# Girvan-Newman Algorithm (Finding Communities)

*Scott Tornquist*
CPSC 450
Spring 2021

## Summary

(1) The algorithm looks at the problem of identifying communities within a graph (2) My implementation approach was a suite of test cases showing the algorithm of large and large graphs. I used the graph environment from our class. I had to change the edge value in the graph from an int to a double for the calculations (3) I have a working Grivan-Newman algorithm with 2 helper functions and 6 test cases.

## 1. ALGORITHM SELECTED

The problem is finding communities within a graph. The Girvan-Newman algorithm does this by calculating the edge betweenness (EBC scores) for each edge and removing the largest edges.

https://www.analyticsvidhya.com/blog/2020/04/community-detection-graphs-networks/

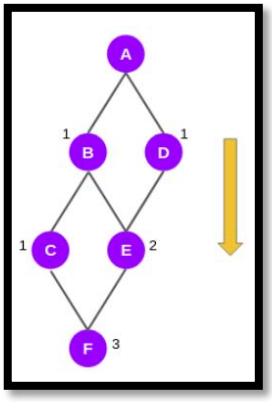
1. Find EBC scores
2. Elimnate largest edges until there are multiple components

This is O(n^2) algorithm because it is a nested for loop that must iterate over each node and find the shortest path for ever node from every node

## 2. BASE IMPLEMENTATION

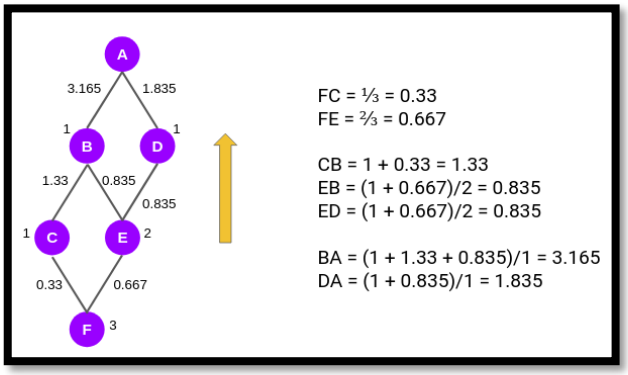
The Girvan-Neman algorithm (bool girvan_newman(Graph& community_graph)) accepts a graph with no edges as input. The function establishes the communities of the graph the function is called upon in this the return graph. The graph that is being mapped must be connected, the function returns false if the graph has more than one weakly connected component. The function determines communities by determining which edges have the most "edge-betweenness" and eliminating those edges. After those edges are the gone, the resulting components are the communities.

1. Find number of shortest paths and ordering by distance.



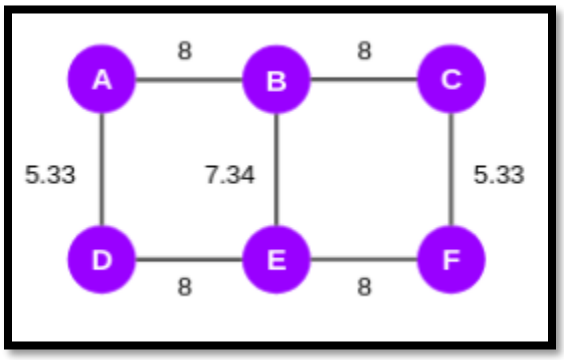

The "edge betweenness score" EBC, is calculated by finding the number of shortest paths from one node to every other node. The void number_of_shortest_paths(int src, Map& num_shortest_paths, Map& distance, std::list<std::pair<int,std::pair<int,int>>>& ordered_by_distance)const function calculates the number of shortest paths to all node from a source node. The values are stored with the node in the ordered_by_distance map and ordered by their distance from the src node.

2. Calculate EBC scores

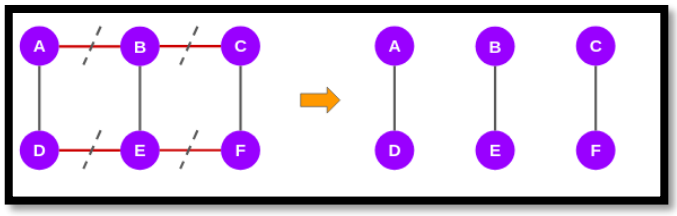


Next, the EBC is calculated for each edge by moving from the farthest from the source node backwards. The first layer of edges are calculated: if the edge is (u,v) → V(# of shortest paths) / U(#number of shortest paths). All other layers are calculated by adding the incoming flow from the farther nodes, adding 1 and dividing amongst all outgoing edges.

3. Repeat step 1 and 2 for all nodes as the source node add all EBC scores for each edge




4. Eliminated the edges with the largest EBC scores and continue until there are multiple components, these components are the communities.




## 3. EXTENDED IMPLEMENTATION

I chose the robust test cases for the extended implementation. The tests and outputs are shown below.

Basic number of shortest paths tests the helper function for find the number of shortest paths. The 5 tests are graphs increasing in size and complexity. The graphs are shown in the project_test_graphs.pdf file. The first and last graphs are of note because they show a balanced graph where 3 communities are created.

## 4. BUILD AND RUN INSTRUCTIONS

Exactly the same as the homework's.

## 5. REFLECTION

This algorithm was very hard. It took me many hours. Much more then any of the homework. This algorithm built upon the homework assignments and made a very cool algo. I ran into a problem with the edge values being integers, so I had to change all the code for adjacency lists to use doubles for the edge values. I was unable to do this with the adjacency matrix.

## 6. RESOURCES

[1] Girvan-Newman Algorithm

[2] Number of shortest path algorithm

## 7. REFERENCES

[1] https://www.baeldung.com/cs/graph-number-of-shortest-paths

[2] https://www.analyticsvidhya.com/blog/2020/04/community-detection-graphs-networks/