Partitioning

(4541.554 Introduction to Computer-Aided Design)

School of EECS

Seoul National University

Introduction

- **Layout System**
  - **Goal**
    - **Constraints**
      - **Design constraints**
        - **e.g. cell area, position, aspect ratio**
      - **Technological constraints**
        - **e.g. design rule, number of routing layers**
      - **Performance constraints**
        - **e.g. timing**
    - **Minimize area (performance, power)**
  - **Problem**
    - **Large set of configurations**
      - **e.g. linear-array cell placement**
        n-cells --> n! configurations
      - **Most layout optimization problems are NP-hard**
    - **Use heuristic algorithms**
      - **Partial search of the configuration space**
        --> local minimum

- **Heuristic Algorithm**
  - **Define topology on the configuration space**
  - **Model: graph G(V,E)**
  - **Define cost function f(v)**
  - **Global minimum:**

    v^* s.t. $f(v^*) <= f(v)$, $v \in V$

    Local minimum:

    v^* s.t. $f(v^*) <= f(v)$, $v \in V$ and $(v^*, v) \in E$
  - **Can be improved by look-ahead**
  - **Example: Linear placement: A B C**

    --> 6 different placements

    1 ABC

    2 BAC

    3 CAB

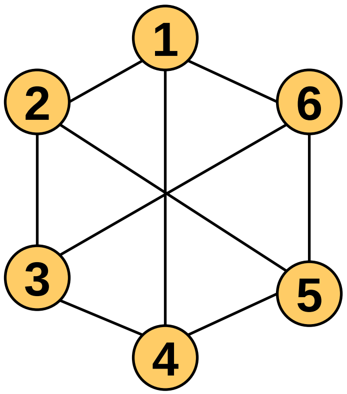
    4 ACB

    5 BCA

    6 CBA

    cost function: c(1)<c(3)<c(5)<c(2)<c(4)<c(6)

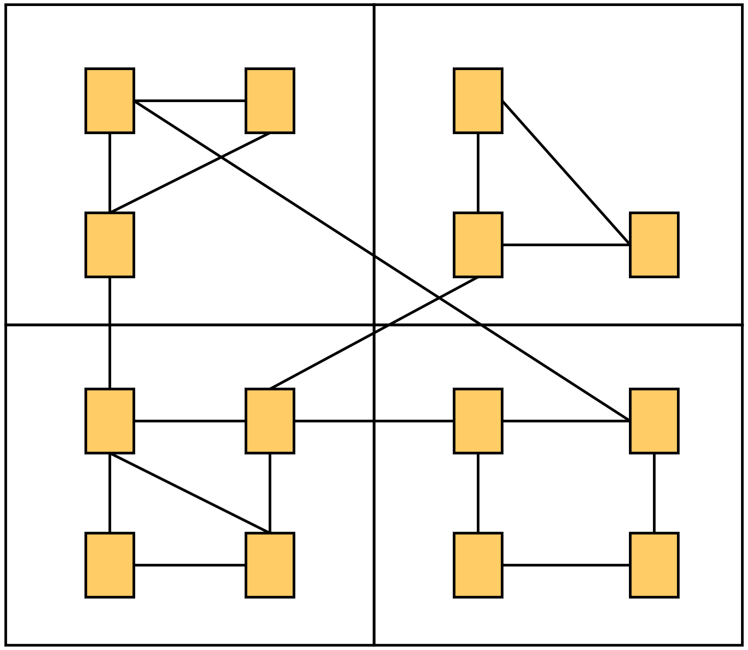
    --> If we start from 3 or 5, we cannot reach the global minimum



- **Major Stages of Layout Process**
  - **Partitioning**
  - **Floor-planning**
  - **Placement**
  - **Routing**
- **Problem Definitions**
  - **Cells (modules): objects with terminals (pins)**
  - **Nets: set of terminals**
  - **Partitioning: break a set of cells into subsets**
  - **Floor-planning: determine relative positions of cells**
  - **Placement: determine absolute positions of cells**
  - **Routing: provide interconnection of terminals**

Partitioning

- **Goals**
  - **Decrease problem size (provides hierarchy)**
  - **Ease placement and routing**



- **Problem Formulation**
  - **Given a set of n modules:**

    $M = \{m_1,\ldots,m_i,\ldots,m_n\}$

    and a set of nets:

    $N = \{n_1,\ldots,n_j,\ldots,n_k\}$

    $n_j = \{m_{j1},\ldots,m_{jl}\}$
  - **Find a partition of M:**

    $\Pi=\{\pi_1,\ldots,\pi_t\},$

    $\pi_i \subset M,\ \cup \pi_i = M,\ \pi_i \cap \pi_j = 0,\ i \neq j$

    subject to capacity constraints:

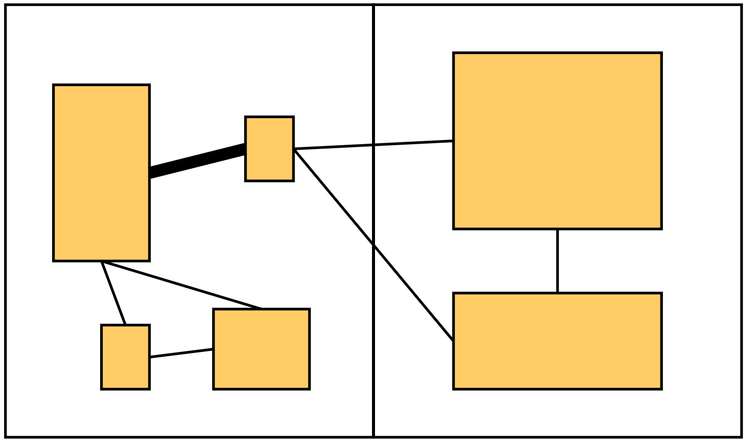
    $|\pi_i| \leq K_i,\ \Sigma K_i \geq n$

    which minimizes cost function (number of nets between partitions):

    $$C(\Pi) = \sum_{m_i \in \pi_h, m_j \in \pi_k, k \neq h} c_{ij},$$

    c_{ij}**=number of nets that connect** m_i **to** m_j

- Generalization
  - Use sizes of modules and weights of nets
- NP-hard
  - Use heuristics: constructive or iterative improvement



Constructive Method

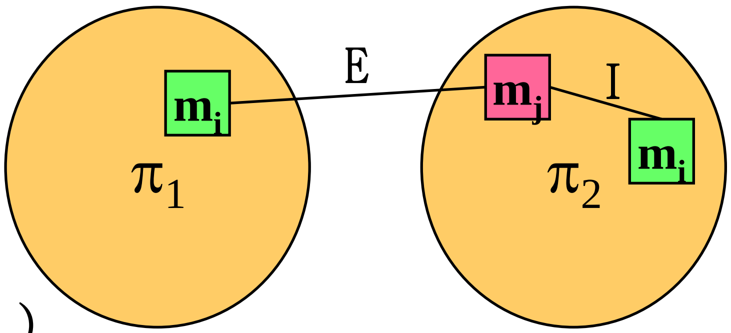
- **Assumption: bi-partition**
- **Definition**

$$\text{Internal cost } I(m_j) = \sum_{m_i \in \pi_2} c_{ij}$$

$$\text{External cost } E(m_j) = \sum_{m_i \in \pi_1} c_{ij}$$

$$\text{Cost function } C(m_j) = I(m_j) - E(m_j)$$

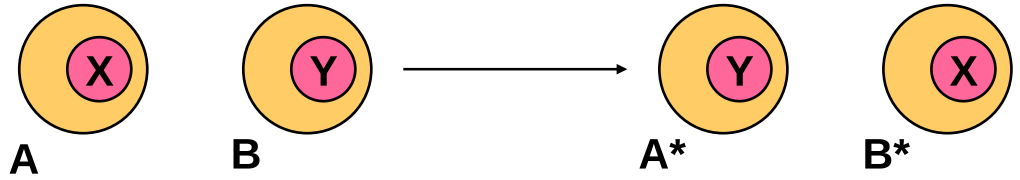
$$\text{Gain } D(m_j) = E(m_j) - I(m_j)$$



- **Algorithm (Greedy Algorithm)**
  - **Select a seed (1st module to be assigned to π_1)**
    - **e.g. select a module with most net connections**
  - **Repeat selecting the next module with minimal cost until size limit is reached**
  - **Fast but the result may not be good**
  - **The result can be a starting point of iterative improvement.**

Iterative Improvement

- **Algorithm**
  - Start from an initial solution
  - Modify incrementally by swapping and monitoring the objective function



  - Random interchange
    - Choose swap at random
    - Accept the swap only if it decreases the cost

Kernighan-Lin Algorithm

- **Definition**
  - **Gain obtained by moving m_i from π_1 to π_2:**

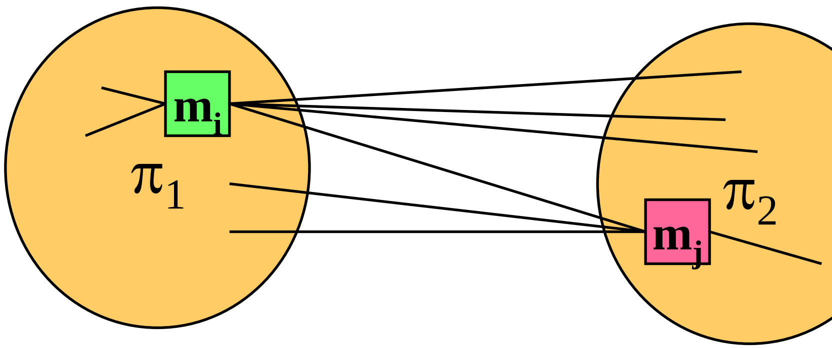
    $D(m_i)=E(m_i)-I(m_i)$
  - **Gain obtained by moving m_j from π_2 to π_1:**

    $D(m_j)=E(m_j)-I(m_j)$
  - **Gain obtained by interchanging $m_i \in \pi_1$ and $m_j \in \pi_2$:**

    $gain=g_k$ (k-th iteration)

    $=D(m_i)+D(m_j)-2c_{ij}$

    $=E(m_i)-I(m_i)+E(m_j)-I(m_j)-2c_{ij}$



gain=4−2+3−1−2=2

- **Algorithm**

  Repeat

  Compute D values for all modules

  Repeat

  Choose $m_i \in \pi_1$ and $m_j \in \pi_2$ such that the gain is maximum

  Fix $m_i \in \pi_2$ and $m_j \in \pi_1$

  Update D values for modules of π_1-m_i and π_2-m_j

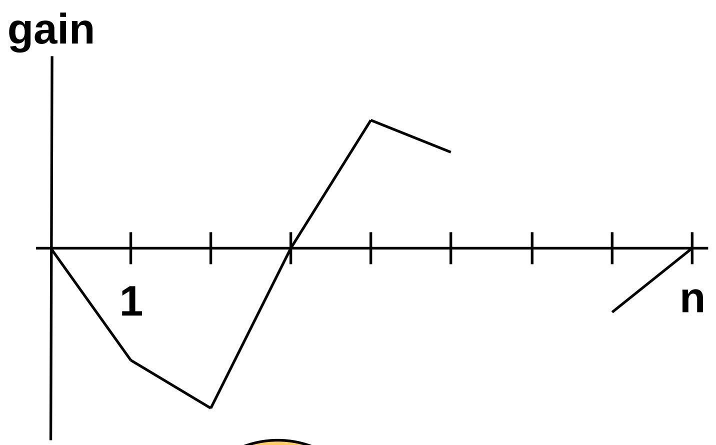
  Compute $$G_k = \sum_{i=1}^{k} g_i$$

  Until all modules are fixed

  Choose k* that maximize G_k

  If $G_{k*}>0$, Swap first k* pairs

  Until $G_{k*}=0$

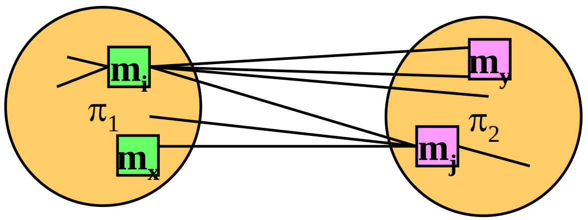


  - **Updating D values**

    $D'(m_x)=D(m_x)+2c_{xi}-2c_{xj}$, $m_x \in \pi_1$-m_i

    $D'(m_y)=D(m_y)+2c_{yj}-2c_{yi}$, $m_y \in \pi_2$-m_j

  - **$G_n=0$ (n=number of modules in a partition)**

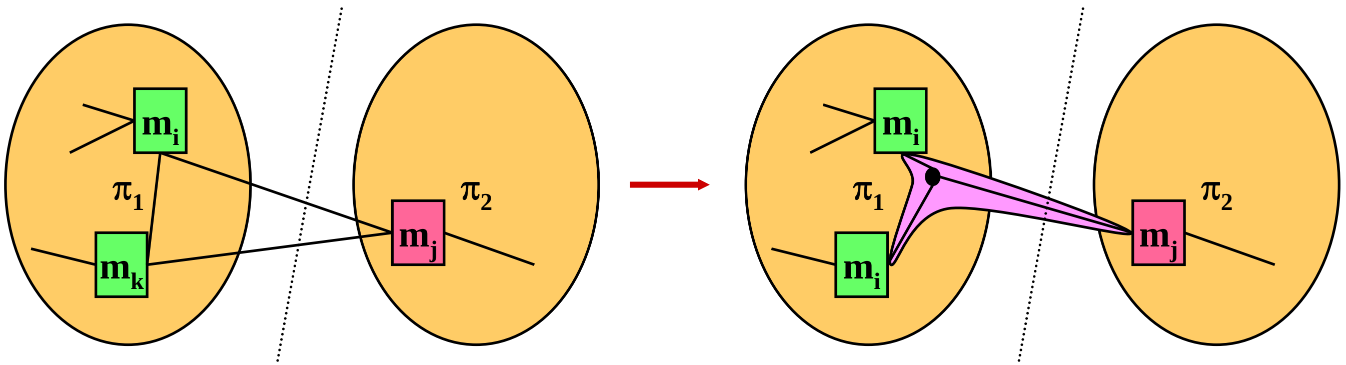


- **Complexity**
  - **Sorting: $O(n \log n)$**
  - **Maximum gain is found rapidly**
    - **$D(m_{x1}) >= D(m_{x2}) >= D(m_{x3})$ ...**
      $D(m_{y1}) >= D(m_{y2}) >= D(m_{y3})$ ...
      Examine module m_{xi} only when
      $$D(m_{y1})+D(m_{xi})>D(m_{x1})+D(m_{y1})-2c_{x1y1}$$
  - **$O(n \log n) + O((n-1) \log (n-1)) + ... = O(n^2 \log n)$**

Fiduccia-Mattheyses Algorithm

- **Modified Version of Kernighan-Lin Algorithm**
  - **Generate balanced partitions**
    - **Non-uniform cell sizes are considered**
    - **Single cell is moved in a single move**
  - **More accurate cost computation**
    - **Consider multi-pin nets**

      --> Extension to hypergraph (cut of nets rather than edges)



  - **Fast algorithm**
    - **Use bucket sorting**

      --> Speed up the sorting process

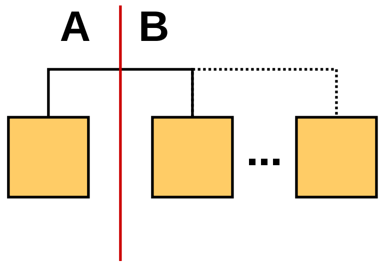
- **Notations**
  - **C: number of cells**
  - **N: number of nets**
  - **n(i): number of cells connected by net i**
  - **s(j): size of cell j**
  - **p(j): number of pins of cell j**
  - **P: total number of pins,** $P = \sum_{j=1}^{C} p(j)$
  - **C=O(P), N=O(P)**
  - **cutstate of a net: {cut, uncut}**
  - **cutset: set of all nets that are cut**
  - **|X|: size of partition X,** $|X| = \sum_{j \in X} s(j)$
  - **g(j):gain of cell j, number of nets by which the cutset would decrease if cell j is moved to the other partition**

$$-p(j) \leq g(j) \leq +p(j)$$

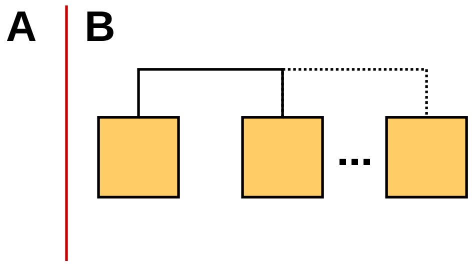
$$-pmax \leq g(j) \leq +pmax, \forall j$$

$$\text{where } pmax = \max_j p(j)$$

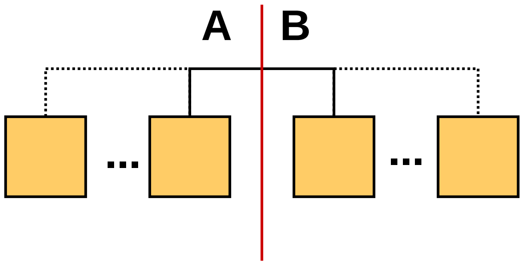
- nc(i,X): number of cells that are in partition X and connected by net i
- critical net: a net connecting a cell whose move changes the net's cutstate
  - a net i is critical iff nc(i,A) or nc(i,B) is either 0 or 1
  - cutstate of a non-critical net is not affected by a move
  - if a net is not critical before and after a move, the gains of its cells due to the net are not affected by the move

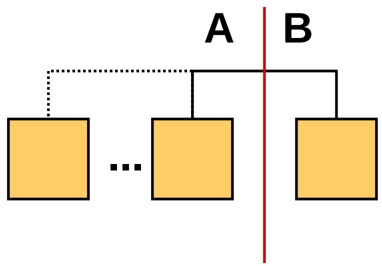


nc(i,A)=1

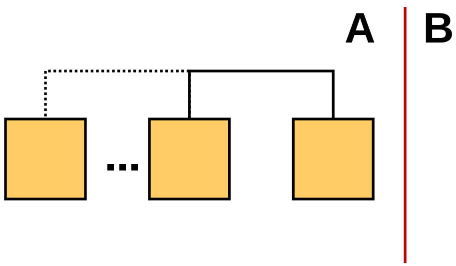


nc(i,A)=0





nc(i,B)=1

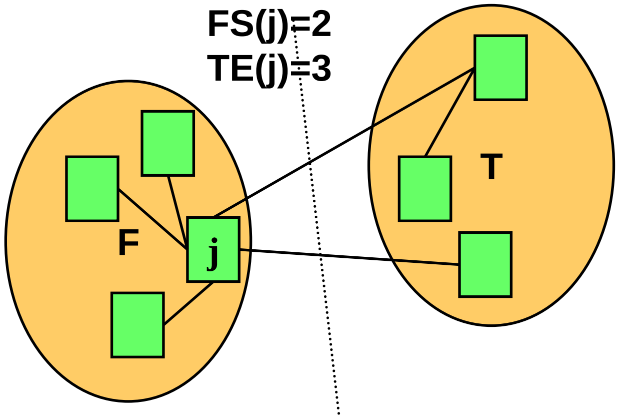


nc(i,B)=0

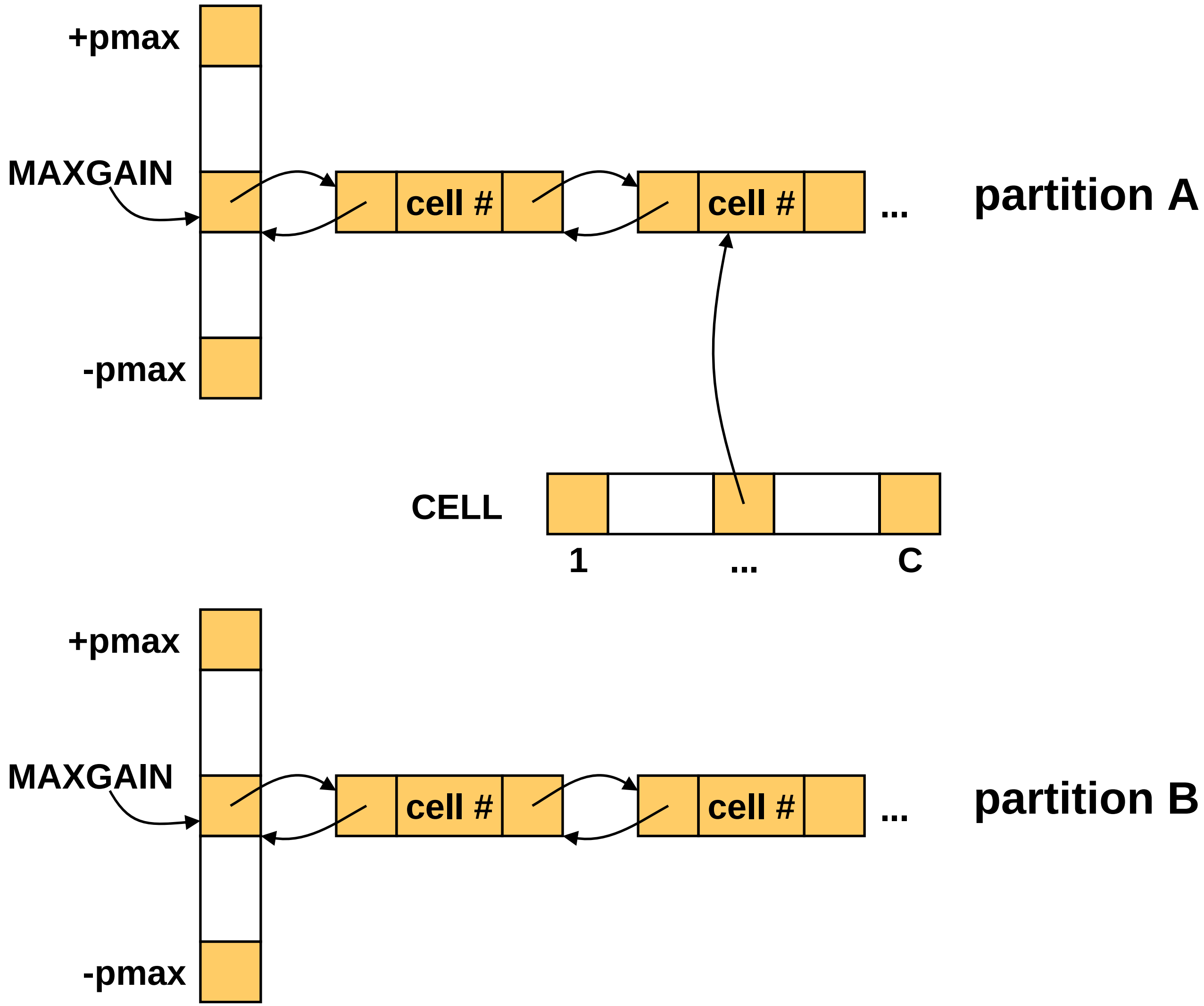
- **Computing Initial Cell Gains**
  - F(j): 'From' partition with respect to cell j
  - T(j): 'To' partition with respect to cell j
  - FS(j): # of nets having cell j as their only F cell
  - TE(j): # of nets having cell j but no T cell
  - g(j)=FS(j)-TE(j)
  - Algorithm

```
FOR each cell j DO
    g(j)=0;
    FOR each net i connecting cell j DO
        IF nc(i,F(j))=1 THEN increment g(j);
        IF nc(i,T(j))=0 THEN decrement g(j);
    END FOR;
END FOR;
```

  - Complexity: O(P) (1)



- **Data Structure**
  - **Sorting: O(C)-->complexity of initialization=O(pmax)+O(C)=O(P) (2)**



- **Establishing Balance**
  - **Given a ratio r, 0<r<1, a partition (A,B) is said to be balanced if**

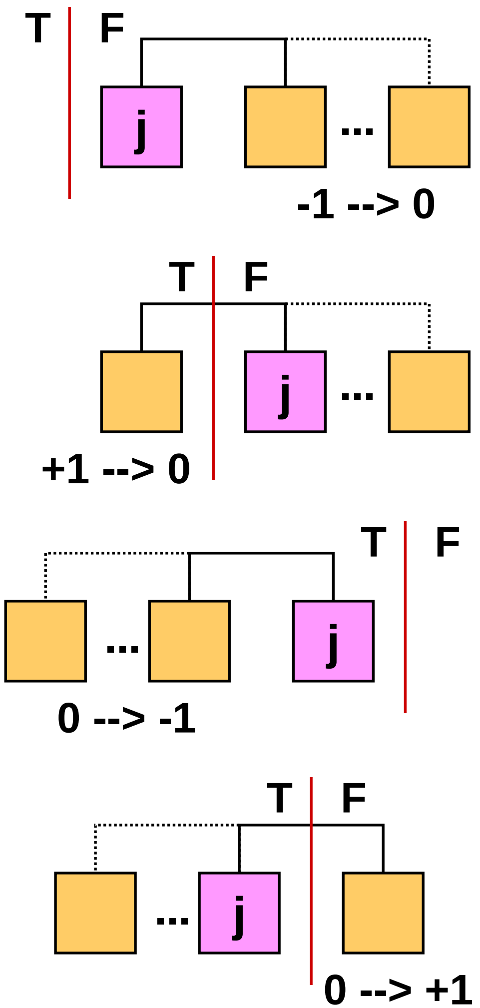
$$rW - smax \leq |A| \leq rW + smax$$

$$\text{where } W = |A| + |B| \text{ and } smax = \max_j s(j)$$

  - **Tolerance of ±k*smax may be used, where k>1 is some slowly growing function of C**
- **Selecting a Cell**
  - **Consider the cell of highest gain from each bucket array**
  - **Reject candidate cells that would cause imbalance**
  - **If neither block has a qualifying cell, stop the current pass**
  - **Among the candidates, choose a cell of highest gain**
  - **Break tie considering balance**

- **Updating Cell Gains**
  - When moving cell j, cell gains of other cells connected to net i change, if nc(i,T(j))=0 or 1 before the move or nc(i,F(j))=0 or 1 after the move (i.e., if i is critical before or after the move)
  - Algorithm

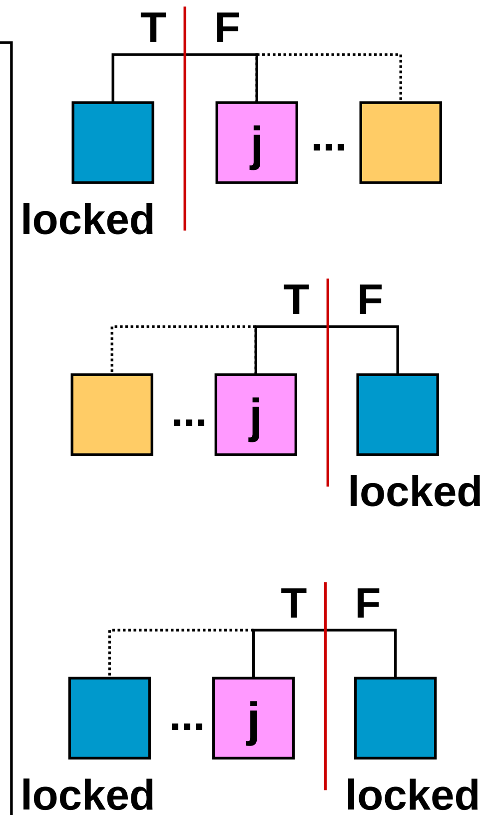
```
FOR each net i connecting the cell j DO
   /* check before the move */
   IF nc(i,T(j))=0 THEN
      increment gains of all free cells connected by net i
   ELSE IF nc(i,T(j))=1 THEN
      decrement gain of the only T cell, if it is free
   /* move */
   decrement nc(i,F(j))
   increment nc(i,T(j))
   /* check after the move */
   IF nc(i,F(j))=0 THEN
      decrement gains of all free cells connected by net i
   ELSE IF nc(i,F(j))=1 THEN
      increment gain of the only F cell, if it is free
END FOR
```



- **Complexity of Updating Cell Gains**
  - **No more than three update operations per net**
  - **Proof**

    nlc(i,X): number of locked cells that are in partition X and connected by net i

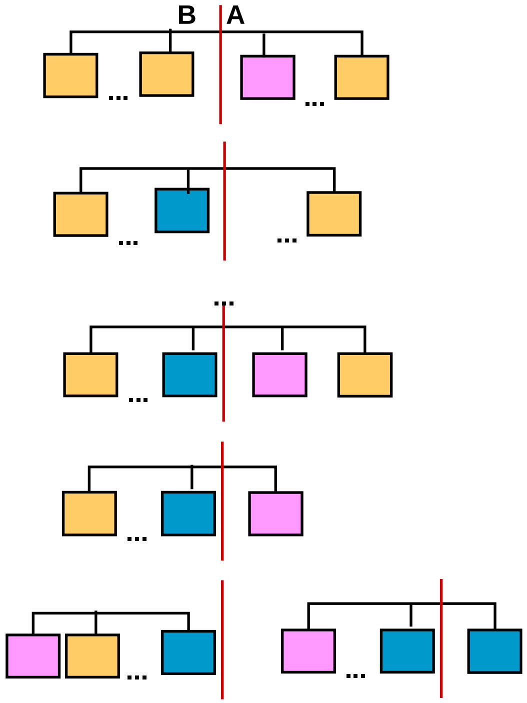
```
/* check before the move */
IF nlc(i,T(j))=0 THEN
    IF nc(i,T(j))=0 THEN
        increment gains of all free cells connected by net i
    ELSE IF nc(i,T(j))=1 THEN
        decrement gain of the only T cell(, if it is free)
/* move */
decrement nc(i,F(j))
increment nc(i,T(j))
/* check after the move */
IF nlc(i,F(j))=0 THEN
    IF nc(i,F(j))=0 THEN
        decrement gains of all free cells connected by net i
    ELSE IF nc(i,F(j))=1 THEN
        increment gain of the only F cell(, if it is free)
```



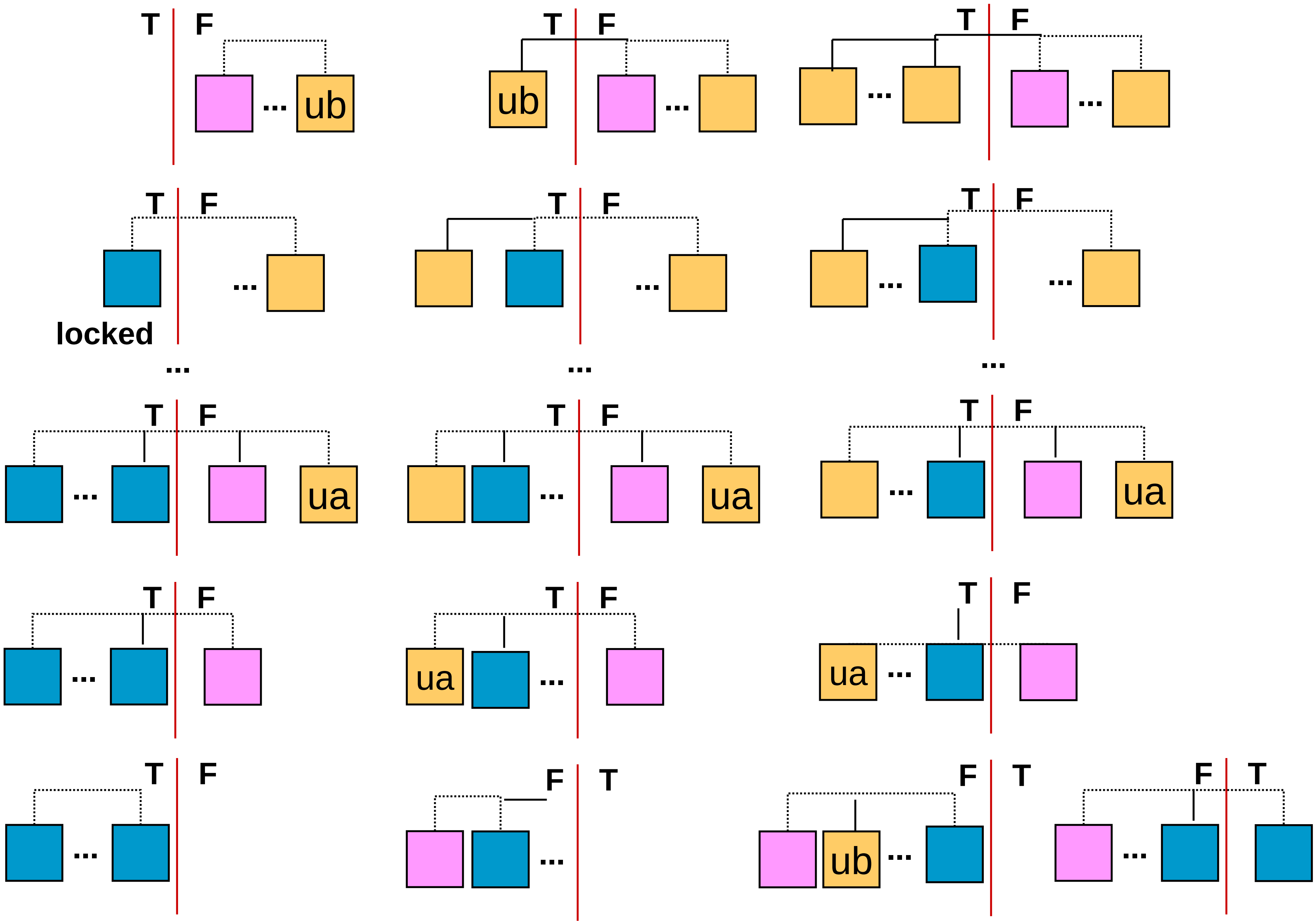
total 4 times

- Consider moving cells from A partition to B partition.
- Initial move will make a locked cell in B, so no pre_update in that direction from now on.
- If we move a cell from B to A, then another locked cell in A and no further update for the net.
- But if we continue moving cells from A to B, there can be two more updates when one cell is left and then no cell is left in A.
- Then if we move a cell from B to A, then A will have a locked cell.
- So total 4 updates

```
/* check before the move */
IF nlc(i,T(j))=0 THEN
   pre_update
/* move */
decrement nc(i,F(j))
increment nc(i,T(j))
/* check after the move */
IF nlc(i,F(j))=0 THEN
   post_update
```



– **In reality, total three updates**



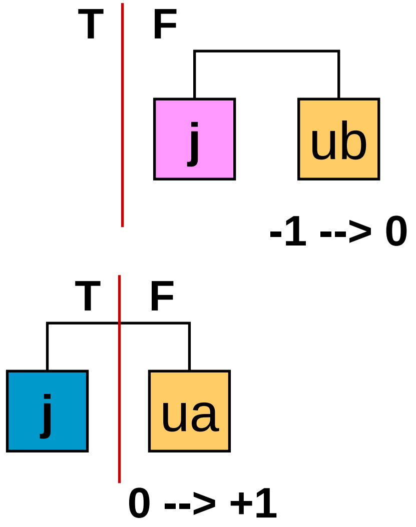
- **Complexity of the Algorithm**
  - **Updating Cell Gains**
    - **Total number of gain adjustments per pass**

      $= O(3 \cdot \sum_{i=1}^{N} n(i)) = O(P)$ **(3)**

    - **During one update, MAXGAIN can be reset to at most MAXGAIN+2**

      --> total amount of MAXGAIN increase

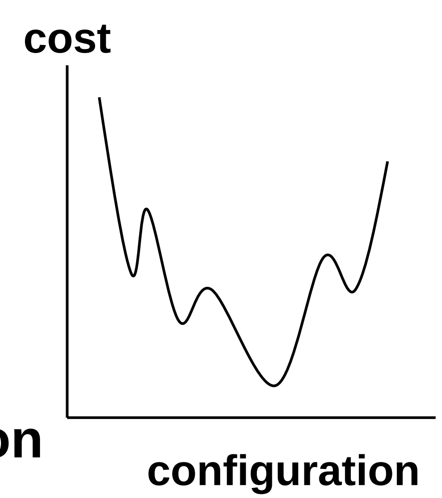
      =O(3*N*2)=O(N)=O(P) **(4)**



  - **Total complexity of one pass**
    - **(1)+(2)+(3)+(O(pmax)+(4))=O(P)**

Simulated Annealing

- **Introduction**
  - General method
  - Applied first to CAD problem (placement and routing) by S.Kirkpatric, C.D.Gelatt, Jr., and M.P.Vecchi, " Optimization by simulated annealing," Science, vol. 220, no. 4598, pp. 671-680, 13 May 1983
  - Random interchange (hill climbing) --> local minimum
  - Escape from the local minimum
  - Probabilistic algorithm



- **Annealing**
  - Method to obtain crystals
  - Warm up to melting point
  - Cool down slowly to allow crystallization
  - Rate of decrease of temperature is very slow around the melting point

- **Simulation of Equilibrium States**
  - **N.Metropolis, A.Rosenbluth, M.Rosenbluth, A.Teller, and E.Teller, "Equation of State Calculations by Fast Computing Machines," Journal of Chemical Physics, June 1953**
  - **Equilibrium at a given temperature**
  - **Algorithm**

    Generate random interchanges

    Compute the difference in energy, dE

    Accept the move with probability

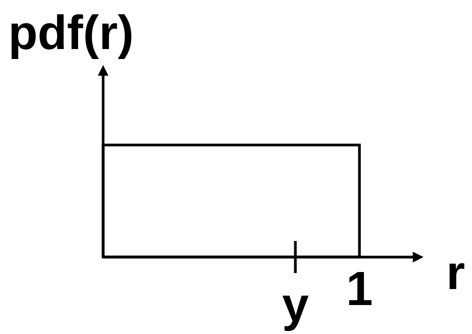
    min(1, exp(-dE/kT))
  - **Downhill moves (dE < 0) are always accepted**
  - **After a large set of moves, the simulated system is in equilibrium at T**
  - **Boltzmann distribution**

- **Simulated Annealing**
  - **Run Metropolis algorithm at decreasing temperatures**
    - **state --> configuration**
      energy --> cost
      ground state --> optimum solution
  - **Problems**
    - **How to decrease temperature**
      --> Cooling schedule
    - **How to accept moves**
      --> min(1, exp(-dE/kT))
    - **How many moves and how wide**
      --> Limit number of moves and ranges
    - **When to stop**
      --> No further improvement

- **Algorithm**

```
Simulated_Annealing(j0, T0) {
  /* Given an initial state s0 and an initial
     temperature T0 */
  T=T0;
  s=s0;
  while(stopping criterion is not satisfied) {
    while(inner loop criterion is not satisfied) {
      snew=generate(s)
      if(accept(c(snew), c(s), T))
        s=snew;
    }
    T=update(T);
  }
}
```

```
accept(c(j), c(i), T) {
  /* returns 1 if the cost variation passes a test
  */
  dE=c(j)-c(i);
  y=f(dE, T); /* exp(-dE/kT)*/
  r=random(0, 1);
  /* random is a function which returns a pseudo
  random number uniformly distributed on the
  interval [0, 1] */
  if(r<y)
     return(1);
  else
     return(0);
}
```



- **Mathematical model:**
  - **Markov chain (memoryless)**
- **Mathematical analysis results:**
  - **Sufficient conditions for reaching global minimum with probability one:**

    (1) At each temperature the process reaches equilibrium

    --> Infinite number of moves at each temperature

    (2) The cooling schedule is

    $T_k = c/\ln(k+a),\ a >= 1$

    --> Temperature drops infinitely slow

    $(dT_k/dk)(1/T_k) = -1/((k+a)\ln(k+a)) \xrightarrow[k \to \infty]{} 0$
  - **Theoretical results only**

    --> Basis for good heuristic