CS612
Algorithms for Electronic Design Automation

Partitioning

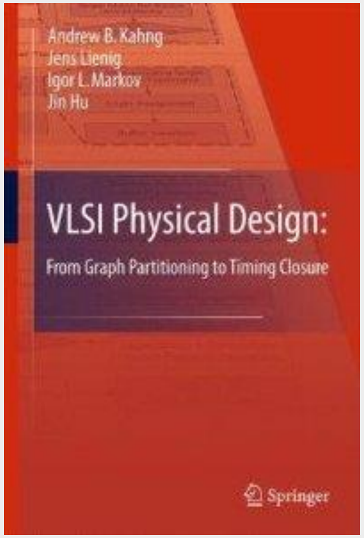
Mustafa Ozdal

MOST SLIDES ARE FROM THE BOOK:

VLSI Physical Design: From Graph Partitioning to Timing Closure

MODIFICATIONS WERE MADE ON THE ORIGINAL SLIDES

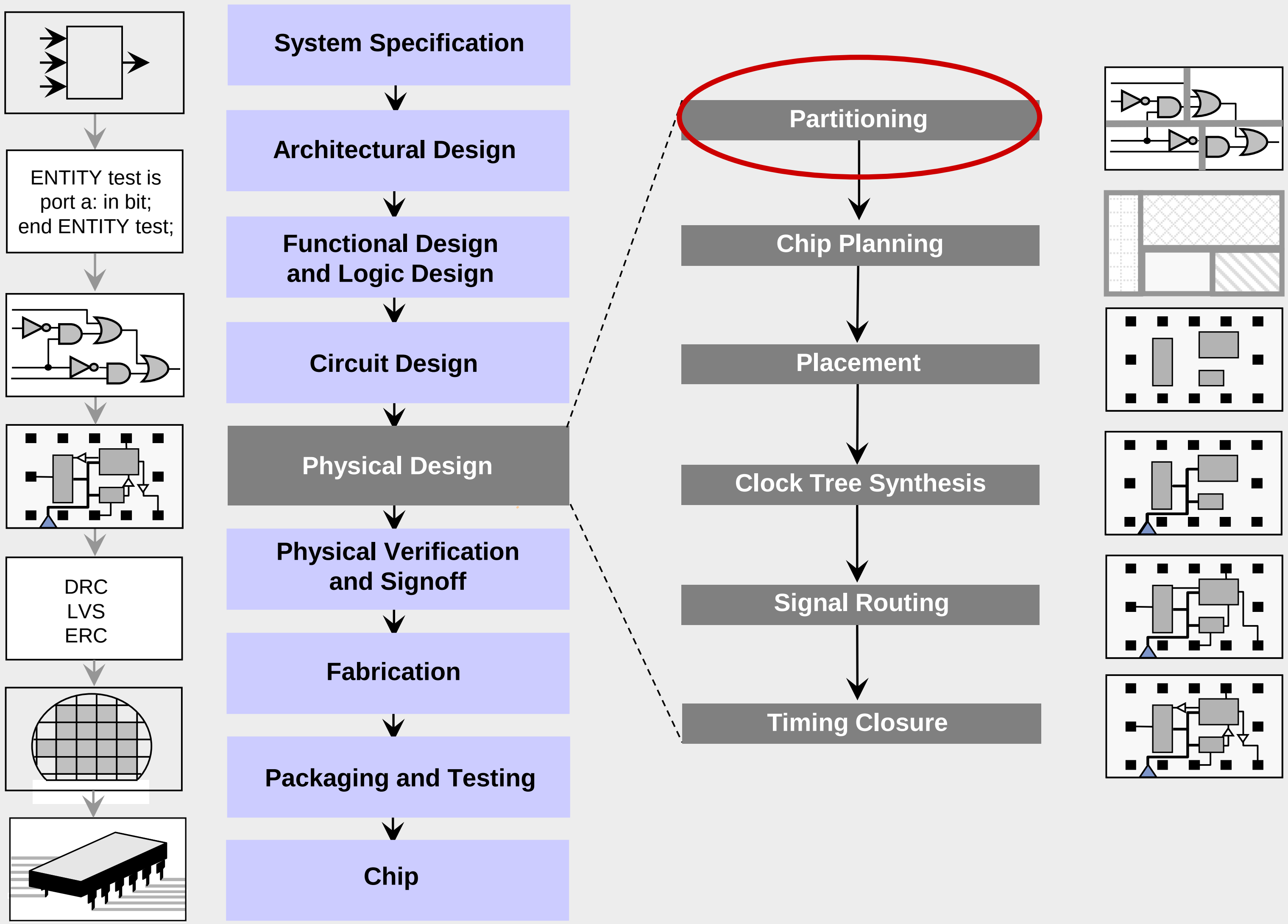
Chapter 2 – Netlist and System Partitioning



Original Authors:

Andrew B. Kahng, Jens Lienig, Igor L. Markov, Jin Hu

2.1 Introduction

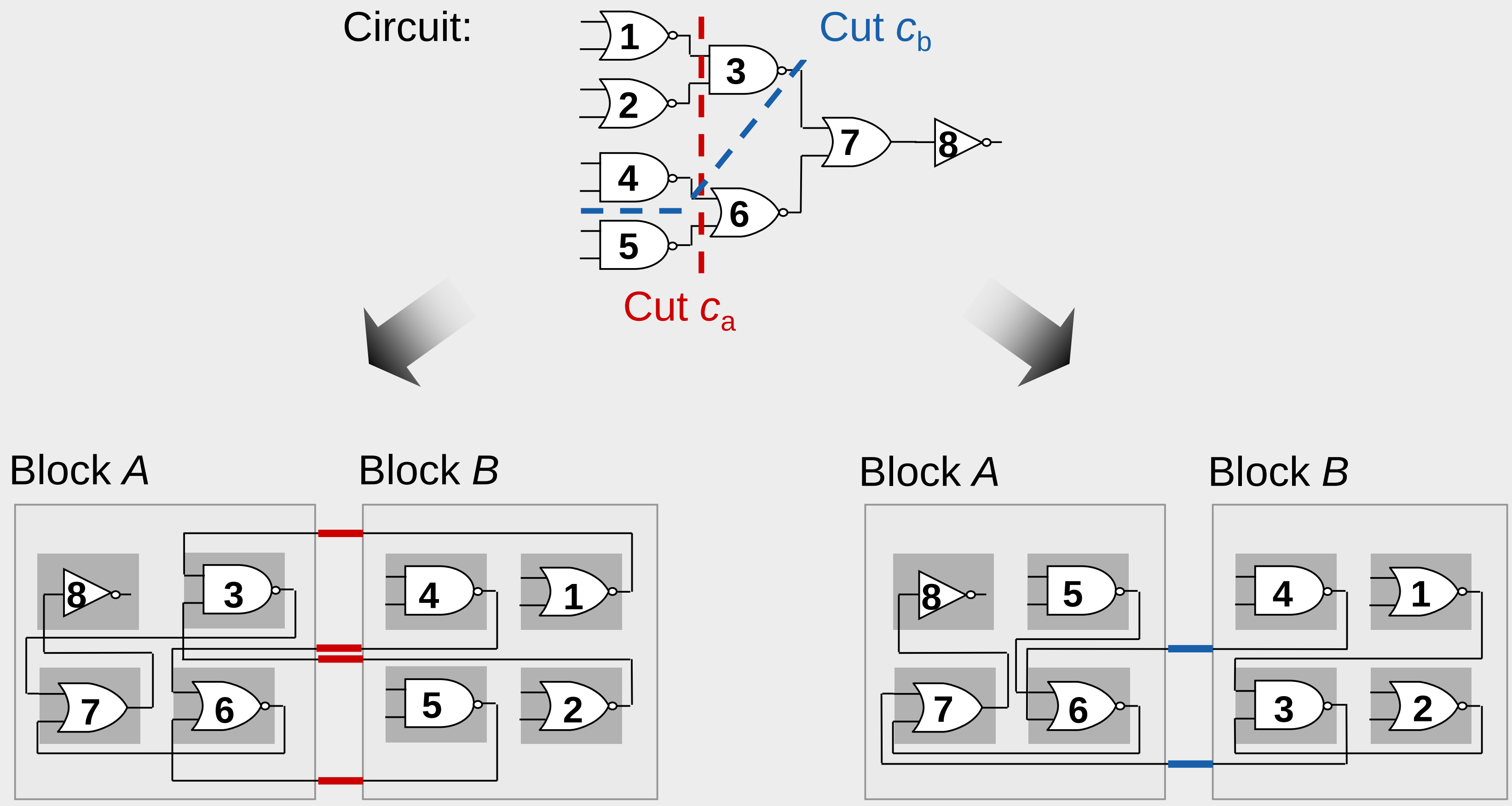


Divide and Conquer Strategy for Chip Design

- Partition the design. Then, work on each partition separately

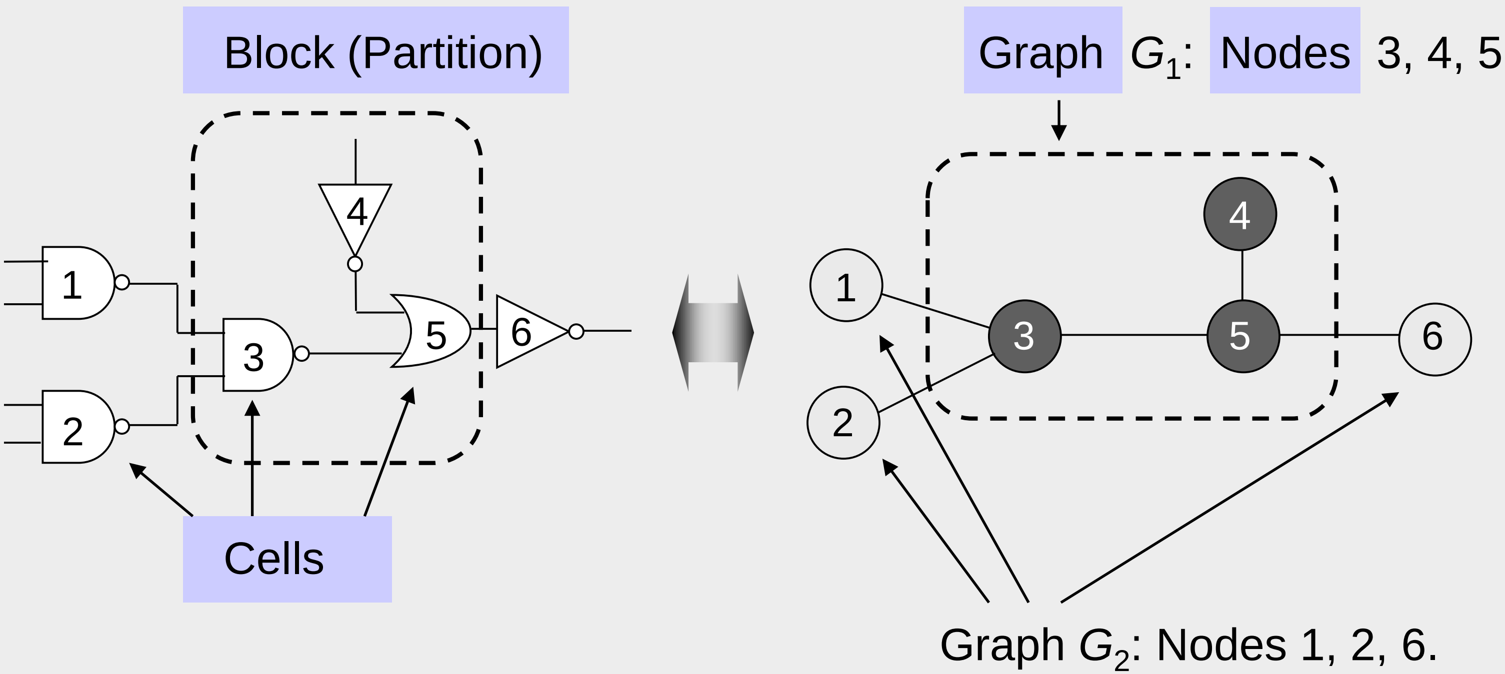
- Advantages:
  - Parallel implementation of each part by different designers
  - Tool capacity issues avoided

- Disadvantages:
  - Potentially less room for optimization
  - Inter-dependency between different partitions
  - Difficulty of combining partitions at the end



Cut c_a: four external connections

Cut c_b: two external connections



Collection of cut edges

Cut set: (1,3), (2,3), (5,6),

cut edge: connects nodes in different partitions

- Given a graph $G(V,E)$ with $|V|$ nodes and $|E|$ edges where each node $v \in V$ and each edge $e \in E$.

- Each node has area $s(v)$ and each edge has cost or weight $w(e)$.

- The objective is to divide the graph G into k disjoint subgraphs such that all optimization goals are achieved:
  - Number (or total weight) of connections between partitions is minimized
  - Partition sizes are balanced

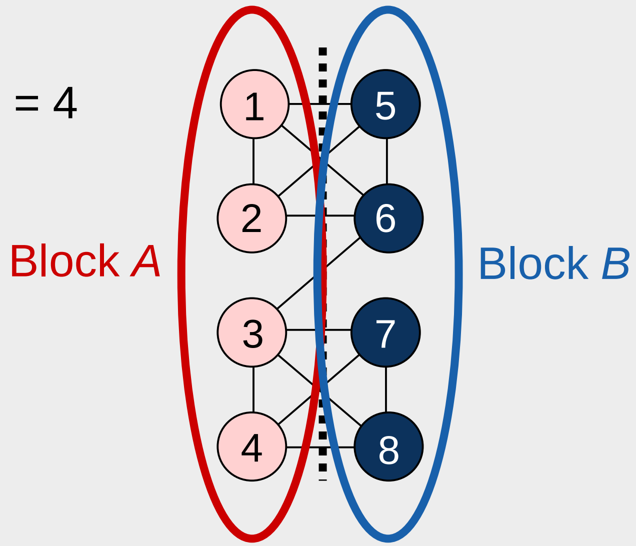
- NP-hard problem
- Efficient heuristics are developed in the 1970s and 1980s.
They are high quality and in low-order polynomial time.

2.4.1 Kernighan-Lin (KL) Algorithm

Given: A graph with $2n$ nodes where each node has the same weight.

Goal: A partition (division) of the graph into two disjoint subsets A and B with minimum cut cost and $|A| = |B| = n$.

Example: $n = 4$



2.4.1 Kernighan-Lin (KL) Algorithm – Terminology

Gain $D(v)$ of a node v

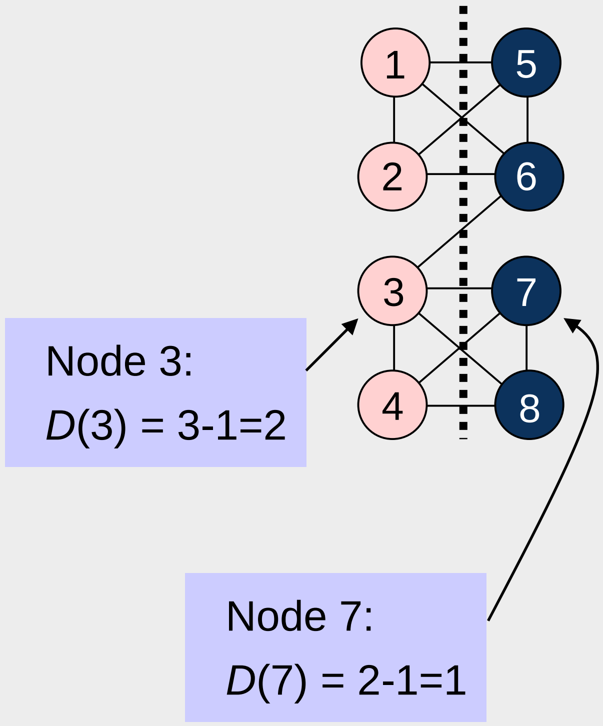
$$D(v) = |E_c(v)| - |E_{nc}(v)| \,,$$

where

$E_c(v)$ is the set of v's incident edges that are cut by the cut line, and

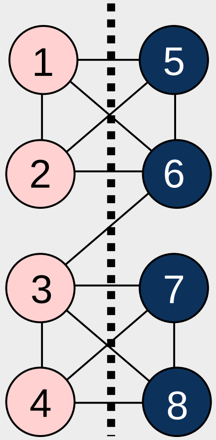
$E_{nc}(v)$ is the set of v's incident edges that are not cut by the cut line.

High gains ($D > 0$) indicate that the node should move, while low gains ($D < 0$) indicate that the node should stay within the same partition.



2.4.1 Kernighan-Lin (KL) Algorithm – Terminology

Gain of swapping a pair of nodes *a* and *b*



$\Delta g = D(a) + D(b) - 2 * c(a,b),$

where

- $D(a)$, $D(b)$ are the respective gains of nodes *a*, *b*
- $c(a,b)$ is the connection weight between *a* and *b*:
  If an edge exists between *a* and *b*,
  then $c(a,b)$ = edge weight (here 1),
  otherwise, $c(a,b) = 0$.

The gain Δg indicates how useful the swap between two nodes will be

The larger Δg, the more the total cut cost will be reduced

2.4.1 Kernighan-Lin (KL) Algorithm – Terminology

Gain of swapping a pair of nodes *a* und *b*

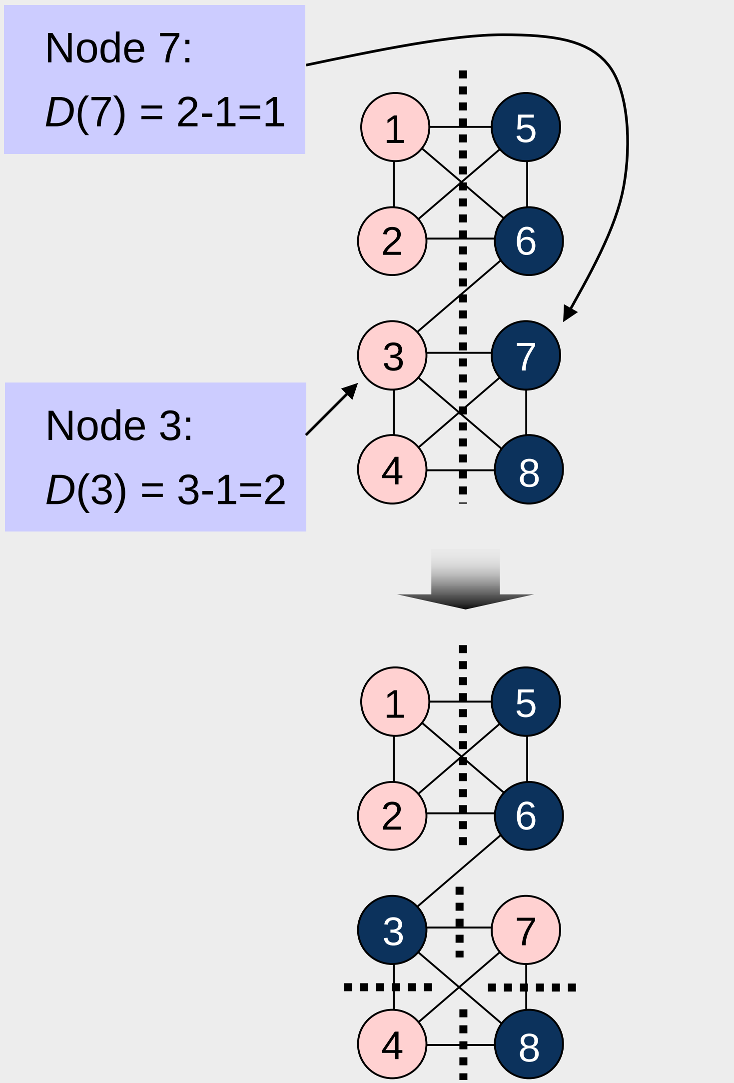
$$\Delta g = D(a) + D(b) - 2 * c(a,b),$$

where

- $D(a)$, $D(b)$ are the respective gains of nodes *a*, *b*
- $c(a,b)$ is the connection weight between *a* and *b*:
  If an edge exists between *a* and *b*,
  then $c(a,b)$ = edge weight (here 1),
  otherwise, $c(a,b) = 0$.

$$\Delta g\,(3,7) = D(3) + D(7) - 2 * c(a,b) = 2 + 1 - 2 = 1$$

=> Swapping nodes 3 and 7 would reduce the cut size by 1



2.4.1 Kernighan-Lin (KL) Algorithm – Terminology

Gain of swapping a pair of nodes *a* und *b*

$$\Delta g = D(a) + D(b) - 2 * c(a,b),$$

where

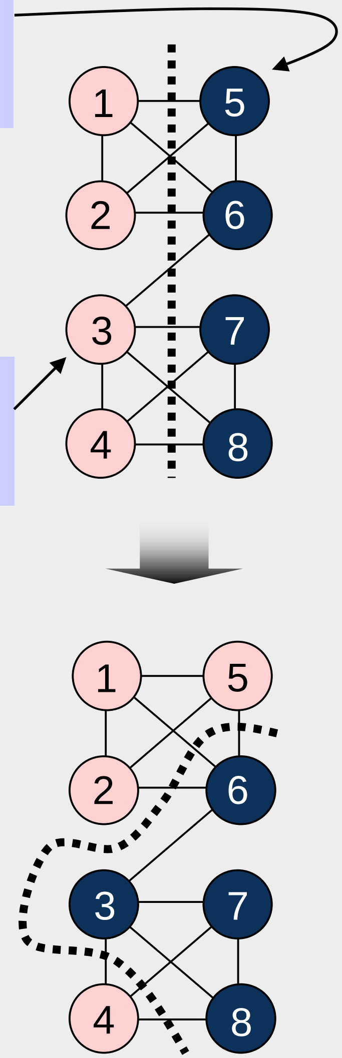
- $D(a)$, $D(b)$ are the respective gains of nodes *a*, *b*
- $c(a,b)$ is the connection weight between *a* and *b*:
  If an edge exists between *a* and *b*,
  then $c(a,b)$ = edge weight (here 1),
  otherwise, $c(a,b) = 0$.

$$\Delta g\,(3,5) = D(3) + D(5) - 2 * c(a,b) = 2 + 1 – 0 = 3$$

=> Swapping nodes 3 and 5 would reduce the cut size by 3

Node 5:
$D(5) = 2\text{-}1=1$

Node 3:
$D(3) = 3\text{-}1=2$



2.4.1 Kernighan-Lin (KL) Algorithm – Terminology

Gain of swapping a pair of nodes *a* and *b*

The goal is to find a pair of nodes *a* and *b* to exchange such that Δg is maximized and swap them.

Maximum positive gain G_m of a pass

The maximum positive gain G_m corresponds to the best prefix of *m* swaps within the swap sequence of a given pass.

These *m* swaps lead to the partition with the minimum cut cost encountered during the pass.

G_m is computed as the sum of Δg values over the first *m* swaps of the pass, with *m* chosen such that G_m is maximized.

$$G_m = \sum_{i=1}^{m} \Delta g_i$$

2.4.1 Kernighan-Lin (KL) Algorithm

Step 0:

– $V = 2n$ nodes
– $\{A, B\}$ is an initial arbitrary partitioning

Step 1:

– $i = 1$
– Compute $D(v)$ for all nodes $v \in V$

Step 2:

– Choose a_i and b_i such that $\Delta g_i = D(a_i) + D(b_i) - 2 * c(a_i b_i)$ is maximized
– Swap and fix a_i and b_i

Step 3:

– If all nodes are fixed, go to Step 4. Otherwise
– Compute and update D values for all nodes that are connected to a_i and b_i and are not fixed.
– $i = i + 1$
– Go to Step 2

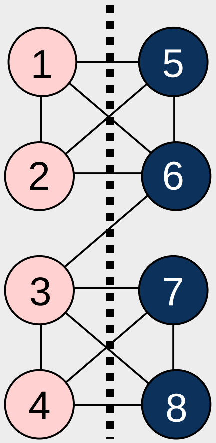
Step 4:

– Find the move sequence 1...m ($1 \le m \le i$), such that $G_m = \sum_{i=1}^{m} \Delta g_i$ is maximized
– If $G_m > 0$, go to Step 5. Otherwise, END

Step 5:

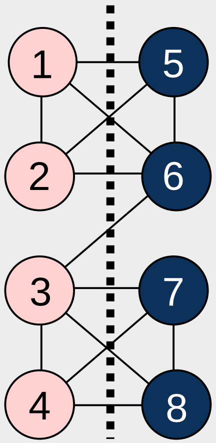
– Execute m swaps, reset remaining nodes
– Go to Step 1

2.4.1 Kernighan-Lin (KL) Algorithm – Example



Cut cost: 9
Not fixed:
1,2,3,4,5,6,7,8

2.4.1 Kernighan-Lin (KL) Algorithm – Example



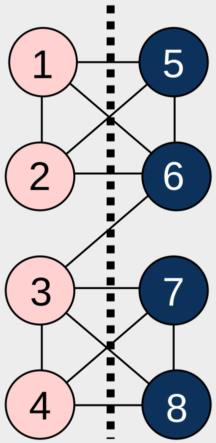
Cut cost: 9
Not fixed:
1,2,3,4,5,6,7,8

gains $D(v)$ of each node:

$D(1) = 1$	$D(5) = 1$
$D(2) = 1$	$D(6) = 2$
$D(3) = 2$	$D(7) = 1$
$D(4) = 1$	$D(8) = 1$

Nodes that lead to maximum gain

2.4.1 Kernighan-Lin (KL) Algorithm – Example



Cut cost: 9
Not fixed:
1,2,3,4,5,6,7,8

gains $D(v)$ of each node:

$D(1) = 1$ $\quad$ $\boldsymbol{D(5) = 1}$
$D(2) = 1$ $\quad$ $D(6) = 2$
$\boldsymbol{D(3) = 2}$ $\quad$ $D(7) = 1$
$D(4) = 1$ $\quad$ $D(8) = 1$

Nodes that lead to maximum gain

$\Delta g_1 = 2+1-0 = 3$

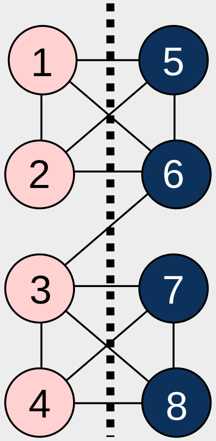
Gain after node swapping

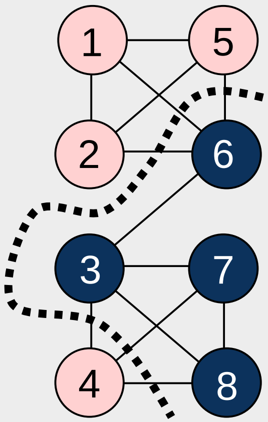
Swap (3,5)

$G_1 = \Delta g_1 = 3$

Gain in the current pass

2.4.1 Kernighan-Lin (KL) Algorithm – Example





Cut cost: 9
Not fixed:
1,2,3,4,5,6,7,8

$D(1) = 1$ **$D(5) = 1$**
$D(2) = 1$ $D(6) = 2$
$D(3) = 2$ $D(7) = 1$
$D(4) = 1$ $D(8) = 1$

Nodes that lead to maximum gain

$\Delta g_1 = 2+1-0 = 3$

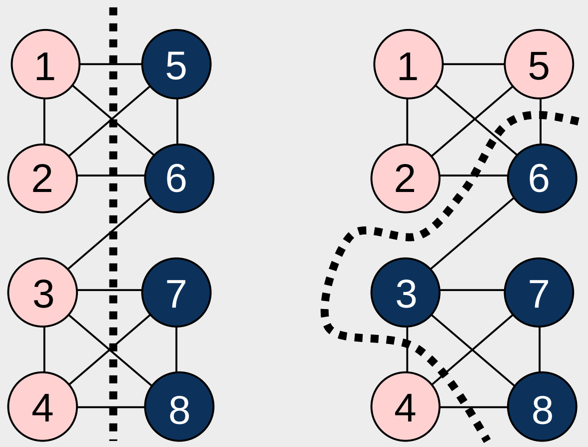
Gain after node swapping

Swap (3,5)

$G_1 = \Delta g_1 = 3$

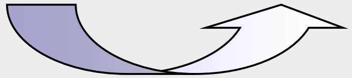
Gain in the current pass

2.4.1 Kernighan-Lin (KL) Algorithm – Example



Cut cost: 9
Not fixed:
1,2,3,4,5,6,7,8

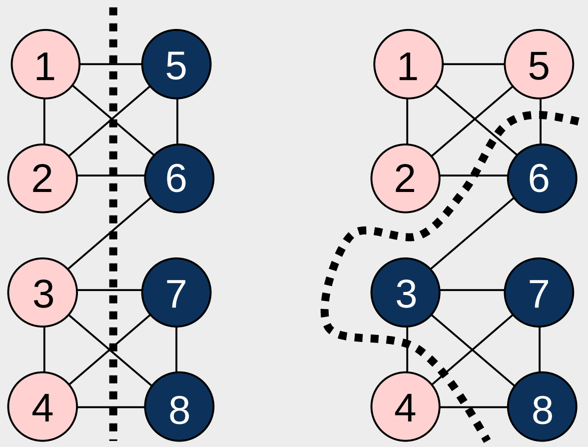
Cut cost: 6
Not fixed:
1,2,4,6,7,8



$D(1) = 1$	$\mathbf{D(5) = 1}$
$D(2) = 1$	$D(6) = 2$
$\mathbf{D(3) = 2}$	$D(7) = 1$
$D(4) = 1$	$D(8) = 1$

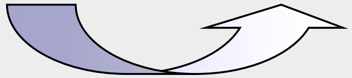
$\Delta g_1 = 2+1-0 = 3$
Swap (3,5)
$G_1 = \Delta g_1 = 3$

2.4.1 Kernighan-Lin (KL) Algorithm – Example



Cut cost: 9
Not fixed:
1,2,3,4,5,6,7,8

Cut cost: 6
Not fixed:
1,2,4,6,7,8

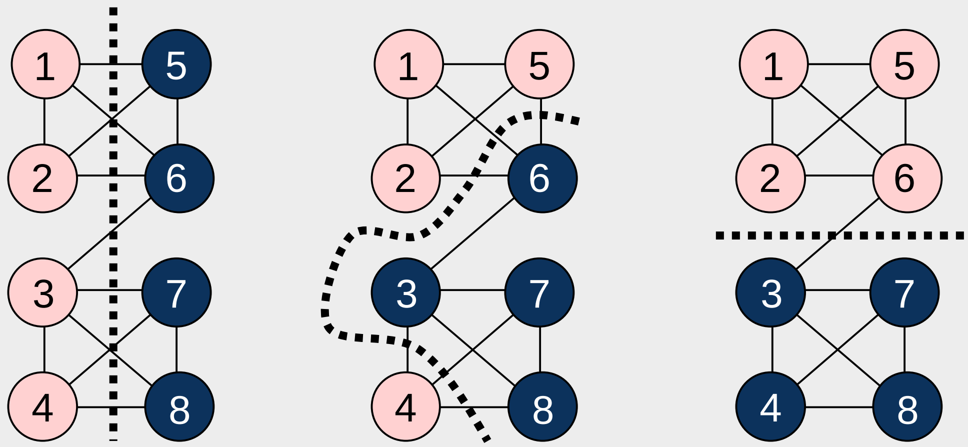


$D(1) = 1$ **$D(5) = 1$**
$D(2) = 1$ $D(6) = 2$
$D(3) = 2$ $D(7) = 1$
$D(4) = 1$ $D(8) = 1$

$\Delta g_1 = 2+1-0 = 3$
Swap (3,5)
$G_1 = \Delta g_1 = 3$

$D(1) = -1$ **$D(6) = 2$**
$D(2) = -1$ $D(7)=-1$
$D(4) = 3$ $D(8)=-1$

2.4.1 Kernighan-Lin (KL) Algorithm – Example



Cut cost: 9
Not fixed:
1,2,3,4,5,6,7,8

Cut cost: 6
Not fixed:
1,2,4,6,7,8

$D(1) = 1$ $\quad$ $\boldsymbol{D(5) = 1}$
$D(2) = 1$ $\quad$ $D(6) = 2$
$\boldsymbol{D(3) = 2}$ $\quad$ $D(7) = 1$
$D(4) = 1$ $\quad$ $D(8) = 1$

$\Delta g_1 = 2+1-0 = 3$
Swap (3,5)
$G_1 = \Delta g_1 = 3$

$D(1) = -1$ $\quad$ $\boldsymbol{D(6) = 2}$
$D(2) = -1$ $\quad$ $D(7)=-1$
$\boldsymbol{D(4) = 3}$ $\quad$ $D(8)=-1$

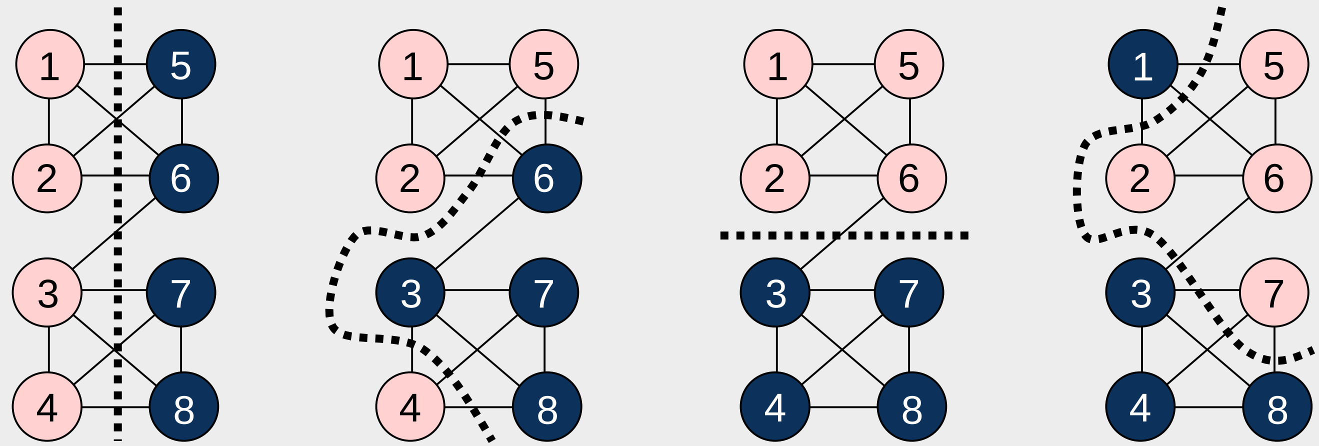
$\Delta g_2 = 3+2-0 = 5$
Swap (4,6)
$G_2 = G_1+\Delta g_2 = 8$

Nodes that lead to maximum gain

Gain after node swapping

Gain in the current pass

2.4.1 Kernighan-Lin (KL) Algorithm – Example



Cut cost: 9 Not fixed: 1,2,3,4,5,6,7,8	Cut cost: 6 Not fixed: 1,2,4,6,7,8	Cut cost: 1 Not fixed: 1,2,7,8	Cut cost: 7 Not fixed: 2,8

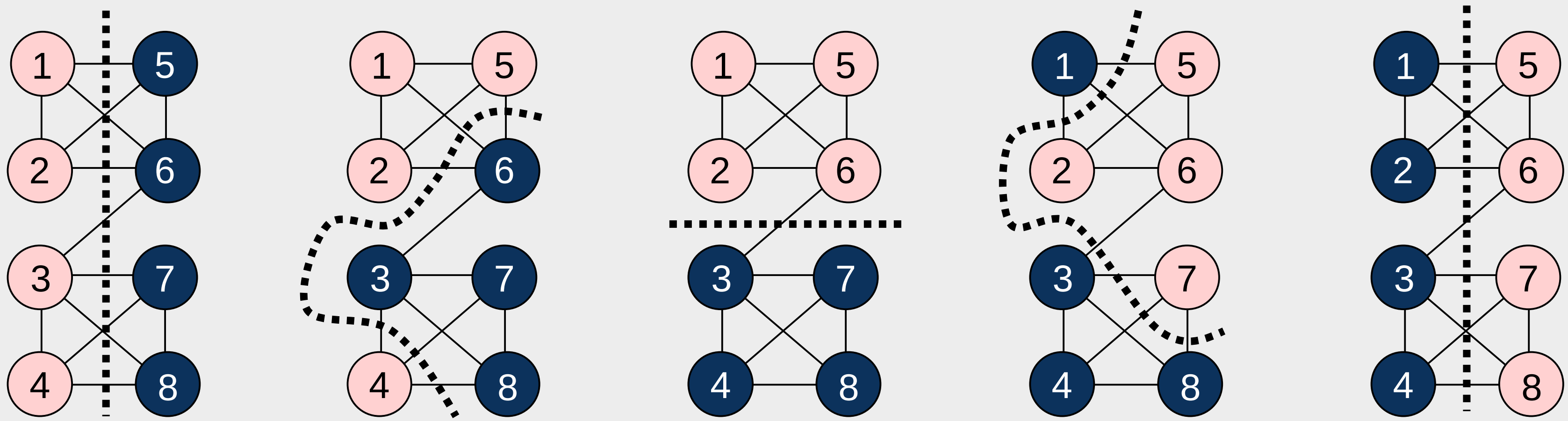
$D(1) = 1$ $\boldsymbol{D(5) = 1}$ $D(2) = 1$ $D(6) = 2$ $\boldsymbol{D(3) = 2}$ $D(7) = 1$ $D(4) = 1$ $D(8) = 1$ $\Delta g_1 = 2+1-0 = 3$ **Swap (3,5)** $G_1 = \Delta g_1 = 3$	$D(1) = -1$ $\boldsymbol{D(6) = 2}$ $D(2) = -1$ $D(7)=-1$ $\boldsymbol{D(4) = 3}$ $D(8)=-1$ $\Delta g_2 = 3+2-0 = 5$ **Swap (4,6)** $G_2 = G_1+\Delta g_2 = 8$	$\boldsymbol{D(1) = -3}$ $\boldsymbol{D(7)=-3}$ $D(2) = -3$ $D(8)=-3$ $\Delta g_3 = -3-3-0 = -6$ **Swap (1,7)** $G_3 = G_2 + \Delta g_3 = 2$

Nodes that lead to maximum gain

Gain after node swapping

Gain in the current pass

2.4.1 Kernighan-Lin (KL) Algorithm – Example



Cut cost: 9
Not fixed:
1,2,3,4,5,6,7,8

Cut cost: 6
Not fixed:
1,2,4,6,7,8

Cut cost: 1
Not fixed:
1,2,7,8

Cut cost: 7
Not fixed:
2,8

Cut cost: 9
Not fixed:
–

$D(1) = 1$ **$D(5) = 1$**
$D(2) = 1$ $D(6) = 2$
$D(3) = 2$ $D(7) = 1$
$D(4) = 1$ $D(8) = 1$

$\Delta g_1 = 2+1-0 = 3$
Swap (3,5)
$G_1 = \Delta g_1 = 3$

$D(1) = -1$ **$D(6) = 2$**
$D(2) = -1$ $D(7) = -1$
$D(4) = 3$ $D(8) = -1$

$\Delta g_2 = 3+2-0 = 5$
Swap (4,6)
$G_2 = G_1 + \Delta g_2 = 8$

$D(1) = -3$ **$D(7) = -3$**
$D(2) = -3$ $D(8) = -3$

$\Delta g_3 = -3-3-0 = -6$
Swap (1,7)
$G_3 = G_2 + \Delta g_3 = 2$

$D(2) = -1$ **$D(8) = -1$**

$\Delta g_4 = -1-1-0 = -2$
Swap (2,8)
$G_4 = G_3 + \Delta g_4 = 0$

2.4.1 Kernighan-Lin (KL) Algorithm – Example

$D(1) = 1$ $D(2) = 1$ $\mathbf{D(3) = 2}$ $D(4) = 1$	$D(1) = -1$ $D(2) = -1$ $\mathbf{D(4) = 3}$	$\mathbf{D(1) = -3}$ $D(2) = -3$	$\mathbf{D(2) = -1}$
$\mathbf{D(5) = 1}$ $D(6) = 2$ $D(7) = 1$ $D(8) = 1$	$\mathbf{D(6) = 2}$ $D(7)=-1$ $D(8)=-1$	$\mathbf{D(7)=-3}$ $D(8)=-3$	$\mathbf{D(8)=-1}$
$\Delta g_1 = 2+1-0 = 3$ **Swap (3,5)** $G_1 = \Delta g_1 = 3$	$\Delta g_2 = 3+2-0 = 5$ **Swap (4,6)** $G_2 = G_1 + \Delta g_2 = 8$	$\Delta g_3 = -3-3-0 = -6$ **Swap (1,7)** $G_3 = G_2 + \Delta g_3 = 2$	$\Delta g_4 = -1-1-0 = -2$ **Swap (2,8)** $G_4 = G_3 + \Delta g_4 = 0$

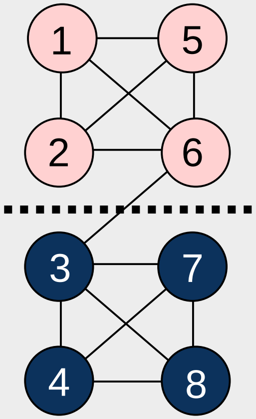
Maximum positive gain $G_m = 8$ with $m = 2$.

2.4.1 Kernighan-Lin (KL) Algorithm – Example

$D(1) = 1$ **$D(5) = 1$** $D(2) = 1$ $D(6) = 2$ **$D(3) = 2$** $D(7) = 1$ $D(4) = 1$ $D(8) = 1$ $\Delta g_1 = 2+1-0 = 3$ **Swap (3,5)** $G_1 = \Delta g_1 = 3$	$D(1) = -1$ **$D(6) = 2$** $D(2) = -1$ $D(7) = -1$ **$D(4) = 3$** $D(8) = -1$ $\Delta g_2 = 3+2-0 = 5$ **Swap (4,6)** $G_2 = G_1 + \Delta g_2 = 8$	**$D(1) = -3$** **$D(7) = -3$** $D(2) = -3$ $D(8) = -3$ $\Delta g_3 = -3-3-0 = -6$ **Swap (1,7)** $G_3 = G_2 + \Delta g_3 = 2$	**$D(2) = -1$** **$D(8) = -1$** $\Delta g_4 = -1-1-0 = -2$ **Swap (2,8)** $G_4 = G_3 + \Delta g_4 = 0$

Maximum positive gain $G_m = 8$ with $m = 2$.

Since $G_m > 0$, the first $m = 2$ swaps (3,5) and (4,6) are executed.



Since $G_m > 0$, more passes are needed until $G_m \leq 0$.

n: # of nodes in each partition

n times

Step 0:
- $V = 2n$ nodes → O(n)
- $\{A, B\}$ is an initial arbitrary partitioning

Step 1:
- $i = 1$
- Compute $D(v)$ for all nodes $v \in V$ → O(n²)

Step 2:
- Choose a_i and b_i such that $\Delta g_i = D(a_i) + D(b_i) - 2 * c(a_i b_i)$ is maximized → O(n²)
- Swap and fix a_i and b_i

Step 3:
- If all nodes are fixed, go to Step 4. Otherwise
- Compute and update D values for all nodes that are connected to a_i and b_i and are not fixed.
- $i = i + 1$
- Go to Step 2 → O(n)

Step 4:
- Find the move sequence 1...m ($1 \le m \le i$), such that $G_m = \sum_{i=1}^{m} \Delta g_i$ is maximized → O(n)
- If $G_m > 0$, go to Step 5. Otherwise, END

Step 5:
- Execute m swaps, reset remaining nodes
- Go to Step 1

1-pass runtime: O(n^3)
Optimized implementation: O(n^2lgn)

2.4.2 Extensions for Kernighan-Lin (KL) Algorithm

- Unequal partition sizes

- Unequal cell sizes or unequal node weights

- *k*-way partitioning (generating *k* partitions)

2.4.3 Fiduccia-Mattheyses (FM) Algorithm

- Single cells are moved independently instead of swapping pairs of cells. Thus, this algorithm is applicable to partitions of unequal size or the presence of initially fixed cells.
- Cut costs are extended to include hypergraphs, i.e., nets with two or more pins. While the KL algorithm aims to minimize cut costs based on edges, the FM algorithm minimizes cut costs based on nets.
- The area of each individual cell is taken into account.
- Nodes and subgraphs are referred to as *cells* and *blocks*, respectively.

2.4.3 Fiduccia-Mattheyses (FM) Algorithm

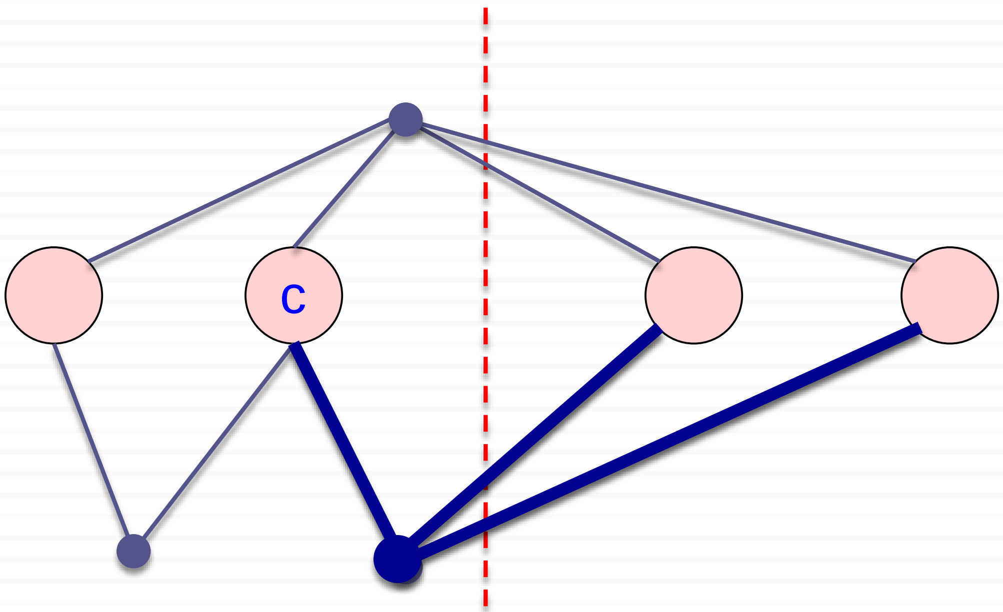
Given: a graph $G(V,E)$ with nodes and *weighted* edges

Goal: to assign all nodes to disjoint partitions, so as to minimize the total cost (weight) of all cut nets while satisfying partition size constraints

Fiduccia-Mattheyses (FM) Algorithm – Terminology

FS(c): the # of cut nets connected only to cell c in c's partition

"*moving force*"

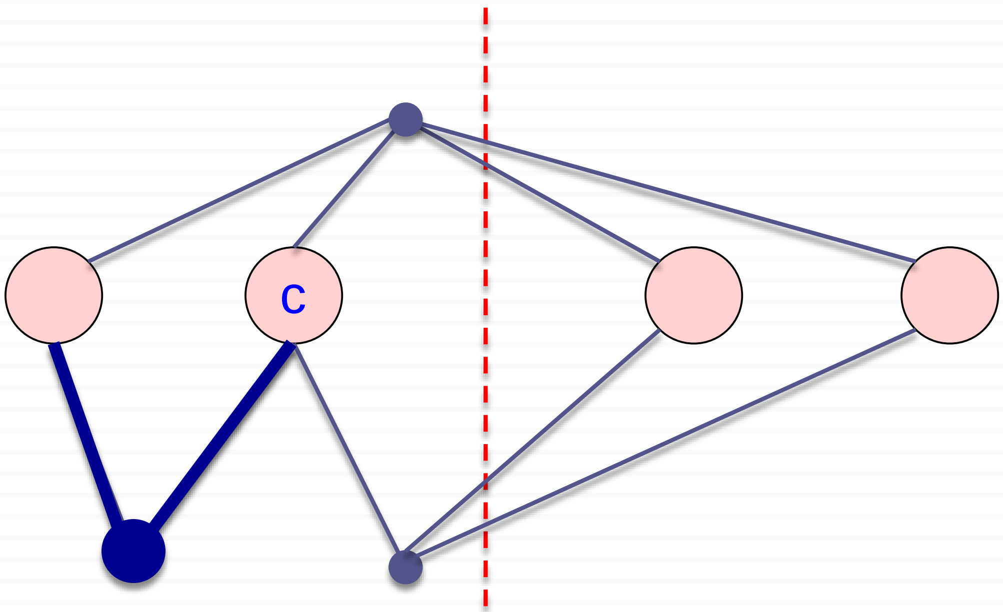


FS(c) = 1

Fiduccia-Mattheyses (FM) Algorithm – Terminology

TE(c): the # of uncut nets connected to cell c

"*retention force*"

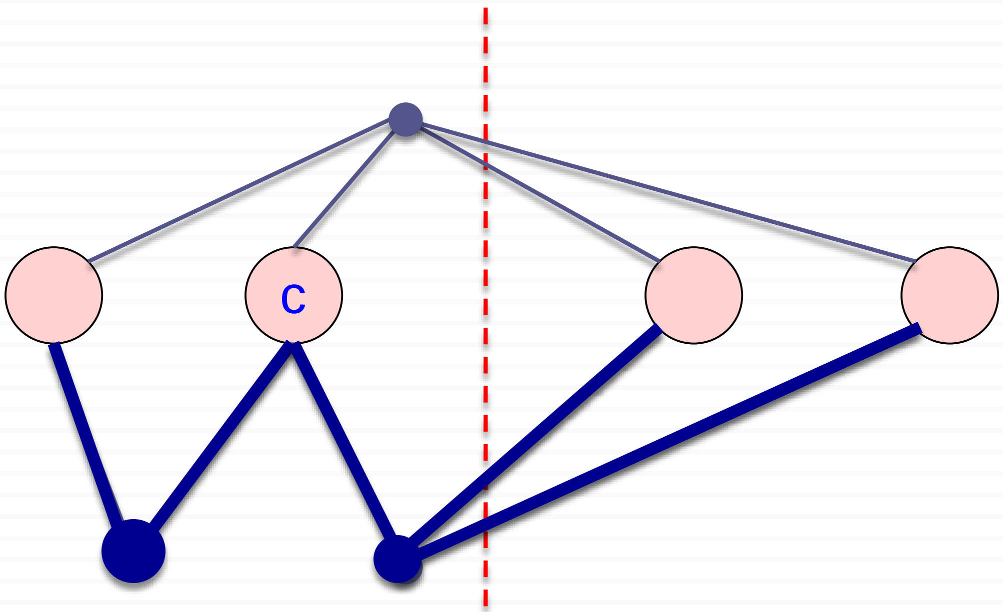


TE(c) = 1

Fiduccia-Mattheyses (FM) Algorithm – Terminology

$\Delta g(c) = FS(c) - TE(c)$

"gain of moving c"



$\Delta g(c) = 0$

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Terminology

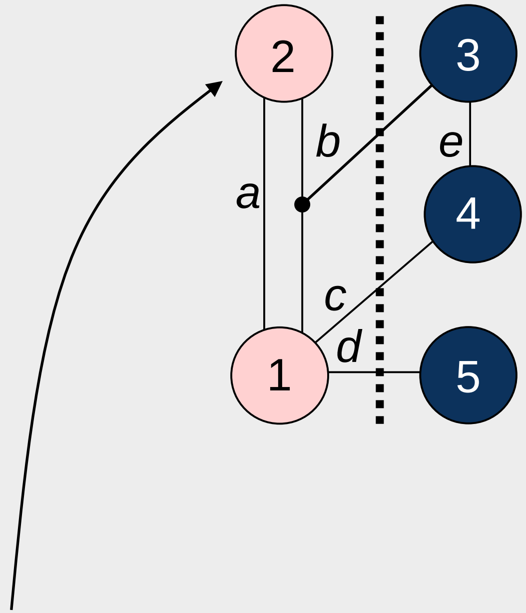
Gain $\Delta g(c)$ for cell c

$\Delta g(c) = FS(c) - TE(c)$,

where

the “moving force“ $FS(c)$ is the number of nets connected to c but not connected to any other cells within c’s partition, i.e., cut nets that connect only to c, and

the “retention force“ $TE(c)$ is the number of *uncut* nets connected to c.



Cell 2: $FS(2) = 0$ $TE(2) = 1$ $\Delta g(2) = -1$

The higher the gain $\Delta g(c)$, the higher is the priority to move the cell c to the other partition.

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Terminology

Gain $\Delta g(c)$ for cell c

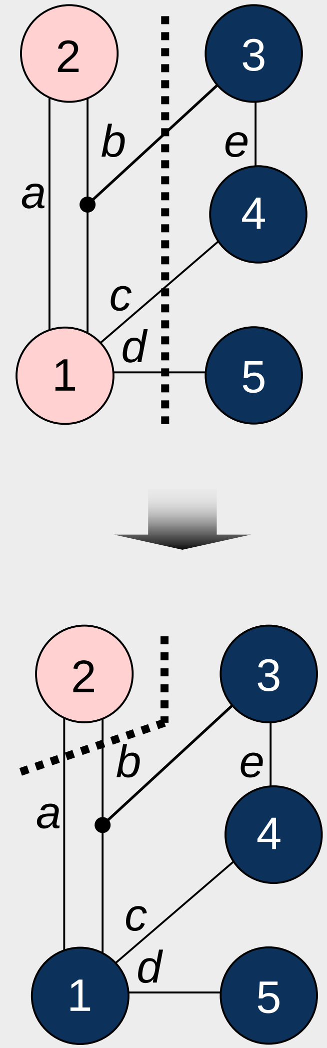
$$\Delta g(c) = FS(c) - TE(c) ,$$

where

the “moving force“ $FS(c)$ is the number of nets connected to c but not connected to any other cells within c’s partition, i.e., cut nets that connect only to c, and

the “retention force“ $TE(c)$ is the number of *uncut* nets connected to c.

Cell 1:	$FS(1) = 2$	$TE(1) = 1$	$\Delta g(1) = 1$
Cell 2:	$FS(2) = 0$	$TE(2) = 1$	$\Delta g(2) = -1$
Cell 3:	$FS(3) = 1$	$TE(3) = 1$	$\Delta g(3) = 0$
Cell 4:	$FS(4) = 1$	$TE(4) = 1$	$\Delta g(4) = 0$
Cell 5:	$FS(5) = 1$	$TE(5) = 0$	$\Delta g(5) = 1$



Maximum positive gain G_m of a pass

The maximum positive gain G_m is the cumulative cell gain of *m* moves that produce a minimum cut cost.

G_m is determined by the maximum sum of cell gains Δg over a prefix of *m* moves in a pass

$$G_m = \sum_{i=1}^{m} \Delta g_i$$

Ratio factor

The *ratio factor* is the relative balance between the two partitions with respect to cell area.

It is used to prevent all cells from clustering into one partition.

The ratio factor r is defined as

$$r = \frac{area(A)}{area(A) + area(B)}$$

where $area(A)$ and $area(B)$ are the total respective areas of partitions A and B

Balance criterion

The balance criterion enforces the ratio factor.

To ensure feasibility, the maximum cell area $area_{max}(V)$ must be taken into account.

A partitioning of V into two partitions A and B is said to be balanced if

$$[\ r \cdot area(V) - area_{max}(V)\ ] \leq area(A) \leq [\ r \cdot area(V) + area_{max}(V)\ ]$$

Base cell

A base cell is a cell c that has maximum cell gain $\Delta g(c)$ among all free cells, and whose move does not violate the balance criterion.

Base cell

Cell 1:	FS(1) = 2	TE(1) = 1	Δg(1) = 1
Cell 2:	$FS(2) = 0$	$TE(2) = 1$	$\Delta g(2) = -1$
Cell 3:	$FS(3) = 1$	$TE(3) = 1$	$\Delta g(3) = 0$
Cell 4:	$FS(4) = 1$	$TE(4) = 1$	$\Delta g(4) = 0$

Fiduccia-Mattheyses (FM) Algorithm – Terminology

- Pin distribution of a *net*: The pair (A(net), B(net))

  where A(net): # of pins in partition A

  B(net): # of pins in partition B

- Critical net: A *net* that contains a cell c whose move *changes the cut state* of the net
  - Either: The net has 1 cell in partition A and all others in B
  - Or: The net has all cells in a single partition
  - For a critical cell, one of the following should hold:

    A(net) = 0 or A(net) = 1 or B(net) = 0 or B(net) = 1

Fiduccia-Mattheyses (FM) Algorithm – Critical Nets

- Critical nets simplify the gain calculations
  - Only the cells belonging to critical nets need to be considered for gain calculations

2.4.3 Fiduccia-Mattheyses (FM) Algorithm

Step 0: Compute the balance criterion

Step 1: Compute the cell gain Δg_1 of each cell

Step 2: $i = 1$

– Choose base cell c_1 that has maximal gain Δg_1, move this cell

Step 3:

– Fix the base cell c_i
– Update all cells' gains that are connected to critical nets via the base cell c_i

Step 4:

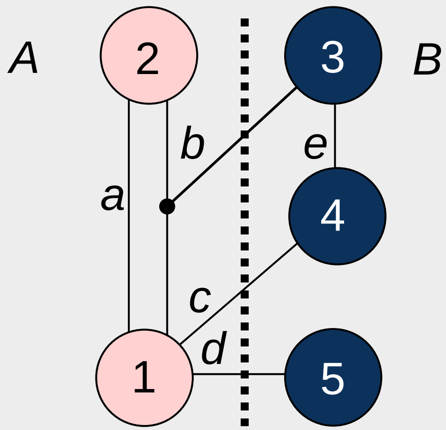
– If all cells are fixed, go to Step 5. If not:
– Choose next base cell c_i with maximal gain Δg_i and move this cell
– $i = i + 1$, go to Step 3

Step 5:

– Determine the best move sequence $c_1, c_2, .., c_m$ $(1 \leq m \leq i)$, so that $G_m = \sum_{i=1}^{m} \Delta g_i$ is maximized
– If $G_m > 0$, go to Step 6. Otherwise, END

Step 6:

– Execute m moves, reset all fixed nodes
– Start with a new pass, go to Step 1



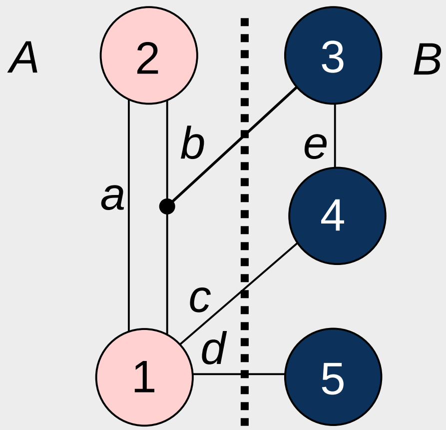
Given:
Ratio factor r = 0,375
area(Cell_1) = 2
area(Cell_2) = 4
area(Cell_3) = 1
area(Cell_4) = 4
area(Cell_5) = 5.

Step 0: Compute the balance criterion

$$[\, r \cdot area(V) - area_{max}(V) \,] \le area(A) \le [\, r \cdot area(V) + area_{max}(V) \,]$$

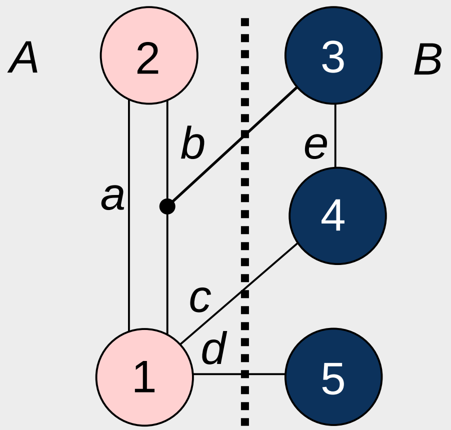
0,375 * 16 – 5 = $1 \le area(A) \le 11$ = 0,375 * 16 +5.

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Example



Step 1: Compute the gains of each cell

Cell 1:	$FS(\text{Cell\_1}) = 2$	$TE(\text{Cell\_1}) = 1$	$\Delta g(\text{Cell\_1}) = 1$
Cell 2:	$FS(\text{Cell\_2}) = 0$	$TE(\text{Cell\_2}) = 1$	$\Delta g(\text{Cell\_2}) = -1$
Cell 3:	$FS(\text{Cell\_3}) = 1$	$TE(\text{Cell\_3}) = 1$	$\Delta g(\text{Cell\_3}) = 0$
Cell 4:	$FS(\text{Cell\_4}) = 1$	$TE(\text{Cell\_4}) = 1$	$\Delta g(\text{Cell\_4}) = 0$
Cell 5:	$FS(\text{Cell\_5}) = 1$	$TE(\text{Cell\_5}) = 0$	$\Delta g(\text{Cell\_5}) = 1$



Cell1:	*FS*(Cell_1) = 2	*TE*(Cell_1) = 1	Δg(Cell_1) = 1
Cell 2:	*FS*(Cell_2) = 0	*TE*(Cell_2) = 1	Δg(Cell_2) = -1
Cell 3:	*FS*(Cell_3) = 1	*TE*(Cell_3) = 1	Δg(Cell_3) = 0
Cell 4:	*FS*(Cell_4) = 1	*TE*(Cell_4) = 1	Δg(Cell_4) = 0
Cell 5:	*FS*(Cell_5) = 1	*TE*(Cell_5) = 0	Δg(Cell_5) = 1

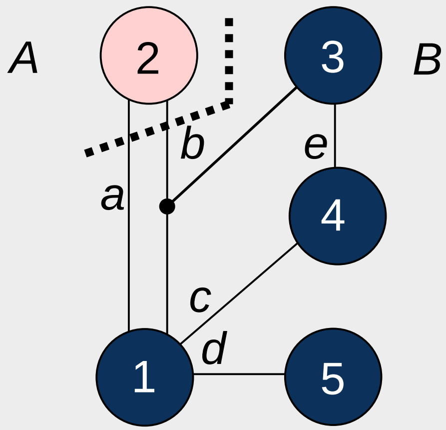
Step 2: Select the base cell

Possible base cells are Cell 1 and Cell 5

Balance criterion after moving Cell 1: *area*(*A*) = *area*(Cell_2) = 4

Balance criterion after moving Cell 5: *area*(*A*) = *area*(*Cell_1*) + *area*(*Cell_2*) + *area*(*Cell_5*) = 11

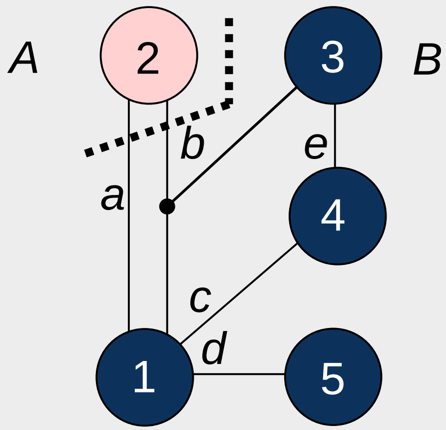
Both moves respect the balance criterion, but Cell 1 is selected, moved, and fixed as a result of the tie-breaking criterion.



Step 3: Fix base cell, update Δg values

Cell 2:	FS(Cell_2) = 2	TE(Cell_2) = 0	Δg(Cell_2) = 2
Cell 3:	FS(Cell_3) = 0	TE(Cell_3) = 1	Δg(Cell_3) = -1
Cell 4:	FS(Cell_4) = 0	TE(Cell_4) = 2	Δg(Cell_4) = -2
Cell 5:	FS(Cell_5) = 0	TE(Cell_5) = 1	Δg(Cell_5) = -1

After Iteration i = 1: Partition $A_1 = \{2\}$, Partition $B_1 = \{1,3,4,5\}$, with fixed cell $\{1\}$.



Iteration $i = 1$

Cell 2:	FS(Cell_2) = 2	TE(Cell_2) = 0	Δg(Cell_2) = 2
Cell 3:	FS(Cell_3) = 0	TE(Cell_3) = 1	Δg(Cell_3) = -1
Cell 4:	FS(Cell_4) = 0	TE(Cell_4) = 2	Δg(Cell_4) = -2
Cell 5:	FS(Cell_5) = 0	TE(Cell_5) = 1	Δg(Cell_5) = -1

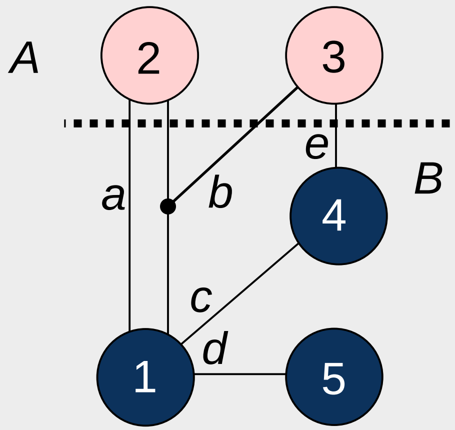
Iteration $i = 2$

Cell *2* has maximum gain $\Delta g_2 = 2$, $area(A) = 0$, balance criterion is violated.

Cell *3* has next maximum gain $\Delta g_2 = -1$, $area(A) = 5$, balance criterion is met.

Cell *5* has next maximum gain $\Delta g_2 = -1$, $area(A) = 9$, balance criterion is met.

Move cell *3*, updated partitions: $A_2 = \{2,3\}$, $B_2 = \{1,4,5\}$, with fixed cells {1,3}



Iteration i = 2

Cell 2: $\Delta g(\text{Cell\_2}) = 1$

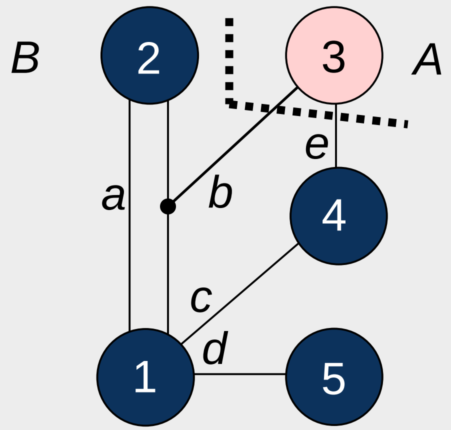
Cell 4: $\Delta g(\text{Cell\_4}) = 0$

Cell 5: $\Delta g(\text{Cell\_5}) = -1$

Iteration i = 3

Cell *2* has maximum gain $\Delta g_3 = 1$, $area(A) = 1$, balance criterion is met.

Move cell *2*, updated partitions: $A_3 = \{3\}$, $B_3 = \{1,2,4,5\}$, with fixed cells {1,2,3}



Iteration i = 3

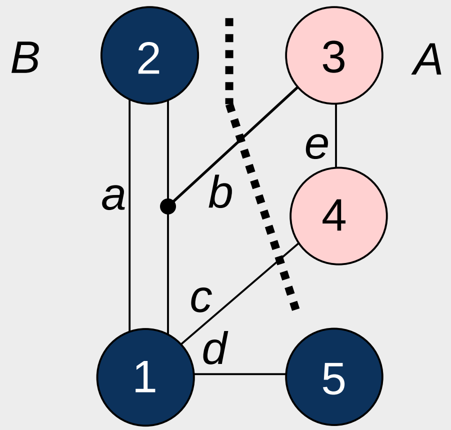
Cell 4: Δg(Cell_4) = 0

Cell 5: Δg(Cell_5) = -1

Iteration i = 4

Cell 4 has maximum gain $\Delta g_4 = 0$, $area(A) = 5$, balance criterion is met.

Move cell 4, updated partitions: $A_4 = \{3,4\}$, $B_3 = \{1,2,5\}$, with fixed cells {1,2,3,4}



Iteration i = 4

Cell 5: Δg(Cell_5) = -1

Iteration i = 5

Cell 5 has maximum gain Δg_5 = -1, *area*(*A*) = 10, balance criterion is met.

Move cell 5, updated partitions: A_4 = {3,4,5}, B_3 = {1,2}, all cells {1,2,3,4,5} fixed.

Step 5: Find best move sequence $c_1 \ldots c_m$

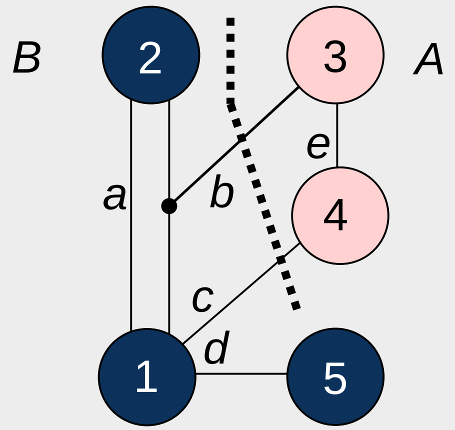
$G_1 = \Delta g_1 = 1$

$G_2 = \Delta g_1 + \Delta g_2 = 0$

$G_3 = \Delta g_1 + \Delta g_2 + \Delta g_3 = 1$

$G_4 = \Delta g_1 + \Delta g_2 + \Delta g_3 + \Delta g_4 = 1$

$G_5 = \Delta g_1 + \Delta g_2 + \Delta g_3 + \Delta g_4 + \Delta g_5 = 0.$



Maximum positive cumulative gain $G_m = \sum_{i=1}^{m} \Delta g_i = 1$

found in iterations 1, 3 and 4.

The move prefix m = 4 is selected due to the better balance ratio (*area*(*A*) = 5); the four cells 1, 2, 3 and 4 are then moved.

Result of Pass 1: Current partitions: *A* = {3,4}, *B* = {*1,2,5*}, cut cost reduced from 3 to 2.

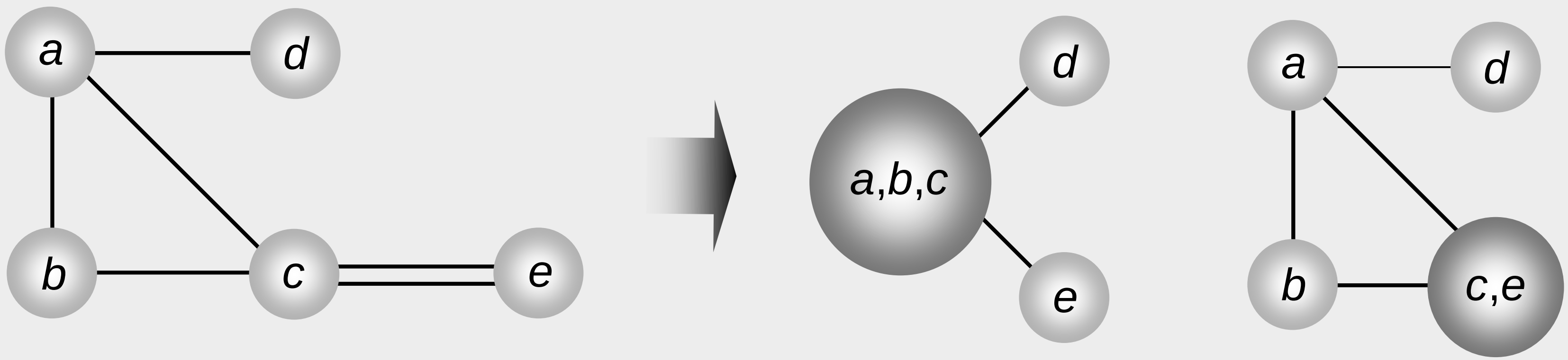
Chapter 2 Supplemental: Difference between KL & FM

- Component dependency of partitioning algorithms
  - KL is based on the number of edges
  - FM is based on the number of nets

- Time complexity of partitioning algorithms
  - KL has cubic time complexity
  - FM has linear time complexity

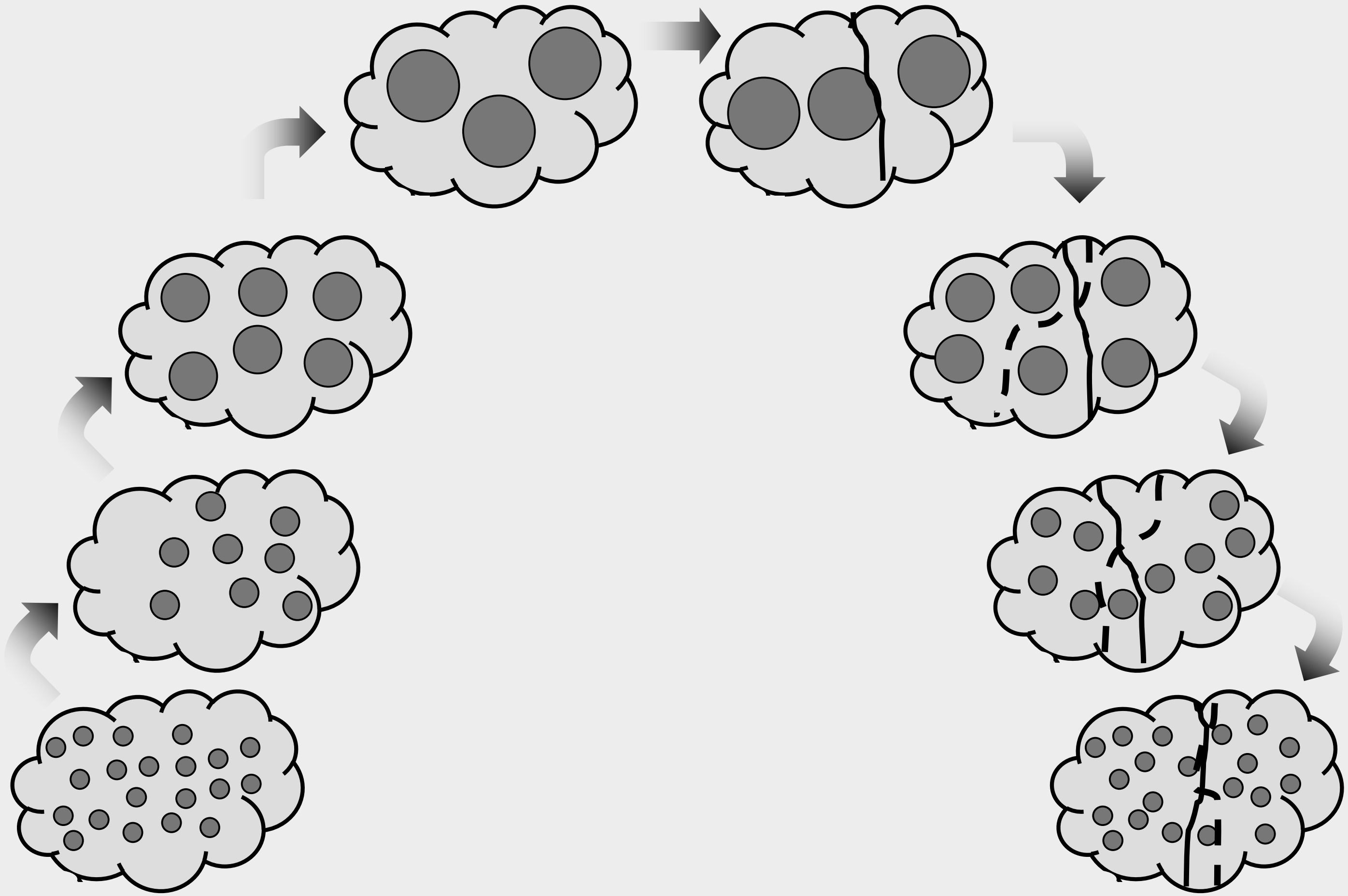
2.5.1 Clustering

- To make things easy, groups of tightly-connected nodes can be clustered, absorbing connections between these nodes
- Size of each cluster is often limited so as to prevent degenerate clustering, i.e. a single large cluster dominates other clusters
- Refinement should satisfy balance criteria



Initital graph

Possible clustering hierarchies of the graph



FPGA Applications: Programmability

- Instead of fabricating a chip, we can program an FPGA

- Advantages:
  - Avoid non-recurring engineering (NRE) costs of IC design
  - Enable later changes easily (e.g. in case of a new media format)

- Disadvantages:
  - Larger area, more power, slower speed

FPGA Applications: System Prototyping

- Typical design cycle:

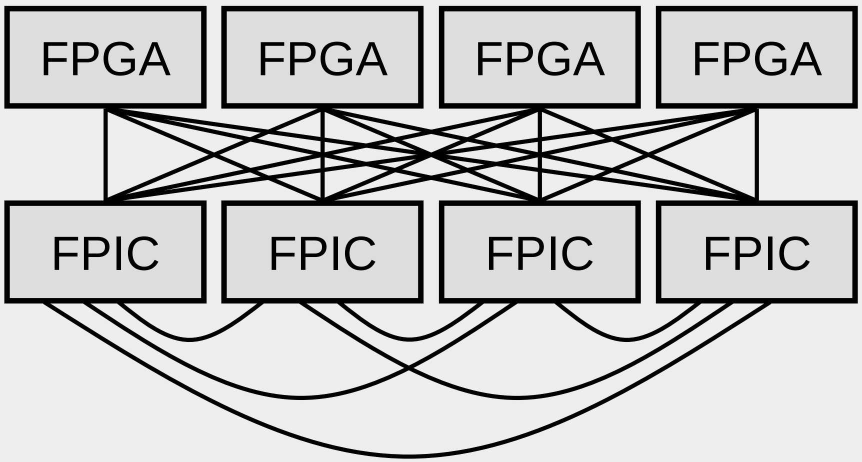
Hardware design | Software design

Arch | RTL | PD | Litho | Fab — Firmware | Drivers | Apps

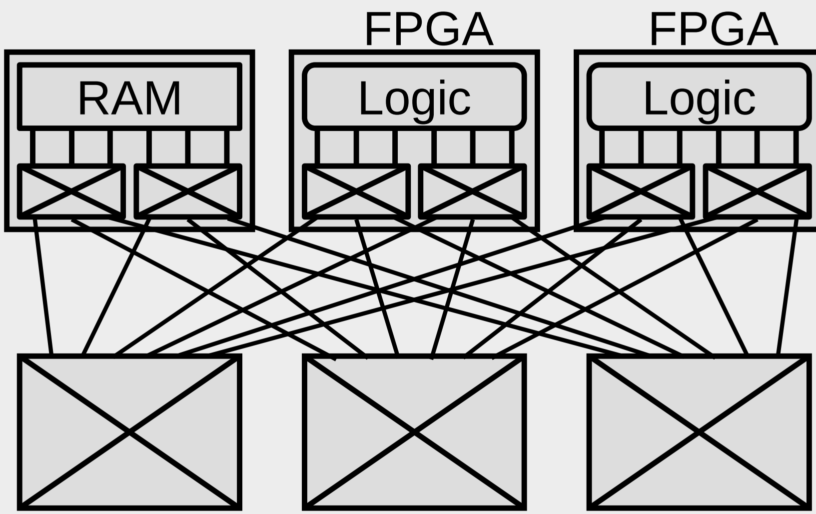
- Reduce time to market with system prototyping:

Arch | RTL | PD | Litho | Fab

Firmware | Drivers | Apps



Reconfigurable system with multiple FPGA and FPIC devices



Mapping of a typical system architecture onto multiple FPGAs

System Partitioning onto Multiple FPGAs

- Challenges:
  - Low utilization of FPGA resources (due to hard I/O pin limits)
  - Low performance (due to interconnect delays between FPGAs)
  - Long runtimes of system partitioning process

- Partitioning formulations:
  - Hard upper bound for total area of each partition
  - Hard upper bound for the cut size

- 3rd dimension for FPGA mapping: time

Summary of Chapter 2

- Circuit netlists can be represented by graphs or hypergraphs
- Partitioning a graph means assigning nodes to disjoint partitions
  - Total size of each partition (number/area of nodes) is limited
  - Objective: minimize the number connections between partitions
- Basic partitioning algorithms
  - Incremental changes organized into passes
  - KL swaps pairs of nodes from different partitions
  - FM re-assigns one node at a time
  - FM is faster, usually more successful
- Multilevel partitioning
  - Clustering
  - FM partitioning
  - Refinement (also uses FM partitioning)
- Application: system partitioning into FPGAs
  - Each FPGA is represented by a partition