

## CIRCUIT PARTITIONING

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

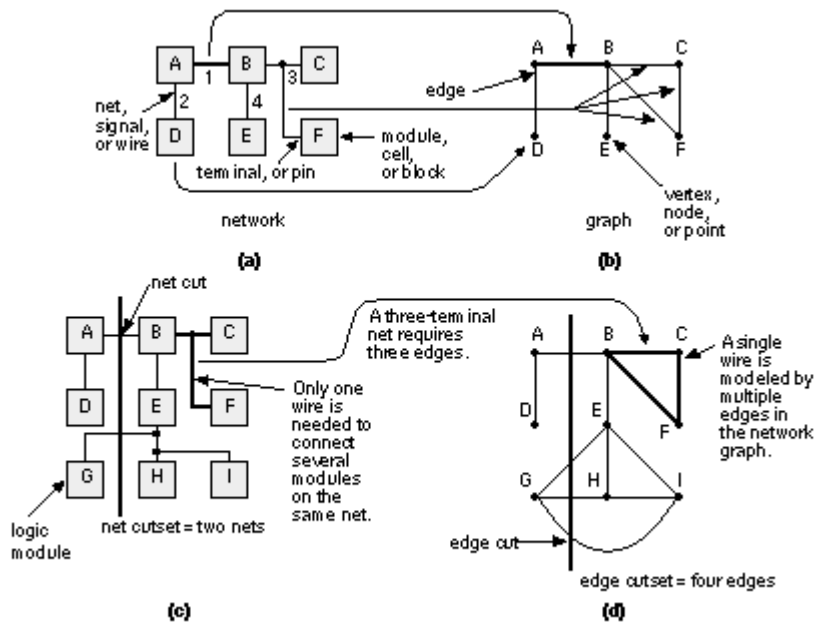
**Problem Definition:** Microelectronic systems typically consist of many functional blocks. If a functional block is too large to fit in one ASIC, we may have to split, or partition, the function into pieces using goals and objectives that we need to specify. For example, we might want to minimize the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



- A network containing circuit logic cells and nets.
- The equivalent graph with vertices 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).
- Partitioning a network and its graph. A network with a net cut that cuts two nets.
- 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. \*

\* illustrates the differences between the nets of a network and the edges in the network graphs.

### Choosing optimal partitioning is NP- complete

- Only known exact algorithms have cost = exponential( n)
- We need good heuristics

**NP Complete Problem** : Partitioning with Minimum connections between 2 ASICs \*  
\*  $\equiv$  MIN-CUT problem

A bisection of a graph  $G=(V, E)$  with an even number of vertices is a pair of disjoint subsets  $A, B \subset V$  of equal size. The cost of bisection is the number of edges  $C = \sum_{(a,b) \in E} \mathbb{1}_{a \in A, b \in B}$ . The problem of Graph Bisection takes as input a graph  $G$  with an even number of vertices, and returns a bisection of minimum cost.

We show that the minimum bisection problem and the problem of approximating it to within a constant factor are NP-complete

### Proof of NP-Completeness

- $\text{MIN-CUT} \in \text{NP}$

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.

- $\text{MIN-CUT} \in \text{NP-Complete}$

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

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.

**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. 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

### **K-L Heuristic**

Iterative improvement based method based on local changes. Take an initial partition and iteratively improve it. • One pass greedily computes  $|N|/2$  possible X and Y to swap, picks best

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

#### **Repeat**

Compute costs  $D(n)$  for all  $n$  in  $N$  ...  $\text{cost} = O(|N|^2)$

Unmark all nodes in  $N$  ...  $\text{cost} = O(|N|)$

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

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

Mark a and b (but do not swap them) ...  $\text{cost} = O(1)$

Update  $D(n)$  for all unmarked  $n$ , as though a and b had been swapped  
...  $\text{cost} = O(|N|)$

#### **Endwhile**

... At this point we have computed a sequence of pairs

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

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

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

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

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

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

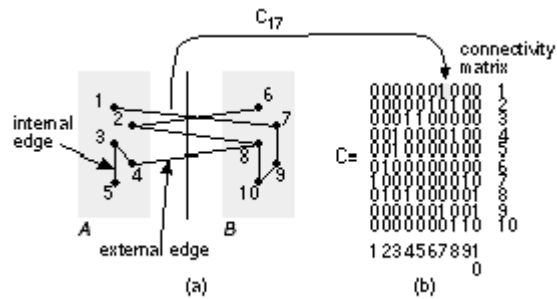
Update  $\text{newB} = B - \{b_1, \dots, b_k\} \cup \{a_1, \dots, a_k\}$  ...  $\text{cost} = O(|N|)$

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

#### **Endif**

Until Gain  $\leq 0$

Example :



(a) An example network graph.

(b) The connectivity matrix, C; the column and rows are labeled to help you see how the matrix entries correspond to the node numbers in the graph.

For example, C 17 (column 1, row 7) equals 1 because nodes 1 and 7 are connected. In this example all edges have an equal weight of 1, but in general the edges may have different weights.

Suppose we already have split a network into two partitions, A and B , each with m nodes (perhaps using a constructed partitioning). Our goal now is to swap nodes between A and B with the objective of minimizing the number of external edges connecting the two partitions. Each external edge may be weighted by a cost, and our objective corresponds to minimizing a cost function that we shall call the total external cost, cut cost, or cut weight, W :

$$W = \sum_{a \in A \text{ and } b \in B} C_{ab}$$

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

In order to simplify the measurement of the change in cut weight when we interchange nodes, we need some more definitions. First, for any node a in partition A , we define an external edge cost, which measures the connections from node a to B ,

$$E_a = \sum_{y \in B} C_{ay}$$

for the given graph  $E_1 = 1$ , and  $E_3 = 0$ . Second, we define the internal edge cost to measure the internal connections to a ,

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

$z \in B$

For the given graph  $I_1 = 0$ , and  $I_3 = 2$ . We define the edge costs for partition B in a similar way (so  $E_8 = 2$ , and  $I_8 = 1$ ). The cost difference is the difference between external edge costs and internal edge costs,

$$D_X = E_X - I_X .$$

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  $g = D_2 + D_8 - 2 C_{28} = 1 + 2 - 2$ .

The K-L algorithm finds a group of node pairs to swap that increases the gain even though swapping individual node pairs from that group might decrease the gain. First we pretend to swap all of the nodes a pair at a time.

This is the algorithm:

1. Find two nodes,  $a_i$  from A , and  $b_i$  from B , so that the gain from swapping them is a maximum. The gain is  $g_i = D_{a_i} + D_{b_i} - 2 C_{a_i b_i}$  .
2. Next pretend swap  $a_i$  and  $b_i$  even if the gain  $g_i$  is zero or negative, and do not consider  $a_i$  and  $b_i$  eligible for being swapped again.
3. Repeat steps 1 and 2 a total of m times until all the nodes of A and B have been pretend swapped. We are back where we started, but we have ordered pairs of nodes in A and B according to the gain from interchanging those pairs.
4. Now we can choose which nodes we shall actually swap. Suppose we only swap the first n pairs of nodes that we found in the preceding process. In other words we swap nodes  $X = a_1, a_2, \dots, a_n$  from A with nodes  $Y = b_1, b_2, \dots, b_n$  from B. The total gain would be

$$G_n = \sum g_i \text{ Where } i = 1 \text{ to } n$$

5. We now choose  $n$  corresponding to the maximum value of  $G_n$ .

If the maximum value of  $G_n > 0$ , then we swap the sets of nodes  $X$  and  $Y$  and thus reduce the cut weight by  $G_n$ . We use this new partitioning to start the process again at the first step. If the maximum value of  $G_n = 0$ , then we cannot improve the current partitioning and we stop. We have found a locally optimum solution

### **Worst Cases**

Multiple disconnected complete graphs

Expanded Hypercubes