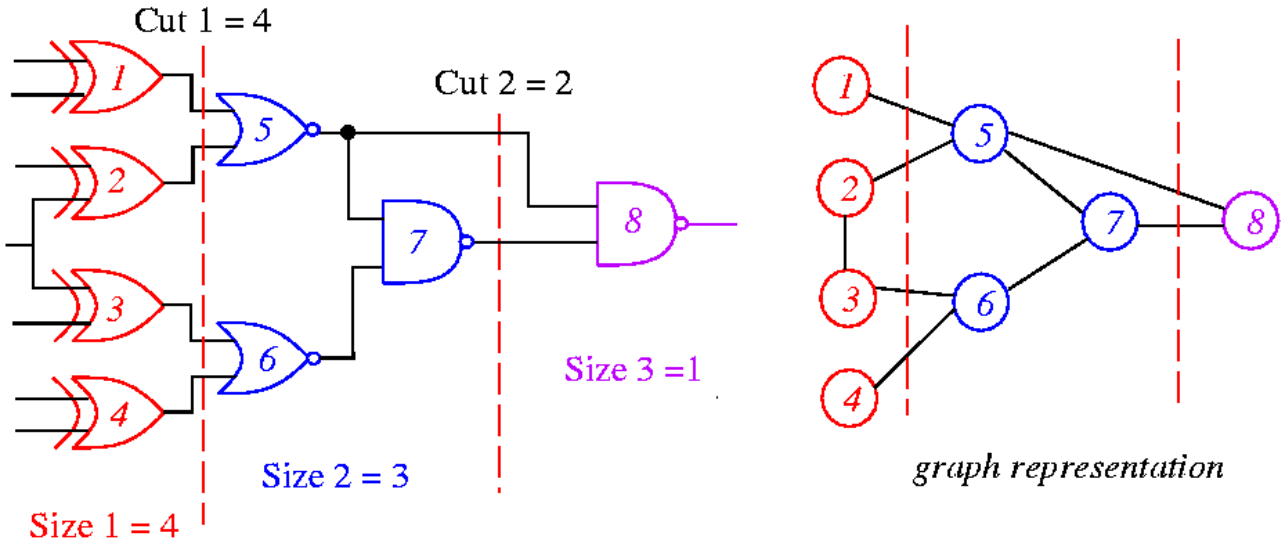
Partitioning

- 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 $V_1, V_2, \ldots, V_k$, such that an objective function is optimized, subject to certain constraints.
- **Bounded size constraint:** The size of the i-th subset is bounded by L_i and $U_i \left(\text{i.e.}, L_i \le \sum_{v \in V_i} s(v) \le U_i \right)$
- **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

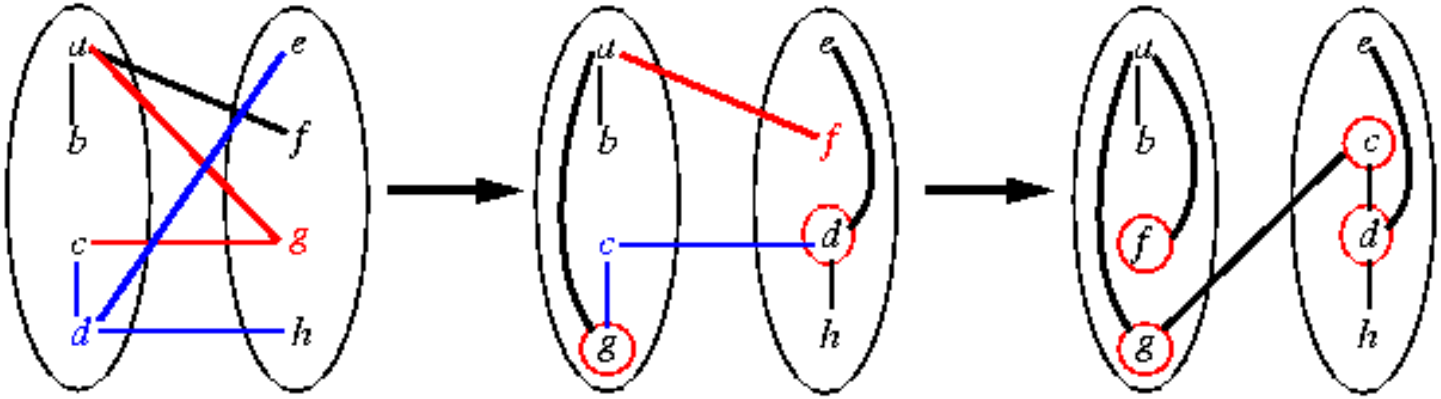
- Kernighan and Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, Feb. 1970.
- An **iterative**, **2-way**, **balanced** partitioning (bi-sectioning) heuristic.
- Restrictions:
  - Assume all vertices are of the same size.
  - Work only 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 $g_1+...+g_k$ 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.



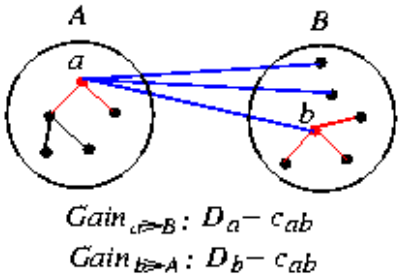
Step #	Vertex pair	Cost reduction	Cut cost
0	-	0	5
1	{d, g}	3	2
2	{c, f}	1	1
3	{b, h}	-2	3
4	{a, e}	-2	5

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

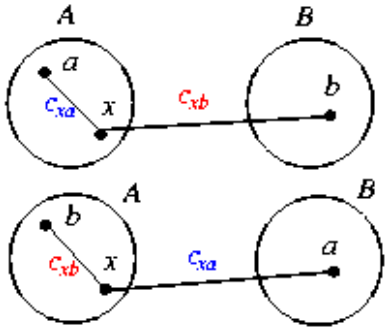
Properties

- 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

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



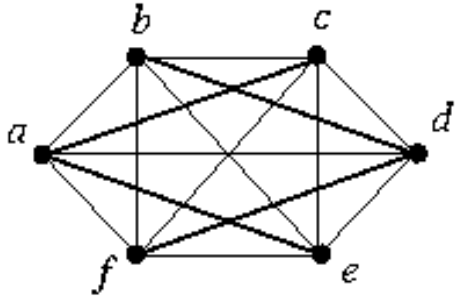
Internal cost vs. External cost



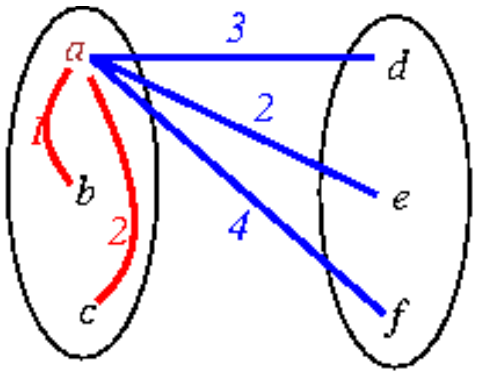
before swap	*after swap*	$\triangle C$
$-c_{xa}$	$+c_{xa}$	$+2c_{xa}$
$+c_{xb}$	$-c_{xb}$	$-2c_{xb}$

updating D–values

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

Initial cut cost = (3+2+4)+(4+2+1)+(3+2+1) = 22

- Iteration 1:

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

Weighted Example (Cont'd)

- Iteration 1:

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

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

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

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

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

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

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

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

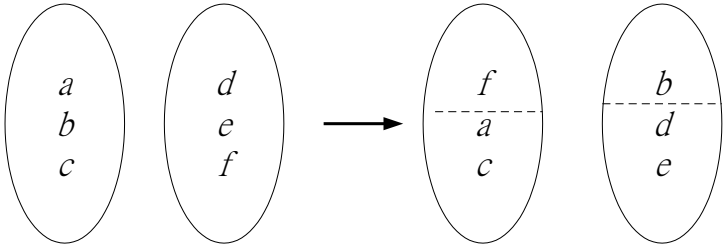
$g_{cd} = 3 + 3 - 2 * 3 = 0$

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

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

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

Weighted Example (Cont'd)



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

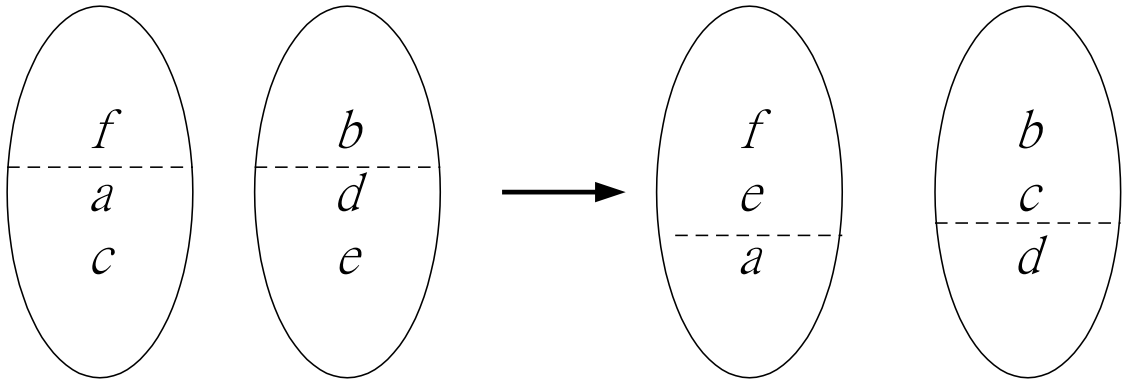
$$D_a^{'} = D_a + 2c_{ab} - 2c_{af} = 6 + 2*1 - 2*4 = 0$$
$$D_c^{'} = D_c + 2c_{cb} - 2c_{cf} = 3 + 2*1 - 2*1 = 3$$
$$D_d^{'} = D_d + 2c_{df} - 2c_{db} = 3 + 2*3 - 2*4 = 1$$
$$D_e^{'} = D_e + 2c_{ef} - 2c_{eb} = 0 + 2*2 - 2*2 = 0$$

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

$$g_{ad} = D_a^{'} + D_d^{'} - 2c_{ad} = 0 + 1 - 2*3 = -5$$
$$g_{ae} = D_a^{'} + D_e^{'} - 2c_{ae} = 0 + 0 - 2*2 = -4$$
$$g_{cd} = D_c^{'} + D_d^{'} - 2c_{cd} = 3 + 1 - 2*3 = -2$$
$$g_{ce} = D_c^{'} + D_e^{'} - 2c_{ce} = 3 + 0 - 2*2 = -1 \text{ (maximum)}$$

- Swap c and e! ($\tilde{g}_2 = -1$)

Weighted Example (Cont'd)



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

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

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

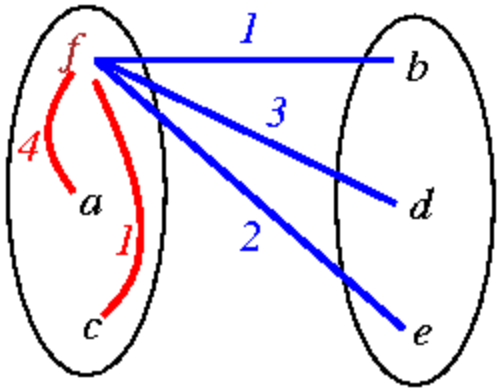
- $g_{xy} = D_x'' + D_y'' - 2x_{xy}.$

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

- Note that this step is redundant $\left(\sum_{i=1}^{n} \hat{g}_i = 0\right)$
- 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= $\max \sum_{i=1}^{k} \hat{g_i} = 0\ (k = 3) \Rightarrow$ 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 $v_{ai},\ldots,v_{ak}$ from V_A to V_B and $v_{bi},\ldots,v_{bk}$ 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 $O(n\text{-}i\text{+}1)^2$ 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, $O(n^3)$.