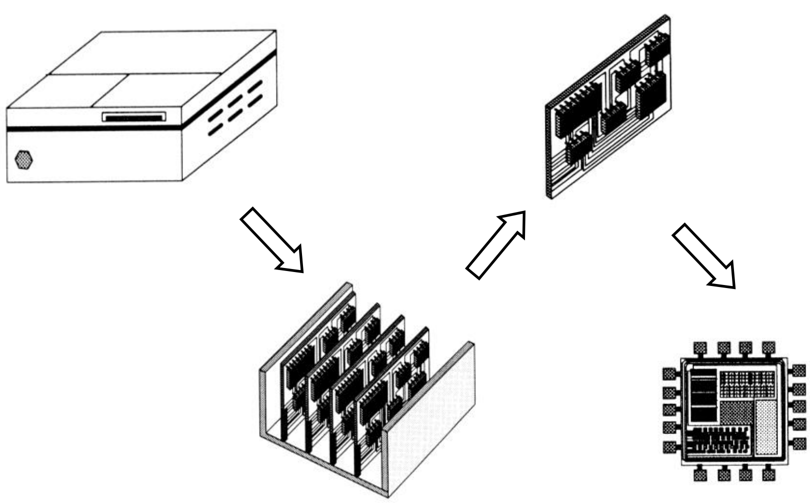
# CAD Algorithms for Physical Design - Partitioning

Christos P Sotiriou



## System Hierarchy

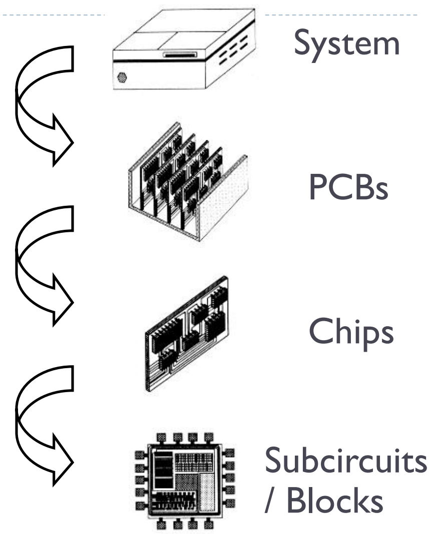


# Partitioning Levels

System Level Partitioning

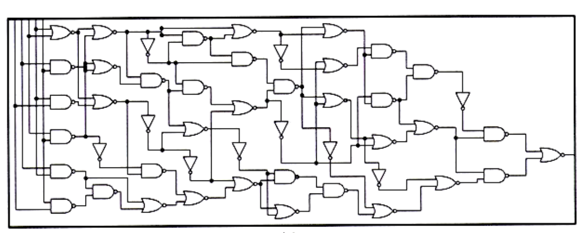
Board Level Partitioning

Chip Level Partitioning

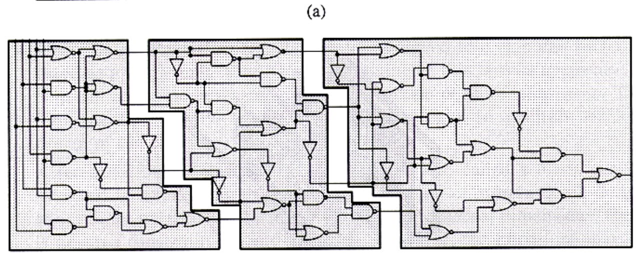





# Circuit Partitioning



(a)



(b)

## Importance of Circuit Partitioning

- Divide-and-conquer methodology
  - The most effective way to solve problems of high complexity
  - E.g.: min-cut based placement, partitioning-based test generation,…
- System-level partitioning for multi-chip designs
  - inter-chip interconnection delay dominates system performance.
- Circuit emulation/parallel simulation
  - partition large circuit into multiple FPGAs (e.g. Quickturn), or multiple special-purpose processors (e.g. Zycad).
- Parallel CAD development
  - Task decomposition and load balancing
- In deep-submicron designs, partitioning defines local and global interconnect, and has significant impact on circuit performance

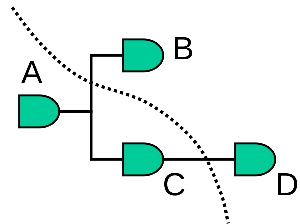


## Terminology

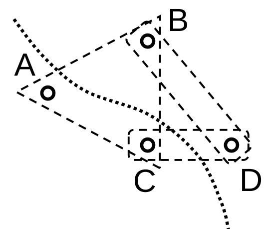
- **Partitioning**: Dividing bigger circuits into a small number of partitions (top down)
- **Clustering**: cluster small cells into bigger clusters (bottom up).
- **Covering / Technology Mapping**: Clustering such that each partitions (clusters) have some special structure (e.g., can be implemented 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}

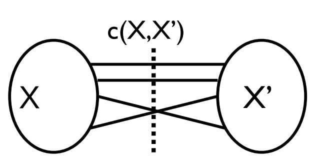



- Vertex label: Gate size/area
- Hyperedge label:
  - Importance of net (weight)



## Circuit Partitioning Formulation

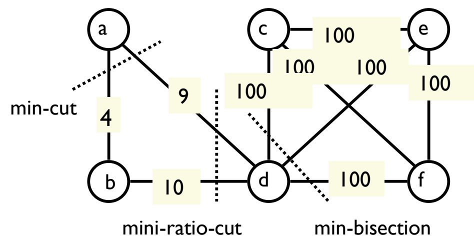
- Bi-partitioning formulation:
  - Minimize interconnections between partitions




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

# Bi-Partitioning Example

- Edge numbers reflect weight, *i.e.* number of connections




- Min-cut size=13
- Min-Bisection size = 300
- Min-ratio-cut size= 19
  - Ratio-cut helps to identify natural clusters



# Circuit Partitioning Formulation - 2

- General multi-way partitioning formulation:
  - Partitioning a network N into N1, N2, …, Nk such that
- Each partition has an area constraint

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

- Each partition has an I/O constraint

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

- Minimize the total interconnection:

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

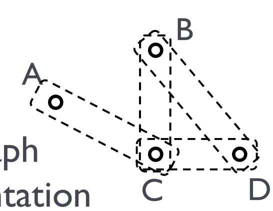
# Types of Partitioning Algorithms

- Combinatorial (Iterative) partitioning algorithms
  - SA-based
  - Most Effective:
    - Kernighan-Lin (KL)
    - Fiduccia-Mattheyses (FM)
- Spectral based partitioning algorithms
- Net partitioning vs. module partitioning
- Multi-way partitioning
- Multi-level partitioning
- Further study in partitioning techniques
  - Timing-driven …

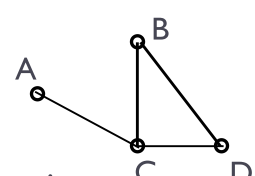


# Restricted Partitioning Problem

- 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 is a **Graph**




Hypergraph Representation




Graph Representation

## Problem Formulation

- Input: A 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.
- NP-Complete Problem



## Partitioning is NP

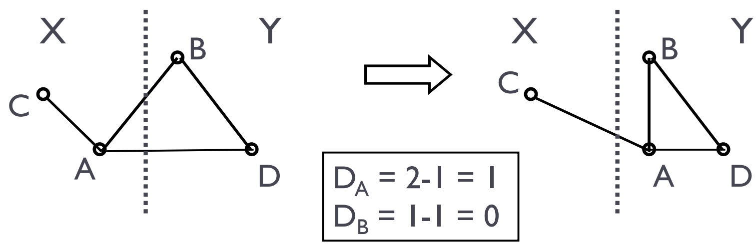
- Try <u>all</u> possible bisections. Find the best one.
- If there are 2n vertices,
  \# of possibilities = $(2n)! / n!^2 = n^{O(n)}$

$$C(n,k) = \frac{P(n,k)}{P(k,k)} = \frac{n!}{(n-k)!k!}.$$

- For 4 vertices (A,B,C,D), 3 possibilities.
  I. 1. X={A,B} & Y={C,D}
  II. 2. X={A,C} & Y={B,D}
  III. 3. X={A,D} & Y={B,C}
- For 100 vertices, $5\text{x}10^{28}$ possibilities.
  - Need $1.59\text{x}10^{13}$ years if one can try 100M possibilities per second.

## KL/FM Ideas - 1

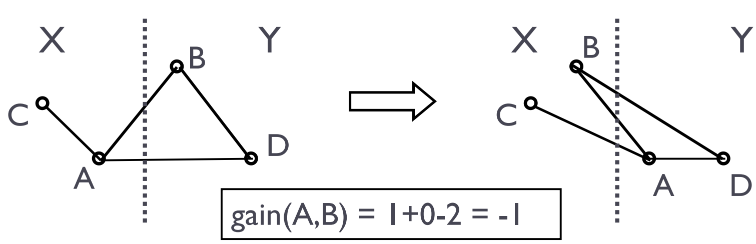
- Define $D_A$ = Decrease in cut value (cost), if moving node A to the alternative partition
  - Divide into
  - **External cost** (connection) $E_A$ – **Internal cost** $I_A$
  - Moving node A from partition X to partition Y would increase the value of the **cutsize** (or cutset) by $E_A$ and decrease it by $I_A$






## KL/FM Ideas - 2

- Specifically, in KL we want to balance two partitions
  - Perform node swaps instead of moves
- If nodes A and B are swapped
  - **gain**(A,B) = $D_A + D_B - 2 \times c_{AB}$
  - where $c_{AB}$ : edge cost for AB



## Kernighan-Lin Algorithm - 1

- Gain-based cell swap
  - Gain represents cutline change for a candidate swap
  - At every swap, algorithms select **maximum gain swap**
- Pass Concept
  - A set of complete swaps, *i.e.* all cells swapped once
  - Swapped cells are **locked**; may not be swapped again
- At the end of a Pass, the best cost through the movements log is selected
  - Limited negative swaps are accepted until the end of the pass
    - Least negative when no positive moves are possible
  - Hill-climbing part of the algorithm



## Kernighan-Lin Algorithm - 2

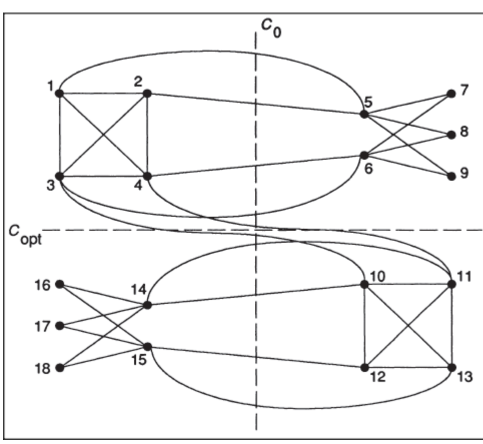
- Start with any initial legal partitions X and Y.
- A pass (exchanging each vertex exactly once) is described below:
  - 1. For i := 1 to n do
    From the unlocked (unexchanged) vertices,
    choose a pair (A,B) s.t. Gain(A,B) is largest.
    Exchange A and B. Lock A and B.
    Let gi = gain(A,B).
  - 2. Find the k s.t. Gain = g1 + ... + gk is maximum.
  - 3. Switch the first k pairs up to the maximum Gain
- Repeat the pass until there is no improvement (G=0).

# Kernighan-Lin Algorithm - 3

```
Pair : array [1 : n/2] of pair of [1 : n];
Cost : array [0 : n/2] of integer;
Locked : array [1 : n] of Boolean;
D : array [1 : n] of integer;
c : array [1 : n, 1 : n] of integer;
BestChange : [1 : n/2];
BestCost : integer;
imin, jmin : [1 : n];

compute the c and D values;
for i from 1 to n do
    Locked[i] := false od;

BestCost := Cost[0] := cutsize(A, B);
BestChange := 0;
for s from 1 to n/2 do
    Cost[s] := ∞;
    for i, j from 1 to n such that v_i ∈ A and Locked[i] = false
                and v_j ∈ B and Locked[j] = false do
        if 2c[i, j] − D[i] − D[j] < Cost[s] then
            Pair[s] := (i, j);
            Cost[s] := 2c[i, j] − D[i] − D[j] fi od;
    (imin, jmin) := Pair[s];
    Locked[imin] := Locked[jmin] := true;
    for i from 1 to n such that Locked[i]= false do
        if v_i ∈ A then
            D[i] := D[i] − c[i, jmin] + c[i, imin]
        else
            D[i] := D[i] − c[i, imin] + c[i, jmin] fi od;
    Cost[s] := Cost[s − 1] + Cost[s];
    if Cost[s] < BestCost then
        BestChange := s;
        BestCost := Cost[s] fi od;

for s from 1 to BestChange do
    exchange Pair[s] od;
```



# KL Example




| Step No. | Vertex Pair | Change | Cutsize |
|---|---|---|---|
| 0 | — | 0 | 10 |
| 1 | {4, 10} | 2 | 12 |
| 2 | {2, 12} | 2 | 12 |
| 3 | {1, 13} | −2 | 8 |
| 4 | {3, 11} | −8 | 2 |
| 5 | {7, 18} | −4 | 6 |
| 6 | {8, 17} | 0 | 10 |
| 7 | {5, 15} | 2 | 12 |
| 8 | {9, 16} | 2 | 12 |
| 9 | {6, 14} | 0 | 10 |

## KL and Hypergraph Representation

- For a hypergraph representation
  - the k-clique model may be used
- A net containing k connections
  - Single gate output fans out to (k – 1) gate inputs forms a k-clique
  - Each edge in the clique gets a weight of 1/(k – 1)
  - If an edge already exists, the weight is added, instead of adding a new parallel edge
- Edges may also possess individual weights
  - Integer or floating-point numbers

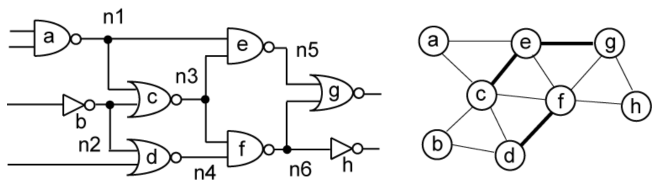


## Complexity of KL Algorithm

- 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.
- Better implementation can get $O(n^2 \log n)$ time per pass.
- Number of passes is usually small.
- Useful Survey Paper
  - Charles Alpert and Andrew Kahng, "Recent Directions in Netlist Partitioning: A Survey", Integration: the VLSI Journal, 19(1-2), 1995, pp. 1-81.

# Kernighan-Lin Algorithm Example

- Perform single KL pass on the following circuit:
  - KL needs undirected graph (clique-based weighting)

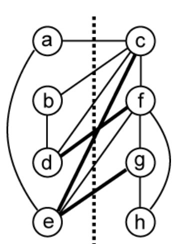



(a) (b)



# Kernighan-Lin Algorithm Example

- First Swap

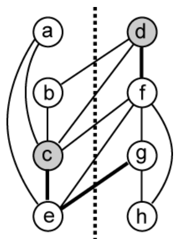



initial partitioning

| pair | $E_x - I_x$ | $E_y - I_y$ | $c(x,y)$ | gain |
|---|---|---|---|---|
| $(a,c)$ | $0.5 - 0.5$ | $2.5 - 0.5$ | 0.5 | 1 |
| $(a,f)$ | $0.5 - 0.5$ | $1.5 - 1.5$ | 0 | 0 |
| $(a,g)$ | $0.5 - 0.5$ | $1 - 1$ | 0 | 0 |
| $(a,h)$ | $0.5 - 0.5$ | $0 - 1$ | 0 | -1 |
| $(b,c)$ | $0.5 - 0.5$ | $2.5 - 0.5$ | 0.5 | 1 |
| $(b,f)$ | $0.5 - 0.5$ | $1.5 - 1.5$ | 0 | 0 |
| $(b,g)$ | $0.5 - 0.5$ | $1 - 1$ | 0 | 0 |
| $(b,h)$ | $0.5 - 0.5$ | $0 - 1$ | 0 | -1 |
| **$(d,c)$** | **$1.5 - 0.5$** | **$2.5 - 0.5$** | **0.5** | **2** |
| $(d,f)$ | $1.5 - 0.5$ | $1.5 - 1.5$ | 1 | -1 |
| $(d,g)$ | $1.5 - 0.5$ | $1 - 1$ | 0 | 1 |
| $(d,h)$ | $1.5 - 0.5$ | $0 - 1$ | 0 | 0 |
| $(e,c)$ | $2.5 - 0.5$ | $2.5 - 0.5$ | 1 | 2 |
| $(e,f)$ | $2.5 - 0.5$ | $1.5 - 1.5$ | 0.5 | 1 |
| $(e,g)$ | $2.5 - 0.5$ | $1 - 1$ | 1 | 0 |
| $(e,h)$ | $2.5 - 0.5$ | $0 - 1$ | 0 | 1 |

# Kernighan-Lin Algorithm Example

- Second Swap

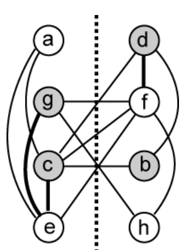



| pair | $E_x - I_x$ | $E_y - I_y$ | $c(x,y)$ | gain |
|---|---|---|---|---|
| $(a,f)$ | $0-1$ | $1-2$ | 0 | -2 |
| $(a,g)$ | $0-1$ | $1-1$ | 0 | -1 |
| $(a,h)$ | $0-1$ | $0-1$ | 0 | -2 |
| $(b,f)$ | $0.5-0.5$ | $1-2$ | 0 | -1 |
| $\mathbf{(b,g)}$ | $\mathbf{0.5-0.5}$ | $\mathbf{1-1}$ | **0** | **0** |
| $(b,h)$ | $0.5-0.5$ | $0-1$ | 0 | -1 |
| $(e,f)$ | $1.5-1.5$ | $1-2$ | 0.5 | -2 |
| $(e,g)$ | $1.5-1.5$ | $1-1$ | 1 | -2 |
| $(e,h)$ | $1.5-1.5$ | $0-1$ | 0 | -1 |



# Kernighan-Lin Algorithm Example

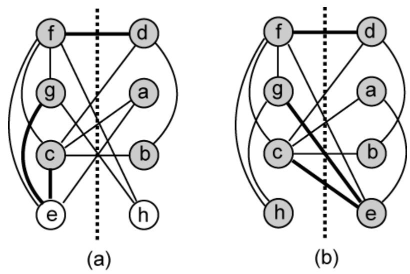
- Third Swap




| pair | $E_x - I_x$ | $E_y - I_y$ | $c(x,y)$ | gain |
|---|---|---|---|---|
| $\mathbf{(a,f)}$ | $\mathbf{0-1}$ | $\mathbf{1.5-1.5}$ | **0** | **-1** |
| $(a,h)$ | $0-1$ | $0.5-0.5$ | 0 | -1 |
| $(e,f)$ | $0.5-2.5$ | $1.5-1.5$ | 0.5 | -3 |
| $(e,h)$ | $0.5-2.5$ | $0.5-0.5$ | 0 | -2 |

# Kernighan-Lin Algorithm Example

- Fourth Swap
  - Last swap does not require gain computation

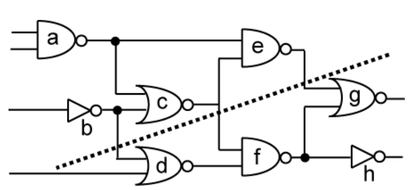





# Kernighan-Lin Algorithm Example

- Cutsize reduced from 5 to 3
  - Two best solutions found (solutions are always area-balanced)

| $i$ | pair | $gain(i)$ | $\sum gain(i)$ | cutsize |
|---|---|---|---|---|
| 0 | - | - | - | 5 |
| **1** | **$(d, c)$** | **2** | **2** | **3** |
| **2** | **$(b, g)$** | **0** | **2** | **3** |
| 3 | $(a, f)$ | -1 | 1 | 4 |
| 4 | $(e, h)$ | -1 | 0 | 5 |



# Fiduccia-Mattheyses Algorithm

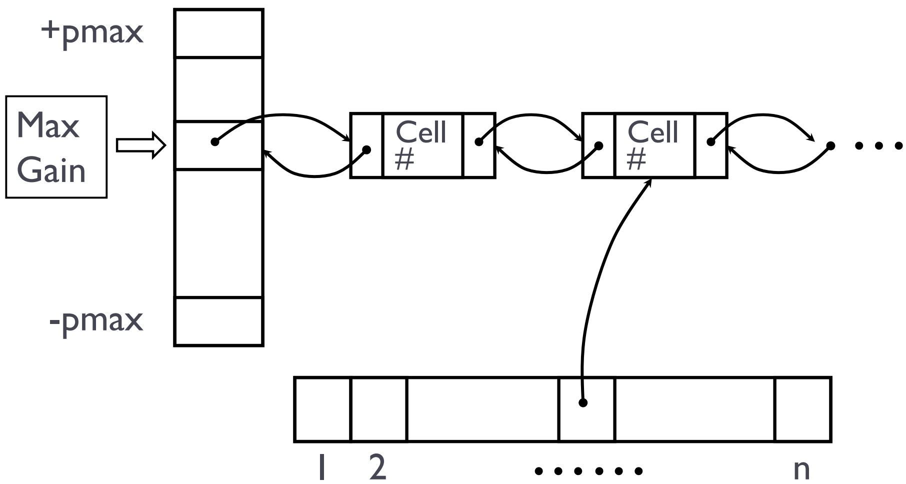
- Modification of KL Algorithm:
  - Can handle non-uniform vertex weights (areas)
  - Allow unbalanced partitions
  - Extended to handle hypergraphs
  - Clever way to select vertices to move, run much faster.
- Input: A hypergraph with
  - Set vertices V (|V| = m)
  - Set of hyperedges E. (total # **nets** in netlist = n)
  - Area $a_u$ for each vertex u in V.
  - Cost $c_e$ for each hyperedge in e.
  - An area ratio r.
- Output: 2 partitions X & Y such that
  - Total cost of hyperedges cut is minimized.
  - area(X) / (area(X) + area(Y)) is about r.



# Fiduccia-Mattheyses Algorithm

- Similar to KL:
  - Work in passes.
  - Lock vertices after moved.
  - Actually, only move those vertices up to the maximum partial sum of gain.
- Difference from KL:
  - Not exchanging pairs of vertices.
    Move only one vertex at each time.
  - The use of gain bucket data structure.

## Gain Bucket Data Structure






## FM External and Internal Vertex Cost

$v_i \in A$.

**Definition 6.3 (External and Internal Hyperedge Cost)** *The* external hyperedge cost *of vertex $v_i$ is defined as*

$$E(i) := \sum_{e \in E_{\text{ext},i}} c(e)$$

*where*

$$E_{\text{ext},i} := \{e \in E \mid \{v_i\} = e \cap A\}$$

*Analogously, the* internal hyperedge cost *of vertex $v_i$ is defined as*

$$I(i) := \sum_{e \in E_{\text{int},i}} c(e)$$

*where*

$$E_{\text{int},i} := \{e \in E \mid v_i \in e \text{ and } e \cap B = \emptyset\}$$

**Definition 6.2 (Gain)** *The* gain *of $v_i$ is defined as*

$$D(i) := E(i) - I(i)$$

- For cell *i* in Partition *P1*
- E(i) = FS(i) =
  - **number of nets that have *i* as the only cell in Partition P1**
- I(i) = TE(i) =
  - **number of nets containing cell *i* and are entirely located in P1**

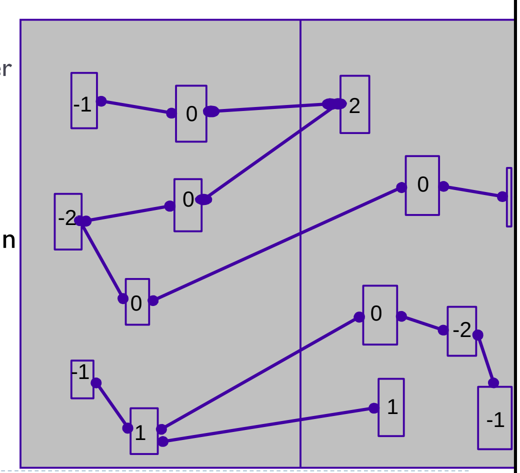
# FM Algorithm in Detail

- Perform the following three steps before the first pass begins:
  - (i) unlock all cells,
  - (ii) compute the gain of all cells based on the initial partitioning,
  - (iii) add the cells to the bucket structure.
- Once the pass begins, Repeat the following four steps at every move until all cells are locked:
  - (i) **we choose the "legal" cell with maximum gain** (A cell move is legal if moving it to the other partition does not violate the area constraint),
  - (ii) move the chosen cell and **lock it** in the destination partition,
  - (iii) update the gain values of the neighbors of the moved cell and update their positions in the bucket, and
  - (iv) record the gain and the current cutsize.
- At the end of the pass, identify and accept the first K moves that lead to minimum cutsize discovered during the entire pass.
- **If the initial cutsize has reduced during the current pass**
  - attempt another pass using the best solution discovered from the current pass as initial solution; otherwise terminate.

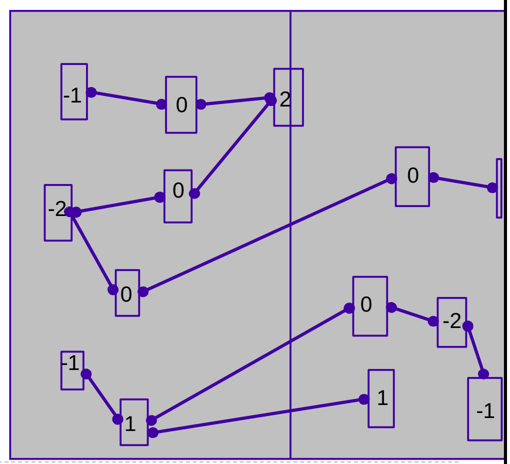


# FM Partitioning Example - 1

- Moves are based on object gain
  - The amount of change in cut crossings that will occur if an object is moved from its current partition into the other partition
- each object is assigned a gain
  - objects are put into a sorted gain list
- the object with the highest gain from the larger of the two sides is selected and moved.
  - the moved object is "locked"
  - gains of "touched" objects are recomputed
  - gain lists are resorted

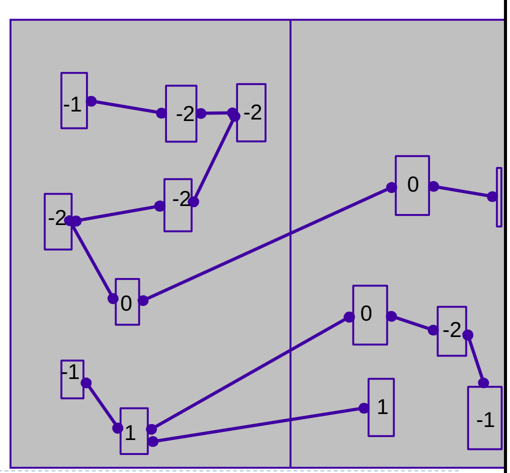


# FM Partitioning Example - 2

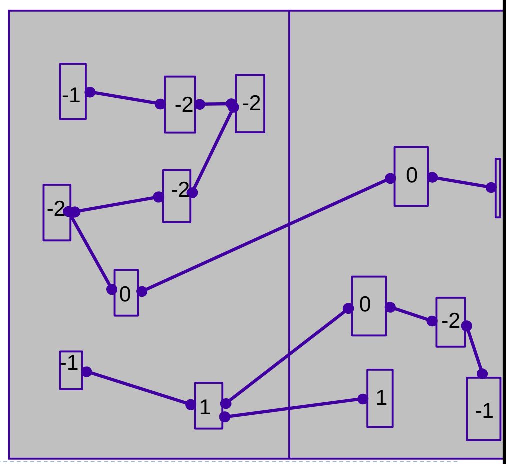





# FM Partitioning Example - 3

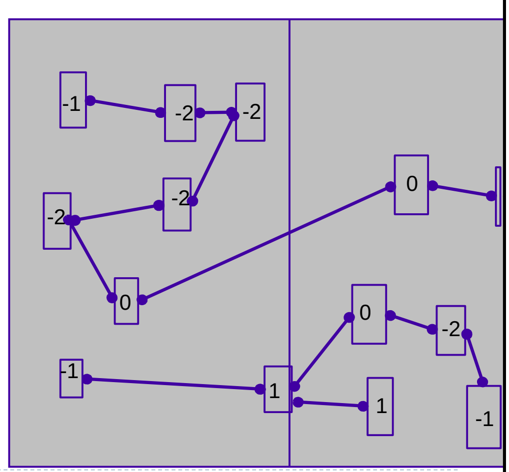


## FM Partitioning Example - 4

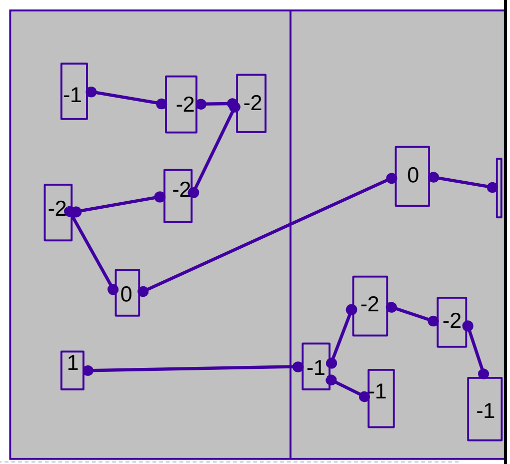





## FM Partitioning Example - 5

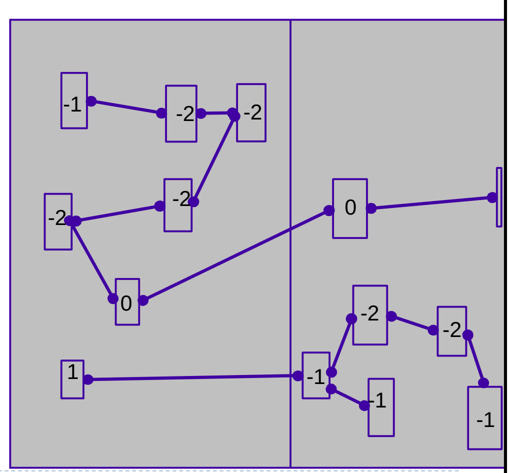


# FM Partitioning Example - 6

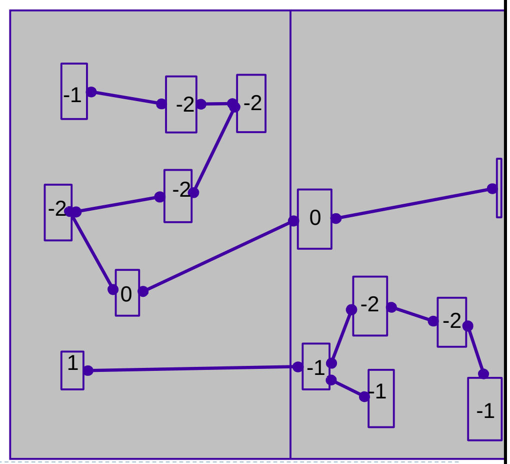





# FM Partitioning Example - 7

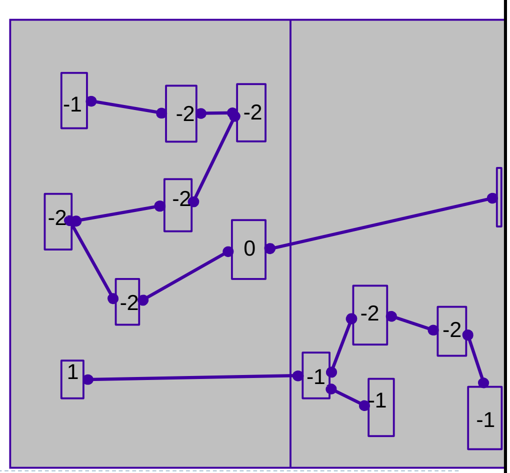


## FM Partitioning Example - 8

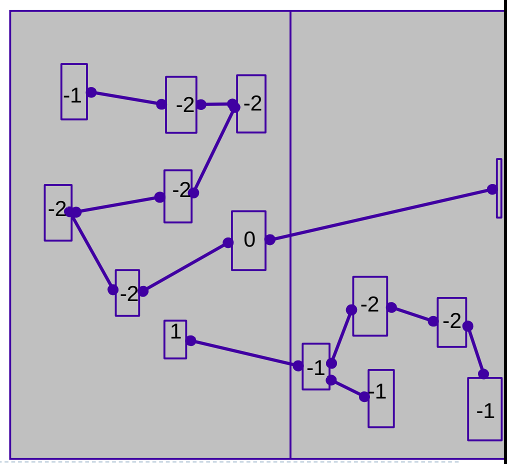





## FM Partitioning Example - 9

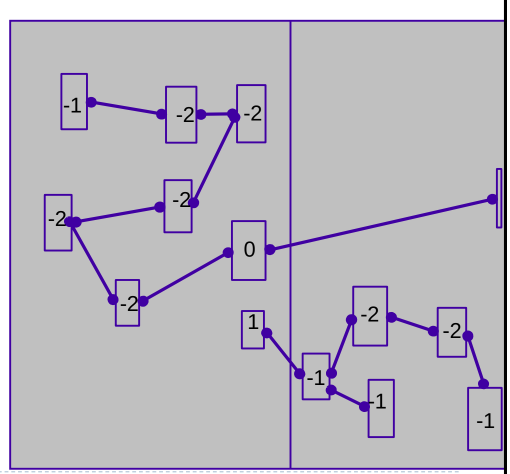


## FM Partitioning Example - 10

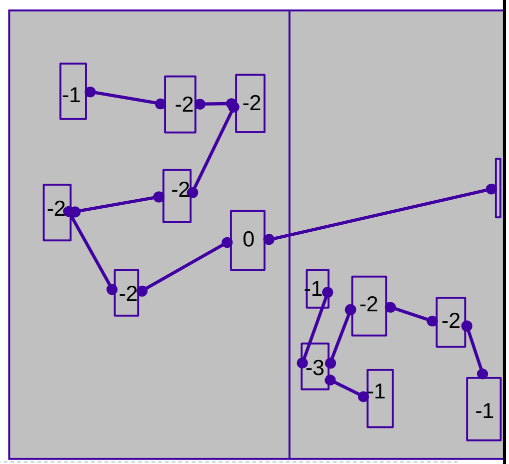





## FM Partitioning Example - 11

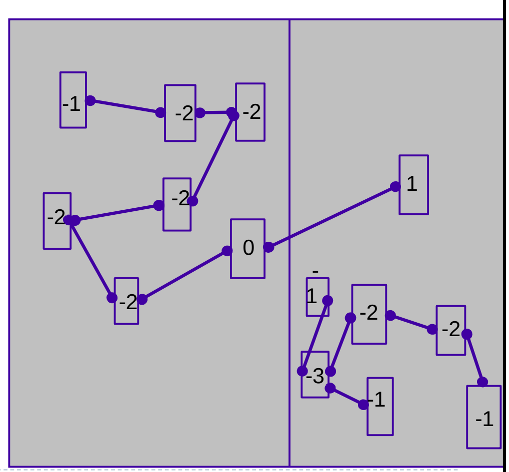


# FM Partitioning Example - 12

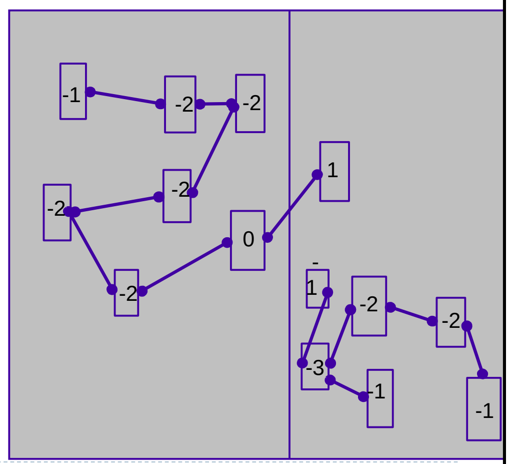





# FM Partitioning Example - 13

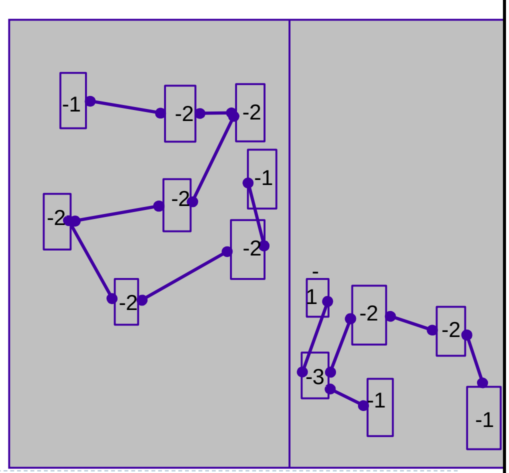


# FM Partitioning Example - 14






# FM Partitioning Example - 15



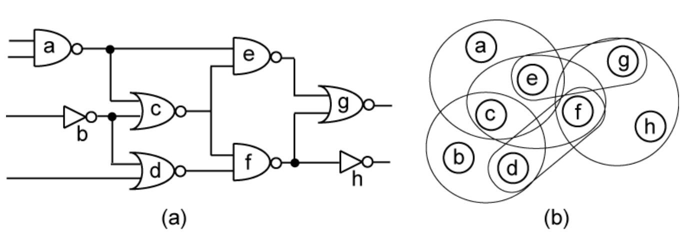
## Complexity of FM

- For each pass,
  - Constant time to find the best vertex to move.
  - After each move, time to update gain buckets is proportional to degree of vertex moved.
  - Total time is O(n), where n is total number of nets
- Number of passes is usually small.



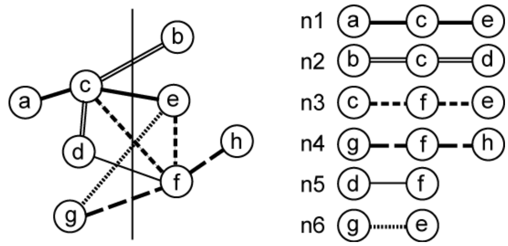
## Fiduccia-Mattheyses Algorithm Example

- Perform FM algorithm on the following circuit:
  - Area constraint = [3,5]
  - Break ties in alphabetical order.



# Fiduccia-Mattheyses Algorithm Example

- Initial Partitioning
  - Random initial partitioning is given.

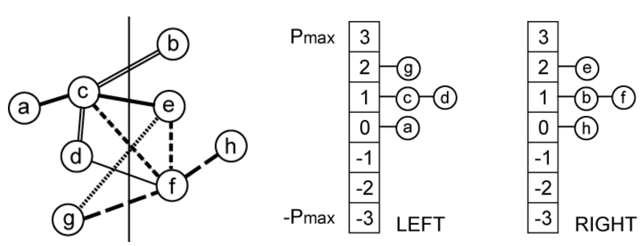





# Fiduccia-Mattheyses Algorithm Example

- Gain Computation and Bucket Set Up

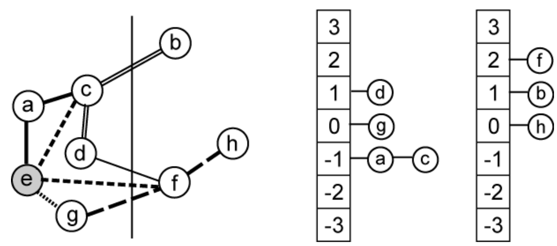
cell $c$: $c$ is contained in net $n_1 = \{a, c, e\}$, $n_2 = \{b, c, d\}$, and $n_3 = \{c, f, e\}$. $n_3$ contains $c$ as its only cell located in the left partition, so $FS(c) = 1$. In addition, none of these three nets are located entirely in the left partition. So, $TE(c) = 0$. Thus, $gain(c) = 1$.



# Fiduccia-Mattheyses Algorithm Example

- First Move

move 1: From the initial bucket we see that both cell $g$ and $e$ have the maximum gain and can be moved without violating the area constraint. We move $e$ based on alphabetical order. We update the gain of the unlocked neighbors of $e$, $N(e) = \{a, c, g, f\}$, as follows: $gain(a) = FS(a) - TE(a) = 0 - 1 = -1$, $gain(c) = 0 - 1 = -1$, $gain(g) = 1 - 1 = 0$, $gain(f) = 2 - 0 = 2$.

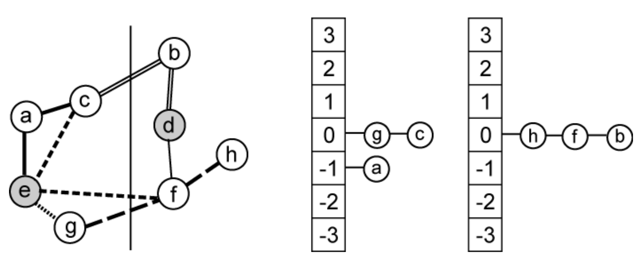





# Fiduccia-Mattheyses Algorithm Example

- Second Move

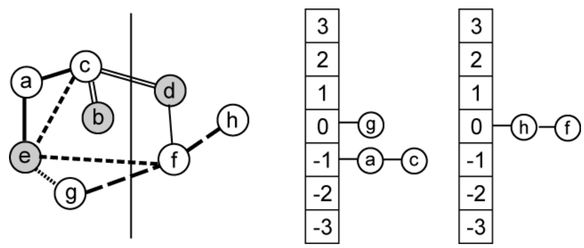
move 2: $f$ has the maximum gain, but moving $f$ will violate the area constraint. So we move $d$. We update the gain of the unlocked neighbors of $d$, $N(d) = \{b, c, f\}$, as follows: $gain(b) = 0 - 0 = 0$, $gain(c) = 1 - 1 = 0$, $gain(f) = 1 - 1 = 0$.



# Fiduccia-Mattheyses Algorithm Example

- Third Move

move 3: Among the maximum gain cells $\{g, c, h, f, b\}$, we choose $b$ based on alphabetical order. We update the gain of the unlocked neighbors of $b$, $N(b) = \{c\}$ as follows: $gain(c) = 0 - 1 = -1$.

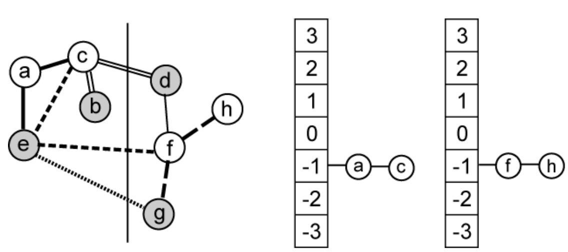





# Fiduccia-Mattheyses Algorithm Example

- Fourth Move

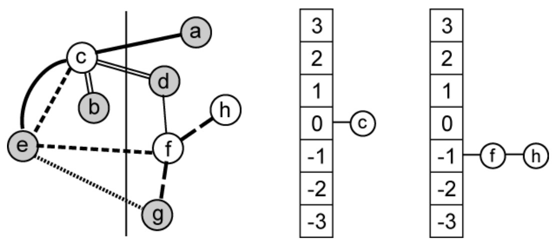
move 4: Among the maximum gain cells $\{g, h, f\}$, we choose $g$ based on the area constraint. We update the gain of the unlocked neighbors of $g$, $N(g) = \{f, h\}$, as follows: $gain(f) = 1 - 2 = -1$, $gain(h) = 0 - 1 = -1$.



# Fiduccia-Mattheyses Algorithm Example

- Fifth Move

move 5: We choose $a$ based on alphabetical order. We update the gain of the unlocked neighbors of $a$, $N(a) = \{c\}$, as follows: $gain(c) = 0 - 0 = 0$.

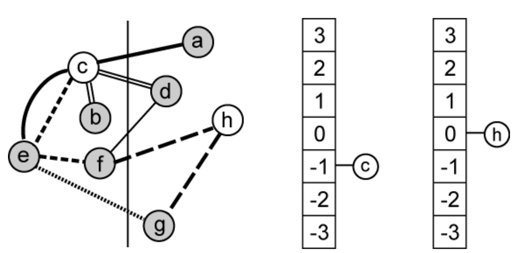





# Fiduccia-Mattheyses Algorithm Example

- Sixth Move

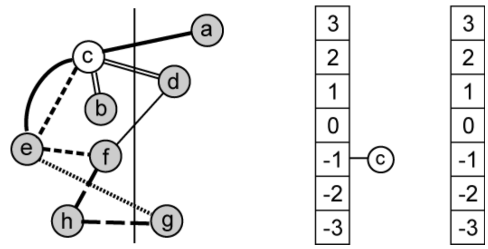
move 6: We choose $f$ based on the area constraint and alphabetical order. We update the gain of the unlocked neighbors of $f$, $N(f) = \{h, c\}$, as follows: $gain(h) = 0 - 0 = 0$, $gain(c) = 0 - 1 = -1$.



# Fiducccia-Mattheyses Algorithm Example

- Seventh Move

move 7: We move $h$. $h$ has no unlocked neighbor.

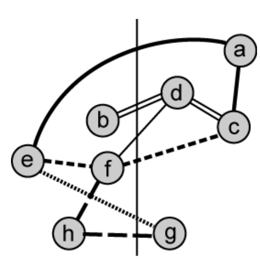





# Fiduccia-Mattheyses Algorithm Example

- Last Move

move 8: We move $c$.



# Fiduccia-Mattheyses Algorithm Example

- Summary
  - Found three best solutions.
    - Cutsize reduced from 6 to 3.
    - Solutions after move 2 and 4 are better balanced.

| $i$ | cell | $g(i)$ | $\sum g(i)$ | cutsize |
|---|---|---|---|---|
| 0 | - | - | - | 6 |
| 1 | $e$ | 2 | 2 | 4 |
| **2** | $\boldsymbol{d}$ | **1** | **3** | **3** |
| **3** | $\boldsymbol{b}$ | **0** | **3** | **3** |
| **4** | $\boldsymbol{g}$ | **0** | **3** | **3** |
| 5 | $a$ | -1 | 2 | 4 |
| 6 | $f$ | -1 | 1 | 5 |
| 7 | $h$ | 0 | 1 | 5 |
| 8 | $c$ | -1 | 0 | 6 |