

Topic :  
**Circuit / System Partitioning in VLSI**  
**– NP Complete**

Heuristics for NP-complete layout problems draw heavily on foundations for more basic problems such as graph partitioning and graph coloring.

E.g.: Layout of VLSI circuits

CAD of digital systems / VLSI depends on theory and results of theoretical community

Automated Layout uses fundamental results in graph algorithm.

**Layout** : Partitioning, Placement, Routing & Floor Planning

Unfortunately, for most of these problems the Computational Complexity is NP-Hard!

As for all NP-Complete problems here also we look for *constraint satisfying solutions*.

- Solution quality is measured by how closely we meet the constraints.

## **Problem: System / Circuit Partitioning**

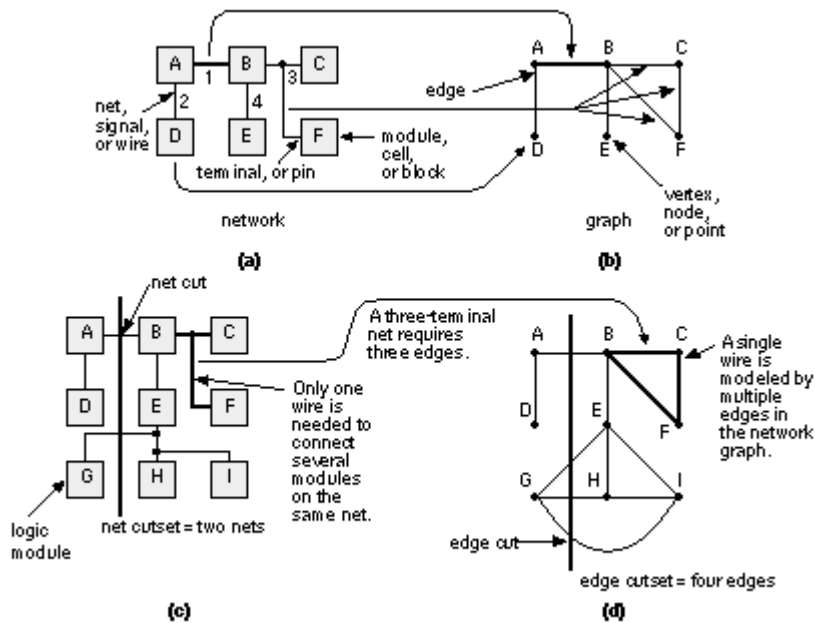
**Problem Definition:** Microelectronic systems typically consist of many functional blocks. If a functional block is too large to fit in one ASIC, we may have to split, or partition, the function into pieces using goals and objectives that we need to specify. For example, we might want to minimize the number of pins for each ASIC to minimize package cost.

### **Partition objectives:**

Set the goal relevant to any or all of the following:

- size for each ASIC
- number of ASICs
- number of connections for each ASIC
- number of total connections between all ASICs

Example :



a. A network containing circuit logic cells and nets.

b. The equivalent graph with vertexes and edges. For example: logic cell D maps to node D in the graph; net 1 maps to the edge (A, B) in the graph. Net 3 (with three connections) maps to three edges in the graph: (B, C), (B, F), and (C, F).

c. Partitioning a network and its graph. A network with a net cut that cuts two nets.

d. The network graph showing the corresponding edge cut. The net cutset in c contains two nets, but the corresponding edge cutset in d contains four edges. This means a graph is not an exact model of a network for partitioning purposes.

**NP Complete Problem** : Partitioning with Minimum connections between 2 ASICs \*

\*  $\equiv$  MIN-CUT problem

**No Efficient heuristics exist for n-way partitioning is found to be useful.**

SPECIAL CASE : BISECTION. So bisection algorithms are used recursively to achieve results closer to the objective.

### **Proof of NP-Completeness**

- MIN-CUT  $\in$  NP

The decision problem for minimum bisection, which asks for a given graph whether it has a bisection of less than a given width, is obviously in NP since given a bisection we can verify quickly its width and the fact that it is a bisection.

- MIN-CUT  $\in$  NP- Complete

$$3SAT \leq_p \text{MAX-CUT} \leq_p \text{MAX-BIS} \leq_p \text{MIN-BIS}$$

We show that maximum cut can be reduced to minimum bisection, thereby showing that minimum bisection is NP-complete.

First note that maximum bisection can easily be reduced to minimum bisection (or vice-versa) because the maximum bisection of a graph is the minimum bisection of its

complement. Note that this duality only works because the number of possible edges across a bisection is constant for a given graph; it fails for max cut and min cut, which is why it makes sense for one to be NP-complete and the other not.

Now we show how to reduce max cut to max bisection. Given a graph  $G$  with  $n$  vertices, we claim that the width of the maximum cut for  $G$  is equal to that of the maximum bisection of the graph  $G'$  given by appending  $n$  isolated vertices to  $G$ . This is because any cut of  $G$  can be made into a bisection of  $G'$  by adding a suitable number of isolated vertices to each side of the cut. This gives an easy reduction of the decision problem for max cut to that for max bisection-the reduction is polynomial time since we only doubled the size of the input.

Some algorithms are available for 2-way partitioning or Bisection are as follows :

## **Greedy Search Algorithms**

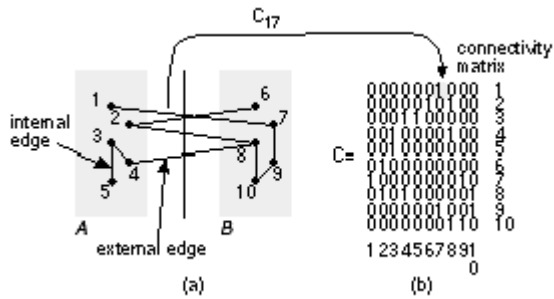
Simple Greedy Method  
Kernighan-Lin Algorithm

## **Spectral Methods**

Simple Spectral Bisection  
Boppana Bisection

## **Randomized Algorithms**

Metropolis and Simulated Annealing  
Genetic Algorithm



$$W = \sum C_{ab}$$

$a \in A$  and  $b \in B$

the cut weight for the given graph is 4 (all the edges have weights of 1).

$$E_a = \sum C_{ay}$$

$y \in B$

for the given graph  $E_1 = 1$ , and  $E_3 = 0$ .

$$I_a = \sum C_{az}$$

$z \in B$

For the given graph  $I_1 = 0$ , and  $I_3 = 2$ .

$$D_X = E_X - I_X$$

Thus, in Figure  $D_1 = 1$ ,  $D_3 = -2$ , and  $D_8 = 1$ .

Now pick any node in A, and any node in B. If we swap these nodes, a and b, we need to measure the reduction in cut weight, which we call the gain, g. We can express g in terms of the edge costs as follows:

$$g = D_a + D_b - 2 C_{ab}$$

The last term accounts for the fact that a and b may be connected. So, in Figure, if we swap nodes 1 and 6, then  $g = D_1 + D_6 - 2 C_{16} = 1 + 1$ . If we swap nodes 2 and 8, then  $g = D_2 + D_8 - 2 C_{28} = 1 + 2 - 2$ .

So, in Figure, if we swap nodes 1 and 6, then  $g = D_1 + D_6 - 2 C_{16} = 1 + 1$ . If we swap nodes 2 and 8, then  $g = D_2 + D_8 - 2 C_{28} = 1 + 2 - 2$ .

## K-L Heuristic

Compute  $T = \text{cost}(A, B)$  for initial  $A, B \dots \text{cost} = O(|N|^2)$

### **Repeat**

Compute costs  $D(n)$  for all  $n$  in  $N \dots O(|N|^2)$

Unmark all nodes in  $N \dots O(|N|)$

**While** there are unmarked nodes  $\dots |N|/2$  iterations

Find an unmarked pair  $(a, b)$  maximizing  $\text{gain}(a, b) \dots O(|N|^2)$

Mark  $a$  and  $b$  (but do not swap them)  $\dots O(1)$

Update  $D(n)$  for all unmarked  $n$ , as though  $a$  and  $b$  had been swapped  
 $\dots O(|N|)$

### **Endwhile**

$\dots$  At this point we have computed a sequence of pairs

$\dots (a_1, b_1), \dots, (a_k, b_k)$  and gains  $\text{gain}(1), \dots, \text{gain}(k)$

$\dots$  for  $k = |N|/2$ , ordered by the order in which we marked them

Pick  $j$  maximizing  $\text{Gain} = \sum_{k=1}^j \text{gain}(k) \dots O(|N|)$

$\dots$  Gain is reduction in cost from swapping  $(a_1, b_1)$  through  $(a_j, b_j)$

**If**  $\text{Gain} > 0$  then  $\dots$  it is worth swapping

Update  $\text{newA} = A - \{a_1, \dots, a_k\} \cup \{b_1, \dots, b_k\} \dots O(|N|)$

Update  $\text{newB} = B - \{b_1, \dots, b_k\} \cup \{a_1, \dots, a_k\} \dots O(|N|)$

Update  $T = T - \text{Gain} \dots O(1)$

### **Endif**

**Until**  $\text{Gain} \leq 0$

### Disadvantages

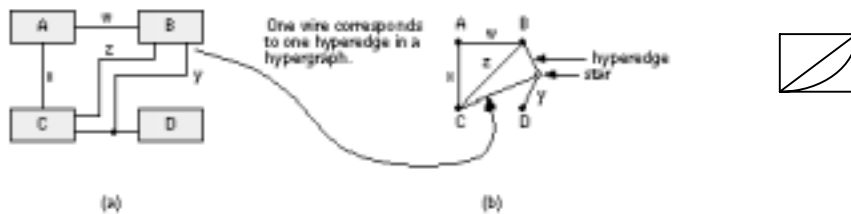
High Time complexity  $O(n^3)$ ....large for VLSI (  $n \cong 10^8$  )

Achieves Local Optimum only

Balanced Partitions

No Weight for vertices

Only on Edges, not on hyper edges



### Worst Cases

Multiple disconnected complete graphs

Expanded Hypercubes

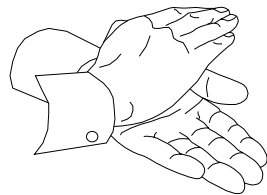
### Better Alternative

*Fiduccia Mattheyses Heuristic*...non balanced partition, takes care of weighted vertices, hyper edges, can be extended for global optimum

...ABOVE all time complexity is reduced to  $O(n^2)$



Questions ?



**END OF PRESENTATION**