Elective II: VLSI Design

Code: CISM 402

Pritha Banerjee

Courtsey for slides: Debasis Mitra, NIT Durgapur

Submission of Assignment

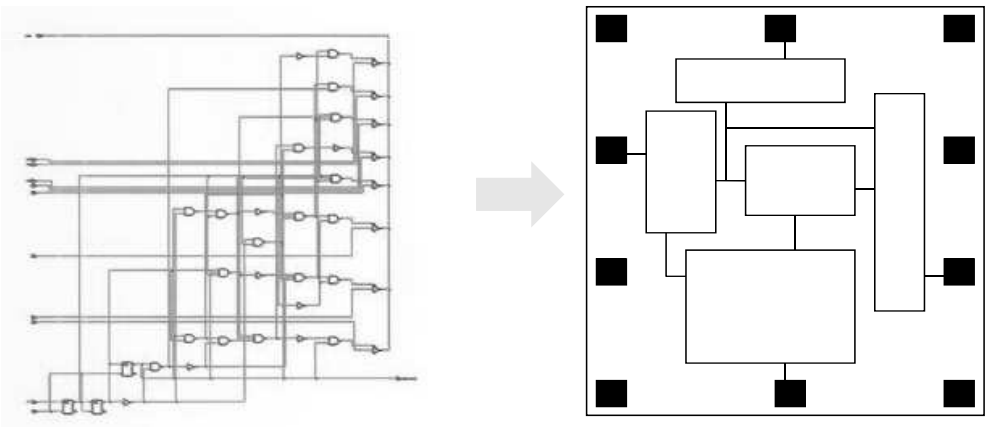
- Assignments I: April 10, 2011
  - Seven segment decoder
  - 4 bit ALU
  - 3-bit counter
- Assignment II: April 30, 2011
  - 8-bit processor

Resources

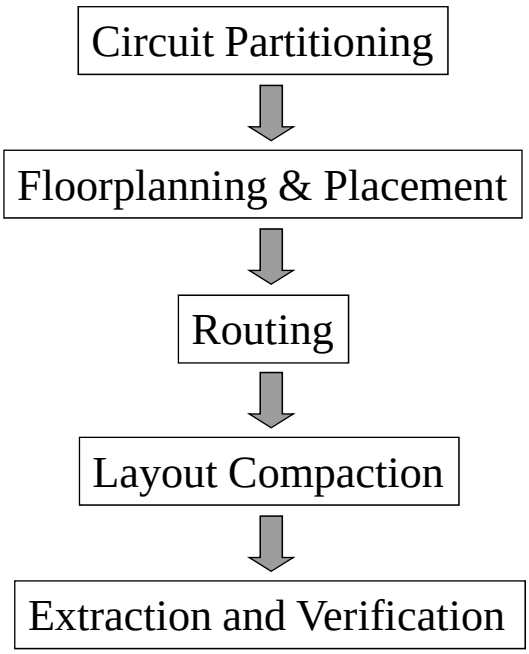
- Books:
  - Chapter 4 of Naveed A. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers
  - Chapter 2 (2.1) of M. Sarafzadeh and C. K. Wong, *An introduction to VLSI Physical Design*, The McGraw Hill Companies, Inc.

Physical Design Cycle

- Physical Design -- Convert the netlist (circuit description) into a geometric representation. The outcome is called a *layout*.



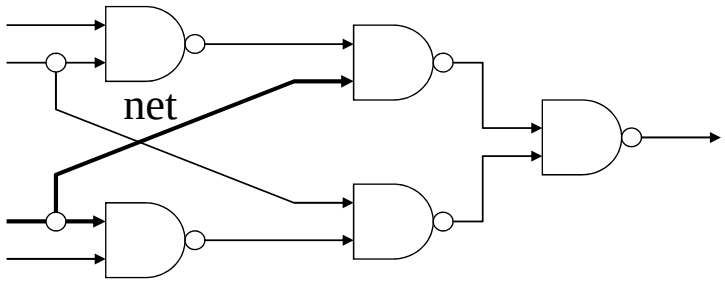
Physical Design Flow



Net and Netlist

Net: a set of terminals which have to be made electrically equivalent.

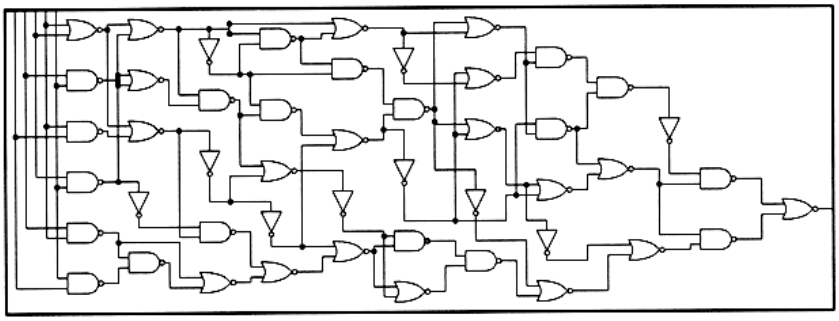
Netlist: a collection of nets



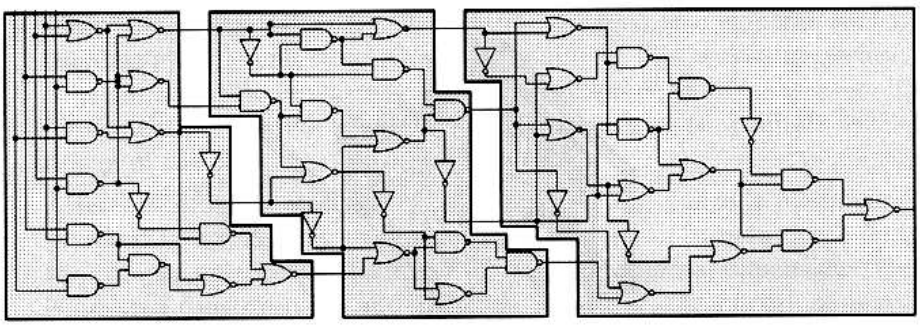
Circuit Partitioning

Partition a large circuit into sub-circuits (called blocks). Factors like # blocks, block sizes, interconnection between blocks, critical delay … are considered.

Circuit Partitioning



(a)

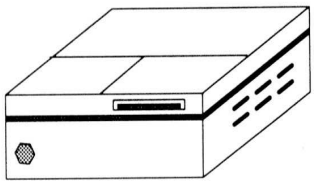


(b)

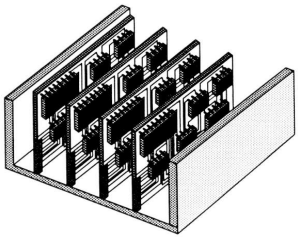
Partitioning in VLSI

A VLSI system is partitioned at several levels due to its complexity

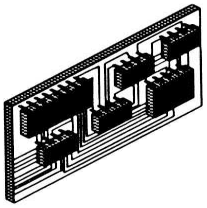
Partitioning in VLSI



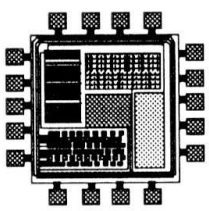
System level



Board level



Chip level



Some Terminology

Partitioning: Dividing bigger circuits into a small number of partitions (top down)

k-way Partitioning: Dividing into k partitions.

Bipartitioning: 2-way partitioning.

Bisectioning: Bipartitioning such that the two partitions have the same size.

Clustering: cluster small cells into bigger clusters (bottom up).

Circuit Partitioning

Objectives

- minimize interconnection between partitions
- minimize delay due to partitioning

Circuit Partitioning

Constraints

- number of terminals in a partition has an upper bound
- area of each partition
- number of partitions within a range

Algorithms for Partitioning

Input -

a) A set of components and

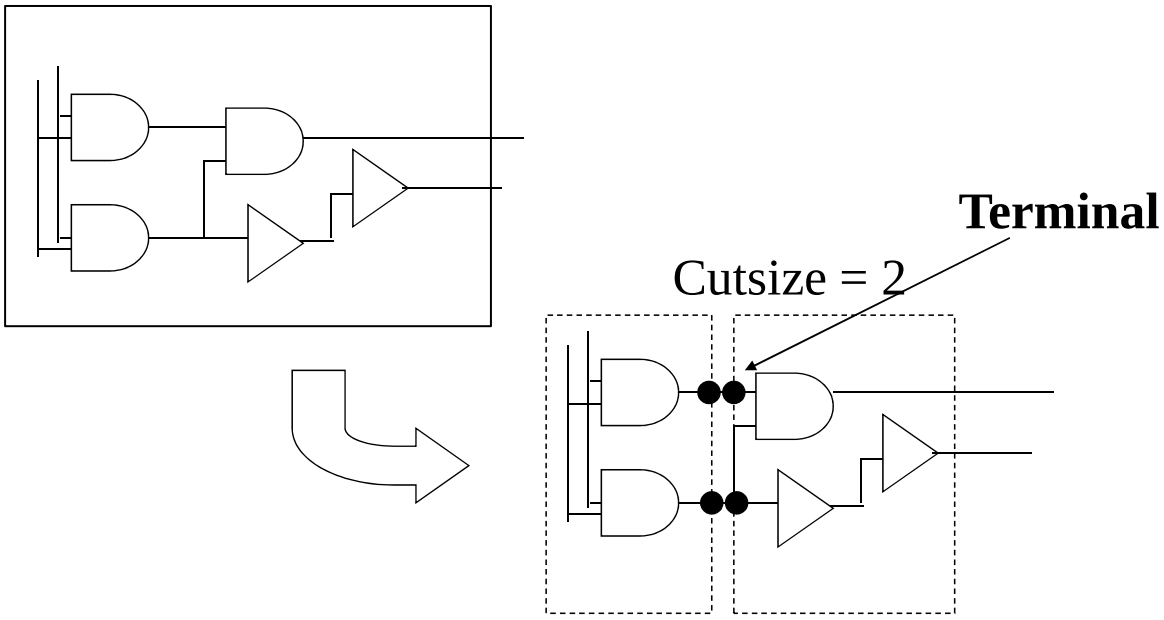
b) A netlist

Output -

a) A set of subcircuits which when connected, function as the original circuit and

b) Terminals required for each subcircuit to connect it to the others

Algorithms for Partitioning



Partitioning Algorithms - Classification

Based on input

- Constructive
  - –typically used for constructing some initial partition which can be improved by using other algorithms (preprocessing)
  - –input is the circuit components and the netlist
  - – output is a set of partitions and the new netlist
- Iterative
  - – accept an initial set of partitions and netlist
  - – generate an improved partitions set and modified netlist
  - –Iterates until the partitions cannot be improved further

Partitioning Algorithms - Classification

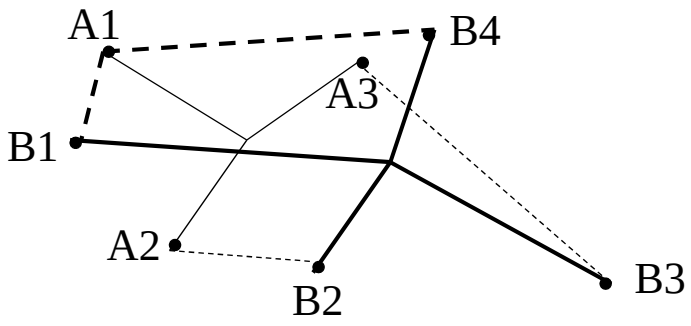
Based on process used

- Group Migration
  - starts with some partitions (usually generated randomly)
  - moves components between partitions to improve the partitioning
- Simulation Based
  - starts with a random solution and then
  - reaches the desired solution through movements from one feasible solution to another
  - uses a cost function, which classifies any feasible solution , and a set of moves, which allows movement to other solutions

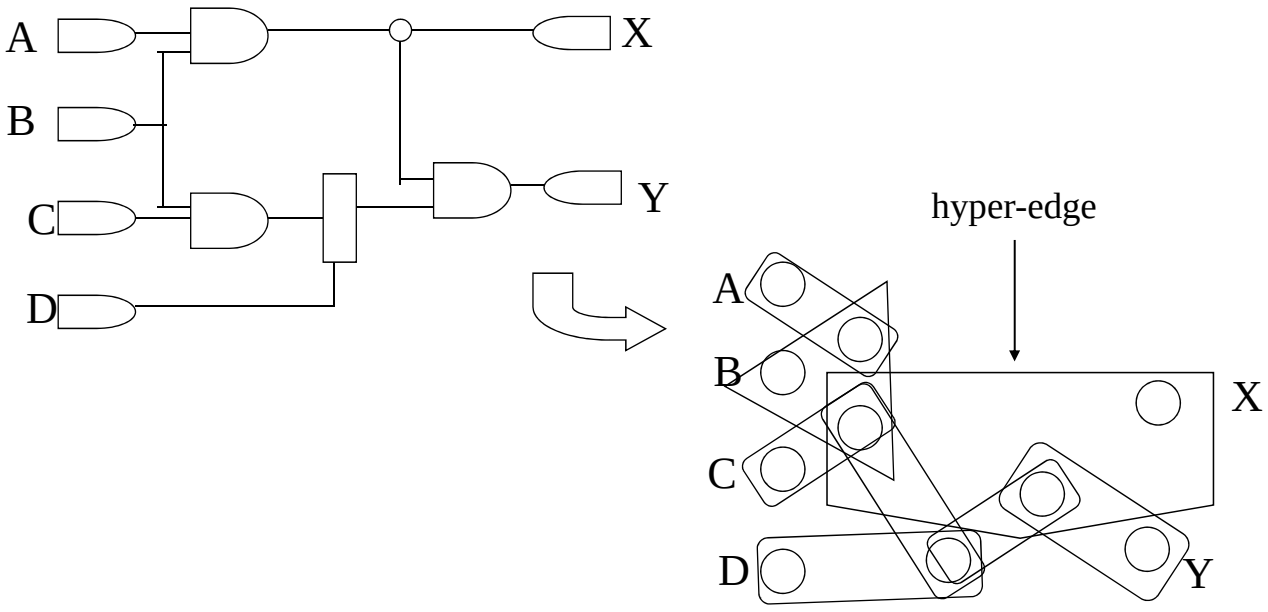
Hyper Graphs

A *hyperedge* is an interconnection of a set of distinct vertices

A *hypergraph* is a set of hyperedges

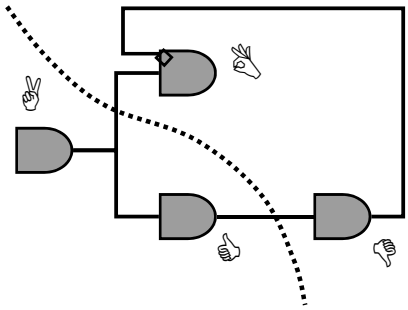


Modeling of a circuit by a Hyper Graph

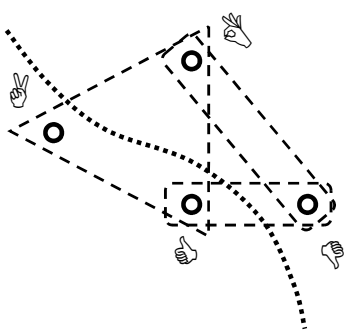


Circuit Representation

- Netlist:
  - Gates: A, B, C, D
  - Nets: {A,B,C}, {B,D}, {C,D}



- Hypergraph:
  - Vertices: A, B, C, D
  - Hyperedges: {A,B,C}, {B,D}, {C,D}



  - Vertex label: Gate size/area
  - Hyperedge label:
    Importance of net (weight)

Kernighan - Lin Algorithm

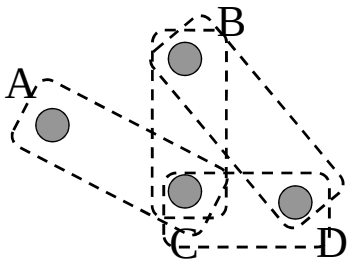
A Bisectioning Algorithm

a circuit can be partitioned into more than two partitions by the iterative application of the algorithm

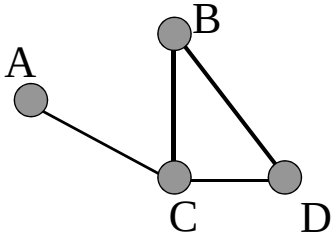
Kernighan - Lin Algorithm

Restricted Partition Problem

- Restrictions:
  - For Bisectioning of circuit
  - Assume all gates are of the same size
  - Works only for 2-terminal nets
    - If all nets are 2-terminal,the Hypergraph reduces to a Graph



Hypergraph Representation



Graph Representation

Problem Formulation

- Input: An unweighted graph with
  - Set vertices V. (|V| = 2n)
  - Set of edges E. (|E| = m)
  - Cost c_{AB} for each edge {A, B} in E.
- Output: 2 partitions X & Y such that
  - Total cost of edges cut is minimized.
  - Each partition has n vertices.

A Trivial Approach

- Try all possible bisections. Find the best one.
- For 4 vertices (A,B,C,D), 3 possibilities.
  1. X={A,B} & Y={C,D}
  2. X={A,C} & Y={B,D}
  3. X={A,D} & Y={B,C}
- If there are 2n vertices,

  # of possibilities = $^{2n}C_n = (2n)! / n!^2$
- For 100 vertices, 5×10^{28} possibilities.
  - Need 1.59×10^{13} years if one can try 100M possibilities per second

Kernighan - Lin Algorithm

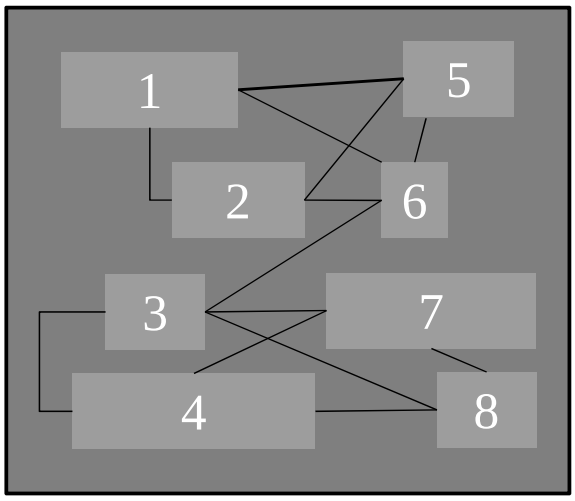
A Bisectioning Algorithm

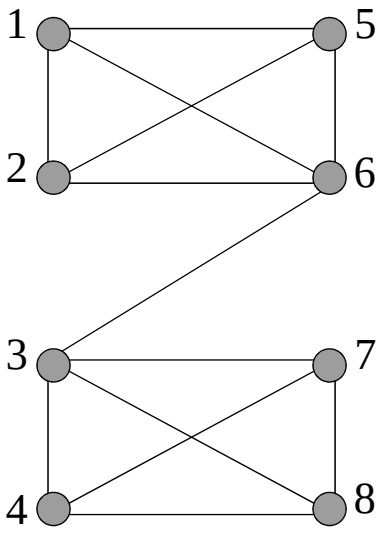
a circuit can be partitioned into more than two partitions by the iterative application of the algorithm

Kernighan - Lin Algorithm

Steps -

1. The circuit to be partitioned is represented as a graph *G(V,E)* with modules representing nodes and the connections between nodes representing as edges

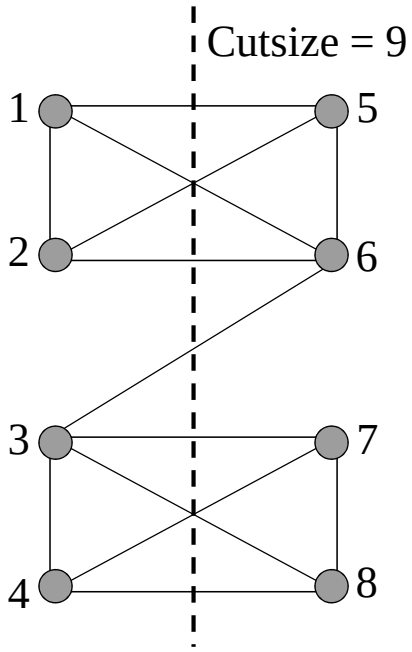




Kernighan - Lin Algorithm

Steps -

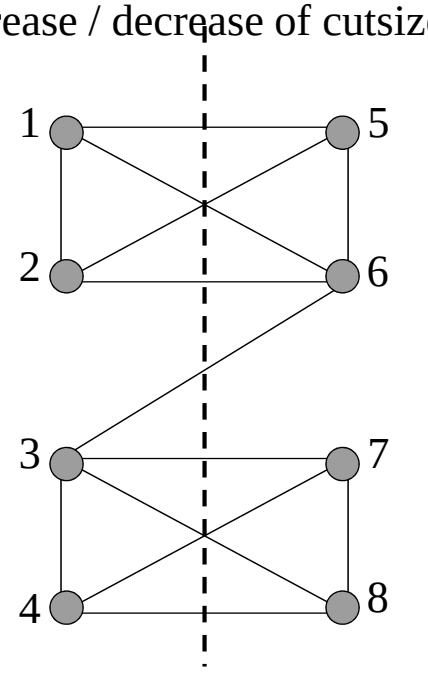
2. The graph *G(V,E)* is partitioned into two subsets *G1*, *G2* of equal sizes (balanced bipartition)



Kernighan - Lin Algorithm

Steps -

3. A vertex pair *v1*, *v2* is chosen from *G1*, *G2* such that it results largest decrease or smallest increase in cutsize, The vertex pair is then exchanged tentatively and increase / decrease of cutsize is recorded in a log

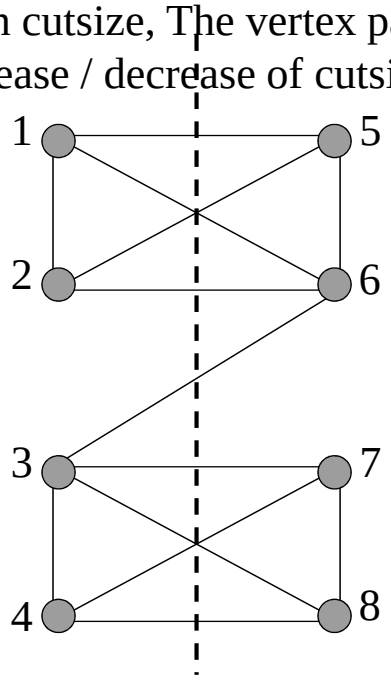


Kernighan - Lin Algorithm

Steps -

3. A vertex pair *v1*, *v2* is chosen from *G1*, *G2* such that it results largest decrease or smallest increase in cutsize, The vertex pair is then exchanged tentatively and increase / decrease of cutsize is recorded in a log

Vetex pair	Cutsize	gain
1, 5	9	0
1, 6	8	1
1,7	7	2
3,5	6	3
…	….	….
4, 8	9	0



Kernighan - Lin Algorithm

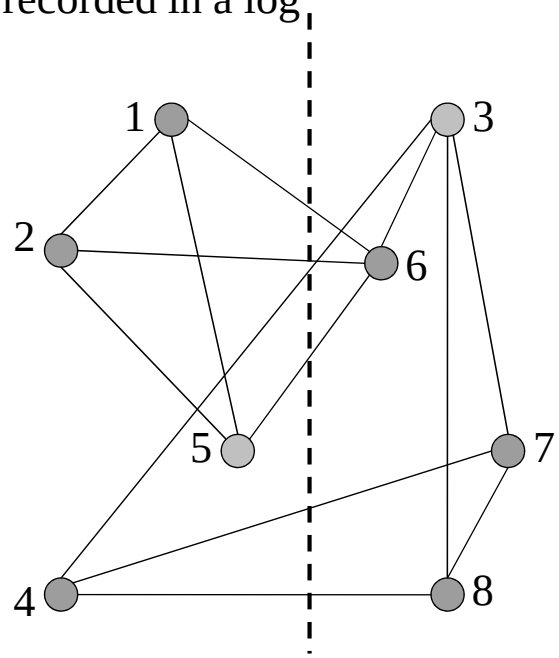
Steps -

3. A vertex pair *v1*, *v2* from *G1*, *G2* is exchanged tentatively and increase / decrease of cutsize is recorded in a log

Gain (decrease in cutsize) g(3,5) = 3

3 and 5
exchanged

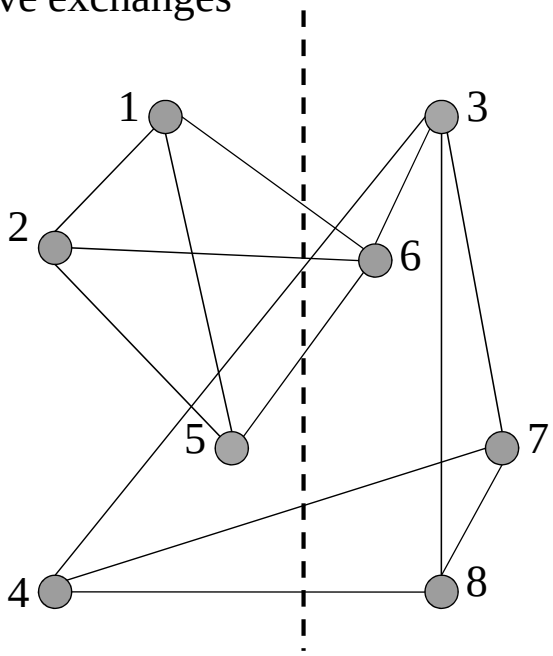
Cutsize = 6



Kernighan - Lin Algorithm

Steps -

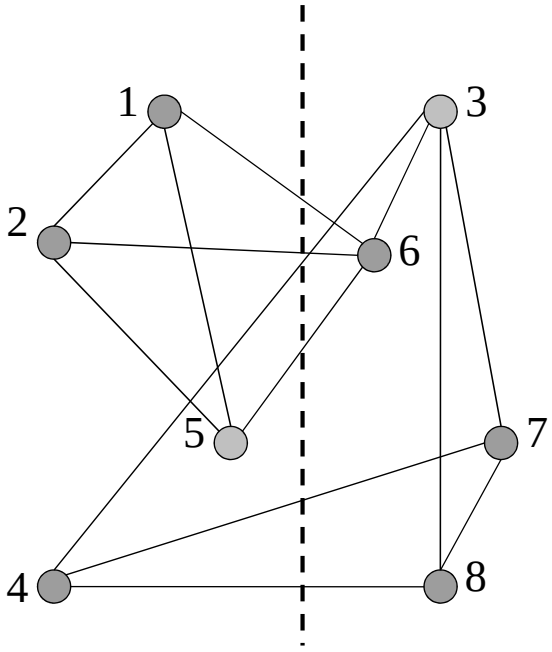
4. The vertex pair *v1, v2* is then locked prohibiting them from taking part in any further tentative exchanges



Kernighan - Lin Algorithm

Steps -

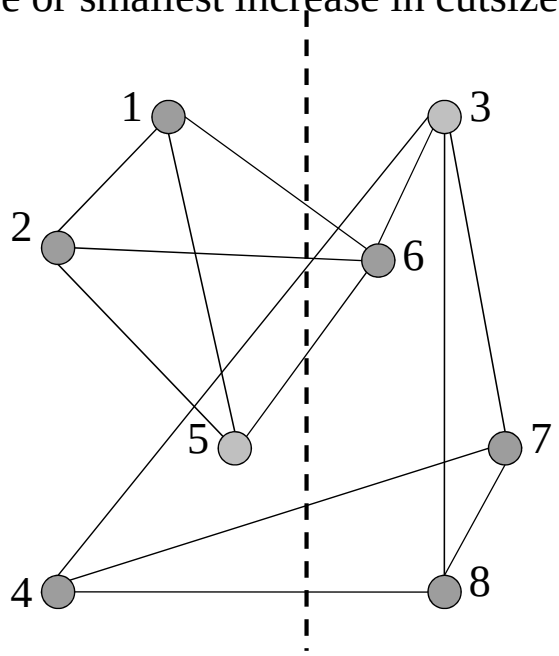
5. Step 3 and 4 is repeatedly applied to the new partitions until all the vertices are locked



Kernighan - Lin Algorithm

Steps -

6. A set of vertex pairs are selected for actual exchange whose exchange results largest decrease or smallest increase in cutsize



Kernighan - Lin Algorithm

Steps -

6. A set of vertex pairs are selected for actual exchange whose exchange results largest decrease or smallest increase in cutsize

Decision is made based on the entries in the log

Tentative Exchange Number (k)	Vertex pair	gain g_k	Partial sum of gain Σg_j j=0 to k	cutsize
0	**-**	**-**	**-**	**9**
1	**3,5**	**3**	**3**	**6**
2	**4,6**	**5**	**8**	**1**
3	**1,7**	**-6**	**2**	**7**
4	**2,8**	**-2**	**0**	**9**

Kernighan - Lin Algorithm

Steps -

6. A set of vertex pairs are selected for actual exchange whose exchange results largest decrease or smallest increase in cutsize

Decision is made based on the entries in the log

Tentative Exchange Number (k)	Vertex pair	gain g_k	Partial sum of gain Σg_j j=0 to k	cutsize
0	-	-	-	9
1	3,5	3	3	6
2	4,6	5	8	1
3	1,7	-6	2	7
4	2,8	-2	0	9

The value of k for which maximum partial sum is achieved is determined

Kernighan - Lin Algorithm

Steps -

6. A set of vertex pairs are selected for actual exchange whose exchange results largest decrease or smallest increase in cutsize

Decision is made based on the entries in the log

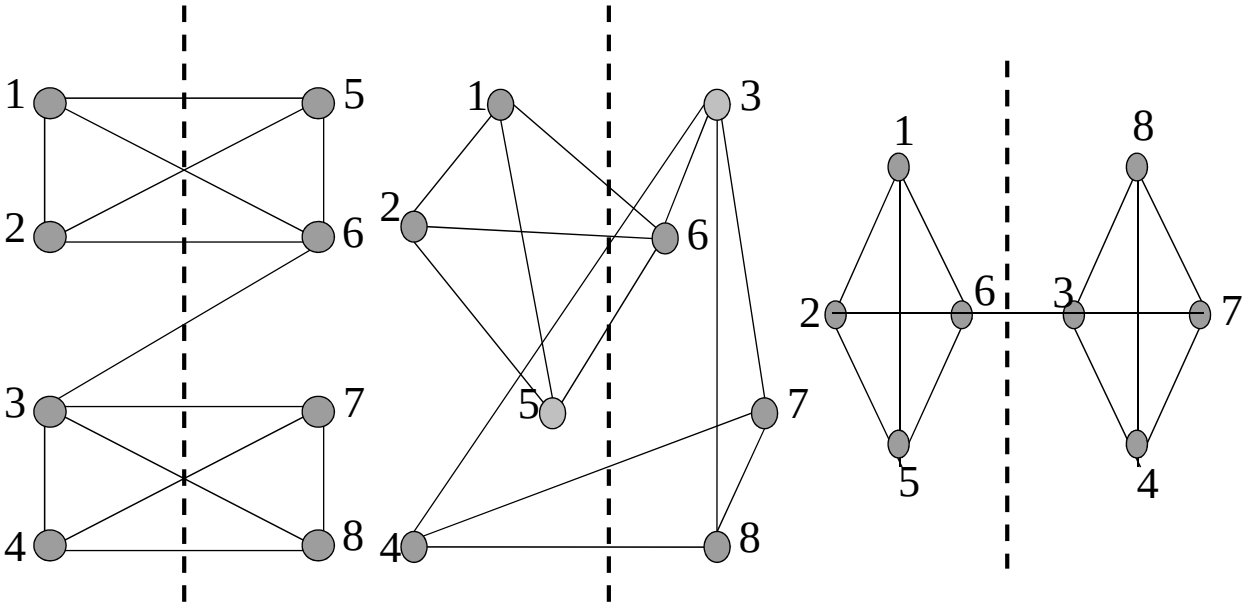
Tentative Exchange Number (k)	Vertex pair	gain g_k	Partial sum of gain Σg_j j=0 to k	cutsize
0	-	-	-	9
1	3,5	3	3	6
2	4,6	5	8	1
3	1,7	-6	2	7
4	2,8	-2	0	9

The first k pairs of vertices are actually exchanged

Kernighan - Lin Algorithm

Steps -

6. A set of vertex pairs are selected for actual exchange whose exchange results largest decrease or smallest increase in cutsize



Kernighan - Lin Algorithm

Steps -

7. A new iteration of steps 2 to 6 is started. The algorithms **stops** when no decrease of cutsize is possible during an iteration

Kernighan - Lin Algorithm

KL(G)

- Bipartition G into 2 groups V1, V2
- Repeat
  - For i = 1 to n/2 do
    - Find a pair of unlocked vertices va in V1 and vb in V2 whose exchange makes largest decrease or smallest increase in cut-cost
    - Mark va, vb as locked, store gain gi
  - Find k s.t $\sum gi = Gain_k$ is maximized
  - If $Gain_k > 0$
    - ✓ move va1, va2, ….vak from V1 to V2
    - ✓ move vb1, vb2, ….vbk from V2 to V1
- Unitil $Gain_k <= 0$

Kernighan - Lin Algorithm

Computing gain formally:

$g(v_i, v_j)$ = decrese in cutsize if v_i changes over to the other partition D(i)

+

decrese in cutsize if v_j changes over to the other partition D(j)

-

2 * no. of edges between v_i and v_j

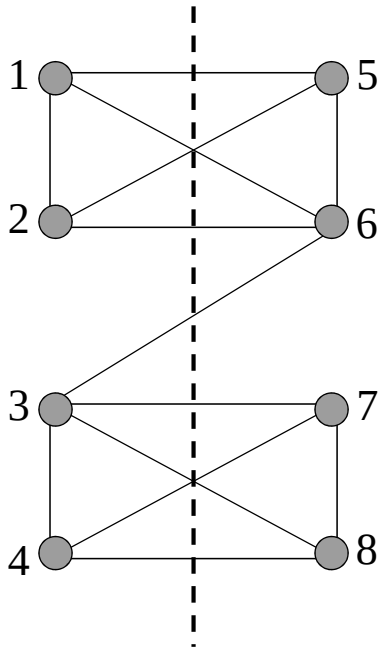
- D(i) - outedge(i) – inedge(i)
- Edge (v_i , v_j) do not contribution on the overall gain, as
  - vi ,vj remain in different partitions and edges between them will still cross the partition.
  - Edge(v_i , v_j) has been considered to contribute in both the cases of individual migration of vi and vj, hence subtract 2.

Kernighan - Lin Algorithm

Computing gain formally:

Example:

D(3) = 3 – 1 = 2
D(5) = 2 – 1 = 1
Hence,
g(3, 5) = 2 + 1 -2*0 = 3



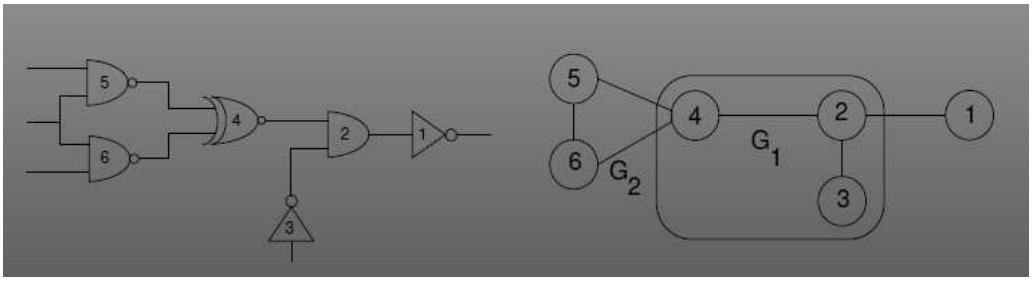
Kernighan - Lin Algorithm

Time Complexity of KL Algorithm: $O(n^3)$

- For each pass,
  - $O(n^2)$ time to find the best pair to exchange.
  - n pairs exchanged.
  - Total time is $O(n^3)$ per pass.
- Number of passes is usually small (can be considered as constant)
- *Extensions: can be extended to r –way partition*
  - Bipartition rC_2 pair of subsets, each taking $(n/r)^3$
  - Time Complexity: $(rC_2)\ (n/r)^3 = O(n^3/r)$

Kernighan - Lin Algorithm

Another example:



- Step 1: Initialization.

  Let the initial partition be a random division of vertices into the partition A={2,3,4} and B={1,5,6}.

  $A' = A = \{2,3,4\}$, and $B' = B = \{1,5,6\}$.

Kernighan - Lin Algorithm

Another example:

- Step 2: Compute $D-$values.

  $D_1 = E_1 - I_1 = 1 - 0 = +1$
  $D_2 = E_2 - I_2 = 1 - 2 = -1$
  $D_3 = E_3 - I_3 = 0 - 1 = -1$
  $D_4 = E_4 - I_4 = 2 - 1 = +1$
  $D_5 = E_5 - I_5 = 1 - 1 = +0$
  $D_6 = E_6 - I_6 = 1 - 1 = +0$

Kernighan - Lin Algorithm

Another example:

- Step 3: Compute gains.

$$g_{21} = D_2 + D_1 - 2c_{21} = (-1) + (+1) - 2(1) = -2$$
$$g_{25} = D_2 + D_5 - 2c_{25} = (-1) + (+0) - 2(0) = -1$$
$$g_{26} = D_2 + D_6 - 2c_{26} = (-1) + (+0) - 2(0) = -1$$
$$g_{31} = D_3 + D_1 - 2c_{31} = (-1) + (+1) - 2(0) = +0$$
$$g_{35} = D_3 + D_5 - 2c_{35} = (-1) + (+0) - 2(0) = -1$$
$$g_{36} = D_3 + D_6 - 2c_{36} = (-1) + (+0) - 2(0) = -1$$
$$g_{41} = D_4 + D_1 - 2c_{41} = (+1) + (+1) - 2(0) = +2$$
$$g_{45} = D_4 + D_5 - 2c_{45} = (+1) + (+0) - 2(1) = -1$$
$$g_{46} = D_4 + D_6 - 2c_{46} = (+1) + (+0) - 2(1) = -1$$

- The largest g value is g_{41}. (a_1, b_1) is $(4, 1)$, the gain $g_{41} = g_1 = 2$, and $A' = A' - \{4\} = \{2,3\}$, $B' = B' - \{1\} = \{5, 6\}$.

Kernighan - Lin Algorithm

Another example:

- Both A' and B' are not empty; then we update the $D-$values in the next step and repeat the procedure from Step 3.
- Step 4: Update $D-$values of nodes connected to (4,1).

The vertices connected to (4,1) are vertex (2) in set A' and vertices (5,6) in set B'. The new $D-$values for vertices of A' and B' are given by

$$D_2' = D_2 + 2c_{24} - 2c_{21} = -1 + 2(1 - 1) = -1$$
$$D_5' = D_5 + 2c_{51} - 2c_{54} = +0 + 2(0 - 1) = -2$$
$$D_6' = D_6 + 2c_{61} - 2c_{64} = +0 + 2(0 - 1) = -2$$

Kernighan - Lin Algorithm

Another example:

- To repeat Step 3, we assign $D_i = D_i^{'}$ and then recompute the gains:

$$g_{25} = D_2 + D_5 - 2c_{25} = (-1) + (-2) - 2(0) = -3$$
$$g_{26} = D_2 + D_6 - 2c_{26} = (-1) + (-2) - 2(0) = -3$$
$$g_{35} = D_3 + D_5 - 2c_{35} = (-1) + (-2) - 2(0) = -3$$
$$g_{36} = D_3 + D_6 - 2c_{36} = (-1) + (-2) - 2(0) = -3$$

- All the g values are equal, so we arbitrarily choose g_{36}, and hence the pair (a_2, b_2) is $(3, 6)$,

$$g_{36} = g_2 = -3,$$
$$A' = A' - \{3\} = \{2\},$$
$$B' = B' - \{6\} = \{5\}.$$

Kernighan - Lin Algorithm

Another example:

- The new $D-$values are:

$$D_2^{'} = D_2 + 2c_{23} - 2c_{26} = -1 + 2(1 - 0) = 1$$
$$D_5^{'} = D_5 + 2c_{56} - 2c_{53} = -2 + 2(1 - 0) = 0$$

- The corresponding new gain is:

$$g_{25} = D_2 + D_5 - 2c_{52} = (+1) + (0) - 2(0) = +1$$

- Therefore the last pair (a_3, b_3) is (2,5) and the corresponding gain is $g_{25} = g_3 = +1$.

Kernighan - Lin Algorithm

Another example:

- Step 5: Determine k.
- We see that $g_1 = +2$, $g_1 + g_2 = -1$, and $g_1 + g_2 + g_3 = 0$.
- The value of k that results in maximum G is 1.
- Therefore, $X = \{a_1\} = \{4\}$ and $Y = \{b_1\} = \{1\}$.
- The new partition that results from moving X to B and Y to A is, $A = \{1, 2, 3\}$ and $B = \{4, 5, 6\}$.
- The entire procedure is repeated again with this new partition as the initial partition.
- Verify that the second iteration of the algorithm is also the last, and that the best solution obtained is $A = \{1, 2, 3\}$ and $B = \{4, 5, 6\}$.

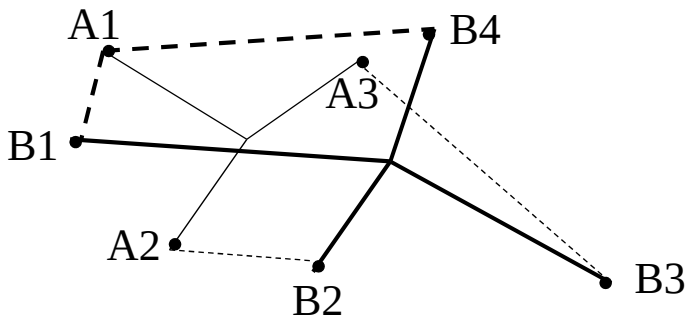
Drawbacks of KL Algorithm

- Considers unit vertex weights
  - Vertex weight: *Represent block sizes, which usually differ from block to block*
- Not applicable to hypergraphs; KL uses edge-cut model (multi terminal nets are represented by complete graph on the set of terminals)
- Partition sizes have to be pre-specified
- High time complexity

How to handle these drawbacks?

Fiduccia-Mattheyses Heuristic Terminology

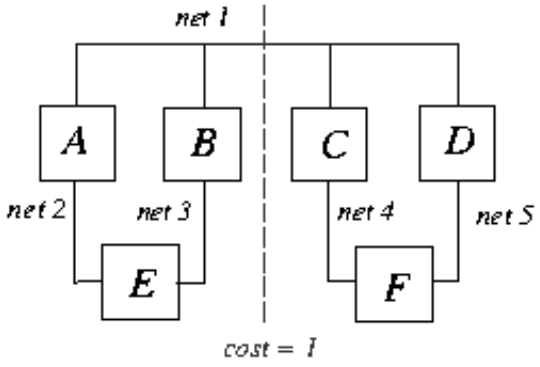
- Improvement over KL : time complexity per pass is O(t), t : # of terminals in G

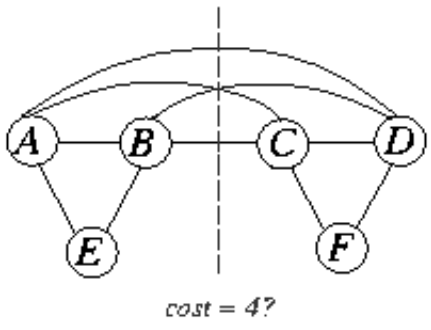


terminals: 3+ 4+ 2 + 2+2+2 = 15

Handling Hypergraphs

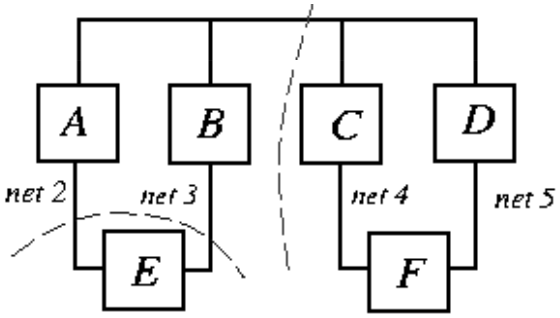
- For multi-terminal nets, net cut is a more accurate measurement for cut cost (i.e., deal with hyperedges).
- {A B E} {C D F} is a good partition
- Should not assign same weight to all edges

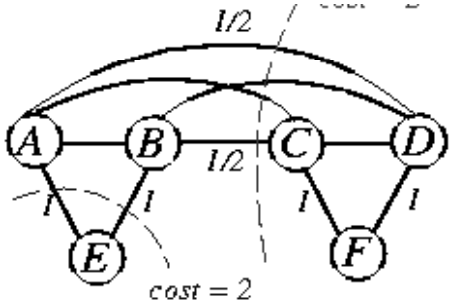




Handling Hypergraphs: net-cut model

- *n(i)* = # of cells associated with Net i.
- Edge weight w_{xy} =2 / n(i) for an edge connecting cells x and y.



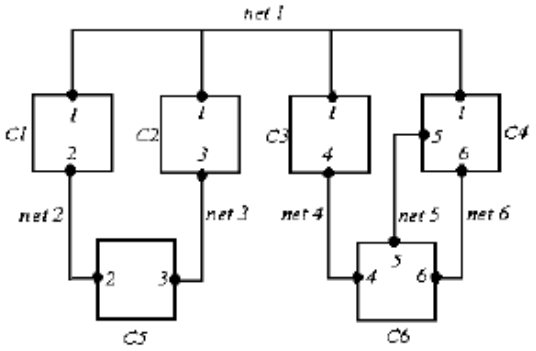


Fiduccia-Mattheyses Heuristic

- Improvement over KL :
  - Aims at reducing net-cut costs; the concept of cutsize is extended to hypergraphs.
  - time complexity per pass is O(t), t : # of terminals in G (A special data structure used to select a vertex to be moved to the other partition)
  - Only a single vertex is moved across the cut in a single move
  - Vertices are weighted
  - Can handle “unbalanced” partitions; (can obtain partitions with given balance factor)
- Balance Factor:
  - $w(v_i)$ = weight of vertex v_i
  - Weight of each partition : $\alpha \sum w(v_i)$, ½ <= α < 1

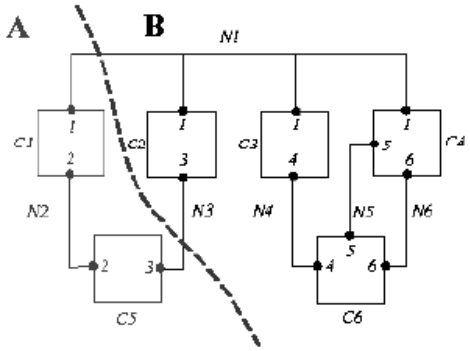
Fiduccia-Mattheyses : Notations

- n(i): # of cells in Net i; e.g., n(1) = 4.
- s(i): size of Cell i (weight of cell)
- t(i): # of pin terminals in Cell i; e.g., t(6)=3.
- C: total # of cells; e.g., C=6.
- N: total # of nets; e.g., N=6.
- t: total # of terminals; t = t(1) + … + t(C) = n(1) + … + n(N).



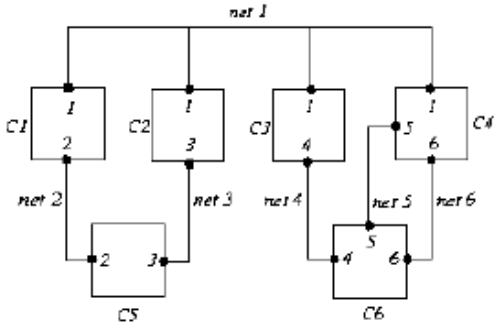
Fiduccia-Mattheyses Heuristic: Problem

- Input : A weighted (vertex and edges) hypergraph, required balance factor b,
- Output: balanced bipartition with minimum cut satisfying the balance factor b
- W(A) , W(B) <= ∑w(v)/2 + max{w(v)}, [if equal size bipartition], W(A): sum of weights of v in partition A
- w(v) : area/size of module represented by v



- Cut set = {N1, N3}
- W(A) = Size(A) = s(1) + s(2) = |A|
- W(B) = Size(B) = s(3)+ s(4)+ s(5)+ s(6) = |B|

Fiduccia-Mattheyses Heuristic: Input Data structure



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

- #of terminals t : 14
- Construction takes O(t) time

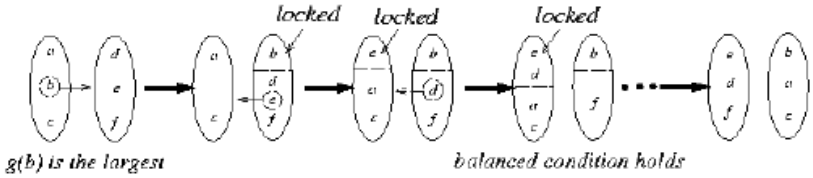
Fiduccia-Mattheyses Heuristic

- Initial Partition : A,B
  - Sort v in decreasing order of size(weight), Put alternate vertices in A and B.
- Movement of a single vertex causes a change of atmost :- pmax to +pmax in the cut-cost (gain).
- pmax: max degree of vertex
- Maintain 2 lists A and B for each partition indexed by the gain
- Choose the max gain vertex from either A or B if its movement does not violate the balance condition

Fiduccia-Mattheyses Heuristic (contd.)

- Move the vertex temporarily as in KL, so lock vertex
- Update the data structure to reflect the change in gains of vertices which are adjacent to the moved vertices
- Continue till all vertices are locked.
- Choose the sequence of moves (Partial sum of gains) s.t cut-cost is minimized
- Continue with another pass till there is no more improvement in cut cost

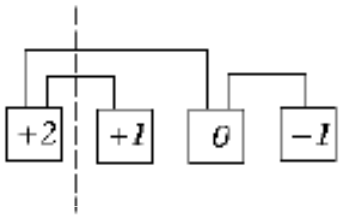
  - $g(i)$: gain in moving cell i to the other set, i.e., size of old cutset - size of new cutset.

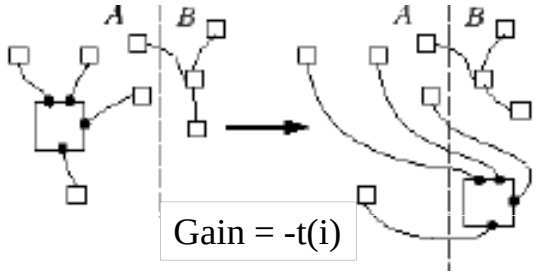


  - Suppose $\widehat{g_i}$'s: $g(b)$, $g(e)$, $g(d)$, $g(a)$, $g(f)$, $g(c)$ and the largest partial sum is $g(b)+g(e)+g(d)$. Then we should move b, e, $d \Rightarrow$ two resulting sets: $\{a, c, e, d\}$, $\{b, f\}$.

Cell gain

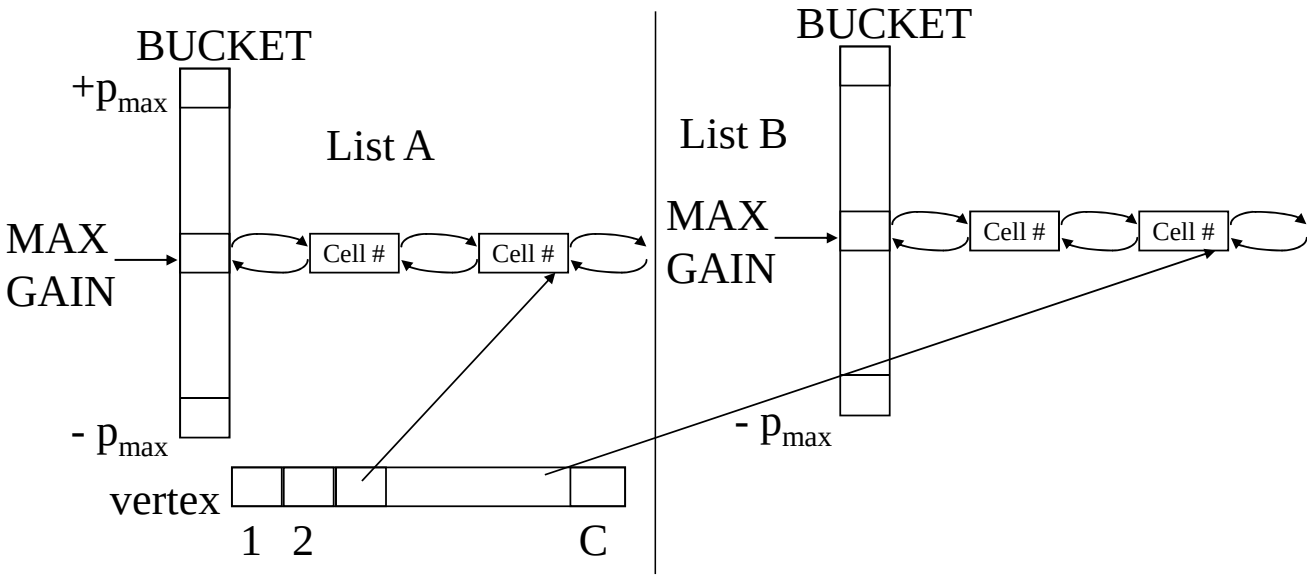
- $-t(i) \leq g(i) \leq t(i)$





Two "bucket list" structures, one for set *A and one for set B* ($P_{max} = \max_i t(i)$).

Fiduccia-Mattheyses Heuristic : Data Structure

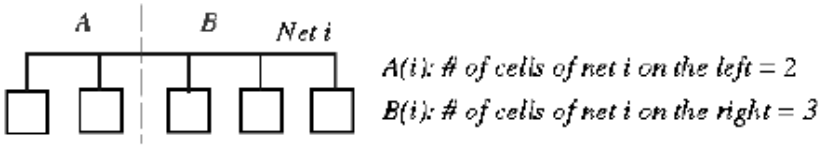


O(1)-time operations:

• find a cell with Max Gain, remove Cell i from the structure, insert Cell i into the structure, update g(i) to g(i)+ Δ,
and update the Max Gain pointer.

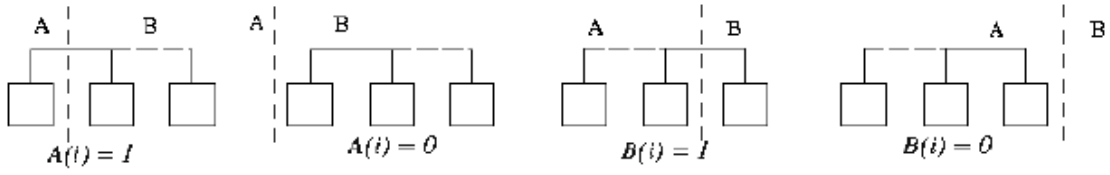
Distribution of nets and critical nets

- Distribution of Net i: (A(i), B(i)) = (2, 3).
  - (A(i), B(i)) for all i can be computed in O(t) time.



•Critical Nets: A net is critical if it has a cell which if moved will change its cut state.

- 4 cases: *A(i) = 0 or 1, B(i) = 0 or 1.*



Gain of a cell depends only on its critical nets.

Compute Cell gains

- Initialization of all cell gains requires O(t) time:

g(i) = 0;

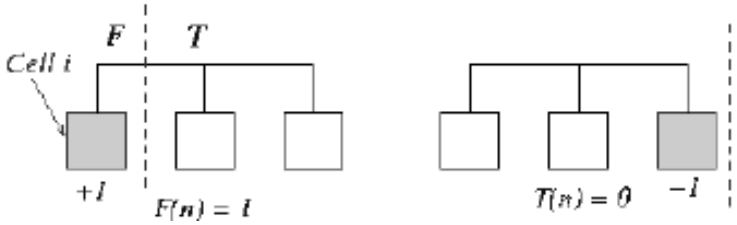
F = the "from partition" of Cell i;

T = the "to partition" of Cell i;

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

if *F(n)=1 then g(i) = g(i)+1;*

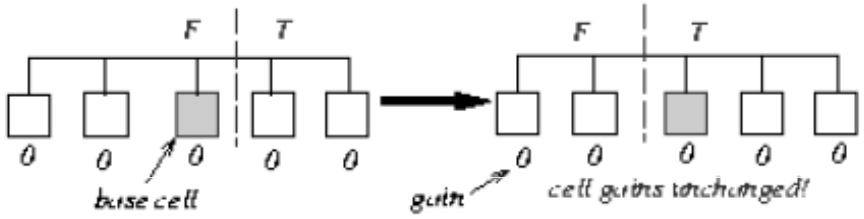
if *T(n)=0 then g(i) = g(i)-1;*



Only need O(t) time to maintain all cell gains in one pass.

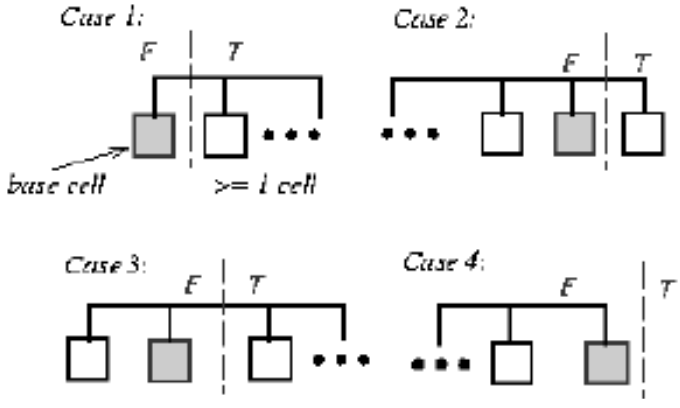
Update Cell gains

- To update the gains, 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

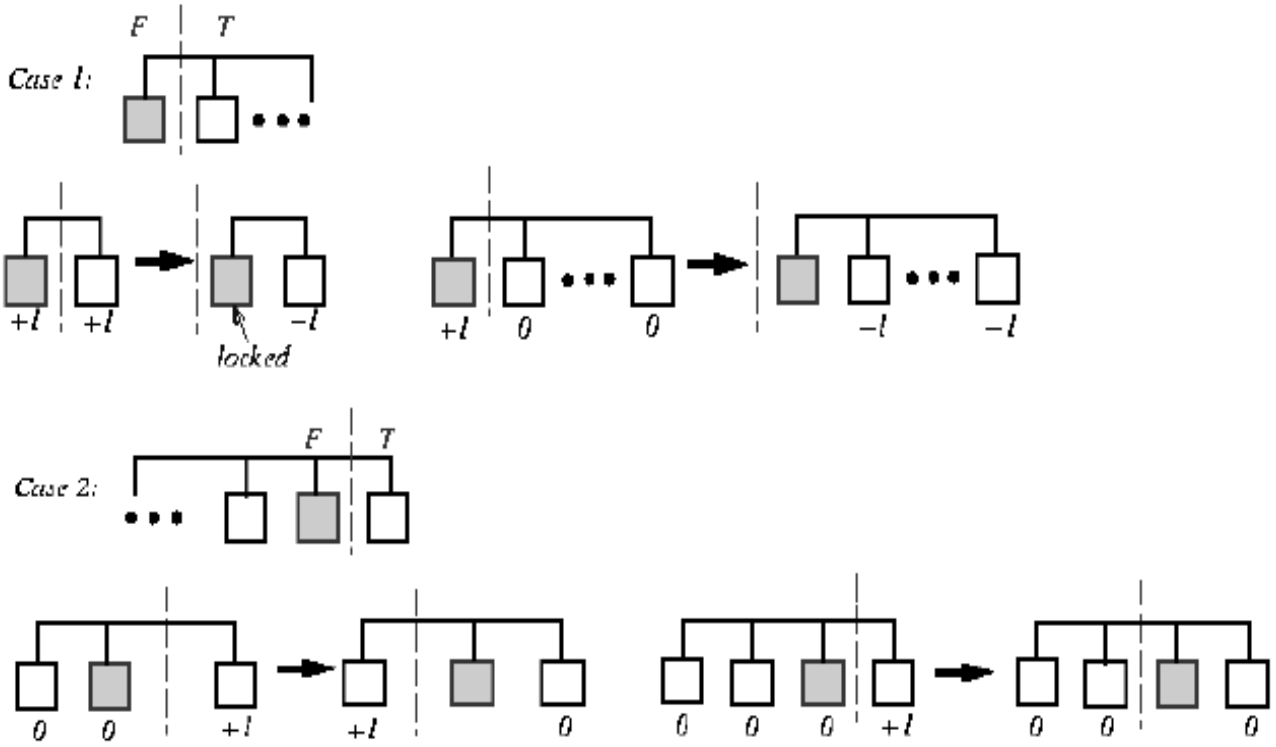


Consider only the case where the base cell is in the left partition.

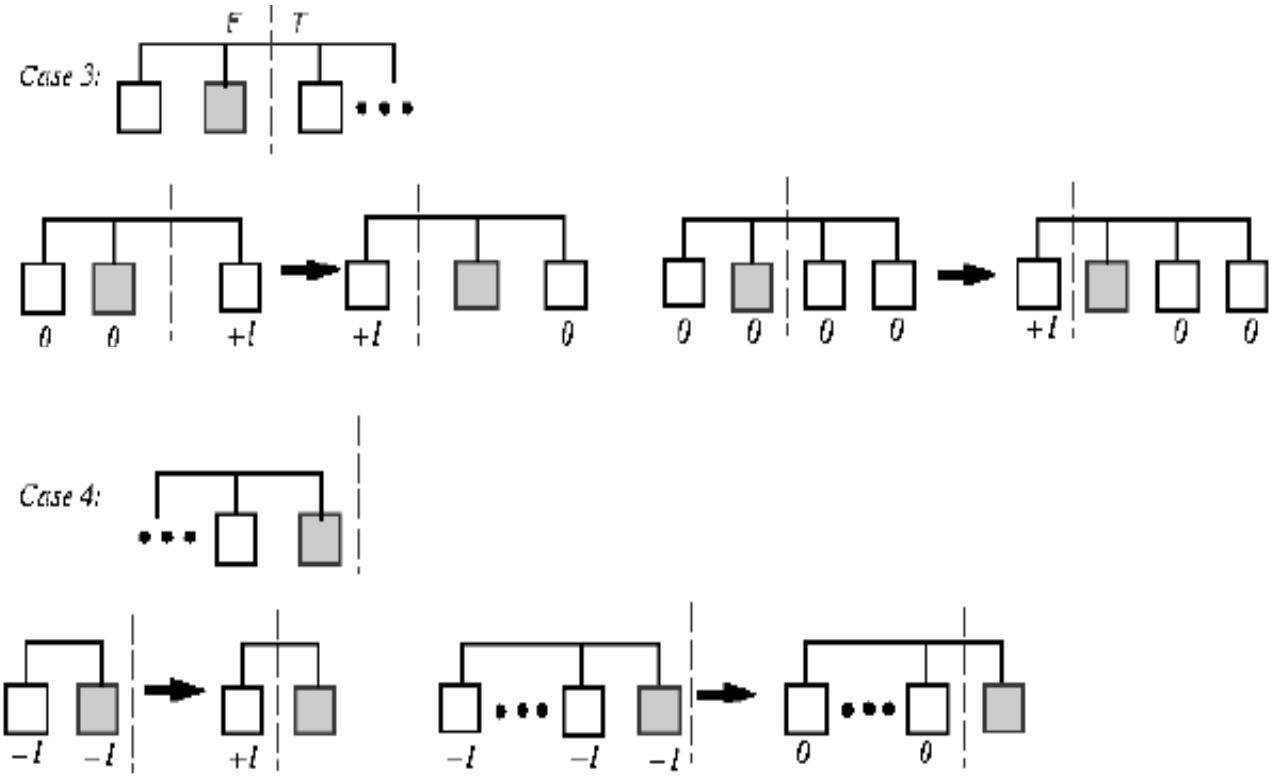
Other case is similar



Update Cell gains

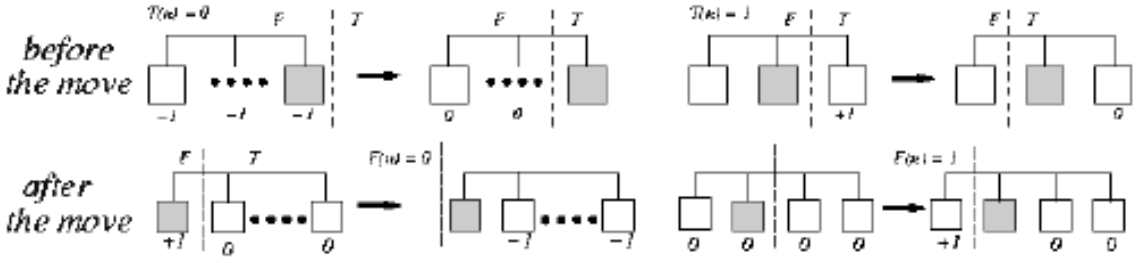


Update Cell gains



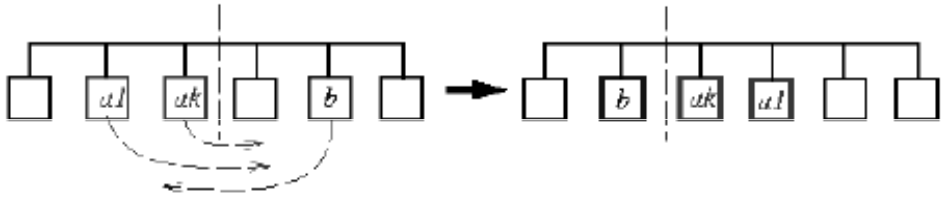
Algorithm : Update Cell gains

```
Algorithm: Update_Gain
1 begin  /* move base cells and update neighbors' gains */
2 F ← the From Block of the base cell;
3 T ← 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
     else if 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) ← F(n) - 1; T(n) ← T(n)+1;
     /* check for critical nets after the move */
8    if F(n)=0  then decrement gains of all free cells on n
     else if F(n) = 1 then increment gain of the only F cell on n,
     if it is free
9 end
```



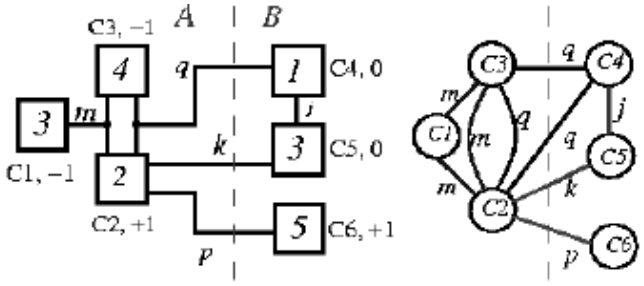
Complexity of updating the cell gains

- Once a net has “locked” cells at both sides, the net will remain cut from now on.
- Suppose we move a1, a2, …, ak from left to right, and then move b from right to left i.e, At most a1, a2, …, ak and b need updating!



- To update the cell gains, it takes O(n(i)) work for Net i., n(i): number of cells in net n
- Total time = n(1)+n(2)+…+n(N) = O(t).

Fiduccia-Mattheyses Heuristic: Example

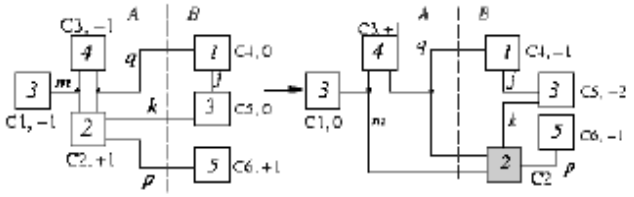


- Computing cell gains: $F(n) = 1 \Rightarrow g(i) + 1$; $T(n)=0 \Rightarrow g(i) - 1$

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: $r|V| - S_{max} \le |A| \le r|V| + S_{max}$. Let $r = 0.4 \Rightarrow |A| = 9$, $|V| = 18$, $S_{max} = 5$, $r|V| = 7.2 \Rightarrow$ Balanced: $2.2 \le 9 \le 12.2$!
- maximum gain: c_2 and balanced: $2.2 \le 9\text{-}2 \le 12.2 \Rightarrow$ Move c_2 from A to B (use size criterion if there is a tie).

Fiduccia-Mattheyses Heuristic: Example



- 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! $(2.2 \le 7\text{-}4 \le 12.2) \rightarrow$ Move c_3 from A to B (use size criterion if there is a tie).

Fiduccia-Mattheyses : Summary

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

- $\hat{g_1} = 1, \hat{g_2} = 1, \hat{g_3} = -1, \hat{g_4} = 1, \hat{g_5} = -2, \hat{g_6} = 0 \Rightarrow$ Maximum partial sum G_k = +2, k = 2 or 4.
- Since k=4 results in a better balanced partition $\Rightarrow$ Move c_1, c_2, c_3, $c_6 \Rightarrow A$={6}, B={1, 2, 3, 4, 5}.
- **Repeat the whole process until new $G_k \leq 0$.**