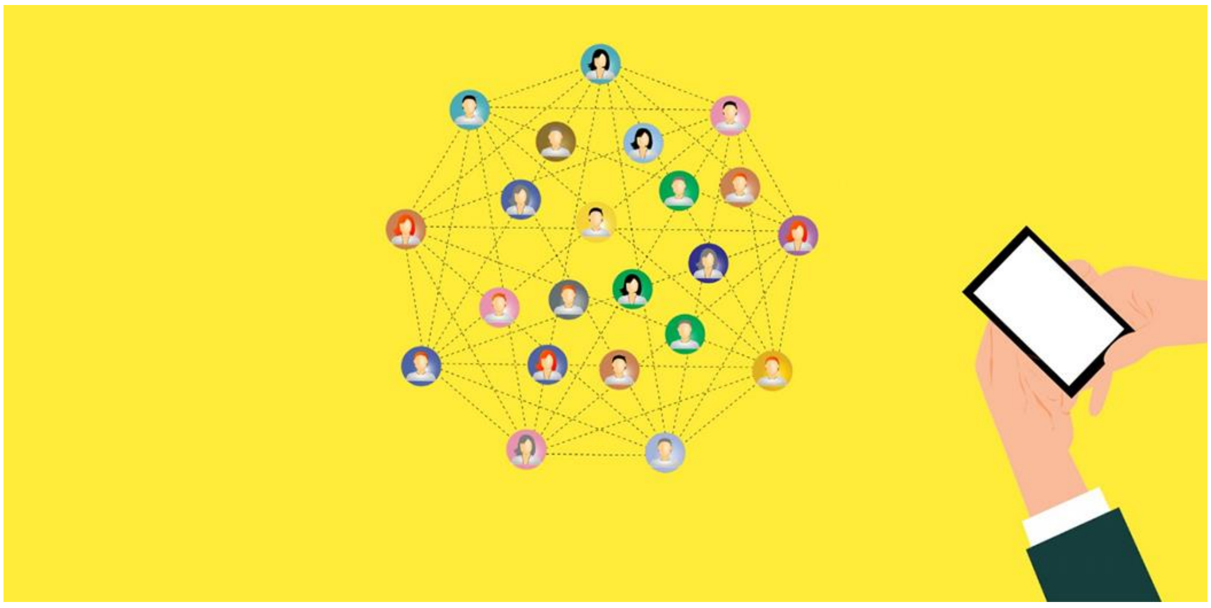
Getting Started with Community Detection in Graphs and Networks

Introduction

Networks surround us, perhaps in the form of your favourite social media hubs: Facebook, Instagram, Twitter, or the intricate web of stock exchanges where buying and selling intertwine. But networks don't just thrive in technology; they're woven into our daily lives. Communities are the heartbeat of these networks, where clusters of closely connected nodes form distinct groups. Picture your social sphere on platforms like Facebook or Instagram – interactions with friends, colleagues, and family – a tight-knit community within the digital world. Join us on an expedition into the realm of community detection, where hidden patterns shape our interconnected world.



What is a Community?

A community, with respect to graphs, can be defined as a subset of nodes that are densely connected to each other and loosely connected to the nodes in the other communities in the same graph.

Think about social media platforms such as Facebook, Instagram, or Twitter, where we try to connect with other people. Eventually, after a while, we end up being connected with people belonging to different social circles. These social circles can be a group of relatives, school mates, colleagues, etc.

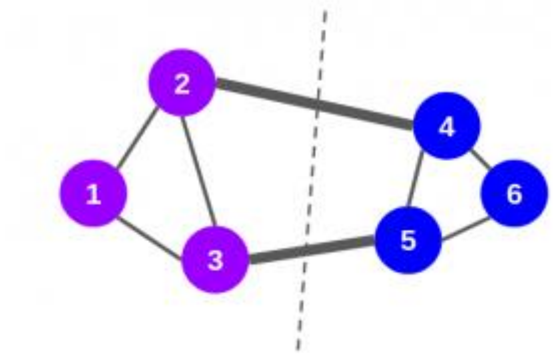
These social circles are nothing but communities!

What is Community Detection?

Community detection methods locate communities based on network structure, such as strongly connected groupings of nodes; however, they often ignore node properties. Nodes of similar types form a community in a network. Intra-community edges are the edges that connect the nodes in a community.

Detecting communities in a network is one of the most important tasks in network analysis. In a large scale network, such as an online social network, we could have millions of nodes and edges. Detecting communities in such networks becomes a herculean task.

Therefore, we need community detection algorithms that can partition the network into multiple communities.



Types of Community Detection

There are primarily two types of methods for detecting communities in graphs:

- Agglomerative Methods
- Divisive Methods

Agglomerative Methods

In agglomerative methods, we start with an empty graph that consists of nodes of the original graph but no edges. Next, the edges are added one-by-one to the graph, starting from “stronger” to “weaker” edges. This strength of the edge, or the weight of the edge, can be calculated in different ways.

Divisive Methods

In divisive methods, we go the other way round. We start with the complete graph and take off the edges iteratively. The edge with the highest weight is removed first. At every step, the edge-weight calculation is repeated, since the weight of the remaining edges changes after an edge is removed. After a certain number of steps, **we get clusters of densely connected nodes.**

In this article, we will cover the Girvan-Newman algorithm – an example of the divisive method.

Girvan-Newman Algorithm for Community Detection

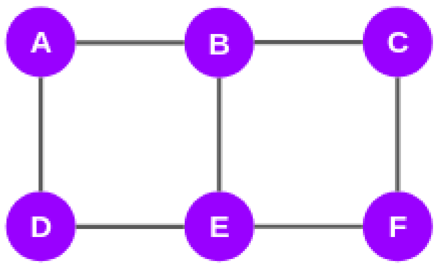
Under the Girvan-Newman algorithm, the communities in a graph are discovered by iteratively removing the edges of the graph, based on the edge betweenness centrality value.

The edge with the highest edge betweenness is removed first. We will cover this algorithm later in the article, but first, let’s understand the concept of “edge betweenness centrality”.

Edge Betweenness Centrality (EBC)

The edge betweenness centrality (EBC) can be defined as the number of shortest paths that pass through an edge in a network. Each and every edge is given an EBC score based on the shortest paths among all the nodes in the graph.

With respect to graphs and networks, the shortest path means the path between any two nodes covering the least amount of distance. Let’s take an example to find how EBC scores are calculated. Consider this graph below:

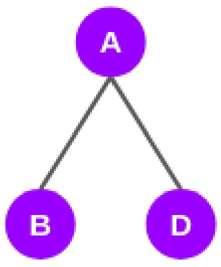


EBC Scores

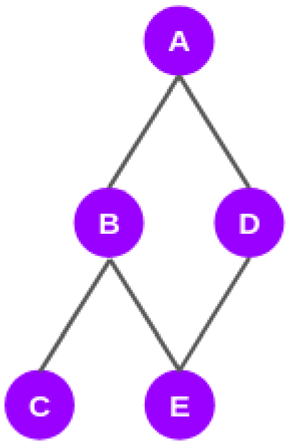
It has 6 nodes and 7 edges, right? Now, we will try to find the EBC scores for all the edges in this graph. Note that it is an iterative process and I've given an outline of it here:

- We will take one node at a time and plot the shortest paths to the other nodes from the selected node
- Based on the shortest paths, we will compute the EBC scores for all the edges
- We need to repeat this process for every node in the graph. As you can see, we have 6 nodes in the graph above. Therefore, there will be 6 iterations of this process
- This means every edge will get 6 scores. These scores will be added edge-wise
- Finally, the total score of each edge will be divided by 2 to get the EBC score

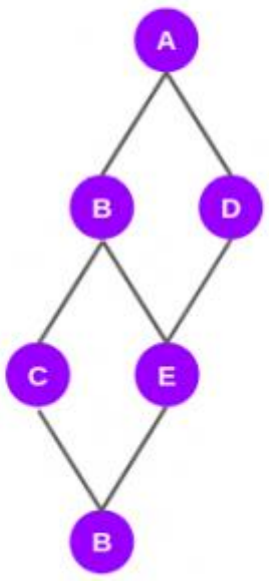
Now, let's start with node A. The directly connected nodes to node A are nodes B and D. So, the shortest paths to B and D from A are AB and AD respectively:



It turns out that the shortest paths to nodes C and E from A go through B and D:



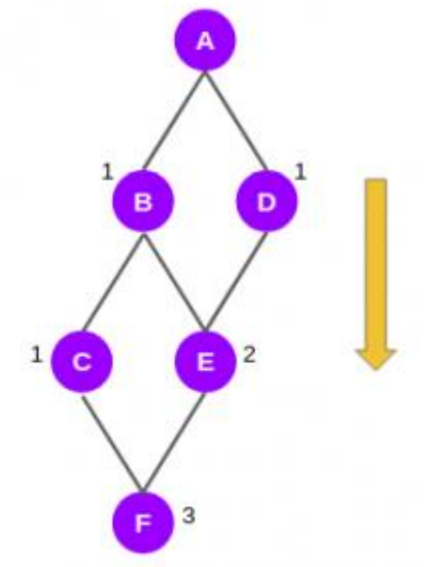
The shortest paths to the last node F from node A, pass through nodes B, D, C, and E:



The graph above depicts only the shortest paths from node A to all the other nodes. Now we will see how edges are scored.

Before giving scores to the edges, we will assign a score to the nodes in the shortest-path-graph. To assign these scores, we will have to traverse the graph from the root node, i.e., node A to the last node (node F).

Assigning Scores to Nodes



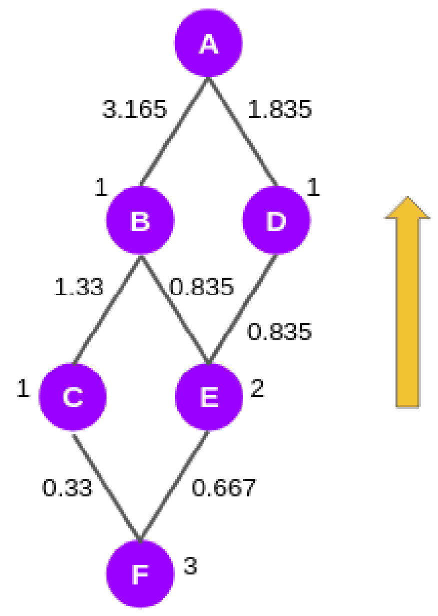
As you can see in the graph above, nodes B and D have been given a score of 1 each. This is because the shortest path to either node from node A is only one. For the very same reason, node C has been given a score of 1 as there is only one shortest path from node A to node C.

Moving on to node E. It is connected to node A through two shortest paths, ABE and ADE. Hence, it gets a score of 2.

The last node F is connected to A through three shortest paths — ABCF, ABEF, and ADEF. So, it gets a score of 3.

Computing Scores for Edges

Next, we will proceed with computing scores for the edges. Here we will move in the backward direction, from node F to node A:



FC = ⅓ = 0.33
FE = ⅔ = 0.667

CB = 1 + 0.33 = 1.33
EB = (1 + 0.667)/2 = 0.835
ED = (1 + 0.667)/2 = 0.835

BA = (1 + 1.33 + 0.835)/1 = 3.165
DA = (1 + 0.835)/1 = 1.835

Step 1

We first compute the score for the edges FC and FE. The edge score for edge FC is the ratio of the node scores of C and F, i.e. 1/3 or 0.33. Similarly, for FE the edge score is 2/3.

Step 2

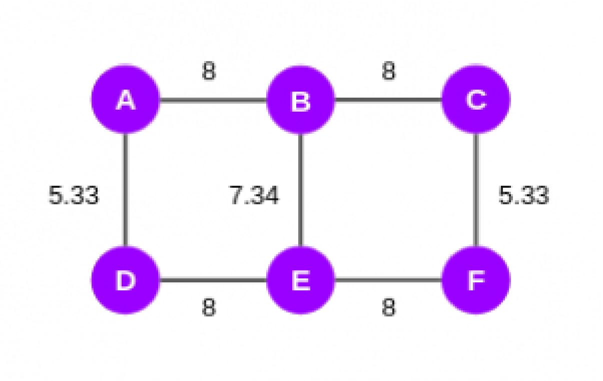
Now we have to calculate the edge score for the edges CB, EB, and ED. According to the Girvan-Newman algorithm, from this level onwards, every node will have a default value of 1 and the edge scores computed in the previous step will be added to this value.

So, the edge score of CB is (1 + 0.33)/1. Similarly, edge score EB or ED is (1 + 0.667)/2.

Step 3

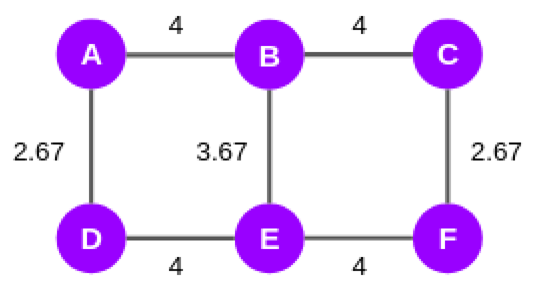
Then we move to the next level to calculate the edge scores for BA and DA.

So far, we have computed the edge scores of the shortest paths with respect to node A. We will have to repeat the same steps again from the other remaining five nodes. In the end, we will get a set of six scores for all the edges in the network. We will add these scores and assign them to the original graph as shown below:

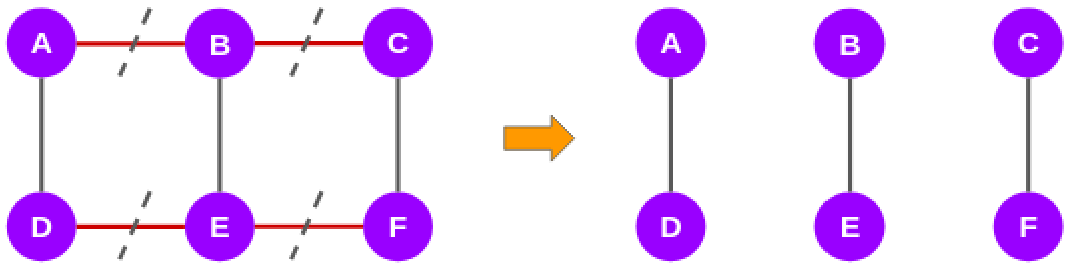


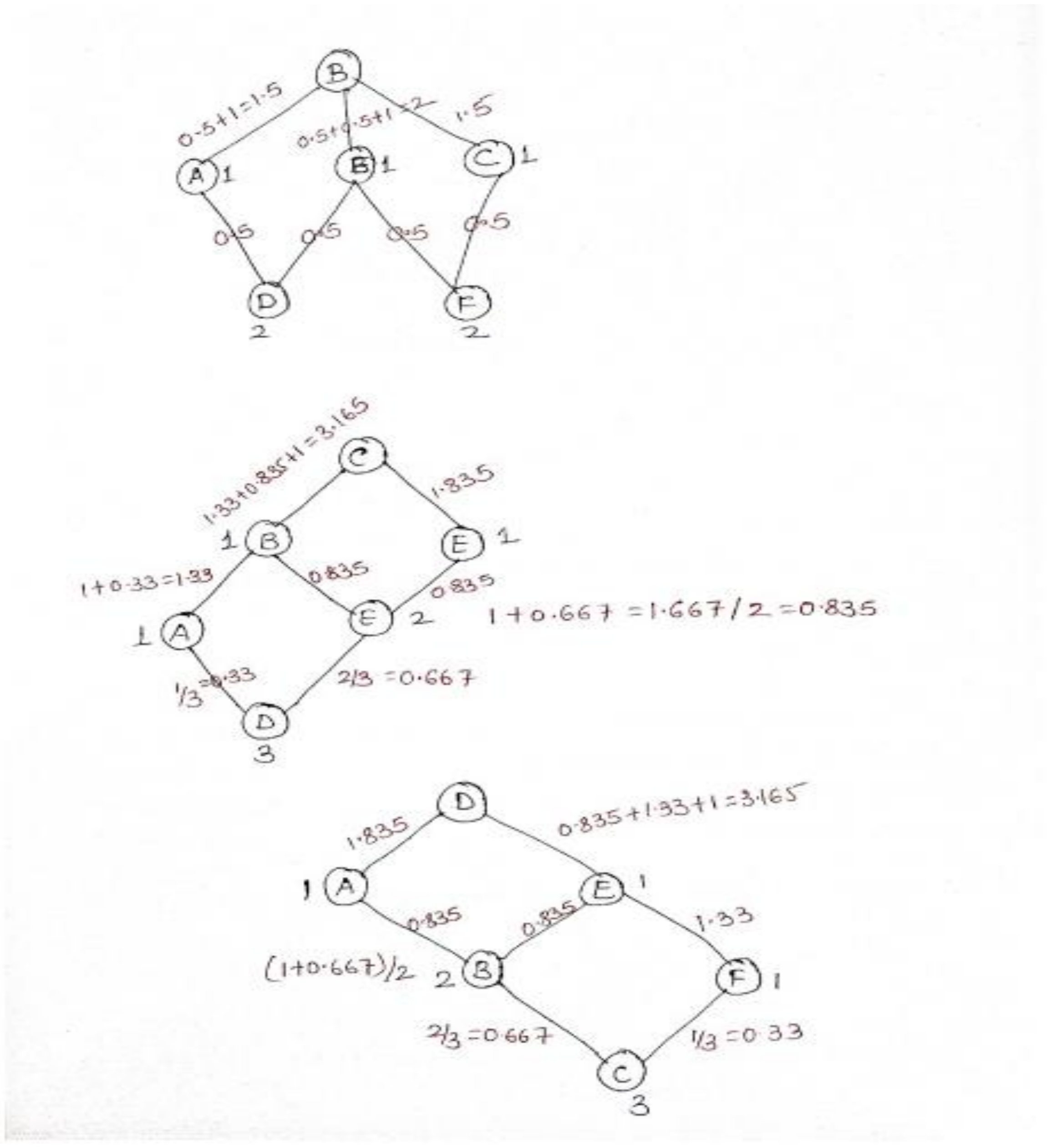
Step 4

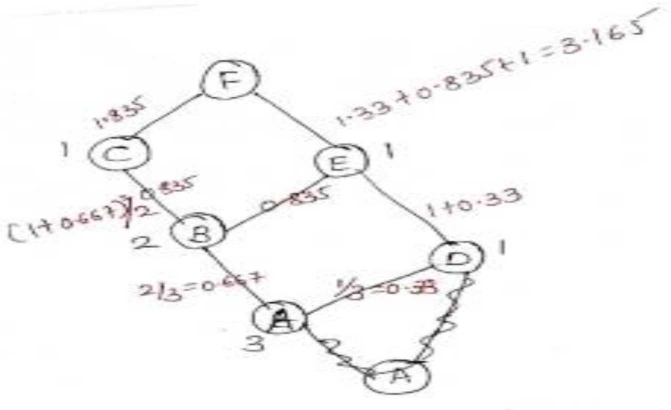
Since it is an undirected graph, we will divide these scores by two and finally, we will get the EBC scores:

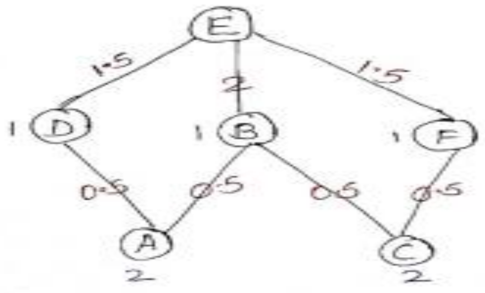


According to the Girvan-Newman algorithm, after computing the EBC scores, the edges with the highest scores will be taken off till the point the graph splits into two. So, in the graph above, we can see that the edges AB, BC, DE, and EF have the highest score, i.e., 4. We will strike off these edges and it gives us 3 subgraphs that we can call communities:


A
B
C
D
E
F
A
B
C
D
E
F


B
0.5+1=1.5
0.5+0.5+1=2
1.5
A 1
B 1
C 1
0.5
0.5
0.5
0.5
D
2
F
2
1.33+0.835+1=3.165
C
1.835
1 B
E 1
1+0.33=1.33
0.835
0.835
1 A
E 2
1+0.667 =1.667/2 =0.835
1/3=0.33
2/3 =0.667
D
3
D
1.835
0.835+1.33+1=3.165
1 A
E 1
0.835
0.835
1.33
(1+0.667)/2
2 B
F 1
2/3 =0.667
1/3 =0.33
C
3





EDGES	EDGE BETWEENESS
AB	3·165+1·5+1·33+0·835+0·5+0·667=8
AD	1·835+0·5+0·33+1·835+0·5+0·33=5·33
BC	3·165+1·5+1·33+0·835+0·5+0·667=8
BE	0·835+2+0·835+0·835+2+0·835=7·34
CF	1·835+0·5+0·33+1·835+0·5+0·33 = 5·33
DE	3·165+1·5+1·33+0·835+0·5+0·667 = 8
EF	3·165+1·5+1·33+0·835+0·5+0·667 = 8