Complex Network: Theory and Application

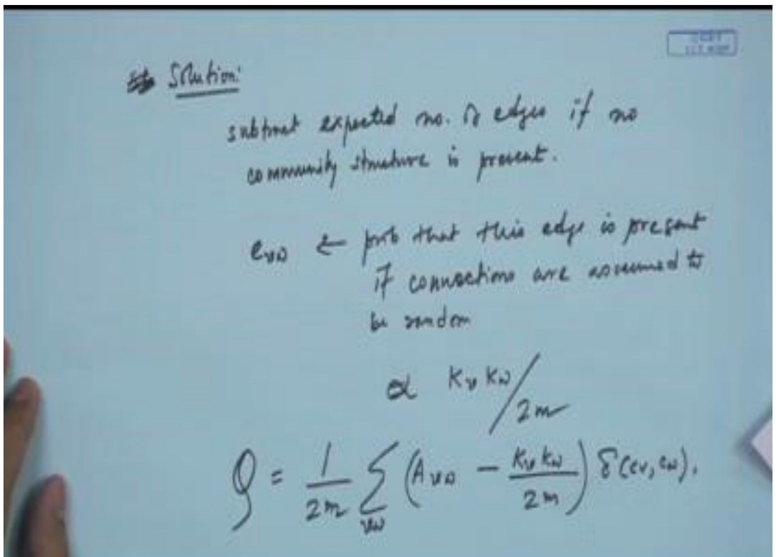
Prof. Animesh Mukherjee
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 15
Community Analysis – IV

In the last lecture I defined the notion of quality function and I told you that modularity can be thought of as a way of identifying the quality of a given partition and then, we were trying to define the modularity. So, in order to do that what we did was we expressed the total number of edges that are represent within communities as a fraction of the total number of edges in the network, but then we saw that this could be problematic. Why? It is because then you can assume the whole network as a single community and your modularity will get to 1. So, therefore, you do not get to it as the actual quality of the partitions.

In order to overcome this problem, what we try to do is again we use a similar concept as is done while estimating correlations or mixing coefficients. You try to remove the part that is expected, you try to discount the part that happens just by chance the expected number of connections between communities that could happen just by random chance. So, basically what we will try to do is, we will have a solution proposed like this.

(Refer Slide Time: 01:39)



Solution:

subtract expected no. of edges if no community structure is present.

e_{vw} ← prob that this edge is present if connections are assumed to be random

$\propto k_v k_w / 2m$

$$Q = \frac{1}{2m} \sum_{vw} \left(A_{vw} - \frac{k_v k_w}{2m} \right) \delta(c_v, c_w).$$

So, the solution is subtracted expected number of edges if no community structure is present. So, we try to estimate even if there is no community structure present. What is the probability that there could be an edge between a pair of nodes or they fall in a partition? So, suppose you have an edge E vw. So, now the probability that this edge exists, edge is present if connections are assumed to be random. So, we assume that you take every pair of nodes and the connection between them is just a random variable. So, the connections are just by chance.

In such a case, the probability that there will be an edge between a pair of nodes is nothing, but will be proportional to k v k w normalized by the total number of edges in the network. So, basically if there are two nodes with certain degree values, then independent of anything else, the probability that they will have an edge between them. If connections are completely random in the graph depends on their degree only and the probabilities just the normalized degree normalized by the role number of edges in the network.

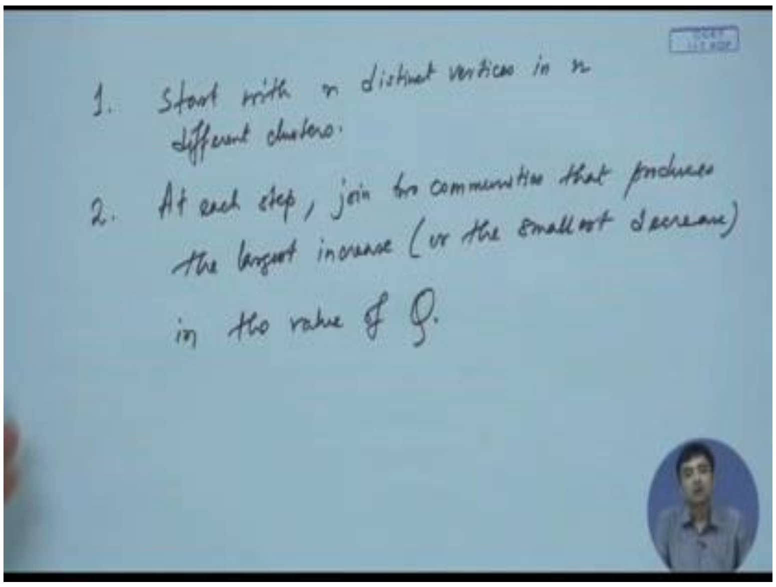
They are normalized by actually the total degree. Basically you have the probability of assuming that there is an edge between a pair of nodes is equivalent to the product of their degrees normalized by the total degree. So, you basically discount this part. So, we assume that by random chance if there is an edge between a pair of nodes, then that might be problematic in estimating the modularity correctly. So, even within a partition, there might be an edge by chance. So, we remove this chance factor from the definition of modularity and therefore, we have the revised definition as follows.

The modularity function q is nothing, but one by twice m as we have already seen a v w that is if there is an edge existing by disc with a discount. Now, if there is an edge existing by random chance over all v w delta cv cw, so we have now used this discounting factor by means of which we are removing that part which is we are subtracting that part which can be expected just by random chance. So, in that way we have now removed the original glitch in the definition.

Now, you see if you include the whole network that would not give you the best quality function that would not give you the best quality induction in indication rather if you actually divide it into nice sub modules where each of the sub modules are much more densely connected, then you will have a larger modularity value. So, given this quality function now, you see like if you have this quality function, these actually also tells you that how good a partition is now you can try to therefore also optimize this function and have community structures.

So, any quality function that way can also be used as a metric to identify community structures. So, as we have defined the community, the quality function in originally quality functions were used mostly to identify the goodness of a partition, but then since they are testing the goodness, they themselves can also be used as an optimization criteria for identifying good communities structures and that was the idea of the algorithm that was proposed by Newman and more in one of their early works, where they actually used the modularity function in order to identify communities structures in a graph and this is a very simple two step algorithm as follows.

(Refer Slide Time: 06:59)



1. Start with n distinct vertices in n different clusters.
2. At each step, join two communities that produces the largest increase (or the smallest decrease) in the value of Q.

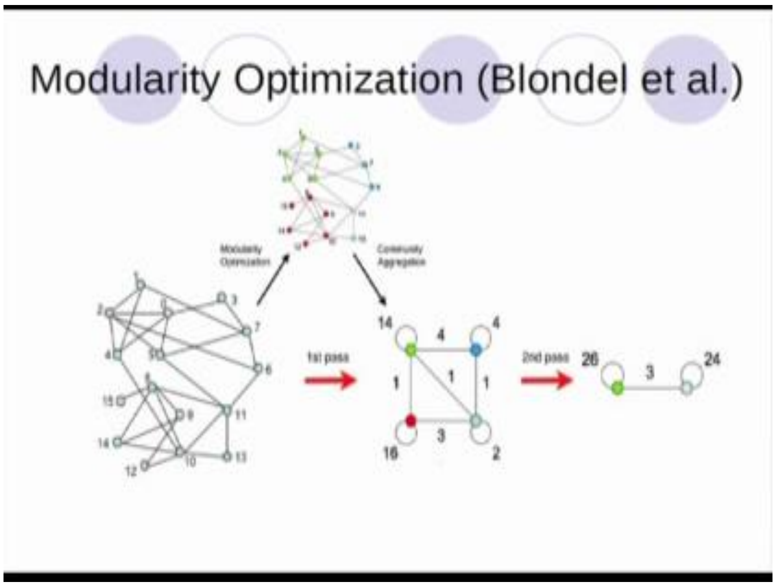
So, the first step is start. So, this is again an agglomerative algorithm. So, start with n distinct vertices in n different clusters. So, you start with as you do in all agglomerative approaches, you start with n if distinct, but this is in n different clusters. Now, at each step join two communities that produce the largest increase or the smallest decrease in the value of q. So, basically you start with each individual node in single community, single term communities.

Now, at every step you actually try to join a pair of communities based on whether when you join the overall modularity of the graph increases or not if the overall modularity of the graph is higher than the previous step or it is at least if it was negative in the previous step, it is lower negative in the next step then or if it is like positive, more positive in the next step, then the earlier step you allow this combination. You allow join of these two communities into a larger communities otherwise not. So, it is again a hill climbing approach and in this way you iteratively improve. So, it is a community optimization technique whereby you optimize the modularity function in each step in order to get better and better communities structures.

So, this is one of the very simple variations of community finding algorithm agglomerative approaches to community finding algorithms using the idea of a quality function and we can use the quality function because they themselves are tuned to identify good partition or the goodness of a partition now later on. So, these particular algorithm was tested on many real world networks and it was shown to work well, but then later on there was an even more sophisticated version of this algorithm which actually became very popular and we will now discuss that particular algorithm which actually has become very popular in the current literature because of its very nicely maintained website.

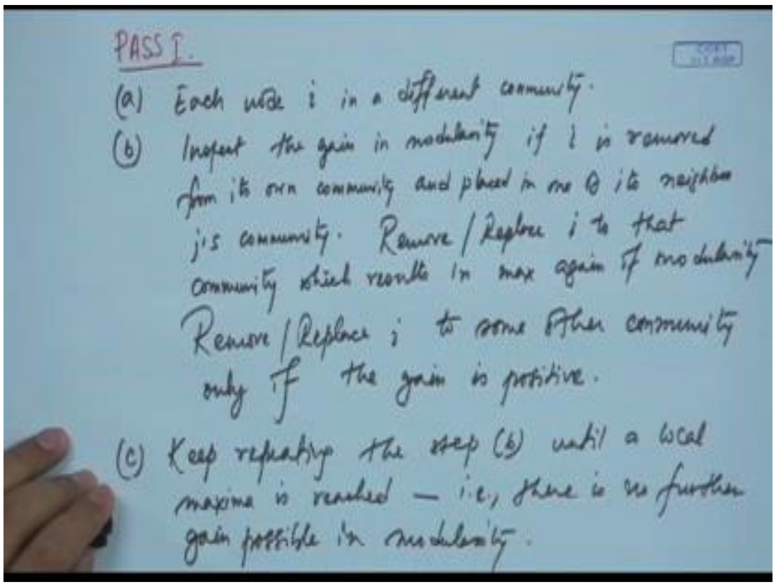
You can yourself go and find out this particular algorithm and execute them. So, it is very nicely documented and this is the algorithm that was developed by a group in (Refer Time: 10:13) and that is why it gets its name. It is also named as the Blondel algorithm. So, this was like proposed by Blondel in one of their early works in a very highly reputed physics journey.

(Refer Slide Time: 10:29)


Modularity Optimization (Blondel et al.)

So, the idea is that here you again use the idea of modularity optimization, but then this is a two pass algorithm. We will first describe the algorithm in steps, in simple steps and then we will take an example and look at its execution. So, the algorithm is as follows. So, as I said it is two-pass algorithm and the algorithm actually toggles between the two passes and in every pass there is an improvement made over the previous pass. So, we will write the algorithm now basically the two different passes of the algorithm.

(Refer Slide Time: 11:23)


PASS I.

(a) Each node i in a different community.

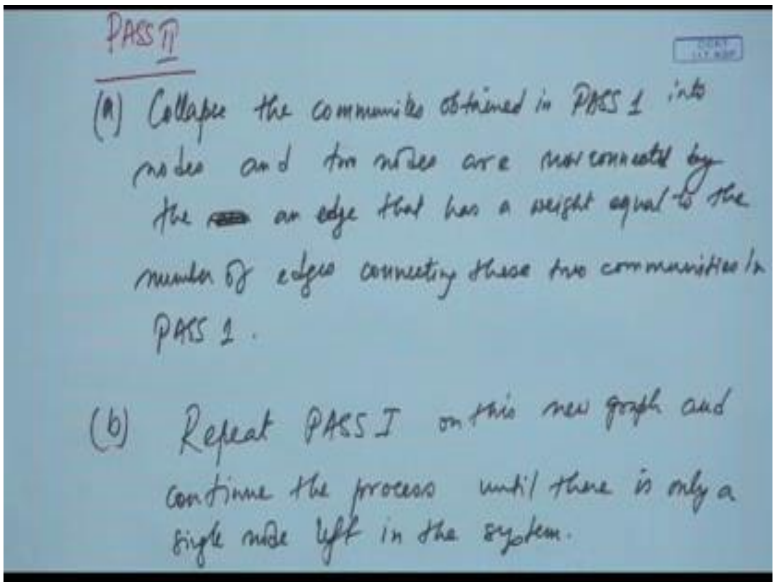
(b) Inspect the gain in modularity if i is removed from its own community and placed in one of its neighbour j's community. Remove/Replace i to that community which results in max gain of modularity. Remove/Replace i to some other community only if the gain is positive.

(c) Keep repeating the step (b) until a local maxima is reached — i.e, there is no further gain possible in modularity.

Pass I - so in this pass you assign each node i in a different community. So, each node i is assigned to a different community. Then, inspect the gain in modularity. If i is removed from it is own community and placed in one of its neighbor js community, then remove or replace i to that community which results in maximum gain of modularity which results in the maximum again of modularity remove or replace i to some other community only if the gain is positive. See keep repeating the step b until local maxima is reached that is there is no further gain possible in modularity. So, this is the first pass of the algorithm.

So, what you do? You take every individual node and each of them are in single tend communities. Now, you try to put it in some other community and see if there is a positive gain in modularity. Now, you see where actually the maximal gain is and you replace the node i to that particular community and you keep on doing this until and unless there is no further movement of nodes possible and a local maxima is achieved. Now, we come to the Pass II of the algorithm.

(Refer Slide Time: 15:14)



PASS II

(a) Collapse the communities obtained in PASS 1 into nodes and two nodes are now connected by the an edge that has a weight equal to the number of edges connecting these two communities in PASS 1.

(b) Repeat PASS I on this new graph and continue the process until there is only a single node left in the system.

Pass II comprises two steps. So, collapse the communities obtained in pass I into nodes and two nodes are now connected by the number by an edge that has a weight equal to the number of edges connecting these two communities in pass I. So, what you do here given the graph structure of pass I. So, you have found out the community structure in pass I. Now, each community is observed into a single node. So, each community that you have got in the pass I is observed into a single node.

Now, you have to also define edges between these two between a pair of nodes now. So, the edge between a pair of nodes has a weight equal to the total number of edges that was passing from one community to the other in pass I. So, in this way you construct concise version of the network. So, you observe the nodes into, you observe the communities into nodes and then you have edges with weights on them and that forms your new graph.

So, with this new graph what do you do? Repeat the pass I on this new graph and continue the process until there is only a single node left in the system. So, as I was telling you have to toggle between passes. So, now you have a new graph. Once you have from the pass I, you got the community structures, you observed each community into a node and the edges between two communities now is equal has an edge weight equal to total of number of edges passing through them in pass I. These forms here second level graph.

Now, you again execute pass I on this graph and you will again get a bunch of community structures now. Then, again you will observe them into single tend into nodes each community into nodes and you will keep on doing this same process until and unless you will end up into one single node and now, if you unfold this structure; you get a very nice organization of all the nodes into community structures. So, this is actually illustrated by an example on the slide. So, suppose you start with this particular graph here. So, from this graph after a first level modularity optimization say you arrive at this particular graph, so this is after the execution of the first pass.

So, there is one module which has all the nodes marked in red, there is another module which is marked in green, a third module marked in blue and the fourth module marked in light blue. So, now from there you do a community aggregation. So, now all the red nodes are collapsed into one single red node here. Similarly, all the green nodes are collapsed into one single green node here, all the blue nodes are collapsed into one single blue node here and all the light blue nodes are collapsed into one single light blue node here. Now, then after that you try to see what is the total number of edges that are passing from one community to another.

So, in this case you see that there is only one edge. So, this particular edge has weight one. So, now, for instance from here from this green module to the weight module, you see that there are 4 edges actually passing, 1 here, 2 here, 3 here and there is 4 here. So, now based on this you have 4 edges that are passing from green to blue.

Similarly, you have one edge from blue to light blue and three edges from light blue to red. Now, apart from this you also have self-loops here because there is large number of edges inside single community. Now, all these edges actually translate in to a self-loop with the edges. So, there are 14 edges inside red, the green community. Similarly there are 16 edges inside the eight communities and so on and so forth. Now, in this way you have the graph structure after the second pass.

At the end of the second pass, now you again implement the first pass on this and then, again you have, so again if you implement you will get some communities structure. They can be collapsed and you have even smaller substructure and this will then in the next pass get into one single node and then, when you unfold this whole structure, you will get the community, the entire community structure of the graph. So, in this way you can estimate the community structure of a graph of a real world network by using this two-step modularity optimization process and this process is shown to give a much better performance than the original modularity optimization algorithm.

Now, we have seen so far methods based on agglomerative approaches laying the hierarchical class string. We have also seen some bisective methods; we have seen some methods based on the idea of optimizing quality functions.

Now, in the last segment in the last lap of this lecture, we will study methods based on spectral bisection. So, as I told you already that one can identify community structures or components in a graph using the Eigen vectors, using the information about the Eigen vectors of the edges and symmetric of that graph. So, most of the graphs that the real world graphs that we deal with actually have positive entries and they are usually symmetric. So, for such graphs there are certainly few interesting properties that are already known and one can leverage all these properties in order to identify community's structures.

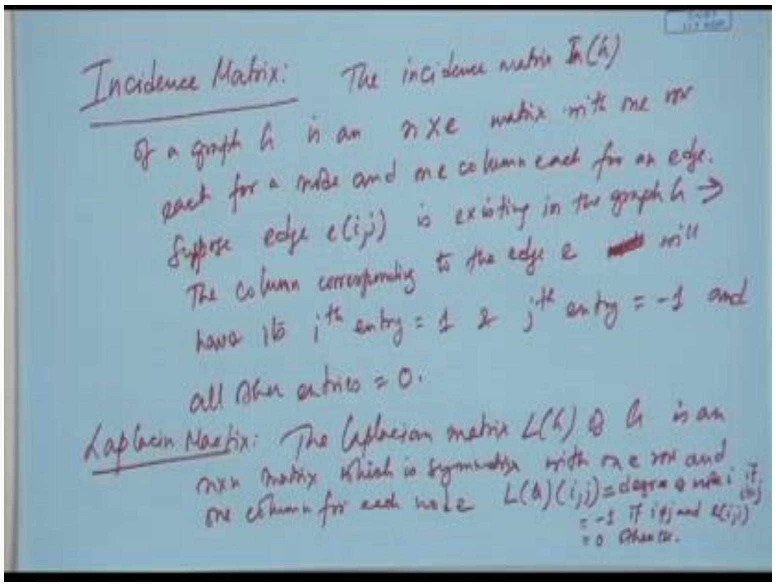
For instance, one of the properties that is known for these graphs is that the principle Eigen vector of this particular of this type of graphs or the adjacency matrix of this graphs, the principle Eigen vector of such adjacency matrices which have only positive entries and which are symmetric, such matrices have a principle Eigen vector whose all entries are positive and not only that the other Eigen vectors are actually orthogonal to the principle Eigen vector. So, every individual Eigen vector for such matrices is orthogonal to each other.

Now, if we know that the principle Eigen vector of the adjacency matrix is all positive, then the other Eigen vectors all the entries cannot be positive and then you cannot have orthogonality maintained. So, there will be some entries which are positive whereas the other entries will be negative in this vector. So, we know that for the principle Eigen vector, all the entries are positive. Now, for the other Eigen vectors also if all the entries are positive, then they would become parallel. So, this is not possible.

So, for other Eigen vectors what is necessary, what is ensured for this particular types of adjacency matrices where all the entries are positive and the matrix is symmetric. So, in such cases it is ensured that the other Eigen vectors are orthogonal to this particular principle Eigen vector and therefore, its entries should contain both positive and negative values and the positive values and the negative values, these two types of values actually nicely separate the graph into two different components.

That is the basic idea of using Eigen vectors, basically the second Eigen vector to identify the community structure of a graph. So, in order to get a better understanding of how this is done, we will try to define certain notions now and then, from there we will build up the algorithm. So, the first thing that we will introduce is two different types of matrices.

(Refer Slide Time: 25:25)



Incidence Matrix: The incidence matrix $In(G)$ of a graph G is an $n \times e$ matrix with one row each for a node and one column each for an edge. Suppose edge $e(i,j)$ is existing in the graph $G \rightarrow$ The column corresponding to the edge e will have its i^{th} entry $= 1$ & j^{th} entry $= -1$ and all other entries $= 0$.

Laplacin Matrix: The Laplacian matrix $L(G)$ of G is an $n \times n$ matrix which is symmetric with one row and one column for each node $L(G)(i,j) =$ degree of node i if $i=j$
$= -1$ if $i \neq j$ and $e(i,j)$
$= 0$ otherwise.

The matrix is called the incidence matrix. So, the incidence matrix is defined as follows. The incidence matrix i and g of graph g is an n cross e, where n is the number of nodes and e is the number of edges matrix with one row each for a node and one column each for n edge. Suppose edge e i j is existing in the graph g, then the column corresponding to the edge e will have its i th entry equal to 1 and j th entry equal to minus 1 and all other entries equal to 0. So, in this way you defined the incidence matrix of a graph.

Similarly, you have a Laplacin matrix of a graph. The l Laplacin matrix l g of the graph g is an n cross n matrix which is symmetric with one row and one column for each node, where l g i j. There is i j the entry of this matrix is defined as follows is equal to degree of node i. If i is equal to j n is equal to minus 1, if i is not equal to j and there is an edge e i j and is equal to 0 otherwise. So, this is an adjacency. This is translation of the adjacency matrix where what you have is you have the entry i that is the diagonal entries as the degree of that node and all the other entries where there is an edge between that node and the other node you have a minus 1, otherwise it is a 0. So, in this way you construct Laplacin matrix.

In the next lecture, we will see how you use these matrices in order to further identify the community structure.