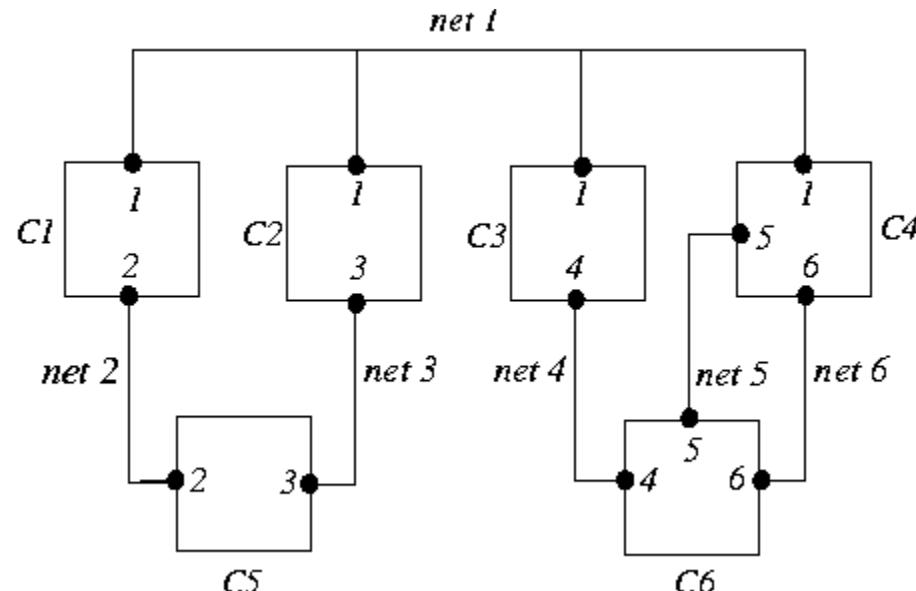


Fiduccia-Mattheyses (FM) Algorithm

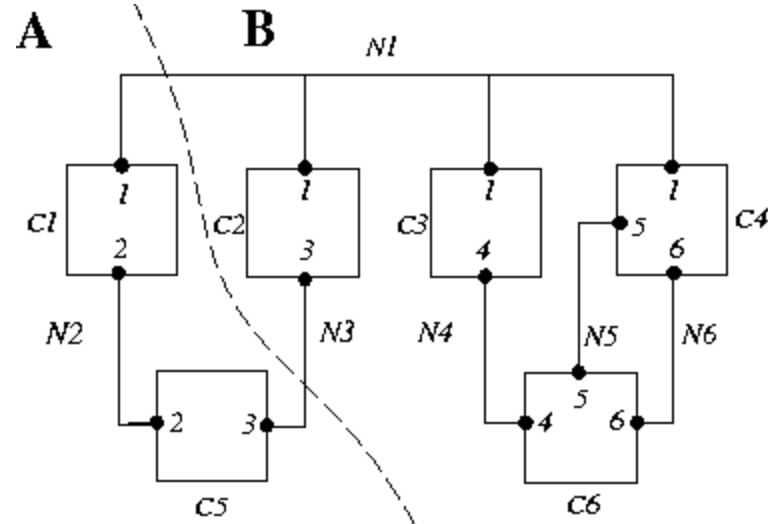
- Fiduccia and Mattheyses. “A linear time heuristic for improving network partitions,” 19th Design Automation Conf., 1982.
- Same as KL:
 - Work in passes.
 - Lock vertices after moved.
 - Actually, only move those vertices up to the maximum partial sum of gain.
- Different from KL:
 - Aim at **reducing 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; $P = p(1)+\dots+p(C) = n(1)+\dots+n(N)$.

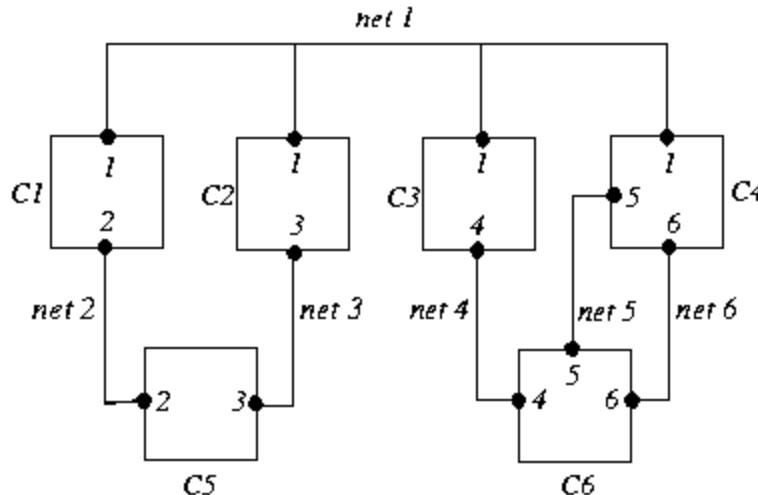


Cut



- **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

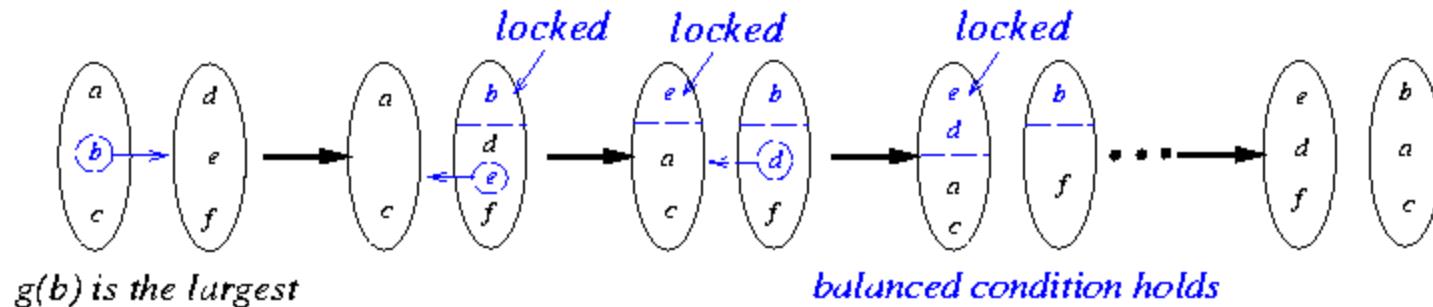
- Only move a cell at a time, preserving “balance.”

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

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

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

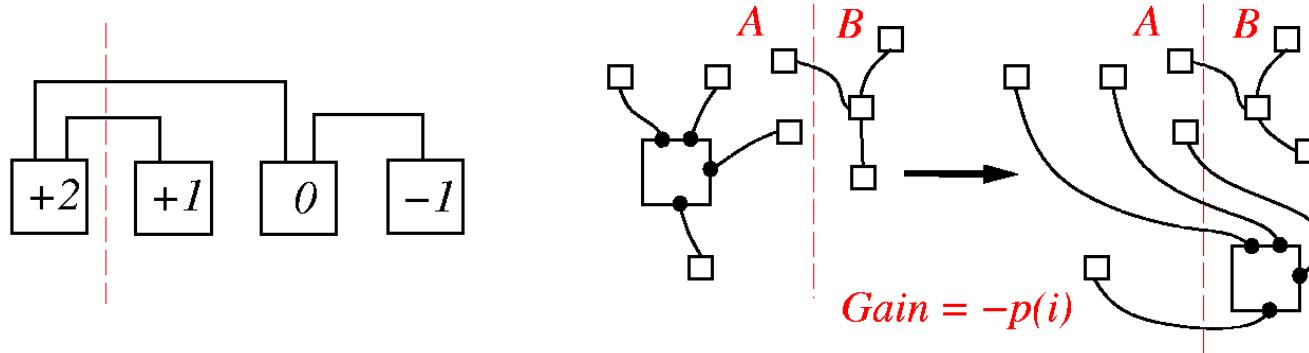
- $g(i)$: gain in moving cell i to the other set,
i.e., size of **old** cutset - size of **new** cutset.



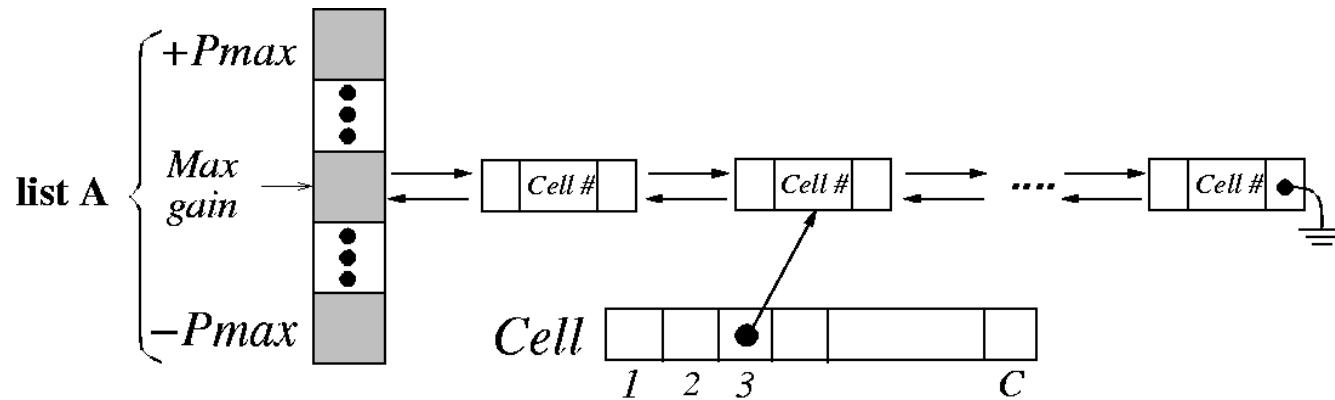
- Suppose \hat{g}_i 's: $g(b), g(e), g(d), g(a), g(f), g(c)$ and the largest partial sum is $g(b) + g(e) + g(d)$. Then we should move $b, e, d \rightarrow$ resulting two sets: $\{a, c, e, d\}, \{b, f\}$

Cell Gains and Data Structure Manipulation

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



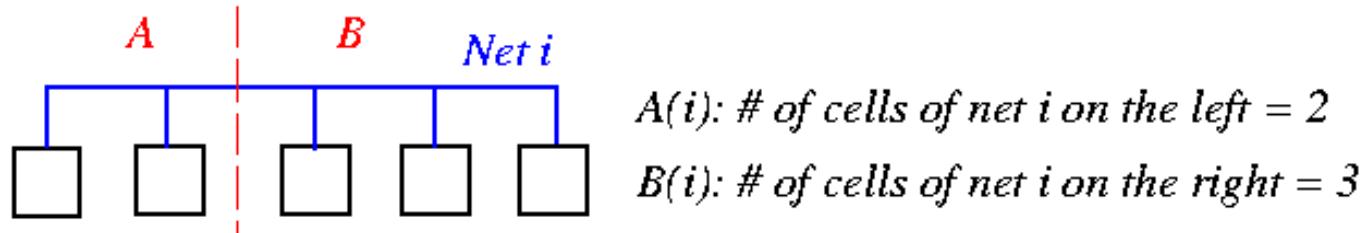
- Two “bucket list” structures, one for set A and one for set B ($P_{max} = \max_i p(i)$).



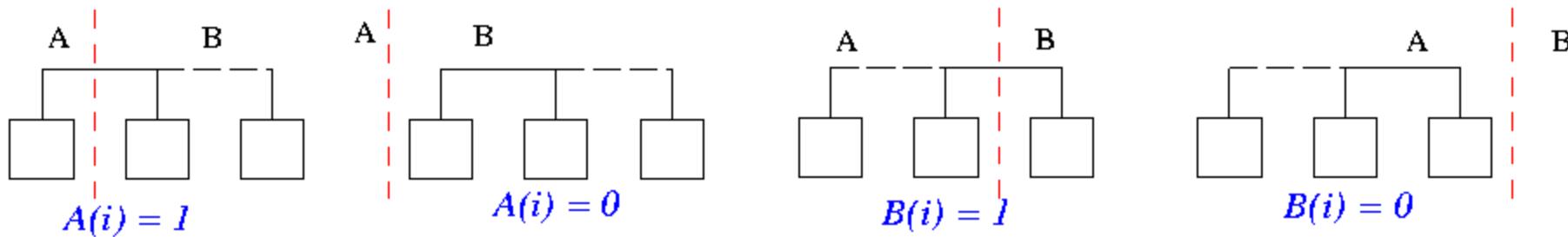
- $O(1)$ -time operations: find a cell with Max Gain, remove Cell i from the structure, insert Cell i into the structure, update $g(i)$ to $g(i) + \Delta$, update the Max Gain pointer.

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:

$g(i) \leftarrow 0;$

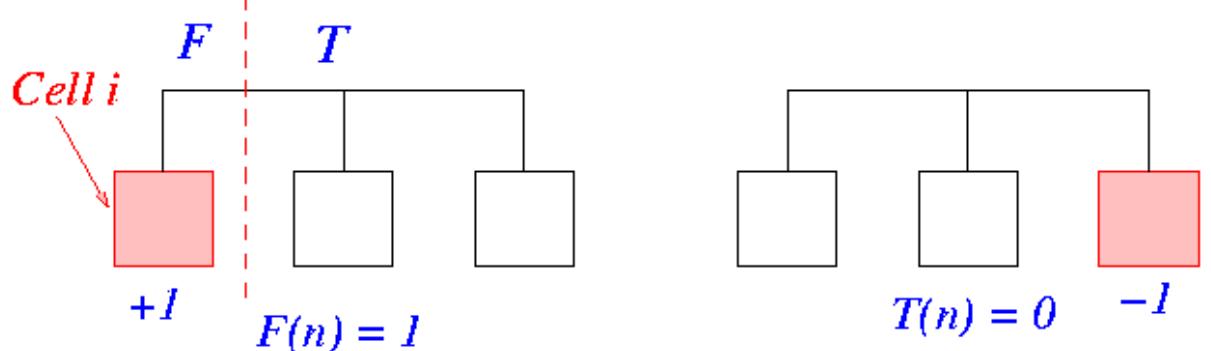
$F \leftarrow$ the “from block” of cell i ;

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

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

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

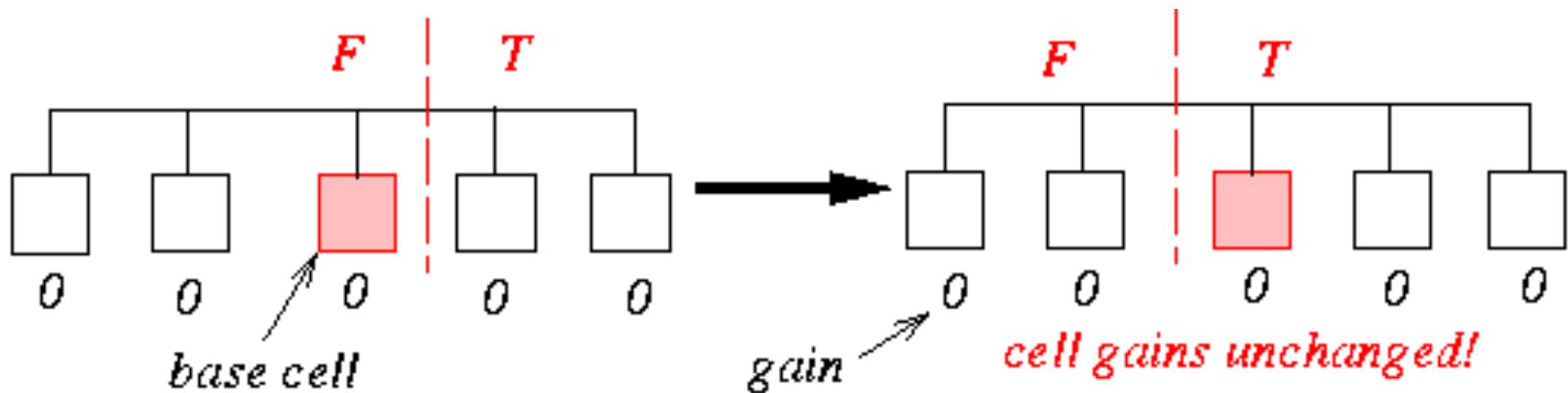
if $T(n) = 0$ then $g(i) \leftarrow g(i) - 1$;



- 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.



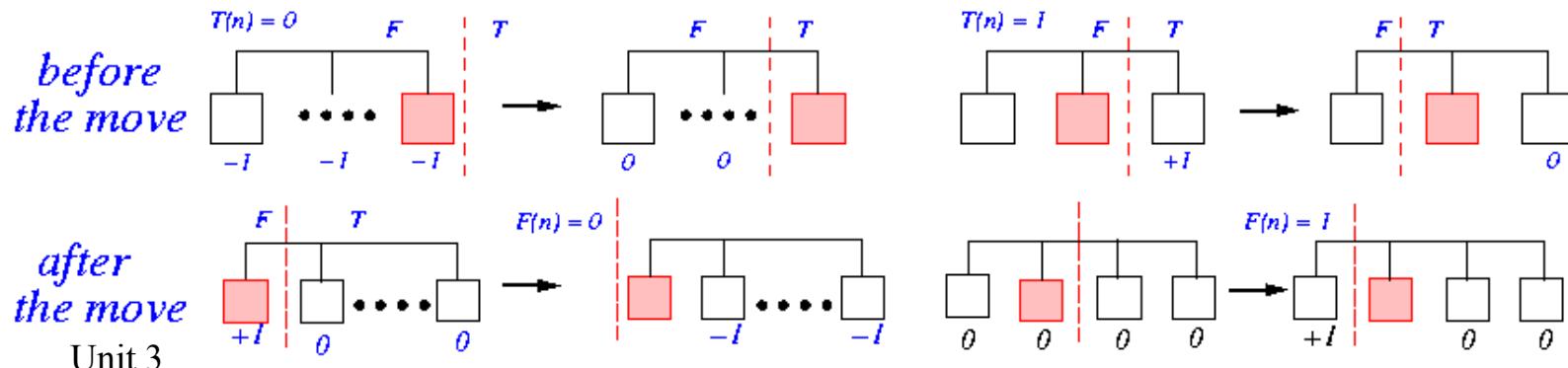
Algorithm for Updating Cell Gains

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

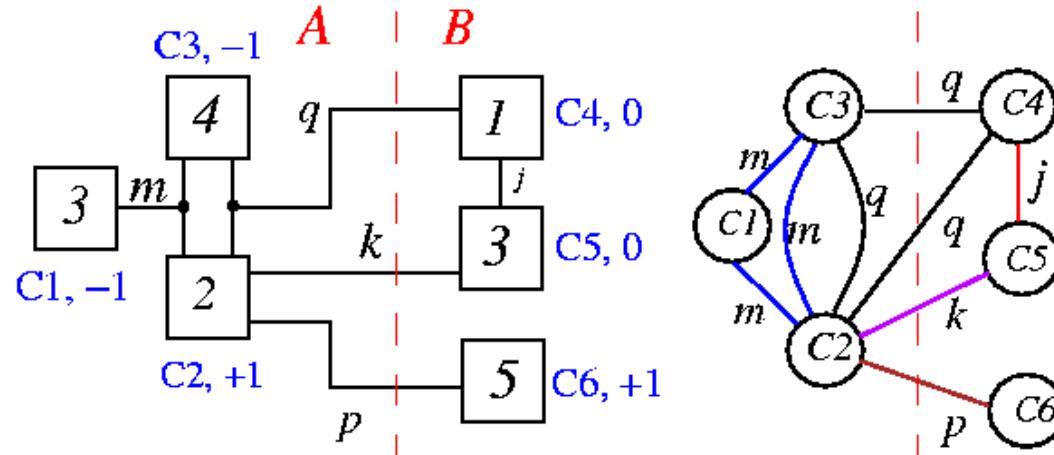
```



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

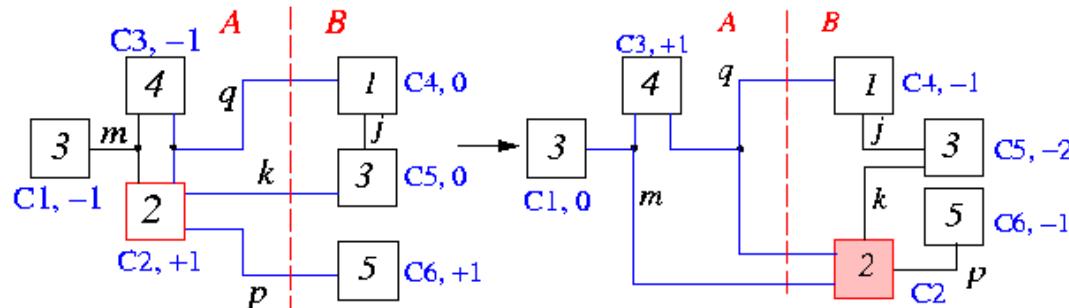


- Computing cell gains: $F(n) = 1 \Rightarrow g(i) + 1$; $T(n) = 0 \Rightarrow g(i) - 1$

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).

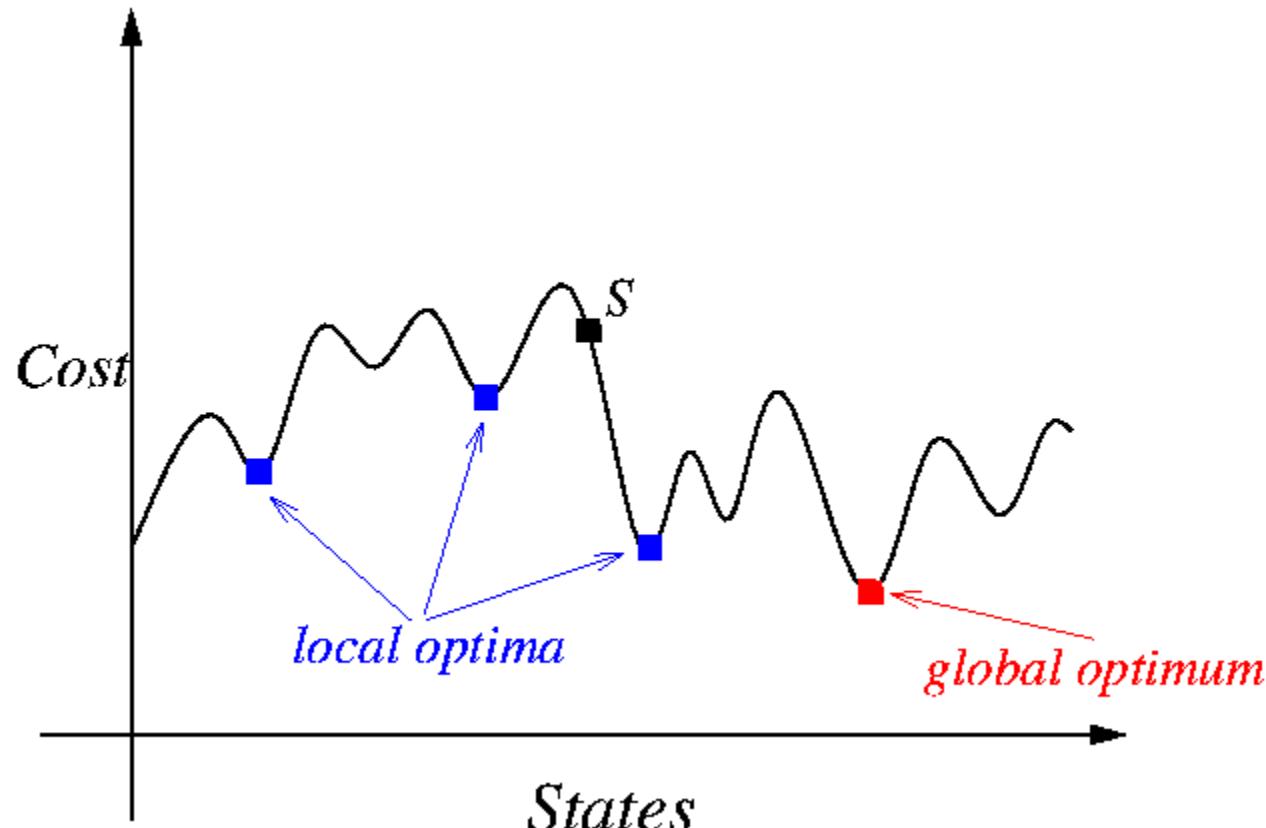
Summary of the Example

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

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

- $\Delta C = cost(S') - cost(S)$
- 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

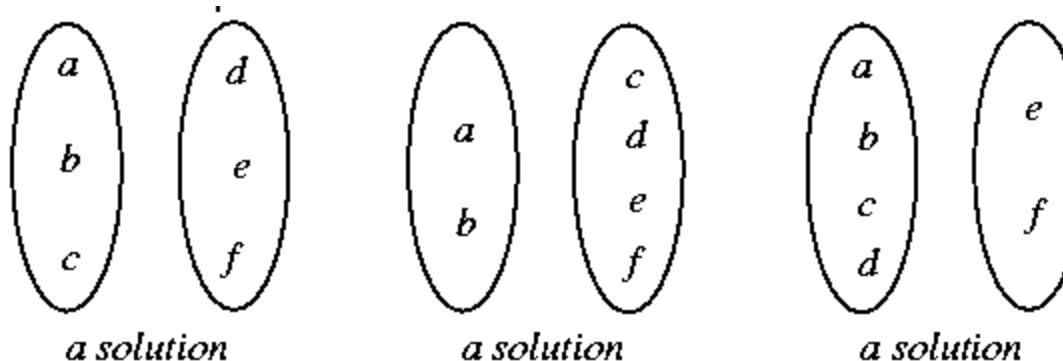
```
1 begin
2 Get an initial solution  $S$ ;  $Best \leftarrow S$ 
3 Get an initial temperature  $T > 0$ ;
4 while not yet “frozen” do
5   for  $1 \leq i \leq P$  do
6     Pick a random neighbor  $S'$  of  $S$ ;
7      $\Delta \leftarrow cost(S') - cost(S)$ ;
     /* down hill move */
8     if  $\Delta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

Basic Ingredients for Simulated Annealing

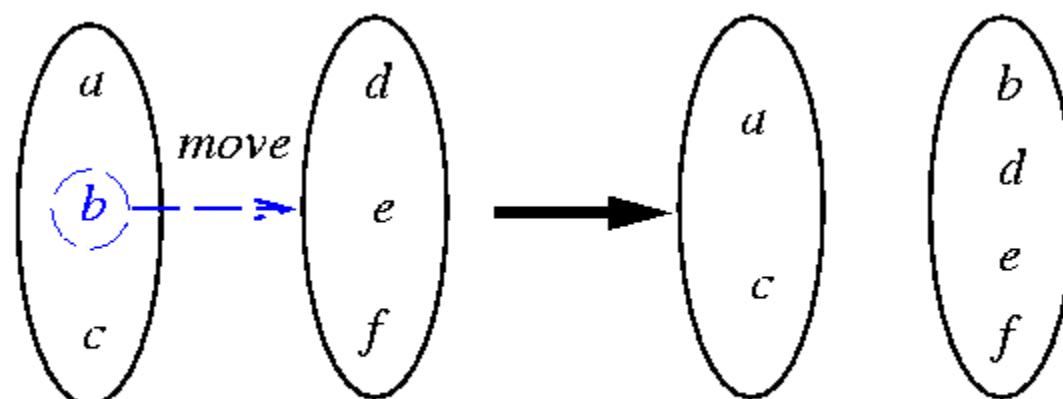
- Solution space
- Neighborhood structure
- Cost function
- Annealing schedule

Partitioning by Simulated Annealing

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



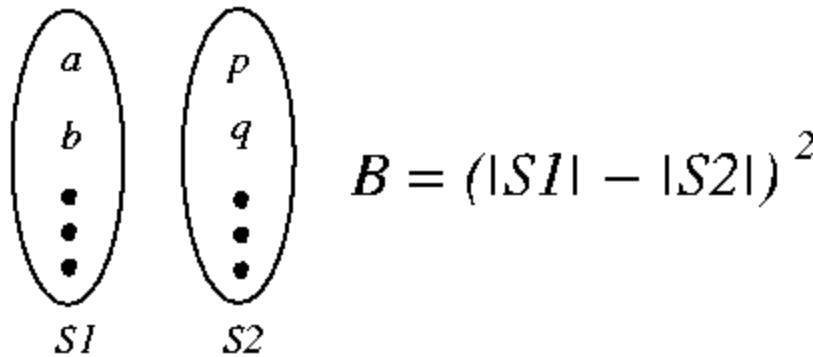
- **Neighborhood structure:**



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 balanced the solution is
 - λ : a constant



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