Seed-based De-anonymization

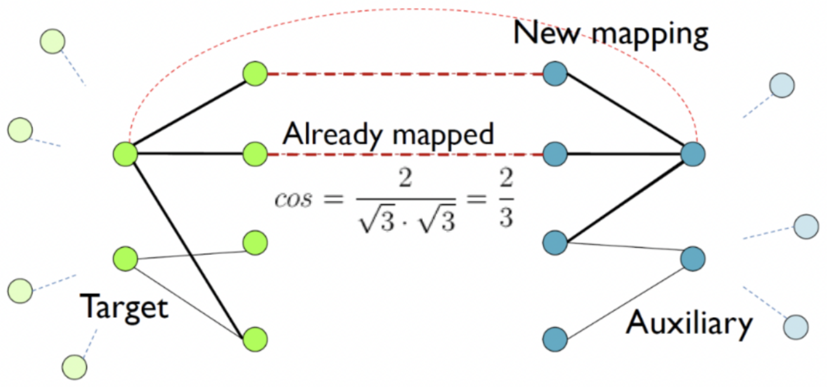
Individual Project 1

De-anonymization is a technique used in privacy that attempts to re-identify encrypted or obscured information. De-anonymization, also referred to as data re-identification, cross-references anonymized information with other available data to identify a person, group, or transaction.

In our algorithm, we have 2 graphs. G1(V, E) and G2(V, E) .

G1(V, E) is an original graph with privacy information
G2(V, E) is an auxiliary graph with identity information.

Some of the nodes are already mapped in this. These nodes are Seed pair nodes. We need to use these already-mapped nodes to find other seed pairs.



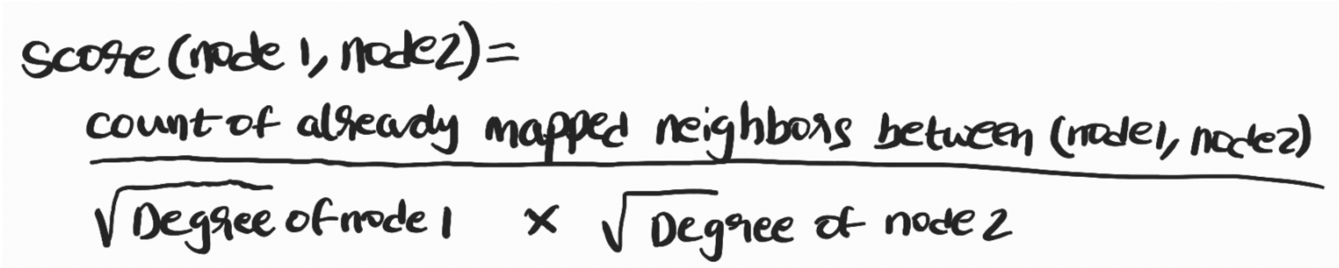
The network's topological structure and the feedback from earlier mapping iterations are used by the algorithm, which intuitively finds new mappings.

In our algorithm, we are importing graph 1 and graph 2 using networkx package by reading the edge list data.

Then we import the seed-pairs data, which is in a txt file in project data and json format in mini-dataset. When we import the json file into a variable, it is directly converted into a dictionary.

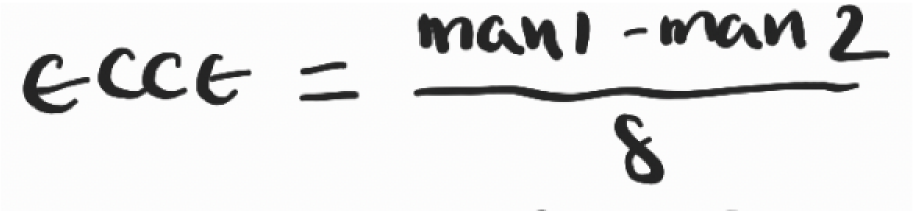
We split the seed-pair dictionary into separate nodes one has all the nodes in the seed-pair list and is graph 1 node. And the other one has all the nodes in the seed-pair list and corresponds to graph 2.

We iterate through each graph 1 node, and we find the degree of that node, then we find the degree of the node in another graph. We consider the neighbors of these nodes and check if these nodes are already mapped and if their pairs exist in the seed pair list. Here we find the unmapped nodes and measure the similarity by calculating the Score.



$$\text{score}(\text{node } 1, \text{node}2) = \frac{\text{count of already mapped neighbors between } (\text{node}1, \text{node}2)}{\sqrt{\text{Degree of node } 1} \times \sqrt{\text{Degree of node } 2}}$$

We find all the scores of these nodes. We store all of these scores into a set, and now we find eccentricity for each iteration.

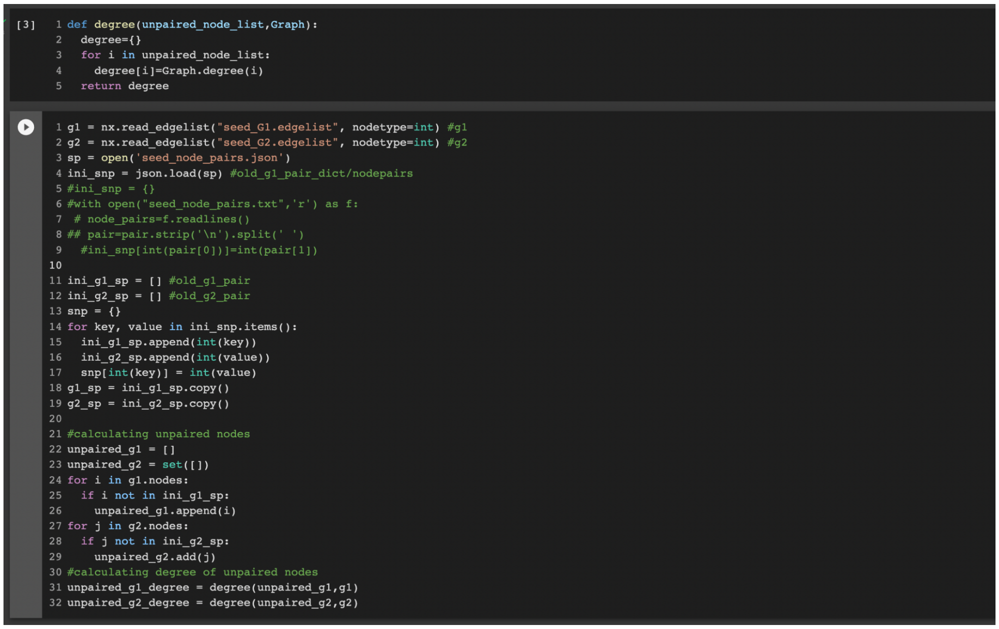


$$ECCE = \frac{\text{max}1 - \text{max}2}{\delta}$$

Ecce = Score(max1)-Score(max2)/Standard Deviation.

We store all the scores in each iteration and find the standard deviation for this set. Then we find the first max score and second max score. We subtract these 2 scores. Then we divide it by the standard deviation. We get eccentricity value. Now we check if ECCE is greater than 0.5, we map Score(max1)'s node to the node of the first graph we are iterating. Do the iteration until convergence. We continue this iteration until there are no other seed pair nodes to map using the above-mentioned algorithm.

After all the iterations, we append all the seed pair nodes to the initial seed-pair dictionary we imported. The final file is this dictionary loaded into a text file.

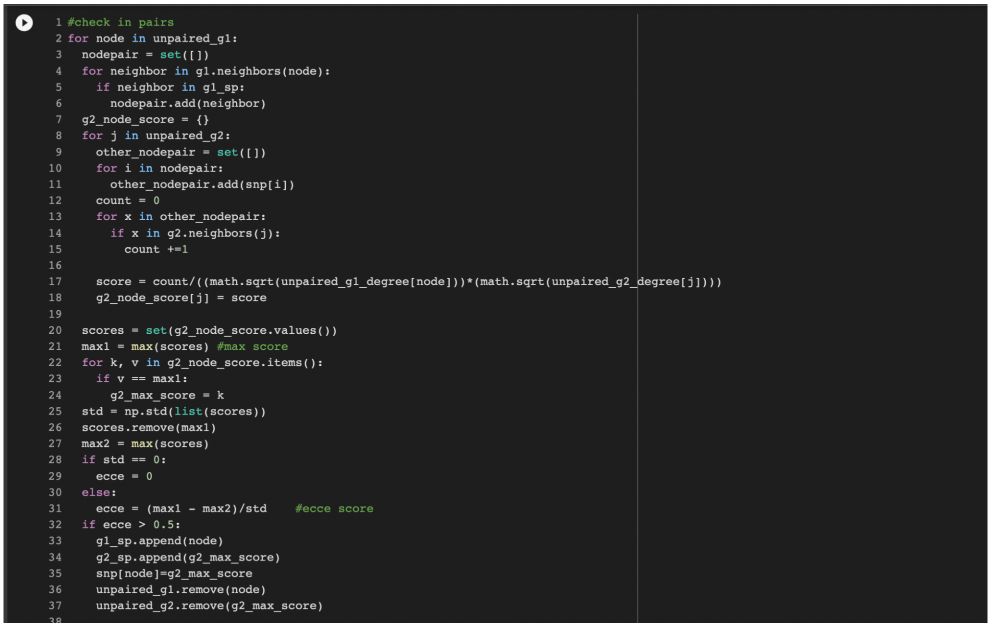


```
def degree(unpaired_node_list,Graph):
  degree={}
  for i in unpaired_node_list:
    degree[i]=Graph.degree(i)
  return degree
```

```
g1 = nx.read_edgelist("seed_G1.edgelist", nodetype=int) #g1
g2 = nx.read_edgelist("seed_G2.edgelist", nodetype=int) #g2
sp = open('seed_node_pairs.json')
ini_snp = json.load(sp) #old_g1_pair_dict/nodepairs
#ini_snp = {}
#with open("seed_node_pairs.txt",'r') as f:
 # node_pairs=f.readlines()
## pair=pair.strip('\n').split(' ')
  #ini_snp[int(pair[0])]=int(pair[1])

ini_g1_sp = [] #old_g1_pair
ini_g2_sp = [] #old_g2_pair
snp = {}
for key, value in ini_snp.items():
  ini_g1_sp.append(int(key))
  ini_g2_sp.append(int(value))
  snp[int(key)] = int(value)
g1_sp = ini_g1_sp.copy()
g2_sp = ini_g2_sp.copy()

#calculating unpaired nodes
unpaired_g1 = []
unpaired_g2 = set([])
for i in g1.nodes:
  if i not in ini_g1_sp:
    unpaired_g1.append(i)
for j in g2.nodes:
  if j not in ini_g2_sp:
    unpaired_g2.add(j)
#calculating degree of unpaired nodes
unpaired_g1_degree = degree(unpaired_g1,g1)
unpaired_g2_degree = degree(unpaired_g2,g2)
```

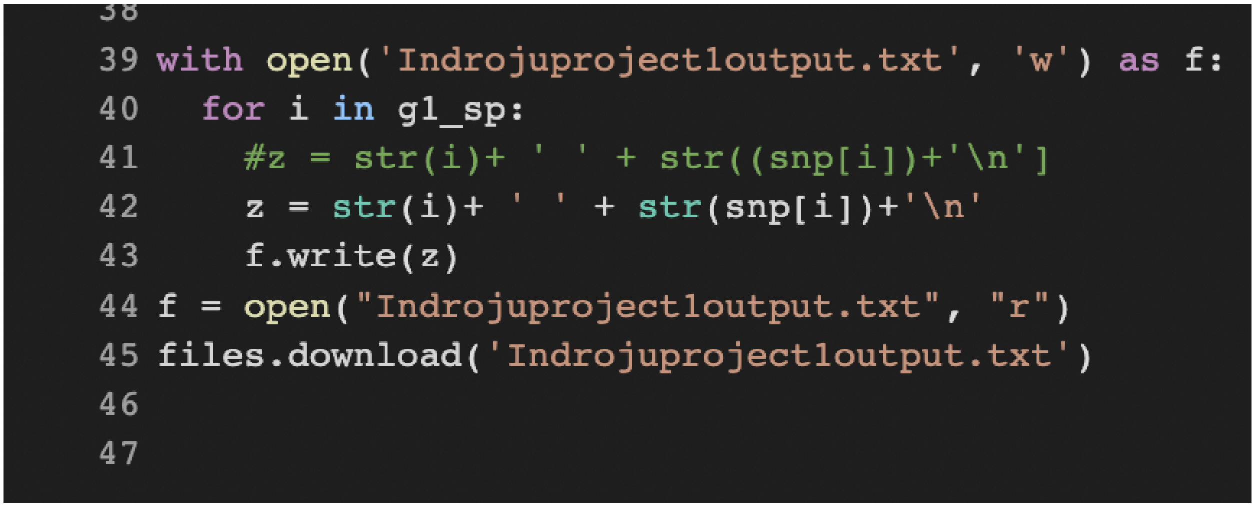


```
#check in pairs
for node in unpaired_g1:
  nodepair = set([])
  for neighbor in g1.neighbors(node):
    if neighbor in g1_sp:
      nodepair.add(neighbor)
  g2_node_score = {}
  for j in unpaired_g2:
    other_nodepair = set([])
    for i in nodepair:
      other_nodepair.add(snp[i])
    count = 0
    for x in other_nodepair:
      if x in g2.neighbors(j):
        count +=1

    score = count/((math.sqrt(unpaired_g1_degree[node]))*(math.sqrt(unpaired_g2_degree[j])))
    g2_node_score[j] = score

  scores = set(g2_node_score.values())
  max1 = max(scores) #max score
  for k, v in g2_node_score.items():
    if v == max1:
      g2_max_score = k
  std = np.std(list(scores))
  scores.remove(max1)
  max2 = max(scores)
  if std == 0:
    ecce = 0
  else:
    ecce = (max1 - max2)/std    #ecce score
  if ecce > 0.5:
    g1_sp.append(node)
    g2_sp.append(g2_max_score)
    snp[node]=g2_max_score
    unpaired_g1.remove(node)
    unpaired_g2.remove(g2_max_score)
```

Here we are writing the output into a text file.



```python
with open('Indrojuproject1output.txt', 'w') as f:
  for i in g1_sp:
    #z = str(i)+ ' ' + str((snp[i])+'\n']
    z = str(i)+ ' ' + str(snp[i])+'\n'
    f.write(z)
f = open("Indrojuproject1output.txt", "r")
files.download('Indrojuproject1output.txt')


```