GRAPH PARTITIONING

1905064 - SAYEM SHAHAD
1905065 - ARNAB BHATTACHARJEE
1905079 - SALMAN SAYEED
1905089 - MAJISHA JAHAN DISHA
1905109 - ARKO SIKDER

AGENDA

DISCUSSION POINTS

01

INTRO AND PROBLEM OVERVIEW

1905079

02

GKLR (GLOBAL KL REFINEMENT)

1905064

03

EXPERIMENTATION AND RESULTS

190565

04

METAHEURISTICS ALGORITHMS

1905089

05

RESULTS AND EXPERIMENTATION

1905109

06

FINAL COMPARISON

1905109

AGENDA 1

INTRO AND PROBLEM OVERVIEW

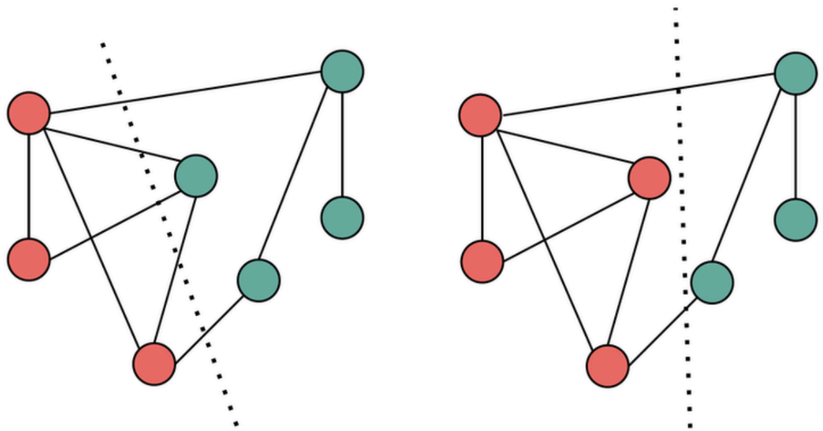
1905079

INTRODUCTION

GRAPH PARTITIONING

GRAPH PARTITION

Reduction of a **graph to a smaller graph** by partitioning its set of nodes into mutually exclusive groups.

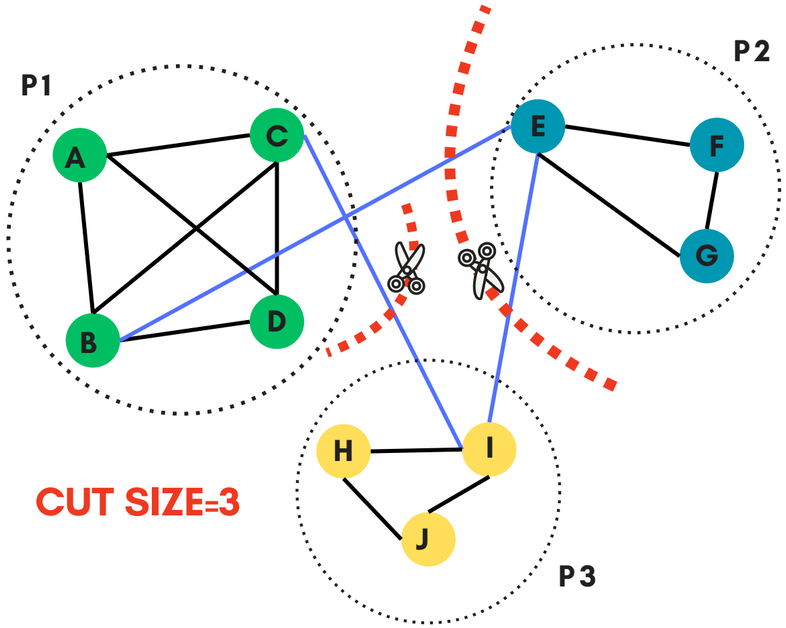


Bad partition

Good partition

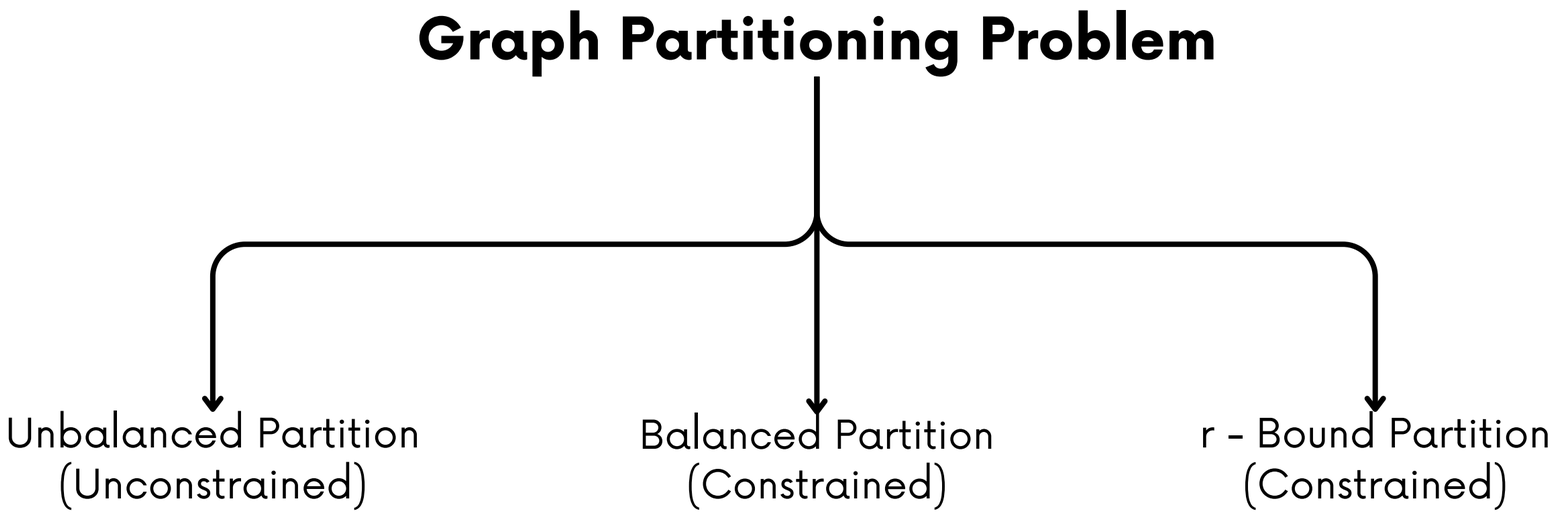
GRAPH PARTITIONING PROBLEM

Tries to **minimize cut size** between different partitions



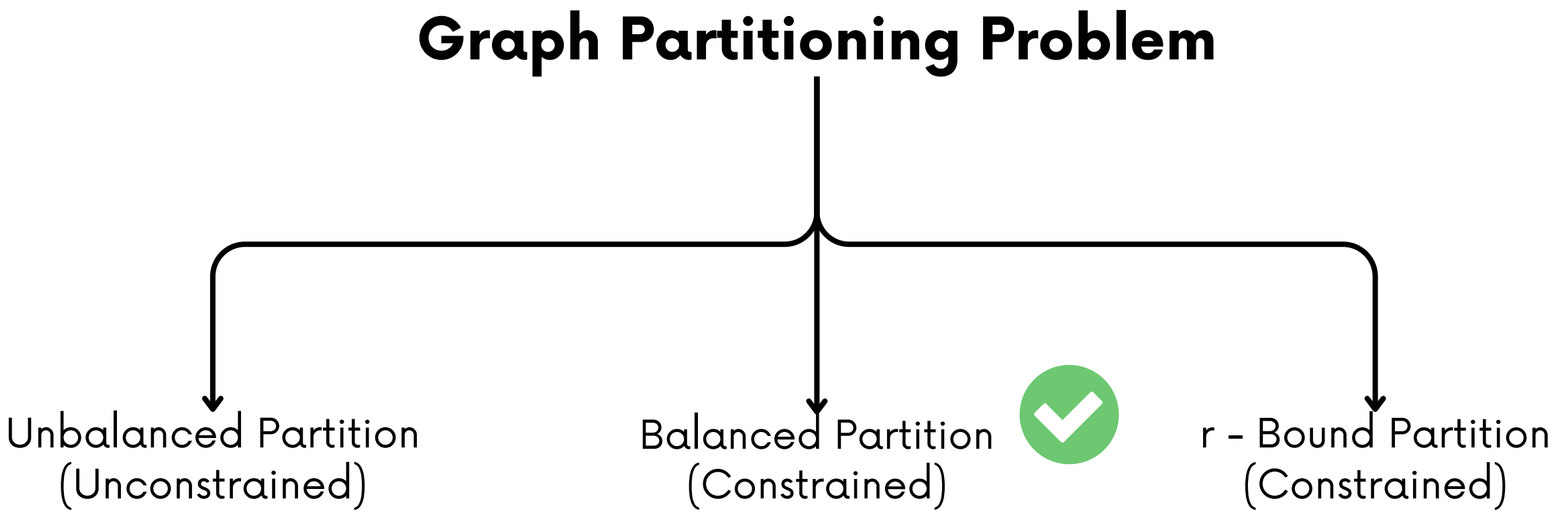
INTRODUCTION

PARTITION TYPE



INTRODUCTION

PARTITION TYPE



INTRODUCTION

GRAPH PARTITIONING

PROBLEM DEFINITION

Instance:

- An undirected graph $G = (V, E)$
- Number of desired partitions $k \in \mathbb{N}$

Optimization:

Find a **balanced** partition of V into k subsets $V_1, V_2, \ldots, V_k$ that **minimizes** **cut**$(V_1, \ldots, V_k)$ = number of edges e crossing between partitions subject to:

- $\forall i \in \{1, \ldots, k\}: |V_i| \leqslant 1 + \lceil |V|/k \rceil$ (balance constraint)
- $\bigcup_i V_i = V$ (all vertices must be assigned)
- $V_i \cap V_j = \varnothing$ for $i \neq j$ (partitions are disjoint)

Complexity:

The Graph Partitioning Problem is **NP-Hard for $k \geqslant 2$**, for both weighted and unweighted graphs.

INTRODUCTION

COMMON USE-CASES

CKT PARTITIONING

SOCIAL NETWORKS

LOAD BALANCING

TRANSPORTATION NETWORK

IMAGE PROCESSING

DATABASE MANAGEMENT

INTRODUCTION

ALGORITHMS WE DISCUSSED

SPECTRAL PARTITIONING

INTRODUCTION

ALGORITHMS WE DISCUSSED

SPECTRAL PARTITIONING

- Constructs Laplacian Matrix
- Computes Eigen value and eigen vector
- For each vertex if eigen vector>0 put it in one partition other in the other

INTRODUCTION

ALGORITHMS WE DISCUSSED

SPECTRAL PARTITIONING

- Constructs Laplacian Matrix
- Computes Eigen value and eigen vector
- For each vertex if eigen vector>0 put it in one partition other in the other

MULTILEVEL GRAPH PARTITIONING

INTRODUCTION

ALGORITHMS WE DISCUSSED

SPECTRAL PARTITIONING

- Constructs Laplacian Matrix
- Computes Eigen value and eigen vector
- For each vertex if eigen vector>0 put it in one partition other in the other

MULTILEVEL GRAPH PARTITIONING

A 3 stage algorithm including:

- Coarsening Phase (reduces size of the graph)
- Initial Partitioning (on smaller graph)
- Uncoarsening and Refinement (uses KL or FM)

INTRODUCTION

ALGORITHMS WE DISCUSSED

SPECTRAL PARTITIONING

MULTILEVEL GRAPH PARTITIONING

SIMULATED ANNEALING

INTRODUCTION

ALGORITHMS WE DISCUSSED

SPECTRAL PARTITIONING

MULTILEVEL GRAPH PARTITIONING

SIMULATED ANNEALING

- Initial random partitioning
- Exploring neighbors and probabilistic shift
- Lowering probability of lower ones by lowering temperature and moving to convergence

INTRODUCTION

ALGORITHMS WE DISCUSSED

SPECTRAL PARTITIONING

MULTILEVEL GRAPH PARTITIONING

SIMULATED ANNEALING

KERNINGHAN-LIN (KL)

- Initial random partitioning
- Exploring neighbors and probabilistic shift
- Lowering probability of lower ones by lowering temperature and moving to convergence

INTRODUCTION

ALGORITHMS WE DISCUSSED

SPECTRAL PARTITIONING

MULTILEVEL GRAPH PARTITIONING

SIMULATED ANNEALING

- Initial random partitioning
- Exploring neighbors and probabilistic shift
- Lowering probability of lower ones by lowering temperature and moving to convergence

KERNINGHAN-LIN (KL)

- Initial partitioning of equal size
- Gain calculation (Reduction in cut size)
- Vertex Pair Swapping betn pairs with max gain
- Lock swapped vertices and continue till convergence

INTRODUCTION

ALGORITHMS WE DISCUSSED

SPECTRAL PARTITIONING

MULTILEVEL GRAPH PARTITIONING

SIMULATED ANNEALING

KERNINGHAN-LIN (KL)

FIDUCCIA-MATTHEYSES (FM)

INTRODUCTION

ALGORITHMS WE DISCUSSED

SPECTRAL PARTITIONING

MULTILEVEL GRAPH PARTITIONING

SIMULATED ANNEALING

KERNINGHAN-LIN (KL)

FIDUCCIA-MATTHEYSES (FM)

- Random initial partition
- Gain Calculation and moving of a single vertex with the highest gain (instead of a pair like KL) and lock the vertex
- Maintain gains in “Buckets”
- Identify the point with the maximum cumulative gain and revert moves made after this point, and repeat until convergence

INTRODUCTION

ALGORITHMS WE DISCUSSED

- SPECTRAL PARTITIONING
- MULTILEVEL GRAPH PARTITIONING
- SIMULATED ANNEALING
- KERNINGHAN-LIN (KL)
- FIDUCCIA-MATTHEYSES (FM)
- GLOBAL KL REFINEMENT (GKLR)

INTRODUCTION

ALGORITHMS WE DISCUSSED

SPECTRAL PARTITIONING

MULTILEVEL GRAPH PARTITIONING

SIMULATED ANNEALING

KERNINGHAN-LIN (KL)

FIDUCCIA-MATTHEYSES (FM)

GLOBAL KL REFINEMENT (GKLR)

GENETIC ALGORITHM

INTRODUCTION

ALGORITHMS WE DISCUSSED

SPECTRAL PARTITIONING

MULTILEVEL GRAPH PARTITIONING

SIMULATED ANNEALING

KERNINGHAN-LIN (KL)

FIDUCCIA-MATTHEYSES (FM)

GLOBAL KL REFINEMENT (GKLR)

GENETIC ALGORITHM

INTRODUCTION

ALGORITHMS WE DISCUSSED

SPECTRAL PARTITIONING

MULTILEVEL GRAPH PARTITIONING

SIMULATED ANNEALING

KERNINGHAN-LIN (KL)

FIDUCCIA-MATTHEYSES (FM)

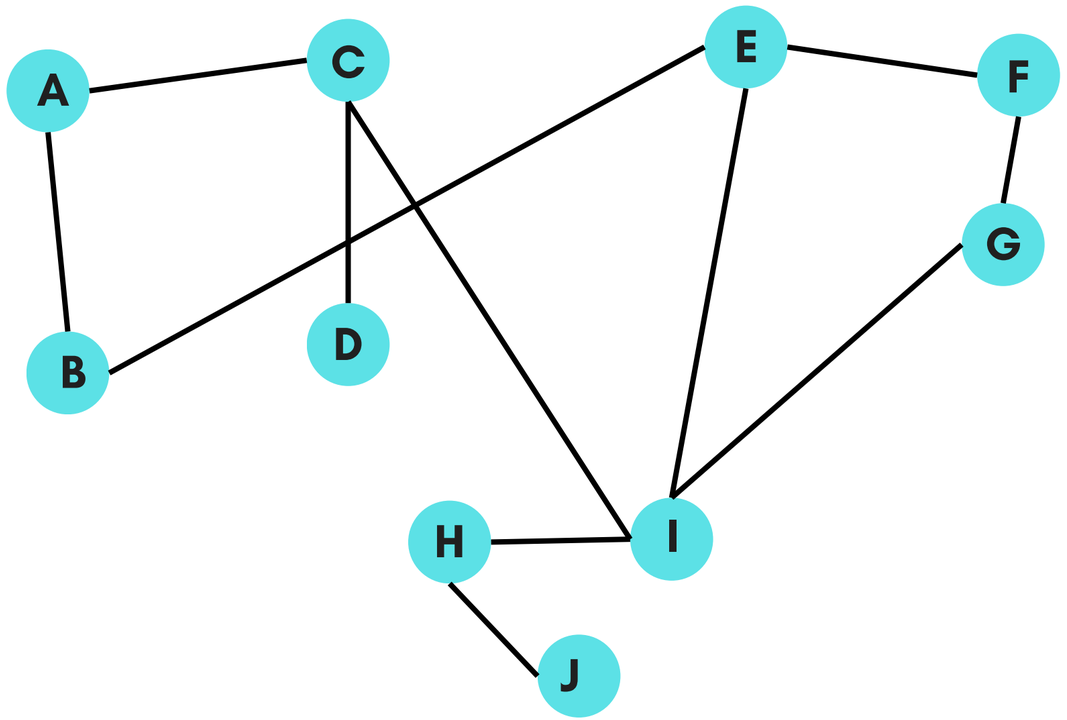
GLOBAL KL REFINEMENT (GKLR) ✓

GENETIC ALGORITHM ✓

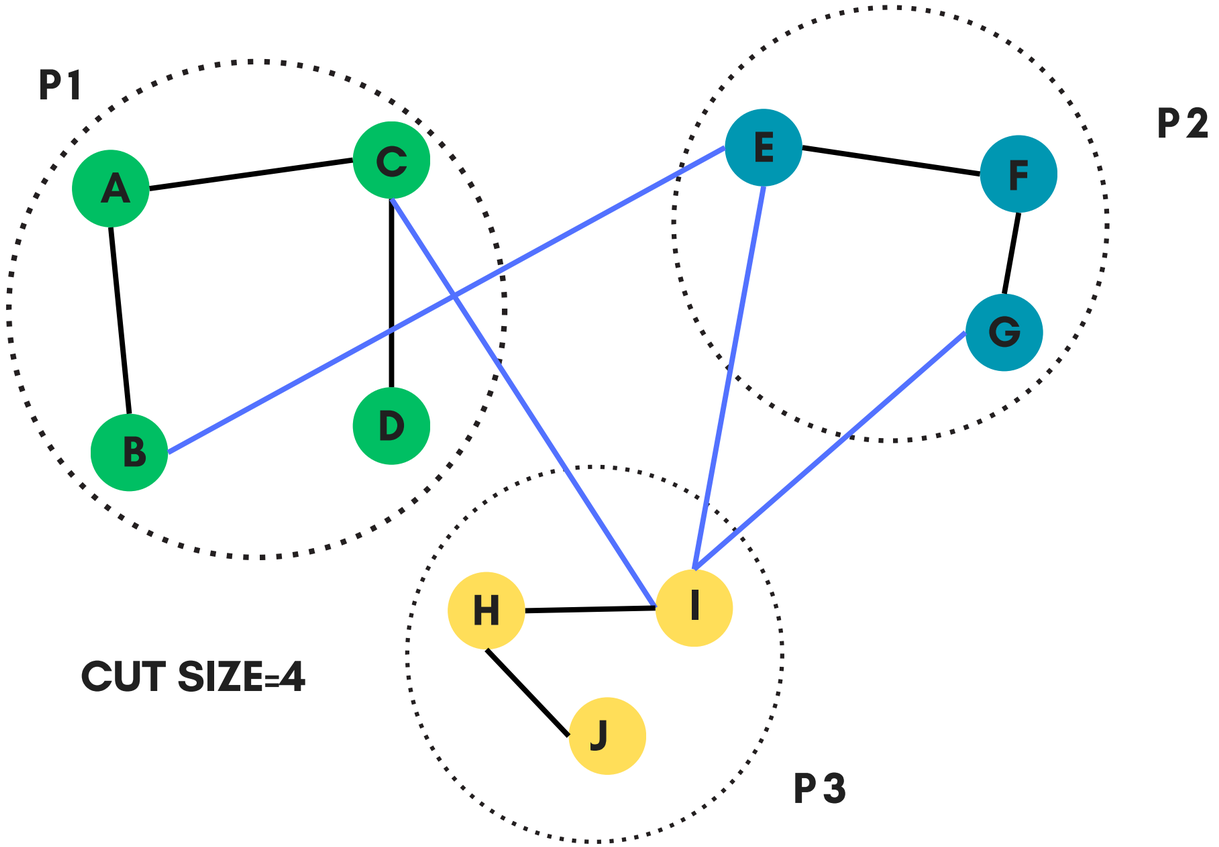
AGENDA 2

GKLR (GLOBAL KL REFINEMENT)

1905064

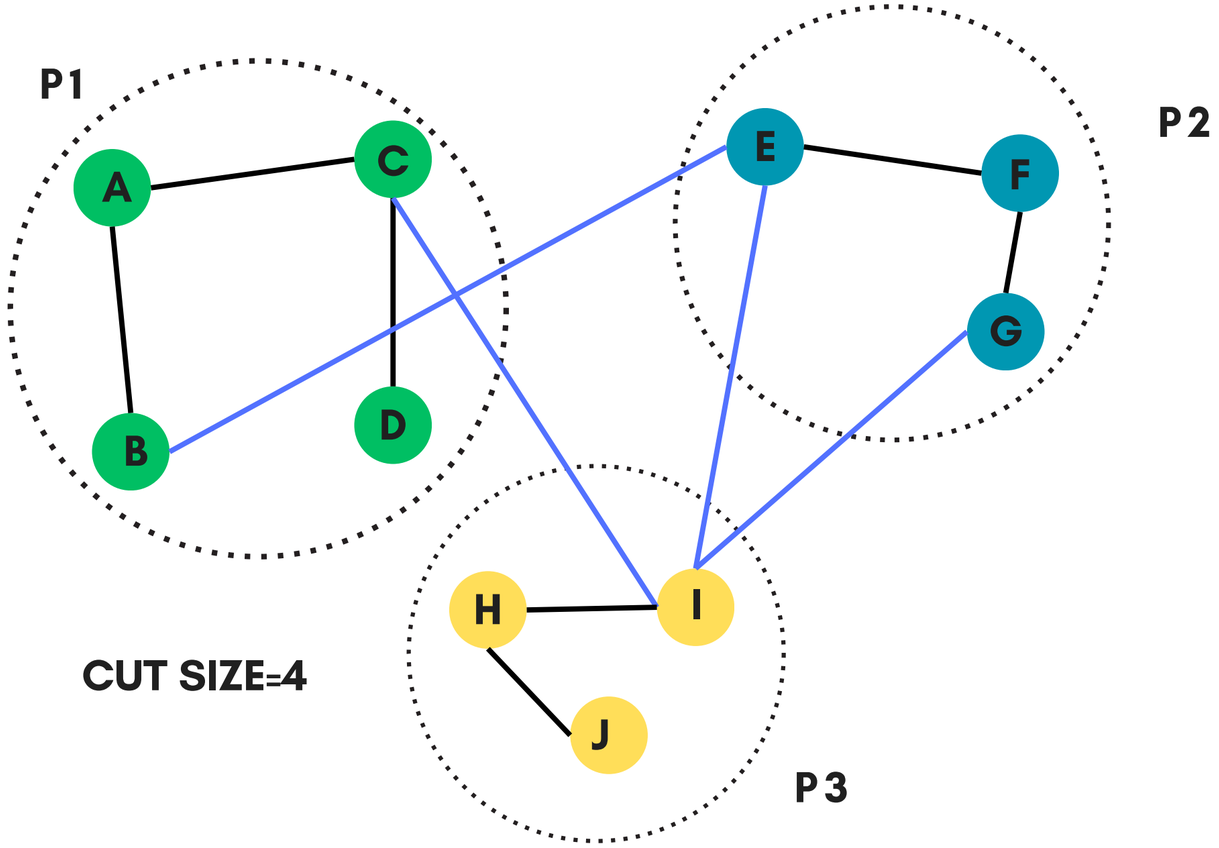

A
C
E
F
B
D
G
H
I
J

START WITH A RANDOM PARTITION



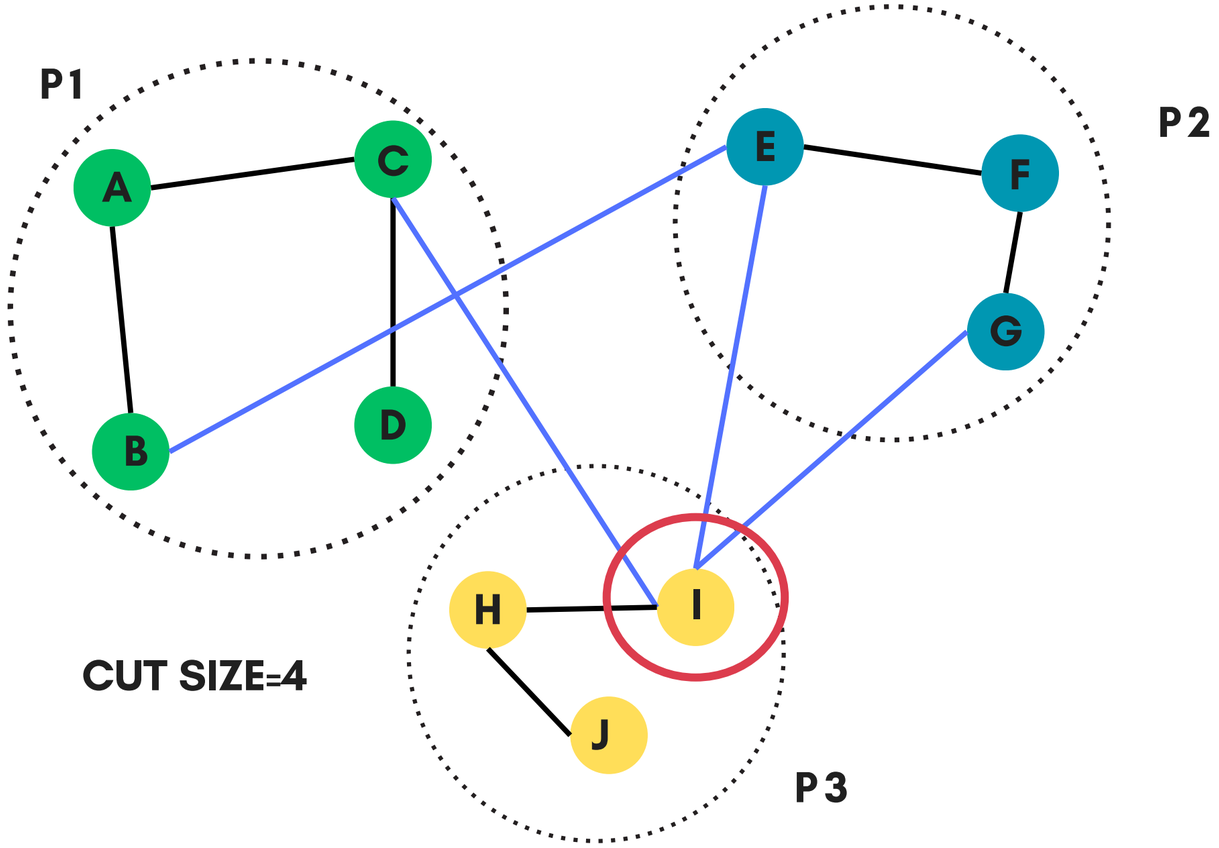
CALCULATE GAIN AND BUILD GAIN TABLE

GAIN = # of external edges - # of internal edges



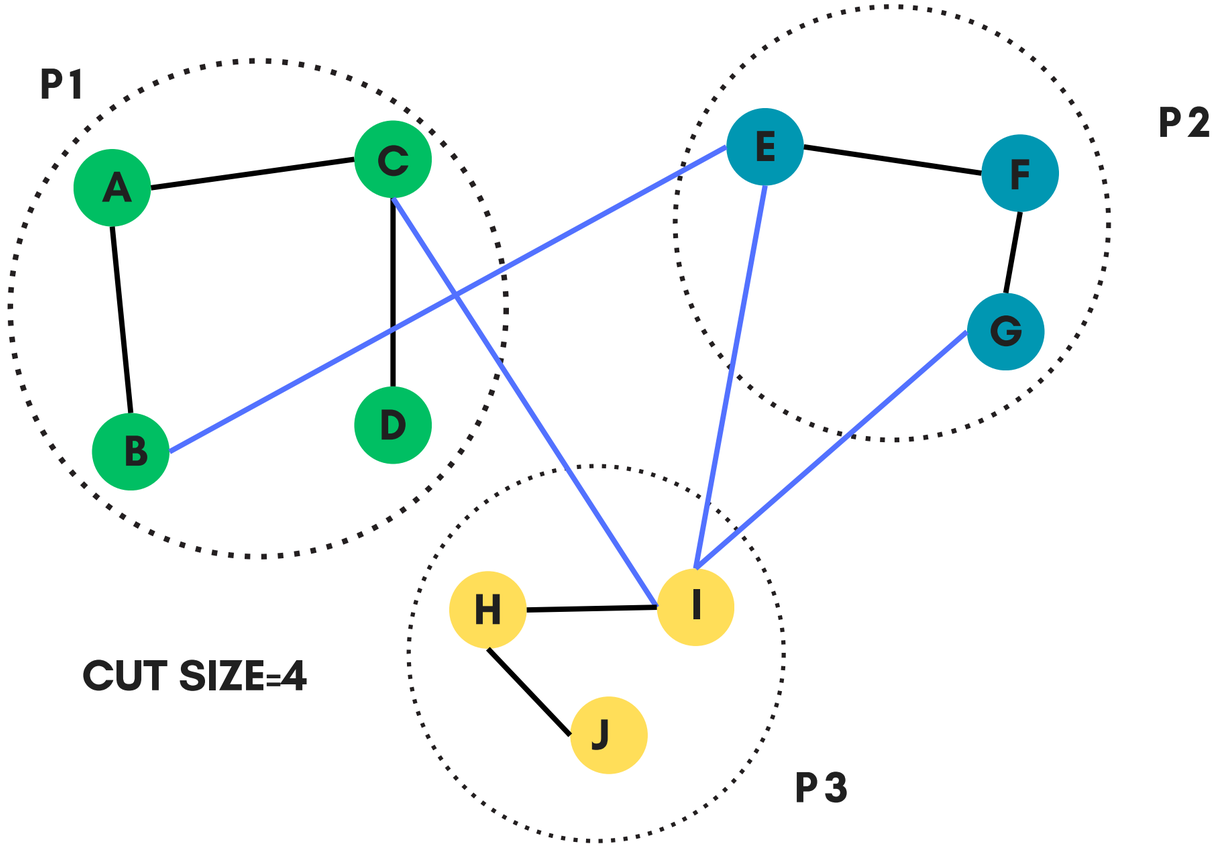
GAIN = # of external edges - # of internal edges

GAIN(I)= 3-1 = 2

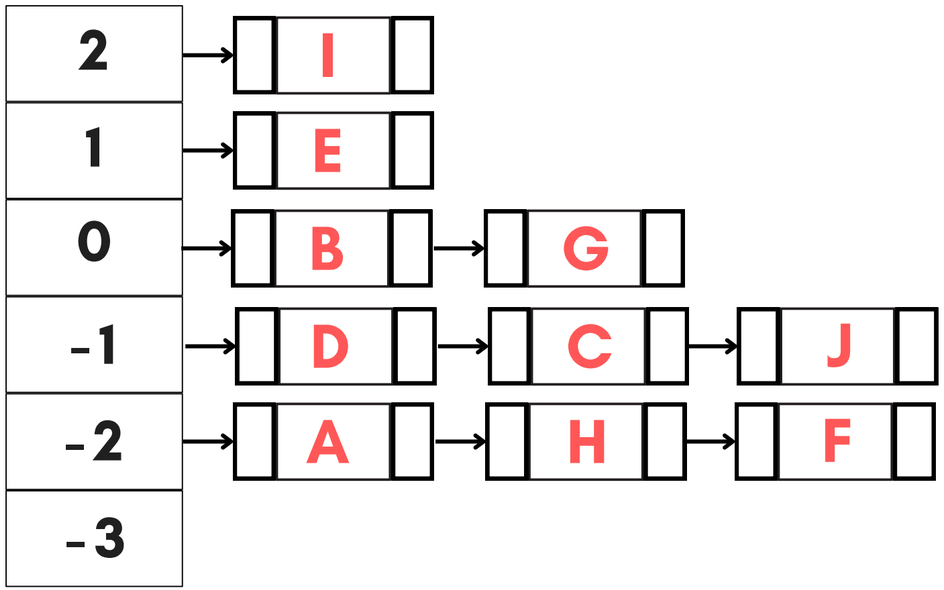


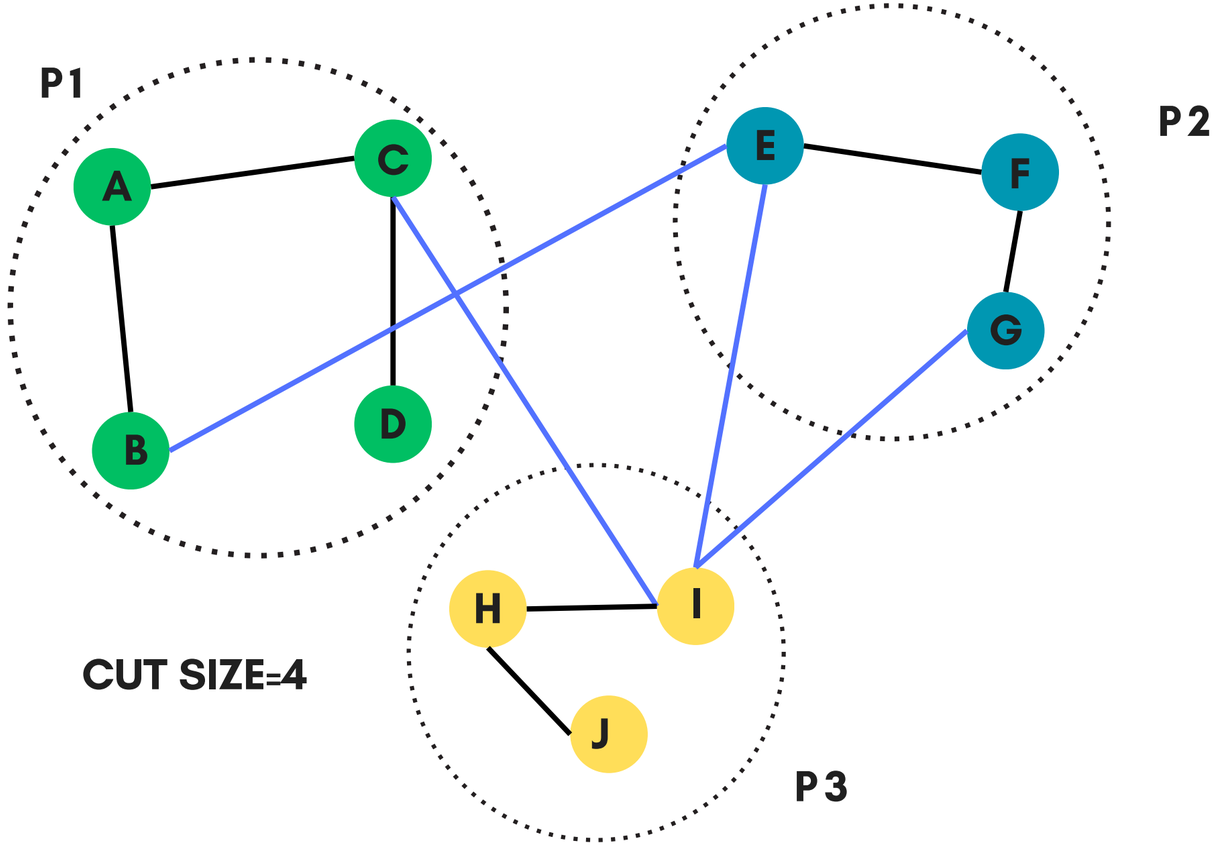
```
For every node ==>
        for each of its neighbors ==>
                increment external or
                internal edge count
```

O(m) complexity to calculate gains

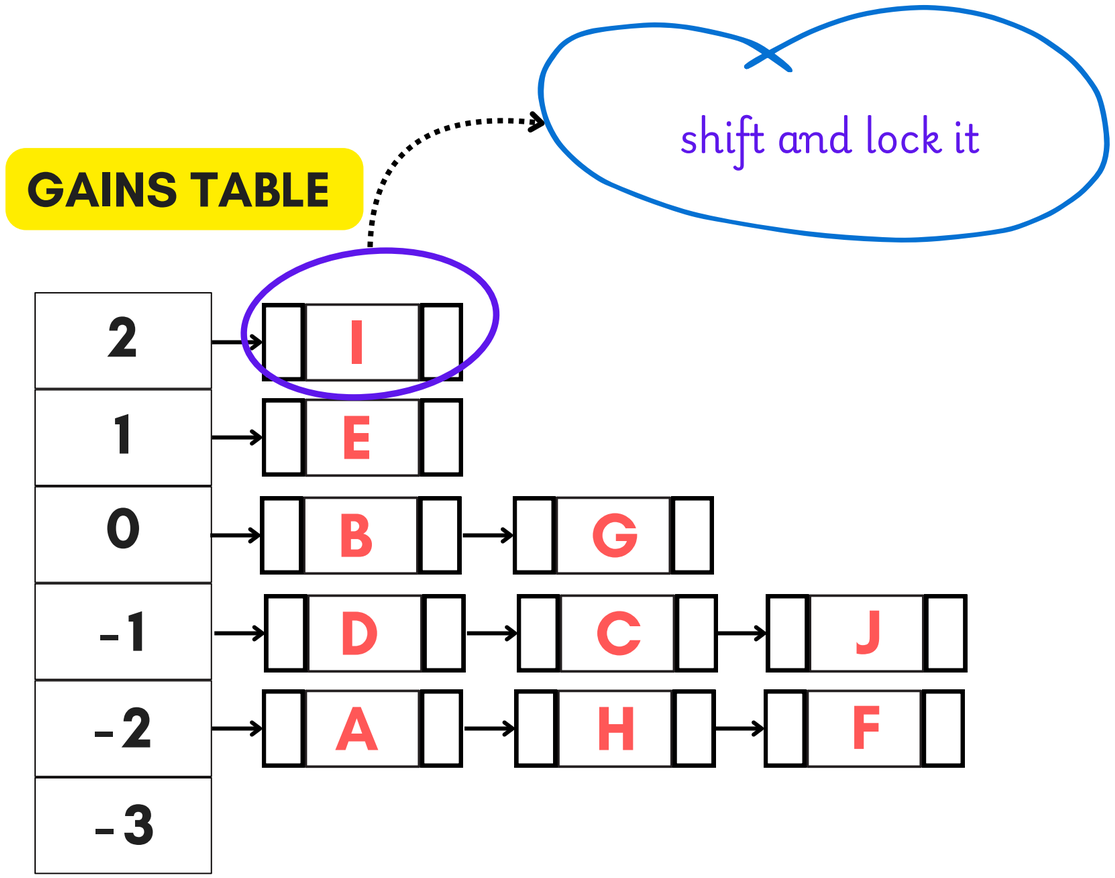


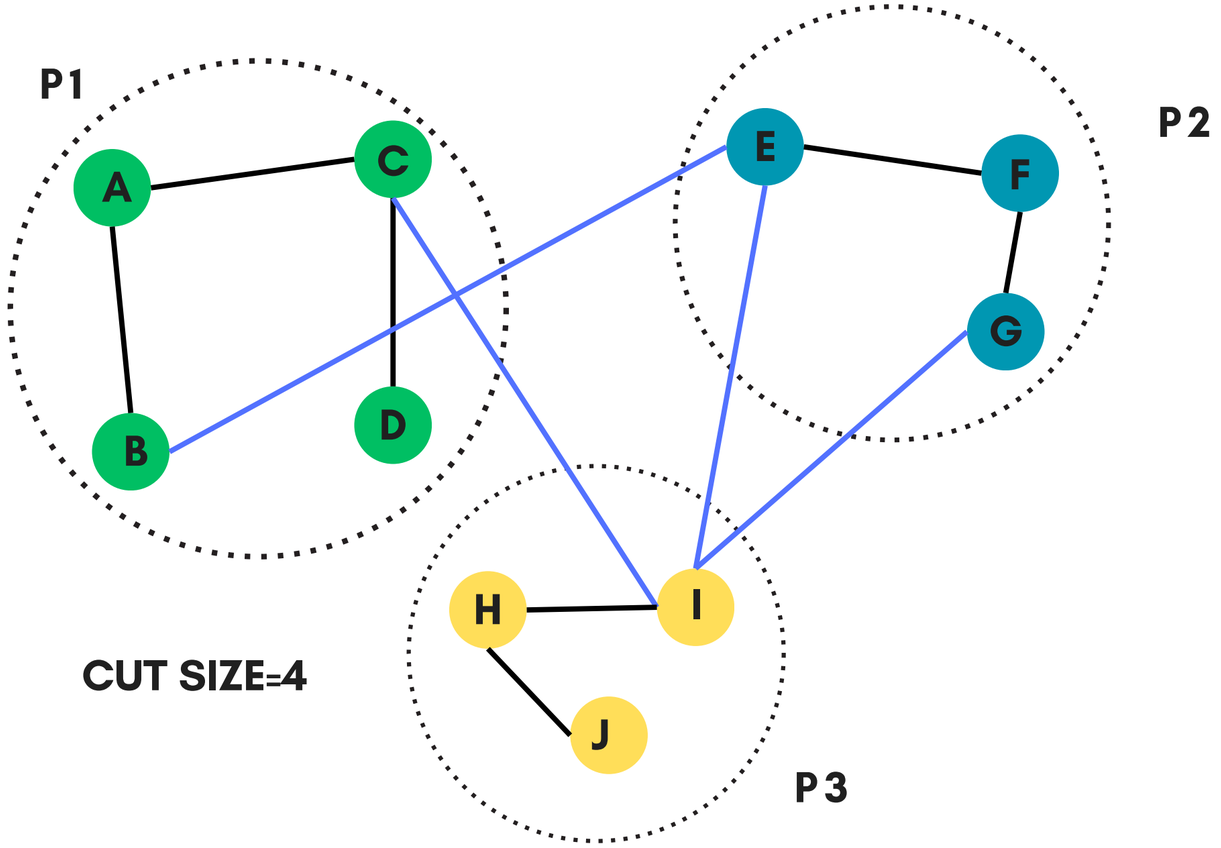
GAINS TABLE





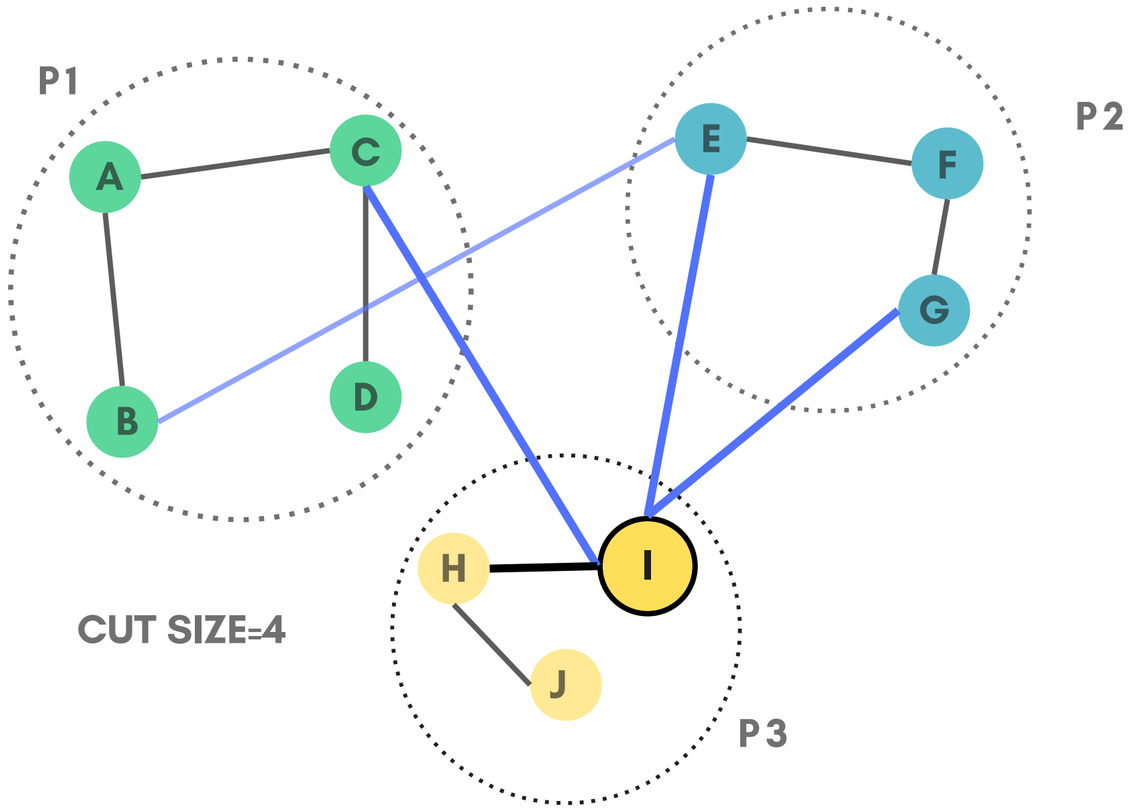
SHIFT AND LOCK





WHICH PARTITION IS OPTIMAL TO MOVE

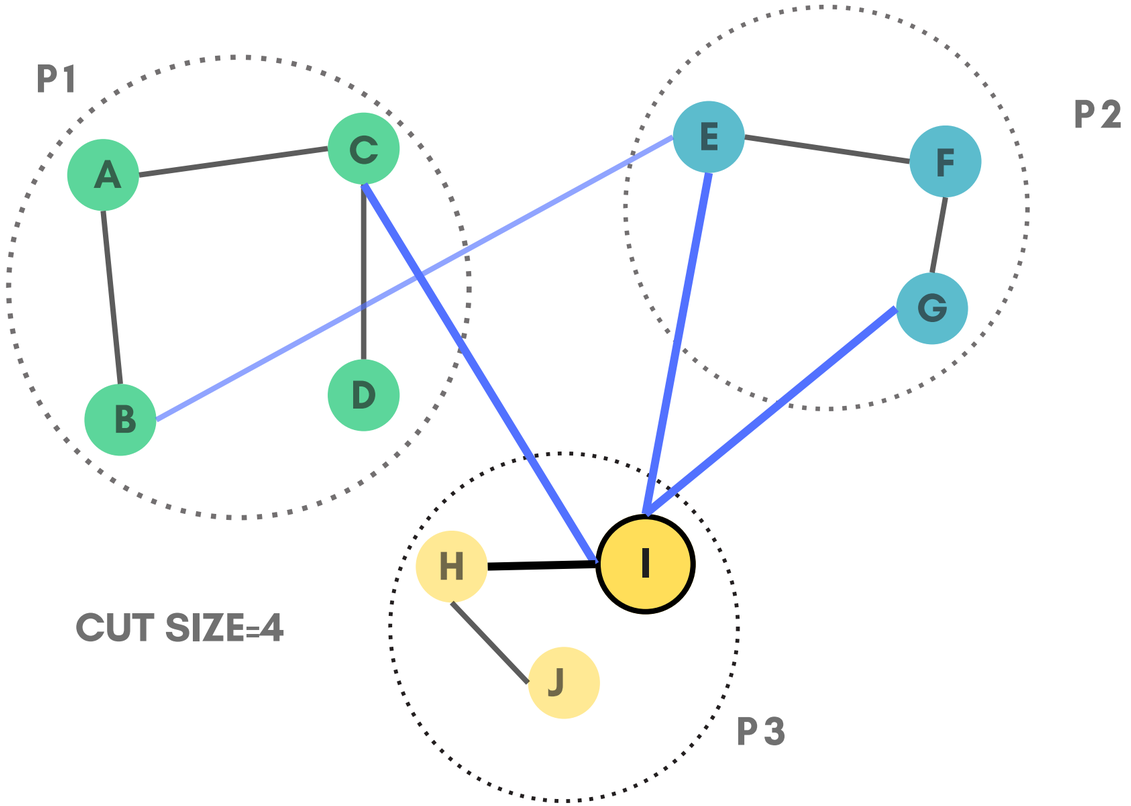
```
find node with maximal gain
if gain is positive
      find a partition such that==>
          gain is minimized &&
          balance is maintained
```



WHICH PARTITION IS OPTIMAL TO MOVE

NODE ‘I’ HAS,

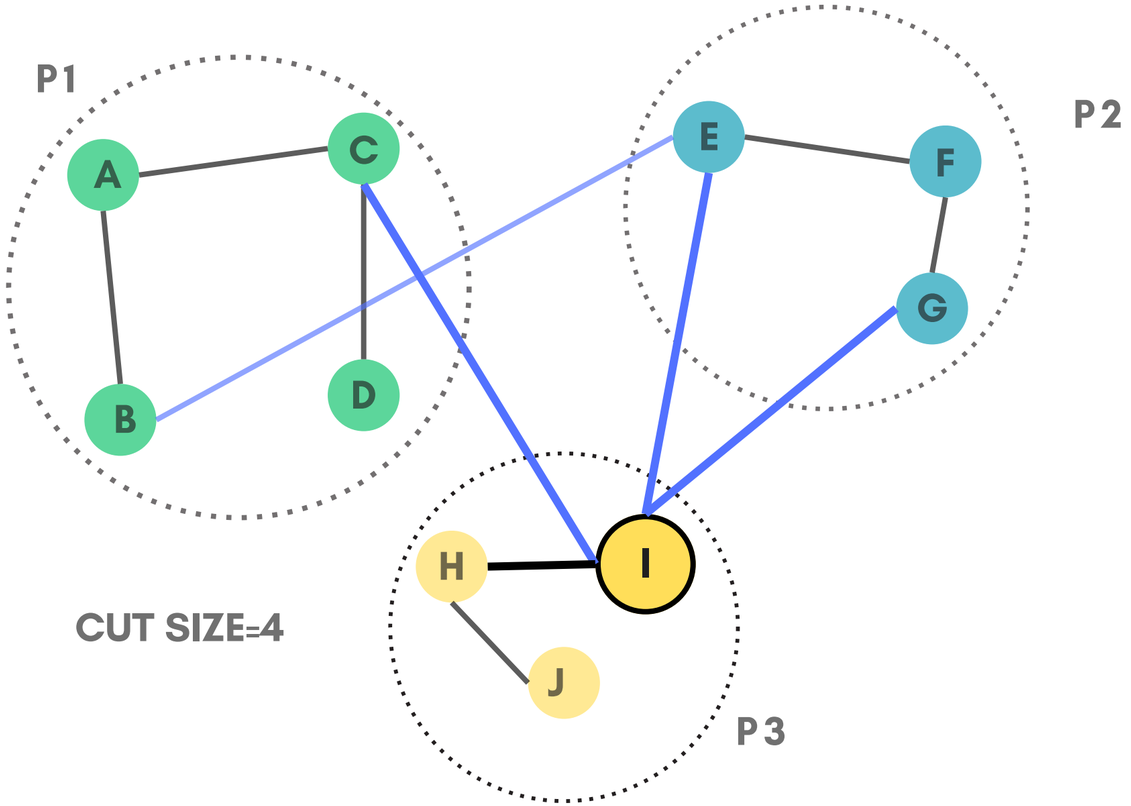
- TWO EXTERNAL EDGES TO PARTITION P2
- ONE EXTERNAL EDGE TO PARTITION P1



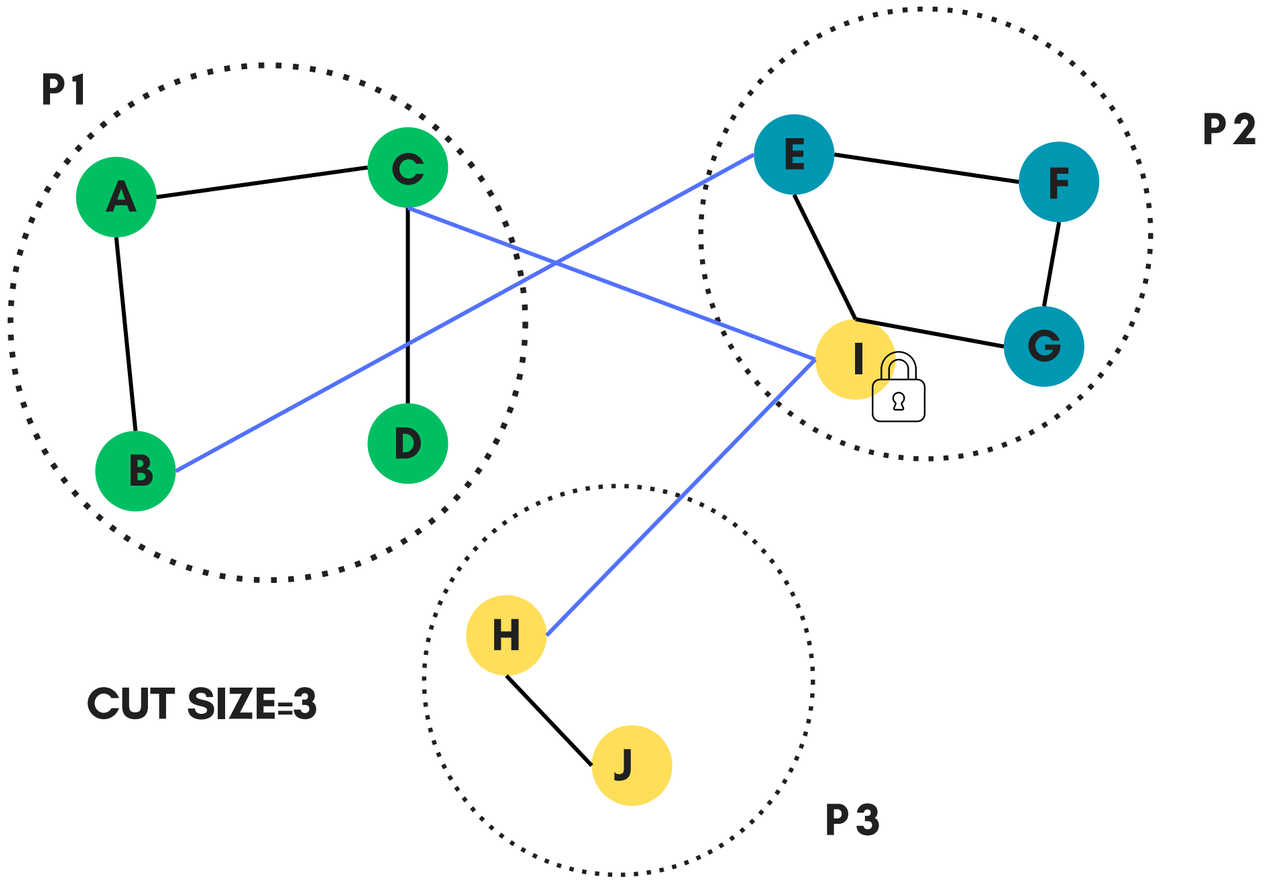
WHICH PARTITION IS OPTIMAL TO MOVE

NODE ‘I’ HAS,

- TWO EXTERNAL EDGES TO PARTITION P2
- ONE EXTERNAL EDGE TO PARTITION P1

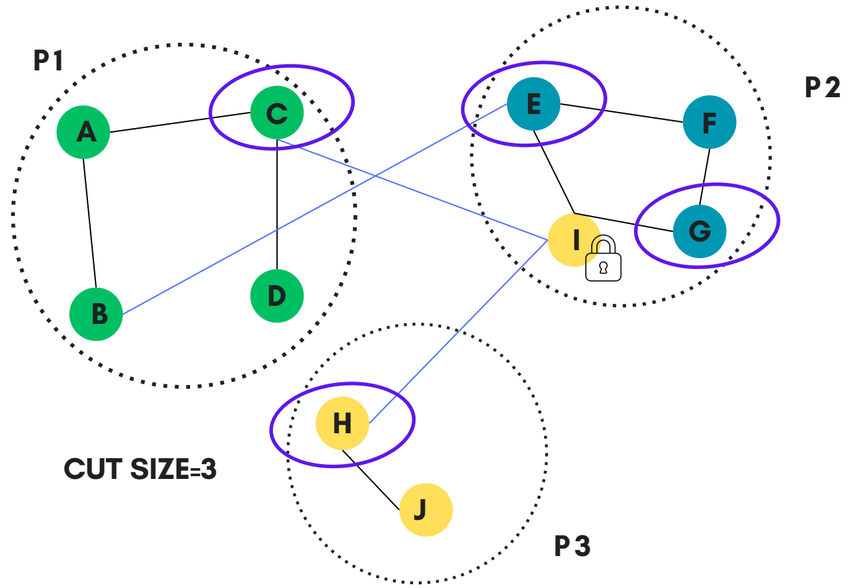


P2 IS THE OPTIMAL ONE

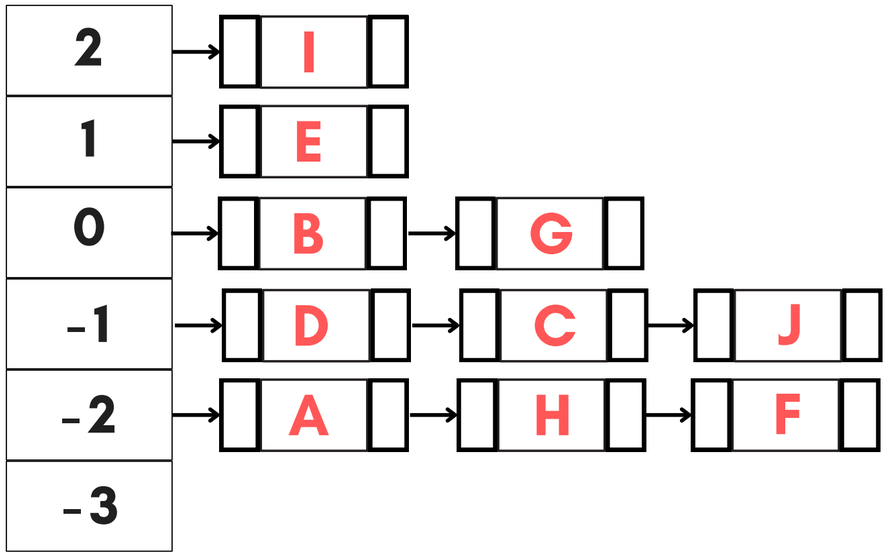

P1
A
C
B
D
P2
E
F
I
G
H
J
P3
CUT SIZE=3

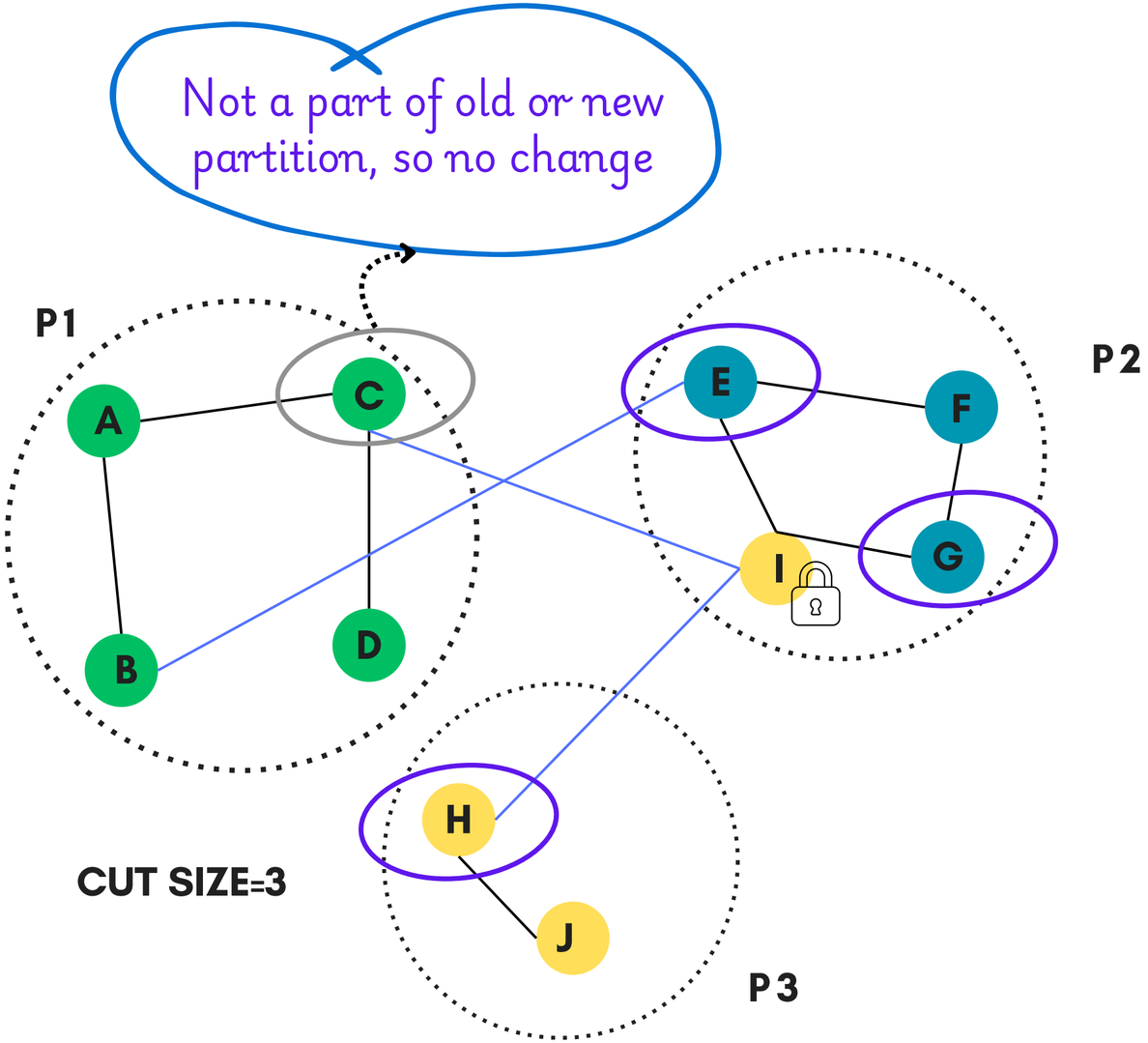
UPDATE GAINS FOR THE NEIGHBOURS

```
// update the gain of the neighbors
    for (int i = 0; i < node->neighbors.size(); i++)
    {
        // neighbour with old partition will have gain +2
        if (node->neighbors[i]->partition == old_partition)
        {
            gainTable->shiftUp(node->neighbors[i]);
            gain_update+=2;
        }
        else if (node->neighbors[i]->partition == partition)
        {
            gainTable->shiftDown(node->neighbors[i]);
            gain_update-=2;
        }
    }
```

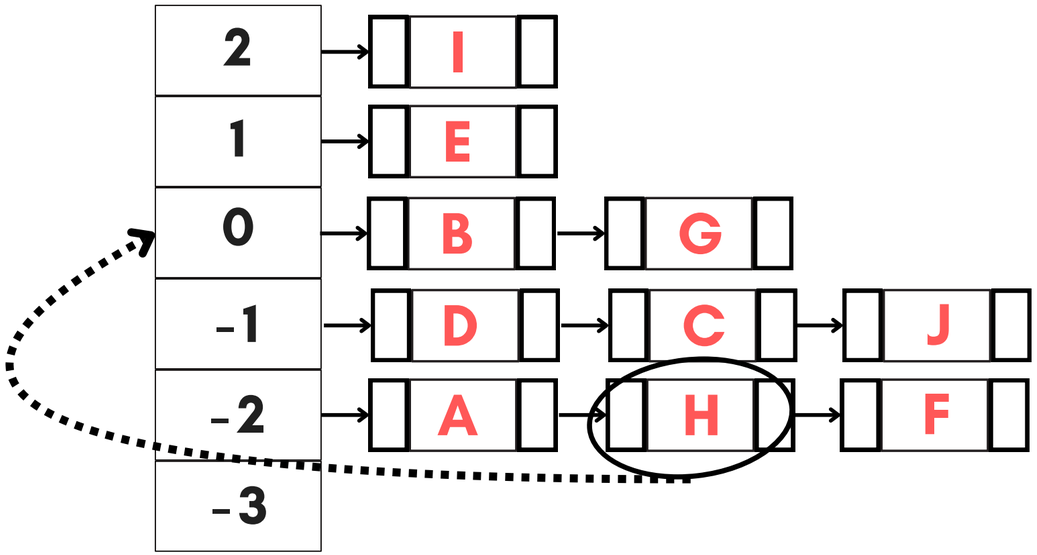


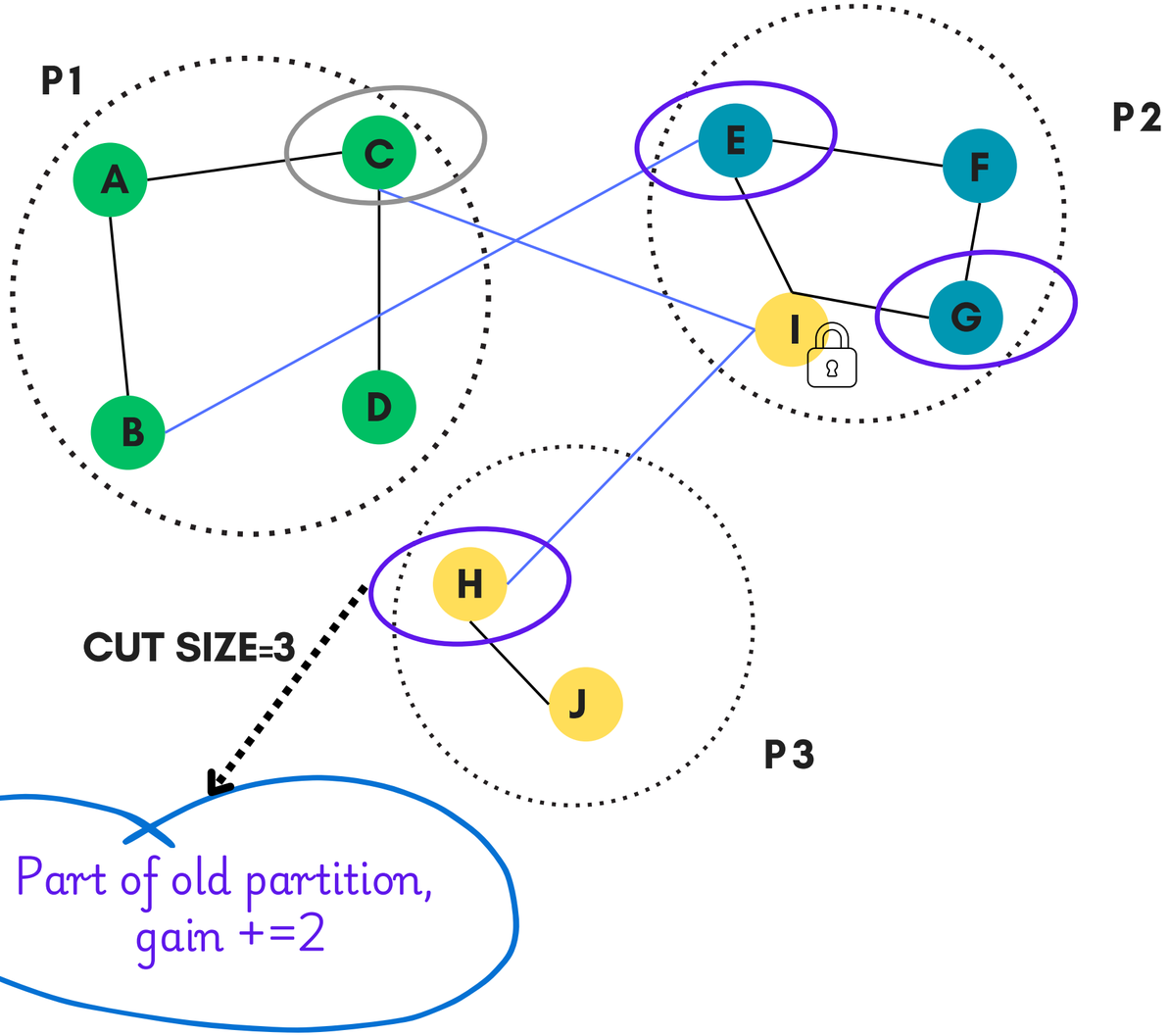
UPDATE GAINS FOR THE NEIGHBOURS



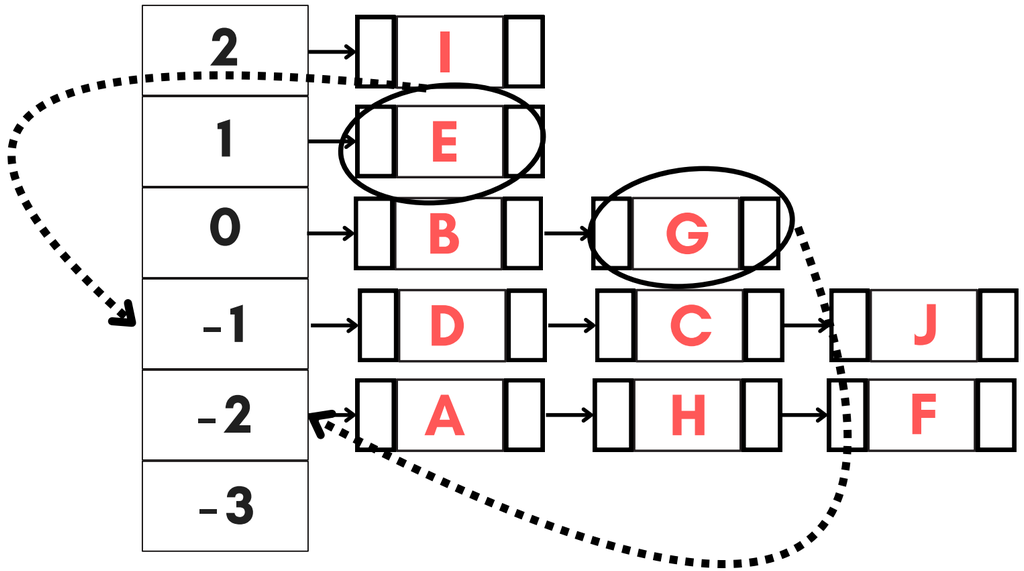


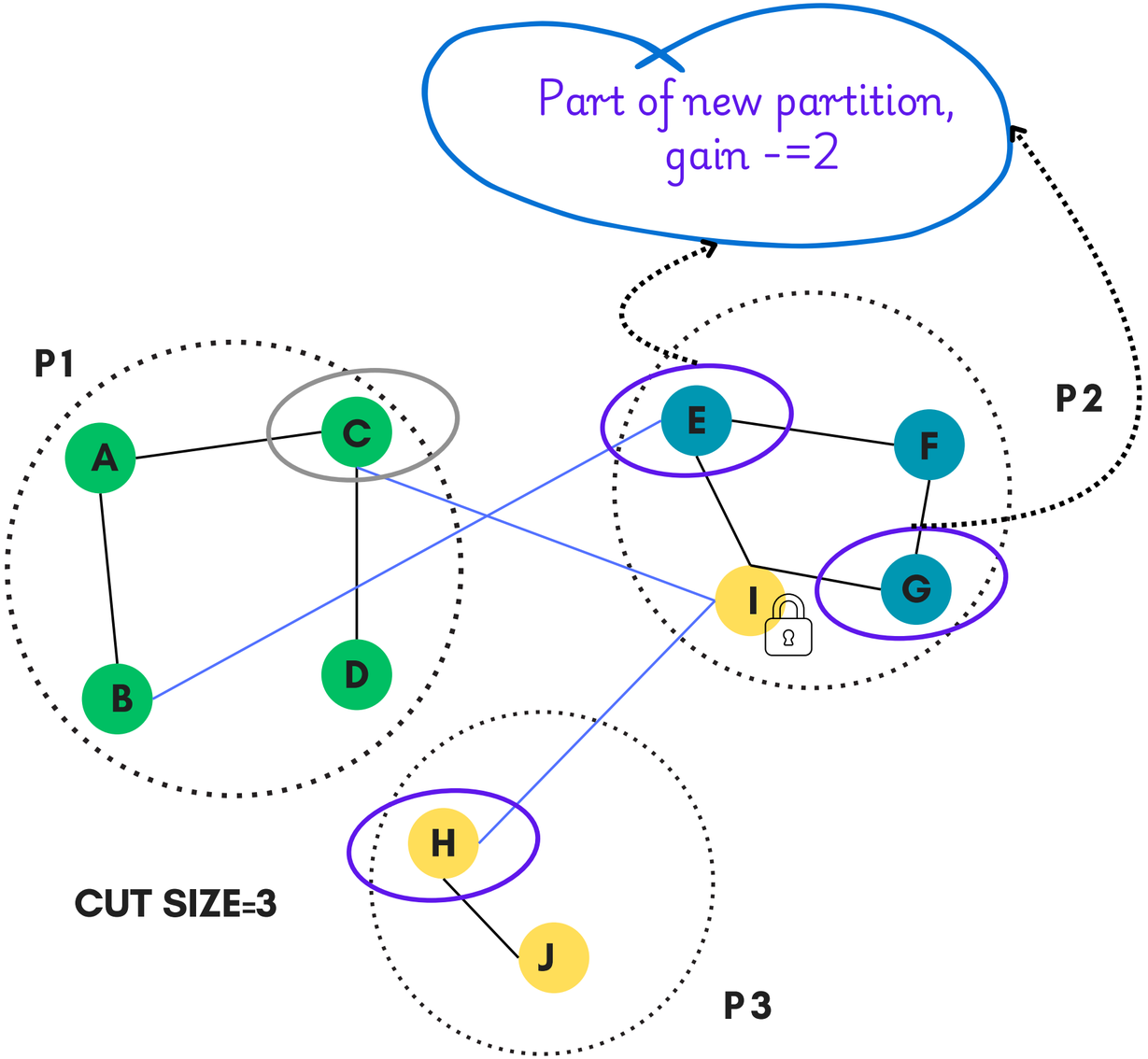
UPDATE GAINS FOR THE NEIGHBOURS

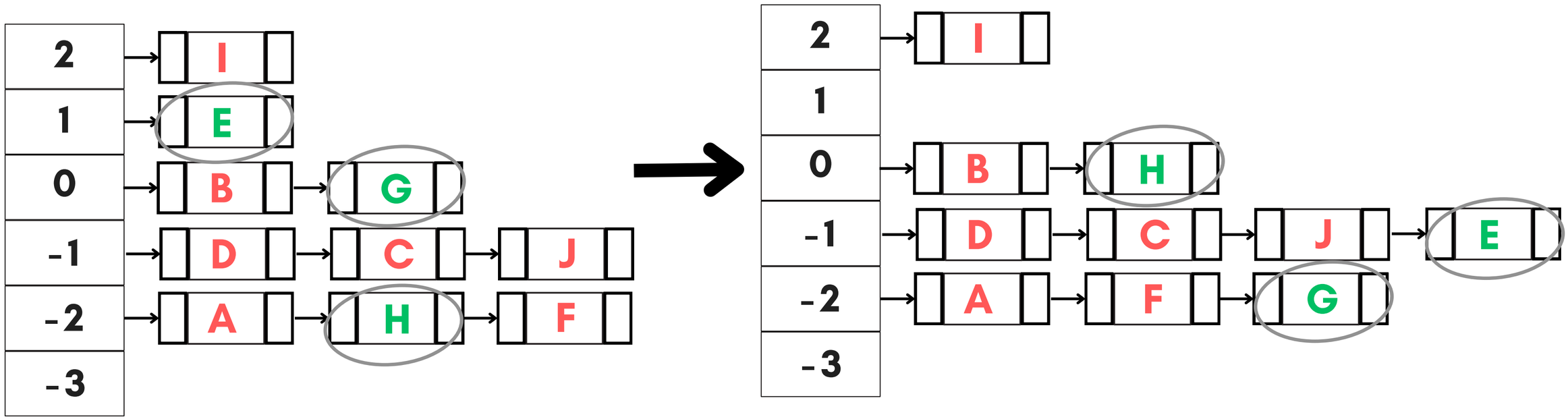




UPDATE GAINS FOR THE NEIGHBOURS

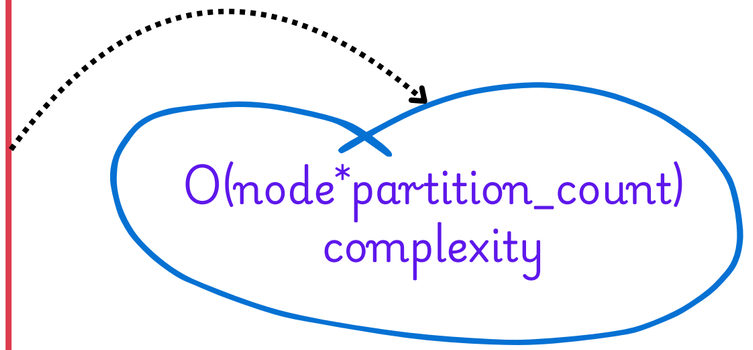





2
1
0
-1
-2
-3
I
E
B
G
D
C
J
A
H
F
2
1
0
-1
-2
-3
I
B
H
D
C
J
E
A
F
G

Algorithm 2.7. The *Global Kernighan-Lin Refinement* algorithm

```
1:  procedure GKLR (graph G = (V, E), partition P_k, max balance maxbal)
2:      repeat "pass"
3:          Create the gains table and the set of unlocked vertices V_v
4:          while V_v ≠ ∅ or dep gains have not been negative do
5:              Select the vertex v of greatest gain. Let V_j ∈ P_k such that v ∈ V_j
6:              for all V_i ∈ P_k − {V_j} such that V_i ∪ {v} maintains maxbal do
7:                  Compute the gain of adding v to V_i
8:              end for
9:              Select the part V_i ∈ P_k of maximal gain g for the vertex v
10:             Record the transfer of v to V_i and the corresponding gain g
11:             for all unlocked vertex v' adjacent to v do
12:                 Update the gains table for v'
13:             end for
14:             Lock v
15:         end while
16:         Select the set of vertices to transfer that maximizes the gain g
17:         if g > 0 then
18:             Alter P_k by adding the set of vertices to transfer
19:         end if
20:     until g ≤ 0 or a predefined number of passes is reached
21:     return P_k
22: end procedure
```



CONTINUE **SHIFT_AND_LOCK** UNTIL

- GAIN CAN'T BE FURTHER MINIMIZED
- ALL VERTICES ARE LOCKED

THEN UNLOCK EVERY NODE AND REPEAT ANOTHER PASS

FURTHER IMPROVEMENTS

Hendrickson – Leland algorithm	• **uses FM algorithm for k-way partitioning** • **uses k(k-1) gain tables** • **used when k is low**
Walshaw-Cross refinement algorithm	• **almost the same as GKLR** • **Uses load distribution algorithms for better balancing**

AGENDA 3

EXPERIMENTATION AND RESULTS

1905065

DATASET

DATASET DIVISIONS

Type	Node-count
Small Graph	200-800
Large Graph	3000-6000

Type	Edge-Probability[1]
Dense Graph	0.8
Sparse Graph	0.2
Random	0.5

1. Edge probability means probability of existence of an edge between 2 specific vertices

DATASET

DATASET DIVISIONS

Type	Node-count
Small Graph	200-800
Large Graph	3000-6000

Type	Edge-Probability
Dense Graph	0.8
Sparse Graph	0.2
Random	0.5

100 Samples in each of the 6 graph types

DATASET

DATASET CREATION

Type	Node-count
Small Graph	200-800
Large Graph	3000-6000

Type	Edge-Probability
Dense Graph	0.8
Sparse Graph	0.2
Random	0.5

```
=> num_nodes = random.randint(num_nodes_min, num_nodes_max)
```

DATASET

DATASET CREATION

Type	Node-count
Small Graph	200-800
Large Graph	3000-6000

Type	Edge-Probability
Dense Graph	0.8
Sparse Graph	0.2
Random	0.5

=> for each pair of vertices u and v:

decide whether there will be an edge between them with Edge-probability

DATASET

DATASET CREATION

Type	Node-count
Small Graph	200-800
Large Graph	3000-6000

Type	Edge-Probability
Dense Graph	0.8
Sparse Graph	0.2
Random	0.5

=> for each pair of vertices u and v:

decide whether there will be an edge between them with Edge-probability

DATASET

DATASET CREATION

Type	Node-count
Small Graph	200-800
Large Graph	3000-6000

Type	Edge-Probability
Dense Graph	0.8
Sparse Graph	0.2
Random	0.5

***Accept this Graph only if it is a connected graph

EXPERIMENTS

MAXIMUM ALLOWABLE DEVIATION

WHAT IF WE INCREASE THE FLEXIBILITY OF PARTION SIZE?

Balance Difference?

Balance Difference= 10%

Let, # vertices= 250

#Partitions = 5

Balanced partitions get 50 vertices each

Balance Difference = 10% means, we will allow

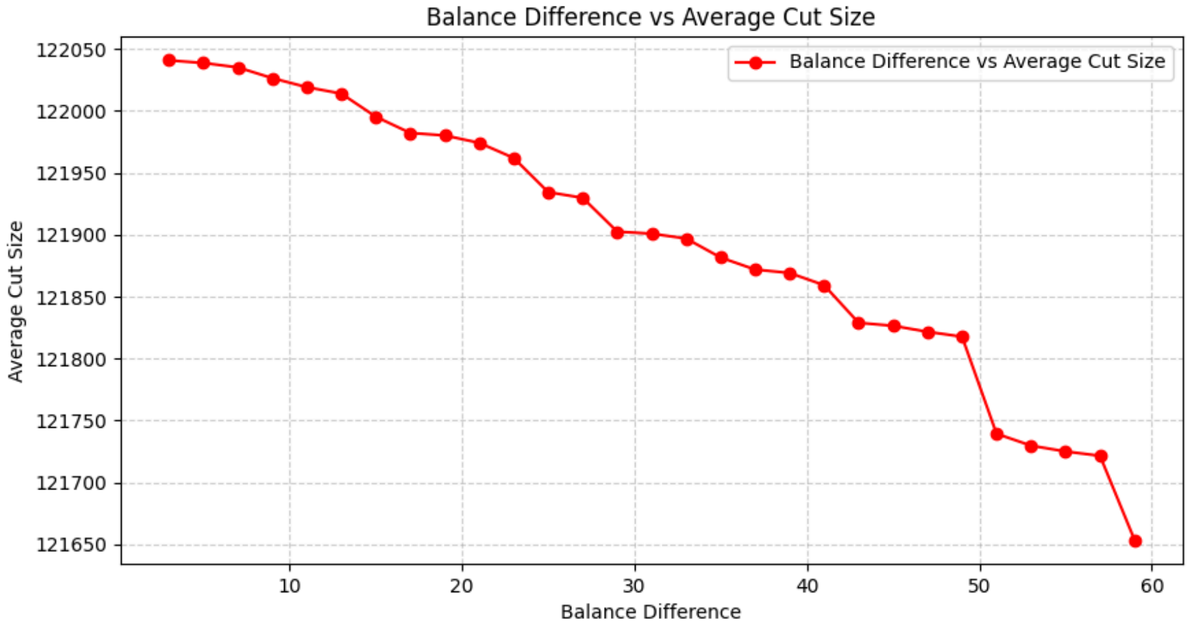
50 + 50*10% = 55 vertices at max in a partition.

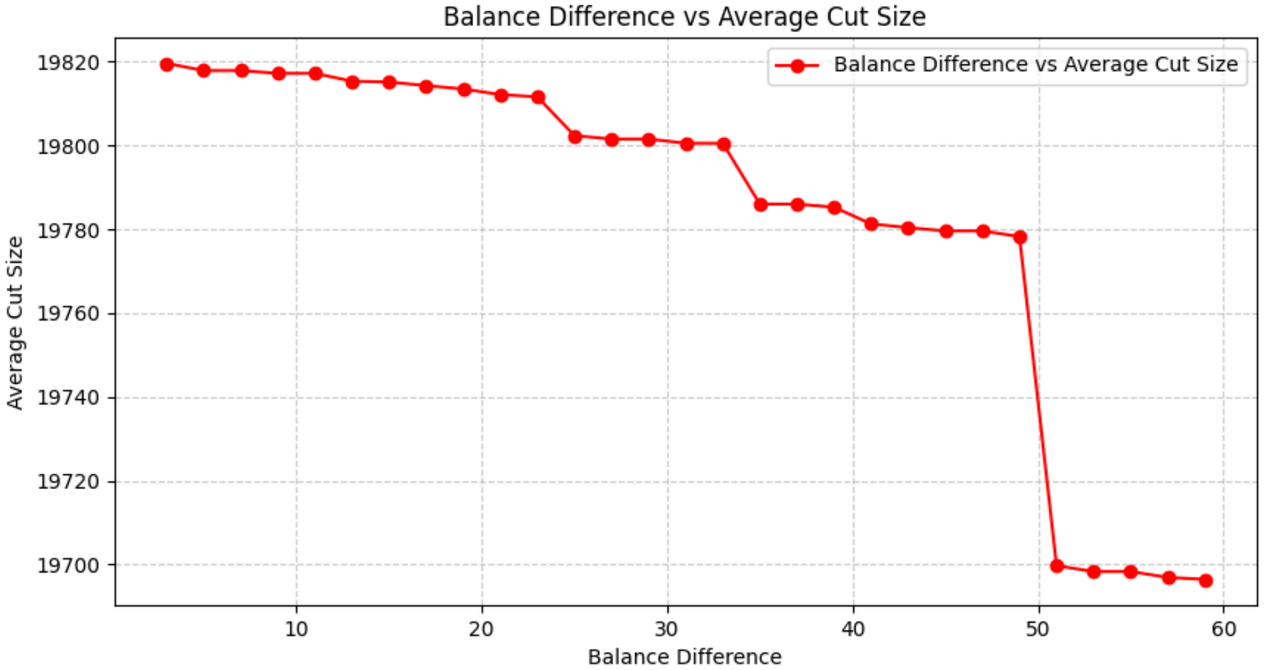
MAXIMUM ALLOWABLE DEVIATION

WHAT IF WE INCREASE THE FLEXIBILITY OF PARTION SIZE?

Balance Difference: 2% to 60%

1. Small-Dense



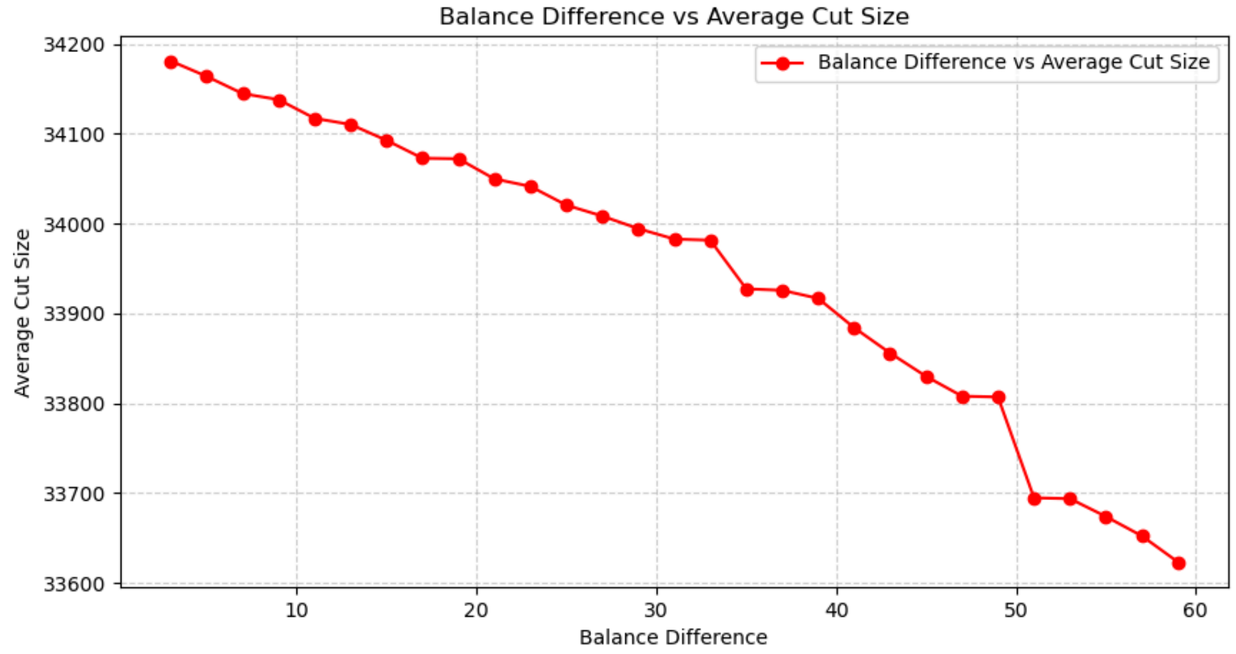


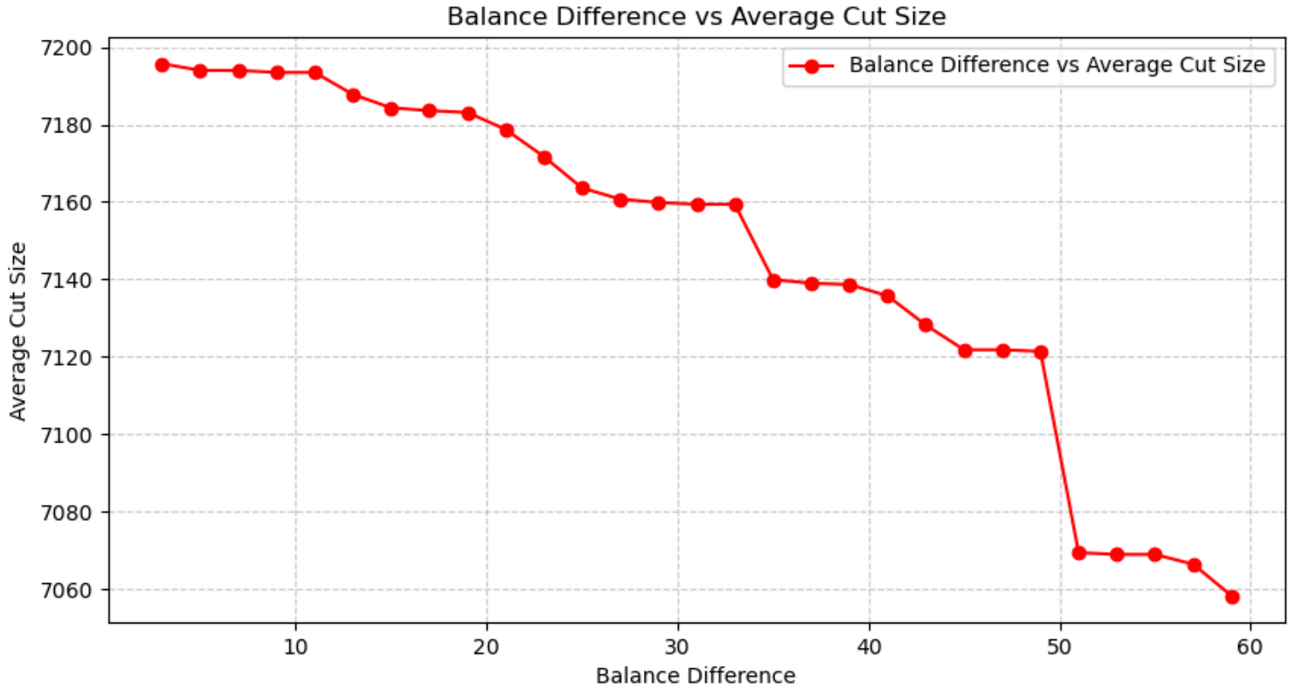
MAXIMUM ALLOWABLE DEVIATION

WHAT IF WE INCREASE THE FLEXIBILITY OF PARTION SIZE?

Balance Difference: 2% to 60%

2. Small-Sparse



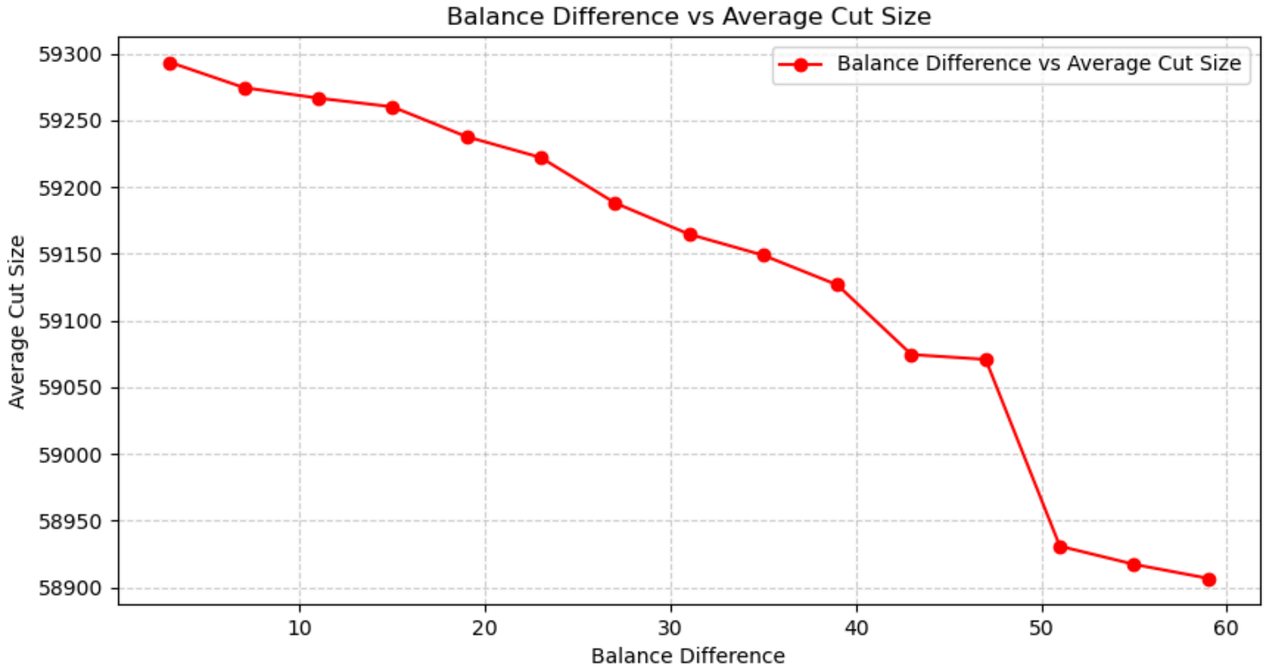


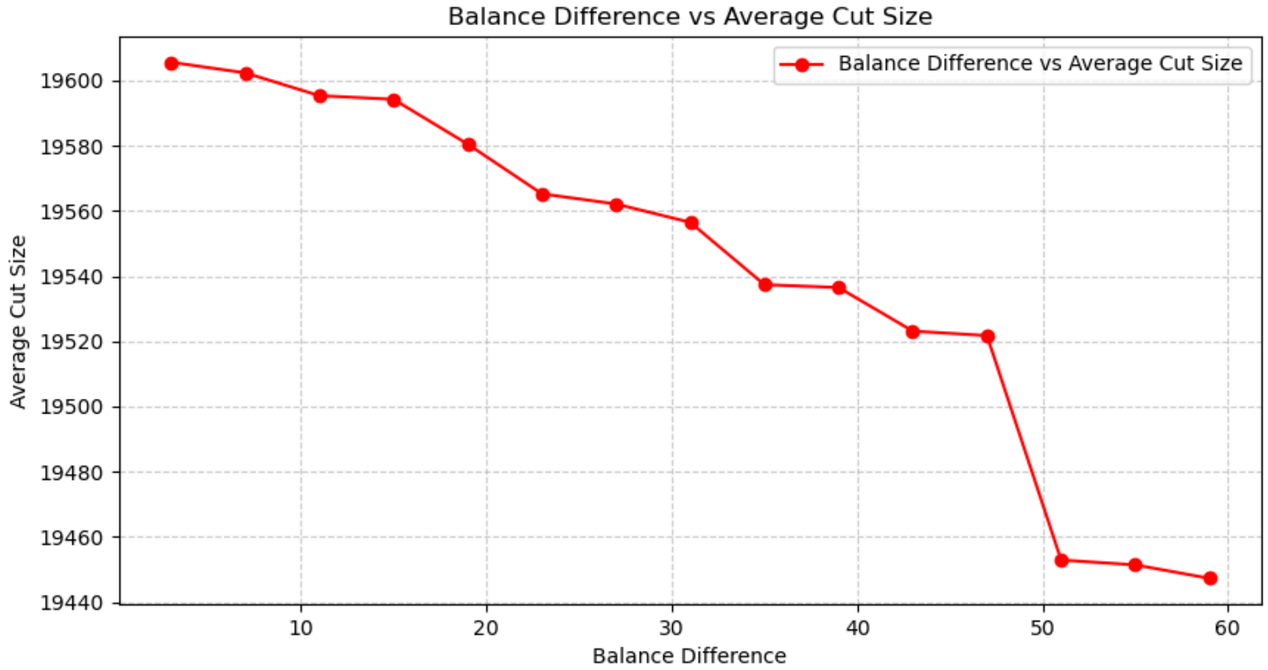
MAXIMUM ALLOWABLE DEVIATION

WHAT IF WE INCREASE THE FLEXIBILITY OF PARTION SIZE?

Balance Difference: 2% to 60%

3. Small-Random

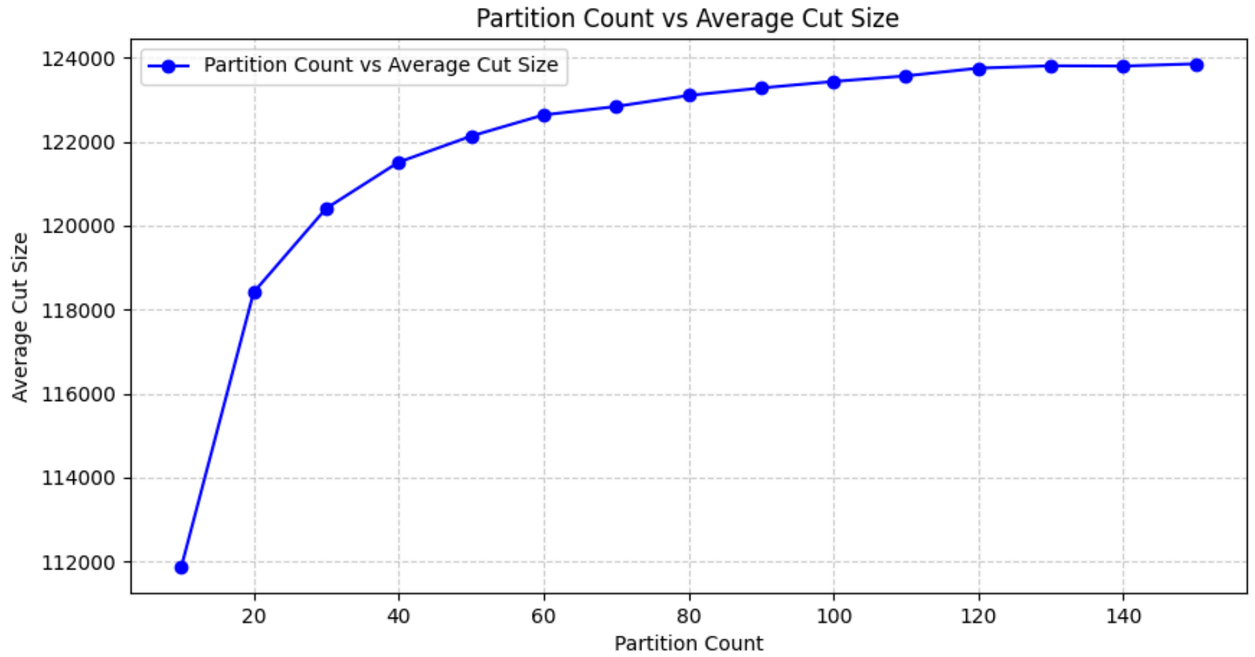


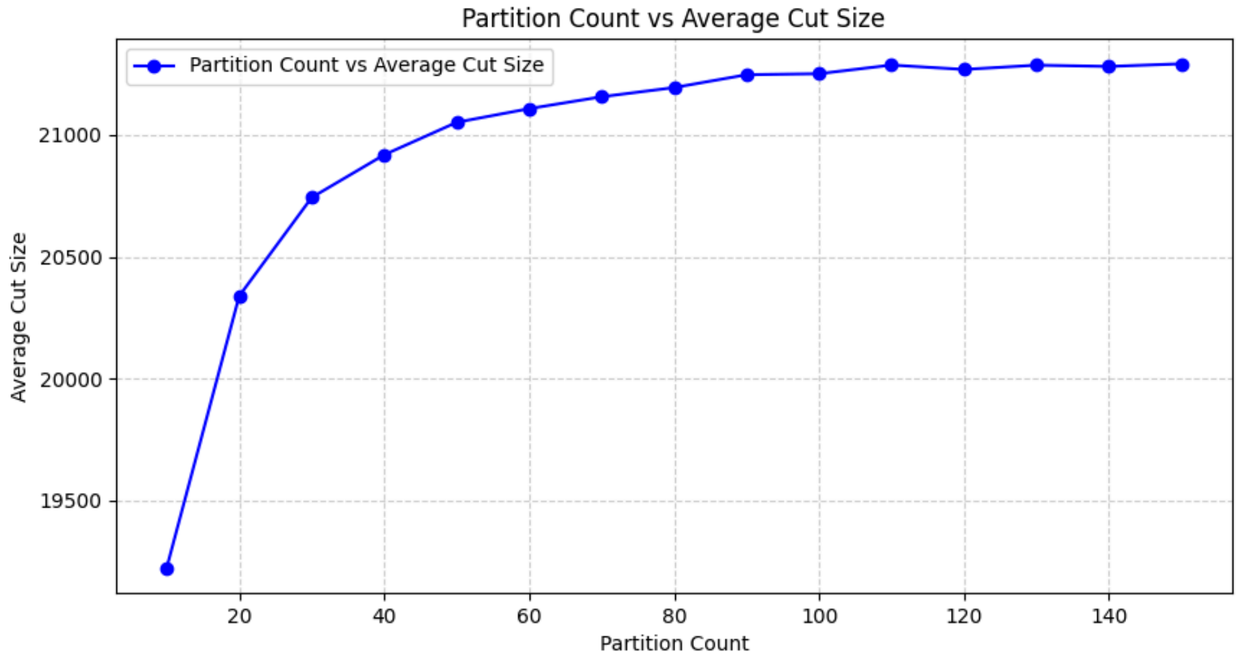


WHAT IF WE INCREASE THE NUMBER OF PARTIONS?

Number of partitions: 10 to 150

1. Small-Dense

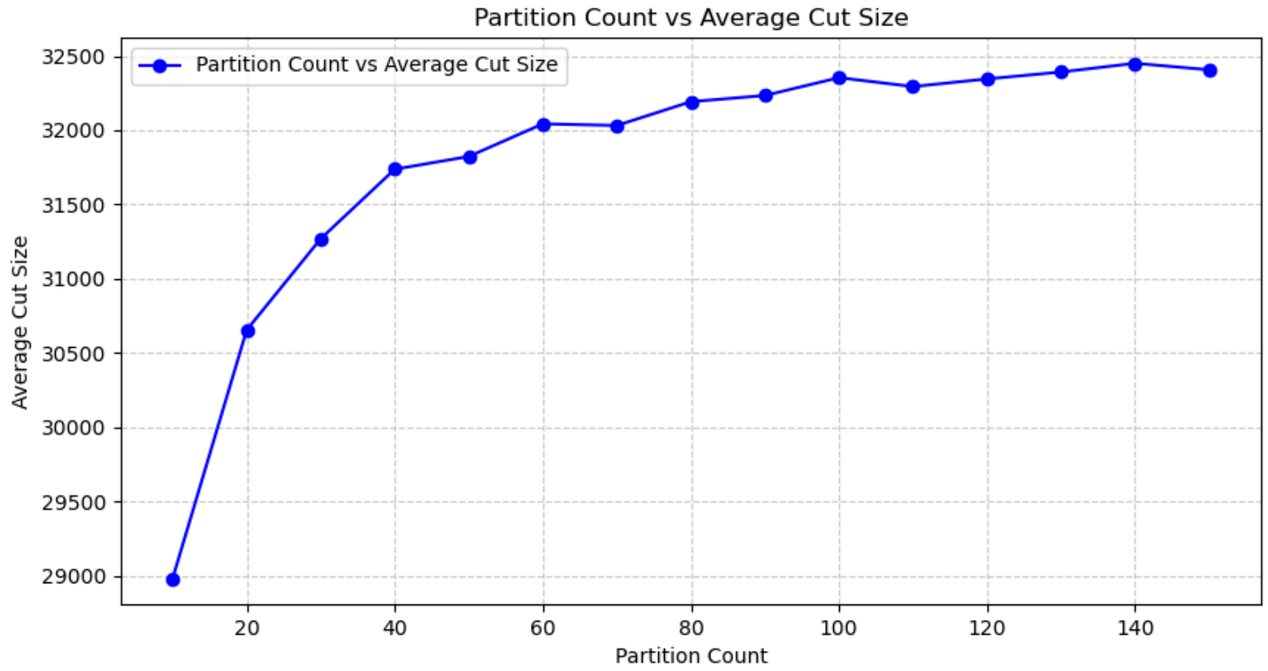


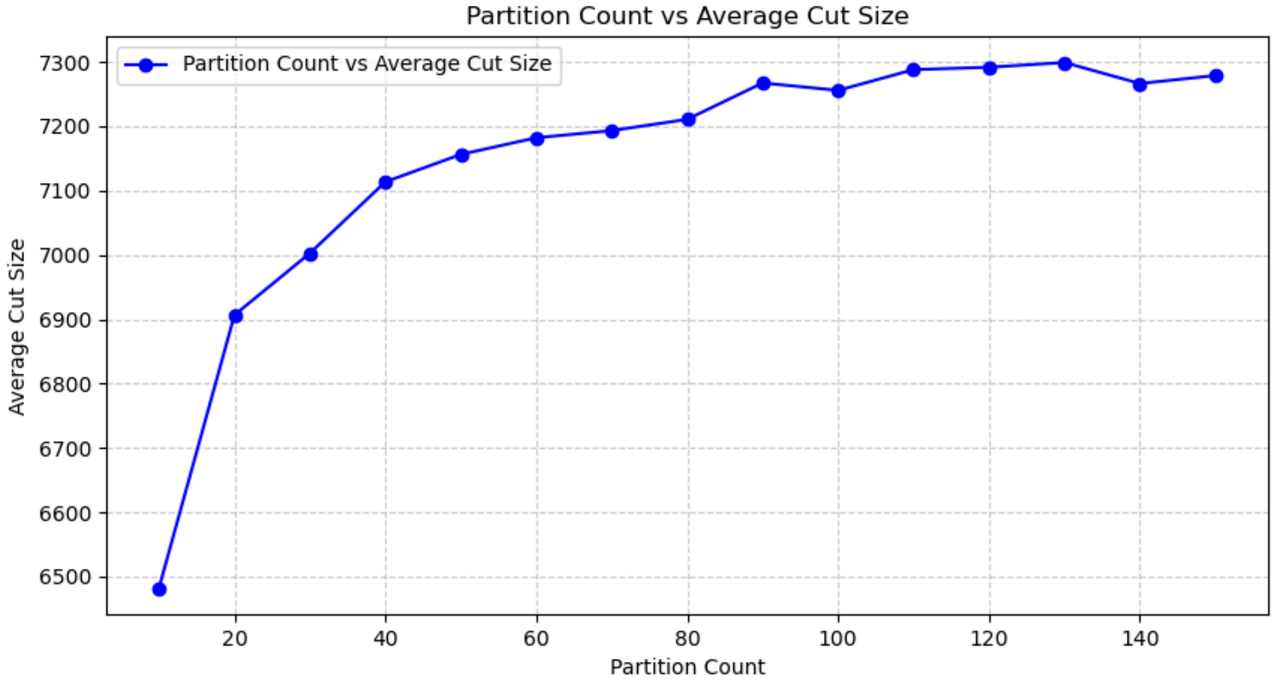


WHAT IF WE INCREASE THE NUMBER OF PARTIONS?

Number of partitions: 10 to 150

2. Small-Sparse

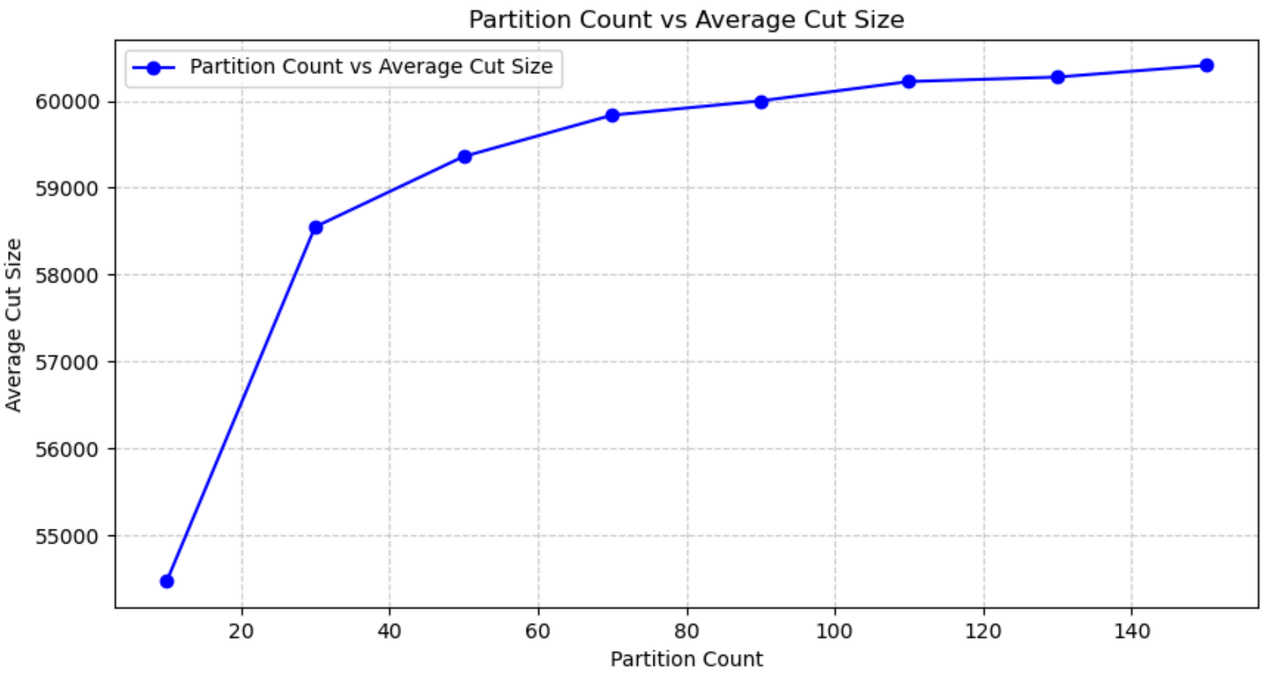


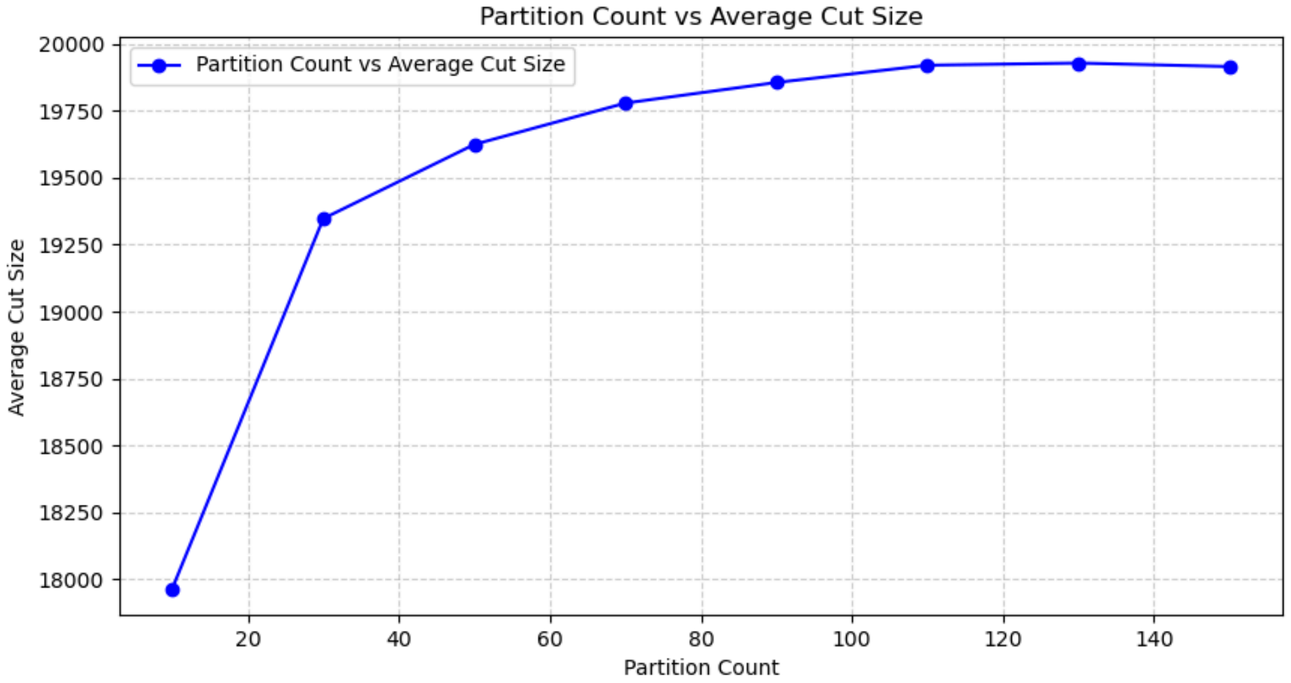


WHAT IF WE INCREASE THE NUMBER OF PARTIONS?

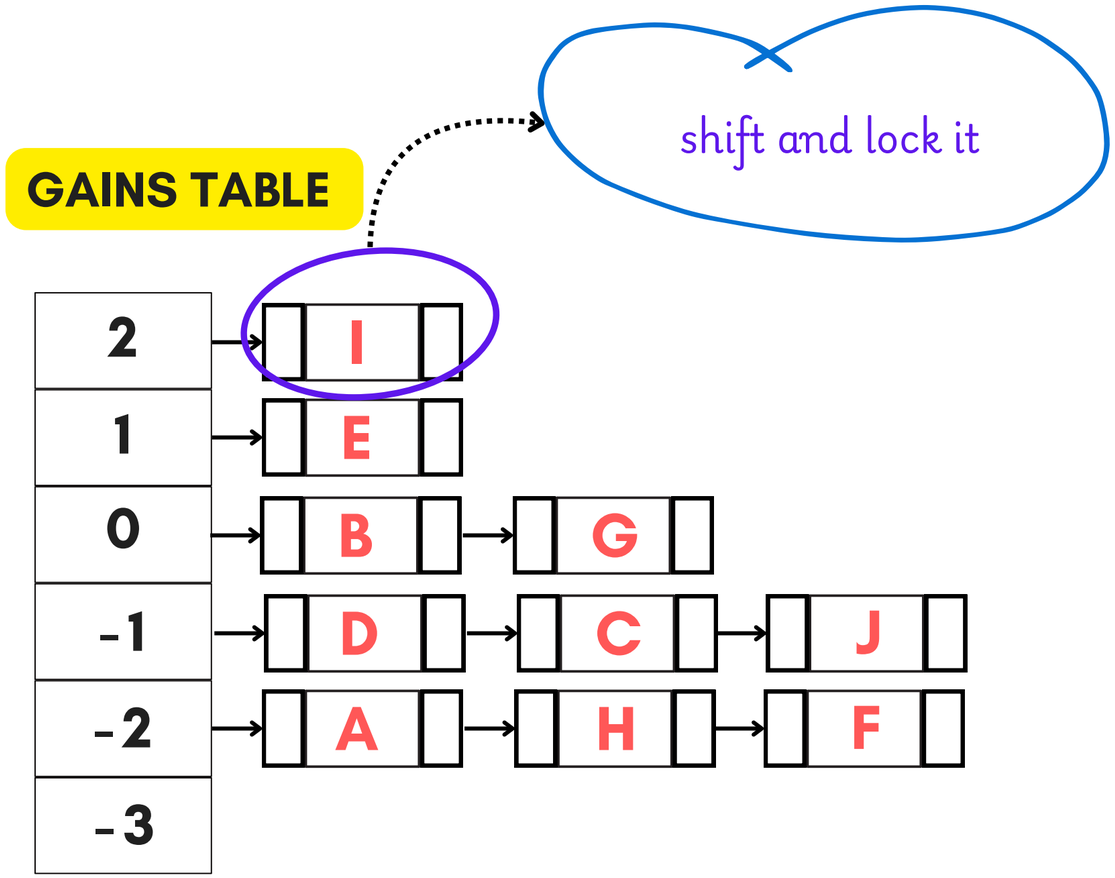
Number of partitions: 10 to 150

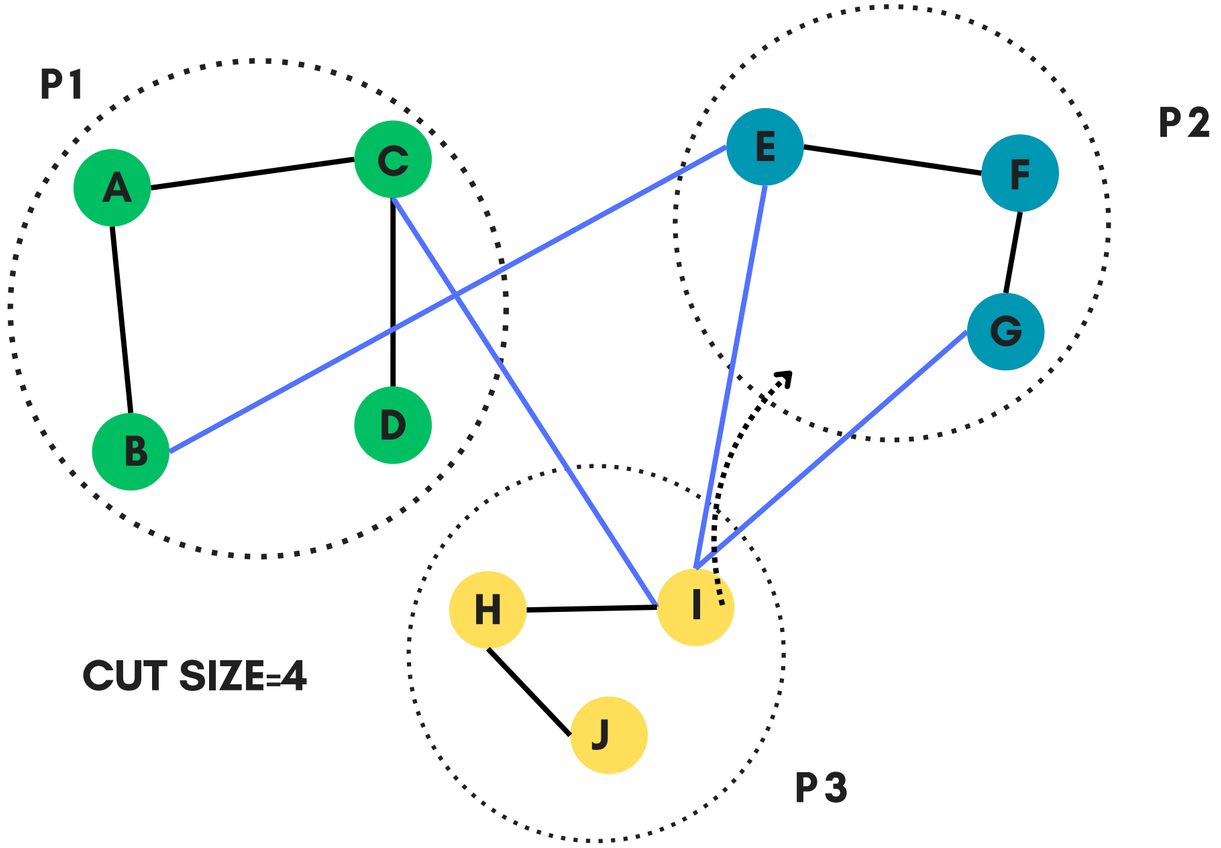
3. Small-Random





SHIFT?

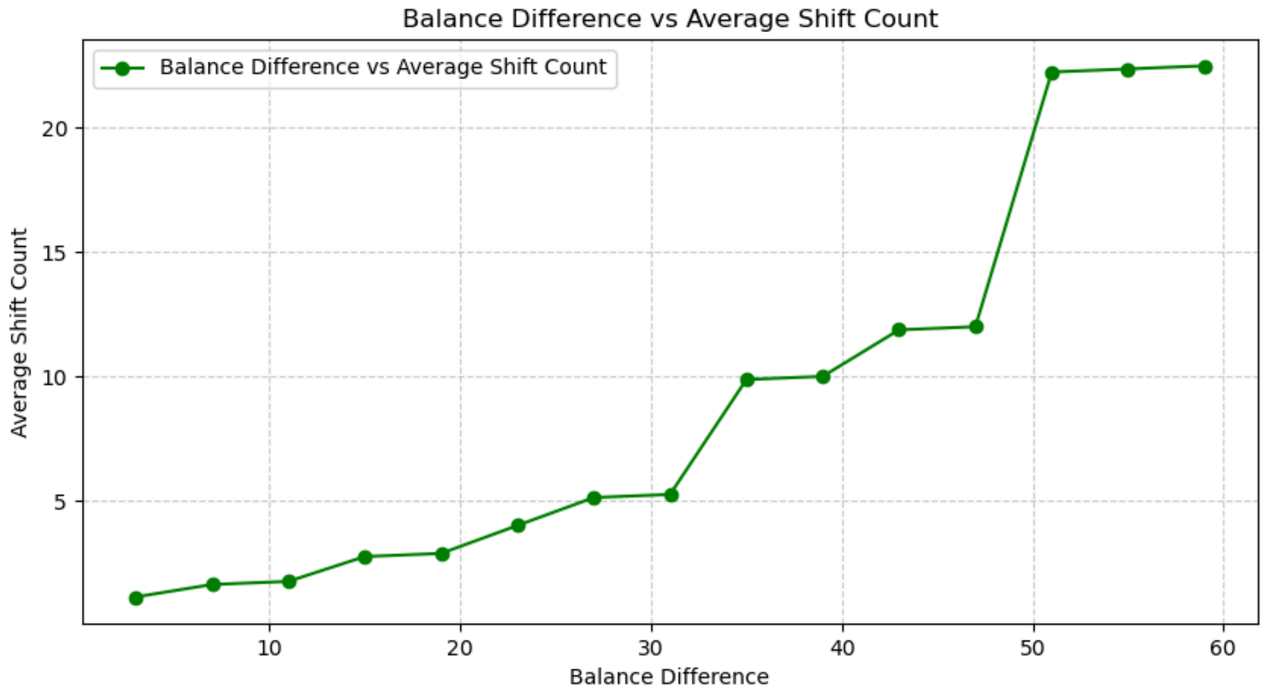

shift and lock it
GAINS TABLE
2
1
0
-1
-2
-3
I
E
B
G
D
C
J
A
H
F

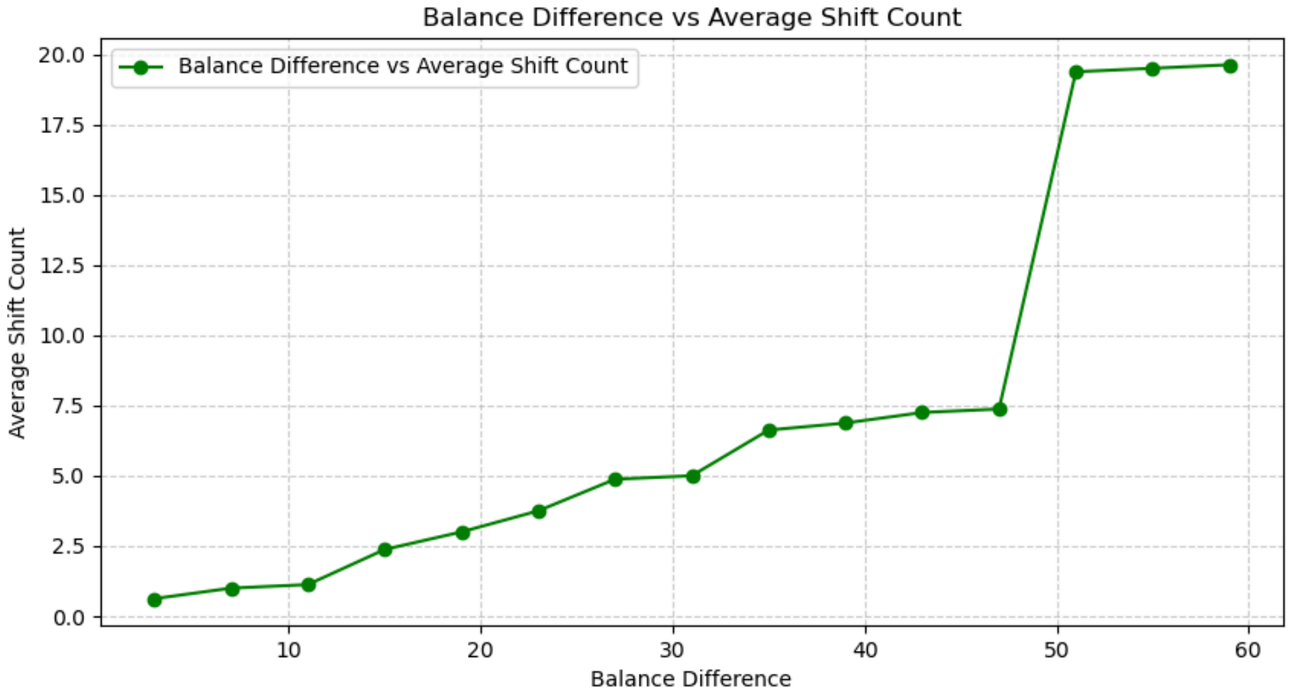

P1
P2
P3
A
C
E
F
B
D
G
H
I
J
CUT SIZE=4

WHAT IS THE EFFECT OF BALANCE-DIFFERENCE ON SHIFT COUNT?

Balance Difference: 2% - 60%

1. Small-Dense

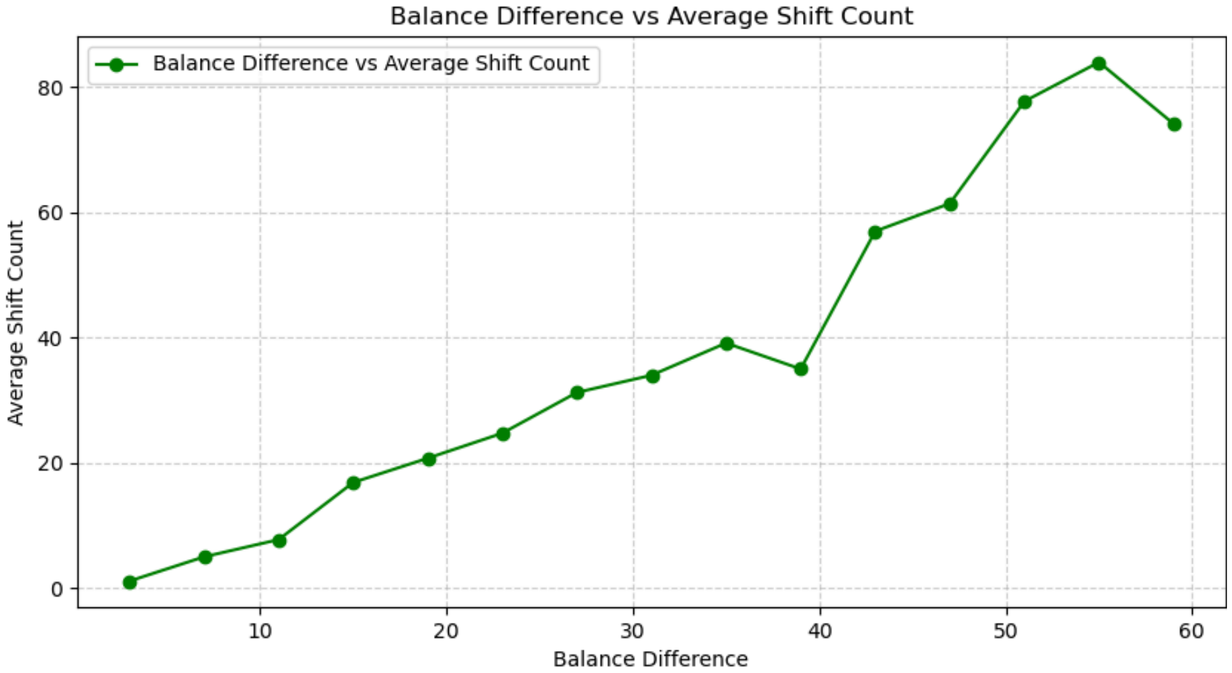




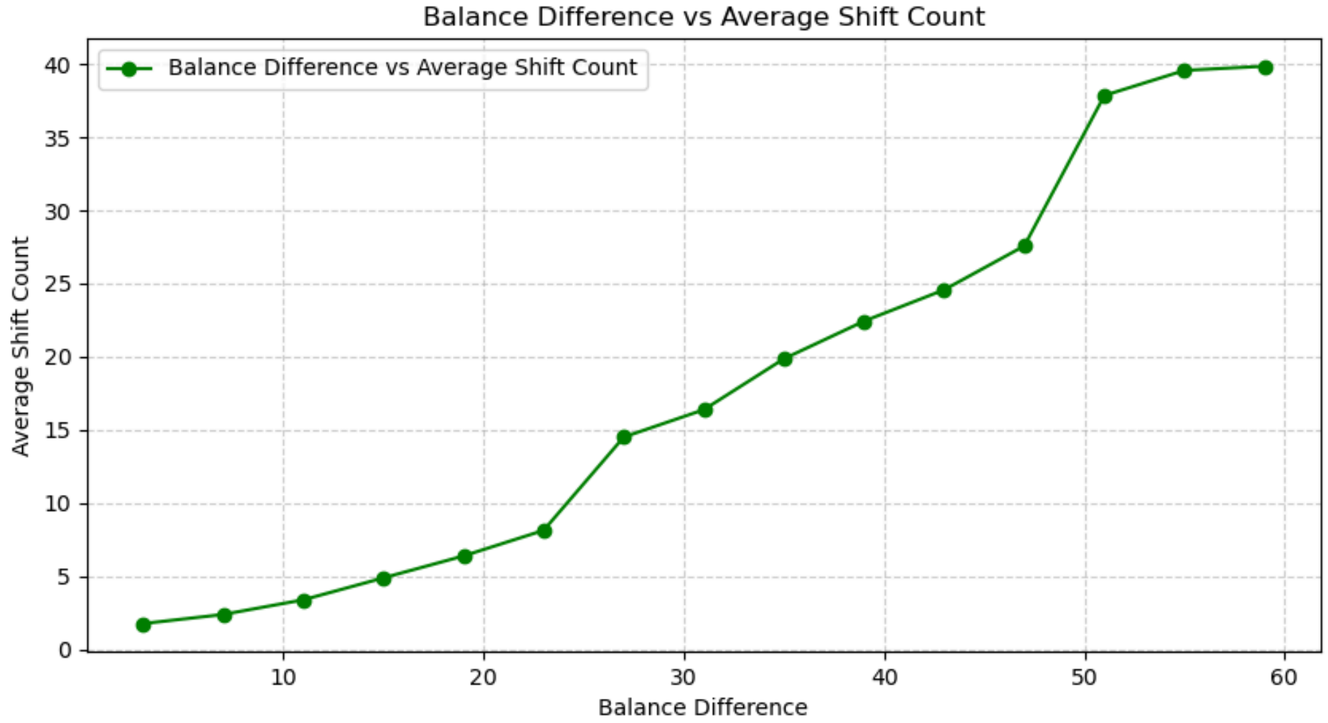
WHAT IS THE EFFECT OF BALANCE-DIFFERENCE ON SHIFT COUNT?

Balance Difference: 2% - 60%

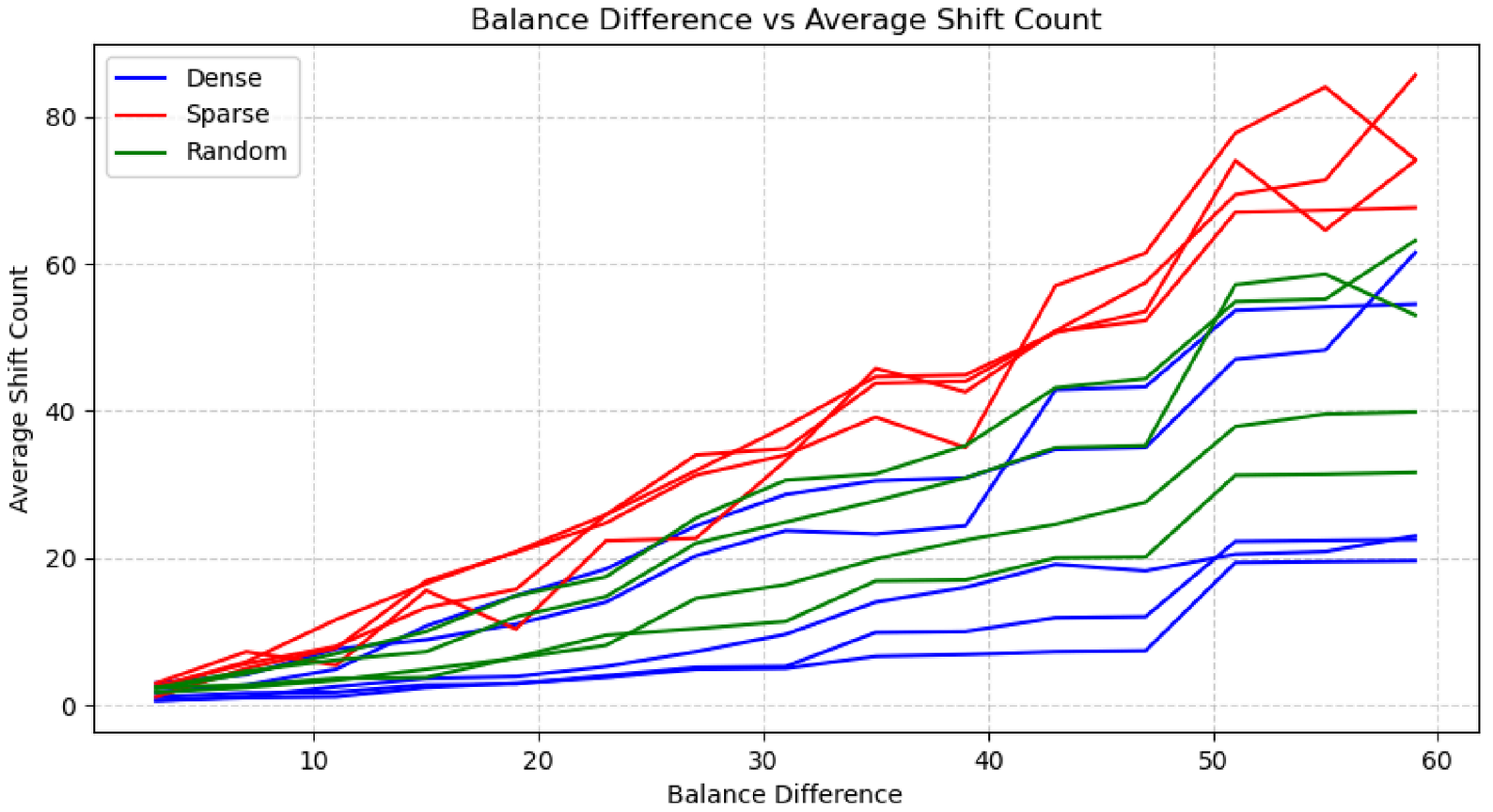
2. Small-Sparse



3. Small-Random

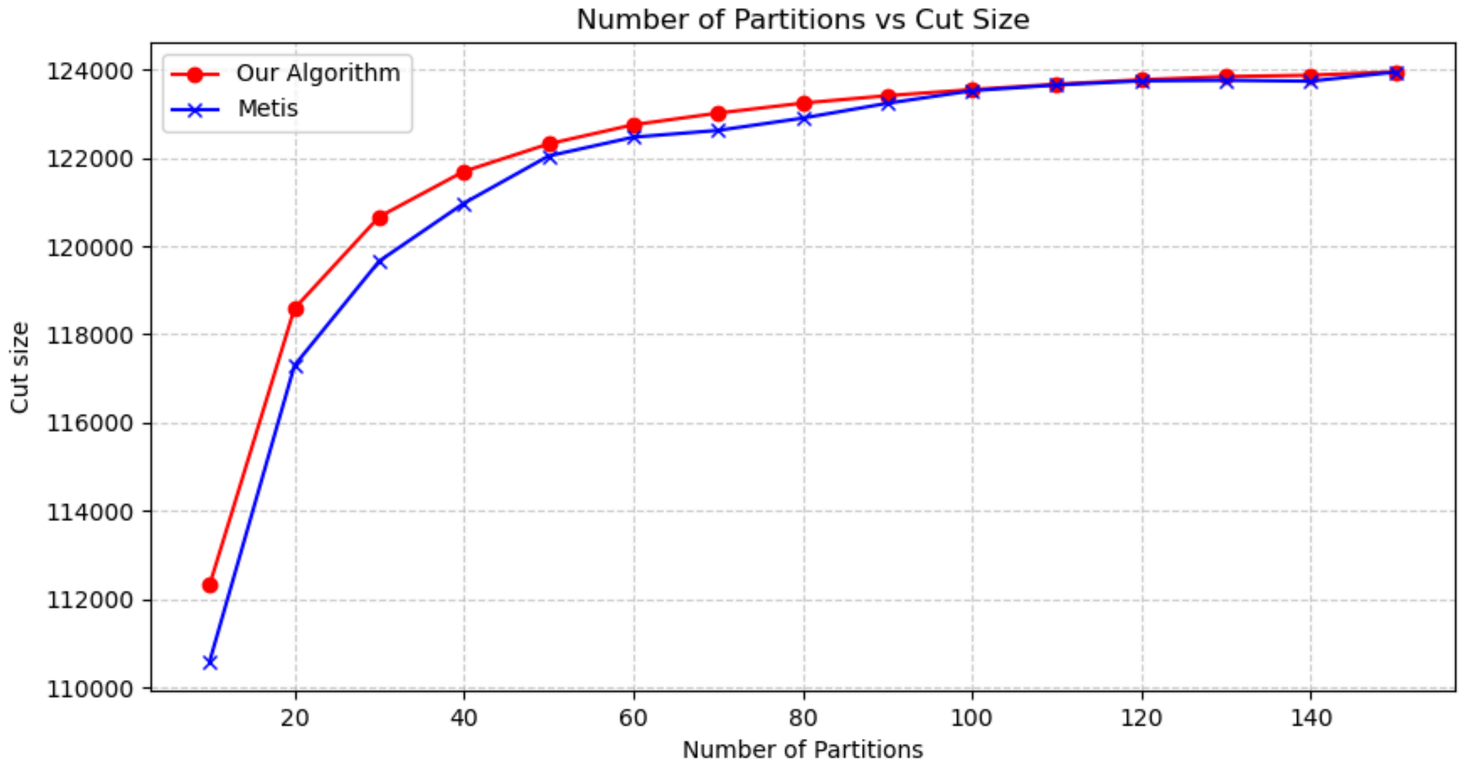


DENSE GRAPH HAS LESS SHIFTS!!!



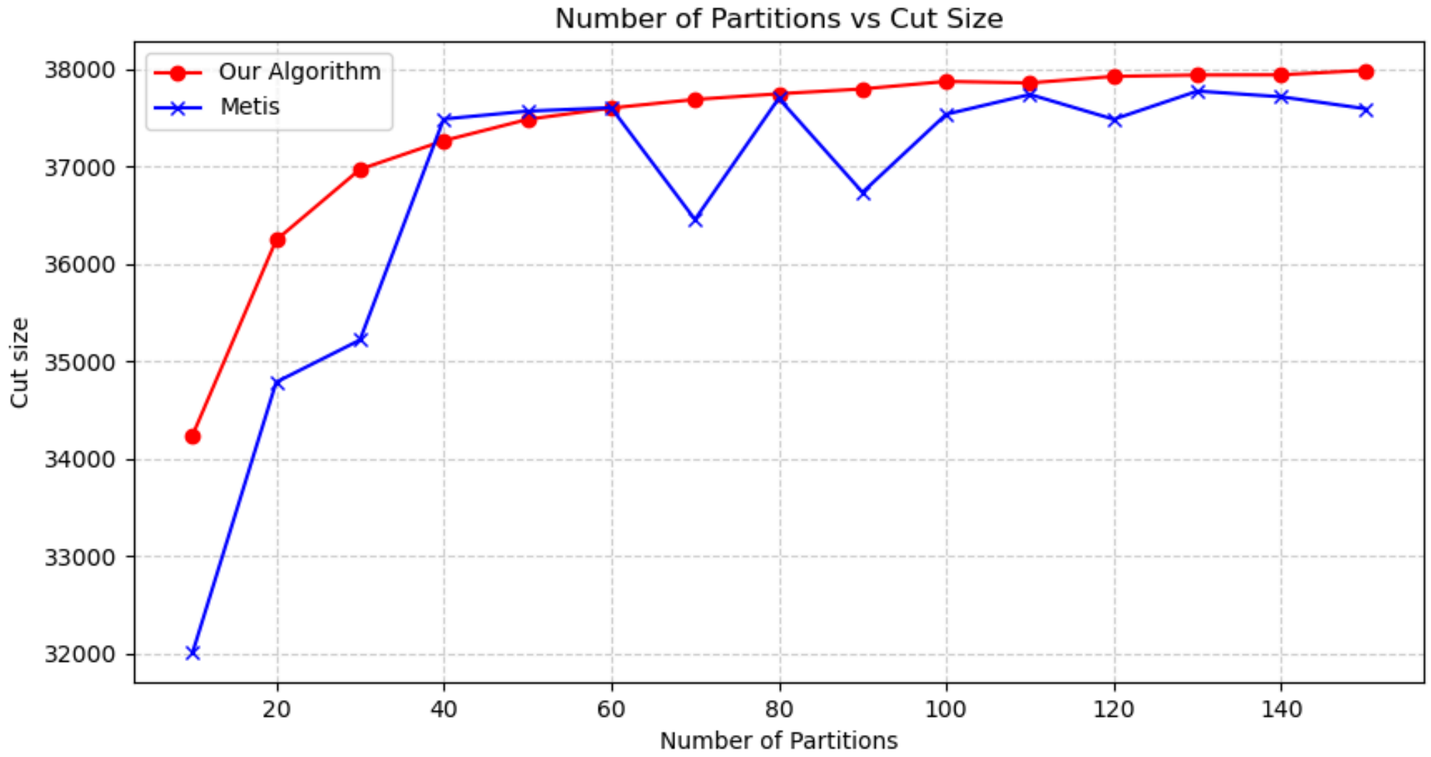
PERFORMANCE COMPARISON USING METIS

1. Small-Dense



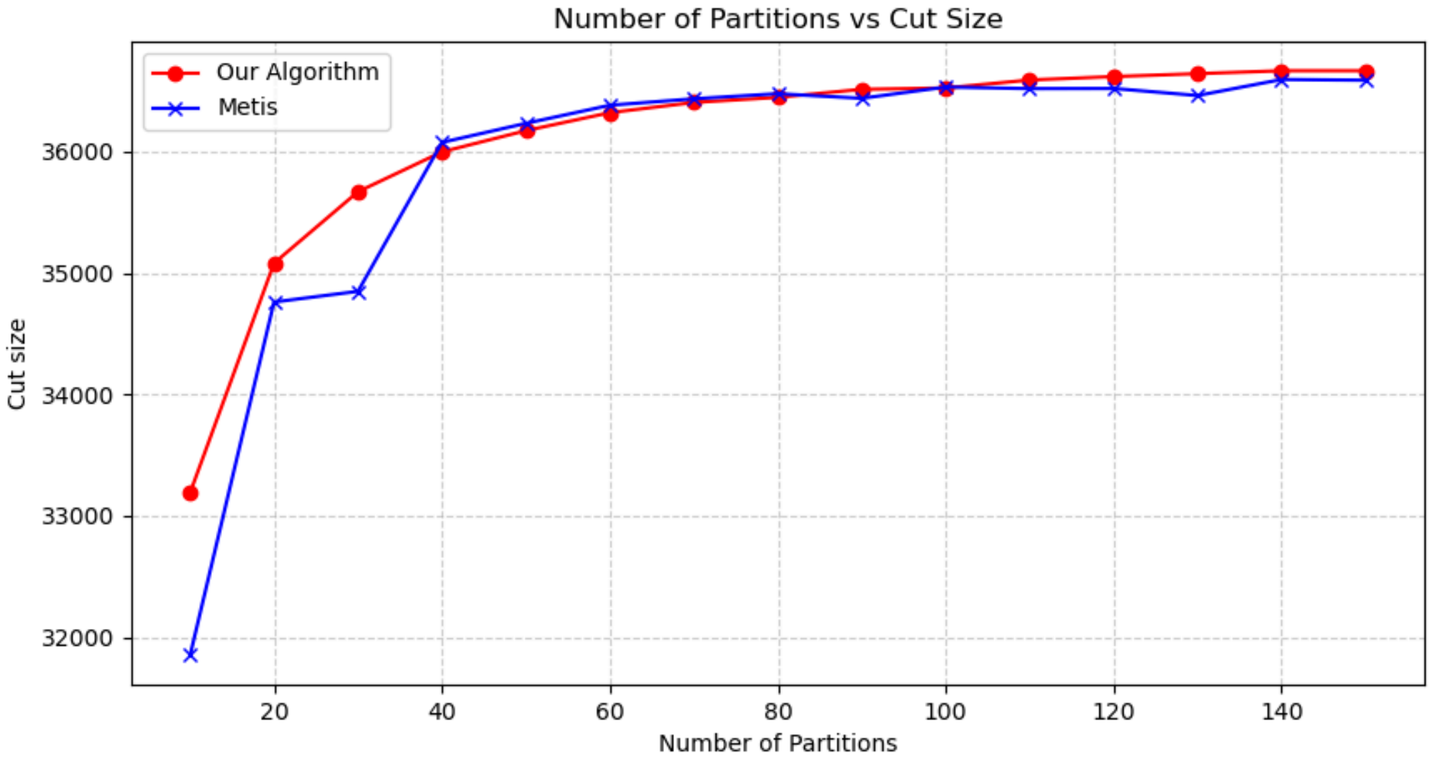
PERFORMANCE COMPARISON USING METIS

2. Small-Sparse



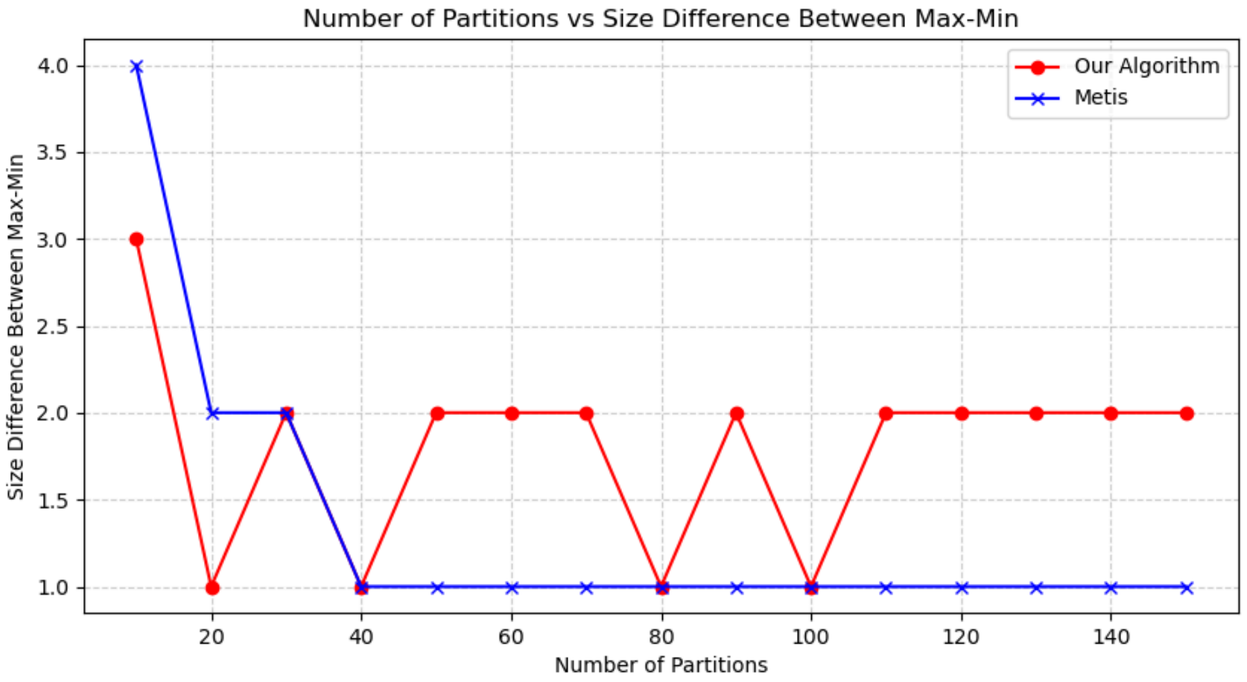
PERFORMANCE COMPARISON USING METIS

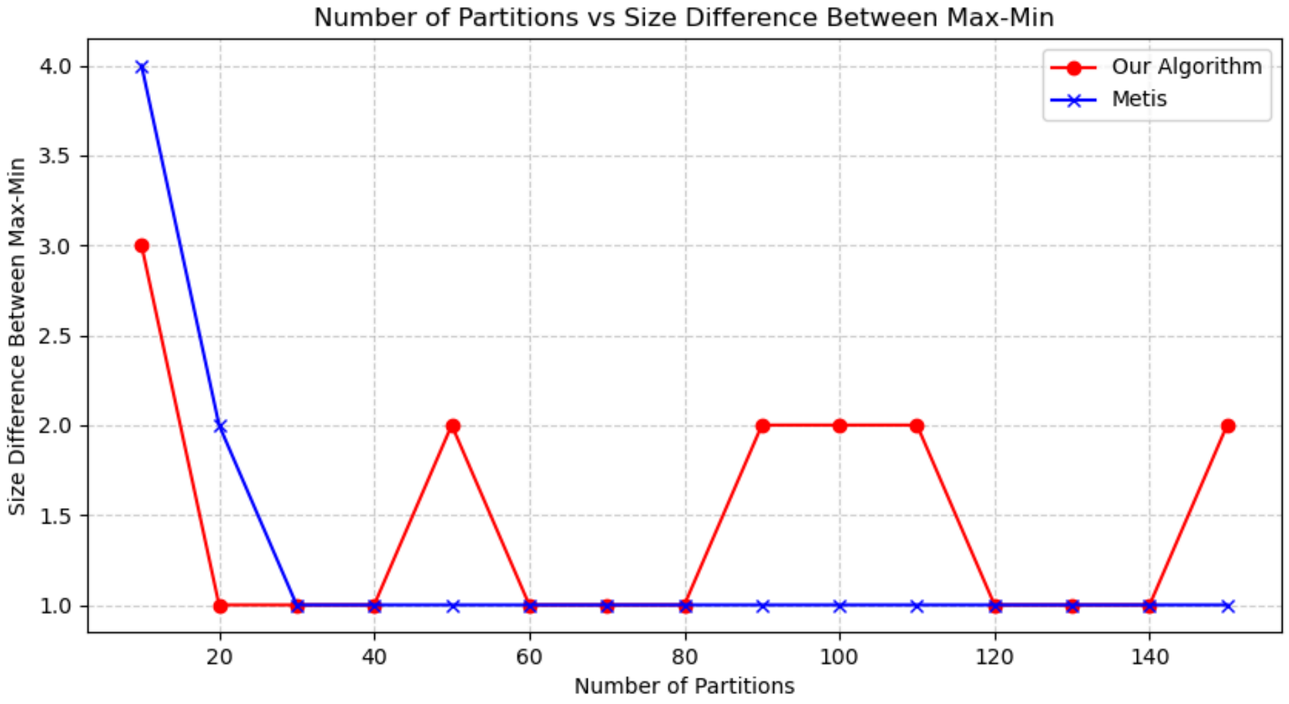
3. Small-Random



PERFORMANCE COMPARISON USING METIS

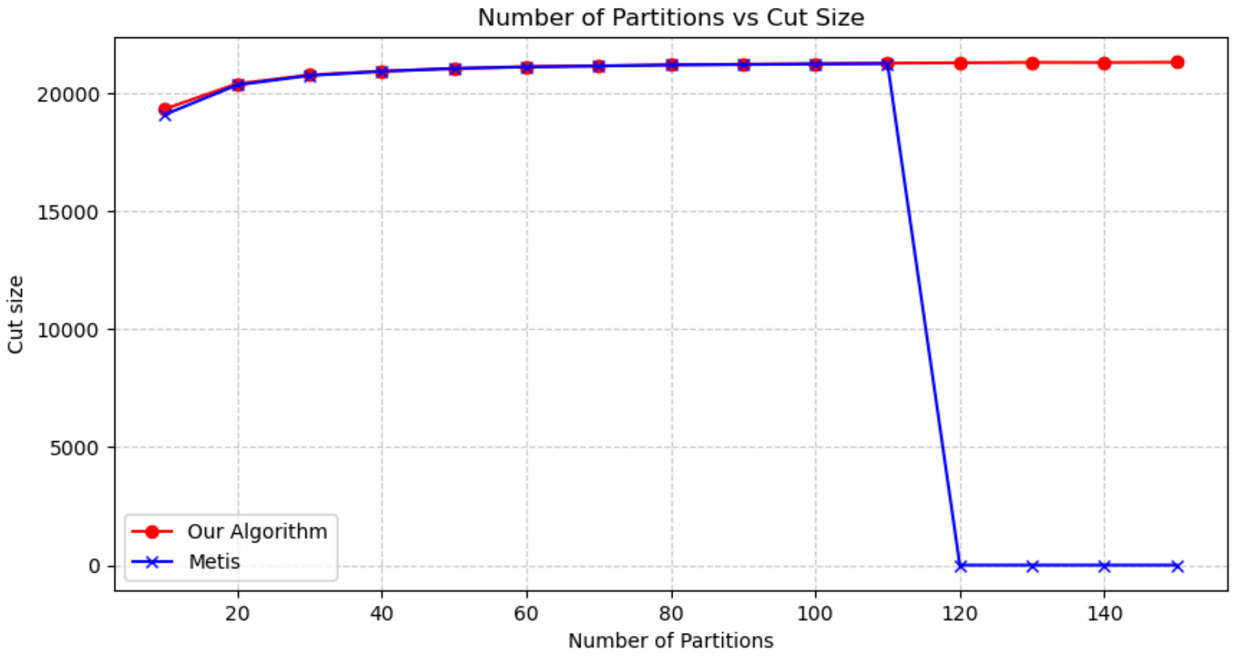
PartitionCount vs Size Difference Between Max And Min Partition:



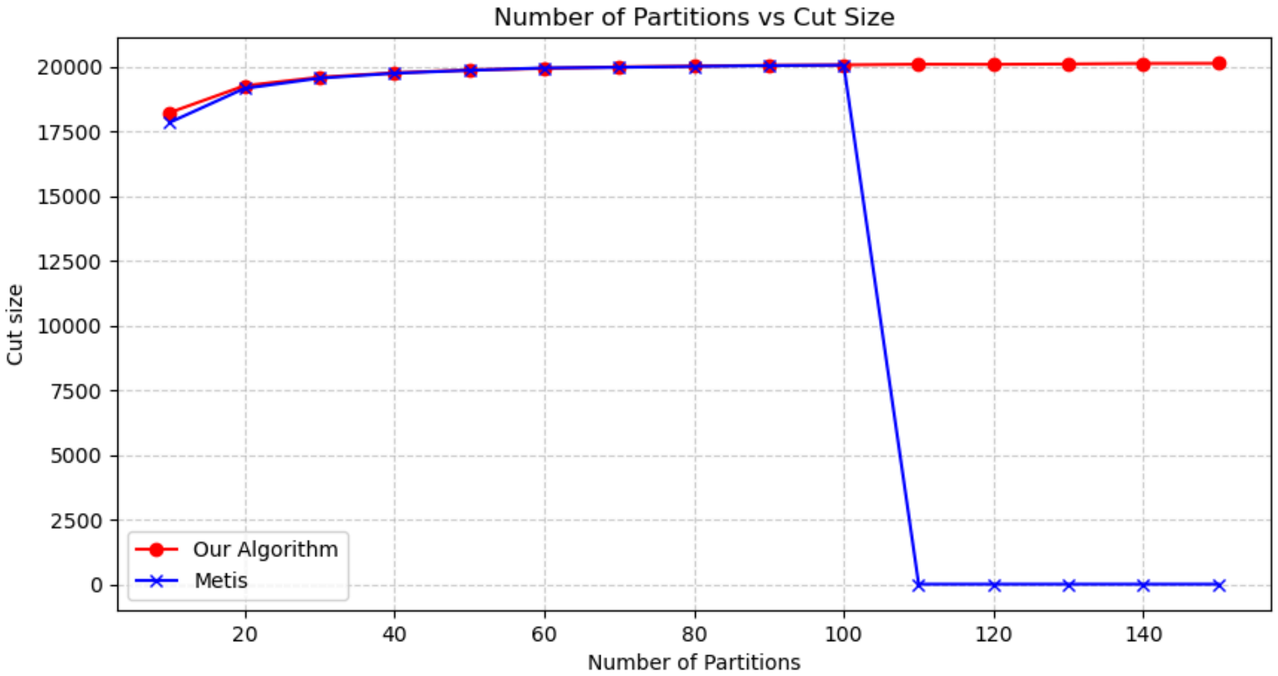


METIS FAILS SOMETIMES!

Nodes = 225
Edges= 20192

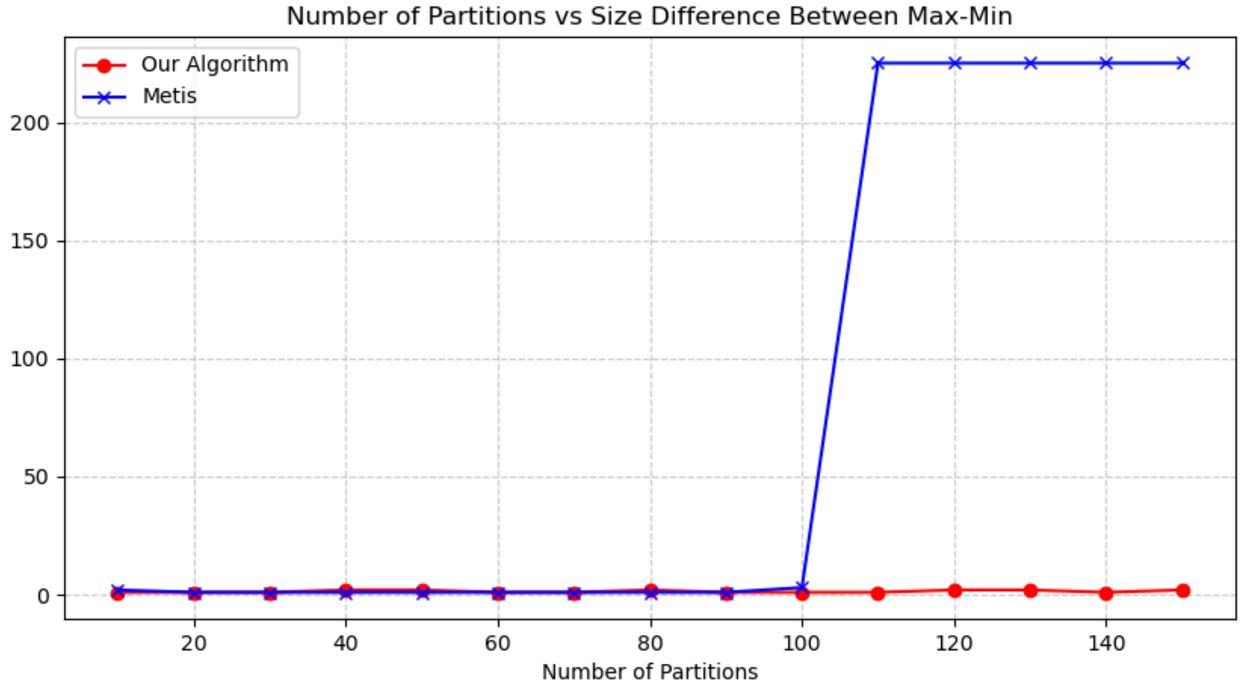


Nodes = 232
Edges= 21404

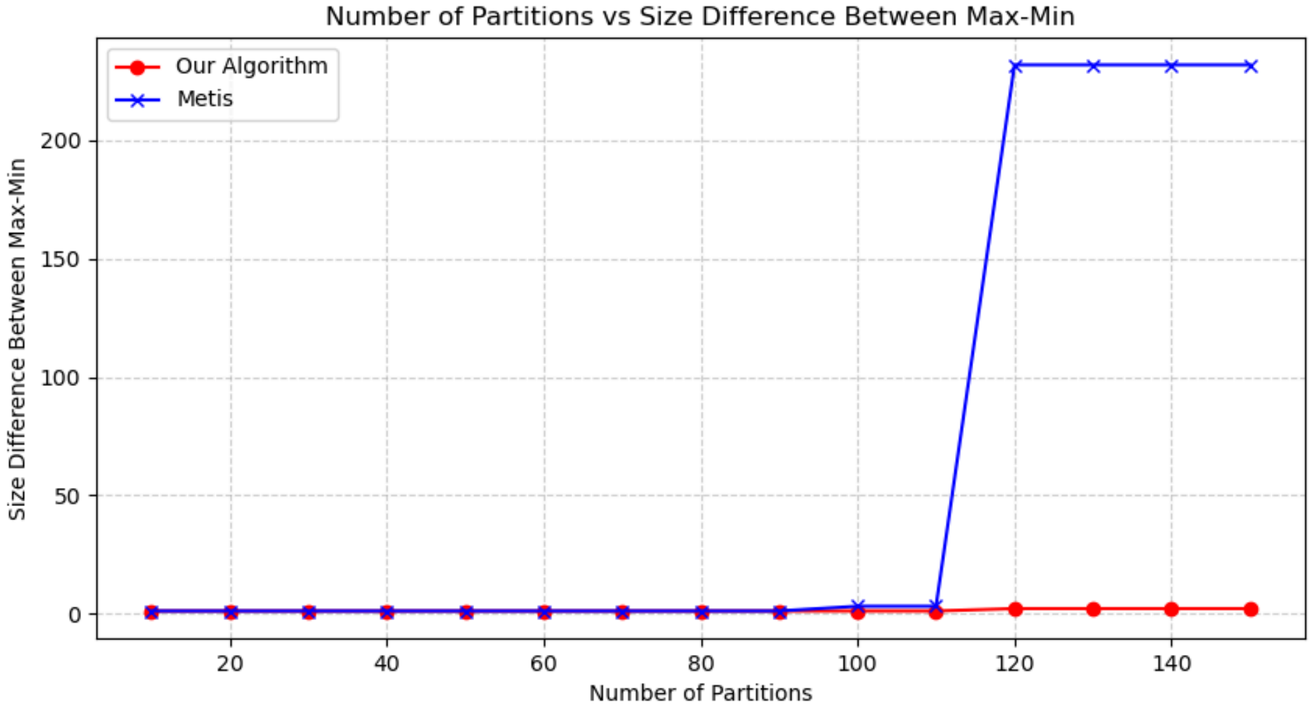


METIS FAILS SOMETIMES!

Nodes = 225
Edges= 20192



Nodes = 232
Edges= 21404



AGENDA 4

GENETIC ALGORITHM

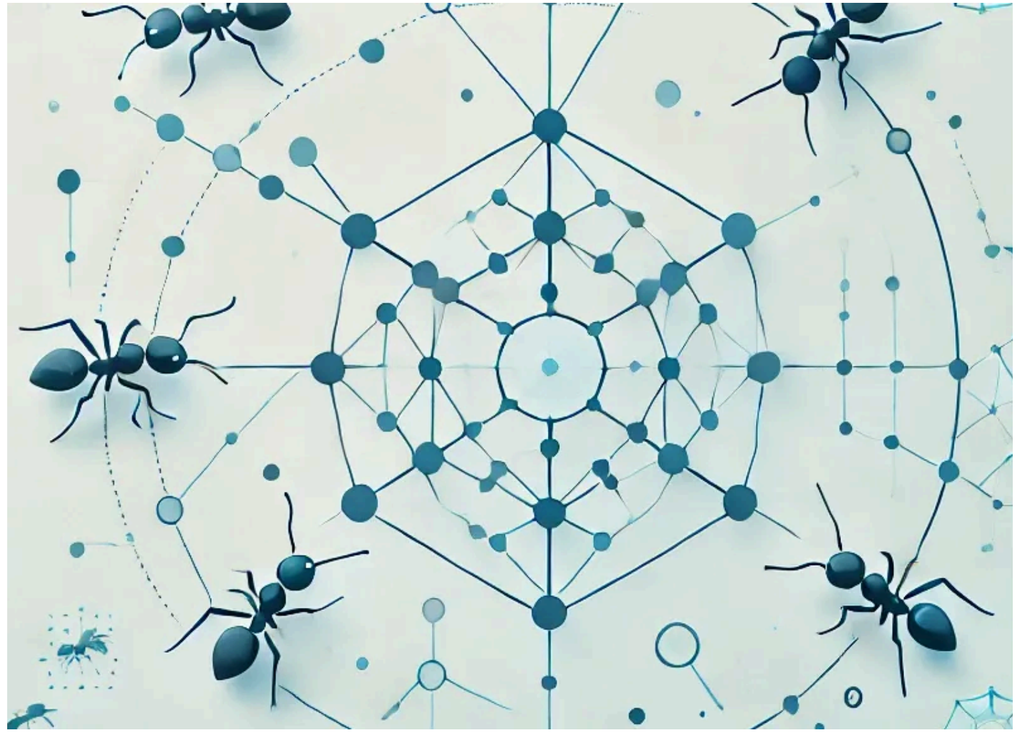
A METAHEURISTIC ALGORITHM

1905089

METAHEURISTIC ALGORITHM

INTRODUCTION

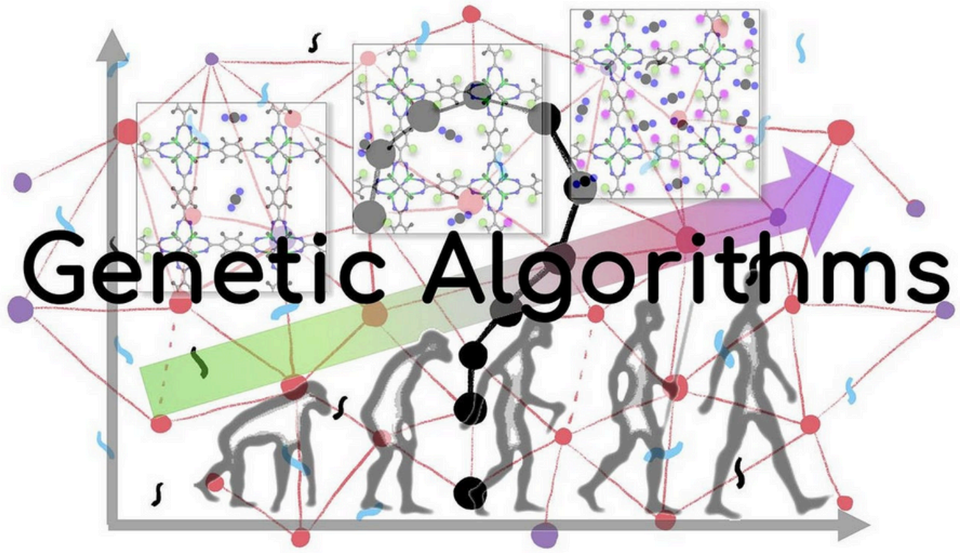
HIGH-LEVEL STRATEGIES FOR SOLVING COMPLEX OPTIMIZATION PROBLEMS BY EXPLORING AND EXPLOITING LARGE SEARCH SPACES EFFICIENTLY.



METAHEURISTIC ALGORITHM

GENETIC ALGORITHM

A SEARCH HEURISTIC INSPIRED BY THE PROCESS OF NATURAL SELECTION THAT IS USED TO GENERATE HIGH-QUALITY SOLUTIONS TO OPTIMIZATION AND SEARCH PROBLEMS.



GENETIC ALGORITHM

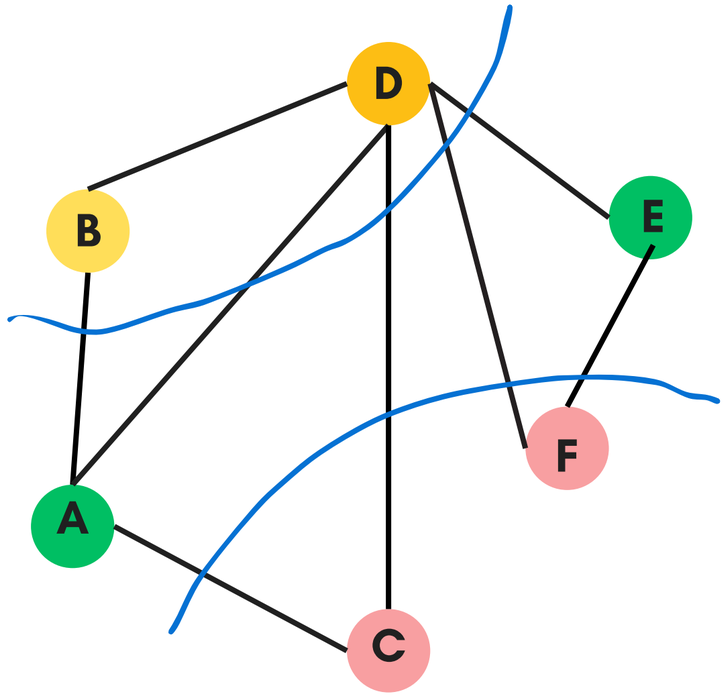
1. INITIALIZATION

START WITH A POPULATION OF POTENTIAL SOLUTIONS, EACH REPRESENTING A POSSIBLE WAY TO PARTITION THE GRAPH.

Freckle Face

VERTEX ENCODING:

- A GENE CORRESPONDS TO A VERTEX.
- IT ASSIGNS EACH GENE A SPECIFIED NUMBER 0 TO K DEPENDING ON WHICH PARTITION THE GENE BELONGS TO.



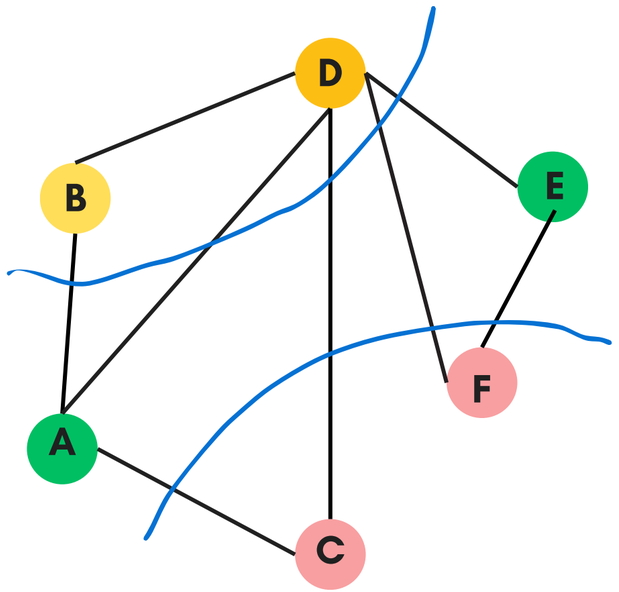
RANDOM PARTITION:
{A,E}, {C,F},{B,D}

VERTEX ENCODING HERE IS : 132312

GENETIC ALGORITHM

2. FITNESS FUNCTION

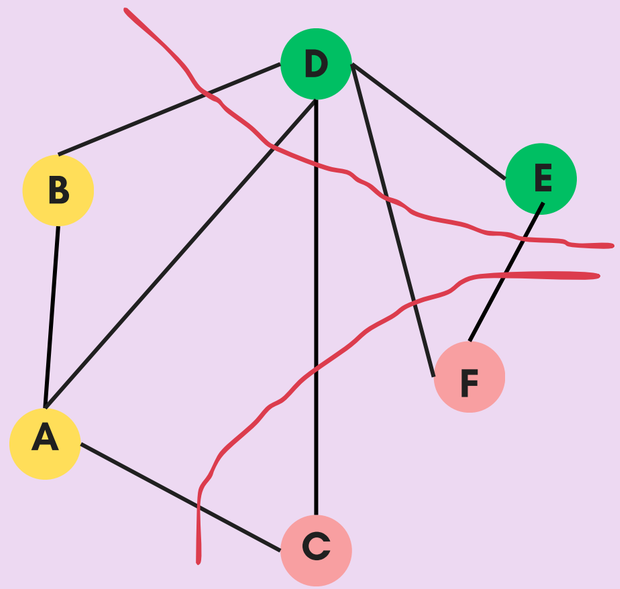
MORE CUT EDGES



RANDOM PARTITION: {A,E}, {C,F},{B,D}

VERTEX ENCODING : 132312

LESS CUT EDGES



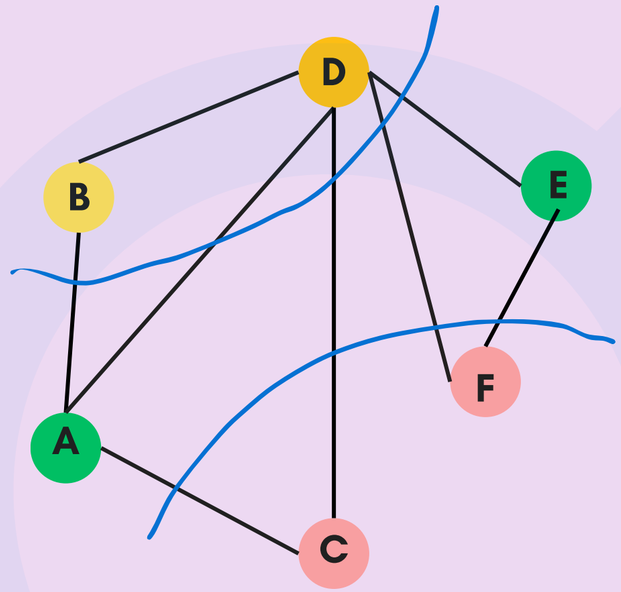
RANDOM PARTITION:{A,B}, {C,F},{E,D}

VERTEX ENCODING : 112332

GENETIC ALGORITHM

3. PARENT SELECTION & THEIR ENCODINGS

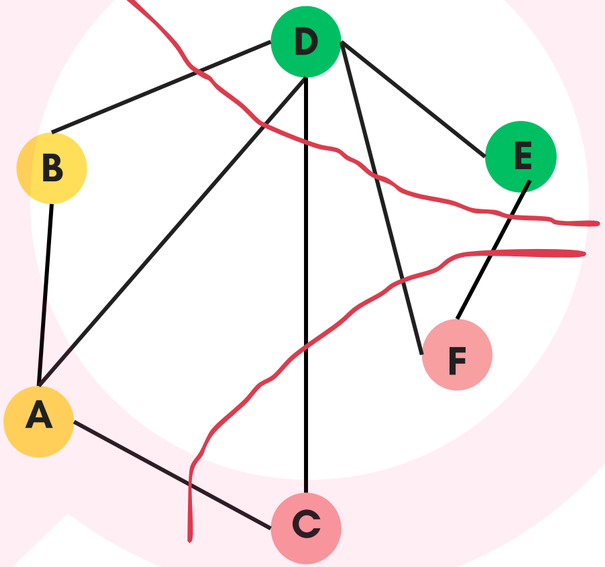
PARENT 1:



RANDOM PARTITION: {A,E}, {C,F},{B,D}

VERTEX ENCODING : 132312

PARENT 2:



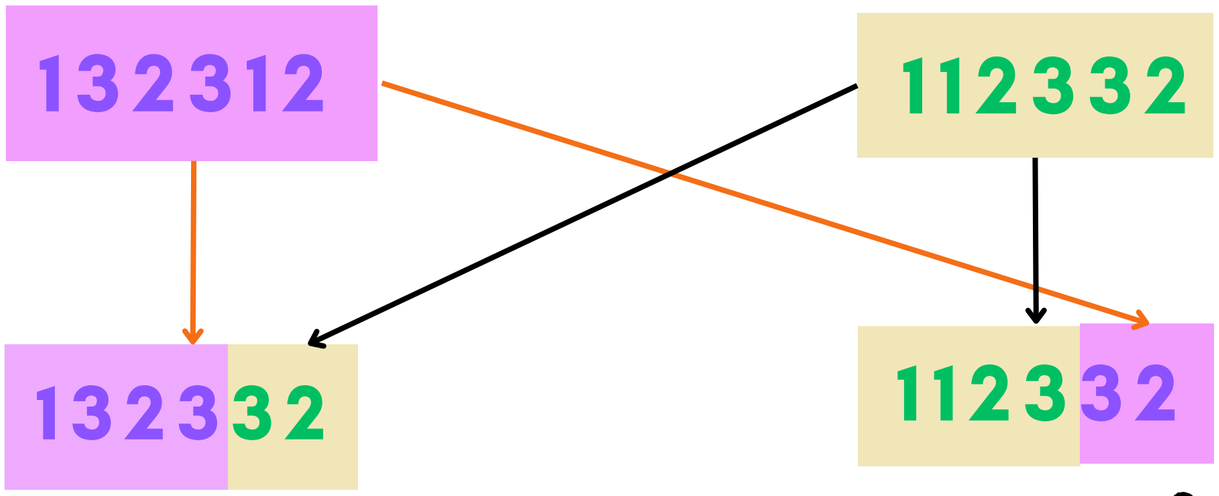
RANDOM PARTITION:{A,B}, {C,F},{E,D}

VERTEX ENCODING : 112332

GENETIC ALGORITHM

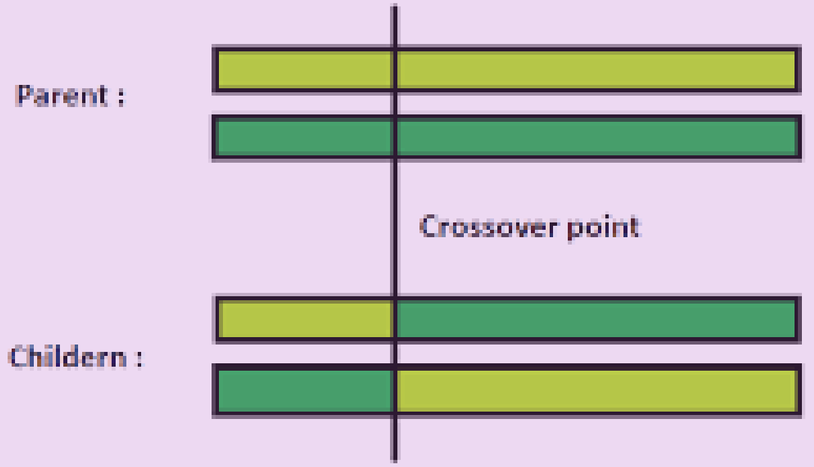
4. CROSSOVER

CROSSOVER COMBINES TWO PARENT SOLUTIONS TO CREATE NEW OFFSPRING WITH MIXED TRAITS



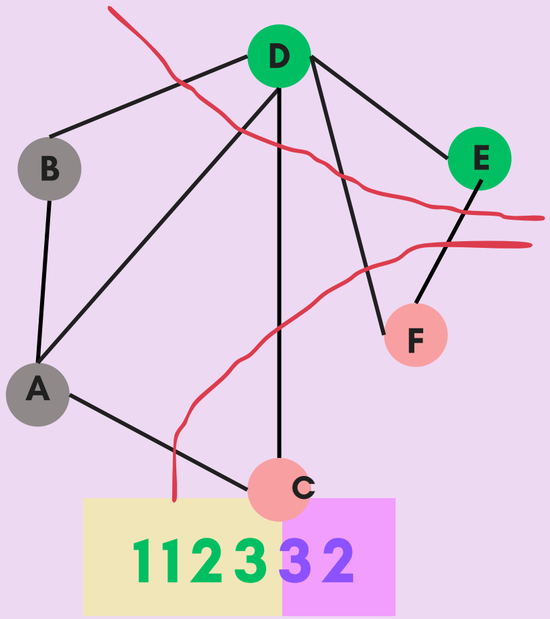
UNBALANCED PARTITION!!!

BALANCED PARTITION!!!

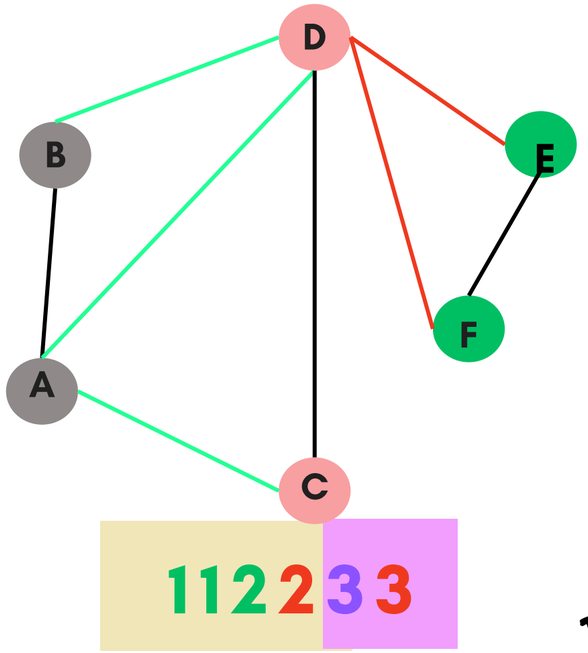


GENETIC ALGORITHM

5.MUTATION



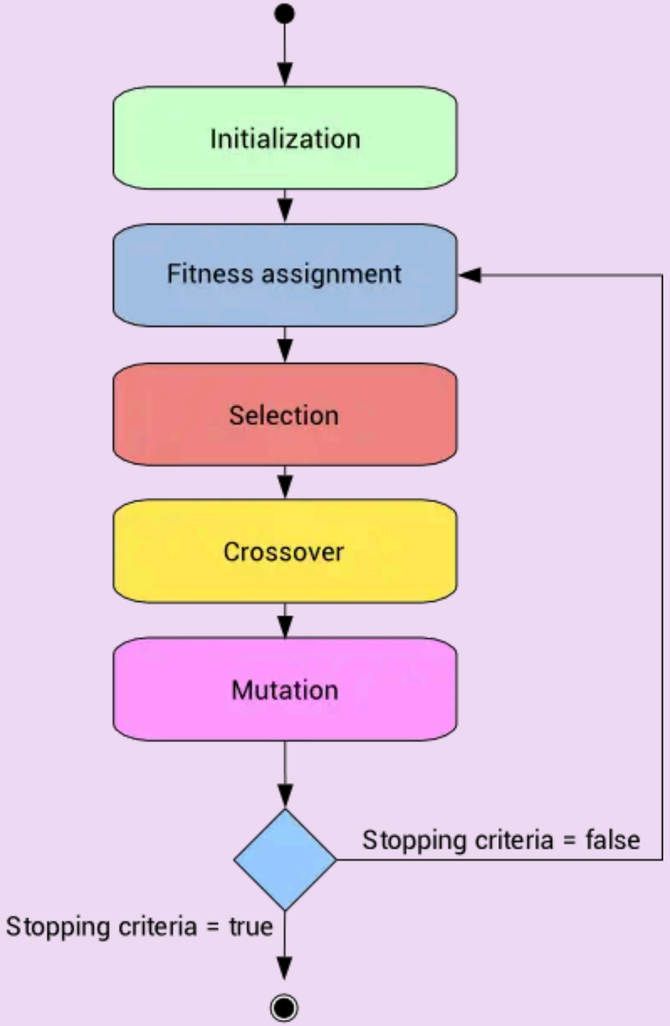
HIGH PENALTY



PENALTY DECREACED

GENETIC ALGORITHM

4.TERMINATION



Time Complexity:

O(generations * population_size * (|V| + |E|))

Space Complexity:

O(population_size * num_nodes)

DATASET

GENETIC ALGORITHM

DATASET DIVISIONS

Type	Edge-Probability
Dense Graph	0.8
Sparse Graph	0.2
Random	0.5

[1]

1. Edge probability means probability of existence of an edge between 2 specific vertices

GENETIC ALGORITHM

DATASET DESCRIPTION

Type	Node	Edge
Sparse	591	34492
	573	32676
Dense	303	36641
	559	124571
Random	493	60711
	384	36839

IMPLEMENTATION DETAILS

GENETIC ALGORITHM

BASICS OF IMPLEMENTATION

LANGUAGE: PYTHON
GRAPHS & CHARTS: PYTHON
GITHUB LINK: Genetic Algorithm Implementation

❋ PSEUDOCODE ❋

```
Initialize population with random partitions

For each generation:
    Select parents from the population
    Create children through crossover
    Mutate some children
    Calculate fitness for each child
    Replace old population with new population

Evaluate and return the best partition
```

GENETIC ALGORITHM

FUNCTION DETAILS

FUNCTION	DETAILS
calculate_fitness	Computes the fitness of a partition based on cut edges and partition balance.
random_partition	Creates a new partition with random node assignments.
crossover	Combines two parent partitions to create a child partition using a random crossover point.
mutate	Alters a partition by randomly changing one node’s partition assignment.
selection	Selects the fitter partition between two randomly chosen individuals from the population.
genetic_algorithm	Runs the main genetic algorithm process (initializes population, performs selection, crossover, mutation, and evaluates fitness).

AGENDA 5

GENETIC ALGORITHM RESULTS AND FINAL COMPARISONS

1905109

GENETIC ALGORITHM

GRAPH PARAMETERS

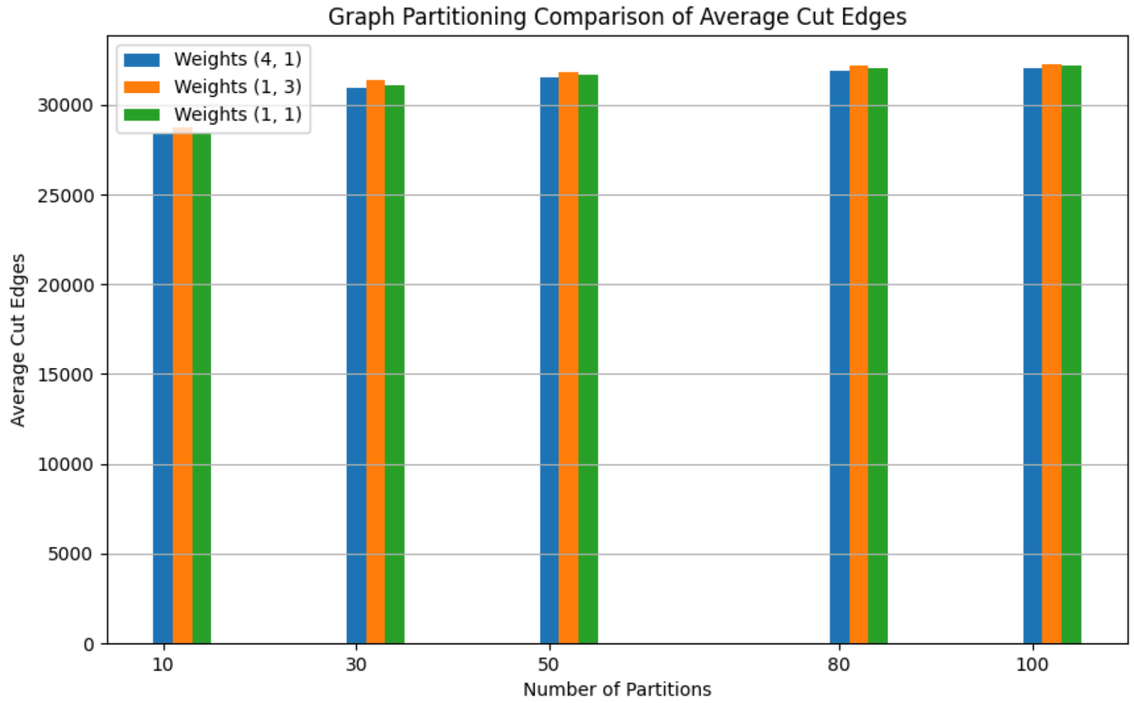
PARTITION SIZE -> [10, 30, 50, 80, 100]

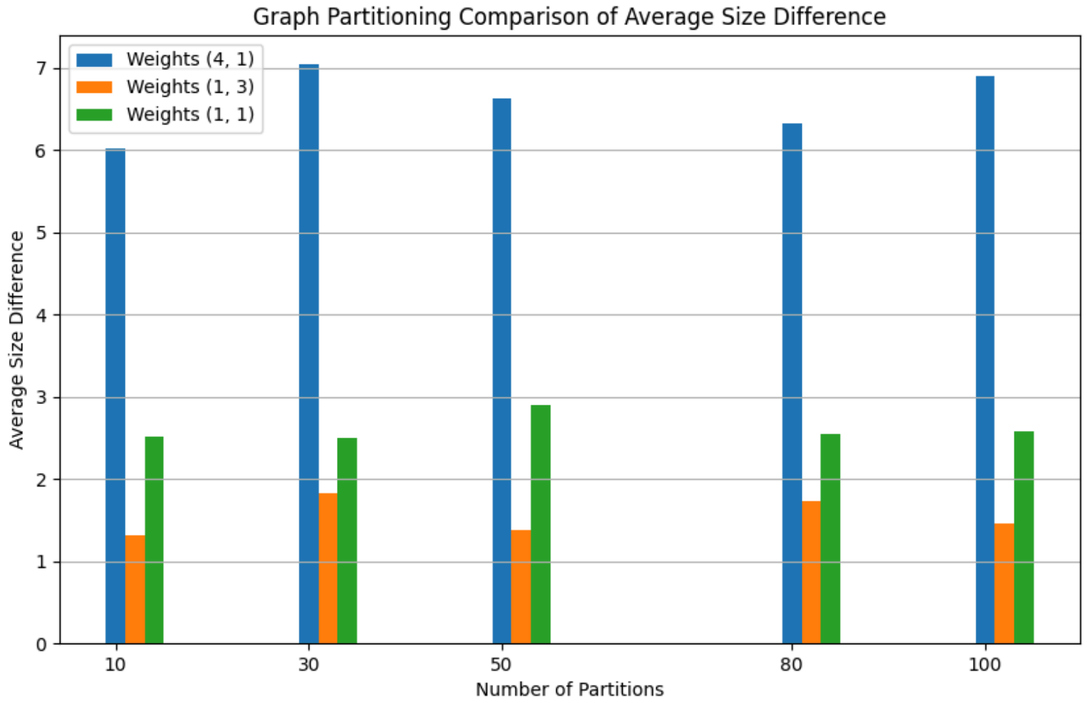
FITNESS FUNCTION = CUT_WEIGHT*CUT_EDGES + BALANCE_WEIGHT*BALANCE_PENALTY

CUT_WEIGHT	BALANCE_WEIGHT
4	1
1	3
1	1

GENETIC ALGORITHM

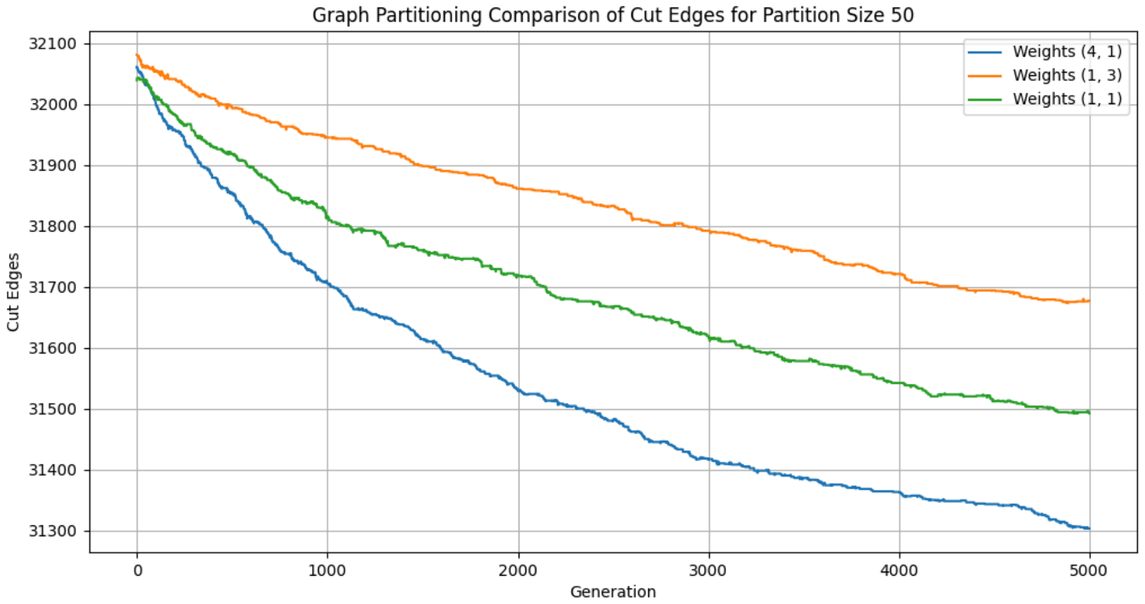
SPARSE GRAPH

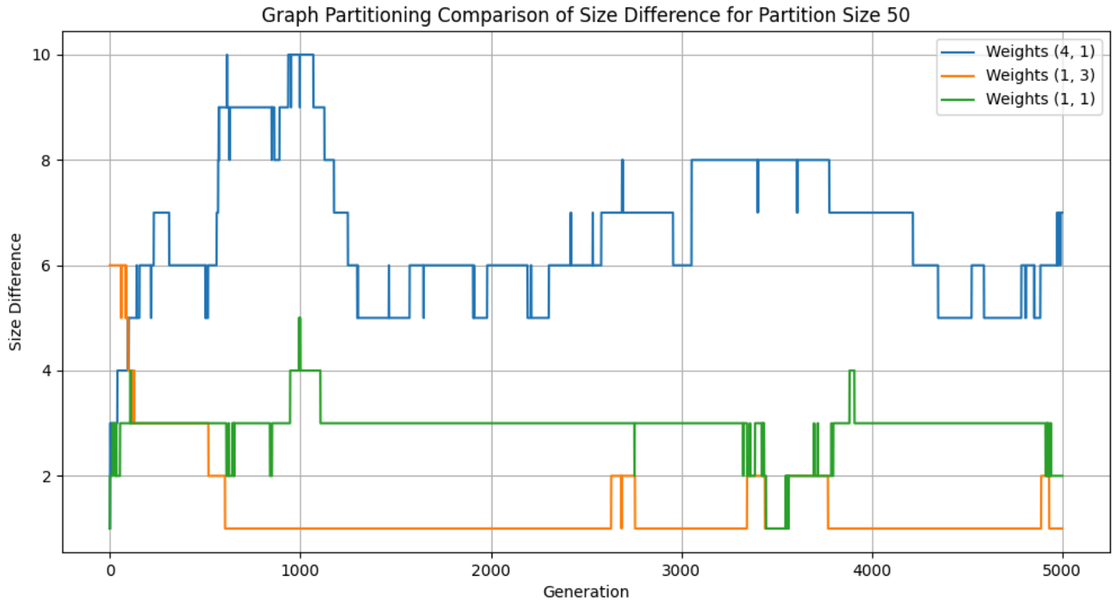




GENETIC ALGORITHM

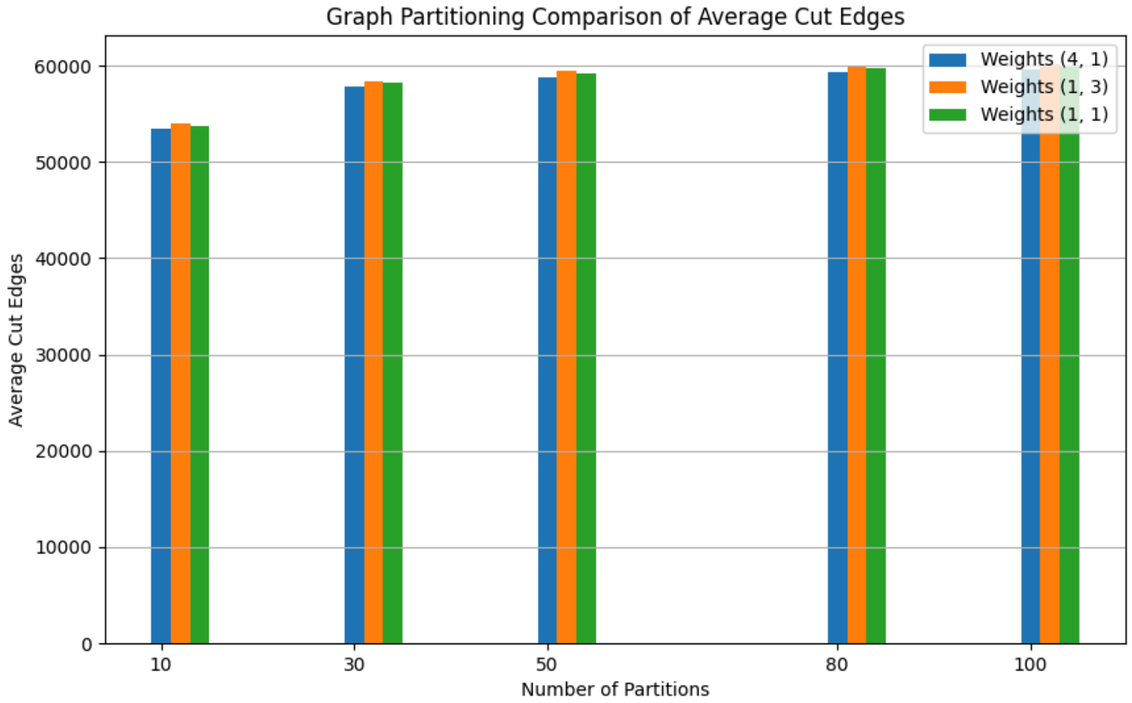
SPARSE GRAPH

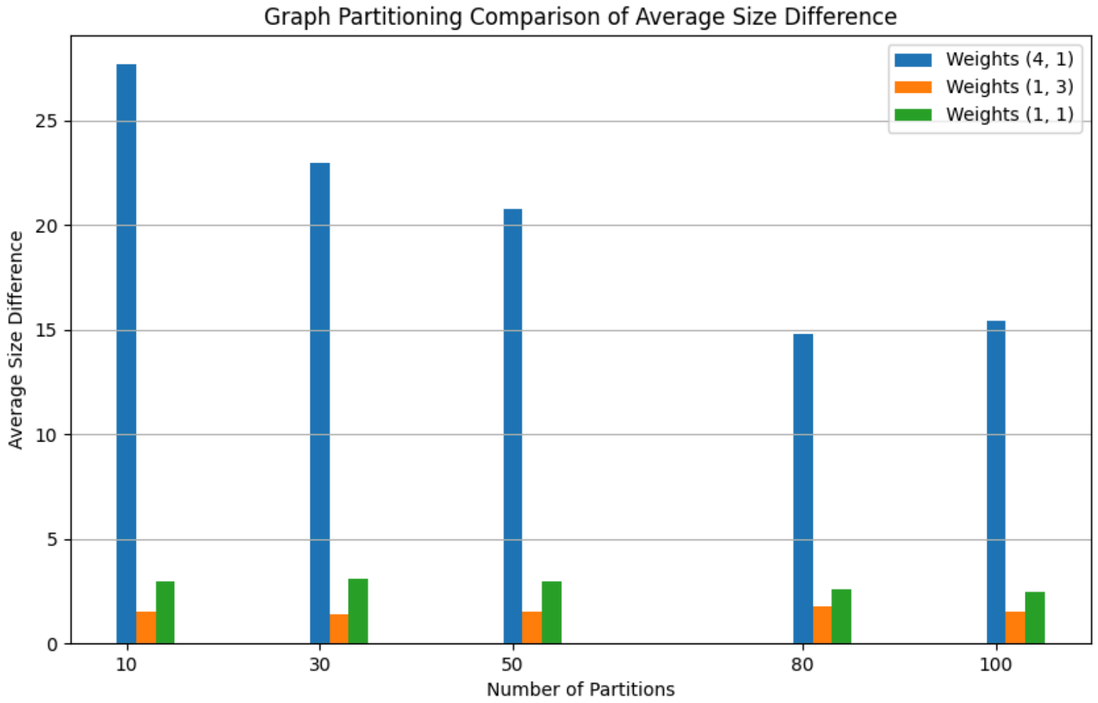




GENETIC ALGORITHM

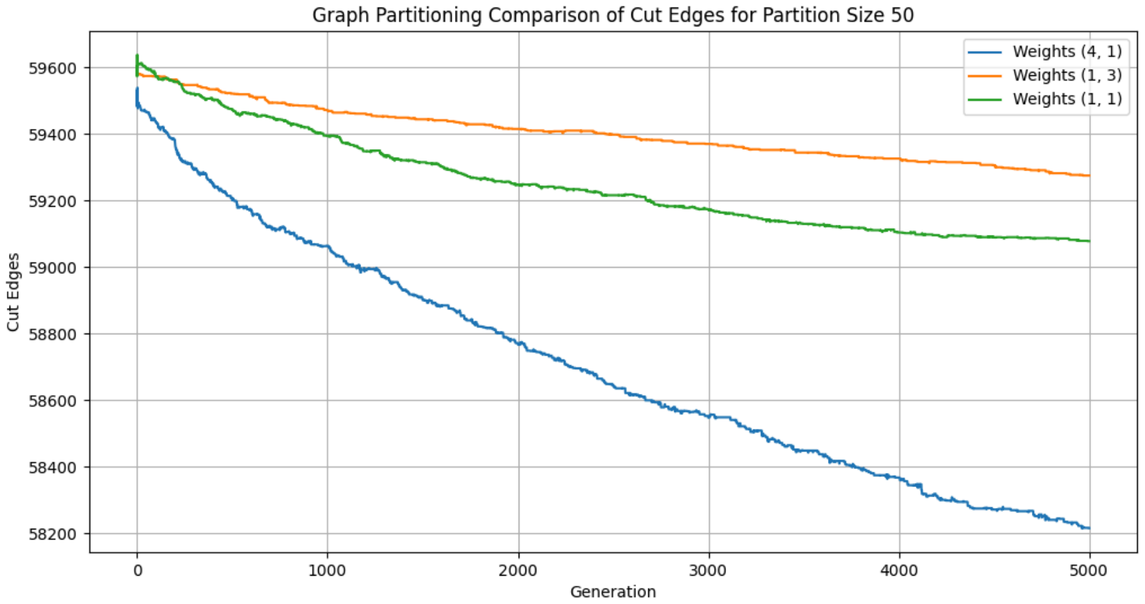
RANDOM GRAPH

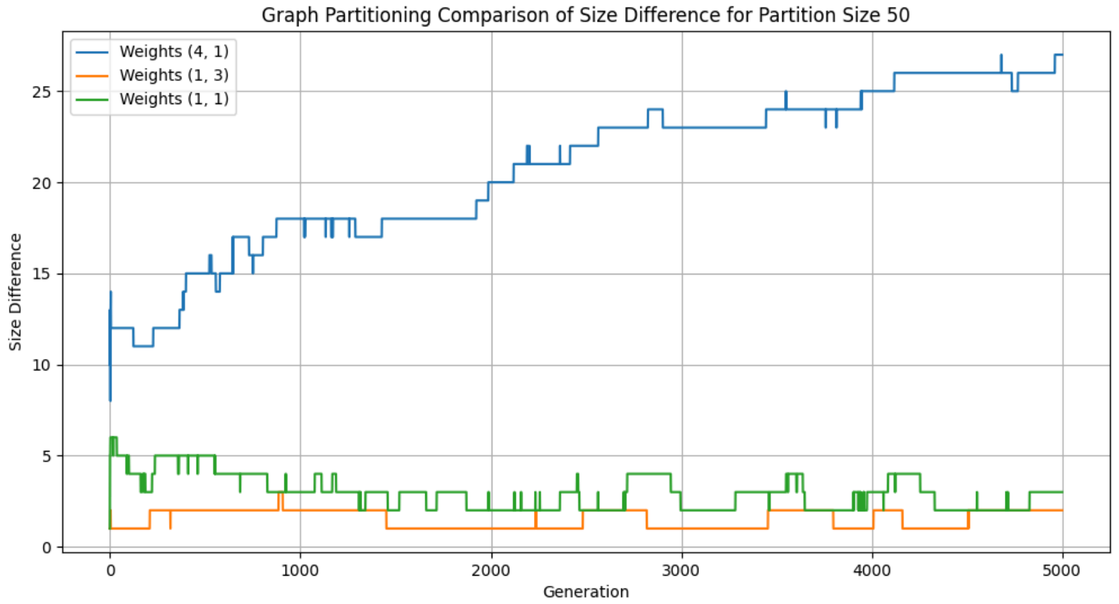




GENETIC ALGORITHM

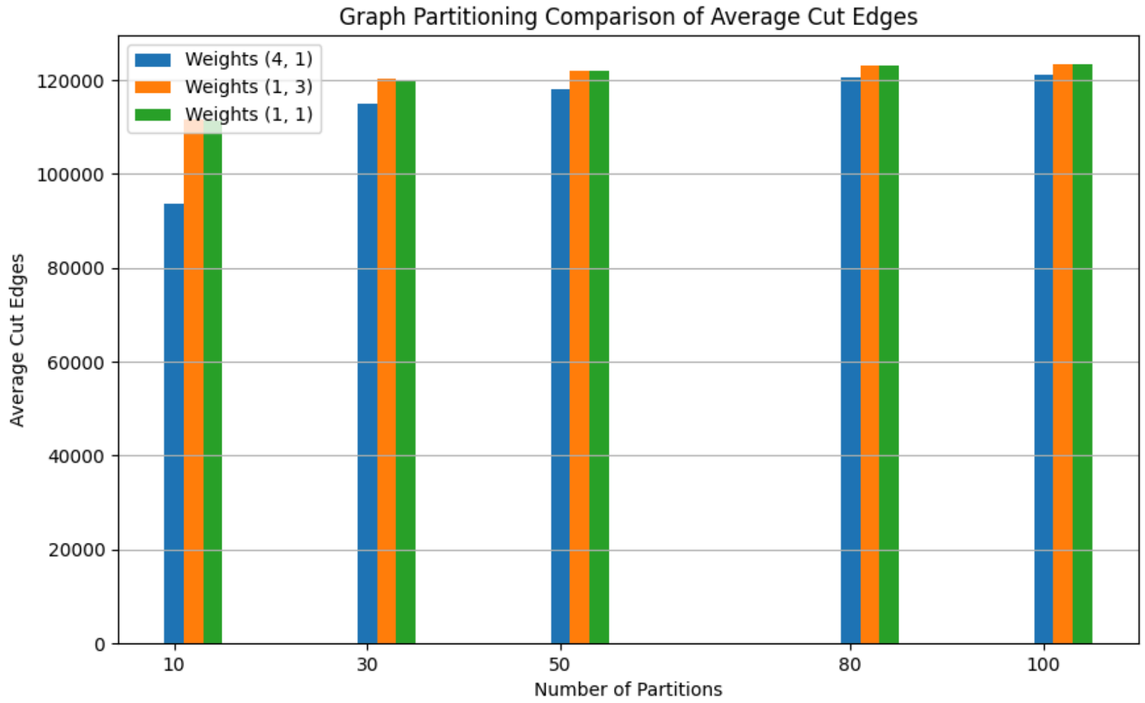
RANDOM GRAPH

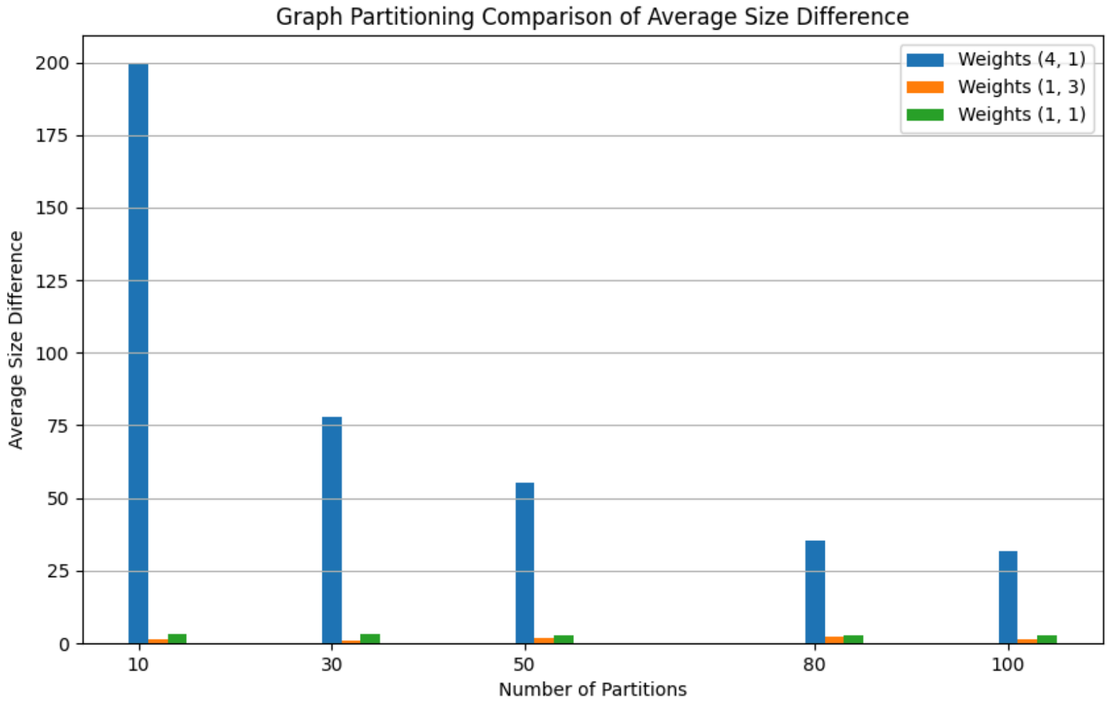




GENETIC ALGORITHM

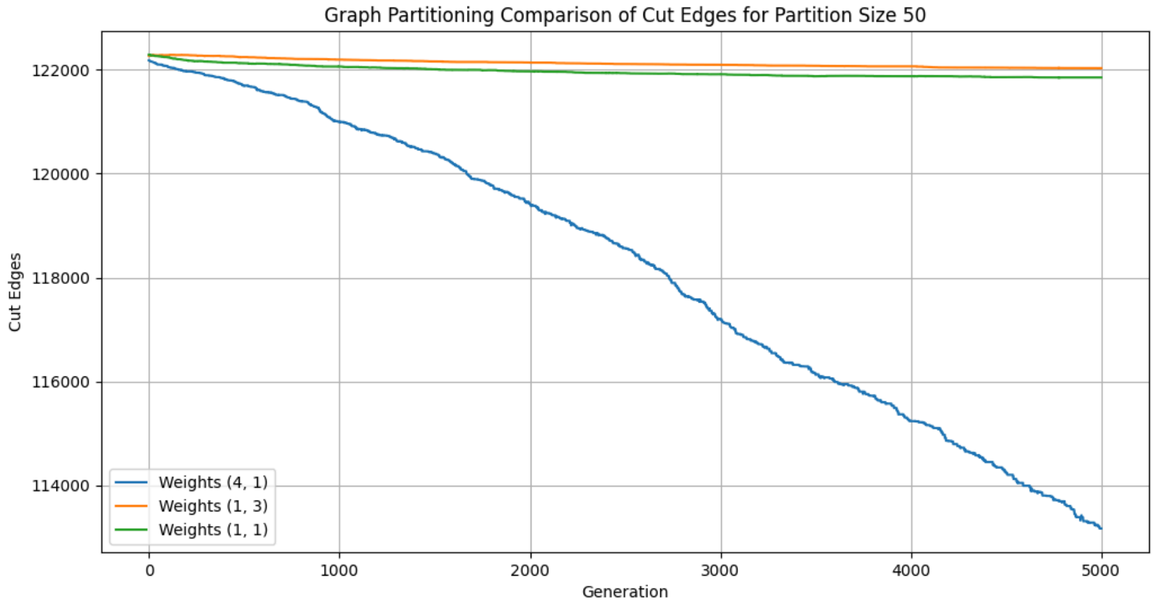
DENSE GRAPH

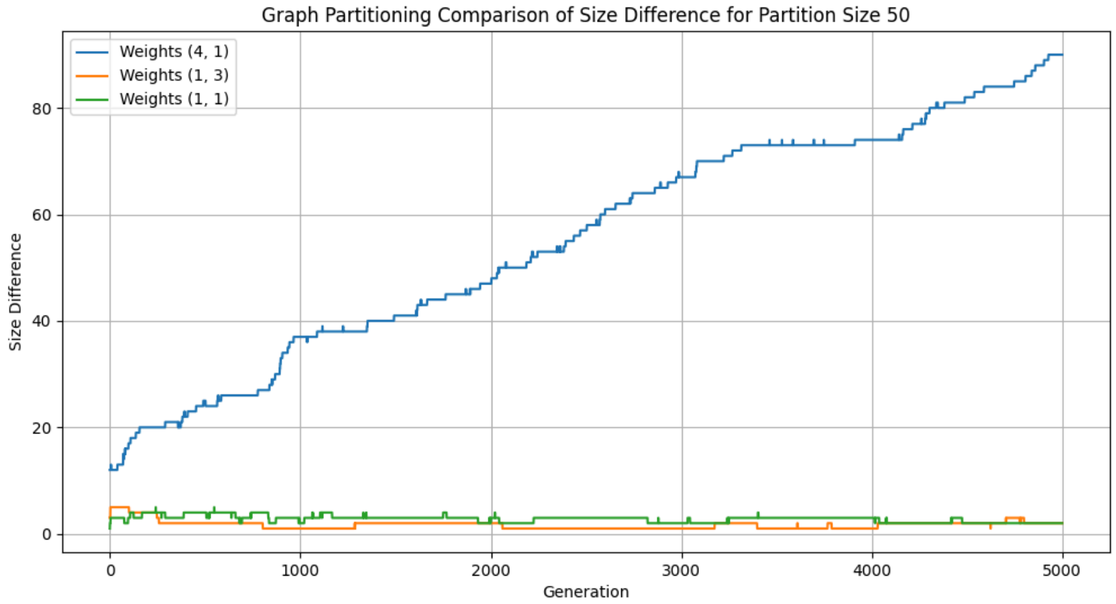




GENETIC ALGORITHM

DENSE GRAPH





FINAL COMPARISONS

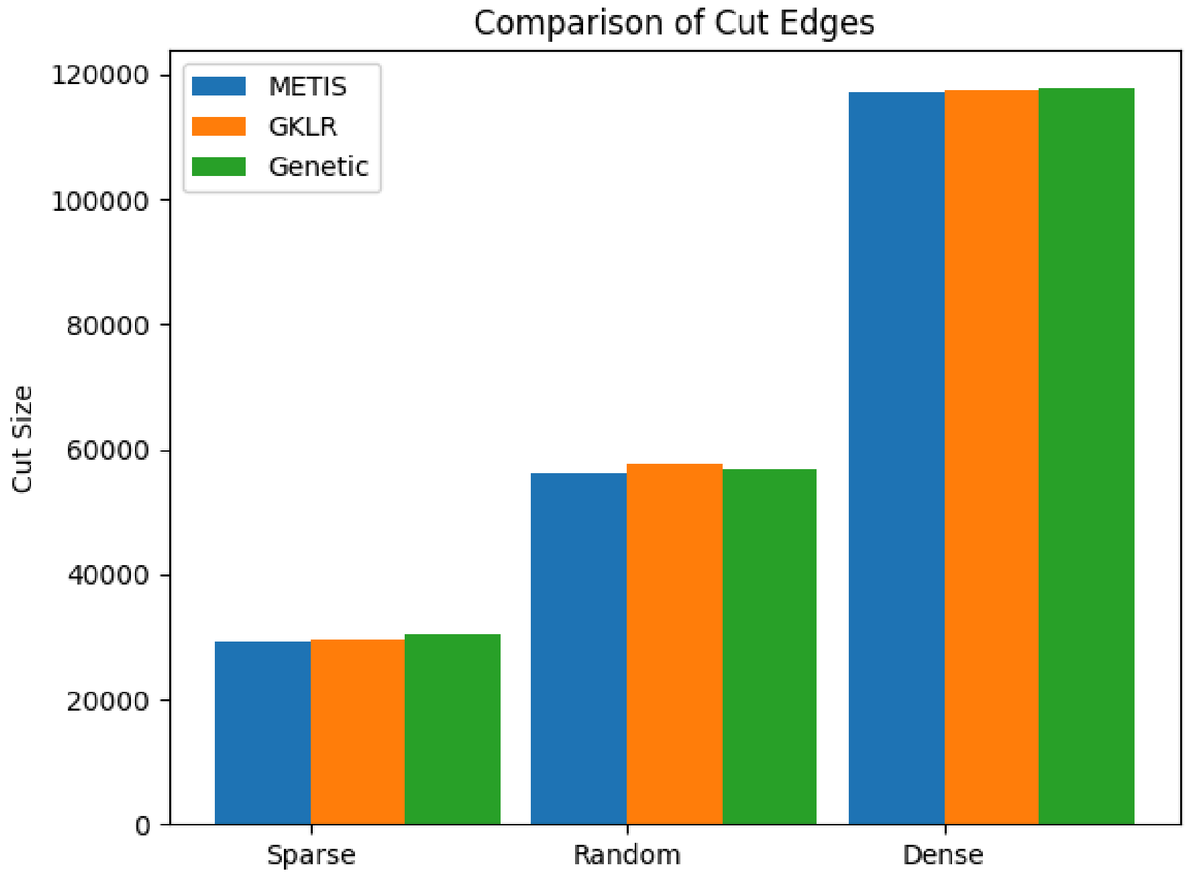
FINAL RESULTS

GRAPH PARAMETERS

Type	Node	Edge
Sparse	573	32676
Random	493	60711
Dense	559	124571

Partition Count -> 20

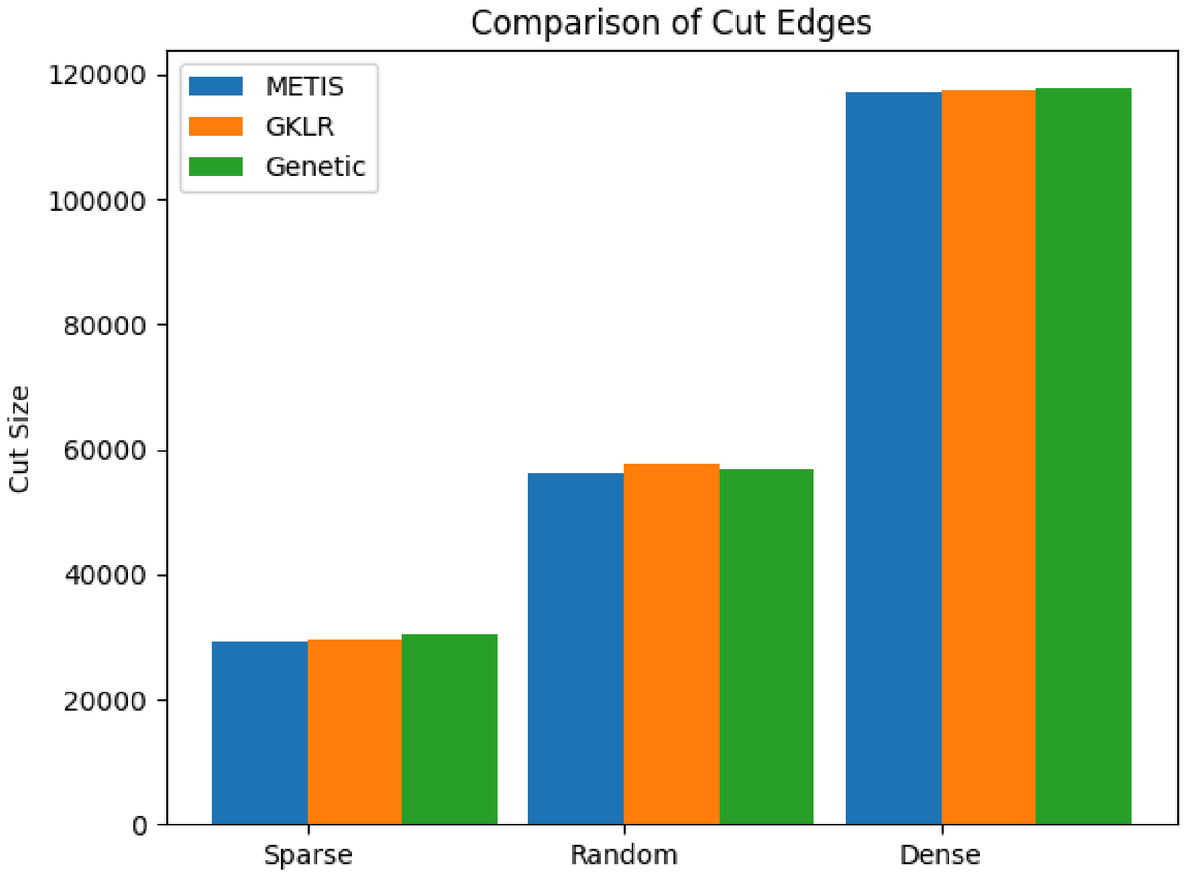
Maximum Partition Size Difference -> 1



FINAL RESULTS

RESULTS

Type	METIS	GKLR	Genetic
Sparse	29269	29635	30578
Random	56172	57738	56802
Dense	117296	117689	117981



FINAL RESULTS

RESULTS

Type	METIS	GKLR	Genetic
Sparse	29269	29635	30578
Random	56172	57738	56802
Dense	117296	117689	117981

$$\frac{GKLR}{METIS} = 1.0115$$

$$\frac{Genetic}{METIS} = 1.013$$

$$\frac{Genetic}{GKLR} = 1.0015$$

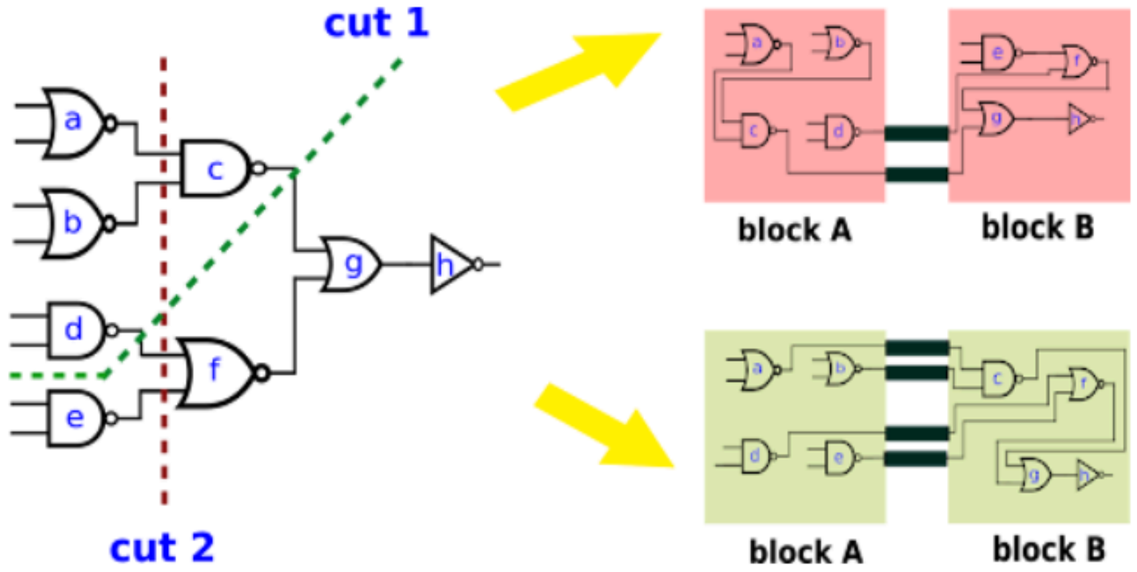
APPENEDIX

APPENDIX

COMMON USE-CASES

Circuit Partitioning

- The goal is to divide a large circuit into smaller, more manageable sub-circuits while minimizing the number of connections between partitions to reduce complexity and optimize performance.
- Circuit components (like gates, modules, flip-flops) are represented as vertices in the graph, while the wires connecting them are represented as edges.
- **Balanced Partitioning**: Each partition should contain approximately equal numbers of components to ensure even distribution of computational load and area utilization.
- **Minimizing Cut Size**: The objective is to minimize the number of edges (interconnections) that cross between different partitions, which directly impacts:
    - Power consumption
    - Signal delay
    - Routing complexity
    - Manufacturing cost



APPENDIX

COMMON USE-CASES

Multi-level Circuit Partitioning:

Complex VLSI circuits are partitioned hierarchically:

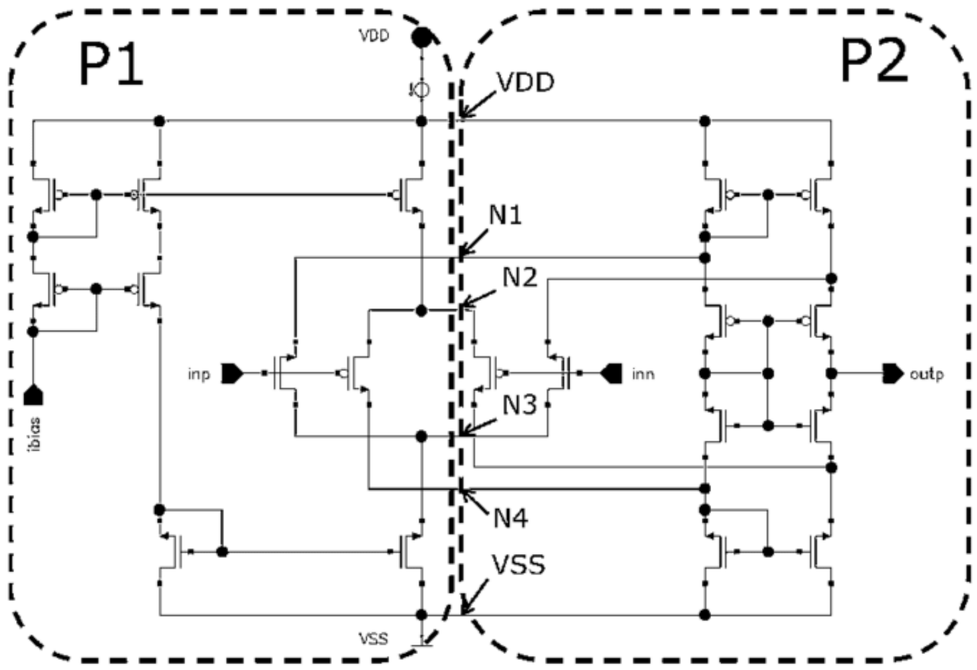
- System level: Dividing into major functional blocks
- Block level: Breaking down functional blocks into smaller modules
- Gate level: Organizing individual gates and components

Chip Floorplanning

- The objective is to arrange circuit modules on a chip while optimizing area utilization and minimizing wire length
- Each partition must consider power distribution, thermal constraints, and signal timing requirements

Physical Design Optimization

- Aims to minimize critical path delays and power consumption through optimal component placement
- Partitions must balance computational load while maintaining signal integrity



Circuit partitioning. The figure illustrates the partitioning of a simplified operational amplifier composed of 19 MOSFETs and one ideal voltage source of zero Volts into two partitions (P1 and P2). The six coupling nodes (VDD, N1, N2, N3, N4, and VSS) are marked by arrows. The first partition P1 has five internal nodes and one current variable (due to the MNA approach each voltage source has one current as MNA variable in the system), whereas the second partition P2 has six internal nodes. The whole circuit has 17 nodes (and thus 18 MNA unknowns) in total.

APPENDIX

COMMON USE-CASES

1. **Load Balancing in Distributed Systems**

- **How Graph Partitioning is Used**: Distributes workload evenly across multiple computing nodes while minimizing the communication overhead between nodes. The system's tasks and their dependencies are modeled as a graph, where balanced partitions ensure efficient resource utilization.
- **Example**: Apache Hadoop uses graph partitioning techniques to distribute large-scale data processing tasks across a cluster, ensuring each node receives a balanced portion of the MapReduce operations while minimizing data transfer between nodes.

2. Simulation Distribution

- **How Graph Partitioning is Used**: Divides large scientific computation tasks into balanced subtasks that can be processed simultaneously across multiple processors while minimizing inter-processor communication. The computation is represented as a graph where nodes are computational tasks and edges represent data dependencies.
- **Example**: Climate modeling systems use graph partitioning to distribute atmospheric and oceanic calculations across supercomputer nodes, where each partition handles calculations for a specific geographic region while maintaining necessary data exchange at boundaries.

APPENDIX

COMMON USE-CASES

SOCIAL NETWORKS:

- **How Graph Partitioning is Used**: Graph partitioning is employed to divide the massive user base into smaller, manageable clusters for efficient data processing, such as analyzing user interactions, recommendations, or querying specific user groups.

- **Example**: In Facebook, graph partitioning is used to cluster users into communities based on friendships, making news feed updates and friend recommendations faster and more efficient.

BIOLOGICAL NETWORKS:

- **How Graph Partitioning is Used:** Used to identify clusters of proteins or genes that are functionally related. In bioinformatics, these partitions help in predicting protein functions and understanding biological pathways

- **Example**: Protein-Protein Interaction **(PPI)** Networks use graph partitioning to identify protein complexes or functional modules involved in biological processes.

APPENDIX

COMMON USE-CASES

TRANSPORTATION NETWORKS:

- **How Graph Partitioning is Used**: It divides a transportation network into smaller regions for efficient route planning and traffic management. This reduces computational complexity when calculating the shortest path.

- **Example**: Google Maps uses graph partitioning techniques to efficiently compute optimal routes by dividing cities into regions and processing them separately for quicker responses.

IMAGE PROCESSING:

- **How Graph Partitioning is Used:** In image segmentation, graph partitioning divides an image into distinct regions or segments, where nodes represent pixels or superpixels, and edges represent the similarity between them. The goal is to group pixels that are similar in terms of color, texture, or intensity.
- **Example:** Normalized Cuts Algorithm is used in image processing to partition an image into segments by minimizing the similarity between different segments while maximizing the similarity within the same segment.

APPENDIX

COMMON USE-CASES

DATABASE MANAGEMENT:

- **How Graph Partitioning is Used:** In distributed databases, graph partitioning divides data into smaller partitions that can be spread across multiple servers to improve query performance and reduce communication overhead between servers.
- **Example:** Neo4j, a graph database, uses graph partitioning to distribute graph data across different servers, allowing for faster querying and load balancing in large-scale database systems. This helps in scenarios such as social network analysis or recommendation engines, where relationships between entities are important.