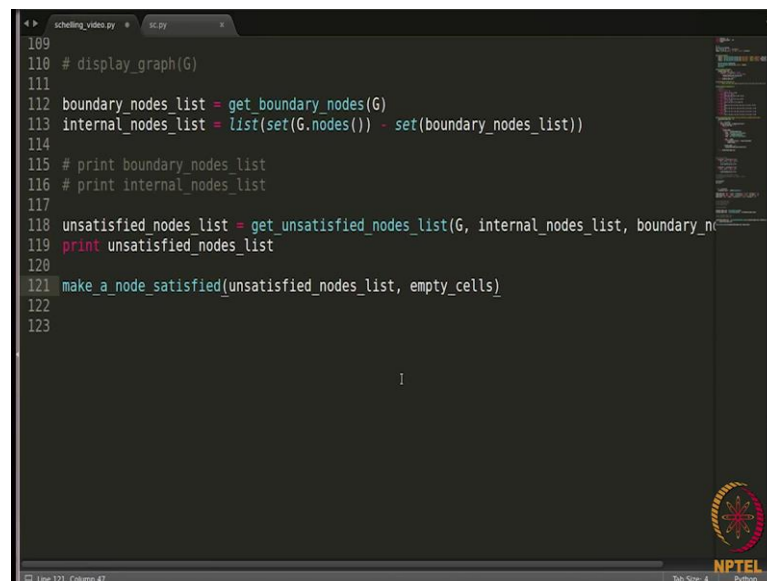


**Social Networks**  
**Prof. S. R. S. Iyengar**  
**Department of Computer Science**  
**Indian Institute of Technology, Ropar**

**Lecture – 61**

**Homophily (Continued) & Positive and Negative Relationships**  
**Schelling Model Implementation - Shifting the Unsatisfied Nodes & Visualizing the Final Graph**

(Refer Slide Time: 00:05)

A screenshot of a Python IDE window titled 'schelling\_video.py'. The code is as follows:

```
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
123
```

The IDE has a dark theme. On the right side, there is a sidebar with a file explorer and a console. At the bottom right, there is an NPTEL logo and the text 'NPTEL Python'. The status bar at the bottom indicates 'Line 121, Column 47' and 'Tab Size: 4 Python'.

We now have list of unsatisfied nodes which we completed in the previous video and we have to now choose one node out of that list and we have to make it satisfied; how do we do that we have to move that node to an empty cell which also we will be randomly choosing. So, let us create this function.

(Refer Slide Time: 00:35)

```
70         if similar_nodes <= t:
71             unsatisfied_nodes_list.append((u,v))
72
73         return unsatisfied_nodes_list
74
75     def make_a_node_satisfied(unsatisfied_nodes_list, empty_cells):
76         if len(unsatisfied_nodes_list) != 0:
77             node_to_shift = random.choice(unsatisfied_nodes_list)
78             new_position = random.choice(empty_cells)
79
80             G.node[new_position]['type'] = G.node[node_to_shift]['type']
81             G.node[node_to_shift]['type'] = 0
82             labels[node_to_shift], labels[new_position] = labels[new_position], labels[node_to_shift]
83         else:
84             pass
85
86
87     I
88     #Add diagonal edges
89     for ((u,v),d) in G.nodes(data=True):
90         if (u+1 <= N-1) and (v+1 <= N-1):
91             # print (u,v), (u+1, v+1)
92             G.add_edge((u,v),(u+1,v+1))
93
94     for ((u,v),d) in G.nodes(data=True):
95         if (u+1 <= N-1) and (v-1 >= 0):
```

So, let us create this function, here I will write this. So, in case the list is empty we will I add just one condition here if length of unsatisfied nodes list is not equal to 0 then we will do or this and what is that we have basically have to choose one node randomly out of this list we will make use of the function rand dot choice.

So, let us write a node to shift the one we which we will be shifting in this particular iteration. So, in one iteration we will be shifting 1 node. So, we will choose the node randomly random dot choice unsatisfied nodes list now where do we shift this node again, we have to choose one empty location randomly. So, maybe we can write new position is equal to random dot choice from empty cells now we got the node and we have to move it what kind of changes we will happen when a node is shifted. So, basically the, so, here we have on the left-hand side assume we have a node which is to be shifted on the right-hand side we have another node which is empty.

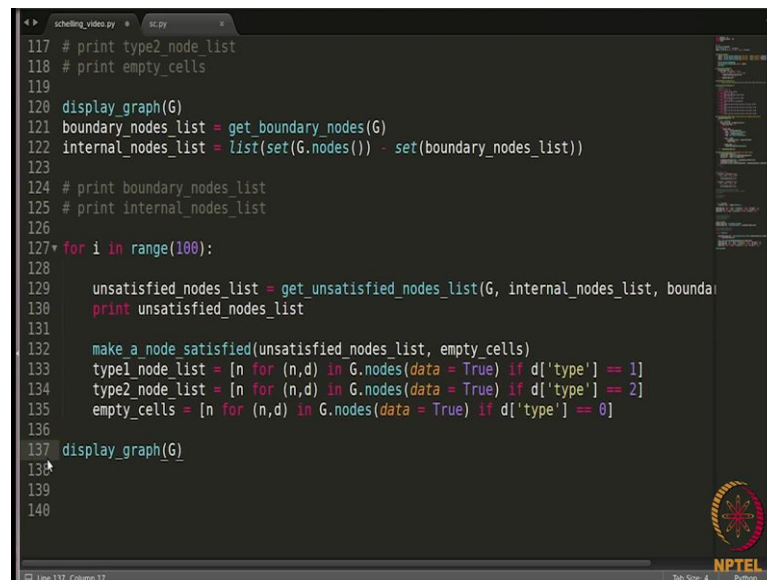
So, the type of the empty node is right now 0. So, the type of this empty node will change to the type of the node which is now coming into this cell that is one thing also when we look at the left hand side when this node is shifted this location will become empty. So, the type of this node will become 0. So, this changing of type will happen as the first thing second thing that has to happen is the changing of the labels. So, the label of the node on the left-hand side and the node on the right-hand side will be exchange will be exchange yeah basically. So, I hope you are able to imagine how I am going to

explain you. So, let us first change the type. So, I am going to change the type of the empty cell to the type of a node which is going to be shifted. So, I will write G dot node new position is basically the location of the news of the empty cell type is going to be equal to the type of the node which is node to shift.

So, now we have change the type of the empty cell to the type of the node which is going to be coming into this cell and the node which got shifted will now be 0 as type. So, I will write G.node to shift it is type will now be 0 done second thing that has to be done is to exchange the labels. So, in python exchanging is very straight forward 1 1 simple command we can just to; now where are we storing the labels if you remember initially, we created this dictionary that is labels, right. So, this dictionary is having all the labels where this is where we have to make changes. So, in this dictionary labels there will be a label for node to shift there will be and in this dictionary labels there will be a label for the node new position already we have to exchange that.

So, let me I am sorry how do we do it x comma y is equal to y comma x as implies that this is how you exchange contents of two variables this no need for any temporary variable. So, that is this while do is well labels node to shift comma labels new position is equal to labels new position comma labels node to shift I think where done in the function let me write the else part of this if. So, in case the there is nothing in the unsatisfied nodes list basically all the nodes are satisfied we are not going to do anything. So, just we will write a pass nothing to be done. So, we are done with this function which is make a node satisfied now let us get back, here we call this function make a node satisfied after which the list of type a nodes and the list of type two nodes and the list of empty cells we will change and we will need the those lists again. So, they need to update them. So, what is going to write is type one I think we did that in this copy paste let me yeah.

(Refer Slide Time: 06:02)

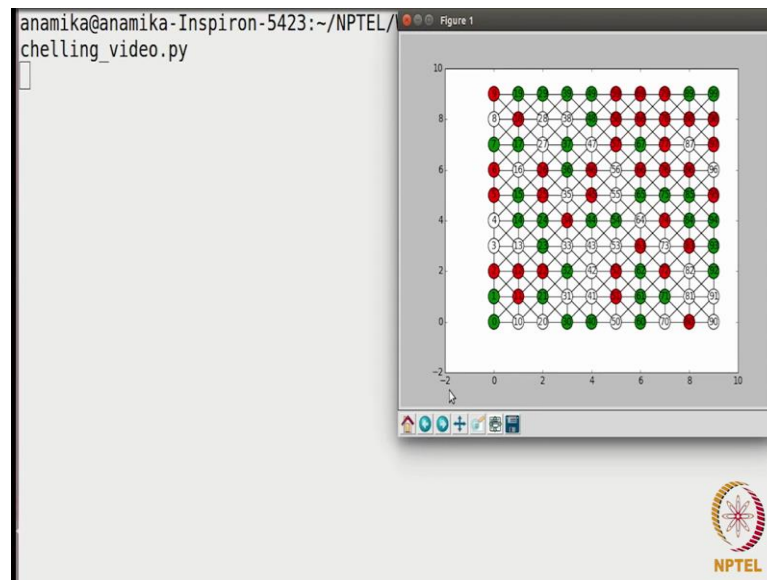


```
117 # print type2_node_list
118 # print empty_cells
119
120 display_graph(G)
121 boundary_nodes_list = get_boundary_nodes(G)
122 internal_nodes_list = list(set(G.nodes()) - set(boundary_nodes_list))
123
124 # print boundary_nodes_list
125 # print internal_nodes_list
126
127 for i in range(100):
128
129     unsatisfied_nodes_list = get_unsatisfied_nodes_list(G, internal_nodes_list, bounda
130     print unsatisfied_nodes_list
131
132     make_a_node_satisfied(unsatisfied_nodes_list, empty_cells)
133     type1_node_list = [n for (n,d) in G.nodes(data = True) if d['type'] == 1]
134     type2_node_list = [n for (n,d) in G.nodes(data = True) if d['type'] == 2]
135     empty_cells = [n for (n,d) in G.nodes(data = True) if d['type'] == 0]
136
137 display_graph(G)
138
139
140
```

So, this is where we computed these 3-list type one node list type two node list empty cells now after node is shifted to another location these lists are going to undergo changes is well. So, we need to re compute them. So, I am just copying pasting here over here now how many times we do and then do we have to do it as you see make a node satisfied only makes one nodes satisfied. So, we have to keep doing this for all the nodes which are unsatisfied. So, maybe we can start a loop here. So, I will right for I in range you can change recording or you can keep a number over here which is dependent on the total number of nodes in the grid I am for now I am keeping it just hundred. So, basically, we have to repeat all these steps until all the nodes gets satisfied. So, this all will go under this loop once this is done, we are going to display the graph again. So, we will call this function display graph G.

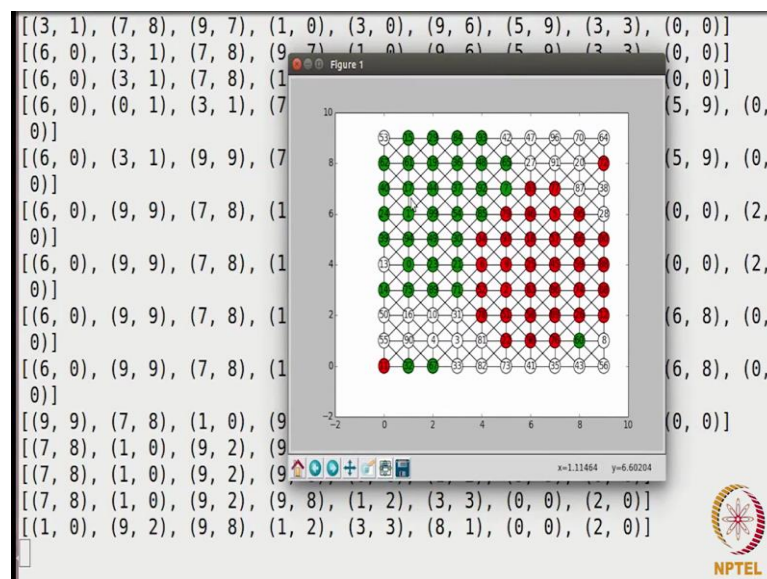
Now, before I will execute let me it removes these least commons common based commons which where displaying the graph in the very first form. So, let me trend the let me display the initial graph is well display graphs. So, we are going to we are just displaying the initial version of the graph and after that we have this loop in which all the shifted will happen and in the end this is how the graph this is the command that will display the final graph. So, this should be displaying the initial graph they should be displaying the final graph and in the middle, they are these hundred iterations let me increase the number of iterations here. So, what should basically happen the nodes should move towards nodes of their own time?

(Refer Slide Time: 08:31)



So, that is what is expected let us see how it works lets executed and see how it works. So, this is the initial graph as you can see the nodes of different colours are scattered. I am closing it and now.

(Refer Slide Time: 08:42)



So, this is the final graph as you can see there are all the green nodes are on the one side all the red nodes are on the other side and there are few other nodes as well with scene unsatisfied is well maybe we can increase in the on the counter there. So, that counter will basically depend on many things it could depend on the size of the network that if

we have taken. Secondly, we have assigned the nodes randomly. So, for different random settings you might require different number of iterations you can also keep a track of the number of hydration that that were required to convert this graph into a graph where the people are same type on together.

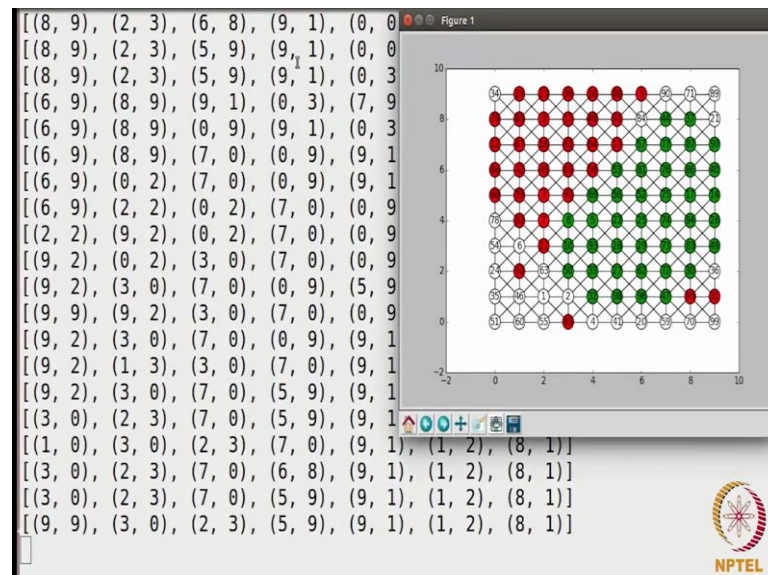
(Refer Slide Time: 09:25)

```

117 # print type2_node_list
118 # print empty_cells
119
120 display_graph(G)
121 boundary_nodes_list = get_boundary_nodes(G)
122 internal_nodes_list = list(set(G.nodes()) - set(boundary_nodes_list))
123
124 # print boundary_nodes_list
125 # print internal_nodes_list
126
127 for i in range(5000):
128
129     unsatisfied_nodes_list = get_unsatisfied_nodes_list(G, internal_nodes_list, bounda
130     print unsatisfied_nodes_list
131
132     make_a_node_satisfied(unsatisfied_nodes_list, empty_cells)
133     type1_node_list = [n for (n,d) in G.nodes(data=True) if d['type'] == 1]
134     type2_node_list = [n for (n,d) in G.nodes(data=True) if d['type'] == 2]
135     empty_cells = [n for (n,d) in G.nodes(data=True) if d['type'] == 0]
136
137     display_graph(G)
138
139
140

```

(Refer Slide Time: 09:29)



10,000 seems too much; may be 5000; 5000 let us try for these many iterations. So, it is going its going good still there are a few unsatisfied nodes, but in more iterations they will converge they will converge basically it. So, happens that all the nodes become

satisfied another thing to note here is that you can also change the value of  $t$ . So, here we have taken  $t$  is equal to 3. So, even for an internal node since it has 8 neighbours even out of 8 neighbours only 3 are of its own type the node will not move to any other place. So, so you can change  $t$  to 4 5 or any other value and you can see the number of iterations that were required to get to the convergence. So, you can I think you have gotten enough tools now you can play around with all these values you can keep changing and you can observe and if needed you can plot things as well as in the number of iterations required for converging the these many number of nodes you can change the value of capital  $N$  which we initially to a capital  $N$  was 10 which means aware 100 cells.

So, that was about this Schelling model I would suggest you code yourself and you know watch the video only when it is needed see in the initial video I think I gave the structure as to what has to be done. So, if for every video you do that in the sense you just listen to the structure and the aim that has to be implemented and then code is code it yourself without just following the video that will be actual learning for you. So, I would suggest you to code yourself and check the video whenever needed only.

Thank you.