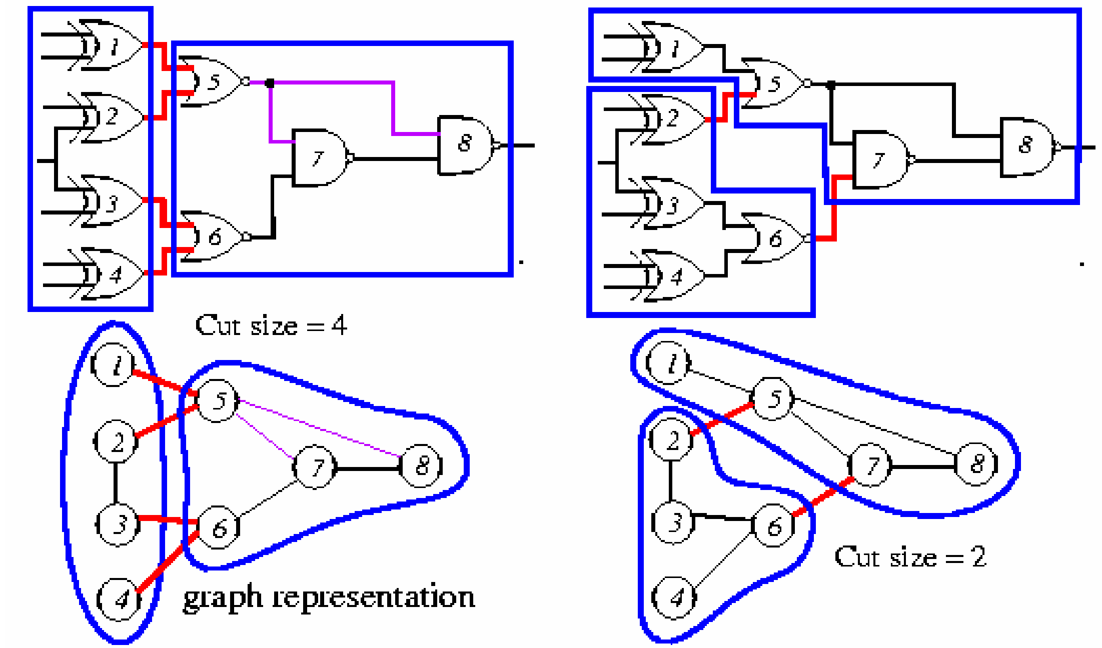
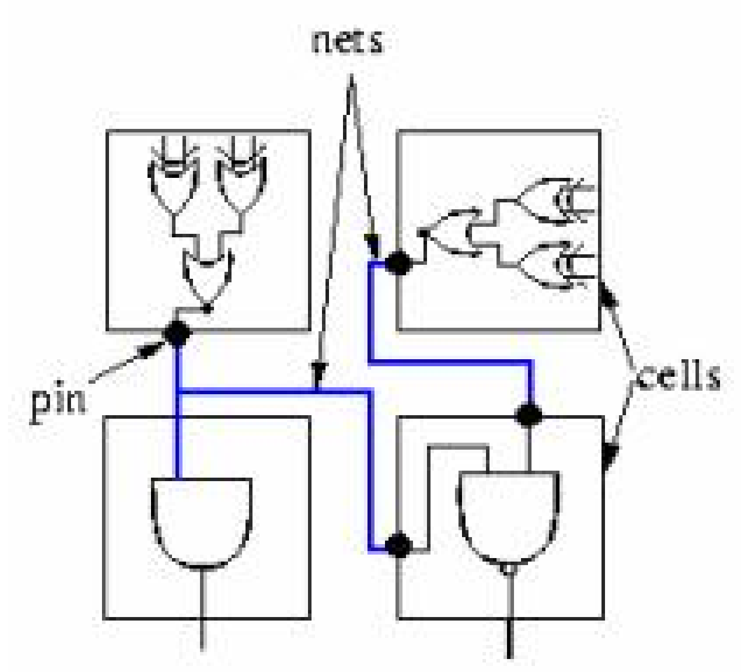
Partitioning

- Course contents:
  - Kernighang-Lin partitioning heuristic
  - Fiduccia-Mattheyses heuristic
- Readings
  - Chapter 7.5



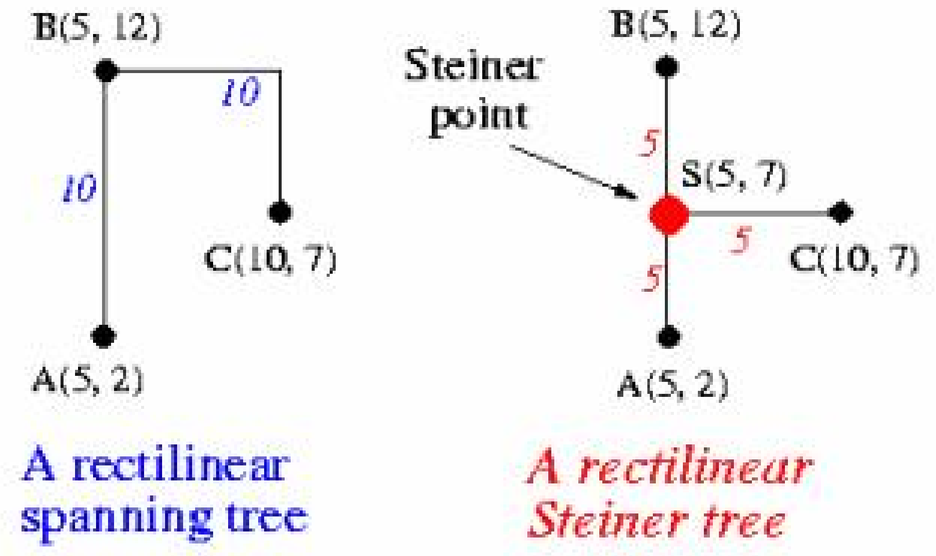
Basic Definitions

- **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.

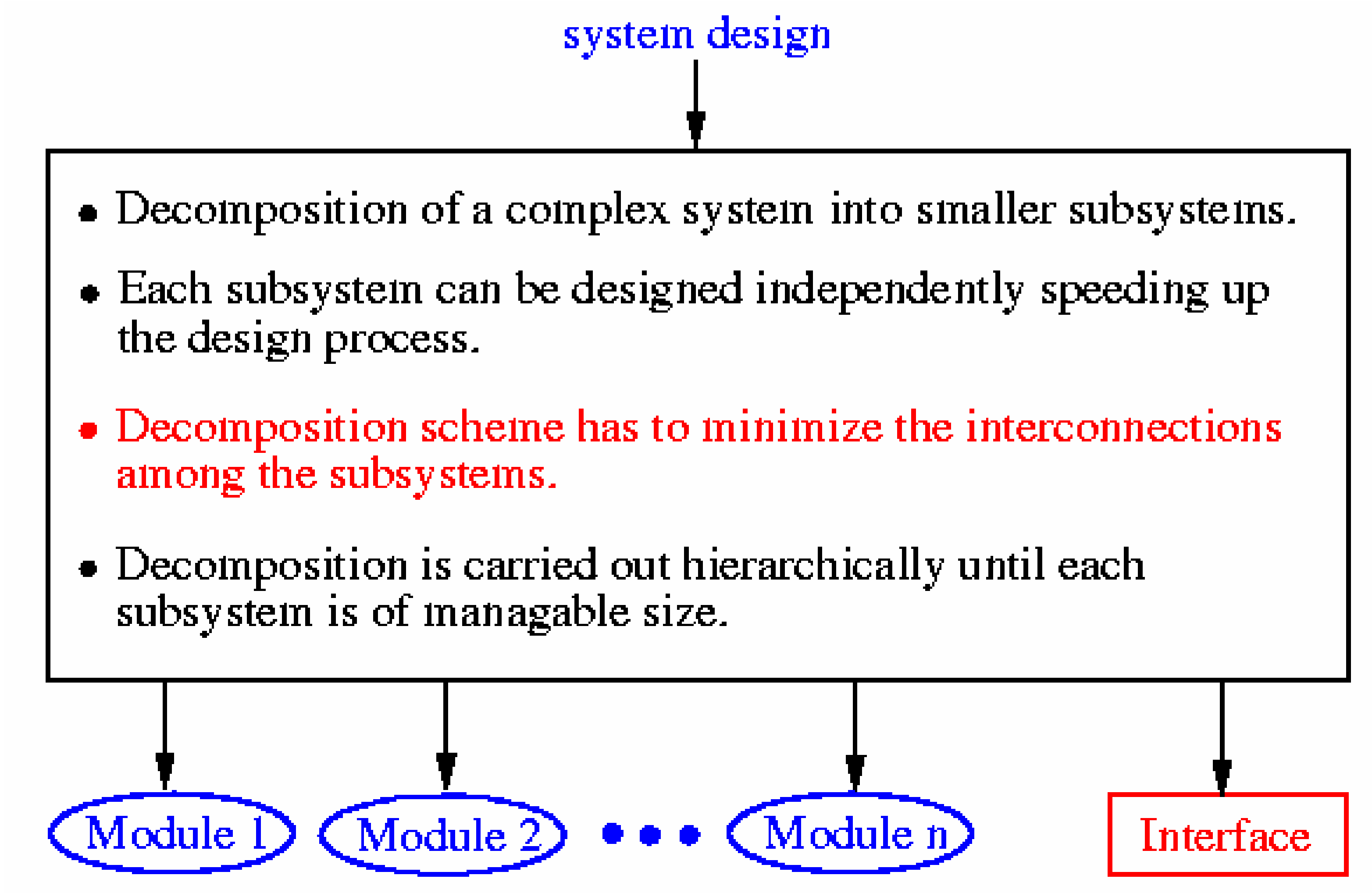


Basic Definitions (cont'd)

- **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.

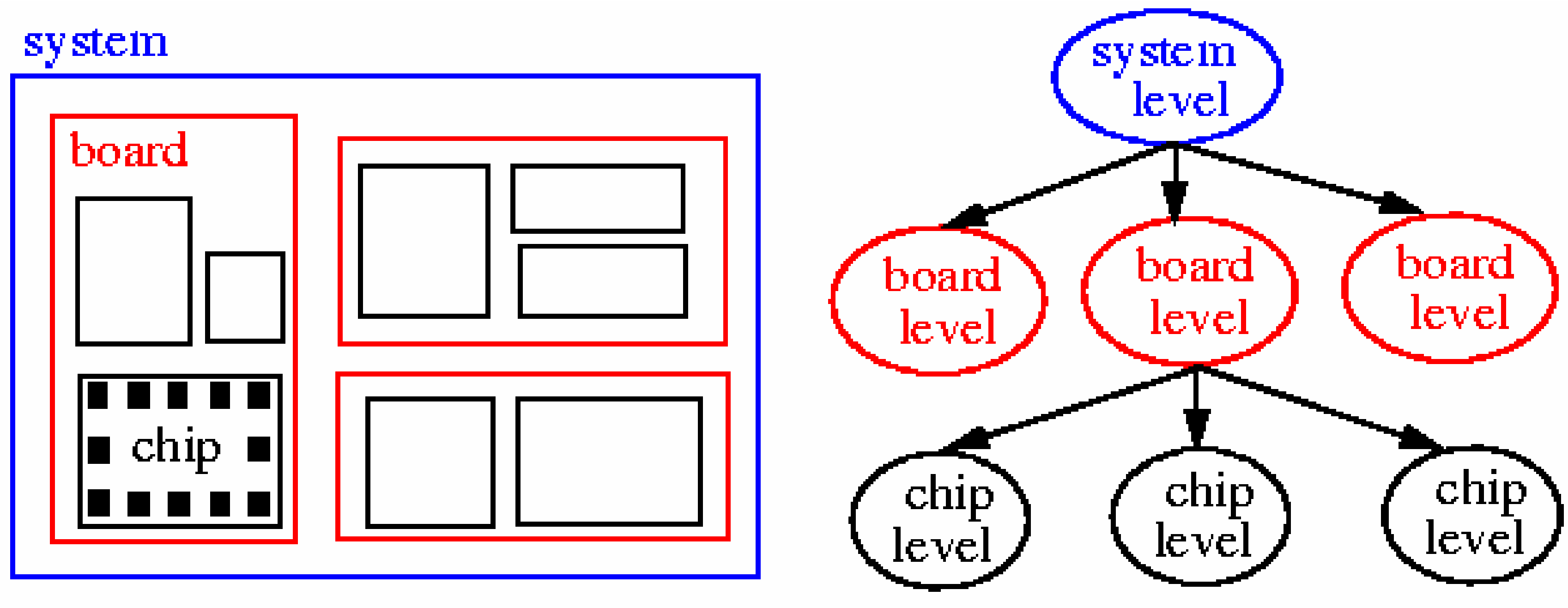


Partitioning



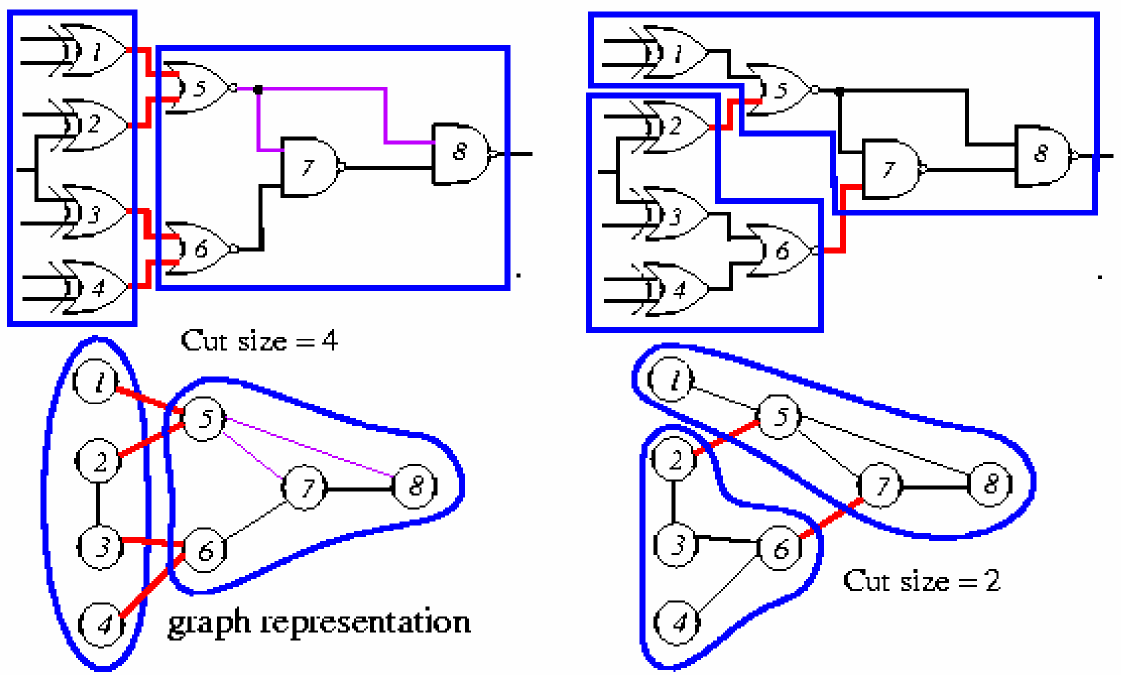
Levels of Partitioning

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



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

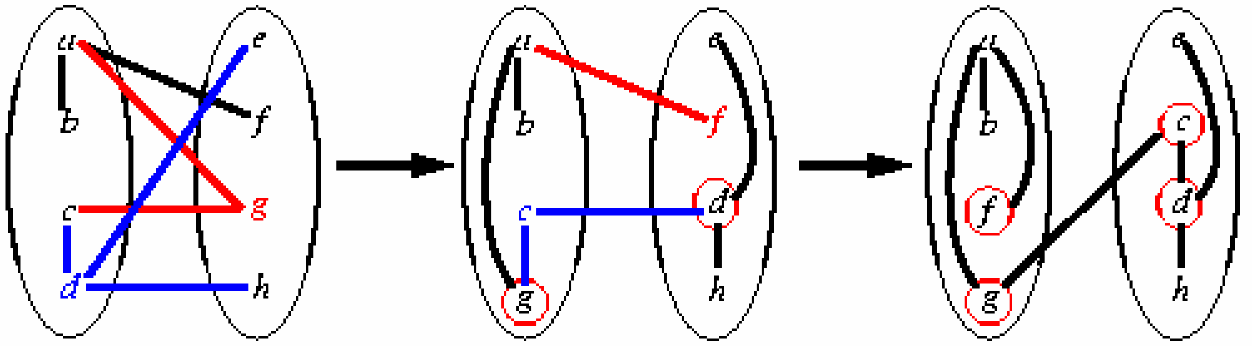
- ***k*-way partitioning:** Given a graph $G(V, E)$, where each vertex v Î V has a **size** $s(v)$ and each edge e Î E has a **weight** $w(e)$, the problem is to divide the set V into k disjoint subsets $V_1, V_2, \ldots, V_k$, such that an objective function is optimized, subject to certain constraints.
- **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 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.



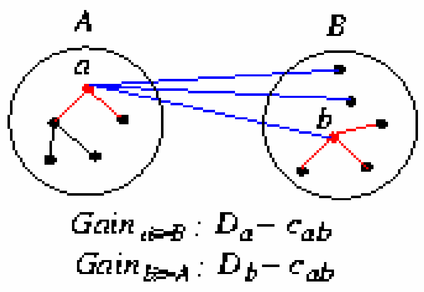
Step #	Vertex pair	Cost reduction	Cut cost
0	-	0	5
1	{d, g}	3	2
2	{c, f}	1	1
3	{b, h}	-2	3
4	{a, e}	-2	5

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

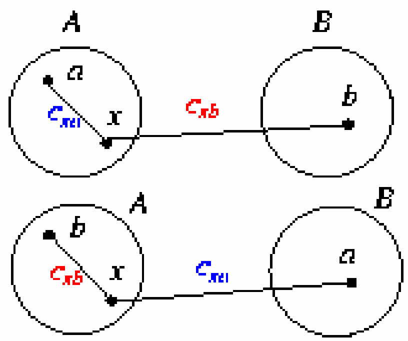
Properties

- Two sets *A* and *B* such that |*A*| = *n* = |*B*| and *A* Ç *B* = Æ.
- **External cost** of *a* Î *A*: $E_a = \sum_{v \in B} c_{av}$.
- **Internal cost** of *a* Î *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* Î A and *b* Î *B* are interchanged, then the new *D*-values, *D*', are given by

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



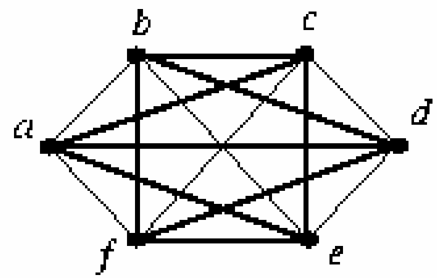
Internal cost vs. External cost



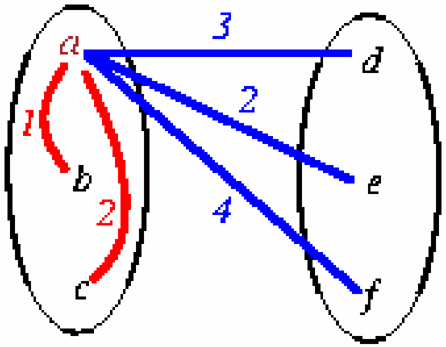
before swap	after swap	$\triangle C$
$-c_{xa}$	$+c_{xa}$	$+2c_{xa}$
$+c_{xb}$	$-c_{xb}$	$-2c_{xb}$

updating D−values

Kernighan-Lin 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

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 (cont'd)

- Iteration 1:

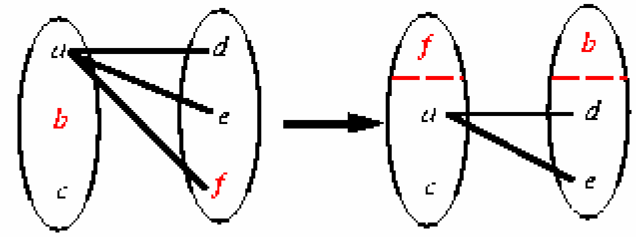
$$
\begin{array}{lll}
I_a = 1 + 2 = 3; & E_a = 3 + 2 + 4 = 9; & D_a = E_a - I_a = 9 - 3 = 6 \\
I_b = 1 + 1 = 2; & E_b = 4 + 2 + 1 = 7; & D_b = E_b - I_b = 7 - 2 = 5 \\
I_c = 2 + 1 = 3; & E_c = 3 + 2 + 1 = 6; & D_c = E_c - I_c = 6 - 3 = 3 \\
I_d = 4 + 3 = 7; & E_d = 3 + 4 + 3 = 10; & D_d = E_d - I_d =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

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

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

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

Weighted Example (cont'd)



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

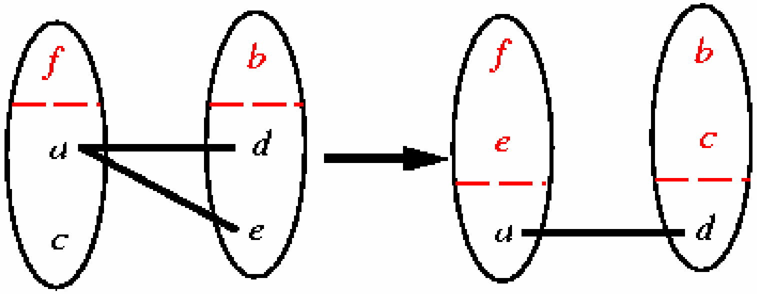
$$
\begin{aligned}
D'_a &= D_a + 2c_{ab} - 2c_{af} = 6 + 2 \times 1 - 2 \times 4 = 0 \\
D'_c &= D_c + 2c_{cb} - 2c_{cf} = 3 + 2 \times 1 - 2 \times 1 = 3 \\
D'_d &= D_d + 2c_{df} - 2c_{db} = 3 + 2 \times 3 - 2 \times 4 = 1 \\
D'_e &= D_e + 2c_{ef} - 2c_{eb} = 0 + 2 \times 2 - 2 \times 2 = 0
\end{aligned}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

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

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

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

Weighted Example (cont'd)



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

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

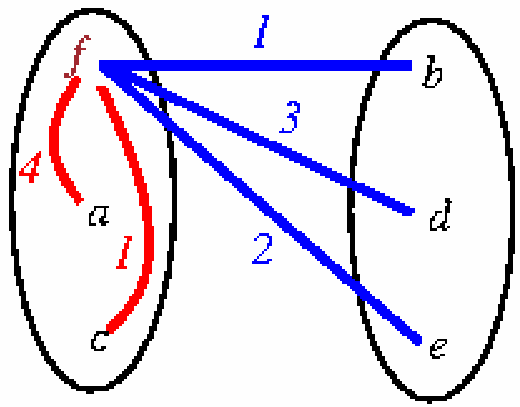
- $g_{xy} = D''_x + D''_y - 2c_{xy}$.

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

- 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*.

Weighted Example (cont'd)

	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: $\hat{g}_1 = g_{ce} = -1,\ \hat{g}_2 = g_{ab} = -3,\ \hat{g}_3 = g_{fd} = 4.$
- Largest partial sum = $\max \sum_{i=1}^{k} \hat{g}_i = 0$ (k=3) Þ 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.

```
begin
Bipartition G into A and B such that |V_A| = |V_B|, V_A ∩ V_B = ∅,
  and V_A ∪ V_B = V.
repeat
  Compute D_v, ∀ v ∈ V.
  for i =1 to n do
    Find a pair of unlocked vertices v_ai ∈ V_A and v_bi ∈ V_B whose
    exchange makes the largest decrease or smallest increase in cut
    cost;
    Mark v_ai and v_bi as locked, store the gain ĝ_i , and compute the new
    D_v, for all unlocked v ∈ V;
  Find k, such that G_k = Σ_{i=1}^{k} ĝ_i  is maximized;
  if G_k > 0 then
    Move v_a1, …, v_ak from V_A to V_B and v_b1, …, v_bk from V_B to V_A;
  Unlock v, ∀ v ∈ V.
until G_k ≤ 0;
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 $(n\text{-}i+1)^2$ 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?

Extensions of K-L Algorithm

1. **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.

- **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.

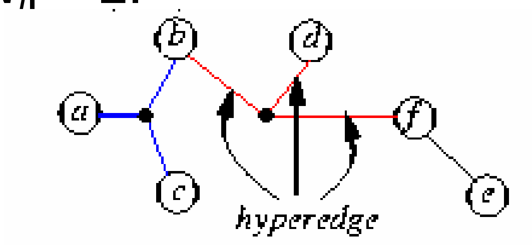
- ***k*-way partition**
  1. Partition the graph into k equal-sized sets.
  2. Apply the Kernighan-Lin algorithm for each pair of subsets.
  3. Time complexity? Can be reduced by recursive bi-partition.

Drawbacks of the Kernighan-Lin Heuristic

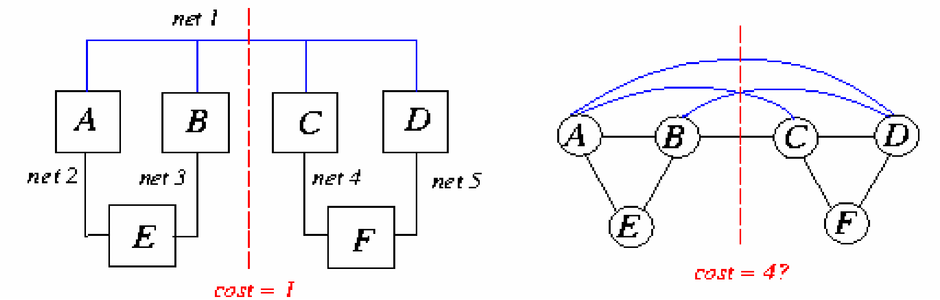
- The K-L heuristic **handles only unit vertex weights**.
  - Vertex weights might represent block sizes, different from blocks to blocks.
  - Reducing a vertex with weight $w(v)$ into a clique with $w(v)$ vertices and edges with a high cost increases the size of the graph substantially.
- The K-L heuristic **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**, $O(n^3)$.

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 $|N_i| \geq 2$.

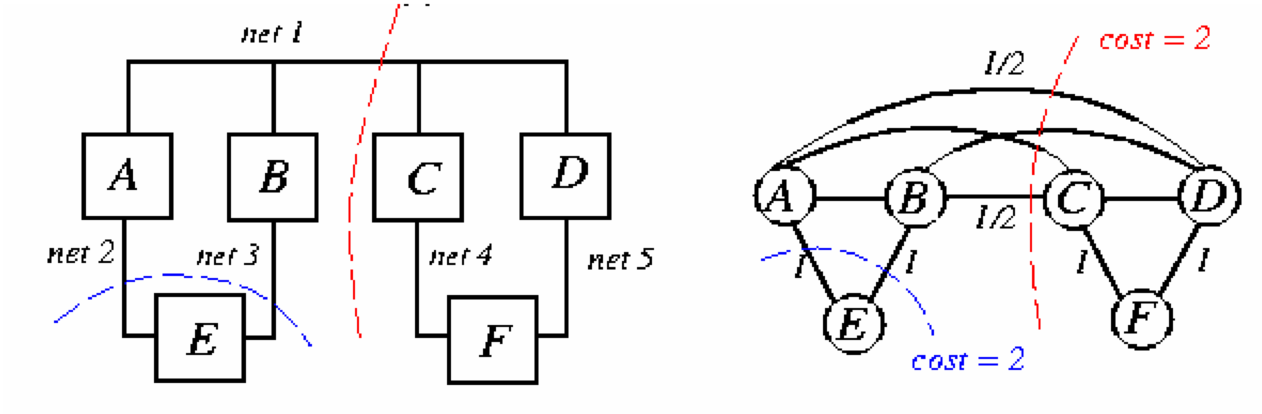


- Schweikert and Kernighan, "A proper model for the partitioning of electrical circuits," 9th Design Automation Workshop, 1972.
- For multi-terminal nets, **net cut** is a more accurate measurement for cut cost (i.e., deal with hyperedges).
  - {A, B, E}, {C, D, F} is a good partition.
  - Should not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

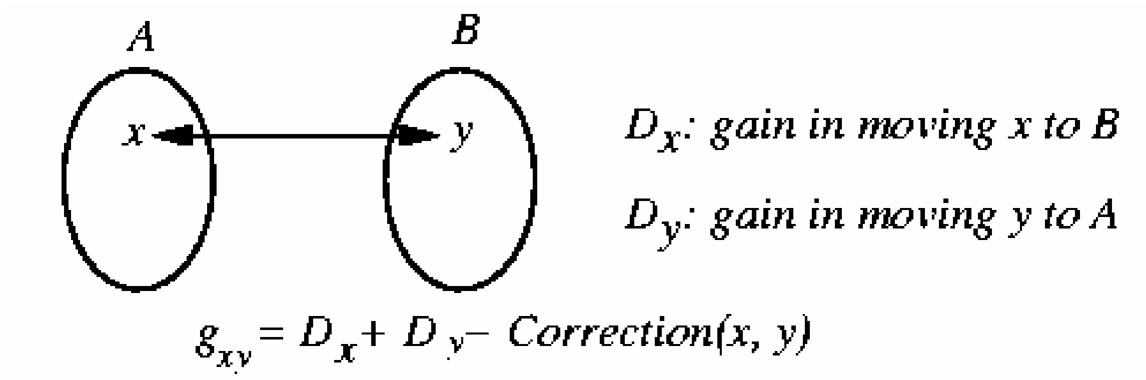


Net-Cut Model

- 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.

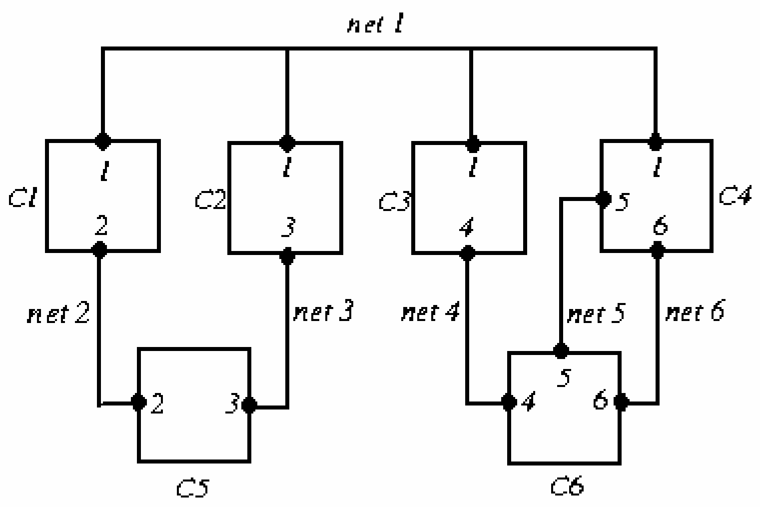


Fiduccia-Mattheyses Heuristic

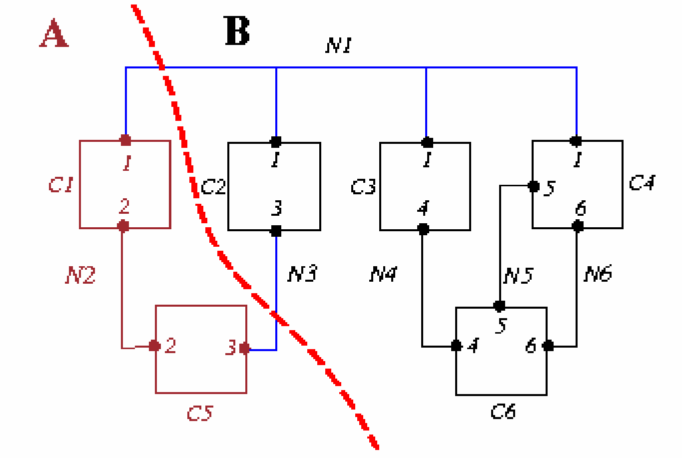
- Fiduccia and Mattheyses, "A linear time heuristic for improving network partitions," DAC-82.
- New features to the K-L heuristic:
  - Aims at **reducing net-cut costs**; the concept of cutsize is extended to hypergraphs.
  - Only a **single vertex** is moved across the cut in a single move.
  - Vertices are weighted.
  - Can handle "unbalanced" partitions; a balance factor is introduced.
  - **Time complexity** $O(P)$, where P is the total # of terminals.

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; $P = p(1) + \ldots + p(C) = n(1) + \ldots + n(N)$.

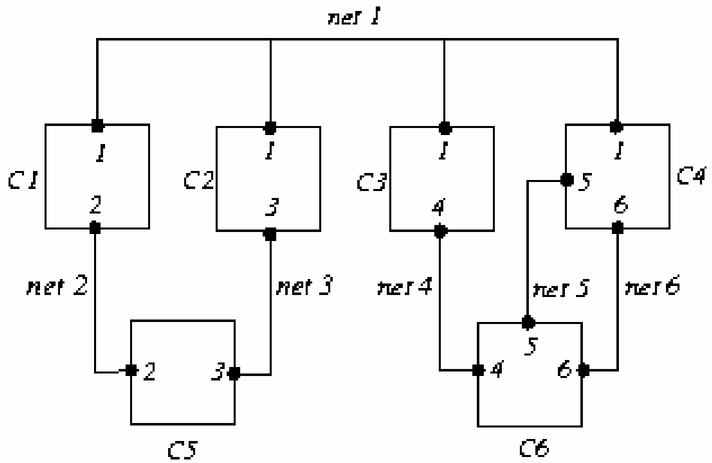


Cut



- **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

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

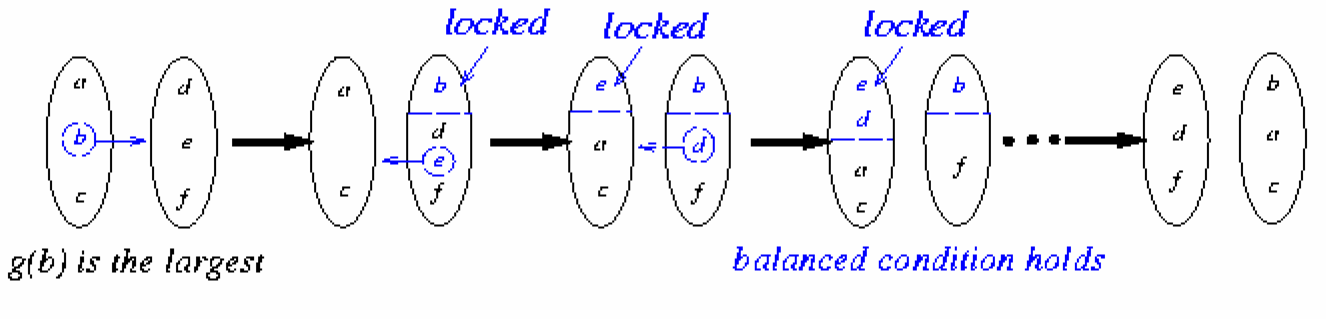
Basic Ideas: Balance and Movement

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

$$
\begin{aligned}
\frac{|A|}{|A|+|B|} &\approx r \\
rW - S_{max} &\le |A| \le rW + S_{max},
\end{aligned}
$$

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

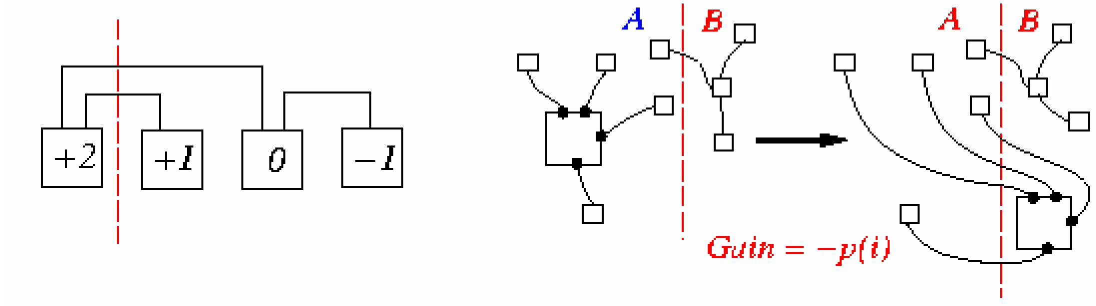
- $g(i)$: gain in moving cell i to the other set, i.e., size of **old** cutset - size of **new** cutset.



- 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 $\Rightarrow$ resulting two sets: $\{a, c, e, d\}$, $\{b, f\}$.

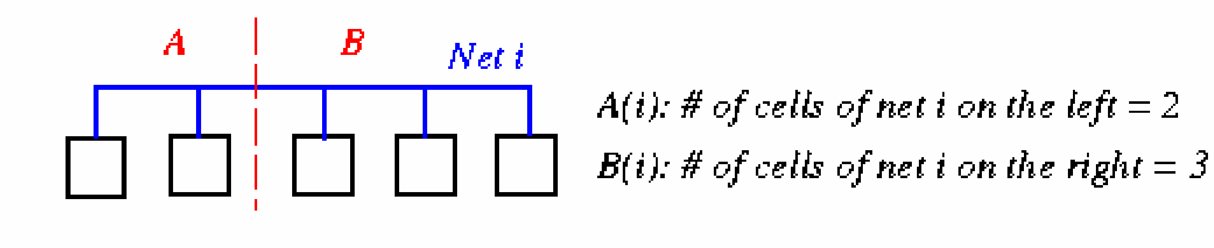
Cell Gains

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

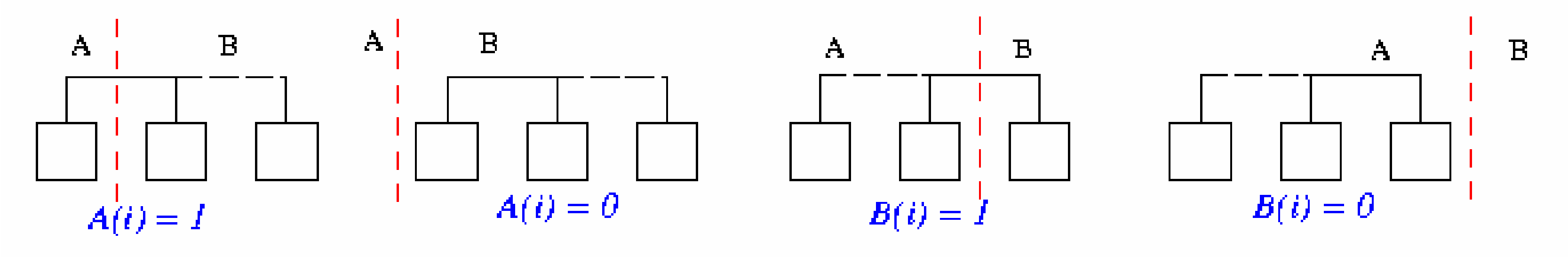


Net Distribution and Critical Nets

- 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.

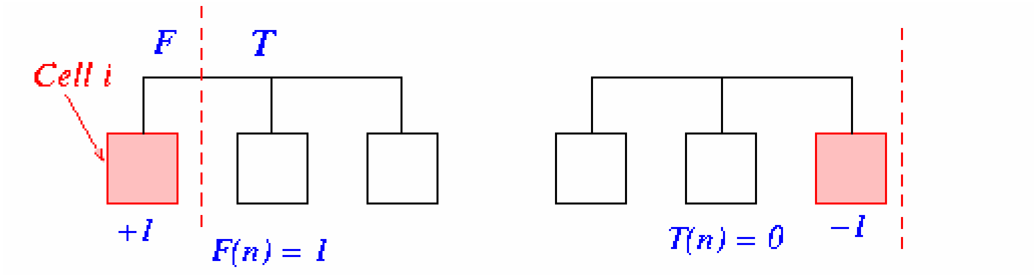


- **Gain of a cell depends only on its critical nets.**

Computing Cell Gains

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

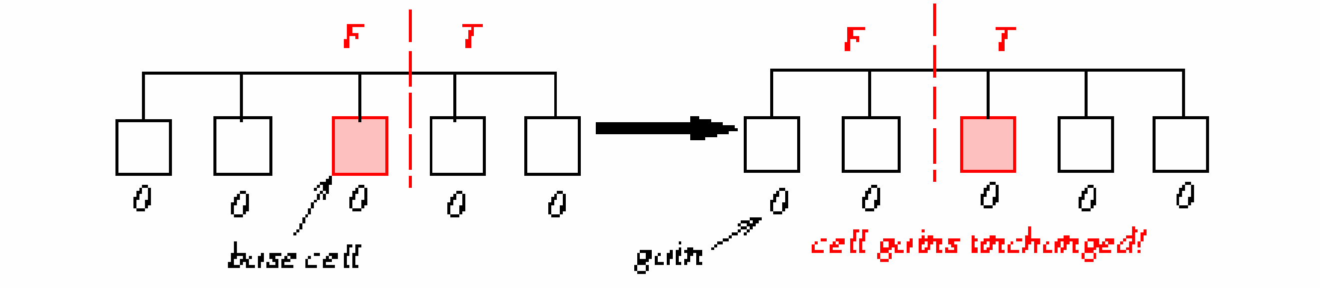
  g(*i*) ¬ 0;
  F ¬ the "from block" of Cell *i*;
  T ¬ the "to block" of Cell *i*;
  for each net *n* on Cell *i* **do**
      if *F*(*n*)=1 **then** *g*(*i*) ¬ *g*(*i*)+1;
      if *T*(*n*)=0 **then** *g*(*i*) ¬ *g*(*i*)-1;



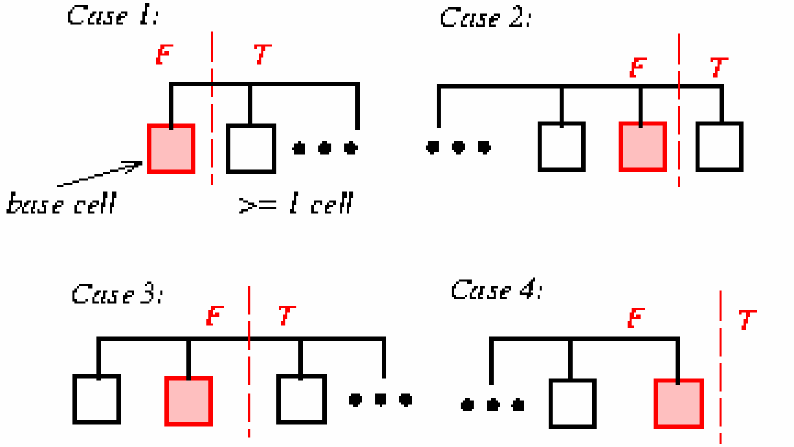
- Will show: Only need *O*(*P*) time to maintain all cell gains in one pass.

Updating Cell Gains

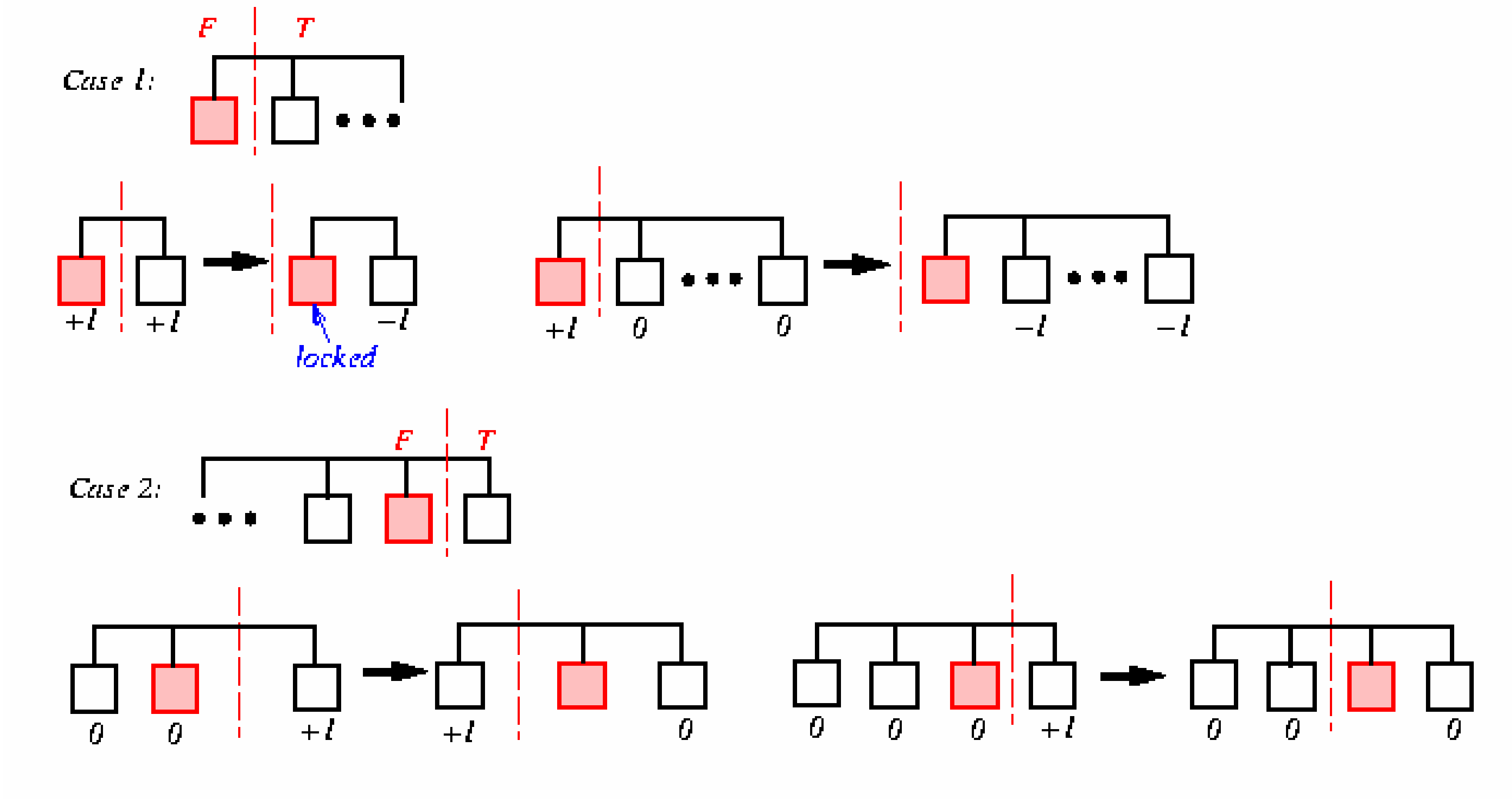
- To update the gains, we only need to look at those nets, connected to the base cell, which are critical **before** or **after** the move.
- **Base cell**: The cell selected for movement from one set to the other.



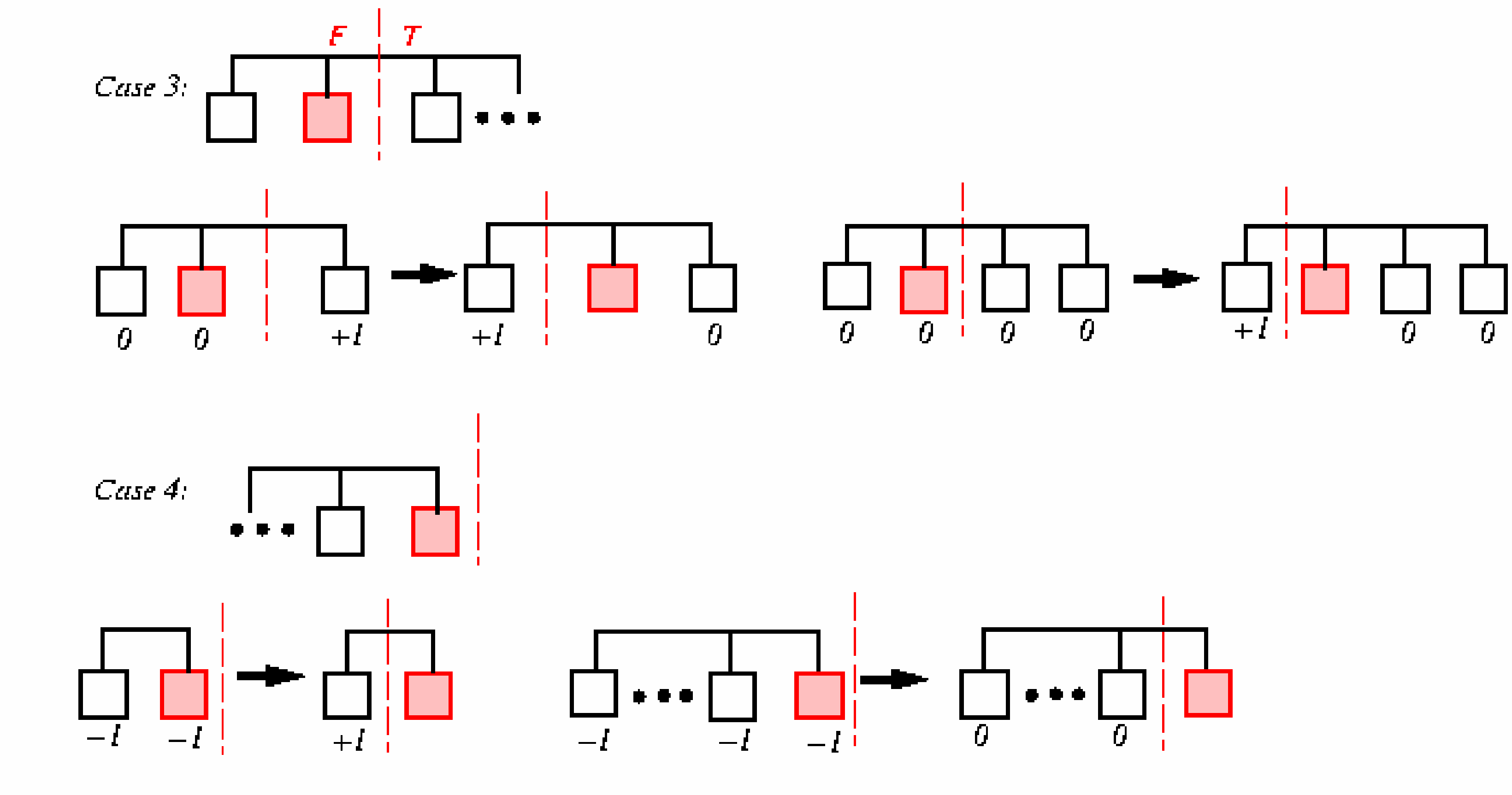
- Consider only the case where the base cell is in the left partition. The other case is similar.



Updating Cell Gains (cont'd)

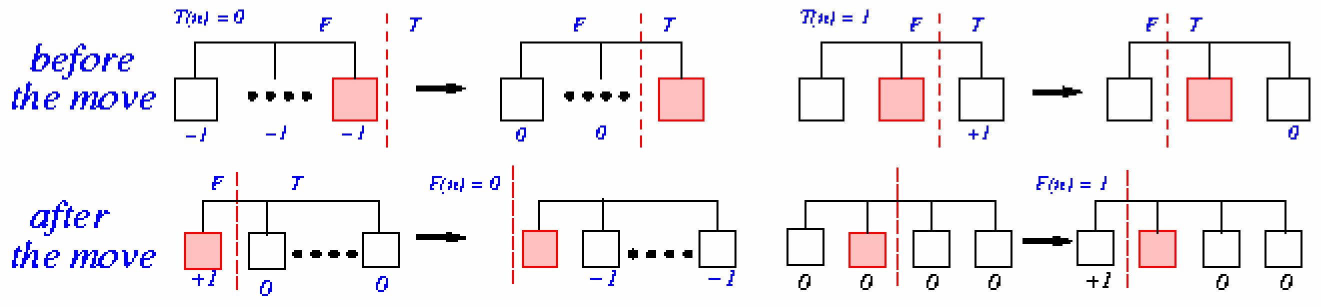


Updating Cell Gains (cont'd)



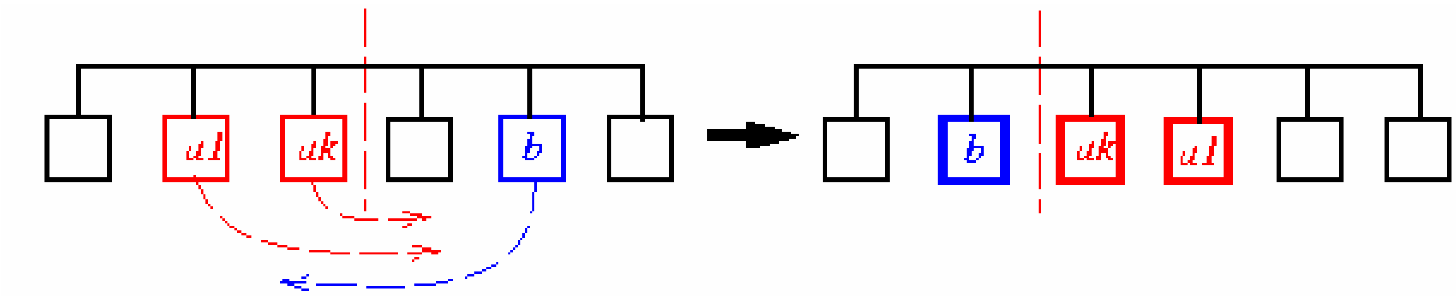
Algorithm for Updating Cell Gains

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*
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*) ← *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*
else if *F*(*n*) = 1 then increment gain of the only *F* cell on *n*,
if it is free
9 end



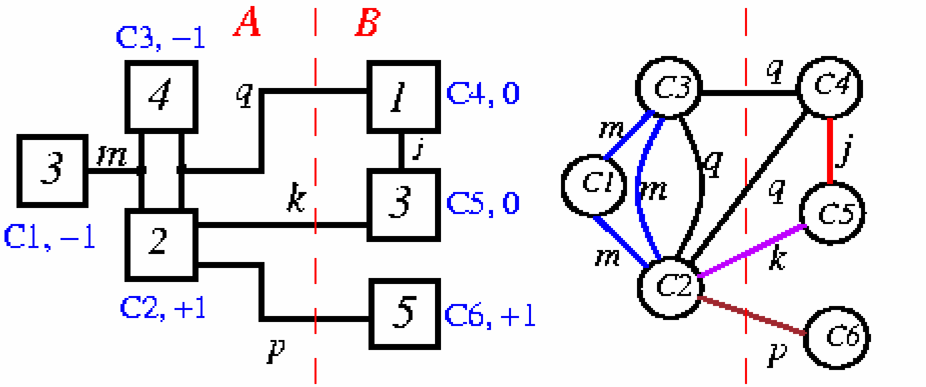
Complexity of Updating Cell Gains

- 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 Þ 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 = $n(1)+n(2)+\ldots+n(N) = O(P)$.

F-M Heuristic: An Example

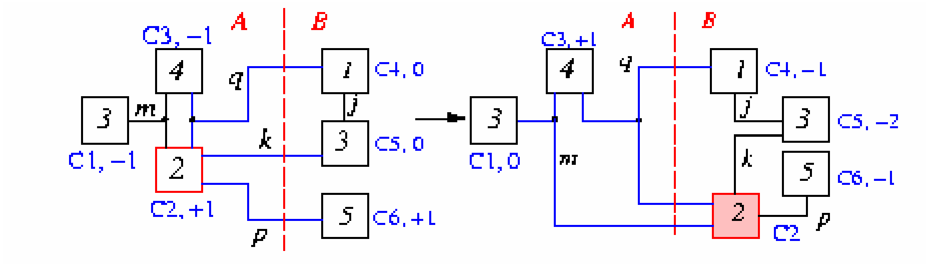


- Computing cell gains: $F(n) = 1$ Þ $g(i) + 1$; $T(n)=0$ Þ $g(i) - 1$

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: $r|V| - S_{max}$ £ $|A|$ £ $r|V| + S_{max}$. Let $r = 0.4$ Þ $|A| = 9$, $|V| = 18$, $S_{max} = 5$, $r|V|=7.2 \Rightarrow$ Balanced: 2.2 £ 9 £ 12.2!
- maximum gain: c_2 and balanced: 2.2 £ 9-2 £ 12.2 Þ 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:

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 £ 7-4 £ 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	$\lvert A \rvert$	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

- $\hat{g_1} = 1, \hat{g_2} = 1, \hat{g_3} = -1, \hat{g_4} = 1, \hat{g_5} = -2, \hat{g_6} = 0$ Þ Maximum partial sum $G_k = +2$, $k = 2$ or 4.
- Since k=4 results in a better balanced Þ 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 £ 0.**