

Social Networks
Prof. S. R. S. Iyengar
Department of Computer Science
Indian Institute of Technology, Ropar

Lecture - 36
Strong and Weak Relationships
Community Detection Using
Girvan Newman Algorithm

Hey everyone, in the previous video we divided the graph into two communities using brute force approach. Brute force approach means we tried all possible divisions of nodes into two communities which was very time-consuming process. So, if the graph is very huge it is going to take whole lot of time and incase of very you know large graphs it might not even run it my get stuck. So, that was not a feasible way to find communities in a given graph.

On the other hand we have this algorithm which is called Girvan Newman Algorithm and this algorithm is basically base on the concept of edge betweenness. Now the key idea is that the edge is which are inter community that is the edge is which are you know linking the nodes from one community to the other community, they tend to have high value of betweenness now; that means, that number of shortest pass through these edges. So, if variable find the edge betweenness of these edges and we keep removing them we tend to get the community structure of the graph. So, this is the whole idea behind this algorithm and this is what we are going to implement and after that will try this algorithm on the sample graph that we took in previous video that is bubble graph, and after that will also execute this algorithm on Zachary karate club as well and will see the communities that get formed.

(Refer Slide Time: 01:55)

```
1 import networkx as nx
2
3 def edge_to_remove(G):
4     dict1 = nx.edge_betweenness centrality(G)
5     list_of_tuples = dict1.items()
6     list_of_tuples.sort(key = lambda x:x[1], reverse = True)
7     return list_of_tuples[0][0] #(a,b)
8
9
10 def girvan(G):
11     c = nx.connected_component_subgraphs(G)
12     l = len(c)
13     print 'The number of connected components are ', l
14
15     while(l != 1):
16         G.remove_edge(*edge_to_remove(G)) #(a,b) --> (a,b)
17         c = nx.connected_component_subgraphs(G)
18         l = len(c)
19         print 'The number of connected components are ', l
20
21     return c
22
23 G = nx.barbell_graph(5,0)
24 c = girvan(G)
```

So, let us get started. So, I am going to import this package import networkx as nx and let us create a function for Girvan algorithm. So, I will write Girvan G. So, let us first of all check whether the graph is connected or not. So, I will write nx.connected. I am sorry connected component sorry component sub graphs G. So, what is function does connected component sub graphs this function returns the connected component of the graph as sub graphs. So, if there are two connected components in the graph G it will return two sub graphs. If there is only one connected component that is graph is connected it is going to return only one sub graph and that is graphic itself.

So, let us store that in this list c. So, if the length of this list is one; that means, graph is connected right because it returns only one connected component, and if the length of the list is more than one that means, the graph is disconnected and there are more than one connected components right. So, let us store the length of this list c and let us print that because we are going to keep checking whether the graph is now connected or not. So, we are going to make use of this function again and just to keep track of algorithm we are going to print whether the graph is connected or not at that particular moment.

So, I am going to print the number of connected components is. So, write l. So, l is number of connected components. So, initially it will be one if the graph is connected of course. Now as I told you we will keep removing the edges from this graph G. So, the

function that we will use to remove the edges from the graph is `G.remove_edge`, and as the parameter we will pass the edge that has to be removed basically we will pass two parameters first is the source of the edge and second is the target of the edge that has to be removed.

Now, the big question is which edge to remove right as I told you we are going to remove the edges based on the betweenness value. So, we have to calculate the betweenness value of the edges first, now since you have to keep removing the edges I am going to keep that code in a function. So, let me better create a function here, I will create a function say edge to remove I will pass the graph `G` here. So, let us first create this function, edge to remove and let us pass the graph here edge to remove.

Now, in order to be able to know which edge to remove we have to first calculate the edge betweenness values, and for that we have a function `nx.edge_betweenness centrality` or of `G`. So, `edge betweenness centrality` is a function in `networkx` which returns a dictionary of values. So, I am going to name it like `dict1`. So, that you know the data structure of the return value. So, what does this dictionary contain? This dictionary contains as a key the edge and as a value it contains the edge betweenness centrality of that edge correct. So, that is what this dictionary contains.

Now, what is our aim? Our aim is to sort the edges based on their betweenness centrality value. So, since dictionary does not keep track of the sorted elements basically there is no association between the element and the index. So, the elements can appear in any order. So, order is not a thing of dictionary. So, let us take the values from this dictionary and keep them in the form of a list. So, what I will do is I will write `dict1.items` and I will store these in the form of list of tuples and going to give similar name to it. So, that you it is easy of your list of tuples is equal to `dict1.items`.

So, you might be knowing that these `items` function basically returns the tuples of the dictionary, where the first element of each tuple is a key and second element of each tuple is the value. So, this list is going to have tuples where the first element will be the edge of a each tuple and the second element of each tuple is going to be the edge betweenness value. Now make sense for us to sort this list as we know sort this list based on the edge betweenness value right now how we do that let me show you list of tuples dot sort this is the function that we will use to sort this list. Now the list is having tuples

and tuples have two values, second value is the edge betweenness value and based on that only we must sort the list. So, whenever we must sort list of tuples there is syntax that we follow let me show you that, `key = lambda x : x[1]`.

Now, let me tell you why I write `x : x[1]` and one more thing `reverse` is equal to `true`. So, this is how we sort list of tuples you can see the documentation of this `lambda` key word and. So, this is basically a shortcut to sort list based on either you know first value of the tuple or the second value of the tuple. So, here we want to sort based on second value of the tuples. So, I am writing `x : x[1]`, in case you want to sort the list based on the first value of the tuple you will write `x : x[0]`. Second thing is we want in the descending order that is we want the edge which has highest betweenness to be on the top. So, I will write `reverse` is equal to `true` in case you want opposite you do not need write anything here.

So, we have now the list which is sorted. So, the first element of the list is the tuple, where the first element of the tuple is the edge and the second element of the tuple is the betweenness value. What do we have to return here by this function? This function should return the edge right. So, what we are going to return is list of tuples first I am sorry first element first tuples first element which is 0 because the index start from 0 I hope I am clear here 0th with element means 0th tuple and in that tuple 0th element which is the first element that is the edge.

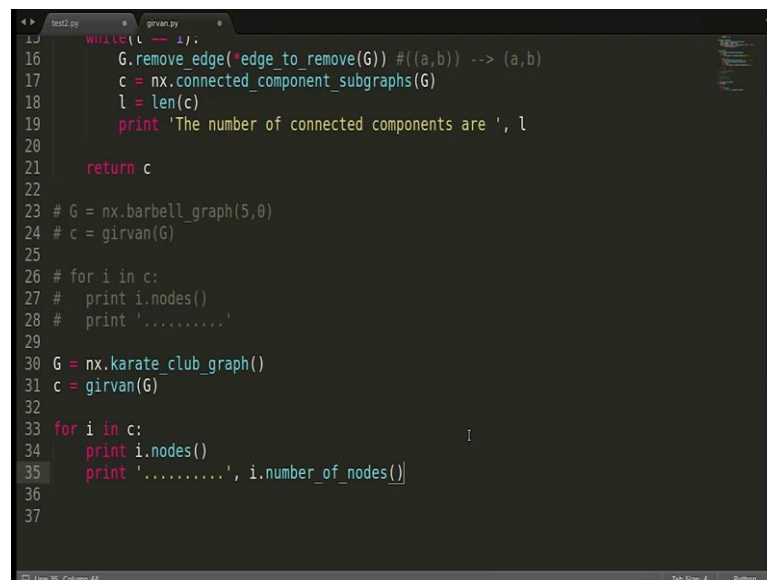
So, we are going to return this alright I am sorry return this, alright. So, you know which edge to remove now, there is one catch here which should be noted this edge to remove is going to return tuple right let me write here as a comment is going to return the edge which is of the form `a comma b` where `a` is the source and `b` is target . So, when we pass it here it becomes like this `a comma b` right this becomes this comes over here instead of this which will give you syntax error. So, what is needed is we instead of this we want this thing `a comma b` I am sorry `a comma b`, basically out of the tuple we have to extract the edge. So, what we can put here is `star`, when we write `star` here, we get the content of the tuple and not the tuple ok.

So, we now we got the edge which has to be removed after we remove the edge from the graph, we again have to check whether the graph is corrected or not right. So, again we will make use of this function and again we will find the length and again we make print whether the as in what are the number of connected components. So, I am going to use it

here, we have to keep removing the edges until the graph becomes disconnected. So, which means we have to start while loop here. So, I will write while l is equal to 1 which means as long as the graph is connected keep doing this and as soon as this l becomes 2 right l becomes 2 means the graph becomes disconnected and number of connected component in the graph is 2 the control will come out this loop and here we have two connected components correct.

So, and these two connected components are there in this list c which is what we are going to return here. So, will write return c. So, we are going to return this c and we will see in while we are calling it we will see how we can extract the connected component out of this c. So, I think we are done with the functions let us try to make use of them, let us take the example graph that we took in previous video nx.barbell_graph with the same values I will take as a from 5 , 0 and I will pass them in this function, and let me take the return values now c is going to have the two connected components right. So, and these two connected components in the form of graph which means we can apply all the functions of graph on these connected components because they are actually graph objects.

(Refer Slide Time: 12:45)



```
16 def girvan(G):
17     G.remove_edge(*edge_to_remove(G)) #((a,b)) --> (a,b)
18     c = nx.connected_component_subgraphs(G)
19     l = len(c)
20     print 'The number of connected components are ', l
21     return c
22
23 # G = nx.barbell_graph(5,0)
24 # c = girvan(G)
25
26 # for i in c:
27 #     print i.nodes()
28 #     print '.....'
29
30 G = nx.karate_club_graph()
31 c = girvan(G)
32
33 for i in c:
34     print i.nodes()
35     print '.....', i.number_of_nodes()
36
37
```

So, I can start to look here, or I can manually do that as well. So, I will write for i in c. So, c was less which has two elements. So, I will write for i and c let me. So, what is our aim are our is to print the nodes which are falling in the first community and in the

second community, community means connected component here right. So, let us print i because i is a graph I can use i.nodes and that is all. So, i goes from 0 to 1 and both the connected components nodes will be printed here let me put some separately here something anything.

(Refer Slide Time: 13:39)

```
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_3_(Strength of weak tie
s)/Videos$ python girvan.py
The number of connected components are 1
The number of connected components are 2
[0, 1, 2, 3, 4]
.....
[8, 9, 5, 6, 7]      I
.....
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_3_(Strength of weak tie
s)/Videos$
```

So, we should get the two connected component the nodes of the two connected components by this let us see, let us check the program what was the name yeah right. So, you see the number of connected components initially was one, and then we removed some edge and the number of connected components became 2.

So, it was very small graph. So, quickly it conversed as we as see how quickly it conversed as compared to the brute force method. In that method we it took good amount of iteration to finally, give us the communities. Here it quickly given as the communities and as we saw in the previous video the communities were 0 1 2 3 4 in the first one and 8 9 5 6 7 in the second community. So, it is giving us correctly the communities in the given graph let us go back to our code and we can also check the karate clubs current communities here ok.

So, let us check this code for the Zachary karate club as well, let me copy paste this and let me common this. So, let us takes the Zachary karate club. So, this is the function that we can use karate club graph. Let me tell you can also follow another method to get the is Zachary karate club in the graph object which I think we used in the previous video

you if you have the karate dot gml or karate club graph in any format you can just read that graph file and then that is how you can you know convert that into graph object.

(Refer Slide Time: 15:43)

```
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_3_(Strength of weak tie
s)/Videos$ python girvan.py
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 2
[32, 33, 2, 8, 9, 14, 15, 18, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
.....
[0, 1, 3, 4, 5, 6, 7, 10, 11, 12, 13, 16, 17, 19, 21]
.....
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_3_(Strength of weak tie
s)/Videos$
```

So, you can use that, or you can use this inbuilt function which is given by networkx. So, we are going to run this again on this graph let us see, let me its let me show you. So, you see the number of connective components in initially was one and then it removed some good number of edges based on the betweenness until it reached to the two connective components part, and this is these are the nodes in the first community and these are the notes in the second community, and yeah I think these are the these are the nodes which actually constitute the two communities in the karate club network, you can also verify you can google and you can see the communities that would actually formed in the karate club network.

So, this given (Refer Time: 16:26) is doing good job of giving the communities here, you can also see the number of nodes if you want to print the number of nodes in each community let us do that. So, as I told you this is i is a graph. So, you can you know use any graph function here number of nodes it should work.

(Refer Slide Time: 16:51)

```
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_3_(Strength of weak tie
s)/Videos$ python girvan.py
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 1
The number of connected components are 2
[32, 33, 2, 8, 9, 14, 15, 18, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
..... 19
[0, 1, 3, 4, 5, 6, 7, 10, 11, 12, 13, 16, 17, 19, 21]
..... 15
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_3_(Strength of weak tie
s)/Videos$
```

So, I want to see the number of nodes in each community. So, there are 19 and 15, I think if I am not wrong there were 16 or 18 or something, but nevertheless this is pretty good approximation of finding the communities in the given graph.