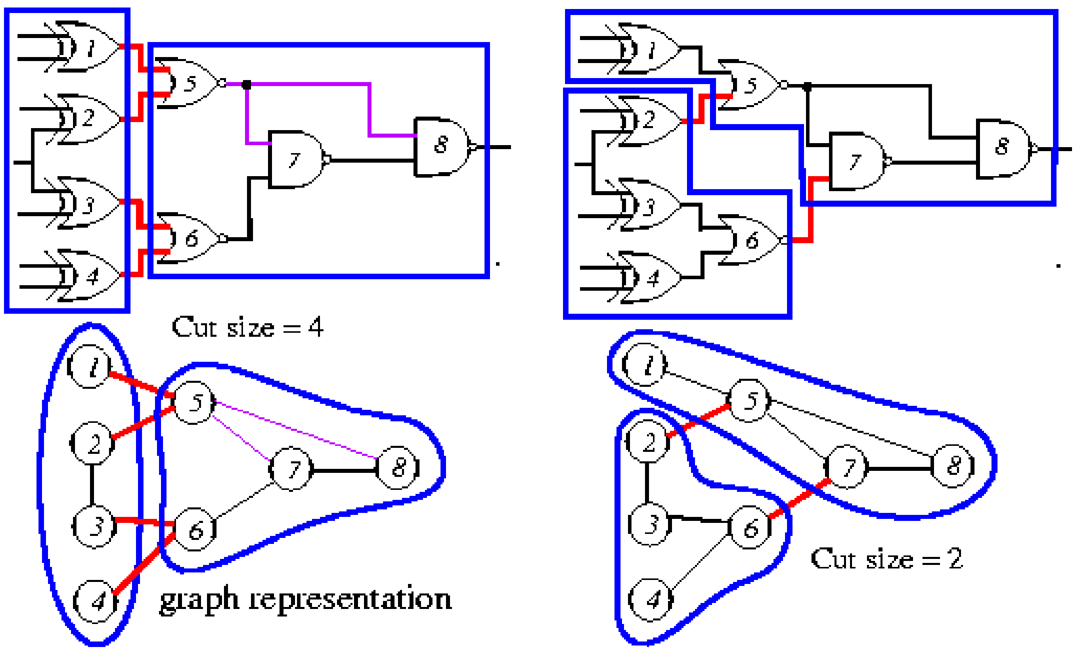
Unit 3: Circuit Partitioning

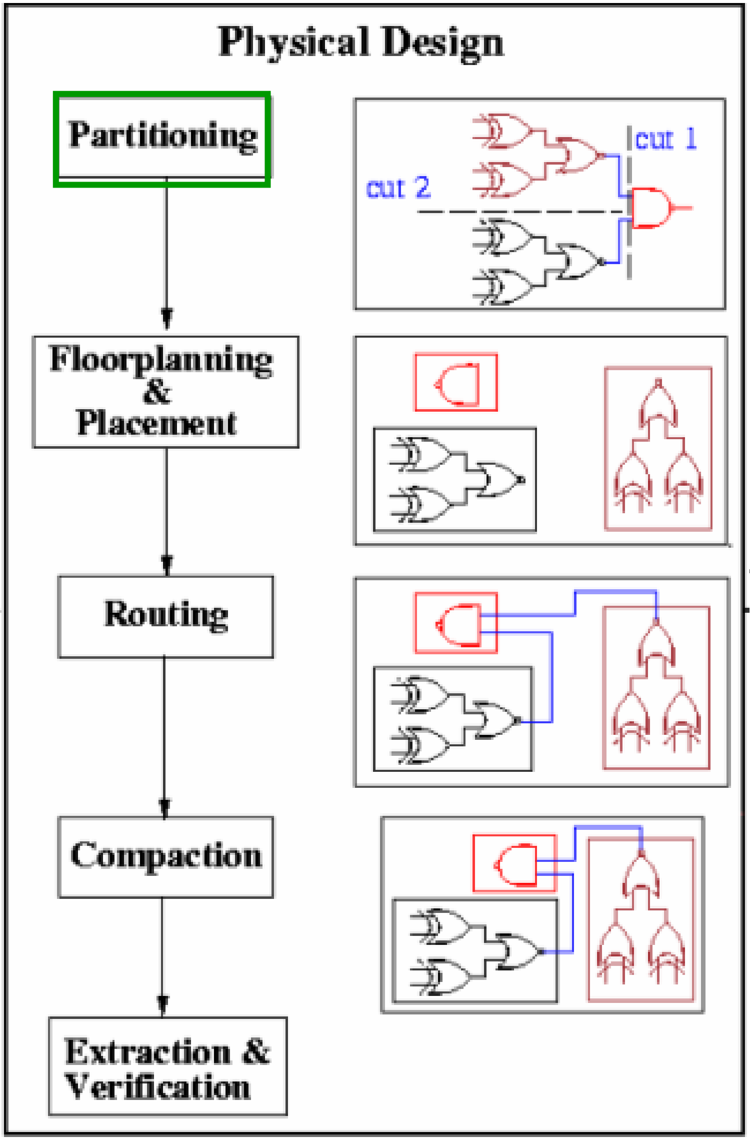
- Course contents:
  - Kernighan-Lin heuristic
  - Simulated annealing based partitioning algorithm
  - Fiduccia-Mattheyses heuristic



Slides Courtesy of Prof. H.-R. Jiang

Position of Partitioning

Front end design

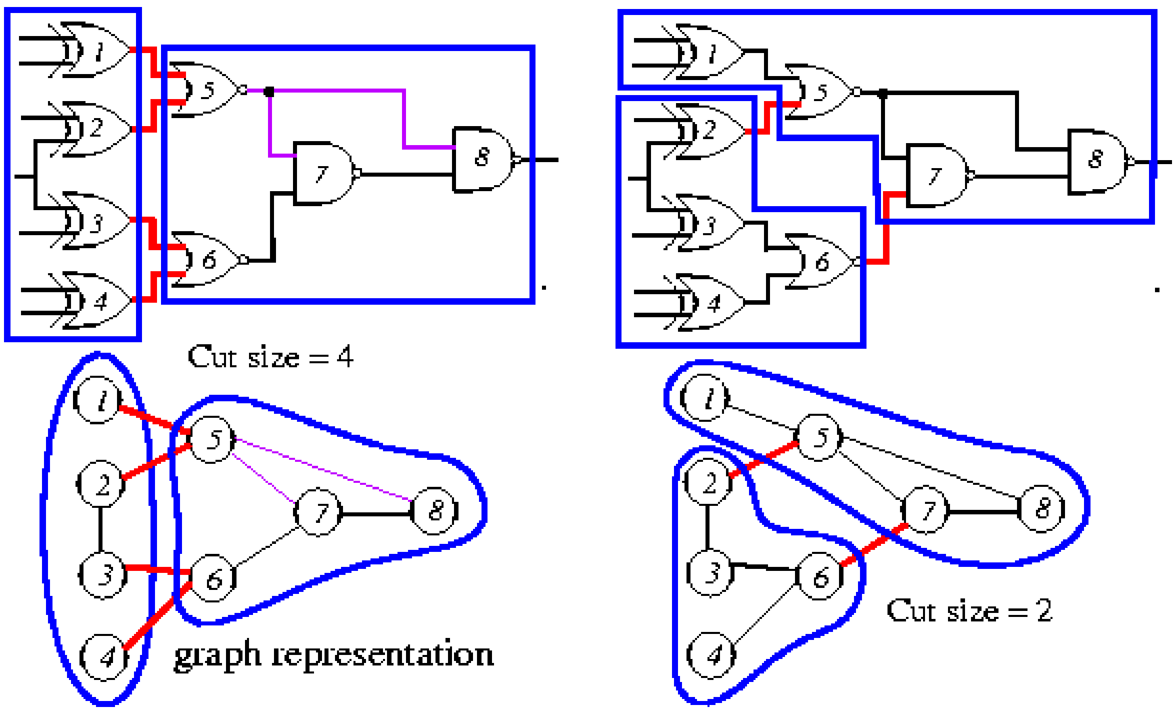


Fabrication

Slides Courtesy of Prof. H.-R. Jiang

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?



Slides Courtesy of Prof. H.-R. Jiang

Problem Definition: Partitioning

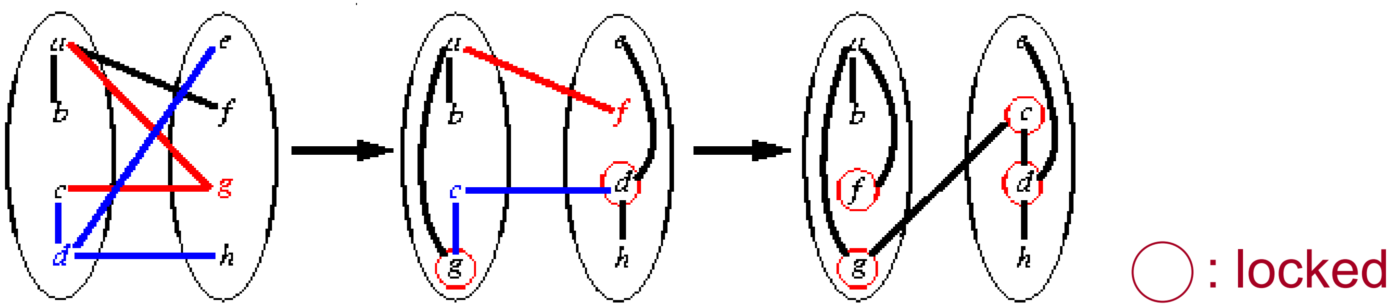
- ***k*-way partitioning:** Given a graph $G(V, E)$, where each vertex $v \in V$ has a **size** $s(v)$ and each edge $e \in E$ has a **weight** $w(e)$, the problem is to divide the set V into k disjoint subsets $V_1, V_2, \ldots, V_k$, such that an objective function is optimized, subject to certain constraints.
- **Bounded size constraint:** The size of the i-th subset is bounded by B_i ($\sum_{v \in V_i} s(v) \leq B_i$).
  - 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.
  - May not be balanced.
- 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.

K-L Algorithm: A Simple Example

- Each edge has a unit weight.



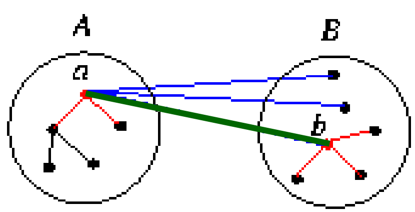
Step #	Vertex pair	Cost reduction	Cut cost
0	-	0	5
1	{d, g}	3	2
2	{c, f}	1	1
3	{b, h}	-2	3
4	{a, e}	-2	5

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

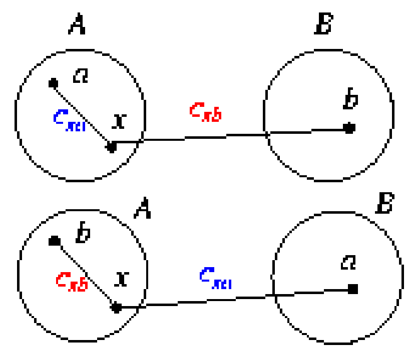
Properties

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

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



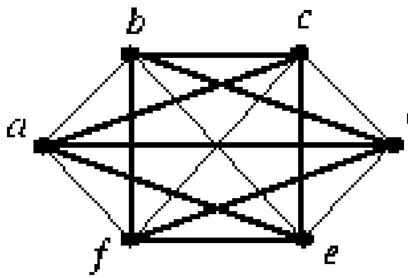
$Gain_{a \to B} : D_a - c_{ab}$
$Gain_{b \to A} : D_b - c_{ab}$
Internal cost vs. External cost



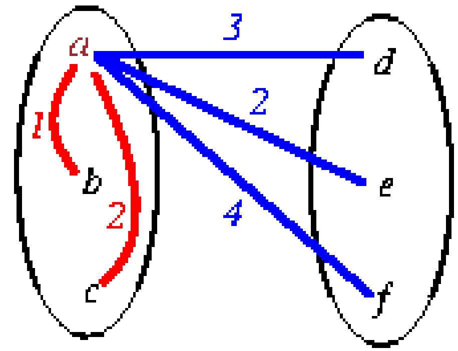
before swap	after swap	$\triangle C$
$-c_{xa}$	$+c_{xa}$	$+2c_{xa}$
$+c_{xb}$	$-c_{xb}$	$-2c_{xb}$

updating D−values

K-L 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:

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

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

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

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

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

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

Computing the g Value

- Iteration 1:

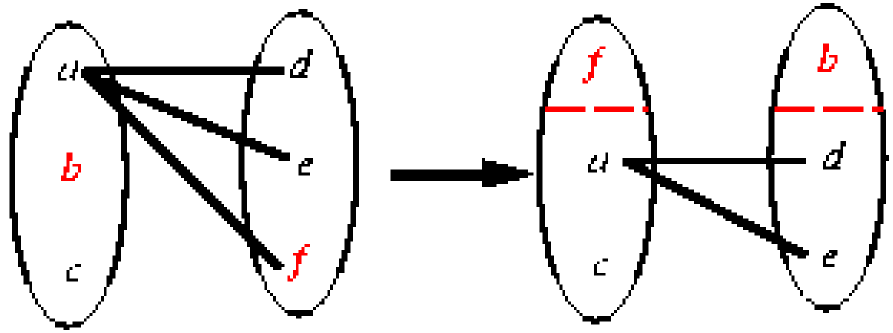
$$\begin{array}{lll}
I_a = 1+2=3; & E_a = 3+2+4=9; & D_a = E_a - I_a = 9-3=6 \\
I_b = 1+1=2; & E_b = 4+2+1=7; & D_b = E_b - I_b = 7-2=5 \\
I_c = 2+1=3; & E_c = 3+2+1=6; & D_c = E_c - I_c = 6-3=3 \\
I_d = 4+3=7; & E_d = 3+4+3=10; & D_d = E_d - I_d = 10-7=3 \\
I_e = 4+2=6; & E_e = 2+2+2=6; & D_e = E_e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

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

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

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

Updating the D Value



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

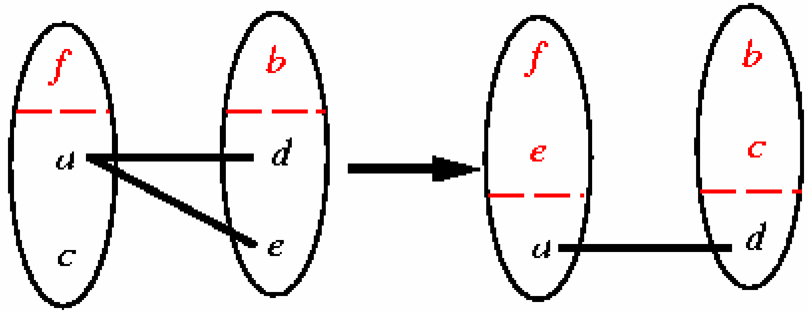
$$\begin{aligned} D'_a &= D_a + 2c_{ab} - 2c_{af} = 6 + 2 \times 1 - 2 \times 4 = 0 \\ D'_c &= D_c + 2c_{cb} - 2c_{cf} = 3 + 2 \times 1 - 2 \times 1 = 3 \\ D'_d &= D_d + 2c_{df} - 2c_{db} = 3 + 2 \times 3 - 2 \times 4 = 1 \\ D'_e &= D_e + 2c_{ef} - 2c_{eb} = 0 + 2 \times 2 - 2 \times 2 = 0 \end{aligned}$$

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

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

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

Determining Swapping Pairs



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

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

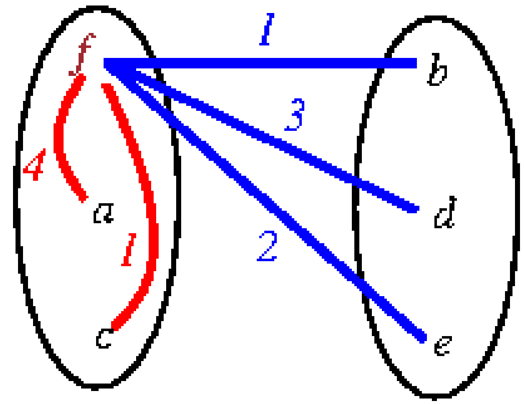
- $g_{xy} = D''_x + D''_y - 2c_{xy}$.

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

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

Next Iteration

	a	b	c	d	e	f
a	0	1	2	3	2	4
b	1	0	1	4	2	1
c	2	1	0	3	2	1
d	3	4	3	0	4	3
e	2	2	2	4	0	2
f	4	1	1	3	2	0



$$Initial\ cut\ cost = (1+3+2)+(1+3+2)+(1+3+2) = 18\ (22-4)$$

- 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!

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 $|V_A| = |V_B|$, $V_A \cap V_B = \emptyset$, and $V_A \cup V_B = V$.
repeat
  Compute D_v, $\forall v \in V$.
  for $i = 1$ **to** n **do**
    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;
    Mark v_{ai} and v_{bi} as locked, store the gain $\hat{g}_i$, and compute the new D_v, for all unlocked $v \in V$;
  Find k, such that $G_k = \sum_{i=1}^{k} \hat{g}_i$ is maximized;
  if $G_k > 0$ **then**
    Move $v_{a1}, \ldots, v_{ak}$ from V_A to V_B and $v_{b1}, \ldots, v_{bk}$ from V_B to V_A;
  Unlock v, $\forall v \in V$.
until $G_k \leq 0$;
end

Time Complexity

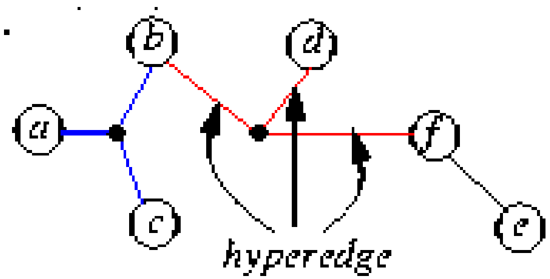
- Line 4: Initial computation of D: $O(n^2)$
- Line 5: The **for**-loop: $O(n)$
- The body of the loop: $O(n^2)$.
  - Lines 6—7: Step i takes $(n - i + 1)^2$ time.
- Lines 4--11: Each pass of the repeat loop: $O(n^3)$.
- Suppose the repeat loop terminates after r passes.
- The total running time: $O(rn^3)$.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

Extensions of K-L Algorithm

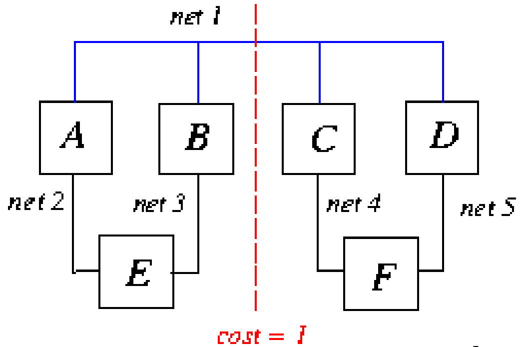
- **Unequal sized subsets** (assume n_1 **<** n_2)
  1. Partition: $|A| = n_1$ and $|B| = n_2$.
  2. Add $n_2 - n_1$ dummy vertices to set A. Dummy vertices have no connections to the original graph.
  3. Apply the Kernighan-Lin algorithm.
  4. Remove all dummy vertices.
- **Unequal sized "vertices"**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

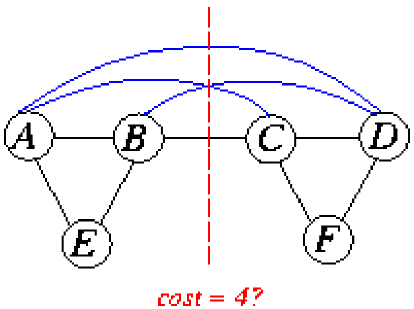
Coping with Hypergraph

- 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 $|N_i| \geq 2$.



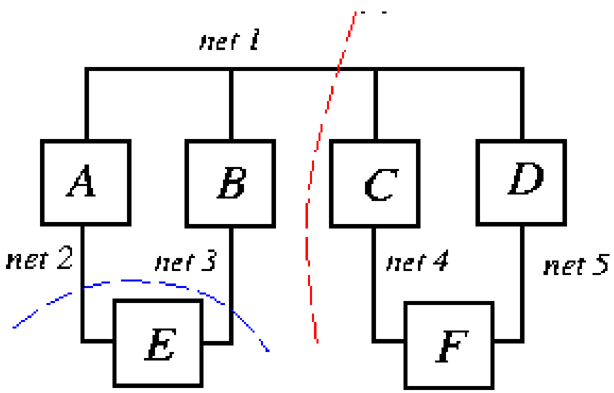
- Schweikert and Kernighan, "A proper model for the partitioning of electrical circuits," 9th Design Automation Workshop, 1972.
- For multi-terminal nets, **net cut** is a more accurate measurement for cut cost (i.e., deal with hyperedges).
  - {A, B, E}, {C, D, F} is a good partition.
  - Should not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

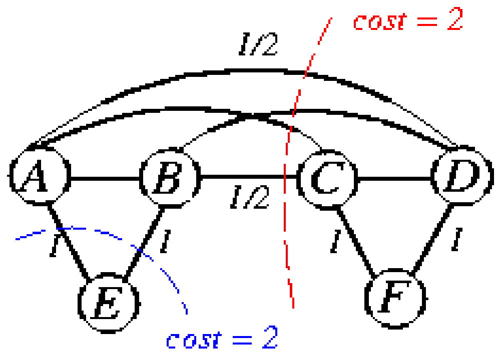




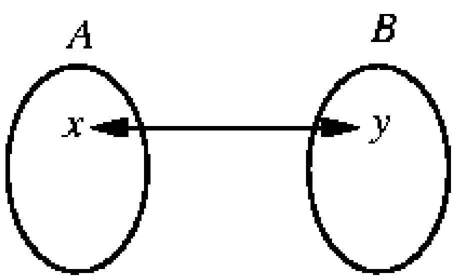
Net-Cut Model

- 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.



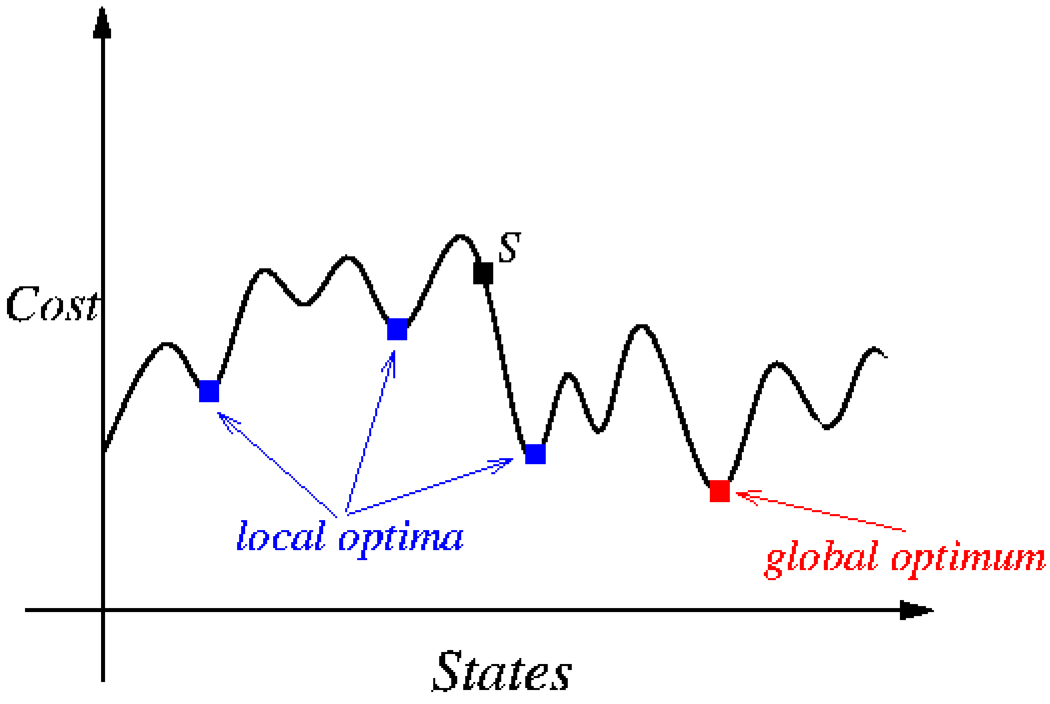
D_x: gain in moving x to B

D_y: gain in moving y to A

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

Simulated Annealing

- Kirkpatrick, Gelatt, and Vecchi, “Optimization by simulated annealing,” *Science*, May 1983.
- Greene and Supowit, “Simulated annealing without rejected moves,” ICCD-84.



Simulated Annealing Basics

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

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

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

Generic Simulated Annealing Algorithm

begin
Get an initial solution S;
Get an initial temperature $T > 0$;
while not yet "frozen" **do**
  for $1 \leq i \leq P$ **do**
    Pick a random neighbor S' of S;
    $\Delta \leftarrow cost(S') - cost(S)$;
    /* downhill move */
    if $\Delta \leq 0$ **then** $S \leftarrow S'$
    /* uphill move */
    if $\Delta > 0$ **then** $S \leftarrow S'$ with probability $e^{-\frac{\Delta}{T}}$;
$T \leftarrow rT$; /* reduce temperature */
return S
end

Slides Courtesy of Prof. H.-R. Jiang

Basic Ingredients for Simulated Annealing

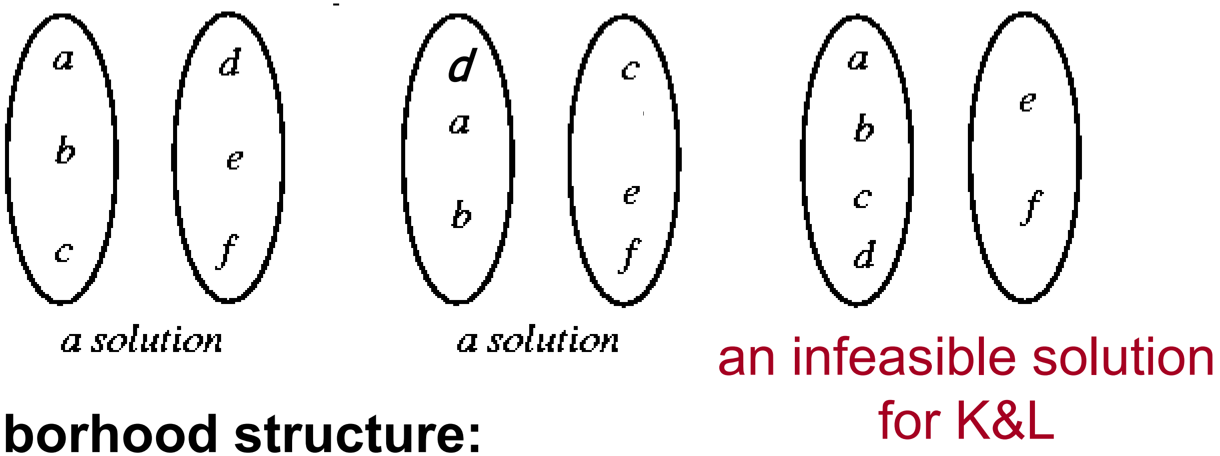
- Analogy:

Physical system	Optimization problem
state	configuration
energy	cost function
ground state	optimal solution
quenching	iterative improvement
careful annealing	simulated annealing

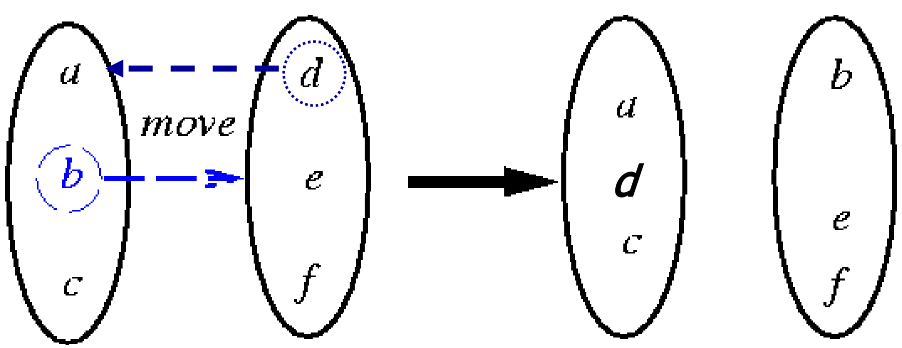
- Basic Ingredients for Simulated Annealing:
  - **Solution space**
  - **Neighborhood structure**
  - **Cost function**
  - **Annealing schedule**

Partition by Simulated Annealing

- Kirkpatrick, Gelatt, and Vecchi, "Optimization by simulated annealing," *Science*, May 1983.
- **Solution space:** set of all partitions



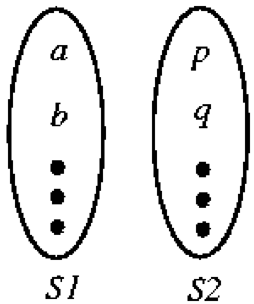
- **Neighborhood structure:**



Randomly swap a pair of cells from each side

Partition by Simulated Annealing (cont'd)

- **Cost function:** $f = C$
  - C: the partition cost as used before.



$$B = (|S1| - |S2|)^2$$

- **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 x total # of cells.
  - The system is "frozen" if very low acceptances at 3 consecutive temperatures.