An Exploration in Creating an Optimal Random Community Detection Algorithm

Rithika .D.

- Communities are subgroups within a network structure that reflect nodes that share more ties amongst their group members than the rest of the graph
- Communities are also reflective of users on social media where those who have more shared interests or viewpoints are more likely to interact with each other and form tight knit communities leading to issues arising such as echo chambers and an increased likelihood of misinformation especially as these views become more harmful.
- My implementation (derived from the optimal community detection algorithm) introduces a random element in forming subgroups amongst a network in order to directly suppress algorithmic biases and the rise of echo chambers as forming randomized user hubs is more likely to offer a more diverse perspective and exchange of thoughts on social media
- Can be potentially powerful when dealing with the rise of misinformation and centralized niche communities across social media as well as directly contradicting the personalized recommendations for alternate posts or influencers for any given user.

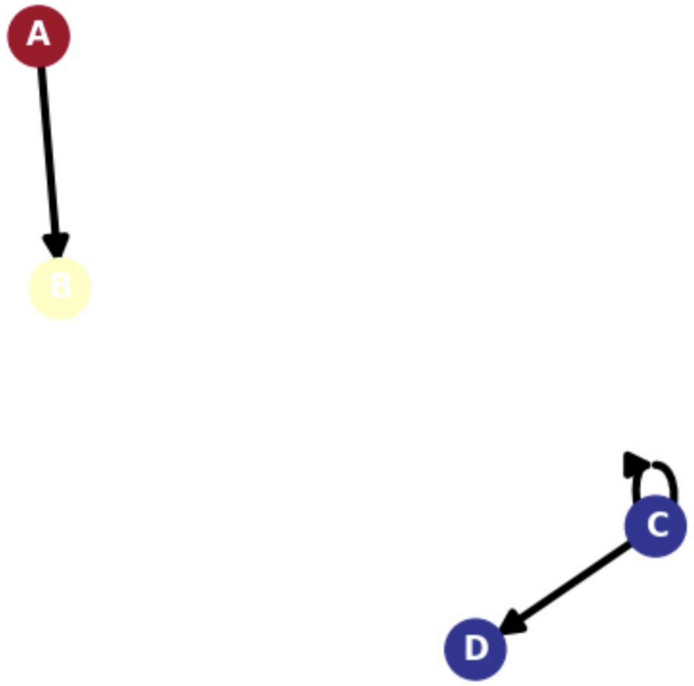
Pseudocode 1#

```
def create_clusters(edgelist):
    e_list = every unique node in the given edgelist
    #create rand_list to contain the number of elements per every subgroup
    rand_list = []
    #now we generate a random number ranging from 1 to the number of nodes we have to determine how many nodes are included in every community and append them to rand_list
    output = the resulting communities that were split according to the random index position
    communities = output after removing every occurrence of an empty sublist
    modularity = built in networkx function that calculates the network modularity of our community structure generated from the communities variable
    group_assignment = []
    #the group_assignment list refers to the index of every community meaning that we number them according to a given group's position amongst all of them starting from community 0 to the last one where we have for example node A associated with the first community
    return(communities, e_list, group_assignment, modularity)
```

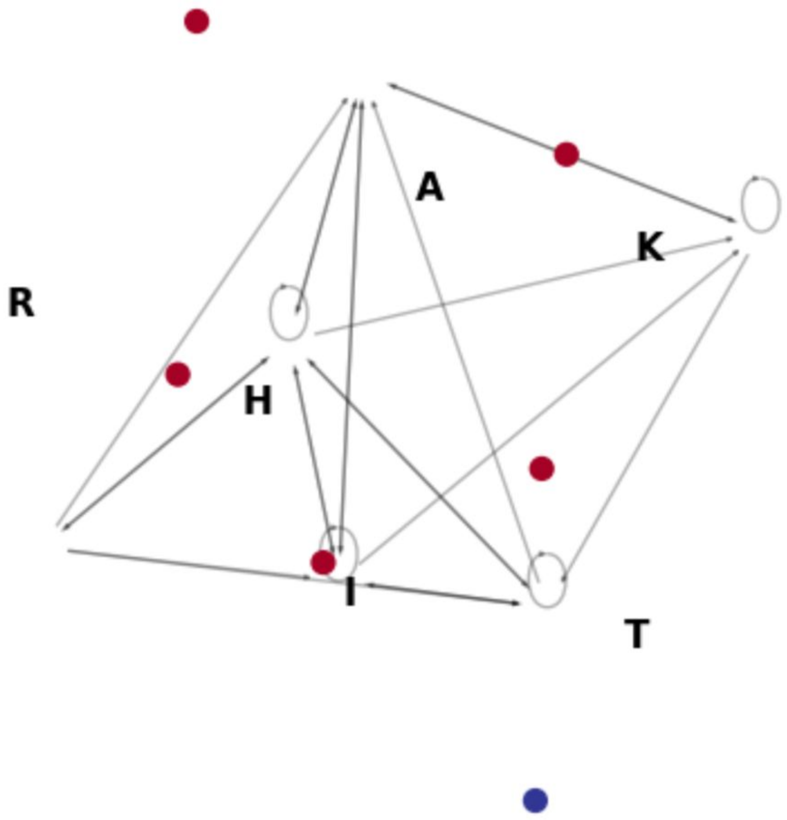
Pseudocode 2#

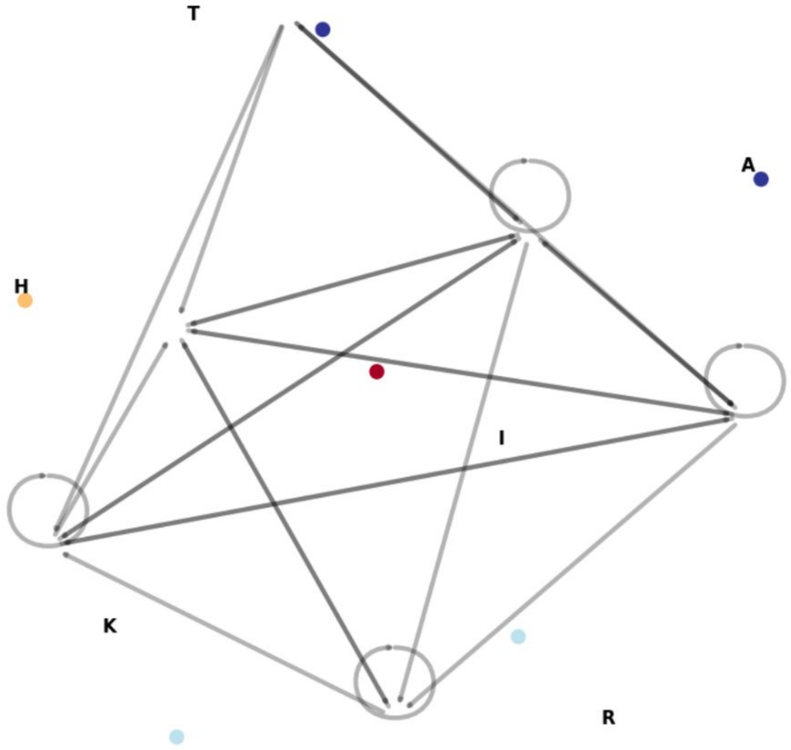
```
def optimal_random(edgelist):
    #create empty list cases
    cases[]
    #now we generate a thousand iterations of community layouts from the create_clusters() function and record every result in cases
    for i in range(1000):
        append iteration i of create_clusters(edgelist) to cases[]
    max = the first modularity score associated with a community assignment as recorded in the information cases[] stores
    #now we iterate through cases and determine the maximum network modularity score and record the index (stored in the variable max_index) associated with that value
    #our final result is the community assignment associated with the highest modularity score otherwise known as the most optimal or tightly knit subgroups amongst the overall network structure
    return(cases[max_index])
```

The output of create_clusters() on a small graph with nodes: A, B, C, D in which A and B were in their own communities while there was a group contained both C and D. Below is the data associated with how the communities were determine with the first element representing the subgroups in the form of sublists, next represents every unique node in the edgelist, third is the associated numerical community for each node based on their positioning, and finally the last value is the network modularity score for the community structure.

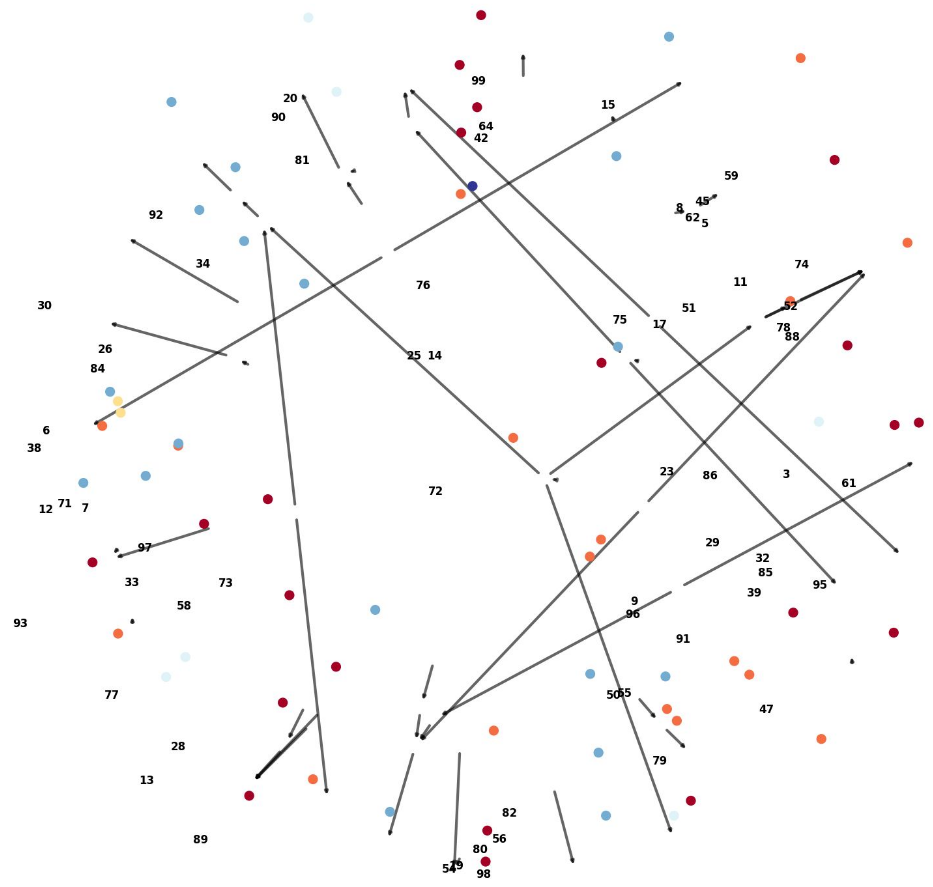


```
([['A'], ['B'], ['C', 'D']],
 NodeView(('A', 'B', 'C', 'D')),
 [0, 1, 2, 2],
 0.2222222222222222)
```





Leftmost graph was generated by one iteration of the create_clusters() function while the rightmost one was created using the results of the optimal_random() implementation that takes into account a thousand runs and uses the best performing community distribution



The result of the optimal_random() implementation on a graph that contained over a hundred nodes with the most cohesive subgroups detected amongst the large network structure.

The algorithm can work on a graph of any size though the time to run it might differ though not significantly