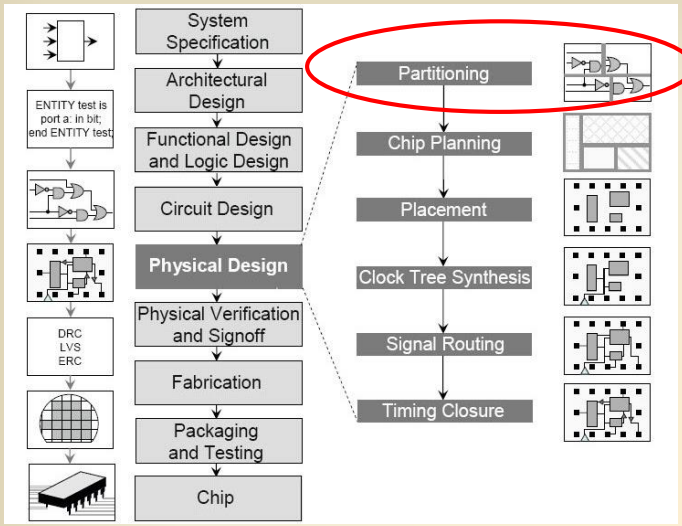
EE6094
CAD for VLSI Design

Chapter 8
Partition

Spring 2022
Andy, Yu-Guang Chen
Assistant Professor, Department of EE
National Central University
andyygchen@ee.ncu.edu.tw

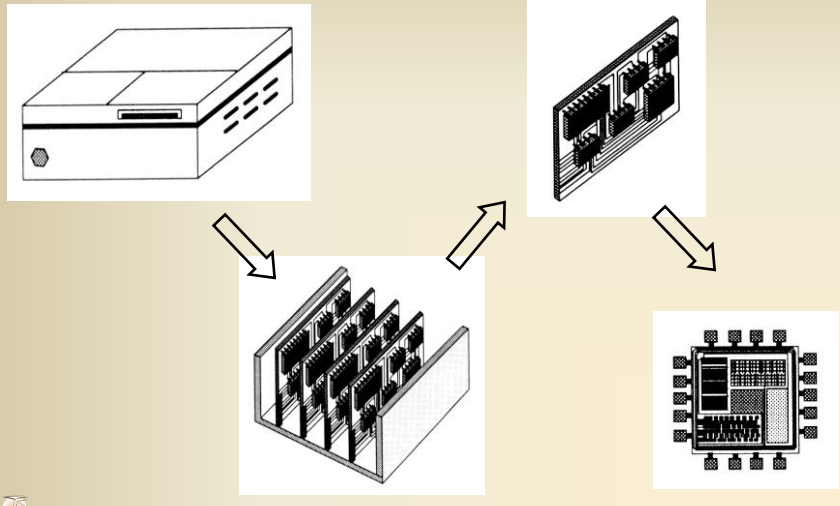
Where are we now…



Outline

- Partition an overview
- Kernighagn & Lin heuristic (KL)
- Fiduccia-Mattheyses heuristic (FM)
- Simulated annealing based method (SA)
- Multilevel circuit partitioning
- Partitioning for wirelength minimization
- Clustering for delay minimization

System Hierarchy

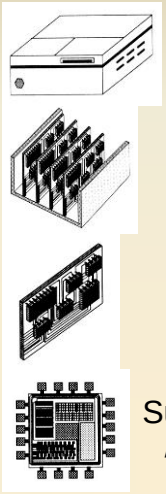


Levels of Partitioning

System Level Partitioning

Board Level Partitioning

Chip Level Partitioning



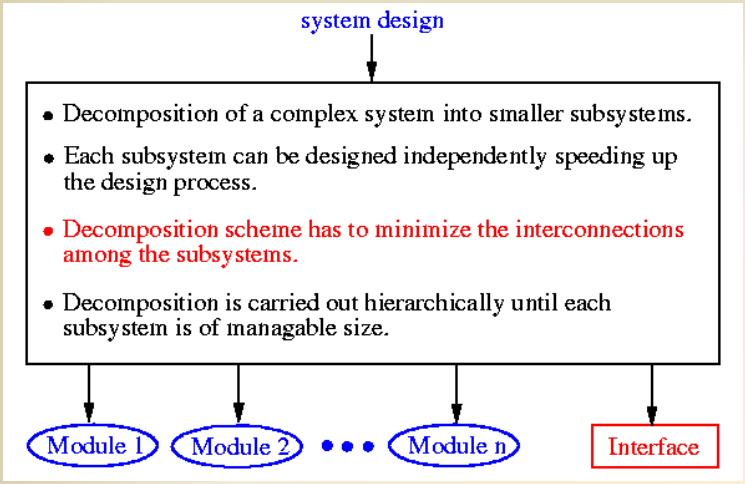
System

PCBs

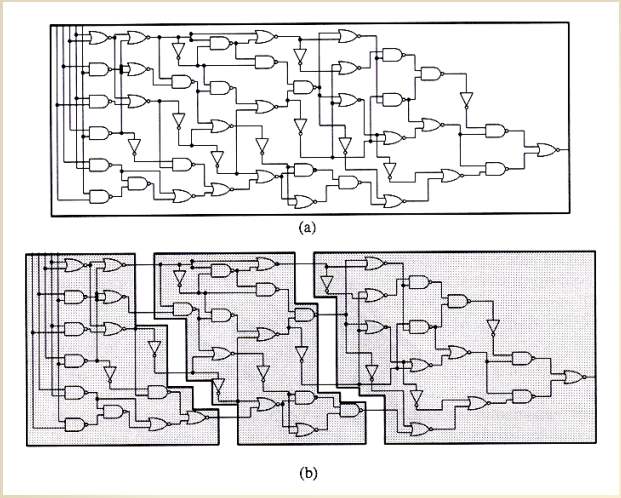
Chips

Subcircuits / Blocks

Partitioning

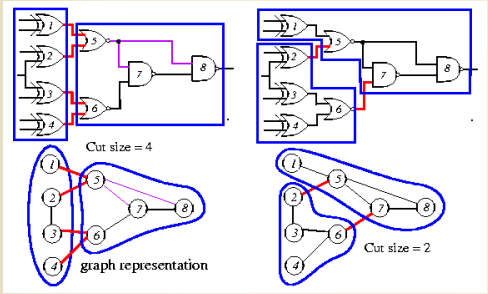


Partitioning of a Circuit



Circuit Partitioning

- **Objective:** Partition a circuit into parts such that every component is within a prescribed range and the # of connections among the components is minimized.
  - More constraints are possible for some applications.
- Cutset? Cut size? Size of a component?

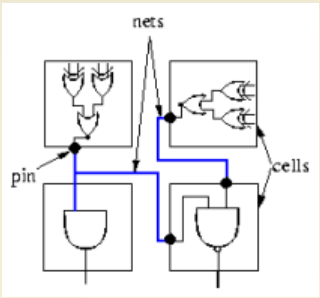


Importance of Circuit Partitioning

- Divide-and-conquer methodology
  - The most effective way to solve problems of high complexity
  - Ex: 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 (ex: Quickturn), or multiple special-purpose processors (ex: 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
- …

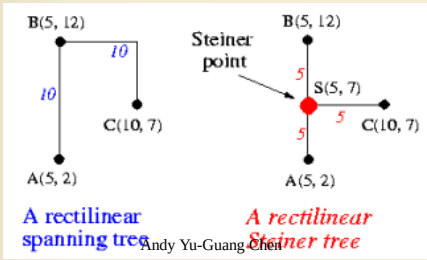
Basic Definitions

- **Cell:** a logic block used to build larger circuits.
- **Pin:** a wire (metal or polysilicon) to which another external wire can be connected.
- **Nets:** a collection of pins which must be electronically connected.
- **Netlist:** a list of all nets in a circuit.



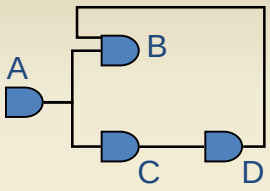
Basic Definitions

- **Manhattan distance:** If two points (pins) are located at coordinates (x_1, y_1) and (x_2, y_2), the Manhattan distance between them is given by $d_{12} = |x_1-x_2| + |y_1-y_2|$.
- **Rectilinear spanning tree:** a spanning tree that connects its pins using Manhattan paths.
- **Steiner tree:** a tree that connects its pins, and additional points **(Steiner points)** are permitted to used for the connections.

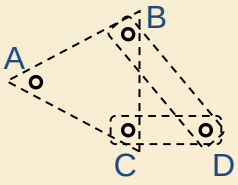


Circuit Representation

- 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)



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 (ex: 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

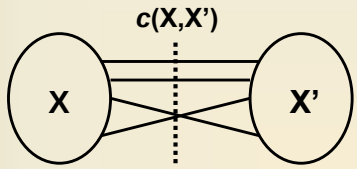
Problem Definition: Partitioning

- ***k*-way partitioning:** Given a graph $G(V, E)$, where each vertex $v \in V$ has a **size** $s(v)$ and each edge $e \in E$ has a **weight** $w(e)$, the problem is to divide the set V into k disjoint subsets $V_1, V_2, \ldots, V_k$, such that an objective function is optimized, subject to certain constraints.
- **Bounded size constraint:** The size of the i-th subset is bounded by B_i ($\sum_{v \in V_i} s(v) \leq B_i$).
  - Is the partition balanced?
- **Min-cut cost between two subsets:** Minimize $\sum_{\forall e=(u,v) \wedge p(u) \neq p(v)} w(e)$, where $p(u)$ is the partition # of node u.
- The 2-way, balanced partitioning problem is NP-complete, even in its simple form with identical vertex sizes and unit edge weights.

Circuit Partitioning Formulation

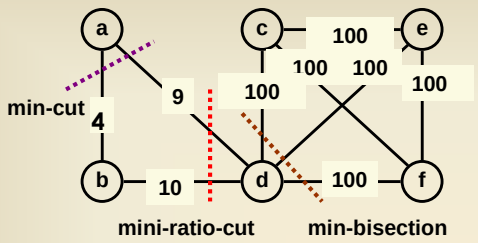
Bi-partitioning formulation:

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: cut size = 13
Min-bisection cut: cut size = 300
Min-ratio-cut: cut size = 19

Ratio-cut helps to identify natural clusters

Circuit Partitioning Formulation

- General multi-way partitioning formulation:
  - Partition a network N into N_1, N_2, …, 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:

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

Partitioning Algorithms

- Iterative partitioning algorithms
- Spectral based partitioning algorithms
- Deterministic vs. probabilistic algorithms
- Net partitioning vs. module partitioning
- Multi-way partitioning
- Multi-level partitioning
- Further study in partitioning techniques (timing-driven …)

Iterative Partitioning Algorithms

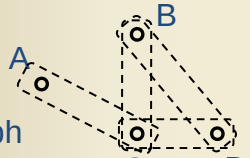
- Greedy iterative improvement method
  - [Kernighan-Lin 1970]
  - [Fiduccia-Mattheyses 1982]
  - [Krishnamurthy 1984]
- Simulated Annealing
  - [Kirkpartrick-Gelatt-Vecchi 1983]
  - [Greene-Supowit 1984]

Kernighan-Lin Algorithm

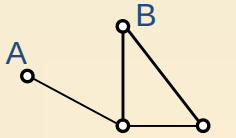
- Kernighan and Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, Feb. 1970.
- An **iterative**, **2-way**, **balanced** partitioning (bi-sectioning) heuristic.
- Till the cut size keeps decreasing
  - Vertex pairs which give the largest decrease or the smallest increase in cut size are exchanged.
  - These vertices are then **locked** (and thus are prohibited from participating in any further exchanges).
  - This process continues until all the vertices are locked.
  - Find the set with the largest partial sum for swapping.
  - Unlock all vertices.

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 is called a graph



Hypergraph Representation



Graph Representation

Problem Formulation

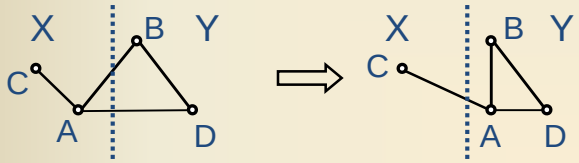
- Input: A graph with
  - Set vertices V ($|V| = 2n$)
  - Set of edges E ($|E| = m$)
  - Cost c_{AB} for each edge $\{A, B\}$ in E
- Output: 2 partitions X & Y such that
  - Total cost of edges cut is minimized
  - Each partition has n vertices
- This problem is NP-Completeness!!!!!

A Trivial Approach

- Try all possible bisections, find the best one
- If there are *2n* vertices,
  # of possibilities = $C_n^{2n} / 2$ = $(2n)! / (2 \times (n!^2)) = n^{O(n)}$
- 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}
- For 100 vertices, 5×10^{28} possibilities
  - Need 1.59×10^{13} years if one can try 100M possibilities per second

Idea of KL Algorithm

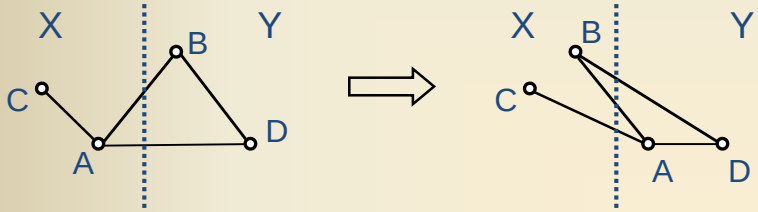
- D_A = Decrease in cut value if moving A
  - External cost (connection) E_A – Internal cost I_A
  - Moving node a from block X to block Y would increase the value of the cutset by E_A and decrease it by I_A



$D_A = 2-1 = 1$
$D_B = 1-1 = 0$

Idea of KL Algorithm

- Note that we want to balance two partitions
- If switch A & B, gain(A,B) = $D_A+D_B-2c_{AB}$
  - c_{AB} : edge cost for AB

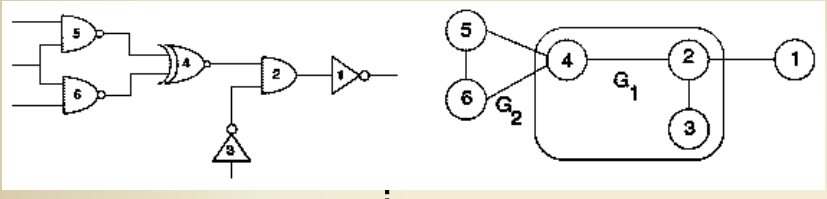


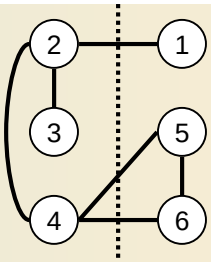
gain(A,B) = 1+0-~~2~~ ~~1~~ -1

Idea of KL Algorithm

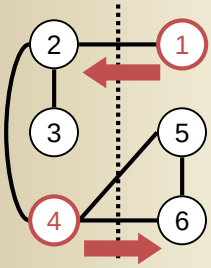
- Start with any initial legal partitions X and Y
- A **pass** (exchanging each vertex exactly once) is described below:
  1. For i := 1 to n do
     From the unlocked (unexchanged) vertices,
     choose a pair (A,B) s.t. gain(A,B) is largest
     Exchange and Lock A, B
     Let g_i = gain(A,B)
  2. Find the k s.t. $G=g_1+...+g_k$ is maximized
  3. Switch the first k pairs
- Repeat the pass until there is no improvement (G=0)

KL Example – 1



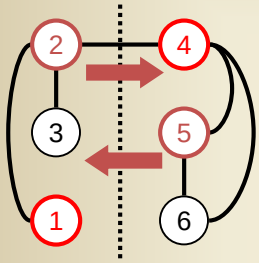


KL Example – 2



- D_1 = 1-0 = 1
- D_2 = 1-2 = -1
- D_3 = 0-1 = -1
- D_4 = 2-1 = 1
- D_5 = 1-1 = 0
- D_6 = 1-1 = 0
- gain(2, 1) = -1 + 1 – 2 = -2
- gain(2, 5) = -1 + 0 – 0 = -1
- gain(2, 6) = -1 + 0 – 0 = -1
- gain(3, 1) = -1 + 1 – 0 = 0
- gain(3, 5) = -1 + 0 – 0 = -1
- gain(3, 6) = -1 + 0 – 0 = -1
- gain(4, 1) = 1 + 1 – 0 = 2 g_1
- gain(4, 5) = 1 + 0 – 2 = -1
- gain(4, 6) = 1 + 0 – 2 = -1

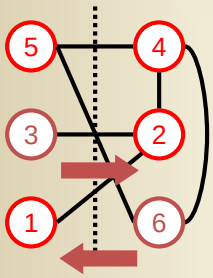
KL Example – 3

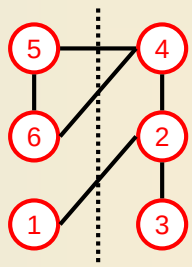


- D_2 = 1 – 2 = -1
- D_3 = 0 – 1 = -1
- D_5 = 0 – 2 = -2
- D_6 = 0 – 2 = -2
- gain(2, 5) = -1 + -2 – 0 = -3 g_2
- gain(2, 6) = -1 + -2 – 0 = -3
- gain(3, 5) = -1 + -2 – 0 = -3
- gain(3, 6) = -1 + -2 – 0 = -3

KL Example – 4

- D_3 = 1 – 0 = 1
- D_6 = 1 – 1 = 0
- gain(3, 6) = 1 – 0 – 0 = 1 g_3





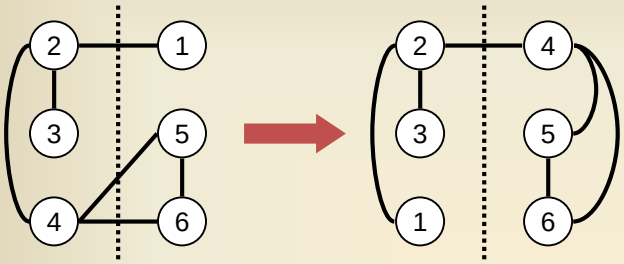
$g_1 = 2,\ g_2 = -3,\ g_3 = 1$

Then ???
We need to find k

- ❑ k = 1: g_1 = 2
- ❑ k = 2: $g_1 + g_2$ = -1
- ❑ k = 3: $g_1 + g_2 + g_3$ = 0

KL Example – 5

- Since k = 1, we only take 1 move → swap (4, 1)



- Finally, we get the result?

NO! We only pass 1 iteration!

- How should I do to get an end?

Time Complexity of KL

- For each pass KL
  - $O(n^2)$ 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

Outline

- Partition an overview
- Kernighagn & Lin heuristic (KL)
- Fiduccia-Mattheyses heuristic (FM)
- Simulated annealing based method (SA)
- Multilevel circuit partitioning
- Partitioning for wirelength minimization
- Clustering for delay minimization

Recap of KL Algorithm

Pair-wise exchange of nodes to reduce cut size

Allow cut size to increase temporarily within a pass

Compute the gain of a swap

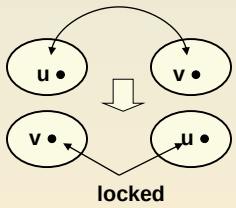
  Repeat

    Perform a feasible swap of max gain

    Mark swapped nodes "locked"

    Update swap gains

  Until no feasible swap



Find max prefix partial sum in gain sequence $g_1, g_2, \ldots, g_m$

Make corresponding swaps permanent

Start another pass if current pass reduces the cut size (usually converge after a few passes)

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
```

Time Complexity

- 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?

A 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

Extensions of K-L Algorithm

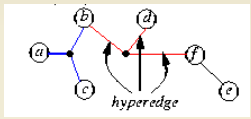
- **Unequal sized subsets** (assume $n_1 < n_2$)
  1. Partition: $|A| = n_1$ and $|B| = n_2$.
  2. Add n_2-n_1 dummy vertices to set A. Dummy vertices have no connections to the original graph.
  3. Apply the Kernighan-Lin algorithm.
  4. Remove all dummy vertices.
- **Unequal sized "vertices"**
  1. Assume that the smallest "vertex" has unit size.
  2. Replace each vertex of size s with s vertices which are fully connected with edges of infinite weight.
  3. Apply the Kernighan-Lin algorithm.
- ***k*-way partition**
  1. Partition the graph into k equal-sized sets.
  2. Apply the Kernighan-Lin algorithm for each pair of subsets.
  3. Time complexity? Can be reduced by recursive bi-partition.

Drawbacks of the Kernighan-Lin Heuristic

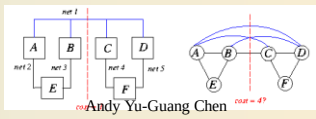
- The K-L heuristic **handles only unit vertex weights**.
  - Vertex weights might represent block sizes, different from blocks to blocks.
  - Reducing a vertex with weight $w(v)$ into a clique with $w(v)$ vertices and edges with a high cost increases the size of the graph substantially.
- The K-L heuristic **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**, $O(n^3)$.

Coping with Hypergraph

- A hypergraph $H=(N, L)$ consists of a set N of vertices and a set L of hyperedges, where each hyperedge corresponds to a **subset** N_i of distinct vertices with $|N_i| \geq 2$.



- Schweikert and Kernighan, "A proper model for the partitioning of electrical circuits," 9th Design Automation Workshop, 1972.
- 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 the same weight for all edges.



Fiduccia-Mattheyses Heuristic

- Fiduccia and Mattheyses, "A linear time heuristic for improving network partitions," DAC-82.
- New features to the K-L heuristic:
  - Aims at **reducing net-cut costs**; the concept of cutsize is extended to hypergraphs.
  - Only a **single vertex** is moved across the cut in a single move.
  - Vertices are weighted.
  - Can handle "unbalanced" partitions; a balance factor is introduced.
  - A special data structure is used to select vertices to be moved across the cut to improve running time.
  - **Time complexity** $O(P)$, where P is the total # of terminals.

Features of FM Algorithm

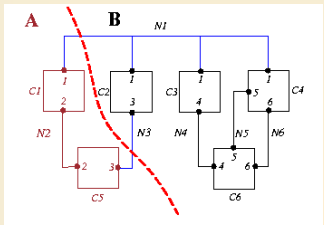
- Modification of KL Algorithm:
  - Can handle non-uniform vertex weights (areas)
  - Allow unbalanced partitions
  - Extended to handle hypergraphs
  - Clever way to select vertices to move, run much faster

Problem Formulation

- Input: A hypergraph with
  - Set vertices V. (|V| = n)
  - Set of hyperedges E (total # pins in netlist = p)
  - 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
- This problem is NP-Complete!!!!!

Cut

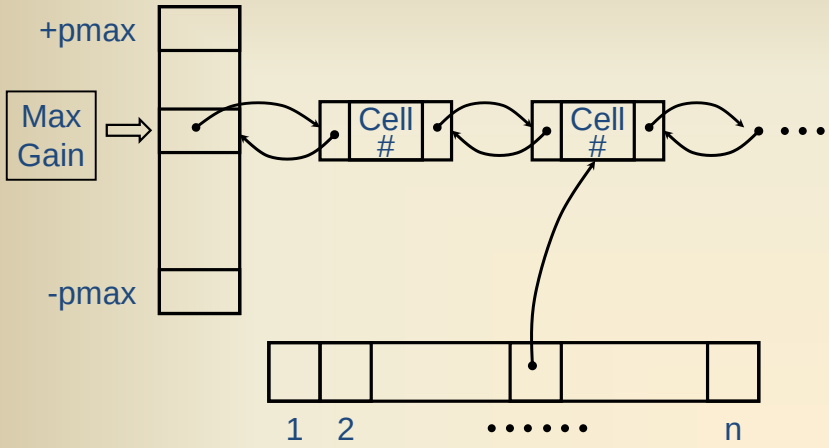
- **Cutstate** of a net:
  - Net 1 and Net 3 are **cut** by the partition.
  - Net 2, Net 4, Net 5, and Net 6 are **uncut**.
- **Cutset** = {Net 1, Net 3}.
- $|A|$ = size of $A = a_1+a_5$; $|B| = a_2+a_3+a_4+a_6$.
- **Balanced 2-way partition:** Given a fraction r, $0 < r < 1$, partition a graph into two sets A and B such that
  - $\frac{|A|}{|A|+|B|} \approx r$.
  - Size of the cutset is minimized.



Ideas of FM 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

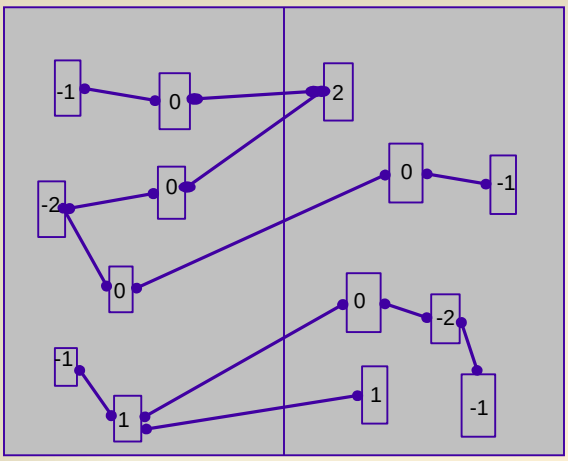


FM Partitioning

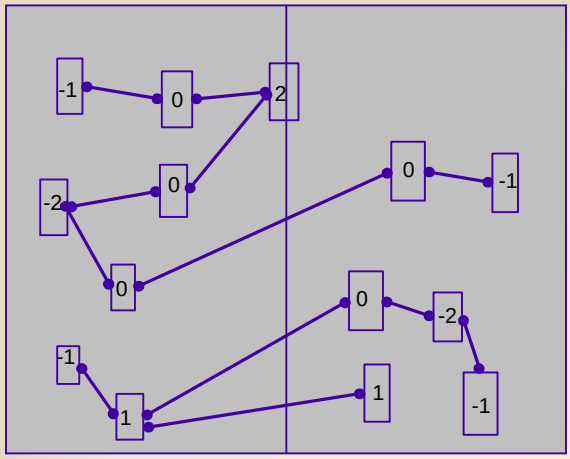
- Moves are made based on object gain
- Object Gain: The amount of change in cut crossings that will occur if an object is moved from its current partition into the other partition
- A pass description

  While there is unlocked object
  1. Each object is assigned a gain
  2. Objects are put into a sorted gain list
  3. The object with the highest gain from the larger of the two sides is selected and moved
  4. The moved object is "locked"
  5. Gains of "touched" objects are recomputed
  6. Gain lists are resorted
- Repeat the pass until there is no improvement

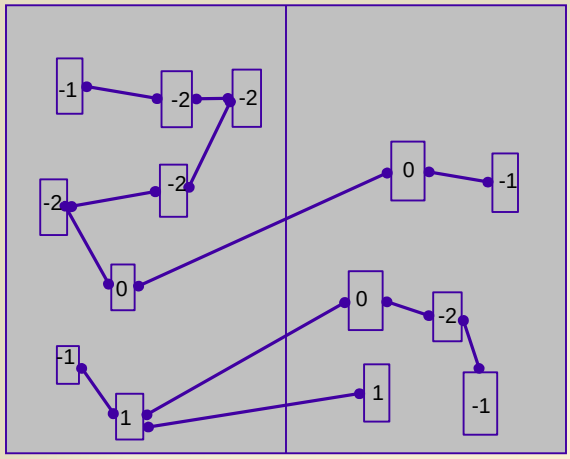
FM Partitioning



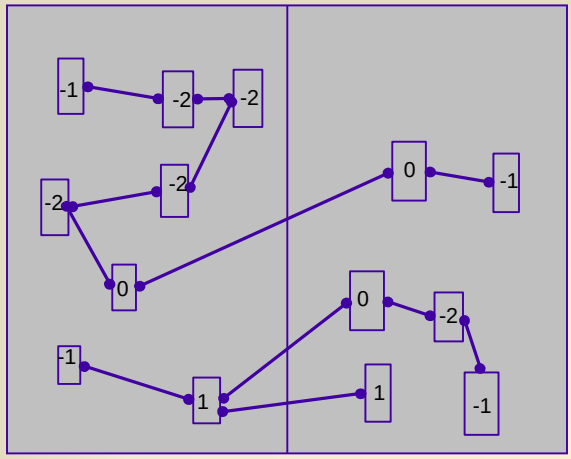
FM Partitioning



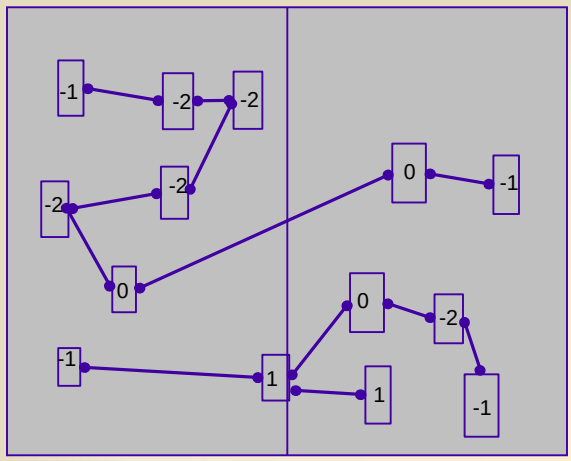
FM Partitioning



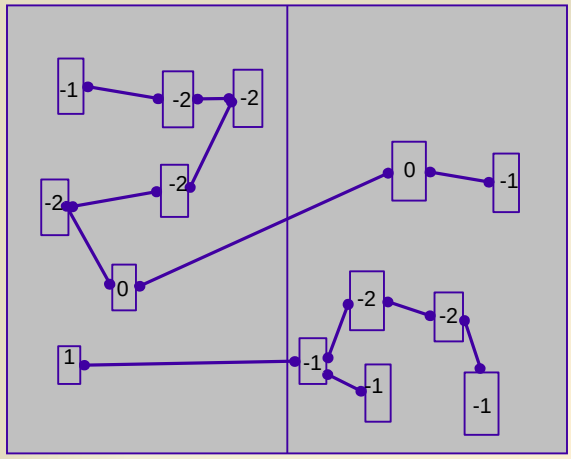
FM Partitioning



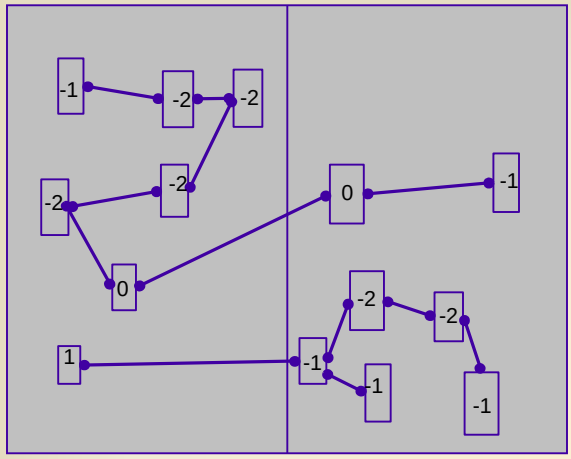
FM Partitioning



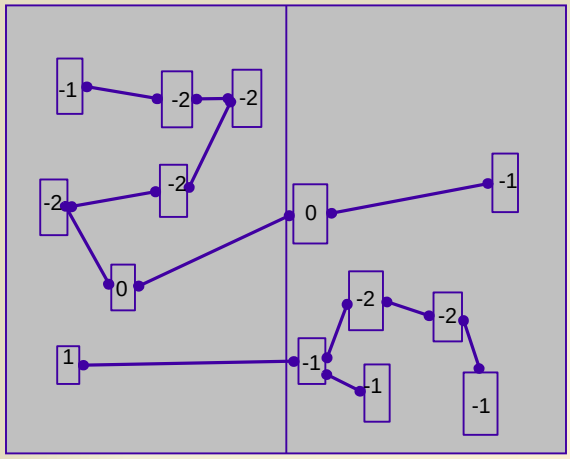
FM Partitioning



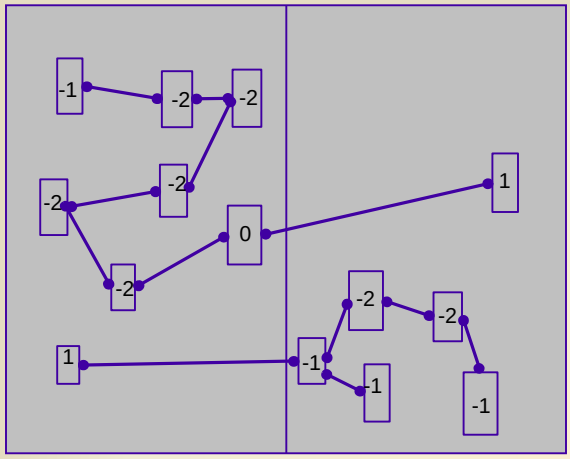
FM Partitioning



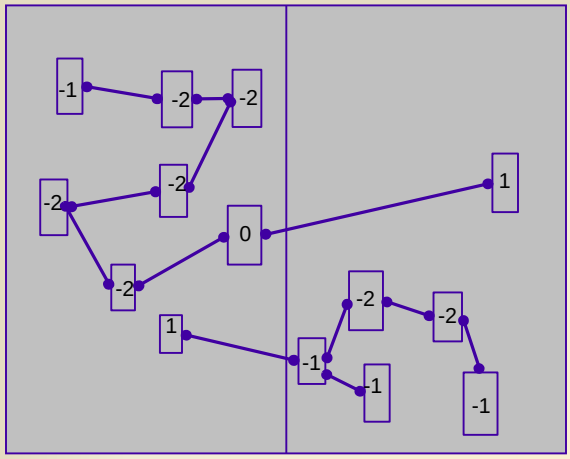
FM Partitioning



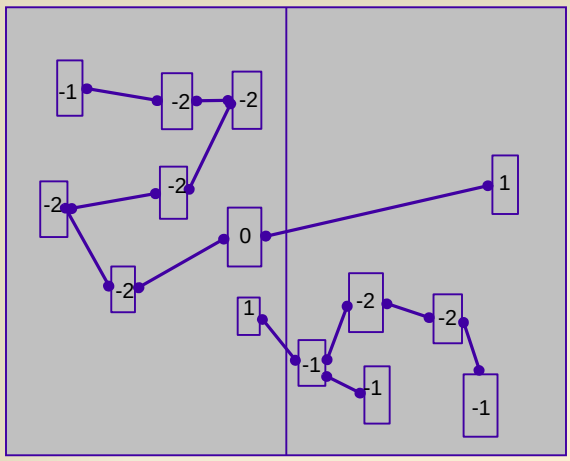
FM Partitioning



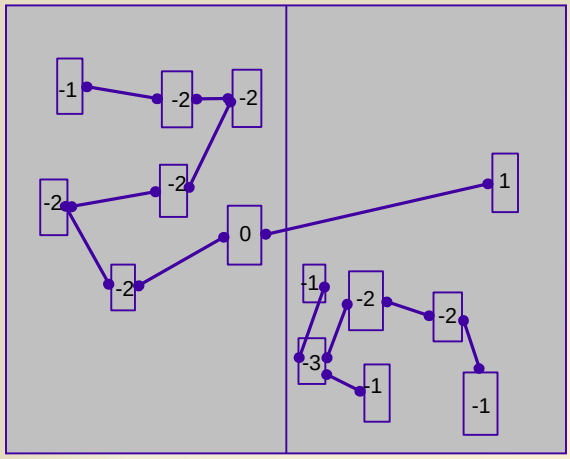
FM Partitioning



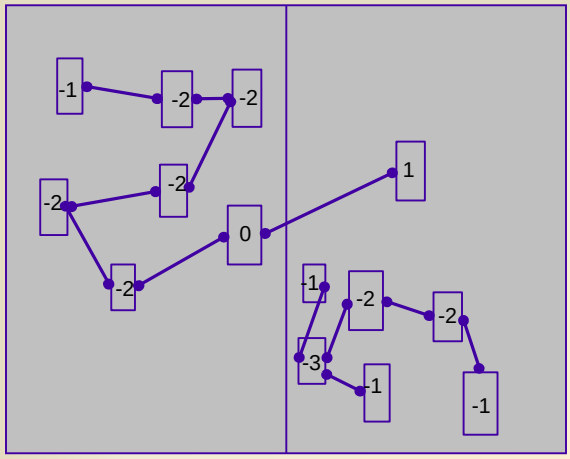
FM Partitioning



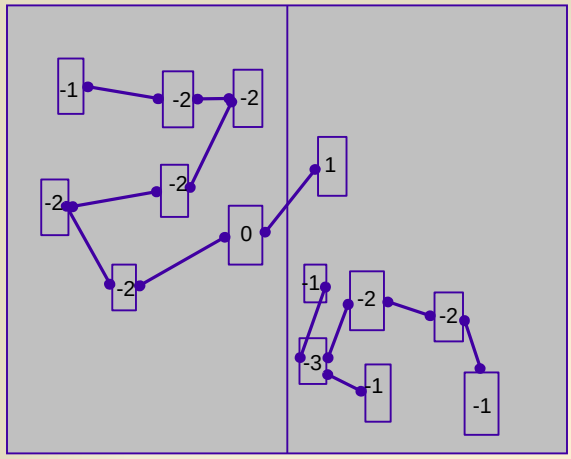
FM Partitioning



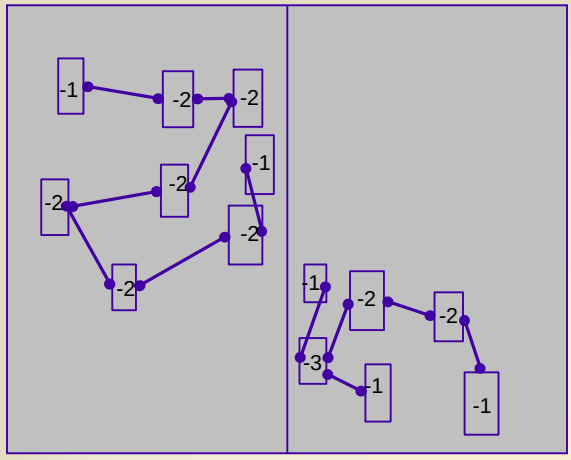
FM Partitioning



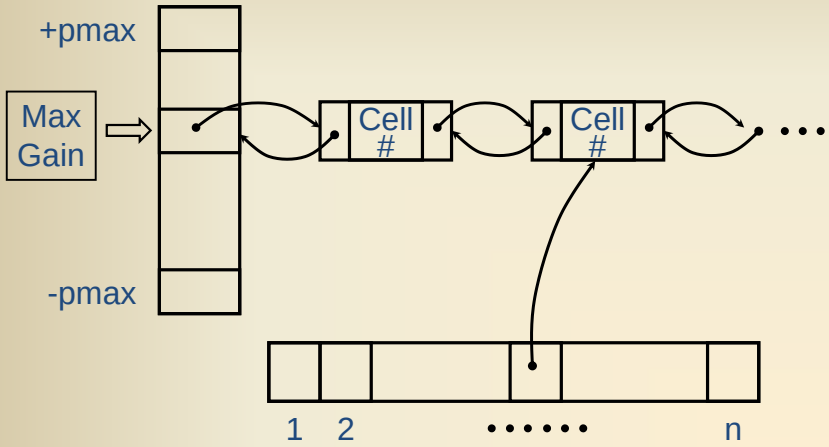
FM Partitioning



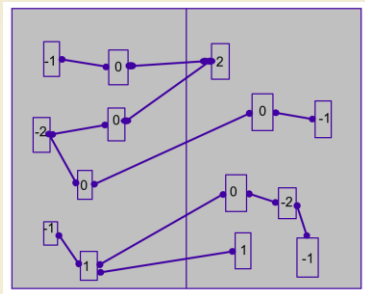
FM Partitioning



Gain Bucket Data Structure

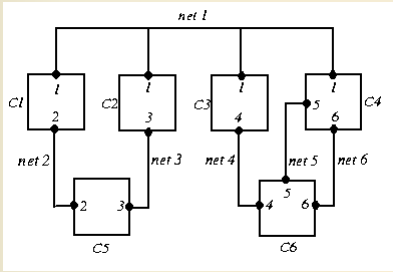


FM Partitioning

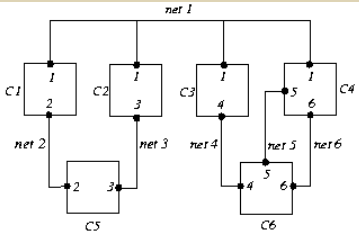


F-M Heuristic: Notation

- $n(i)$: # of cells in Net i; e.g., $n(1) = 4$.
- $s(i)$: size of Cell i.
- $p(i)$: # of pin terminals in Cell i; e.g., $p(6)=3$.
- C: total # of cells; e.g., $C=6$.
- N: total # of nets; e.g., $N=6$.
- P: total # of pins; $P = p(1) + \ldots + p(C) = n(1) + \ldots + n(N)$.



Input Data Structures



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

- Size of the network: $P = \sum_{i=1}^{6} n(i) = 14$
- Construction of the two arrays takes $O(P)$ time.

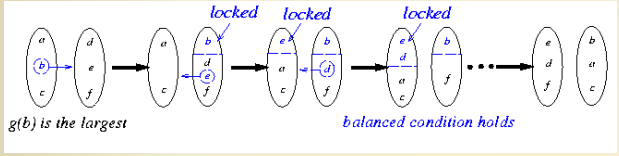
Basic Ideas: Balance and Movement

- Only move a cell at a time, preserving "balance."

$$\frac{|A|}{|A|+|B|} \approx r$$
$$rW - S_{max} \leq |A| \leq rW + S_{max},$$

where $W=|A|+|B|$; $S_{max}=\max_i s(i)$.

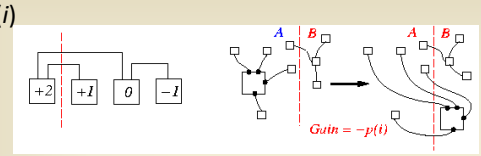
- $g(i)$: gain in moving cell i to the other set, i.e., size of **old** cutset - size of **new** cutset.



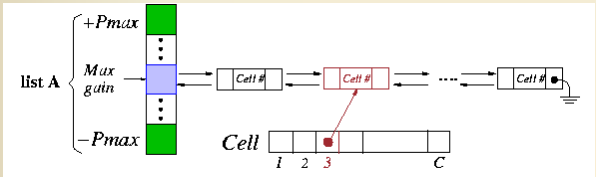
- Suppose g_i's: $g(b)$, $g(e)$, $g(d)$, $g(a)$, $g(f)$, $g(c)$ and the largest partial sum is $g(b)+g(e)+g(d)$. Then we should move b, e, d resulting two sets: {a, c, e, d}, {b, f}.

Cell Gains and Data Structure Manipulation

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



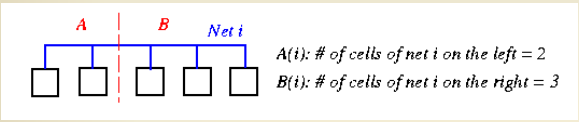
- Two "bucket list" structures, one for set A and one for set B ($P_{max} = \max_i p(i)$).



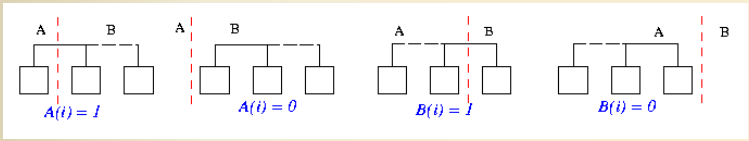
- $O(1)$-time operations: find a cell with Max Gain, remove Cell i from the structure, insert Cell i into another structure, update $g(i)$ to $g(i)+\Delta$, update the Max Gain pointer.

Net Distribution and Critical Nets

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



- **Critical Nets**: A net is critical if it has a cell which if moved will change its cutstate.
  - 4 cases: $A(i) = 0$ or 1, $B(i) = 0$ or 1.



- **Gain of a cell depends only on its critical nets.**

Computing Cell Gains

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

  $g(i) \leftarrow 0$;

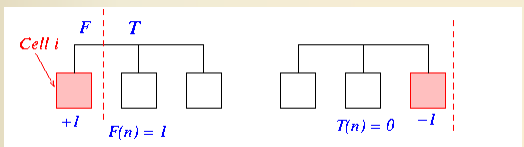
  $F \leftarrow$ the "from block" of Cell i;

  $T \leftarrow$ the "to block" of Cell i;

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

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

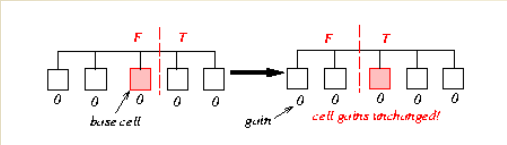
  if $T(n)=0$ **then** $g(i) \leftarrow g(i)-1$;



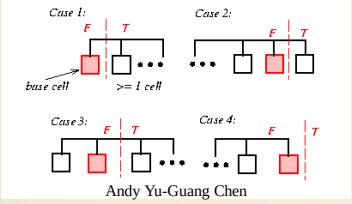
- Will show: Only need $O(P)$ time to maintain all cell gains in one pass.

Updating Cell Gains

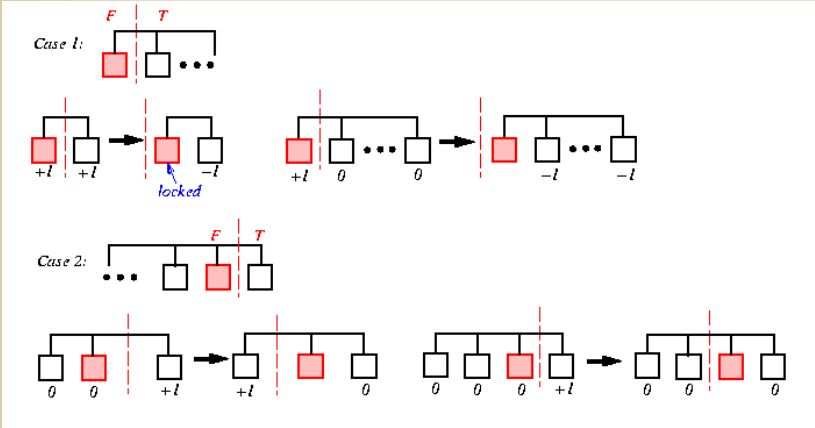
- To update the gains, we only need to look at those nets, connected to the base cell, which are critical **before** or **after** the move.
- **Base cell**: The cell selected for movement from one set to the other.



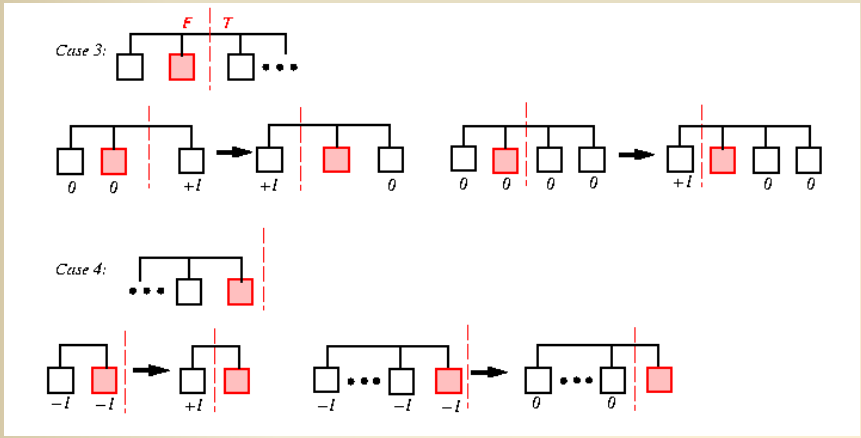
- Consider only the case where the base cell is in the left partition. The other case is similar.



Updating Cell Gains

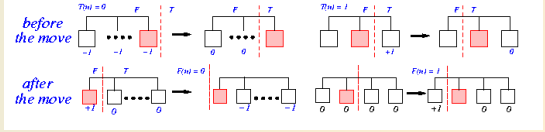


Updating Cell Gains



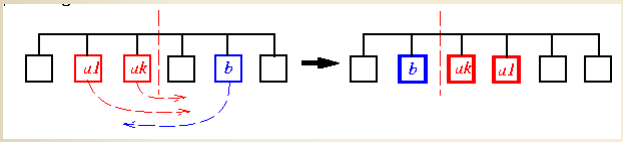
Algorithm for Updating Cell Gains

```
Algorithm: Update_Gain
1 begin  /* move base cell 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 Cell Gains

- Once a net has "locked' cells at both sides, the net will remain cut from now on.
- Suppose we move a_1, a_2, …, a_k from left to right, and then move b from right to left
- At most only moving a_1, a_2, …, a_k and b need updating!



- To update the cell gains, it takes $O(n(i))$ work for Net i.
- Total time = $n(1)+n(2)+…+n(N) = O(P)$.

Time 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(p), where p is total number of pins
- Number of passes is usually small

Extension by Krishnamurthy

- "An Improved Min-Cut Algorithm for Partitioning VLSI Networks", IEEE Trans. Computer, 33(5):438-446, 1984.

Tie-Breaking Strategy

- For each vertex, instead of having a gain bucket, a gain vector is used
- Gain vector is a sequence of potential gain values corresponding to numbers of possible moves into the future
- Therefore, r^{th} entry looks r moves ahead
- Time complexity is O(pr), where r is max # of look-ahead moves stored in gain vector
- If ties still occur, some researchers observe that LIFO order improves solution quality

Ratio Cut Objective by Wei and Cheng

- "Towards Efficient Hierarchical Designs by Ratio Cut Partitioning", ICCAD, pages 1:298-301, 1989.

Ratio Cut Objective

- It is not desirable to have some pre-defined ratio on the partition sizes
- Wei and Cheng proposed the Ratio Cut objective
- Try to locate natural clusters in circuit and force the partitions to be of similar sizes at the same time
- Ratio Cut $R_{XY} = C_{XY}/(|X| \bullet |Y|)$
- A heuristic based on FM was proposed

Sanchis Algorithm

- "Multiple-way Network Partitioning", IEEE Trans. Computers, 38(1):62-81, 1989.

Multi-Way Partitioning

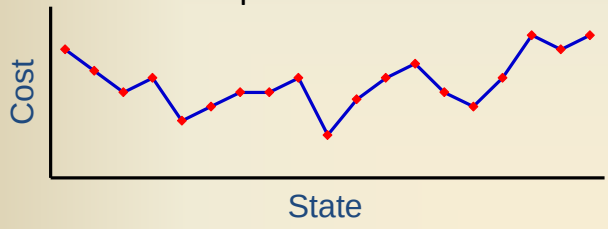
- Dividing into more than 2 partitions
- Algorithm by extending the idea of FM + Krishnamurthy

Greedy Algorithm

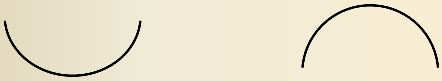
- A very simple technique for State Space Search Problem
- Start from any state
- Always move to a neighbor with the min cost (assume minimization problem)
- Stop when all neighbors have a higher cost than the current state

Problem with Greedy Algorithms

- Easily get stuck at local minimum
- Will obtain non-optimal solutions

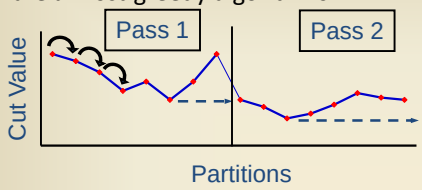


- Optimal only for convex (or concave for maximization) functions

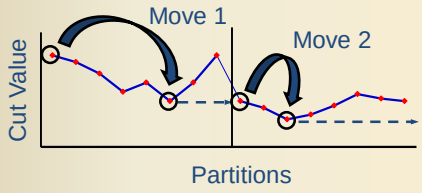


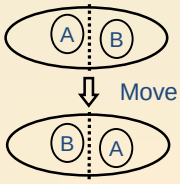
Greedy Nature of KL & FM

- KL and FM are *almost* greedy algorithms



- Purely greedy if we consider a pass as a "move"





Simulated Annealing

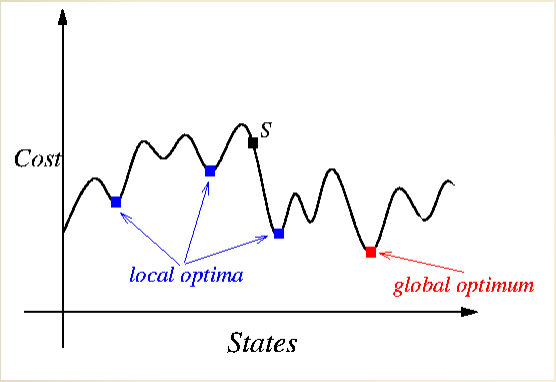
- Very general search technique
- Try to avoid being trapped in local minimum by making probabilistic moves
- Popularize as a heuristic for optimization by:
  - Kirkpatrick, Gelatt and Vecchi, "Optimization by Simulated Annealing", Science, 220(4598):498-516, May 1983

Basic Idea of SA

- Inspired by the *Annealing Process*:
  - The process of carefully cooling molten metals in order to obtain a good crystal structure
  - First, metal is heated to a very high temperature
  - Then slowly cooled
  - By cooling at a proper rate, atoms will have an increased chance to regain proper crystal structure
- Attaining a min cost state in simulated annealing is analogous to attaining a good crystal structure in annealing

Simulated Annealing

- Kirkpatrick, Gelatt, and Vecchi, "Optimization by simulated annealing," *Science*, May 1983.
- Greene and Supowit, "Simulated annealing without rejected moves," ICCD-84.



The SA Procedure

Let *t* be the initial temperature

Repeat

  Repeat

  - Pick a neighbor of the current state randomly
  - Let *c* = cost of current state

    Let *c'* = cost of the neighbour picked
  - If *c'* < *c*, then move to the neighbour (downhill move)
  - If *c'* > *c*, then move to the neighbour with probablility $e^{-(c'-c)/t}$ (uphill move)

  Until equilibrium is reached

  Reduce *t* according to cooling schedule

Until Freezing point is reached

Simulated Annealing Basics

- Non-zero probability for "up-hill" moves.
- Probability depends on
  1. magnitude of the "up-hill" movement
  2. total search time

$$Prob(S \rightarrow S') = \begin{cases} 1 & \text{if } \Delta C \leq 0 \quad /* \text{ "down - hill" moves } */ \\ e^{-\frac{\Delta C}{T}} & \text{if } \Delta C > 0 \quad /* \text{ "up - hill" moves } */ \end{cases}$$

- $\Delta C = cost(S') - Cost(S)$
- T: Control parameter (temperature)
- Annealing schedule: $T=T_0, T_1, T_2, \ldots$, where $T_i = r^i T_0$, $r < 1$.

Things to decide when using SA

- When solving a combinatorial problem, we have to decide:
  - The state space
  - The neighborhood structure
  - The cost function
  - The initial state
  - The initial temperature
  - The cooling schedule (how to change t)
  - The freezing point

Basic Ingredients for Simulated Annealing

- Analogy:

Physical system	Optimization problem
state	configuration
energy	cost function
ground state	optimal solution
quenching	iterative improvement
careful annealing	simulated annealing

- Basic Ingredients for Simulated Annealing:
  - **Solution space**
  - **Neighborhood structure**
  - **Cost function**
  - **Annealing schedule**

State Space Search Problem

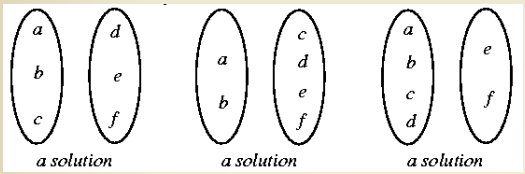
- Combinatorial optimization problems (like partitioning) can be thought as a State Space Search Problem
- A State is just a configuration of the combinatorial objects involved
- The State Space is the set of all possible states (configurations)
- A Neighborhood Structure is also defined (which states can one go in one step)
- There is a cost corresponding to each state
- Search for the min (or max) cost state

Common Cooling Schedules

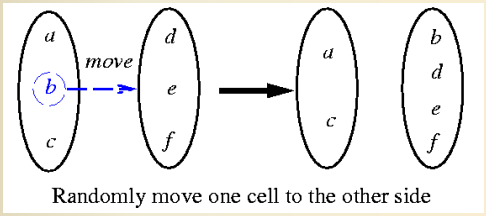
- Initial temperature, Cooling schedule, and freezing point are usually experimentally determined
- Some common cooling schedules:
  - $t = \alpha t$, where α is typically around 0.95
  - $t = e^{-\beta t}\, t$, where β is typically around 0.7
  - …

Partition by Simulated Annealing

- Kirkpatrick, Gelatt, and Vecchi, "Optimization by simulated annealing," *Science*, May 1983.
- **Solution space:** set of all partitions



- **Neighborhood structure:**



Randomly move one cell to the other side

Bisectioning using SA

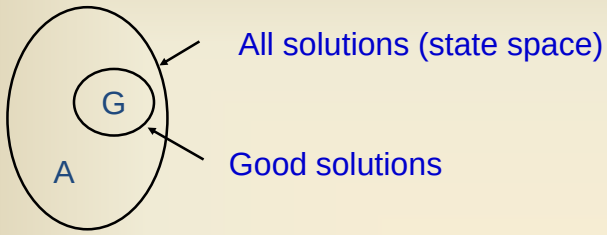
- Johnson, Aragon, McGeoch and Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation Part I, Graph Partitioning", Operations Research, 37:865-892, 1989.

SA – The Evaluation

- An extensive empirical study of Simulated Annealing versus Iterative Improvement Approaches
- Conclusion: SA is a competitive approach, getting better solutions than KL for random graphs
- Remarks:
  - Netlists are not random graphs, but sparse graphs with local structure
  - SA is too slow. So KL/FM variants are still most popular
  - Multiple runs of KL/FM variants with random initial solutions may be preferable to SA

The Use of Randomness

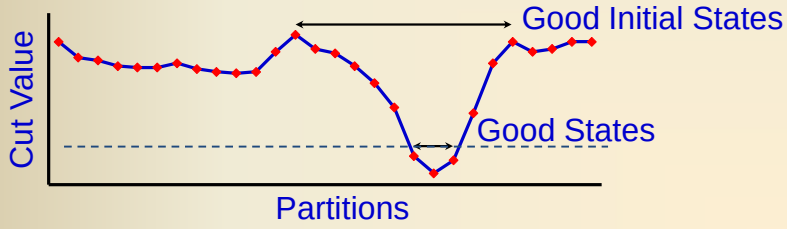
- For any partitioning problem:



- Suppose solutions are picked randomly
- If $|G|/|A| = r$, Pr(at least 1 good in 5/r trials) = $1-(1-r)^{5/r}$
- If $|G|/|A| = 0.001$, Pr(at least 1 good in 5000 trials) = $1-(1-0.001)^{5000} = 0.9933$

Adding Randomness to KL/FM

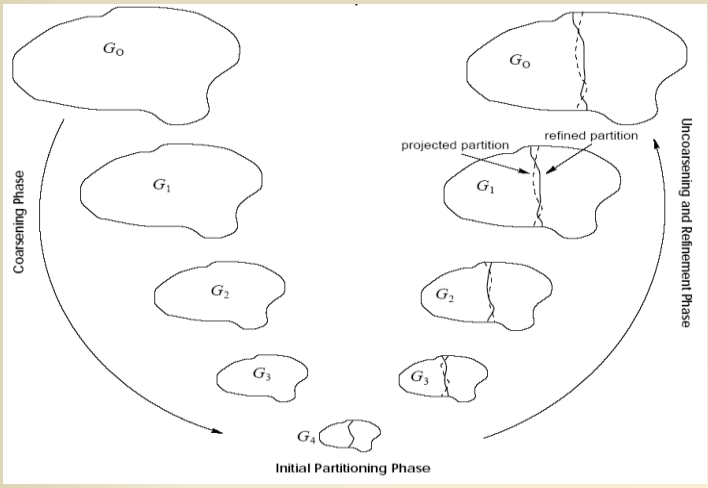
- In fact, # of good states are extremely few. Therefore, r is extremely small.
- Need extremely long time if just picking states randomly (without doing KL/FM).
- Running KL/FM variants several times with random initial solutions is a good idea.



Some Other Approaches

- KL/FM-SA Hybrid: Use KL/FM variant to find a good initial solution for SA, then improve that solution by SA at low temperature
- Tabu Search
- Genetic Algorithm
- Spectral Methods (finding Eigenvectors)
- Network Flows
- Quadratic Programming
- …

Multi-Level Partitioning

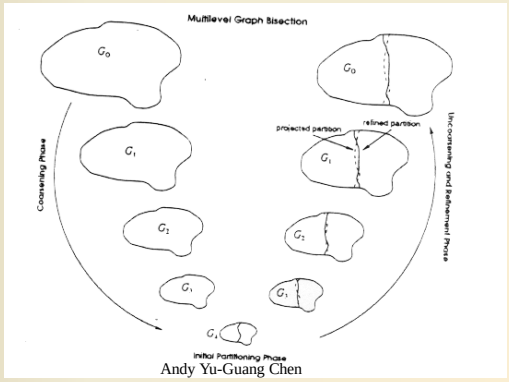


Multi-Level Partitioning

- G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Domain", DAC 1997

Multilevel Partitioning

- **Three phases** (for bipartitioning)
  - **Coarsening:** construct a sequence of smaller (coarser) graphs.
  - **Initial partitioning:** construct a bipartitioning solution for the coarsest graph.
  - **Uncoarsening & refinement:** the bipartitioning solution is successively projected to the next-level finer graph, and at each level an iterative refinement algorithm (such as KL or FM) is used to further improve the solution.

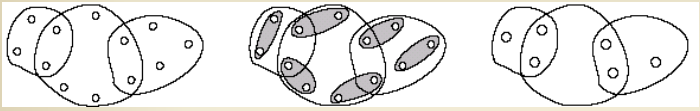


hMETIS

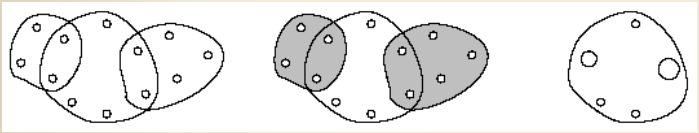
- Three coarsening algorithms:
  - **Edge coarsening:** A maximal matching of the vertices.
  - **Hyperedge coarsening**: a set of hyperedges is selected, and the vertices belonging to a selected hyperedge are merged into a cluster. (Preference: hyperedges with large weights and hyperedges of small size.)
  - **Modified hyperedge coarsening:** hyperedge coarsening + merging the remaining vertices of each hyperedge into a cluster.

Coarsening Phase

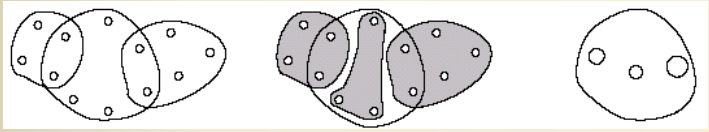
- Edge Coarsening



- Hyper-edge Coarsening (HEC)



- Modified Hyperedge Coarsening (MHEC)



Uncoarsening and Refinement Phase

- FM:
  - Based on FM with two simplifications:
    - Limit number of passes to 2
    - Early-Exit FM (FM-EE), stop each pass if k vertex moves do not improve the cut
- HER (Hyperedge Refinement)
  - Move a group of vertices between partitions so that an entire hyperedge is removed from the cut

hMETIS Algorithm

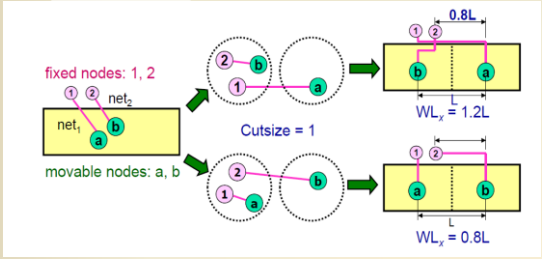
- Software implementation available for free download from Web
- hMETIS-EE_{20}
  - 20 random initial partitions
  - with 10 runs using HEC for coarsening
  - with 10 runs using MHEC for coarsening
  - FM-EE for refinement
- hMETIS-FM_{20}
  - 20 random initial partitions
  - with 10 runs using HEC for coarsening
  - with 10 runs using MHEC for coarsening
  - FM for refinement

Experimental Results

- Compared with five previous algorithms
- hMETIS-EE_{20} is:
  - 4.1% to 21.4% better
  - On average 0.5% better than the best of the 5 algorithms
  - Roughly 1 to 15 times faster
- hMETIS-FM_{20} is:
  - On average 1.1% better than hMETIS-EE_{20}
  - Improve the best-known bisections for 9 out of 23 test circuits
  - Twice as slow as hMETIS-EE_{20}

Partitioning for Wirelength Minimization

- Chen, Chang, Lin, "IMF: Interconnection-driven floorplanning for large-scale building-module designs," ICCAD-05
- Minimizing cut size is *not* equivalent to minimizing wirelength (WL)



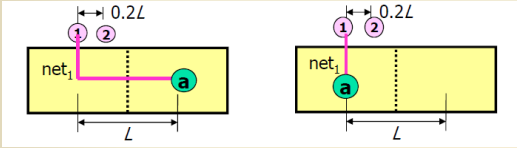
- Problem: **hyperedge weight is a constant value!**
  - Shall map the min-cut cost to wirelength (WL) change
  - Shall assign the hyperedge weight as the value of wirelength contribution if the hyperedge is cut

Net Weight Assignment

- net_1 connects a movable node *a* and a fixed node 1.

  Weight(net_1) = WL(net_1 is cut) – WL(net_1 is not cut)

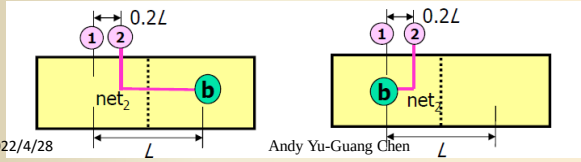
  = *L* – 0*L* = *L*



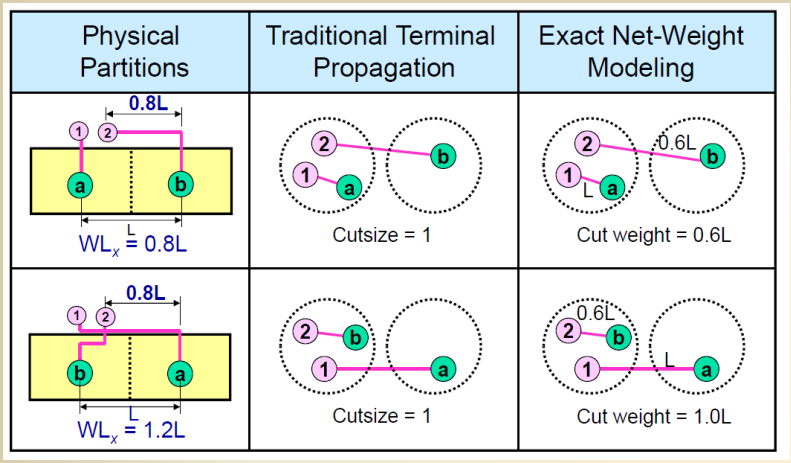
- net_2 connects a movable node *b* and a fixed node 2.

  Weight(net_2) = WL(net_2 is cut) – WL(net_2 is not cut)

  = *0.8L – 0.2L = 0.6L*



Examples



Cut weight is proportitional to the wirelength (WL)

WL = Cut weight + 0.2L

(0.2L is the WL lower bound: placing a & b in the left side)

Relationship Between WL and Cut Weight

- Theorem: $WL_i = w_{1,i} + n_{cut,i}$
  - $n_{cut,i}$: cut weight for net i
  - $w_{1,j}$: the wirelength lower bound for net i
- Then, we have

Constant

$$min(\sum WL_i) = min(\sum(w_{1,i} + n_{cut,i})) = \sum w_{1,i} + min(\sum n_{cut,i})$$

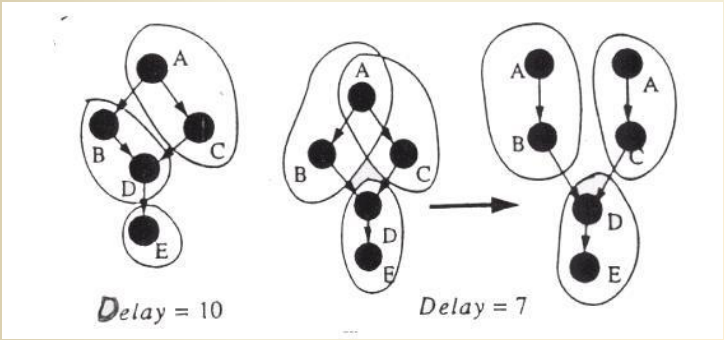
Finding the minimum wirelength is equivalent to finding the cut weight

Clustering for Delay Minimization

- Allow gate duplication.
- Gate duplication may help reduce delay.

D=3; M=2; $\delta(v)$=1, $w(v)$=1, for each v.

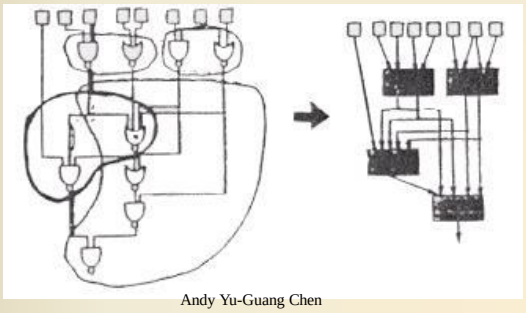
Without gate duplication | With gate duplication



Unit Delay Model

- No gate delay.
- No interconnection delay within a cluster.
- Delay of 1 unit for an interconnection between 2 clusters.
- An optimal algorithm for area constraint only (Lawler, Levitt and Turner, IEEE TC, 1966).
- An optimal algorithm for pin constraint only (Cong and Ding, ICCAD, 1992).

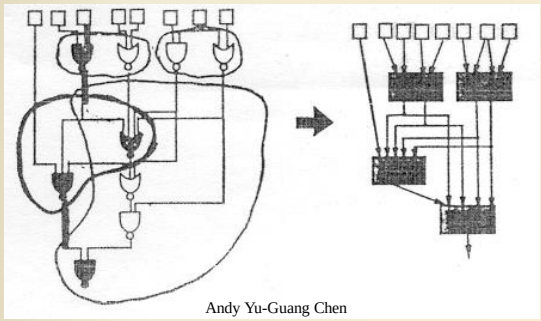
Circuit delay = 3.



General Delay Model

- Each gate has a delay.
- No interconnection delay within a cluster.
- Delay of D units for an interconnection between 2 clusters.
- A heuristic algorithm for area constraint only (Murgai, Brayton and Sangivanni-Vincentelli, ICCAD, 1991).

$D = 2$, $\delta(v) = 1$, circuit delay = 6+4 = 10.



General Delay Model (Cont'd)

- Rajaraman and Wong, "Optimal clustering for delay minimization," DAC, 1993.
- Optimal algorithm: $O(n^2 \log n + nm)$, where n is # of gates, m is # of interconnections.
- Definitions:
  - M: the area constraint on a cluster.
  - $W(C)$: the total area of the gates in cluster C.
  - N: a given combinational circuit.
  - N_v: v and all its *predecessors* in N.
  - $\delta(v)$: the delay of v.
  - $\Delta(u,v)$: maximum delay along any path from the output of u to the output of v, ignoring delays on interconnections.
  - $w(v)$: the area of v.
  - $l(v)$: the delay at v in an optimal clustering of N_v.
    For each *primary input* v, $l(v)=\delta(v)$.
  - $l'(u)=l(u)+\Delta(u,v)$, for each u in $N_v-\{v\}$.

General Delay Model (Cont'd)

- Algorithm: labeling phase + clustering phase.
- **Labeling phase:** compute $l(v)$ for each v in a topological order.
  - P: the set of nodes in N_v-{v} sorted in non-increasing order in the value of l'.

```
Algorithm Labeling(v);
begin
done ← false;
cluster(v) ← {v};
while (not done)
        Remove the first node u in P;
        if (W(cluster(v)) + w(u)) ≤ M)
                cluster(v) ← cluster(v) ∪ {u};
                if P is empty
                        done ← true;
                endif
        else
                done ← true;
        endif
endwhile
l1(v) ← max{l'(x) | x ∈ cluster(v) ∩ PI};
l2(v) ← ~~l(u) + D~~; max{l'(u)+D | u ∈ Nv - cluster(v)}
l(v) ← max{l1(v), l2(v)};
end
```

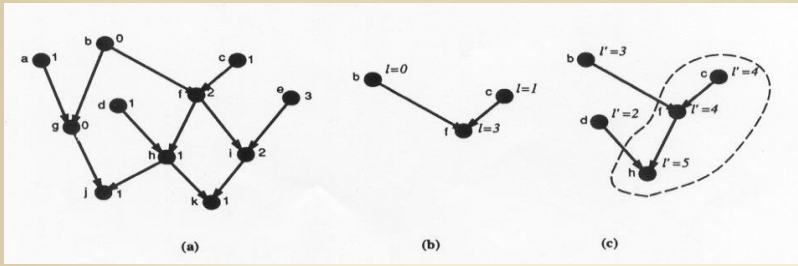

General Delay Model (Cont'd)

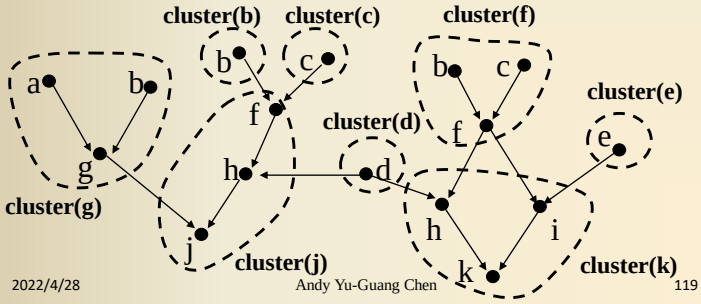
- **Clustering phase:** generate the clusters based on the information obtained in the labeling phase.
- Overall algorithm:

```
begin
Compute the maximum delay matrix Δ. Δ(i, j) is the
maximum delay along any path from the output of i
to the output of j;
for each PI i, do l(i) ← δ(i);
Sort the non-PI nodes of N in topological order
to obtain list T;
while T is non-empty
        Remove the first node v from T;
        Compute Nv;
        for each node u ∈ Nv\{v} do
                l'(u) ← l(u) + Δ(u, v);
        Sort the nodes in Nv\{v} in order of
        decreasing value of l' to form list P;
        Call Labeling(v);
endwhile
L ← PO;
S ← φ;
while L is not empty
        Remove a node v from L;
        S ← S ∪ {cluster(v)};   N - cluster(v)
        for all nodes x in N, such that x is adjacent
        to y, for some y ∈ cluster(v), L ← L ∪ {x};
endwhile
end
```

General Delay Model

- An example: M=3, D=3





Summary

- Partition an overview
- Kernighagn & Lin heuristic (KL)
- Fiduccia-Mattheyses heuristic (FM)
- Simulated annealing based method (SA)
- Multilevel circuit partitioning
- Partitioning for wirelength minimization
- Clustering for delay minimization