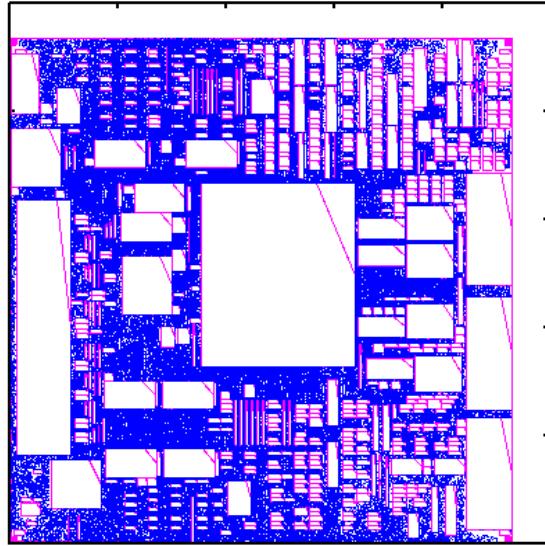
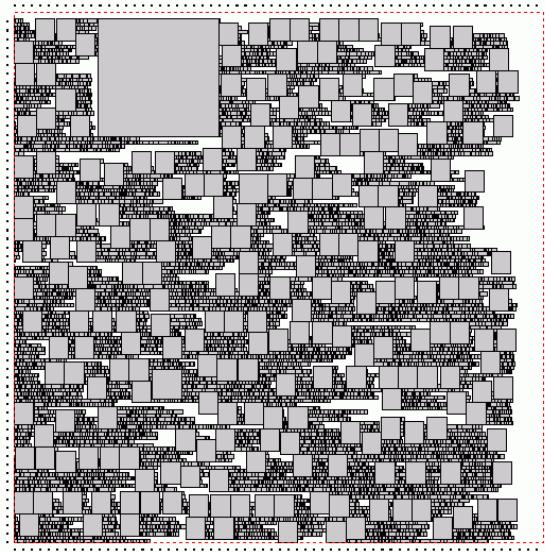
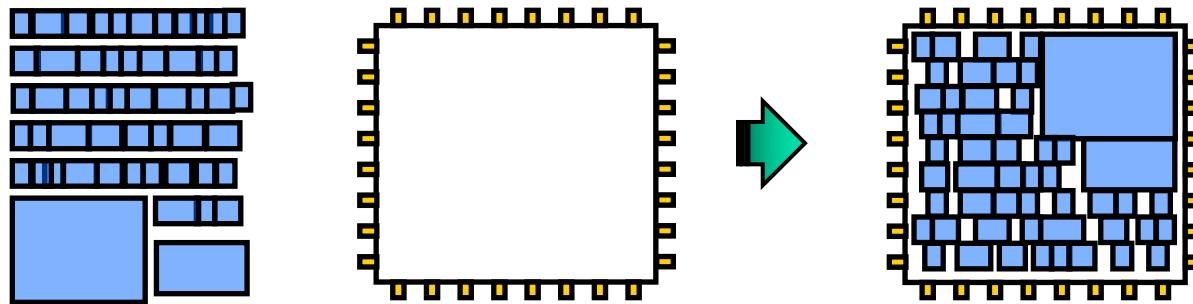


Unit 5: Placement

- Course contents:
 - Partitioning-based method
 - Min-cut placement
 - Nondeterministic methods:
 - Simulated annealing (SA)
 - Genetic algorithm (GA)
 - Analytical methods:
 - Force-directed placement
 - Quadratic placement (QP)
 - Non-quadratic placement
 - Mixed-size placement
 - Appendix: Linear assignment placement
- Readings
 - W&C&C: Chapter 11
 - S&Y: Chapter 4



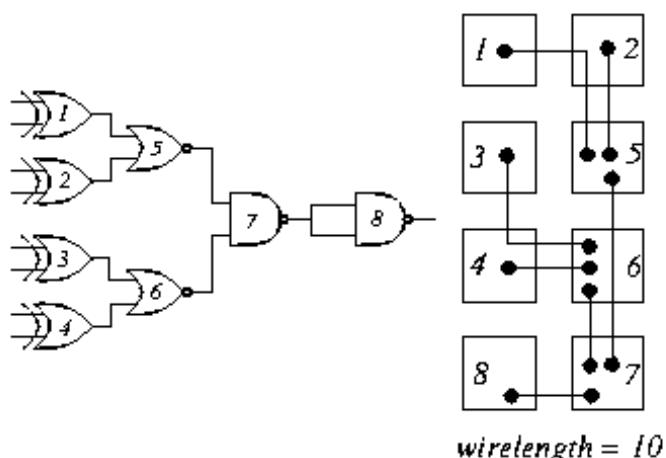
VLSI Placement



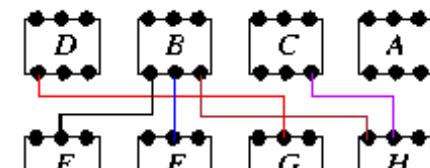
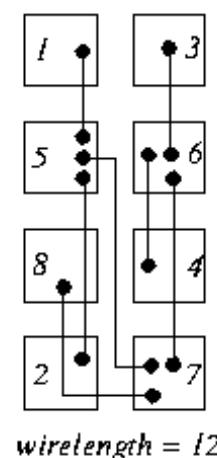
- **Placement:** assign cells to positions on the chip, such that no two cells overlap with each other (**legalization**) and some cost function (e.g., wirelength) is optimized.
- A major step in physical design that has been studied for 40+ years.
 - *EETimes* (4/10/2003): 1.46--2.38 X from the optimal wirelength
 - Is still far away from optimal??
- More than 16 new academic placers since 2000
- ACM ISPD Placement Contests in 2005 and 2006

Placement Problem

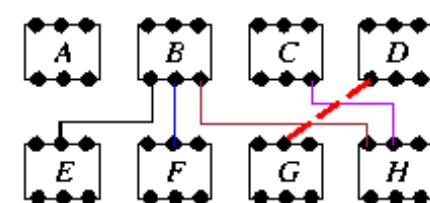
- Inputs: A set of **fixed** cells/modules, a netlist.
- Goal: Find the best position for each module on the chip according to appropriate cost functions.
 - Considerations: **placement**也要考慮後續的**routing**可行性, **routability/channel density**, **wirelength**, power, timing, thermal, I/O pads, manufacturability, etc.



相同的面積下 · 以wire length 評估 ·
左側結果較好



Density = 2 (2 tracks required)

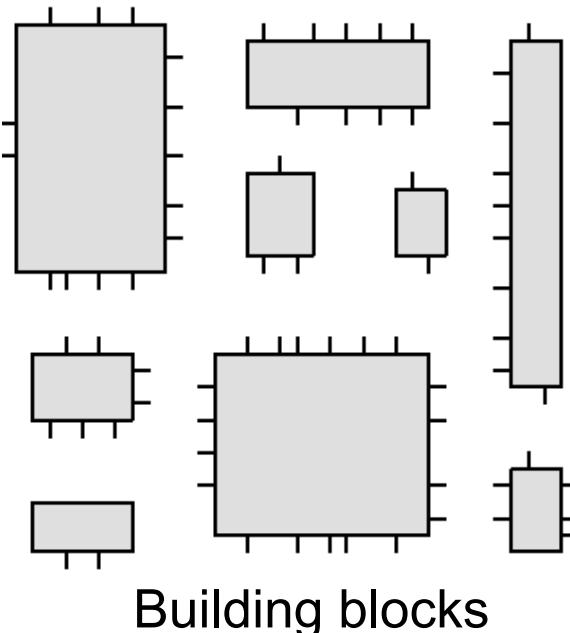


Shorter wirelength, 3 tracks required.

中間只有兩個channel的話 · 以
下方的placement無法繞線

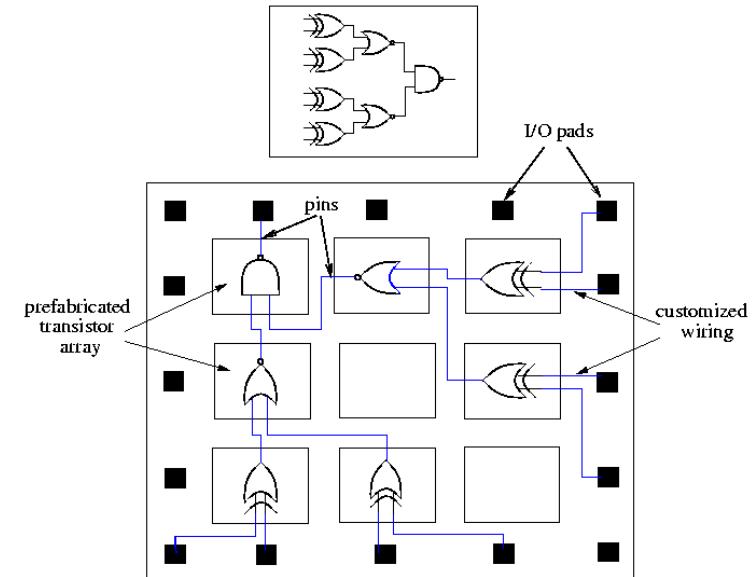
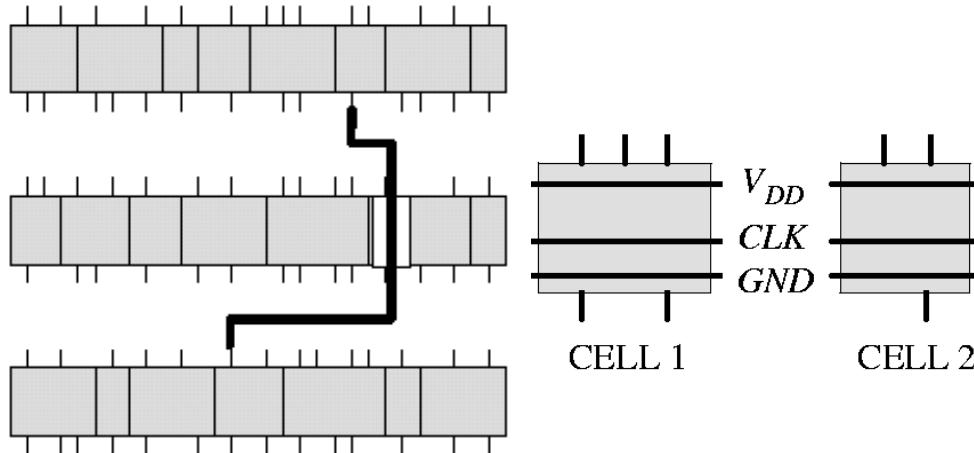
Placement Style: Building Blocks

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



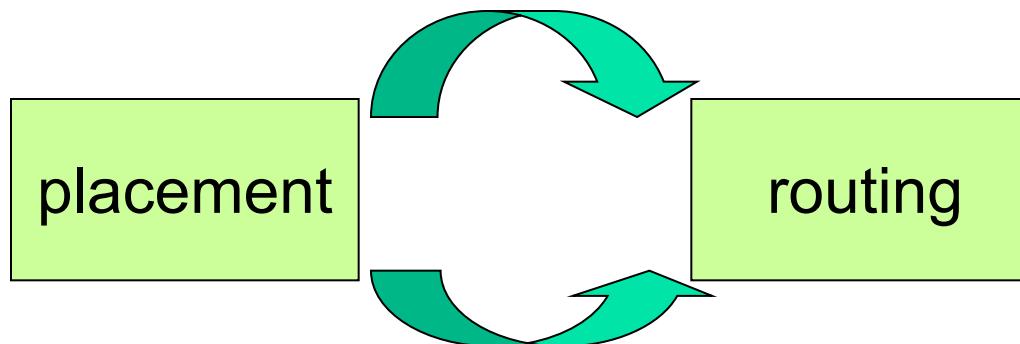
Placement Styles: Standard Cell & Gate Array

- Standard cells: power and clock connections run horizontally through the cell, and other I/O leaves the cell from the top or bottom sides.
 - The cells are placed in rows.
 - Sometimes *feedthrough* cells are added to ease wiring.
- Gate array: placement should be able to deal with fixed channel capacities



Relation with Routing

- Ideally, placement and routing (P&R) should be performed simultaneously as they depend on each other's results; it, however, often too complicated.
 - Pan and Chu, “IPR: An integrated placement and routing algorithm,” DAC-07.
- Approximation: Placement estimates the wire length of a net using some *wirelength metric*.



placement與routing其實應該要同時並進，但是placement已經很複雜了，再加上routing會更難，因此實際在作的時候都是先做placement

Basic Wirelength Models

- **Half-perimeter wirelength (HPWL):** Half the perimeter of the bounding rectangle that encloses all the pins of the net to be connected. Most widely used approximation! 用最上下左右的點畫出一個矩形，看這個矩形的半周長(長+寬)-->簡單好用又快速
- **Squared Euclidean distance:** squares of all pairwise terminal distances in a net using a quadratic cost function

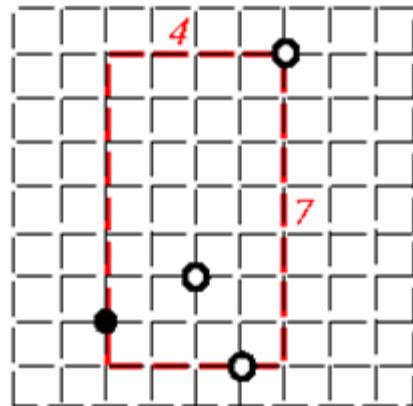
$$\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \gamma_{ij} [(x_i - x_j)^2 + (y_i - y_j)^2]$$

相對來說計算時間較久，此方法較不實用

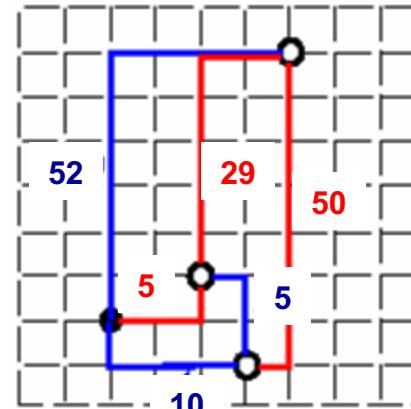
- routing的時候才比較適用steiner tree(用steiner tree才能找出最佳解)，在placement的時候會有點太花時間
- **Steiner-tree approximation:** Computationally expensive.
spanning tree的準確度會比HPWL更好，但是還是有點太花時間了，所以大家還是用HPWL
 - **Minimum spanning tree:** Good approximation to Steiner trees.
 - **Complete graph:** Since #edges in a complete graph is $\binom{n(n-1)}{2} = \frac{n}{2} \times \# \text{ of tree edges } (n-1)$, $wirelength \approx \frac{2}{n} \sum_{(i,j) \in net} dist(i,j)$.

Wirelength Estimation Examples

可以發現HPWL估算出來的結果與steiner很接近，而且實作起來簡單，因此被公認是最有效的做法

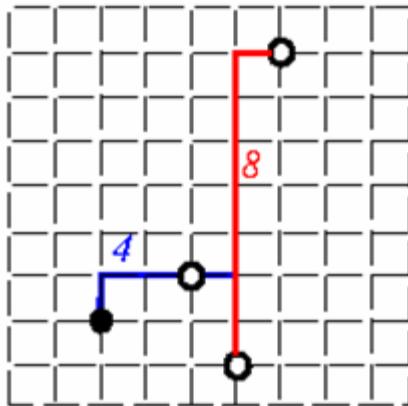


half perimeter
wirelength = 11

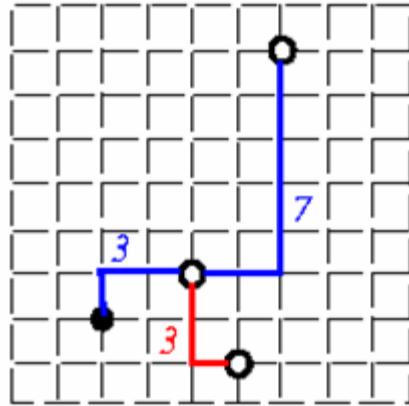


squared Euclidean
distance = 15.1 ($\gamma_{ij} = 0.1$)

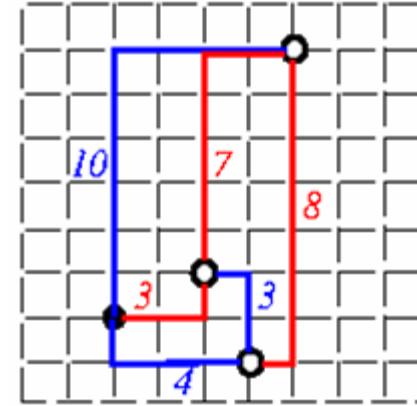
最精確的解答



min Steiner tree
wirelength = 12



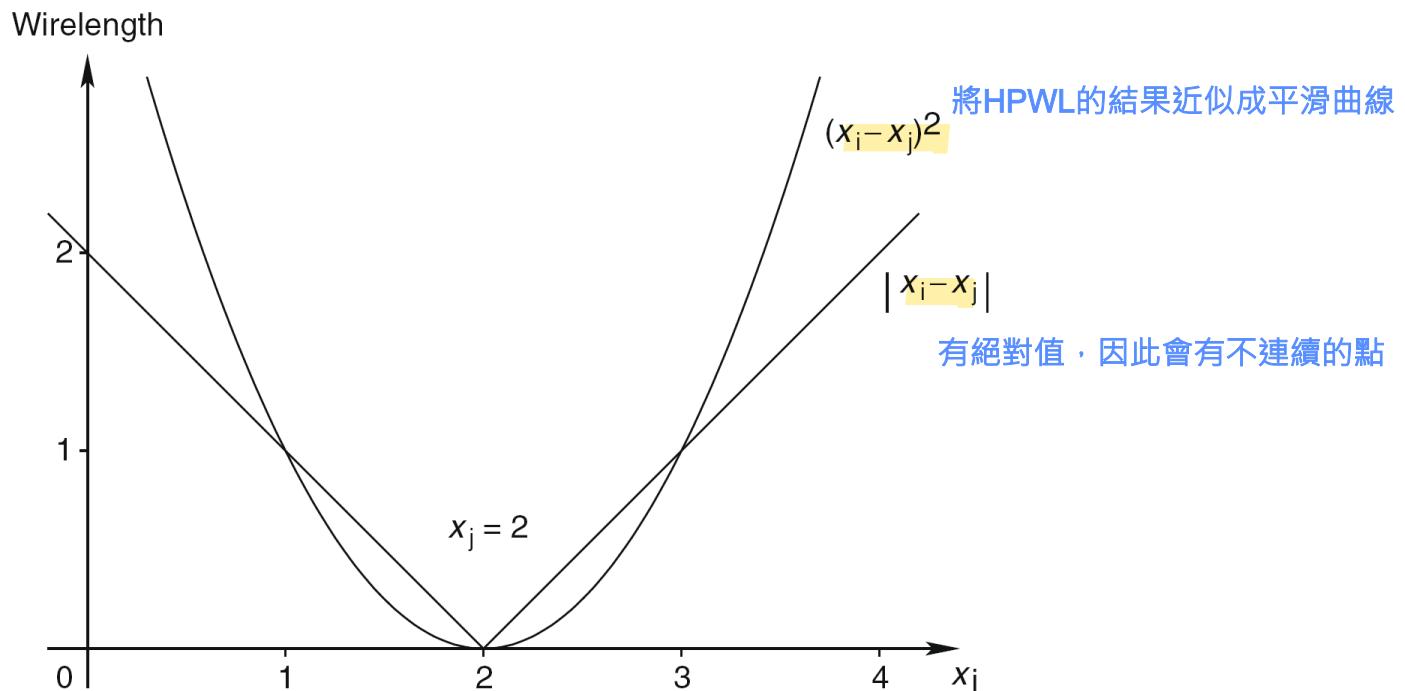
min spanning tree
wirelength = 13



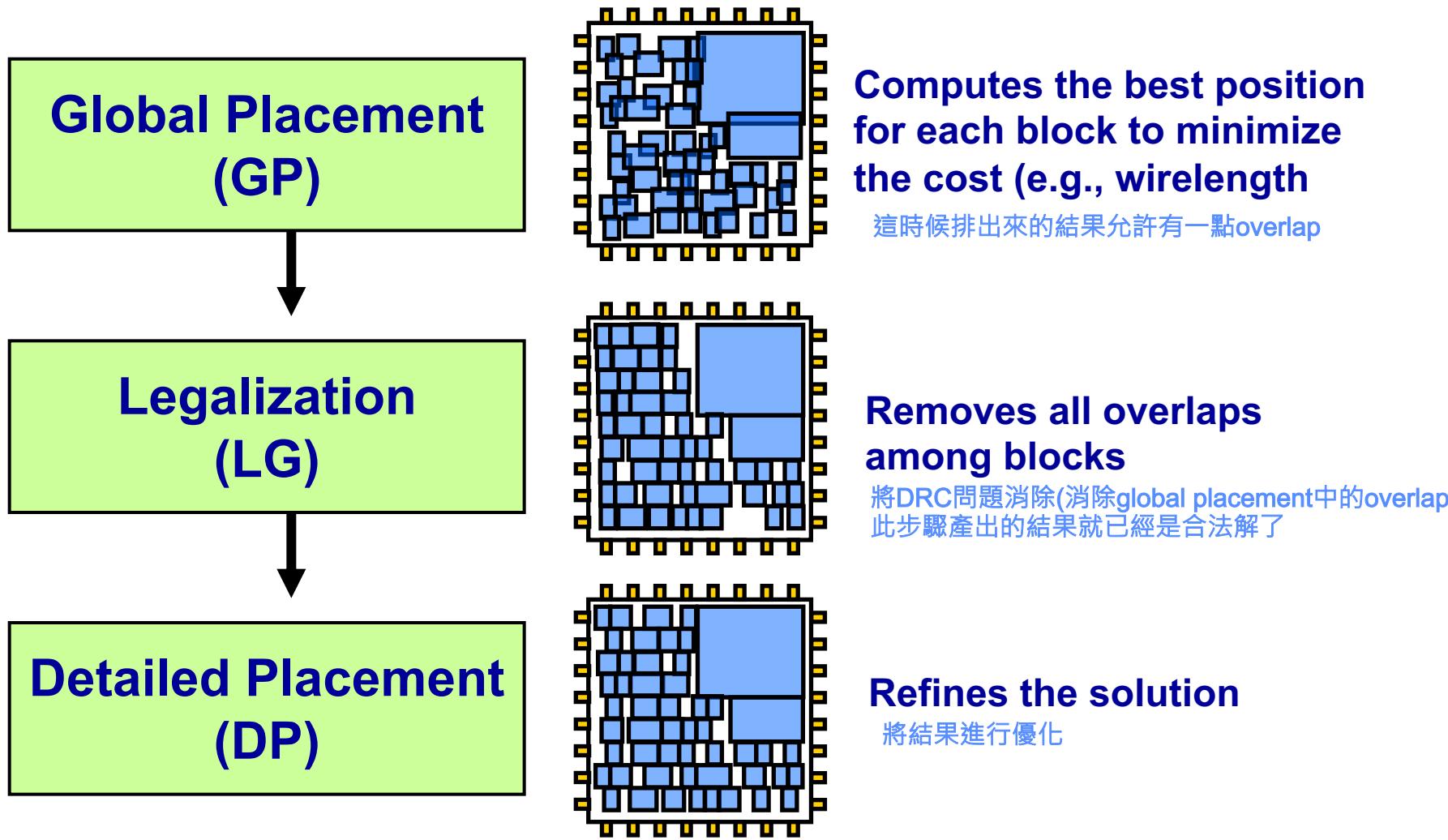
complete graph
wirelength * 2 / n = 17.5

Quadratic Wirelength vs. Linear HPWL

- HPWL is convex, but not differentiable
HPWL的曲線會是不平滑的折線(因為有絕對值) · 不可微-->因此會將其結果近似成另外一條平滑曲線
- Quadratic wirelength is a convex, smooth function → easy to minimize
- Both correlate “well” with each other



Typical Placement Flow

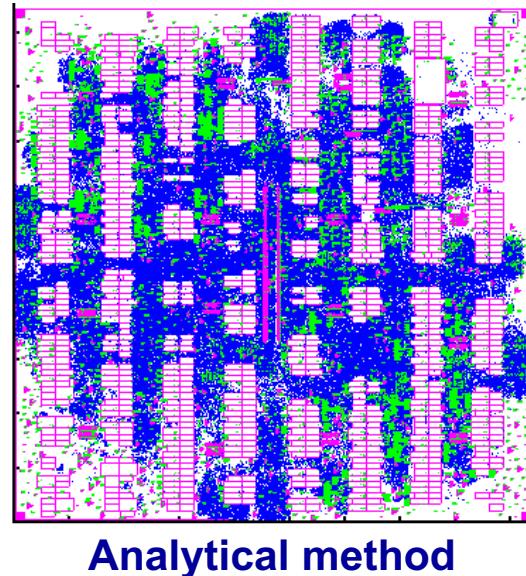
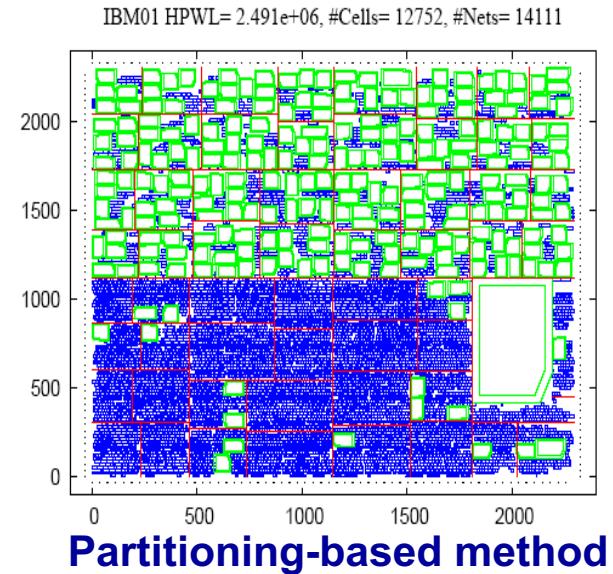


Global Placement Algorithms: Operation Perspective

- The placement problem is NP-complete.
- Popular placement algorithms:
 - **Constructive algorithms:** once the position of a cell is fixed, it is not modified anymore. cell一擺上去就不再動
 - Cluster growth, linear assignment, min cut, QP, etc.
 - **Iterative algorithms:** intermediate placements are modified in an attempt to improve the cost function.
 - Force-directed method, etc 將某一次步驟的最佳解作為下次iteration的初始狀態，不斷迭代-->ex: partition的FM
 - **Nondeterministic approaches:** simulated annealing, genetic algorithm, etc. 也算是iterative的演算法，只是他們包含隨機的因素在裡面
- Can combine multiple elements:
 - Constructive algorithms are used to obtain an initial placement.
 - The initial placement is followed by iterative improvement.
 - The results can further be improved by simulated annealing.

Global Placement Algorithms: Algorithmic Perspective

- Partitioning-based method
 - Min-cut placement
- Nondeterministic methods
 - Simulated annealing (SA)
 - Genetic algorithm (GA)
- Analytical methods
 - Force-directed placement
 - Quadratic placement (QP)
 - Non-quadratic placement

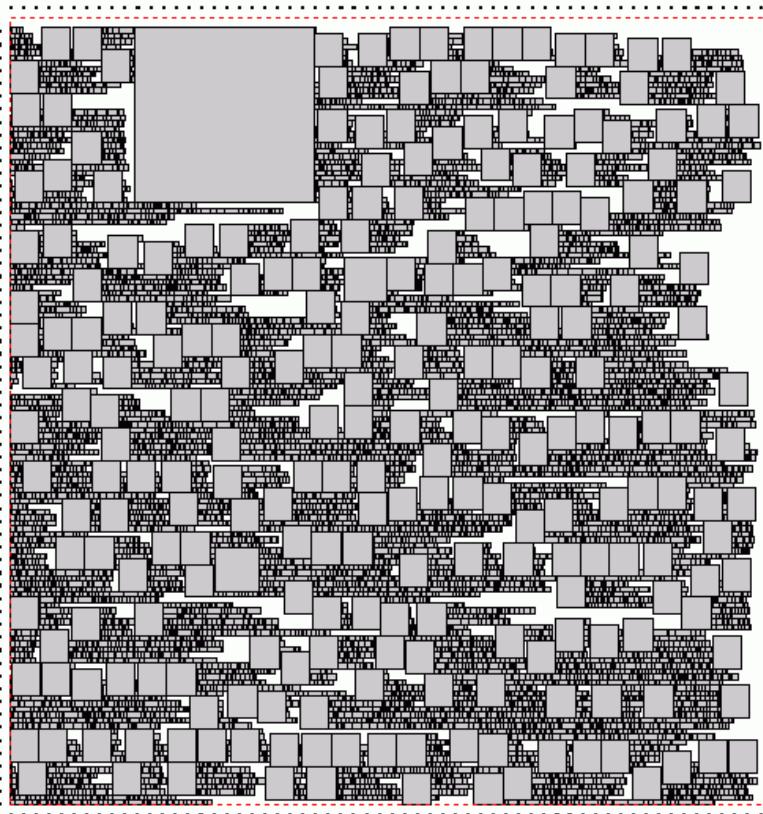


Example Academic Placers

- Partitioning/clustering based placers
 - NTUplace1 (ISPD-05): <http://eda.ee.ntu.edu.tw/research.htm>
 - Capo (DAC-00, ICCAD-04, ISPD-05, ISPD-06):
<http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Placement/Capo/>
 - FungShui (ISPD-04, -05)
- Simulated annealing based placers
 - Dragon (ICCAD-00, ISPD-05, ISPD-06): <http://er.cs.ucla.edu/Dragon/>
- Analytical placers
 - GORDIAN (ICCAD-90)
 - Aplace (ICCAD-04, ISPD-05, ICCAD-05, ISPD-06)
 - Kraftwerk (DAC-98), Kraftwerk&Domino (ISPD-05, ICCAD-06)
 - mPL (ICCAD-03, ISPD-05, ISPD-06)
 - FastPlace (ISPD-04, -05, ASPDAC-06, ASPDAC-07)
 - NTUplace3 (ICCAD-06)
 - RQL (DAC-07)
- Hybrid placers: NTUplace2 (ISPD-06; partitioning + analytical), Dragon (ISPD-05; SA + partitioning)

Example Placements

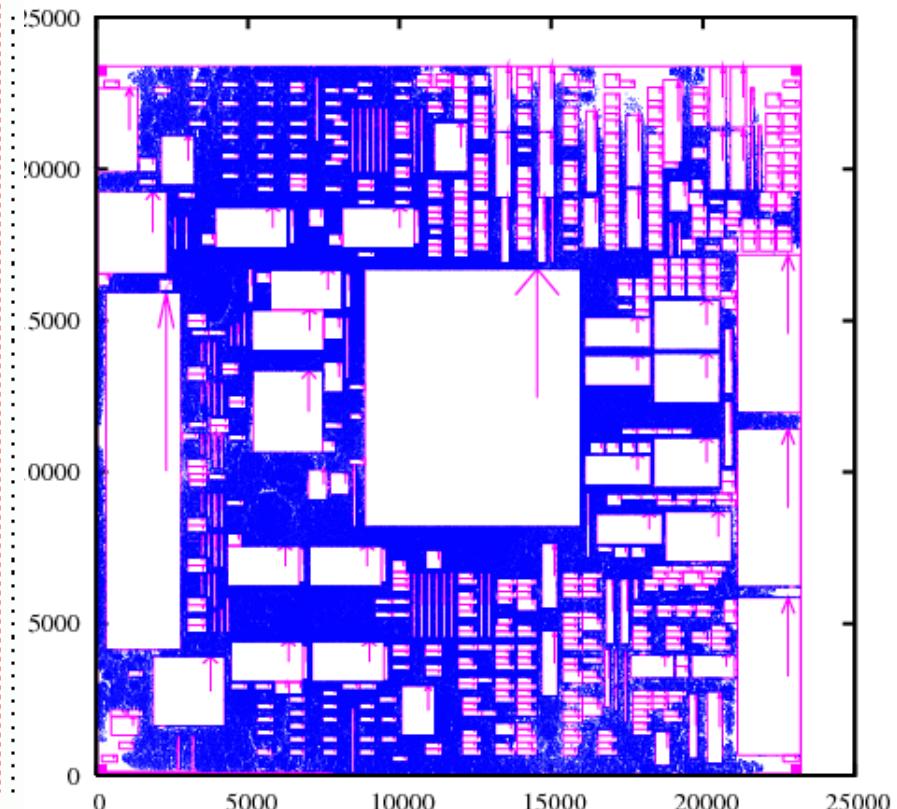
ISPD98 ibm01



12,752 cells, 247 macros
Amax/Amin = 8416

ISPD'06

adaptec5.plt, block= 843224, net= 867798, HPWL= 387222315

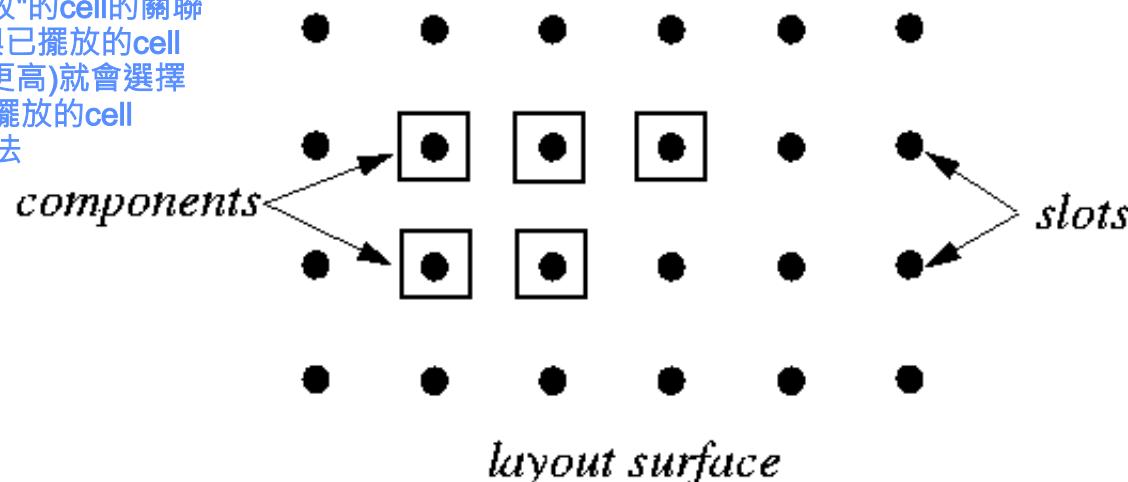


842K movable cells
646 fixed macros
868K nets

Bottom-Up Placement by Cluster Growth

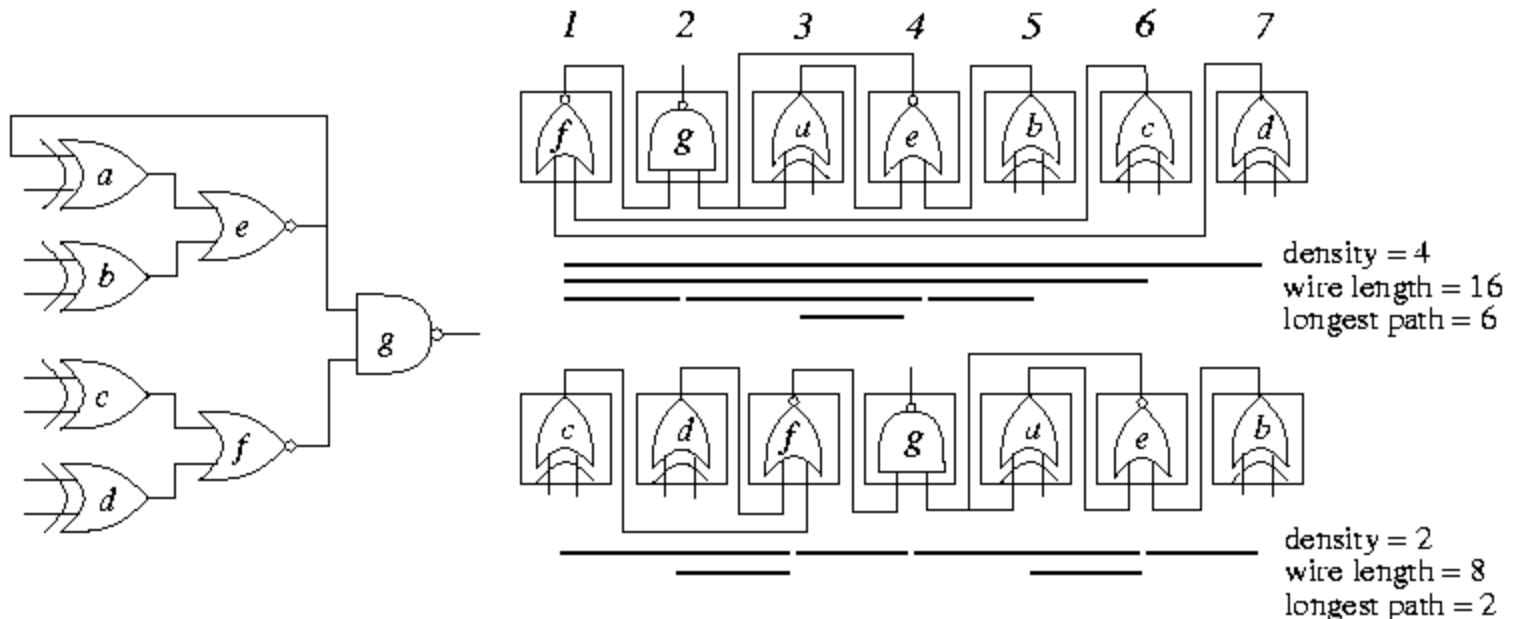
- **Greedy method:** Selects unplaced components and places them in available slots.
 - **SELECT:** Choose the unplaced component that is most strongly connected to all of the placed components (or most strongly connected to any single placed component).
 - **PLACE:** Place the selected component at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

SELECT: 從還沒擺放的cell中挑一個出來，會根據與“已擺放”的cell的關聯性來進行選擇，如果與已擺放的cell有更多的連線(關聯性更高)就會選擇這個cell做為下一個要擺放的cell
(greedy) --> 不好的方法



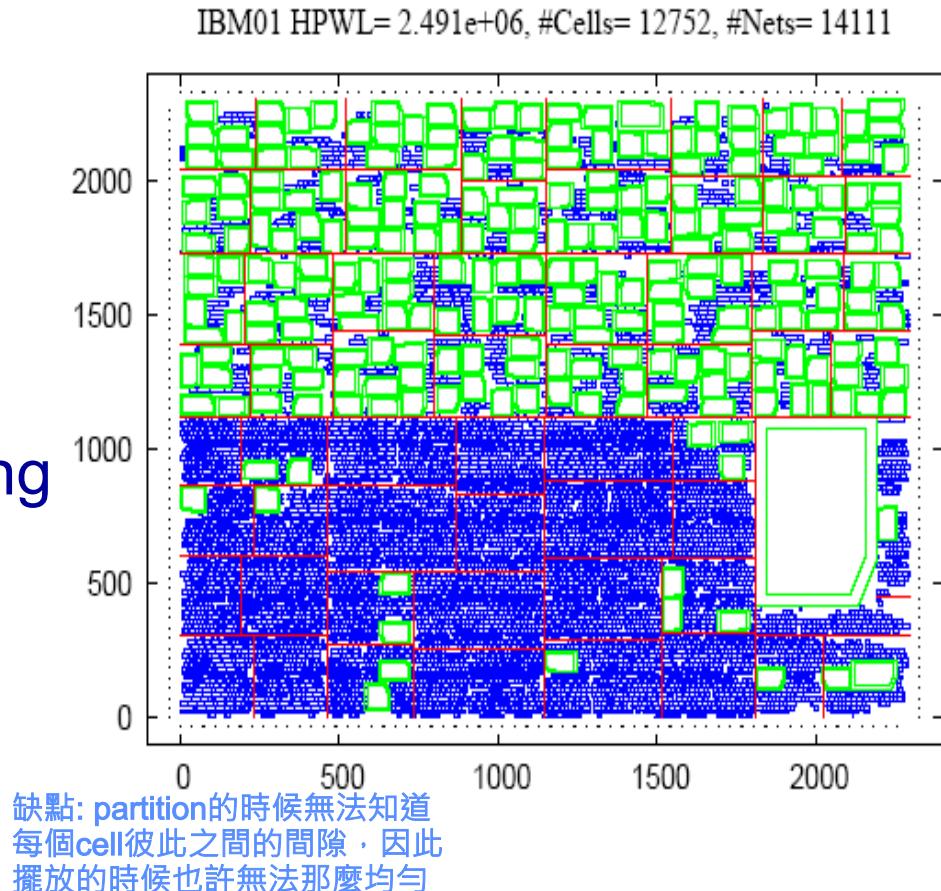
Cluster Growth Example

- # of other terminals connected: $c_a=3$, $c_b=1$, $c_c=1$, $c_d=1$, $c_e=4$, $c_f=3$, and $c_g=3 \Rightarrow e$ has the most connectivity.
- Place e in the center, slot 4. a, b, g are connected to e , and $\hat{c}_{ae} = 2, \hat{c}_{be} = \hat{c}_{eg} = 1 \Rightarrow$ Place a next to e (say, slot 3). Continue until all cells are placed.
- Further improve the placement by swapping the gates.



Partitioning Based Placement

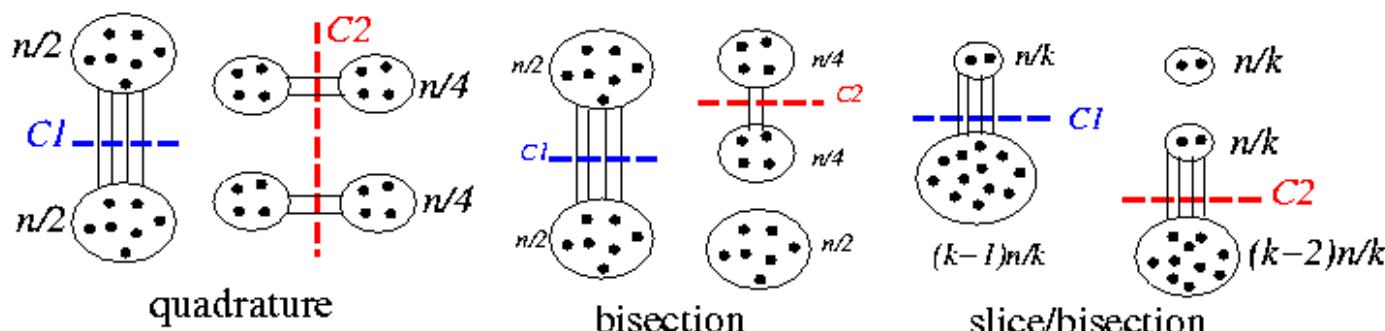
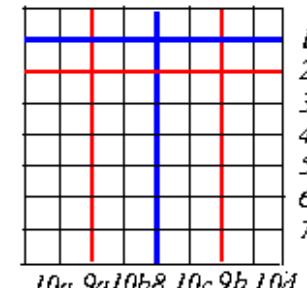
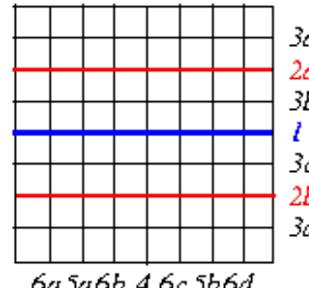
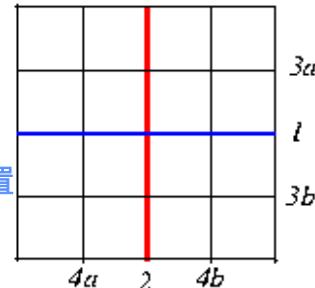
- Rely on a good partitioner (e.g., hMetis) and partition metric (e.g., exact net-weight modeling)
- Pros
 - 直接把Hmetis的結果拿來用-->好快
 - Is very fast
 - Has very good scalability for handling large-scale designs
- Cons
 - Relatively harder to handle design with large white spaces



Top-Down Min-Cut Placement

- Breuer, “A class of min-cut placement algorithms,” DAC-77.
- **Quadrature:** suitable for circuits with high density in the center. 一刀直的一刀橫的 · 交錯切
- **Bisection:** good for standard-cell placement. 水平切
- **Slice/Bisection:** good for cells with high interconnection on the periphery. 一次切一小段(不是切在中間) · 不是均勻地切割

遞迴的方式去進行切割，
邊切邊決定每個cell的位置



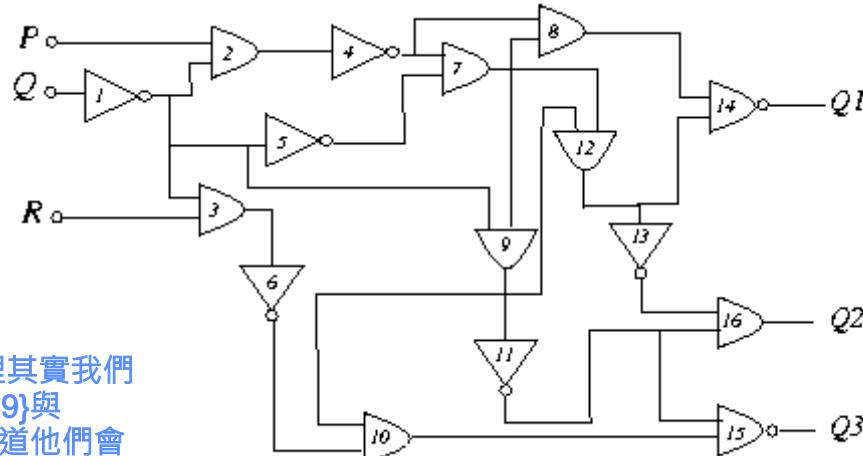
Algorithm for Min-Cut Placement

```
Algorithm: Min_Cut_Placement( $N$ ,  $n$ ,  $C$ )
/*  $N$ : the layout surface */
/*  $n$  : # of cells to be placed */
/*  $n_0$ : # of cells in a slot */
/*  $C$ : the connectivity matrix */

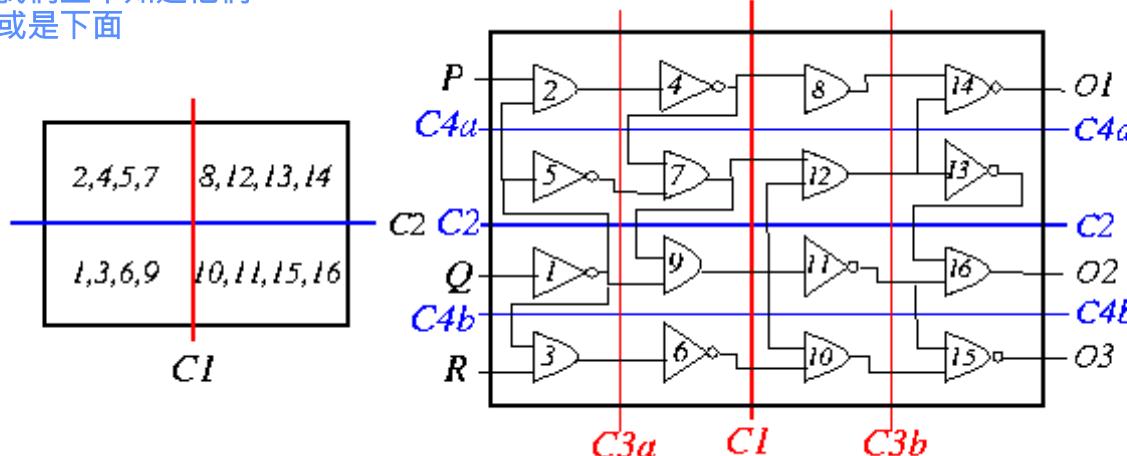
1 begin
2 if ( $n \leq n_0$ ) then PlaceCells( $N$ ,  $n$ ,  $C$ ) 切到大小足夠小的時候就可以直接擺進去(不用再切)
3 else
4    $(N_1, N_2) \leftarrow \text{CutSurface}(N);$ 
5    $(n_1, C_1), (n_2, C_2) \leftarrow \text{Partition}(n, C);$ 
6   Call Min_Cut_Placement( $N_1$ ,  $n_1$ ,  $C_1$ );
7   Call Min_Cut_Placement( $N_2$ ,  $n_2$ ,  $C_2$ );
8 end
```

Quadrature Placement Example

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



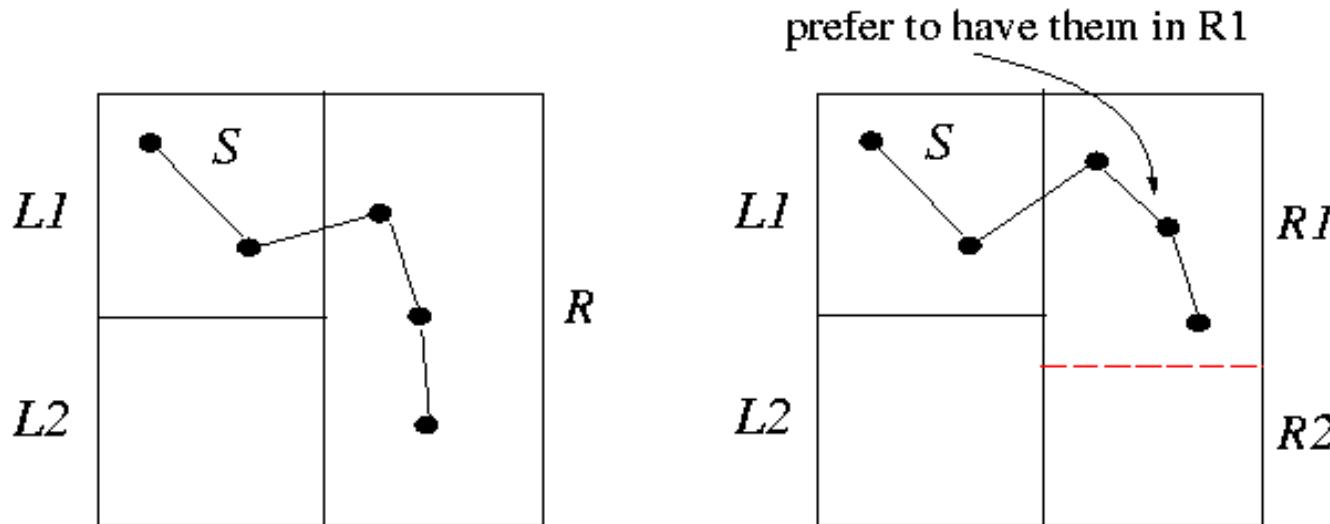
切割出來的東西該擺在哪裡其實我們是不知道的，以圖中 $\{1,3,6,9\}$ 與 $\{2,4,5,7\}$ 來說，雖然我們知道他們會被分成兩半，但是我們並不知道他們兩個誰該擺在上面或是下面。



What if we swap $\{1, 3, 6, 9\}$ and $\{2, 4, 5, 7\}$??

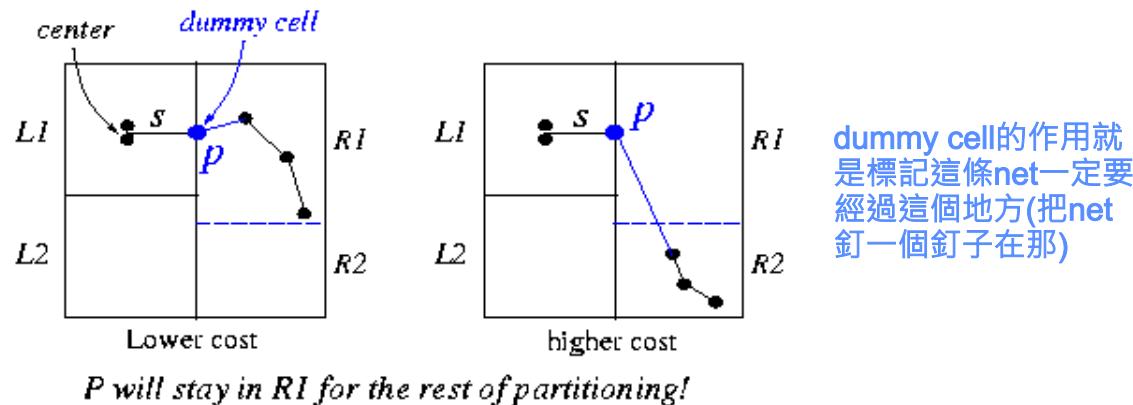
Min-Cut Placement with Terminal Propagation

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

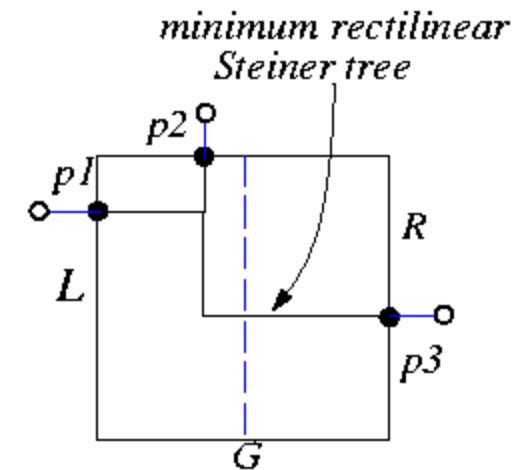
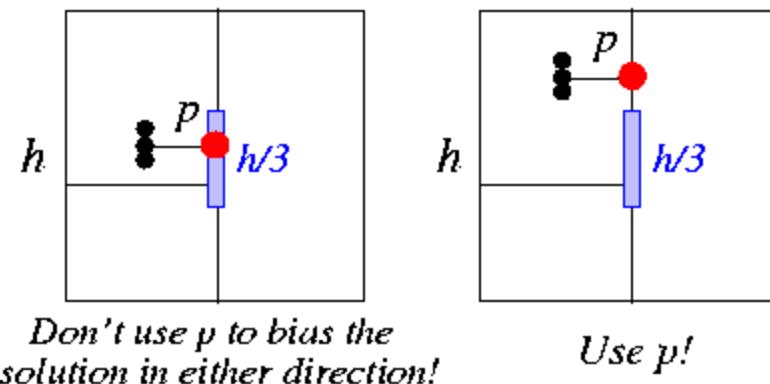


Terminal Propagation

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

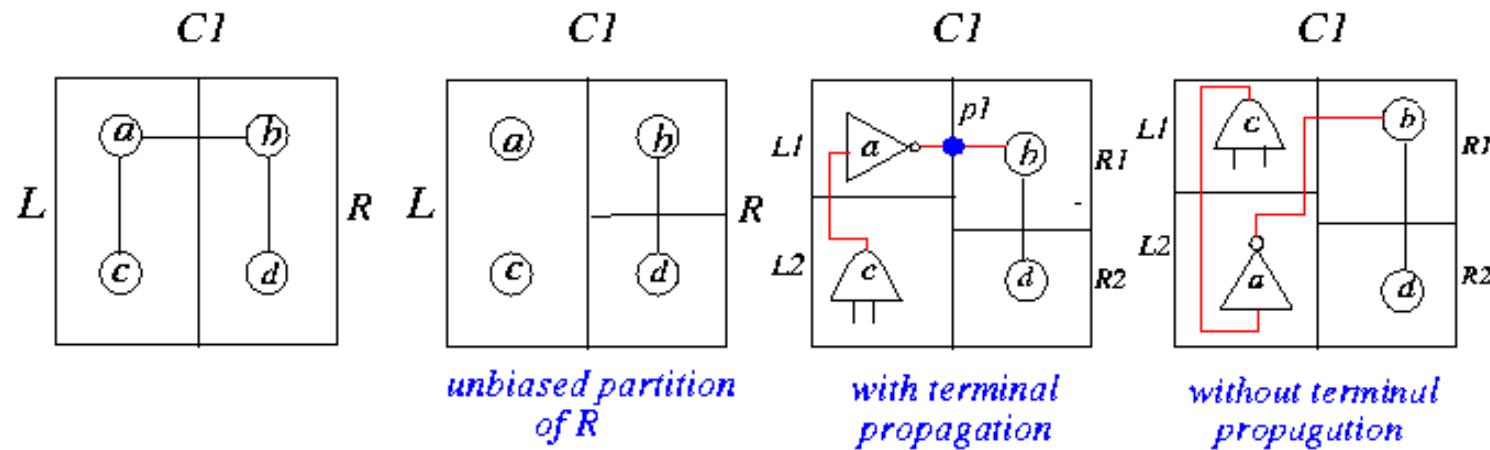
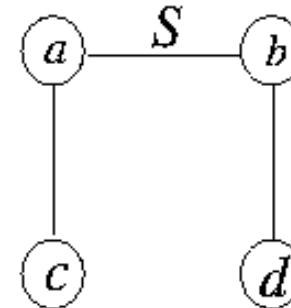
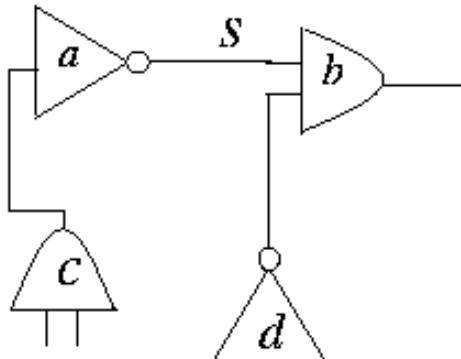


- When not to use p to bias partitioning? Net s has cells in many groups?



Terminal Propagation Example

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



Mincut Placement : Quadrature Mincut

- Perform quadrature mincut onto 4×4 grid
 - Start with vertical cut first

$$n_1 = \{e, f\}$$

$$n_2 = \{a, e, i\}$$

$$n_3 = \{b, f, g\}$$

$$n_4 = \{c, g, l\}$$

$$n_5 = \{d, l, h\}$$

$$n_6 = \{e, i, j\}$$

$$n_7 = \{f, j\}$$

$$n_8 = \{g, j, k\}$$

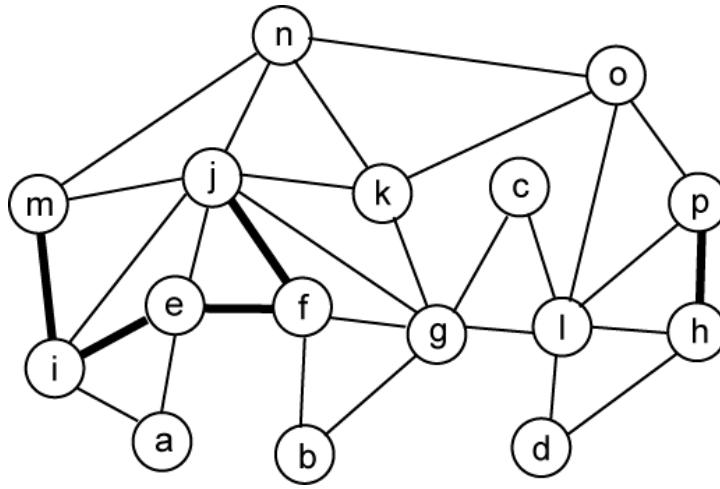
$$n_9 = \{l, o, p\}$$

$$n_{10} = \{h, p\}$$

$$n_{11} = \{i, m\}$$

$$n_{12} = \{j, m, n\}$$

$$n_{13} = \{k, n, o\}$$

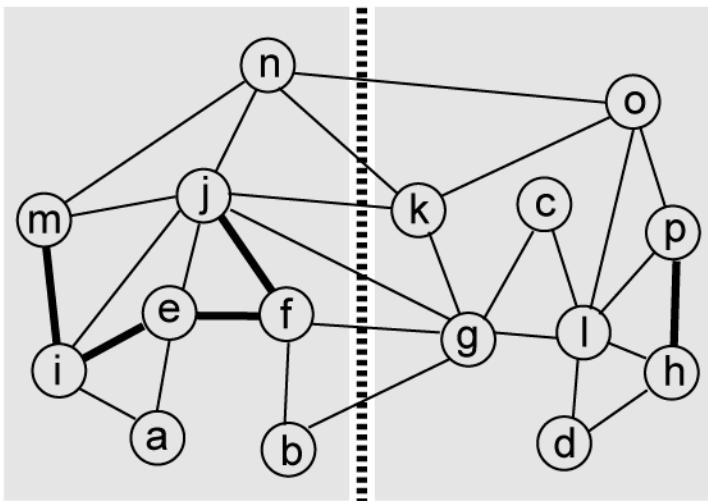


Clique-based graph model

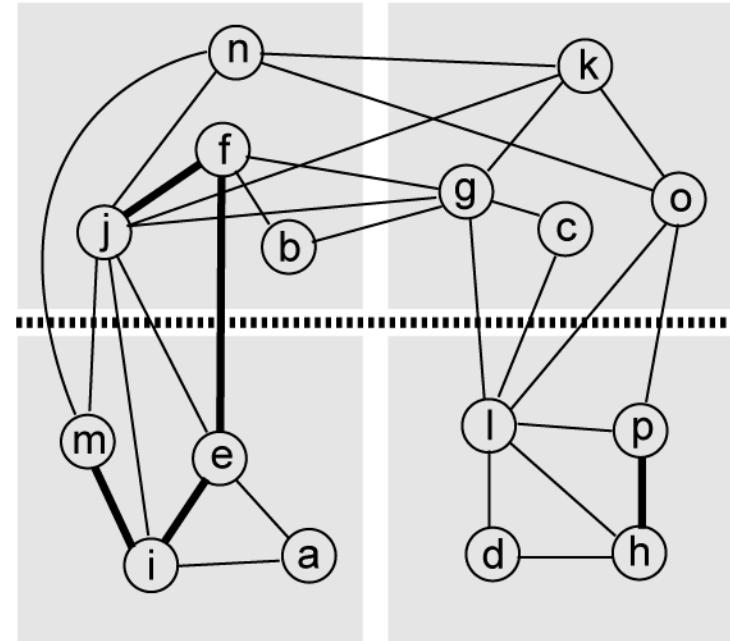
thin edges = weight 0.5, thick edges = weight 1

Cut 1 and 2

- First cut has min-cutsize of 3 (not unique)
 - Both cuts 1 and 2 divide the entire chip



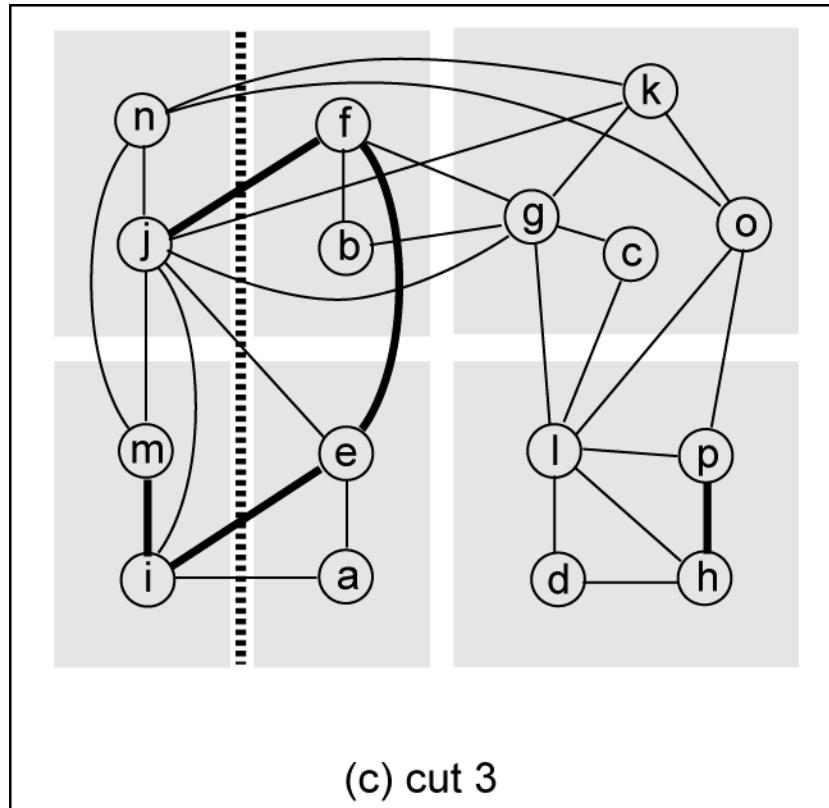
(a) cut 1



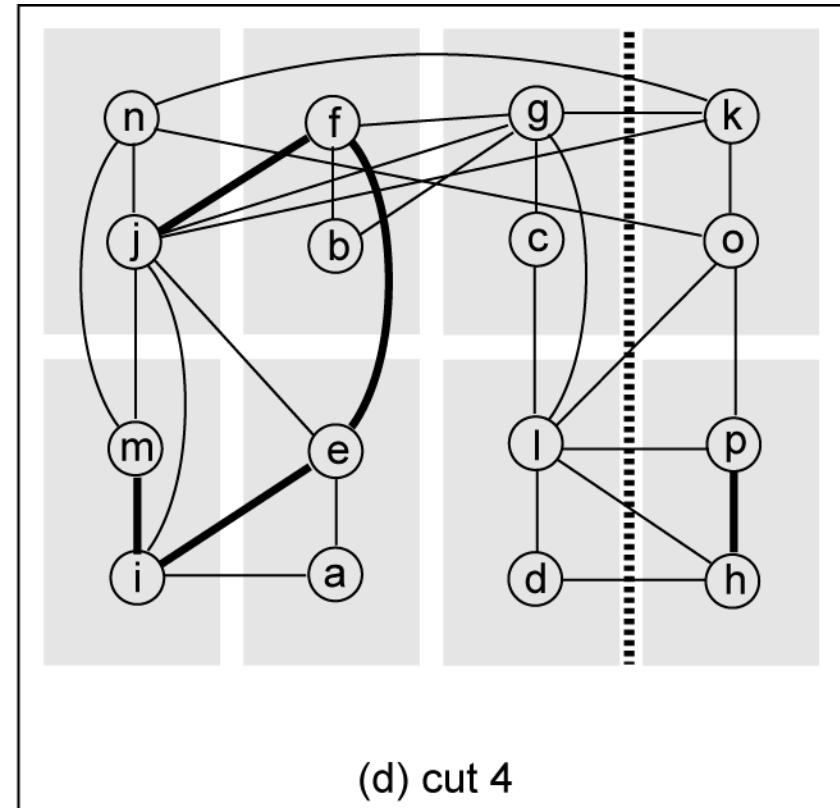
(b) cut 2, 1st-level quadrants formed

Cut 3 and 4

- Each cut minimizes cutsize
 - Helps reduce overall wirelength



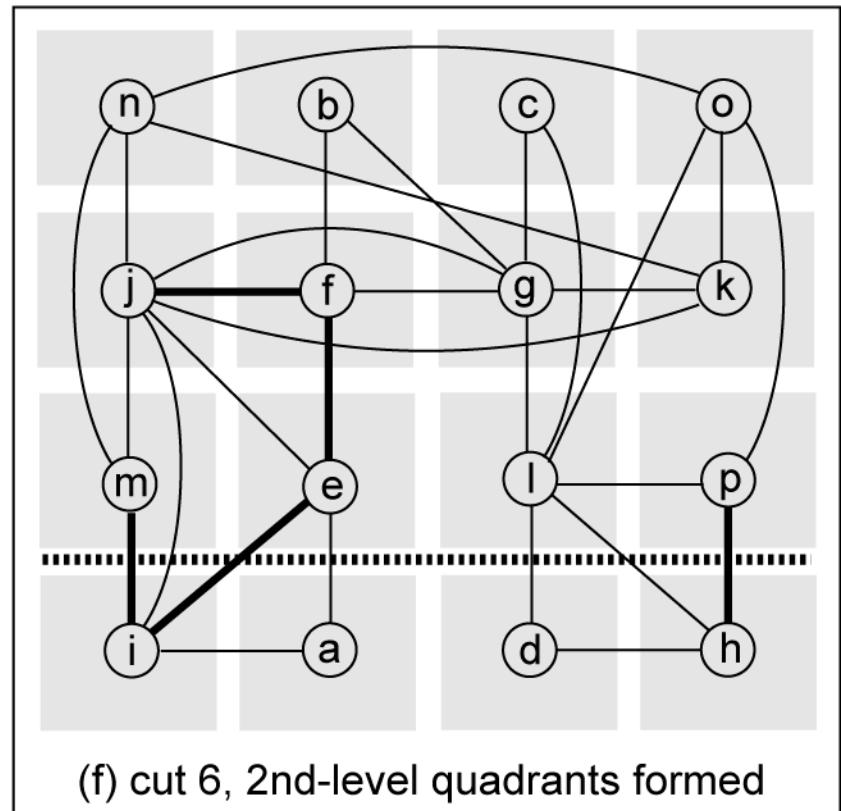
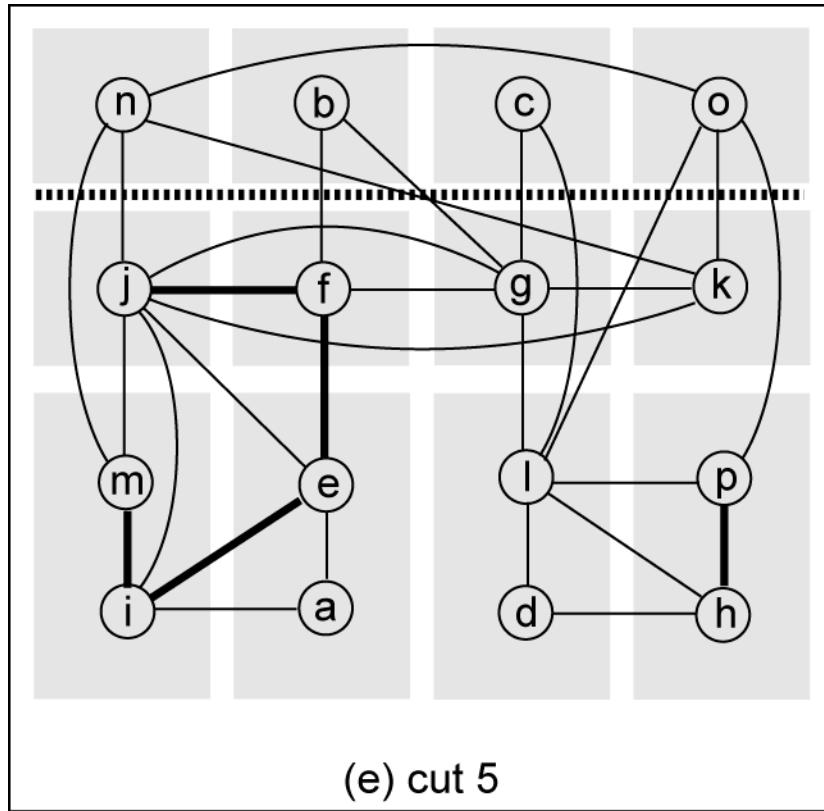
(c) cut 3



(d) cut 4

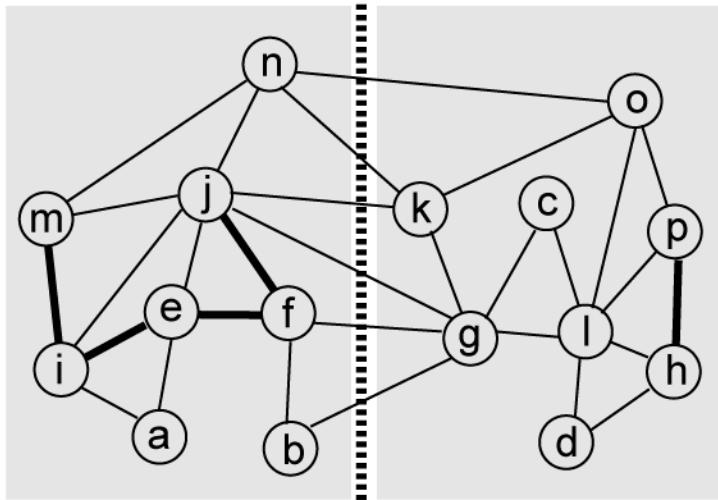
Cut 5 and 6

- 16 partitions generated by 6 cuts
 - HPBB wirelength = 27

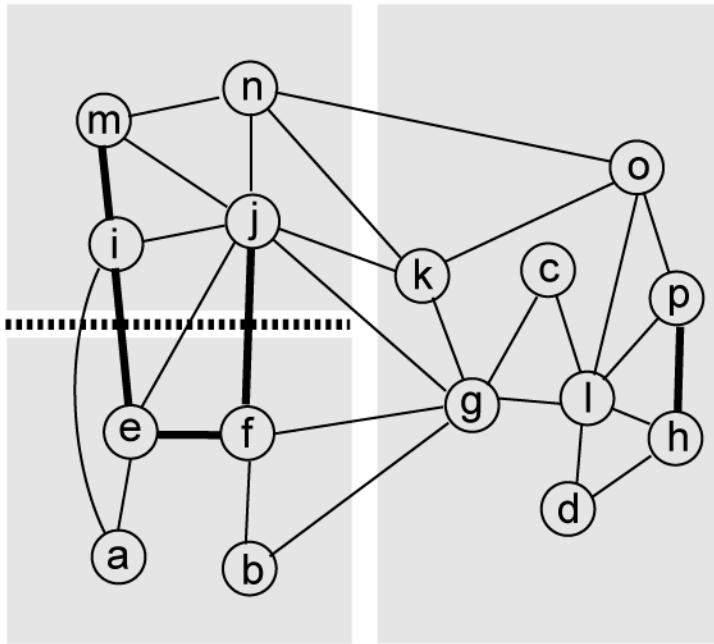


Mincut Placement: Recursive Bisection

- Start with vertical cut



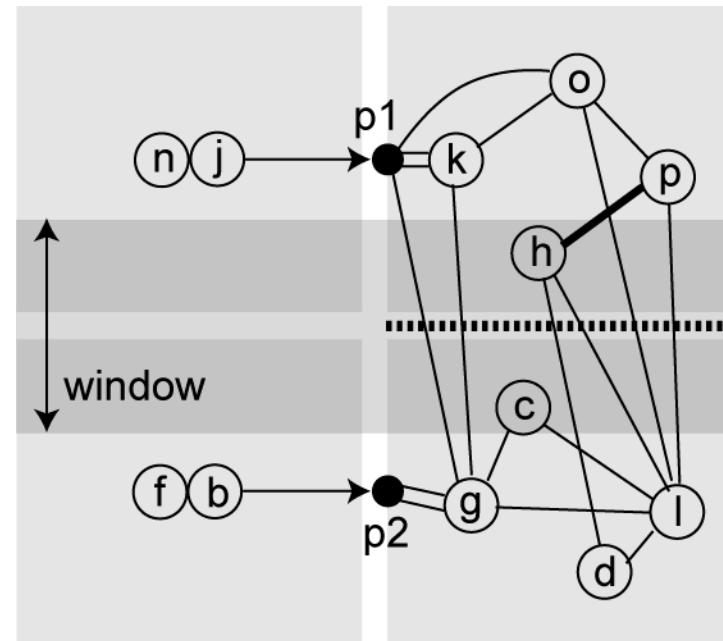
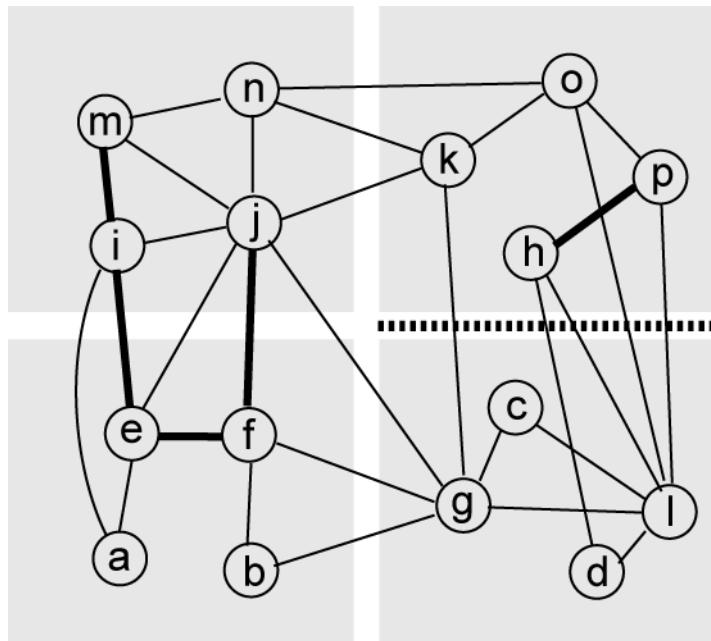
(a) cut 1



(b) cut 2

Cut 3: Terminal Propagation

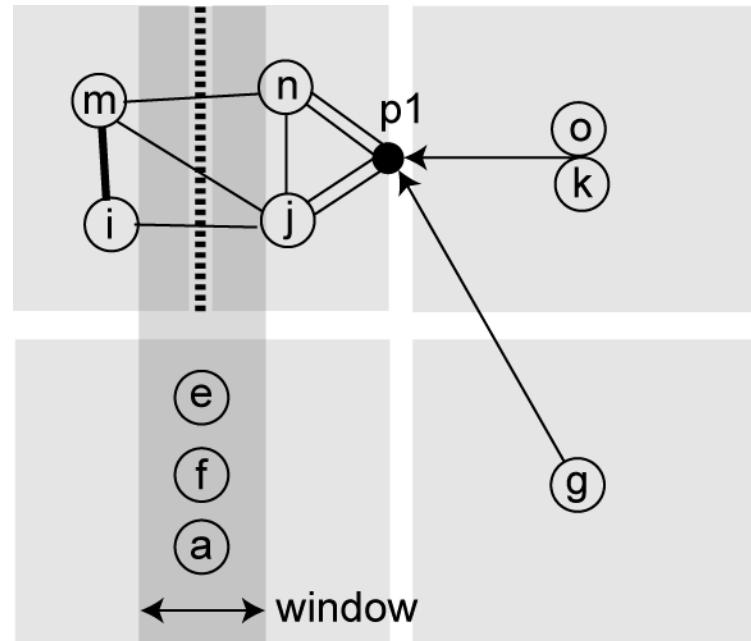
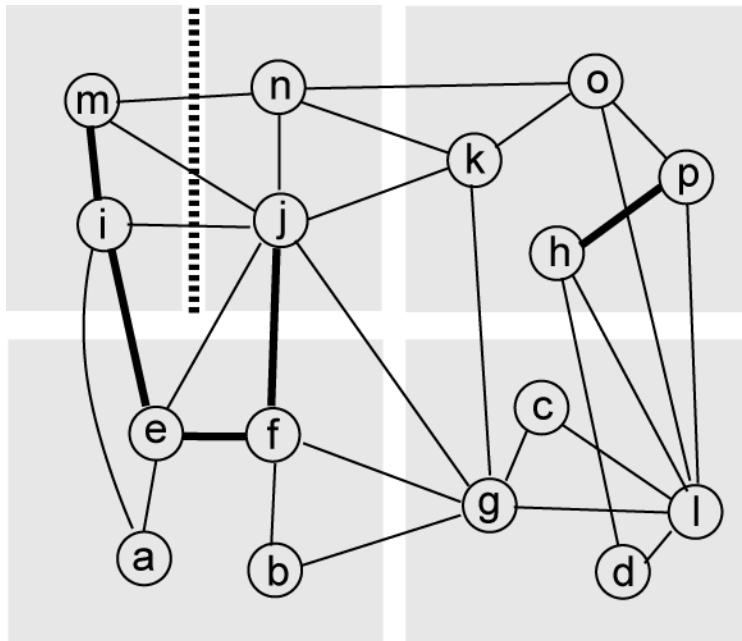
- Two terminals are propagated: “pulling” nodes
 - Node k and o connect to n and j : p_1 propagated (outside window)
 - Node g connect to j , f and b : p_2 propagated (outside window)
 - Terminal p_1 pulls $k/o/g$ to top partition, and p_2 pulls g to bottom



每次要進行切割之前，都會去看dummy node的位置
猜想：把dummy node當成一個node(兩個set中都會有一些dummy node)，也是放在某個set中且lock住，進行partition的時候也要將其考慮進去來算其他cell的gain

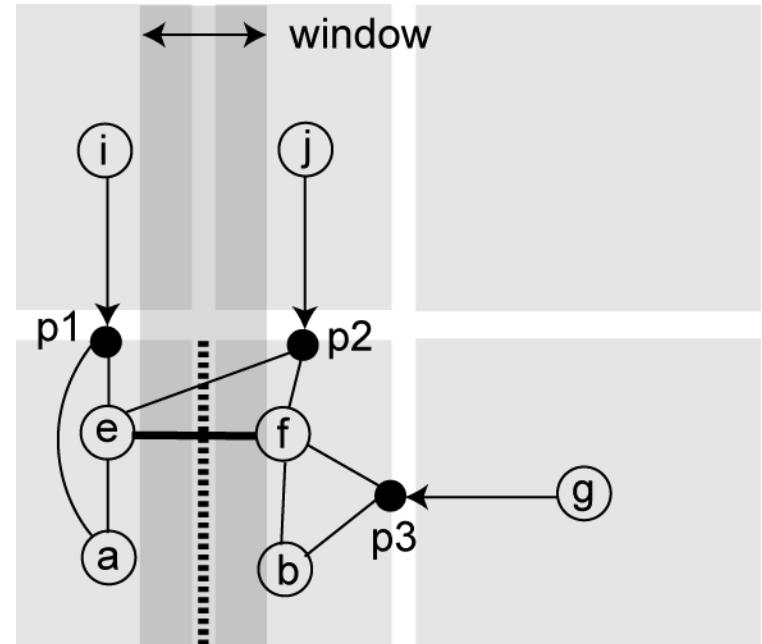
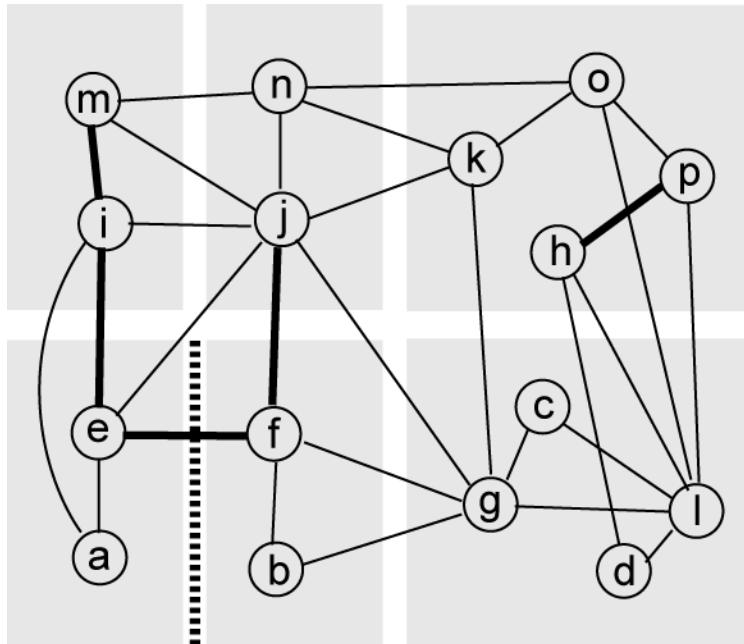
Cut 4: Terminal Propagation

- One terminal propagated
 - Node n and j connect to $o/k/g$: p_1 propagated
 - Node i and j connect to $e/f/a$: no propagation (inside window)
 - Terminal p_1 pulls n and j to right partition



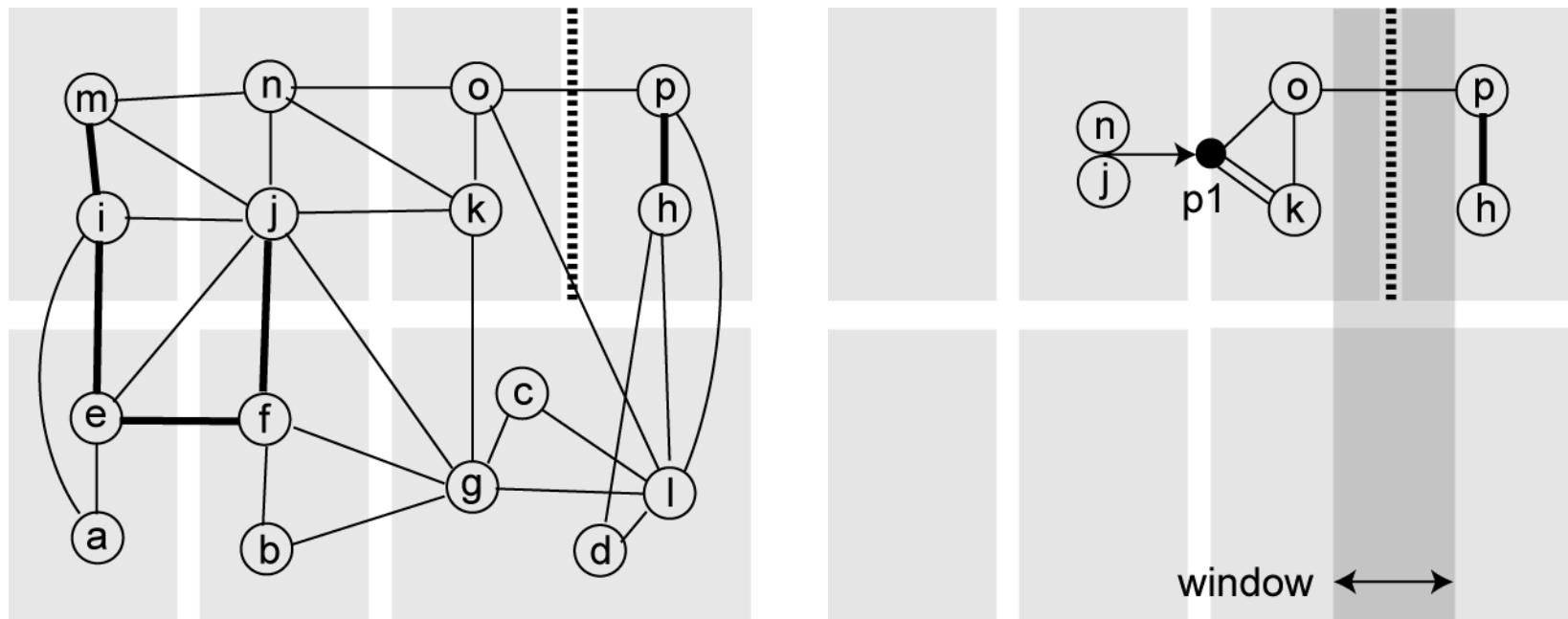
Cut 5: Terminal Propagation

- Three terminals propagated
 - Node i propagated to p_1 , j to p_2 , and g to p_3
 - Terminal p_1 pulls e and a to left partition
 - Terminal p_2 and p_3 pull $f/b/e$ to right partition



Cut 6: Terminal Propagation

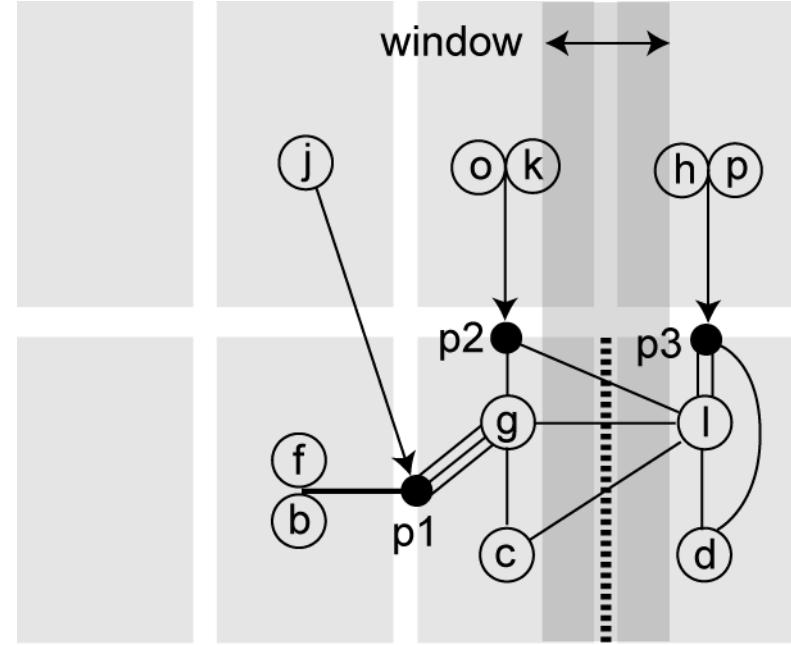
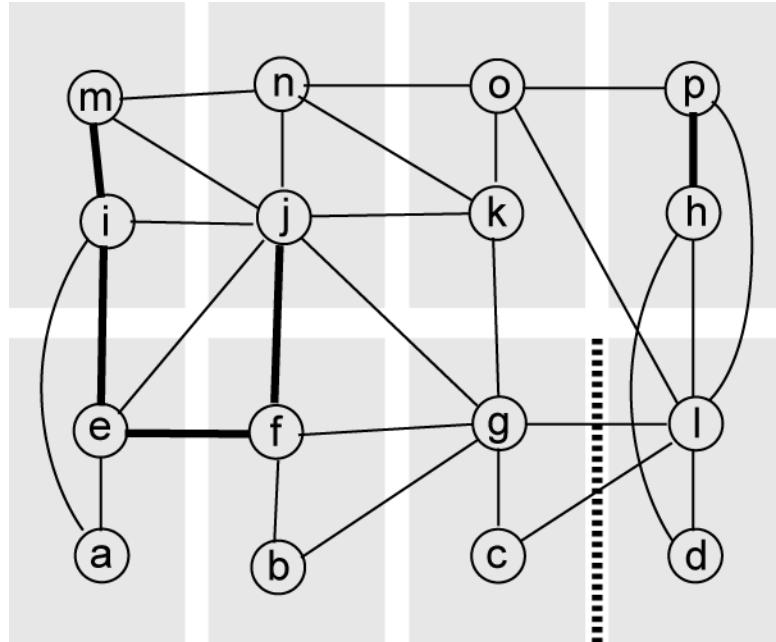
- One terminal propagated
 - Node n and j are propagated to p_1
 - Terminal p_1 pulls o and k to left partition



Cut 7: Terminal Propagation

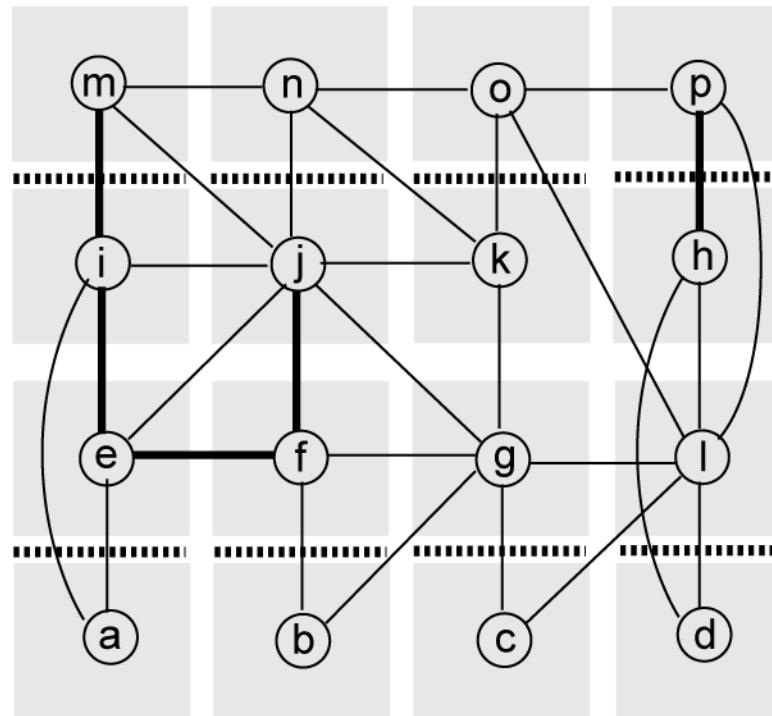
- Three terminals propagated

- Node $j/f/b$ propagated to p_1 , o/k to p_2 , and h/p to p_3
- Terminal p_1 and p_2 pull g and l to left partition
- Terminal p_3 pull l and d to right partition



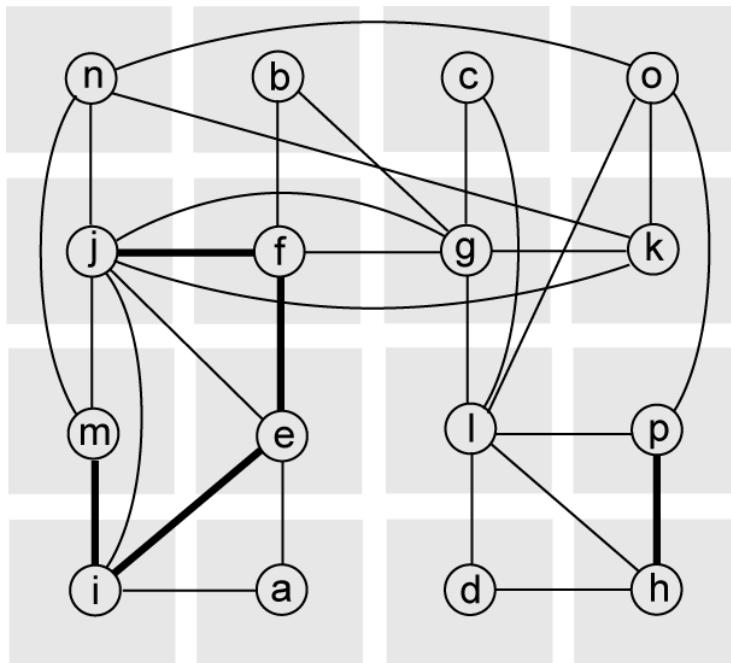
Cut 8 to 15

- 16 partitions generated by 15 cuts
 - HPBB wirelength = 23

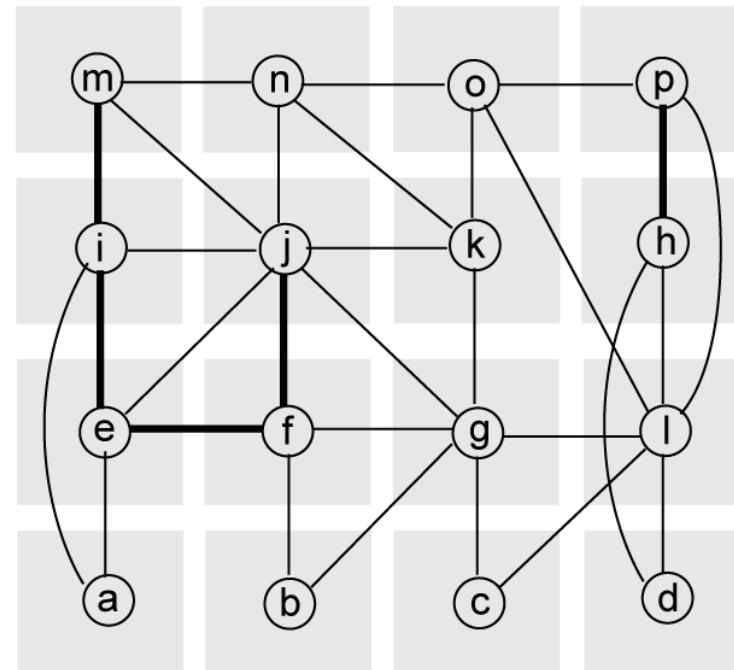


Comparison

- Quadrature vs. recursive bisection + terminal propagation
 - Number of cuts: 6 vs. 15
 - Wirelength: 27 vs. 23



quadrature

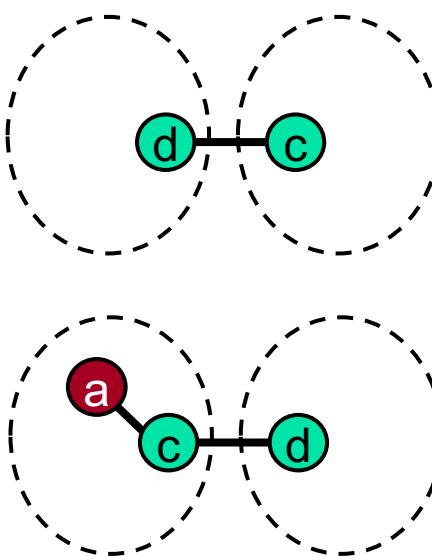
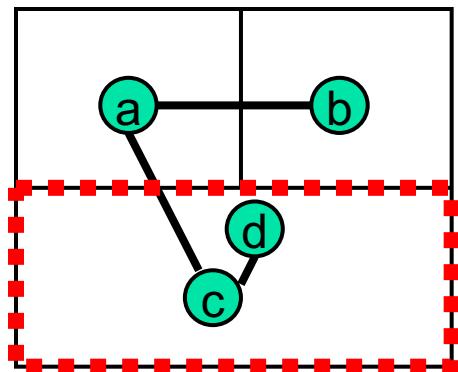


bisection

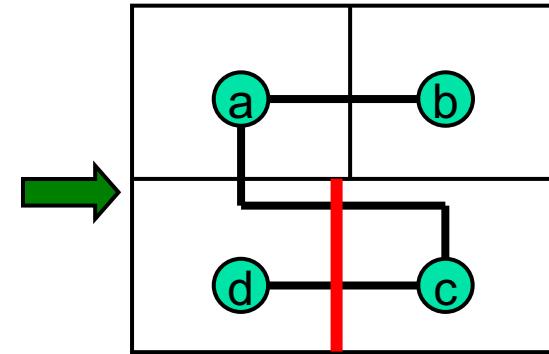
Exact Net-Weight Modeling

- Chen, Chang, Lin, “IMF: Interconnect-driven multilevel floorplanning for large-scale building-module designs,” ICCAD-05 (TCAD-08).
- After partitioning nodes a and b , how do we assign the regions for c and d to minimize the wire length?

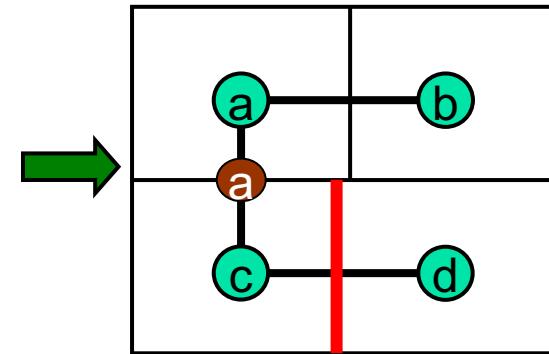
A possible solution **without**
using terminal propagation



With terminal propagation
(Node a is a fixed node)



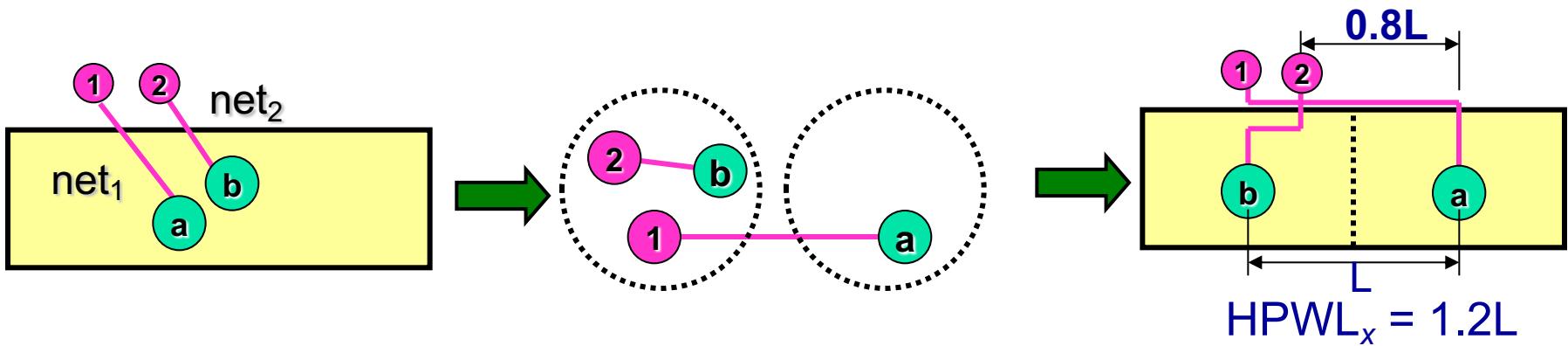
cut size = 2



Obtain a better result.
cut size = 1

Traditional Terminal Propagation (TP)

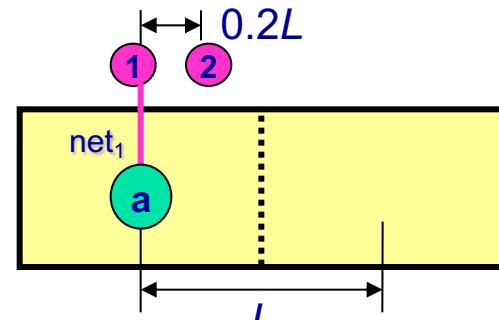
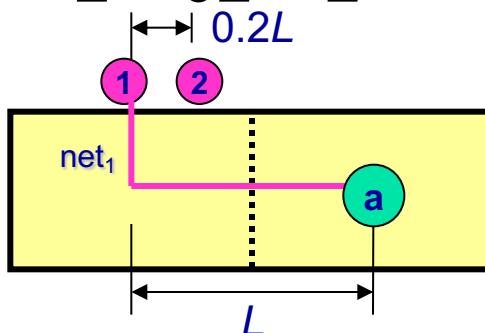
- Traditional TP considers only the cut size, which is *not* equivalent to finding the minimum HPWL exactly.
- Traditional TP might lead to a sub-optimal solution.
 - Nodes 1 and 2 are *fixed*, partitioning *a* and *b* to different regions gives the same cut size but incurs different HPWL's.



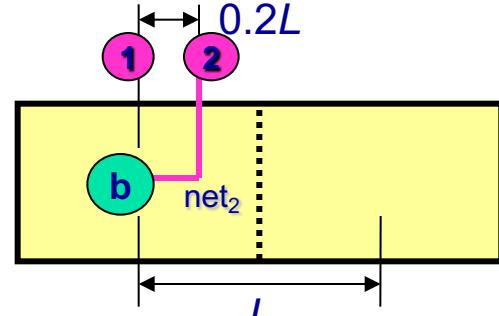
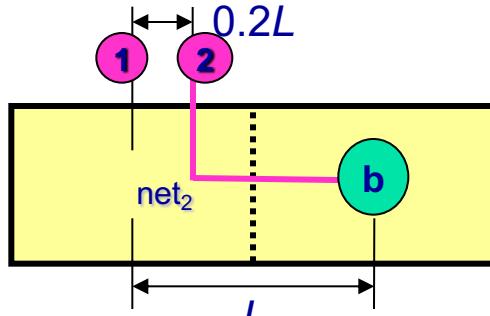
- Problem: edge weight is a **constant** value!
 - Shall map the min-cut cost to the HPWL change.
 - Shall assign the edge weight as the value of the HPWL contribution if the edge is cut.

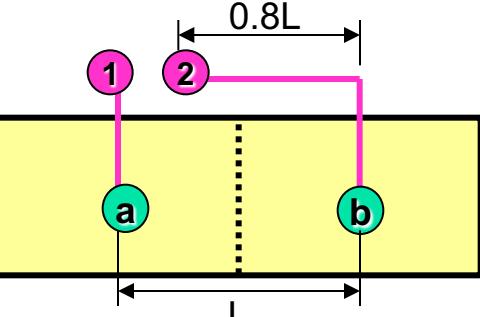
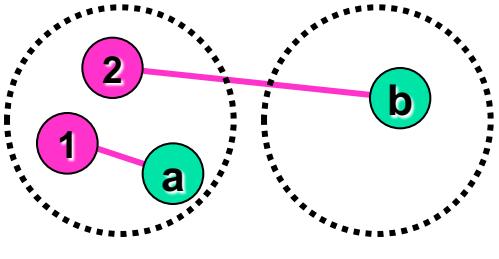
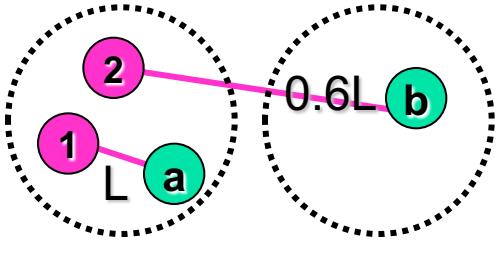
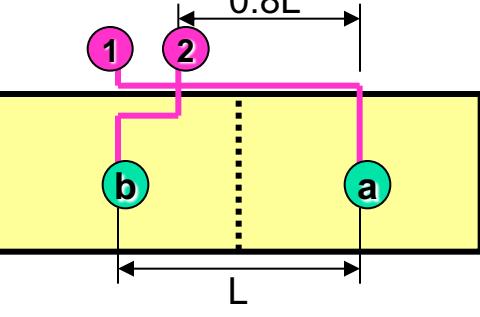
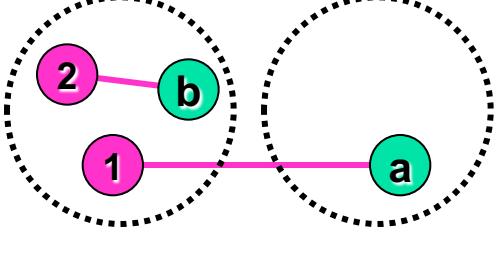
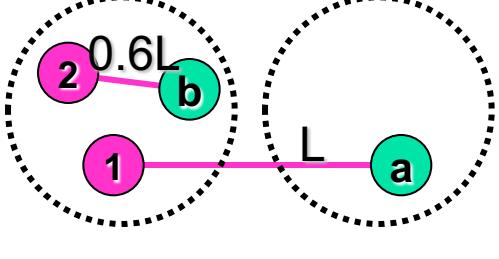
Net Weight Assignment

- Set edge weight to the value of HPWL if the edge is cut.
- net_1 connects a movable node **a** and a fixed node **1**.
$$\begin{aligned}\text{Weight}(\text{net}_1) &= \text{Length}(\text{net}_1 \text{ is cut}) - \text{Length}(\text{net}_1 \text{ is not cut}) \\ &= L - 0L = L\end{aligned}$$



- net_2 connects a movable node **b** and a fixed node **2**.
$$\begin{aligned}\text{Weight}(\text{net}_2) &= \text{Length}(\text{net}_2 \text{ is cut}) - \text{Length}(\text{net}_2 \text{ is not cut}) \\ &= 0.8L - 0.2L = 0.6L\end{aligned}$$



Physical Partitions	Traditional Terminal Propagation	Exact Net-Weight Modeling
 <p>$HPWL_x = 0.8L$</p>	 <p>Cut size = 1</p>	 <p>Cut weight = 0.6L</p>
 <p>$HPWL_x = 1.2L$</p>	 <p>Cut size = 1</p>	 <p>Cut weight = 1.0L</p>

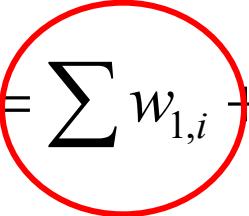
Cut weight is proportional to the HPWL!

$$HPWL = Cut\ weight + 0.2L$$

(0.2L is the lower bound for the net's HPWL)

Exact Net-Weight Modeling

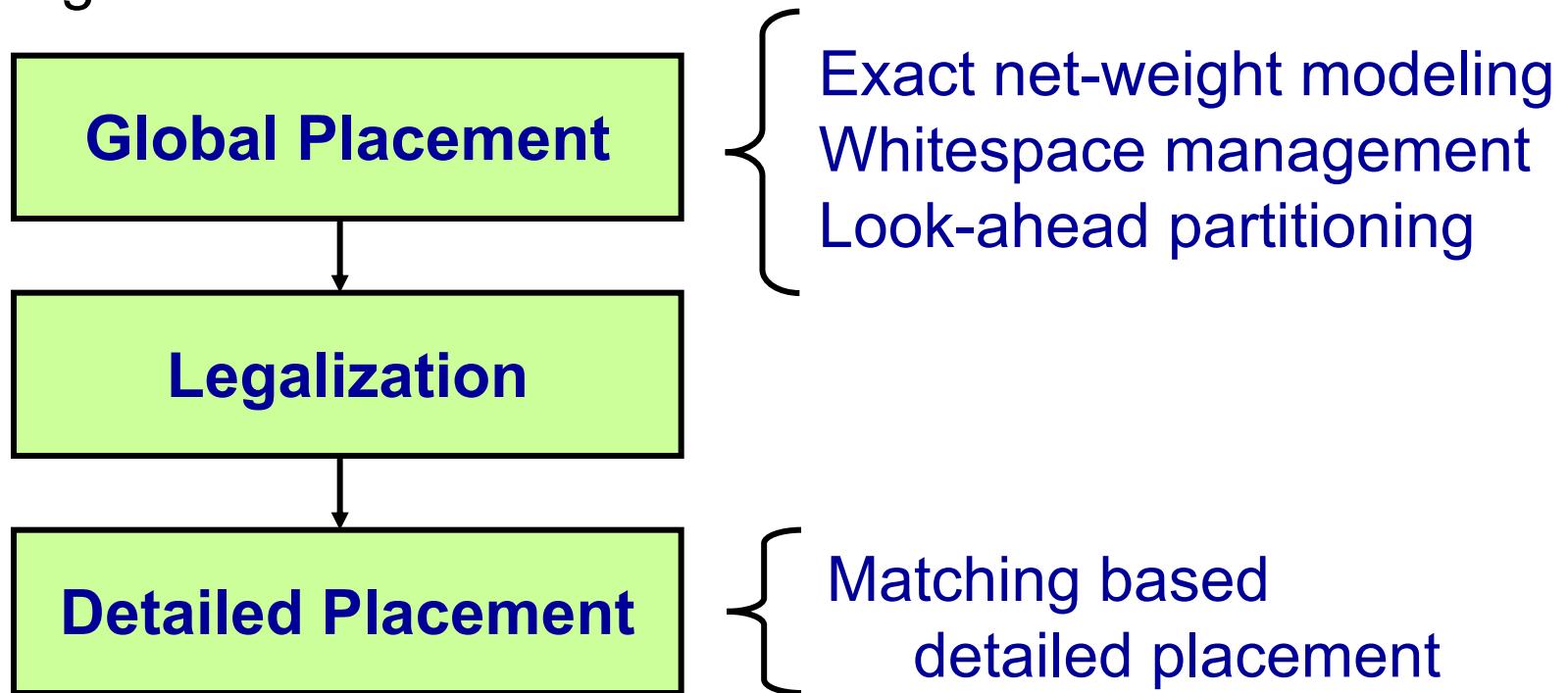
- Chen, Chang, Lin, “IMF: Interconnect-driven multilevel floorplanning for large-scale building-module designs,” ICCAD-05 (TCAD-08).
- Theorem: $HPWL_i = w_{1,i} + n_{cut,i}$
 - $n_{cut,i}$: cut weight for net i
 - $w_{1,i}$: the HPWL lower bound for net i (movable modules are all at the side that leads to the smallest HPWL)
- Then, we have Constant!

$$\min\left(\sum HPWL_i\right) = \min\left(\sum (w_{1,i} + n_{cut,i})\right) = \sum w_{1,i} - \min\left(\sum n_{cut,i}\right)$$


Finding the minimum HPWL is equivalent to finding the min-cut!!

NTUplace1 Overview

- Chen, Hsu, Jiang, Chang, “A ratio partitioning based placement algorithm for large-scale mixed-size design,” ISPD-05
- Is based on the min-cut partitioning technique
- Algorithm overview



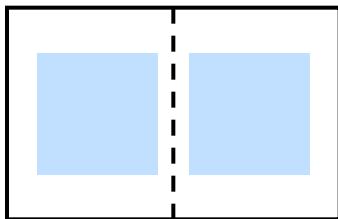
Whitespace Management (1/2)

- Traditional min-cut placers uniformly distribute whitespace and tend to produce excessive wirelength when the whitespace is large
- Adya, Markov, Villarrubia use dummy cells to control the whitespace allocation [ICCAD-03]
 - Add dummy cells to increase the utilization ratio
 - Whitespace is distributed according to the dummy cell locations
 - However, their method tends to increase the number of cells, leading to longer running time and larger memory usage

Whitespace Management (2/2)

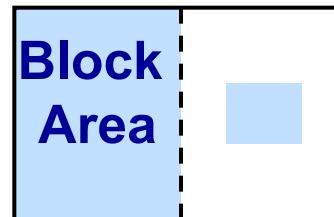
- NTPlace1 directly controls the balance criteria during partitioning using the available free space
- Relaxing the balance criteria leads to smaller cutsize and thus smaller wirelength
- The balance criterion satisfies that the utilization of each partition is less than or equal to 1
- The criterion is then fed into the partitioner to allocate whitespace

Uniform whitespace distribution



Both utilization < 1

Left partition Utilization = 1.0



**Left util. = 1,
right util. < 1**

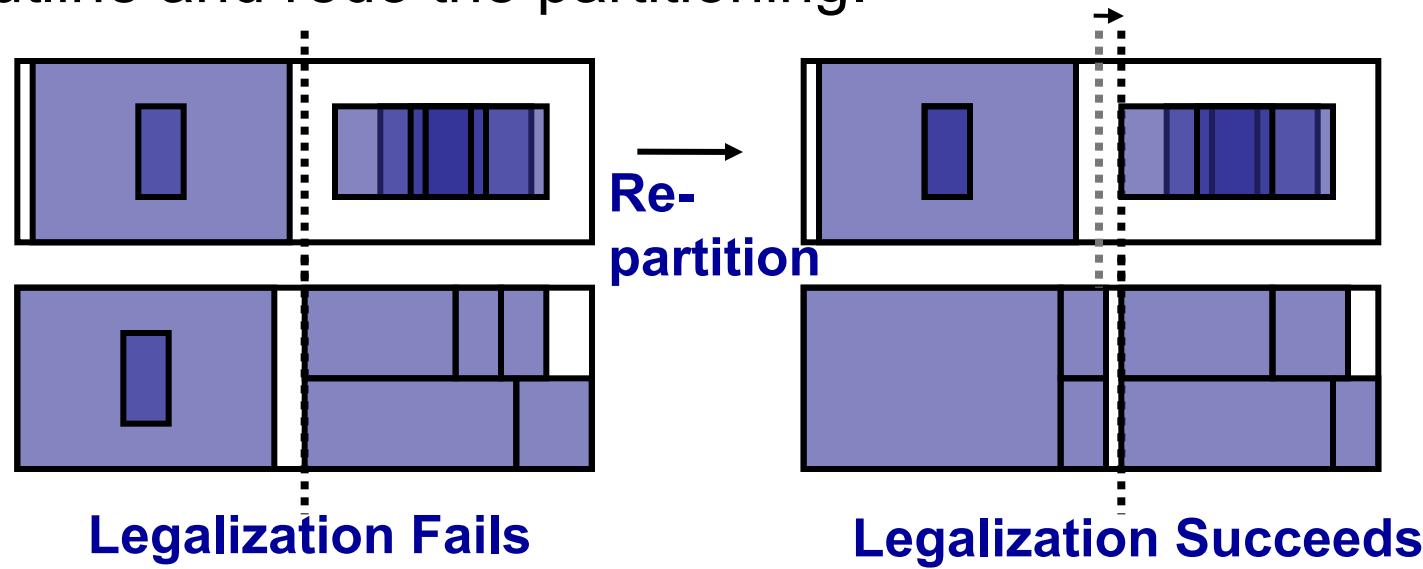
Right partition Utilization = 1.0



**Left util. < 1,
right util. = 1**

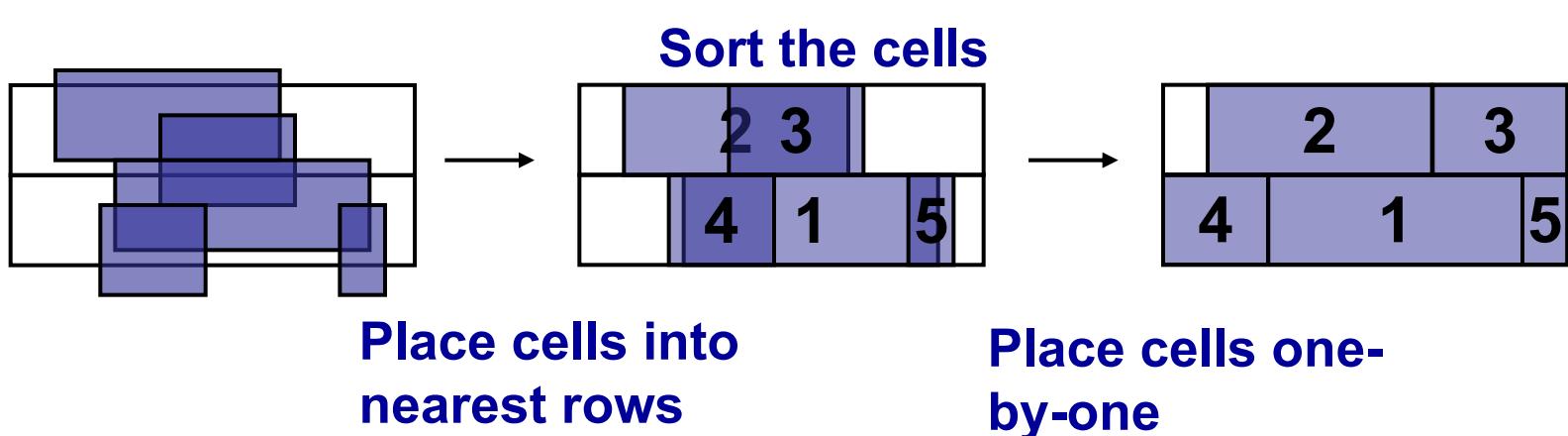
Look-Ahead Partitioning

- Simplify the idea in Cong et al., “Fast floorplanning by look-ahead enabled recursive bipartitioning,” ASPDAC-2005
- Use the first-fit bin-packing heuristic to check if the subpartition can be legalized
 - Increase the chance of legalizing macro blocks
- If the subpartition cannot be legalized, we move the cutline and redo the partitioning.



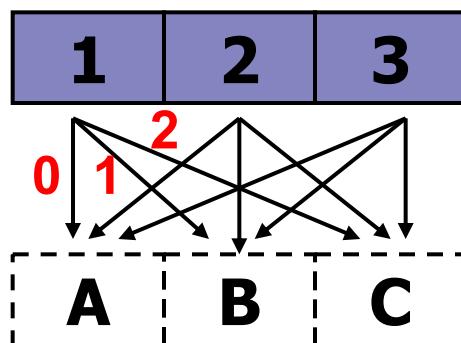
Legalization

- Place all cells in the rows to obtain a feasible solution
 - 1) Place cells into their nearest rows
 - 2) Sort all standard cells according to their sizes, from the largest to the smallest
 - 3) Assign the x-coordinates for all cells according to the sorted order. If overlap occurs, we will find a nearest empty slot to place the cell



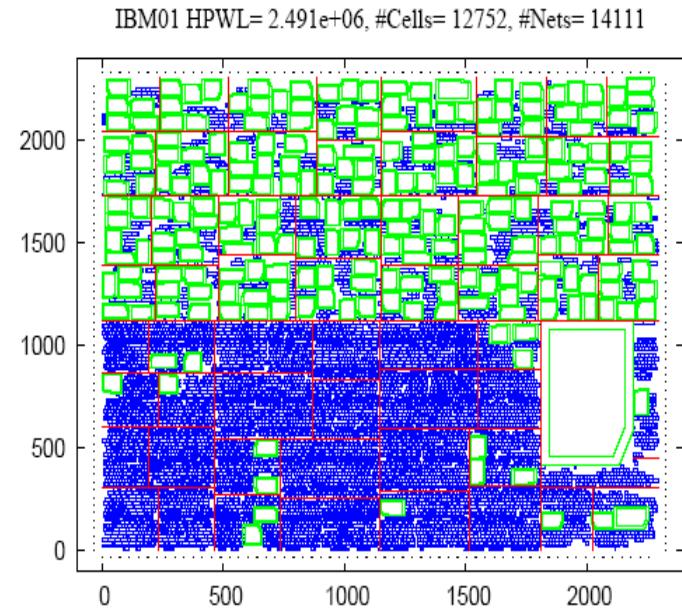
Detailed Placement

- Is based on cell location assignment (matching)
 - Each cell has different costs at different locations
 - Minimize total cost: $O(n^3)$ time for n cells
 - Is better than $O(n!)$ time for a branch & bound (BB) detailed placer
 - Can use a much larger window (> 64 cells)



Capo Overview

- Pure recursive bi-partitioning placer
 - Multi-level FM for instances with > 200 cells
 - Flat FM for instances with 35-200 cells
 - Branch-and-bound for instances with < 35 cells
- Careful handling of partitioning tolerance
 - Uncorking: Prevent large cells from blocking smaller cells to move
 - Repartitioning: Several FM calls with decreasing tolerance
 - Block splitting heuristics: Higher tolerance for vertical cut
 - Hierarchical tolerance computation: Instance with more whitespace can have a bigger partitioning tolerance
- Implementation with good interface (LEF/DEF and GSRC bookshelf) available on web

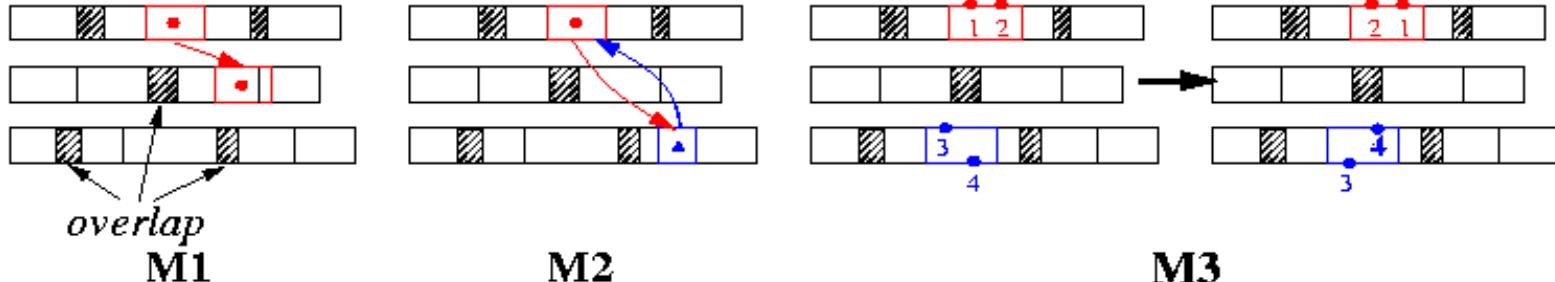


Placement by Simulated Annealing

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

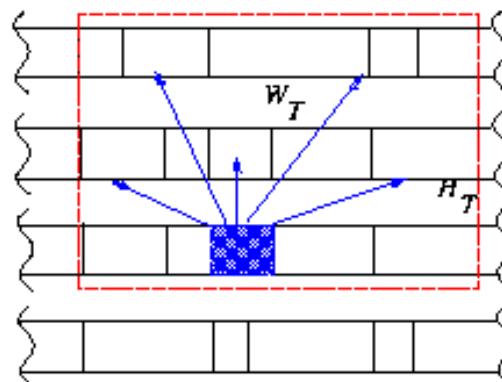
Solution Space & Neighborhood Structure

- **Solution Space:** All possible arrangements of the modules into rows, possibly with overlaps. 直接透過座標來代表solution
- **Neighborhood Structure:** 3 types of moves
 - M_1 : Displace a module to a new location.
 - M_2 : Interchange two modules.
 - M_3 : Change the orientation of a module. 左右鏡射-->也許可以減少wire length



Neighborhood Structure

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



Cost Function

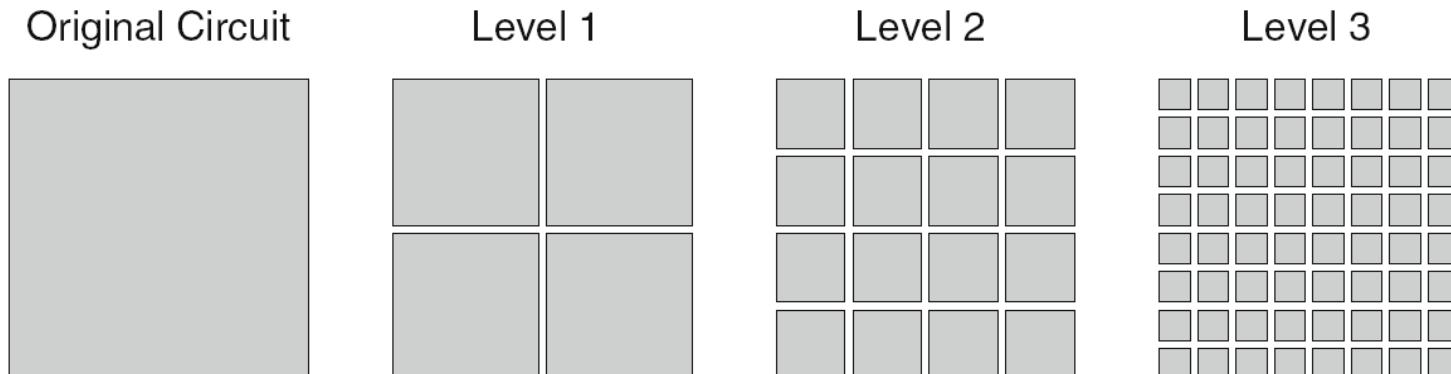
- Cost function: $C = C_1 + C_2 + C_3$.
- C_1 : **total estimated wirelength**.
 - $C_1 = \sum_{i \in \text{Nets}} (\alpha_i w_i + \beta_i h_i)$
 - α_i, β_i are horizontal and vertical weights, respectively. ($\alpha_i=1, \beta_i=1 \Rightarrow (1/2) \times$ perimeter of the bounding box of Net i .)
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

Annealing Schedule

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

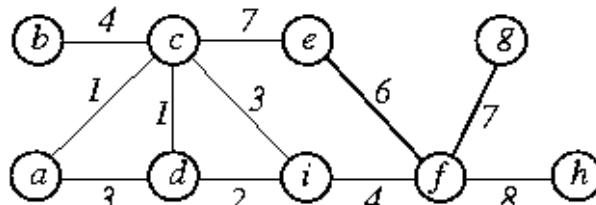
Dragon Overview

- Dragon takes a hybrid approach that combines simulated annealing and partitioning
 - Recursive partitioning to reduce the problem size
 - Annealing to refine the solution generated by partitioning
- Top-down hierarchical placement
 - hMetis to recursively quadrisect into 4^h bins at level h
 - Swapping of bins at each level by SA to minimize WL
 - Terminates when each bin contains < 7 cells
 - At final stage of GP, switch single cells locally among bins by SA to further minimize WL
- Detailed placement is done by greedy algorithm



Placement by the Genetic Algorithm

- Cohoon & Paris, “Genetic placement,” ICCAD-86.
- **Genetic algorithm:** A search technique that emulates the biological evolution process to find the optimum.
- Generic approaches:
 - Start with an initial set of random configurations (**population**); each individual is a string of symbol (symbol string \leftrightarrow **chromosome**: a solution to the optimization problem, symbol \leftrightarrow **gene**).
 - During each iteration (**generation**), the individuals are evaluated using a **fitness** measurement.
 - Two fitter individuals (**parents**) at a time are selected to generate new solutions (**offsprings**).
 - Genetic operators: **crossover**, **mutation**, **inversion**
- In the example, string = [aghcbidef]; fitness value = $1/\sum_{(i, j) \in E} w_{ij} d_{ij} = 1/85$.



6	7	8
3	4	5
0	1	2

d	e	f
c	b	i
a	g	h

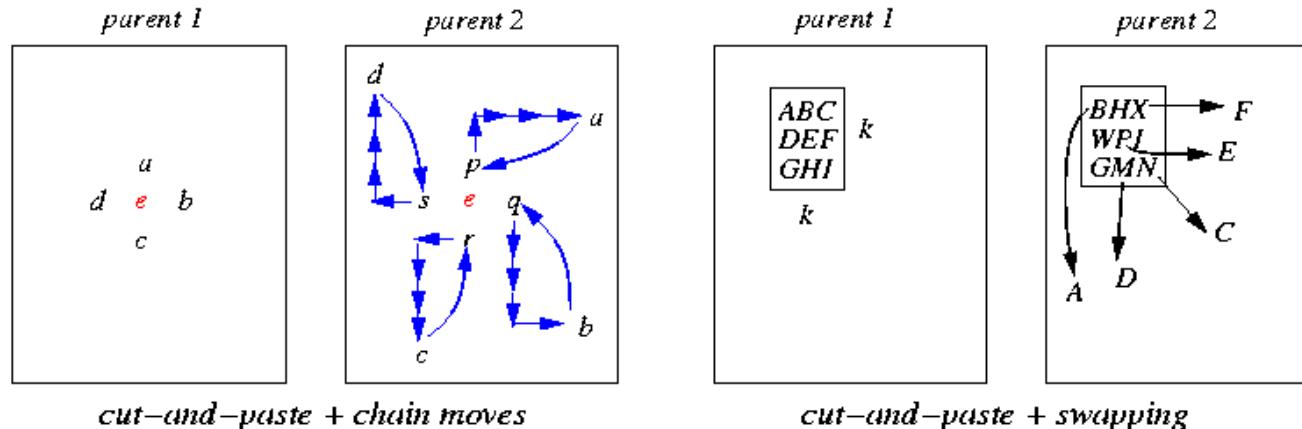
string: aghc**b**idef

Genetic Operator: Crossover

- Main genetic operator: Operate on two individuals and generates an offspring.
 - $[bidef|aghc](\frac{1}{86}) + [bdefi|gcha](\frac{1}{110}) \rightarrow [bidefgcha](\frac{1}{63})$.
 - Need to avoid repeated symbols in the solution string!
- **Partially mapped crossover** for avoiding repeated symbols:
 - $[bidef|gcha](\frac{1}{86}) + [aghcb|idef](\frac{1}{85}) \rightarrow [bgcha|idef]$.
 - Copy *idef* to the offspring; scan *[bidef|gcha]* from the left, and then copy all unrepeated genes.

Two More Crossover Operations

- Cut-and-paste + Chain moves:
 - Copy a randomly selected cell e and its four neighbors from parent 1 to parent 2.
 - The cells that earlier occupied the neighboring locations in parent 2 are shifted outwards.
- Cut-and-paste + Swapping:
 - Copy $k \times k$ square modules from parent 1 to parent 2 (k : random # from a normal distribution with mean 3 and variance 1).
 - Swap cells not in both square modules.



Genetic Operators: Mutation & Inversion

- **Mutation:** prevents loss of diversity by introducing new solutions.
 - Incremental random changes in the offspring generated by the crossover.
 - A commonly used mutation: pairwise interchange.
- **Inversion:** $[bid|efgch|a] \rightarrow [bid|hcgfe|a]$.
- Apply mutation and inversion with probability P_μ and P_i respectively.

Algorithm: Genetic_Placement(N_p , N_g , N_o , P_i , $P\mu$)

```
/*  $N_p$ : population size; */  
/*  $N_g$ : # of generation; */  
/*  $N_o$ : # of offsprings; */  
/*  $P_i$  : inversion probability; */  
/*  $P\mu$  : mutation probability; */  
  
1 begin  
2 ConstructPopulation( $N_p$ ); /* randomly generate the initial population */  
3 for  $j \leftarrow 1$  to  $N_g$   
4   Evaluate Fitness( $population(j)$ );  
5 for  $i \leftarrow 1$  to  $N_o$   
6   for  $j \leftarrow 1$  to  $N_p$   
7     ( $x, y$ )  $\leftarrow$  ChooseParents; /* choose parents with probability  $\propto$  fitness value */  
8      $offspring(j) \leftarrow$  GenerateOffspring( $x, y$ ); /* perform crossover to generate offspring */  
9   for  $h \leftarrow 1$  to  $N_p$   
10    With probability  $P\mu$ , apply Mutation( $population(h)$ );  
11    for  $h \leftarrow 1$  to  $N_p$   
12      With probability  $P_i$ , apply Inversion( $population(h)$ );  
13      Evaluate Fitness( $offspring(j)$ );  
14     $population \leftarrow$  Select( $population$ ,  $offspring$ ,  $N_p$ );  
15 return the highest scoring configuration in  $population$ ;  
16 end
```

Genetic Placement Experiment: GINIE

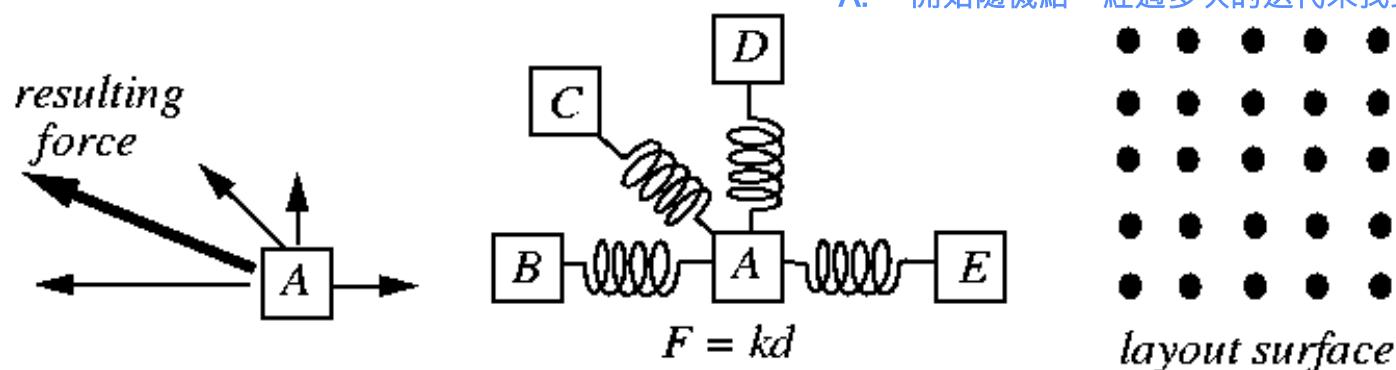
- Termination condition: no improvement in the best solution for 10,000 generations.
- Population size: 50. (Each generation: 50 unchanged throughout the process.)
- Each generation creates 12 offsprings.
- Comparisons with simulated annealing:
 - Similar quality of solutions and running time.

Analytical Placement

- Key: Solve a relaxed placement problem “optimally”
 - Ignore overlaps (fixed at later stages)
 - Adopt linear or nonlinear wirelength estimation
 - Need pads to pull the cells outwards
- Approaches for overlap removal
 - Cell spreading
 - Legalization
- Pro: Obtain very good quality for existing benchmarks
- Con: hard to handle cell rotation, big macros, region constraints, etc.
- Example tools
 - APlace (Kahng's group @ UCSD, ISPD-04, ICCAD-05)
 - FastPlace (Chu's group @ Iowa State, ISPD-04, ICCAD-05, DAC-07)
 - GORDIAN (Johannes's group @ TU of Munich, ICCAD-90)
 - Kraftwerk (Johannes's group @ TU of Munich, DAC-98, ICCAD-06)
 - mPL (Cong's group @ UCLA, ISPD-04, ISPD-05)
 - NTUplace3 (Chang's group @ Nat'l Taiwan U, ICCAD-06)

Placement by the Force-Directed Method

- Hanan & Kurtzberg, “Placement techniques,” in *Design Automation of Digital Systems*, Breuer, Ed, 1972.
- Quinn, Jr. & Breuer, “A force directed component placement procedure for printed circuit boards,” *IEEE Trans. Circuits and Systems*, June 1979.
- Reducing the placement problem to solving a set of simultaneous linear equations to determine equilibrium locations for cells.
- Analogy to Hooke's law: $F = kd$, F : force, k : spring constant, d : distance. 用彈簧的概念實現floorplanning --> 嘗試用物理的觀念來解演算法問題
- Goal: Map cells to the layout surface. 已知BCDE的位置，可以根據彈力平衡來決定A的位置
Q: 但是要如何得知BCDE的初始位置?
A: 一開始隨機給，經過多次的迭代來找到最後的最佳位置



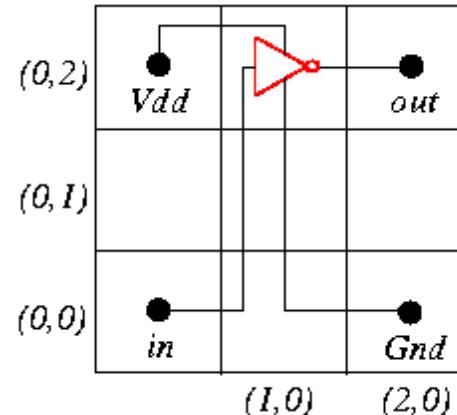
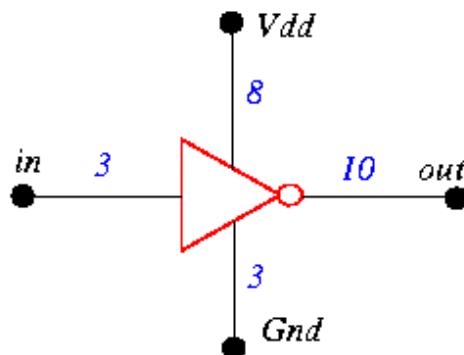
Finding the Zero-Force Target Location

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

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

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

- In the example, $\hat{x}_i = \frac{8 \times 0 + 10 \times 2 + 3 \times 0 + 3 \times 2}{8 + 10 + 3 + 3} = 1.083$ and $\hat{y}_i = 1.50$.



Force-Directed Placement

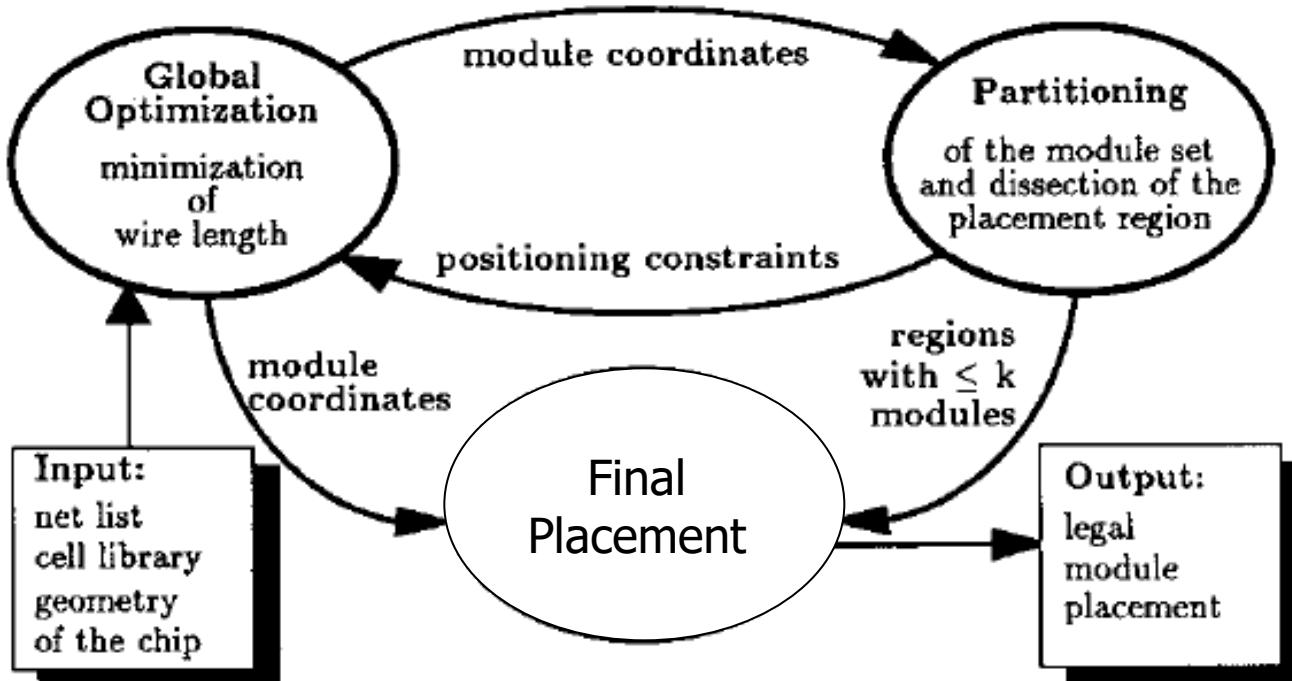
- Approach I (constructive):
 - Start with an initial placement.
 - Compute the zero-force locations for all cells.
 - Apply **linear assignment (matching)** to determine the “ideal” locations for the cells.
- Approach II (can be constructive or iterative):
 - Start with an initial placement.
 - Select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

Algorithm: Force-Directed_Placement

```
1 begin
2 Compute the connectivity for each cell;
3 Sort the cells in decreasing order of their connectivities into list  $L$ ;
4 while ( $IterationCount < IterationLimit$ ) do
5   Seed  $\leftarrow$  next module from  $L$ ;
6   Declare the position of the seed vacant;
7   while ( $EndRipple = FALSE$ ) do
8     Compute target location of the seed;
9     case the target location
10    VACANT:
11      Move seed to the target location and lock;
12       $EndRipple \leftarrow TRUE$ ;  $AbortCount \leftarrow 0$ ;
13    SAME AS PRESENT LOCATION:
14       $EndRipple \leftarrow TRUE$ ;  $AbortCount \leftarrow 0$ ;
15    LOCKED:
16      Move selected cell to the nearest vacant location;
17       $EndRipple \leftarrow TRUE$ ;  $AbortCount \leftarrow AbortCount + 1$ ;
18      if ( $AbortCount > AbortLimit$ ) then
19        Unlock all cell locations;
20         $IterationCount \leftarrow IterationCount + 1$ ;
21    OCCUPIED AND NOT LOCKED:
22      Select cell as the target location for next move;
23      Move seed cell to target location and lock the target location;
24       $EndRipple \leftarrow FALSE$ ;  $AbortCount \leftarrow 0$ ;
25
26 end
```

GORDIAN Placement

- Kleinhans, Sigl and Johannes, “GORDIAN: VLSI placement by quadratic programming and slicing optimization,” TCAD, October 91 (ICCAD-90). 雖然quality沒到最好，但是速度很快
- Global placement (analytical scheme) + detailed placement (combinatorial search)

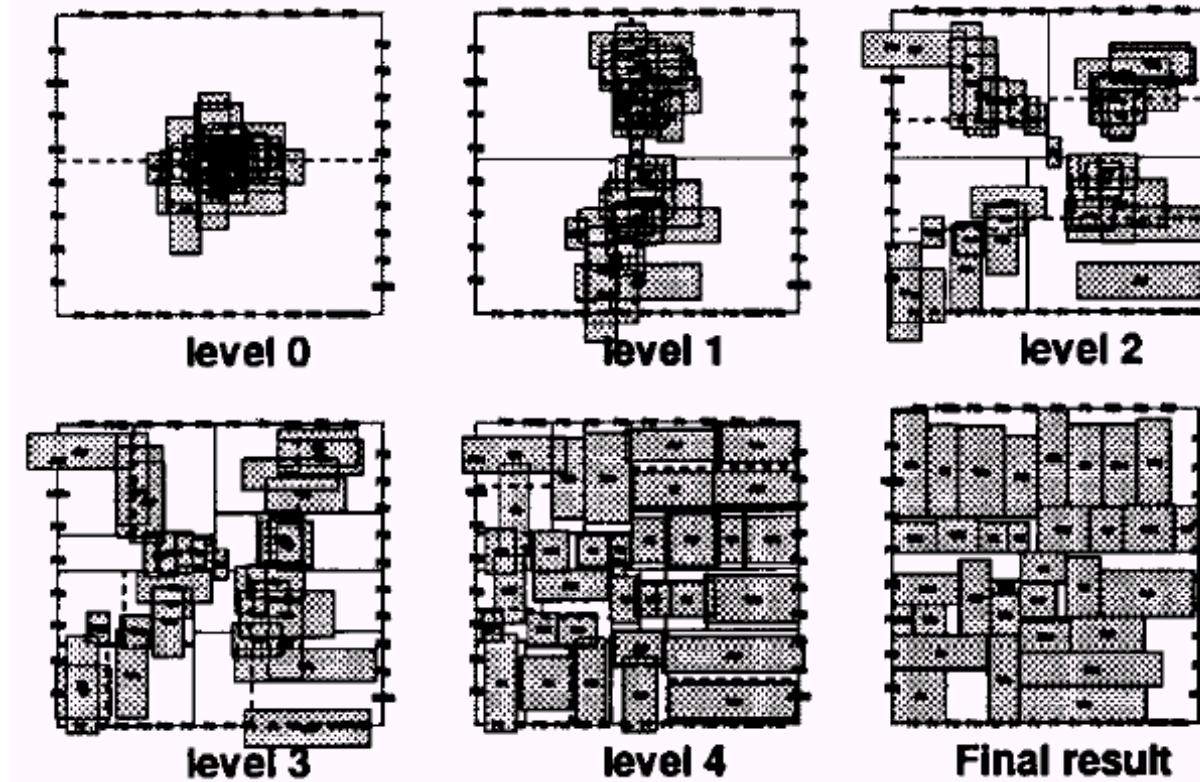


Data flow in the placement procedure GORDIAN.

Global Placement Optimization

- Apply quadratic programming.
- Objective function: wirelength.
 - Squared distance between modules and nets.
- Constraints: the centers of the regions on a level of partitioning for the movable modules.

level 0: 先做analytical placement得到初始座標(都很靠近中間)
level 1: 做partition，將level 0中比較偏上的cell都往上面的set擺，剩下的就放下面
level 2~n: 繼續partition，然後這些cell就會慢慢散開



Objective Function

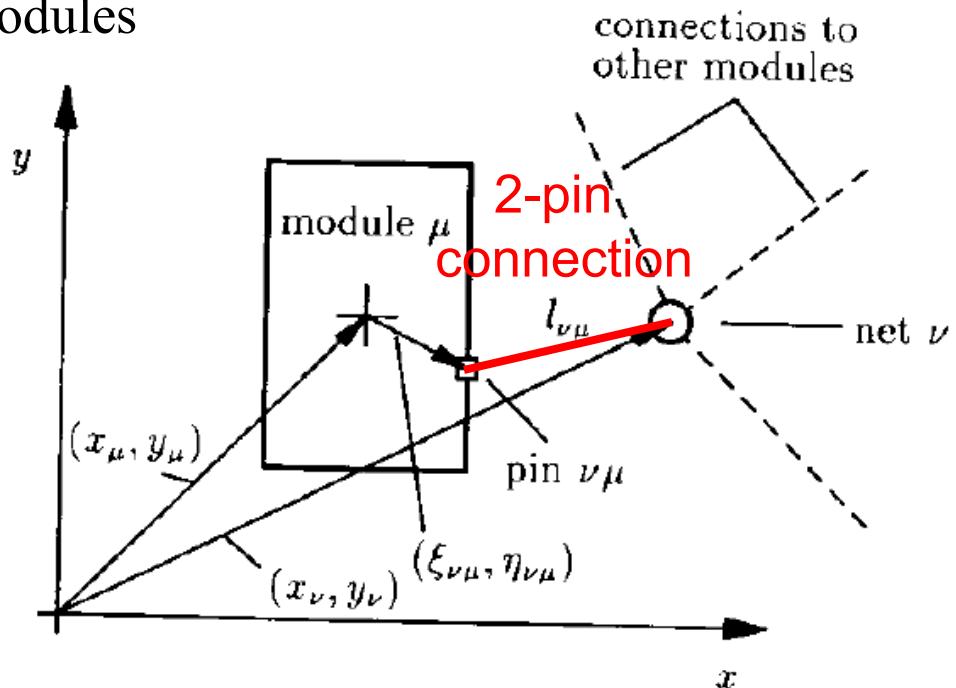
- Objective function: Squared distance between modules and nets. $\phi(x, y) = \frac{1}{2} \sum_{\nu \in N} \sum_{\mu \in M} [(x_\mu + \xi_{\nu\mu} - x_\nu)^2 + (y_\mu + \eta_{\nu\mu} - y_\nu)^2] \cdot t_{\nu\mu} \cdot w_\nu$

- w_ν is the weight of net ν .
- $t_{\nu u} = 1$ if net ν connects to module u ; $t_{\nu u} = 0$, otherwise.
- $\phi(x, y) = \frac{1}{2} x^T Q x + d_x^T x + \frac{1}{2} y^T Q y + d_y^T y$, 實作的時候 x, y 是分開做的

x and y : coordinate of modules

Q : connectivity matrix,

d : originates from the contributions of the pads and the relative pin coordinates.



More on the Objective Function

(x_i, y_i) = Coordinates of the center of cell i

c_{ij} = Weight of the net between cells i and j

\mathbf{x}, \mathbf{y} = Solution vectors

Cost of the net between cells i and j

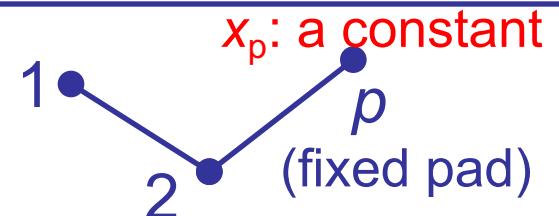
$$= \frac{1}{2} c_{ij} ((x_i - x_j)^2 + (y_i - y_j)^2)$$

$$\text{Total cost} = \frac{1}{2} \mathbf{x}^T Q \mathbf{x} + \mathbf{d}_x^T \mathbf{x} + \frac{1}{2} \mathbf{y}^T Q \mathbf{y} + \mathbf{d}_y^T \mathbf{y} + \text{constant}$$

Horizontal cost

$$= \frac{1}{2} c_{12} (x_1 - x_2)^2 + \frac{1}{2} c_{2p} (x_2 - x_p)^2$$

$$= \frac{1}{2} \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} c_{12} & -c_{12} \\ -c_{12} & c_{12} + c_{2p} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 & -c_{2p} x_p \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \frac{1}{2} c_{2p} x_p^2$$



Solution of Quadratic Placement

$$\text{Total cost } \tilde{L} = \frac{1}{2} \mathbf{x}^T Q \mathbf{x} + \mathbf{d}_x^T \mathbf{x} + \frac{1}{2} \mathbf{y}^T Q \mathbf{y} + \mathbf{d}_y^T \mathbf{y} + \text{const}$$

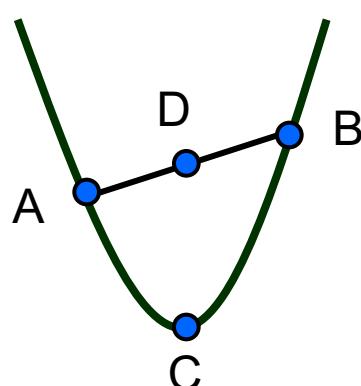
- The problems in x- and y-directions can be separated and solved independently
- *Ignore non-overlapping and other constraints for the time being*
- Minimize $\tilde{L}_x = \frac{1}{2} \mathbf{x}^T Q \mathbf{x} + \mathbf{d}_x^T \mathbf{x}$
- Q can be proved to be positive definite
- Thus, the function is convex
- Minimum solution can be found by setting derivatives to 0:

$$Q \mathbf{x} + \mathbf{d}_x = \mathbf{0}$$

Convexity

- The objective function is a **convex quadratic** function.
- Convexity:
 - If \mathbf{x} is a single variable, $f''(\mathbf{x}) \geq 0$.
 - If $\mathbf{x} = [x_1, x_2, \dots, x_n]$, $\mathbf{x}^T \mathbf{H}_f \mathbf{x} \geq 0$.

$$H_f = \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1n} \\ f_{21} & f_{22} & & \vdots \\ \vdots & & & \vdots \\ f_{n1} & \cdots & \cdots & f_{nn} \end{pmatrix}$$



A: $(x, f(x))$

B: $(x', f(x'))$

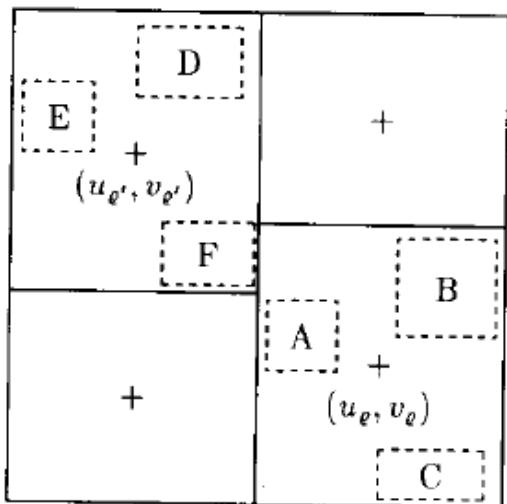
C: $(cx + (1-c)x', f(cx + (1-c)x'))$

D: $(cx + (1-c)x', cf(x) + (1-c)f(x'))$

Convex: $cf(x) + (1-c)f(x') \geq f(cx + (1-c)x')$

Constraints

- Constraints: make the center of module gravity correspond to the center of the region, $A^{(l)}x = u^{(l)}$; each entry in A gives the area ratio of the module in the corresponding region.
- Constraints are linear (and thus also convex).



$$A^{(l)} = \begin{bmatrix} A & B & C & D & E & F & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varrho & * & * & * & 0 & 0 & 0 \\ \varrho' & 0 & 0 & 0 & * & * & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

$$x_a f_a + x_b f_b + x_c f_c = u_\rho (f_a + f_b + f_c)$$

$$x_d f_d + x_e f_e + x_f f_f = u_{\rho'} (f_d + f_e + f_f)$$

f_m : the area of module m, u_ρ : the center of a region ρ ,
 x_m : the coordinate of the center of module m.

Quadratic Programming

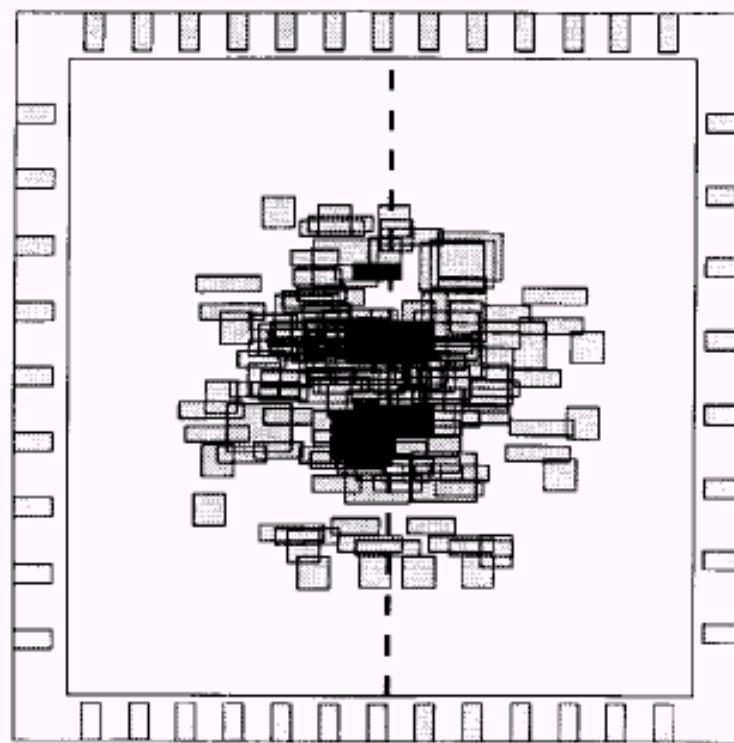
- The objective function is a **convex quadratic** function.
 - C is positive definite.
- Constraints are **linear** (and thus **convex**).
- QP has a unique global minimum $\phi(x^*)$.
- **QP can be solved optimally in polynomial time.**
- Ignore the non-overlapping constraint at the global placement stage.
- The problem in x- and y-directions can be separated and solved independently.

$$\text{QP: } \min_x \left\{ \phi(x) = \frac{1}{2} x^T C x + d^T x \mid A^{(1)} x = u^{(1)} \right\}.$$

- Minimum solution can be found by setting derivatives to 0.

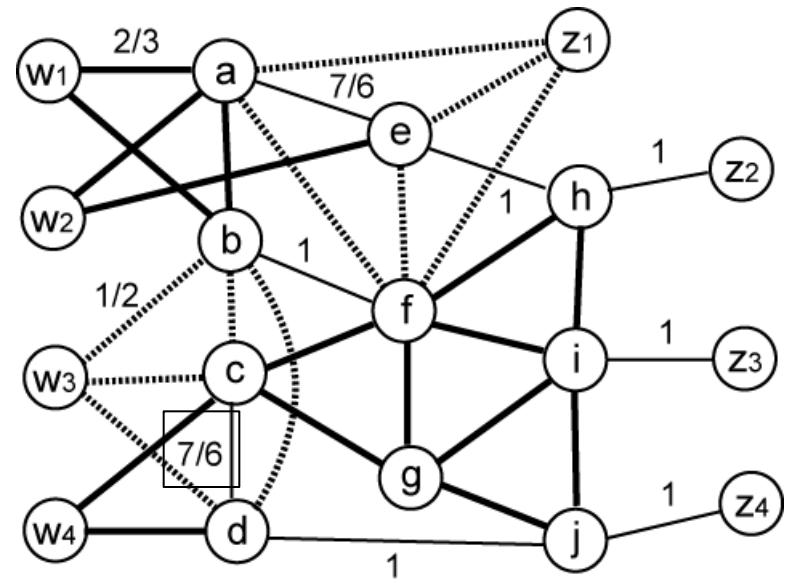
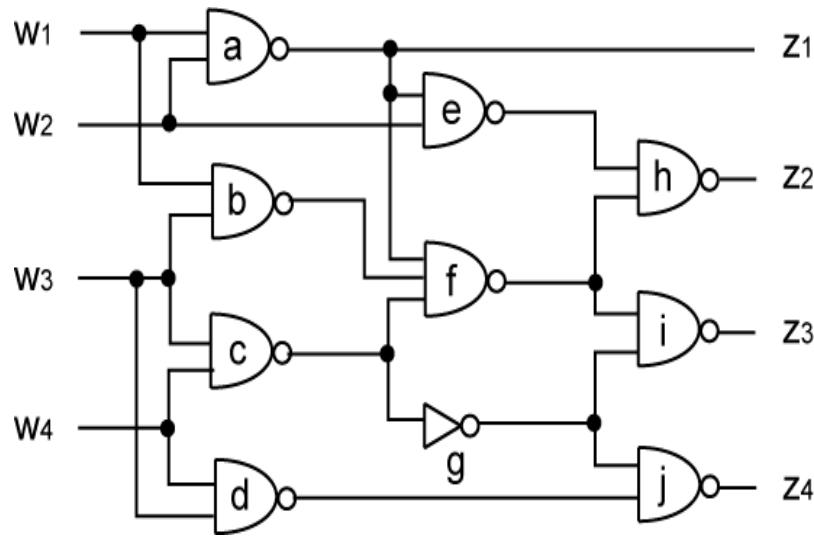
Partitioning

- After global optimization, modules are divided into two parts such that the module areas of both subsets are about the same.
 - Sort the x- (y-) coordinates of the modules and cut the region vertically (horizontally).



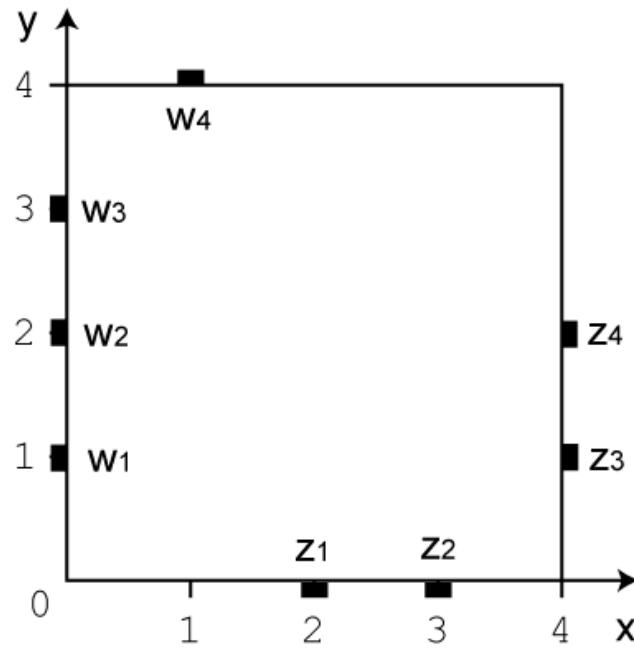
Practice Problem

- Perform GORDIAN placement
 - Uniform area and net weight, area balance factor = 0.5
 - Undirected graph model: each edge in k -clique gets weight $2/k$



IO Placement

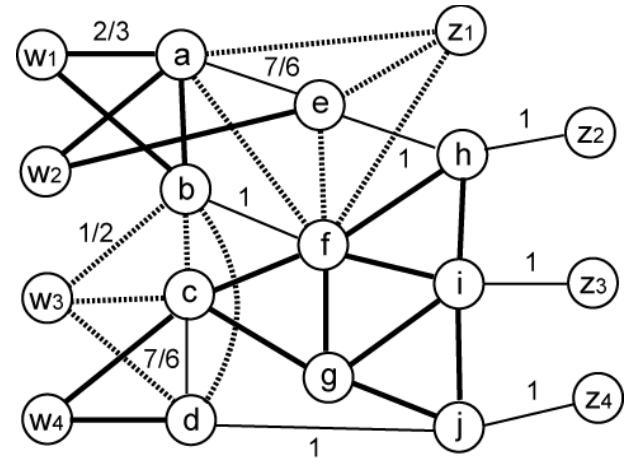
- Necessary for GORDIAN to work



Adjacency Matrix

- Shows connections among movable nodes
 - Among nodes a to j

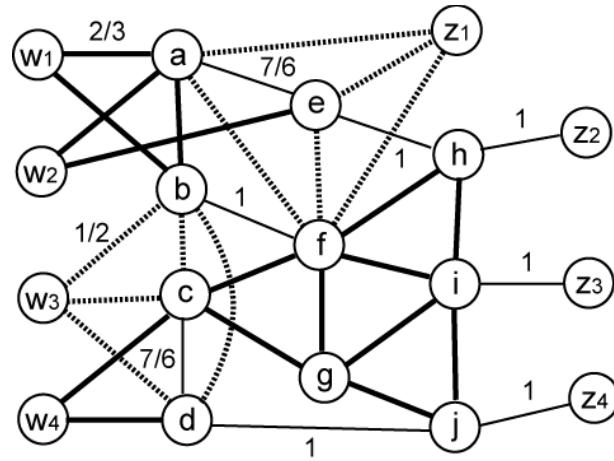
$$\begin{pmatrix} 0 & \frac{2}{3} & 0 & 0 & \frac{7}{6} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{2}{3} & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{7}{6} & 0 & \frac{2}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{7}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{7}{6} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 1 & 0 & 0 \\ \frac{1}{2} & 1 & \frac{2}{3} & 0 & \frac{1}{2} & 0 & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & 0 \\ 0 & 0 & \frac{2}{3} & 0 & 0 & \frac{2}{3} & 0 & 0 & \frac{2}{3} & \frac{2}{3} \\ 0 & 0 & 0 & 0 & 1 & \frac{2}{3} & 0 & 0 & \frac{2}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & 0 & \frac{2}{3} \\ 0 & 0 & 0 & 1 & 0 & 0 & \frac{2}{3} & 0 & \frac{2}{3} & 0 \end{pmatrix}$$



Pin Connection Matrix

- Shows connections between movable nodes and IO
 - Rows = movable nodes, columns = IO (fixed)

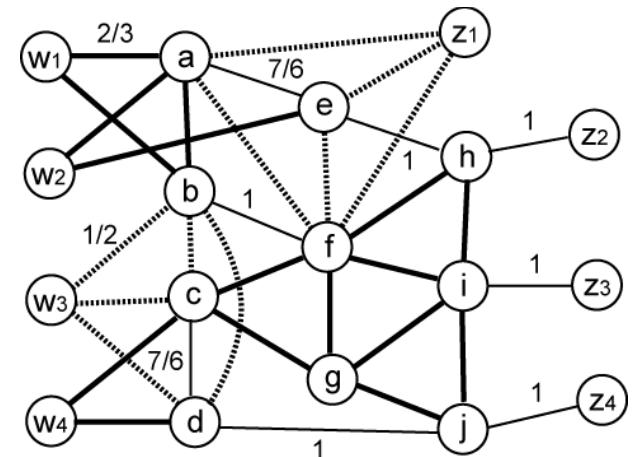
$$\begin{pmatrix} \frac{2}{3} & \frac{2}{3} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{2}{3} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{2}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{2}{3} & 0 & 0 & 0 & 0 \\ 0 & \frac{2}{3} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$



Degree Matrix

- Based on both adjacency and pin connection matrices
 - Sum of entries in the same row (= node degree)

$$\begin{pmatrix} \frac{25}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{23}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{25}{6} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{23}{6} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{23}{6} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{31}{6} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{8}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{10}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{11}{3} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{10}{3} \end{pmatrix}$$



Connectivity/Laplacian Matrix

- Degree matrix minus adjacency matrix

$$\begin{pmatrix} \frac{25}{6} & -\frac{2}{3} & 0 & 0 & -\frac{7}{6} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ -\frac{2}{3} & \frac{23}{6} & -\frac{1}{2} & -\frac{1}{2} & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & \frac{25}{6} & -\frac{7}{6} & 0 & -\frac{2}{3} & -\frac{2}{3} & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & -\frac{7}{6} & \frac{23}{6} & 0 & 0 & 0 & 0 & 0 & -1 \\ -\frac{7}{6} & 0 & 0 & \frac{23}{6} & -\frac{1}{2} & 0 & -1 & 0 & 0 & 0 \\ -\frac{1}{2} & -1 & -\frac{2}{3} & 0 & -\frac{1}{2} & \frac{31}{6} & -\frac{2}{3} & -\frac{2}{3} & -\frac{2}{3} & 0 \\ 0 & 0 & -\frac{2}{3} & 0 & 0 & -\frac{2}{3} & \frac{8}{3} & 0 & -\frac{2}{3} & -\frac{2}{3} \\ 0 & 0 & 0 & 0 & -1 & -\frac{2}{3} & 0 & \frac{10}{3} & -\frac{2}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{2}{3} & -\frac{2}{3} & -\frac{2}{3} & \frac{11}{3} & -\frac{2}{3} \\ 0 & 0 & 0 & -1 & 0 & 0 & -\frac{2}{3} & 0 & -\frac{2}{3} & \frac{10}{3} \end{pmatrix}$$

Fixed Pin Vectors

- Based on pin connection matrix and IO location

Each entry i in d_x , denoted $d_{x,i}$, is computed as follows:

$$d_{x,i} = - \sum_j p_{ij} \cdot x(p_j)$$

where p_{ij} denotes the entry of the pin connection matrix, and $x(p_j)$ is the x -coordinate of the corresponding IO pin j .

— ~~Y-direction is defined similarly~~

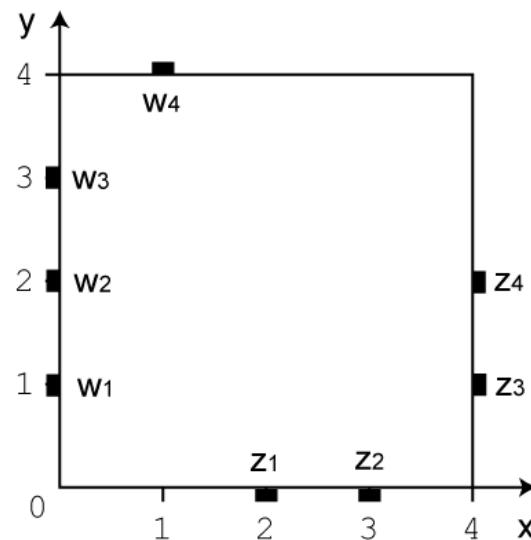
Fixed Pin Vectors (cont)

$$d_{x,1} = -\left(\frac{2}{3} \cdot 0 + \frac{2}{3} \cdot 0 + 0 \cdot 0 + 0 \cdot 1 + \frac{1}{2} \cdot 2 + 0 \cdot 3 + 0 \cdot 4 + 0 \cdot 4\right) = -1$$

By examining the remaining 9 movable cells, we get

$$d_x^T = \left(-1 \quad 0 \quad -\frac{2}{3} \quad -\frac{2}{3} \quad -1 \quad -1 \quad 0 \quad -3 \quad -4 \quad -4\right)$$

$$\begin{pmatrix} \frac{2}{3} & \frac{2}{3} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{2}{3} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{2}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{2}{3} & 0 & 0 & 0 & 0 \\ 0 & \frac{2}{3} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$



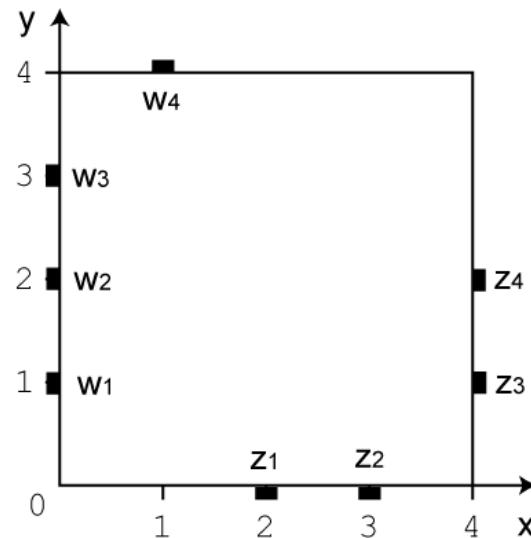
Fixed Pin Vectors (cont)

$$d_{y,1} = -\left(\frac{2}{3} \cdot 1 + \frac{2}{3} \cdot 2 + 0 \cdot 3 + 0 \cdot 4 + \frac{1}{2} \cdot 0 + 0 \cdot 0 + 0 \cdot 1 + 0 \cdot 2\right) = -2$$

By examining the remaining 9 movable cells, we get

$$d_y^T = \left(-2 \quad -\frac{13}{6} \quad -\frac{25}{6} \quad -\frac{25}{6} \quad -\frac{4}{3} \quad 0 \quad 0 \quad 0 \quad -1 \quad -2\right)$$

$$\begin{pmatrix} \frac{2}{3} & \frac{2}{3} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{2}{3} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{2}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{2}{3} & 0 & 0 & 0 & 0 \\ 0 & \frac{2}{3} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$



Level 0 QP Formulation

- No constraint necessary

Minimize

$$\phi(x) = \frac{1}{2}x^T Cx + d_x^T x$$

and

$$\phi(y) = \frac{1}{2}y^T Cy + d_y^T y$$

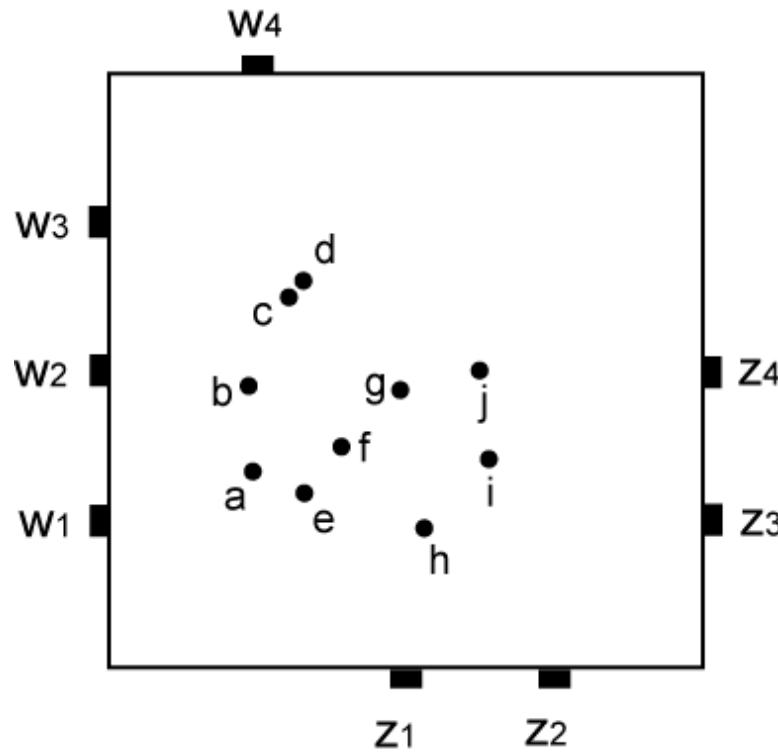
We use MOSEK and obtain the following solution:

$$x^T = (0.95 \quad 0.92 \quad 1.21 \quad 1.32 \quad 1.32 \quad 1.61 \quad 1.98 \quad 2.13 \quad 2.59 \quad 2.51)$$

$$y^T = (1.27 \quad 1.83 \quad 2.48 \quad 2.61 \quad 1.16 \quad 1.45 \quad 1.84 \quad 0.92 \quad 1.41 \quad 2.03)$$

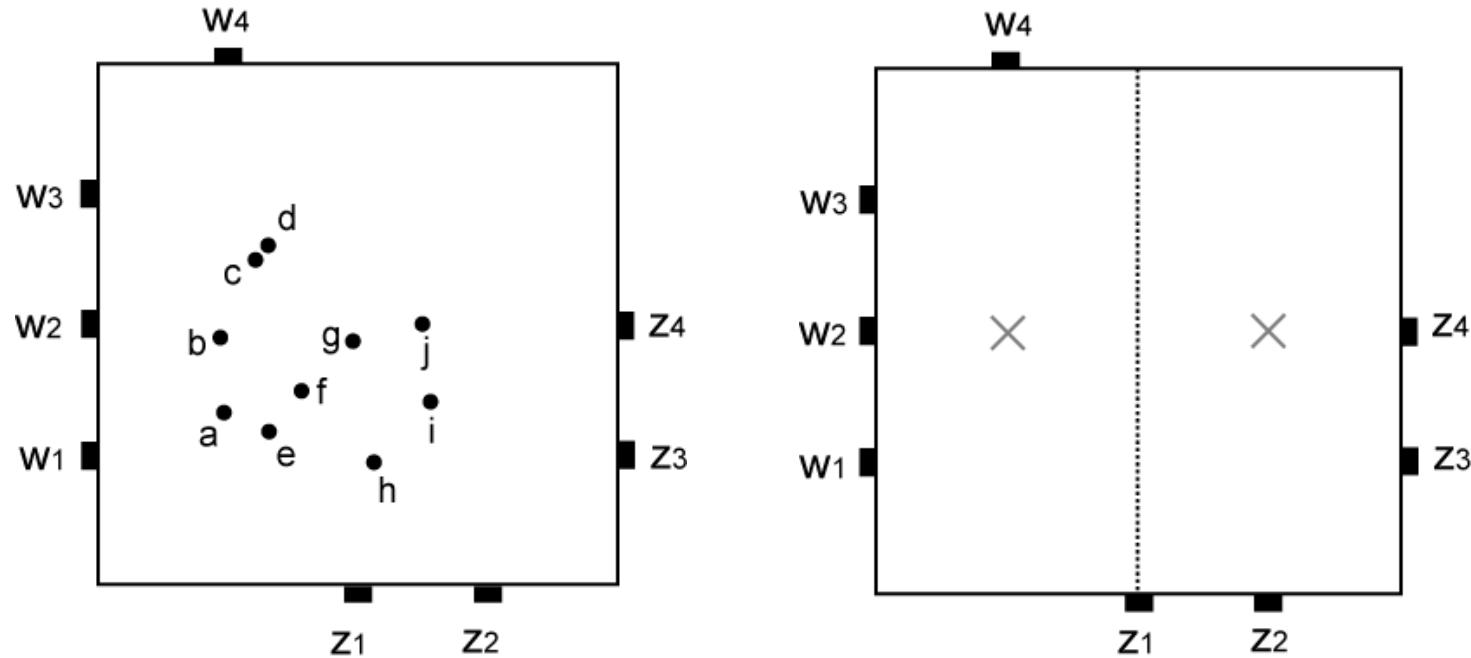
Level 0 Placement

- Cells with real dimension will overlap



Level 1 Partitioning

- Perform level 1 partitioning
 - Obtain center locations for center-of-gravity constraints



Level 1 Constraint

We first sort the nodes based on their x -coordinates:

$$\{b, a, c, e, d, f, g, h, j, i\}$$

We perform partitioning under $\alpha = 0.5$:

$$S_{\rho'} = \{b, a, c, e, d\}, \quad S_{\rho''} = \{f, g, h, j, i\}$$

The center location vectors are:

$$u_x^{(1)} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}, \quad u_y^{(1)} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

We build the matrix $A^{(1)}$ for the center-of-gravity constraint at level $l = 1$:

$$A^{(1)} = \begin{pmatrix} \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{pmatrix}$$

Level 1 LQP Formulation

We now solve the following Linearly constrained QP (LQP) to obtain the new placement for the movable nodes:

$$\text{Minimize } \phi(x) = \frac{1}{2}x^T C x + d_x^T x, \text{ subject to } A^{(1)} \cdot x = u_x^{(1)}$$

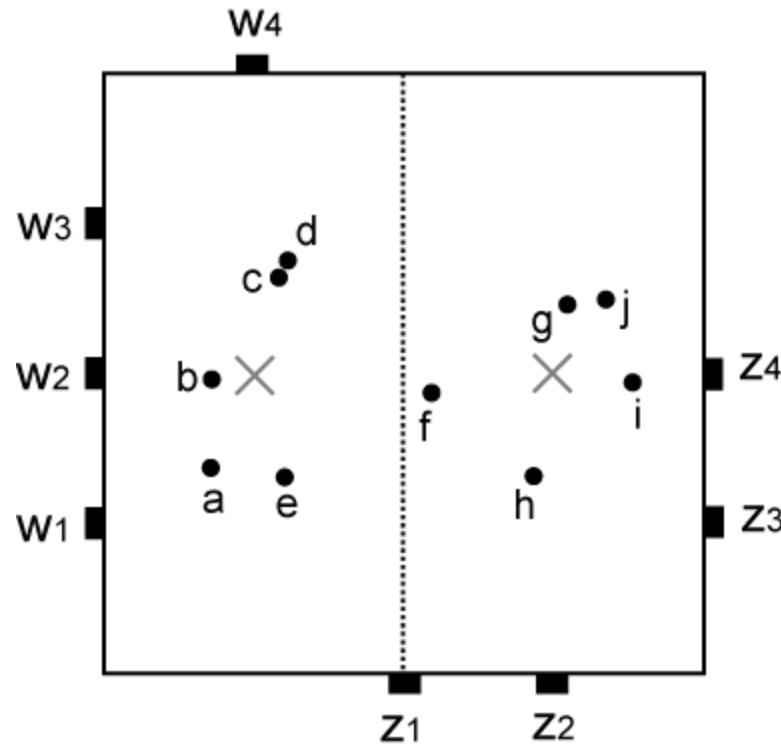
$$\text{Minimize } \phi(y) = \frac{1}{2}y^T C y + d_y^T y, \text{ subject to } A^{(1)} \cdot y = u_y^{(1)}$$

The solutions are as follows:

$$x^T = (0.70 \quad 0.71 \quad 1.17 \quad 1.21 \quad 1.22 \quad 2.17 \quad 3.10 \quad 2.84 \quad 3.56 \quad 3.33)$$

$$y^T = (1.34 \quad 1.94 \quad 2.66 \quad 2.76 \quad 1.30 \quad 1.83 \quad 2.45 \quad 1.32 \quad 1.91 \quad 2.49)$$

Level 1 Placement



Verification

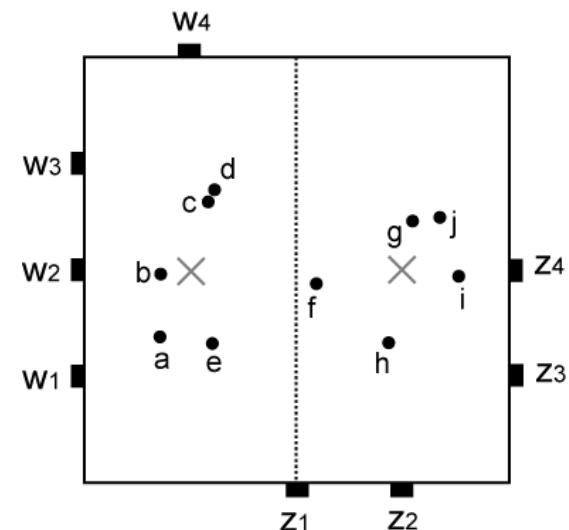
- Verify that the constraints are satisfied in the left partition

The following cells are partitioned to the left: $a(0.70, 1.34)$, $b(0.71, 1.94)$, $c(1.17, 2.66)$, $d(1.21, 2.76)$, and $e(1.22, 1.30)$. Thus, the center of gravity is located at:

$$\frac{0.70 + 0.71 + 1.17 + 1.21 + 1.22}{5} = 1.00$$

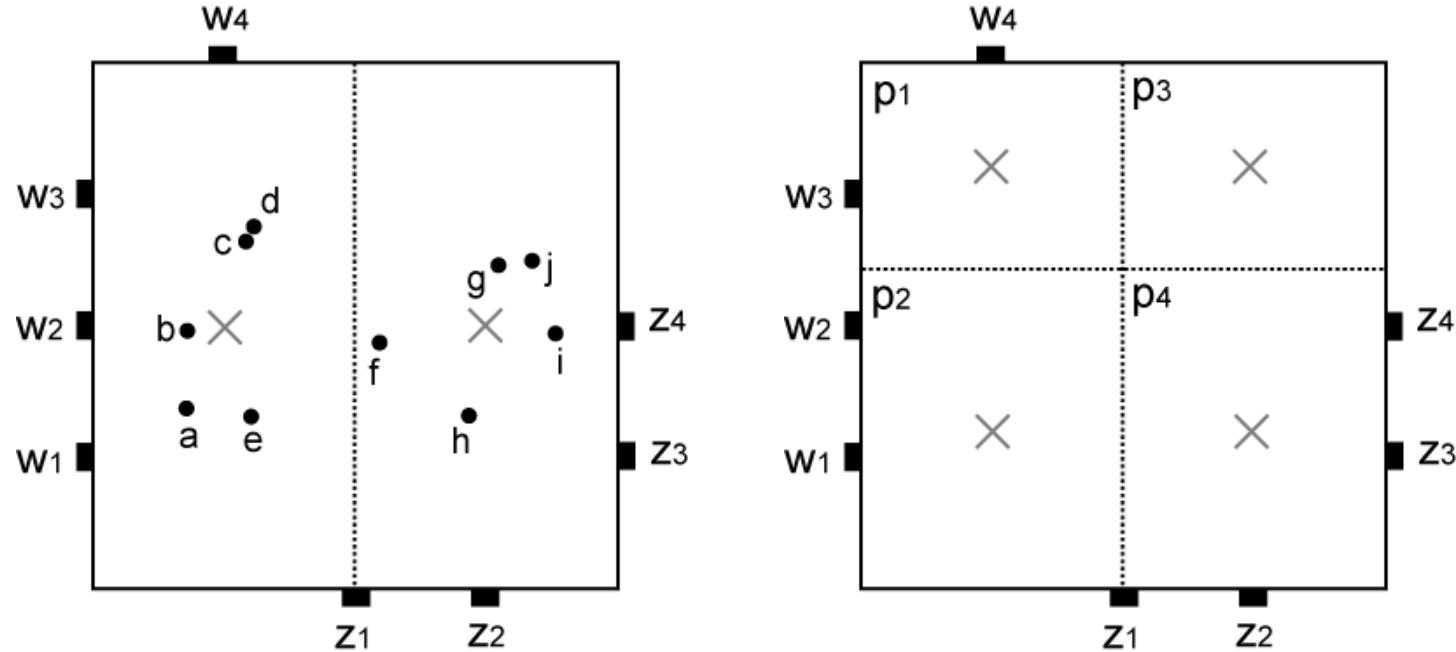
$$\frac{1.34 + 1.94 + 2.66 + 2.76 + 1.30}{5} = 2.00$$

This agrees with the center location (1, 2).



Level 2 Partitioning

- Add two more cut-lines
 - This results in $p_1=\{c,d\}$, $p_2=\{a,b,e\}$, $p_3=\{g,j\}$, $p_4=\{f,h,i\}$



Level 2 Constraint

The center location vectors are:

$$u_x^{(2)} = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 3 \end{pmatrix}, \quad u_y^{(2)} = \begin{pmatrix} 3.2 \\ 1.2 \\ 3.2 \\ 1.2 \end{pmatrix}$$

Next, we build the matrix $A^{(2)}$ for the center-of-gravity constraint at level $l = 2$. Recall that $p_1 = \{c, d\}$, $p_2 = \{a, b, e\}$, $p_3 = \{g, j\}$, $p_4 = \{f, h, i\}$. Thus,

$$A^{(2)} = \begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

where the rows denote the partitions p_1 through p_4 , and the columns denote the cells a through j .

Level 2 LQP Formulation

We now solve the following LQP to obtain the placement of the movable nodes:

$$\text{Minimize } \phi(x) = \frac{1}{2}x^T C x + d_x^T x, \text{ subject to } A^{(2)} \cdot x = u_x^{(2)}$$

$$\text{Minimize } \phi(y) = \frac{1}{2}y^T C y + d_y^T y, \text{ subject to } A^{(2)} \cdot y = u_y^{(2)}$$

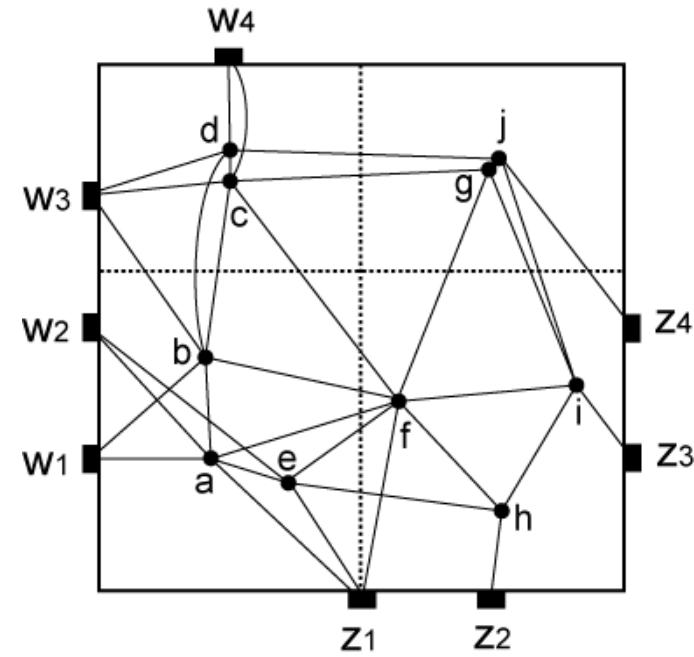
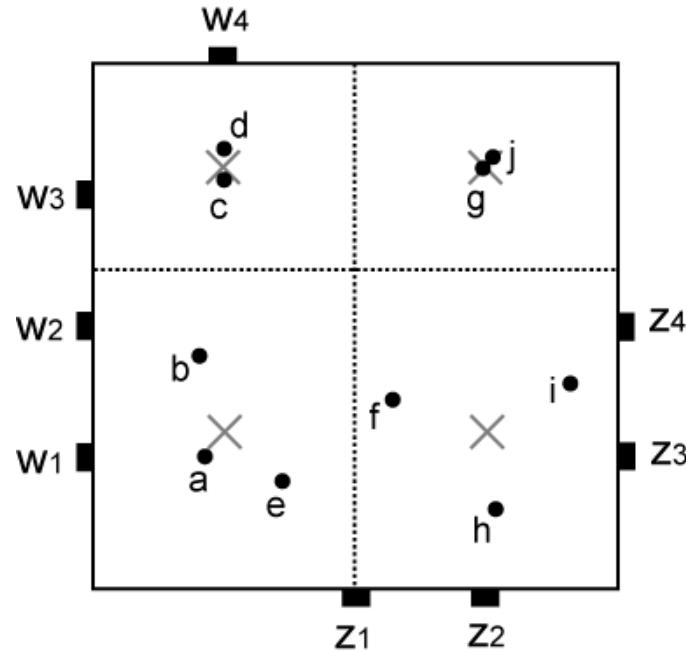
The solutions are as follows:

$$x^T = (0.83 \quad 0.78 \quad 1.00 \quad 1.00 \quad 1.39 \quad 2.28 \quad 2.89 \quad 3.06 \quad 3.66 \quad 3.11)$$

$$y^T = (1.01 \quad 1.78 \quad 3.08 \quad 3.32 \quad 0.82 \quad 1.44 \quad 3.18 \quad 0.59 \quad 1.57 \quad 3.22)$$

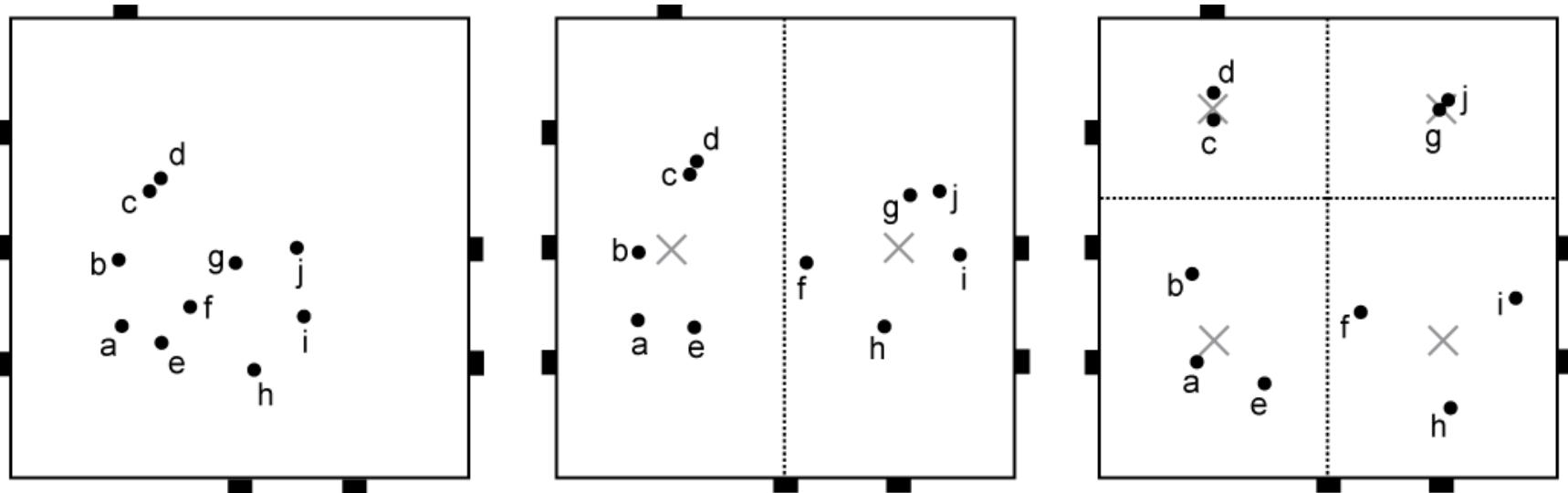
Level 2 Placement

- Clique-based wiring is shown



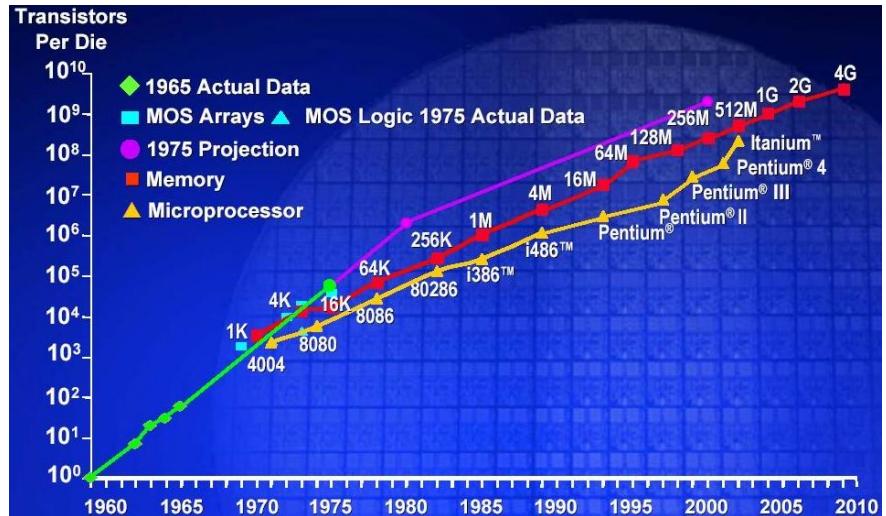
Summary

- Center-of-gravity constraint
 - Helps spread the cells evenly while monitoring wirelength
 - Removes overlaps among the cells (with real dimension)

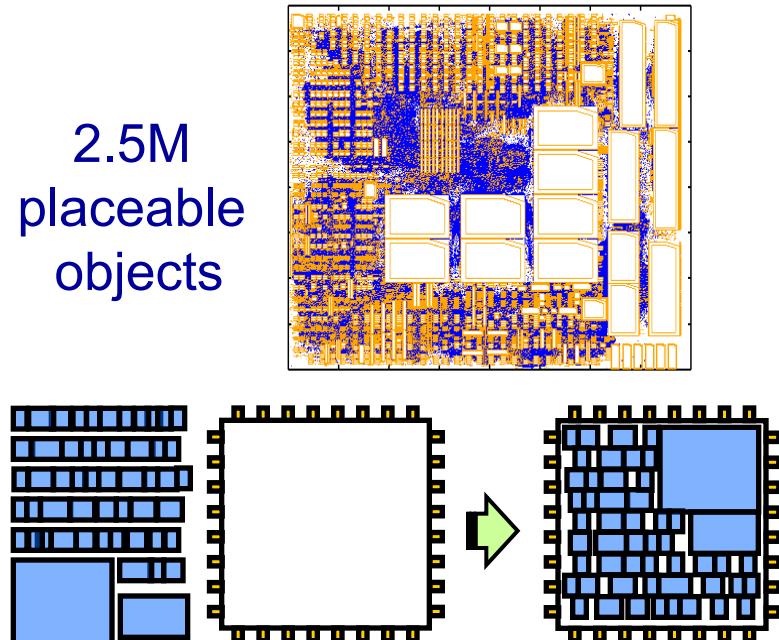


Modern Placement Challenges

- High complexity
 - Millions of objects to be placed
- Mixed-size placement
 - Hundreds of movable macro blocks with millions of small standard cells
- Placement constraints
 - Preplaced blocks
 - Chip density
 - etc.

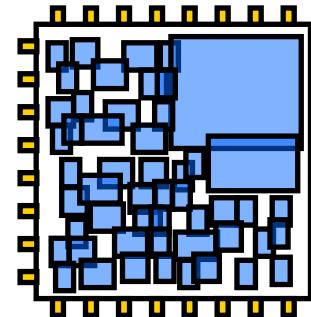
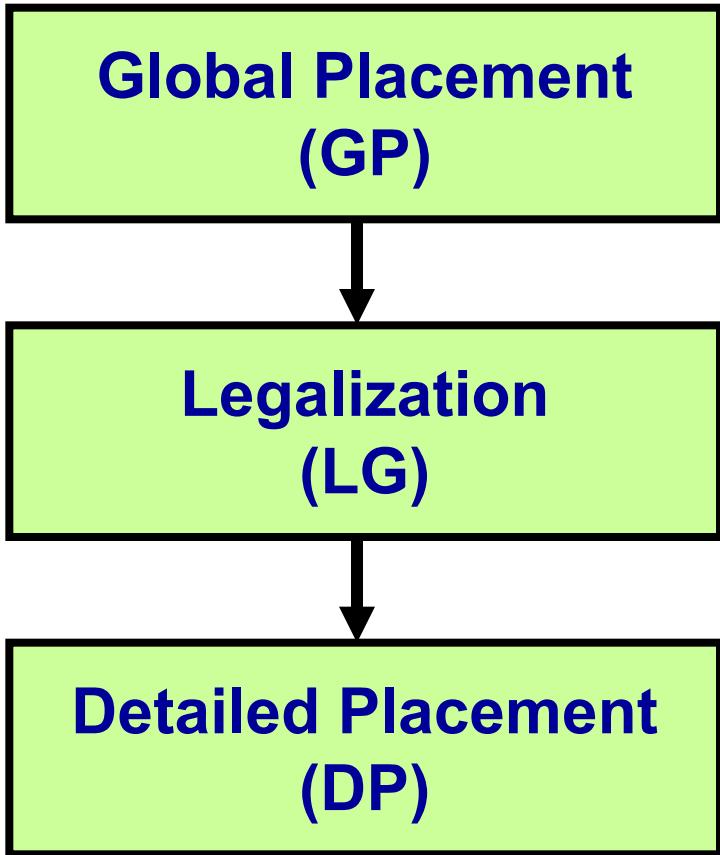


2.5M
placeable
objects

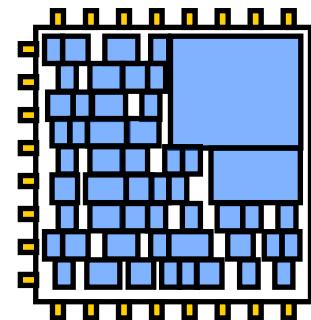
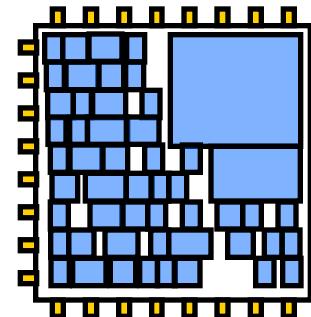


NTUPlace3 Standard-Cell Placement Flow

- Chen, et al., “A high quality analytical placer considering preplaced blocks and density constraint,” ICCAD-06 (TCAD-08).



prototyping



Placement with Density Constraint

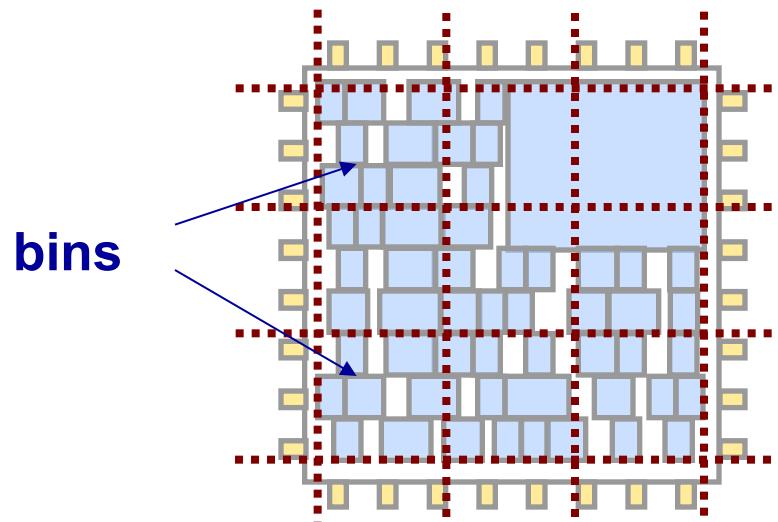
- Given the chip region and block dimensions, divide the placement region into bins
- Determine (x, y) for all movable blocks

$\min W(x, y)$ -- wirelength function

s.t. 1. $\text{Density}_b(x, y) \leq \text{MaximumDensity}_b$

希望cell可以均匀分散在chip中，不會全部擠在一起，避免繞線時的麻煩 for each bin b

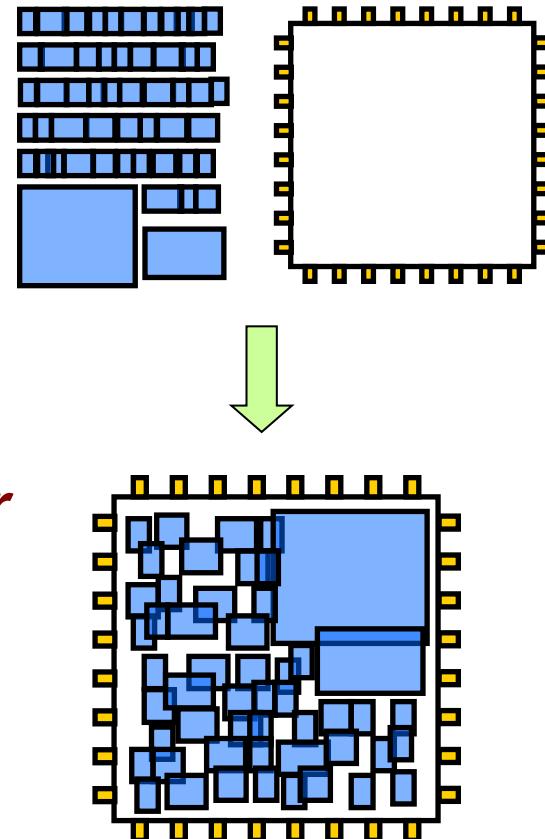
2. No overlap between blocks



$$\text{Density} = \frac{A_{\text{block}}}{A_{\text{bin}}}$$

Global Placement

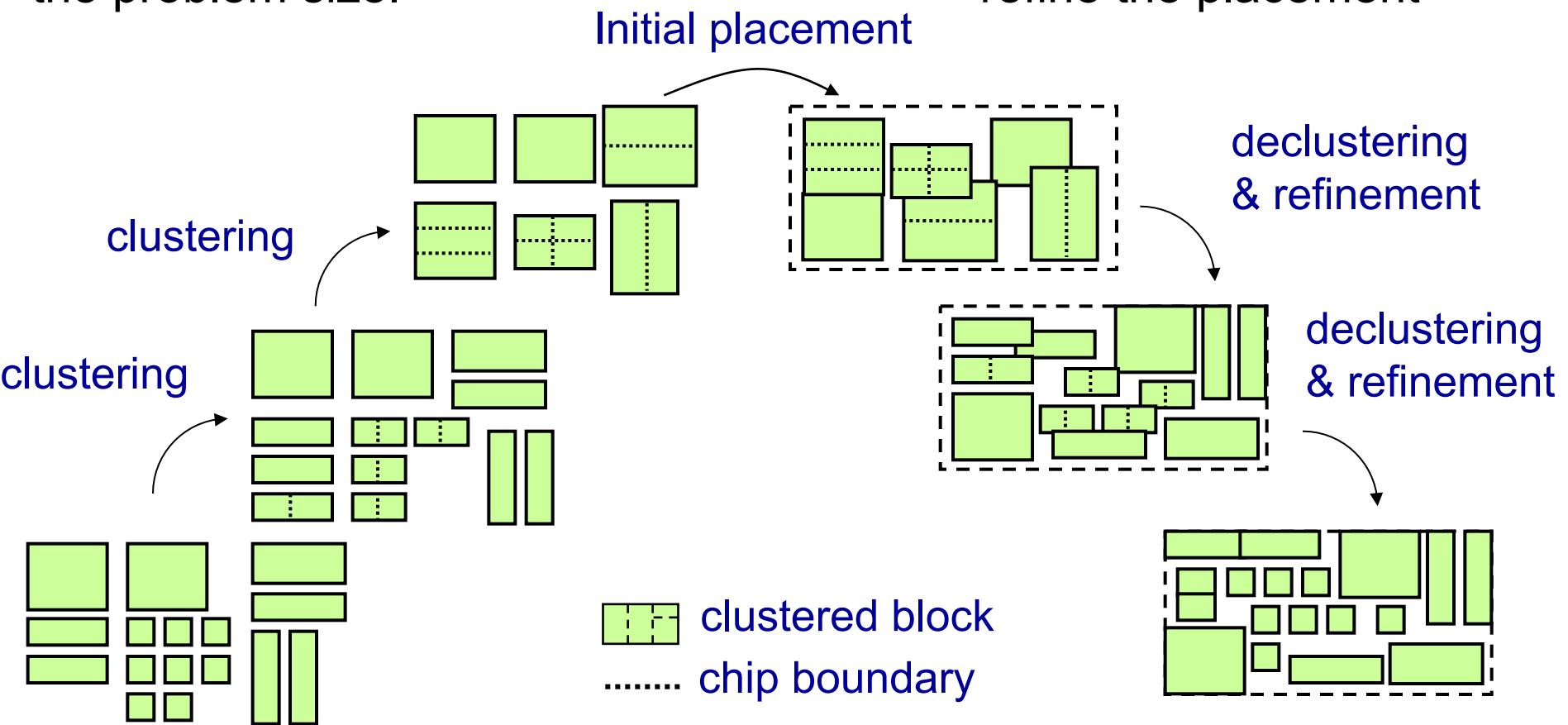
- Placement flow
 - ***Global placement***
 - ***Multilevel framework***
 - ***Analytical placement with a nonlinear objective function***
 - ***Free-space allocation for density control***
 - ***Smoothing functions for preplaced blocks***
 - Legalization
 - Detailed placement



Multilevel Global Placement

Cluster the blocks based on connectivity/size to reduce the problem size.

Iteratively decluster the clusters and further refine the placement



Analytical Placement Model

- Analytical placement during declustering
- Global placement problem (allow overlaps)

$$\min W(x, y)$$

$$\text{s.t. } D_b(x, y) \leq M_b$$

Minimize HPWL

D_b : density for bin b

M_b : max density for bin b

- Relax the constraints into the objective function

$$\min W(x, y) + \lambda \sum (D_b(x, y) - M_b)^2$$

- Use the gradient method to solve it
- Increase λ gradually to find the optimal (x, y)

Gradient Solver

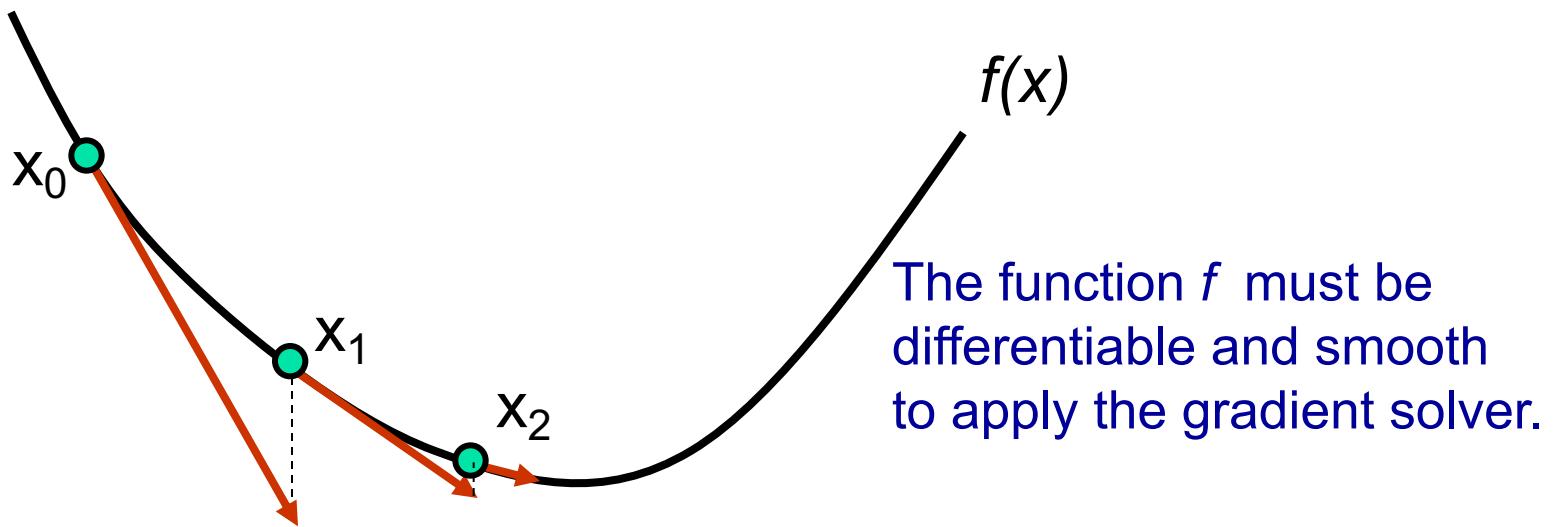
$$\min f(x)$$

[Gradient Solver]

$x_0 \leftarrow$ initial value

Repeat until convergence

$$x_{i+1} = x_i - f'(x)|_{x=x_i} * \text{stepsize}$$



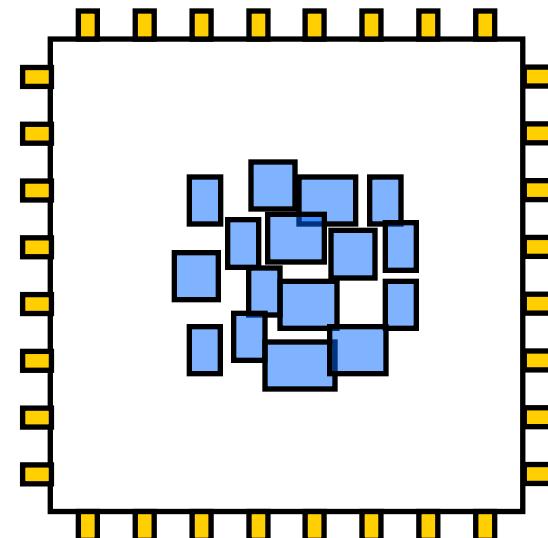
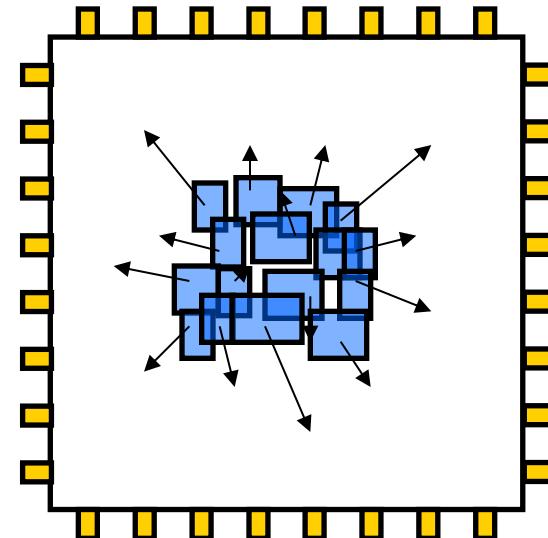
Dynamic Step-Size Control

- Step size is too large
 - May not converge to a good solution
- Step size is too small
 - Incur long running time
- Adjust the step size s.t. the average Euclidean movement of all blocks is a fixed value

$$\text{step size } \alpha_k = \frac{s}{\|\mathbf{d}_k\|_2}$$

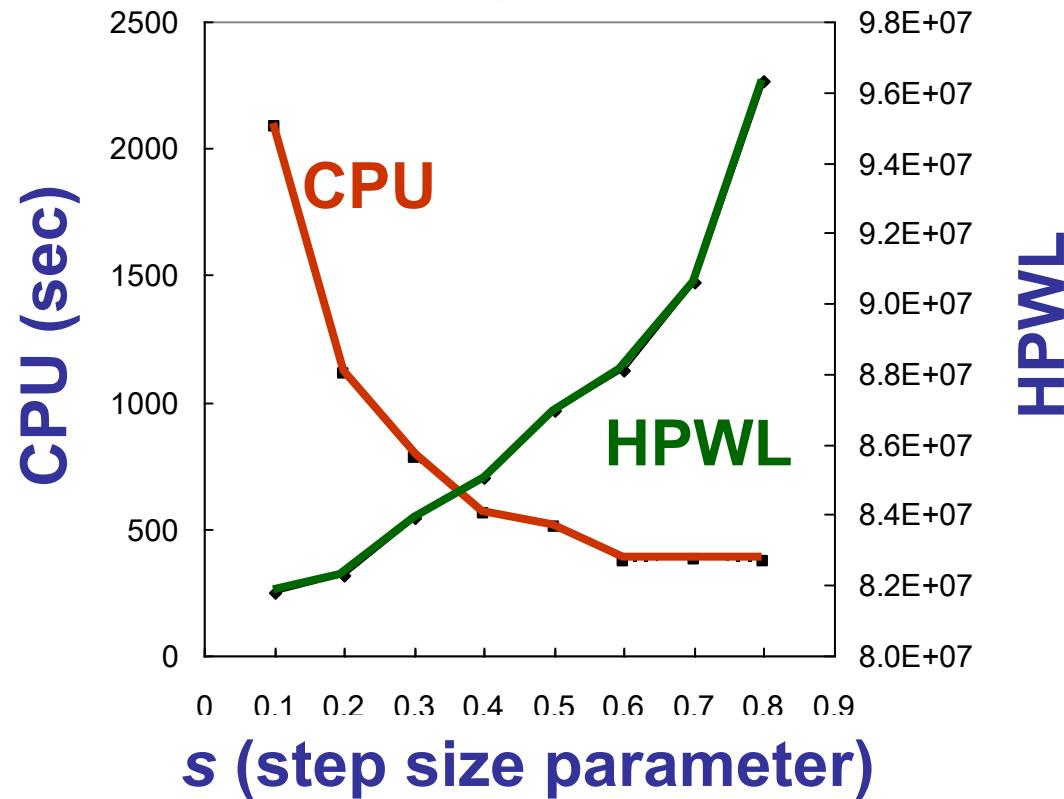
\mathbf{d}_k conjugate directions

s a user-specified value



Effects of the Step-Size Parameter

- Small step size
 - Larger runtime, smaller HPWL
- Large step size
 - Smaller runtime, larger HPWL



Wirelength Model

- Half perimeter wirelength (HPWL): not smooth/differentiable

$$W(\mathbf{x}, \mathbf{y}) = \sum_{\text{net } e} \left(\max_{v_i, v_j \in e} |x_i - x_j| + \max_{v_i, v_j \in e} |y_i - y_j| \right)$$

- Apply the **log-sum-exp wirelength model**: NTUplace3, mPL, APlace

$$\begin{aligned} & \gamma \sum_{e \in E} \left(\log \sum_{v_k \in e} \exp(x_k/\gamma) + \log \sum_{v_k \in e} \exp(-x_k/\gamma) + \right. \\ & \quad \left. \log \sum_{v_k \in e} \exp(y_k/\gamma) + \log \sum_{v_k \in e} \exp(-y_k/\gamma) \right). \end{aligned}$$

- Is a patented method proposed by Naylor et al.
- Is an effective smooth & differentiable function for HPWL approximation
- Approaches exact HPWL when $\gamma \rightarrow 0$
- Achieves the best result among recent academic placers

Other Smooth Wire Models

- Quadratic model: BonnPlace, FastPlace, Kraftwerk, mFAR
 - Not exact wirelength modeling

$$\sum_{e \in E} \left(\sum_{v_i, v_j \in e, i < j} w_{ij} (x_i - x_j)^2 + \sum_{v_i, v_j \in e, i < j} w_{ij} (y_i - y_j)^2 \right)$$

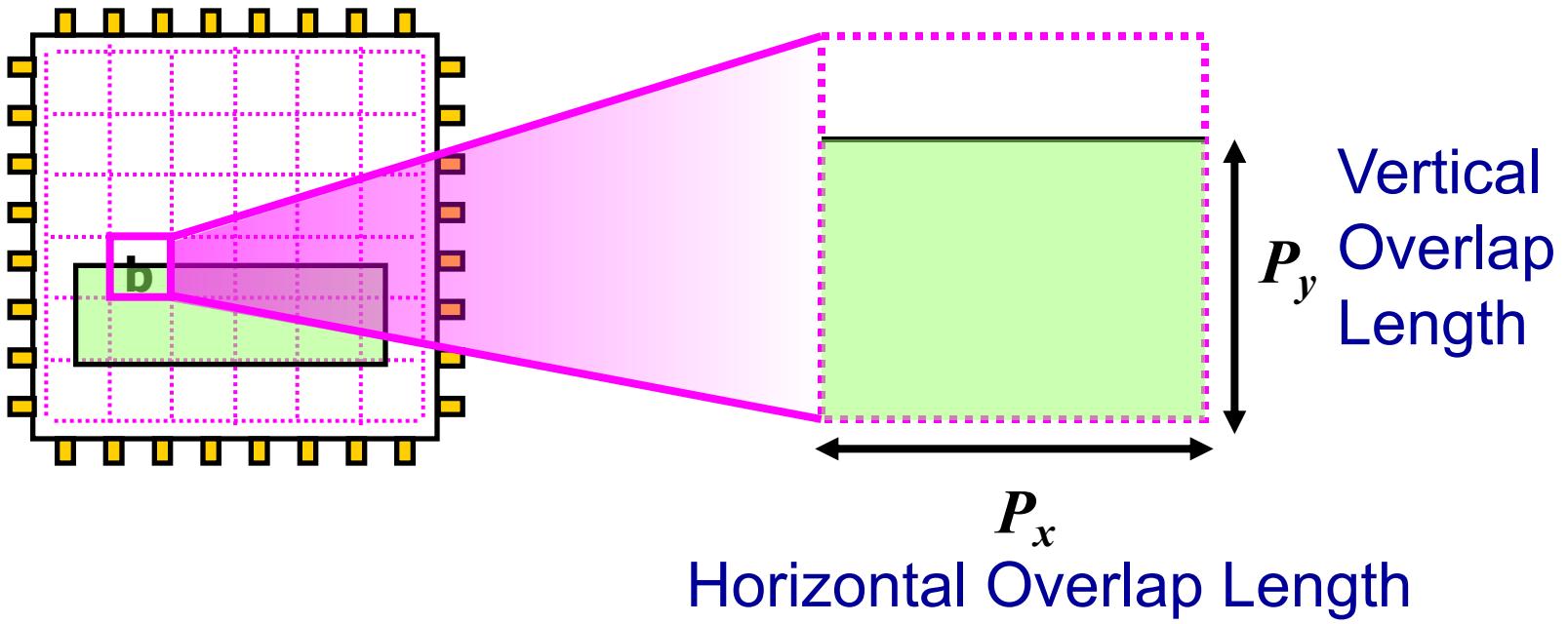
- L_p-norm model
 - Approaches exact HPWL when $p \rightarrow \infty$
 - Needs more time to evaluate the exponential function

$$\sum_{e \in E} \left(\left(\sum_{v_k \in e} x_k^p \right)^{\frac{1}{p}} - \left(\sum_{v_k \in e} x_k^{-p} \right)^{-\frac{1}{p}} + \left(\sum_{v_k \in e} y_k^p \right)^{\frac{1}{p}} - \left(\sum_{v_k \in e} y_k^{-p} \right)^{-\frac{1}{p}} \right)$$

- Typical quality & speed ranking: **log-sum-exp > L_p-norm > Quadratic**

Density Model

- Compute the block area in each bin to obtain the bin density

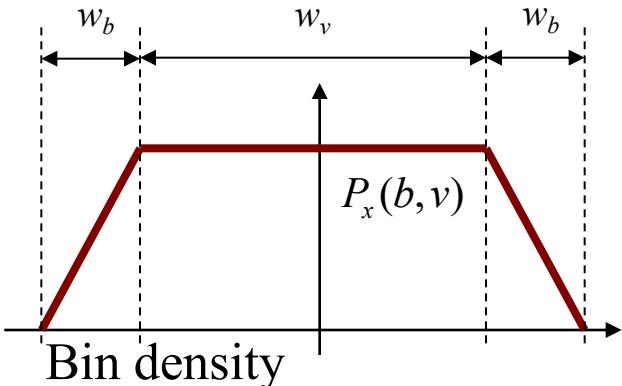


Bin density

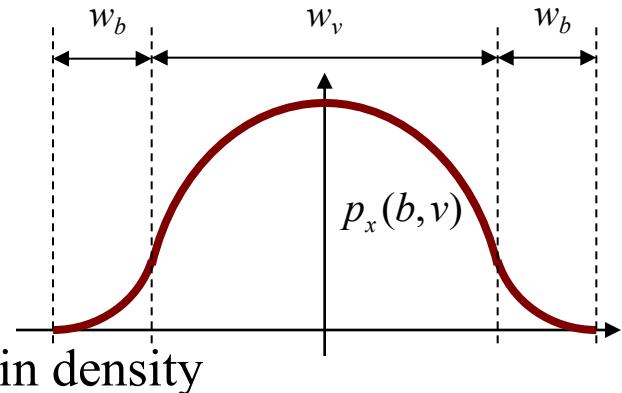
$$D_b(\mathbf{x}, \mathbf{y}) = \sum_{v \in V} P_x(b, v)P_y(b, v)$$

Density Smoothing

- Apply the bell-shaped function to make bin density function smooth (Kahng & Wang)



Bell-shaped smooth function
 Continuous & differentiable



Overlap length function

$$D_b(\mathbf{x}, \mathbf{y}) = \sum_{v \in V} P_x(b, v) P_y(b, v)$$

$$D'_b(\mathbf{x}, \mathbf{y}) = \sum_{v \in V} c_v p_x(b, v) p_y(b, v)$$

$$p_x(b, v) = \begin{cases} 1 - ad_x^2, & 0 \leq d_x \leq w_v/2 + w_b \\ b(d_x - 2w_b - 2w_g)^2, & w_v/2 + w_b \leq d_x \leq w_v/2 + 2w_b \\ 0, & w_v/2 + 2w_b \leq d_x, \end{cases}$$

where

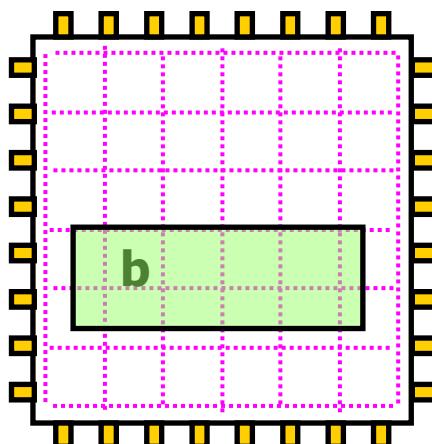
$$a = 4/((w_v + 2w_b)(w_v + 4w_b))$$

$$b = 2/(w_b(w_v + 4w_b)),$$

w_b bin width

w_v block width

normalization factor

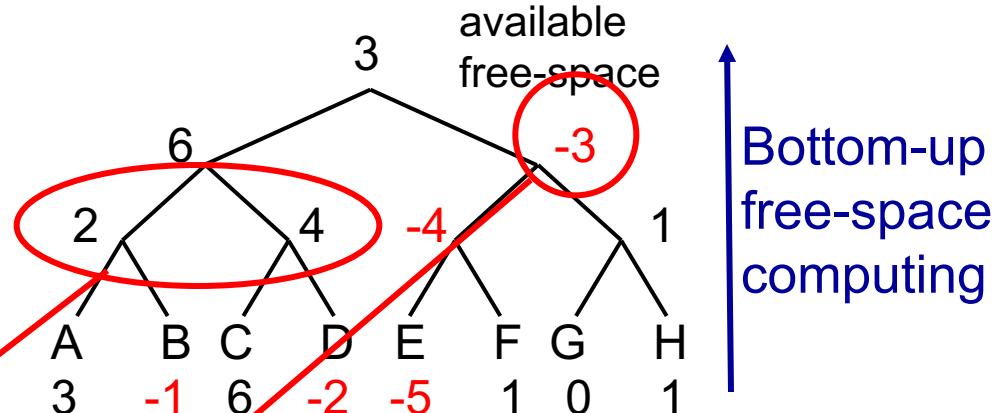


Free-Space Allocation for Density Control

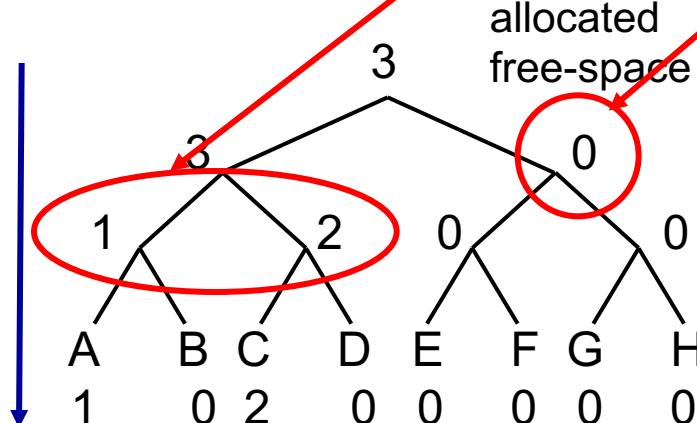
- Reallocate free-space to the density overflow bins

density overflow bins

A 30	B 30	E 30	F 30
C 30	D 30	G 30	H 30



Top-down free-space allocation



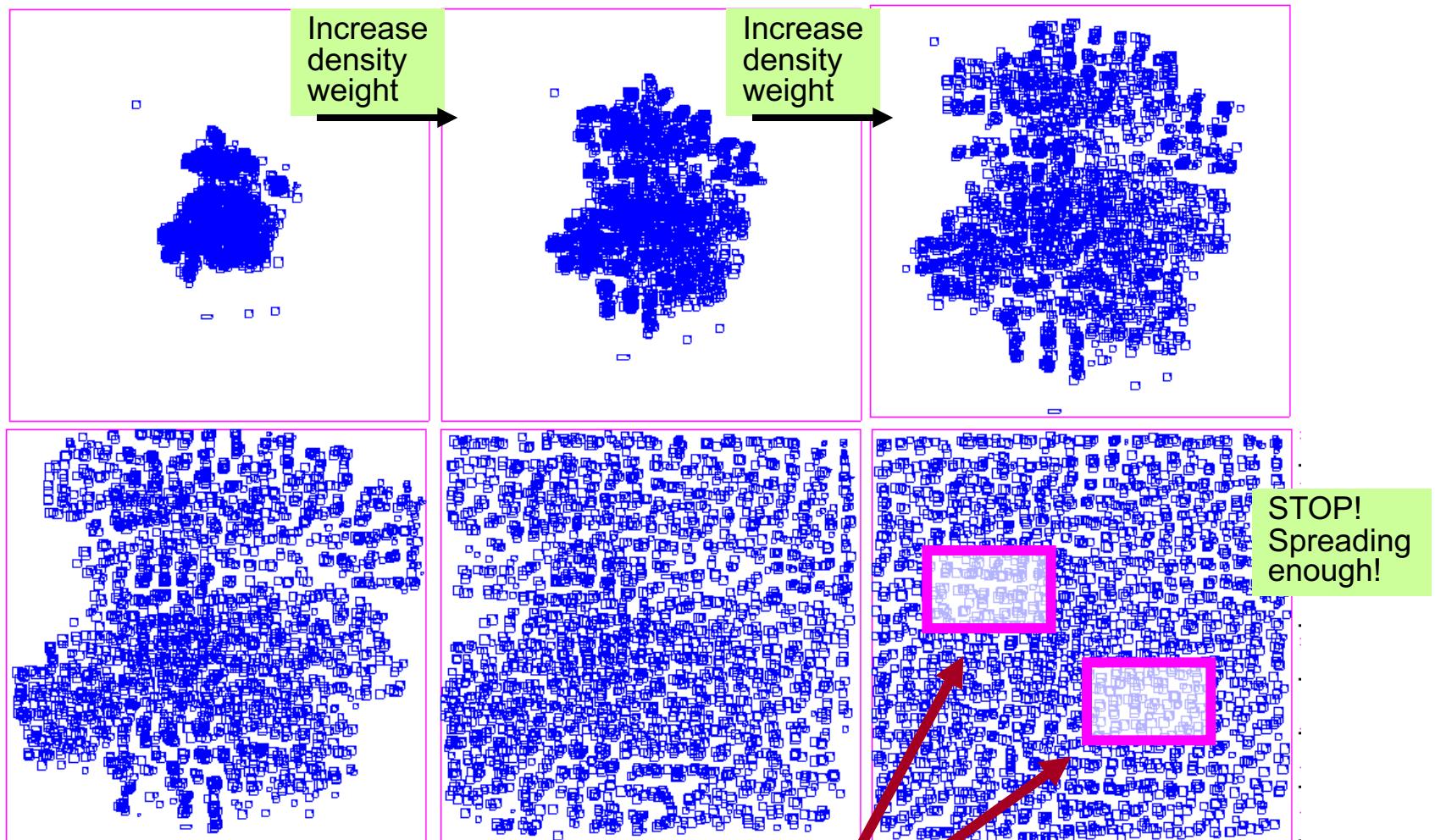
Example:

$$A'_E = A_E + 0 - (-5) = 35$$

A 28	B 31	E 35	F 29
C 26	D 32	G 30	H 29

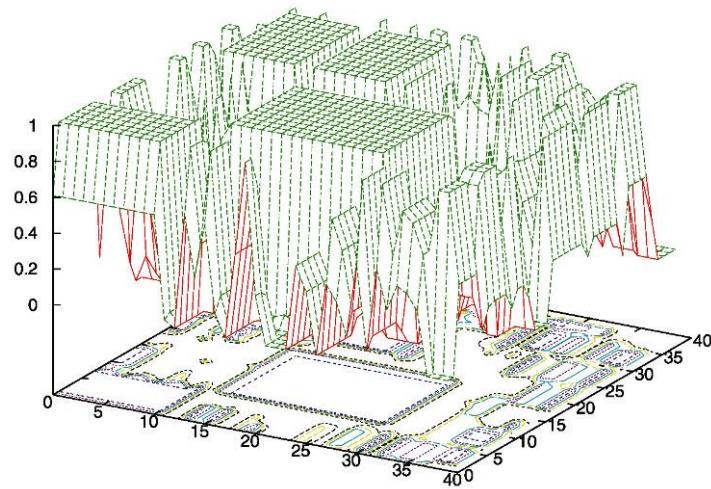
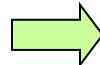
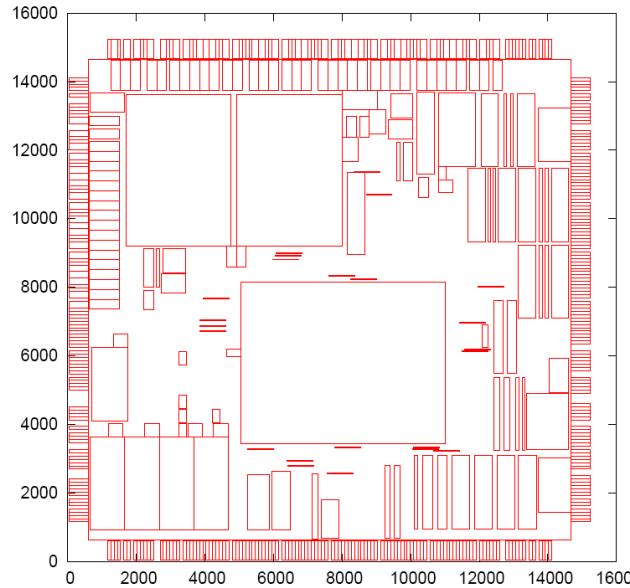
Determine cut-lines according to the allocated free-space

Placement Process



- How to handle preplaced blocks?
 - Pre-defined density makes cell spreading harder.

Density Map with Preplaced Blocks



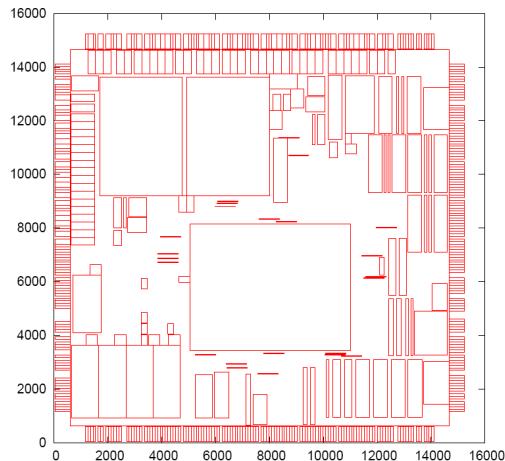
**Placement with
preplaced blocks**

Two problems

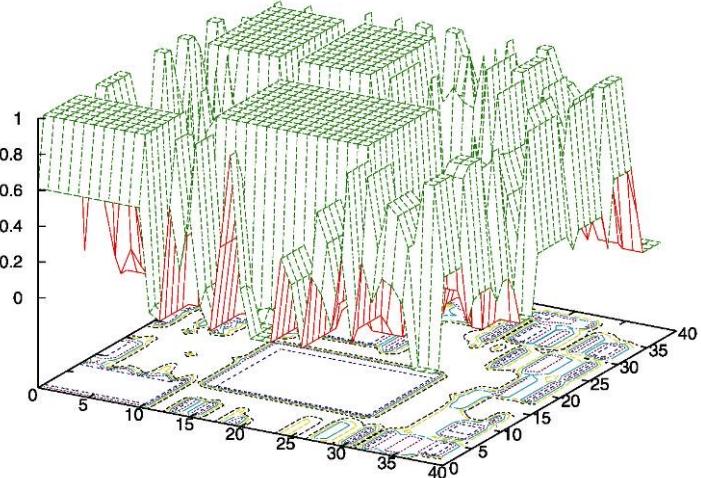
- ⌚ the density map is not smooth
- ⌚ the density is too high to spread movable blocks over the mountains

**Corresponding
density map**

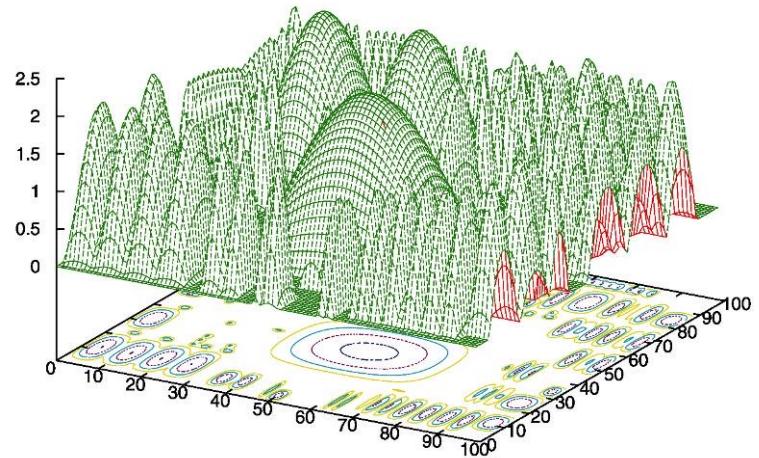
Bell-Shaped Block Smoothing??



- ⌚ mountain heights are quite different
- ⌚ there are many valleys among mountains
- ➔ might mis-guide the movement of blocks

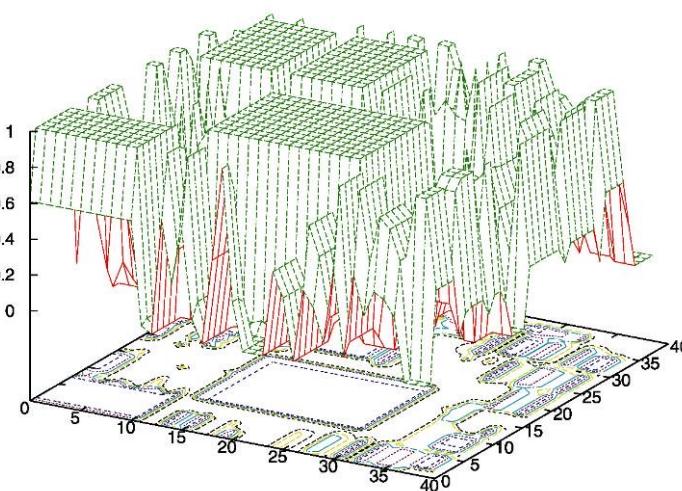
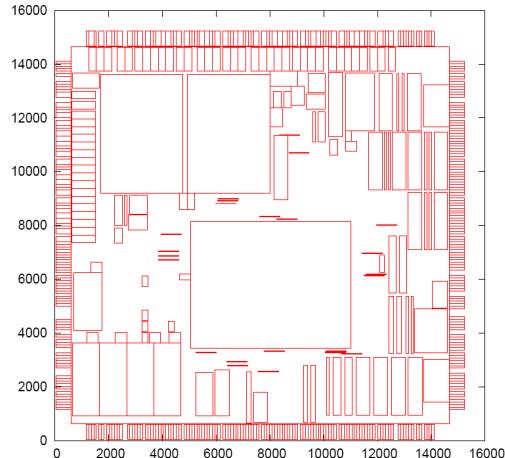


Apply the
bell-shaped
function



Bell-shaped smoothing
(Kahng et al.)

Preplaced Block Smoothing

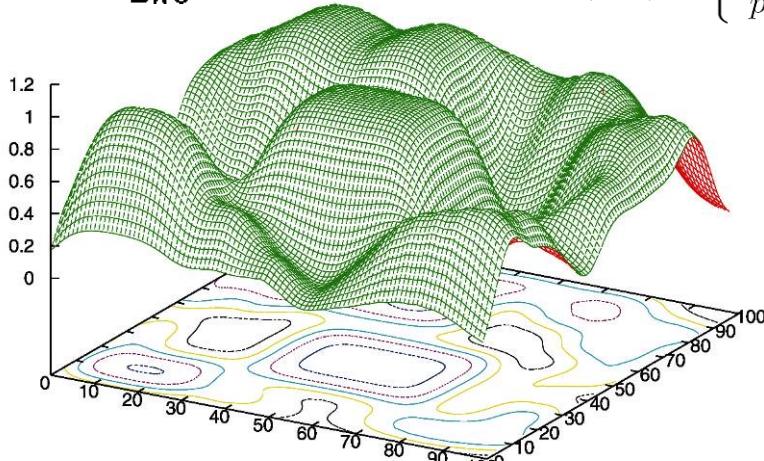


Convolute with Gaussian

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

無法微分 · 因此要平滑化

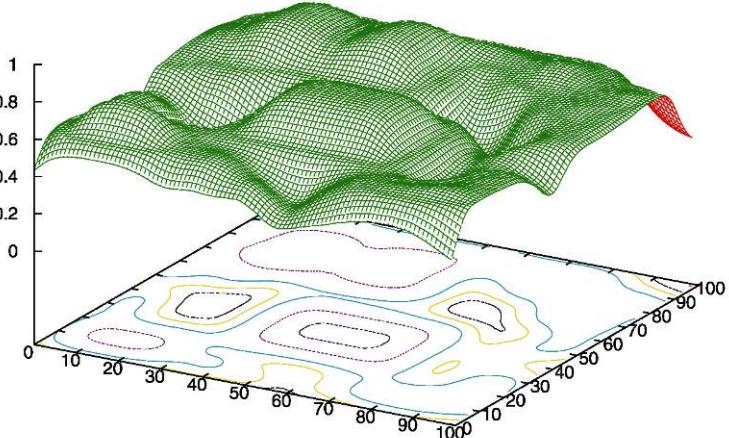
$$p'(x, y) = \begin{cases} \bar{p} + (p(x, y) - \bar{p})^\delta & \text{if } p(x, y) \geq \bar{p} \\ \bar{p} - (\bar{p} - p(x, y))^\delta & \text{if } p(x, y) \leq \bar{p} \end{cases}$$



Gaussian smoothing



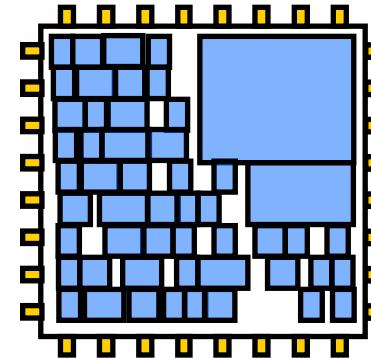
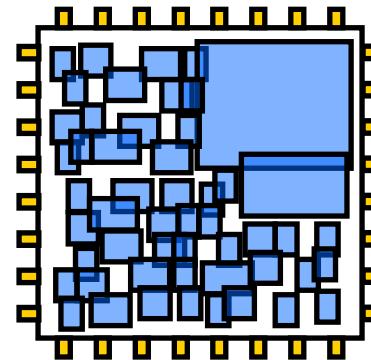
Level
smoothing



Gaussian + Level smoothing

Legalization

- Placement flow
 - Global placement
 - **Legalization**
 - **Mixed-size legalization**
 - **Look-ahead legalization**
 - Detailed placement



Legalization

- Simplified Tetris algorithm
[Hill, US Patent 6370673, 2002]
 - Sort modules based on x-coordinate
 - Search the current nearest available position for each module
- Pros and cons
 - ☺ fast
 - ☺ larger blocks are fixed at a position ahead of nearby small cells
 - ☹ best applied when modules are distributed evenly
 - ☹ may fail if the global placement has many overlaps among macros

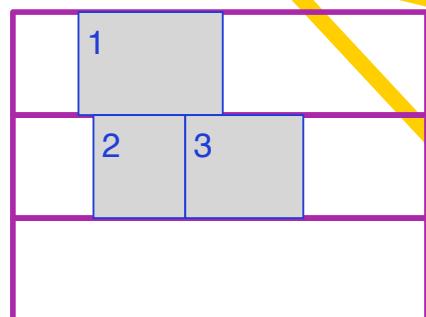
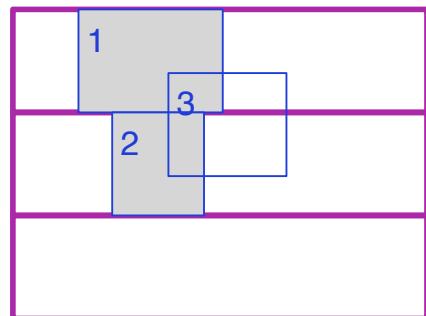
Tetris Algorithm

- First, sort modules in ascending x-coordinate
- Then pack modules to the left one at a time into the row that minimizes the total displacement for that module
- Commonly used and extremely fast
- But may result in very uneven row length and may fail to pack all modules inside placement region

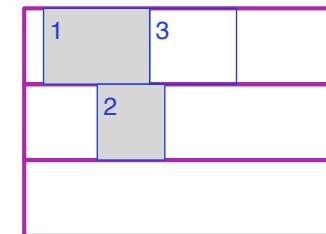
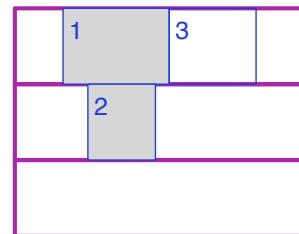
轉90度的俄羅斯方塊
(右邊丟到左邊)

Abacus (ISPD'08)

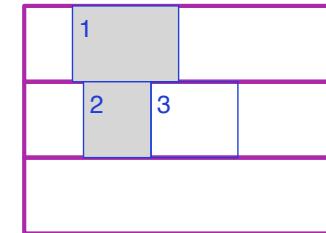
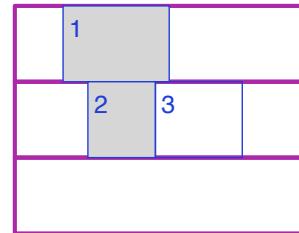
- The objective is to minimize displacement.
- Cells are sorted according to their x coords of global placement, and then cells are legalized one at a time.
- The legalized cell is inserted into different rows by **PlaceRow()**, and the cost is stored.
- After the cell is moved over the rows, it is inserted to the best row.
- After all cells are inserted to their best row, the legalization is finished.



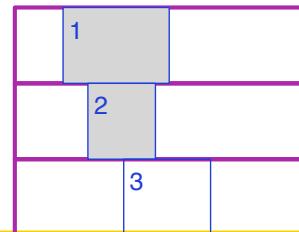
Insert cell 3 into 1st row



Insert cell 3 into 2nd row



Insert cell 3 into 3rd row

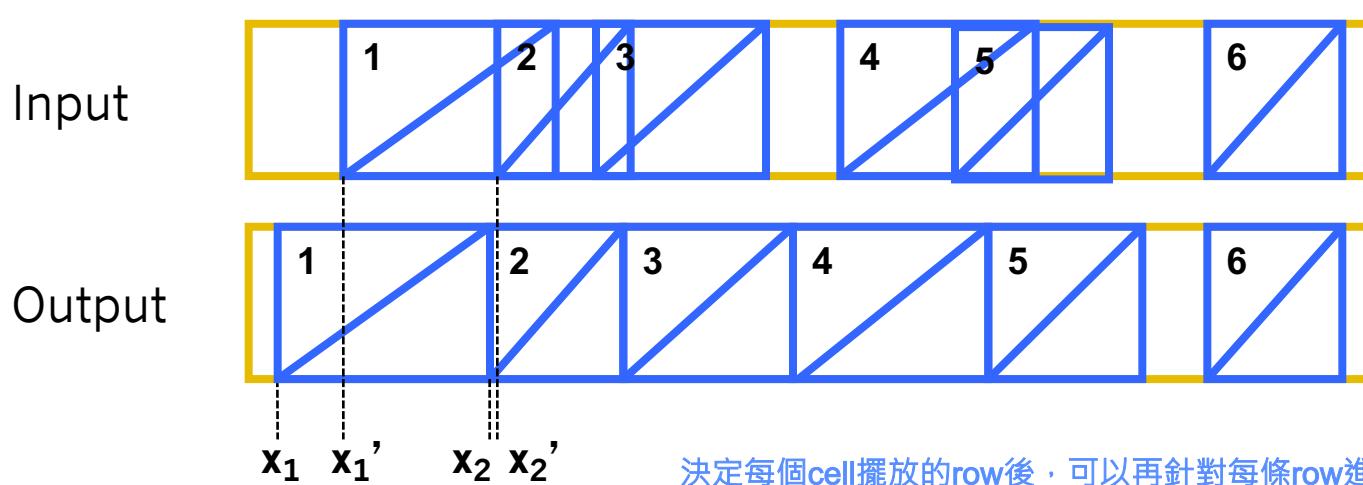


Legalization Approach

Algorithm 1: Our legalization approach “Abacus”

```
1 Sort cells according to x-position;  
2 foreach cell i do  
3    $c_{best} \leftarrow \infty$ ;  
4   foreach row r do  
5     Insert cell i into row r;  
6     PlaceRow r (trial);  
7     Determine cost c;  
8     if c <  $c_{best}$  then  $c_{best} = c$ ,  $r_{best} = r$ ;  
9     Remove cell i from row r;  
10    end  
11    Insert Cell i to row  $r_{best}$ ;  
12    PlaceRow  $r_{best}$  (final);  
13 end
```

Optimizing the Total Movement: PlaceRow()



$$\min \sum_{i=1}^{N_r} e(i) [x(i) - x'(i)]^2 \quad (1)$$

$$\text{s.t. } x(i) - x(i-1) \geq w(i-1) \quad i = 2, \dots, N_r \quad (2)$$

Given a cell ordering, legalization can be formulated as QP, and Abacus uses DP to speedup

PlaceRow() by DP

Objective is to minimize total displacement:

$$\min \sum_{i=1}^{N_r} e(i) [x(i) - x'(i)]^2$$

Property	Explanation
$x'(i), y'(i)$	Position in global placement
$x(i), y(i)$	Position in legal placement
$w(i)$	Width
$e(i)$	Weight (e.g., number of pins)

When all cells of a row are abutting, the minimum of objective is obtained by setting its derivative with respect to $x(1)$ to zero, which gives:

$$x(i) = x(1) + \sum_{k=1}^{i-1} w(k) \quad i = 2, \dots, N_r \quad \dots \dots \dots \text{Cells are abutting in a row}$$

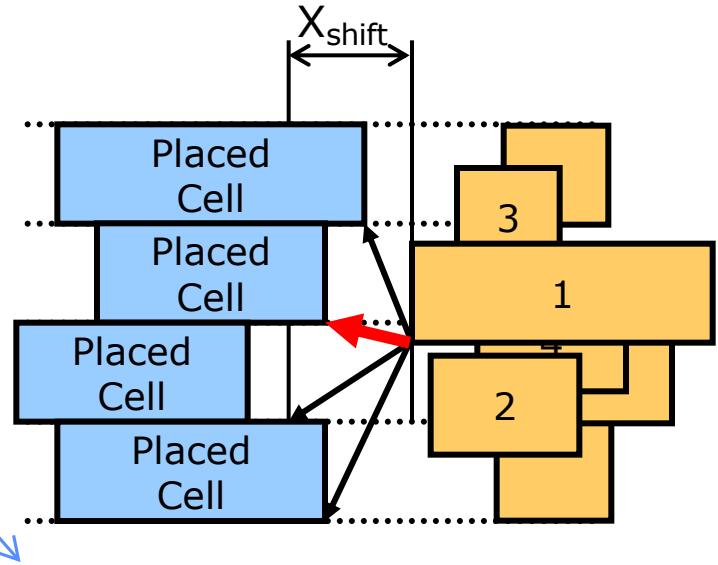
$$\underbrace{\sum_{i=1}^{N_r} e(i)}_{e_c} x(1) - \underbrace{\left[e(1)x'(1) + \sum_{i=2}^{N_r} e(i) \left[x'(i) - \sum_{k=1}^{i-1} w(k) \right] \right]}_{q_c} = 0 \quad \dots \dots \dots \text{Set derivative of objective to zero}$$

The global position of the cluster is simply solved:

$$x(1)e_c - q_c = 0 \quad \Leftrightarrow \quad x(1) = \frac{q_c}{e_c}$$

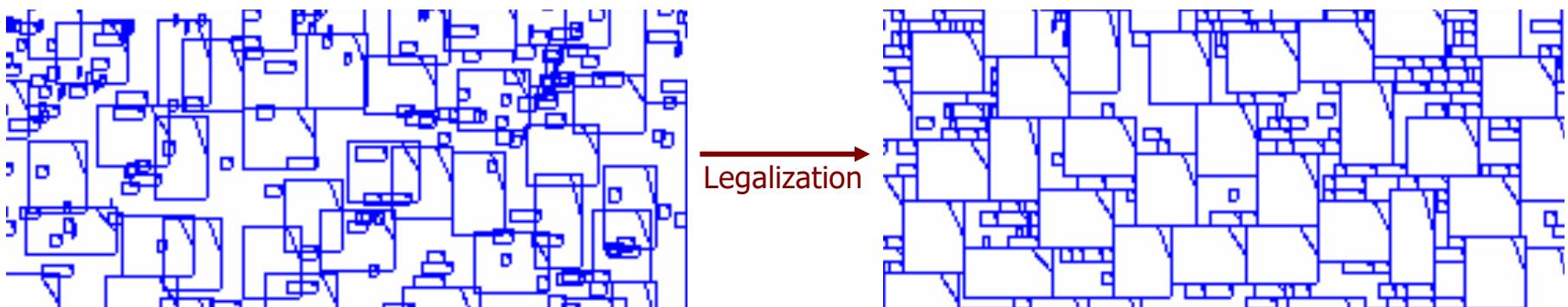
Mixed-Size Legalization

- Determine the block legalization sequence by the **x coordinate** and the **block size**
 - Priority = $k_1 x_i + k_2 w_i + k_3 h_i$
 - x_i : the **x coordinate** of block i
 - $w_i(h_i)$: the width (height) of the block i
 - Large blocks are legalized earlier



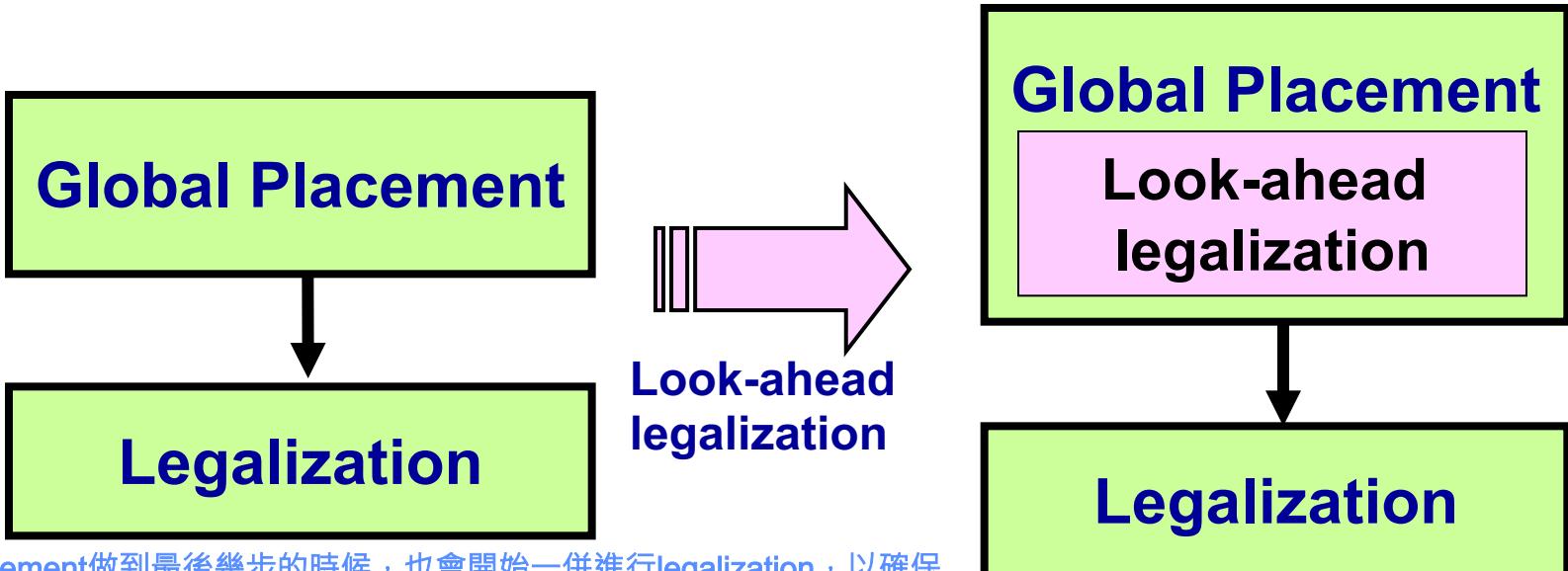
最基本的就是只考慮x座標進行擺放，但是如果每個cell的大小不太一樣，就也要將他們的寬、高也納入考量

- Select the position with the smallest wirelength within the x_{shift} range
大的cell先擺放，小的cell後擺放較容易成功



Look-ahead Legalization

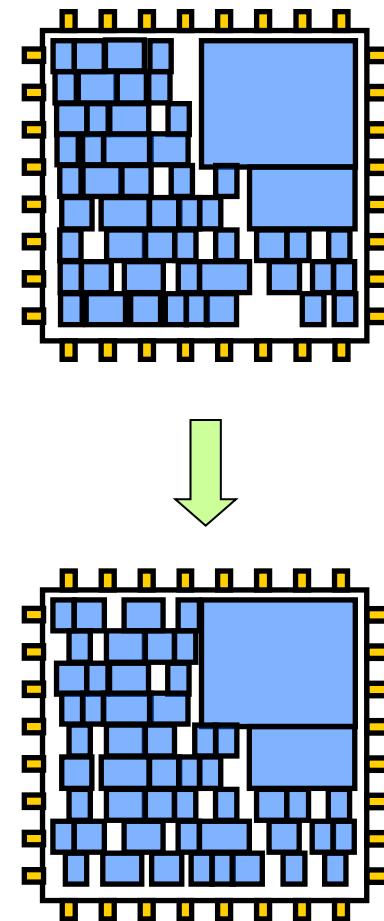
- Blocks not spread enough or spread too much lead to poor legalization results
- Incorporate look-ahead legalization into global placement
 - Perform legalization several times until blocks are spread enough
 - Reduce the gap between global placement and legalization



placement做到最後幾步的時候，也會開始一併進行legalization，以確保該次global placement是可以被legalize的，如果發現最後幾步的global placement是無法被legalize的話，就再往回進行global placement

Detailed Placement

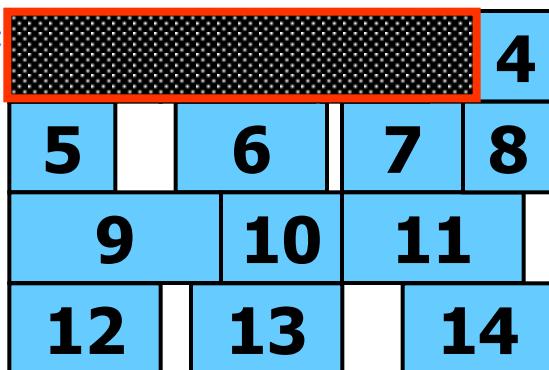
- Placement flow
 - Global placement
 - Legalization
 - ***Detailed placement***
 - ***Cell matching for wirelength minimization***
 - ***Cell sliding for density optimization***



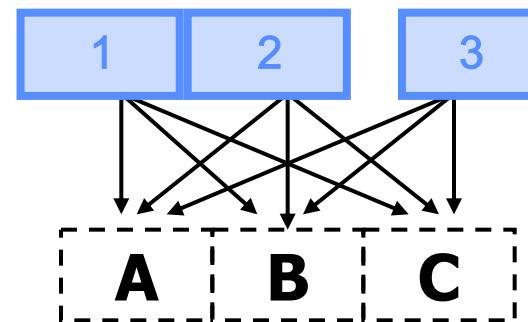
Cell Matching

- For wirelength minimization
- Steps
 1. Select a window
 2. Select blocks from the window
 3. Create a bipartite matching problem (edge weight = wirelength)
 4. Find the minimum weighted matching to optimize the wirelength
 5. Update block positions
- Handle **200-300** cells at one time
 - Compared to branch-and-bound which can handle only 6 cells at one time due to its high time complexity.

紅色的框框就是
window



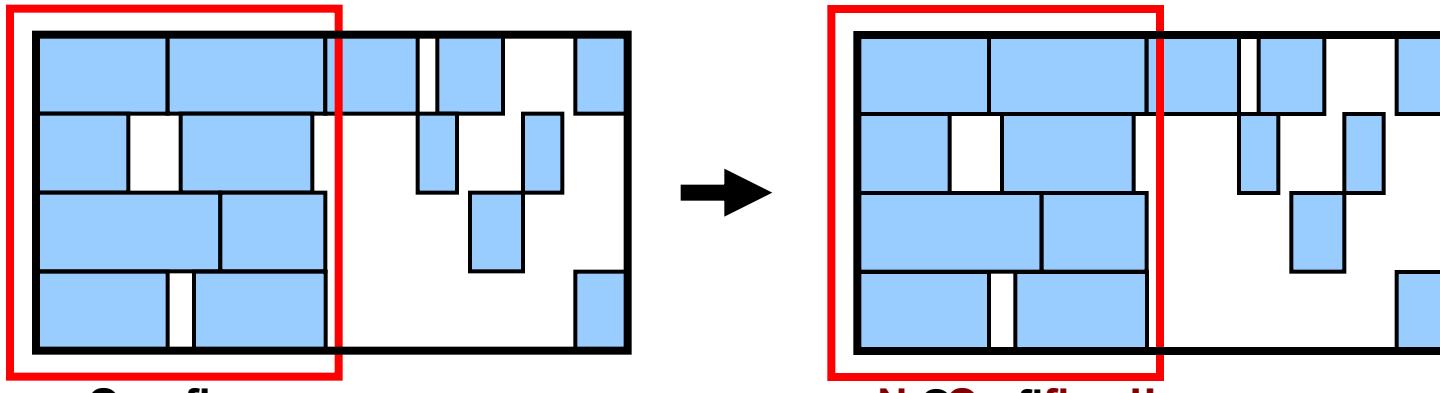
window內的
block拿出來做
bipartite
matching



Assign cells {1,2,3} to locations {A,B,C}

Cell Sliding

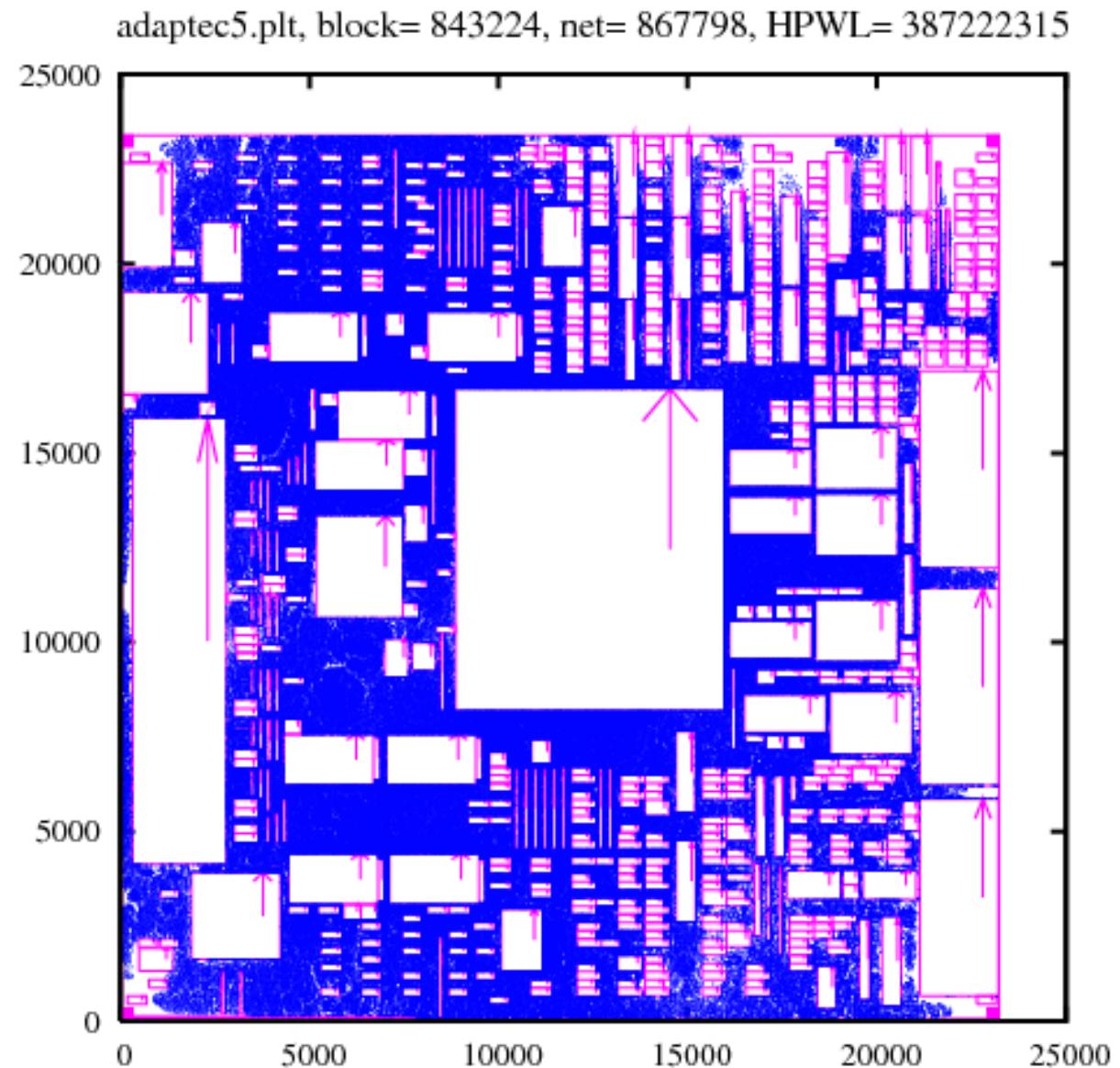
- For density optimization
- Steps
 1. Select an overflow window
 2. Calculate the amount of area to shift
 3. Move cells left/right to reduce the density
- Is fast and effective



將此window中的某些block向右滑動，讓出空間才能順利routing
(這張圖沒有畫出來)

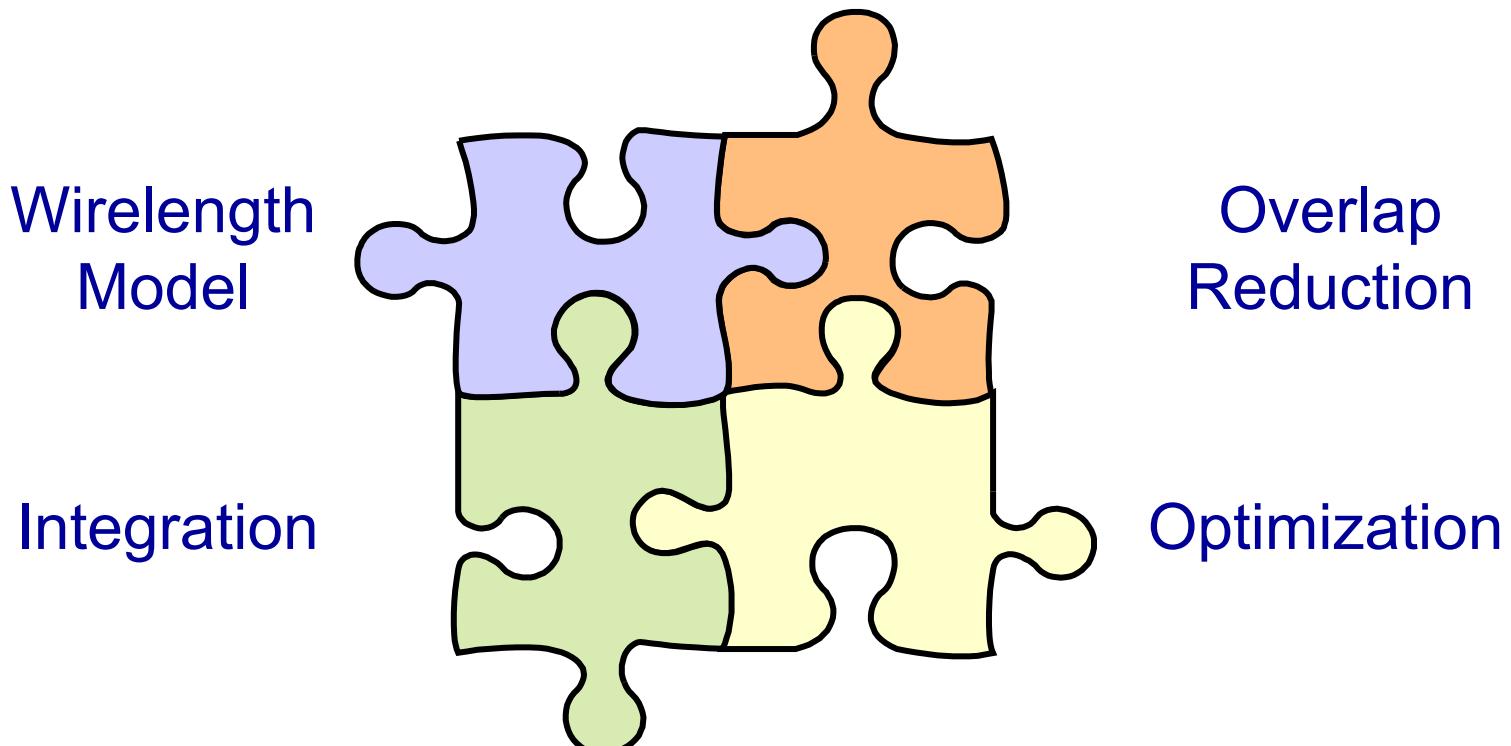
Demo (adaptec5)

- GP: 5 levels
- #Movable obj.
= 842K
- #Fixed obj.
= 646
- #Nets = 868K



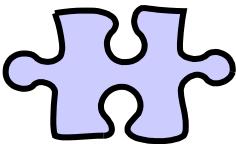
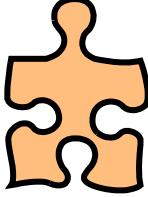
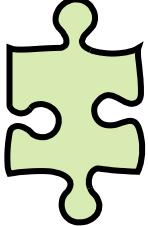
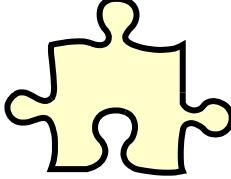
Essential Issues in Analytical Placement

- Chang, Jiang & Chen, “Essential issues in analytical placement algorithms,” IPSJ Trans. System LSI Design Methodology, August 2009

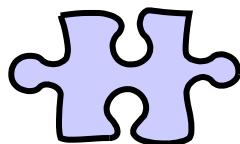


Slides: courtesy T.-C. Chen

NTPlace3 Example

心法	招式
	Wirelength Model: Log-Sum-Exp (LSE) function
	Overlap Reduction: Bell-shaped density model
	Integration: Penalty method
	Optimization: Nonlinear

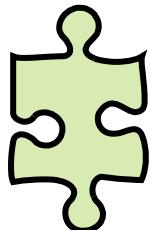
GODIAN Example



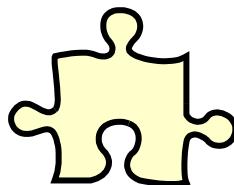
Wirelength Model: Quadratic function



Overlap Reduction: Partitioning

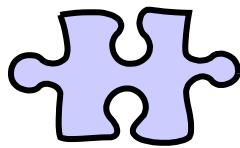


Integration: Region Constraint

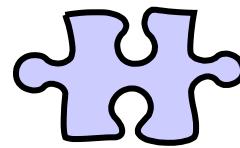


Optimization: Quadratic

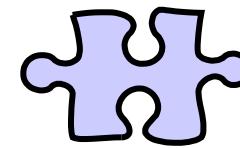
Other Combinations?



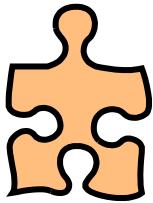
Quadratic
Wirelength
function



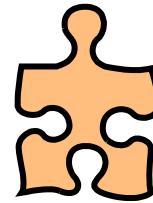
LSE
function



???



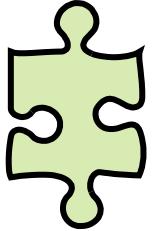
Partitioning



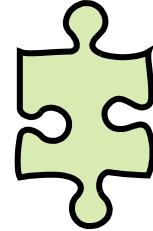
Bell-shaped
density model



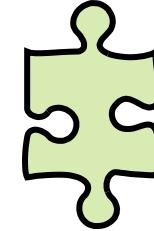
???



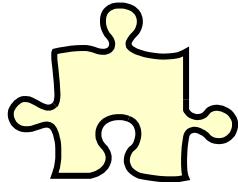
Region
constraint



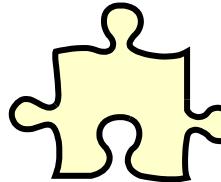
Penalty
method



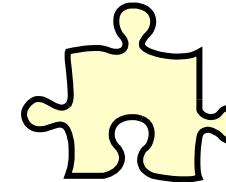
???



Quadratic

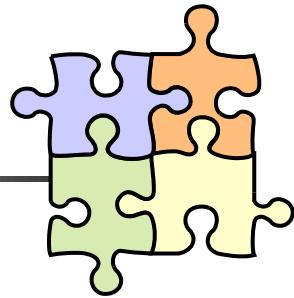


Nonlinear



???

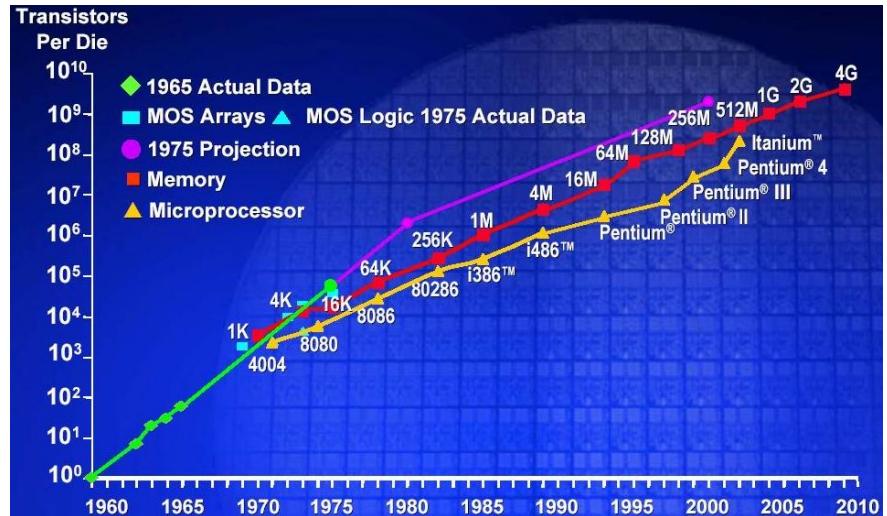
Modern Academic Analytical Placers



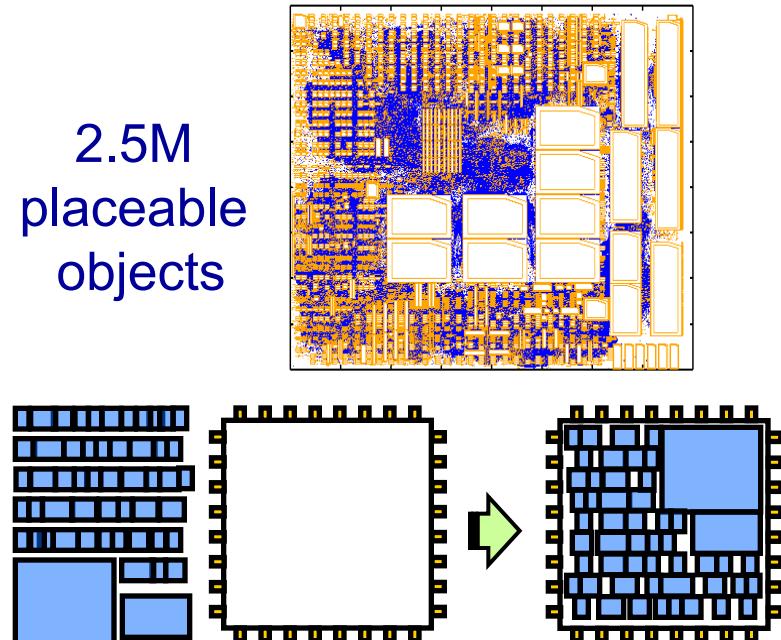
Placer	Wirelength Model	Overlap Reduction	Integration	Optimization
APlace	LSE	Density	Penalty Method	Nonlinear
BonnPlace	Quadratic	Partitioning	Region Constraint	Quadratic
DPlace	Quadratic	Diffusion	Fixed Point	Quadratic
FastPlace	Quadratic	Cell Shifting	Fixed Point	Quadratic
FDP	Quadratic	Density	Fixed Point	Quadratic
Gordian	Quadratic	Partitioning	Region Constraint	Quadratic
hATP	Quadratic	Partitioning	Region Constraint	Quadratic
Kraftwerk2	Bound2Bound	Density	Fixed Point	Quadratic
mFAR	Quadratic	Density	Fixed Point	Quadratic
mPL6	LSE	Density	Penalty Method	Nonlinear
NTUplace3	LSE	Density	Penalty Method	Nonlinear
RQL	Quadratic	Cell Shifting	Fixed Point	Quadratic
Vassu	LSE	Assignment	Fixed Point	Nonlinear

Modern Placement Challenges Revisited

- High complexity
 - Millions of objects to be placed
- Mixed-size placement
 - Hundreds of movable macro blocks with millions of small standard cells
- Placement constraints
 - Preplaced blocks
 - Chip density
 - etc.

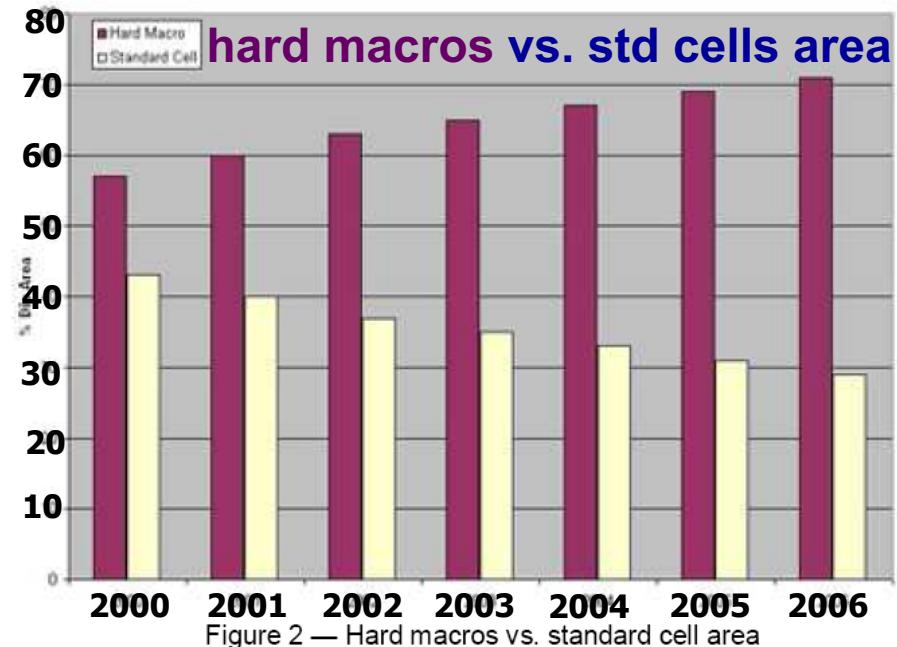
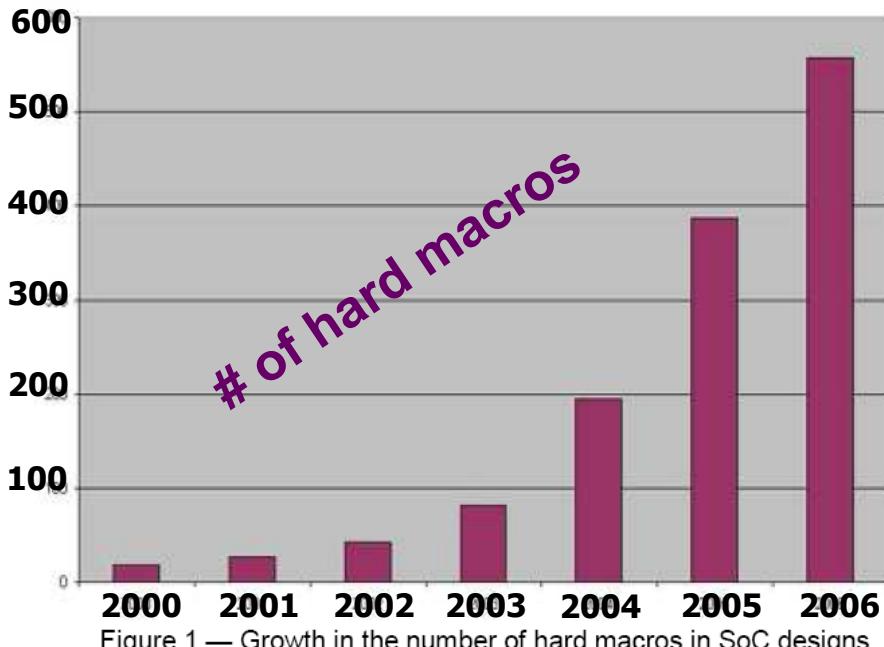


2.5M
placeable
objects



“Hard Macros Will Revolutionize SoC Design”

- Wein & Benkoski, *EE Times*, Aug. 20, 2004
- An SoC often contains hundreds of pre-designed macros
 - Embedded memories, analog circuits, IP blocks, etc.
- The number of macros grows dramatically.
- Existing layout tools are having problems with macros

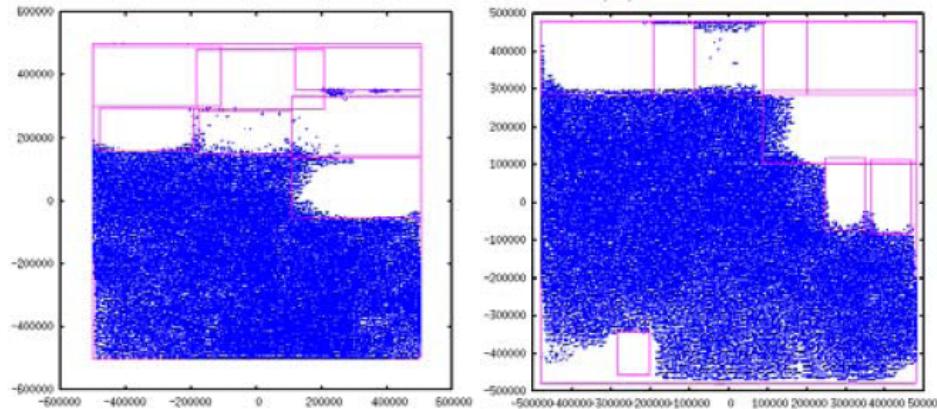


Existing Works on Mixed-Size Designs

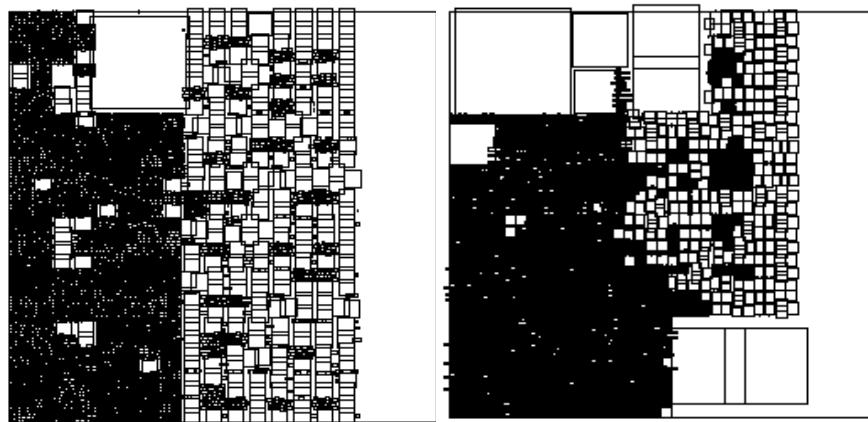
- Type 1: Simultaneous macro and cell placement
 - Allow macro overlaps during the placement process
 - Need a robust legalizer
 - Include most mixed-size placers
 - APlace, Dragon, FengShui, Kraftwerk, mPL, NTUplace3, etc.
- Type 2: Constructive macro placement
 - Keep macro overlap-free during the placement process
 - Include Capo, PATOMA, FLOP
 - It is harder to achieve a high-quality solution
- Type 3: Two-stage macro placement
 - Consist of (1) macro placement and then (2) cell placement
 - Is robust in finding legal placement
 - Is widely used in the industry
 - Include MP-tree, CG
 - **Need a good macro placer!!**

Problem with Simultaneous Macro and Cell Placement

- Is hard to guarantee overlap-free placement
- Placement generated by APlace 2.0



- Placement generated by FengShui

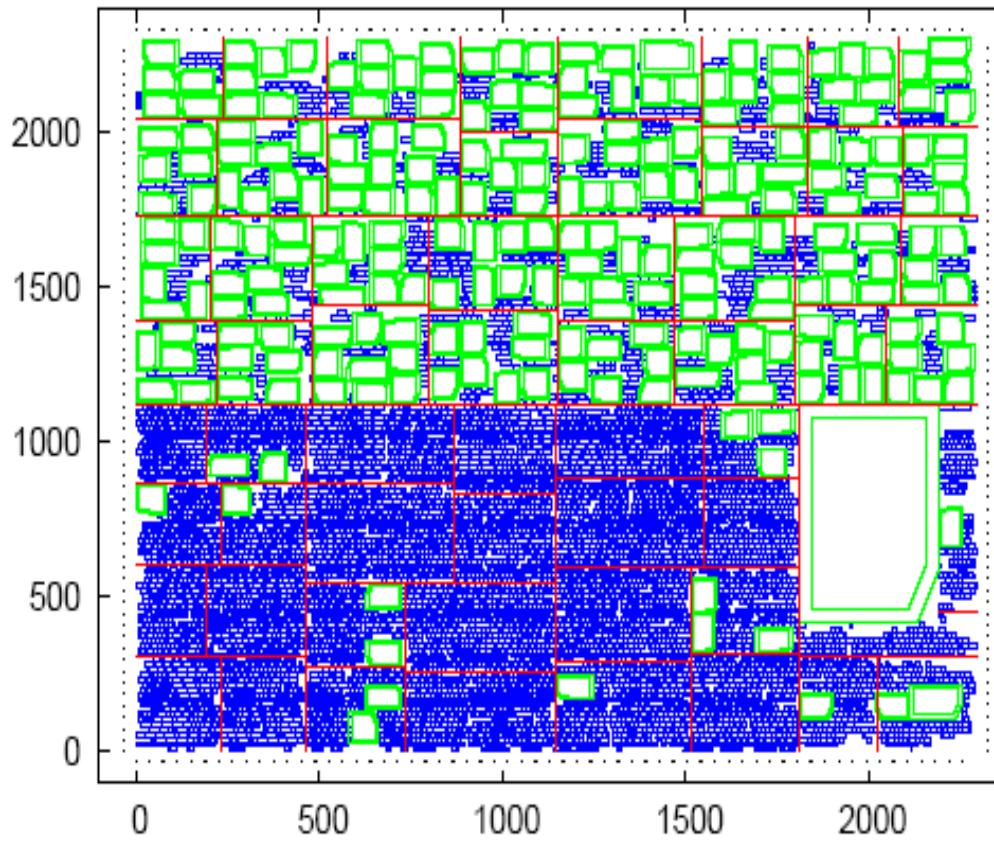


Courtesy of Prof. Y.-wv. Chang & H.-M. Chen

Constructive Macro Placement

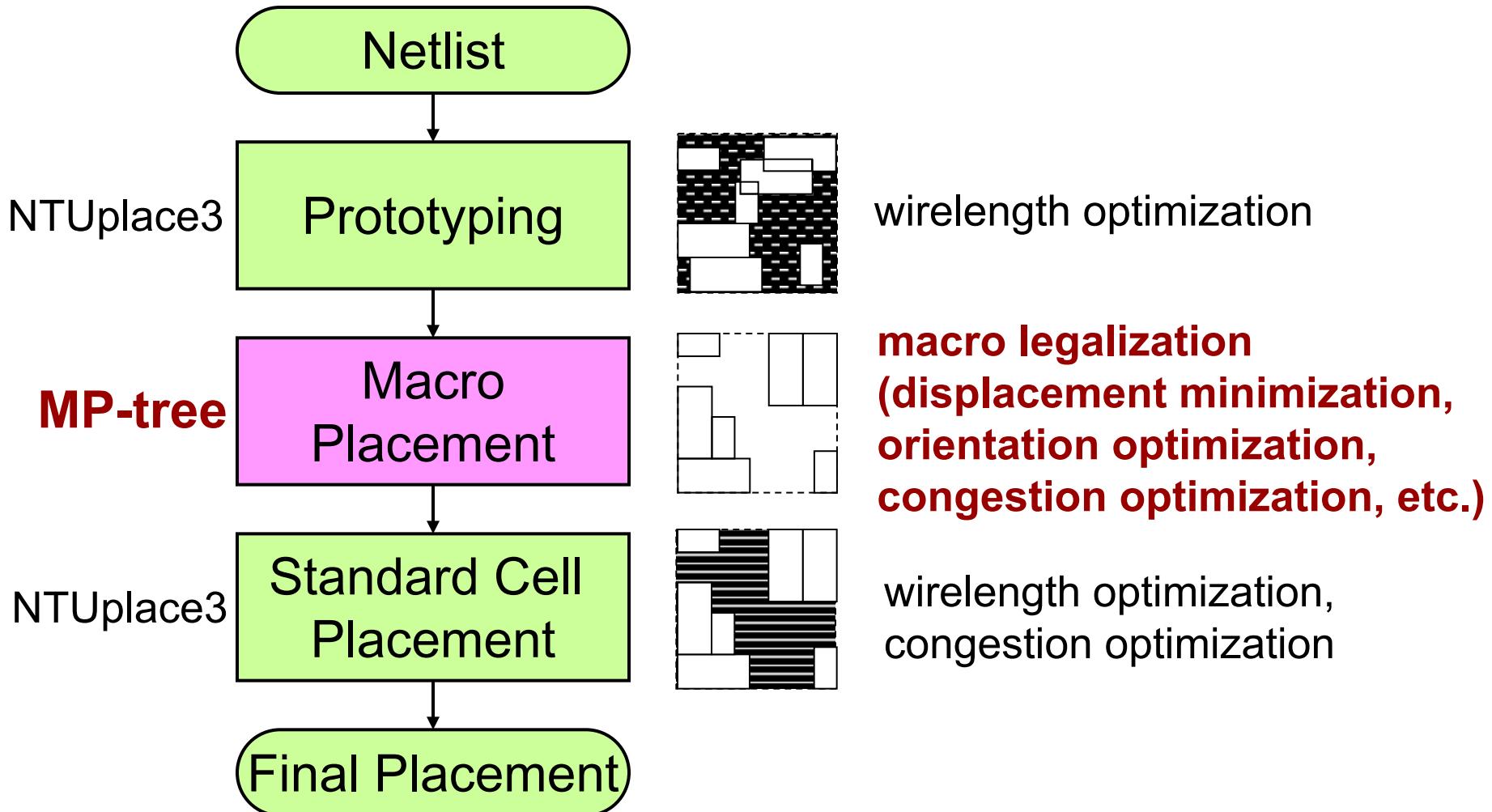
- Capo: recursive min-cut bisection
- Floorplanning is performed when necessary in the sub-region.

IBM01 HPWL= 2.491e+06, #Cells= 12752, #Nets= 14111



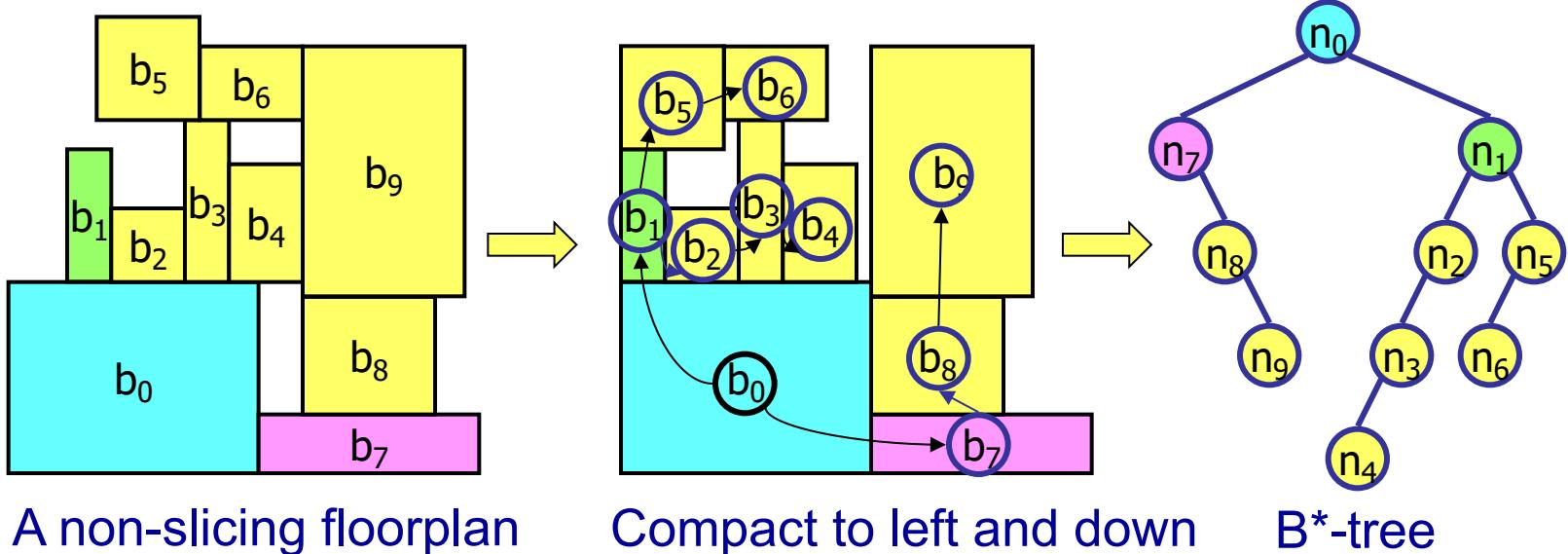
Our Mixed-Size Placement Flow

- Based on two-stage macro placement



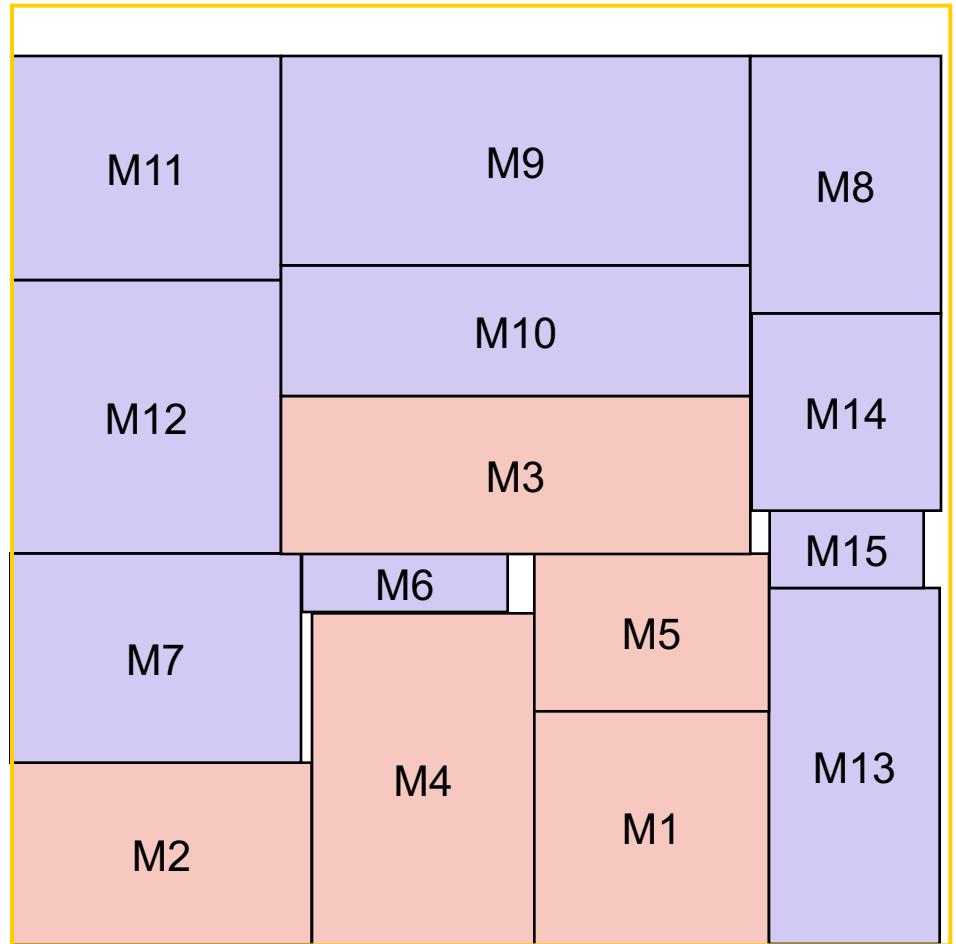
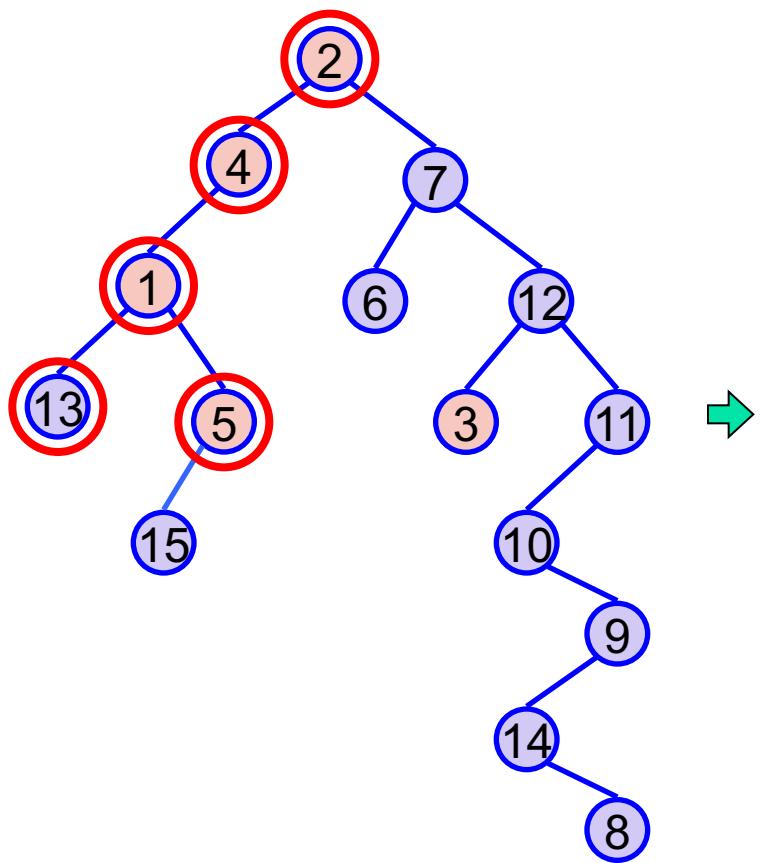
Macro Placement Using Multiple B*-Trees

1. Compact macros to left and bottom.
2. Construct an ordered binary tree.
 - Left child: the lowest, adjacent macro on the right ($x_j = x_i + w_i$).
 - Right child: the first macro above, with the same x-coordinate ($x_j = x_i$).

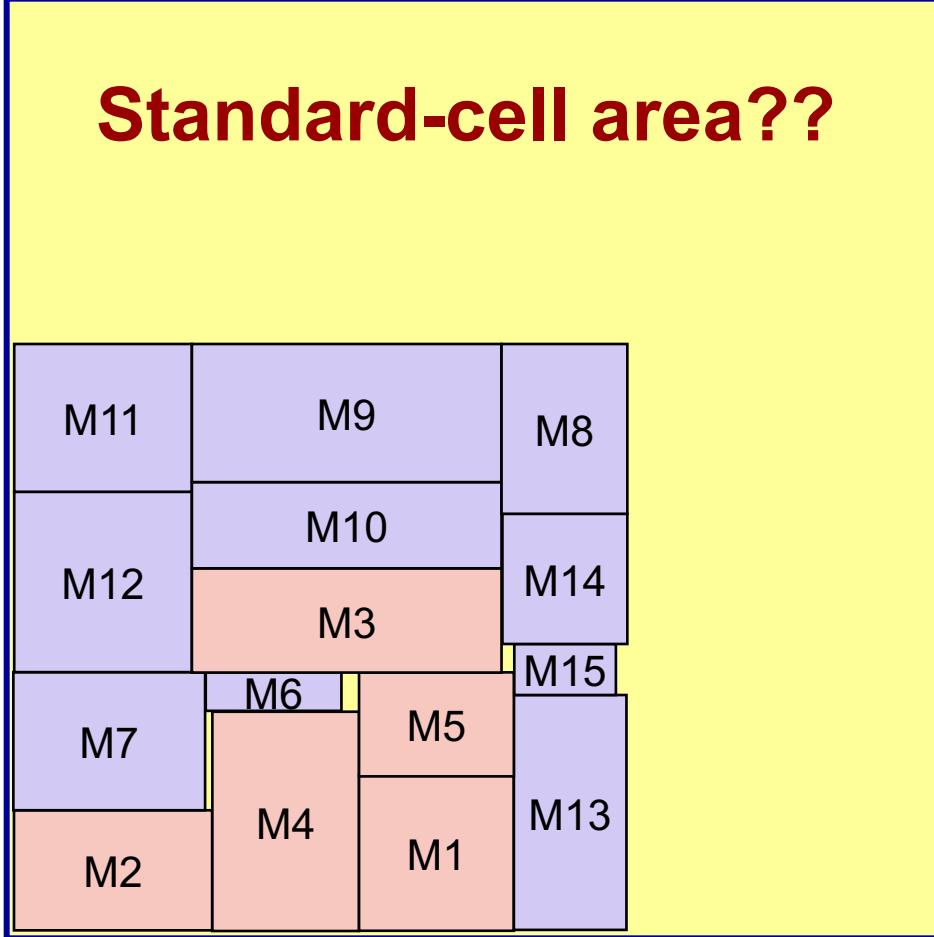


B*-Tree Packing Revisited

- Preorder traversal, complexity: $O(n)$



But What If %Macro Area Is Not High?

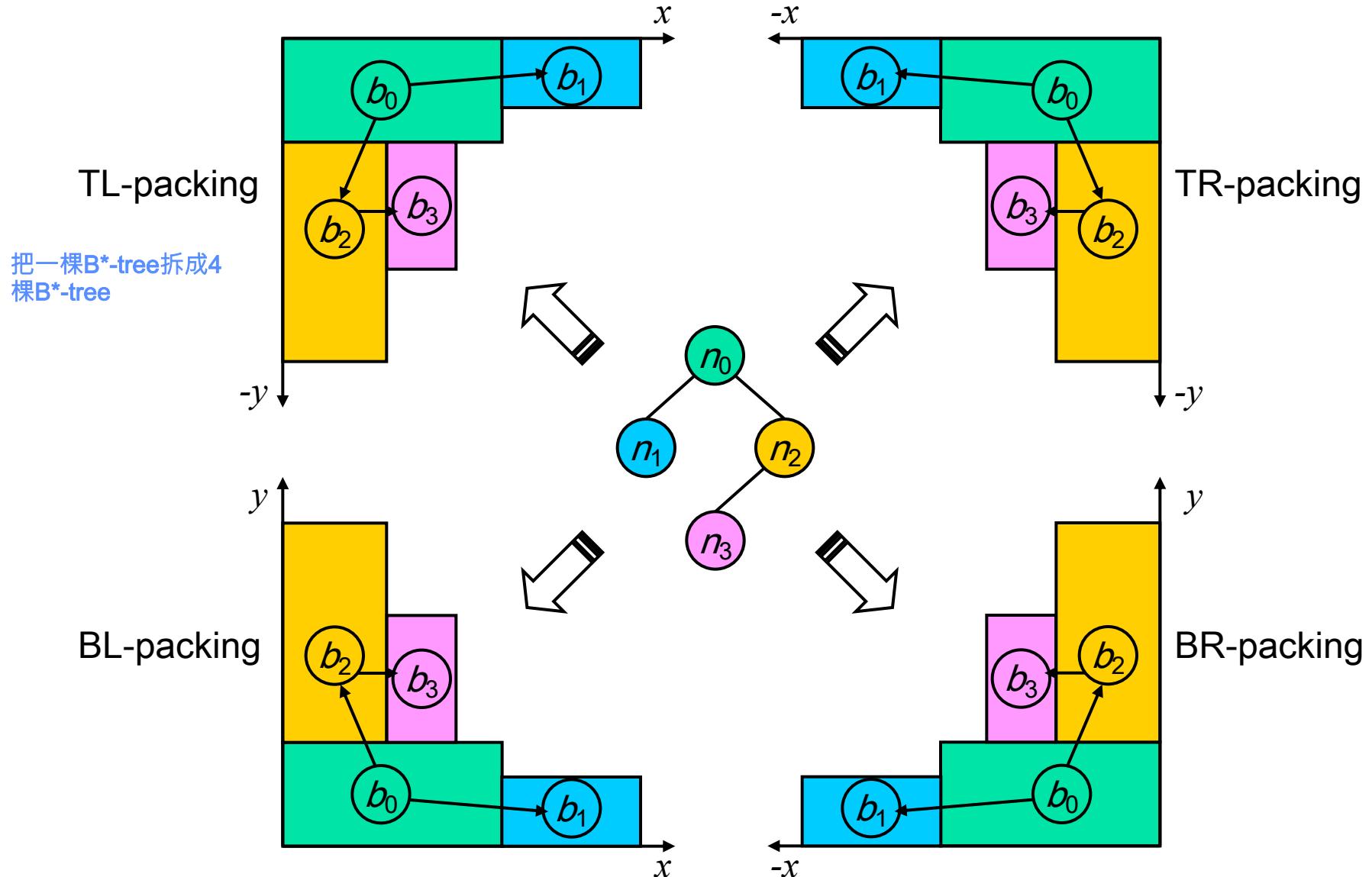


All macros will
be packed
together!!

Chip outline

把所有macro往左下角packing，這樣剩下能擺放standard cell的空間不會很理想

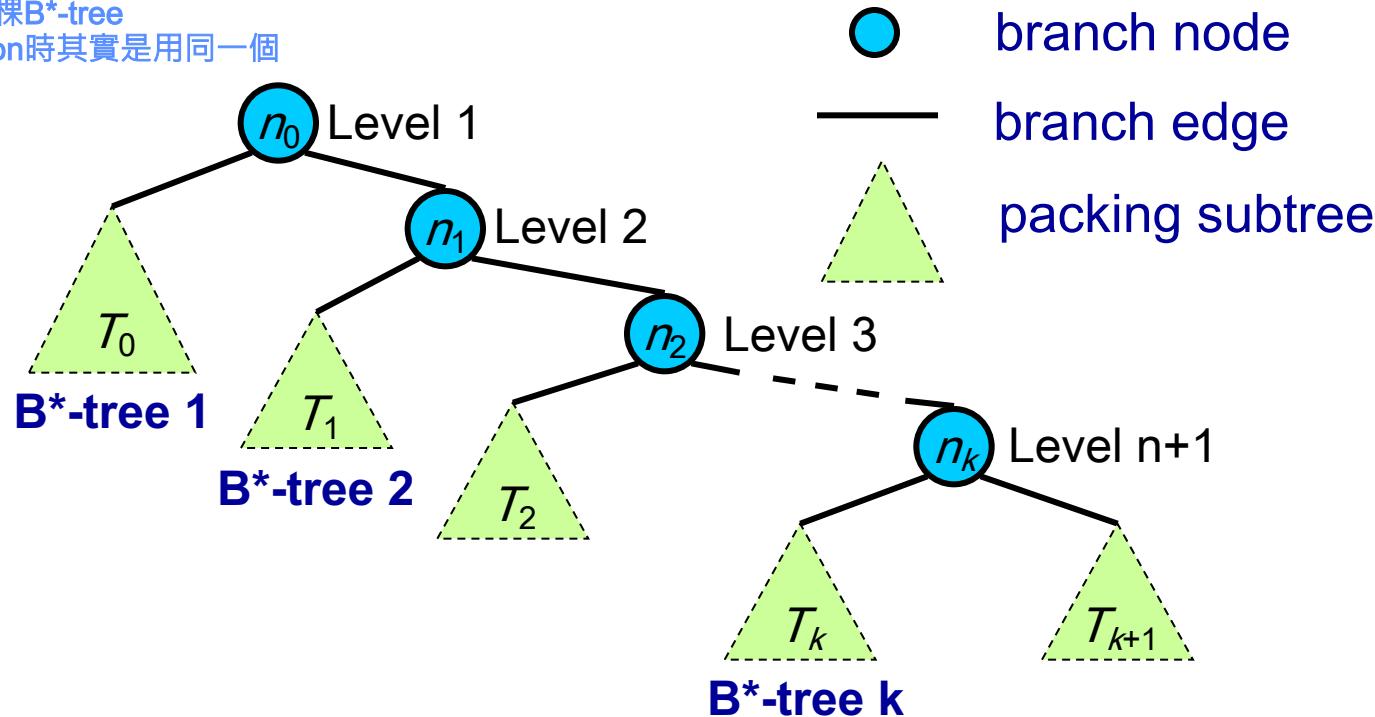
Multi-Packing (MP) Tree Representation



Generalized MP-Tree Representation

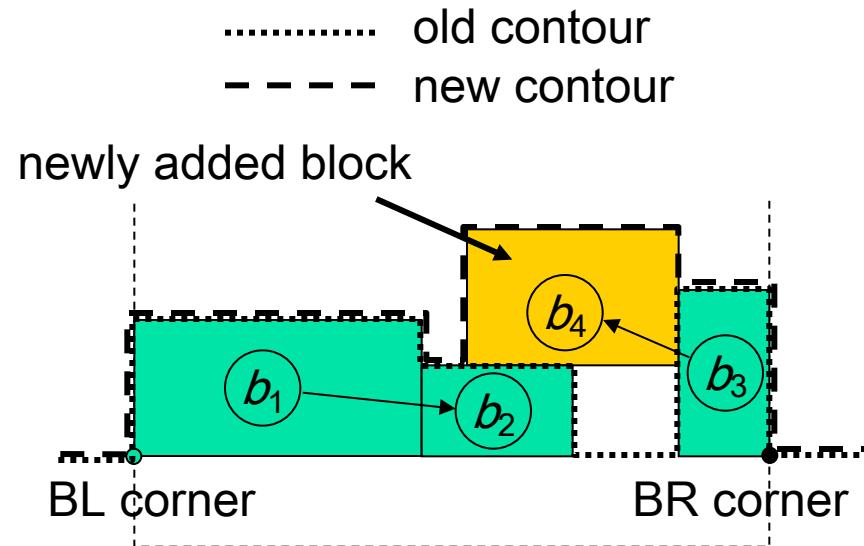
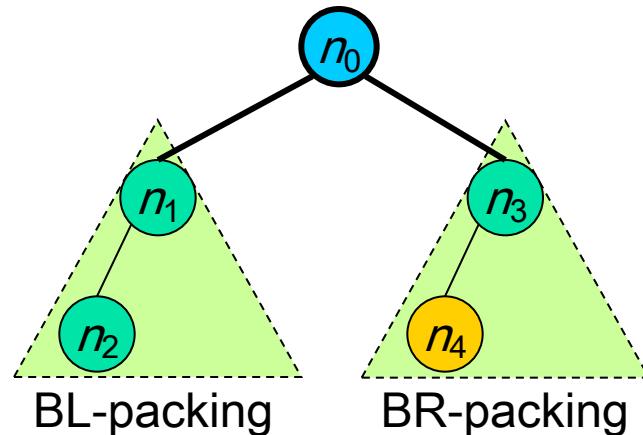
- Working on independent packing trees may not obtain desired solution
 - Lack global interactions among different subproblems
- Key: Combine several packing trees packing to different corners
- Use the right skewed branch for easier implementation

串聯拆開的那幾棵B*-tree
因此在perturbation時其實是用同一個
tree的結構



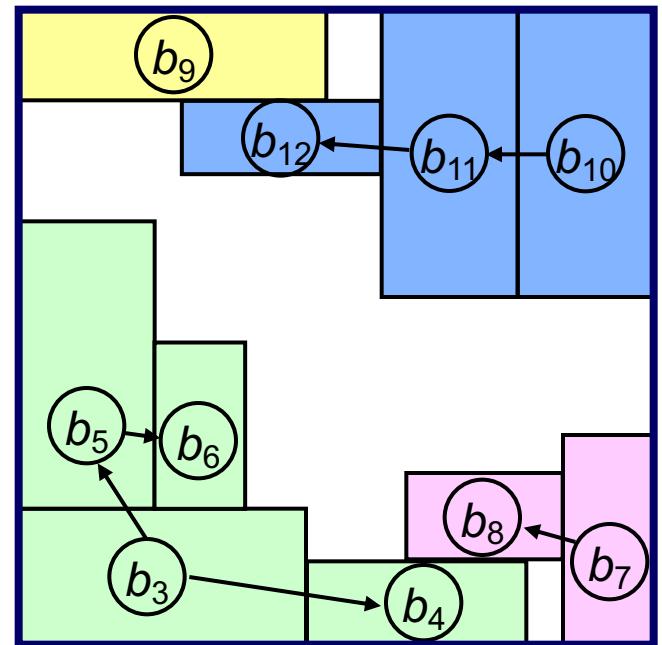
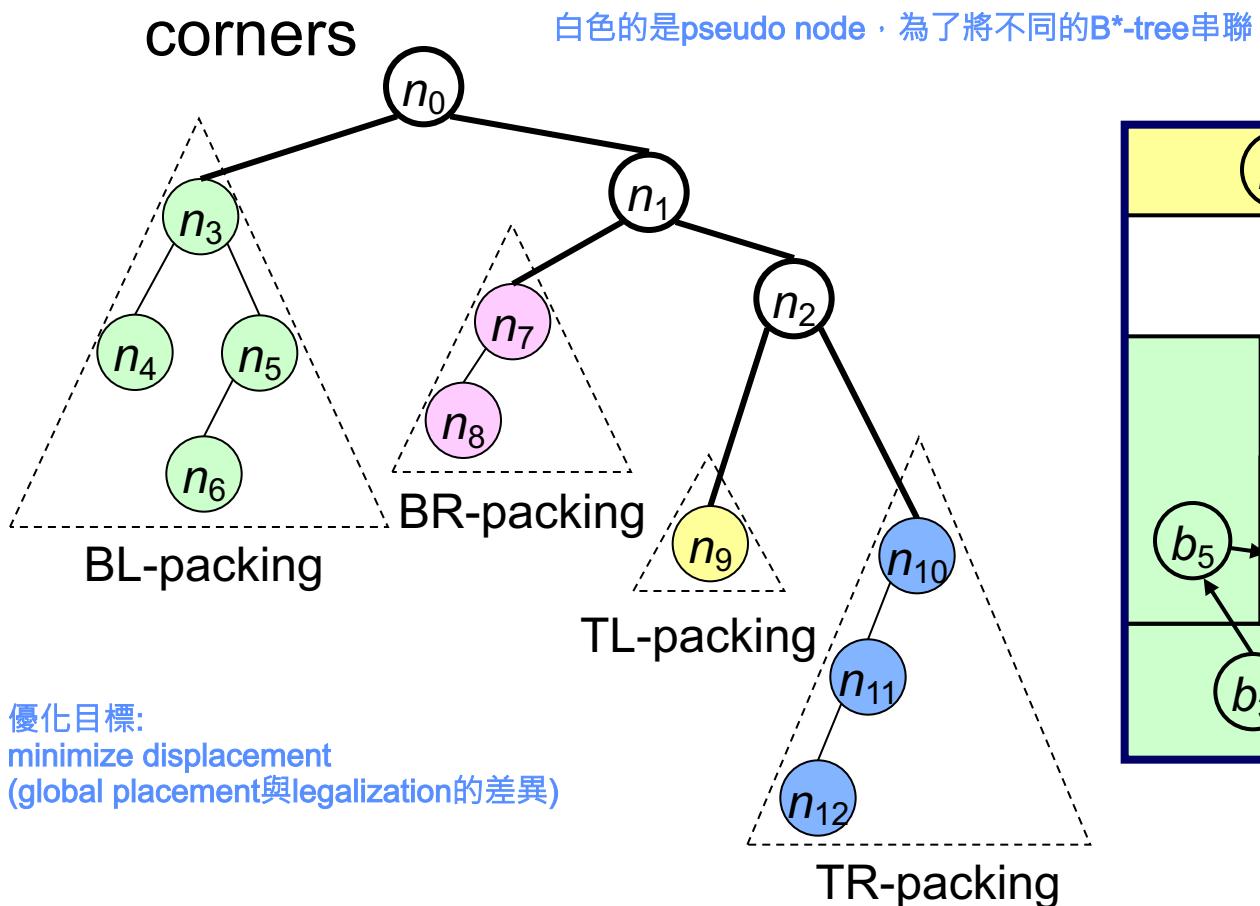
MP-tree Packing

- All BL- and BR-packing subtrees use the bottom-contour data structure
 - All TL- and TR-packing subtrees use the top contour data structure
 - An example of BL- and BR-packing



MP-tree Example

- Use four packing subtrees to handle a rectangular chip
- Applies to a floorplan region with an arbitrary number of corners

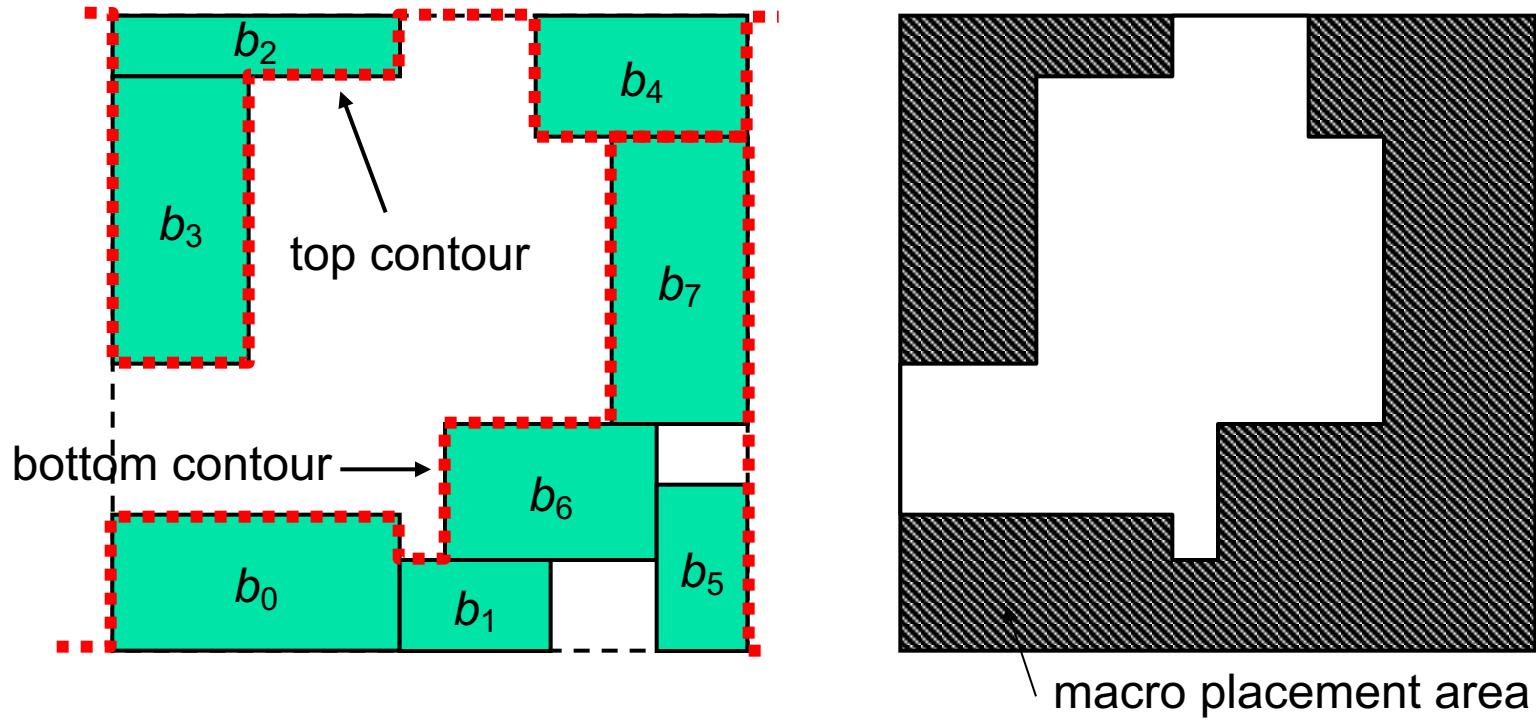


Operations on MP-tree

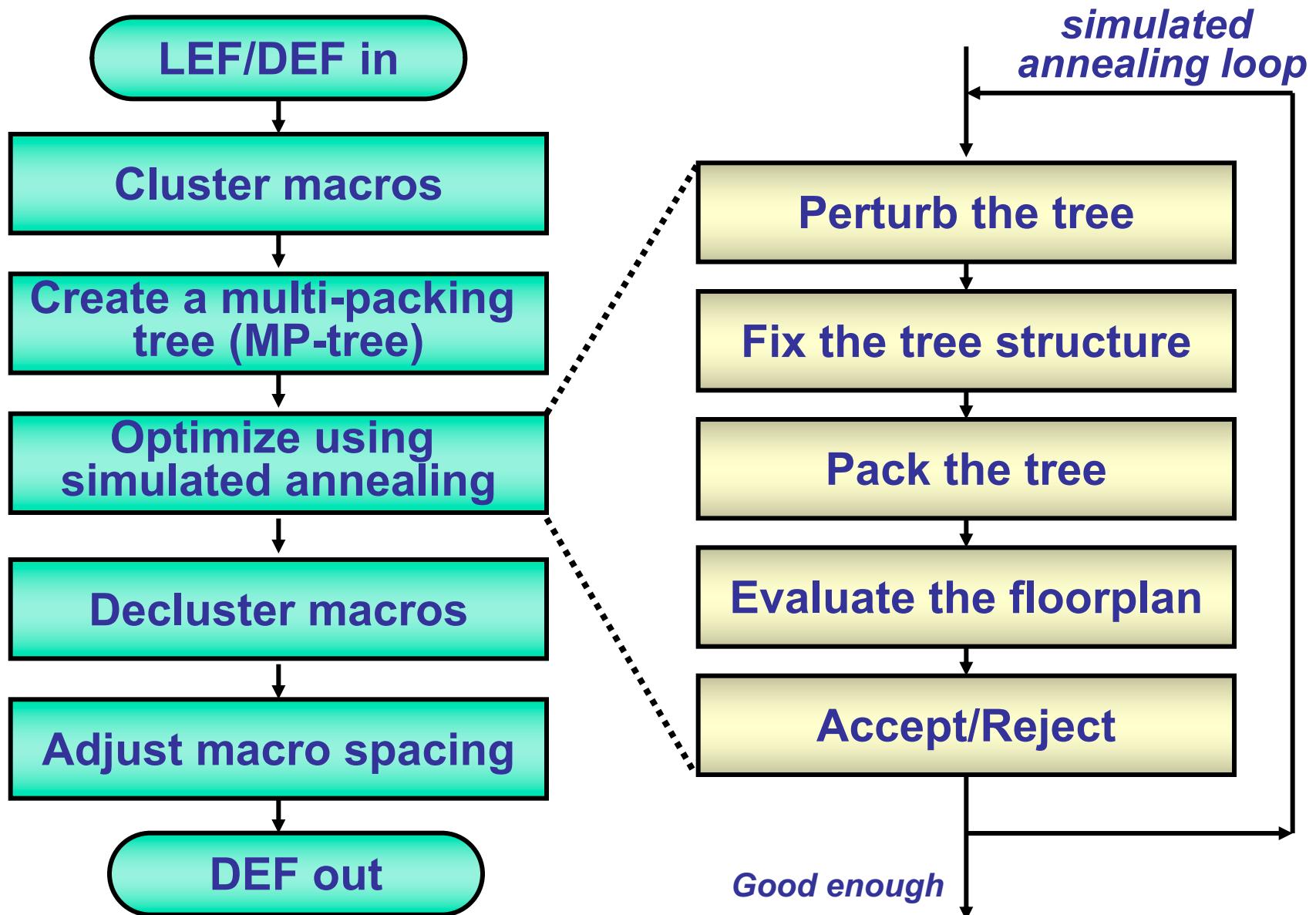
- Operations for simulated annealing to perturb one MP-tree to another
- Op1: Rotate a block or a cluster
- Op2: Resize a cluster
- Op3: Move a node in a packing subtree to another place
- Op4: Swap two nodes within one or two packing subtrees
- Op5: Swap two packing subtrees

Evaluation of a Macro Placement

- Macro placement area
- Wirelength
- Macro displacement



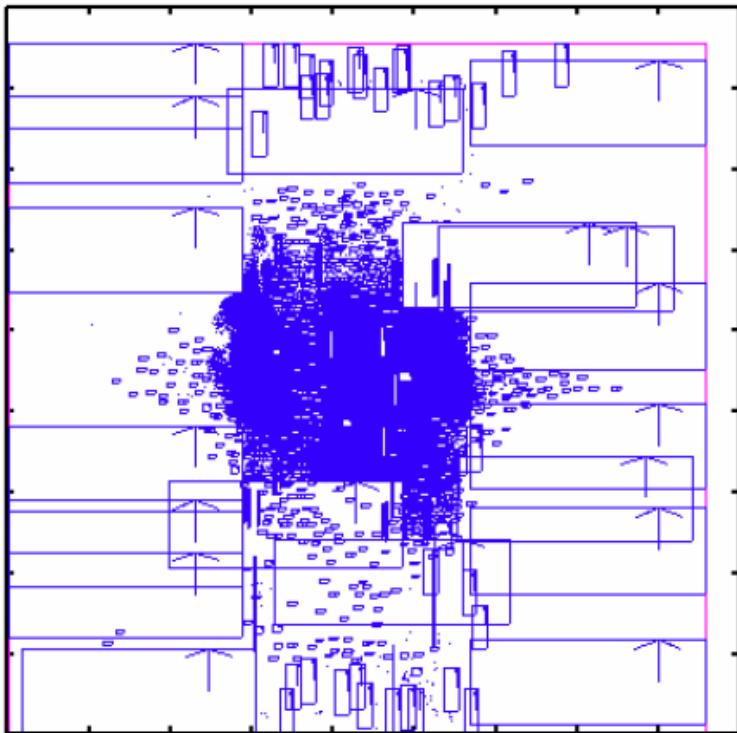
Macro Placement Flow



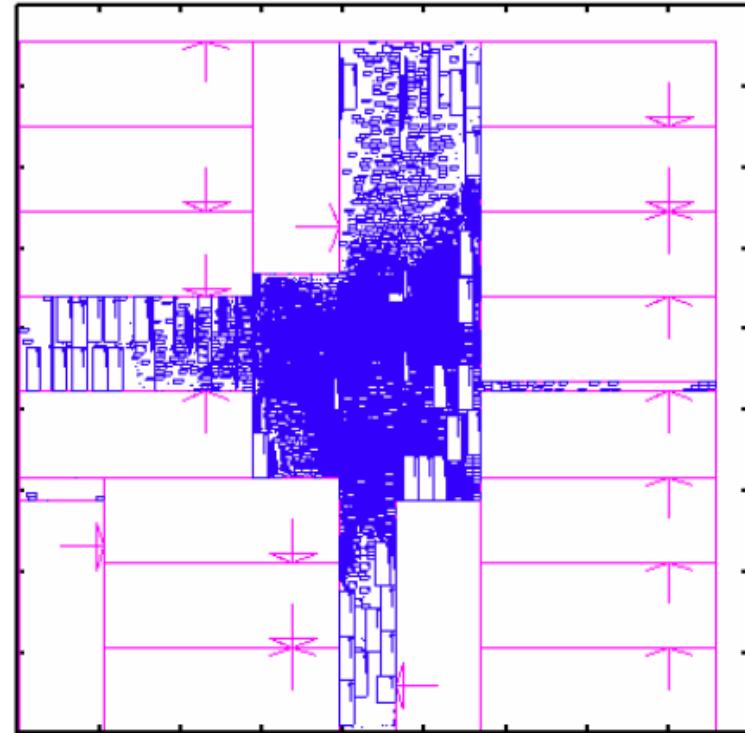
MP-tree Macro Placer

- Is very **fast** for operations and packing based on binary trees (extension of B*-trees)
- Considers **macro orientations** and **spacing** between macros
 - Lead to smaller wirelength and less congestion
- Can handle various **placement constraints**
 - Pre-placed blocks, corner block, and placement blockages
- Can be combined with state-of-the-art standard cell placers to obtain better mixed-size placement

ISPD-06 newblue3 Layouts

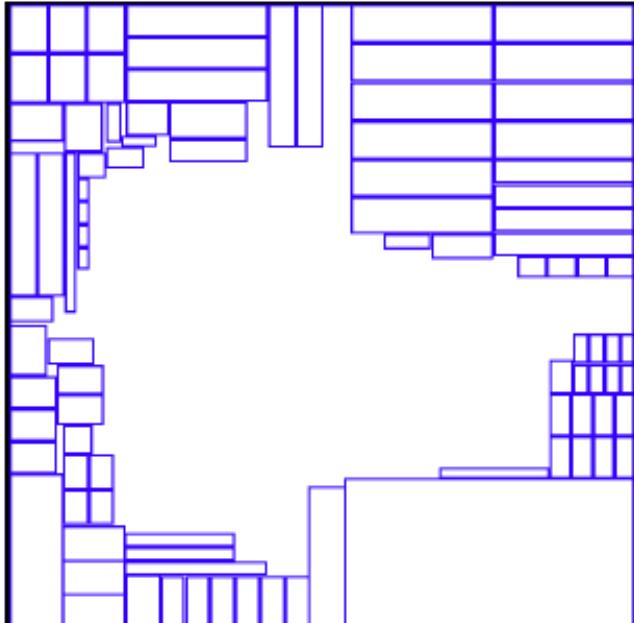


**NTUplace3 alone
(failed to find a
legal placement)**



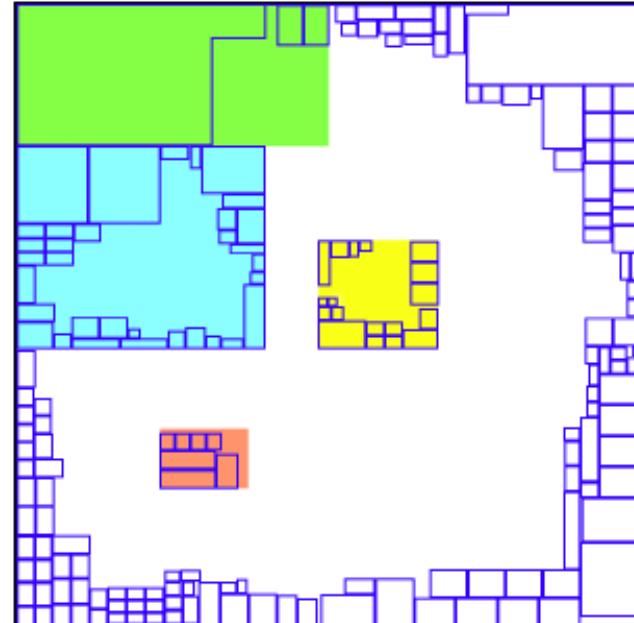
MP-tree + NTUplace3

Mchip Benchmark Results



(a) mchip2

95 Macros

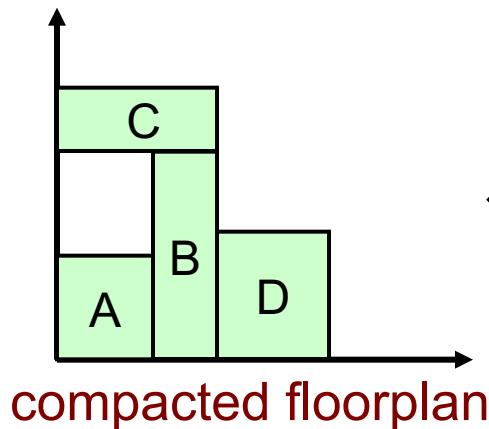


(b) mchip4

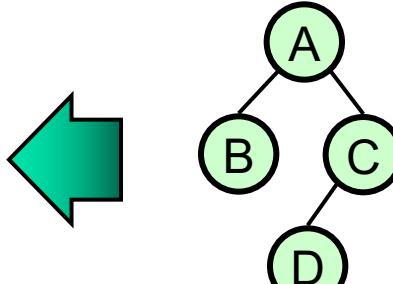
**380 Macros w/
4 region constraints**

Limitation of the MP-tree

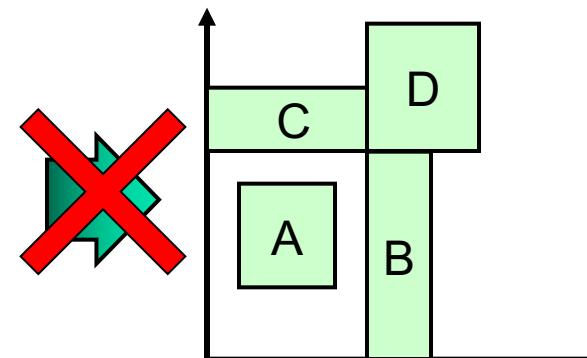
- MP-tree can generate only compacted macro placement results, **not desirable for low-density designs**



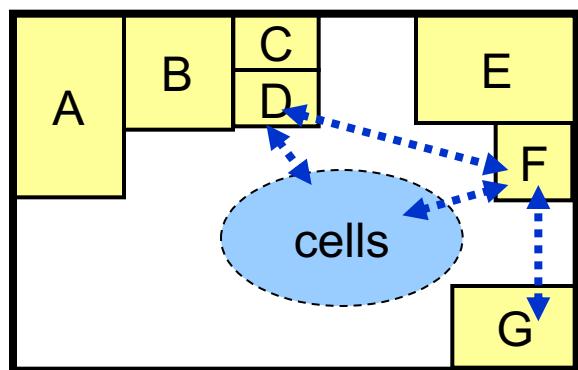
compacted floorplan



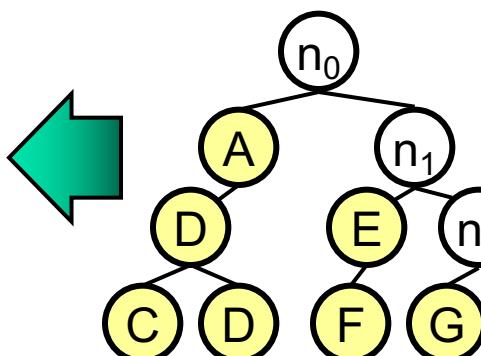
B*-tree



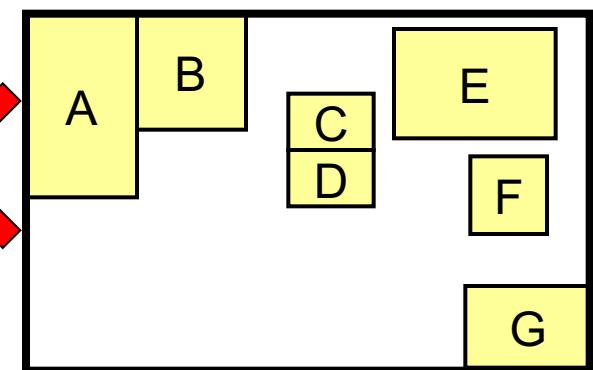
non-compacted floorplan



compacted placement



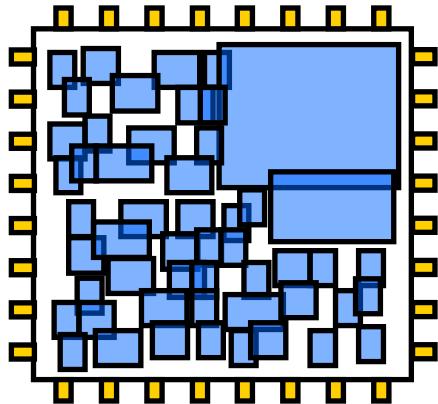
MP-tree



non-compacted placement

Techniques to Overcome the Limitations

- Chen, Chuang, Chang, Chang, ICCAD-08.



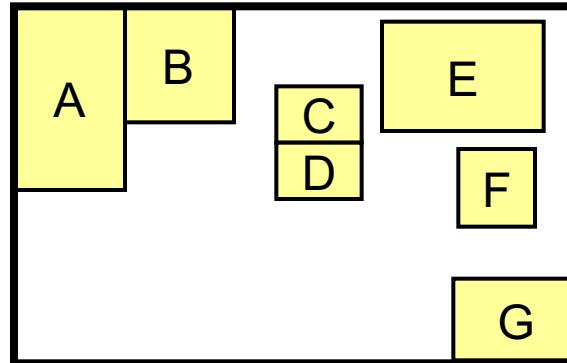
Legal placement?



Floorplan techniques



Constraint Graph (e.g., TCG)



Non-compacted macro placement?



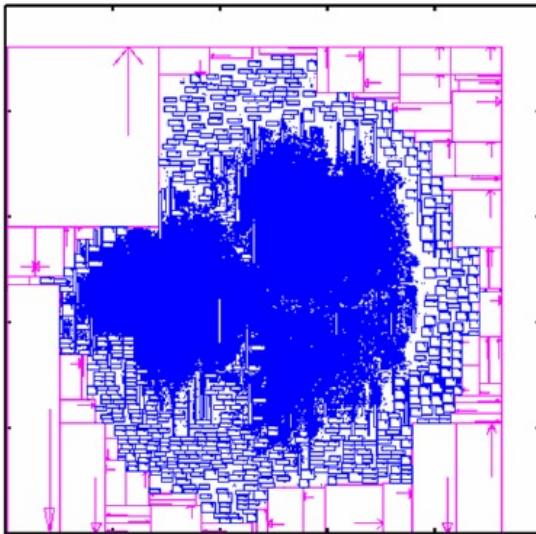
Placement prototype



Linear Programming

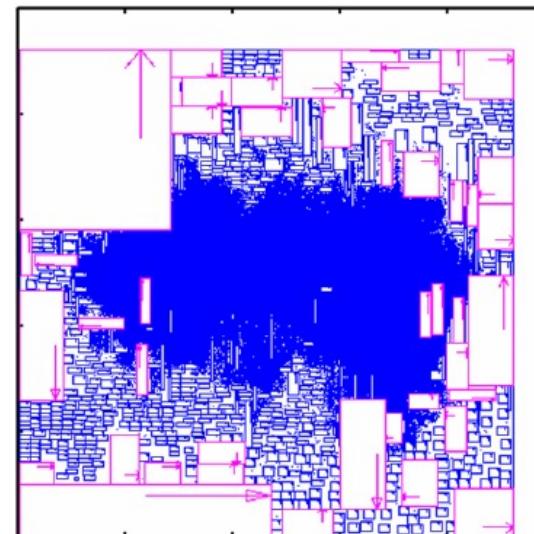
Layout Example: adaptec5

- Integrated with NTUpPlace3
- 80% utilization rate



MP-tree

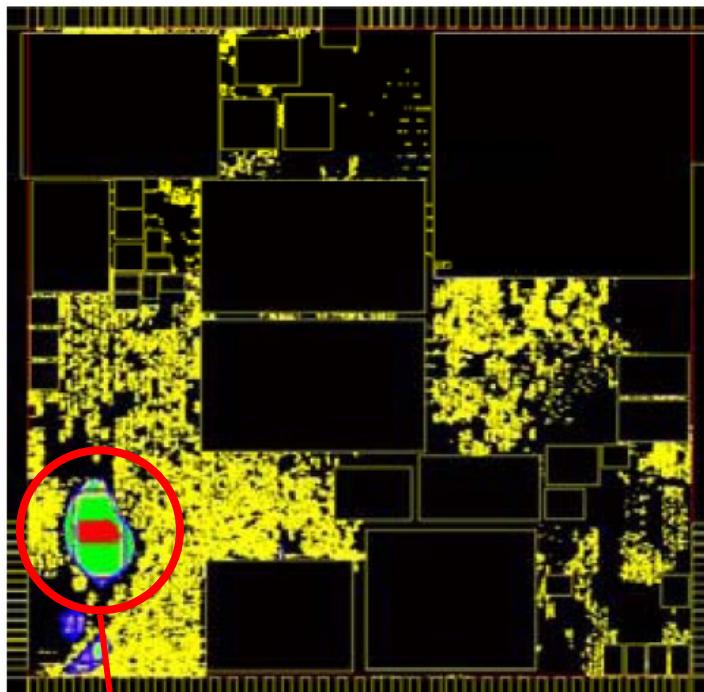
HPWL: 32.72×10^7



TCG

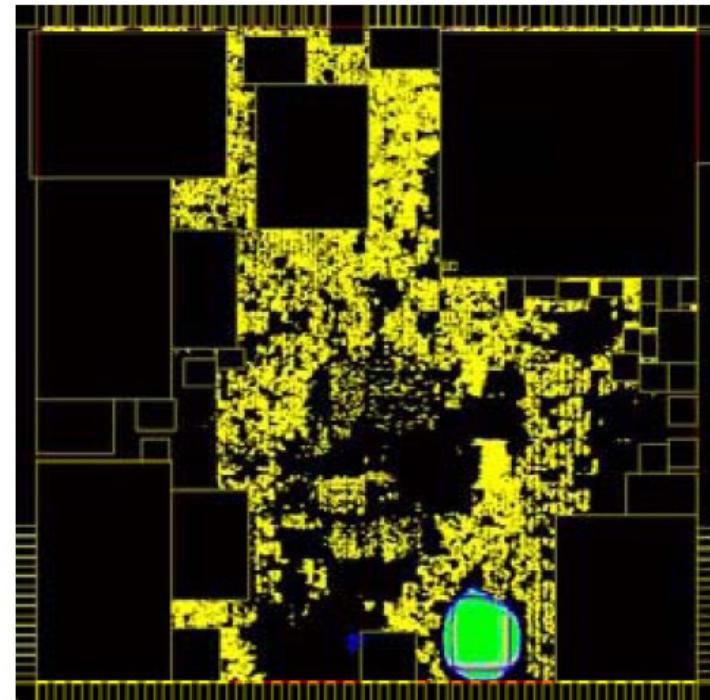
HPWL: 29.72×10^7

Congestion Map - gchip



(a) From Tool A

congestion hot spot



(b) From our macro placer

Summary: Placement

- Popular methods: partitioning-based, simulated annealing, analytical placement
- Major steps: global placement -> legalization -> detailed placement
- Design Styles
 - Gate Array / FPGA
 - Standard cell
 - Macro block
 - Mixed-size (block and cells)
- Integrated with the design flow
 - Placement with clock-tree synthesis, power/ground network synthesis, scan-chain reordering, etc
- Many modern design challenges, e.g.,
 - Mixed-size large-scale placement
 - Other objectives: routability, timing, power (voltage islands), thermal, manufacturability, reliability
 - 3D IC placement (thermal vias?)

Popular Wirelength Models

HPWL
$$W(\mathbf{x}, \mathbf{y}) = \sum_{\text{net } e} \left(\max_{v_i, v_j \in e} |x_i - x_j| + \max_{v_i, v_j \in e} |y_i - y_j| \right)$$

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

Log-sum-exp
$$\begin{aligned} & \gamma \sum_{e \in E} \left(\log \sum_{v_k \in e} \exp(x_k/\gamma) + \log \sum_{v_k \in e} \exp(-x_k/\gamma) + \right. \\ & \quad \left. \log \sum_{v_k \in e} \exp(y_k/\gamma) + \log \sum_{v_k \in e} \exp(-y_k/\gamma) \right). \end{aligned}$$

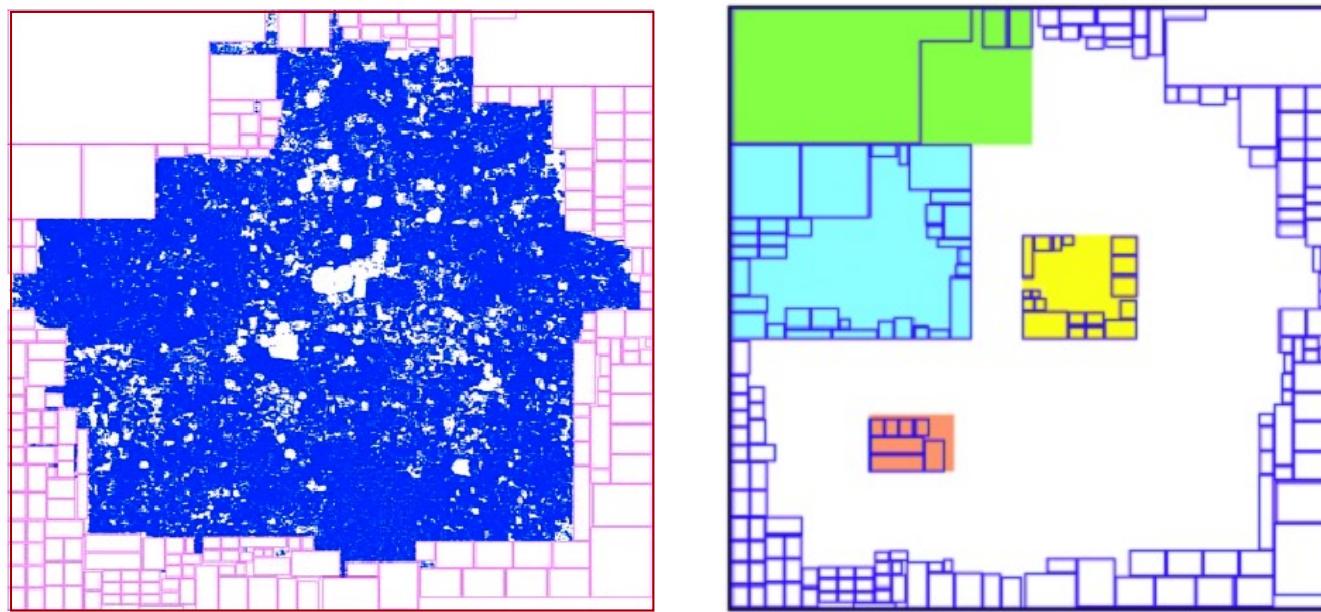
L_p-norm
$$\sum_{e \in E} \left(\left(\sum_{v_k \in e} x_k^p \right)^{\frac{1}{p}} - \left(\sum_{v_k \in e} x_k^{-p} \right)^{-\frac{1}{p}} + \left(\sum_{v_k \in e} y_k^p \right)^{\frac{1}{p}} - \left(\sum_{v_k \in e} y_k^{-p} \right)^{-\frac{1}{p}} \right)$$

Placement Benchmarks

- ISPD 2005 and 2006 benchmark suites
 - Derived from IBM ASIC designs
 - Mixed-size designs with both fixed and movable modules
 - Up to 2.5 million movable modules
- IBM-PLACE 2.0
 - Derived from smaller IBM circuits released in 1998
 - Has standard cells only and no connection to I/O pads
- IBM-MSwPins
 - Derived from the same circuits as IBM-PLACE 2.0
 - Contain large movable macros and fixed pads
- Faraday mixed-sized benchmarks
 - Have sufficient routing information to run industrial routers
- PEKO benchmark suites
 - Synthetic placement benchmarks with known optimal wirelength
 - Many variations

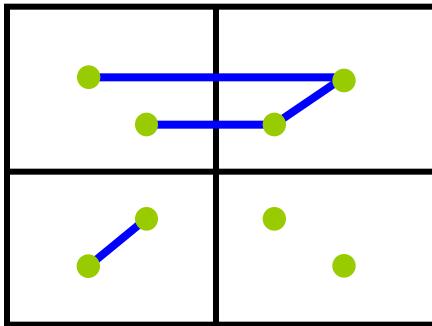
Large-Scale Mixed-Size Placement

- We still have a long road to go for large-scale mixed-size placement!!
 - Is the two-stage approach the best alternative?
 - Should we pack macros along the chip boundaries?
 - Need to consider many other placement constraints?

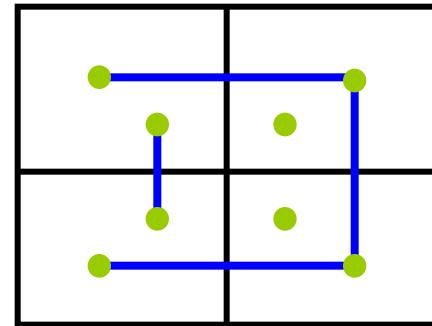


Routability-Driven Placement

- Most placement algorithms focus on wirelength minimization alone
- Congestion objective differs from the wirelength one
 - Jiang, Su, Chang, DAC-08



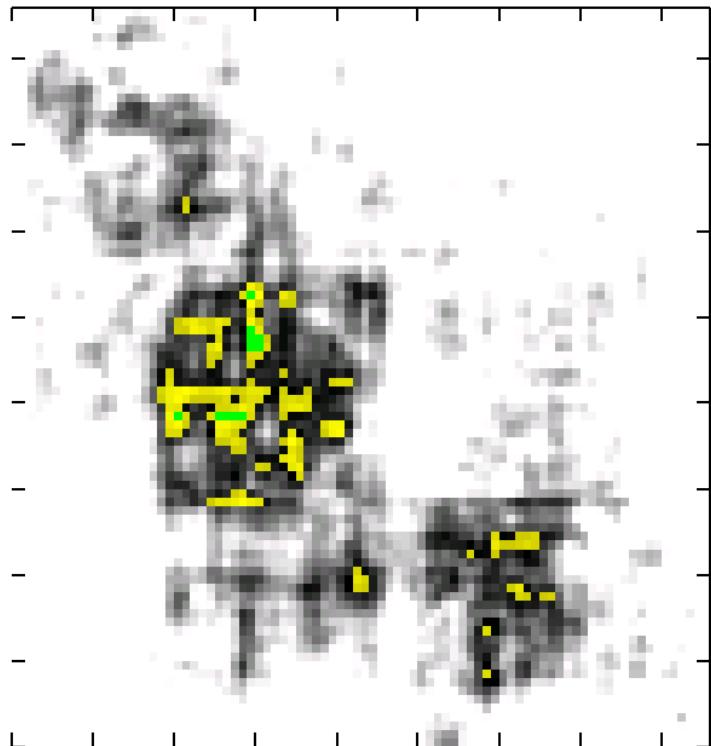
Wirelength minimization
Maximum congestion = 2



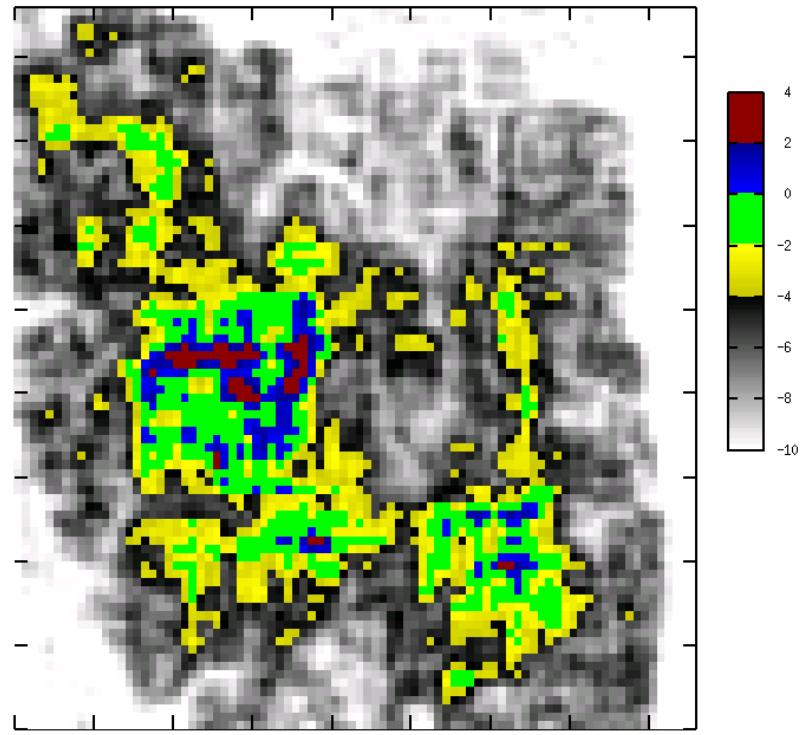
Congestion optimization
Maximum congestion = 1

- Further, placement should interact with routing
 - Fast and accurate routing demand estimation for placement?
 - Simultaneous placement and routing?
 - Pan & Chu, DAC-07

Example Congestion Maps



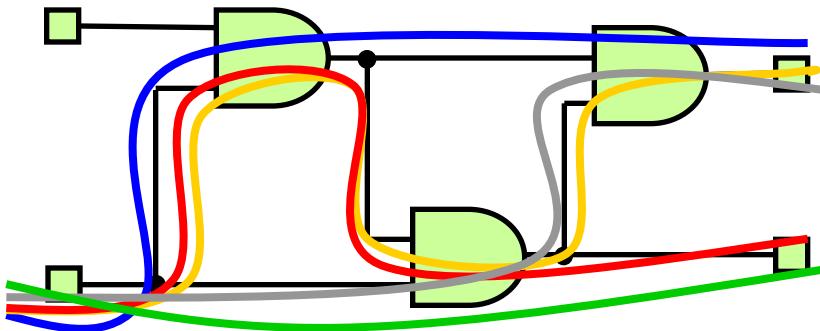
Congestion optimized result
violations = 0



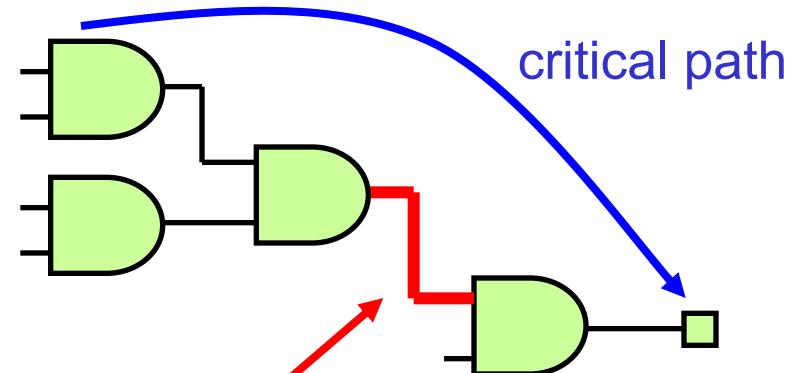
Wirelength minimized result
violations = 6057

Timing-Driven Placement

- Two major techniques for timing-driven placement
 - Path-based methods incur prohibitively time complexity due to the exponentially increasing number of paths
 - Net-based methods lack the global view of the full path



timing paths increases exponentially

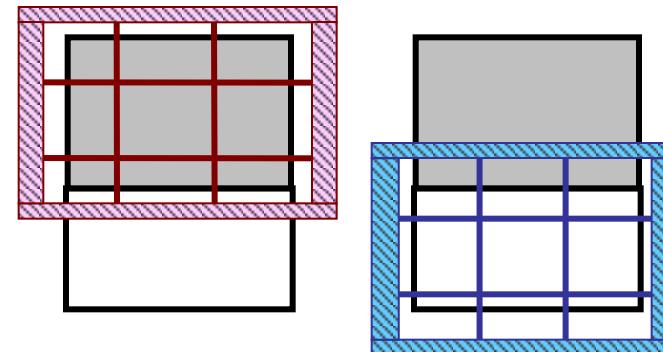
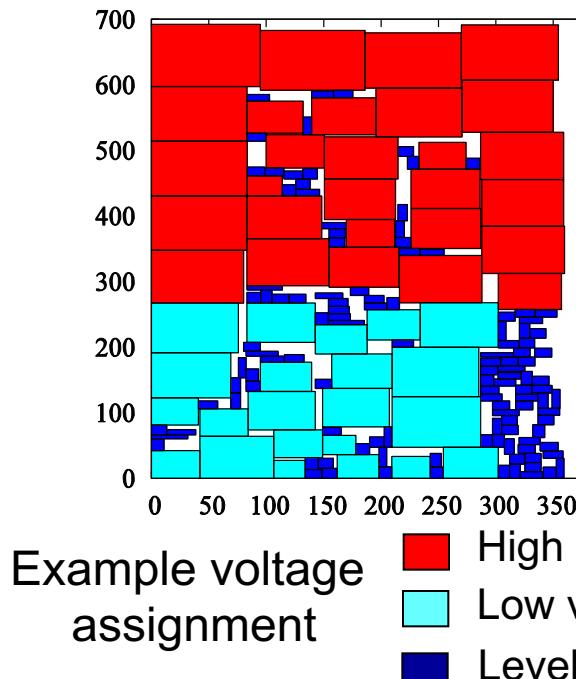


Net based method may not optimize timing effectively

- A timing optimization technique with low complexity and high controllability is desired

Power-Aware Placement

- Multiple supply voltages are widely used for low-power design
 - Voltages can be assigned during floorplanning/placement
- Integration of voltage assignment and placement can further reduce the power consumption
 - Lee, Liu, Chang, ICCAD-06, ICCAD-07

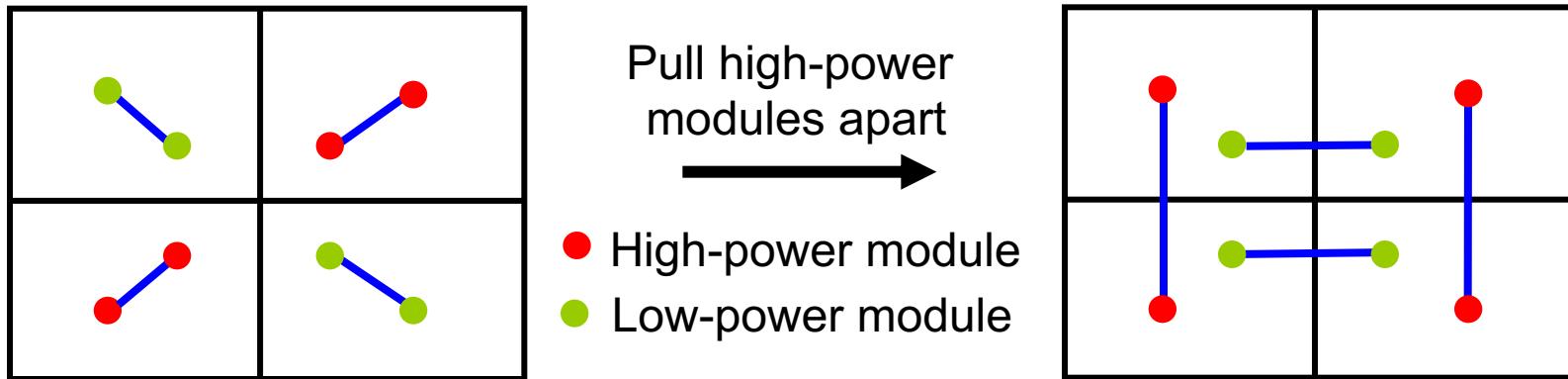


Resulting power ring structure

- High voltage power ring
- Low voltage power ring

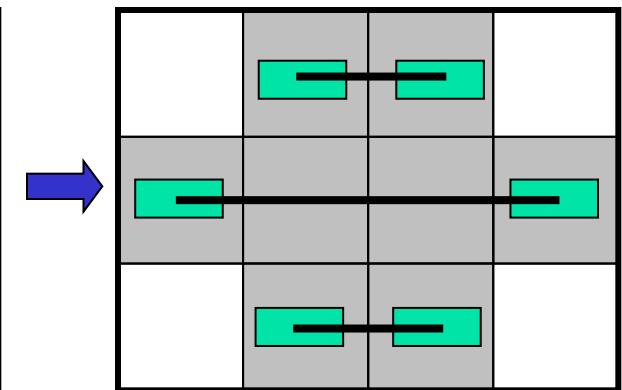
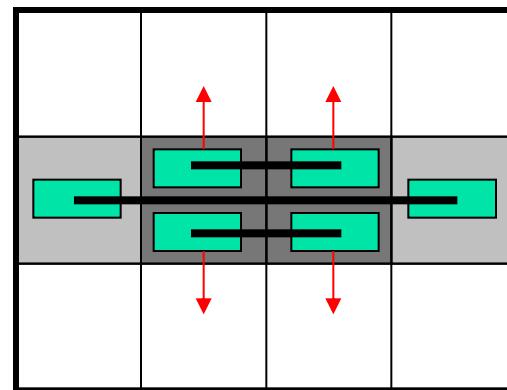
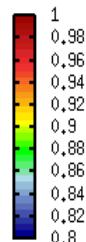
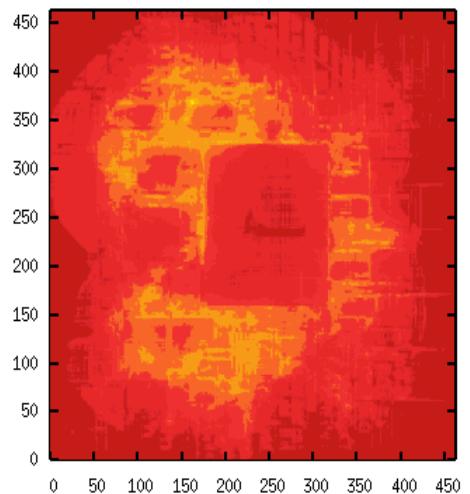
Thermal-Aware Placement

- The increasing integration density and clock frequency raise the chip temperature significantly
- Power density is a dominant factor to reduce the chip temperature
 - Reducing power consumption alone is not sufficient
- Need to spread hot cells to lower chip temperature and thus to increase chip reliability



Manufacturability-Aware Placement

- Predictive Chemical Mechanical Polishing (CMP) model
 - The number of dummy fills and normalized copper thickness are functions of wire density [Cho et al, ICCAD-06]
- Wire density optimization is limited by pin locations
- Shall move cells/pins out of the high wire-density regions during placement
 - Chen, et al., ISPD-08 (TCAD, 2008)

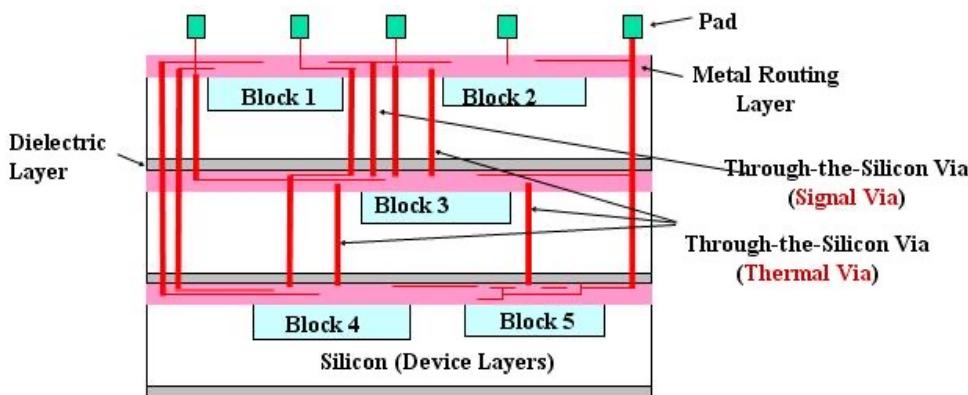


Better wire distribution

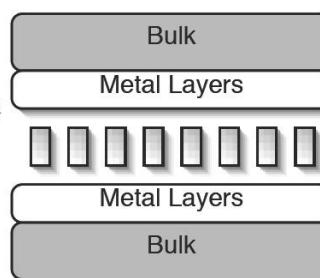
Normalized Copper Thickness Map

Placement for 3D (2.5D) IC's

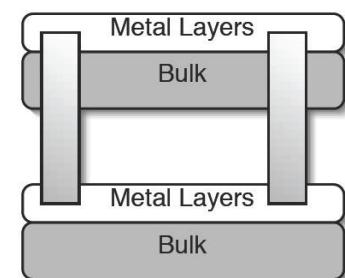
- 3D (2.5D) IC technology has been proposed to alleviate interconnect problems
- Thermal dissipation is the major challenge for 2.5D design.
- Dummy thermal vias are often inserted to dissipate heat
- Two integration strategies for 2.5D ICs: face-to-face (F2F) and face-to-back (F2B).
 - F2F is more popular due to its lower cost
- Problem: Place cells to optimize wirelength and thermal



Cross section view of a 2.5D IC



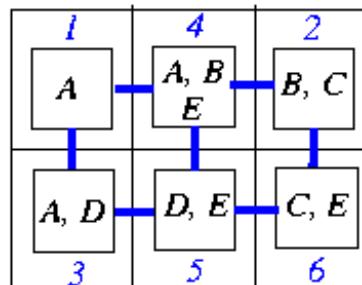
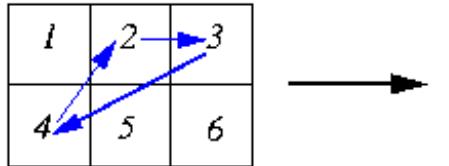
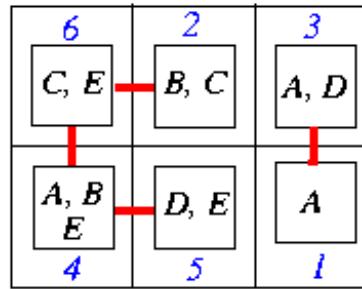
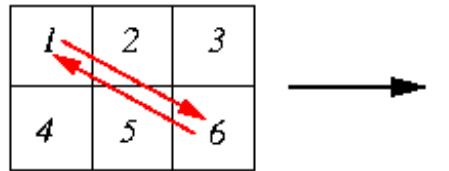
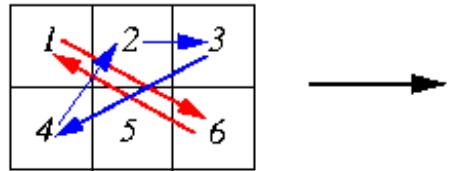
(a) face-to-face



(b) face-to-back

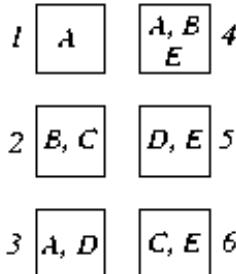
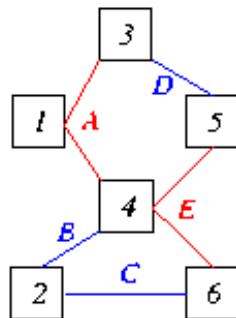
Two types of 2.5D IC's

Appendix A: Placement by Linear Assignment



Linear Assignment

- Problem Definition:
 - Instance: An $n \times n$ matrix $D=((d_{ij}))$ with value from $\mathbb{R}^+ \cup \{0\}$.
 - Configurations: All permutations $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$
 - Solutions: All configurations
 - Maximize or Minimize $D(\pi) \leftarrow \sum_{i=1}^n d_{i,\pi(i)}$
 - D can be interpreted as an edge-weighted complete bipartite graph G with n vertices on each side.
- The linear assignment problem can be solved in $O(n^3)$ time (known fastest algorithm by Fredman & Tarjan: $O(n^2 \log n + mn)$ time).



a	b	c
d	e	f

	a	b	c	d	e	f
t	1	1	1	0	0	0
2	1	0	1	1	0	1
3	1	1	1	0	0	0
4	1	0	1	0	0	0
5	1	1	1	1	1	1
6	1	0	1	1	0	1

1. signal A in top row
2. even #'s only in corners

1	3	4
2	5	6

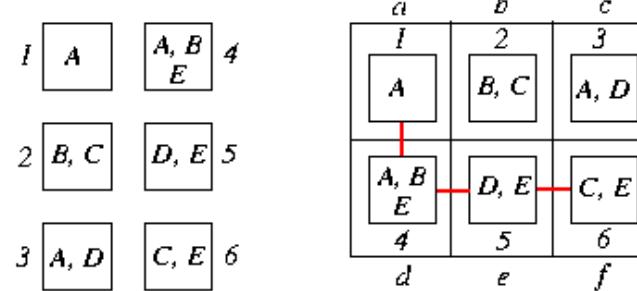
	a	b	c	d	e	f
t	1	1	1	0	0	0
2	1	0	0	1	0	0
3	1	1	1	0	0	0
4	1	0	1	0	0	0
5	1	1	1	1	1	1
6	1	0	1	1	0	1

1. signal A in top row
2. even #'s only in corners
3. signal C not on right

No feasible assignment!

Placement by Linear Assignment

- Akers, “On the use of linear assignment algorithm in module placement,” DAC-81.
- Total # of signal adjacencies for the placement = 3 (Signal A: between modules 1 and 4; Signal E: modules 4 and 5; Signal E: modules 5 and 6)



$$\mathbf{J}:$$
$$I \ 2 \ 3 \ 4 \ 5 \ 6$$
$$J = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

$$\mathbf{A}:$$
$$a \ b \ c \ d \ e \ f$$
$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 2 & 0 \\ 1 & 2 & 0 & 2 & 1 & 1 \\ 1 & 3 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 0 & 2 & 1 \\ 2 & 1 & 1 & 1 & 2 & 1 \end{bmatrix}$$

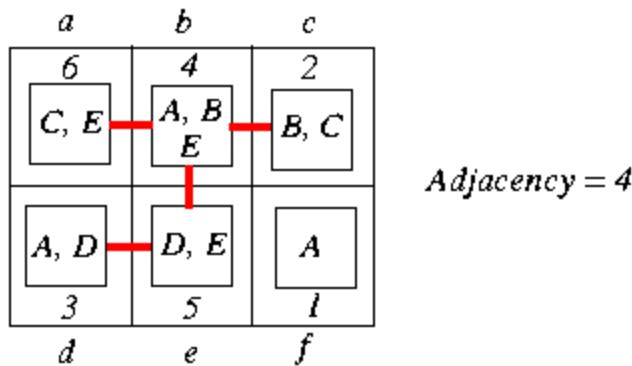
- J_{ij} : # of common signals between modules i and j .
- A_{ix} : # of signal adjacencies between module i and its neighbors if we move it to location x .
- Example: $A_{6a} = J_{62} + J_{64} = 2$; $A_{4e} = J_{42} + J_{44} + J_{46} = 2$.

Linear Assignment

$$\mathbf{A}: \begin{array}{ccccccc} & a & b & c & d & e & f \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} & \left[\begin{matrix} 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 2 & 0 \\ 1 & 2 & 0 & 2 & 1 & 1 \\ 1 & 3 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 0 & 2 & 1 \\ 2 & 1 & 1 & 1 & 2 & 1 \end{matrix} \right] & \end{array}$$

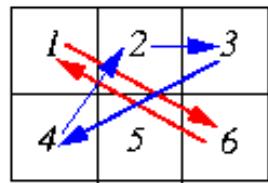
max linear assignment
 $= 1+1+2+3+2+2 = 11$

Bad approach: Rearrange the placement according to this solution

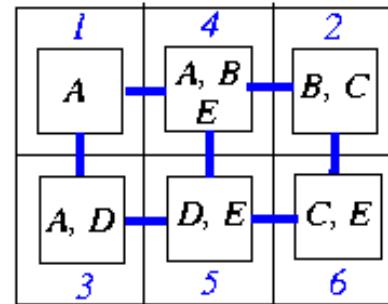
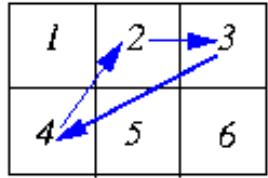
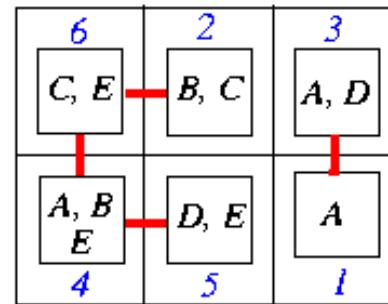
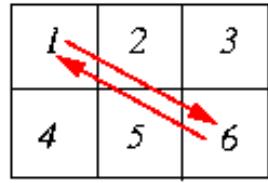


- Minor improvement.
 - Reason: When a module reached a “good” position, the module previously adjacent to that position had gone elsewhere!

Better Approach for Linear Assignment



6	4	2
3	5	1



- Modify the placement based on a cycle that yields maximum improvement.
- Repeat the process all over again.