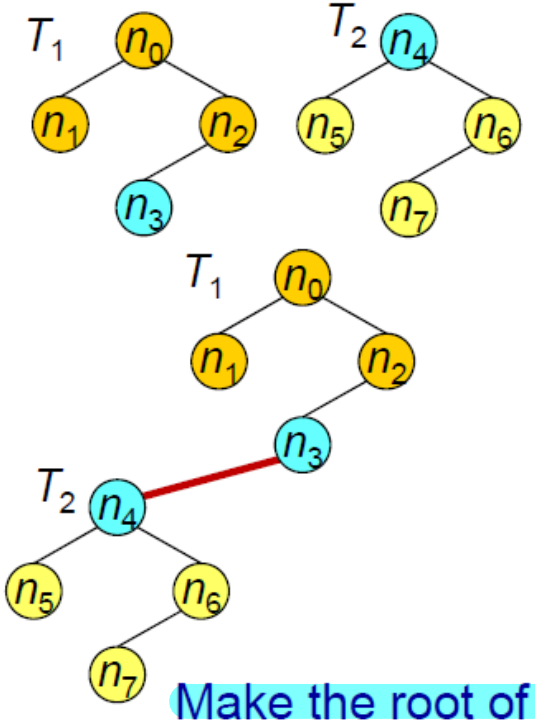
↳ 合併完，有這個特性。

$$height_{new} \le height_1 + height_2$$
$$width_{new} = \max(width_1, width_2)$$

Horizontal Merging

↳ 被"垂直的 cut"切開，則"水平合併"。





$$height_{new} = max(height_1, height_2)$$
$$width_{new} = width_1 + width_2$$

↳ 合併完，有這個特性。

Make the root of the right B*-tree as the left child of the node corresponding to the right-most module of the left B*-tree.

↳ "最右邊的 module"。

早期作法：①先放 modules ② 再把一些 standard cells 當成 soft module 放。⇒ 缺點："最後 standard cell 放的位置不好"。
↳ 沒確定"soft module"內的"standard cells"怎麼放。

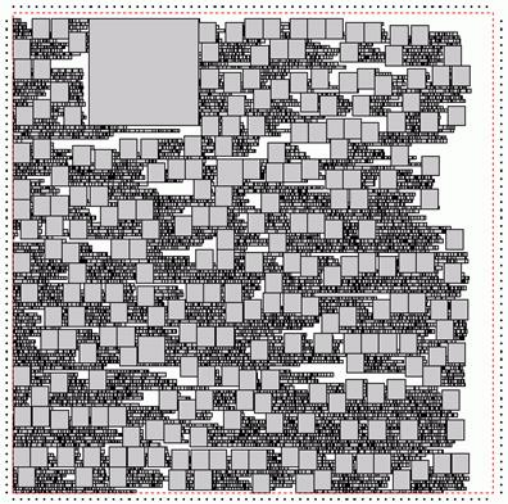
Macro Placement/Floorplanning

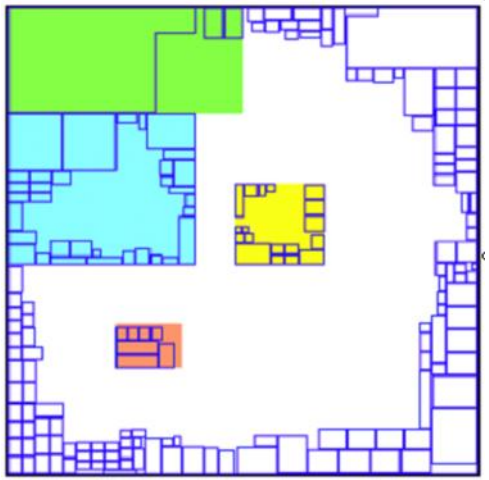
作法：① 同時放 module 和 standard cell，確認大概的位置。② 固定 module 的位置，去除 module 之間的 overlap ③ 再放 standard cell，module 變障礙物。

- Mixed sized cell/block placement/floorplanning: apply floorplanning techniques for macros to address various design constraints, e.g., range constraints, block rotation, block sizing

↳ 設一個區域，限制哪些 module 可以放

↳ 調整 soft module 的大小。





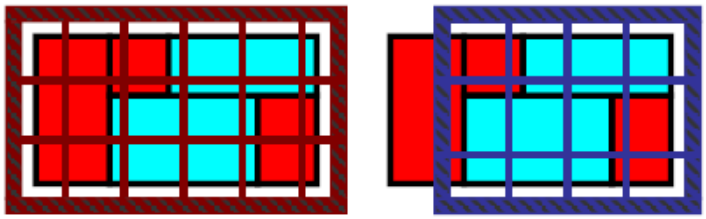
作 floorplan 時，考慮 power supply 形成的 voltage Island

Voltage Island Aware Floorplanning

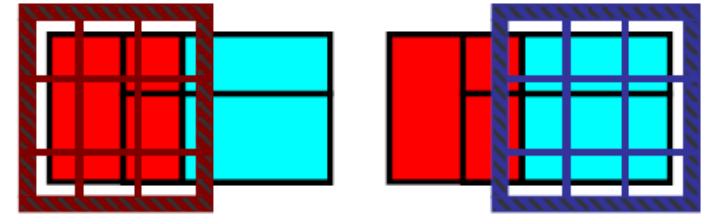
↳ Ex: 1V, 0.8V, 0.75V。⇒"不同 module"需要的 power supply 可能會不同。

- Multiple supply voltages (voltage islands)

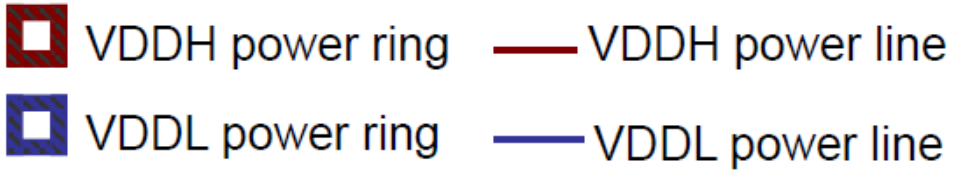
沒考慮 power supply 的 floorplan
↳ module 會形成不同的 Island
↳ "在 metal layer"上的 "power plane"會佔用較多 routing resource。

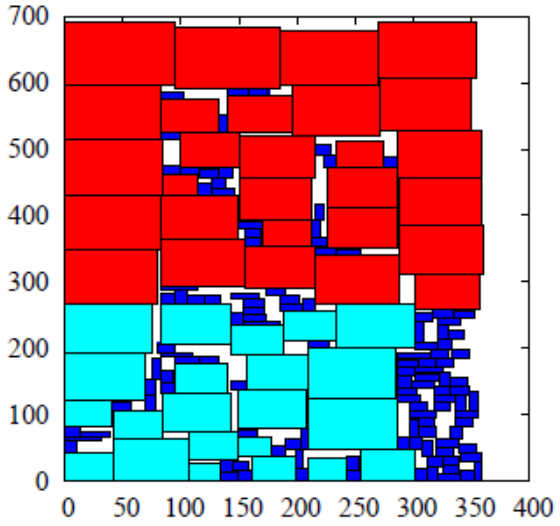


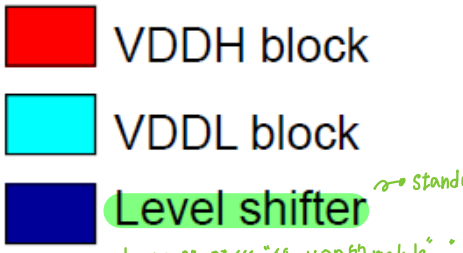
better floorplan



有考慮 power supply 的 floorplan
↳ 相同顏色的 module 會形成 Island
↳ "在 metal layer"上的 "power plane"會佔用較少 routing resource。







↳ standard cell

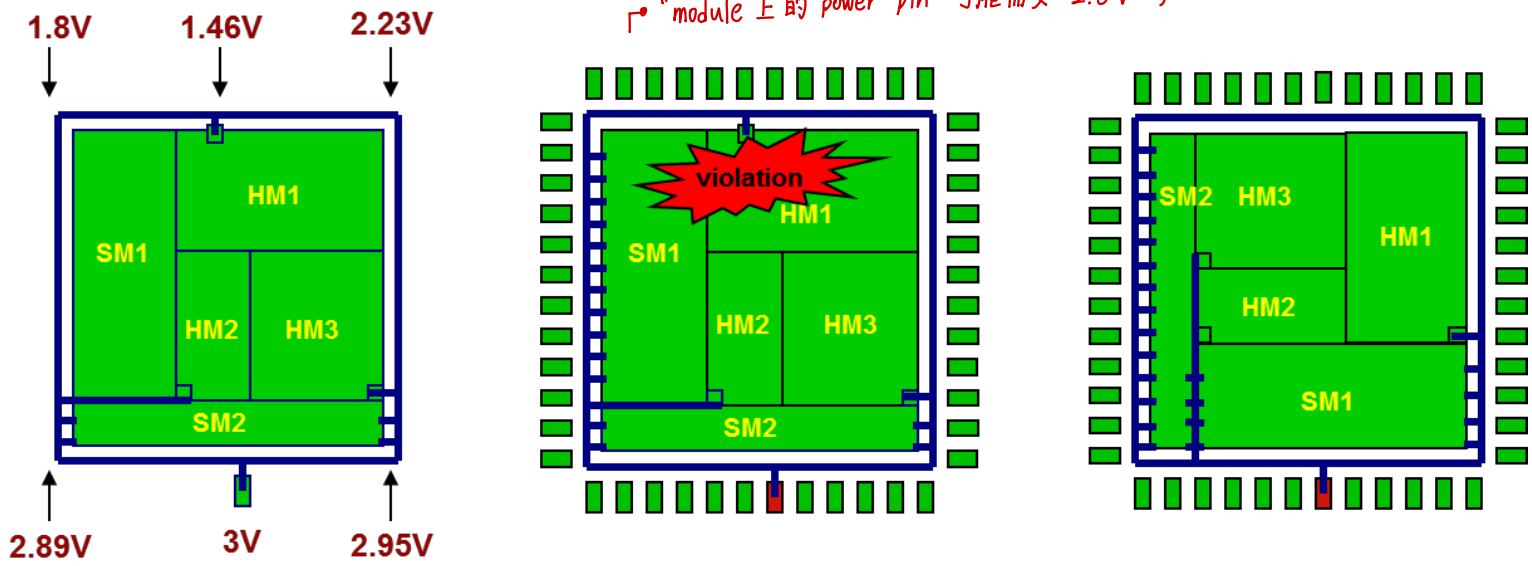
↳ 訊號若從"低 VDD的module","走到"高 VDD的module",則需要"Level shifter"提高電壓。

作 floorplan 時，考慮 voltage drop

Voltage Drop Aware Floorplanning

IR Drop ⇒ Ex: 從"3V的 power supply"，流入金屬線，金屬線會有"電流(I)，電阻(R)"。
∴在不同位置上的金屬線，不一定能拿到 3V 的電壓。(離 power supply 愈遠，電壓可能愈低)

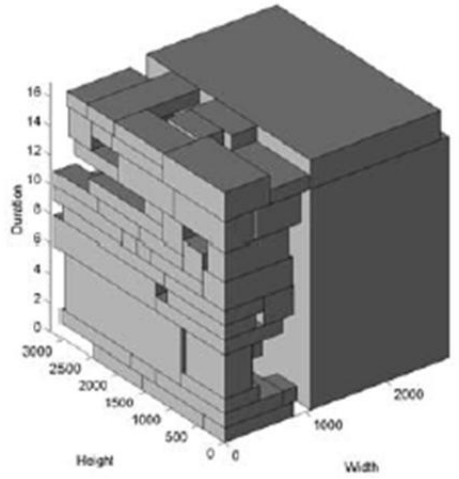
- Power/ground networks for static/dynamic IR drop minimization (voltage drop aware floorplanning)



"module 上的 power pin" 可能需要"2.5V"，金屬線可能只剩"0.8V"。

Beyond 2D Floorplanning

- Floorplanning for reconfigurable computing

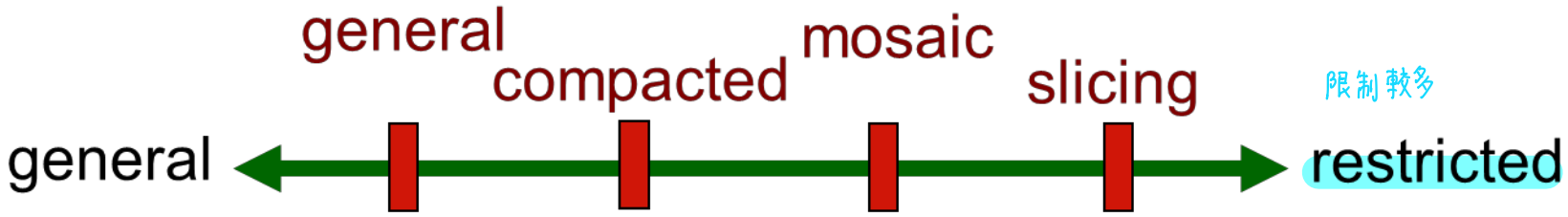


- Floorplanning for digital microfluidic biochips
- SiP/2.5D/3D floorplanning

2025 PDA，13葉，1:29:29

Existing Floorplan Representations

- Slicing: slicing tree, normalized Polished expression
- Mosaic: corner block list (ICCAD'00), twin binary tree (ISPD'01)
- Compacted: O-tree, B*-tree, corner sequence (TVLSI'03)
- General: sequence pair, bounded-sliceline grid (ICCAD'96), transitive closure graph (DAC'01), TCG-S (DAC'02), adjacent constraint graph (ICCD'04)



Comparison

從"floorplan表示法"，得到 floorplan，需要的時間

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

Existing Floorplanning Problems

- Outline free (variable die) ~ 在意"面積"
- Fixed outline (fixed die)
- Hard modules only
- Soft (and hard) modules
- Large scale
- Mixed size ~ 考慮 module 和 standard cell
- Pre-placed modules ~ 有些 module 已確定位置，有些 module 還沒
- Range-constrained modules ~ module 放置的區域會被限制
- Boundary-constrained modules ~ module 要被放在邊界
- Abutment-constrained modules ~ 有些 modules 要貼在一起放。
- Symmetry-constrained modules ~ 有些 modules 要沿著"對稱軸"放。⇒會出現在 analog placement。
- Rectilinear modules ~ module 的形狀：只有"水平邊，垂直邊"，"邊的數量 > 4"
- Analog placement
- Beyond 2D ~ 多個 Die 之間的 floorplan。
  Ex: 2.5D, 3D

Existing Floorplanning Problems (cont'd)

- Co-synthesis with
  - Voltage islands
  - Power supply planning (voltage drop)
  - Interconnect planning
  - Bus planning
  - Buffer planning
  - Noise
  - SoC test scheduling
  - Micro-architecture pipelining
  - Double patterning

作 floorplan 時，同時可以考慮的 issue

訊號之間的 interconnect

考慮 address bus

有 2 種：① ② → 要有"偶數個"訊號不能錯

有時候需要在訊號線上插 buffer。線太長，導致訊號傳太慢

DFM 相關的東西。