EE677 Foundations of VLSI CAD
Implementation, Visualisation and Analysis of Various Circuit Partitioning Algorithms

Rohan Rajesh Kalbag, 20D170033
Anubhav Bhatla, 200070008

December 9, 2022

Abstract: Often the VLSI design schematic of a system cannot be emulated/verified on a single FPGA, due to the finite number of programmable logic elements present in the FPGA. Interconnections between circuit elements can be conveniently represented using graphs, Logic gates, LUTs, FFs, and other entities present in the design can be modelled as graph nodes, and the interconnections are modelled as edges, if there are multiple parallel interconnects between two entities, we can represent the same using weighted graphs. Suppose we wish to computerize dividing the design among 2 FPGAs using CAD; we can model the problem as a graph partitioning problem.

1 Acknowledgement

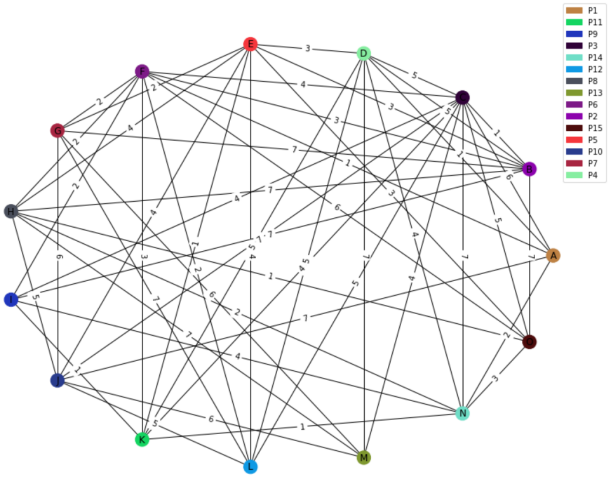
We want to thank **Prof. Virendra Singh** for giving us this opportunity to explore the extremely important field of VLSI Computer-Aided Design Automation through a project such as this.

2 Objectives of Project

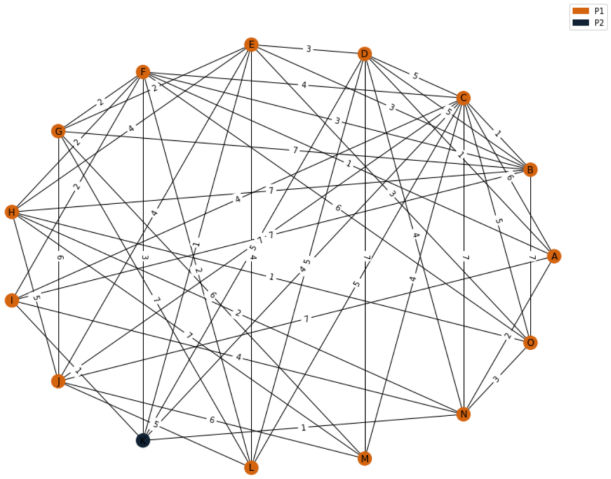
In this project, we implement graph partitioning algorithms and heuristics like the **Kernighan-Lin Algorithm**, **Clustering Based Heuristic**, **Hagen Kahng EIG Algorithm** and compare, analyse their capability in partitioning a given graph network into two partitions and visualise them using plotting tools of `matplotlib`, `networkx` libraries of `Python`.

3 Clustering-based Partitioning

In this heuristic partitioning method, we start with each node having its own unique partition. We look for the edge with the highest weight, and the two nodes connected to this edge are joined into a single partition, the second node is assigned the partition name of the first node of the collapsed edge. This is repeated till we are left with only two partitions. Consider the graph given below:



On applying the clustering algorithm, we get the following final graph:

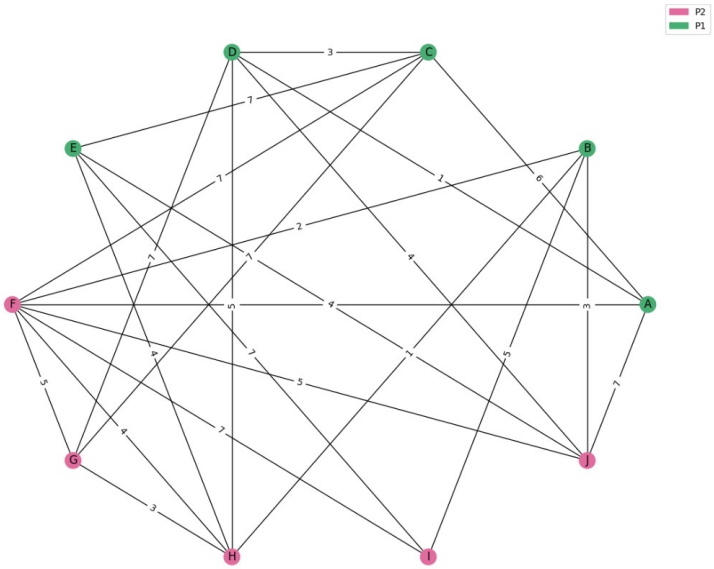


The final cost of this partition comes out to be 15 units. In general, the clustering-based partition method leads to **highly unbalanced** partitions which majorly consist of (n - 1, 1) splits such as the above example.

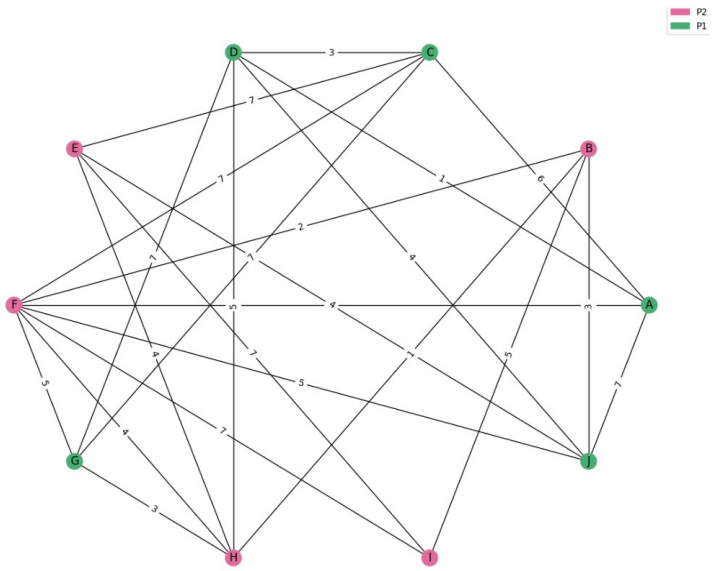
4 Kernighan-Lin (KL) Partitioning

4.1 Equal Partitioning

In this method of partitioning for a graph of $2n$ nodes, we take a random equal initial split of n nodes each and partition the graph into two partitions P_1 and P_2. We iterate over n, and for every node a, we calculate the internal and external edge costs for it denoted by E_a and E_b. We then calculate $D_a = E_a - I_a$ for the node. We calculate the **cost reduction metric** for swapping nodes a and b denoted by $g_{ab} = D_a + D_b - 2c_{ab}$ where c_{ab} is the weight corresponding to an edge between a and b. We swap the nodes with the max value of $g_{sel} = g_{ab}$, then we calculate the updated values of D_a for all nodes a and so on. We keep track of the prefix sum G of sum of all g_{sel} over all iterations. We define the maximal index m for which the prefix sum is maximized and only perform the swaps corresponding to g_{ab} for all iterations $i \leq m$ and ignore the rest. This gives a configuration with the minimized cut size but the same partition. Consider the graph given below, which has an initial cut size of **64**

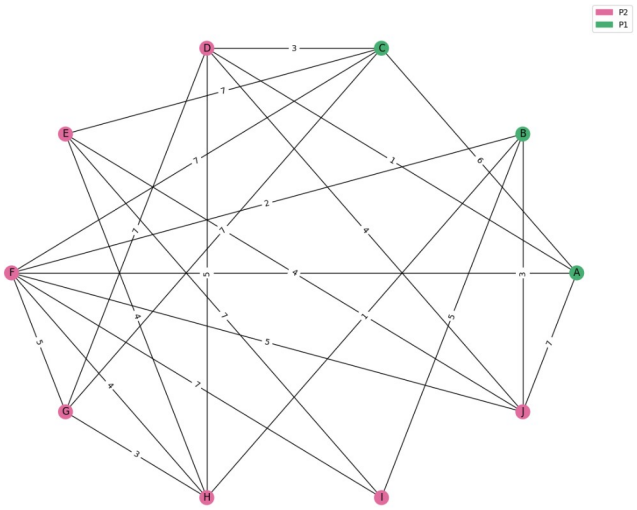


After applying the KL algorithm, we obtain the below graph which has a reduced corresponding cut size of **40**.

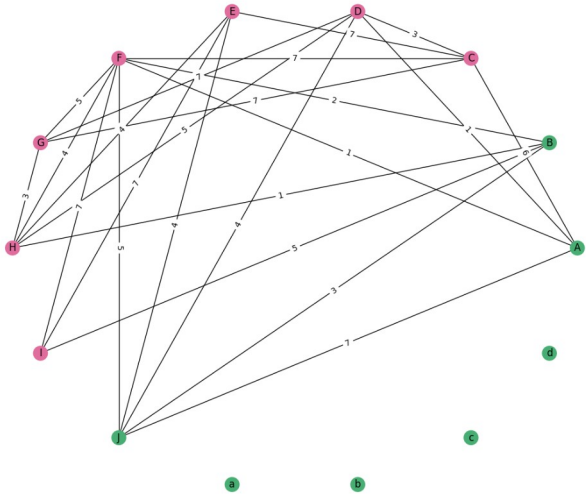


4.2 Unequal Partitioning

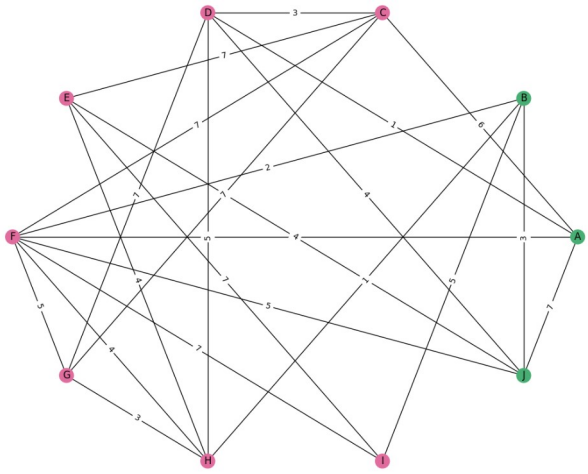
We partition the graph into n_1 and n_2 nodes each and then add $|n_1 - n_2|$ nodes to the smaller partition with no interconnections (edges). We then apply the KL algorithm for equal partitions as described above, perform the swaps and remove the dummy added edges after the swaps are completed to obtain the minimized cut size configuration. Consider the unequal partitioning of the below graph



The optimized partition after adding the dummy nodes and performing KL algorithm can be found below



The optimized partition after removing the dummy nodes is as shown below and works as intended and the cut size reduces.

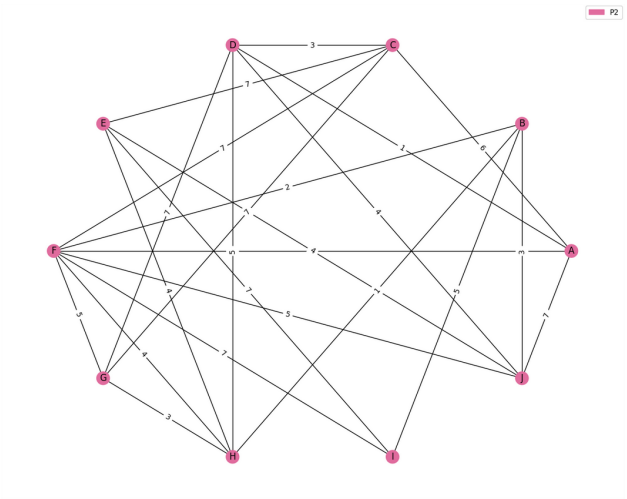


5 Hagen-Kahng EIG Partitioning

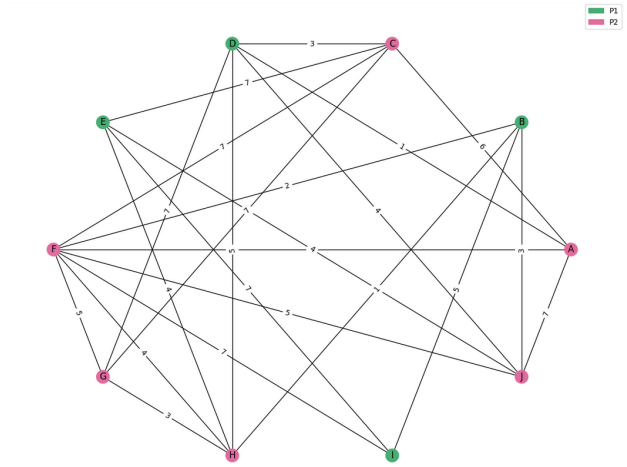
We start by defining the Degree Matrix wherein the entry $Deg[i][i]$ is equal to the sum of weights of the edges attached to the i^{th} node and the remaining entries being 0. Similarly, we also define the Adjacency Matrix as $Adj[i][j]$ equal to the weight of the edge between nodes i and j and the remaining entries being 0. We now define the Laplacian matrix as:

$$Lap = Deg - Adj \tag{1}$$

We now find the eigenvalues and eigenvectors for this Laplacian matrix and choose the **second smallest eigenvalue**. The eigenvector corresponding to this eigenvalue now consists of values $\in [-1, 1]$. This vector's i^{th} value corresponds to the i^{th} node. We aim to find the best partition between these values, dividing the nodes into two partitions. Consider the graph below:



On applying the EIG algorithm, we get the following final graph:



This partition is obtained by moving the nodes corresponding to the three smallest values in the above-mentioned eigenvector to partition P1. The final cost of this partition comes out to be 48 units.

6 Benchmarking

We generate connected **Erdos - Renyi** random graphs with a specified number of edges with randomly seeded edge weights varying from 1 to 7.

We measure metrics for 16 generated graphs, varying from 10 to 25 nodes with random weights. We decided not to compare the Clustering-based algorithm with the other two partitioning algorithms because the Clustering algorithm produced **highly unequal partitions** consisting majorly of (n-1, 1) splits, which may is not practical for real-life systems and beats the goal of partitioning. We continue with the other two algorithms, comparing them using the metrics and tabulate **cut size** and **ratio cut** for each configuration of the benchmark graph. The ratio cut is given by $Ratio\ Cut\ = \frac{Cut\,Size}{(s)(n-s)}$, where s and $n-s$ are the sizes of the two partitions.

Let's start by considering the results of the benchmark graph with 15 nodes. Given below are all the **cut size** and **ratio cut** for each configuration start of the benchmark graph for either of the algorithms:

Partition Sizes	KL		EIG	
P1-P2	Cutsize	Ratio Cut	Cutsize	Ratio Cut
1-14	11	0.786	30	2.143
2-13	29	1.115	59	2.269
3-12	40	1.111	66	1.833
4-11	50	1.136	72	1.636
5-10	63	1.260	94	1.880
6-9	63	1.167	114	2.111
7-8	66	1.179	107	1.911
8-7	66	1.179	116	2.071
9-6	60	1.111	107	1.981
10-5	47	0.940	94	1.880
11-4	42	0.955	91	2.068
12-3	38	1.056	65	1.806
13-2	28	1.077	40	1.538
14-1	11	0.786	18	1.286

In the above table, we observe that KL gives us better overall cut sizes and ratio cuts than EIG and thus leads to more cost-efficient partitioning. However, KL takes a total of $579\,ms$ to run the entire benchmark compared to $102\,ms$ for the EIG algorithm. Therefore, we observe a trade-off between cost and time for the two algorithms, with KL being more cost-efficient and EIG being more time-efficient.