**Note:** I did this lab assignment by working with Kiante. Therefore, the results and the codes are similar.

**Note:** The reason that I uploaded the results a bit late is that I was waiting for the code to compile one last time.

## Code explanation

The first thing that was obstacle in this lab was the binary files. Binary files were not running on macOS, so we had to re-compile them to get their outputs. This was done by cloning GKlib, METIS, snap, and mlrmcl1.2 and compiling them using Terminal.

To get the outputs from the CNM algorithm for the YouTube dataset, I tried deleting nodes less than a threshold, which I tried deleting nodes with a degree less than 2 to 15, but the problem would remain, and the runtime was not changing much.

On the other hand, we could get the outputs for the other algorithms, and we changed the outputs' formats so we could use them for NetworkX pre-defined functions. These functions are in the Code Listing 1.

Code Listing 1: Functions for output formats

```
def convert_communities_v2(G, communities):

    directed = False
    dic = {}
    id_map = {}
    node_num = 4039
    edge_num = 88234

    in_file = open(G, "rt")

    for line in in_file:
        if("#" in line):
            if("Nodes" in line):
                str_split = line.strip().split()
                node_num = int(str_split[2])
                edge_num = int(str_split[4])
                print(node_num, edge_num)
            continue
        str_split = [int(ele) for ele in line.strip().split()]
        if(str_split[0] == str_split[1]):
            continue
        if(str_split[0] in dic):
            dic[str_split[0]].append(str_split[1])
        else:
            dic[str_split[0]] = []
            dic[str_split[0]].append(str_split[1])
        if(directed == False):
```

```
            if(str_split[1] in dic):
                dic[str_split[1]].append(str_split[0])
            else:
                dic[str_split[1]] = []
                dic[str_split[1]].append(str_split[0])

    count = 1
    key_set = [ele for ele in dic]
    for ele in key_set:
        id_map[ele] = count
        count += 1

    cres = []

    communityDict=defaultdict(set) # creates a dict of keys as community number and values as set of nodeIDs belonging to that community
    communityFile = open(communities, "rt") # the output file obtained from community detection algorithm is given in argv[2]

    outcount=1
    # finds the nodeID for the metics file by looping through the mapping dictionary of the metis file
    for ele in communityFile:
        for node, counts in id_map.items():
            if(outcount==counts):
                communityDict[ele].add(node)
        outcount+= 1

    for key, value in communityDict.items():
        cres.append(frozenset(value))

        return cres
```

Moreover, we used the following functions to get conductance and normalized cuts.

Code Listing 2: Functions for conductance and normalized cuts

```

def conductance_scores(g, c, plot = True):
    c =  convert_communities(g,c)
    #how to get the conductance for all the communities
    conductance_scores_temp = [nx.conductance(g,communities_i, weight='weight') for communities_i in c]
    if plot == True:
        pd.Series(conductance_scores_temp).plot.hist(grid=True, bins=20, color='deepskyblue')
        plt.title('Conductance scores for each community')
        plt.xlabel('Conductance')
        plt.ylabel('Counts')
        plt.style.use('ggplot')
        plt.grid(axis='y', alpha=0.75)
        plt.show()
    return conductance_scores_temp
```

```
15
16 def ncs_scores(g, c, plot = False):
17     c =  convert_communities(g,c)
18     #how to get the conductance for all the communities
19     ncs_scores_temp = [nx.normalized_cut_size(g,communities_i, weight='
           ↪ weight') for communities_i in c]
20     if plot == True:
21         pd.Series(ncs_scores_temp).plot.hist(grid=True, bins=20, color='
               ↪ deepskyblue')
22         plt.title('Normalized cut scores for each community')
23         plt.xlabel('Normalized cut')
24         plt.ylabel('Counts')
25         plt.style.use('ggplot')
26         plt.grid(axis='y', alpha=0.75)
27         plt.show()
28     return ncs_scores_temp
```

## Results and comparisons

In figure 1 we plotted the degree distribution to decide on how to delete the nodes from youtube dataset. The best that out code gave us was the pruned youtube dataset with removing node with degrees less than 20. The runtime of this result was about 8 minutes.

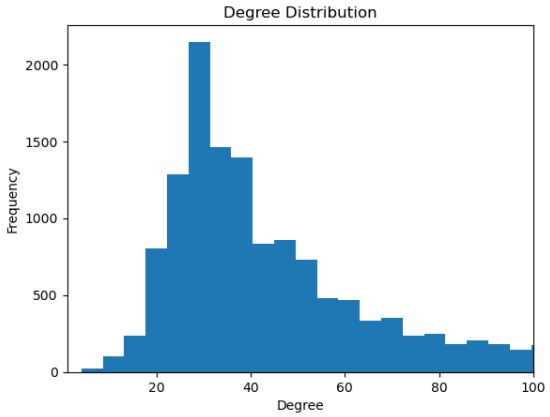


Figure 1: Degree distributions for youtube dataset for degrees between 0 and 100

As we can see in figure 1, deleting the nodes with degrees than 20 seems good. The results have 40852 nodes and 834676 edges with a modularity score of 0.58 and 51 communities.

After getting the desired format for the outputs, we used the networkx.algorithm.community library to compute the modularity of each output. In figure 2, we can see the bar plot for each dataset, except the YouTube dataset, and in figure 3, we can see two plots from the first lab assignment.

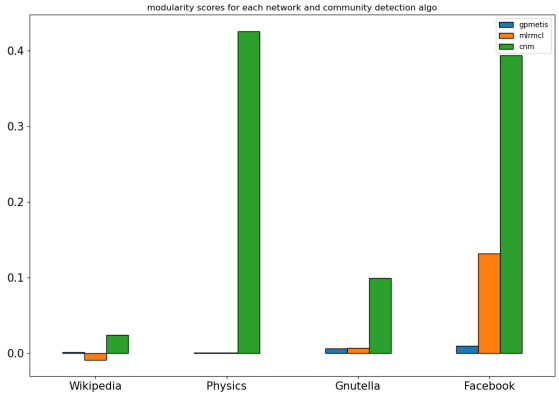


Figure 2: Modularity scores computed by different algorithms for each dataset.

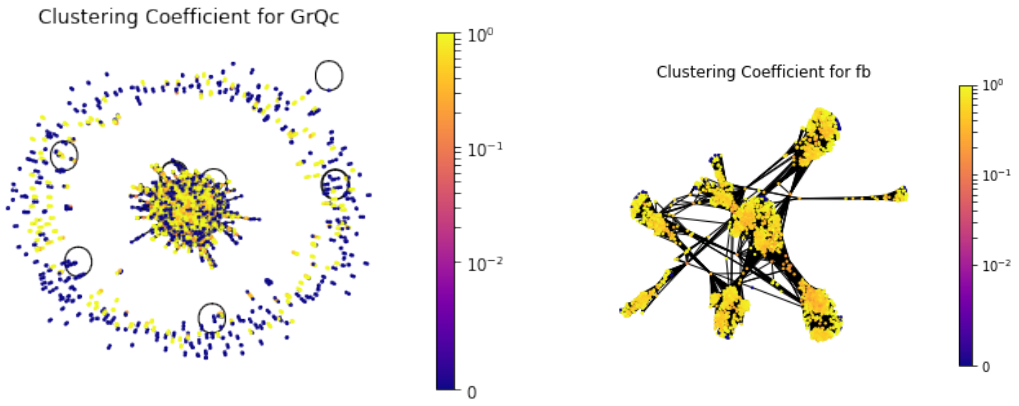


Figure 3: Heatmap for degree centrality and an approximate visualization of the network for Physics (left) and facebook (right).

A high modularity score in a network means that the network can be divided into distinct modules or communities, with nodes within each module being more densely connected than nodes in other modules. In figure 3, we expect our algorithms to have high modularity scores in facebook and physics datasets. This is the case for the CNM algorithm, but we can't say they are successful for MLR-MCL and gpmetis.

Furthermore, by changing the -c 500 for MLR-MCL to -c 1000 and -c 10000 we got different results that are in figure 4.

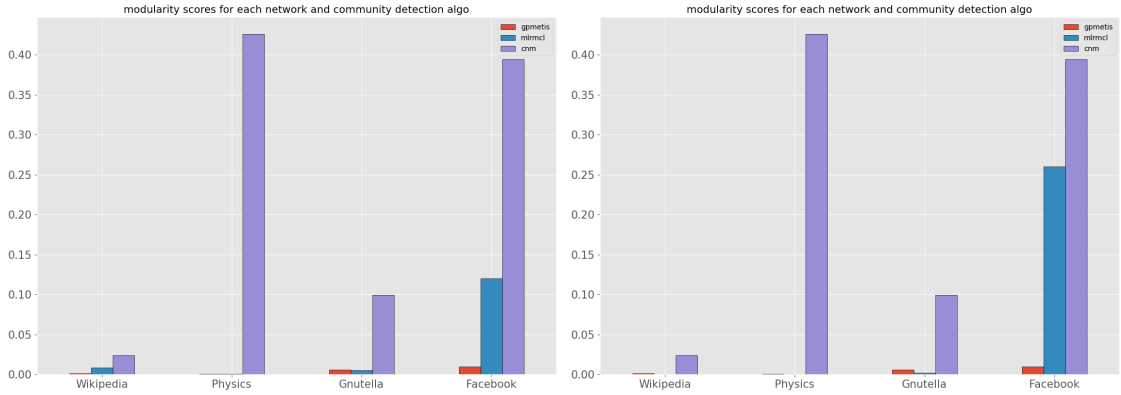


Figure 4: Changing parameters on MLR-MCL to -c 1000 (left) and -c 10000 (right).

As we can see, the score for facebook dataset grows but almost stays the same for other datasets.

Another measure computed was conductance. Conductance is a measure of the connectivity between a subgraph of a network and the rest of the network. Specifically, it measures the ratio of the number of edges that connect the nodes within the subgraph

to nodes outside the subgraph, to the total number of edges incident on the nodes in the subgraph.

Intuitively, a subgraph with high conductance has many edges that connect it to the rest of the network, indicating that it is not well-separated from the rest of the network. A subgraph with low conductance, on the other hand, has relatively few edges that connect it to the rest of the network, indicating that it is more isolated or distinct from the rest of the network.

In figure 5 we can see the conductance for all algorithms and datasets, except the YouTube algorithm which took too long to get the results, and was the reason I uploaded this file late!

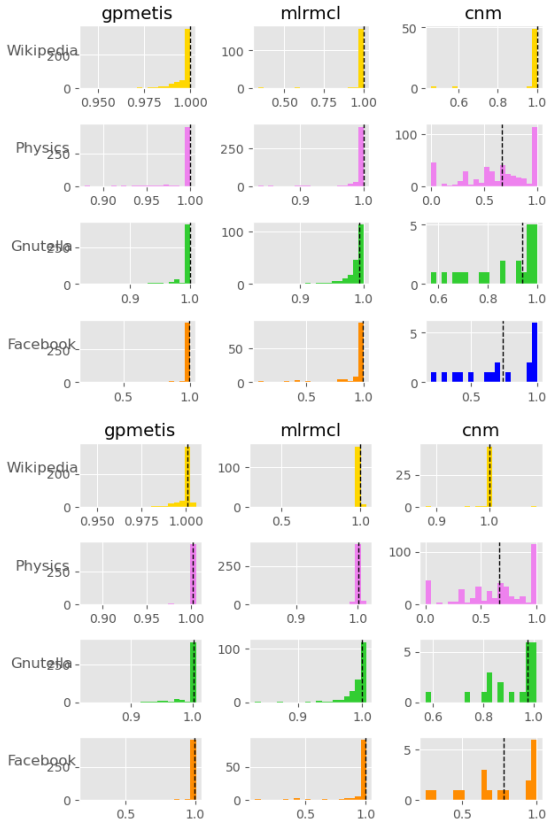


Figure 5: Conductance (upper) and normalized cuts (lower) for different algorithms on different datasets.

**Final note:** I waited for the final results but since the assignment is closed, I will send the files as they are and I will send the updated files after I have them.

## References

[1] Easley, David; Kleinberg Jon. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*. Cambridge University Press, 2010. Print.

[2] "Stanford Network Analysis Project". SNAP, Stanford. Online. `http://snap.stanford.edu/index.html`

[3] "METIS". Online. `https://github.com/KarypisLab/METIS`

[4] "GKlib". Online. `https://github.com/KarypisLab/GKlib`

[5] "Network Analysis in Python". NetworkX. Online. `https://networkx.org/documentation/stable/reference/introduction.html`