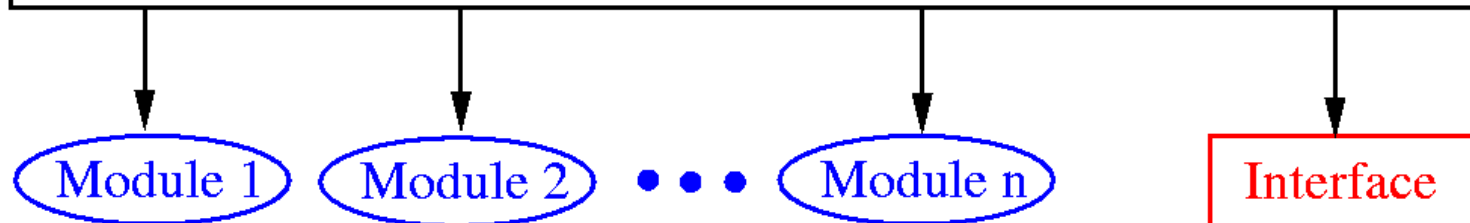


Partitioning

system design

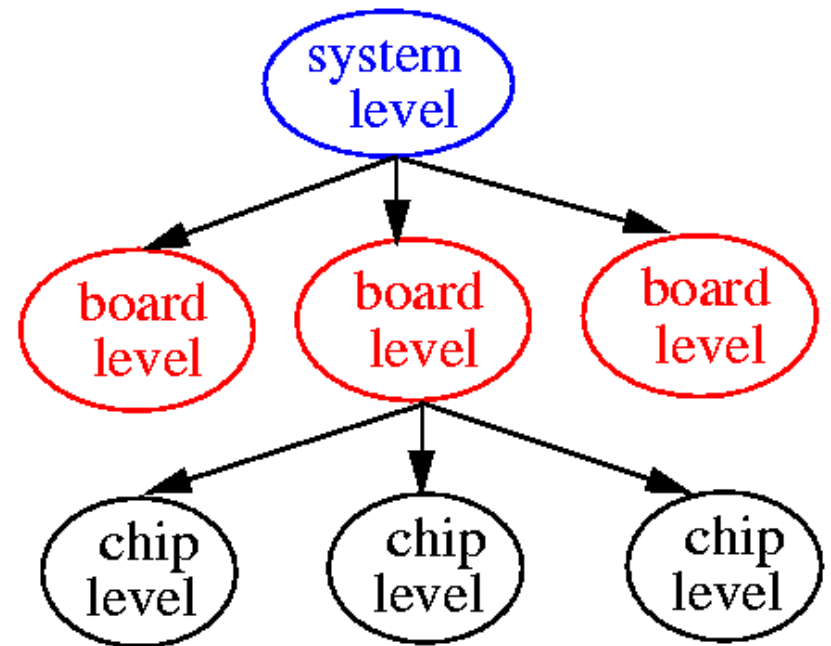
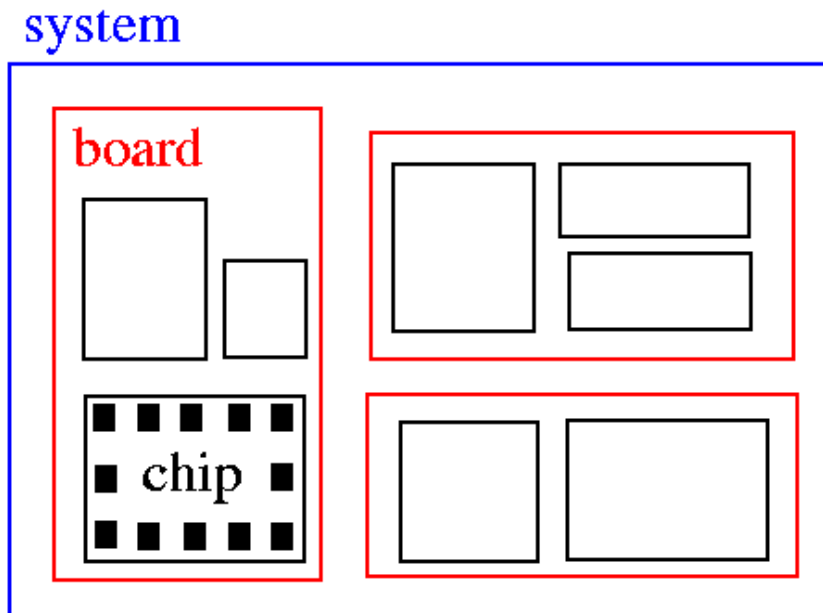


- Decomposition of a complex system into smaller subsystems.
- Each subsystem can be designed independently speeding up the design process.
- Decomposition scheme has to minimize the interconnections among the subsystems.
- Decomposition is carried out hierarchically until each subsystem is of manageable size.



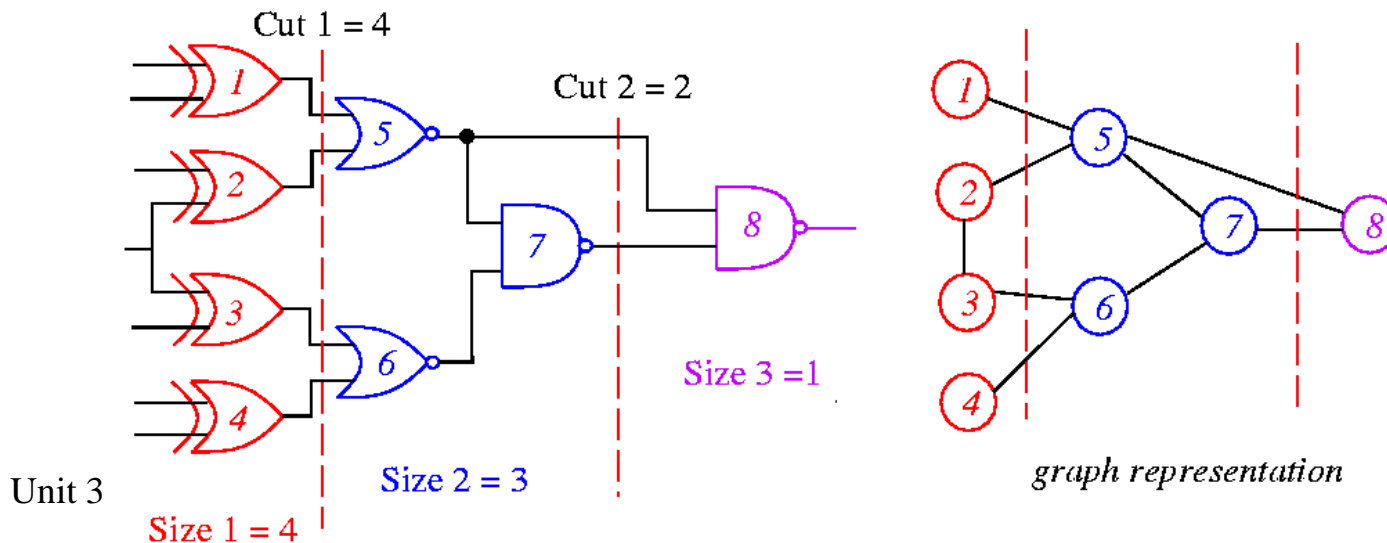
Levels of Partitioning

- The levels of partitioning: System, Board, Chip.



Partitioning of a Circuit

- The task of cutting a circuit into smaller parts.
- **Objective:** Partition the circuit into sub-circuits such that every sub-circuit is within a pre-specified size and the # of connections among sub-circuits is minimized.
 - Other possible constraints (e.g., # of pins in a sub-circuit)
 - Other possible objectives (e.g., critical path delay)
- Cutset? Cut size? Size of a sub-circuit?



Problem Definition

- **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, \dots, 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 L_i and U_i (i.e., $L_i \leq \sum_{v \in V_i} s(v) \leq U_i$)
- **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 subset where u is.
- The 2-way, size-constrained partitioning problem is NP-hard, even in its simple form with identical vertex sizes and unit edge weights.

Kernighan-Lin (KL) Algorithm

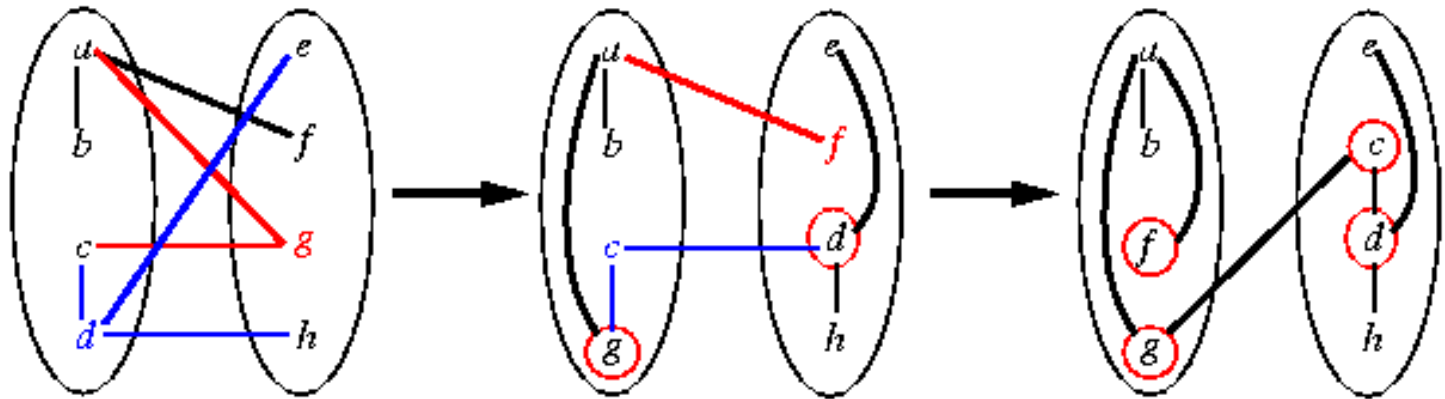
- Kernighan and Lin, “An efficient heuristic procedure for partitioning graphs,” *The Bell System Technical Journal*, vol. 49, no. 2, Feb. 1970.
- An **iterative, 2-way, balanced** partitioning (bi-sectioning) heuristic.
- Restrictions:
 - Assume all vertices are of the same size.
 - Work only for 2-terminal nets.

Key Idea of KL Algorithm

- Start with any initial partitions A and B .
- A **pass** (exchanging each vertex exactly once) consists of:
 - Exchange a vertex pair which gives the **maximum gain** g_i (i.e., largest decrease or smallest increase in cut size), and **lock** them (which thus are prohibited from participating in any further exchanges).
 - This process continues until all vertices are locked.
 - Find the largest partial sum G (i.e., find k such that $g_1 + \dots + g_k$ is maximized).
 - Exchange the first k pairs.
- Repeat the pass until there is no improvement (i.e., $G=0$).

KL 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

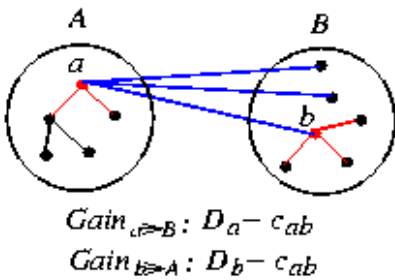
- Questions: How to compute cost reduction? What pairs to be swapped?

Properties

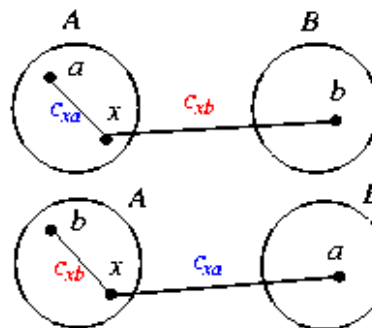
- Two sets A and B such that $|A|=n=|B|$ and $A \cap B = \emptyset$.
- External cost** of $a \in A$: $E_a = \sum_{v \in B} c_{av}$.
- Internal cost** of $a \in A$: $I_a = \sum_{v \in A} c_{av}$.
- D -value of a vertex a : $D_a = E_a - I_a$ (cost reduction for moving a).
- Cost reduction (**gain**) for swapping a and b : $g_{ab} = D_a + D_b - 2c_{ab}$.
- If $a \in A$ and $b \in B$ are interchanged, then the new D -values, D' , are given by

$$D'_x = D_x + 2c_{xa} - 2c_{xb}, \forall x \in A - \{a\}$$

$$D'_y = D_y + 2c_{yb} - 2c_{ya}, \forall y \in B - \{b\}.$$



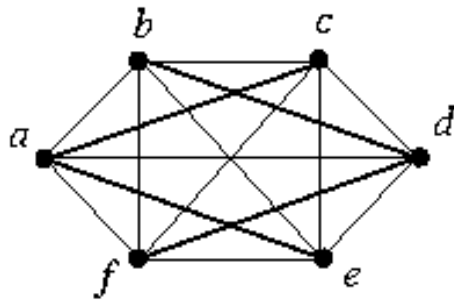
Internal cost vs. External cost



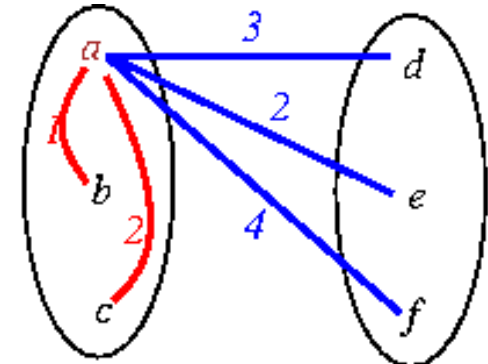
updating D -values

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

KL Algorithm: 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



costs associated with a

$$\text{Initial cut cost} = (3+2+4) + (4+2+1) + (3+2+1) = 22$$

- Iteration 1:

$$I_a = 1 + 2 = 3; \quad E_a = 3 + 2 + 4 = 9; \quad D_a = E_a - I_a = 9 - 3 = 6$$

$$I_b = 1 + 1 = 2; \quad E_b = 4 + 2 + 1 = 7; \quad D_b = E_b - I_b = 7 - 2 = 5$$

$$I_c = 2 + 1 = 3; \quad E_c = 3 + 2 + 1 = 6; \quad D_c = E_c - I_c = 6 - 3 = 3$$

$$I_d = 4 + 3 = 7; \quad E_d = 3 + 4 + 3 = 10; \quad D_d = E_d - I_d = 10 - 7 = 3$$

$$I_e = 4 + 2 = 6; \quad E_e = 2 + 2 + 2 = 6; \quad D_e = E_e - I_e = 6 - 6 = 0$$

$$I_f = 3 + 2 = 5; \quad E_f = 4 + 1 + 1 = 6; \quad D_f = E_f - I_f = 6 - 5 = 1$$

Weighted Example (Cont'd)

- Iteration 1:

$$I_a=1+2=3; \quad E_a=3+2+4=9; \quad D_a=E_a-I_a=9-3=6$$

$$I_b=1+1=2; \quad E_b=4+2+1=7; \quad D_b=E_b-I_b=7-2=5$$

$$I_c=2+1=3; \quad E_c=3+2+1=6; \quad D_c=E_c-I_c=6-3=3$$

$$I_d=4+3=7; \quad E_d=3+4+3=10; \quad D_d=E_d-I_d=10-7=3$$

$$I_e=4+2=6; \quad E_e=2+2+2=6; \quad D_e=E_e-I_e=6-6=0$$

$$I_f=3+2=5; \quad E_f=4+1+1=6; \quad D_f=E_f-I_f=6-5=1$$

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

$$g_{ad} = D_a + D_d - 2c_{ad} = 6 + 3 - 2 * 3 = 3$$

$$g_{ae} = 6 + 0 - 2 * 2 = 2$$

$$g_{af} = 6 + 1 - 2 * 4 = -1$$

$$g_{bd} = 5 + 3 - 2 * 4 = 0$$

$$g_{be} = 5 + 0 - 2 * 2 = 1$$

$$g_{bf} = 5 + 1 - 2 * 1 = 4 \text{ (maximum)}$$

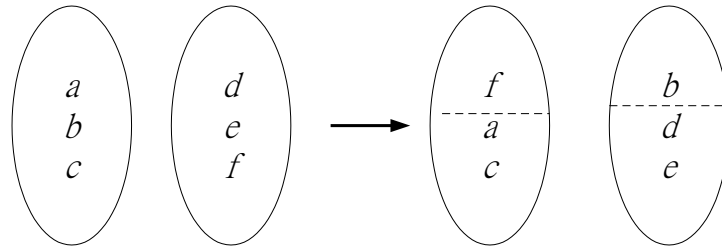
$$g_{cd} = 3 + 3 - 2 * 3 = 0$$

$$g_{ce} = 3 + 0 - 2 * 2 = -1$$

$$g_{cf} = 3 + 1 - 2 * 1 = 2$$

- Swap b and f ! ($\tilde{g}_1=4$)

Weighted Example (Cont'd)



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

$$D'_a = D_a + 2c_{ab} - 2c_{af} = 6 + 2*1 - 2*4 = 0$$

$$D'_c = D_c + 2c_{cb} - 2c_{cf} = 3 + 2*1 - 2*1 = 3$$

$$D'_d = D_d + 2c_{df} - 2c_{db} = 3 + 2*3 - 2*4 = 1$$

$$D'_e = D_e + 2c_{ef} - 2c_{eb} = 0 + 2*2 - 2*2 = 0$$

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

$$g_{ad} = D'_a + D'_d - 2c_{ad} = 0 + 1 - 2*3 = -5$$

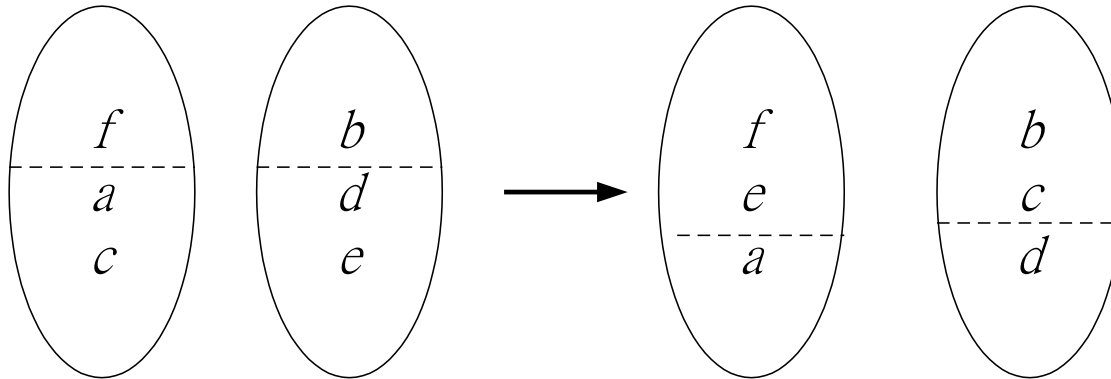
$$g_{ae} = D'_a + D'_e - 2c_{ae} = 0 + 0 - 2*2 = -4$$

$$g_{cd} = D'_c + D'_d - 2c_{cd} = 3 + 1 - 2*3 = -2$$

$$g_{ce} = D'_c + D'_e - 2c_{ce} = 3 + 0 - 2*2 = -1 \text{ (maximum)}$$

- Swap c and $e!$ ($g_{ce} = -1$)

Weighted Example (Cont'd)



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

$$D_a'' = D_a' + 2c_{ac} - 2c_{ae} = 0 + 2*2 - 2*2 = 0$$

$$D_d'' = D_d' + 2c_{de} - 2c_{dc} = 1 + 2*4 - 2*3 = 3$$

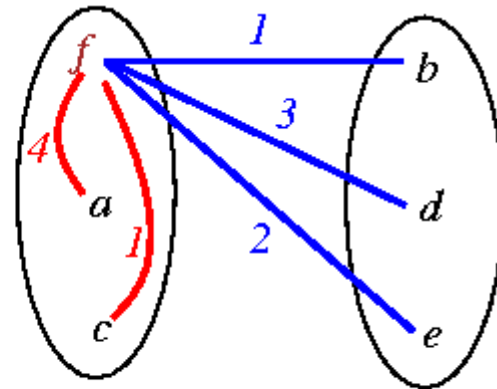
- $g_{xy} = D_x'' + D_y'' - 2x_{xy}$

$$g_{ad} = D_a'' + D_d'' - 2c_{ad} = 0 + 3 - 2*3 = -3 \quad (\hat{g}_3 = -3)$$

- 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 $\max \sum_{i=1}^k \hat{g}_i = 4 \quad (k = 1) \Rightarrow \text{Swap } b \text{ and } f$.

Weighted Example (Cont'd)

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
<i>a</i>	0	1	2	3	2	4
<i>b</i>	1	0	1	4	2	1
<i>c</i>	2	1	0	3	2	1
<i>d</i>	3	4	3	0	4	3
<i>e</i>	2	2	2	4	0	2
<i>f</i>	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: $\hat{g}_1 = g_{ce} = -1$, $\hat{g}_2 = g_{ab} = -3$, $\hat{g}_3 = g_{fd} = 4$.
- Largest partial sum = $\max \sum_{i=1}^k \hat{g}_i = 0$ ($k = 3$) \Rightarrow Stop!

Algorithm: Kernighan-Lin(G)

Input: $G=(V,E), |V|=2n$.

Output: Balanced bi-partition A and B with “small” cut cost.

```
1 begin
2 Bi-partition  $G$  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  $\hat{g}_i$ , and compute the new  $D_v$ , for all
       unlocked  $v \in V$ ;
8     Find  $k$ , such that  $G_k = \sum_{i=1}^k \hat{g}_i$  is maximized;
9     if  $G_k > 0$  then
10      Move  $v_{ai}, \dots, v_{ak}$  from  $V_A$  to  $V_B$  and  $v_{bi}, \dots, v_{bk}$  from  $V_B$  to  $V_A$ ;
11      Unlock  $v, \forall v \in V$ .
12 until  $G_k \leq 0$ ;
13 end
```

Time Complexity

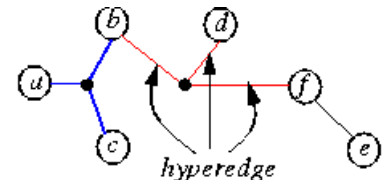
- Line 4: Initial computation of D : $O(n^2)$
- Line 5: The **for**-loop: $O(n)$
- The body of the loop: $O(n^2)$
 - Lines 6-7: Step i takes $O(n-i+1)^2$ time.
- Lines 4-11: Each pass of the repeat loop: $O(n^3)$.
- Suppose the repeat loop terminates after r passes.
- The total running time: $O(rn^3)$.

Extension of KL Algorithm

- k -way partitioning
 1. Partition the graph into k equal-sized sets. Apply the KL algorithm for each pair of subsets.
 2. Apply the KL algorithm recursively.

Drawbacks of KL Algorithm

- KL 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.
- KL handles only exact bisections.
 - Need dummy vertices to handle the unbalanced problem.
- KL cannot handle hypergraphs.
 - A hypergraph consists of a set of vertices and a set of hyperedges, where each hyperedge e_i corresponds to a subset N_i of distinct vertices with $|N_i| \geq 2$
- The time complexity of a pass is high, $O(n^3)$.

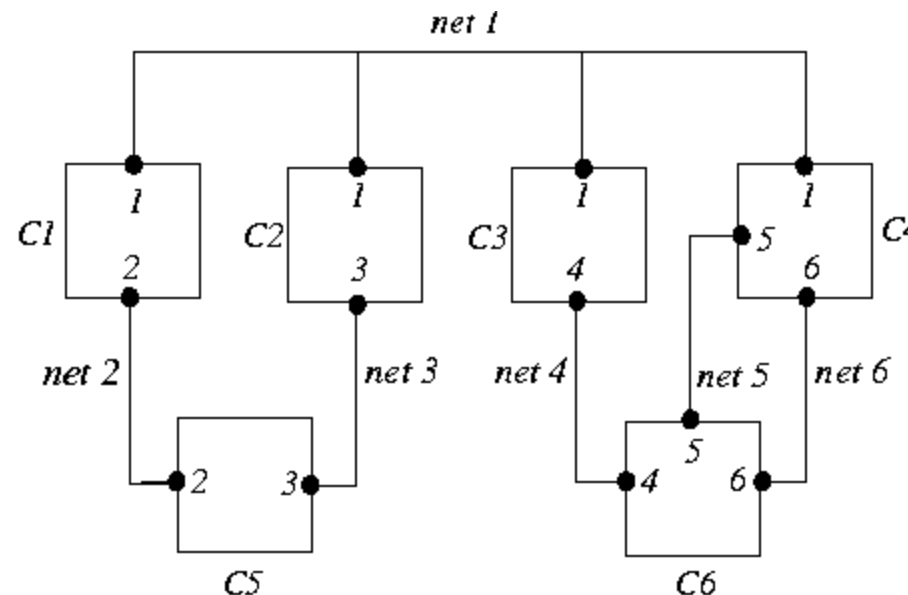


Fiduccia-Mattheyses (FM) Algorithm

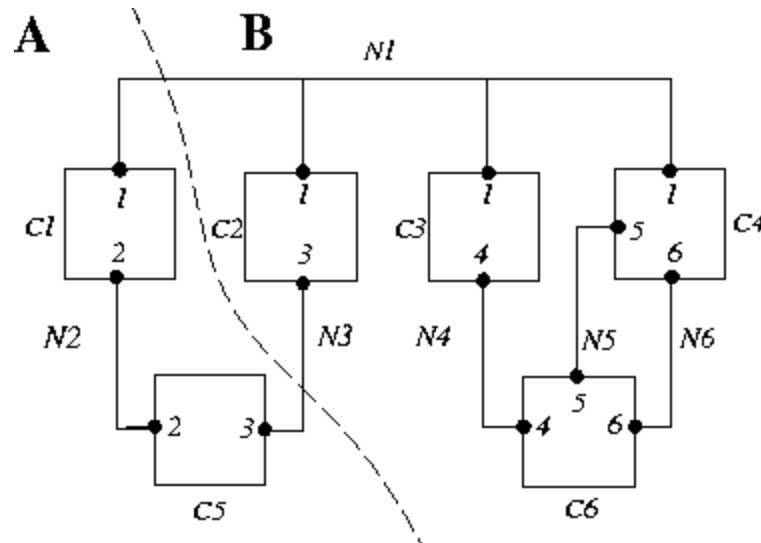
- Fiduccia and Mattheyses. “A linear time heuristic for improving network partitions,” 19th Design Automation Conf., 1982.
- Same as KL:
 - Work in passes.
 - Lock vertices after moved.
 - Actually, only move those vertices up to the maximum partial sum of gain.
- Different from KL:
 - Aim at **reducing net-cut costs**; the concept of cut size is extended to hypergraphs.
 - Only a **single vertex** is moved across the cut in a 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 of a pass is $O(P)$** , where P is the total # of pins.

FM: Notation

- $n(i)$: # of cells in Net i ; e.g., $n(1)=4$.
- $s(i)$: size of Cell i .
- $p(i)$: # of pins 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; $P = p(1)+\dots+p(C) = n(1)+\dots+n(N)$.

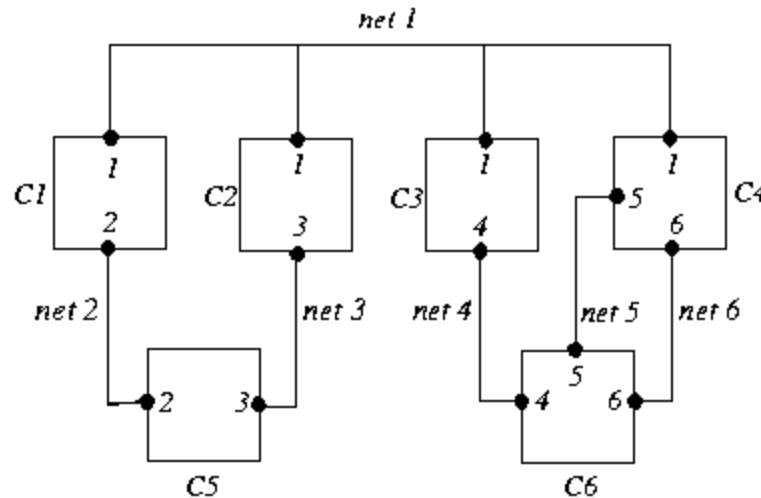


Cut



- **Cutstate** of a net:
 - Net 1 and Net 3 are **cut**.
 - 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 partitioning**: 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

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

Basic Ideas: Balance and Movement

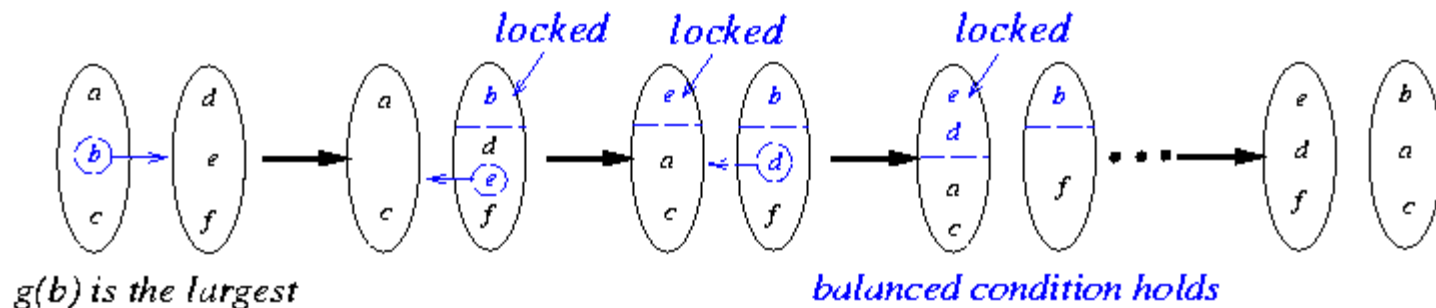
- Only move a cell at a time, preserving “balance.”

$$\frac{|A|}{|A| + |B|} \approx r$$

$$rW - S_{\max} \leq |A| \leq rW + S_{\max},$$

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

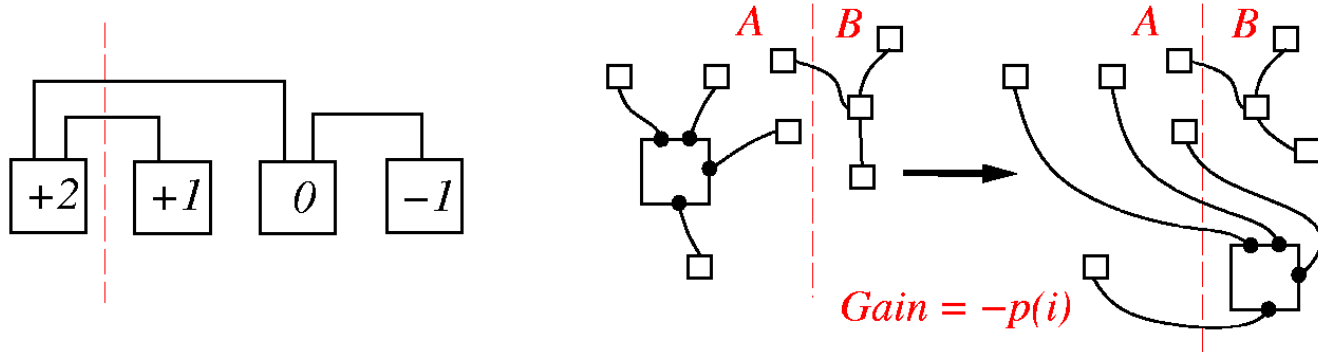
- $g(i)$: gain in moving cell i to the other set,
i.e., size of **old** cutset - size of **new** cutset.



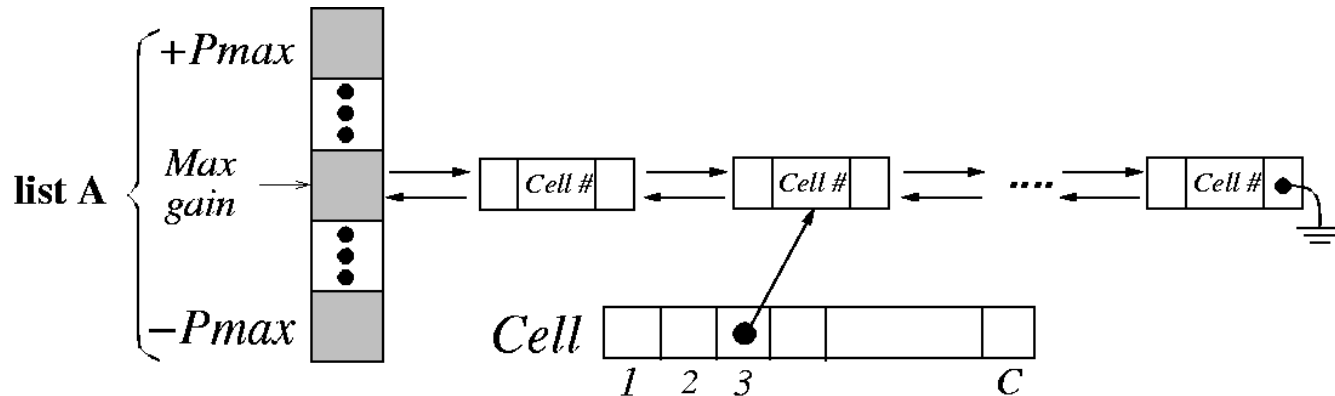
- Suppose \hat{g}_i 's: $g(b)$, $g(e)$, $g(d)$, $g(a)$, $g(f)$, $g(c)$ and the largest partial sum is $g(b) + g(e) + g(d)$. Then we should move $b, e, d \rightarrow$ resulting two sets: $\{a, c, e, d\}, \{b, f\}$

Cell Gains and Data Structure Manipulation

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



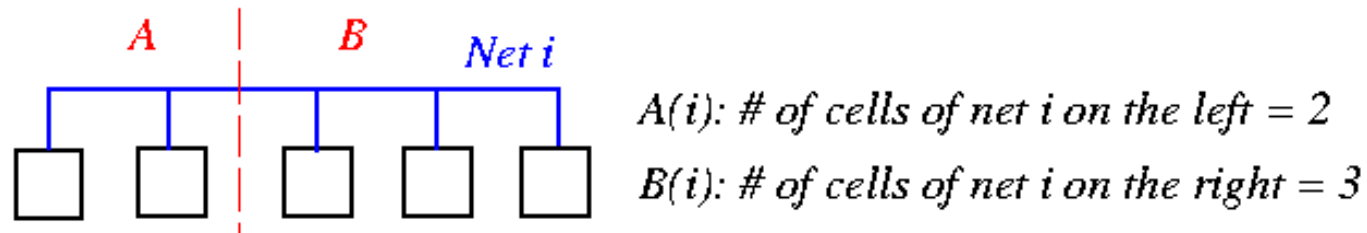
- Two “bucket list” structures, one for set A and one for set B ($P_{\max} = \max_i p(i)$).



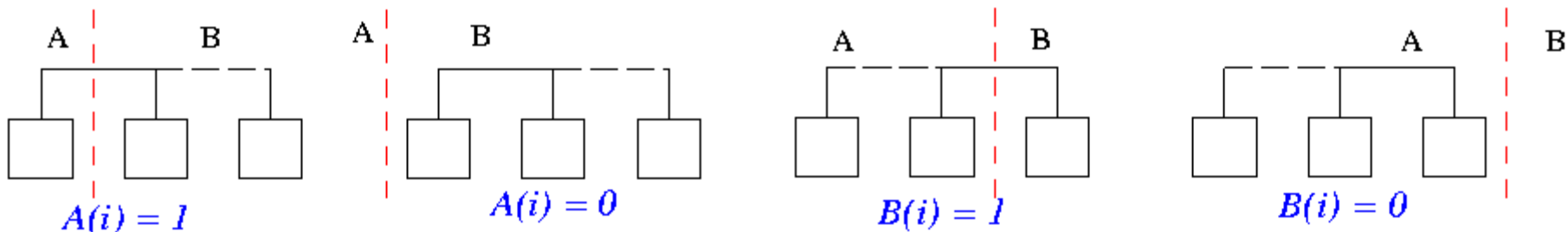
- $O(1)$ -time operations: find a cell with Max Gain, remove Cell i from the structure, insert Cell i into the structure, update $g(i)$ to $g(i) + \Delta$, update the Max Gain pointer.

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.



- **Critical Nets:** A net is critical if it has a cell which if moved will change its cutstate.
 - 4 cases: $A(i) = 0$ or 1, $B(i) = 0$ or 1.



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

Computing Cell Gains

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

$g(i) \leftarrow 0$;

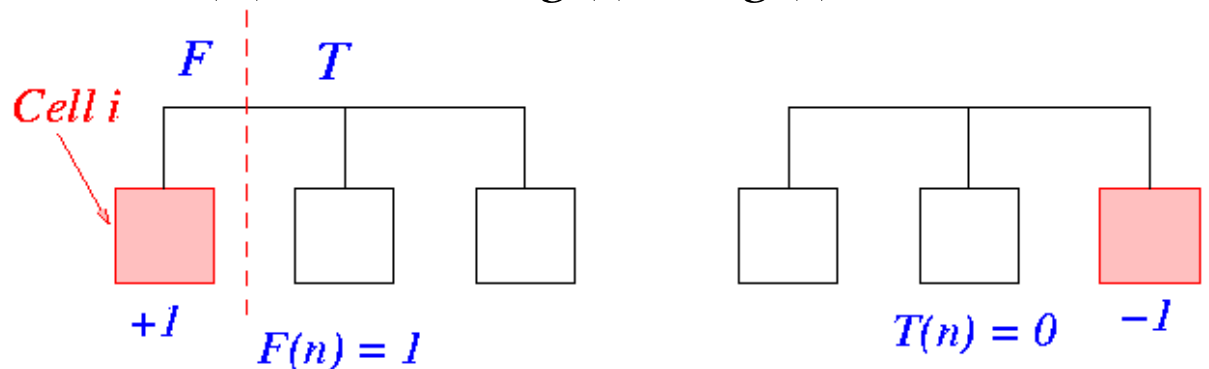
$F \leftarrow$ the “from block” of cell i ;

$T \leftarrow$ the “to block” of cell i ;

for each net n on Cell i **do**

if $F(n) = 1$ **then** $g(i) \leftarrow g(i) + 1$;

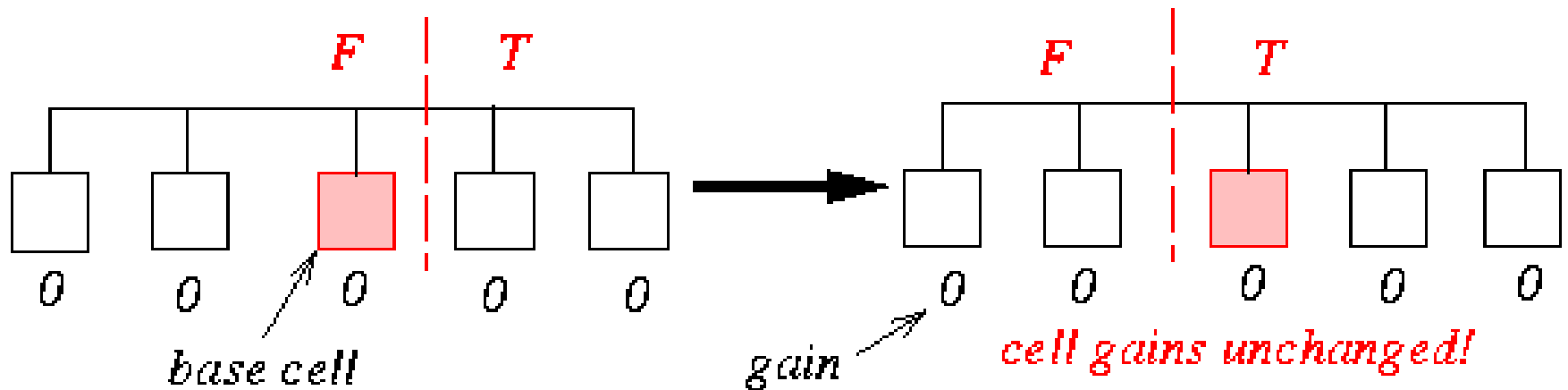
if $T(n) = 0$ **then** $g(i) \leftarrow g(i) - 1$;



- 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.

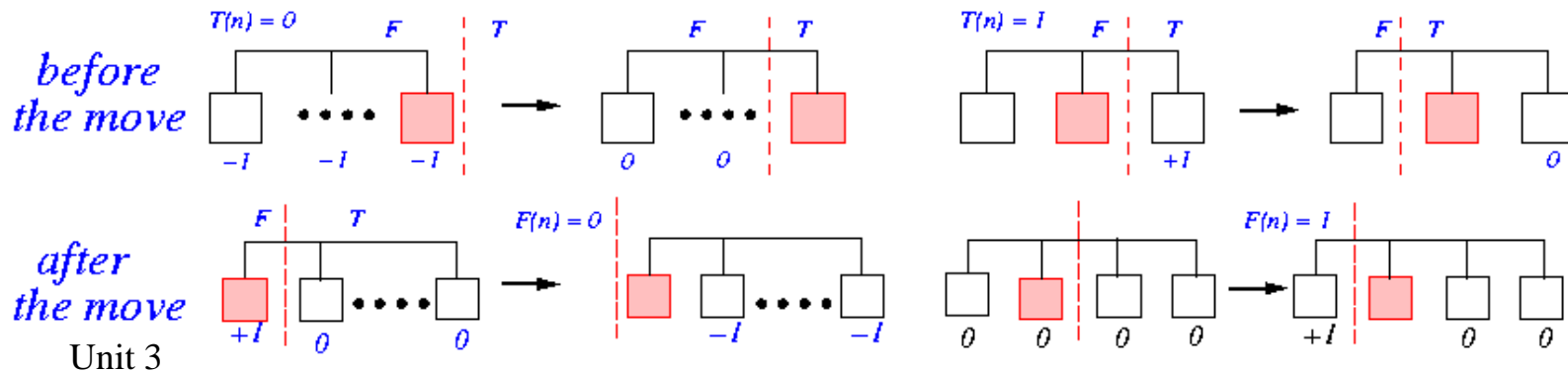


Algorithm for Updating Cell Gains

Algorithm: Update_Gain

```

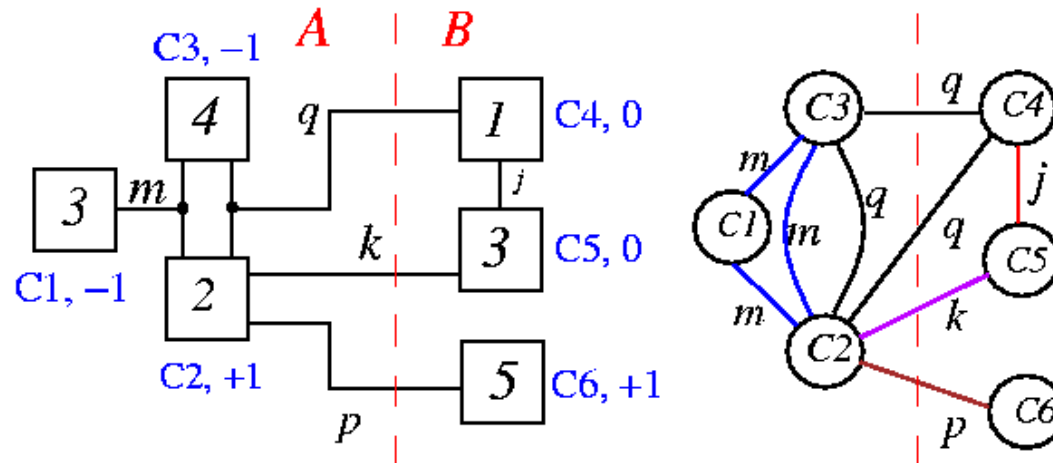
1 begin /* move base cell and update neighbors' gains */
2  $F \leftarrow$  the Front Block of the base cell;
3  $T \leftarrow$  the To Block of the base cell;
4 Lock the base cell and complement its block;
5 for each net  $n$  on the base cell do
    /* check critical nets before the move */
6   if  $T(n) = 0$  then increment gains of all free cells on  $n$ 
   elseif  $T(n) = 1$  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  $n$ 
   elseif  $F(n) = 1$  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.
- To update the cell gains, it takes $O(n(i))$ work for Net i .
- Total time = $n(1) + n(2) + \dots + n(N) = O(P)$

FM: An Example

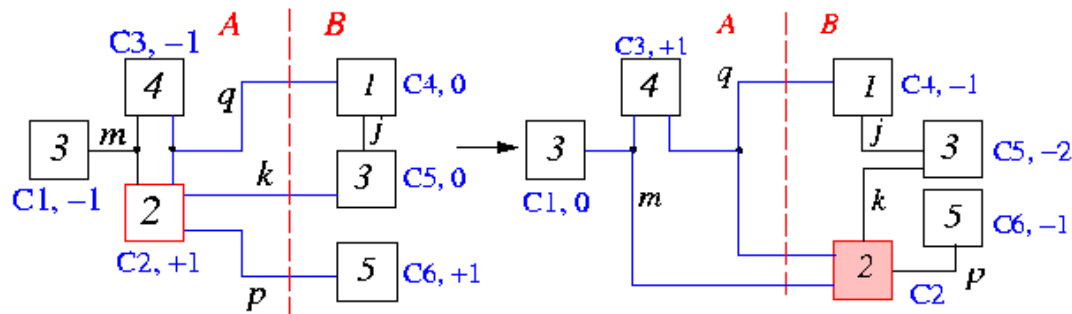


- Computing cell gains: $F(n) = 1 \Rightarrow g(i) + 1$; $T(n) = 0 \Rightarrow g(i) - 1$

Cell	m		q		k		p		j		$g(i)$
	F	T	F	T	F	T	F	T	F	T	
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 $r = 0.4 \Rightarrow |A|=9$, $|V|=18$, $S_{max} = 5$, $r|V| = 7.2 \Rightarrow$ Balanced: $2.2 \leq 9 \leq 12.2$!
- Maximum gain: c_2 and balanced: $2.2 \leq 9-2 \leq 12.2 \Rightarrow$ Move c_2 from A to B (use size criterion if there is a tie).

FM: An Example (Cont'd)



- Changes in net distribution:

Net	Before move		After move	
	F	T	F'	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 $T(n)$				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! ($2.2 \leq 7-4 \leq 12.2$)

Unit 3 \Rightarrow Move c_3 from A to B (use size criterion if there is a tie).

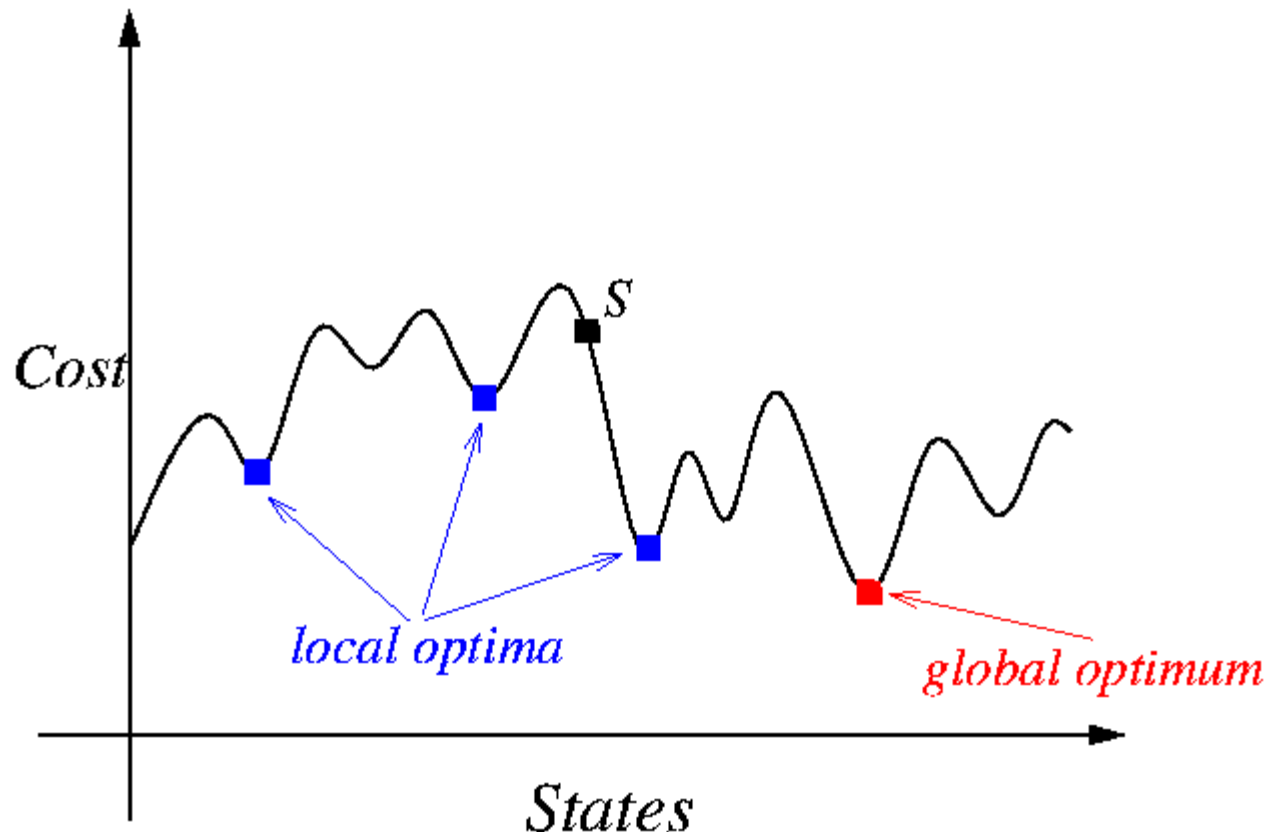
Summary of the Example

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

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

Simulated Annealing Revisited

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



Simulated Annealing Basics

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

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

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

Generic Simulated Annealing Algorithm

Algorithm: Simulated_Annealing

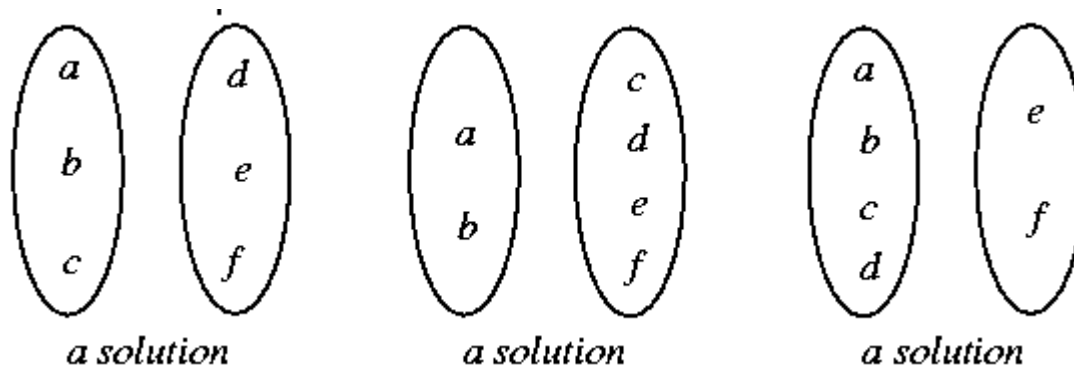
```
1 begin
2 Get an initial solution  $S$ ;
3 Get an initial temperature  $T > 0$ ;
4 while not yet “frozen” do
5   for  $1 \leq i \leq P$  do
6     Pick a random neighbor  $S'$  of  $S$ ;
7      $\Delta \leftarrow cost(S') - cost(S)$ ;
8     /* down hill move */
9     if  $\Delta \leq 0$  then  $S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

Basic Ingredients for Simulated Annealing

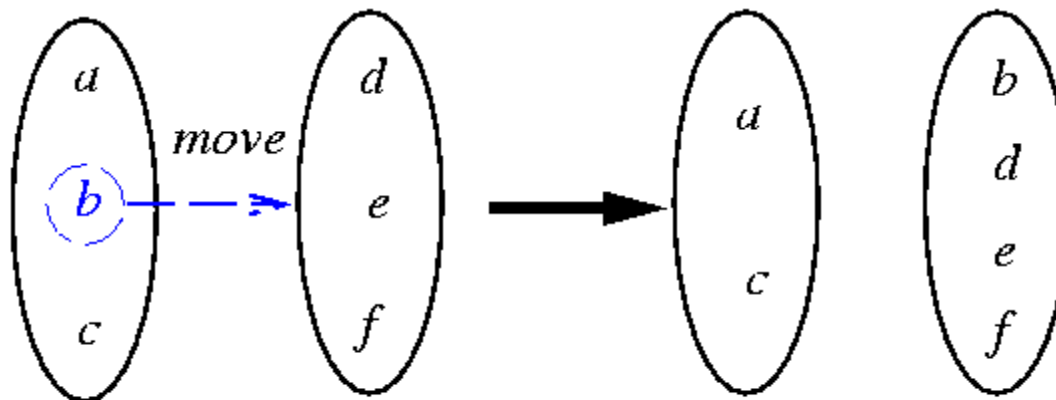
- Solution space
- Neighborhood structure
- Cost function
- Annealing schedule

Partitioning by Simulated Annealing

- Kirkpatrick, Gelatt, and Vecchi, “Optimization by simulated annealing,” *Science*, May 1983.
- **Solution space:** set of all partitioning solutions



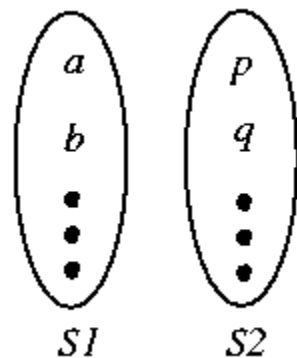
- **Neighborhood structure:**



Randomly move one cell to the other side

Partitioning by Simulated Annealing (cont'd)

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

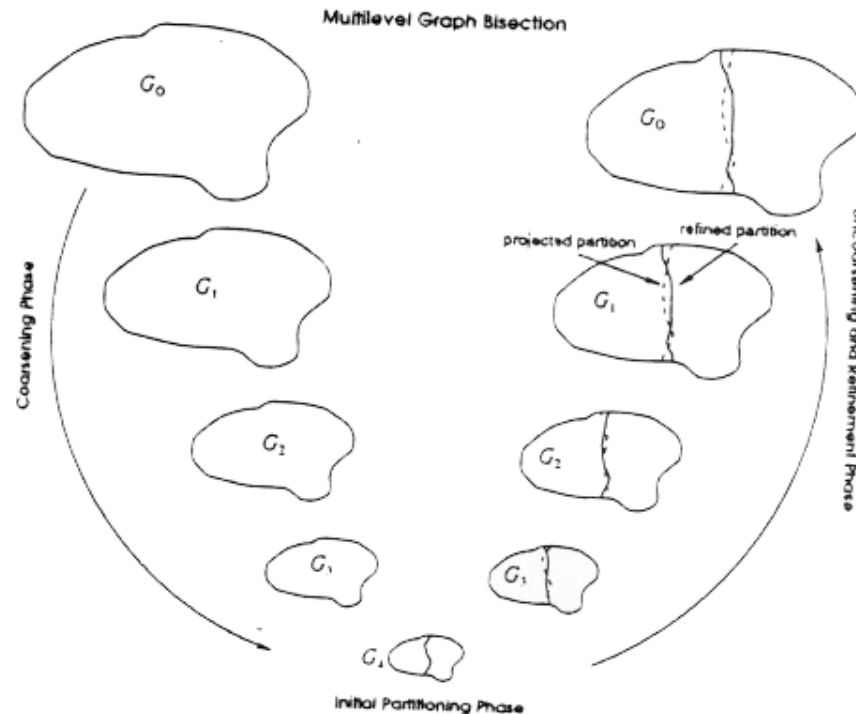


The diagram shows two vertical ovals representing sets $S1$ and $S2$. Set $S1$ contains elements a , b , and three dots. Set $S2$ contains elements p , q , and three dots. To the right of the ovals is the equation $B = (|S1| - |S2|)^2$.

- **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 ≥ 100 * total # of cells.
 - The system is “frozen” if very low acceptances at 3 consecutive temperatures.

Multilevel Partitioning

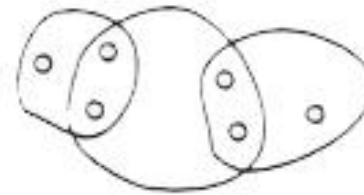
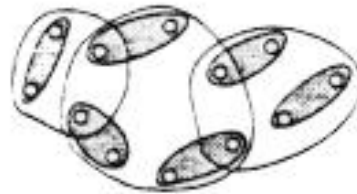
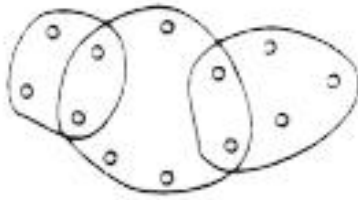
- **Three phases** (for bipartitioning)
 - **Coarsening**: construct a sequence of smaller (coarser) graphs.
 - **Initial partitioning**: construct a bipartitioning solution for the coarsest graph.
 - **Uncoarsening & refinement**: the bipartitioning solution is successively projected to the next-level finer graph, and at each level an iterative refinement algorithm (such as KL or FM) is used to further improve the solution.



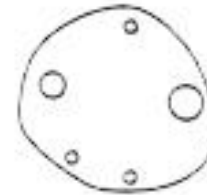
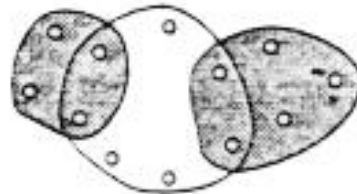
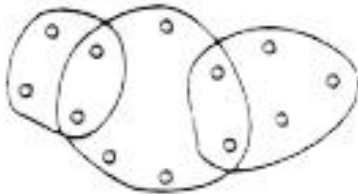
hMETIS

- Kayrpi, Aggarwal, Kumar and Shekhar, “Multilevel hypergraph partitioning: application in VLSI domain,” DAC, 1997.
- Three coarsening algorithms:
 - **Edge coarsening:** A maximal matching of the vertices.
 - **Hyperedge coarsening:** a set of hyperedges is selected, and the vertices belonging to a selected hyperedge are merged into a cluster. (Preference: hyperedges with large weights and hyperedges of small size.)
 - **Modified hyperedge coarsening:** hyperedge coarsening + merging the remaining vertices of each hyperedge into a cluster.

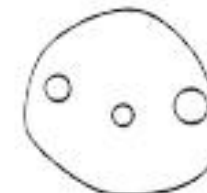
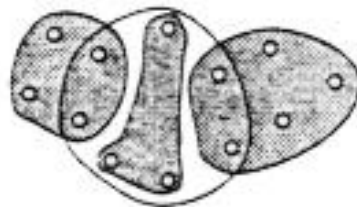
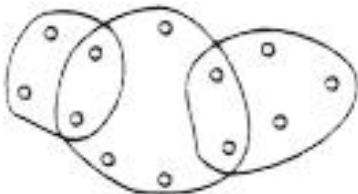
hMETIS (Cont'd)



(a) Edge Coarsening



(b) Hyperedge Coarsening



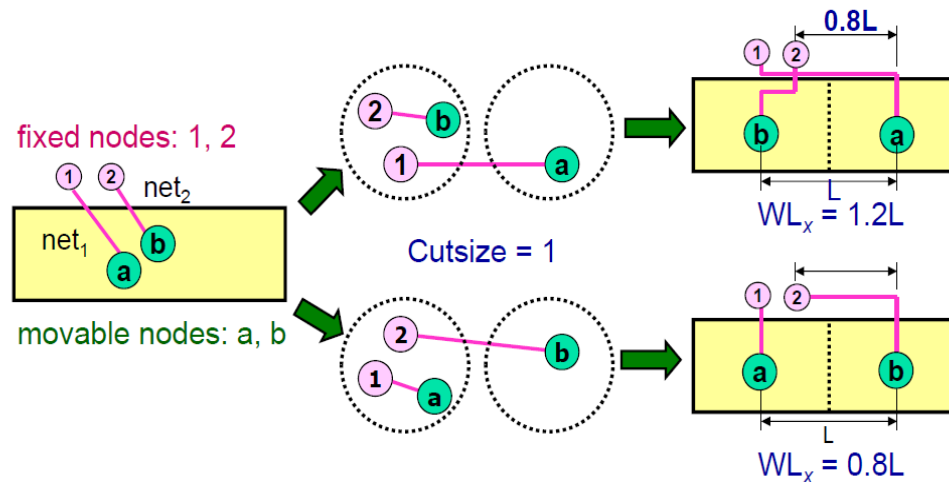
(c) Modified Hyperedge Coarsening

hMETIS (Cont'd)

- Two uncoarsening & refinement algorithms:
 - FM algorithm with modifications:
 - * Restrict the maximum number of passes to 2.
 - * Stop each pass when no improvement is made from the first k moves.
 - Hyperedge refinement: move groups of vertices between subsets so that an entire hyperedge is removed from the cut set.

Partitioning for Wirelength Minimization

- Chen, Chang, Lin, “IMF: Interconnection-driven floorplanning for large-scale building-module designs,” ICCAD-05
- Minimizing cut size is *not* equivalent to minimizing wirelength (WL)

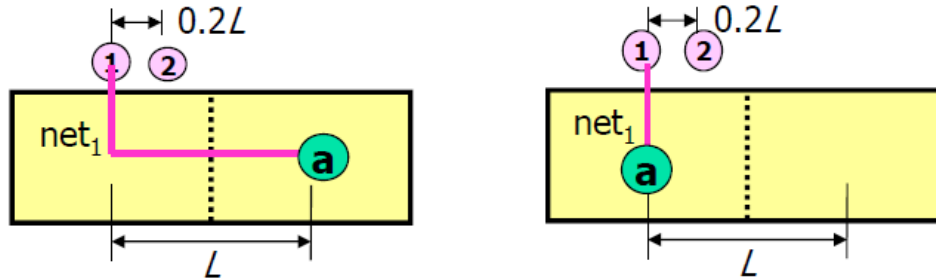


- Problem: **hyperedge weight is a constant value!**
 - Shall map the min-cut cost to wirelength (WL) change
 - Shall assign the hyperedge weight as the value of wirelength contribution if the hyperedge is cut

Net Weight Assignment

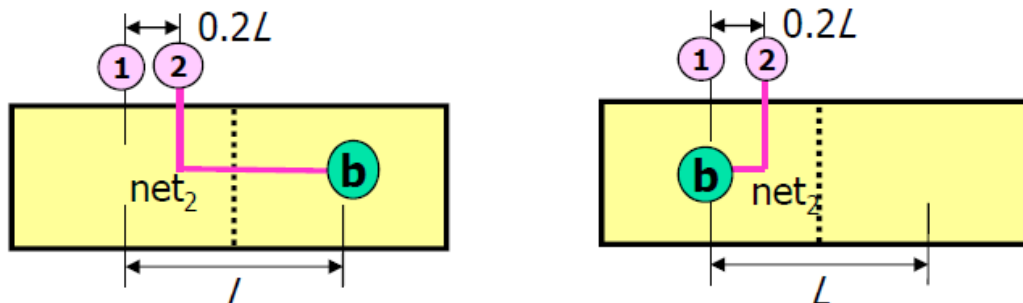
- net₁ connects a movable node *a* and a fixed node 1.

$$\begin{aligned}\text{Weight}(\text{net}_1) &= \text{WL}(\text{net}_1 \text{ is cut}) - \text{WL}(\text{net}_1 \text{ is not cut}) \\ &= L - 0L = L\end{aligned}$$

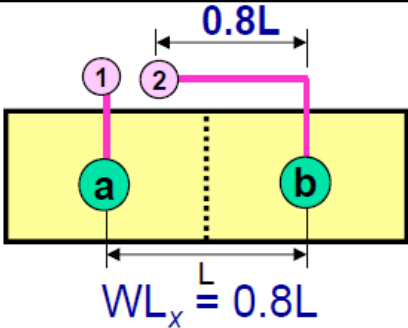
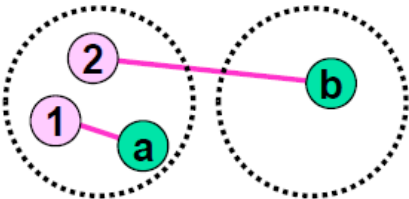
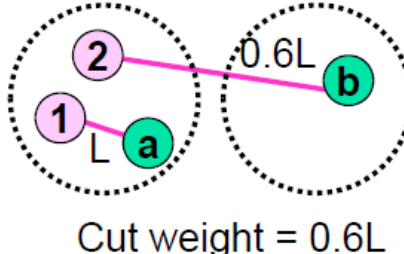
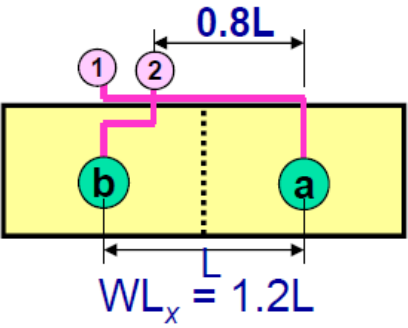
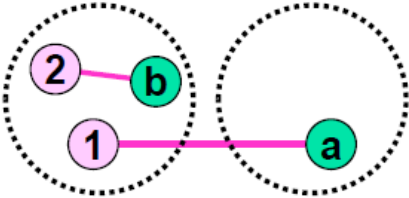
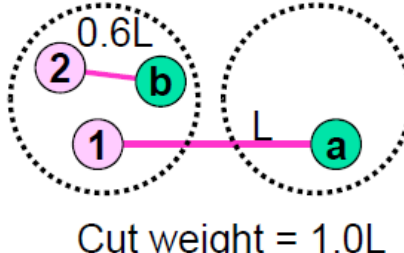


- net₂ connects a movable node *b* and a fixed node 2.

$$\begin{aligned}\text{Weight}(\text{net}_2) &= \text{WL}(\text{net}_2 \text{ is cut}) - \text{WL}(\text{net}_2 \text{ is not cut}) \\ &= 0.8L - 0.2L = 0.6L\end{aligned}$$



Examples

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

Cut weight is proportional to the wirelength (WL)

$$WL = \text{Cut weight} + 0.2L$$

($0.2L$ is the WL lower bound: placing a & b in the left side)

Relationship Between WL and Cut Weight

- Theorem: $WL_i = w_{1,i} + n_{cut,i}$
 - $n_{cut,i}$: cut weight for net i
 - $w_{1,i}$: the wirelength lower bound for net i

- Then, we have

Constant

$$\min(\sum WL_i) = \min(\sum (w_{1,i} + n_{cut,i})) = \underbrace{\sum w_{1,i}}_{\text{Constant}} + \min(\sum n_{cut,i})$$

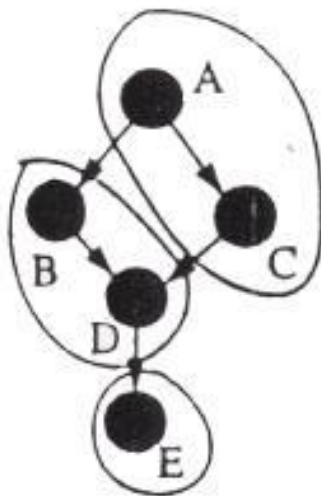
Finding the minimum wirelength is equivalent to finding the cut weight

Clustering for Delay Minimization

- Allow gate duplication.
- Gate duplication may help reduce delay.

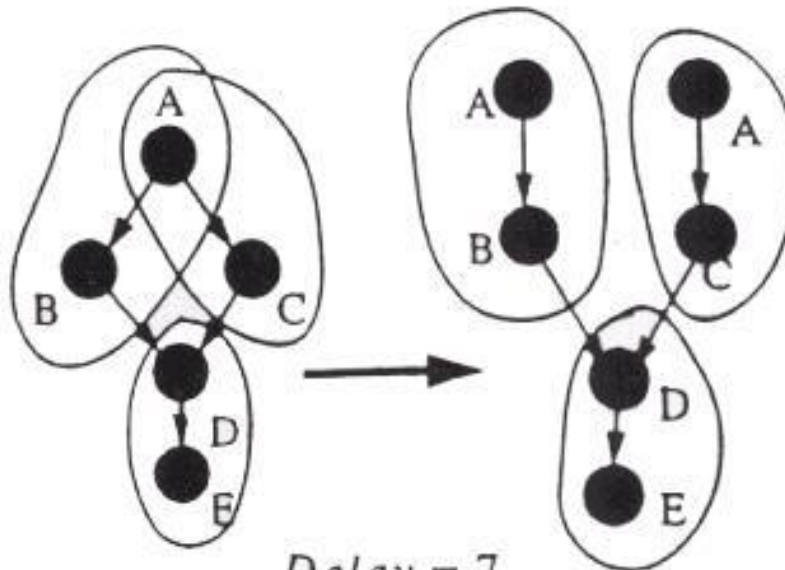
$D=3; M=2; \delta(v)=1, w(v)=1$, for each v .

Without gate duplication



Delay = 10

With gate duplication

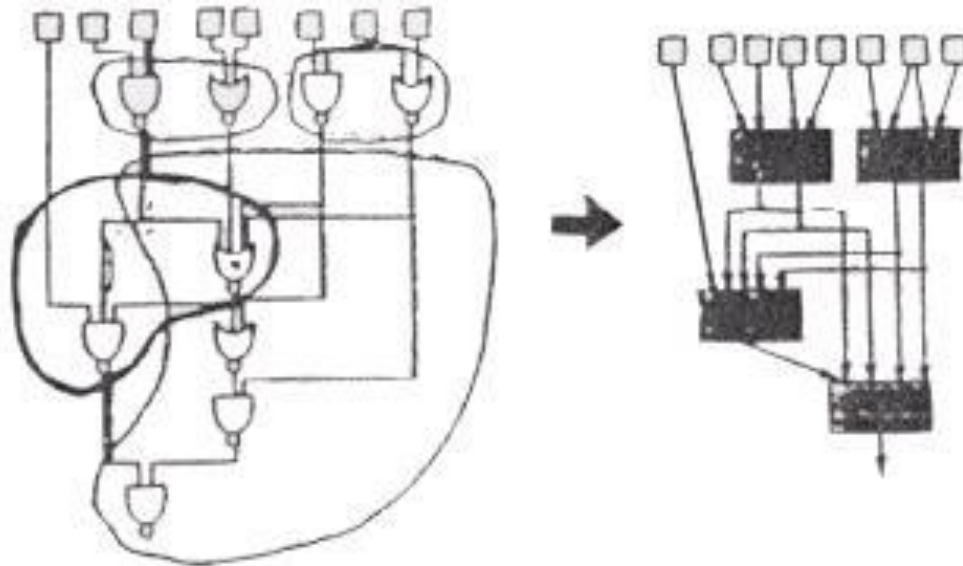


Delay = 7

Unit Delay Model

- No gate delay.
- No interconnection delay within a cluster.
- Delay of 1 unit for an interconnection between 2 clusters.
- An optimal algorithm for area constraint only (Lawler, Levitt and Turner, IEEE TC, 1966).
- An optimal algorithm for pin constraint only (Cong and Ding, ICCAD, 1992).

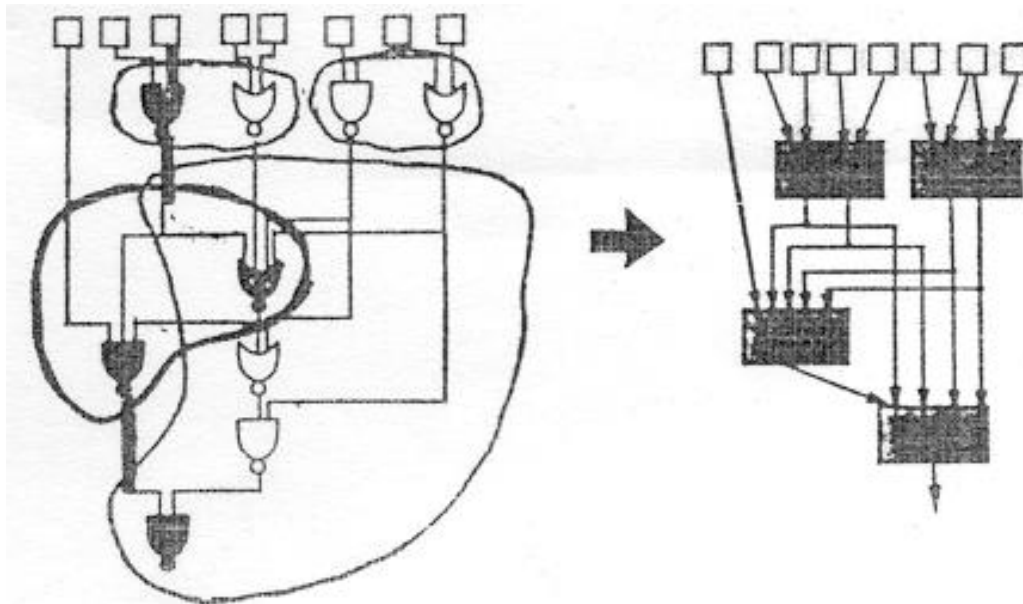
Circuit delay = 3.



General Delay Model

- Each gate has a delay.
- No interconnection delay within a cluster.
- Delay of D units for an interconnection between 2 clusters.
- A heuristic algorithm for area constraint only (Murgai, Brayton and Sangivanni-Vincentelli, ICCAD, 1991).

$D = 2$, $\delta(v) = 1$, circuit delay = $6+4 = 10$.



General Delay Model (Cont'd)

- Rajaraman and Wong, “Optimal clustering for delay minimization,” DAC, 1993.
 - Optimal algorithm: $O(n^2 \log n + nm)$, where n is # of gates, m is # of interconnections.
 - Definitions:
 - M : the area constraint on a cluster.
 - $W(C)$: the total area of the gates in cluster C .
 - N : a given combinational circuit.
 - N_v : v and all its *predecessors* in N .
 - $\delta(v)$: the delay of v .
 - $\Delta(u, v)$: maximum delay along any path from the output of u to the output of v , ignoring delays on interconnections.
 - $w(v)$: the area of v .
 - $l(v)$: the delay at v in an optimal clustering of N_v .
- For each *primary input* v , $l(v) = \delta(v)$.
- $l'(u) = l(u) + \Delta(u, v)$, for each u in $N_v - \{v\}$.

General Delay Model (Cont'd)

- Algorithm: labeling phase + clustering phase.
- **Labeling phase:** compute $l(v)$ for each v in a topological order.
 - P : the set of nodes in $N_v - \{v\}$ sorted in non-increasing order in the value of l' .

```

Algorithm Labeling( $v$ );
begin
  done  $\leftarrow$  false;
  cluster( $v$ )  $\leftarrow$  { $v$ };
  while (not done)
    Remove the first node  $u$  in  $P$ ;
    if ( $W(\text{cluster}(v)) + w(u) \leq M$ )
      cluster( $v$ )  $\leftarrow$  cluster( $v$ )  $\cup$  { $u$ };
      if  $P$  is empty
        done  $\leftarrow$  true;
      endif
    else
      done  $\leftarrow$  true;
    endif
  endwhile
   $l_1(v) \leftarrow \max\{l'(x) \mid x \in \text{cluster}(v) \cap PI\}$ ;
   $l_2(v) \leftarrow \cancel{l'(v) + D}, \max\{l'(u) + D \mid u \in N_v - \text{cluster}(v)\}$ ;
   $l(v) \leftarrow \max\{l_1(v), l_2(v)\}$ ;
end
    
```

General Delay Model (Cont'd)

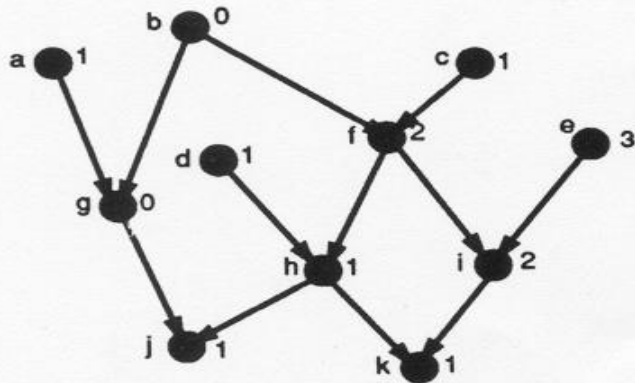
- **Clustering phase:** generate the clusters based on the information obtained in the labeling phase.
- Overall algorithm:

```

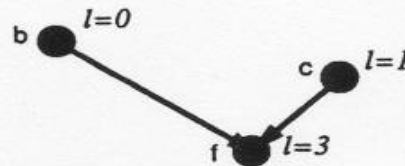
begin
  Compute the maximum delay matrix  $\Delta$ .  $\Delta(i, j)$  is the
  maximum delay along any path from the output of  $i$ 
  to the output of  $j$ ;
  for each PI  $i$ , do  $l(i) \leftarrow \delta(i)$ ;
  Sort the non-PI nodes of  $N$  in topological order
  to obtain list  $T$ ;
  while  $T$  is non-empty
    Remove the first node  $v$  from  $T$ ;
    Compute  $N_v$ ;
    for each node  $u \in N_v \setminus \{v\}$  do
       $l'(u) \leftarrow l(u) + \Delta(u, v)$ ;
    Sort the nodes in  $N_v \setminus \{v\}$  in order of
    decreasing value of  $l'$  to form list  $P$ ;
    Call Labeling( $v$ );
  endwhile
   $L \leftarrow \mathcal{PO}$ ;
   $S \leftarrow \phi$ ;
  while  $L$  is not empty
    Remove a node  $v$  from  $L$ ;  $N$ -cluster( $v$ )
     $S \leftarrow S \cup \{cluster(v)\}$ ;
    for all nodes  $x$  in  $N$ , such that  $x$  is adjacent
    to  $v$ , for some  $y \in cluster(v)$ ,  $L \leftarrow L \cup \{x\}$ ;
  endwhile
end
  
```

General Delay Model (Cont'd)

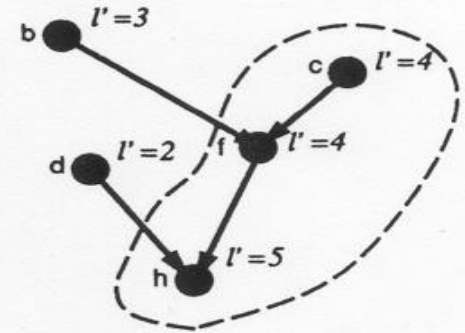
- An example: $M=3, D=3$



(a)



(b)



(c)

