Assignment ReportTopology\_Mapping

short line

Jamaleldin Ahmed Khalaf 15

Hazem Morsy 16

# Introduction

Topology mapping is a partitioning technique that maps the simulated nodes to different physical nodes. In this assignment, we will use spectral clustering to partition a given network topology on the available physical nodes.

The network topology is a graph of N nodes communicating with each other by sending data traffic through a set of edges. An edge in the topology is weighted by the traffic(Mbps) passing through it.

Our goal is to **find the cut that minimizes the traffic** between different partitions , not to balance nodes in different partitions.

# 

# 

# Dataset

sets:

Sets is a list that holds numpy arrays each numpy array represents a cluster resulted from cutting the graph

To build sets a function called get\_set(labels, groundTruth, k, GT\_bool) where labels is each node from the graph labeled to which cluster it belongs and ground truth of that graph to a specific k and GT\_bool is used if we want the sets to contain the ground truth within the sets or not

Basically if GT\_bool is True then each cluster will hold the number of the node then ground truth and the label of the cluster numpy.array(node, groundTruth, label)

Each cluster then will be added to a list called sets then return sets

But if GT\_bool is false then we discard the ground truth so each cluster will be numpy.array(node, label)

For j in range(k):  
 For i in labels.size:  
 If GT\_bool:  
 Then cluster.add([i, groundTruth[i], labels[i]])  
 Else  
 Then cluster.add([i, labels[i]])  
 sets.add(cluster)  
Return sets

# Conditional Entropy

condtional\_entropy(sets, k):  
 Size = sets.size()  
 Entropy = 0  
 For cluster in sets:  
   
   
   
 For i in range(k):  
 #number of nodes that  
 has the label j in the ground truth column  
   
   
   
 Return Entropy

# F\_Measure

F\_measure(sets, K):  
   
 For cluster in sets:  
   
   
   
   
   
   
   
 Return Fmeasure

# Normalized Cut

Normalized\_Cut(adj\_matrix, sets, K):  
 proximity = zeros(k,k)  
 For i → adj\_matrix.rowSize():  
 C1 = getClusterNumber(index=i, sets, k)  
 For j → adj\_matrix.columnSize():  
 If adj\_matrix[i][j] 0:  
 C2 = getClusterNumber(index=j, sets, k)  
 proximity[C1][C2] = + proximity[C1][C2]  
 If C1 C2:  
 Do the same but for proximity[C2][C1]  
   
   
   
 Return NCut

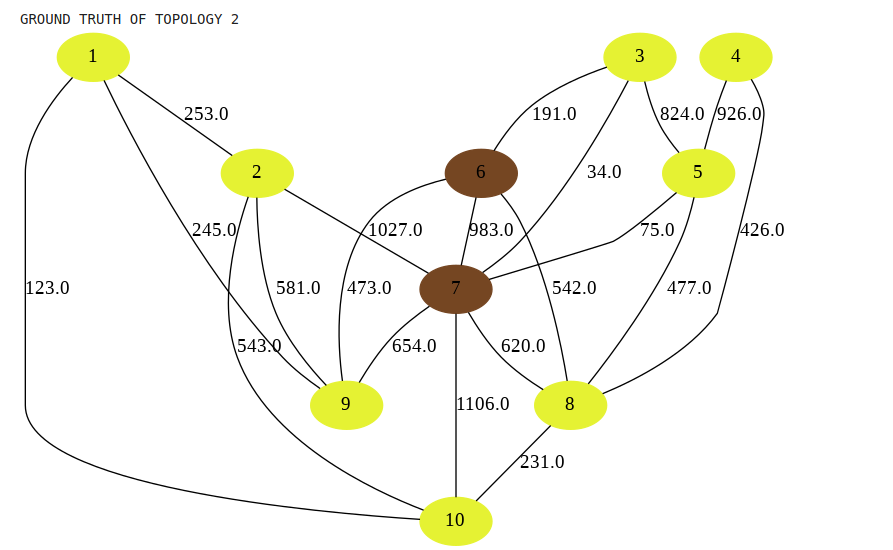
# 

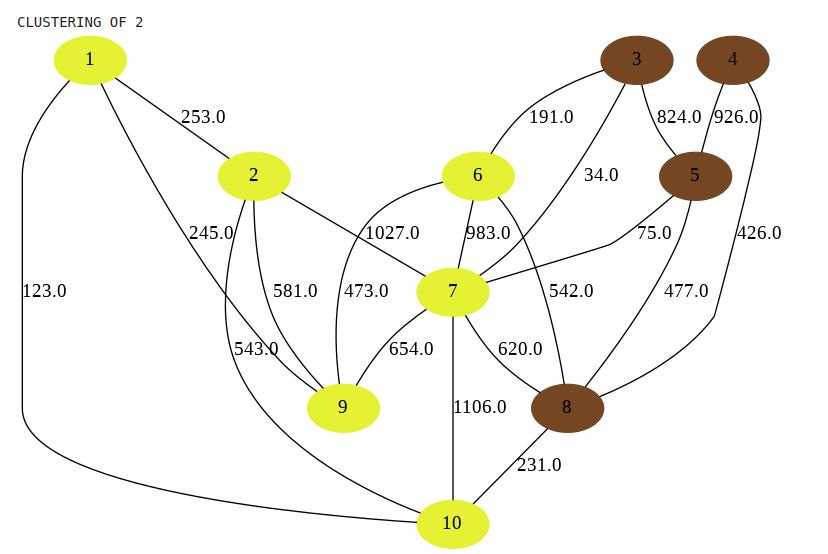
# Results:

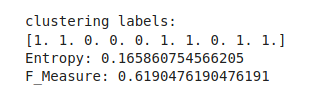
**For topologies of size 10 and 50:**

We used external measures for evaluation (entropy and F-Measure) ,

Entropy measures the noise of data and F-Measure measures the purity and recall of data so the higher the F-Measure and The lower the entropy , the better the clustering.





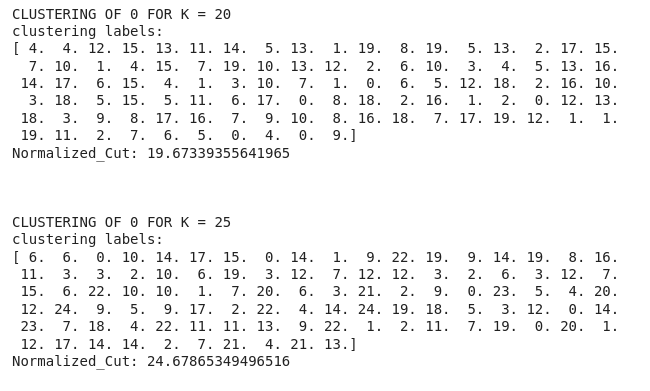


**For topologies of size 100 :**

We used the internal measures for evaluation (Normalized-Cut).

Normalized-Cut measure the balancing of nodes in each partition , the higher the N-Cut the lower the ratio between the internal and external weights.

By increasing k clusters the N-cut increases as the higher k separates the nodes well.



**For measuring the traffic between different clusters:**

We constructed 2d-array of shape ( k-cluster \* k-cluster) in which each entry measures the traffic between cluster of index (i in rows) and (j in columns).

