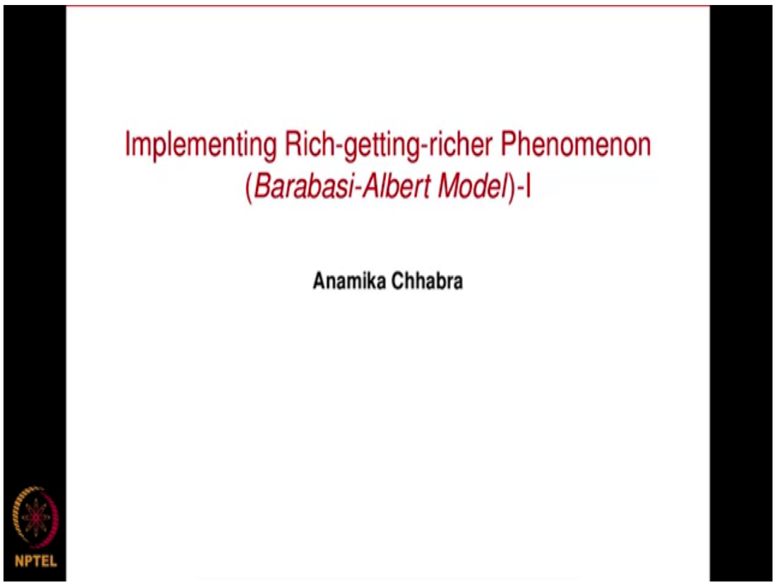
Social Networks

Prof. S. R. S. Iyengar
Prof. Anamika Chhabra
Department of Computer Science
Indian Institute of Technology, Ropar

Rich Get Richer Phenomenon
Lecture - 121
Implementing Rich-getting-richer Phenomenon (Barabasi-Albert Model)-1

(Refer Slide Time: 00:05)

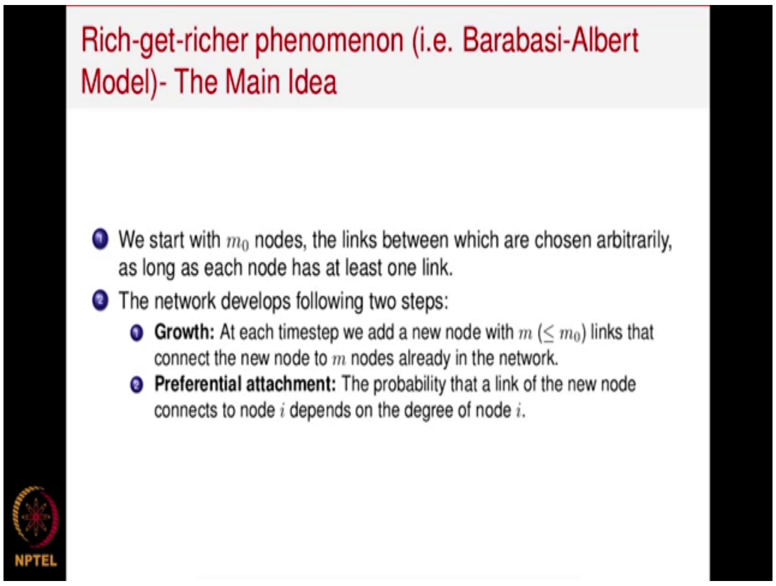


Implementing Rich-getting-richer Phenomenon
(*Barabasi-Albert Model*)-I

Anamika Chhabra

Hey everyone, in this video sequence we are going to implement an interesting phenomenon that is called Rich getting richer Phenomenon. It is also called Matthew effect and it is also called Preferential Attachment. Now, there is a particular model that is called Barabasi Albert model which is precisely based on this method that is called preferential attachment. So, we are going to implement Barabasi Albert model in this subsequent videos and that be same as implementing rich getting richer phenomena as well. Before we get into the steps of implementation, let us look at the main idea behind this concept.

(Refer Slide Time: 00:40)



Rich-get-richer phenomenon (i.e. Barabasi-Albert Model)- The Main Idea

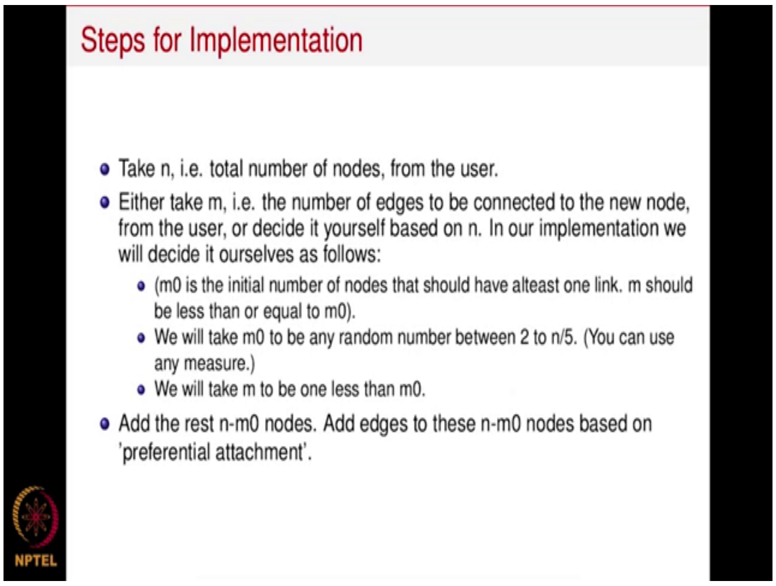
- We start with m_0 nodes, the links between which are chosen arbitrarily, as long as each node has at least one link.
- The network develops following two steps:
  - **Growth:** At each timestep we add a new node with m ($\leq m_0$) links that connect the new node to m nodes already in the network.
  - **Preferential attachment:** The probability that a link of the new node connects to node i depends on the degree of node i.

So, the main idea is that we are going to start with some m naught number of nodes. And, these nodes are going to have some edges across each other. The only condition is that every node should have at least one edge; this means that we start with an existing network of m naught nodes. And, the new node and the new notes get attached to this existing network one by one. Now, how do they get attached? There are two properties that every node follows and the first thing is that every node gets attached to some m number of existing nodes. So, for example, say m naught is equal to say 5 and m has to be less than equal to m naught.

So, assume m naught is equal to 5 and m is equal to say 3. So, we will start with an existing network of 5 nodes and 1 new node will come and it will get attached to some 3 existing nodes. And, then second node will come that will also get attached to these existing 3 nodes and so on so, this keeps happening. Now, how do we decide which 3 existing nodes should get attached to the new node, that is decided by a preferential attachment. Now, what is preferential attachment? This means that the more connected a node is, the more likely it is to receive links with a new node.

In other words more degree an existing node has more probability it will have of connecting to the new node. So, this is how we will keep adding new nodes to the existing network. This was the main idea behind rich getting richer phenomenon. Now, let us look at this steps that we are going to follow for the implementation.

(Refer Slide Time: 02:26)



Steps for Implementation

- Take n, i.e. total number of nodes, from the user.
- Either take m, i.e. the number of edges to be connected to the new node, from the user, or decide it yourself based on n. In our implementation we will decide it ourselves as follows:
  - (m0 is the initial number of nodes that should have alteast one link. m should be less than or equal to m0).
  - We will take m0 to be any random number between 2 to n/5. (You can use any measure.)
  - We will take m to be one less than m0.
- Add the rest n-m0 nodes. Add edges to these n-m0 nodes based on 'preferential attachment'.

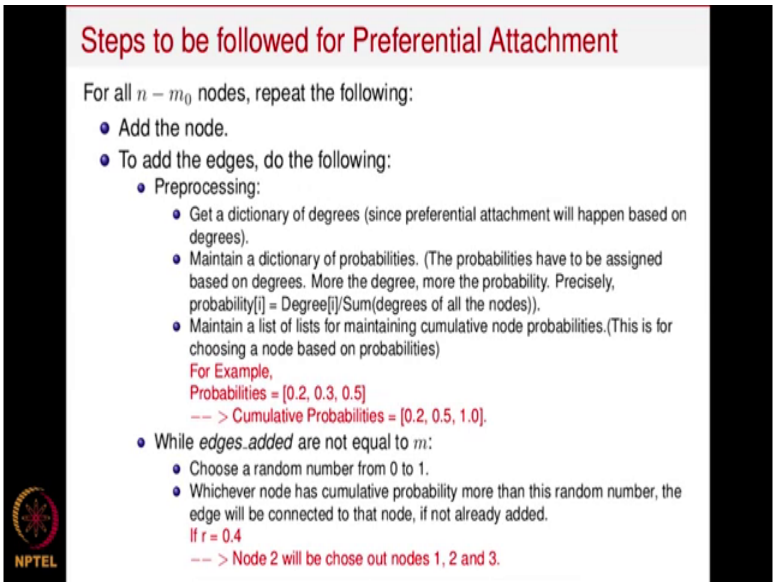
So, I am going to required value of n, n that is the total number of nodes in the network. We will take the value of n from the user, the other thing that we need is the value of m that is equal to the number of edges that every new node is going to get attached to; basically the number of edges that every node will introduced into the network. So, we will we can take value of m from the user or we can have our code decide the value of m automatically.

In our code we will decide the value of m based on n, it is up to you can do whatever you feel like ok. The other thing that we need is m naught, as I told you m naught is the number of nodes in the existing network, in the initial network. The only condition is that m should be less than equal to m naught. So, we need n, we need m and we need m naught; these 3 values we need and the only condition is that m should be less that equal to m naught.

So, in our code we are going to have m is equal to m naught minus 1, you can just fix it on any a new value that you wish to have. So, in our code we will have m naught to be any random number between 2 to n n by 5, this is this is just an assumption we are taking; you can go ahead with any value of m naught. You can fix on a value m naught is equal to 4. So, every time you will start with an initial network of 4 nodes and m we will take be m naught minus 1 you can take any value.

Now, we have a network of m naught nodes, the next step is to keep adding new nodes to this existing network. So, the number of nodes that are going to be attached is equal to n minus m naught because, n is the total number of nodes. So, these n minus m naught nodes are going to be attached to existing network based on preferential attachment as I explained you. Now, let us see how we are going to implement preferential attachment ok.

(Refer Slide Time: 04:38)



Steps to be followed for Preferential Attachment

For all $n - m_0$ nodes, repeat the following:

- Add the node.
- To add the edges, do the following:
  - Preprocessing:
    - Get a dictionary of degrees (since preferential attachment will happen based on degrees).
    - Maintain a dictionary of probabilities. (The probabilities have to be assigned based on degrees. More the degree, more the probability. Precisely, probability[i] = Degree[i]/Sum(degrees of all the nodes)).
    - Maintain a list of lists for maintaining cumulative node probabilities.(This is for choosing a node based on probabilities)
      For Example,
      Probabilities = [0.2, 0.3, 0.5]
      −− > Cumulative Probabilities = [0.2, 0.5, 1.0].
  - While *edges_added* are not equal to m:
    - Choose a random number from 0 to 1.
    - Whichever node has cumulative probability more than this random number, the edge will be connected to that node, if not already added.
      If r = 0.4
      −− > Node 2 will be chose out nodes 1, 2 and 3.

So, we have to add n minus m naught nodes one by one. So firstly, we will add the node to the network. The next step is to add m edges to this node to the existing nodes in the network. Now, how are we going to decide the nodes that are going to be attached to this existing node? As I told you we need we need the probabilities and the probabilities are going to be decided based on the degrees that the nodes have; more the degree more the probability right. So, for that pattern we need to keep a track of the degrees of the nodes. So, we will do some pre processing for this keeping track. We will maintain a dictionary of degrees for every node because, we the degrees basically going to decide the probabilities.

Second thing is we will maintain a dictionary of probabilities. So, probability of a node getting attached to a new node will be equal to the degree of that node divided by the sum of the degrees of all the nodes right. So, more the degree more the probability so, we are going to maintain a track of the probabilities in a dictionary. Now, apart from that we are going to maintain a third thing, we will basically maintain a list of list because we want to keep an order of the things and order of nodes dictionary does not let you maintain an order. So, we are preferring to keep a list of list for maintaining cumulative node probabilities ok.

Now, let me tell you why we need that. So, whenever you have to choose a node based on a probability this is a standard technique that we use, although you can device some other method is well. We are going to follow this method of making use of cumulative probabilities. Now, let me explain you what that is using an example. Assume there are 3 nodes and they have the properties 0.2 0.3 and 0.5, we are going to maintain a list that is cumulative probabilities.

So, the first node we have cumulative probability 0.2, the second we will have 0.2 plus 0.3 that is 0.5, the third will have 0.5 plus 0.5 that is 1.0. So, this is the less that we are going to maintain, I will show you how we are going to use this list. So, when new node has come, we have to connect this new node to existing m nodes for that we will choose random number between 0 and 1 ok. And, then whichever node has cumulative probability more than this random number is actually the node which is going to be connected to the new node.

Let me show that one example. So, assume r comes out to be 0.5 0.4 let us look at the cumulative probabilities list that we just created. So, first nodes cumulative probabilities 0.2 which is less than 0.4; so, we will go ahead we will check the second one, 0.5 is the second value, 0.5 is more than 0.4. So, we will stop there since r is less than or equal to 0.5 the second node is basically the node which is going to get connected to the new node. You would have understood this method, but do you understand the main idea behind it.

So, the main idea is that more probability node has more likely it will get connected to the node, this is what we have to implement. Now, try to understand how this particular method is helping us implement the same thing. So, basically more probability a node has more window it will have in the cumulative probabilities list. Now, what do I mean by that? So, as you can see just take a look at the cumulative probabilities list. So, we have first nodes cumulative probability to be 0.2. So, it is window will be 0 to 0.2, the window for the second node is 0.2 to 0.5 ok, the window for the third node is 0.5 to 1.0 ok.

So, now you can see that the window for the third node is more than the window for the second node and the window for the second node is more than the window for the first node. So, more wide a window is more likely it is that the random number is going to fall into that window ok. So, assume a node has probability 0.1, it is window will be 0.1 big, if a node has probability 0.6 it is window will be 0.6 wide and more likely it is that the random number is going to fall into that window ok.

And this precisely we are implementing here. So, if a node has more probability the random number is going to fall into that window more likely and that is how we will connect that node to the new node. So, that is the main idea. This is how we are going to add the edges to the new node and this is how we are going to add new nodes up to the point that we get total n nodes in the network. So, this is the main idea.

In fact, not the main idea this is detailed steps of implementation that I have discussed with you. I would suggest you since I have gone to detail into the steps of implementation, it will be nice if you just start of yourself, coding yourself and implement yourself and do not follow the subsequent videos line by line. And, just compare the results with the results of the subsequent videos that will be a nice practice. So, let us get started with the coding part.