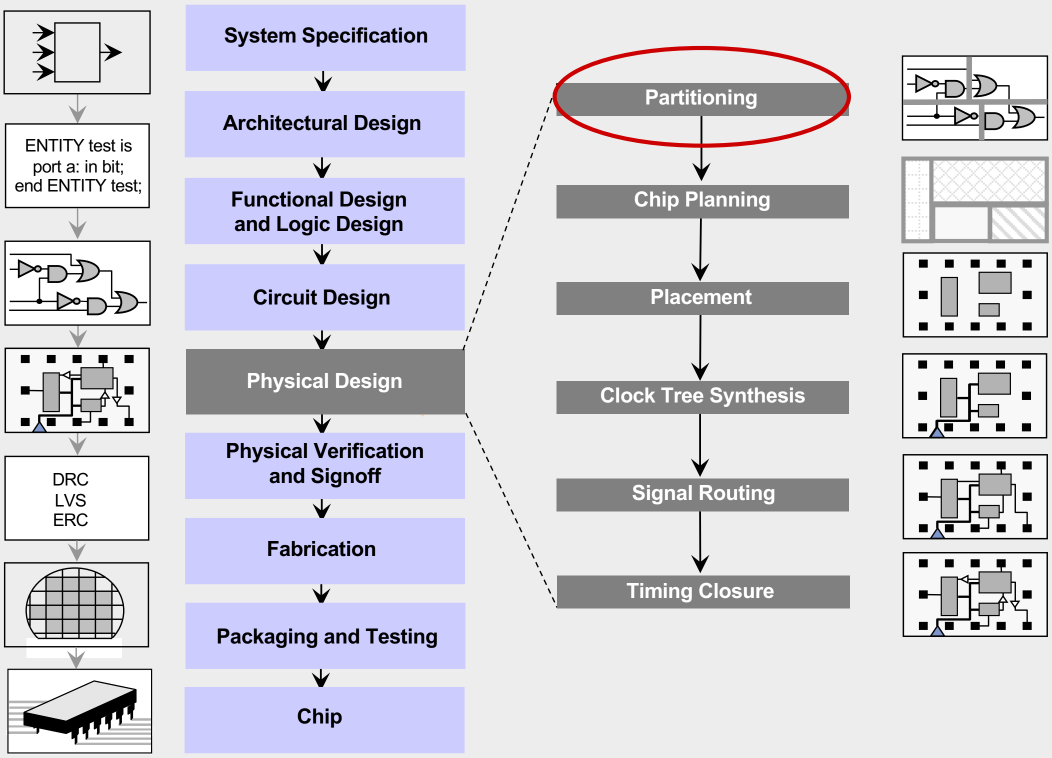
CSC688E: Algorithms for VLSI Design Automation

Lecture 2 – Netlist and System Partitioning

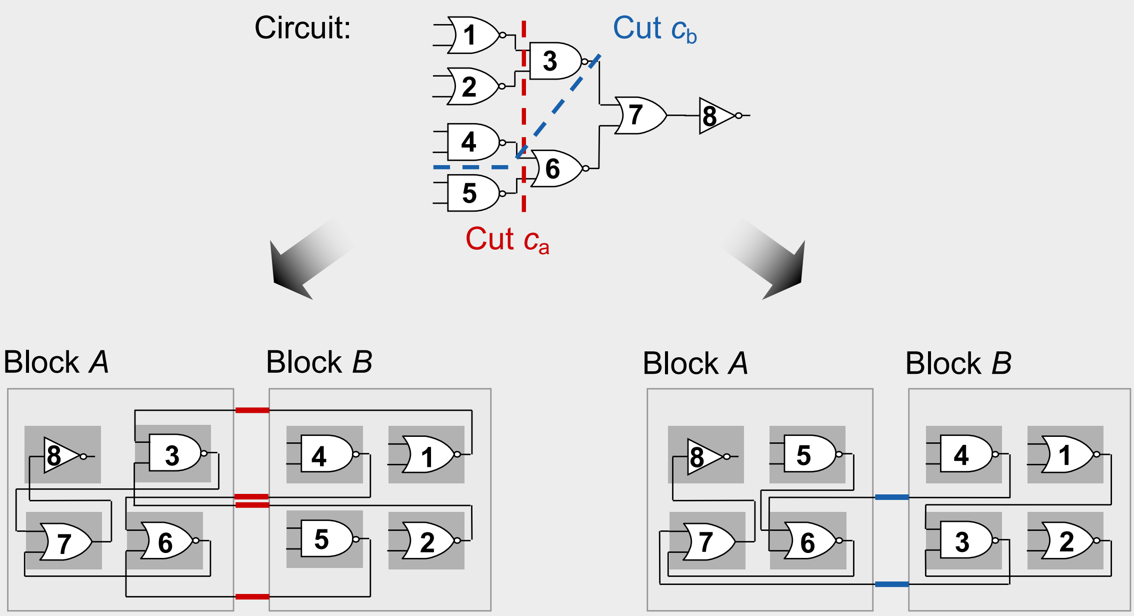
Lecture 2 – Netlist and System Partitioning

2.1 Introduction
2.2 Terminology
2.3 Optimization Goals
2.4 Partitioning Algorithms
  2.4.1 Kernighan-Lin (KL) Algorithm
  2.4.2 Extensions of the Kernighan-Lin Algorithm
  2.4.3 Fiduccia-Mattheyses (FM) Algorithm
2.5 Framework for Multilevel Partitioning
  2.5.1 Clustering
  2.5.2 Multilevel Partitioning
2.6 System Partitioning onto Multiple FPGAs

2.1 Introduction



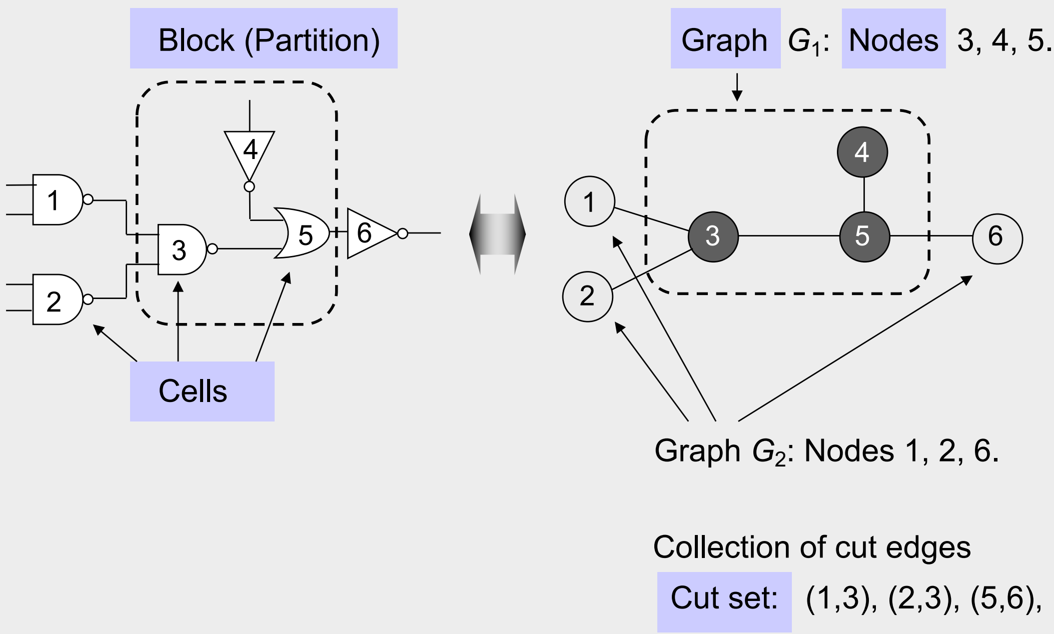
2.1 Introduction



Cut c_a: four external connections

Cut c_b: two external connections

2.2 Terminology



2.3 Optimization Goals

- 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 and all original edge relations are respected.

2.3 Optimization Goals

- In detail, what are the optimization goals?
  - Number of connections between partitions is minimized
  - Each partition meets all design constraints (size, number of external connections..)
  - Balance every partition as well as possible

- How can we meet these goals?
  - Unfortunately, this problem is NP-hard
  - Efficient heuristics are developed in the 1970s and 1980s.
    They are high quality and in low-order polynomial time.

Classification of Partitioning Algorithms

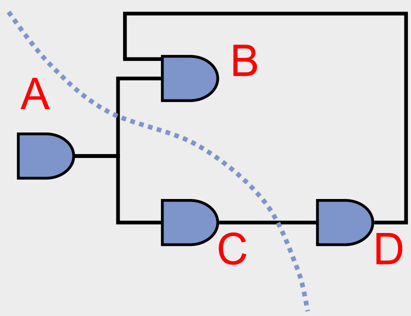
- Constructive algorithms versus iterative improvement algorithms
- Deterministic versus probabilistic algorithms

Some Terminology

- Partitioning: Dividing bigger circuits into a small number of partitions (top down)
- Clustering: cluster small cells into bigger clusters (bottom up).
- Covering / Technology Mapping: Clustering such that each partitions (clusters) have some special structure (e.g., can be implemented by a cell in a cell library).
- k-way Partitioning: Dividing into k partitions.
- Bipartitioning: 2-way partitioning.
- Bisectioning: Bipartitioning such that the two partitions have the same size.

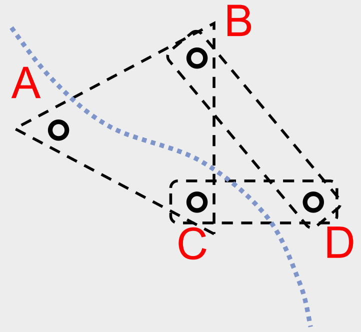
Circuit Representation

- 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)



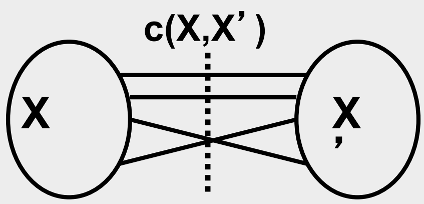
Bi-partitioning problem

- Also known as min cut partitioning
- Number of partitions = 2
- Minimize the nets crossing the partitions
- Size of the two partitions is equal
- Given a graph with N nodes, calculate the number of different bi-partitions!

Circuit Partitioning Formulation

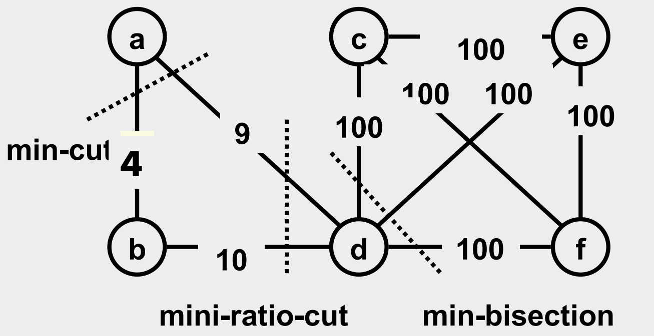
Bi-partitioning formulation:

Minimize interconnections between partitions



- **Minimum cut:** **min c(x, x')**
- **minimum bisection:** **min c(x, x') with |x|= |x'|**
- **minimum ratio-cut:** **min c(x, x') / |x||x'|**

A Bi-Partitioning Example



Min-cut size=13
Min-Bisection size = 300
Min-ratio-cut size= 19

Ratio-cut helps to identify natural clusters

Circuit Partitioning Formulation (Cont'd)

- General multi-way partitioning formulation:
- Partitioning a network N into $N_1, N_2, \ldots, N_k$ such that
- Each partition has an area constraint

$$\sum_{v \in N_i} a(v) \leq A_i$$

- Each partition has an I/O constraint

$$c(N_i, N - N_i) \leq I_i$$

Minimize the total interconnection:

$$\sum_{N_i} c(N_i, N - N_i)$$

Iterative Partitioning Algorithms

- **Greedy iterative improvement method**
  [Kernighan-Lin 1970]
  [Fiduccia-Mattheyses 1982]
  [krishnamurthy 1984]
- **Simulated Annealing**
  [Kirkpartrick-Gelatt-Vecchi 1983]
  [Greene-Supowit 1984]

Chapter 2 – Netlist and System Partitioning

2.1 Introduction
2.2 Terminology
2.3 Optimization Goals
2.4 Partitioning Algorithms
2.4.1 Kernighan-Lin (KL) Algorithm
2.4.2 Extensions of the Kernighan-Lin Algorithm
2.4.3 Fiduccia-Mattheyses (FM) Algorithm
2.5 Framework for Multilevel Partitioning
2.5.1 Clustering
2.5.2 Multilevel Partitioning
2.6 System Partitioning onto Multiple FPGAs

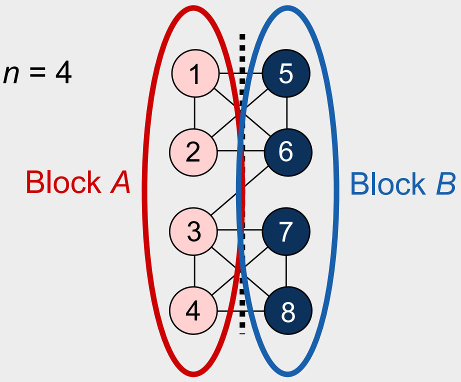
2.4.1 Kernighan-Lin (KL) Algorithm

"An Efficient Heuristic Procedure for Partitioning Graphs," The Bell System Tech. Journal, 49(2):291-307, 1970

Given: A graph with 2*n* 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: $n = 4$



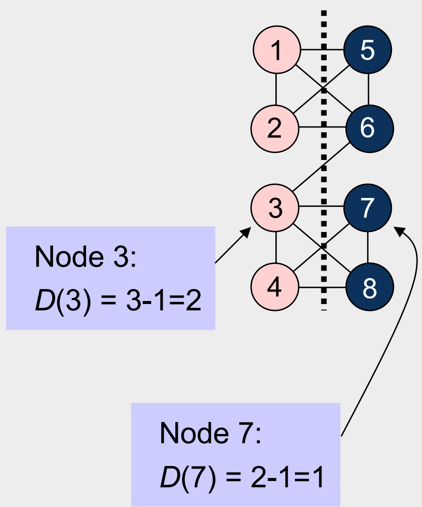
2.4.1 Kernighan-Lin (KL) Algorithm – Terminology

Cost $D(v)$ of moving 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 costs ($D > 0$) indicate that the node should move, while low costs ($D < 0$) indicate that the node should stay within the same partition.

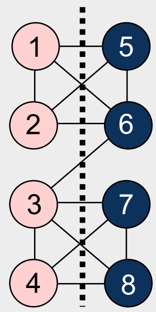


2.4.1 Kernighan-Lin (KL) Algorithm – Terminology

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

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

where
- $D(a)$, $D(b)$ are the respective costs 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

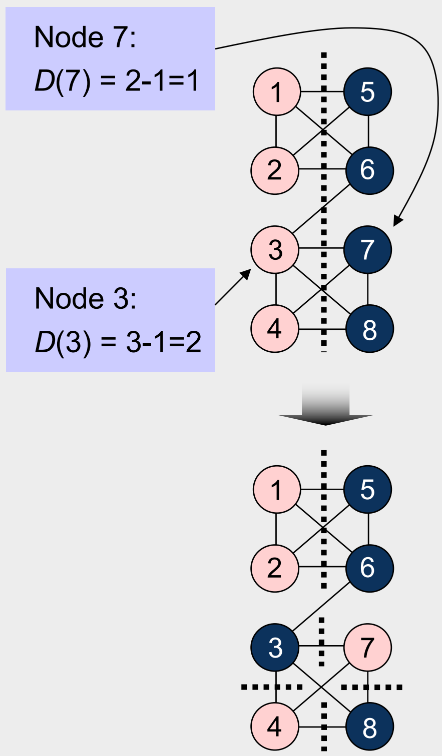
Gain of swapping a pair of nodes *a* and *b*

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

where
- $D(a)$, $D(b)$ are the respective costs 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, $c(a,b) = 0$.

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

=> 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* and *b*

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

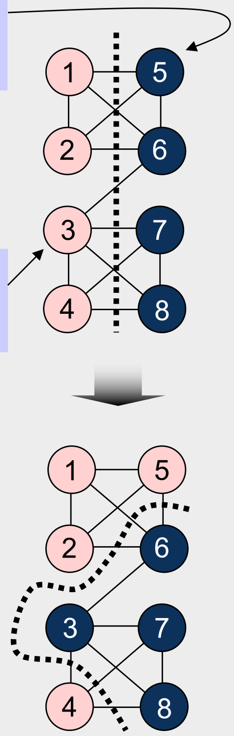
where
- $D(a)$, $D(b)$ are the respective costs of nodes *a*, *b*
- $c(a,b)$ is the connection weight between *a* and *b*:
  If an edge exists between *a* and *b*,
  then $c(a,b)$ = edge weight (here 1),
  otherwise, $c(a,b) = 0$.

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

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

Node 5:
$D(5)$ = 2-1=1

Node 3:
$D(3)$ = 3-1=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.

2.4.1 Kernighan-Lin (KL) Algorithm – Terminology

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 – One pass

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

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

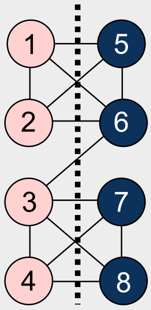
Step 2:
- Choose a_i and b_i such that $\Delta g_i = D(a_i) + D(b_i) - 2 \cdot c(a_i b_i)$ is maximized
- Swap and fix a_i and b_i

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

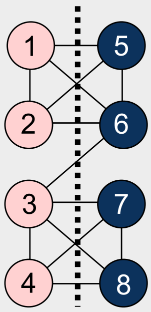
Step 5:
- Execute *m* swaps, reset remaining nodes
- Go to Step 1

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



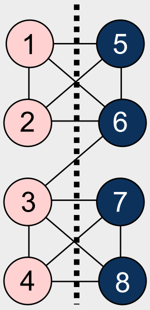
Cut cost: 9
Not fixed:
1,2,3,4,5,6,7,8

Costs $D(v)$ 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

Costs $D(v)$ of each node:

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

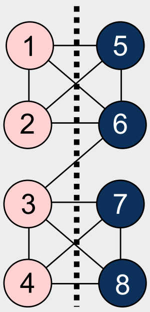
Nodes that lead to maximum gain

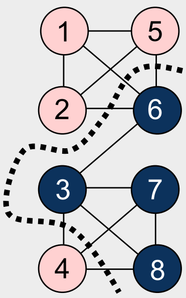
$\Delta g_1 = 2+1-0 = 3$ — Gain after node swapping

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

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

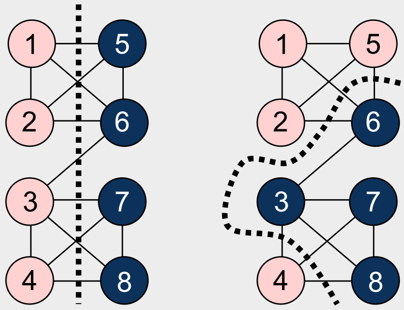
Nodes that lead to maximum gain

$\Delta g_1 = 2+1-0 = 3$ — 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



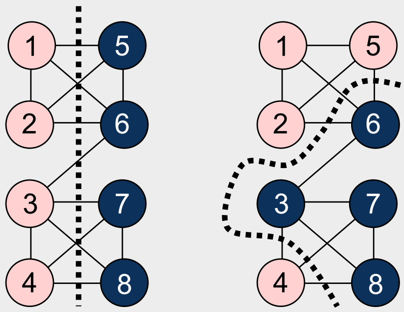
Cut cost: 9
Not fixed:
1,2,3,4,5,6,7,8

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

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

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

2.4.1 Kernighan-Lin (KL) Algorithm – Example



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

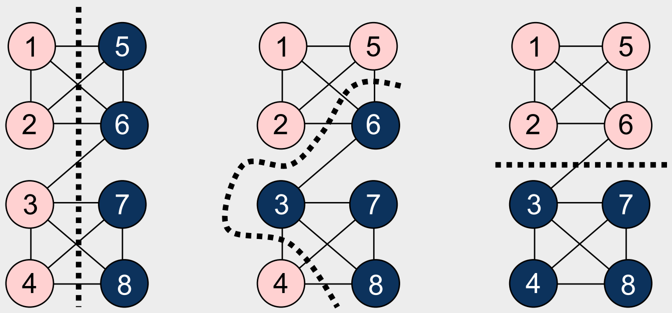
Cut cost: 6
Not fixed:
1,2,4,6,7,8

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

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

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

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

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

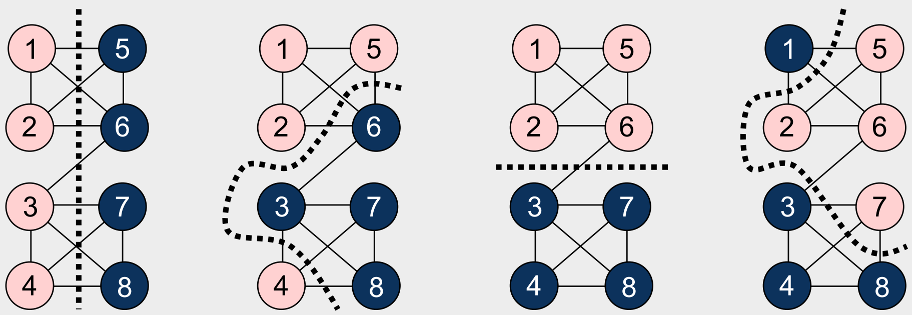
$\Delta g_2 = 3+2-0 = 5$
Swap (4,6)
$G_2 = G_1+\Delta g_2 = 8$

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

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

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

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

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

$\mathbf{D(1) = -3}$ $\mathbf{D(7)=-3}$
$D(2) = -3$ $D(8)=-3$

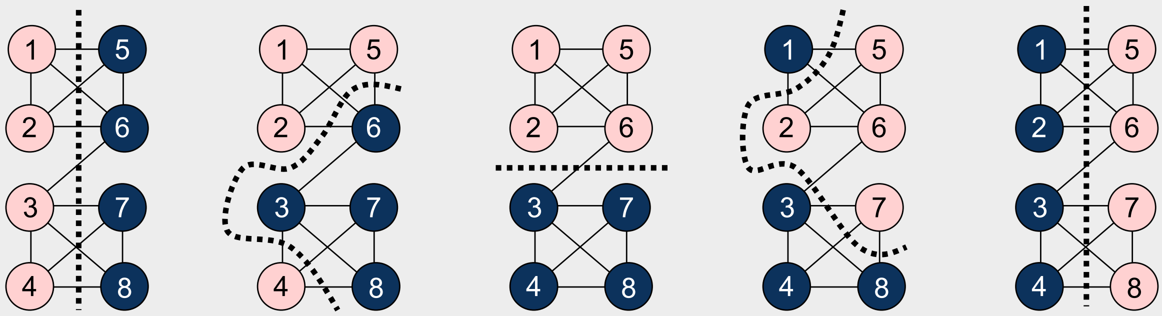
$\Delta g_3 = -3-3-0 = -6$
Swap (1,7)
$G_3 = G_2 + \Delta g_3 = 2$

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

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

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

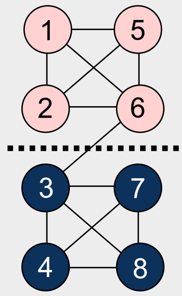
2.4.1 Kernighan-Lin (KL) Algorithm – Example

$D(1) = 1$ **$D(5) = 1$** $D(2) = 1$ $D(6) = 2$ **$D(3) = 2$** $D(7) = 1$ $D(4) = 1$ $D(8) = 1$ $\Delta g_1 = 2+1-0 = 3$ **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 $G_m \leq 0$.



Kernighan-Lin Algorithm

Algorithm: Kernighan-Lin(G)
Input: $G = (V, E)$, $|V| = 2n$.
Output: Balanced bi-partition A and B with "small" cut cost.

```
1 begin
2 Bipartition G into A and B such that |V_A| = |V_B|, V_A ∩ V_B = ∅,
  and V_A ∪ V_B = V.
3 repeat
4   Compute D_v, ∀ v ∈ V.
5   for i = 1 to n do
6     Find a pair of unlocked vertices v_ai ∈ V_A and v_bi ∈ V_B whose
      exchange makes the largest decrease or smallest increase in cut
      cost;
7     Mark v_ai and v_bi as locked, store the gain ĝ_i, and compute the new
      D_v, for all unlocked v ∈ V;
8   Find k, such that G_k = Σ_{i=1}^{k} ĝ_i is maximized;
9   if G_k > 0 then
10      Move v_a1, ..., v_ak from V_A to V_B and v_b1, ..., v_bk from V_B to V_A;
11  Unlock v, ∀ v ∈ V.
12 until G_k ≤ 0;
13 end
```

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

Drawbacks of the Kernighan-Lin Heuristic

- The K-L heuristic **handles only unit vertex weights**.
  - Vertex weights might represent block sizes, different from blocks to blocks.
  - Reducing a vertex with weight $w(v)$ into a clique with $w(v)$ vertices and edges with a high cost increases the size of the graph substantially.
- The K-L heuristic **handles only exact bisections**.
  - Need dummy vertices to handle the unbalanced problem.
- The K-L heuristic **cannot handle hypergraphs**.
  - Need to handle multi-terminal nets directly.
- The **time complexity of a pass is high**, $O(n^3)$.

2.4.2 Extensions of the Kernighan-Lin (KL) Algorithm

- Unequal partition sizes
  - Apply the KL algorithm with only min($|A|$,$|B|$) pairs swapped
  - May want to insert a dummy node.
- Unequal node weights
  - Try to rescale weights to integers, e.g., as multiples of *the greatest common divisor* of all node weights
  - Maintain area balance or allow a *one-move deviation* from balance
- *k*-way partitioning (generating *k* partitions)
  - Apply the KL two-way partitioning algorithm to all possible pairs of partitions
  - Recursive partitioning (convenient when k is a power of two)
  - Direct k-way extensions exist

2.4.3 Fiduccia-Mattheyses (FM) Algorithm

- Modification of KL Algorithm:
  - Can handle non-uniform vertex weights (areas)
  - Allow unbalanced partitions
  - Extended to handle hypergraphs
  - Clever way to select vertices to move, run much faster.

"A Linear-time Heuristics for Improving Network Partitions," 19th DAC, pages 175-181, 1982.

2.4.3 Fiduccia-Mattheyses (FM) Algorithm

- Single cells are moved independently instead of swapping pairs of cells --- cannot and do not need to maintain exact partition balance
  - The area of each individual cell is taken into account
  - Applicable to partitions of unequal size and in the presence of initially fixed cells
- Cut costs are extended to include hypergraphs
  - nets with 2+ pins
- While the KL algorithm aims to minimize cut costs based on edges, the FM algorithm minimizes cut costs based on nets
- Nodes and subgraphs are referred to as *cells* and *blocks*, respectively

2.4.3 Fiduccia-Mattheyses (FM) Algorithm

Given: a hypergraph $G(V,H)$ with nodes and *weighted* hyperedges
partition size constraints

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*

This problem is NP-Complete!!!

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Terminology

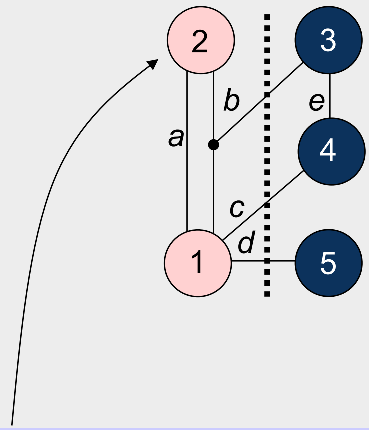
Gain $\Delta g(c)$ for cell c

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

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.

A net is *cut* if its cells occupy more than one partition. Otherwise, the net is *uncut*

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Terminology

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

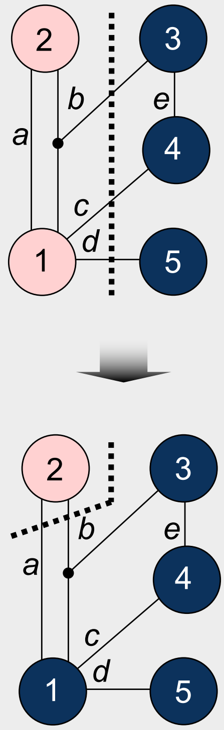
$\Delta g(c) = FS(c) - TE(c)$,

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:	$FS(5) = 1$	$TE(5) = 0$	$\Delta g(5) = 1$



2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Terminology

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

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

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Terminology

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

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

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Terminology

Balance criterion - *To avoid having all cells migrate to one block*

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

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

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Terminology

Base cell

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

Base cell

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

2.4.3 Fiduccia-Mattheyses (FM) Algorithm - One pass

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:

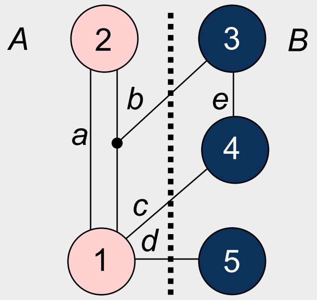
– If all cells are fixed, go to Step 5. If not:
– Choose next base cell c_i with maximal gain Δg_i and move this cell
– $i = i + 1$, go to Step 3

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:

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Example



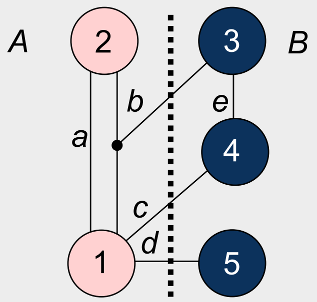
Given:
Ratio factor r = 0,375
area(Cell_1) = 2
area(Cell_2) = 4
area(Cell_3) = 1
area(Cell_4) = 4
area(Cell_5) = 5.

Step 0: Compute the balance criterion

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

$0{,}375 * 16 - 5 = 1 \leq area(A) \leq 11 = 0{,}375 * 16 + 5.$

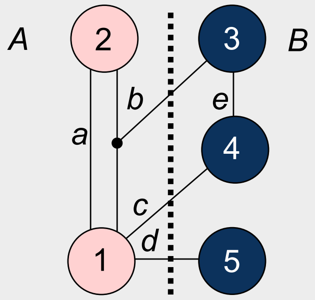
2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Example



Step 1: Compute the gains of each cell

Cell 1:	*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

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Example



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

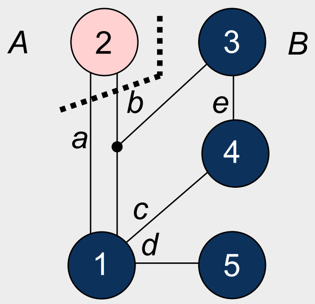
Possible base cells are Cell 1 and Cell 5

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.

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Example

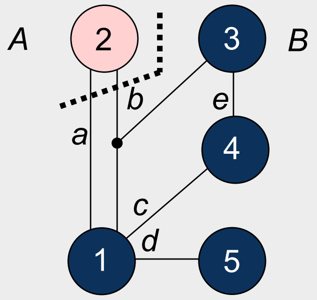


Step 3: Fix base cell, update Δg values

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}.

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Example



Iteration i = 1

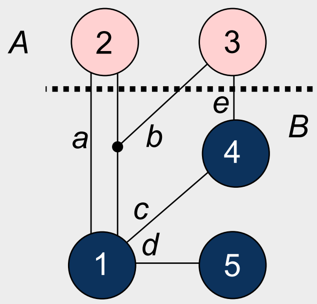
Cell 2:	FS(Cell_2) = 2	TE(Cell_2) = 0	Δg(Cell_2) = 2
Cell 3:	FS(Cell_3) = 0	TE(Cell_3) = 1	Δg(Cell_3) = -1
Cell 4:	FS(Cell_4) = 0	TE(Cell_4) = 2	Δg(Cell_4) = -2
Cell 5:	FS(Cell_5) = 0	TE(Cell_5) = 1	Δg(Cell_5) = -1

Iteration i = 2

Cell *2* has maximum gain Δg_2 = 2, *area*(*A*) = 0, balance criterion is violated.
Cell *3* has next maximum gain Δg_2 = -1, *area*(*A*) = 5, balance criterion is met.
Cell *5* has next maximum gain Δg_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}

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Example



Iteration i = 2

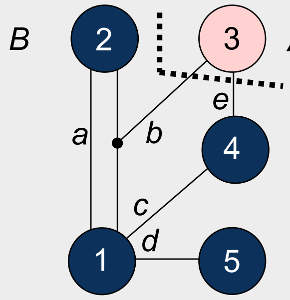
Cell 2:	Δg(Cell_2) = 1
Cell 4:	Δg(Cell_4) = 0
Cell 5:	Δg(Cell_5) = -1

Iteration i = 3

Cell *2* has maximum gain Δg_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}

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Example



Iteration i = 3

Cell 4: Δg(Cell_4) = 0

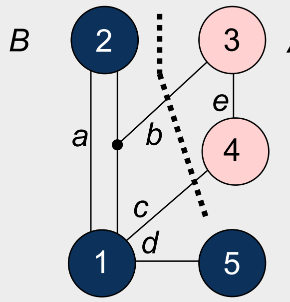
Cell 5: Δg(Cell_5) = -1

Iteration i = 4

Cell 4 has maximum gain Δg_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}

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Example



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.

2.4.3 Fiduccia-Mattheyses (FM) Algorithm – Example

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

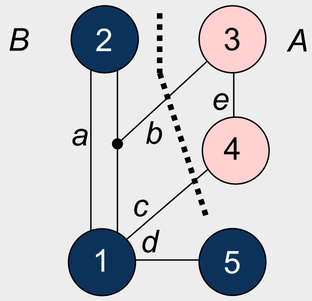
$G_1 = \Delta g_1 = 1$

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

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

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

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



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.

Runtime difference between KL & FM

- Runtime of partitioning algorithms
  - KL is sensitive to the number of nodes and edges
  - FM is sensitive to the number of nodes and nets (hyperedges)
- Asymptotic complexity of partitioning algorithms
  - KL has cubic time complexity *per pass*
  - FM has linear time complexity *per pass*

Chapter 2 – Netlist and System Partitioning

2.1 Introduction

2.2 Terminology

2.3 Optimization Goals

2.4 Partitioning Algorithms
- 2.4.1 Kernighan-Lin (KL) Algorithm
- 2.4.2 Extensions of the Kernighan-Lin Algorithm
- 2.4.3 Fiduccia-Mattheyses (FM) Algorithm

2.5 Framework for Multilevel Partitioning
- 2.5.1 Clustering
- 2.5.2 Multilevel Partitioning

2.6 System Partitioning onto Multiple FPGAs

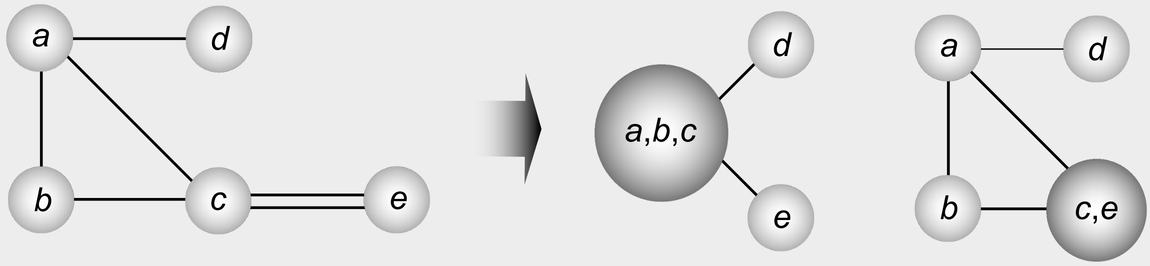
2.5.1 Clustering

- To simplify the problem, 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

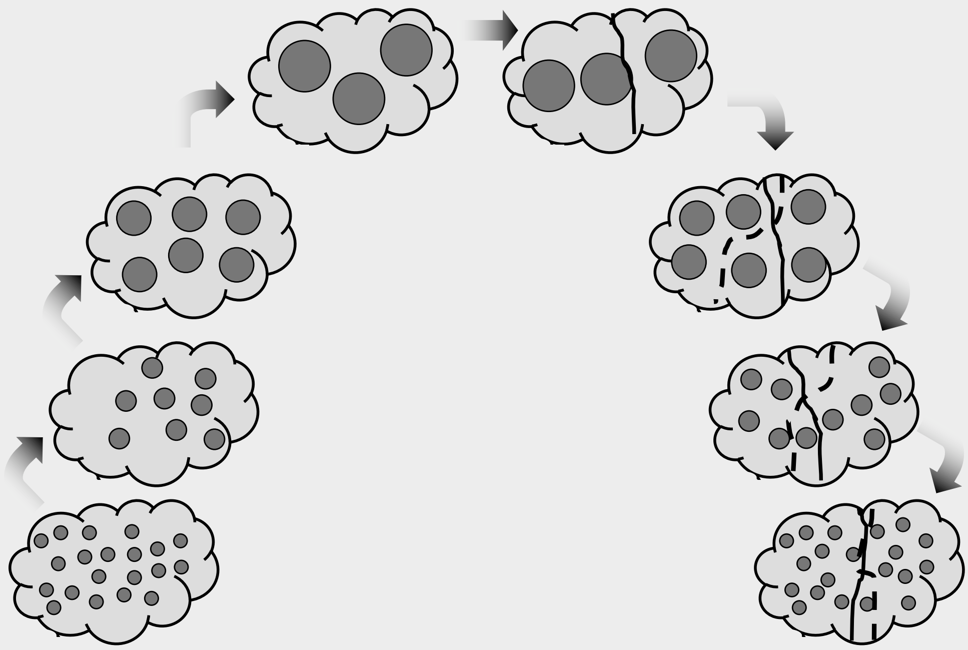
2.5.1 Clustering

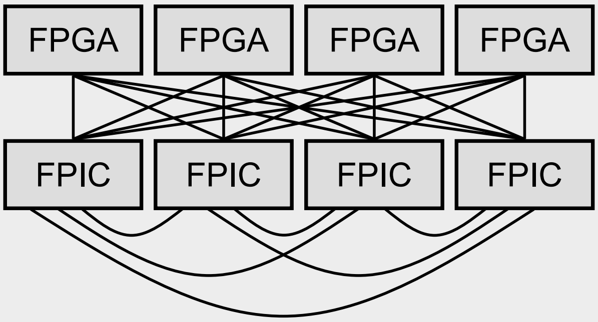


Initital graph

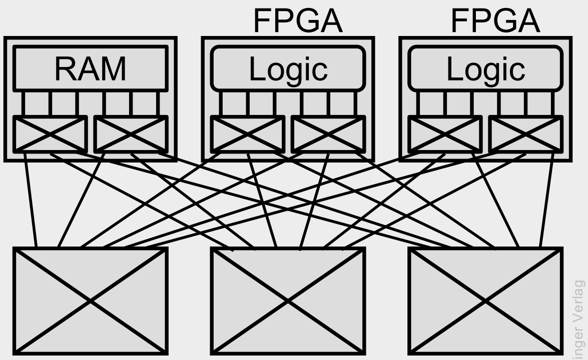
Possible clustering hierarchies of the graph

2.5.2 Multilevel Partitioning





Reconfigurable system with multiple FPGA and FPIC devices



Mapping of a typical system architecture onto multiple FPGAs

Summary of Lecture 2

- Circuit netlists can be represented by graphs
- 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
  - Move-based, move are 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