EE582
Physical Design Automation of VLSI Circuits and Systems

Prof. Dae Hyun Kim
School of Electrical Engineering and Computer Science
Washington State University

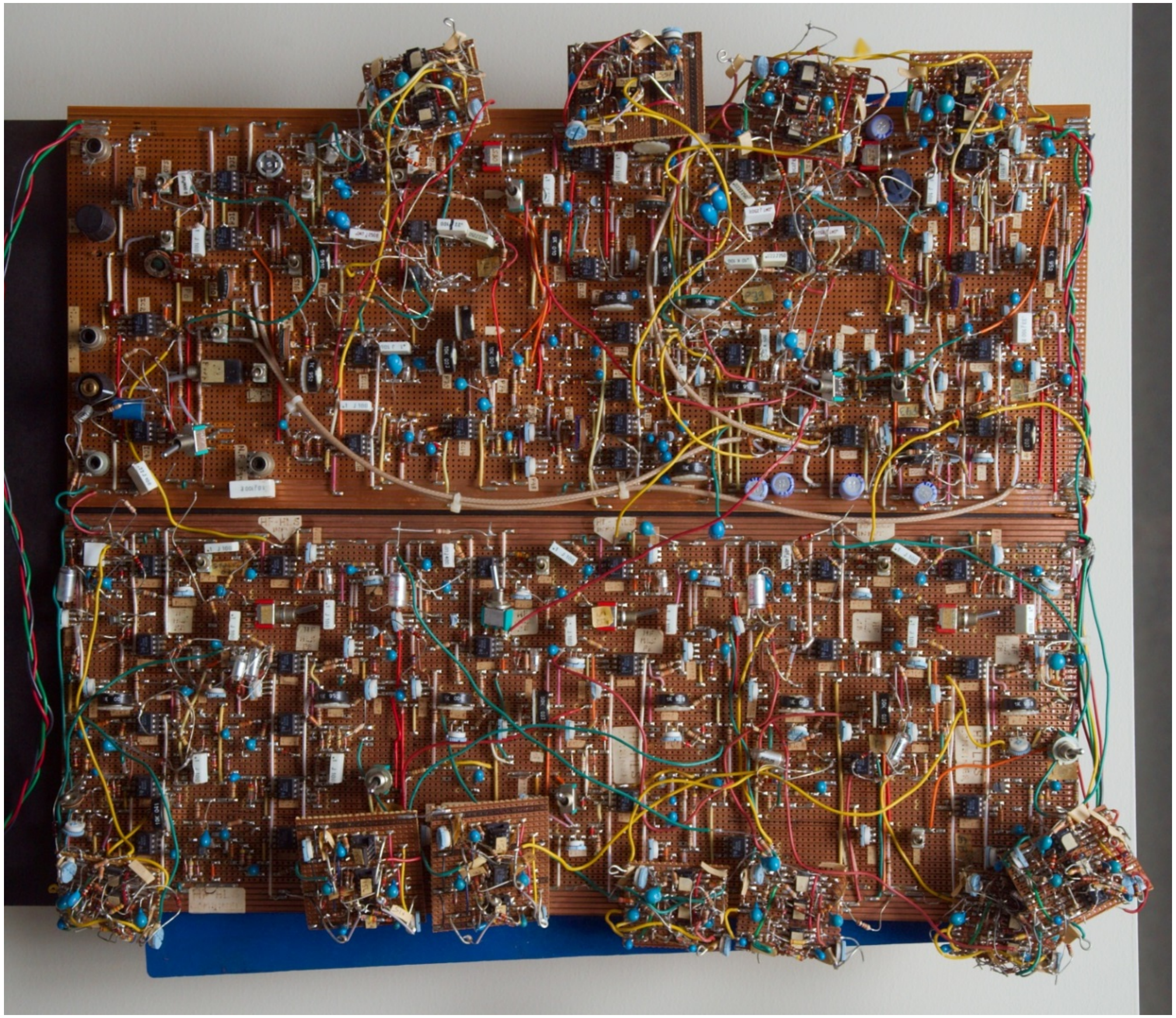
Partitioning

WASHINGTON STATE UNIVERSITY

What We Will Study

- Partitioning
  - Practical examples
  - Problem definition
  - Deterministic algorithms
    - Kernighan-Lin (KL)
    - Fiduccia-Mattheyses (FM)
    - h-Metis
  - Stochastic algorithms
    - Simulated-annealing

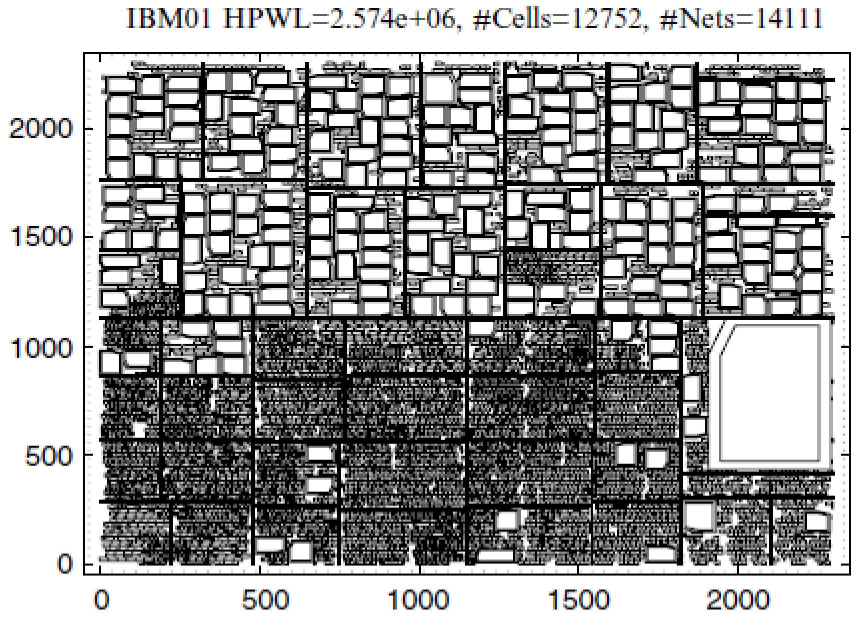
Example



Source: http://upload.wikimedia.org/wikipedia/commons/3/37/Dolby_SR_breadboard.jpg

Example

- ASIC Placement



Source: Nam, "Modern Circuit Placement"

VLSI Circuits

- # interconnections
  - Intra-module: many
  - Inter-module: few

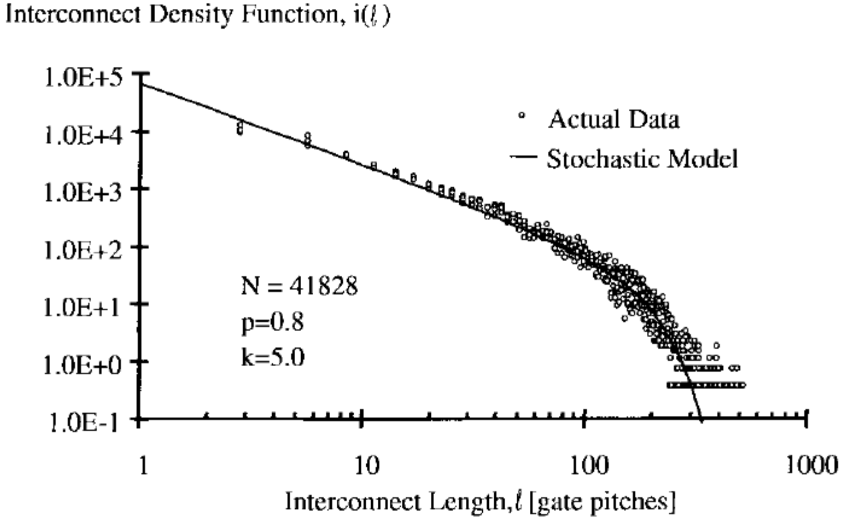


Fig. 9. Interconnect density function compared to the interconnect histogram for microprocessor *A*.

Source: David, "A Stochastic Wire-Length Distribution for Gigascale Integration (GSI) – Part I: Derivation and Validation," TCAD'98

Problem Definition

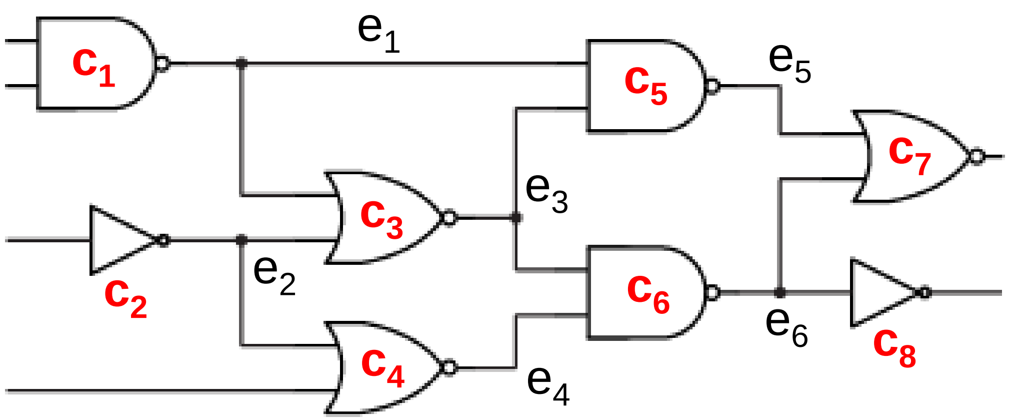
- Given
  - A set of cells: $T = \{c_1, c_2, \ldots, c_n\}$. $|W|=n$.
  - A set of edges (netlist): $R = \{e_1, e_2, \ldots, e_m\}$. $|R|=m$.
  - Cell size: $s(c_i)$
  - Edge weight: $w(e_j)$
  - # partitions: k (k-way partitioning). $P = \{P_1, \ldots, P_k\}$
  - Minimum partition size: $b \leq s(P_i)$
  - Balancing factor: $\max(s(P_i)) - \min(s(P_j)) \leq B$
  - Graph representation: edges / hyper-edges

- Find k partitions
  - $P = \{P_1, \ldots, P_k\}$

- Minimize
  - Cut size: $\sum_{\forall e(u_1,\ldots,ud) \in p(ui) \neq p(uj)} w(e)$

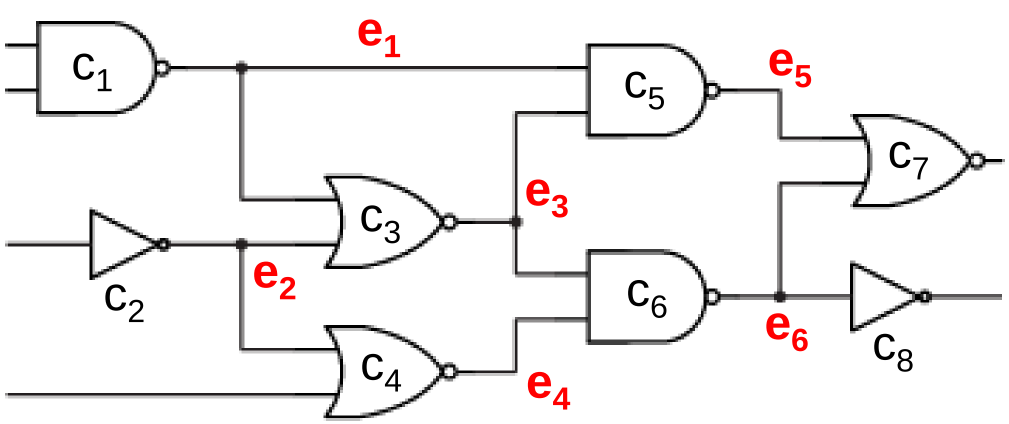
Problem Definition

- A set of cells
  - T = {c_1, c_2, …, c_n}. |W|=n



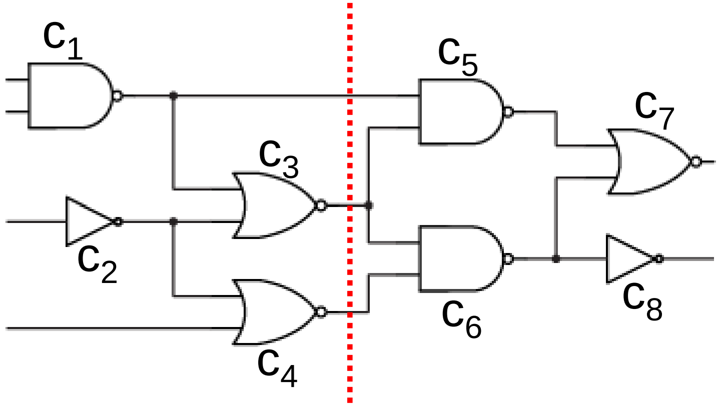
Problem Definition

- A set of edges (netlist, connectivity)
  - $R = \{e_1, e_2, \ldots, e_m\}$. $|R|=m$



k-way Partitioning

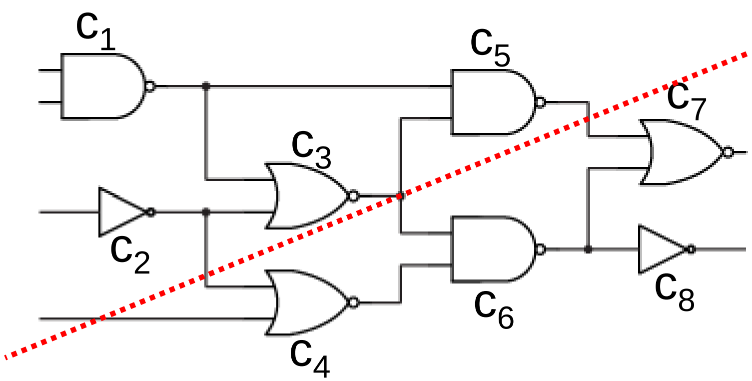
- k=2, $|P_i|$=4



$P_1 = \{c_1, c_2, c_3, c_4\}$

$P_2 = \{c_5, c_6, c_7, c_8\}$

Cut size = 3



$P_1 = \{c_1, c_2, c_3, c_5\}$

$P_2 = \{c_4, c_6, c_7, c_8\}$

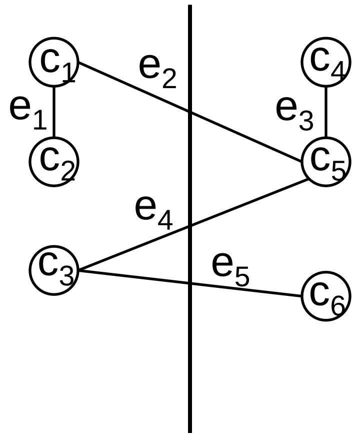
Cut size = 3

Kernighan-Lin (KL) Algorithm

- Problem definition
  - Given
    - A set of vertices (cell list): $V = \{c_1, c_2, \ldots, c_{2n}\}$. $|T|=2n$.
    - A set of two-pin edges (netlist): $E = \{e_1, e_2, \ldots, e_m\}$. $|E|=m$.
    - Weight of each edge: $w(e_j)$
    - Vertex size: $s(c_i) = 1$
  - Constraints
    - # partitions: 2 (two-way partitioning). $P = \{A, B\}$
    - Balanced partitioning: $|A| = |B| = n$
  - Minimize
    - Cutsize

Kernighan-Lin (KL) Algorithm

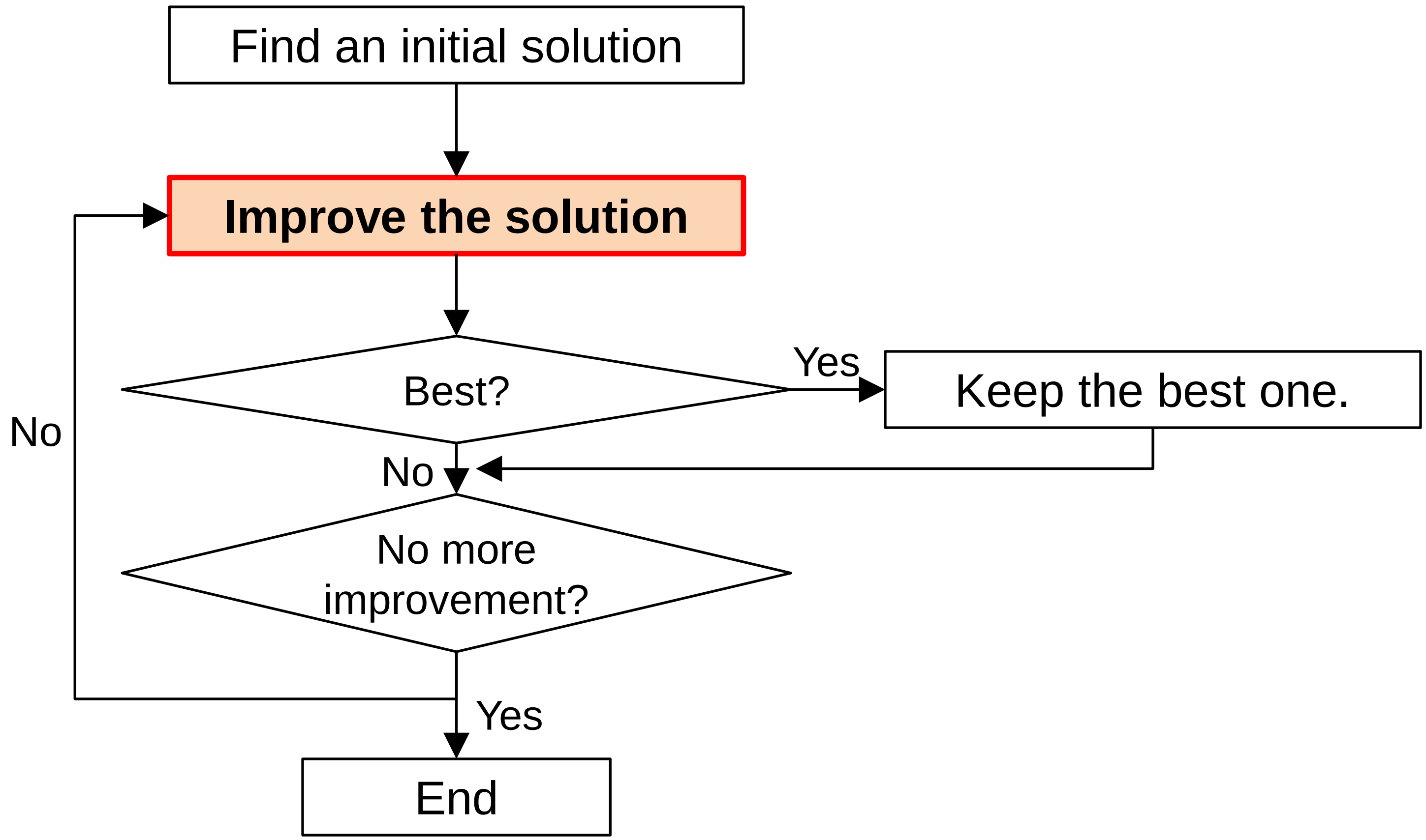
- Cost function: cutsize = $\sum_{e \in \psi} w(e)$
  - Ψ: cut set = $\{e_2, e_4, e_5\}$
  - Cutsize = $w(e_2) + w(e_4) + w(e_5)$



$A=\{c_1, c_2, c_3\}$ $B=\{c_4, c_5, c_6\}$

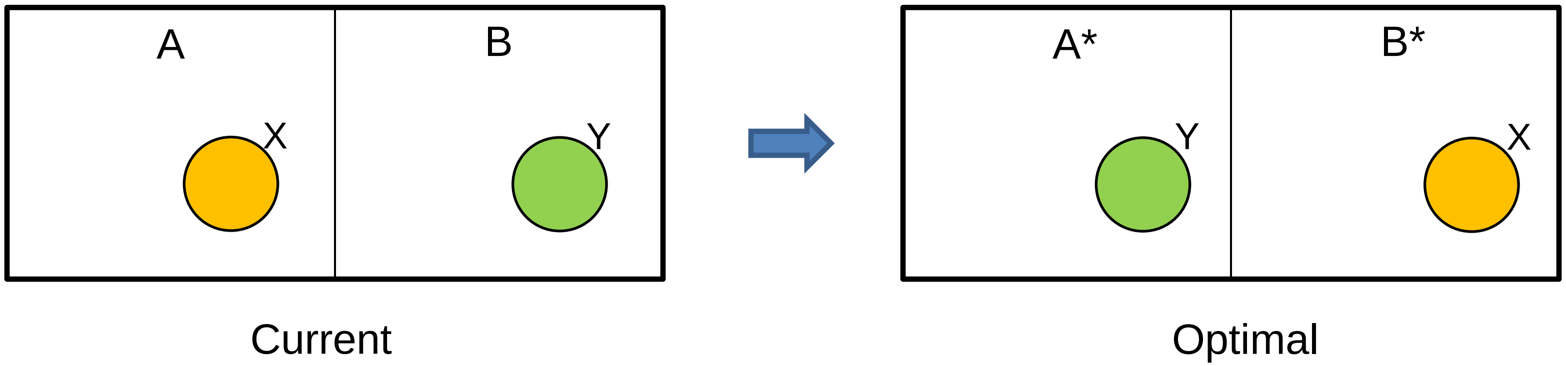
Kernighan-Lin (KL) Algorithm

- Algorithm



Kernighan-Lin (KL) Algorithm

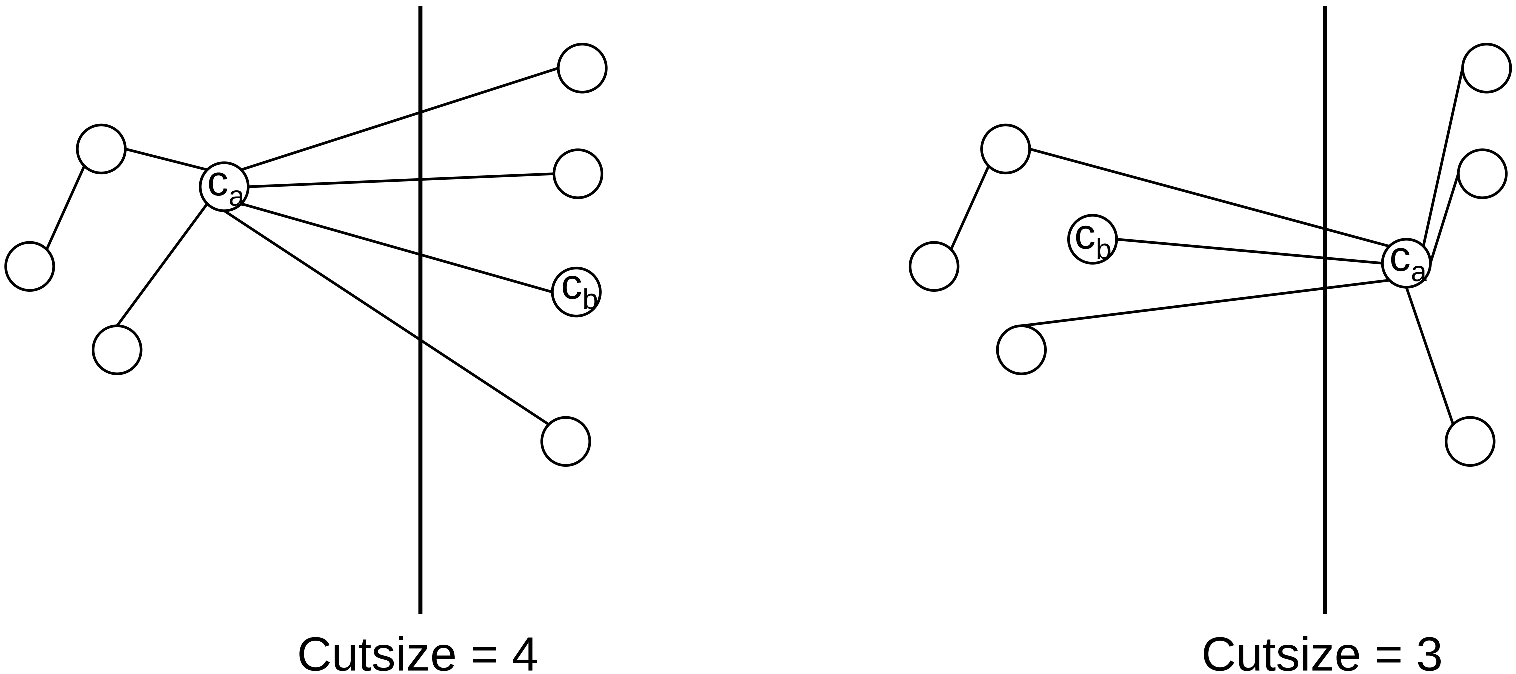
- Iterative improvement



How can we find X and Y?

Kernighan-Lin (KL) Algorithm

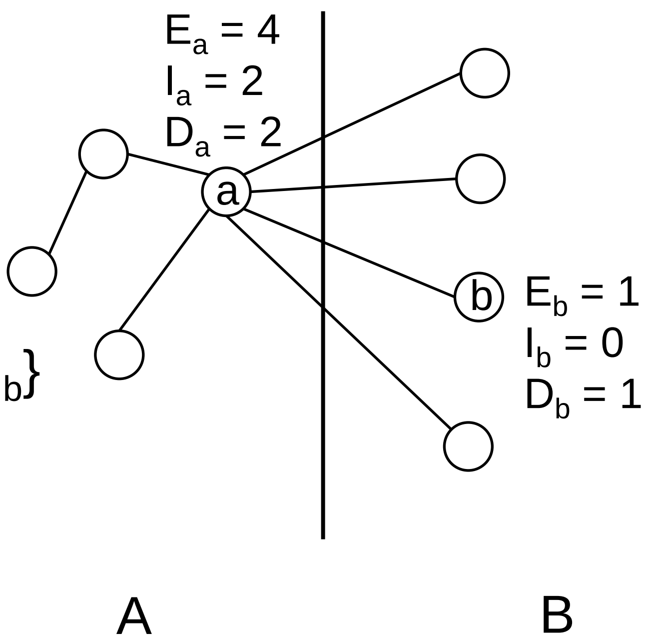
- Iterative improvement
  - Find a pair of vertices such that <u>swapping</u> the two vertices reduces the cutsize.



Kernighan-Lin (KL) Algorithm

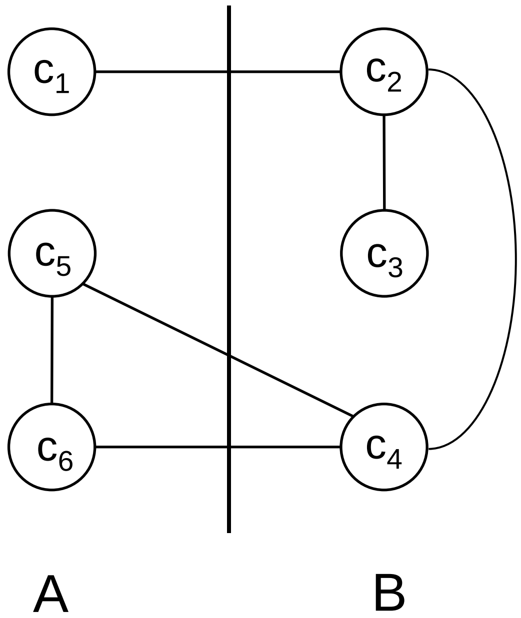
- Gain computation
  - For each cell a
    - External cost = $E_a = \sum w(eav)$ for all v∈B = $\sum w(\text{external edges})$
    - Internal cost = $I_a = \sum w(eav)$ for all v∈A = $\sum w(\text{internal edges})$
    - D-value (a) = $D_a = E_a - I_a$
  - For each pair (a, b)
    - Gain = $g_{ab} = D_a + D_b - 2w(e_{ab})$
  - $g_{ab} = \{(E_a - w(e_{ab})) - I_a\} + \{(E_b - w(e_{ab})) - I_b\}$

    $= D_a + D_b - 2w(e_{ab})$



Kernighan-Lin (KL) Algorithm

- Find a best-gain pair among all the gate pairs



Cutsize = 3

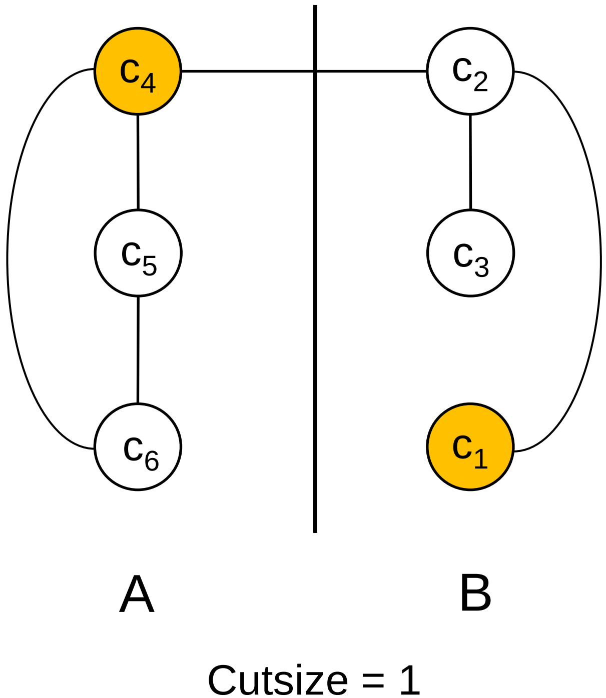
	E_a	I_a	D_a
c_1	1	0	1
c_2	1	2	-1
c_3	0	1	-1
c_4	2	1	1
c_5	1	1	0
c_6	1	1	0

$g_{12} = 1 – 1 – 2 = -2$
$g_{13} = 1 – 1 – 0 = 0$
$g_{14} = 1 + 1 – 0 = +2$
$g_{52} = 0 – 1 – 0 = -1$
$g_{53} = 0 – 1 – 0 = -1$
$g_{54} = 0 + 1 – 2 = -1$
$g_{62} = 0 – 1 – 0 = -1$
$g_{63} = 0 – 1 – 0 = -1$
$g_{64} = 0 + 1 – 2 = -1$

$g_{ab} = D_a + D_b – 2w(e_{ab})$

Kernighan-Lin (KL) Algorithm

- Swap and Lock
  - After swapping, we lock the swapped cells. The locked cells will not be moved further.

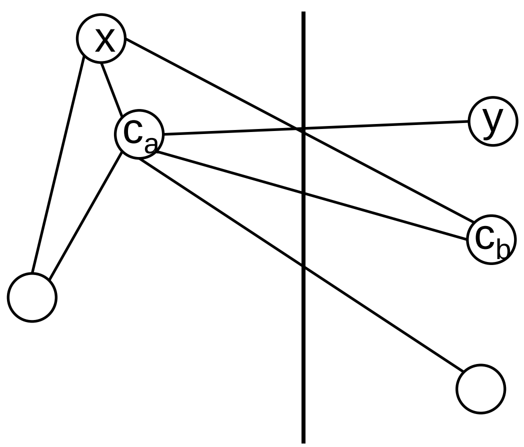


	E_a	I_a	D_a
c_1	1	0	1
c_2	1	2	-1
c_3	0	1	-1
c_4	2	1	1
c_5	1	1	0
c_6	1	1	0

Need update!

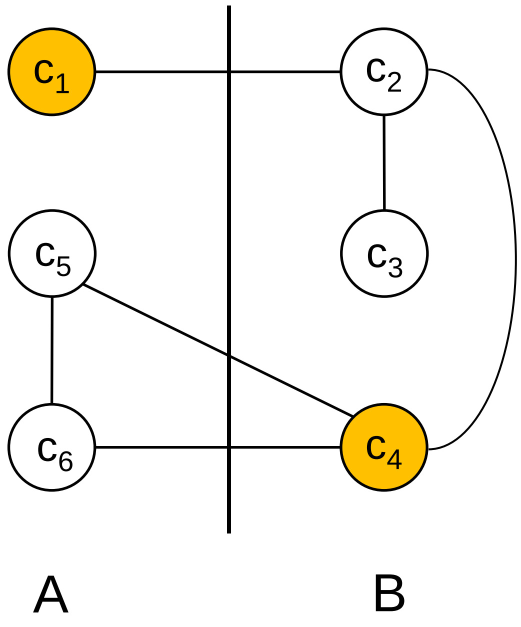
Kernighan-Lin (KL) Algorithm

- Update of the D-value
  - Update the D-value of the cells affected by the move.
    - $D_x' = E_x' - I_x' = \{E_x - w(e_{xb}) + w(e_{xa})\} - \{I_x + w(e_{xb}) - w(e_{xa})\}$
      $= (E_x - I_x) + 2w(e_{xa}) - 2w(e_{xb}) = D_x + 2w(e_{xa}) - 2w(e_{xb})$
    - $D_y' = (E_y - I_y) + 2w(e_{yb}) - 2w(e_{ya}) = D_y + 2w(e_{yb}) - 2w(e_{ya})$



Kernighan-Lin (KL) Algorithm

- Update

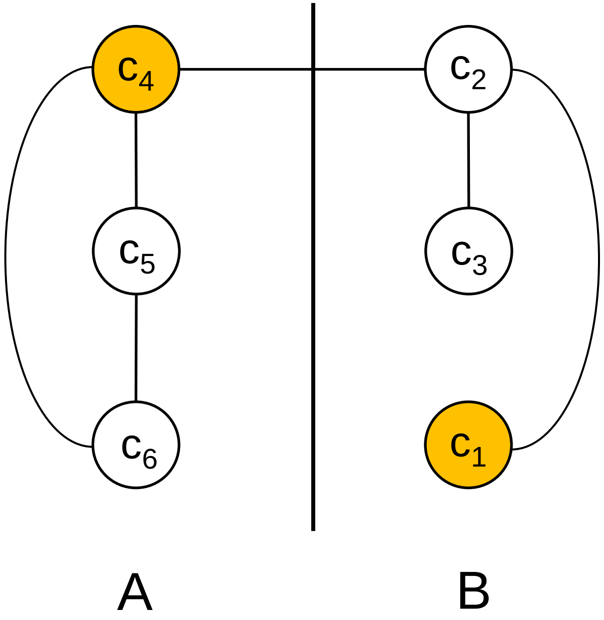


	D_a	D_a'
c_1	1	
c_2	-1	-1 + 2 – 2 = -1
c_3	-1	-1
c_4	1	
c_5	0	0 + 0 – 2 = -2
c_6	0	0 + 0 – 2 = -2

$D_x' = D_x + 2*w(e_{xa}) – 2*w(e_{xb})$
$D_y' = D_y + 2*w(e_{yb}) – 2*w(e_{ya})$

Kernighan-Lin (KL) Algorithm

- Gain computation and pair selection



	D_a'
c_1	
c_2	-1
c_3	-1
c_4	
c_5	-2
c_6	-2

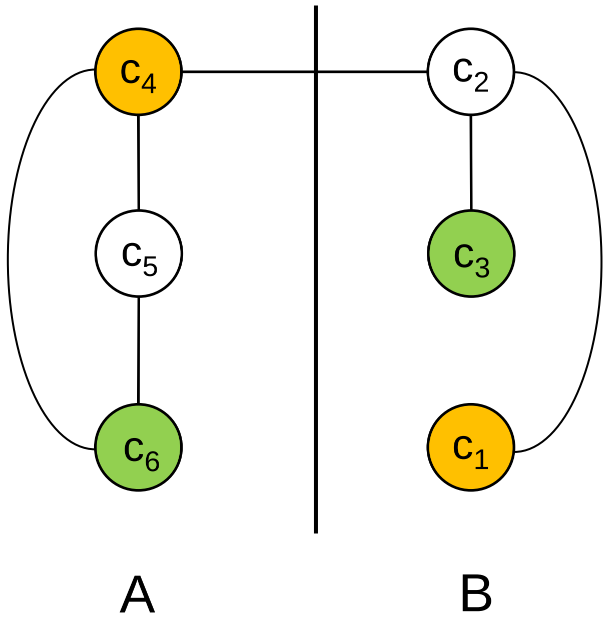
$g_{52} = -2 - 1 - 0 = -3$
$g_{53} = -2 - 1 - 0 = -3$
$g_{62} = -2 - 1 - 0 = -3$
$g_{63} = -2 - 1 - 0 = -3$

$g_{ab} = D_a + D_b - 2w(e_{ab})$

Cutsize = 1

Kernighan-Lin (KL) Algorithm

- Swap and update



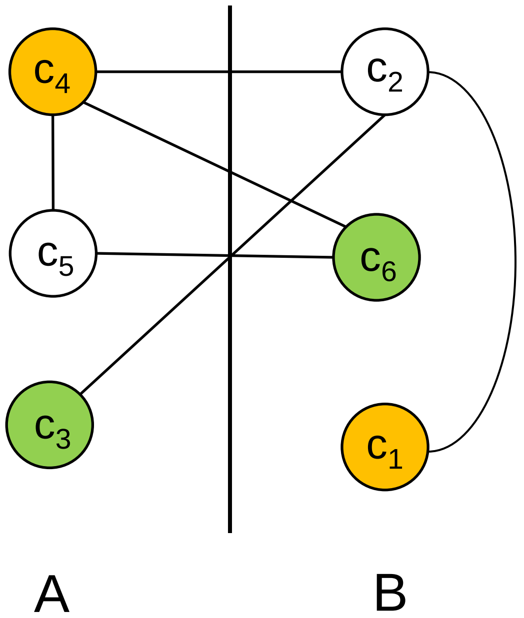
	D_a	D_a'
c_1		
c_2	-1	-1 + 2 – 0 = +1
c_3	-1	
c_4		
c_5	-2	-2 + 2 – 0 = 0
c_6	-2	

Cutsize = 1

$D_x' = D_x + 2*w(e_{xa}) - 2*w(e_{xb})$
$D_y' = D_y + 2*w(e_{yb}) - 2*w(e_{ya})$

Kernighan-Lin (KL) Algorithm

- Swap and update



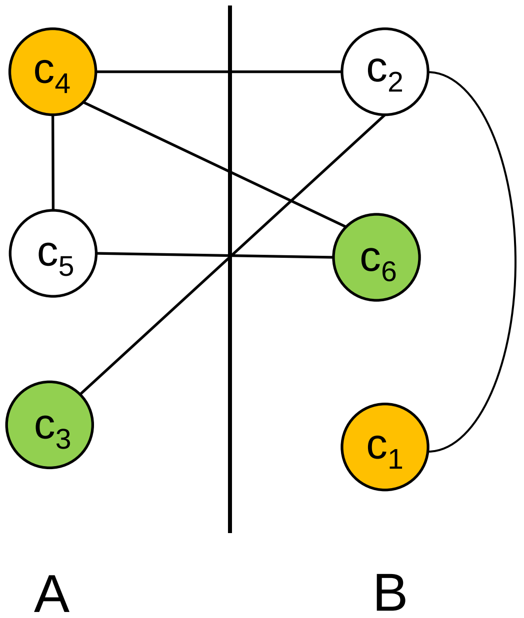
	D_a
c_1	
c_2	+1
c_3	
c_4	
c_5	0
c_6	

Cutsize = 4

$D_x' = D_x + 2*w(e_{xa}) – 2*w(e_{xb})$
$D_y' = D_y + 2*w(e_{yb}) – 2*w(e_{ya})$

Kernighan-Lin (KL) Algorithm

- Gain computation



Cutsize = 4

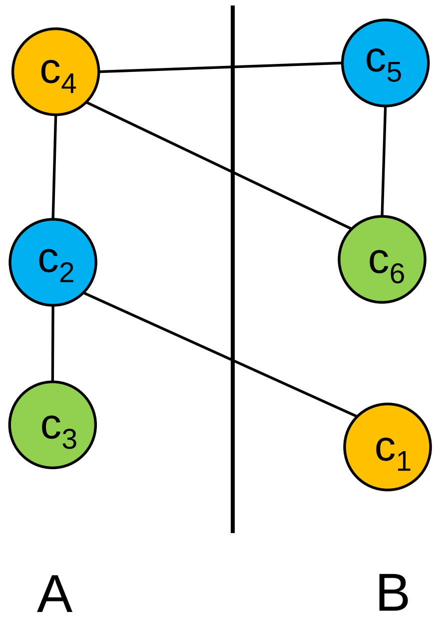
	D_a
c_1	
c_2	+1
c_3	
c_4	
c_5	0
c_6	

g_{52} = +1 + 0 – 0 = +1

$g_{ab} = D_a + D_b - 2w(e_{ab})$

Kernighan-Lin (KL) Algorithm

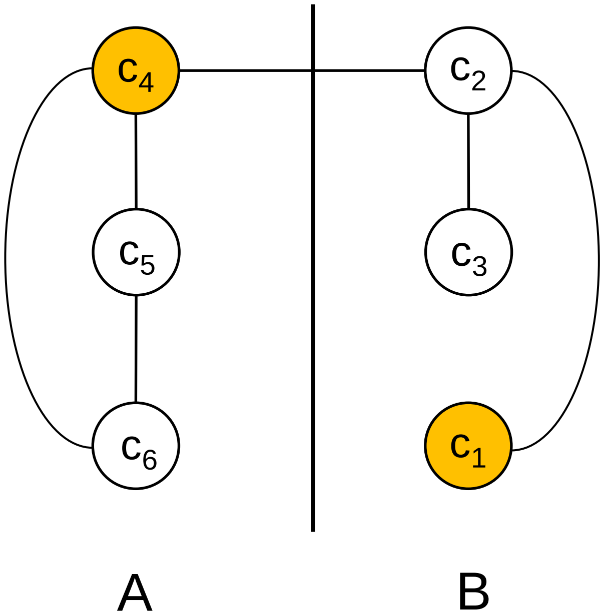
- Swap



Cutsize = 3

Kernighan-Lin (KL) Algorithm

- Cutsize
  - Initial: 3
    - $g_1 = +2$
  - After 1st swap: 1
    - $g_2 = -3$
  - After 2nd swap: 4
    - $g_3 = +1$
  - After 3rd swap: 3



Cutsize = 1

Kernighan-Lin (KL) Algorithm

- Algorithm (a single iteration)
  1. $V = \{c_1, c_2, \ldots, c_{2n}\}$
     {A, B}: initial partition
  2. Compute D_v for all $v \in V$
     queue = {}, i = 1, A'=A, B'=B
  3. Compute gain and choose the best-gain pair (a_i, b_i).
     queue += (a_i, b_i), A' = A'-$\{a_i\}$, B'=B'-$\{b_i\}$
  4. If A' and B' are empty, go to step 5.
     Otherwise, update D for A' and B' and go to step 3.
  5. Find k maximizing $G=\sum_{i=1}^{k} g_i$

Kernighan-Lin (KL) Algorithm

- Algorithm (overall)
  1. Run a single iteration.
  2. Get the best partitioning result in the iteration.
  3. <u>Unlock</u> all the cells.
  4. Re-start the iteration. Use the best partitioning result for the initial partitioning of the next iteration.

- Stop criteria
  - Max. # iterations
  - Max. runtime
  - Δ Cutsize between the two consecutive iterations.
  - Target cutsize
  - …

Kernighan-Lin (KL) Algorithm

- Complexity analysis
  1. $V = \{c_1, c_2, \ldots, c_{2n}\}$
     {A, B}: initial partition
  2. Compute D_v for all $v \in V$
     queue = {}, i = 1, A'=A, B'=B
  3. Compute gain and choose the best-gain pair (a_i, b_i).
     queue += (a_i, b_i), A' = A'-$\{a_i\}$, B'=B'-$\{b_i\}$
  4. If A' and B' are empty, go to step 5.
     Otherwise, update D for A' and B' and go to step 3.
  5. Find k maximizing $G=\sum_{i=1}^{k} g_i$

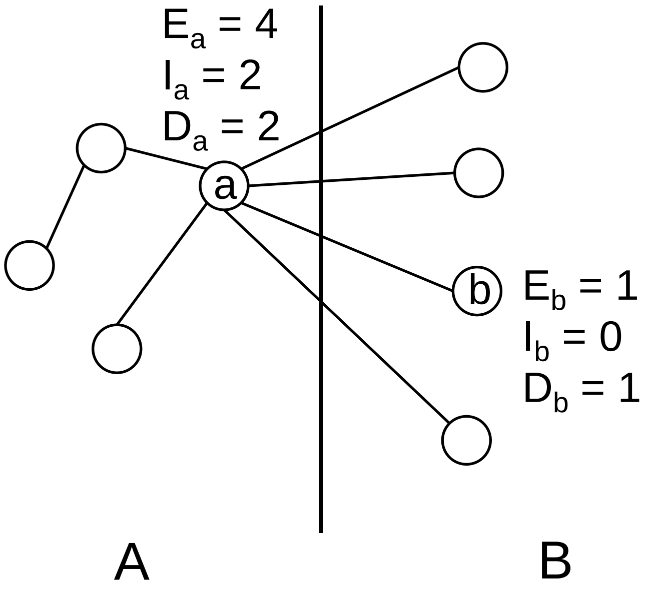
Kernighan-Lin (KL) Algorithm

- Complexity of the D-value computation
  - External cost (a) = $E_a = \sum w(e_{av})$ for all v∈B
  - Internal cost (a) = $I_a = \sum w(e_{av})$ for all v∈A
  - D-value (a) = $D_a = E_a - I_a$

For each cell (node) a
  For each net connected to cell a
    Compute E_a and I_a

Practically O(n)

n: # nodes



Kernighan-Lin (KL) Algorithm

- Complexity of the gain computation
  - $g_{ab} = D_a + D_b - 2w(e_{ab})$

For each pair (a, b)
$g_{ab} = D_a + D_b - 2w(e_{ab})$

Complexity: $O((2n - 2i + 2)^2)$

Kernighan-Lin (KL) Algorithm

- Complexity of the D-value update for swapping a and b
  - $D_x' = D_x + 2w(e_{xa}) - 2w(e_{xb})$
  - $D_y' = D_y + 2w(e_{yb}) - 2w(e_{ya})$

For a (and b)
  For each cell x connected to cell a (and b)
    Update D_x and D_y

Practically O(1)

Kernighan-Lin (KL) Algorithm

- Complexity analysis
  1. $V = \{c_1, c_2, \ldots, c_{2n}\}$
     $\{A, B\}$: initial partition
  2. Compute D_v for all $v \in V$
     queue = {}, i = 1, A'=A, B'=B → O(n)

Loop. # iterations: n

  3. Compute gain and choose the best-gain pair (a_i, b_i).
     queue += (a_i, b_i), A' = A'-$\{a_i\}$, B'=B'-$\{b_i\}$ → $O((2n-2i+2)^2)$
  4. If A' and B' are empty, go to step 5.
     Otherwise, update D for A' and B' and go to step 3.
  5. Find k maximizing $G=\sum_{i=1}^{k} g_i$ → O(1)

Overall: $O(n^3)$

Kernighan-Lin (KL) Algorithm

- Reduce the runtime
  - The most expensive step: gain computation ($O(n^2)$)
    - Compute the gain of each pair: $g_{ab} = D_a + D_b - 2w(e_{ab})$
  - How to expedite the process
    - Sort the cells in the decreasing order of the D-value
      - $D_{a1} \geq D_{a2} \geq D_{a3} \geq \ldots$ in partition A
      - $D_{b1} \geq D_{b2} \geq D_{b3} \geq \ldots$ in partition B
    - Keep the max. gain (g_{max}) found until now.
    - When computing the gain of (D_{al}, D_{bm})
      - If $D_{al} + D_{bm} < g_{max}$, we don't need to compute the gain for all the pairs (D_{ak}, D_{bp}) s.t. k>l and p>m.
      - Practically, it takes O(1).
  - Complexity: O(n*logn) for sorting.

Kernighan-Lin (KL) Algorithm

- Complexity analysis
  1. $V = \{c_1, c_2, \ldots, c_{2n}\}$
     {A, B}: initial partition
  2. Compute D_v for all $v \in V$
     queue = {}, i = 1, A'=A, B'=B → O(n)

Loop. # iterations: n

  3. Compute gain and choose the best-gain pair (a_i, b_i).
     queue += (a_i, b_i), A' = A'-$\{a_i\}$, B'=B'-$\{b_i\}$ → O(n log n)
  4. If A' and B' are empty, go to step 5.
     Otherwise, update D for A' and B' and go to step 3.
  5. Find k maximizing $G=\sum_{i=1}^{k} g_i$ → O(1)

Overall: $O(n^2 \log n)$

Questions

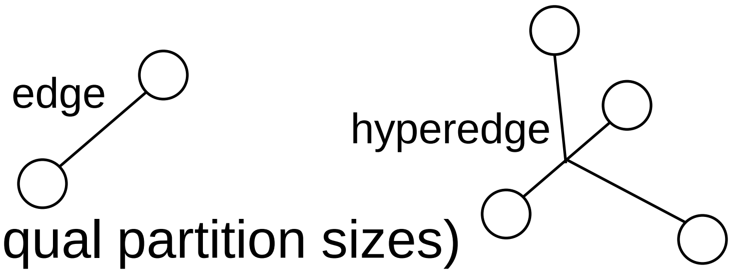
- Can the KL algorithm handle
  - hyperedges?
  - unbalanced partitions?
  - fixed cells (e.g., cell k should be in partition A)?
  - …

Questions

- Intentionally left blank

Fiduccia-Mattheyses (FM) Algorithm

- Handles
  - Hyperedges

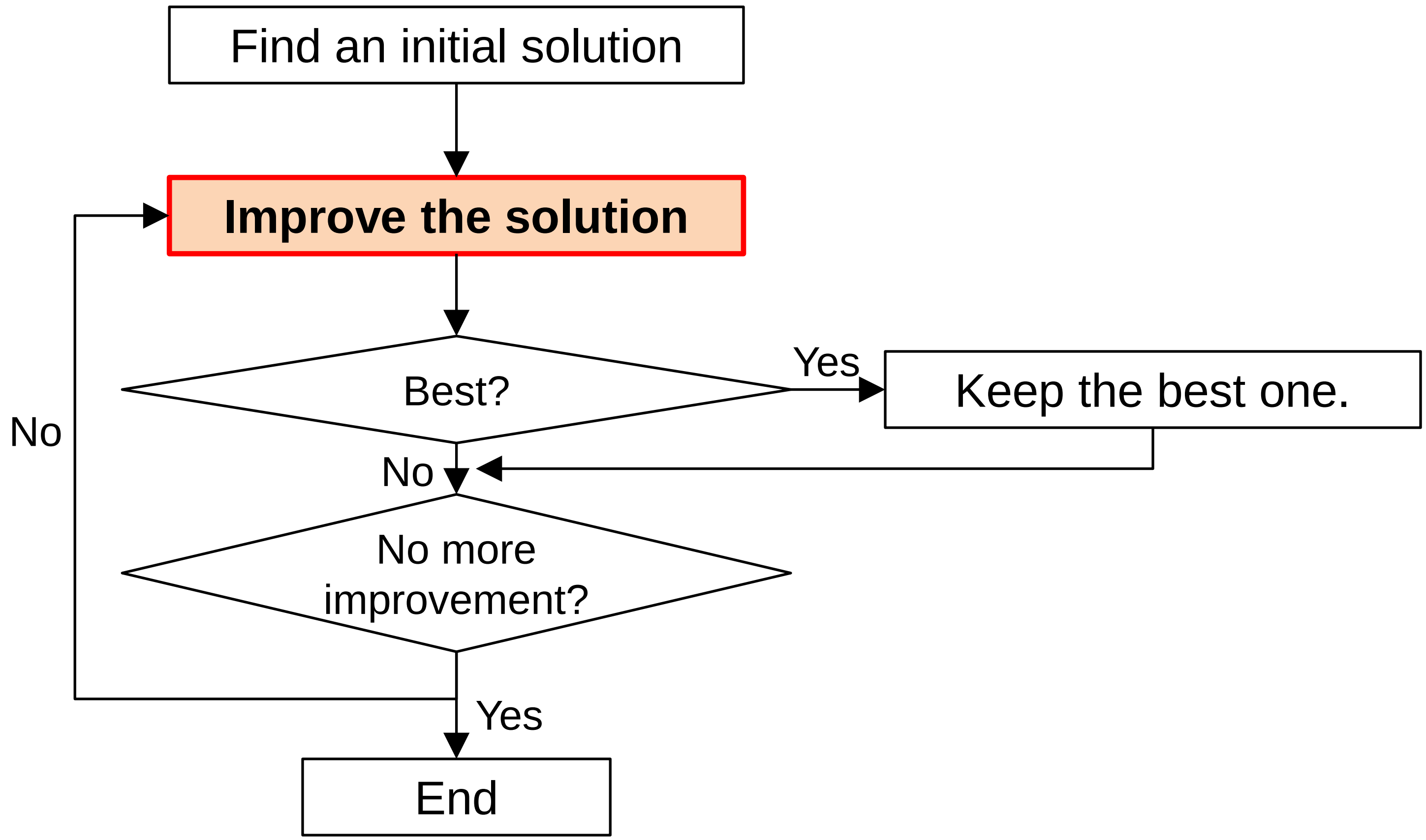


  - Imbalance (unequal partition sizes)
  - Runtime: O(n)

- Idea
  - Move a cell instead of swapping two cells.

Fiduccia-Mattheyses (FM) Algorithm

- Algorithm

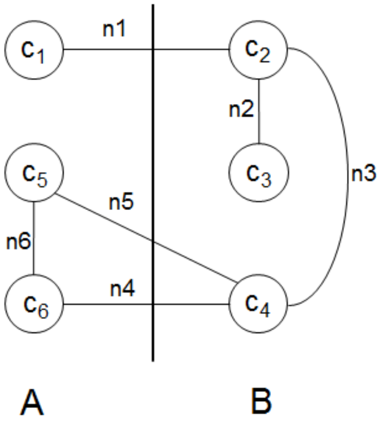


Fiduccia-Mattheyses (FM) Algorithm

- Definitions
  - Cutstate(net)
    - uncut: the net has all the cells in a single partition.
    - cut: the net has cells in both the two partitions.
  - Gain of cell: # nets by which the cutsize will decrease if the cell were to be moved.
  - Balance criterion: To avoid having all cells migrate to one block.
    - $r \cdot |V| - s_{max} \leq |A| \leq r \cdot |V| + s_{max}$ Max cell size
    - $|A| + |B| = |V|$
  - Base cell: The cell selected for movement.
    - The max-gain cell that doesn't violate the balance criterion.

Fiduccia-Mattheyses (FM) Algorithm

- Definitions (continued)
  - Distribution (net): (A(n), B(n))
    - A(n): # cells connected to n in A
    - B(n): # cells connected to n in B

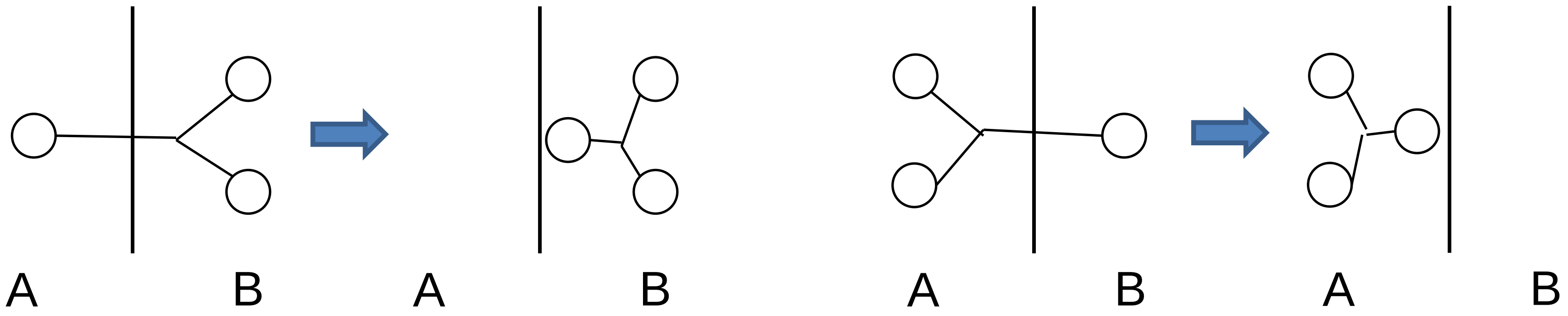


Distribution (n1) = (1, 1)
Distribution (n2) = (0, 1)
Distribution (n3) = (0, 2)
Distribution (n4) = (2, 1)
Distribution (n5) = (1, 1)
Distribution (n6) = (2, 0)

  - Critical net
    - A net is critical if it has a cell that if moved will change its cutstate.
      - cut to uncut
      - uncut to cut
    - The distribution of the net is (0,x), (1,x), (x,0), or (x,1).

Fiducciia-Mattheyses (FM) Algorithm

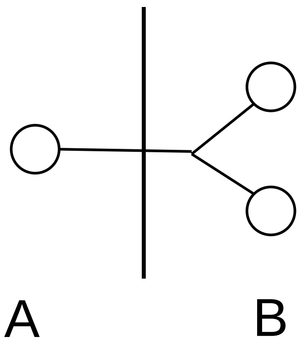
- Critical net
  - Moving a cell connected to the net changes the cutstate of the net.



Fiducccia-Mattheyses (FM) Algorithm

- Algorithm
  1. Gain computation
     - Compute the gain of each cell move
  2. Select a base cell (a max-gain cell)
  3. Move and lock the base cell and update gain.

Fiduccia-Mattheyses (FM) Algorithm



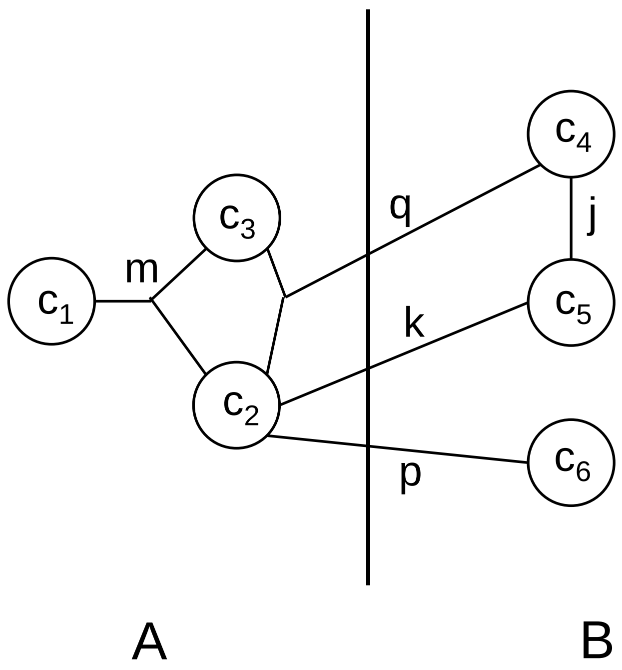
- Gain computation
  - F(c): From_block (either A or B)
  - T(c): To_block (either B or A)
  - gain = g(c) = FS(c) – TE(c)
    - FS(c): |P| s.t. P = {n | c ∈ n and dis(n) = (F(c), T(c)) = (1, x)}
    - TE(c): |P| s.t. P = {n | c ∈ n and dis(n) = (F(c), T(c)) = (x, 0)}

Fiduccia-Mattheyses (FM) Algorithm

- Gain computation

gain = g(c) = FS(c) – TE(c)
FS(c): # nets whose dis. = (F(c), T(c)) = (1, x)}
TE(c): # nets whose dis. = (F(c), T(c)) = (x, 0)}

For each net n connected to c
  if F(n) = 1, g(c)++;
  if T(n) = 0, g(c)--;



Distribution of the nets (A, B)

m: (3, 0)
q: (2, 1)
k: (1, 1)
p: (1, 1)
j: (0, 2)

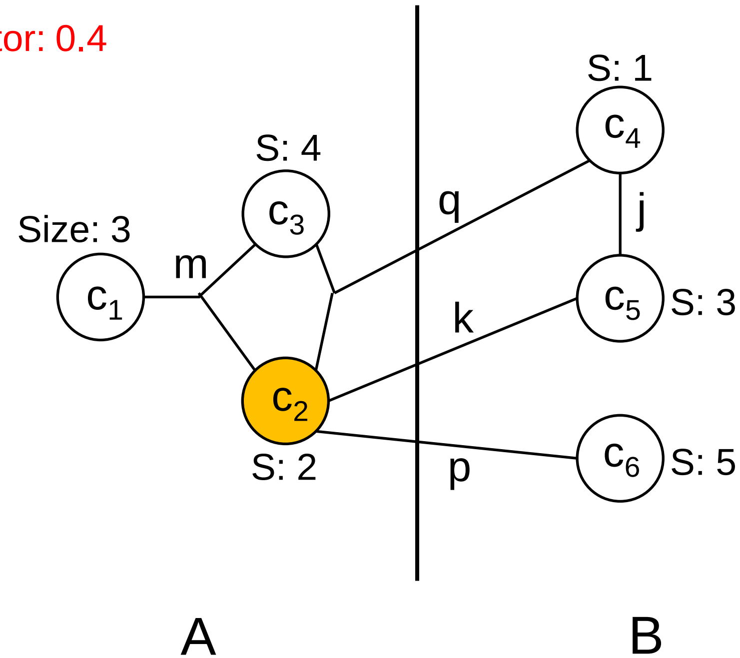
Gain

$g(c_1) = 0 – 1 = -1$
$g(c_2) = 2 – 1 = +1$
$g(c_3) = 0 – 1 = -1$
$g(c_4) = 1 – 1 = 0$
$g(c_5) = 1 – 1 = 0$
$g(c_6) = 1 – 0 = +1$

Fiduccia-Mattheyses (FM) Algorithm

- Select a base (best-gain) cell.

Balance factor: 0.4



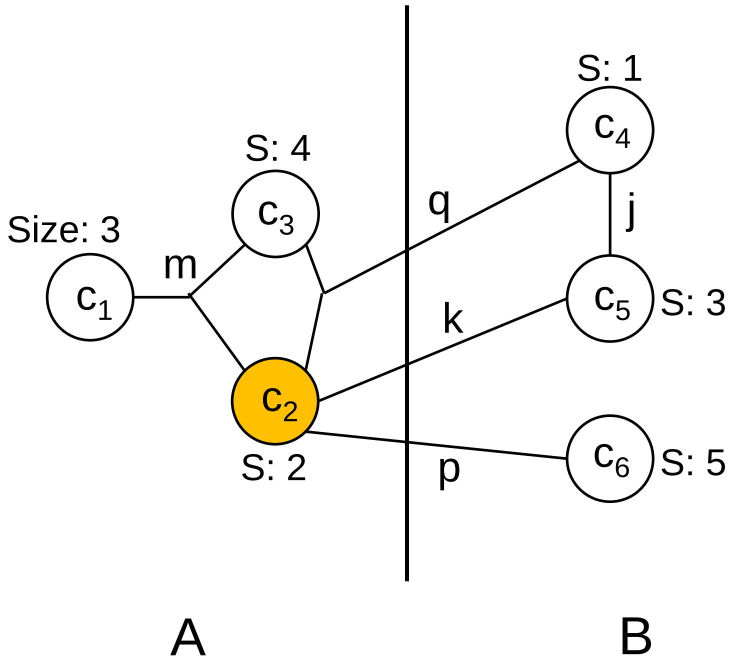
Gain

$g(c_1) = 0 – 1 = -1$
$g(c_2) = 2 – 1 = +1$
$g(c_3) = 0 – 1 = -1$
$g(c_4) = 1 – 1 = 0$
$g(c_5) = 1 – 1 = 0$
$g(c_6) = 1 – 0 = +1$

S(A) = 9, S(B) = 9
Area criterion: [0.4*18-5, 0.4*18+5] = [2.2, 12.2]

Fiduccia-Mattheyses (FM) Algorithm

- Before move, update the gain of all the other cells.



Fiduccia-Mattheyses (FM) Algorithm

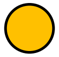
- Original code for gain update
  - F: From_block of the base cell
  - T: To_block of the base cell
  - For each net n connected to the base cell
    - If T(n) = 0
      - gain(c)++; // for c ∈ n
    - Else if T(n) = 1
      - gain(c)--; // for c ∈ n & T
    - F(n)--;
    - T(n)++;
    - If F(n) = 0
      - gain(c)--; // for c ∈ n
    - Else if F(n) = 1
      - gain(c)++; // for c ∈ n & F

Fiduccia-Mattheyses (FM) Algorithm

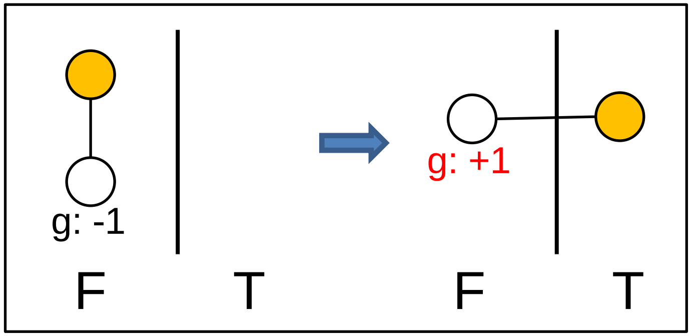
- Gain update (reformulated)
  - F: From_block of the base cell
  - T: To_block of the base cell
  - For each net n connected to the base cell
    - If T(n) = 0
      - gain(c)++; // for c ∈ n
    - Else if T(n) = 1
      - gain(c)--; // for c ∈ n & T
    - If F(n) = 1
      - gain(c)--; // for c ∈ n
    - Else if F(n) = 2
      - gain(c)++; // for c ∈ n & F
    - F(n)--;
    - T(n)++;

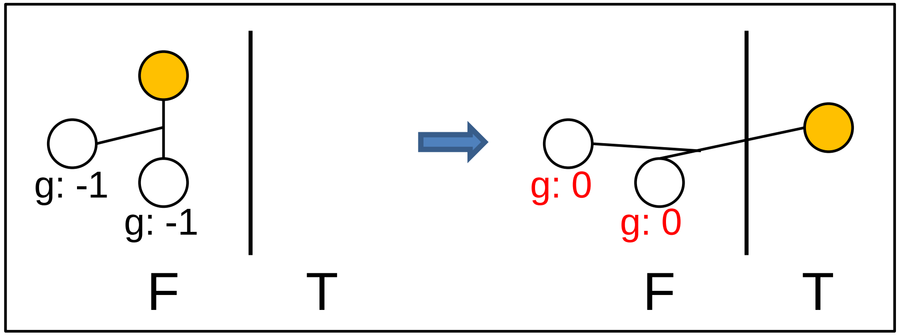
Fiduccia-Mattheyses (FM) Algorithm

- Gain update



Case 1) T(n) = 0



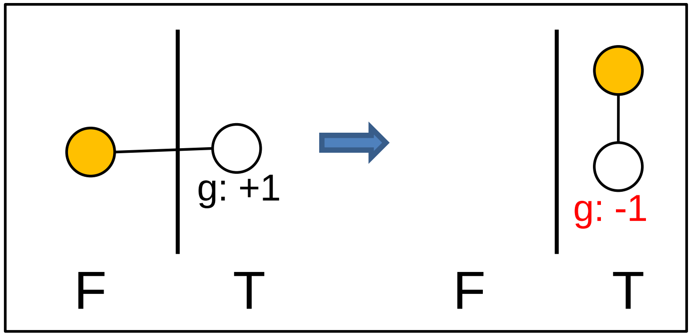


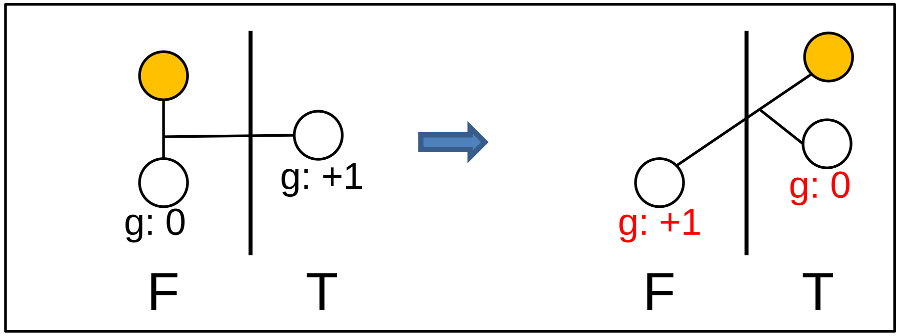
Fiduccia-Mattheyses (FM) Algorithm

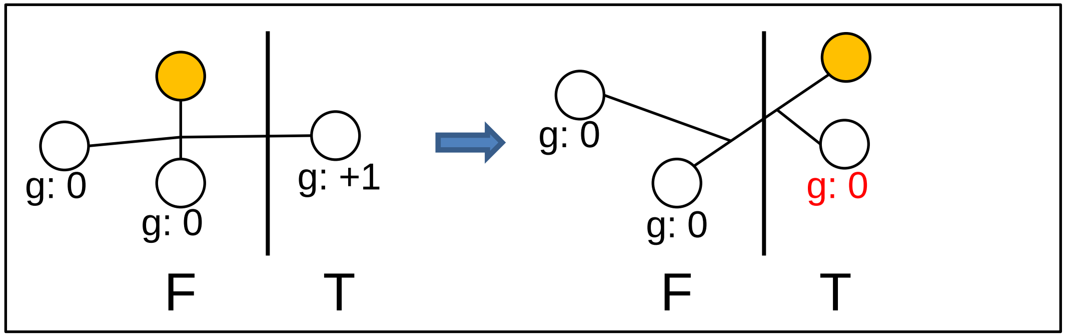
- Gain update

base cell

Case 2) T(n) = 1

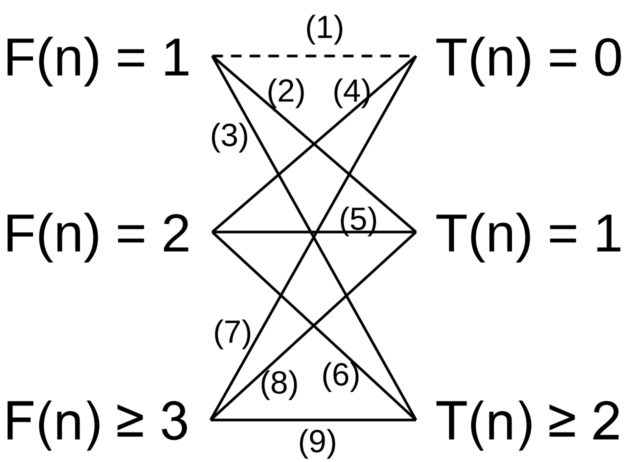






Fiduccia-Mattheyses (FM) Algorithm

- Instead of enumerating all the cases, we consider the following combinations.



	Δg(c) in F	Δg(c) in T		F(n)	T(n)
(1)					
(2)		-2		1	1
(3)		-1		1	≥ 2
(4)	+2			2	0
(5)	+1	-1		2	1
(6)	+1			2	≥ 2
(7)	+1			≥ 3	0
(8)		-1		≥ 3	1
(9)					

Fiduccia-Mattheyses (FM) Algorithm

- Conversion from the table to a source code.

	Δg(c) in F	Δg(c) in T		F(n)	T(n)
(1)					
(2)		-2		1	1
(3)		-1		1	≥ 2
(4)	+2			2	0
(5)	+1	-1		2	1
(6)	+1			2	≥ 2
(7)	+1			≥ 3	0
(8)		-1		≥ 3	1
(9)					

Fiduccia-Mattheyses (FM) Algorithm

- Conversion from the table to a source code.

	Δg(c) in F	Δg(c) in T		F(n)	T(n)
(1)					
(2)		-2		1	1
(3)		-1		1	≥ 2
(4)	$\boxed{+2}$			2	0
(5)	+1	-1		2	1
(6)	+1			2	≥ 2
(7)	$\boxed{+1}$			≥ 3	0
(8)		-1		≥ 3	1
(9)					

T(n) = 0 : g(c)++; // c ∈ n & F

Fiduccia-Mattheyses (FM) Algorithm

- Conversion from the table to a source code.

	Δg(c) in F	Δg(c) in T		F(n)	T(n)
(1)					
(2)		$\boxed{-2}$		1	1
(3)		-1		1	≥ 2
(4)	$\boxed{+2}$			2	0
(5)	+1	$\boxed{-1}$		2	1
(6)	+1			2	≥ 2
(7)	$\boxed{+1}$			≥ 3	0
(8)		$\boxed{-1}$		≥ 3	1
(9)					

T(n) = 1 : g(c)--; // c ∈ n & T

Fiduccia-Mattheyses (FM) Algorithm

- Conversion from the table to a source code.

	Δg(c) in F	Δg(c) in T		F(n)	T(n)
(1)					
(2)		-2		1	1
(3)		-1		1	≥ 2
(4)	+2			2	0
(5)	+1	-1		2	1
(6)	+1			2	≥ 2
(7)	+1			≥ 3	0
(8)		-1		≥ 3	1
(9)					

F(n) = 1 : g(c)--; // c ∈ n & T

Fiduccia-Mattheyses (FM) Algorithm

- Conversion from the table to a source code.

	Δg(c) in F	Δg(c) in T		F(n)	T(n)
(1)					
(2)		-2		1	1
(3)		-1		1	≥ 2
(4)	+2			2	0
(5)	+1	-1		2	1
(6)	+1			2	≥ 2
(7)	+1			≥ 3	0
(8)		-1		≥ 3	1
(9)					

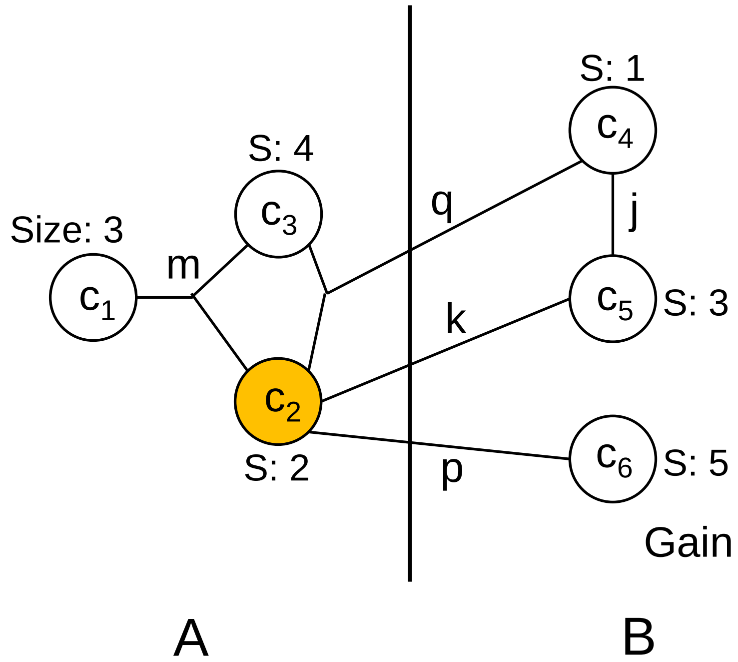
F(n) = 2 : g(c)++; // c ∈ n & F

Fiduccia-Mattheyses (FM) Algorithm

- Gain update
  - F: From_block of the base cell
  - T: To_block of the base cell
  - For each net n connected to the base cell
    - If T(n) = 0
      - gain(c)++; // for $c \in n$
    - If T(n) = 1
      - gain(c)--; // for $c \in n$
    - If F(n) = 1
      - gain(c)--; // for $c \in n$
    - If F(n) = 2
      - gain(c)++; // for $c \in n$
    - F(n)--;
    - T(n)++;

Fiduccia-Mattheyses (FM) Algorithm

- Before move, update the gain of all the other cells.



	Δg(c) in F	Δg(c) in T		F(n)	T(n)
(2)		-2		1	1
(3)		-1		1	≥ 2
(4)	+2			2	0
(5)	+1	-1		2	1
(6)	+1			2	≥ 2
(7)	+1			≥ 3	0
(8)		-1		≥ 3	1

Gain (before move)

$g(c_1) = 0 – 1 = -1$
$g(c_2) = 2 – 1 = +1$
$g(c_3) = 0 – 1 = -1$
$g(c_4) = 1 – 1 = 0$
$g(c_5) = 1 – 1 = 0$
$g(c_6) = 1 – 0 = +1$

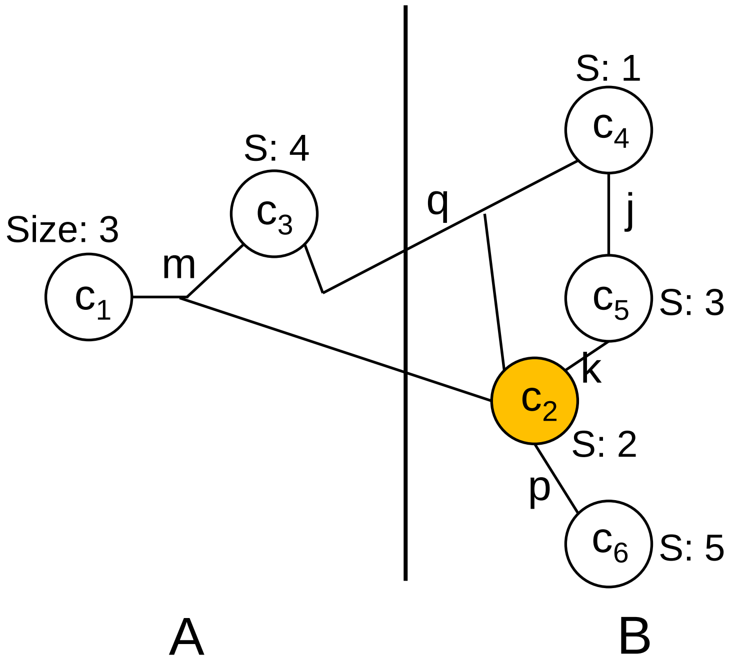
Gain (after move)

$g(c_1) = -1 + 1 = 0$

$g(c_3) = -1 + 1 + 1 = +1$
$g(c_4) = 0 – 1 = -1$
$g(c_5) = 0 – 2 = -2$
$g(c_6) = 1 – 2 = -1$

Fiduccia-Mattheyses (FM) Algorithm

- Move and lock the base cell.



Gain (after move)

$g(c_1) =$ $-1 + 1 = 0$

$g(c_3) = -1 + 1 + 1 = +1$
$g(c_4) = 0 - 1 = -1$
$g(c_5) = 0 - 2 = -2$
$g(c_6) = 1 - 2 = -1$

Fiduccia-Mattheyses (FM) Algorithm

- Choose the next base cell (except the locked cells).
- Update the gain of the other cells.
- Move and lock the base cell.
- Repeat this process.

- Find the best move sequence.

Fiduccia-Mattheyses (FM) Algorithm

For each net n connected to c
if F(n) = 1, g(c)++;
if T(n) = 0, g(c)--;

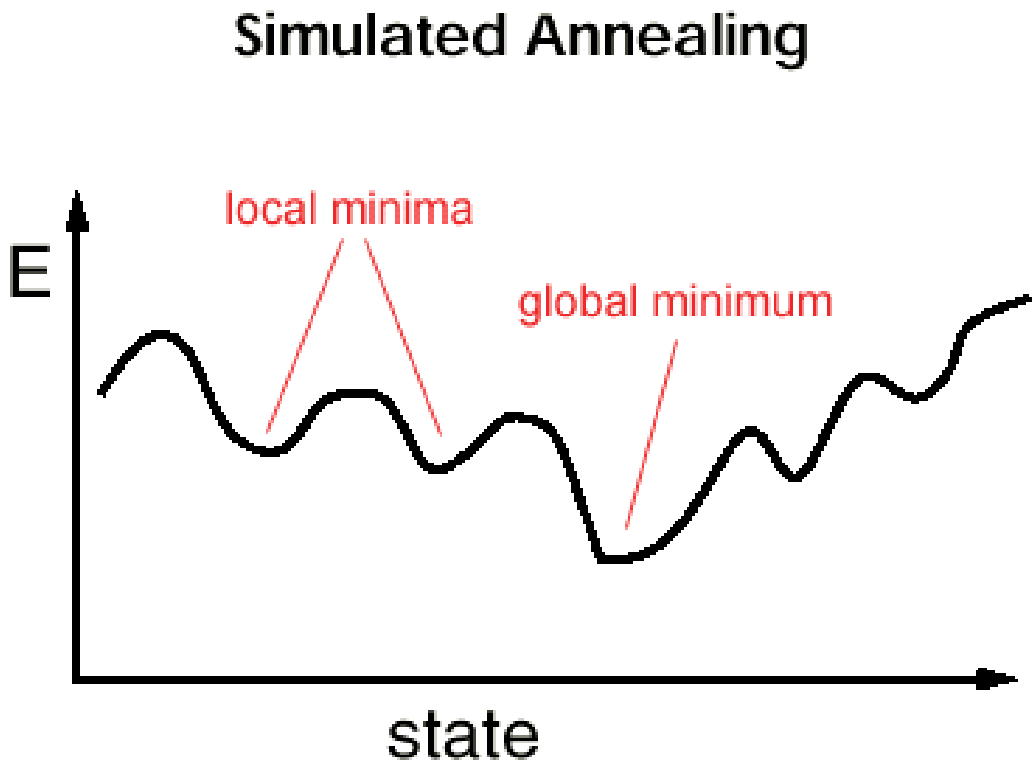
- Complexity analysis
  1. Gain computation
     - Compute the gain of each cell move → Practically O(# cells or # nets)
  2. Select a base cell → O(1)
  3. Move and lock the base cell and update gain. → Practically O(1)

iterations: # cells

Overall: **O(n or c)**

Simulated Annealing

- Borrowed from chemical process



Simulated Annealing

- Algorithm

T = T_0 (initial temperature)

S = S_0 (initial solution)

Time = 0

```
repeat
    Call Metropolis (S, T, M);
    Time = Time + M;
    T = α · T;  // α: cooling rate (α < 1)
    M = β · M;
until (Time ≥ maxTime);
```

Simulated Annealing

- Algorithm

  Metropolis (S, T, M) // M: # iterations

  repeat

  NewS = neighbor(S); // get a new solution by perturbation

  Δh = cost(NewS) – cost(S);

  If (($\Delta h < 0$) or (random < $e^{-\Delta h/T}$))

  S = NewS; // accept the new solution

  M = M – 1;

  until (M==0)

Simulated Annealing

- Cost function for partition (A, B)
  - Imbalance(A, B) = Size(A) – Size(B)
  - Cutsize(A, B) = Σw_n for $n \in \psi$
  - Cost = $W_c \cdot$ Cutsize(A, B) + $W_s \cdot$ Imbalance(A, B)
    - W_c and W_s: weighting factors

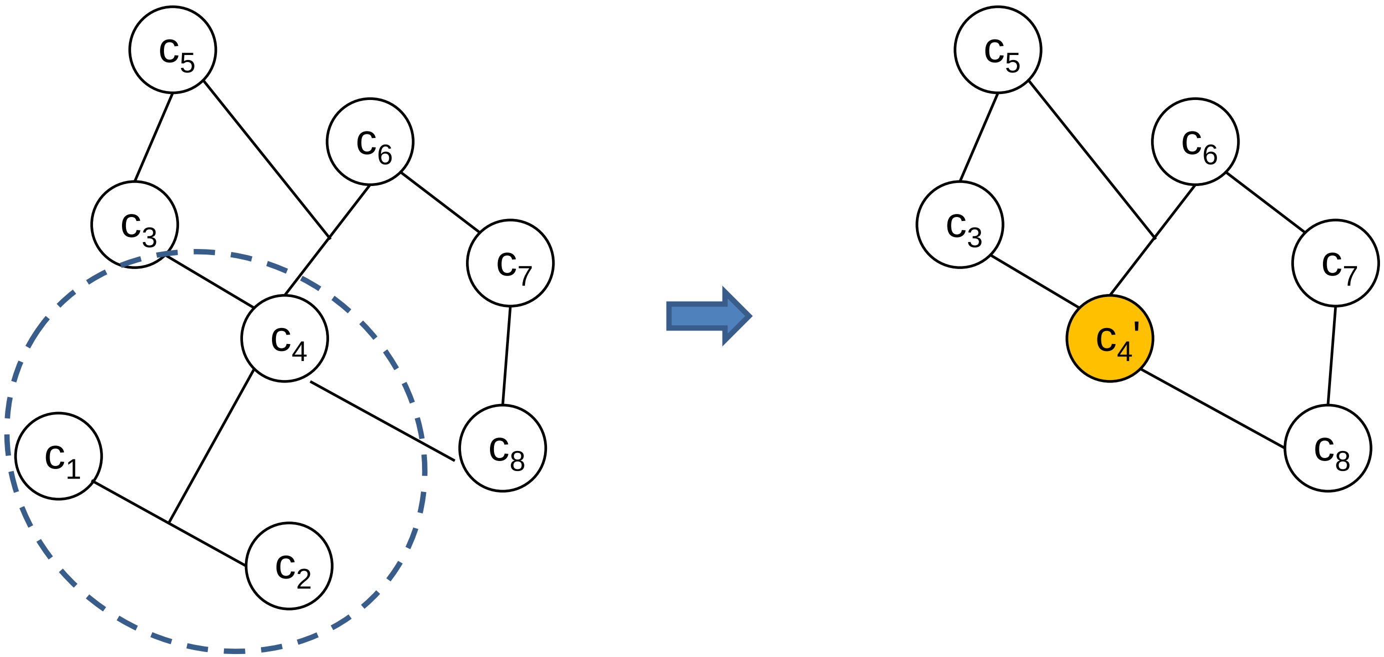
- Neighbor(S)
  - Solution perturbation
    - Example: move a free cell.

hMetis

- Clustering-based partitioning
  - Coarsening (grouping) by clustering
  - Uncoarsening and refinement for cut-size minimization

hMetis

- Coarsening



hMetis

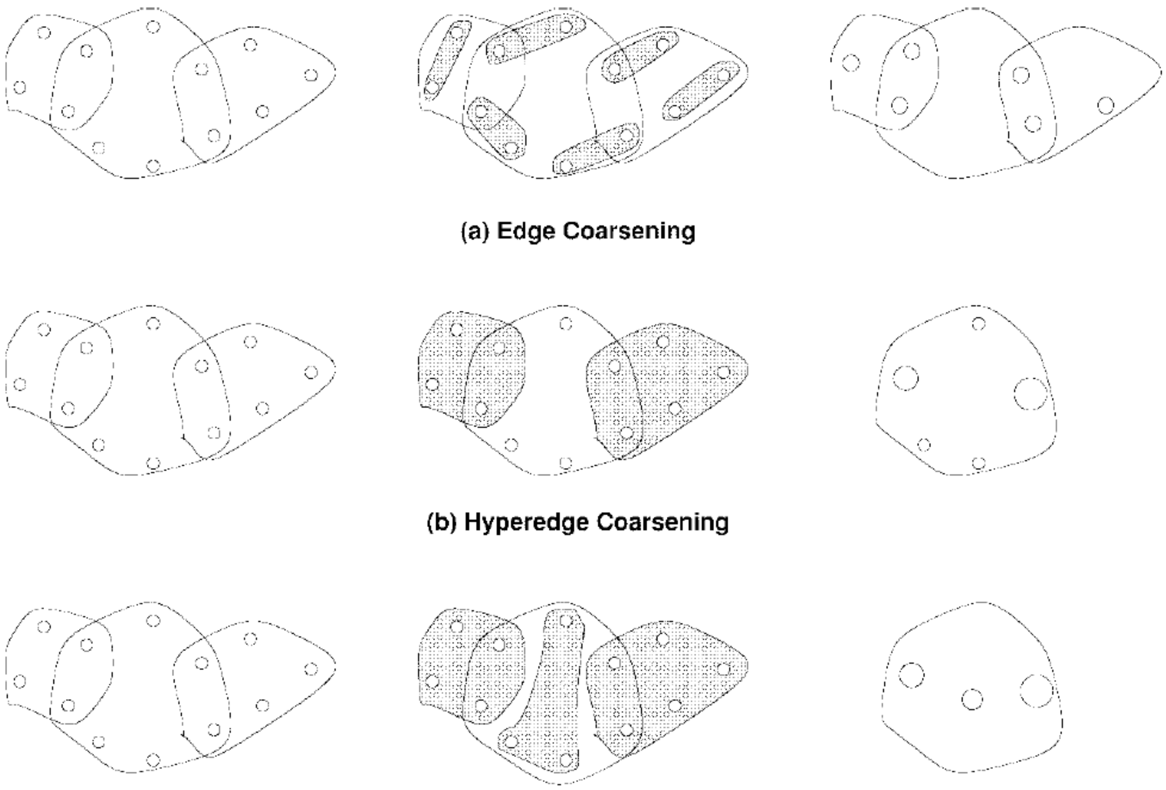
- Coarsening
  - Reduces the problem size
    - Make sub-problems smaller and easier.
      - Better runtime
      - Higher probability for optimality
  - Finds circuit hierarchy

hMetis

- Algorithm
  1. Coarsening
  2. Initial solution generation
     - Run partitioning for the top-level clusters.
  3. Uncoarsening and refinement
     - Flatten clusters at each level (uncoarsening).
     - Apply partitioning algorithms to refine the solution.

hMetis

- Three coarsening methods.



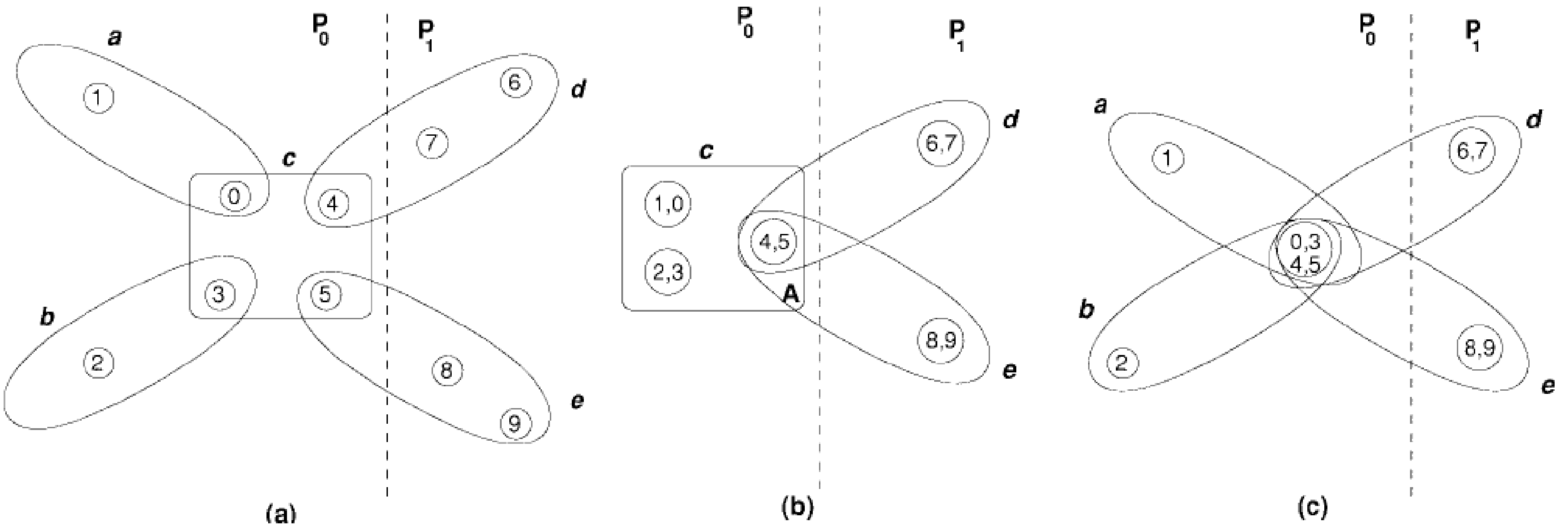
(a) Edge Coarsening

(b) Hyperedge Coarsening

(c) Modified Hyperedge Coarsening

hMetis

- Re-clustering (V- and v-cycles)
  - Different clustering gives different cutsizes.



hMetis

- Final results

Benchmark	PROP	CDIP-LA3$_f$	CLIP-PROP$_f$	PARABOLI	GFM	GMetis	Opt. KLFM	Best	hMETIS-EE$_{20}$	hMETIS-FM$_{20}$	hMETIS-EE$_{10vV}$	hMETIS-FM$_{20vV}$
balu	27	27	27	41	27	27	–	27	27	27	27	27
p1	47	47	51	53	47	47	–	47	52	50	49	49
bm1	50	47	47	–	–	48	–	47	51	51	51	51
t4	52	48	52	–	–	49	–	48	51	51	48	48
t3	59	57	57	–	–	62	–	57	58	58	59	58
t2	90	89	87	–	–	95	–	87	91	88	92	88
t6	76	60	60	–	–	94	–	60	62	60	63	60
struct	33	36	33	40	41	33	–	33	33	33	33	33
t5	79	74	77	–	–	104	–	74	71	71	71	71
19ks	105	104	104	–	–	106	–	104	107	106	106	105
p2	143	151	152	146	139	142	–	139	148	145	148	145
s9234	41	44	42	74	41	43	45	41	40	40	40	40
biomed	83	83	84	135	84	102	–	83	83	83	83	83
s13207	75	69	71	91	66	74	62	62	55	55	61	53
s15850	65	59	56	91	63	53	46	46	42	42	42	42
industry2	220	182	192	193	211	177	–	177	174	167	169	168
industry3	–	243	243	267	241	243	–	241	255	254	252	241
s35932	–	73	42	62	41	57	46	41	42	42	42	42
s38584	–	47	51	55	47	53	52	47	47	47	48	48
avq.small	–	139	144	224	–	144	–	139	136	130	128	127
s38417	–	74	65	49	81	69	–	49	52	51	54	50
avq.large	–	137	143	139	–	145	–	137	129	127	134	127
golem3	–	–	–	1629	–	2111	–	1629	1447	1445	1425	1424

hMetis

- Final results

hMETIS	Quality improvement											
EE_{20}	6.2%	5.3%	4.1%	21.4%	7.8%	10.0%	9.9%	0.3%				
FM_{20}	7.2%	6.4%	5.2%	22.4%	8.7%	11.0%	9.9%	1.4%	1.1%			
EE_{10uV}	6.4%	5.4%	4.1%	21.3%	7.5%	10.1%	7.6%	0.3%	-0.1%	-1.2%		
FM_{20uV}	7.9%	7.3%	6.1%	23.1%	9.4%	11.9%	10.1%	2.3%	2.0%	0.9%	2.0%	

Runtime Comparison. The times are in seconds on the specified machines												
	Sparc5	Sparc5	Sparc5	Dec3000 500AXP	Sparc10	Sparc5	Sparc IPX		SGI R10000	SGI R10000	SGI R10000	SGI R1000(
5 circuits							5606		95	125	62	180
13 circuits					46376				283	390	173	508
16 circuits	2383								158	224	103	303
16 circuits				37570					874	1593	382	1442
22 circuits		15850	16206						445	637	249	733
23 circuits						3357			913	1654	409	1513