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

Lecture – 60
Homophily (Continued) & Positive and Negative Relationships
Schelling Model Implementation – Getting a list of unsatisfied nodes

(Refer Slide Time: 00:06)

In the previous video we computed a list of boundary nodes and we computed a list of internal nodes as well. Now what is the next step? We have to find a list of unsatisfied nodes so that we can move to a new location one by one. Now how do we get a list of unsatisfied nodes? So, basically we will take all the nodes from G dot nodes one by one and we will test whether they are unsatisfied or not. Now how do we do that? We need to get a list of neighbours of that node if it is a boundary node it will have a different set of neighbours if it is an internal node it will have a different set of neighbours.

So, the first step is to get a list of neighbours for the given node. So, we might need to create 2 functions for that one function for getting a list of neighbours if the node is a boundary node and another function for getting a list of neighbours if the node is an internal node. Now let me show you how we can do that let me comment this I want the previous version of the graph here.

(Refer Slide Time: 01:20)

So, let me show you that once, I will just draw the graph the normal way here. So, I will write n x dot draw G and pos position also I want to pos and then p l t dot show.

(Refer Slide Time: 01:37)
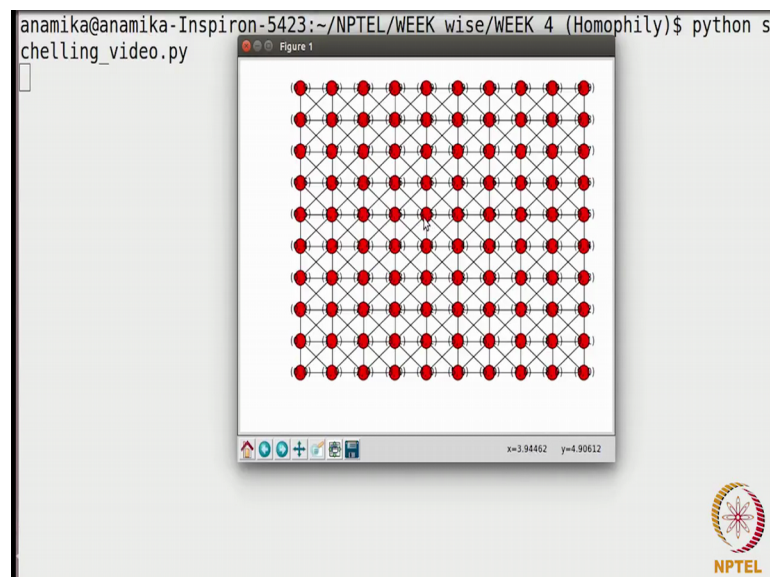
Now, let us see what we get yeah. So, see if it is an internal node for example, let us take the case of 4 comma 5.

Now, what are its neighbours 4 comma 5 has 8 neighbours now what are these; the left one that is 3 comma 5 the right one that is 5 comma 5 the bottom one the top one now how do we get them. So, given a node u comma v how do we get the neighbours u

comma v we will have a neighbour which is u minus 1 comma v it will have a neighbour which is u plus 1 comma v it will have a neighbour which is u comma v plus 1 it will have another neighbour which should be u comma v minus 1. Now next come the diagonal nodes. So, they will be u minus 1 comma v plus 1 u plus 1 comma v minus 1 u plus 1 comma v plus 1 u minus 1 comma v minus 1.

So, whatever I told you are going to be the neighbours of an internal node which is u comma v. So, that was about the internal nodes. So, we can write a function where we pos u comma v and we get the list of all the neighbours now if the node is a boundary node it will have different set of neighbours. So, we need to create a separate function for that now in case again there are 2 kinds of boundary nodes the nodes which are in the corner the 4 nodes and the rest of the nodes now again there will be special cases there if the node is 0 comma 0 I do not think they can generalize that. So, I think they going to have to do that case by case.

So, if the node is 0 comma 0 its neighbours will be 0 1 1 1 1 0 if the node is a nine comma 0. So, if you want to generalize you can just write if the node is N minus 1 comma 0. So, you remember we passed N is equal ten here. So, N minus 1 comma 0 is going to be corner node similarly N minus 1 comma N minus 1 is going to be in other corner node and 0 comma N minus 1 is going to be another corner node now these for special cases for which the neighbours are going to be only 3. So, we can write these special cases there the rest of the boundary nodes are these once. So, one list of one chunk of boundary nodes are going to be the once where u is equal to 0 right and in those cases the boundary the neighbours are going to be 1 2 3 4 5.

So, if it is 0 comma v it will be 0 comma v plus 1 0 comma v minus 1 1 comma v plus 1 1 comma v 1 comma v minus 1 you see how I am doing that I think you can do that for the rest of the cases where let me give you one more example. So, if it is amongst the nodes where v is equal to and minus 1 as you can see it will it will give us this row of boundary nodes right. So, again and those cases you can write the and the neighbours that are going to be there and this is going to be the row where u is equal to N minus 1 and this is going to be the row where v is equal to 0. So, you can write all those special cases and you can get a list of the neighbours for the given boundary node.

Let us go back now to save time I have already written all the cases. So, that we do not spend a lot of time in the video making. So, let me show you all though I have already explained to you. So, this is the function get neighbours for an internal node. So, we are posing u comma v and for N internal node the cases are very straight forward. So, then you completely generalized it. So, you are you are returning a list there which is having all the 8 neighbours. So, that is straight forward another function that I have created already is forgetting a list of neighbours for a boundary node as I told you.

(Refer Slide Time: 06:02)

There are 1 2 3 4 8 cases. So, this you can get. So, there are; this also I comment. So, you see there are 8 cases here 1 2 3 4 5 6 8 cases all the cases that their as I told you first 4 cases are the ones for the corner nodes and the rest 4 cases are the ones for the rest of the boundary nodes.

So, I do not think I need to explain much there. In fact, I would like you to try yourself N case you need help we can just check the values here. So, these are all possible values you can just open the graph the way I opened and then you can note down the neighbours are going to be there that that should be simple to do now we have to get a list of unsatisfied nodes.

(Refer Slide Time: 06:56)


```python
48            return [(u, v-1), (u-1, v), (u+1, v), (u-1, v-1), (u+1, v-1)]
49        elif v == 0:
50            return [(u-1, v), (u+1, v), (u, v+1), (u-1, v+1), (u+1, v+1)]
51
52    def get_unsatisfied_nodes_list(G, internal_nodes_list, boundary_nodes_list):
53        unsatisfied_nodes_list = []
54        t = 3
55        for u,v in G.nodes():
56            type_of_this_node = G.node[(u,v)]['type']
57            if type_of_this_node == 0:
58                continue
59            else:
60                similar_nodes = 0
61                if (u,v) in internal_nodes_list:
62                    neigh = get_neigh_for_internal(u,v)
63                elif (u,v) in boundary_nodes_list:
64                    neigh = get_neigh_for_boundary(u,v)
65
66                for each in neigh:
67                    if G.node[each]['type'] == type_of_this_node:
68                        similar_nodes += 1
69
70                if similar_nodes <= t:
71                    unsatisfied_nodes_list.append((u,v))
72
73        return unsatisfied_nodes_list
```

So, we will create a function for that we will write get unsatisfied nodes list lets pos graph here. Now we will check all the nodes one by one and we will check how many neighbours of the given node have the same type as the as the type of the given node right. So, let us create a list; unsatisfied nodes list and (Refer Time: 07:34) initially now we have to check for each node. So, we will write for u comma v N G dot nodes.

We have to keep a node of the type of this node that is u v. So, we can write type of this node because we will be comparing write type of this node is equal to G dot node, we have to see the type of u v. So, you write u v and then type here there should give us the type of the node. So, we have stop here in case the type of this node is equal to (Refer Time: 08:18) what are we going to do we basically do not have to do anything as in if the type of this node is equal to 0 it can never be unsatisfied right. So, we will just continue and we will check the next node. So, we will write if type of this node is equal to 0 we just continue and check the next node as for now we are going to do.

We are going to see in the list of neighbours of this node how many nodes are having a similar type as this nodes type. So, we will have to keep drag of that. So, we will write similar nodes is equal to 0 initiate initially now we have to check whether this node u v is a internal node or is it a boundary node because accordingly we will get the list of neighbours. So, we have to check here now do we check whether or node is internal node

or boundary node we have already created a list of internal; internal nodes in boundary nodes here, right.

(Refer Slide Time: 09:22)


```python
 97
 98  for n in G.nodes():
 99      G.node[n]['type'] = random.randint(0,2)
100
101  type1_node_list = [n for (n,d) in G.nodes(data = True) if d['type'] == 1]
102  type2_node_list = [n for (n,d) in G.nodes(data = True) if d['type'] == 2]
103  empty_cells = [n for (n,d) in G.nodes(data = True) if d['type'] == 0]
104
105
106  # print type1_node_list
107  # print type2_node_list                          I
108  # print empty_cells
109
110  # display_graph(G)
111
112  boundary_nodes_list = get_boundary_nodes(G)
113  internal_nodes_list = list(set(G.nodes()) - set(boundary_nodes_list))
114
115  # print boundary_nodes_list
116  # print internal_nodes_list
117
118  unsatisfied_nodes_list = get_unsatisfied_nodes_list(G, internal_nodes_list, boundary_n
119  print unsatisfied_nodes_list
120
121  make_a_node_satisfied(unsatisfied_nodes_list, empty_cells)
122
```

So, it will be nice if you pass this these lists into the function boundary nodes list. So, we will pass these 2 lists in the function and we will pass these lists and we will check whether the node belongs to this list or not that we will tell us whether the node is boundary node or internal node.

Internal nodes list; so, what we have into do here we will check if u comma v sorry in internal nodes list which will which will mean that it will it is an internal node then its neighbours. So, you have to get all its neighbours how do we get its neighbours we have already created a function for getting the neighbours of an internal node we will write get neighbours for internal, right. So, we are going to pos the node here in case this u is not an internal nodes list it will (Refer Time: 10:33) in the other list that is boundary nodes lists. So, we will write l f u comma v in boundary nodes list what do we do now we get a list of neighbours again we have created a function for that get neighbour of boundary right for boundary yeah get neighbour for boundary and we will pass the node here you will.

So, now we now got a list of neighbours what do we have to do now we have to check amongst these neighbours how many of them are having the same type as the type of this node that is what we have stored already. So, we will start a loop for all the nodes in the

neighbour. So, you write for each in neighbour if G dot node. So, we have to check the type of each. So, I will write if I am sorry if this is equal to type of this node we have we have to increment the counter that is we already initial right similar nodes plus is equal to one. So, number of similar loads nodes well be incremented after this loop is over we will have a count of the similar nodes. So, if that similar node is less than equal to the threshold. So, maybe you can initialize the threshold here maybe t is equal to 3 I will write here.

So, if the same number similar node is less than equal to t this should mean that the node is unsatisfied. So, we will at this append this node to the list of unsatisfied nodes. So, write unsatisfied node list or append u v u v and this function is going to written the list of unsatisfied nodes I will write return unsatisfied nodes list. Now let us call this function here. So, we will write unsatisfied nodes list is equal to get unsatisfied no nodes list and the parameters are going to be the graph and internal nodes list and boundary nodes list yeah in order to verify let us print whether it works it works fine or not. So, I will print unsatisfied nodes list let me comment this I do not want to see them. So, like see. So, this is the graph we are getting the list of unsatisfied nodes here. So, we are doing good. Now what we have to do?

(Refer Slide Time: 13:19)
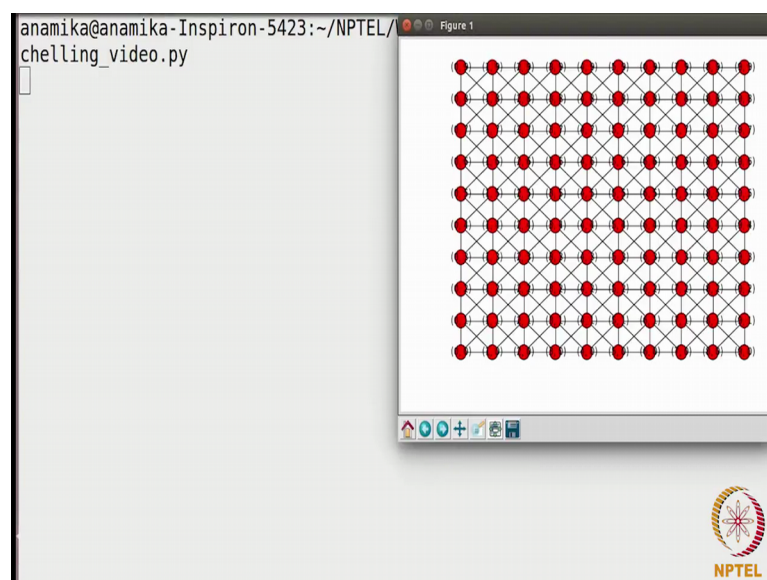
(Refer Slide Time: 13:22)


```
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_4_(Homophily)$ python s
chelling_video.py
[(0, 7), (1, 6), (5, 8), (4, 0), (5, 5), (7, 6), (0, 4), (1, 1), (2, 6), (9,
 3), (6, 0), (7, 5), (0, 1), (3, 1), (9, 9), (8, 9), (9, 4), (7, 2), (1, 5),
 (3, 6), (9, 7), (0, 8), (4, 6), (9, 2), (6, 1), (5, 7), (7, 4), (0, 2), (1,
 3), (3, 0), (2, 8), (9, 8), (1, 4), (3, 9), (2, 3), (9, 6), (6, 5), (3, 4),
 (2, 4), (4, 7), (9, 1), (6, 6), (7, 7), (0, 3), (4, 9), (3, 3), (8, 1), (7,
 9), (2, 0), (8, 8), (9, 5)]
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_4_(Homophily)$
```

After getting a list of unsatisfied nodes we have to take the nodes take one of the unsatisfied nodes and we have to move it to a new place now that new place should be empty, right. So, we also have to keep a track of the empty places we initially did that we kept a list of empty cells. So, that we have already. So, what we will do is I will go back what we will do is from this list of unsatisfied nodes we will choose one node randomly from the list of empty cells we will choose one place randomly we and then we will put the randomly chosen node from the unsatisfied list into this randomly chosen position from the empty cells and after that; obviously, the number of unsatisfied and satisfied nodes we will change. So, the graph we will actually change after that we will re compute all the values that is unsatisfied empty satisfied everything will be re computed and then we will see how the graph looks like this we will keep on repeating for several iterations and in the end we will compare how we are how the graph is looking like and how the graph is looking like initially.

So, as I said the next step is to make or node satisfied we can create a function for that make a node satisfied what do we need to pass their we; obviously, need to pass the list of unsatisfied nodes because we will randomly choose one out of that and second thing that has to pass just the anti cells list because we will run issues one position out of that I think this is good. So, in the next video, we will implement this function make a nodes satisfied.