MINI PROJECT

By Phuong Nguyen, Sarah Erian, Craig Neely

Introduction:

The code reads a graph from a file, and repeatedly finds new communities until all nodes are assigned to a community. In the first iteration, it starts with a node with the highest degree. Then, in each iteration, it adds neighbors with the highest overlap with the current community. If there is a tie between 2 nodes, it picks the one with the higher degree. The process stops when the density of the resulting subgraph drops below a threshold value (0.7). Finally, it updates the graph by removing nodes that are not linked to any other nodes outside the new community and appends the new community to a list of communities found so far.

Description:

- FIRST METHOD

1. This code reads a graph file and finds the communities within the graph. It uses a graph library 'dijkstar' to represent graphs in python. The following methods are used in this code:
2. FindHighestDegreeNode" function takes a graph as an input and returns a node with the highest degree, where the degree is the number of edges connected to the node.
3. The "overlap" function takes two arrays (a1, a2) as an input and returns the count of common elements between both arrays.
4. The "FindHighestDegreeNeighbor" function takes a graph and community as an input and returns a neighbor with the highest degree outside the community. An important point to note is that this neighbor should share the maximum overlap with the community.
5. The "CalculateDensity" function takes a graph object and a list of nodes representing a community and calculates the density of that community.
6. The "updateGraph" function takes a graph object and a list of nodes representing a community and removes all nodes (with their edges) that are not linked to any other nodes outside that community.
7. The code creates an empty list "community_list" to store the resulting communities and then enters a while loop that continues until there are no more nodes left in the graph. Inside this loop, it first creates an empty list "new_community" and finds the node with the highest degree using `FindHighestDegreeNode` function. It appends this

node to the “new_community” list and then keeps adding neighbors to the community as long as the calculated density of the community is greater than or equal to 0.7. It uses the “FindHighestDegreeNeighbor” function to find the next node to add to the community and uses the “CalculateDensity” function to calculate the updated density of the community after each iteration.

8. Finally, it calls the “updateGraph” function to remove the nodes that are not connected to any other nodes outside the community, updates the original graph object, and adds the resulting “new_community” list to the “community_list”.

ð Overall, the code implements a community detection algorithm that forms communities based on high-degree nodes and their neighbors and removes any nodes that are isolated from the communities.

- SECOND METHOD

o For the second method, it is an attempt to obtain communities our first method misses or to still show results when our first method gets stuck in an infinite loop (which can happen for certain graphs).

o This method is similar to how the cluster coefficient can be found, in that the first node in the list is selected and then the degree is then found.

o If this node has a degree greater than one, the first neighbor of this node is then selected, and they are both added to a new community.

o We then find the neighbor for the first neighbor node and check to see if it is also a neighbor to the original node selected; if this is the case it is then added to the community and the density is calculated.

o This smaller neighbor selection process repeats for this first node until the community has a density lower than 0.7 or all neighbors have been added.

o Finally, the entire process is repeated for every node, with duplicate communities not being added.

o At the end of the code, the original method's communities are compared with the second method’s communities and the differences are combined together to show all communities with a density greater than 0.7. This does include some smaller communities that make up larger communities.

Limitations and Improvement:

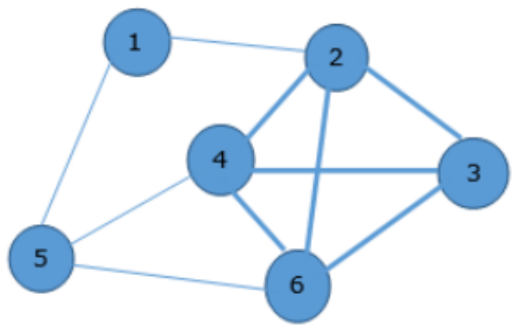
There is room for improvement to find and handle more cases:

· Since the algorithm adds only one node at a time and stops when the density drops below the threshold, it would miss the following case:

o Adding one node would make the density drop but adding 3-4 more nodes increases the overall density.

o Since the algorithm has a specific starting point and removes the nodes not connected to any other nodes in the community after each community detected, it would miss a community in the following graph:

o The detected communities in this graph are {2,3,4,5,6} and {1,2}. It couldn't detect{1,2,4,5,6}



o There are some instances where our original method can get stuck in an infinite loop. There are also some instances where each method seems to come up with different communities, which shows neither method is able to discover all communities desired; however, together they can provide a more complete answer than each of them by itself.

o Another improvement would be efficiency as both these methods have steps that are most likely not necessary to be repeated as some duplicates are found, and some code can probably be trimmed out.