Project 2

Pranav Menon

April 25, 2024

1 Question

In this final part of the question, we will deal with how the impression network can help make people popular.

Or in other words what can we deduce about the impressions. Assume A wants B to do something, or wants to make an impression on B. It might be possible that A has not met B at all.

There are two ways in which A can go about it:

- Either A can directly go and talk to B. But the chances that B would like A or do something for A are less in this case.
- Another way to go about it. Now that we are given the impression network. And we know that there is a person C in the equation. It is known that C finds A impressionable, and B finds C impressionable. So A can go through B. If C is ready to help A, then the chances that he/she will be able to persuade B would increase.

These and one another interesting observation would be observed in the following report.

2 Prepossessing :

The prepocessing has been done in the same way, as it was done from problems 1 and 2. The below graph was generated:

This makes the plot for the graph.

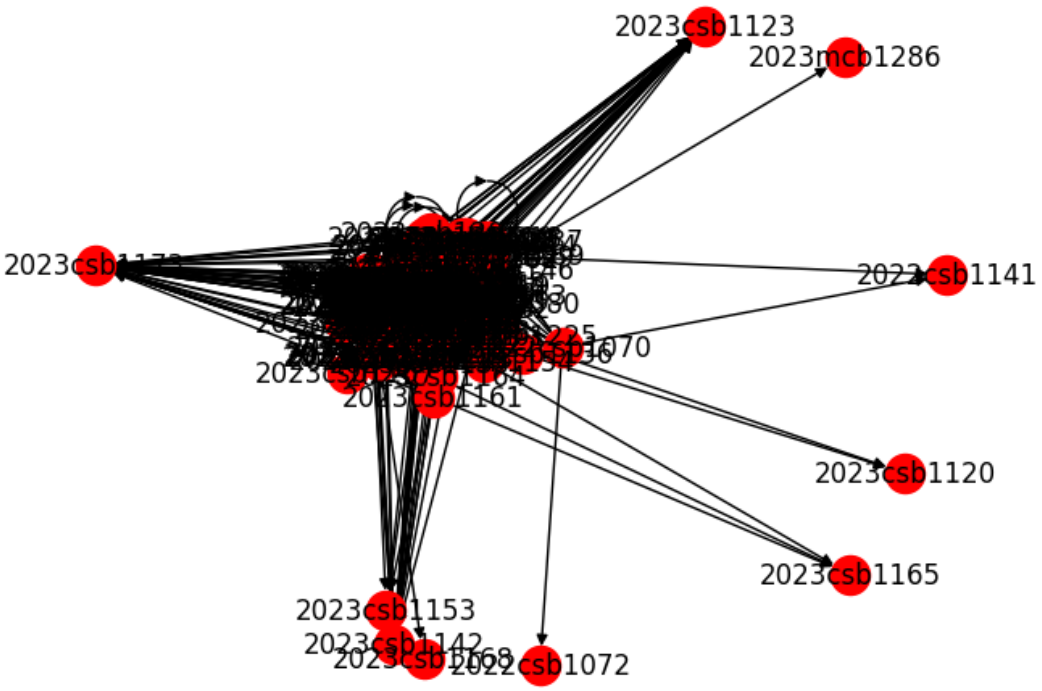

Figure 1: Image for the question.

3 Observation

In the graph, a very interesting point was observed. There were two nodes in the graph which had many incoming links (More than 20) which means that there were many people which have seen these two people during the experiment but, there was no row in which they had filled their impressions of people.

Out of 3376 impressions which were filled in the graph, 2344 were such impressions which were symmetric. Which means that between two nodes A and B both A and B found each other impressive.

These amount to:

$$\frac{(2344)}{3376} * 100 = 69.43\% \tag{1}$$

This can be found using the simple code:

```
matrix = nx.to_numpy_array(G)
total = 0
total_impr = 0
for i in range(143):
  for j in range(143):
    if(matrix[i,j] == matrix[j,i] and matrix[i,j] == 1):
      total += 1
    if(matrix[i,j] == 1):
      total_impr += 1
print(total_impr, " ", total)
```

Hence due to human tendency, people perhaps were attracted towards the same group of people which they already knew. Meaning that if someone meant someone they were inclined to include them in the impression network.

Now, the following code delects certain anomalies between nodes:

```
# One of the things is to find unusual nodes:
# indegree and outdegree
in_degree = dict(G.in_degree())
out_degree = dict(G.out_degree())

in_degree = {node: int(degree) for node, degree in in_degree.items()}
out_degree = {node: int(degree) for node, degree in out_degree.items()}

unusual_difference = [(node, abs(in_deg - out_degree[node])) for node, in_deg in in_degre

if unusual_difference:
    for node, difference in unusual_difference:
        print(node, " ", difference)
else:
    print("None of the nodes were found")
```

The above code outputs the following nodes:

- "2023csb1123" which has 28 incoming links but 0 outgoing links
- "2023csb1173" which has 39 incoming links but 0 outgoing links

This most probably means, that the data for the above nodes is missing.

4 Impression Part-1:

Now consider, the problem in hand is this. A wants B to do something for him. One way is to ask B directly. But A is smart and being equipped with the impression network, A knows that B has a good impression of C and hence chances that B agrees to C are greater.

Now let A, for the sake of fun, not have the knowledge about the impression network but he posses the knowledge that such a network exists. What he decides that he would go to every node who found him impressionable.

One way to go about it is that he goes to everyone who he talked to and ask them to go to a person who found them impressionable. They can go to multiple people but we cannot expect them to go to every node who found them impressionable. This is basic human tendency. A will work the hardest but the other nodes might not work as hard as A. They might only talk to 1 person, 2 people.

4.1 Simplified Version:

The above problem might be too complex to find a coherent solution. Let's reduce the problem to the scale by taking some assumptions. One of them is that the person only talks to a fix amount of people. The below section deals with various values of these people.

A case we need to deal with is that we are not able to reach B using the above algorithm.

Also, please note there is no condition that a unique node is chosen for being added to the queue. Meaning, the node which has already been visited might be visited (talked to) by some other person and he might pat himself on the back on the work well done, when in actuality he did nothing.

4.2 Assumptions:

The following assumptions have been taken:

- All the nodes where we reach must talk to exactly k number of nodes, except dead ends.
- All nodes agree to the process.

4.3 For 1 person

```
from collections import deque
broke = 0
path_ = 0
total = 0
for i in range(10000):
  path = 0
  A = random.choice(all_nodes)
  B = random.choice(all_nodes)
  visited = set()
  queue = deque([(A, 0)])
  flag = 0
  i = 0
  for i in range(1000):
    if queue:
      current_node, last_dist = queue.popleft()
      if current_node == B:
        path_ += last_dist
        total+=1
        break
      if current_node not in visited:
        visited.add(current_node)
        if flag == 0:
          flag = 1
          for node in G[current_node]:
            if node not in visited:
              queue.append((node, last_dist + 1))
        else:
          out_edges = list(G.out_edges(current_node))
          if len(out_edges) == 0:
            continue
          else:
            random_edge = random.choice(out_edges)
            next_node = random_edge[1]
            if next_node not in visited:
              queue.append((next_node, last_dist + 1))
    else:
      broke += 1
      break
print("It broke : ", broke, " number of time and the average distance
between two nodes was : ", (path_/total))

print(total)
print(path_)
```

Now, we can just add conditions for talking to different number of nodes.

4.4 Result for various thresholds:

Table 1: Various Thresholds

Number of people	Miss rate	Average path length	Total Iterations
1	5999	2.315	10000
2	2998	2.910	10000
3	1887	2.680	10000
4	1404	2.491	10000

It is evident from the above fact, more people you can get onboard, and greater the number of dedicated volunteers you have, the better chances would be that you can reach the end person. Also, path length is kind of proportional to the time it takes. This is almost constant across all the variables.

4.5 Dead Ends:

These nodes are the main point of trouble for our code. Assume a node with only incoming edges. There is no where we can go from this point forth. These dead ends won't be able to propagate the message forward, but the person who talked to this person would again think his work is done.

```
# Dead end... -> Information goes lost
# The nodes which have no out-degree. Or these people were the least impressionable
for i in out_degree.items():
  if i[1] == 0:
    print(i)
```

The above code prints the dead ends.

- 2023csb1123
- 2023csb1142
- 2023csb1173
- 2023csb1165
- 2023csb1153
- 2023mcb1286
- 2023csb1168
- 2023csb1120
- 2022csb1141
- 2022csb1072

These are the dead ends. Information isn't propagated further.

5 Impression Part-2

Now, let us consider an ever bigger problem. Assume a node A decides to run for the President of the student council.

The same mode of communication is observed. A goes and talks to all the people he made an impression on during the activity. He tells them to spread the word about him contesting for elections. They would go ahead and talk to more people. Now, the amount of effort they put might vary. For the sake of generality, we assume that the amount of effort (directly proportional to the number of nodes they talk to) is chosen randomly between 0 and 1.

Assumption: We are assuming that all the people are ready to help. Although the amount of effort might come out to be ≈ 0.

Now, our result is to find the number of nodes which we can reach using this. Please note since this is not a normal BFS, we might actually not be able to reach all the nodes.

```
# Going a step further:
# Election
from collections import deque
broke = 0
path_ = 0
```

```
total = 0
for i in range(10000):
  path = 0
  A = random.choice(all_nodes)
  visited = set()
  queue = deque([(A, 0)])
  flag = 0
  i = 0
  for i in range(1000):
    if queue:
      current_node, last_dist = queue.popleft()
      if current_node not in visited:
        visited.add(current_node)
        if flag == 0:
          flag = 1
          for node in G[current_node]:
            if node not in visited:
              queue.append((node, last_dist + 1))
        else:
          out_edges = list(G.out_edges(current_node))
          if len(out_edges) == 0:
            continue
          else:
            random_prob = random.random()
            total_nodes = len(out_edges) * random_prob
            total_nodes = int(total_nodes)

            for i in range(total_nodes):
              random_edge = random.choice(out_edges)
              next_node = random_edge[1]
              if next_node not in visited:
                queue.append((next_node, last_dist+1))
    else:
      path_ += len(visited)
      broke += 1
      break
path_ = path_/10000
print(path_)
print("The percentage of nodes left", (143-path_)*(100/143))
```

The above does the following:

- It runs for a total of 10,000 iterations.
- It chooses a random A, (Anyone can hope to become the president for the student council)
- It then maintains a visited set, a queue for the modified Breadth first search
- If there is an element in queue, it makes it the current node. For A it adds all the elements to queue.
- Else, it chooses a random number between 0 and 1, the amount of effort and calculates the number of nodes that the current_node talks to. Then it chooses these many random edges. If they were never visited we add it to the queue.
- We iterate until the queue is full. If it becomes empty we just break out of the process.

5.1 Output

The output for the file is:

Table 2: Number of nodes

S. No.	Number of nodes left	Percentage of node left
1	129.213	9.640
2	128.619	10.05
3	129.123	9.703
4	128.1801	10.36

Hence, we can know that we reach atleast, on an average about 128 nodes, which are about 90% of the total nodes.

6 Conclusion:

Hence we study the effect of the impression network. How we can reach a single node using the impression network and how many people can we influence using the methods discussed in the report.