Chapter 3

VLSI Physical Design

何宗易
Tsung-Yi Ho
tyho@cs.nthu.edu.tw
http://theta.cs.nthu.edu.tw
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan

Basic Concept

*schematic diagrams: transistor level的diagram (NMOS, PMOS)

- Physical design
  - Creating circuits on silicon.
  - Schematic diagrams are translated into sets of geometric patterns.
  - Every layer is defined by a distinct pattern.

- The topology of the transistor network establishes the logic function.

Basic Concept

- The process of physical design is performed using tool called layout editor.
- Specify the shape, dimensions and placement of every polygon on every layer of the chip.
- Reduce the complexity by using the concept of library.
- Library cells are used as building blocks by creating copies of the basic cells. A copy of a cell is called an instance.

Basic Concept

- Designer's goal is to obtain a fast circuit in the minimum amount of area.
- Small changes in the shape will affect the electrical characteristics of the circuit.
- Circuit simulation also helps to ensure that the layout is accurate and provides a network that meets specification.

CAD toolsets

- Layout editor: draw transistors and wiring patterns made up of polygon. Each layer has a distinct color or fill pattern on the screen.

- The electrical behavior of the design is simulated by first using an extraction tool which translate the polygon patterns into equivalent electrical network in SPICE format.

- Extraction provide important parameters such as the drawn channel width and length for each FET. And how transistors are wired together.

CAD toolsets

- Circuit simulation by SPICE.

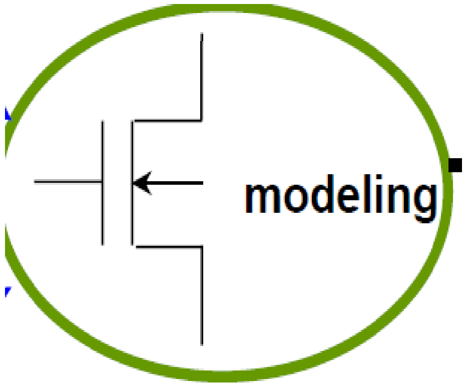
- Layout versus schematic (LVS): check the layout against the schematic diagram. To verify the layout corresponds the intended circuit.

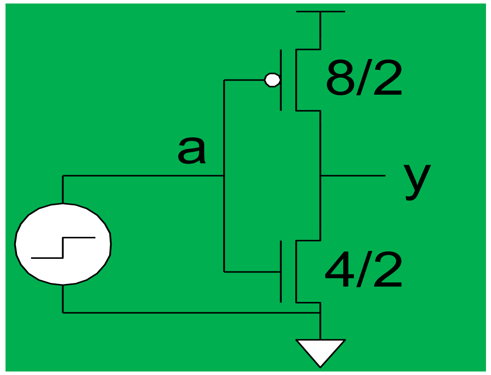
- Design rule checker (DRC): check every occurrence of the design rule list on the layout. Design can be fabricated within the limitation of the process.

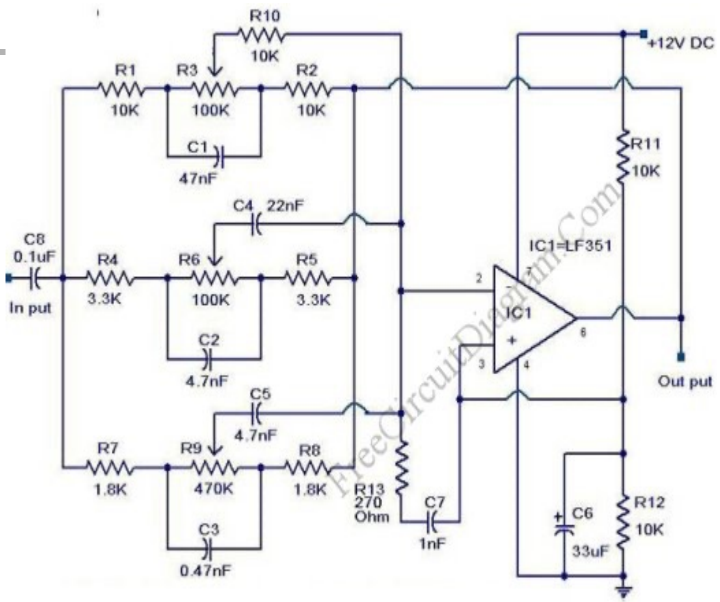
Introduction to SPICE

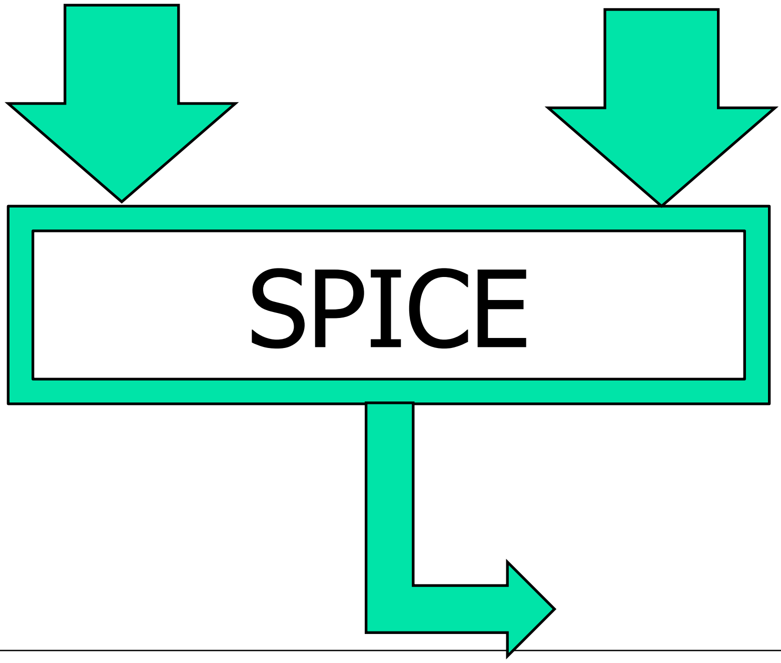
- **S**imulation **P**rogram with **I**ntegrated **C**ircuit **E**mphasis
  - Developed in 1970’s at Berkeley
  - Many commercial versions are available
  - HSPICE is a robust industry standard
    - Has many enhancements that we will use

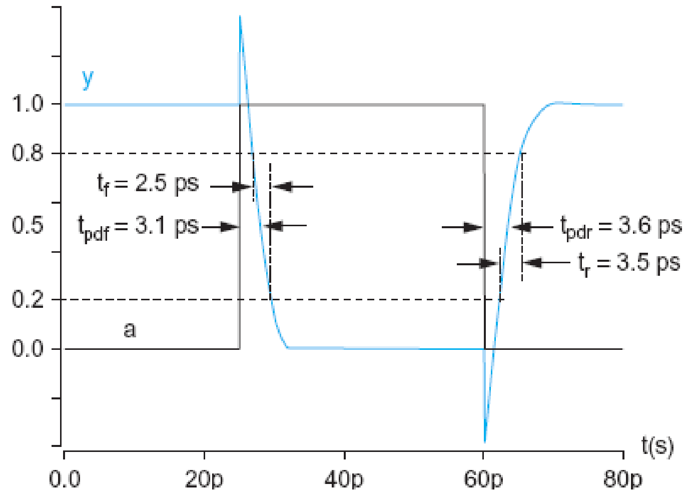
SPICE


modeling


a
8/2
y
4/2


R10
10K
R1
10K
R3
100K
R2
10K
+12V DC
C1
47nF
R11
10K
C4 22nF
C8
0.1uF
In put
R4
3.3K
R6
100K
R5
3.3K
IC1=LF351
IC1
C2
4.7nF
Out put
C5
4.7nF
R7
1.8K
R9
470K
R8
1.8K
R13
270
Ohm
C7
1nF
C6
33uF
R12
10K
C3
0.47nF
FreeCircuitDiagram.Com


SPICE


y
1.0
0.8
0.5
0.2
0.0
a
t_f = 2.5 ps
t_{pdf} = 3.1 ps
t_{pdr} = 3.6 ps
t_r = 3.5 ps
t(s)
0.0
20p
40p
60p
80p

SPICE

- Take a text netlist describing the circuit elements (transistors, resistors, capacitors, etc.) and their connections, and translate this description into equations to be solved.
- The general equations produced are nonlinear differential algebraic equations which are solved using implicit integration methods, Newton's method and sparse matrix techniques.

Writing Spice Decks

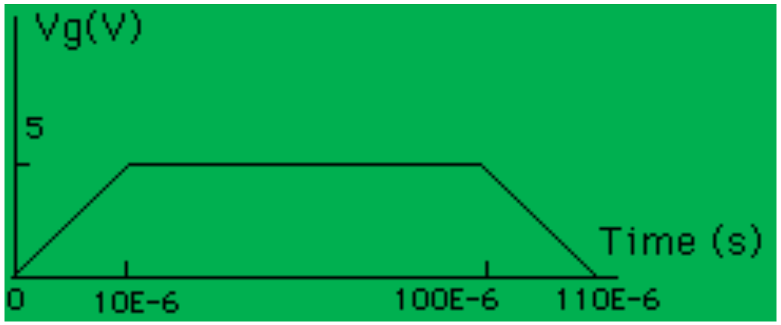
- Writing a SPICE deck is like writing a good program
  - Plan: sketch schematic on paper or in editor
    - Modify existing decks whenever possible
  - Code: strive for clarity
    - Start with name, email, date, purpose
    - Generously comment
  - Test:
    - Predict what results should be
    - Compare with actual
    - *Garbage In, Garbage Out!*

MOSFET Elements

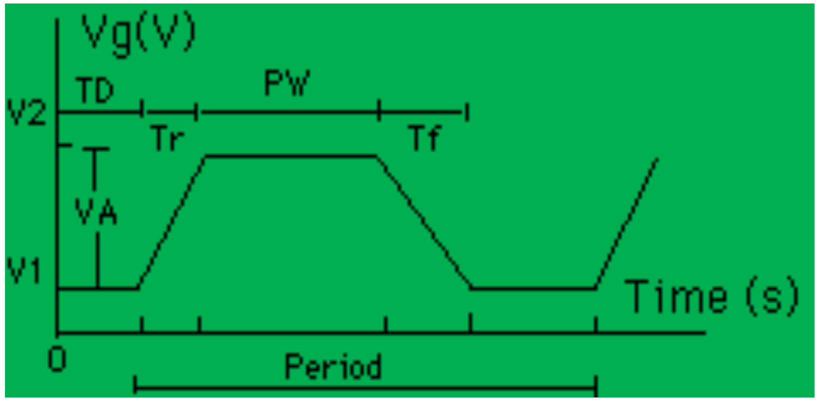
M element for MOSFET

```
Mname drain gate source body type
+ W=<width> L=<length>
+ AS=<area source> AD = <area drain>
+ PS=<perimeter source> PD=<perimeter drain>
```

- Piecewise Linear Source Function - PWL or PL：
  - Vname N1 N2 PWL(T1 V1 T2 V2 T3 V3 ...)

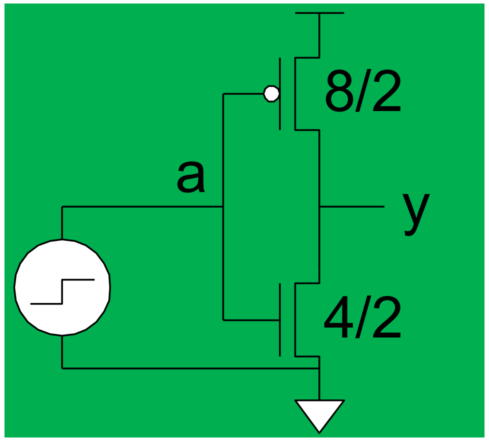


- **Pulse**
  - Vname N1 N2 PULSE(V1 V2 TD Tr Tf PW Period)



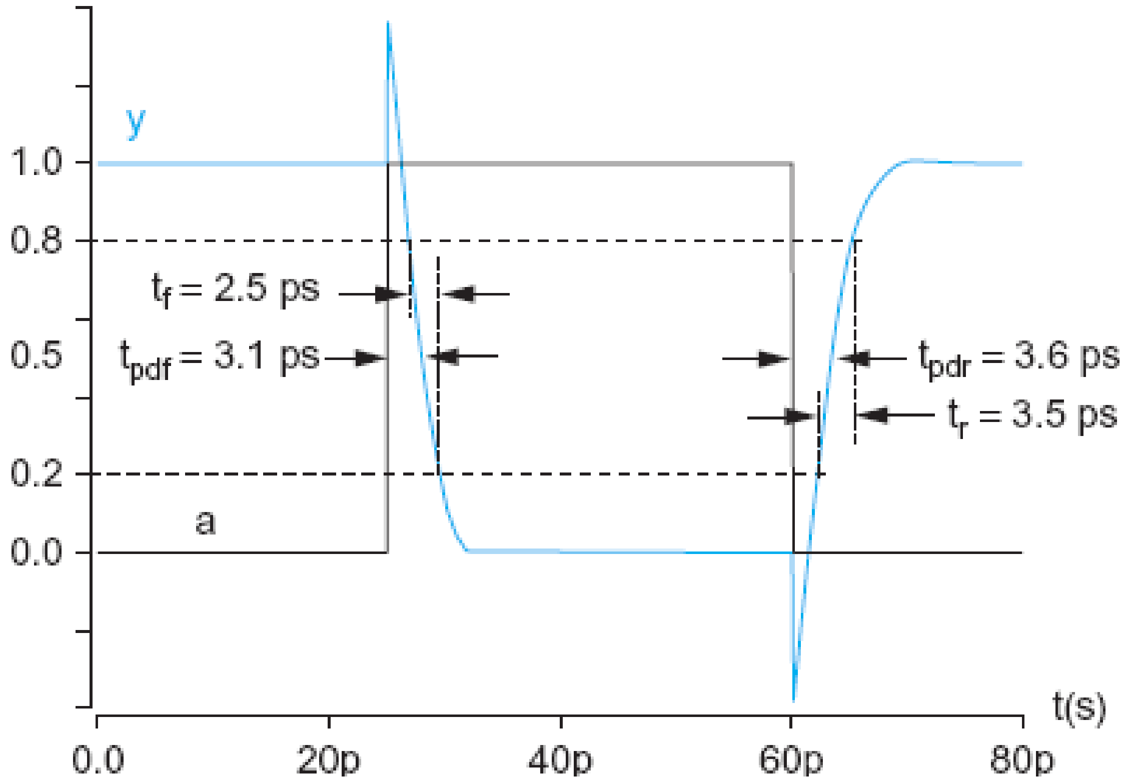
Transient Analysis

```
* inv.sp

* Parameters and models
*----------------------------------------------------------------------
.param SUPPLY=1.0
.option scale=25n
.include '../models/ibm065/models.sp'
.temp 70
.option post

* Simulation netlist
*----------------------------------------------------------------------
Vdd     vdd     gnd     'SUPPLY'
Vin     a       gnd     PULSE    0 'SUPPLY' 50ps 0ps 0ps 100ps 200ps
M1      y       a       gnd      gnd      NMOS     W=4      L=2
+ AS=20 PS=18 AD=20 PD=18
M2      y       a       vdd      vdd      PMOS     W=8      L=2
+ AS=40 PS=26 AD=40 PD=26

* Stimulus
*----------------------TSTEP TSTOP -------------------------
.tran 0.1ps 80ps
.end
```



Transient Results

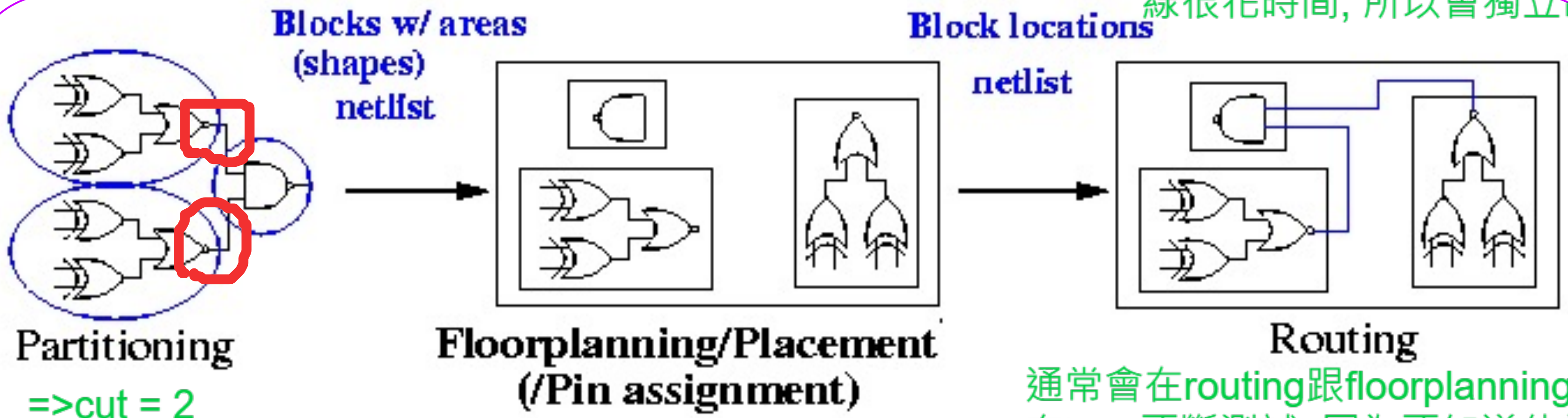
- Unloaded inverter
  - Overshoot
  - Very fast edges



partitioning: 切成給定的n塊, 使得塊跟塊之間的cut(連線)最少, Floorplanning: 根據切好的塊數, 在給定的層內進行排列擺放, 使得Area最小(block跟block距離近, 連線短, 速度快), 並估計從哪裡開始連線

Physical Design Flow

=>邊floorplanning擺block邊繞線很花時間, 所以會獨立出來

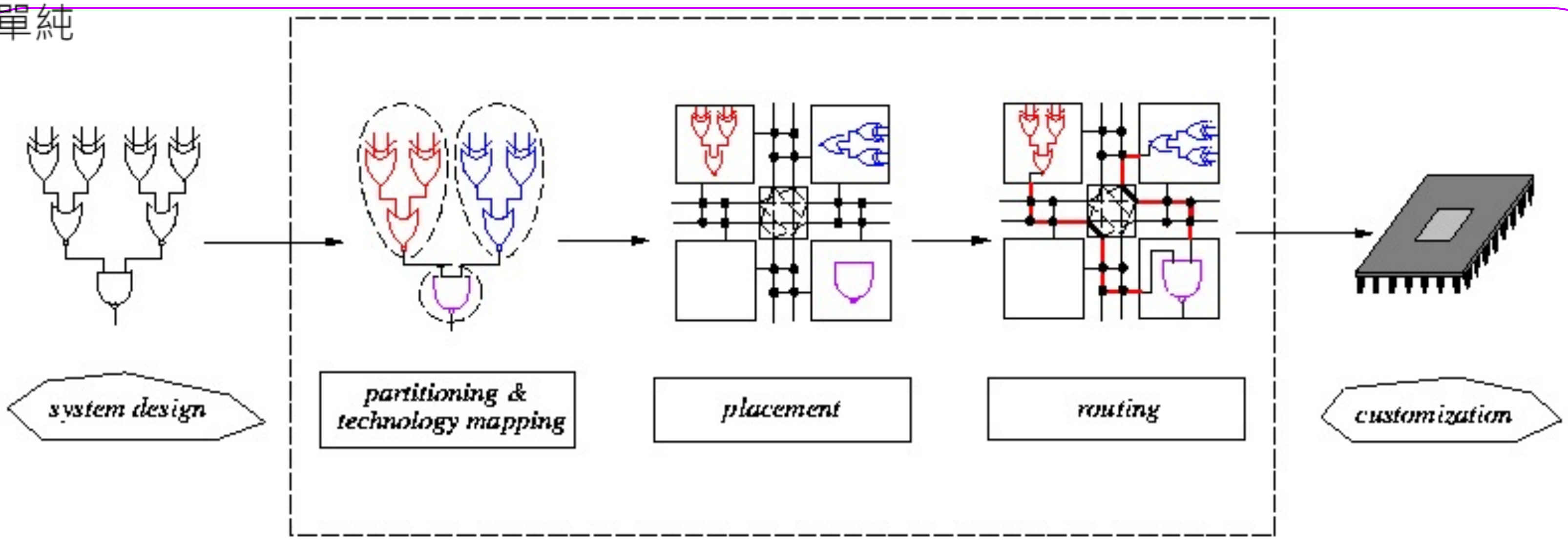


=>cut = 2

把cut數減低, 在floorplan時要考慮到的線段數就較少, 較單純

Cell-Based

通常會在routing跟floorplanning之間有loop不斷測試, 因為不知道估計的到底符不符合實際繞的結果



FPGA

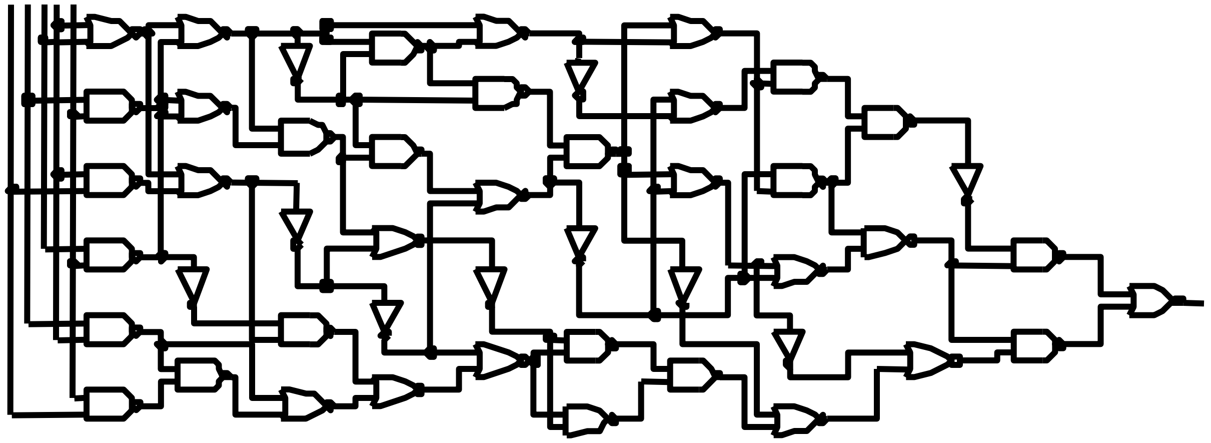
Partitioning

- Decomposition of a complex system into smaller subsystems
  - Done hierarchically
  - Partitioning done until each subsystem has manageable size
  - Each subsystem can be designed independently
- Interconnections between partitions minimized=>減少cut數量
  - Less hassle interfacing the subsystems
  - Communication between subsystems usually costly

=>切完之後擺得不好的話導致放太遠, 線太長cost會太高

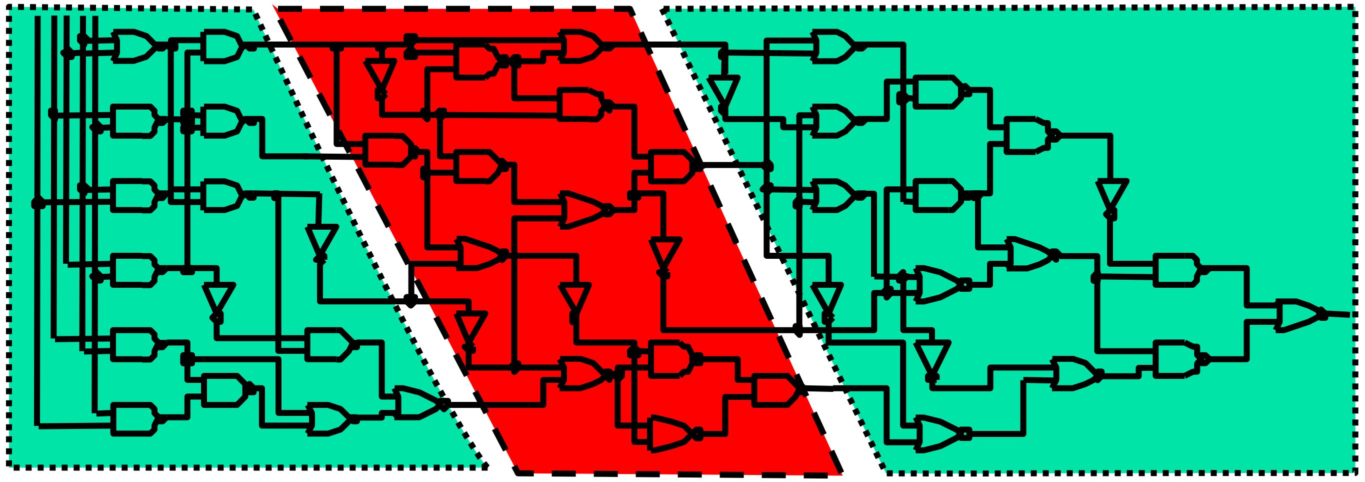
Example: Partitioning of a Circuit

Input size: 48



Cut 1=4 Cut 2=4

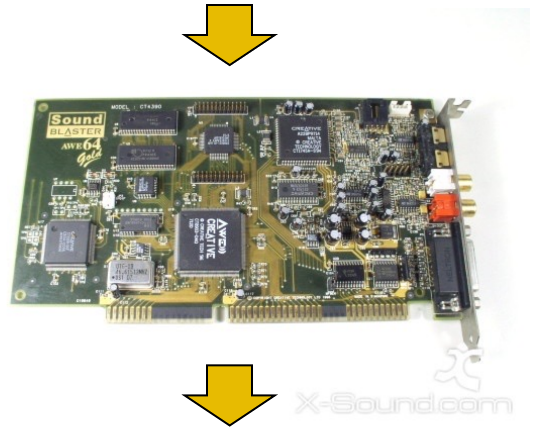
Size 1=15 Size 2=16 Size 3=17



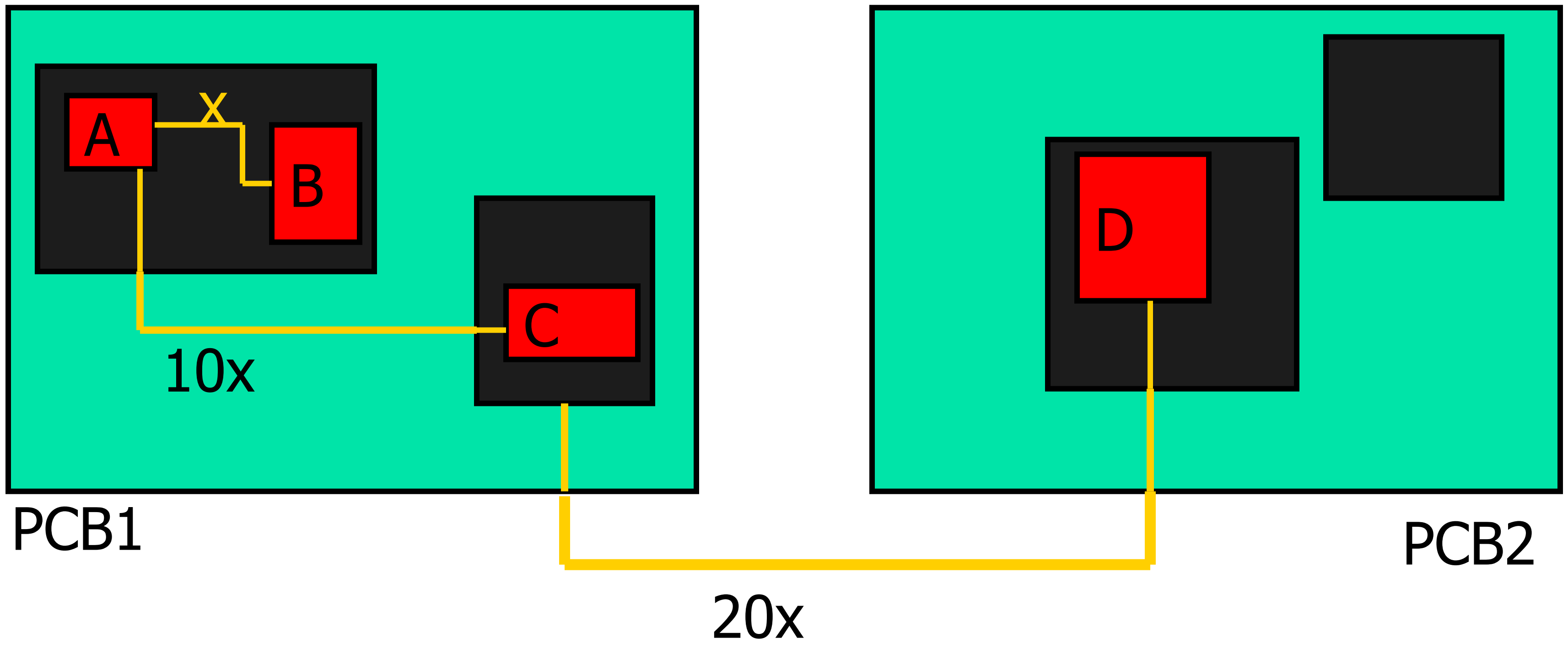
Hierarchical Partitioning

*有些常做且固定的function可以直接做成hardware chip, 當chip之間有沒有訊號, chip要做什麼都給定, 就可以做system-level的partition

- Levels of partitioning:
  - System-level partitioning:
    Each sub-system can be designed as a single printed circuit board (PCB)
  - Board-level partitioning:
    Circuit assigned to a PCB is partitioned sub-circuits
    each fabricated as a VLSI chip
  - Chip-level partitioning:
    Circuit assigned to the chip is divided into manageable sub-circuits
    NOTE: physically not necessary



Delay at Different Levels of Partitions



PCB = Printed Circuit Board

Partitioning: Formal Definition

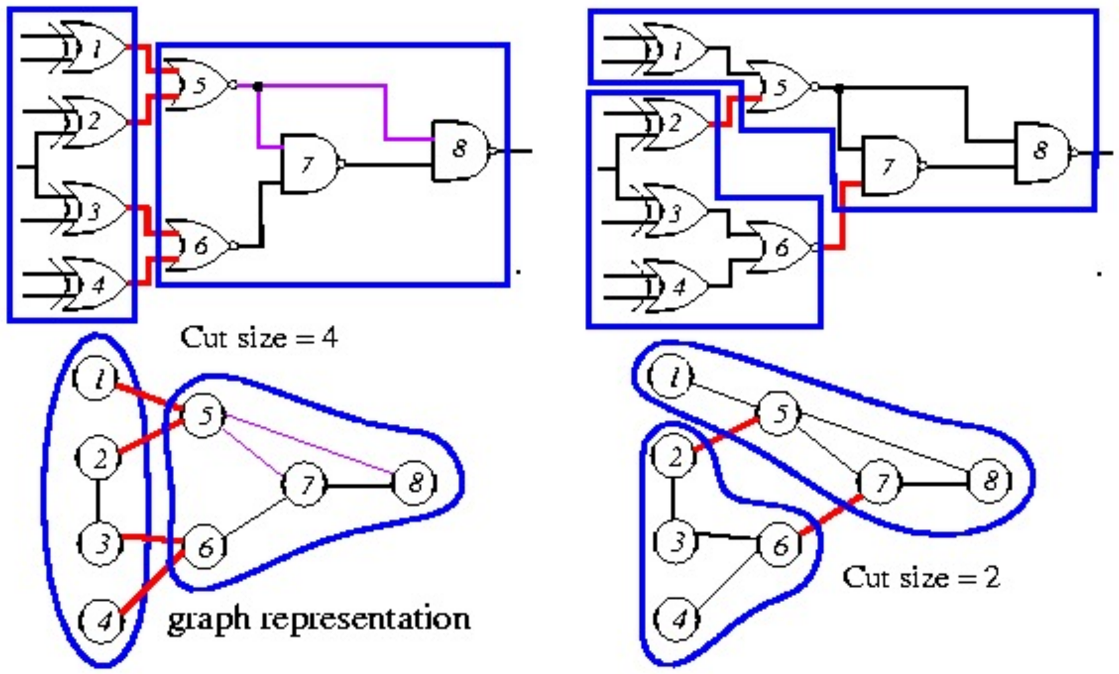
Graph內每個node就是一個gate

- Input:
  - Graph or hypergraph
  - Usually with vertex weights (sizes)
  - Usually weighted edges
- Constraints
  - Number of partitions (K-way partitioning)
  - Maximum capacity of each partition
    OR
    maximum allowable difference between partitions
- Objective
  - Assign nodes to partitions subject to constraints s.t. the cutsize is minimized
- Tractability
  - Is NP-complete ☹

限制: 分成K份, 同時每份內有限制說最大能包含的gate數, 或份跟份之間gate數的最大差異
=> balance每份的數量

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 ($\sum_{v \in V_i} s(v) \le B_i$).
  - 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 Heuristic

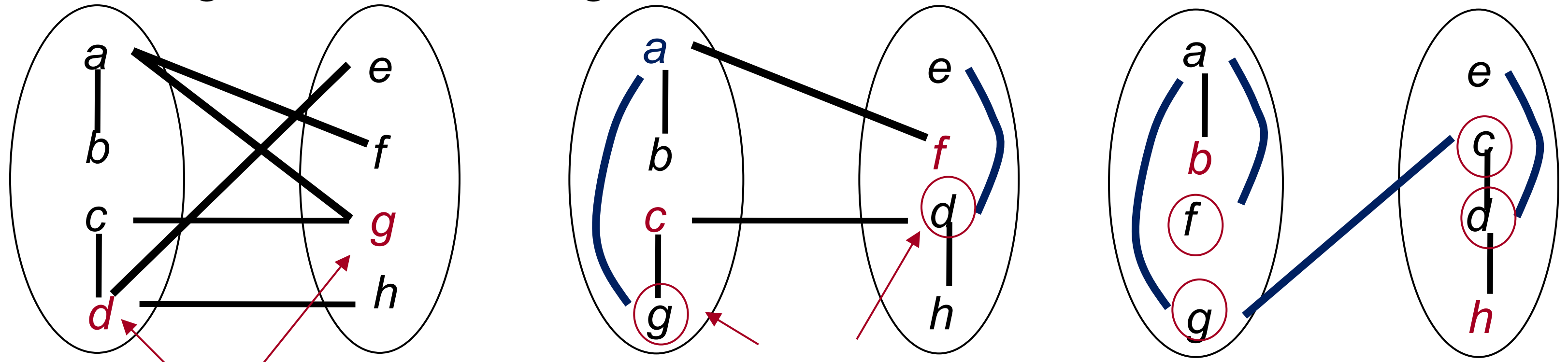
- Kernighan and Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, Feb. 1970.
- An **iterative**, **2-way**, **balanced** partitioning (bi-sectioning) heuristic.
- Till the cut size keeps decreasing
  - Vertex pairs which give the largest 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.

smallest increase仍要選的原因為, 之後可能還會產生decrease, 所以不能當全部都會增加cut時就不繼續執行

先各一半分成兩區, ex. 100個先照1~50放左, 51~100放右. 左右之間所有點跟點之間的連線數就是cut數. 最後目標就是找到兩邊最好的排列使得cut最小
先算一開始兩邊的cut數, 之後每次都找到pairs, 使得他們exchange後cut數會減最多或提升最少
等到全部都被跑過之後, 找到set with largest partial sum, 解鎖所有vertices. 可以再繼續跑KL多次來找到真正更好的解, partial sum是指到這個step時, 所獲得的累積reduction

Kernighan-Lin Heuristic: A Simple Example

- Each edge has a unit weight.



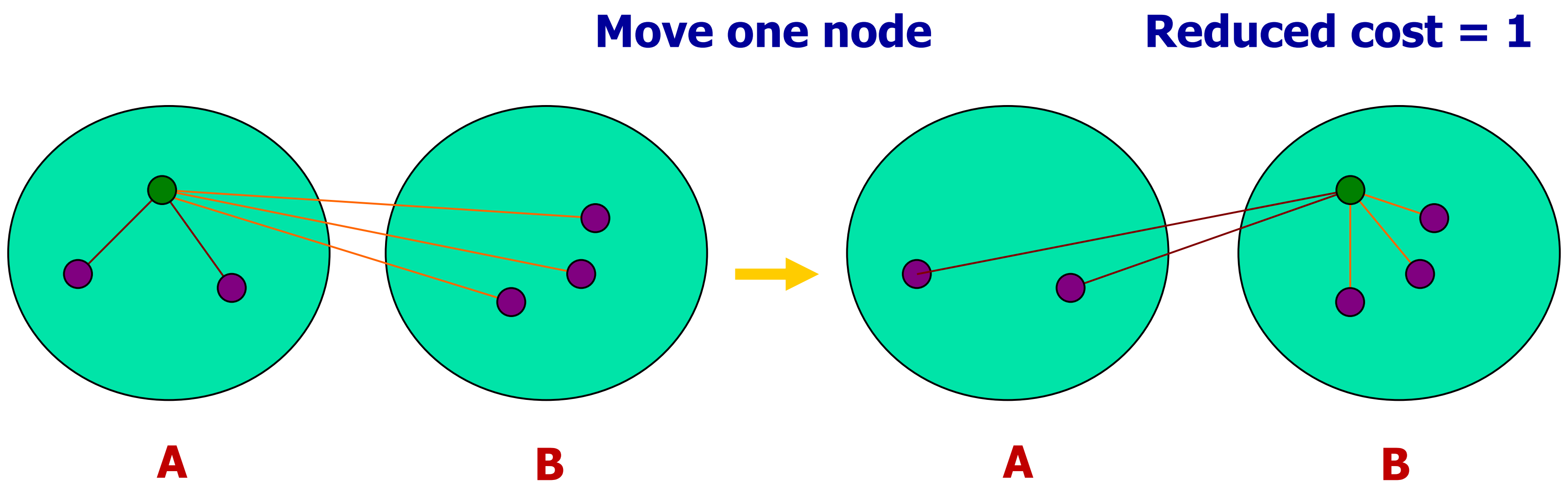
Pairs with the largest gain.   Locked !!!

Step #	Vertex pair	Cost reduction	Cut cost
0	-	0 partial sum_1 = 0	5
1	{*d*, *g*}	3 partial sum_1 = 3	2
2	{*c*, *f* }	1 partial sum_2 = 4	1
3	{*b*, *h*}	-2 partial sum_3 = 2	3
4	{*a*, *e*}	-2 partial sum_4 = 0	5

Step 2交換c, f後有largest partial sum, 因此會以此狀態為下次起始, 並解鎖所有nodes, 之後可以繼續run KL algo =>會做很多iteration

- **Questions:** How to compute cost reduction? What pairs to be swapped? =>如何計算pair之間的cost reduction?
  - Consider the change of internal & external connections.

Observation



- Two sets A and B such that $|A| = n = |B|$ and $A \cap B = \varnothing$.
- **External cost** of $a \in A$: $E_a = \sum_{v \in B} c_{av}$.
- **Internal cost** of $a \in A$: $I_a = \sum_{v \in A} c_{av}$.
- ***D*-value** of a vertex a: $D_a = E_a - I_a$ (cost reduction for moving a).

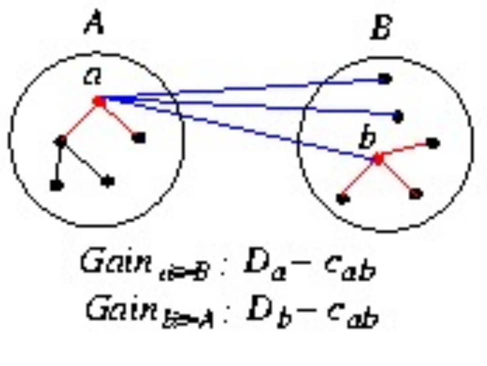
Reduced cost = External cost - Internal cost

Properties

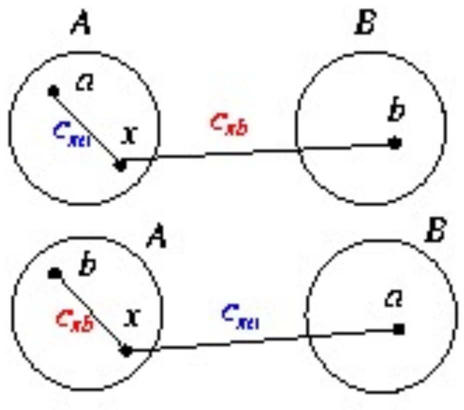
- Cost reduction (gain) for swapping *a* and *b*:
  $g_{ab} = D_a + D_b - 2c_{ab}$ (c_ab是兩者之間的共同連線, 不然會重複計算)
- If $a \in A$ and $b \in B$ are interchanged, then the new *D*-values, *D*', are given by

=>在a b交換之後, 不是所有nodes的D都要計算, 只有跟a b有連線的才要重算

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


A
B
a
b
Gain $_{a \Rightarrow B}$: $D_a - c_{ab}$
Gain $_{b \Rightarrow A}$: $D_b - c_{ab}$

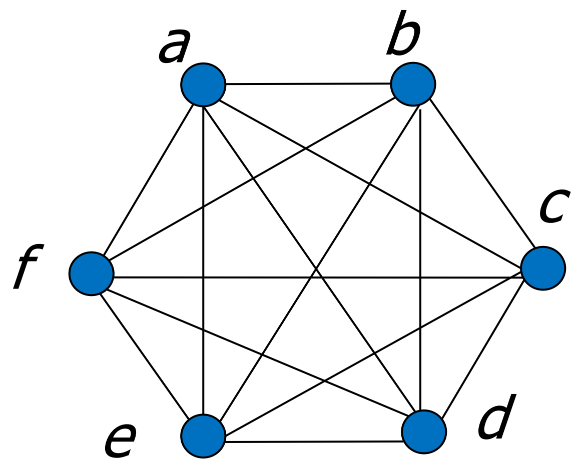
Internal cost vs. External cost



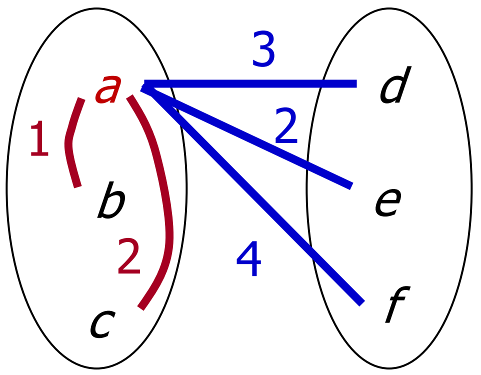
before swap	after swap	ΔC
$-c_{xa}$	$+c_{xa}$	$+2c_{xa}$
$+c_{xb}$	$-c_{xb}$	$-2c_{xb}$

updating D−values

Kernighan-Lin Heuristic: A Weighted Example



	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



cost associated with *a*

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

- Iteration 1:

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

g-Value Computation

- Iteration 1:

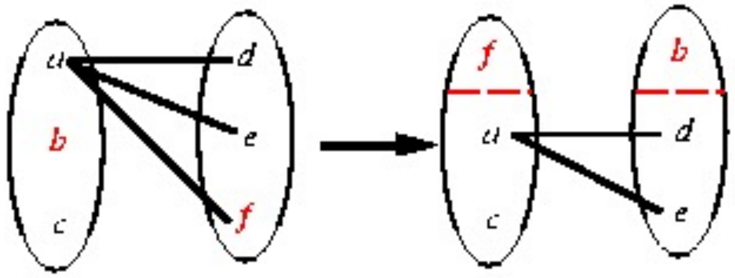
$$\begin{array}{lll}
I_a = 1 + 2 = 3; & E_a = 3 + 2 + 4 = 9; & D_a = E_a - I_a = 9 - 3 = 6 \\
I_b = 1 + 1 = 2; & E_b = 4 + 2 + 1 = 7; & D_b = E_b - I_b = 7 - 2 = 5 \\
I_c = 2 + 1 = 3; & E_c = 3 + 2 + 1 = 6; & D_c = E_c - I_c = 6 - 3 = 3 \\
I_d = 4 + 3 = 7; & E_d = 3 + 4 + 3 = 10; & D_d = E_d - I_d =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

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

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

- Swap *b* and *f* ! (g_1'=4)

D-Value Computation



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

=> 只有跟b f有連線的才要進行重算

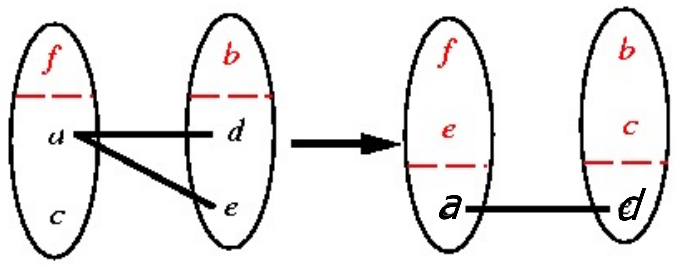
$$\begin{aligned}
D'_a &= D_a + 2c_{ab} - 2c_{af} = 6 + 2 \times 1 - 2 \times 4 = 0 \\
D'_c &= D_c + 2c_{cb} - 2c_{cf} = 3 + 2 \times 1 - 2 \times 1 = 3 \\
D'_d &= D_d + 2c_{df} - 2c_{db} = 3 + 2 \times 3 - 2 \times 4 = 1 \\
D'_e &= D_e + 2c_{ef} - 2c_{eb} = 0 + 2 \times 2 - 2 \times 2 = 0
\end{aligned}$$

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

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

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

Swapping Pair Determination



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

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

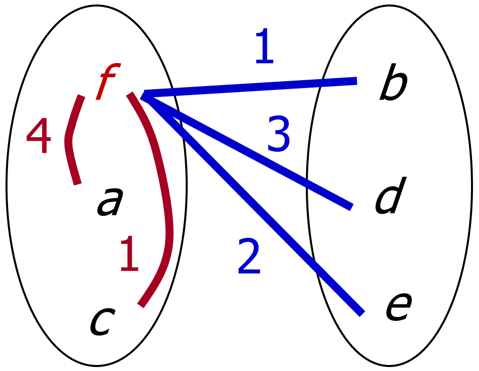
- $g_{xy} = D''_x + D''_y - 2c_{xy}$.

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

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

Next Iteration

	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



- Iteration 2: Repeat what we did at Iteration 1 (Initial cost = 22-4 =18).
- Summary: $g_1' = g_{ce} = -1$, $g_2' = g_{ab} = -3$, $g_3' = g_{fd} = 4$

❖ Largest partial sum = $\max \sum_{i=1}^{k} g_i' = 0$ (k=3) $\Rightarrow$ Stop!

當進入下一round的repeat後, 算出來的largest partial sum <= 0的話, 代表不管再怎麼交換都不會使cut數降低 => terminate

Kernighan-Lin Heuristic

```
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 $|V_A| = |V_B|$, $V_A \cap V_B = \varnothing$,
  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  $g'$ , and compute the new $D_v$,
         for all unlocked $v \in V$;
8    Find $k$, such that $G_k = \sum_{i=1}^{k} g_i'$ 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 $G_k \le 0$;
13 end
```

Time Complexity

- 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: $O(n^3)$.
- Suppose the repeat loop terminates after r passes.
- The total running time: $O(rn^3)$.
  - Polynomial-time algorithm?

=> 會做r個iterations直到largest partial sum <= 0

=> 要做幾round的r未知, 與input n無關
=> 不完全是polynomial time
=> pseudo polynomial

Extensions of K-L Heuristic

- **Unequal sized subsets** (assume $n_1 < n_2$)
  1. Partition: $|A| = n_1$ and $|B| = n_2$.
  2. Add n_2-n_1 dummy vertices to set A. Dummy vertices have no connections to the original graph.
  3. Apply the Kernighan-Lin algorithm.
  4. Remove all dummy vertices.

兩邊subset size不等的話, 加dummy nodes到較少的那邊直到兩邊相等, 執行完後再把dummy移除

- **Unequal sized "vertices"**
  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.

=> s個替代的vertices fully connected and edges inf 不太容易被動到

- ***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.

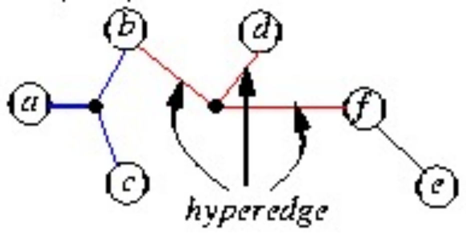
Drawbacks of the Kernighan-Lin Heuristic

- 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**, $O(rn^3)$.

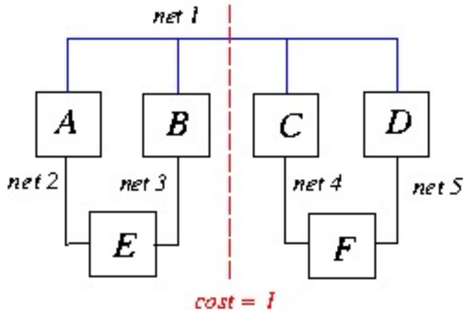
Coping with Hypergraph

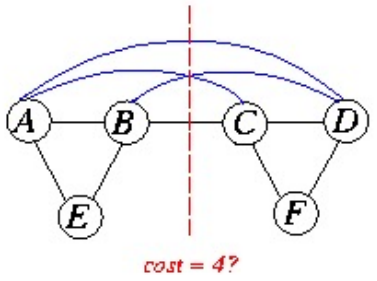
- A hypergraph $H=(N, L)$ consists of a set N of vertices and a set L of hyperedges, where each hyperedge corresponds to a **subset** N_i of distinct vertices with $|N_i| \geq 2$.

hyperedge - 一條edge連到多個 nodes



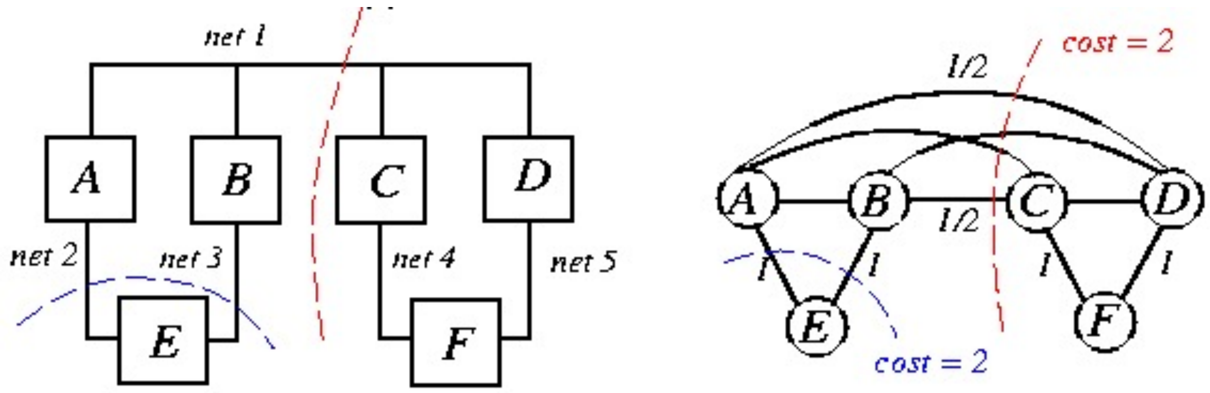
- Schweikert and Kernighan, "A proper model for the partitioning of electrical circuits," 9th Design Automation Workshop, 1972.
- For multi-terminal nets, **net cut** is a more accurate measurement for cut cost (i.e., deal with hyperedges).
  - {A, B, E}, {C, D, F} is a good partition.
  - Should not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



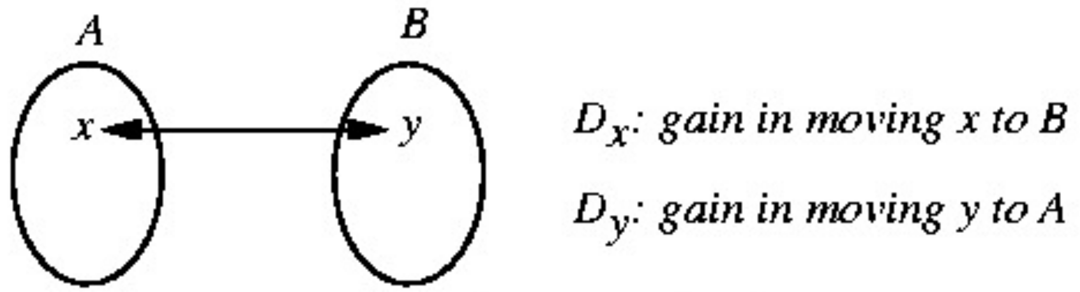


Net-Cut Model

- 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.



- Easy modification of the K-L heuristic.



D_x: gain in moving x to B

D_y: gain in moving y to A

$$g_{xy} = D_x + D_y - Correction(x, y)$$

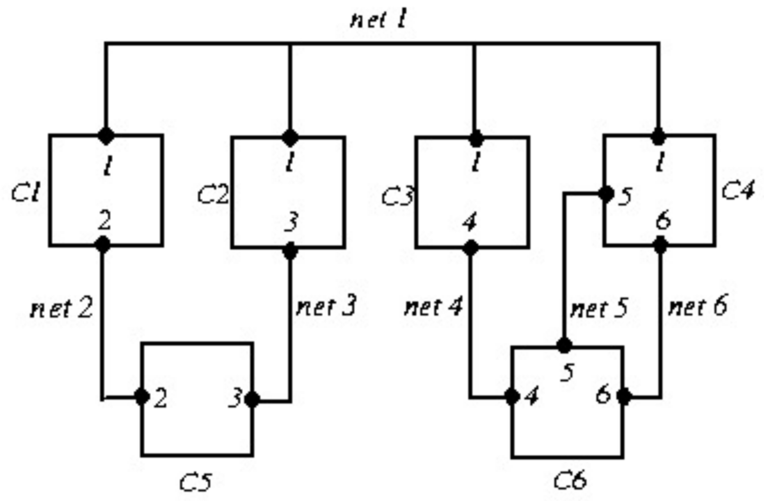
Fiduccia-Mattheyses Heuristic

- 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.
  - 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 pins.

balance factor - 允許的兩邊最不balance的比例

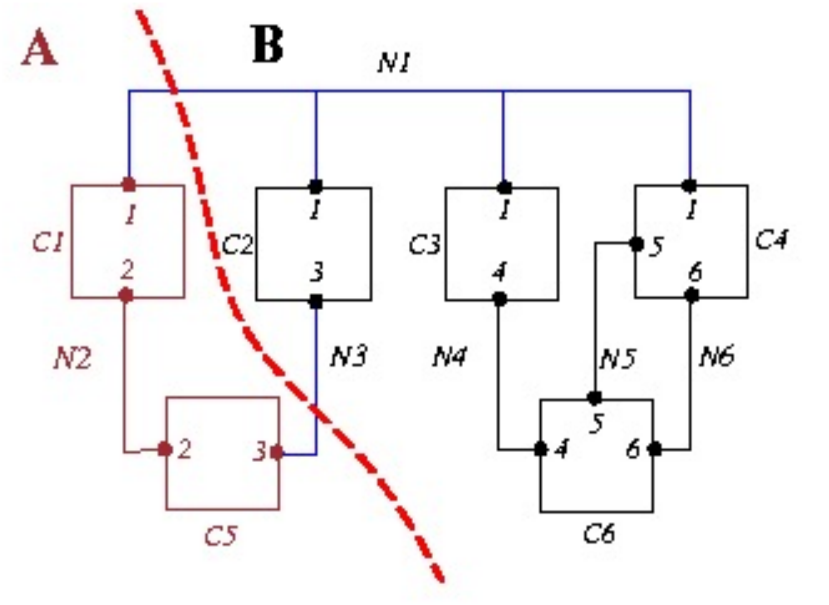
F-M Heuristic: Notation

- $n(i)$: # 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; $P = p(1) + \ldots + p(C) = n(1) + \ldots + n(N)$.



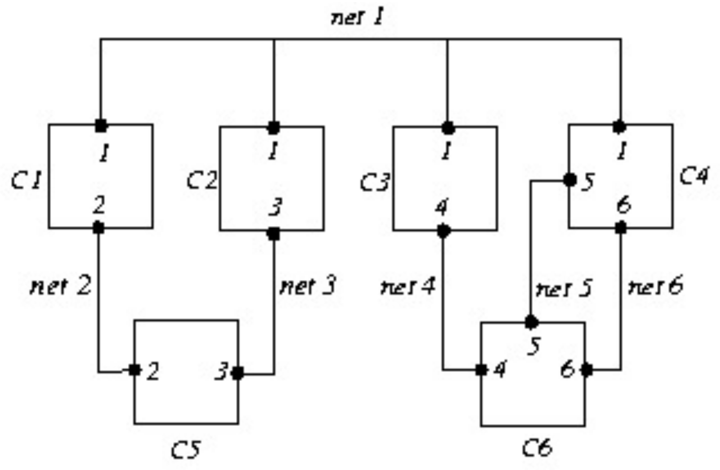
Cut

- **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.



r自己設置, 可以自己設定兩邊的balance

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

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

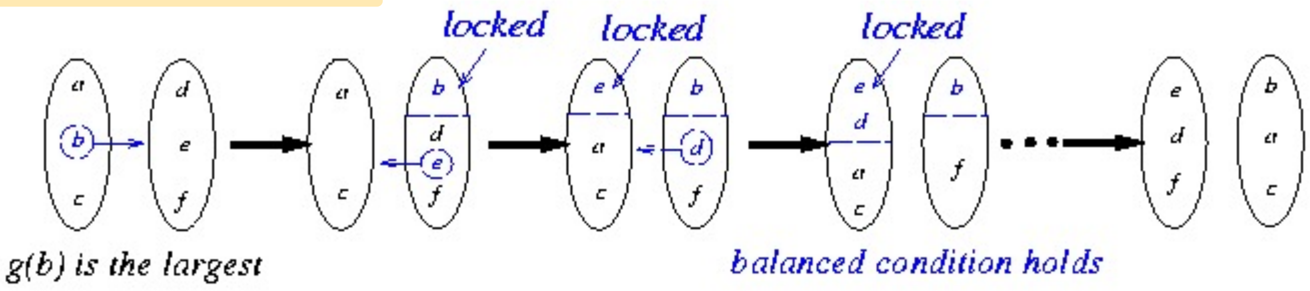
Basic Ideas: Balance and Movement

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

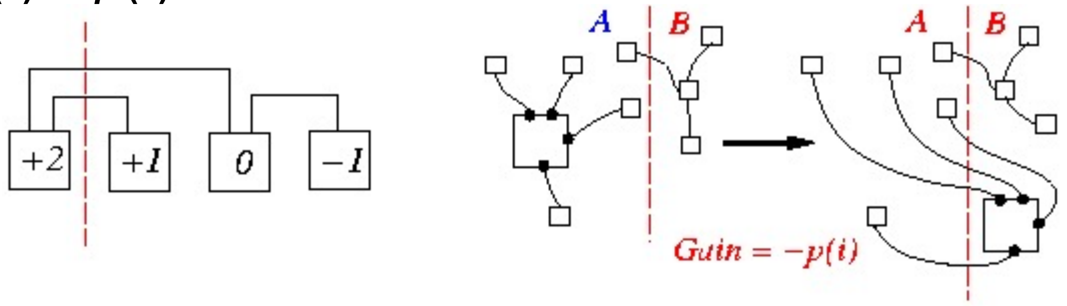
- $g(i)$: gain in moving cell i to the other set, i.e., size of **old** cutset - size of **new** cutset.



- 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 ⇒ two resulting sets: {a, c, e, d}, {b, f}.

Cell Gains and Data Structure Manipulation

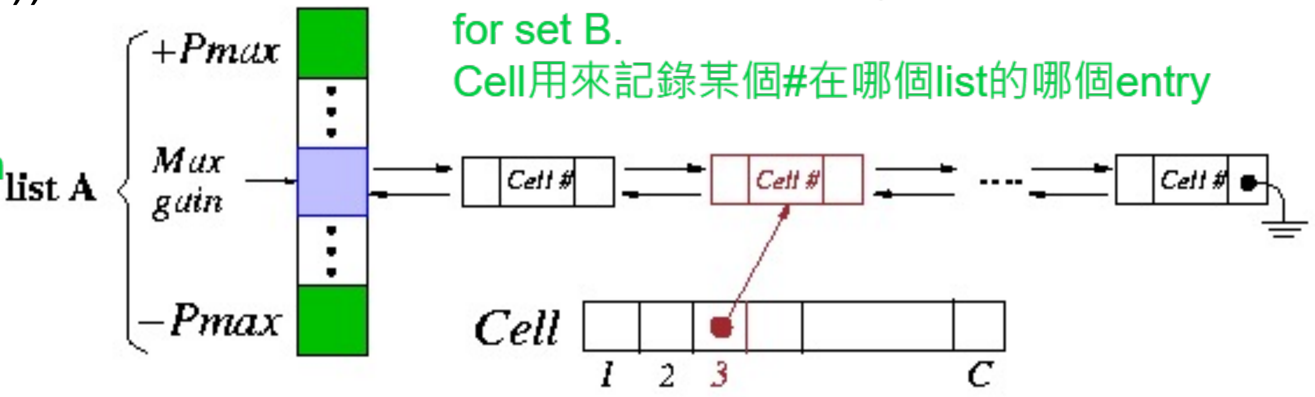
- $-p(i) \le g(i) \le p(i)$ => 每個node的gain根據port #而決定上下限



- Two "bucket list" structures, one for set *A* and one for set *B* ($P_{max} = \max_i p(i)$).

=> 即double linked-list, 除了list A還會有list B for set B.
Cell用來記錄某個#在哪個list的哪個entry

Pmax根據nodes內最大的port #而定.
並且根據目前的gain串到不同的entry

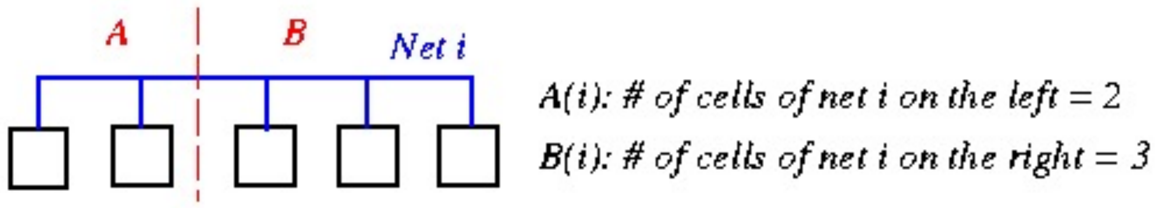


- ***O*(1)-time operations**: find a cell with Max Gain, remove Cell *i* from the structure, insert Cell *i* into the structure, update $g(i)$ to $g(i)+ \Delta$, and update the Max Gain pointer.

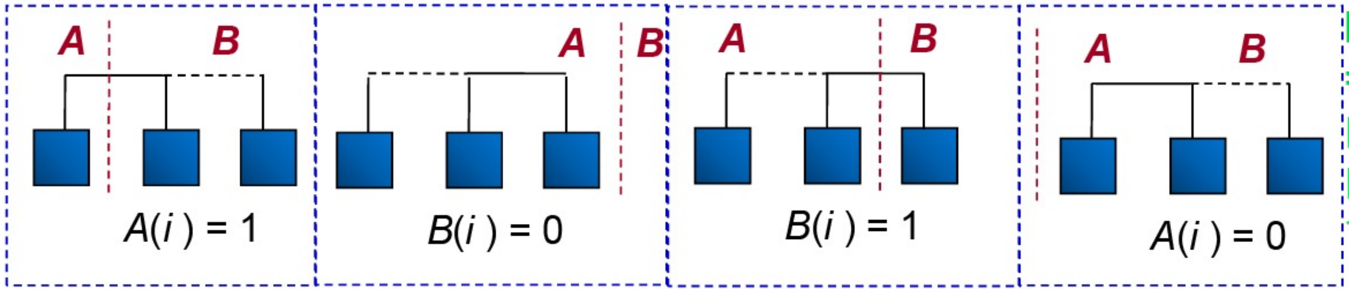
Net Distribution and Critical Nets

- Distribution of Net *i*: (*A*(*i*), *B*(*i*)) = (2, 3).
  - (*A*(*i*), *B*(*i*)) for all *i* can be computed in *O*(*P*) time.

=>並不是所有的vertex都需要去移動


A
B
Net i
A(i): # of cells of net i on the left = 2
B(i): # of cells of net i on the right = 3

- **Critical Nets**: A net is critical if it has a cell which if moved will change its cutstate.
  - ❖ For cells in *left*: *A*(*i*) = 1 or *B*(*i*) = 0.
  - ❖ For cells in *right*: *B*(*i*) = 1 or *A*(*i*) = 0.



hyperedge的情況
=> 只有partition線的某端有1個, 0個的情況去移動才會有影響cut

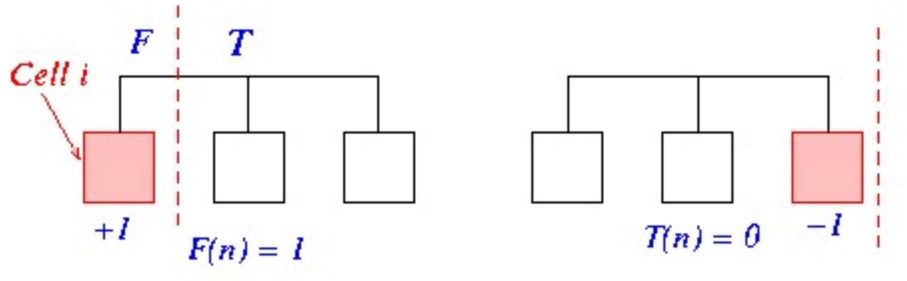
- **Gain of a cell depends only on its critical nets.**

Computing Cell Gains

- Initialization of all cell gains requires $O(P)$ time:

```
$g(i) \leftarrow 0$;
$F \leftarrow$ 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 $g(i) \leftarrow g(i)-1$;
```

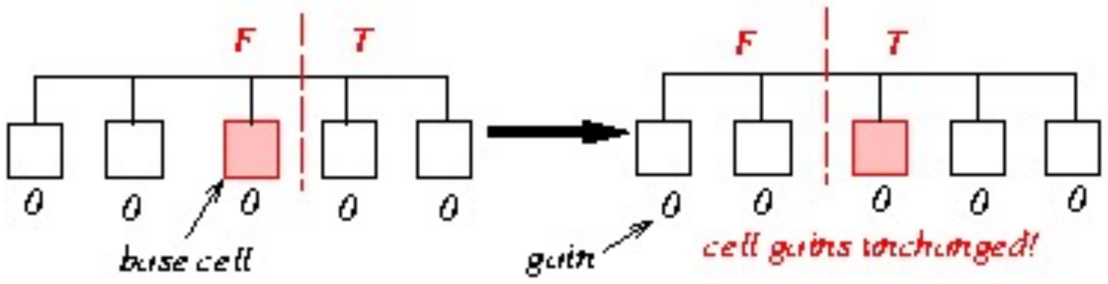
根據一開始的distributions可以輕鬆算出gain值, 選出gain最大的進行移動並更新所有可能更動到的gain



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

Updating Cell Gains

- 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.

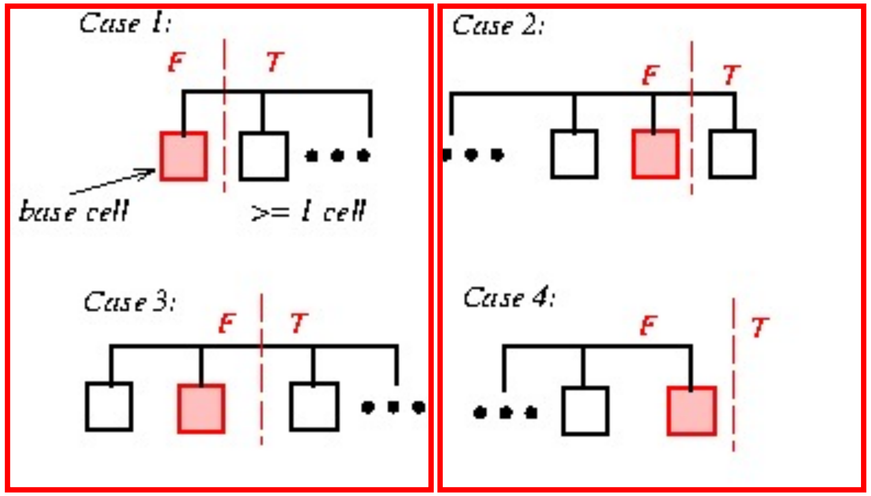


- Consider only the case where the base cell is in the left partition. The other case is similar.

After move

F(n) = 0 or 1

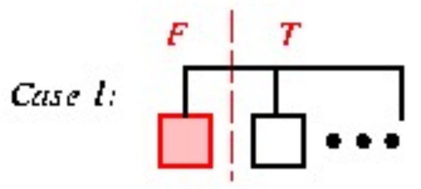
=>考慮的是其他cells要update而不是base cell



Before move

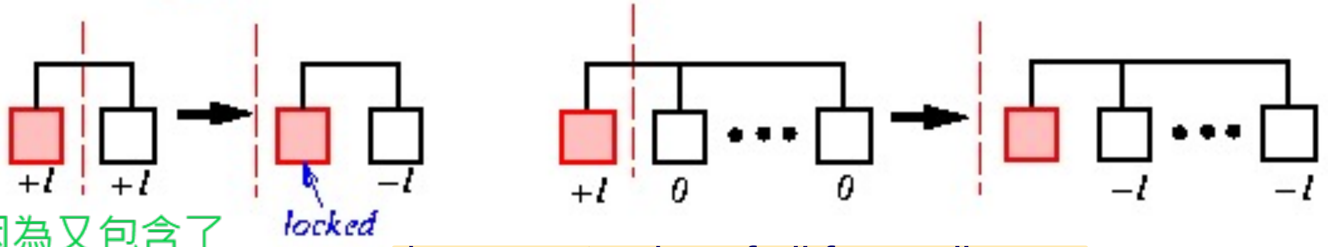
T(n) = 0 or 1

Updating Cell Gains (cont'd)



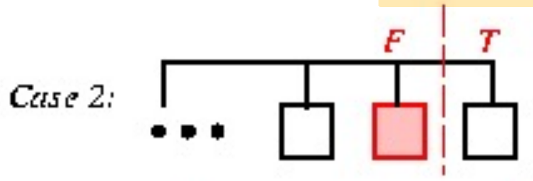
After move

F(n) = 0



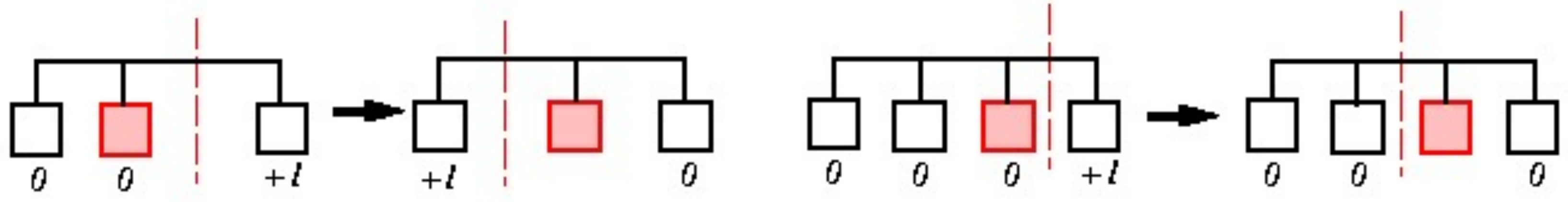
=> -2因為又包含了case 2

decrement gains of all free cells on *n*



Before move

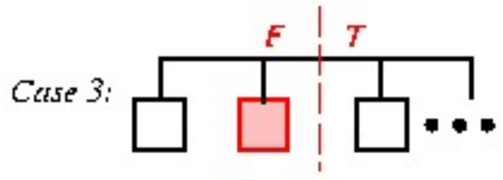
T(n) = 1



=> 左邊cell +1是case 3

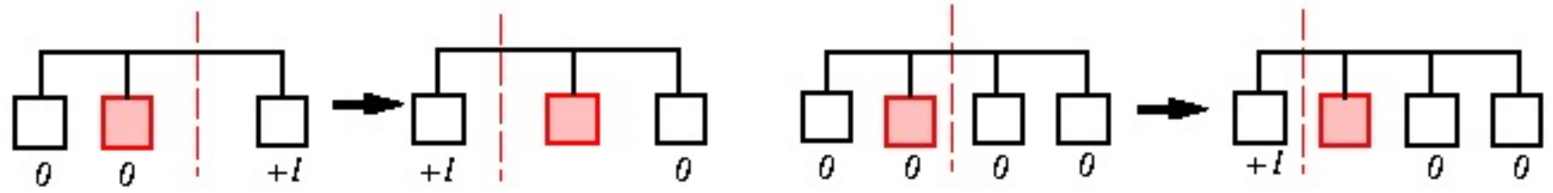
decrement gain of the only *T* cell on *n*

Updating Cell Gains (cont'd)



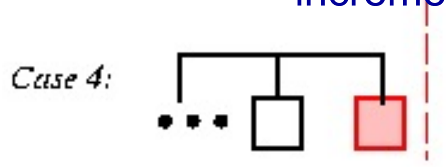
After move

F(n) = 1



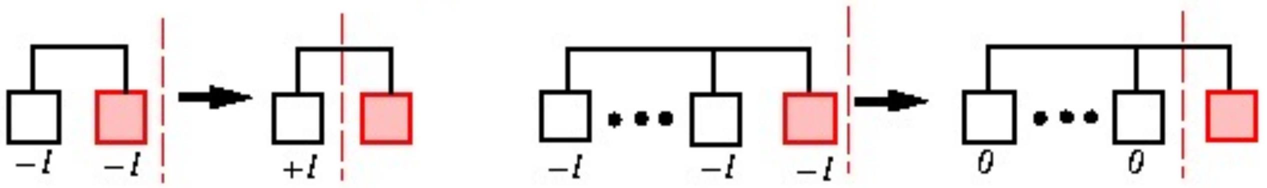
=> 包含 case 2

increment gain of the only *F* cell on *n*



Before move

T(n) = 0

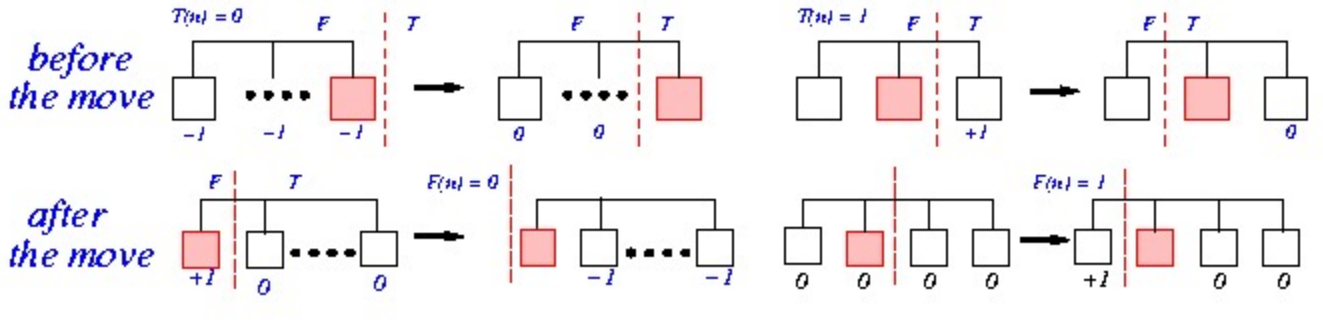


=> 包含case 3

increment gains of all free cells on *n*

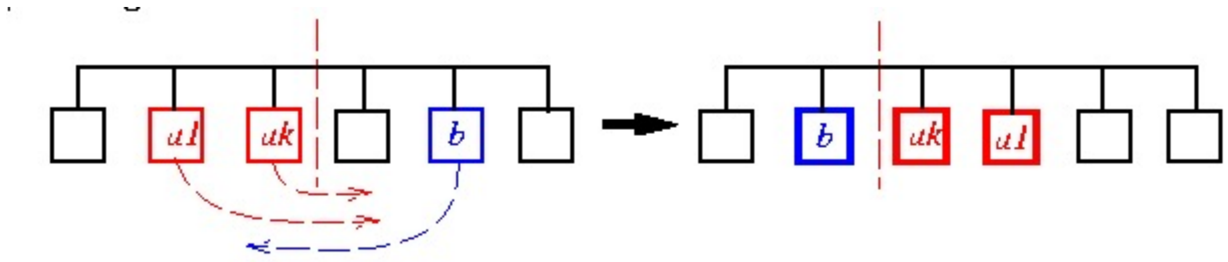
Algorithm for Updating Cell Gains

```
Algorithm: Update_Gain
1 begin  /* move base cells and update neighbors' gains */
2 $F \leftarrow$ the From Block of the base cell;
3 $T \leftarrow$ 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 $n$
      else if $T(n)=1$ then decrement gain of the only $T$ cell on $n$,
      if it is free
       /* change $F(n)$ and $T(n)$ to reflect the move */
7      $F(n) \leftarrow F(n) - 1$; $T(n) \leftarrow T(n)+1$;
      /* check for critical nets after the move */
8     if $F(n)=0$  then decrement gains of all free cells on $n$
      else if $F(n) = 1$ then increment gain of the only $F$ cell on $n$,
      if it is free
9 end
```



Complexity of Updating Cell Gains

- Once a net has “locked” cells at both sides, the net will remain cut from now on.
- Suppose we move a_1, a_2, …, a_k from left to right, and then move *b* from right to left ⍰ At most only moving a_1, a_2, …, a_k and *b* need updating!

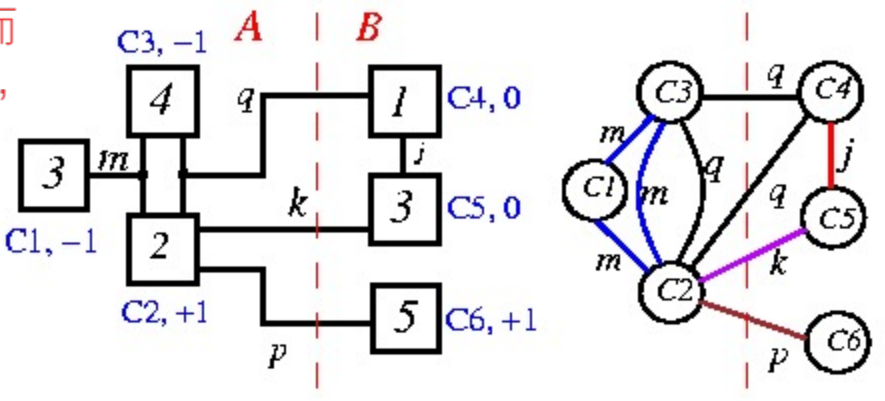


- To update the cell gains, it takes $O(n(i))$ work for Net *i*.
- Total time = $n(1)+n(2)+\ldots+n(N) = O(P)$.

cell內的數字
為cell的size

F-M Heuristic: An Example

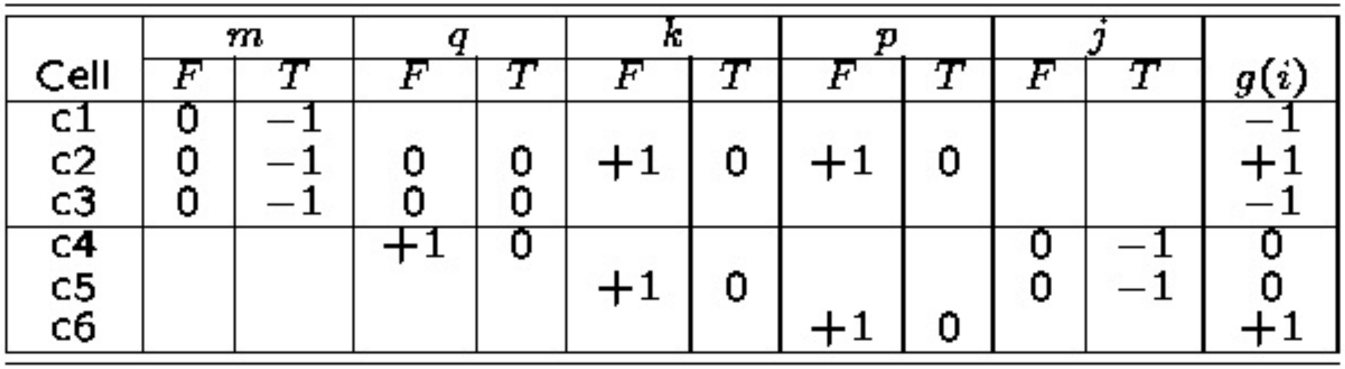
以C1為例, C1連接net m, 而 net m連接的所有cell都在A, 所以net m連到的所有cells 移過去都會賠1 => T = -1



(F = from, 移過去賺 = 1,
T = to, 移過去賠 = -1)

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

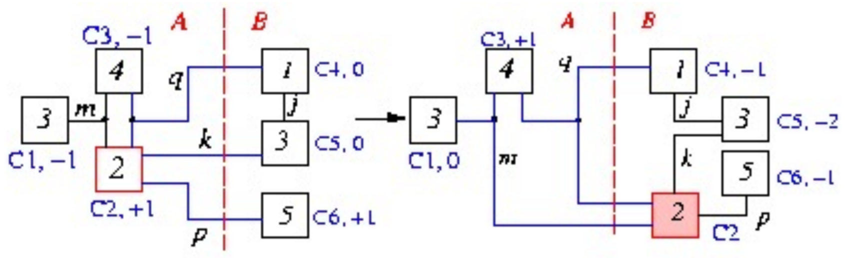
g(i)要根據所有的net得出來的值加起來
=> multiple net的cells



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

F-M Heuristic: An Example (cont'd)



- Changes in net distribution:

ex. C2移過去後, net k連到的cells原本從1F1T to 2T

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

update cell gains只要根據移動過去的cell有關的net distribution的改變去決定要update那些net連到的cells的gain就好

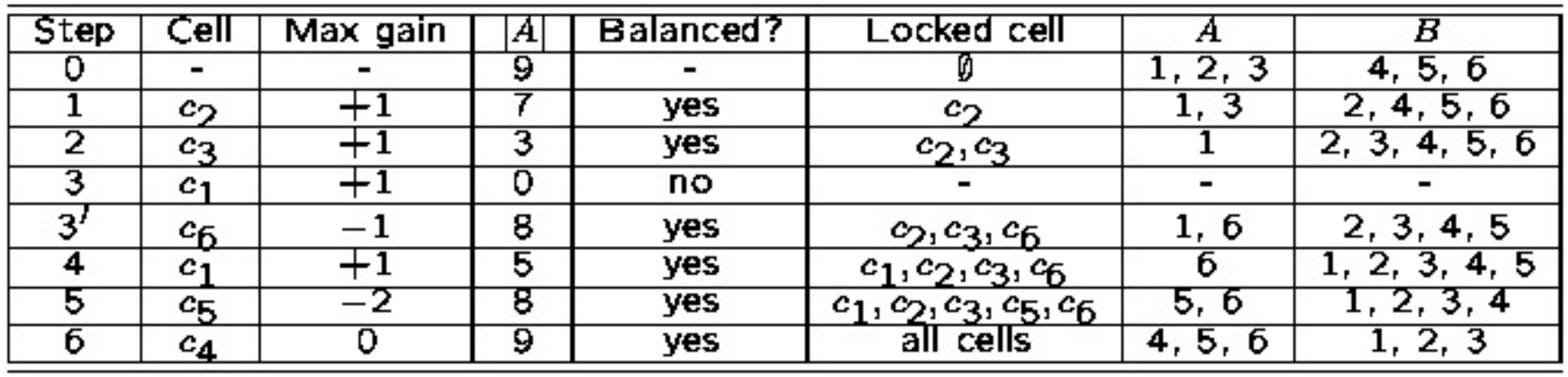
- Updating cell gains on critical nets (run Algorithm Update_Gain):

以net m為例: 原本三個都在F, T = 0
C2過去T後, 其他在F的gain都要+1

Cells	Gains due to $T(n)$				Gain due to $F(n)$				Gain changes	
	k	m	q	p	k	m	q	p	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

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

Summary of the Example

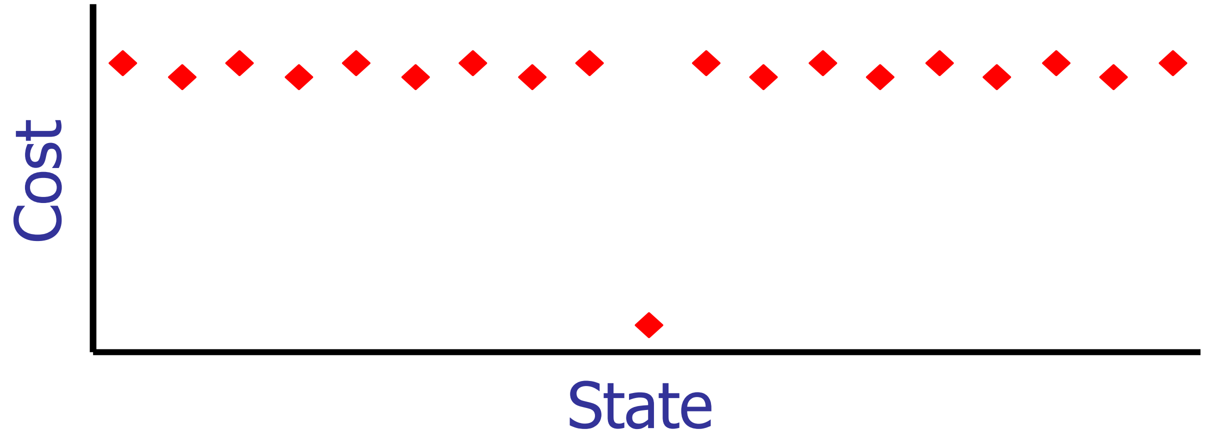


Step	Cell	Max gain	$\|A\|$	Balanced?	Locked cell	A	B
0	-	-	9	-	$\emptyset$	1, 2, 3	4, 5, 6
1	c_2	$+1$	7	yes	c_2	1, 3	2, 4, 5, 6
2	c_3	$+1$	3	yes	c_2, c_3	1	2, 3, 4, 5, 6
3	c_1	$+1$	0	no	-	-	-
3'	c_6	-1	8	yes	c_2, c_3, c_6	1, 6	2, 3, 4, 5
4	c_1	$+1$	5	yes	c_1, c_2, c_3, c_6	6	1, 2, 3, 4, 5
5	c_5	-2	8	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 partition ⍰ Move c_1, c_2, c_3, c_6 ⍰ A={6}, B={1, 2, 3, 4, 5}.
- **Repeat the whole process until new $G_k \leq 0$.**

Problem with Greedy Algorithms

- Easily get stuck at local minimum.
- Will obtain non-optimal solutions.



- Optimal only for convex (or concave for maximization) funtions.

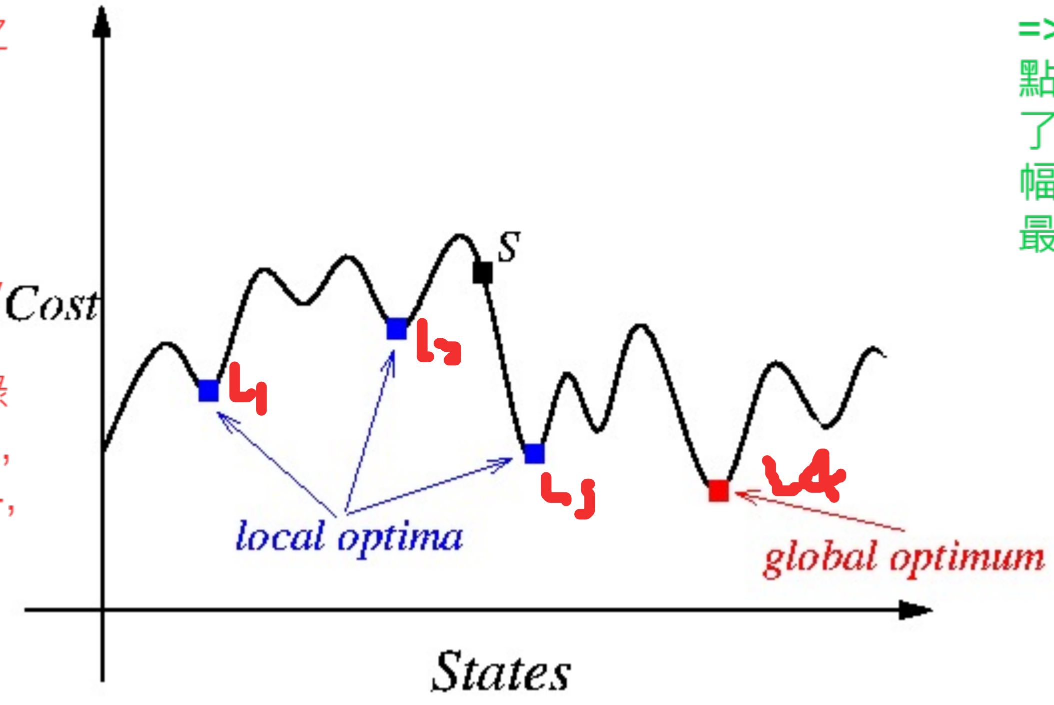
只有convex的問題用greedy才會
是optimal, ex. MST, shortest path,
partition是像下一頁的圖所以greedy會卡在local optimal

Simulated Annealing

- Kirkpatrick, Gelatt, and Vecchi, "Optimization by simulated annealing," *Science*, May 1983.

=> 從S開始做greedy(KL, FM)後往下到L3後, 會因為largest partial sum開始<=0, 就停止執行, 因此會卡在L3, not global optimal

SA - 讓起始點亂跳, 之後再開始執行greedy, 有機會達到global optimal, 但不保證.
例如S可能跳到L4旁的曲線, 就能在greedy達到L4.
從S開始後到L3會記錄下來, 直到找到更好的, 有可能亂跳時, 跳到L4, 又亂跳出來導致沒有辦法達到L4



=>一開始S亂跳去找起始點的幅度可以很大, 但停了開始run之後, 下降的幅度就變小, 以期望找到最佳解

=>一開始溫度高, 往較差的地方跑的prob都可以接受, 因此S可跳脫目前local optima並跑到其他的點, 之後溫度越來越低, 能跑去其他點的機率就減少

Simulated Annealing Basics

- Non-zero probability for “up-hill” moves. =>有機會接受"往上", 不一定是往下走
- Probability depends on
  1. magnitude of the “up-hill” movement
  2. total search time

delta C <= 0, 代表cut數減少, 此時prob = 1, 代表都會接受讓它發生. delta C >= 0, 則去計算, T越大, delta C越小, 越容易接受

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

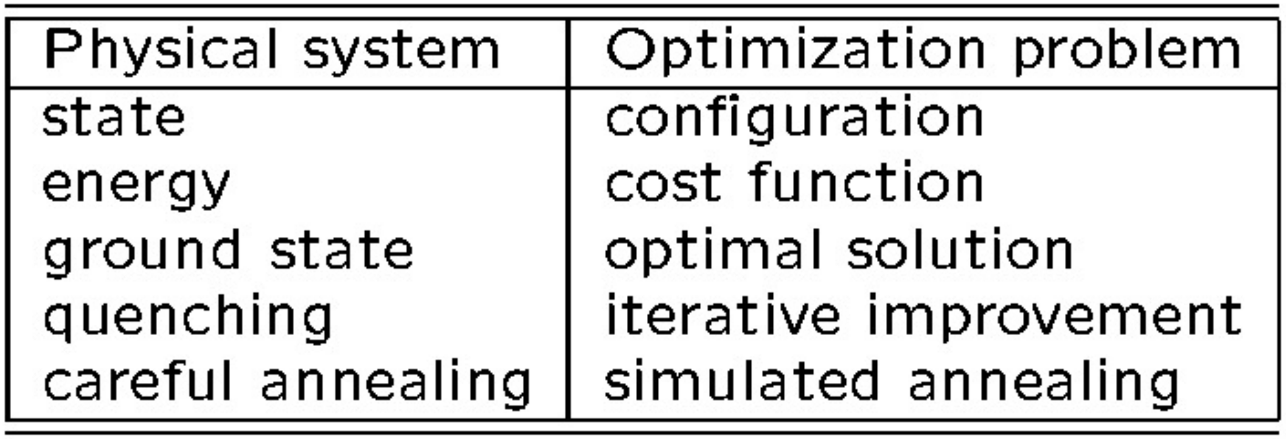
- $\Delta C = cost(S') - Cost(S)$
- T: Control parameter (temperature)
- Annealing schedule: $T=T_0, T_1, T_2, \ldots$, where $T_i = r^i T_0$, $r < 1$.

Generic Simulated Annealing Algorithm

```
1 begin
2 Get an initial solution $S$;
3 Get an initial temperature $T > 0$;
4 while not yet "frozen" do
5     for $1 \le i \le P$ do
6         Pick a random neighbor $S'$ of $S$;
7         $\Delta \leftarrow cost(S') - cost(S)$;
          /* downhill move */
8         if $\Delta \le 0$ then $S \leftarrow S'$
          /* uphill move */
9         if $\Delta > 0$ then $S \leftarrow S'$ with probability $e^{-\frac{\Delta}{T}}$ ;
10 $T \leftarrow rT$;  /* reduce temperature */
11 return $S$
12 end
```

Basic Ingredients for Simulated Annealing

- Analogy:



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**(solution的曲線)
  - **Neighborhood structure**(往下走或往上走的下個位置)
  - **Cost function**(前後cost的差異)
  - **Annealing schedule**(根據一開始給的溫度, 讓S去亂跳, 算delta, 並依據cost function跟去算probability來決定要不要從這, 重複幾次之後reduce溫度, 並繼續重複直到溫度冷卻)