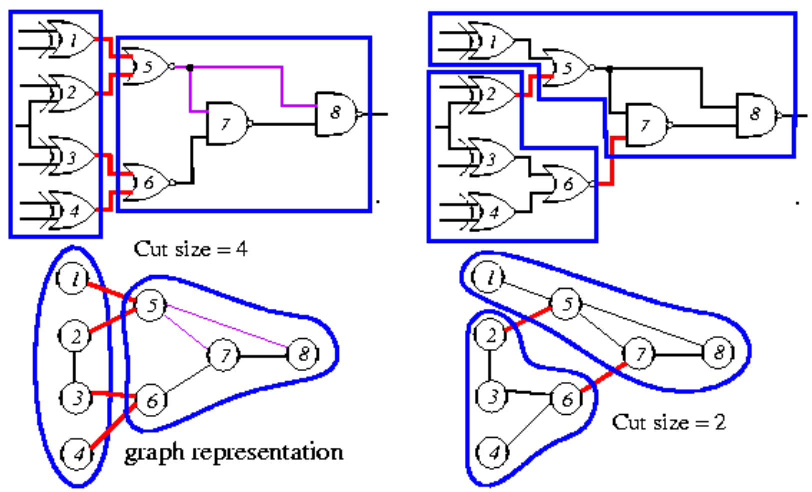
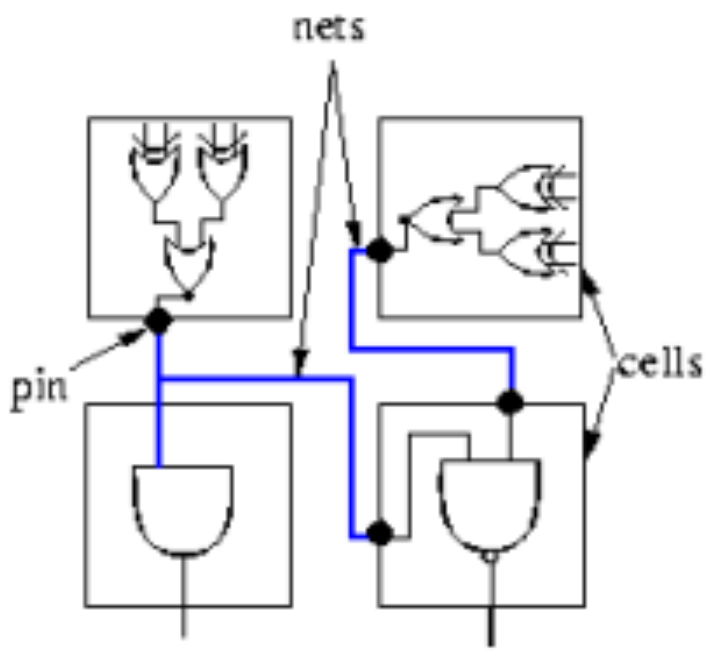
Partitioning

- Course contents:
  - Kernighagn & Lin heuristic
  - Fiduccia-Mattheyses heuristic
  - Simulated annealing based method
  - Network-flow based method
  - Multilevel circuit partitioning
  - Clustering for partition-based placement



Basic Definitions in Physical Design

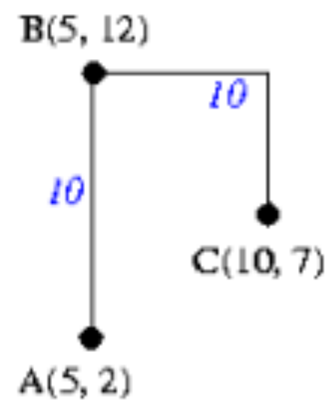
- **Cell:** a logic block used to build larger circuits.
- **Pin:** a wire (metal or polysilicon) to which another external wire can be connected.
- **Nets:** a collection of pins which must be electronically connected.
- **Netlist:** a list of all nets in a circuit.



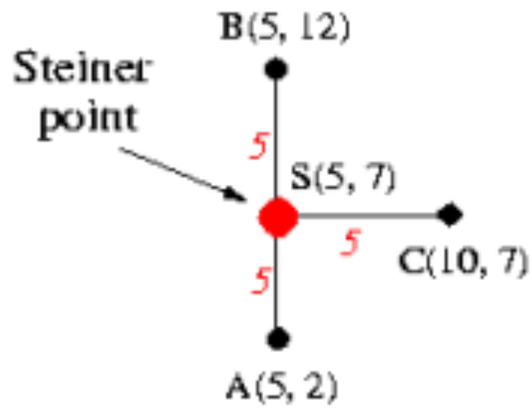
Courtesy of Prof. Y.-W. Chang and H.-M. Chen

Basic Definitions in Physical Design (cont)

- **Manhattan distance:** If two points (pins) are located at coordinates (x_1, y_1) and (x_2, y_2), the Manhattan distance between them is given by $d_{12} = |x_1 - x_2| + |y_1 - y_2|$.
- **Rectilinear spanning tree:** a spanning tree that connects its pins using Manhattan paths.
- **Steiner tree:** a tree that connects its pins, and additional points (**Steiner points**) are permitted to used for the connections.



A rectilinear spanning tree

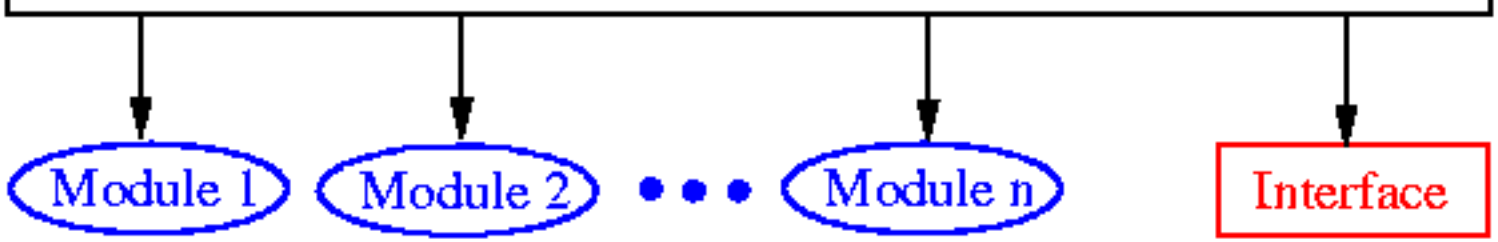


A rectilinear Steiner tree

What is Partitioning?

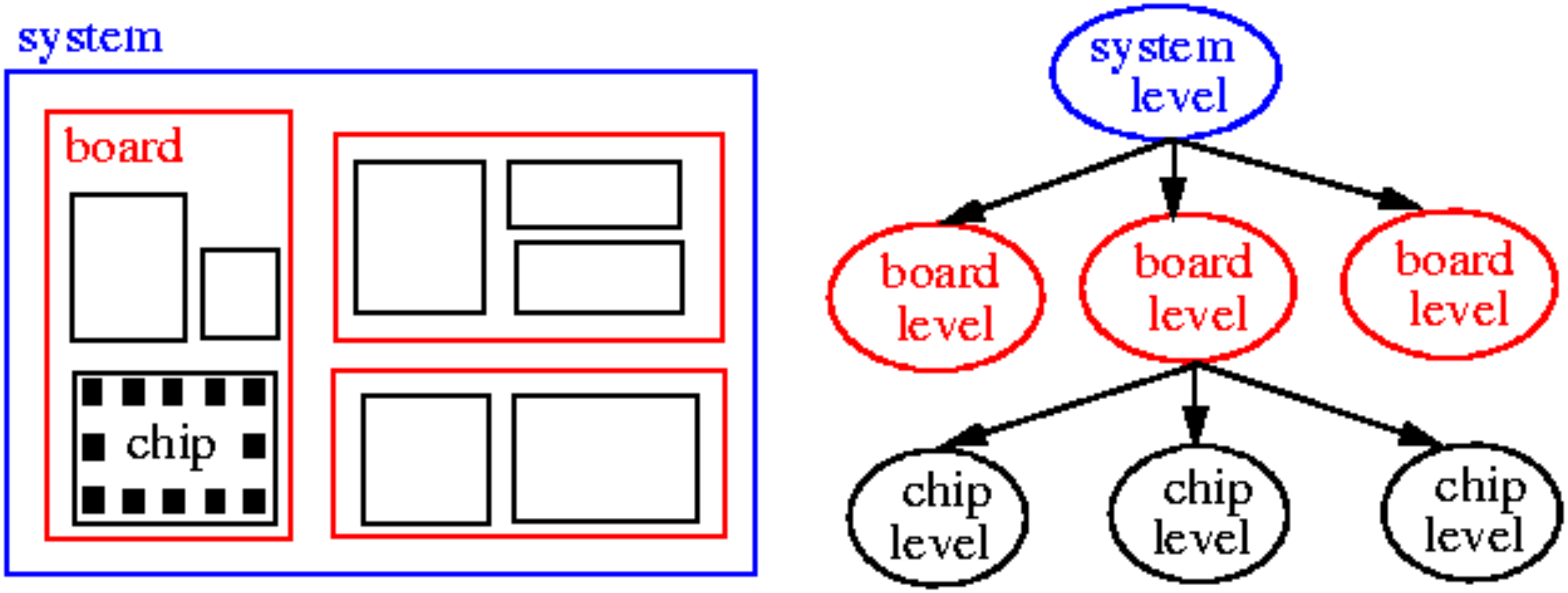
system design

- Decomposition of a complex system into smaller subsystems.
- Each subsystem can be designed independently speeding up the design process.
- Decomposition scheme has to minimize the interconnections among the subsystems. sub problem之間盡量要independent
- Decomposition is carried out hierarchically until each subsystem is of managable size.



Levels of Partitioning

- The levels of partitioning: system, board, chip.
- Hierarchical partitioning: higher costs for higher levels.

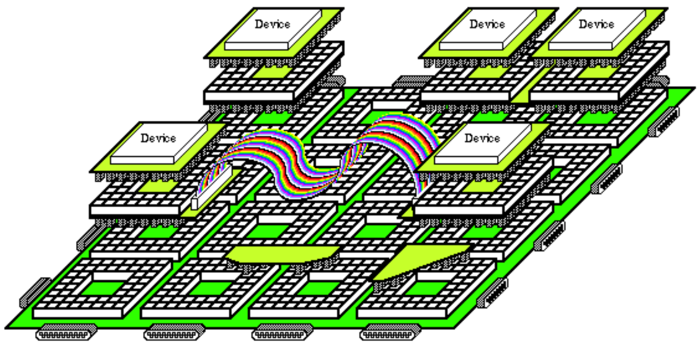


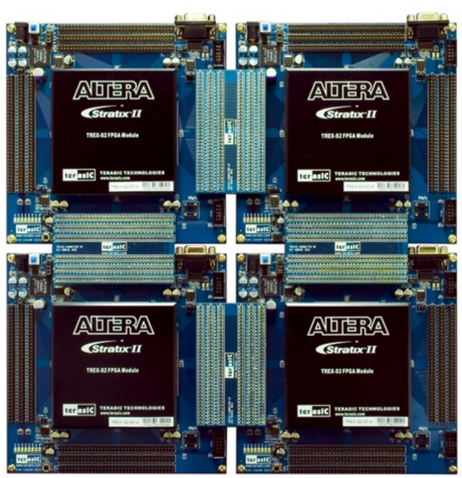
Example Applications

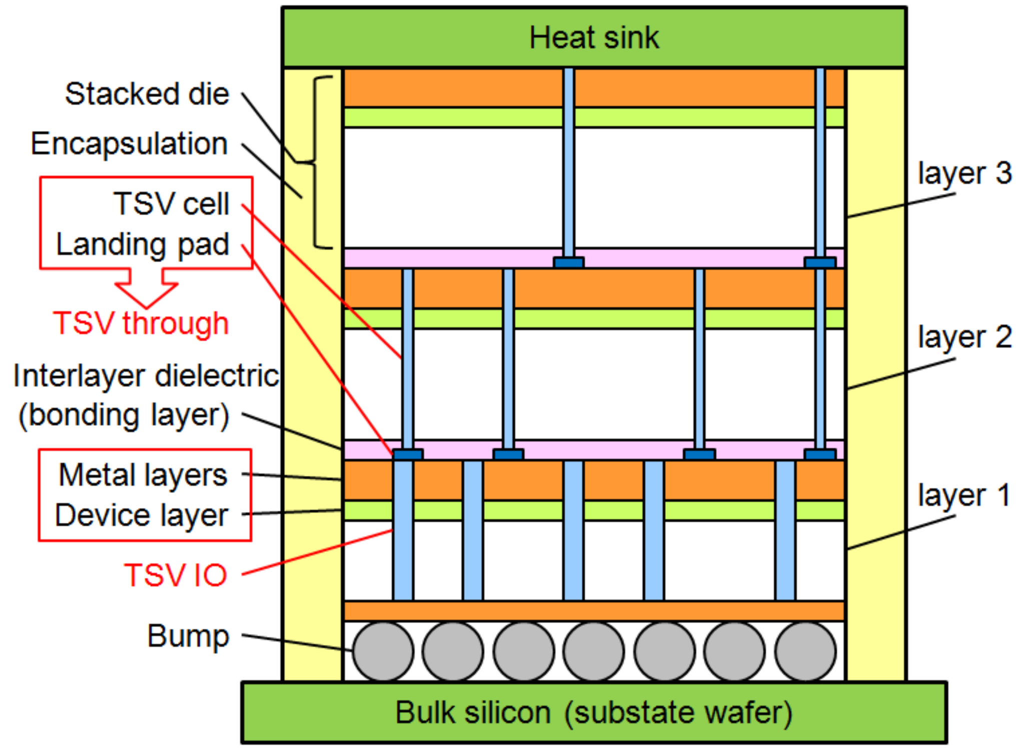
Multi-FPGA Systems

FPGA有容量限制，因此若要燒錄更巨大的電路進去，則必須要先進行切割

3D IC



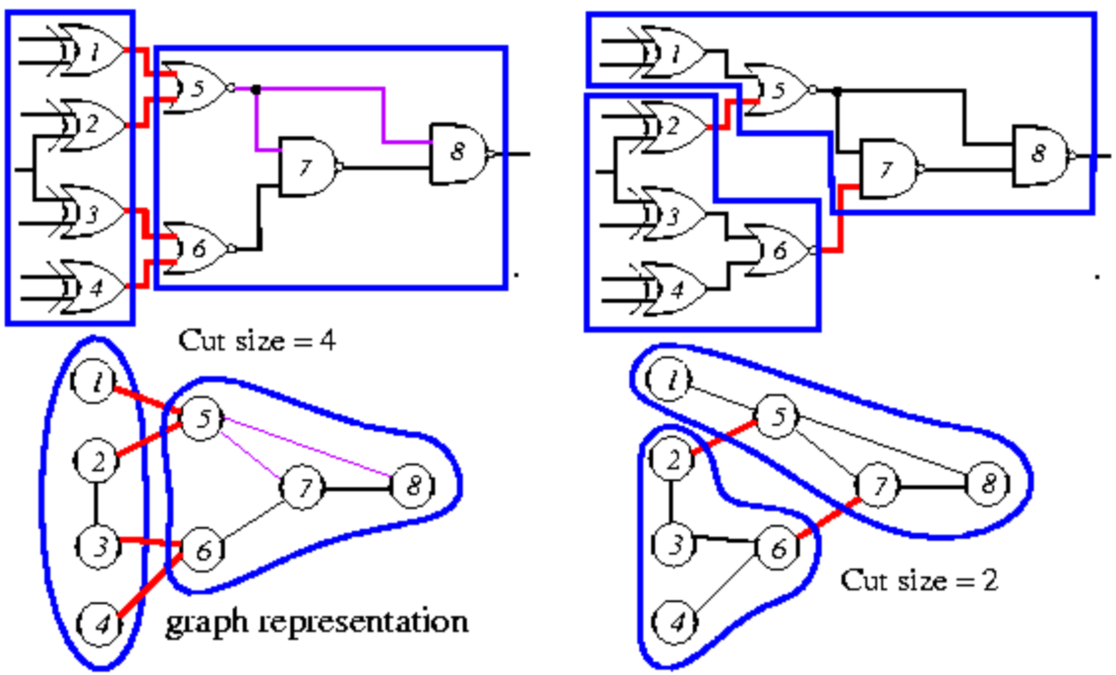




Courtesy of Prof. Y.-W. Chang and H.-M. Chen

Circuit Partitioning

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



Problem Definition: Partitioning

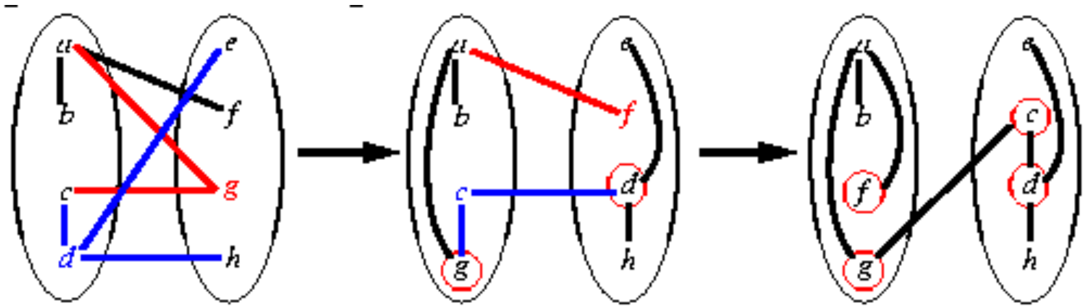
- ***k*-way partitioning:** Given a graph $G(V, E)$, where each vertex $v \in V$ has a **size** $s(v)$ and each edge $e \in E$ has a **weight** $w(e)$, the problem is to divide the set V into k disjoint subsets $V_1, V_2, \ldots, V_k$, such that an objective function is optimized, subject to certain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

Kernighan-Lin Algorithm

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

Kernighan-Lin Algorithm: A Simple Example

- Each edge has a unit weight.



Step #	Vertex pair	Cost reduction	Cut cost
0	-	0	5
1	{d, g}	3	2
2	{c, f}	1	1
3	{b, h}	-2	3
4	{a, e}	-2	5

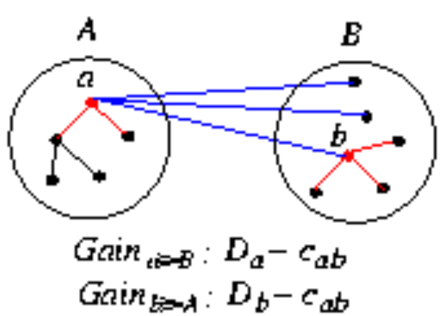
- Questions: How to compute cost reduction? What pairs to be swapped?
  - Consider the change of internal & external connections.

每次結束後的最佳解，都會變成下次iteration的初始狀態，再重新進行partition

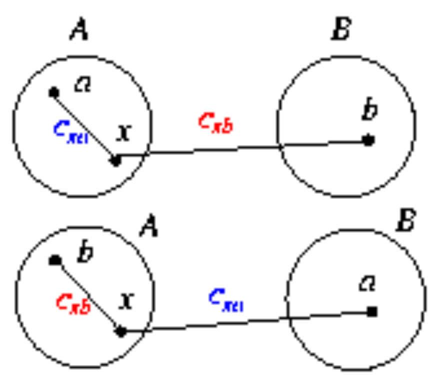
Properties

- Two sets *A* and *B* such that |*A*| = *n* = |*B*| and $A \cap B = \varnothing$.
- **External cost** of $a \in A$: $E_a = \sum_{v \in B} c_{av}$. c為該條edge的weight
- **Internal cost** of $a \in A$: $I_a = \sum_{v \in A} c_{av}$.
- *D*-value of vertex *a*: $D_a = E_a - I_a$ (benefit for moving *a*).
- Reduction in the cost (gain) for swapping *a* and *b*: $g_{ab} = D_a + D_b - 2c_{ab}$. a與b的連線在計算其個別的external cost都有被計算到，但他們交換之後彼此間的連線並沒有減少，因此要將多計算的那兩次扣掉
- If $a \in$ A and $b \in B$ are interchanged, then the new *D*-values for vertices other than *a* and *b*, ***D'***, are given by

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



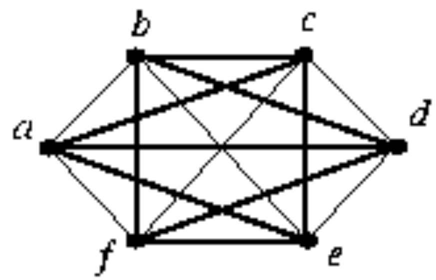
Internal cost vs. External cost



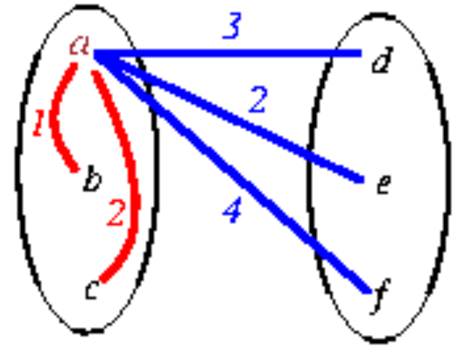
before swap	after swap	ΔC
$-c_{xa}$	$+c_{xa}$	$+2c_{xa}$
$+c_{xb}$	$-c_{xb}$	$-2c_{xb}$
I'x	E'x	

updating D−values

Kernighan-Lin Algorithm: A Weighted Example (1/5)



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



costs associated with *a*

Initial cut cost = (3+2+4)+(4+2+1)+(3+2+1) = 22

- Iteration 1:

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

Weighted Example (2/5)

- Iteration 1:

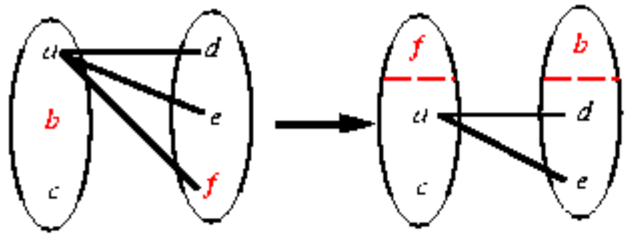
$$
\begin{array}{lll}
I_a = 1+2 = 3; & E_a = 3+2+4 = 9; & D_a = E_a - I_a = 9-3 = 6 \\
I_b = 1+1 = 2; & E_b = 4+2+1 = 7; & D_b = E_b - I_b = 7-2 = 5 \\
I_c = 2+1 = 3; & E_c = 3+2+1 = 6; & D_c = E_c - I_c = 6-3 = 3 \\
I_d = 4+3 = 7; & E_d = 3+4+3 = 10; & D_d = E_d - I_d = 10-7 = 3 \\
I_e = 4+2 = 6; &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

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

$$
\begin{aligned}
g_{ad} &= D_a + D_d - 2c_{ad} = 6+3-2\times 3 = 3 \\
g_{ae} &= 6+0-2\times 2 = 2 \\
g_{af} &= 6+1-2\times 4 = -1 \\
g_{bd} &= 5+3-2\times 4 = 0 \\
g_{be} &= 5+0-2\times 2 = 1 \\
g_{bf} &= 5+1-2\times 1 = 4 \ (maximum) \\
g_{cd} &= 3+3-2\times 3 = 0 \\
g_{ce} &= 3+0-2\times 2 = -1 \\
g_{cf} &= 3+1-2\times 1 = 2
\end{aligned}
$$

- Swap *b* and *f*! $(\widehat{g_1} = 4)$

Weighted Example (3/5)



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

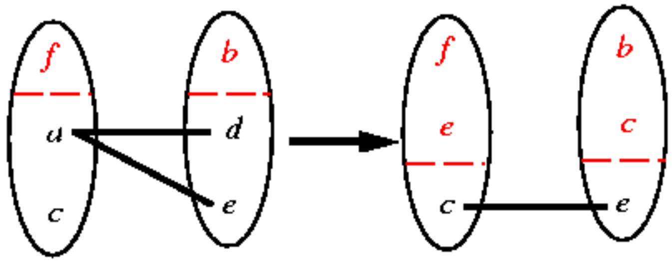
$$
\begin{aligned}
D'_a &= D_a + 2c_{ab} - 2c_{af} = 6 + 2 \times 1 - 2 \times 4 = 0 \\
D'_c &= D_c + 2c_{cb} - 2c_{cf} = 3 + 2 \times 1 - 2 \times 1 = 3 \\
D'_d &= D_d + 2c_{df} - 2c_{db} = 3 + 2 \times 3 - 2 \times 4 = 1 \\
D'_e &= D_e + 2c_{ef} - 2c_{eb} = 0 + 2 \times 2 - 2 \times 2 = 0
\end{aligned}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

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

$$
\begin{aligned}
g_{ad} &= D'_a + D'_d - 2c_{ad} = 0 + 1 - 2 \times 3 = -5 \\
g_{ae} &= D'_a + D'_e - 2c_{ae} = 0 + 0 - 2 \times 2 = -4 \\
g_{cd} &= D'_c + D'_d - 2c_{cd} = 3 + 1 - 2 \times 3 = -2 \\
g_{ce} &= D'_c + D'_e - 2c_{ce} = 3 + 0 - 2 \times 2 = -1 \ \ (maximum)
\end{aligned}
$$

- Swap c and e! $(\widehat{g_2} = -1)$

Weighted Example (4/5)



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

$$\begin{aligned} D''_a &= D'_a + 2c_{ac} - 2c_{ae} = 0 + 2 \times 2 - 2 \times 2 = 0 \\ D''_d &= D'_d + 2c_{de} - 2c_{dc} = 1 + 2 \times 4 - 2 \times 3 = 3 \end{aligned}$$

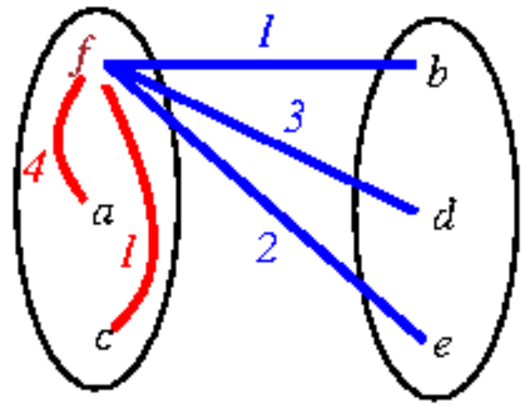
- $g_{xy} = D''_x + D''_y - 2c_{xy}$.

$$g_{ad} = D''_a + D''_d - 2c_{ad} = 0 + 3 - 2 \times 3 = -3 (\widehat{g_3} = -3)$$

- Note that this step is redundant $(\sum_{i=1}^{n} \widehat{g_i} = 0)$.
- Summary: $\widehat{g_1} = g_{bf} = 4,\ \widehat{g_2} = g_{ce} = -1,\ \widehat{g_3} = g_{ad} = -3.$
- Largest partial sum $\max \sum_{i=1}^{k} \widehat{g_i} = 4$ $(k = 1) \Rightarrow$ Swap *b* and *f*.

Weighted Example (5/5)

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



Initial cut cost = (1+3+2)+(1+3+2)+(1+3+2) = 18 (22−4)

- Iteration 2: Repeat what we did at Iteration 1 (Initial cost = 22-4 =18).
- Summary: $\widehat{g_1} = g_{ce} = -1,\ \widehat{g_2} = g_{ab} = -3,\ \widehat{g_3} = g_{fd} = 4.$
- Largest partial sum = $\max \sum_{i=1}^{k} \widehat{g_i} = 0$ (k=3) ⇒ Stop!

Kernighan-Lin Algorithm

Algorithm: Kernighan-Lin(G)
Input: $G = (V, E)$, $|V| = 2n$.
Output: Balanced bi-partition A and B with "small" cut cost.

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

Time Complexity of K-L Algorithm

- Line 4: Initial computation of D: $O(n^2)$
- Line 5: The **for**-loop: $O(n)$
- The body of the loop: $O(n^2)$.
  - Lines 6--7: Step i takes $(n-i+1)^2$ time.
- Lines 4--11: Each pass of the repeat loop: $O(n^3)$.
  - If sorting the D-values in a non-increasing order -> $O(n\log n)$
  - if more greedy (no sorting, just get the max of D-values) -> $O(n^2)$
- Suppose the repeat loop terminates after r passes.
- The total running time: $O(rn^3)$.
  - Polynomial-time algorithm? 不是個好的演算法

Line2: O(n)

Extensions of K-L Algorithm

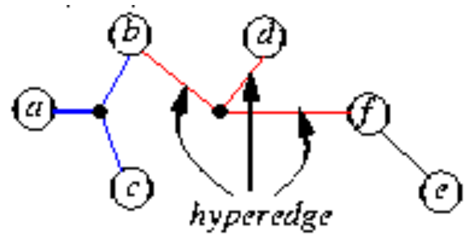
- **Unequal sized subsets** (assume $n_1 < n_2$)
  - Partition: $|A| = n_1$ and $|B| = n_2$.
  - Add n_2-n_1 dummy vertices to set A. Dummy vertices have no connections to the original graph.
  - Apply the Kernighan-Lin algorithm.
  - Remove all dummy vertices.
- **Unequal sized "vertices"** 把一個比較大的vertex換成多個unit vertex來表示，而這幾個unit vertex必兩兩相連
  1. Assume that the smallest "vertex" has unit size.
  2. Replace each vertex of size s with s vertices which are fully connected with edges of infinite weight.
  3. Apply the Kernighan-Lin algorithm.
- ***k*-way partition**
  1. Partition the graph into k equal-sized sets.
  2. Apply the Kernighan-Lin algorithm for each pair of subsets.
  3. Time complexity? Can be reduced by recursive bi-partition.

Drawbacks of the Kernighan-Lin Heuristic

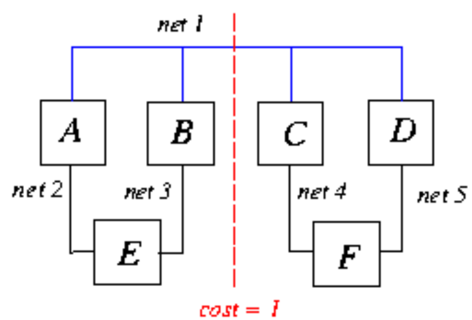
- The K-L heuristic **handles only unit vertex weights**.
  - Vertex weights might represent block sizes, different from blocks to blocks.
  - Reducing a vertex with weight $w(v)$ into a clique with $w(v)$ vertices and edges with a high cost increases the size of the graph substantially.
- The K-L heuristic **handles only exact bisections**.
  - Need dummy vertices to handle the unbalanced problem.
- The K-L heuristic **cannot handle hypergraphs**.
  - Need to handle multi-terminal nets directly.
- The **time complexity of a pass is high**, $O(n^3)$.
- Sensitive to initial partition

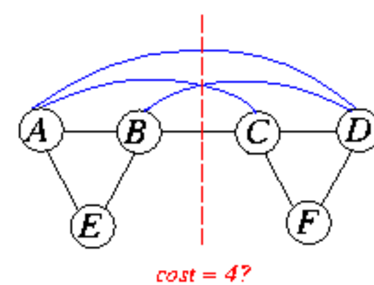
Coping with Hypergraph

- A hypergraph $H=(N, L)$ consists of a set N of vertices and a set L of hyperedges, where each hyperedge corresponds to a **subset** N_i of distinct vertices with $|N_i| \geq 2$.



- Schweikert and Kernighan, "A proper model for the partitioning of electrical circuits," 9th Design Automation Workshop, 1972.
- For multi-terminal nets, **net cut** is a more accurate measurement for cut cost (i.e., deal with hyperedges).
  - {A, B, E}, {C, D, F} is a good partition.
  - Should not assign the same weight for all edges.





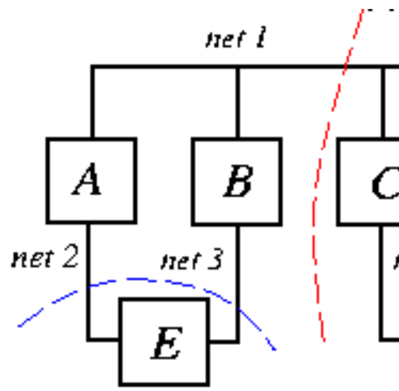
這樣表示是不準的，
實際cut cost只有1

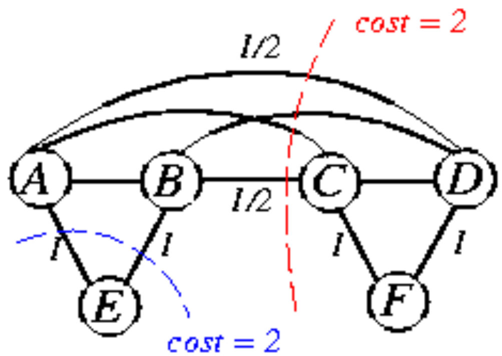
Courtesy of Prof. Y.-W. Chang and H.-M. Chen

Net-Cut Model

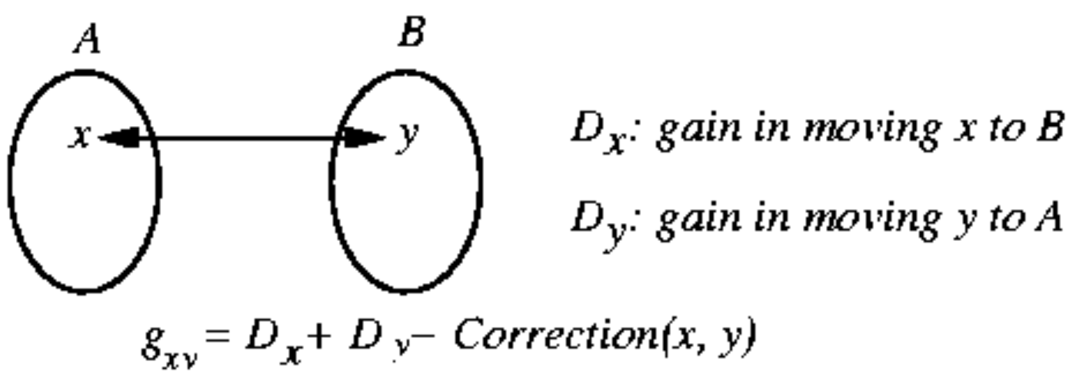
- Let $n(i)$ = # of cells associated with Net i.
- Edge weight $w_{xy} = \frac{2}{n(i)}$ for an edge connecting cells x and y.

也還是不太準





- Easy modification of the K-L heuristic.

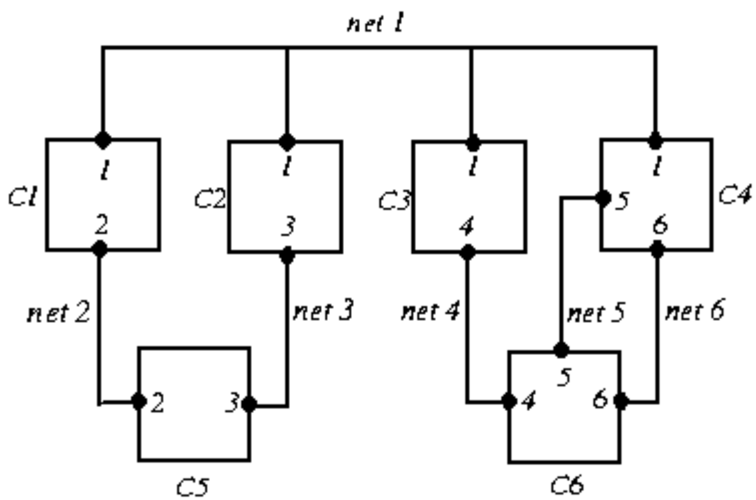


Fiduccia-Mattheyses Heuristic

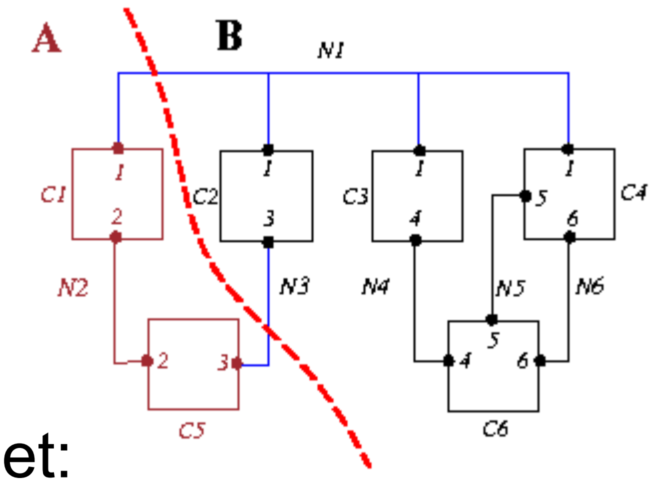
- Fiduccia and Mattheyses, "A linear time heuristic for improving network partitions," DAC-82.
- New features to the K-L heuristic:
  - Aims at **reducing net-cut costs**; the concept of cutsize is extended to hypergraphs. 與KL最大的差異，FM可以處理hypergraph
  - Only a **single vertex** is moved across the cut in a single move.
  - Vertices are weighted.
  - Can handle "unbalanced" partitions; a balance factor is introduced.
  - A special data structure is used to select vertices to be moved across the cut to improve running time.
  - **Time complexity** $O(P)$, where P is the total # of terminals. 重點是資料結構的設計

F-M Heuristic: Notation

- $n(i)$: # of cells in Net i; e.g., $n(1) = 4$.
- $s(i)$: size of Cell i.
- $p(i)$: # of pin terminals in Cell i; e.g., $p(6)=3$.
- C: total # of cells; e.g., $C=6$.
- N: total # of nets; e.g., $N=6$.
- P: total # of pins; $P = p(1) + \ldots + p(C) = n(1) + \ldots + n(N)$.

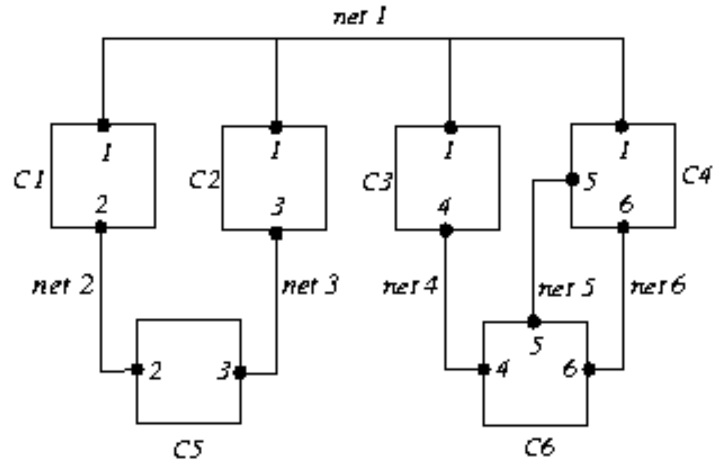


Cut



- **Cutstate** of a net:
  - Net 1 and Net 3 are **cut** by the partition.
  - Net 2, Net 4, Net 5, and Net 6 are **uncut**.
- **Cutset** = {Net 1, Net 3}.
- $|A|$ = size of $A = s(1)+s(5)$; $|B| = s(2)+s(3)+s(4)+s(6)$.
- **Balanced 2-way partition:** Given a fraction r, $0 < r < 1$, partition a graph into two sets A and B such that
  - $\frac{|A|}{|A|+|B|} \approx r$
  - Size of the cutset is minimized.

Input Data Structures



Cell array		Net array	
C1	Nets 1, 2	Net 1	C1, C2, C3, C4
C2	Nets 1, 3	Net 2	C1, C5
C3	Nets 1, 4	Net 3	C2, C5
C4	Nets 1, 5, 6	Net 4	C3, C6
C5	Nets 2, 3	Net 5	C4, C6
C6	Nets 4, 5, 6	Net 6	C4, C6

每個cell會記錄本身連結到那些net

每個net也會儲存本身包含哪些cell

用一個vector存所有的cell　用一個vector儲存所有的net

- Size of the network: $P = \sum_{i=1}^{6} n(i) = 14$
- Construction of the two arrays takes $O(P)$ time.

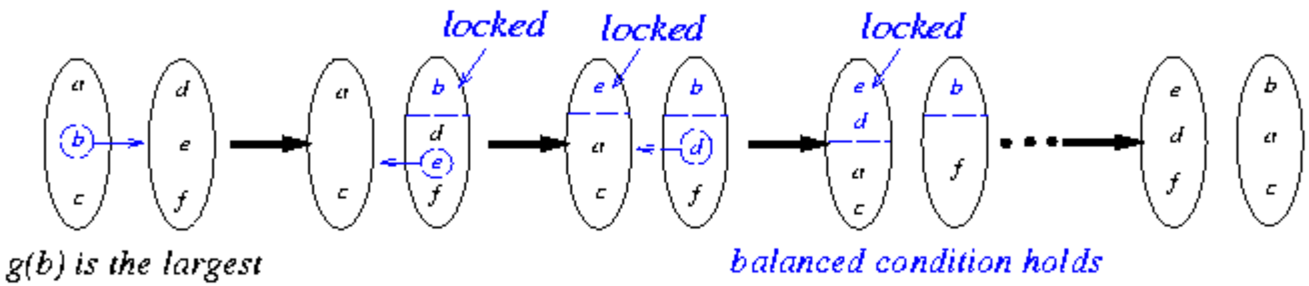
Basic Ideas: Balance and Movement

- Only move a cell at a time, preserving "balance."

$$\frac{|A|}{|A|+|B|} \approx r$$
$$rW - S_{max} \leq |A| \leq rW + S_{max},$$

where $W=|A|+|B|$; $S_{max}=\max_i s(i)$.

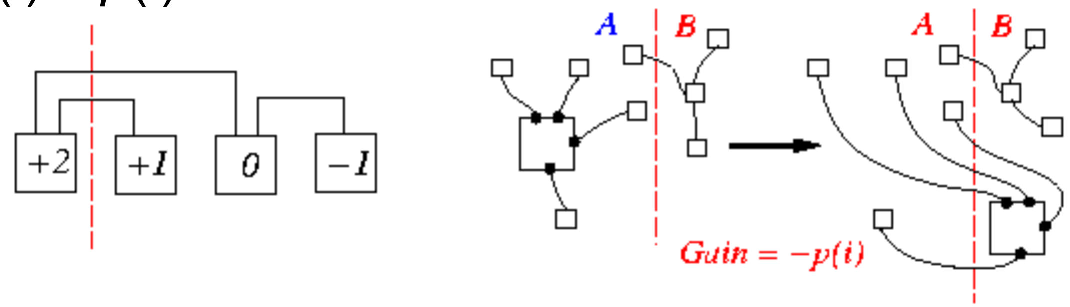
- $g(i)$: gain in moving cell i to the other set, i.e., size of **old** cutset - size of **new** cutset.



- Suppose $\hat{g}_i$'s: $g(b)$, $g(e)$, $g(d)$, $g(a)$, $g(f)$, $g(c)$ and the largest partial sum is $g(b)+g(e)+g(d)$. Then we should move b, e, d ⍰ resulting two sets: {a, c, e, d}, {b, f}.

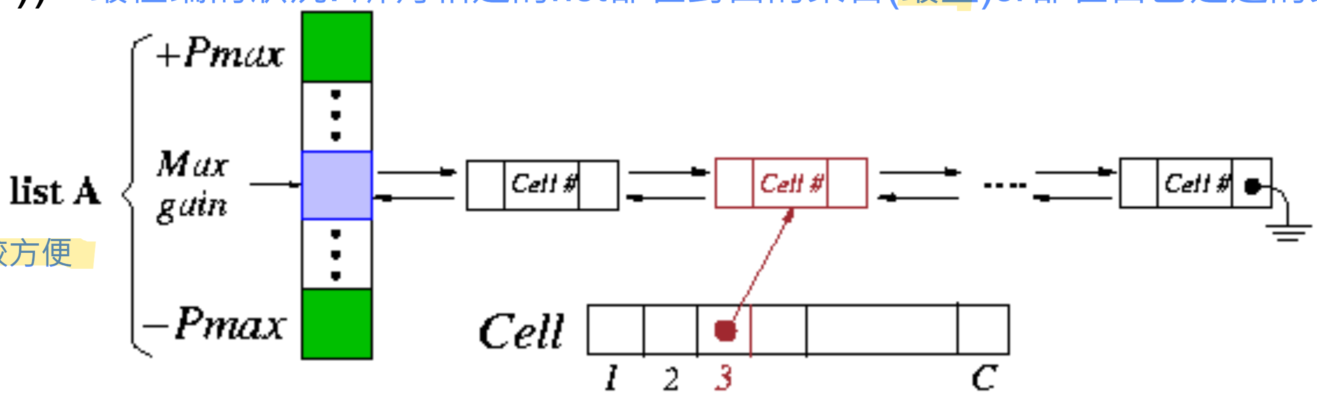
Cell Gains and Data Structure Manipulation

- $-p(i) \le g(i) \le p(i)$



- Two "bucket list" structures, one for set *A* and one for set *B* ($P_{max} = \max_i p(i)$). 最極端的狀況: 所有相連的net都在對面的集合(最正)or都在自己這邊的集合(最負)

listA可以用vector做比較方便



- $O(1)$-time operations: find a cell with Max Gain, remove Cell *i* from the structure, insert Cell *i* into the structure, update $g(i)$ to $g(i)+\Delta$, update the Max Gain pointer.

將cell從bucketlist remove後，有必要再insert嗎?
(當前想法:
方法1.因為移動後都要lock，不如就直接remove後不insert，在該iteration結束後再整個重算bucketlist (似乎比較慢)
方法2.與演算法相同，持續維護bucketlist)

Courtesy of Prof. Y.-W. Chang and H.-M. Chen

Computing Initial Gains of All Free Cells

- Initialization of all cell gains requires $O(P)$ time (efficient algorithm shown below):

$g(i) \leftarrow 0;$
$F \leftarrow$ the "from block" of Cell i;
$T \leftarrow$ the "to block" of Cell i;
for each net n on Cell i **do**
    if $F(n)=1$ **then** $g(i) \leftarrow g(i)+1$;
    if $T(n)=0$ **then** $g(i) \leftarrow g(i)-1$;

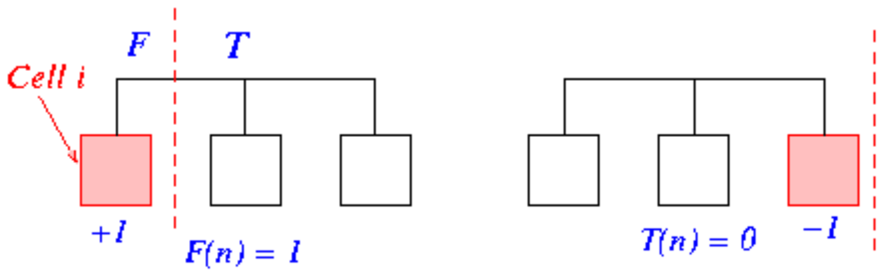
FS(i): # of nets that have cell i as the only cell in From Block

TE(i): # of nets that contain cell i and are entirely located in From Block

gain(i) = FS(i) – TE(i)

F(n)/T(n): # of cells on net n in the From/To Block

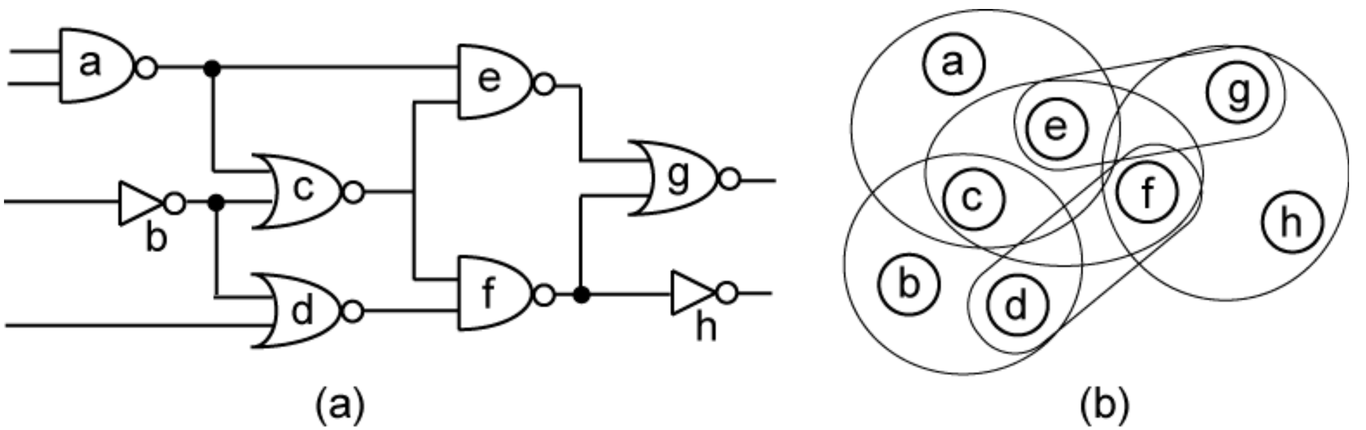
此處應是假設每條線的weight都是1



- Will show: Only need $O(P)$ time to maintain all cell gains in one pass.

Fiduccia-Mattheyses Algorithm

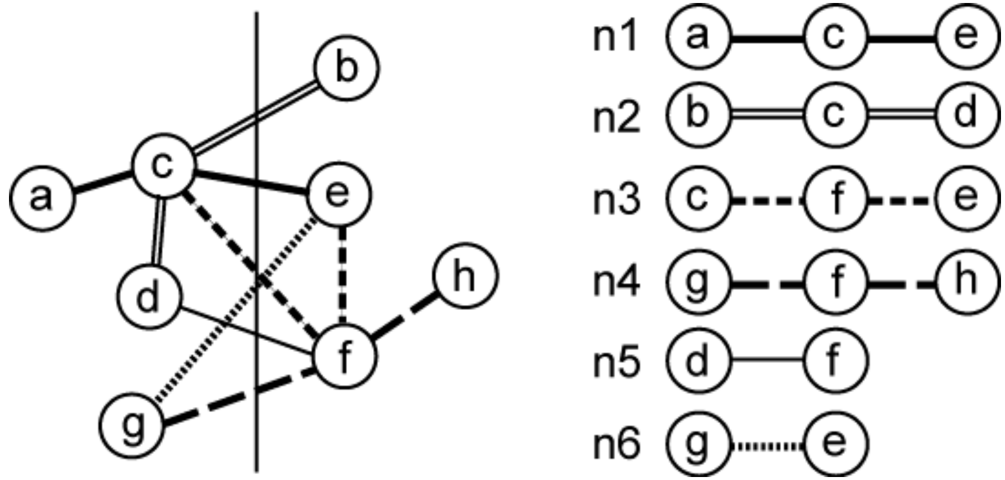
- Perform FM algorithm on the following circuit:
  - Area constraint = [3,5]
  - Break ties in alphabetical order.



Initial Partitioning

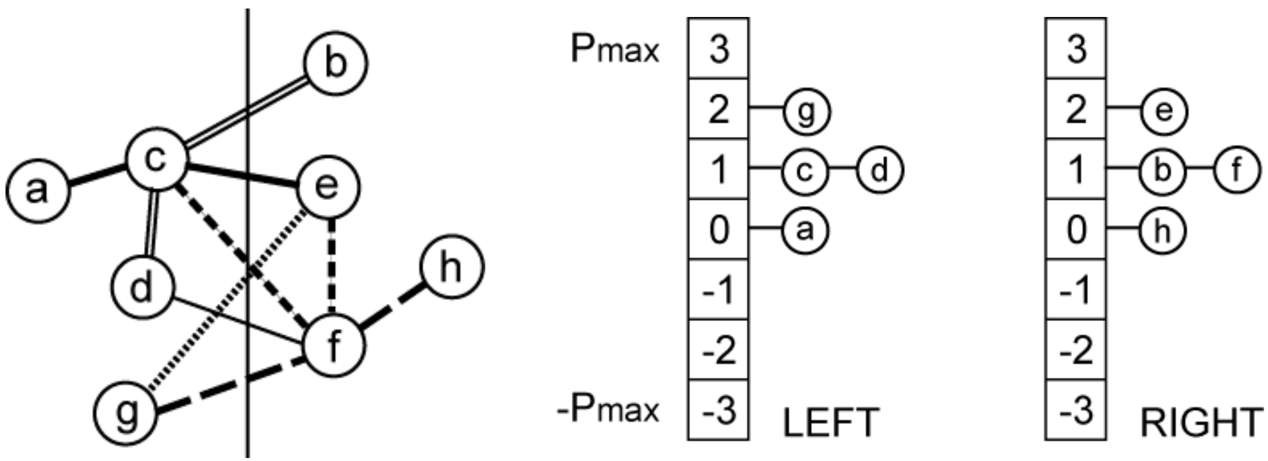
思考: 終止條件
理想狀態下，所有cell被lock後就可以進入下個iteration
但有可能有cell沒有被lock，但是由於balace factor的限制而不能移動，此時需要額外判斷剩下的cell是否皆無法移動，若是這種情況也要視為當前iteration結束

- Random initial partitioning is given.



Gain Computation and Bucket Set Up

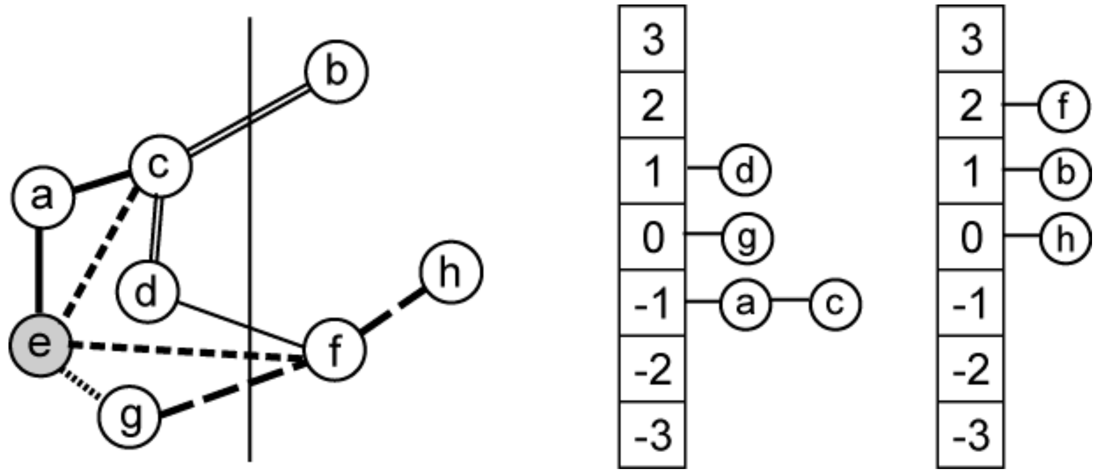
cell c: c is contained in net $n_1 = \{a, c, e\}$, $n_2 = \{b, c, d\}$, and $n_3 = \{c, f, e\}$. n_3 contains c as its only cell located in the left partition, so $FS(c) = 1$. In addition, none of these three nets are located entirely in the left partition. So, $TE(c) = 0$. Thus, $gain(c) = 1$.



FS(x): # of nets that have x as the only cell in LEFT
TE(x): # of nets that contain x and are entirely located in LEFT
gain(x) = FS(x) – TE(x)

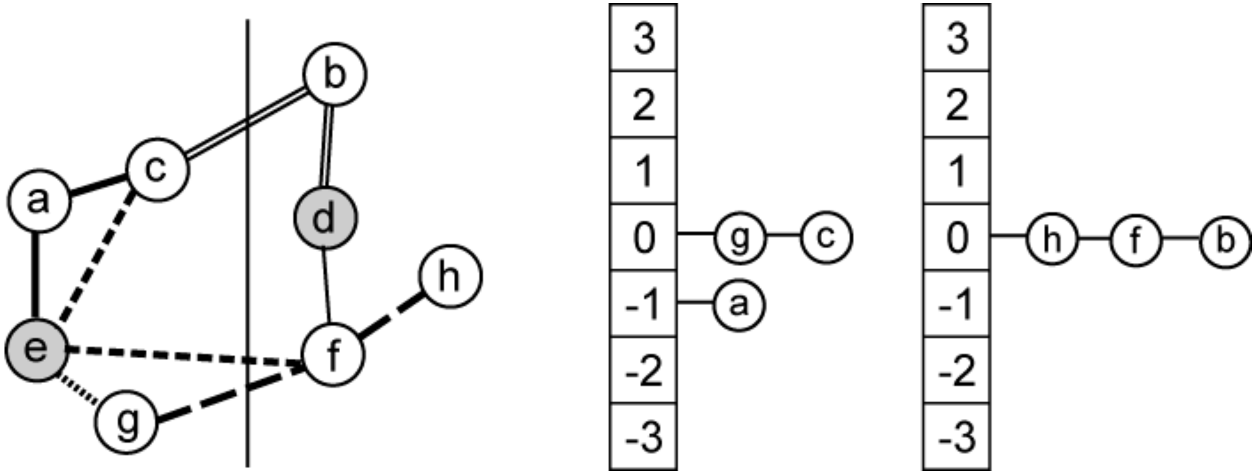
First Move

move 1: From the initial bucket we see that both cell g and e have the maximum gain and can be moved without violating the area constraint. We move e based on alphabetical order. We update the gain of the unlocked neighbors of e, $N(e) = \{a, c, g, f\}$, as follows: $gain(a) = FS(a) - TE(a) = 0 - 1 = -1$, $gain(c) = 0 - 1 = -1$, $gain(g) = 1 - 1 = 0$, $gain(f) = 2 - 0 = 2$.



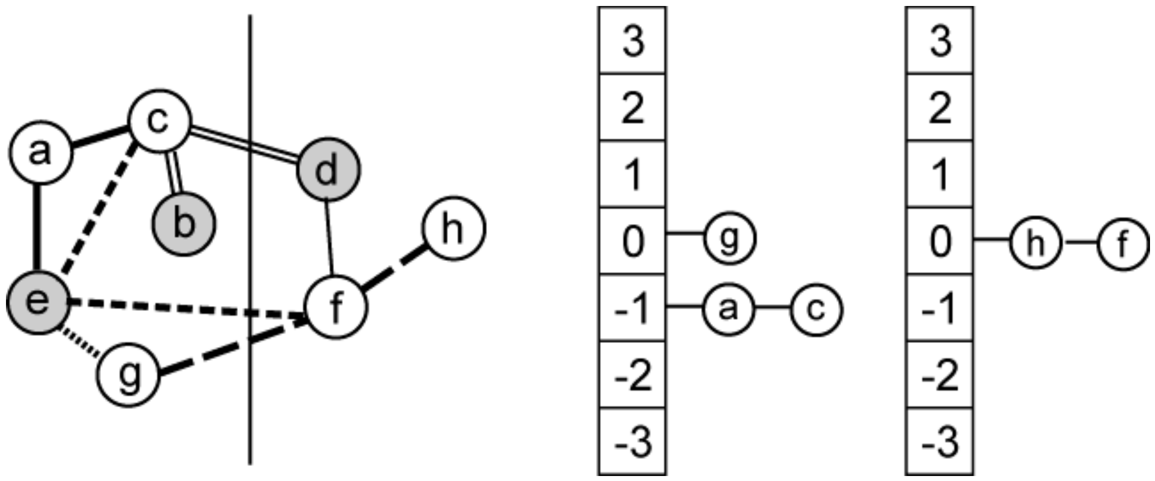
Second Move

move 2: f has the maximum gain, but moving f will violate the area constraint. So we move d. We update the gain of the unlocked neighbors of d, $N(d) = \{b, c, f\}$, as follows: $gain(b) = 0 - 0 = 0$, $gain(c) = 1 - 1 = 0$, $gain(f) = 1 - 1 = 0$.



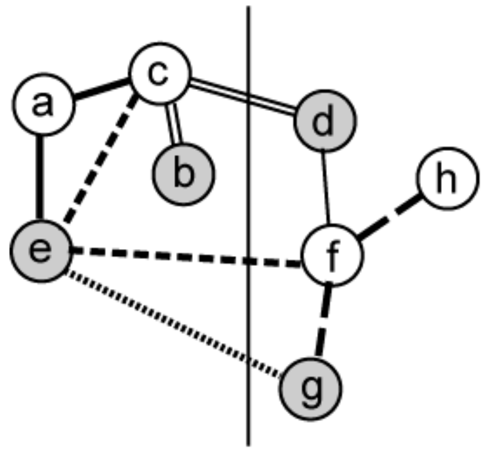
Third Move

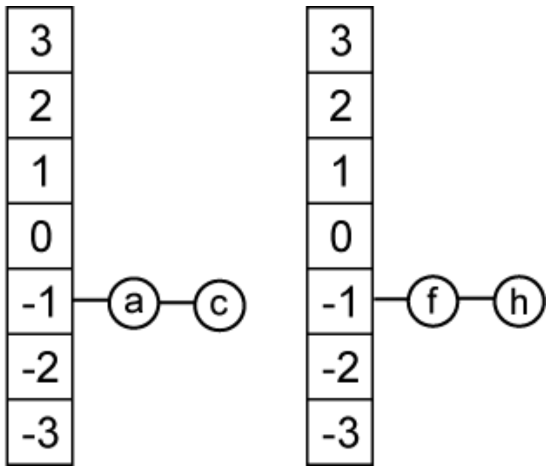
move 3: Among the maximum gain cells $\{g, c, h, f, b\}$, we choose b based on alphabetical order. We update the gain of the unlocked neighbors of b, $N(b) = \{c\}$ as follows: $gain(c) = 0 - 1 = -1$.



Fourth Move

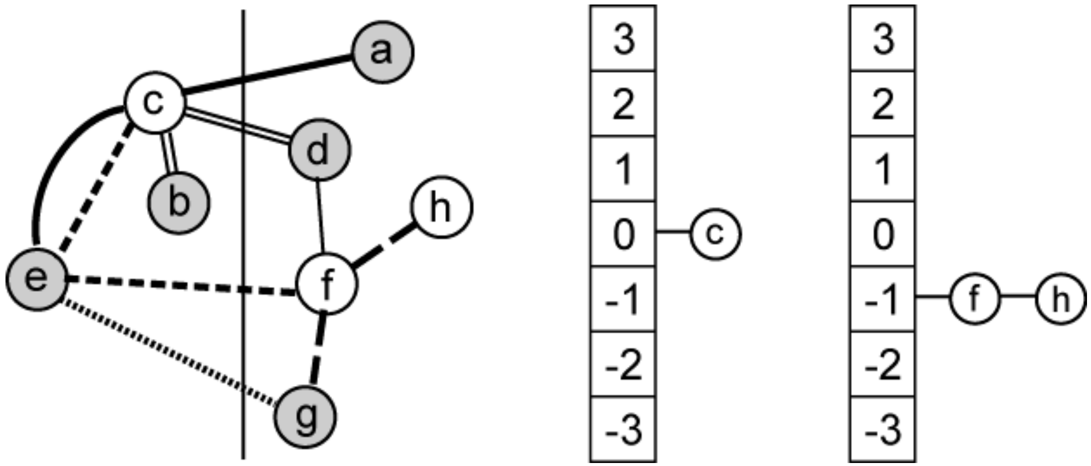
move 4: Among the maximum gain cells $\{g, h, f\}$, we choose g based on the area constraint. We update the gain of the unlocked neighbors of g, $N(g) = \{f, h\}$, as follows: $gain(f) = 1 - 2 = -1$, $gain(h) = 0 - 1 = -1$.





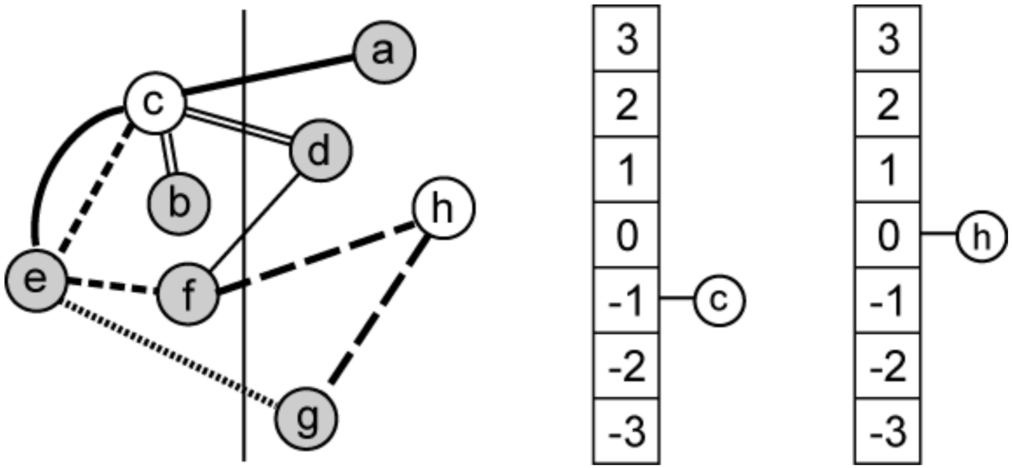
Fifth Move

move 5: We choose a based on alphabetical order. We update the gain of the unlocked neighbors of a, $N(a) = \{c\}$, as follows: $gain(c) = 0 - 0 = 0$.



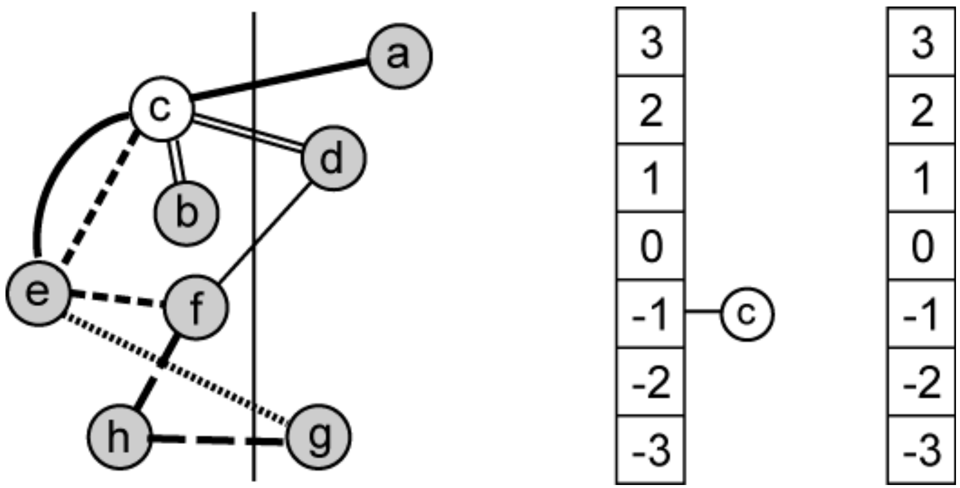
Sixth Move

move 6: We choose f based on the area constraint and alphabetical order. We update the gain of the unlocked neighbors of f, $N(f) = \{h, c\}$, as follows: $gain(h) = 0 - 0 = 0$, $gain(c) = 0 - 1 = -1$.



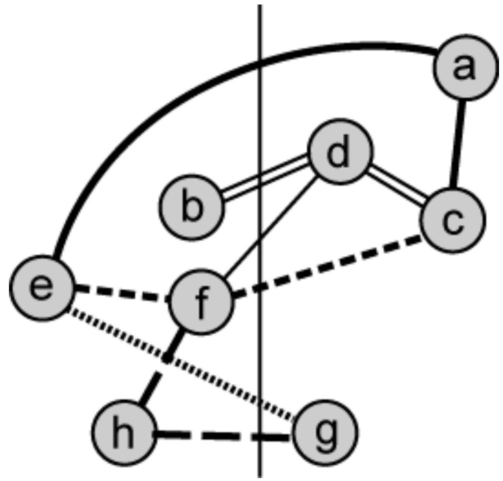
Seventh Move

move 7: We move h. h has no unlocked neighbor.



Last Move

move 8: We move c.



Summary

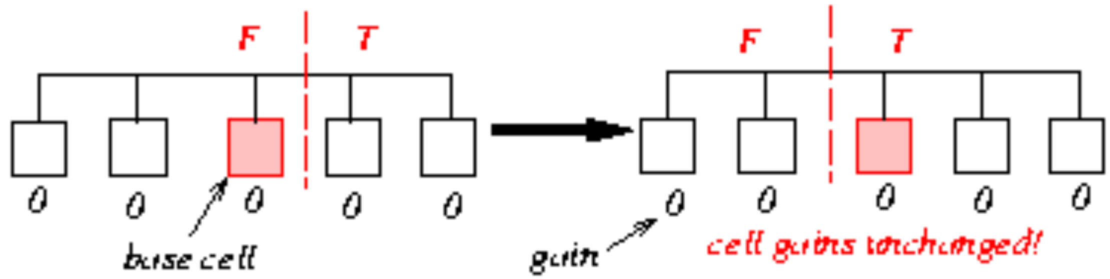
要記錄每一步的移動以及最佳解發生的時間點(trace back時要用)，因此可能需要用到stack來記錄每次移動的cell的from或to

- Found three best solutions.
  - Cutsize reduced from 6 to 3.
  - Solutions after move 2 and 4 are better balanced.

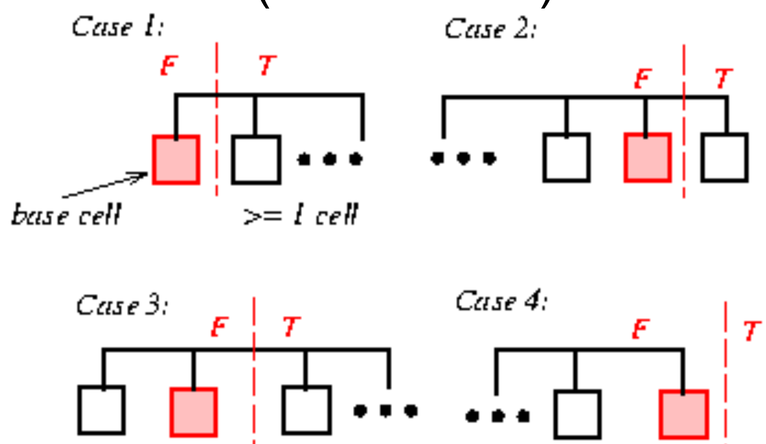
i	cell	$g(i)$	$\sum g(i)$	cutsize
0	-	-	-	6
1	e	2	2	4
2	$\boldsymbol{d}$	**1**	**3**	**3**
3	$\boldsymbol{b}$	**0**	**3**	**3**
4	$\boldsymbol{g}$	**0**	**3**	**3**
5	a	-1	2	4
6	f	-1	1	5
7	h	0	1	5
8	c	-1	0	6

Updating Cell Gains (1/3)

- To update the gains, we only need to look at those nets, connected to the base cell, which are critical **before** or **after** the move.
- **Base cell**: The cell selected for movement from one set to the other.

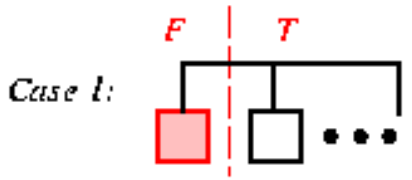


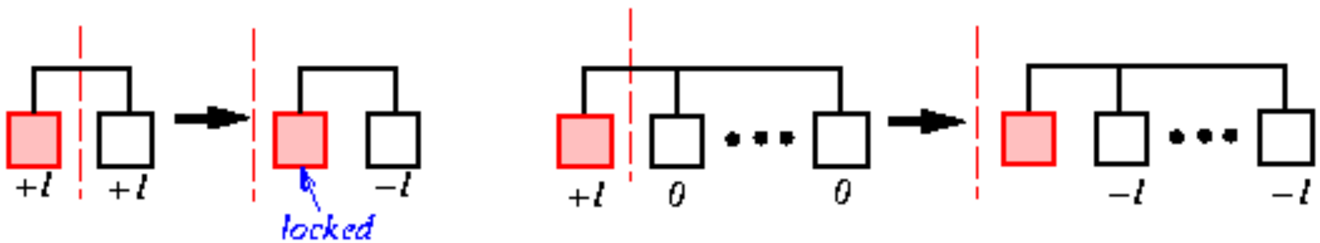
- Consider only the case where the base cell is in the left partition. The other case is similar. (critical nets)

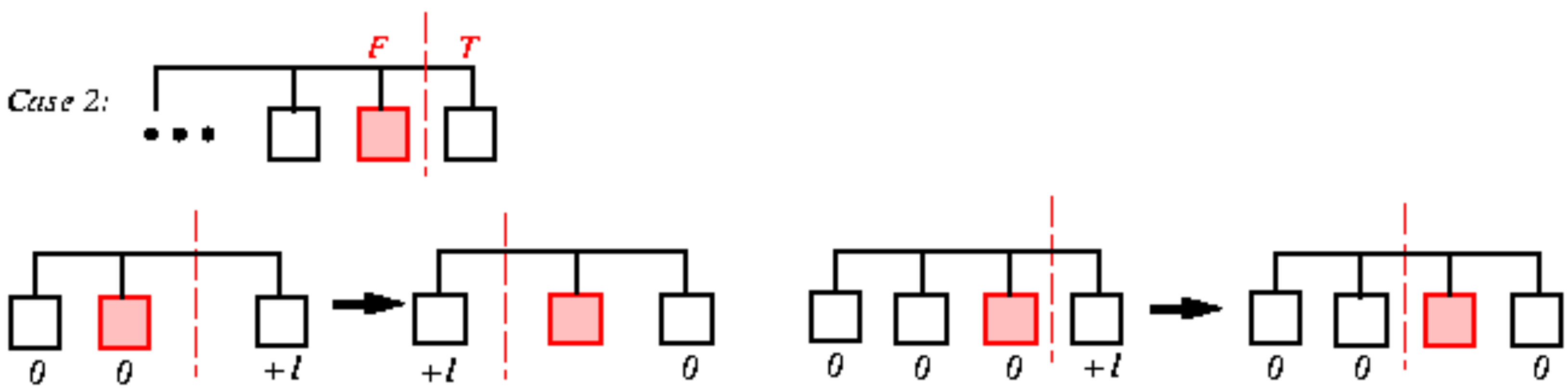


Updating Cell Gains (2/3)

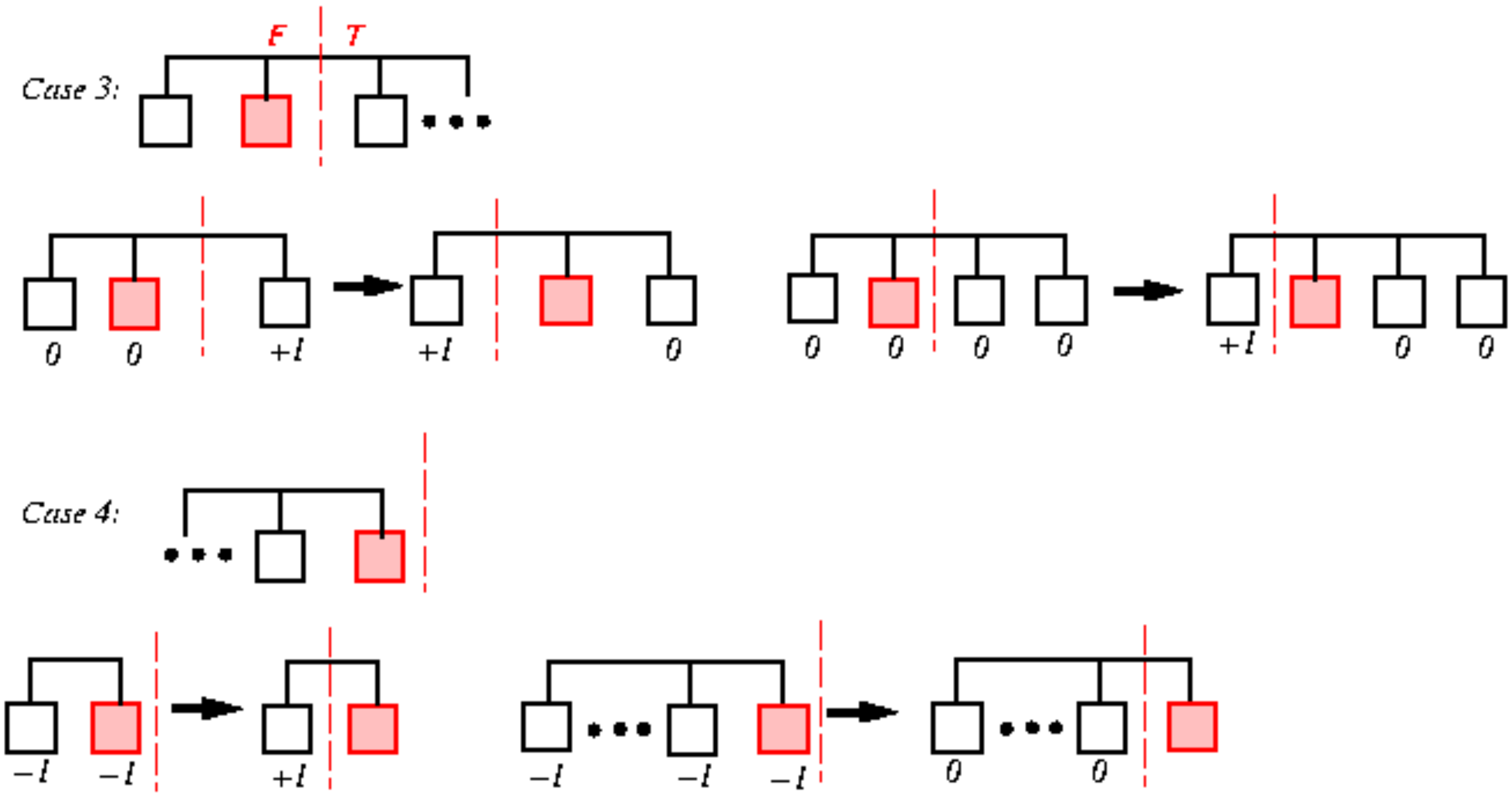
與被移動的cell相連的所有net中的cell都要檢查一下是否為下列四種情況，若是的話就要更新gain







Updating Cell Gains (3/3)

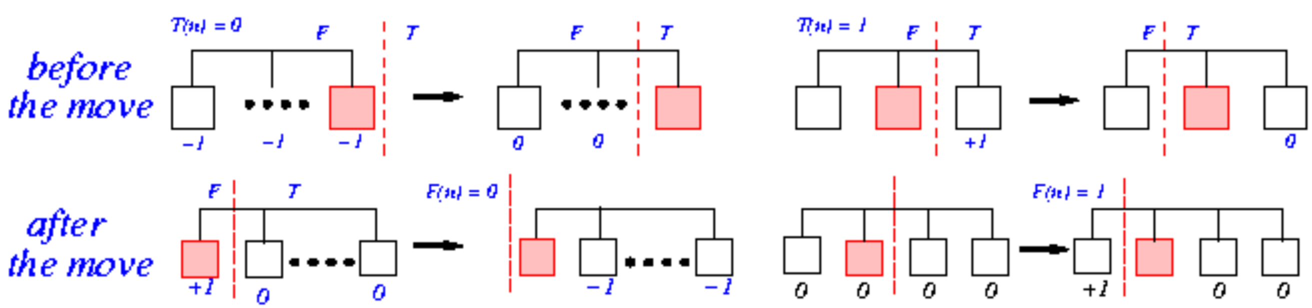


Courtesy of Prof. Y.-W. Chang and H.-M. Chen

Algorithm for Updating Cell Gains

Algorithm: Update_Gain
1 **begin** /* *move base cell and update neighbors' gains* */
2 *F* ← the *Front Block* of the base cell;
3 *T* ← the *To Block* of the base cell;
4 Lock the base cell and complement its block;
5 **for** each net *n* on the base cell **do**
  /* check critical nets before the move */
6   **if** *T*(*n*) = 0 **then** increment gains of all free cells on *n* (case 4)
  else if *T*(*n*)=1 **then** decrement gain of the only *T* cell on *n*,
  if it is free (case 1,2)
  /* change *F*(*n*) and *T*(*n*) to reflect the move */
7   *F*(*n*) ← *F*(*n*) - 1; *T*(*n*) ← *T*(*n*)+1;
  /* *check for critical nets after the move* */
8   **if** *F*(*n*)=0 **then** decrement gains of all free cells on *n* (case 1)
  else if *F*(*n*) = 1 **then** increment gain of the only *F* cell on *n*,
  if it is free (case 3,4)
9 **end**

移動前:
T(n) = 0 --> 這條net所有東西++
T(n) = 1 --> T中唯一的那個--
移動後:
F(n) = 0 --> 這條net所有東西--
F(n) = 1 --> F中唯一的那個++



Complexity of Updating Cell Gains

- To update the cell gains, it takes $O(p(i))$ work for cell i.
  - Find the best cell i to move in O(1)
  - After each move, update gain buckets in $O(p(i))$
- Total time = $p(1)+p(2)+\ldots+p(C) = O(P)$.

F-M Algorithm

- Start with any initial partitions A and B
- A pass is described below: (moving each vertex exactly once)
  1. for i := 1 to 2n do

     From the unlocked (unmoved) vertices,

        Choose a vertex V such that D_v is largest and moving V will not violate the area constraint

     Move V. Lock V.

     Let $g_i = D_v$
  2. Find the k s.t. $G = g_1 + g_2 + \ldots + g_k$ is maximized
  3. Switch the first k vertices
- Repeat the pass until there is no improvement

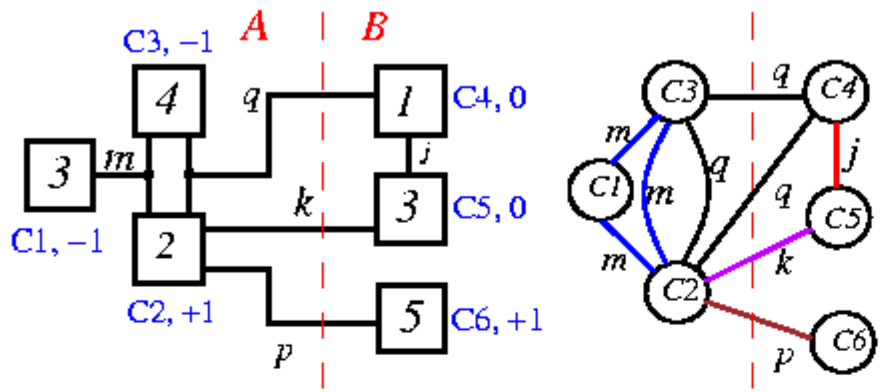
  (max $G \leq 0$)

From Prof. Chu in Iowa St. U.

考試可能會考

Another F-M Heuristic Rundown (1/3)

s(c1)=3
s(c2)=2
s(c3)=4
s(c4)=1
s(c5)=3
s(c6)=5

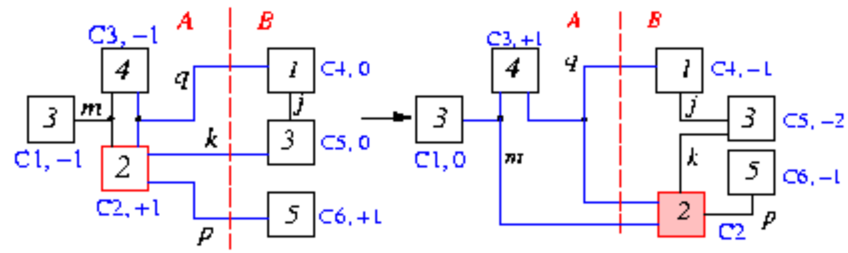


- Computing cell gains: $F(n) = 1 \rightarrow g(i) + 1$; $T(n)=0 \rightarrow g(i) - 1$

Cell	m F	m T	q F	q T	k F	k T	p F	p T	j F	j T	$g(i)$
c1	0	−1									−1
c2	0	−1	0	0	+1	0	+1	0			+1
c3	0	−1	0	0							−1
c4			+1	0					0	−1	0
c5					+1	0			0	−1	0
c6							+1	0			+1

- Balanced criterion: $r|V| - S_{max} \leq |A| \leq r|V| + S_{max}$. Let $r = 0.4$, $|A| = 9$, $|V|= 18$, $S_{\max} = 5$, $r|V|=7.2 \rightarrow$ Balanced: $2.2 \leq 9 \leq 12.2$!
- maximum gain: c_2 and balanced: $2.2 \leq 9\text{-}2 \leq 12.2 \rightarrow$ Move c_2 from A to B (use size criterion if there is a tie).

F-M Heuristic Example (2/3)



- Changes in net distribution:

更新每條net的cell_in_A, cell_in_B，以這個例子來說F = set A, T = set B

Net	Before move F	Before move T	After move F'	After move T'
k	1	1	0	2
m	3	0	2	1
q	2	1	1	2
p	1	1	0	2

- Updating cell gains on critical nets (run Algorithm Update_Gain):

Cells	Gains due to $T(n)$: k	m	q	p	Gain due to $F(n)$: k	m	q	p	Gain changes: Old	New
c_1		+1							−1	0
c_3		+1					+1		−1	+1
c_4			−1						0	−1
c_5	−1				−1				0	−2
c_6				−1				−1	+1	−1

去trace update_gain的pseudo code就好

技巧: 直接一次看一條net，跟寫程式的時候一樣，不要一個cell一個cell看

- Maximum gain: c_3 and balanced! ($2.2 \le 7\text{-}4 \le 12.2$) $\rightarrow$ Move c_3 from A to B (use size criterion if there is a tie).

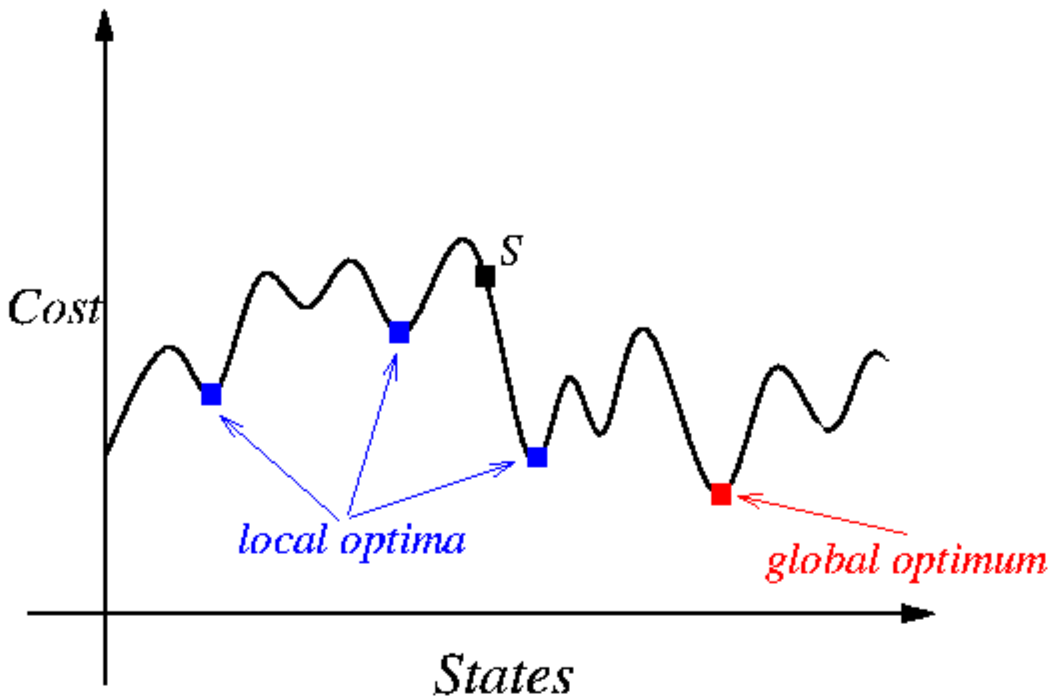
Summary of this Example (3/3)

Step	Cell	Max gain	$\lvert A \rvert$	Balanced?	Locked cell	A	B
0	-	-	9	-	$\emptyset$	1, 2, 3	4, 5, 6
1	c_2	$+1$	7	yes	c_2	1, 3	2, 4, 5, 6
2	c_3	$+1$	3	yes	c_2, c_3	1	2, 3, 4, 5, 6
3	c_1	$+1$	0	no	-	-	-
$3'$	c_6	-1	8	yes	c_2, c_3, c_6	1, 6	2, 3, 4, 5
4	c_1	$+1$	5	yes	c_1, c_2, c_3, c_6	6	1, 2, 3, 4, 5
5	c_5	-2	8	yes	c_1, c_2, c_3, c_5, c_6	5, 6	1, 2, 3, 4
6	c_4	0	9	yes	all cells	4, 5, 6	1, 2, 3

- $\hat{g_1} = 1, \hat{g_2} = 1, \hat{g_3} = -1, \hat{g_4} = 1, \hat{g_5} = -2, \hat{g_6} = 0$ ⇒ Maximum partial sum G_k = +2, k = 2 or 4.
- Since k=4 results in a better balanced ⇒ Move c_1, c_2, c_3, c_6 ⇒ A={6}, B={1, 2, 3, 4, 5}.
- **Repeat the whole process until new $G_k \le 0$.**

Simulated Annealing

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



Courtesy of Prof. Y.-W. Chang and H.-M. Chen

Simulated Annealing Basics

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

狀態變好

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

狀態變差

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

Generic Simulated Annealing Algorithm (from Metropolis 1953)

begin
Get an initial solution S;
Get an initial temperature $T > 0$;
while not yet "frozen" **do**
  for $1 \leq i \leq P$ **do**
    Pick a random neighbor S' of S;
    $\Delta \leftarrow cost(S') - cost(S)$;
    /* downhill move */
    if $\Delta \leq 0$ **then** $S \leftarrow S'$
    /* uphill move */
    如果新的結果比原先結果來得差，就會計算機率判斷是否要將新的結果取代原先的結果
    if $\Delta > 0$ **then** $S \leftarrow S'$ with probability $e^{-\frac{\Delta}{T}}$;
$T \leftarrow rT$; /* reduce temperature */
return S
end

Basic Ingredients for Simulated Annealing

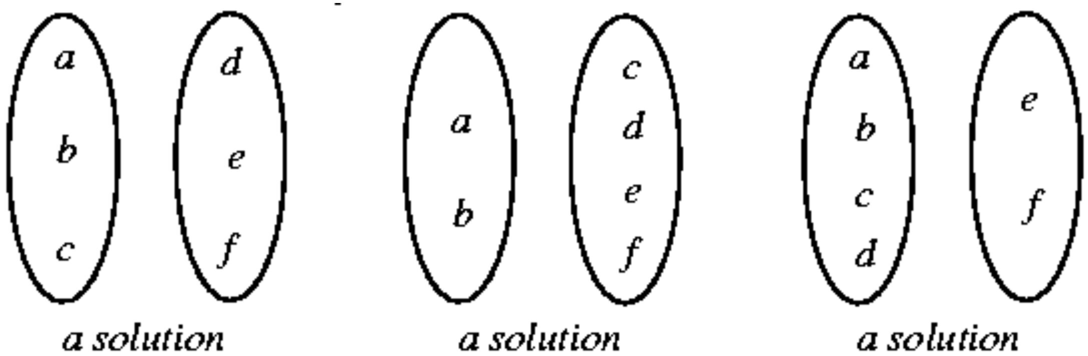
- Analogy: 使用SA之前要先定義好下列的參數

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

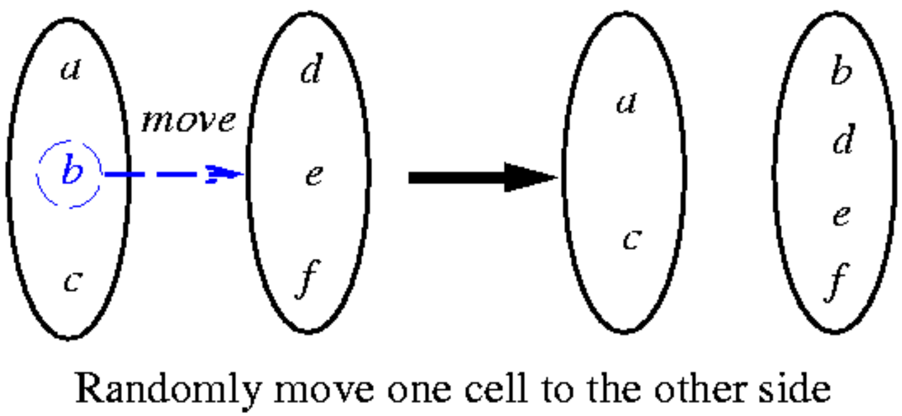
- Basic Ingredients for Simulated Annealing:
  - **Solution space**
  - **Neighborhood structure**
  - **Cost function**
  - **Annealing schedule**

Partition by Simulated Annealing

- **Solution space:** set of all partitions

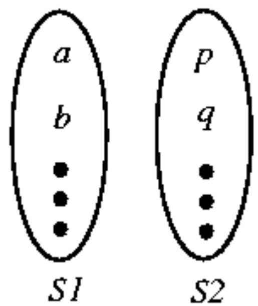


- **Neighborhood structure:**



Partition by Simulated Annealing (cont)

- **Cost function:** $f = C + \lambda B$
  - C: the partition cost as used before.
  - B: a measure of how balance the partition is
  - λ: a constant



$$B = (|S1| - |S2|)^2$$

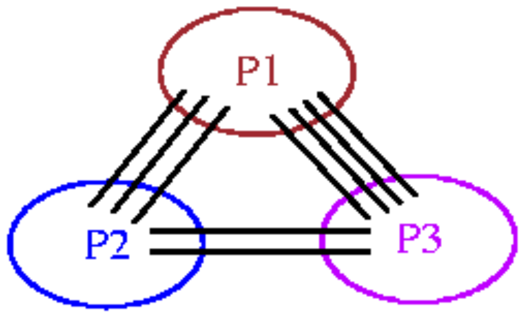
SA似乎也有很多細節..?
可以多去研究一下

- **Annealing schedule:**
  - $T_n = r^n T_0$, $r = 0.9$.
  - At each temperature, either
    1. there are 10 accepted moves/cell on the average, or
    2. # of attempts $\geq 100 \times$ total # of cells.
  - The system is "frozen" if very low acceptances at 3 consecutive temperatures.

Network Flow Based Partitioning

- Yang and Wong, "Efficient network-flow based min-cut balanced partitioning," ICCAD-94.
  - Based on max-flow min-cut theorem.

這個演算法在解決partition的問題時並
沒有很有效率(performance 也不好)
此演算法通常用於其他地方



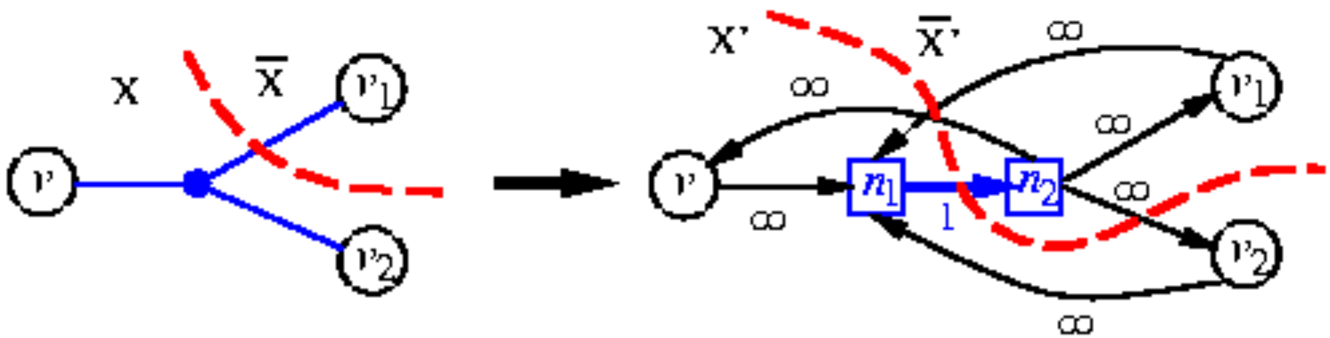
- Gate replication for partitioning: Yang and Wong, ICCAD-95.
- Multi-way partitioning with area and pin constraints: Liu and Wong, ISPD-97.
- Multi-resource partitioning: Liu, Zhu, and Wong, FPGA-98.

Network Flow Based Partitioning

- Why was the technique not wisely used in partitioning?
  - Works on graphs, not hypergraphs.
  - Results in unbalanced partitions; repeated min-cut for balance: $|V|$ max-flows, time-consuming!
- Yang & Wong, ICCAD-94 (also in *The Best of ICCAD*)
  - Exact **net** modeling by flow network.
  - Optimal algorithm for min-net-cut bipartition (unbalanced).
  - Efficient implementation for repeated min-net-cut: same asymptotic time complexity as **one** max-flow computation
    - Through the recycling of augmenting paths from the previous iterations

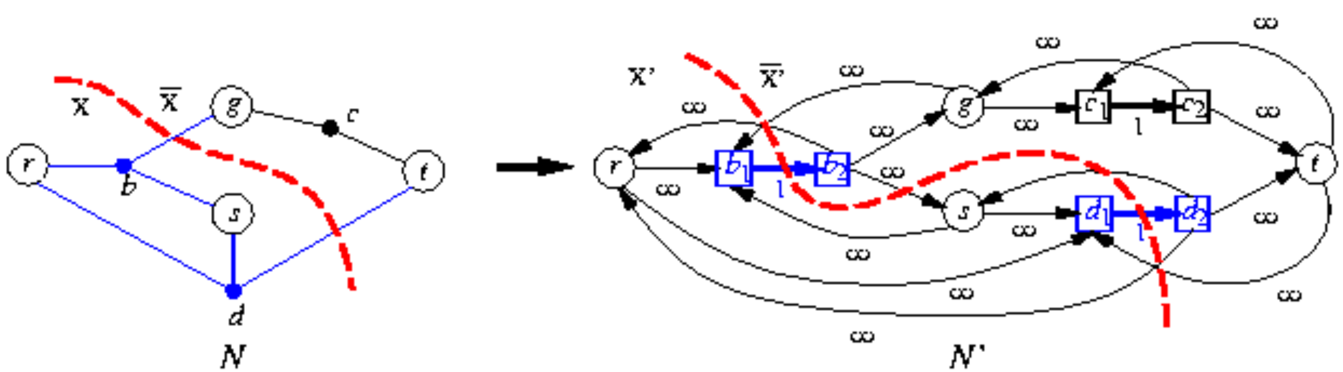
Min-Net-Cut Bipartition (not balanced)

- Net modeling by flow network:



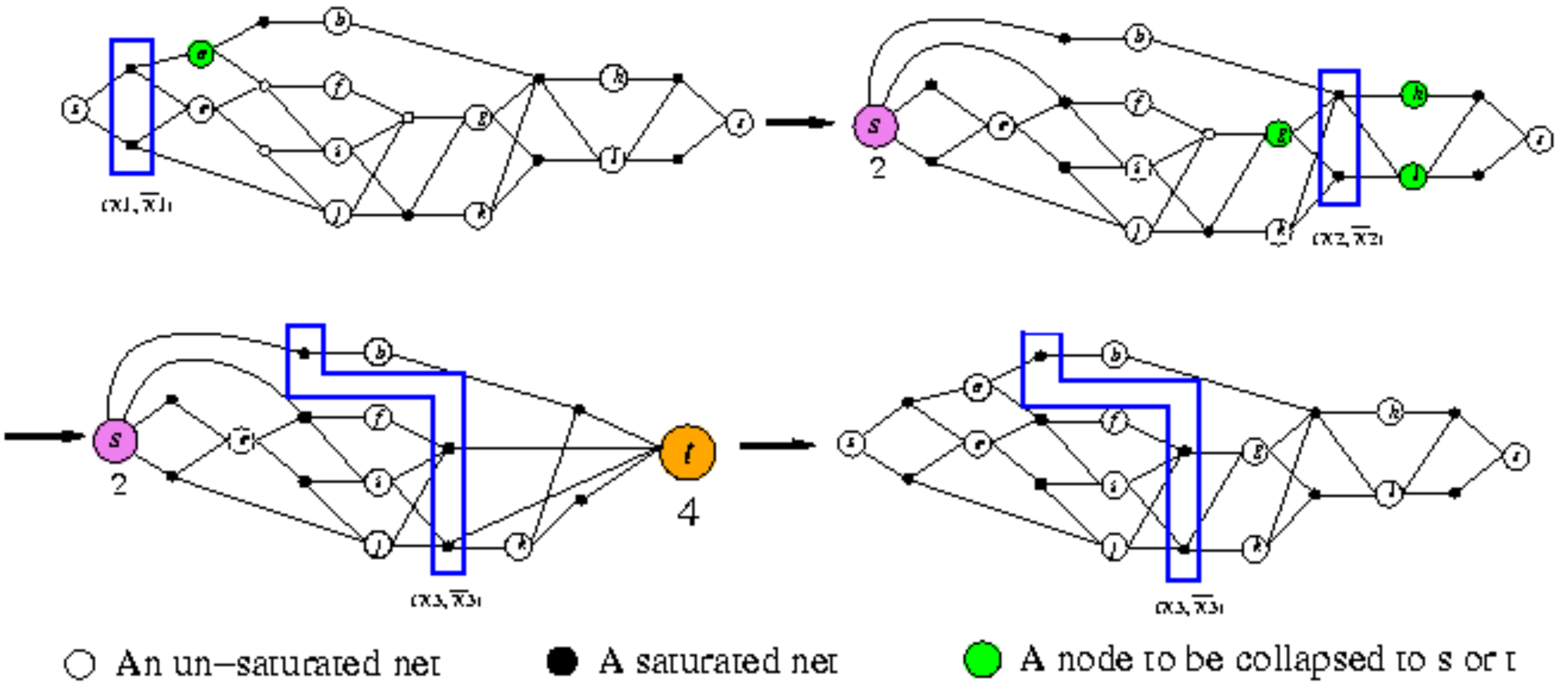
- A min-net-cut ($X, \bar{X}$) in *N* ⇔ A min-capacity-cut ($X', \bar{X}'$) in *N*'.
- Size of flow network: $|V'| \leq 3|V|$, $|E'| \leq 2|E| + 3|V|$.
- Time complexity: O(min-net-cut-size) × $|E| = O(|V||E|)$.

Time for finding augmenting path



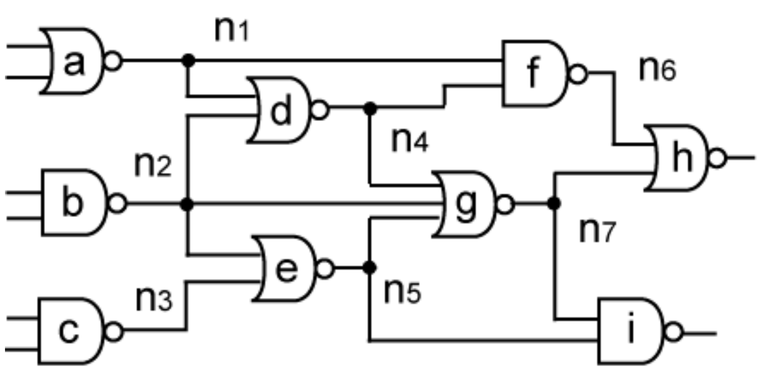
Repeated Min-Cut for Balanced Bipartition (Flow-Balanced-Bipartition, FBB)

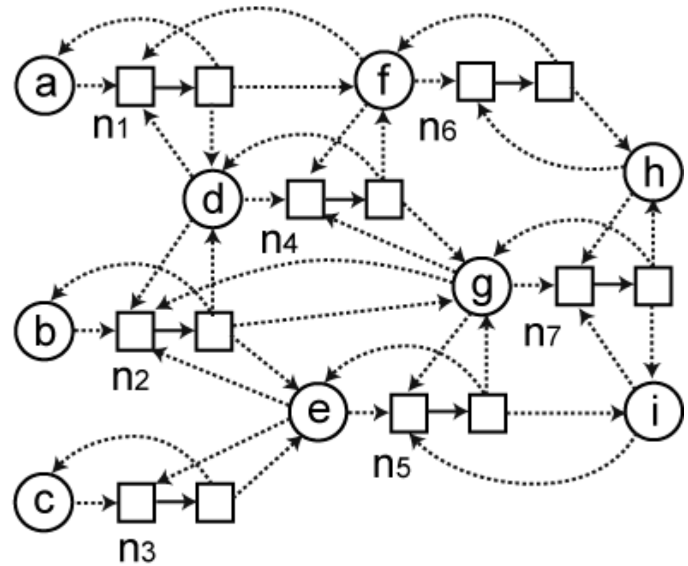
- For most *r*-balanced min-cut bipartition problem (different from min-cut balanced bipartition)
- Allow component weights to deviate from $(1 - \varepsilon)rW$ to $(1 + \varepsilon)rW$.
- Repeatedly compute max-flow: very time-consuming -> incremental flow



Network Flow-based Bipartitioning

- Perform flow-based bipartitioning under:
  - Area constraint [4,5]
  - Source = *a*, sink = *i*
  - Break ties alphabetically

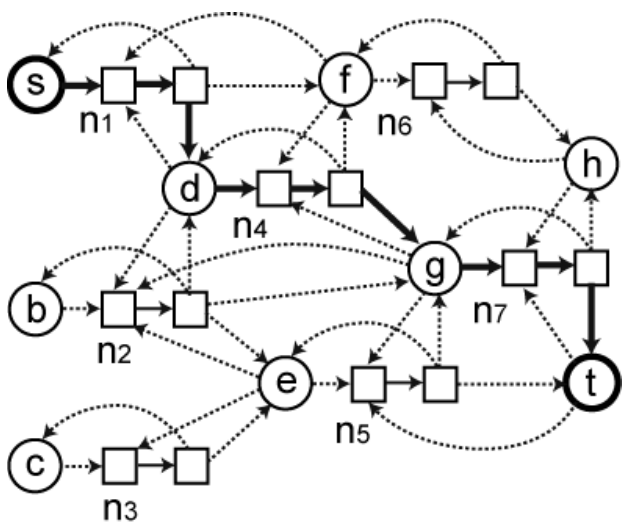




First Max-Flow and Its Cut

Figure 2.25 shows a maximum flow value of 1 (= not unique). Net n_1, n_4, and n_7 are saturated and define the partitioning solutions shown in Table 2.7. For example, removal of n_1 leads to a a-i mincut. But, removal of n_4 or n_7 does not lead to a a-i mincut. Thus, we cut n_1 and obtain the following solution:

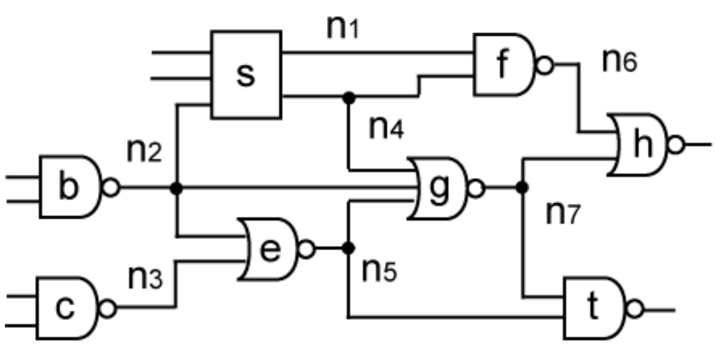
$$P_s = \{s\}, P_t = \{b, c, d, e, f, g, h, t\}$$



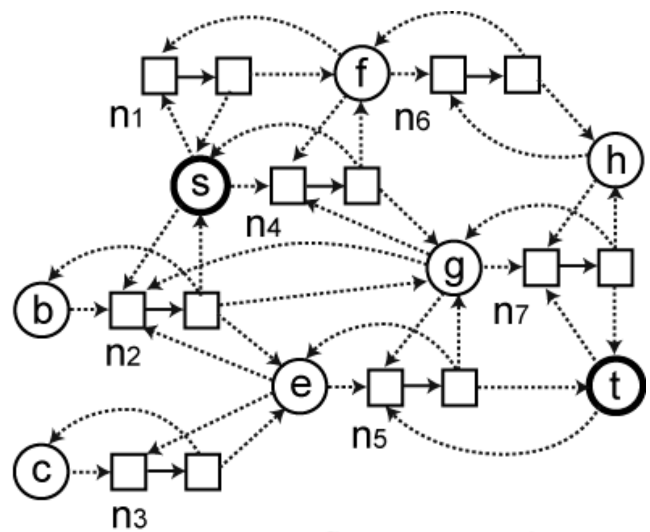
cut net	source partition	sink partition
n_1	s	b, c, d, e, f, g, h, t
n_4	no cut	no cut
n_7	no cut	no cut

First Node Merging

We chose $P_s = \{s\}, P_t = \{b, c, d, e, f, g, h, t\}$. Since the area of P_s is smaller than the lower bound of 4, we choose a node from the sink side. In this case, the node should be contained in the cut net n_1. Since $n_1 = \{a, d, f\}$, we choose d based on alphabetical order.



circuit after merging *s* and *d*

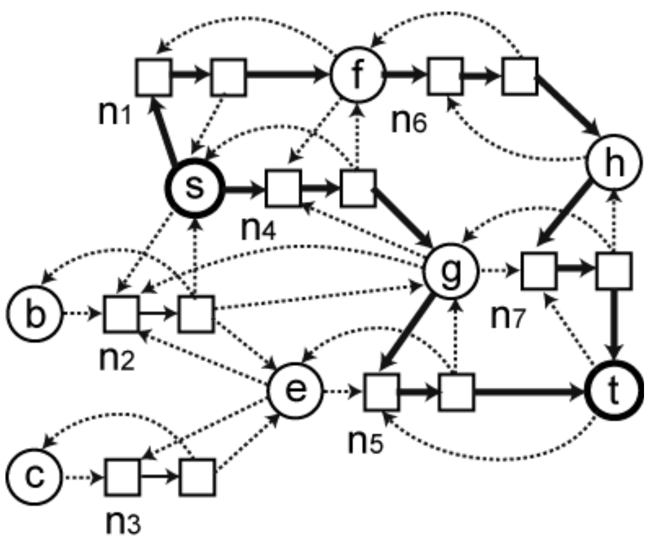


flow-network

Second Max-Flow and Its Cut

Figure 2.27 shows the augmenting paths, and the maximum flow (value = 2). Net n_1, n_6, n_7, n_4, and n_5 are saturated and define the partitioning solutions shown in Table 2.8. Since the max-flow value is 2, our cutset contains two nets n_7 and n_5. This results in:

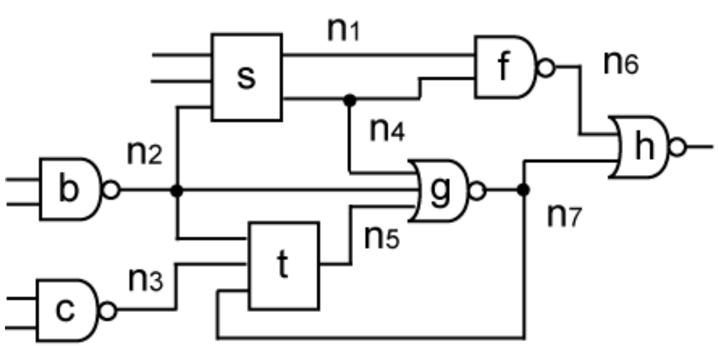
$$P_s = \{s, b, c, e, f, g, h\}, P_t = \{t\}$$



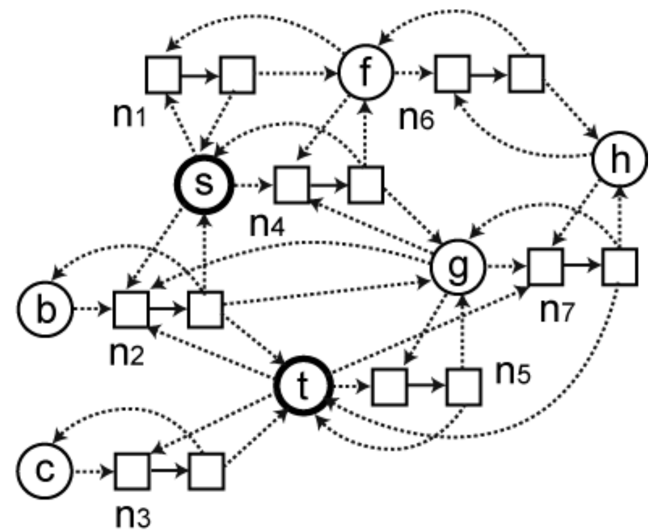
cut net	source partition	sink partition
n_1, n_4	no cut	no cut
n_1, n_5	no cut	no cut
n_6, n_4	no cut	no cut
n_6, n_5	no cut	no cut
n_7, n_4	no cut	no cut
n_7, n_5	s, b, c, e, f, g, h	t

Second Node Merging

We chose $P_s = \{s, b, c, e, f, g, h\}, P_t = \{t\}$. Since the area of source partition is larger than the upper bound of 5 above, we choose a node from the source side. The set of nodes contained in n_7, n_5 and partitioned into the source side include $\{g, h, e\}$. Thus, we choose e to merge with t based on alphabetical order.



circuit after merging *e* and *t*

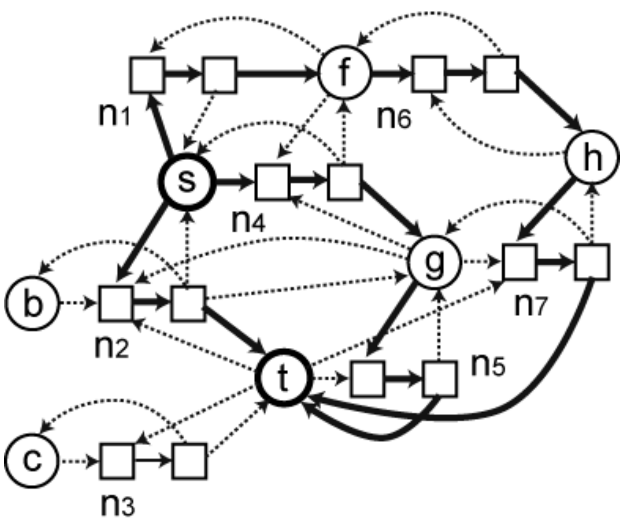


flow-network

Third Max-Flow and Its Cut

Figure 2.29 shows the augmenting paths, and the maximum flow (value = 3). Net n_1, n_6, n_7, n_4, n_5, and n_2 are saturated and define the partitioning solutions shown in Table 2.9. We found three balanced partitioning solutions with the cutsize of 3.

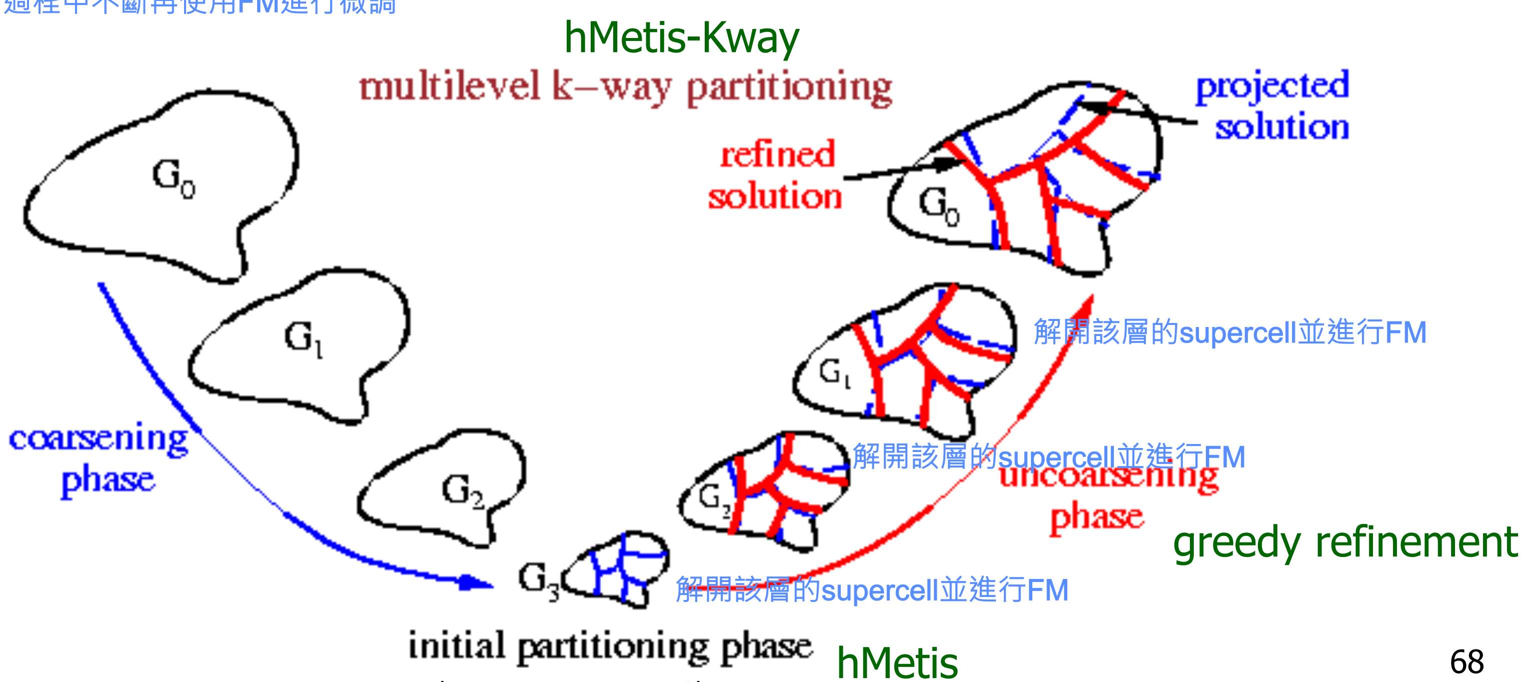
$$(\{a,b,d,f\},\{c,e,g,h,i\}),(\{a,b,d,f,h\},\{c,e,g,i\})$$
$$(\{a,d,f,g,h\},\{b,c,e,i\})$$



cut net	source partition	sink partition
n_1, n_4, n_2	s, b	c, t, g, f, h
n_1, n_5, n_2	no cut	no cut
n_6, n_4, n_2	s, b, f	c, t, g, h
n_6, n_5, n_2	no cut	no cut
n_7, n_4, n_2	s, f, h, b	c, t, g
n_7, n_5, n_2	s, f, g, h	c, t, b

Large-scale Circuit Partitioning

- Keys for large-scale circuits: **clustering, multilevel**
- **Clustering:** Reduce the problem size by grouping highly connected components and treat them as a super node.
- **Multilevel optimization**
  - **Coarsening/clustering:** Recursively clusters the instance until its size is smaller than a given threshold. 把關聯性很強的cell(比如說在同一條net上的cell)包裝成一個super cell，並且多次迭帶
  - **Uncoarsening/partitioning:** Declusters the instance while applying a partitioning refinement algorithm (e.g., F-M or greedy approach).
    解開剛剛包裝的supercell，並且在uncoarsening的過程中不斷再使用FM進行微調



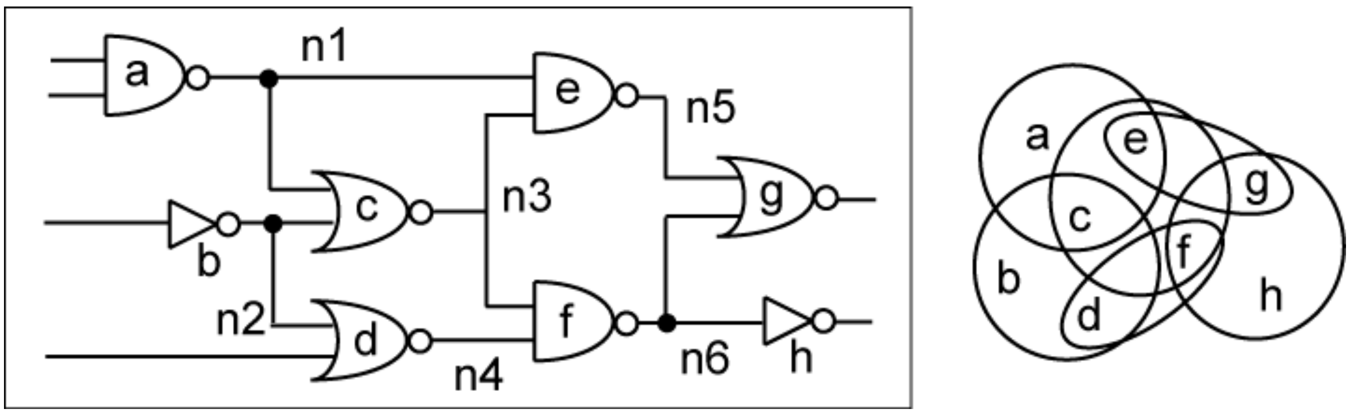
最牛的partition

hMetis (1997) Multilevel Coarsening

- hMetis algorithm utilizes three algorithms to compute the multi-level cluster hierarchy
  - Edge coarsening (EC)
  - Hyperedge coarsening (HEC)
  - Modified hyperedge coarsening (MHEC)

Multi-level Coarsening Algorithms

- Perform Edge Coarsening (EC)
  - Visit nodes and break ties in alphabetical order
  - Explicit clique-based graph model is not necessary



1. Edge Coarsening

|n1|: n1此條net中cell的數量

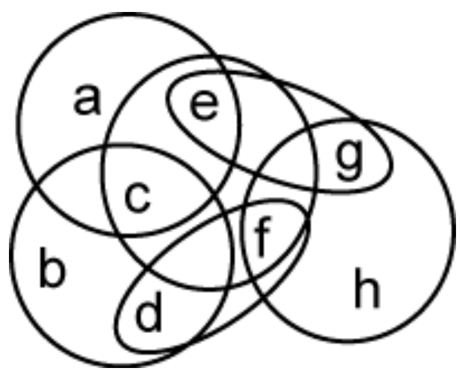
(a) visit a: Note that a is contained in n_1 only. So, $neighbor(a) = \{c, e\}$. The weight of $(a, c) = 1/(|n_1| - 1) = 0.5$. The weight of $(a, e) = 1/(|n_1| - 1) = 0.5$. Thus, we break the tie based on alphabetical order. So, a merges with c. We form $C_1 = \{a, c\}$ and mark a and c.

權重經過比較後，圈選權重較大的cell

(b) visit b: Note that b is contained in n_2 only. So, $neighbor(b) = \{c, d\}$. Since c is already marked, b merges with d. We form $C_2 = \{b, d\}$ and mark b and d.

(c) since c and d are marked, we skip them.

一次圈一個新的cell(不會
一次圈到兩個)
也許會圈很多次?



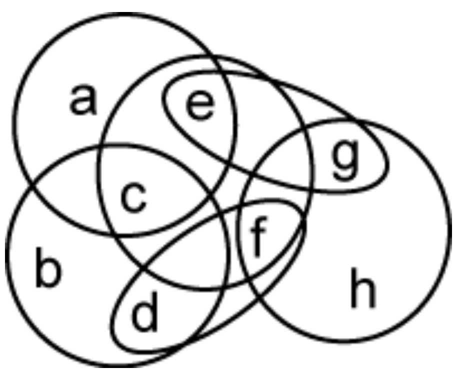
cluster	nodes
C_1	$\{a, c\}$
C_2	$\{b, d\}$
C_3	$\{e, g\}$
C_4	$\{f, h\}$

Edge Coarsening (cont)

(d) visit e: the unmarked neighbors of e are g and f. We see that $w(e, g) = 1$ and $w(e, f) = 0.5$. So, e merges with g. We form $C_3 = \{e, g\}$ and mark e and g.

(e) visit f: Node f is contained in n_3, n_4, and n_6. So, $neighbor(f) = \{c, d, e, g, h\}$. But, the only unmarked neighbor is h. So, f merges with h. We form $C_4 = \{f, h\}$ and mark f and h.

(f) since g and h are marked, we skip them.



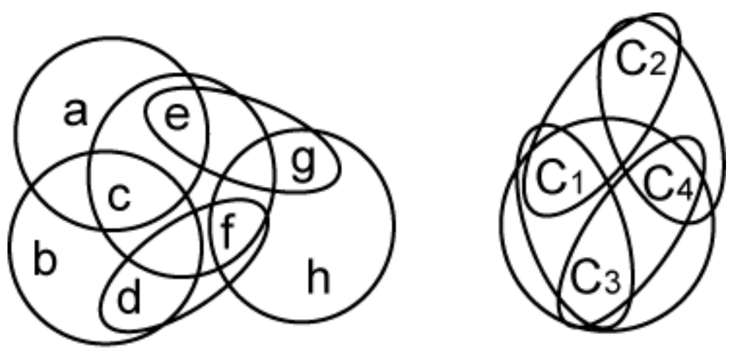
cluster	nodes
C_1	$\{a, c\}$
C_2	$\{b, d\}$
C_3	$\{e, g\}$
C_4	$\{f, h\}$

Obtaining Clustered-level Netlist

- # of nodes/hyperedges reduced: 4 nodes, 5 hyperedges

net	gate-level	cluster-level	final
n_1	$\{a, c, e\}$	$\{C_1, C_1, C_3\}$	$\{C_1, C_3\}$
n_2	$\{b, c, d\}$	$\{C_2, C_1, C_2\}$	$\{C_1, C_2\}$
n_3	$\{c, e, f\}$	$\{C_1, C_3, C_4\}$	$\{C_1, C_3, C_4\}$
n_4	$\{d, f\}$	$\{C_2, C_4\}$	$\{C_2, C_4\}$
n_5	$\{e, g\}$	$\{C_3, C_3\}$	$\emptyset$
n_6	$\{f, g, h\}$	$\{C_4, C_3, C_4\}$	$\{C_3, C_4\}$

cluster	nodes
C_1	$\{a, c\}$
C_2	$\{b, d\}$
C_3	$\{e, g\}$
C_4	$\{f, h\}$



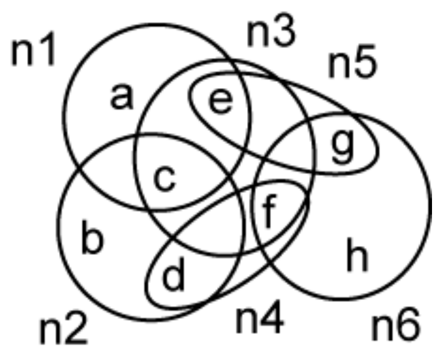
(a) (b)

2. Hyperedge Coarsening

- Initial setup

將所有net用hyperedge進行排序
如何算出hyperedge?

  - Sort hyper-edges in increasing size: n_4, n_5, n_1, n_2, n_3, n_6
  - Unmark all nodes



Hyperedge Coarsening

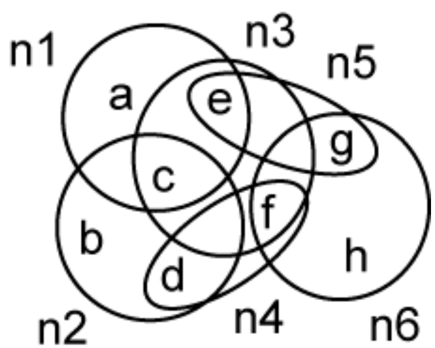
從較小的net開始看

(a) visit $n_4 = \{d, f\}$: since d and f are not marked yet, we form $C_1 = \{d, f\}$ and mark d and f.

(b) visit $n_5 = \{e, g\}$: since e and g are not marked yet, we form $C_2 = \{e, g\}$ and mark e and g.

(c) visit $n_1 = \{a, c, e\}$: since e is already marked, we skip n_1.

直接跳過不圈
-->a跟c先單獨擺放



cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a\}$
C_4	$\{b\}$
C_5	$\{c\}$
C_6	$\{h\}$

Hyperedge Coarsening

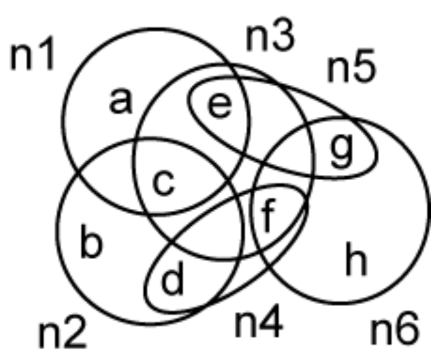
b,c先單獨擺放

(d) visit $n_2 = \{b, c, d\}$: since d is already marked, we skip n_2.

(e) visit $n_3 = \{c, e, f\}$: since e and f are already marked, we skip n_3.

(f) visit $n_6 = \{f, g, h\}$: since f and g are already marked, we skip n_6.

因此最後a,b,c,h都會單獨擺放，因此cluster的效果沒有很好



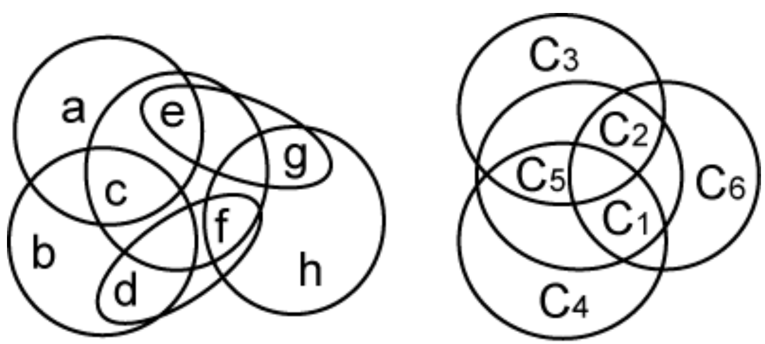
cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a\}$
C_4	$\{b\}$
C_5	$\{c\}$
C_6	$\{h\}$

Obtaining Clustered-level Netlist

- # of nodes/hyperedges reduced: 6 nodes, 4 hyperedges

net	gate-level	cluster-level	final
n_1	$\{a, c, e\}$	$\{C_3, C_5, C_2\}$	$\{C_3, C_5, C_2\}$
n_2	$\{b, c, d\}$	$\{C_4, C_5, C_1\}$	$\{C_4, C_5, C_1\}$
n_3	$\{c, e, f\}$	$\{C_5, C_2, C_1\}$	$\{C_5, C_2, C_1\}$
n_4	$\{d, f\}$	$\{C_1, C_1\}$	$\emptyset$ 自己跟自己圈就不算
n_5	$\{e, g\}$	$\{C_2, C_2\}$	$\emptyset$
n_6	$\{f, g, h\}$	$\{C_1, C_2, C_6\}$	$\{C_1, C_2, C_6\}$

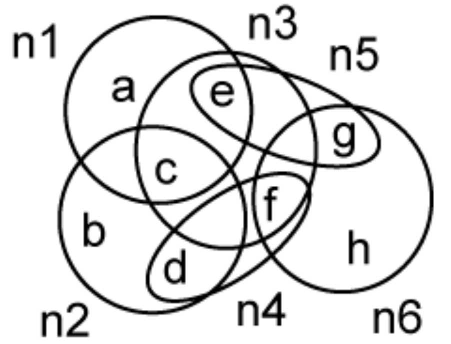
cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a\}$
C_4	$\{b\}$
C_5	$\{c\}$
C_6	$\{h\}$

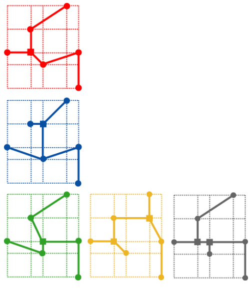


(a) (b)

3. Modified Hyperedge Coarsening

- Revisit skipped nets during hyperedge coarsening
  - We skipped n_1, n_2, n_3, n_6
  - Coarsen un-coarsened nodes in each net





Modified Hyperedge Coarsening

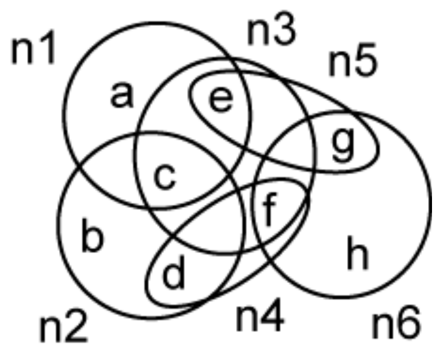
(a) visit $n_1 = \{a, c, e\}$: since e is already marked during HEC, we group the remaining unmarked nodes a and c. We form $C_3 = \{a, c\}$ and mark a and c. e被拿走了，就把剩下的a,c圈再一起

(b) visit $n_2 = \{b, c, d\}$: since d is marked during HEC and c during MHEC as above, we form $C_4 = \{b\}$ and mark b.

(c) visit $n_3 = \{c, e, f\}$: all nodes are already marked, so we skip n_3.

(d) visit $n_6 = \{f, g, h\}$: since f and g are already marked, we form $C_5 = \{h\}$ and mark h.

這樣圈會比普通的hyperedge coarsening效果好一點



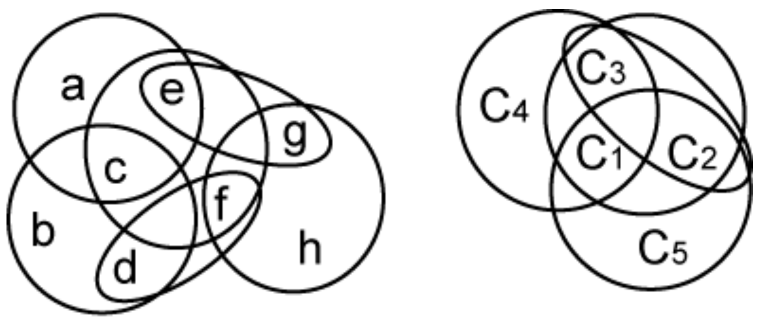
cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a, c\}$
C_4	$\{b\}$
C_5	$\{h\}$

Obtaining Clustered-level Netlist

- # of nodes/hyperedges reduced: 5 nodes, 4 hyperedges

net	gate-level	cluster-level	final
n_1	$\{a, c, e\}$	$\{C_3, C_3, C_2\}$	$\{C_3, C_2\}$
n_2	$\{b, c, d\}$	$\{C_4, C_3, C_1\}$	$\{C_4, C_3, C_1\}$
n_3	$\{c, e, f\}$	$\{C_3, C_2, C_1\}$	$\{C_3, C_2, C_1\}$
n_4	$\{d, f\}$	$\{C_1, C_1\}$	$\emptyset$
n_5	$\{e, g\}$	$\{C_2, C_2\}$	$\emptyset$
n_6	$\{f, g, h\}$	$\{C_1, C_2, C_5\}$	$\{C_1, C_2, C_5\}$

cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a, c\}$
C_4	$\{b\}$
C_5	$\{h\}$



(a) (b)

Clustering for Partitioned-based Placement

各種不同的clustering方式

- First choice
  - *Multilevel k-way Hypergraph Partitioning*, DAC99
  - Similar to EC
- Best choice
  - *A Semi-Persistent Clustering Technique for VLSI Circuit Placement*, ISPD05
  - Used in CPLACE
- Safe choice
  - *SafeChoice: A Novel Approach to Hypergraph Clustering for Wirelength-Driven Placement*, TCAD July 2011
  - Used in SCPlace

跳過了

Best Choice-ISPD05

- Identify the **globally best pair of objects** to cluster.
- Manage a **priority-queue data structure** with the clustering score as a key.

Input: Flat Netlist
Output: Clustered Netlist

1. Until *target object number* is reached:
    2. Find *closest pair* of objects
    3. Cluster them
    4. Update netlist

Fig. 4. Bottom–up clustering.

跳過了

Phase 1: PQ initialization

- For each object ***u*** in the netlist, the closest object ***v*** and its associated clustering score ***d*** are calculated.
- The tuple *(u, v, d)* is inserted to the PQ with ***d*** as a comparison key.
  - For each u, only one tuple with the closest object v is inserted.

Input: Flat Netlist
Output: Clustered Netlist

Phase I. Priority-queue PQ Initialization:
1. For each object u:
2. Find *closest object* v, and its associated clustering score d
3. Insert tuple (u, v, d) into PQ with d as key

Phase II. Clustering:
1. While *target object number* is not reached and top tuple's score $d > 0$:
   2. Pick top tuple (u, v, d) of PQ
   3. Cluster u and v into new object u'
   4. Update netlist
   5. Find *closest object* v' to u' with its clustering score d'
   6. Insert tuple (u', v', d') into PQ with d' as key
   7. Update clustering scores of all neighbors of u'

Fig. 5. BC clustering algorithm.

跳過了

Phase 2: Clustering

- The top tuple (u, v, d) in the PQ is picked up, and the pair of objects (u, v) are clustered creating a new object u'.
  - Update the netlist, the closest object v' to u' and its score d' are calculated, and a new tuple (u', v', d') is inserted to the PQ.
  - The scores of the neighbors of the new object u' (all neighbors of u and v) need to be recalculated.

Input: Flat Netlist
Output: Clustered Netlist

Phase I. Priority-queue PQ Initialization:

1. For each object u:
2. Find *closest object* v, and its associated clustering score d
3. Insert tuple (u, v, d) into PQ with d as key

Phase II. Clustering:

1. While *target object number* is not reached and top tuple's score $d > 0$:
    2. Pick top tuple (u, v, d) of PQ
    3. Cluster u and v into new object u'
    4. Update netlist
    5. Find *closest object* v' to u' with its clustering score d'
    6. Insert tuple (u', v', d') into PQ with d' as key
    7. Update clustering scores of all neighbors of u'

Fig. 5. BC clustering algorithm.

跳過啦

Score Function in Best Choice

- Clustering score *d(u, v)* : How close two nodes *u* and *v* are
  - The weight w_e of a hyper-edge *e* is defined as 1/|e|.
  - Clustering score d(u, v) between two objects *u* and *v*:

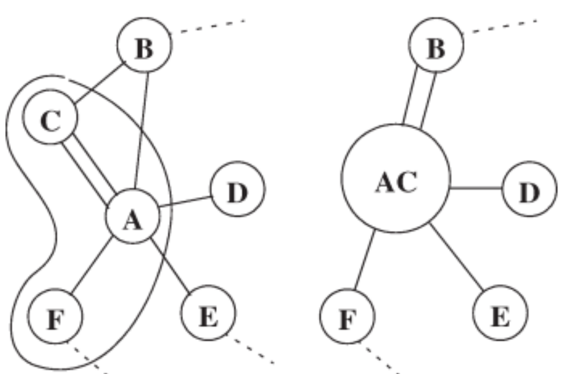
$$d(u,v) = \sum_{e \in E | u,v \in e} \frac{w_e}{(a(u) + a(v))}$$

    - e: a hyper-edge connecting objects u and v
    - w_e: a corresponding edge weight
    - a(u) and a(v): the areas of u and v
- c(u): The closest object to u.
  - The neighbor object with the highest clustering score to u
  - $c(u) = v$ such that $d(u, v) = \max\{d(u, z) | z \in N_u\}$, N_u is the set of the neighboring objects to a given object u

跳過了

Example of Best Choice

- Assume the input netlist with six objects {A, B, C, D, E, F} and eight hyper-edges {A, B}, {A, C}, {A, D}, {A, E}, {A, F}, {A, C}, {B,C}, and {A, C, F} as in Figure(a). The size of each object is 1.
- Since d(A, C) is the highest score in the PQ, A will be clustered with C and the circuit netlist will be updated as shown in Figure(b).



(a) (b)

Clustering score of A and neighbors from (a)	
d(A, B)	1/4
d(A, C)	**2/3**
d(A, D)	1/4
d(A, E)	1/4
d(A, F)	5/12

Clustering score of AC and neighbors from (b)	
d(AC, F)	1/3
d(AC, E)	1/6
d(AC, D)	1/6
d(AC, B)	1/3

跳過了

Safe Choice-TCAD11

- Guarantees that clustering would **not degrade the placement quality**
- Safe condition: If two objects satisfy the safe condition, clustering them would not degrade the wirelength
  - Safe clustering 1: If the optimal wirelength of the netlist generated by clustering a set of vertices is the same as the original netlist, then it is safe to cluster the vertices.
    - NP-hard
  - Safe clustering 2: If a set of vertices can be moved to the same location without increasing the wirelength, then it is safe to cluster the vertices.

跳過了

Safe Condition

- SafeChoice algorithm: Globally ranks and chooses potential clusters via a priority-queue based on their safeness and area
- Maintain a global PQ, cost function:

$$C(a, b) = S^* + \theta \times \frac{A_a + A_b}{\overline{A}_s}$$

  - S*: Safeness of clustering a and b

— Stops clustering when generating more clusters would degrade the placement wirelength

Summary: Partitioning

- Mostly used in placement
- Discussed methods: **group migration** (K-L, F-M), **network flow** (FBB), **simulated annealing**.
- Other important partitioning approaches
  - **Spectral method (ratio cut):** Barnes, *SIAM J. Algebraic and Discrete Methods*, 1982; Alpert & Kahng, DAC-95, DAC-96, etc.
  - **Probabilistic approach:** Dutt & Deng, DAC-96; Chao, et. al., ICCAD-99.
  - **Mathematical programming:** Shih & Kuh, DAC-93 (quadratic programming); Wu et al., TCAD, Oct . 2001 (ILP)
  - **Unified approach:** Network flow + Spectral, Li, et al, ICCAD-95.
  - **Net partitioning:** Cong, et. al., DAC-92
  - **Neural network**
- k-way partitioning: Sanchis, TC, 1989; Cong & Lim, ISPD-98.
- Clustering: Cong, et. al., ICCAD-97; Chao, et. al., ICCAD-99
- Multi-level circuit partitioning: Alpert, et. al., TCAD, Aug. 1998; **Karypis & Kumar, DAC-99 (First choice)**
  - Cong et. al, ISPD-03: Current results are almost “good enough.”
- An earlier survey: Alpert & Kahng, *Integration*, 1995.

MOE IC/CAD Contest Problems

- 2000 MOE IC/CAD contest problem 2 : 2-way mincut partitioning
  - **Input:** A net-list for a circuit
  - **Objective:** To partition the circuit to two subcircuits A and B so that the cut-set of subcircuits A and B is minimized under the constraint |size(A) – size(B)| < n/100, where n is the number of cells in the circuit.
- 2001 MOE IC/CAD contest problem 3 : k-way netlist partitioning
  - Partition the set *C* of *n* cells into *K disjoint, balanced* groups *G1*, *G2*, *G3*, …,*GK* so that the overall cut size is minimized; in other words, no cell replication is allowed.