Complex Network: Theory and Application
Prof. Animesh Mukherjee
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

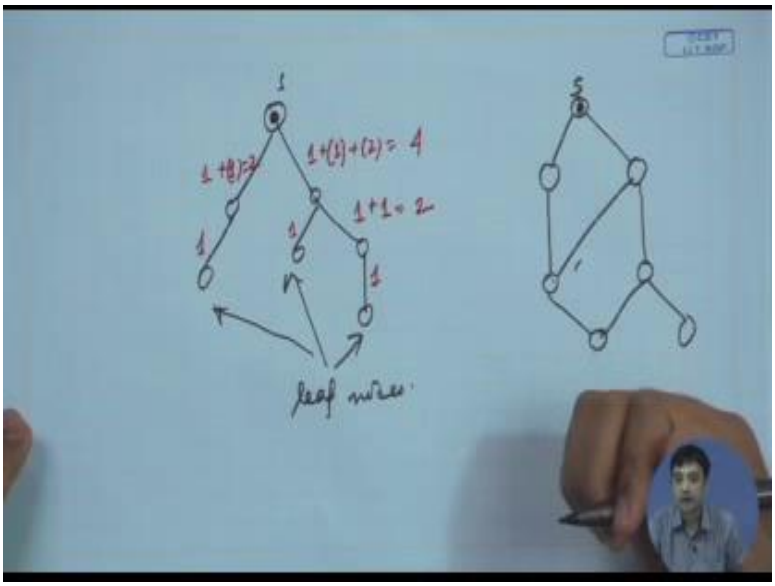
Lecture – 14
Community Analysis – III

Welcome back. So, in this last lecture we have started looking into this idea of doing a bisection of the graph in order to find out community structure, and we have been proposing this method of edge betweenness in order to execute this bisection.

So, and we said that it is a very compute intensive process and therefore, we were trying to look into methods in which one can reduce the computation over to some extent, and so for that we introduce this idea of having single source short dispatch identification using BFS and we saw in the simple case where you have BFS tree, a shortest path tree is not very difficult to find out the path counts passing through every edge and from that to have a aggregate view of the total edge betweenness of every individual as edge.

Now, we say that we will talk about the generalised version of this in this current lecture. So, again if you look at this previous example, we had a graph snapshot like this.

(Refer Slide Time: 01:29)



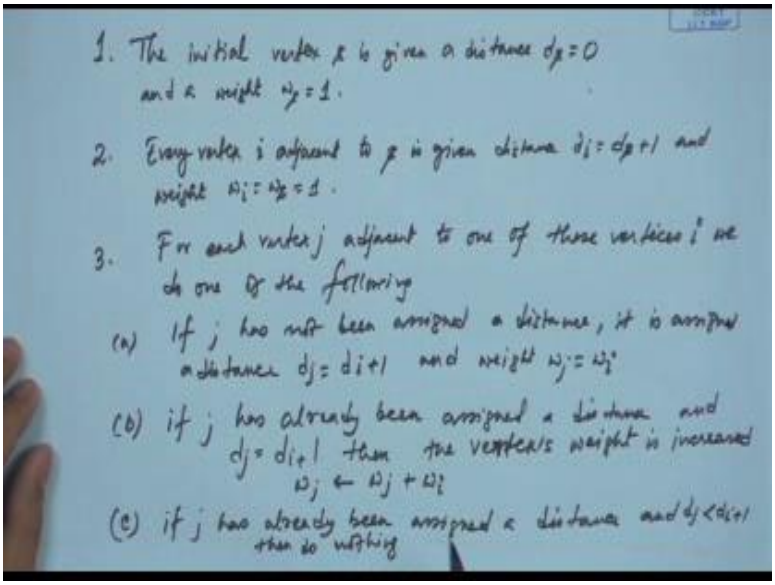
Which was just like a shortest path tree and therefore, it was very easy for us to calculate the comp edge betweenness components where every individual edge, but imagine if we

had a slightly more general case where it is no longer a shortest path tree, but a shortest path graph basically, which means that this implies that from a sources there can be multiple shortest paths to other nodes in the graph. So, we will take a small example again.

Suppose, if I modify the previous example into something like this. So, now, there are at least two shortest paths from S to this particular node and therefore, we have to identify then. So, also if to this node you have to shortest paths. Therefore, we have to identify like a mechanism to estimate the path counts, the promotional path counts or the promotional edge betweenness contribution of each individual edge in this generalise settings.

So, we will first discuss the algorithm and then we will execute it on this particular graph and see how things work. So, it is a very simple variation of the straight forward BFS algorithm that we have already come across in undergraduate classes.

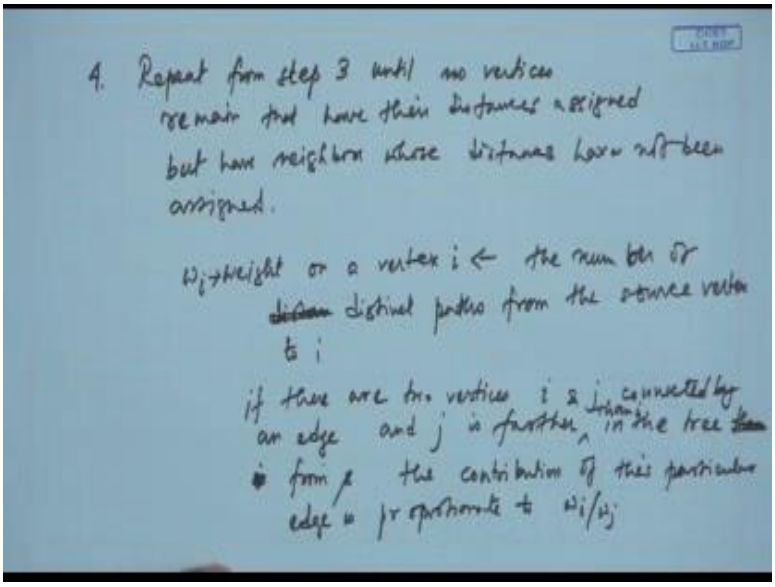
(Refer Slide Time: 03:07)



The first step is as follows the initial vertex s is given are distance ds equal to 0 and a weight ws equal to 1. Step 2; every vertex i adjacent to s is given distance di is equal to ds plus 1 and weight wi is equal to ws is equal to 1. Now, then comes the most important step for each vertex j adjacent to one of these vertices i, we do one of the following; if j has not been assigned a distance that is j has not been explored distance it is assigned a distance dj is equal to di plus 1 and weight wj is equal to wi.

However, if j has already been assigned a distance that is, it has been already explored and it is such that dj is equal to di plus 1 then the vertices weight is increased that is there is a. So, these actually confirm that there are actually two shortest paths leading from that particular vertex. So, that is why we increase the weight of that vertex to whatever the weight of the vertex was plus wi. There could be a third case, if j has already been assigned a distance and dj is less than di plus 1 then do nothing, that is we are probably back tracking. So, that is why we do not do anything; now, the last step.

(Refer Slide Time: 07:04)



Step 4, repeat from step 3 until no vertices remain that have their distances assigned, but have neighbours whose distances have not been assigned.

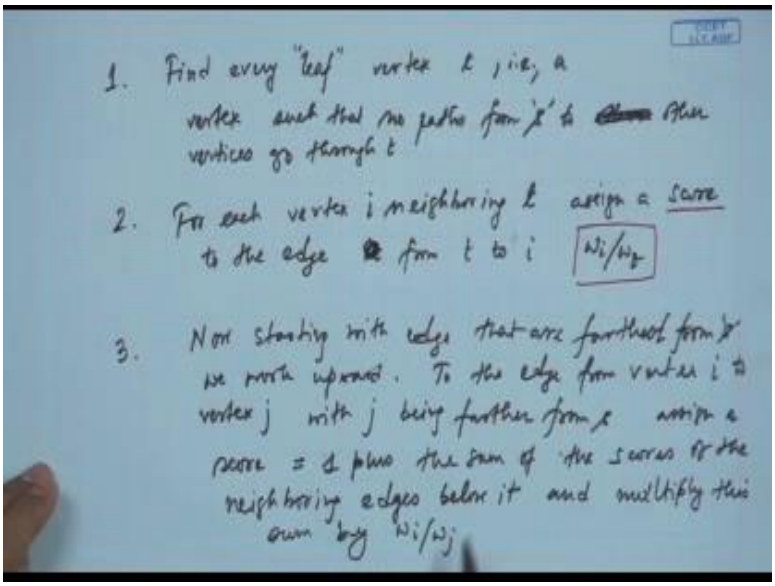
So, you repeat this process until and unless all the vertices and its neighbours have been assigned a distance as well as the weight. So, basically this entire algorithm can be actually implemented using a queue using a queue structure just like BFS, now what does the how do we interpret the weight. So, what is the weight? So, weight on vertex i what does this indicate this is the number of distinct as I was telling you this is the number of distinct paths from the source vertex to i. So, basically that is being counted in the variable wi. So, the weight of the vertex i wi, which is the weight of the vertex i is nothing, but the number of distinct shortest paths from the source to the node i.

Now, how do we calculate the contribution of edge betweenness? So, this can be done very easily. So, what you do is if there are two vertices i and j connected by an edge and

j is further in the tree than i. So, j is further in the tree from s, j is actually further then i in the tree from s. If this is the situation that j is from s if you start from s j is further and i is nearer and there is an edge between i and direct edge between i and j then the contribution of this particular edge is proportion at to wi by wj.

So, basically what you find out is, what is the fraction that is going through i and out of which how much is going through j basically? So, that is actually expressed by this fraction wi by wj. So, therefore, we can again write down how to calculate the edge betweenness from the shortest paths counts using the following three step algorithm. So, the first step of the algorithm is finding every 'leaf' vertex.

(Refer Slide Time: 11:35)



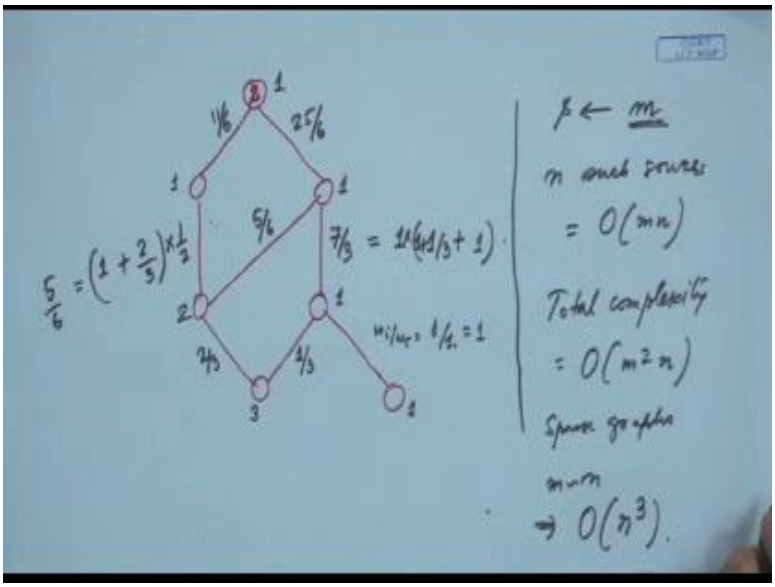
As we have done in the case of the shortest path tree, every leaf vertex tree that is a vertex such that no paths from s to other vertices no path from the source s to other vertices go through t. The next step is for each vertex i neighbouring t assign a score to the edge it to the edge actually you can write from to the edge from t to i wi by wt. So, this is the weight you assign for each vertex i neighbouring to t assign a score. So, using this score we will calculate the edge betweenness you assign a score of wi by wt. So, now, then you have to work upwards as we did in the earlier case.

Now, starting with edges that are farthest from s, we work upwards. To the edge from vertex i to vertex j with j being further from s assign a score equal to one plus you assign the score one plus as you are doing in the last case one plus the sum of the scores of the

neighbouring edges below it and multiply this sum by wi by wj. So, this is your making it proportion it to the number of path counts that passes through that particular edge ij.

So, given the previous example, we will now see how to calculate the edge betweenness.

(Refer Slide Time: 15:31)



So, we have the source node s. So, this is the example that we looked in. So, then what as per the algorithm what we will do. So, we still have the distance and so, we will initially assign the weights. So, the weight here is 1, the weight here is also 1, the weight here is 1, the weight here becomes 2, the weight here is 1, the weight here is 300 and the weight here is 1. So, this is the first part of the algorithm through which we calculate, we said the distances and the weights on the vertices.

So, once we have the weights on the vertices set, now we will have to calculate the contribution of the edge betweenness factors. So, for this particular edge you have. So, this is a because this node is a leaf node this edge actually will be wi by wj which is nothing. So, this is the leaf node. So, it is t. So, and this is another internal node i. So, the fraction here will be nothing, but wi by wt which is equal to 1 by 1 in this case that is equal to 1.

Now, for this particular example it is wi by wt which is nothing, but 1 by 3 for this particular edge you again have wi by wt in this particular case it is 2 by 3 in this way now you have to work upwards. So, for instance let us take this particular edge. So, this

is you have 1 plus coming from bottom is 2 by 3 and this is multiplied by 1 by 2 this actually is equal to 5 by 6 in this way you can calculate the contributions of each of the edges just by following the second part of the algorithm and you can verify that we will get contribution for each of the edge as we have written here. So, so in each of this it is very, very simple state. So, these 7 by 3, again we can repeat this is equal to 1 plus what is coming from bottom, there is 1 from this edge contribution that is 1 by 3 plus, this is coming there is one coming from here.

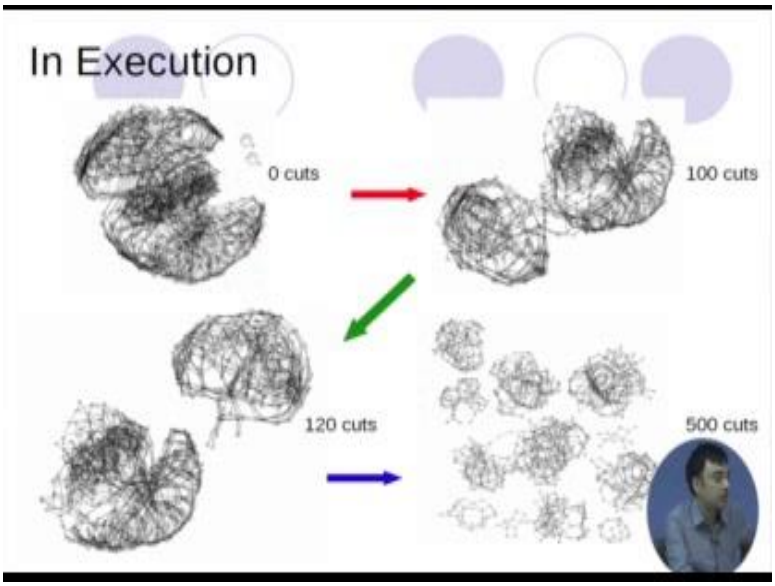
So, in this way you can calculate the contributions from all the different edges that lie below it and from that you can construct the contribution sorry, this is 1 plus 1 plus 1 by 3 and multiplied by 1 by 1. So, that is 1. So, this gives you 7 by 3. So, in this way you can find out the contribution of each individual way each individual edge towards the edge betweenness. So, this is for single source now you can consider every individual node as a single source and from there you can calculate you can repeat this process and find out the total contribution of edge betweenness of each individual edge ok

So, by this you see that we have actually reduced the complexity of the algorithm by at least one order of n, why because now from s from every individual s you can calculate the shortest paths to all other nodes in the fs time, which is equivalent to the number of edges now you have to do it. So, there are n such sources. So, the complexity becomes order m into n because for every single source you need m time. There are n such sources in the network if there are n nodes in the network that will give you order mn and now you have to execute this for every individual edge until and unless all the edges have been removed.

So, the total complexity is order of m square n. Now, again for sparse graphs as we know we can assume that m is very similar to n that implies the total complexity of the algorithm is order n cube which is at least one order less than the naive version of the approach.

In this way, actually Newman and Girvan were successful in computing the community structure of various real world networks. Again, one of the examples in execution, they started with this particular networks here if you look at the slides. So, start with this particular network.

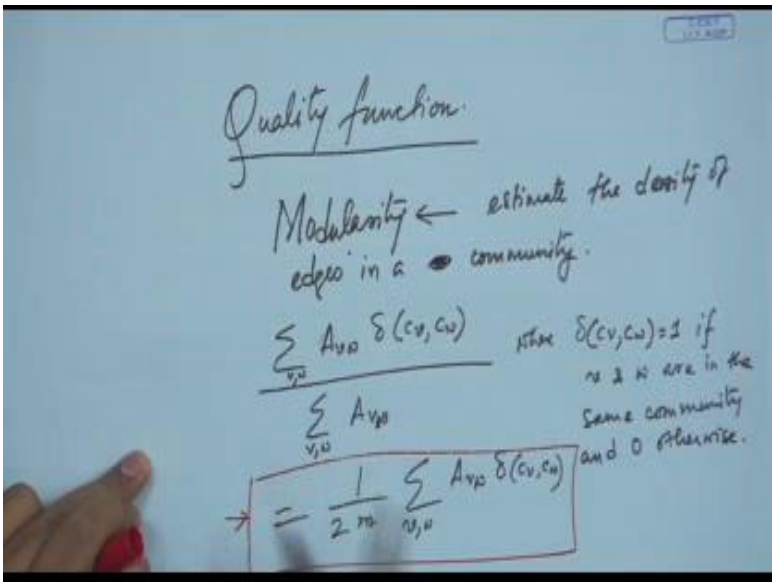
(Refer Slide Time: 21:15)



So, and then after 100, when there are no cuts in the network this is how the networks look after 100 edges have been removed, 100 edges with top betweenness and treaty have been removed the network looks like this then there are 120 cuts you already see that there are at least two components and then from there if you have even more like 500 cuts, you see that there are small sub modules that get generated and each of this are much more dense sub modules which can be thought of as correlates of the community structure of the network. So, in this way you can actually envisage to construct the community structure of various real world networks using this bisective method.

Now, since we have learnt quite a few techniques to identify the community structure of networks, the next obvious question is like once we have got this community structure how do we test that whatever we have got is good partitioning that we have made is a good partition. How do, when do we call a partition good partition? So, basically in order to do this, in order to identify if a partition is a good partition, we need to define the notion of something called the quality function.

(Refer Slide Time: 23:00)



So, quality function can be defined is in various different ways, but one of the earliest ways and one of the very successful and popular ways in which this was defined was by again Girvan and Newman in 2004 in one of their papers, where they defined the concept of modularity which is actually, basically it tries to estimate the density of edges in a community. So, it tries to estimate the density of the number of edges within a community.

So, that is the basic intuition and if given a community there are large number of internal edges inside the community; that means that this is a good partition in that is people who are more densely connected. You have been able to successfully partition them into sub structure that is the idea. So, if we try to write the formula it is a very simple formula. We start with the basic notion and then we see what could go wrong and then we will correct the formula. So, basically as I said it measures the density of the edges inside the community.

So, this is how we can try to define the notion of modularity. So, what you write at the denominator is the sum of the total number of edges in the numerator you have in the numerator, you have you see if there is an edge between a particular pair of vertices v and w and you also check if v and w are part of the same community. So, delta is a indicator function which tells you if delta cv cw are in the same community if v and w are in the same community then delta cv cw is equal to 1 otherwise it is 0.

So, basically what you try to count here is, what is the number of edges that are present between the nodes inside the community and you submit overall pair of nodes and that you normalise by the total number of edges in the network? So, this translates to 1 by 2 m sum of v w a v w delta cv cw.

This works as, where m is the total number. We are assuming that m is the total number of edges in the network. So, then if you look at this formula you immediately find that there is a problem hidden there. So, basically we are trying to find out the quality of a partition and we are trying to find out the density of the edges inside that partition and that would be an indicator of its quality of its goodness quality, but then if you have this particular formula what would happen is that you can actually include the entire network into a single community and then you will have these values equal to 1.

So, you can make artificially make the modularity equal to 1, if you have all the nodes of the networks in a single community. So, this is a very big problem big glitch of this formula and we have to somehow this problem.

In the next lecture we will see how we actually adapt this formula in order to take care of this particular problem and have a final formula which actually works well as quality function.