Solutions to Assignment 2
Network Science
PHYS 7332, Fall 2024

Chelsea Ajunwa*

1. *Question 1.*

(a) Redis uses a variety of data structures, including key-value pairs, strings, sets, and hashes. It is most useful for high-speed applications, caching, real-time analytics, and messaging. An example of a bad use case is one that requires a large amount of memory. This is because it is an in-memory data store; if your data exceeds what can be stored in memory, Redis is not appropriate.

(b) MongoDB structures data as BSON (JSON-like) documents that are organized into collections. It is most useful for large and complex data and for querying. An example of a bad use case is one that requires high speeds; MongoDB is slow compared to databases like Redis.

(c) Cassandra structures data in partitioned rows. It can be used for structures, semi-structured, and unstructured data, and is also useful for querying. An example of a bad use case is one in which the data is simpler and doesn't require the scalability of Cassandra. It is more efficient to use a diferent database in this case.

(d) Neo4j structures data as graphs with nodes and edges connecting the nodes. It is most useful for interconnected data. Social network data is one example. A bad use case would be one with simple, non-interconnected data. In this case, a different database would be more efficient.

2. *Question 2.*

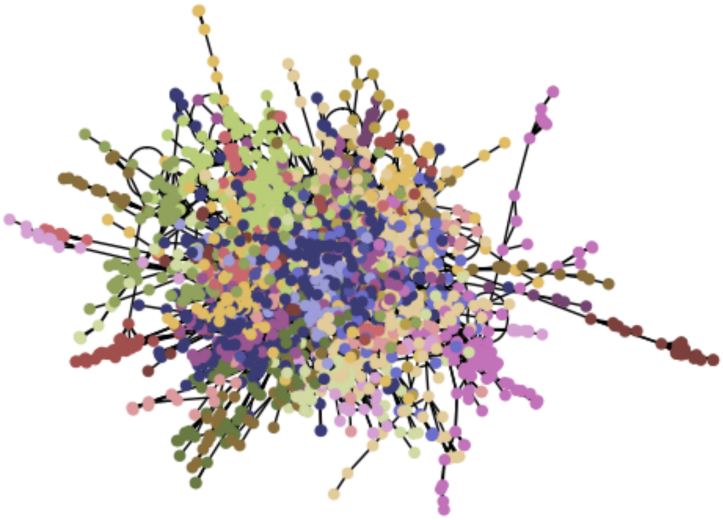
I downloaded a social network dataset from networkrepository.com. The nodes are TV show Facebook pages and the edges are mutual likes.

```
#Creating graph from edge file

G = nx.read_edgelist("fb-pages-tvshow/fb-pages-tvshow.edges",
    nodetype=str,delimiter=",")
# nx.draw(G)

#adding nodes
nodes = {}
```

* **ADD CLASS REPO FORK** https://github.com/cajunwa/phys7332_fa25

```
with open("fb-pages-tvshow/fb-pages-tvshow.nodes") as f:
    for line in f:
        cols = line.strip().split(",")
        # print(cols)
        node_id = cols[2]
        name = cols[1]
        nodes[node_id] = {"name": name}

nx.set_node_attributes(G, nodes)
```

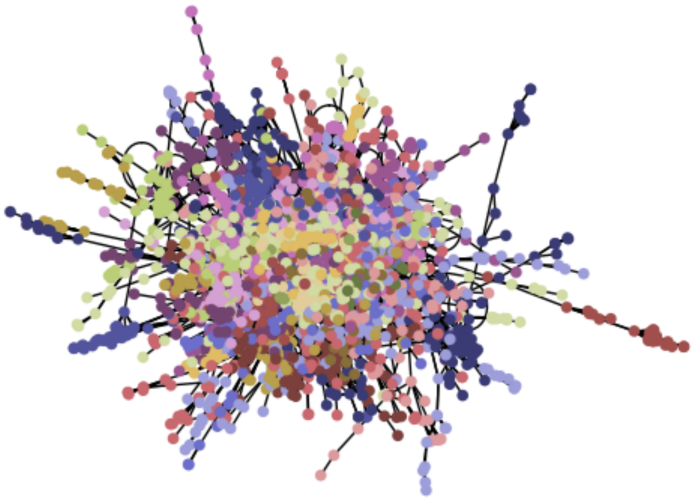
(a) i.



```
#Problem 2i
#Modularity maximization

partition = community.best_partition(G)


colors_sns =
    plt.cm.tab20b(np.linspace(0,1,len(np.unique(list(partition.values())))))
colors_nodes = [colors_sns[partition[node]] for node in G.nodes()]

fig, ax = plt.subplots(1,1,figsize=(7,5),dpi=100)

pos = nx.spring_layout(G)

nx.draw(G, pos=pos,node_color=colors_nodes, ax=ax,node_size=20)
```

ii. $Q = 0.872$

iii. 48 communities were detected.

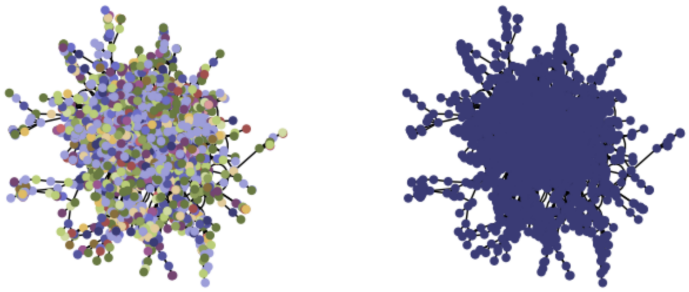


(b)

```
#Using graph-tool to create a partition

#converting to graph-tool graph
gtg = pyintergraph.nx2gt(G, labelname="node_label")

#creating partition of network
state = gt.minimize_blockmodel_dl(gtg)



#storing partition as dictionary and change node labels back to
    networkx format

b = state.get_blocks()

gt_partition = {vertex:0 for vertex in list(gtg.get_vertices())}

for vertex in list(gtg.get_vertices()):
    gt_partition[vertex] = b[vertex]


#visualizing network

colors_sns =
    plt.cm.tab20b(np.linspace(0,1,len(np.unique(list(gt_partition.values())))))

map_communities={y:x for x,y in enumerate(set(gt_partition.values()))}
```

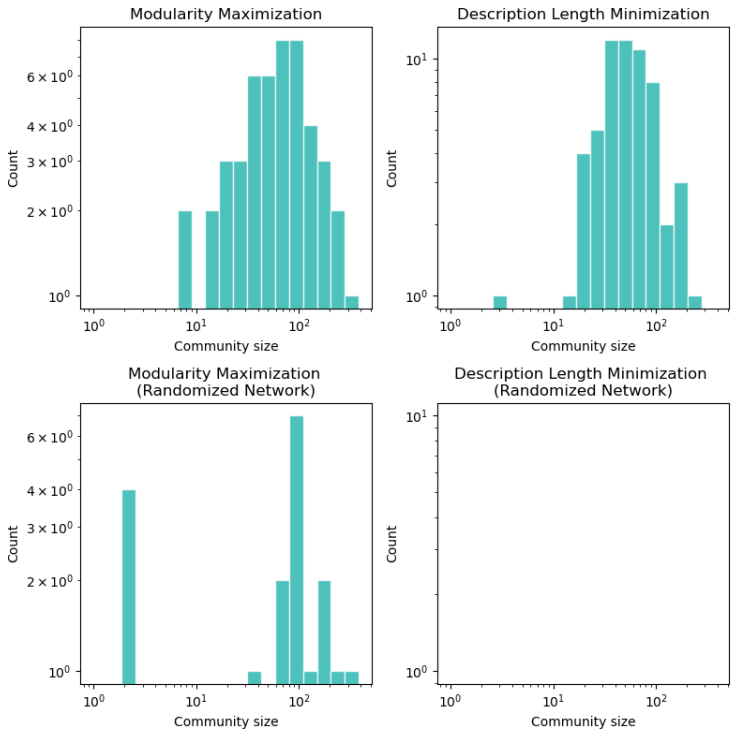
```
colors_nodes = [colors_sns[map_communities[gt_partition[node]]] for
    node in list(gtg.vertices())]

fig, ax = plt.subplots(1,1,figsize=(7,5),dpi=100)

nx.draw(G, pos=pos,node_color=colors_nodes, ax=ax,node_size=20)

#number of communities
gt_num_communities=len(set(list(gt_partition.values())))
print(f'Number of communites: {gt_num_communities}')
```

(c)



```
#First get degree sequence

degree_seq = [dict(G.degree)[node] for node in G.nodes()]

#Next, create degree-preserving randomization

G_rand = nx.random_degree_sequence_graph(degree_seq)

#running modularity maximization
partition_rand = community.best_partition(G_rand)

#running description length minimization
gtg_rand = pyintergraph.nx2gt(G_rand, labelname="node_label")
state = gt.minimize_blockmodel_dl(gtg_rand)

b = state.get_blocks()

gt_partition_rand = {vertex:0 for vertex in
    list(gtg_rand.get_vertices())}

for vertex in list(gtg_rand.get_vertices()):
    gt_partition_rand[vertex] = b[vertex]
```

(d)




```
#plotting community size distributions


def get_comm_sizes(partition):
    """
    Gets list of community sizes for a given network partition.


    Inputs:
    partition - Network partition in dictionary form {node id:
        community id}.

    Outputs:
    comm_sizes - List of community sizes.


    """

    communities = list(set(list(partition.values())))

    comm_sizes_dict = {community:0 for community in communities}
    for node in list(partition.keys()):
        for community in communities:
            if partition[node] == community:
```

```
                comm_sizes_dict[community] += 1

    comm_sizes = list(comm_sizes_dict.values())

    return comm_sizes

#plotting
fig,ax = plt.subplots(2,2,figsize=(8,8),dpi=100)

ax[0,0].hist(get_comm_sizes(partition),color='lightseagreen',
    edgecolor='white',alpha=0.8, bins=np.logspace(0,
    np.log10(max(get_comm_sizes(partition))), 20))
ax[0,0].set_xscale('log')
ax[0,0].set_yscale('log')
ax[0,0].set_xlabel("Community size")
ax[0,0].set_ylabel("Count")
ax[0,0].set_title("Modularity Maximization")



ax[0,1].hist(get_comm_sizes(gt_partition),color='lightseagreen',
    edgecolor='white',alpha=0.8, bins=np.logspace(0,
    np.log10(max(get_comm_sizes(partition))), 20))
ax[0,1].set_xscale('log')
ax[0,1].set_yscale('log')
ax[0,1].set_xlabel("Community size")
ax[0,1].set_ylabel("Count")
ax[0,1].set_title("Description Length Minimization")



ax[1,0].hist(get_comm_sizes(partition_rand),color='lightseagreen',
    edgecolor='white',alpha=0.8, bins=np.logspace(0,
    np.log10(max(get_comm_sizes(partition))), 20))
ax[1,0].set_xscale('log')
ax[1,0].set_yscale('log')
ax[1,0].set_xlabel("Community size")
ax[1,0].set_ylabel("Count")
ax[1,0].set_title("Modularity Maximization \n(Randomized Network)")


ax[1,1].hist(get_comm_sizes(gt_partition_rand),color='lightseagreen',
    edgecolor='white',alpha=0.8, bins=np.logspace(0,
    np.log10(max(get_comm_sizes(partition))), 20))
ax[1,1].set_xscale('log')
```

```
ax[1,1].set_yscale('log')
ax[1,1].set_xlabel("Community size")
ax[1,1].set_ylabel("Count")
ax[1,1].set_title("Description Length Minimization \n(Randomized
    Network)")

plt.tight_layout()
plt.savefig('comm_sizes.png')
plt.show()
```

(e) I could compare partitions using an information-theoretic measure of similarity such as the one proposed in Meilă, 2007 (https://doi.org/10.1016/j.jmva.2006.11.013). They proposed a metric called variation of information. The definition of this metric is as follows: $VI(P1, P2) = H(P1|P2) + H(P2|P1)$. If I were to write a function that takes two partitions as input and calculates the variation of information, it would first check the partitions to ensure they are valid and in dictionary form. It would then calculate the necessary probabilities and entropies and then output a float.

3. *Question 3.*

(a) This paper was motivated by the lack of an adequate benchmark for community detection algorithms. Community structure is a core feature of many networks. By the time this paper was published, many methods of community detection had been devised, but it was unclear which was best at identifying meaningful modules. This is where benchmarks come in. Benchmarks allow us to test community detection algorithms with a network whose community structure is known in order to assess the performance of the algorithms.

The most famous benchmark prior to this paper was the Girvan and Newman benchmark. However, the Girvan-Newman benchmark's nodes were all very similar in degree, the communities were the same size, and the network was small. It was therefore dissimilar from typical networks encountered in the real world. The authors of this paper set out to design a benchmark that was more similar to these real networks.

The main contribution of this paper was a new benchmark network with heterogeneity in node degree (with a degree distribution that decays as a power law), a node number parameter, and a distribution of community sizes that follow a power law. The inputs to this benchmark are number of nodes (N), average degree k, and exponents γ and β. The output is a network.

(b) i.

```
# Question 3bi
# Write a function that outputs a powerlaw degree sequence

def powerlaw_degree_sequence(n,gamma,k):
```

```
"""
Outputs a power law degree sequence {k1,k2,...kn}, which can be
    used as input in a configuration model.

Inputs:
n (int) :
    number of nodes
gamma (int) :
    power law degree exponent
k (int) :
    average degree

Output:
degrees (array) :
    list of degrees

"""



#generate raw samples
seq_orig = scipy.stats.pareto.rvs(b=gamma-1,size=n)

#convert to integers
degrees = np.round(seq_orig).astype(int)

#scale to match average degree
current_mean = np.mean(degrees)
scaler = k/current_mean
degrees = np.round(degrees*scaler).astype(int)


return degrees
```

```
#Question 3bii


import numpy as np

def degree_split(degree_sequence, mu):
"""
Assigns fraction of node's degrees to be internal community edges
    based on mu value.
```

```
    Inputs:
    degree_sequence (list):
        list of node degrees
    mu (float):
        degree assignment parameter. Fraction between 0 and 1.


    Outputs:
    in_degrees (list) :
        List of node degrees that are in the node's community.
    out_degrees (list):
        List of node degrees that are outside the node's community.
    """


    in_degrees = np.round(degree_sequence * (1 - mu)).astype(int)


    out_degrees = degree_sequence - in_degrees



    return in_degrees, out_degrees
```

iii. In order to implement this step, I'd need to include another input (β) in the function. The function would also need to include some code that ensures the sum of all the community sizes are equal to the number of nodes in the graph. And it would need to constrain the maximum and minimum community size such that they are larger than the maximum and minimum degrees, respectively.

(c) Increasing the mixing parameter, mu, seems to increase the probability of success. When mu is too small, the communities within the networks may be too weak to detect given the resolution limit of modularity maximization approaches.

```
#example 1
#G=nx.LFR_benchmark_graph(n=40,tau1=2,tau2=3,mu=0.3,average_degree=7)
#attempts:
# Successful
# ExceededMaxIterations: Could not assign communities; try increasing
    min_community
# ExceededMaxIterations: Could not assign communities; try increasing
    min_community
# Successful
```

```
# Successful

#example 2 (same but with min_community increased)
#G=nx.LFR_benchmark_graph(n=40,tau1=2,tau2=3,mu=0.3,average_degree=7,min_community=3)
#attempts:
# ExceededMaxIterations: Could not assign communities; try increasing
    min_community
# ExceededMaxIterations: Could not assign communities; try increasing
    min_community
# ExceededMaxIterations: Could not assign communities; try increasing
    min_community

#example 3 (n and min_community increased)
#G=nx.LFR_benchmark_graph(n=1000,tau1=2,tau2=3,mu=0.3,average_degree=7,min_community=
#attempts:
# ExceededMaxIterations: Could not assign communities; try increasing
    min_community
# ExceededMaxIterations: Could not assign communities; try increasing
    min_community

#example 4 ( min_community increased further)
#G=nx.LFR_benchmark_graph(n=1000,tau1=2,tau2=3,mu=0.3,average_degree=7,min_community=
#attempts:
# Does not converge
# Successful
# ExceededMaxIterations: Could not assign communities; try increasing
    min_community

#example 5 ( larger mu)
#G=nx.LFR_benchmark_graph(n=100,tau1=2,tau2=3,mu=0.8,average_degree=7,min_community=5
#attempts:
# Successful
# ExceededMaxIterations: Could not assign communities; try increasing
    min_community
# Successful
# Successful

#example 6 ( even larger mu)
#G=nx.LFR_benchmark_graph(n=100,tau1=2,tau2=3,mu=0.9,average_degree=7,min_community=5
# Successful
# Successful
# Successful

#example 7 (high mu and higher average degree and min community)
#G=nx.LFR_benchmark_graph(n=1000,tau1=2,tau2=3,mu=0.9,average_degree=16,min_community
```

```
# Successful
# ExceededMaxIterations: Could not assign communities; try increasing
    min_community
# Successful

#example 7 (same but low mu)
G=nx.LFR_benchmark_graph(n=1000,tau1=2,tau2=3,mu=0.1,average_degree=16,min_community=
#attempts:
# ExceededMaxIterations: Could not assign communities; try increasing
    min_community
# ExceededMaxIterations: Could not assign communities; try increasing
    min_community
# ExceededMaxIterations: Could not assign communities; try increasing
    min_community
```

---

4. *Question 4.*

   (a) See Github https://github.com/cajunwa/graphwork/tree/main

   (b) See Github https://github.com/cajunwa/graphwork/tree/main

5. *(Optional) Question 5 (5 bonus points).*

   For every problem set this semester, there will be an opportunity to earn extra credit. Since this is a new course, we are eager to receive genuine feedback on the material, pace, instruction style, etc. of the course so far. Please provide any feedback about the lectures in recent weeks and/or this assignment.

   ...

If you reference anything, add it `main.bib` and cite using `\cite{Hartle2020wegd}`.