19ZO02-Social and Economic Network Analysis

Project Report

NETWORK ANALYSIS ON BITCOIN OTC TRUSTABILITY NETWORK

Team Members:

19Z201 AKASH A

19Z204 ASHWIN R

19Z205 BALA BHARAT RAAJ G S

20Z434 SRUTHI S

20Z435 UDHAYAKUMARAN H

BACHELOR OF ENGINEERING

KNOWLEDGE AND SERVICE

1951

Branch: COMPUTER SCIENCE AND ENGINEERING

Of Anna University

PROBLEM STATEMENT

To analyze the Bitcoin OTC trust weighted signed network using various concepts of Social and economic network analysis. The bitcoin users ,called Bitcoin OTC , are very anonymous so there is the need to keep a close track on the users' reputation and past transaction history to find such swindlers and prevent doing any such transaction with them . On the grounds of this , the project aims to apply models to find goodness and fairness of a user and perform a link analysis between the users in the network .

DATASET

People who trade using Bitcoin on a website called Bitcoin OTC are connected through a trust network. On a scale from -10 (total mistrust) to +10 (total trust), members of Bitcoin OTC rate one another.

DATA FORMAT

Each line has one rating, sorted by time, with the following format:

```
SOURCE, TARGET, RATING, TIME
```

where

SOURCE: node id of source, i.e., rater

TARGET: node id of target, i.e., ratee

RATING: the source's rating for the target, ranging from -10 to +10 in steps of 1

TIME: the time of the rating, measured as seconds since Epoch.

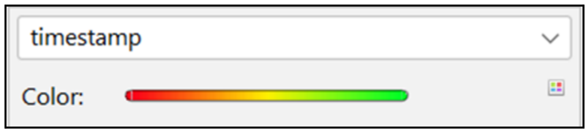
DATASET LINK

https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html

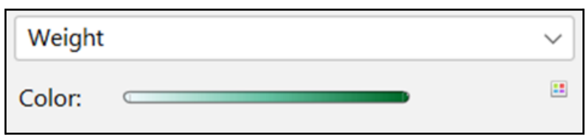
VISUALIZATION OF DATASET

COLOURING:

1. **EDGE COLOUR BASED ON TIMESTAMP VALUES**

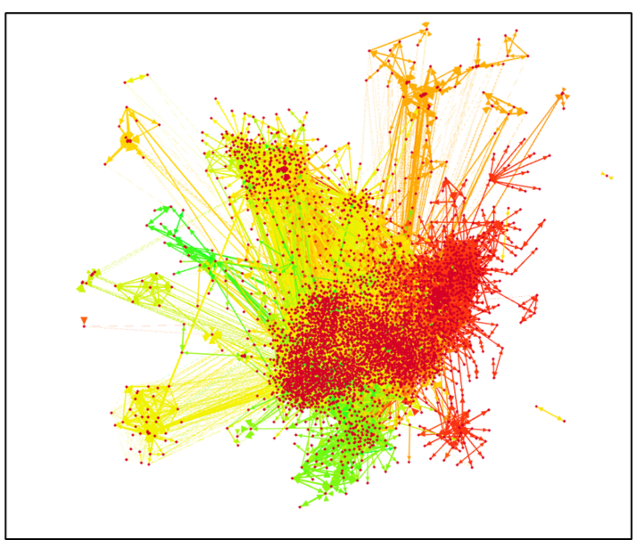


2. **EDGE COLOUR BASED ON TRUSTABILITY VALUES**

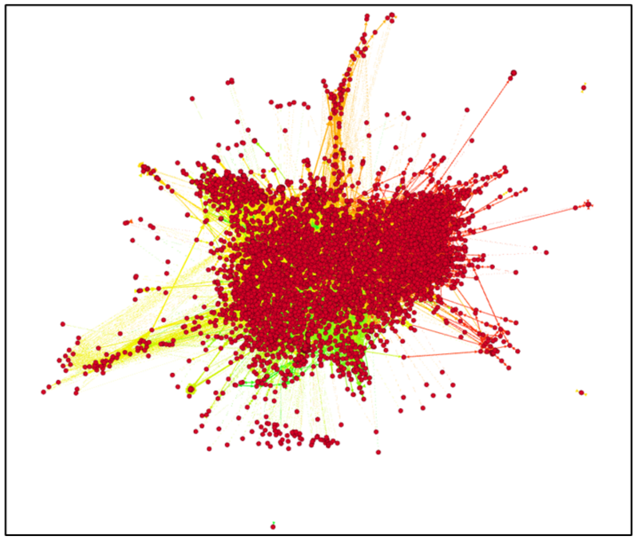


VISUALIZATIONS WITH EDGES BASED ON TIMESTAMP VALUES:

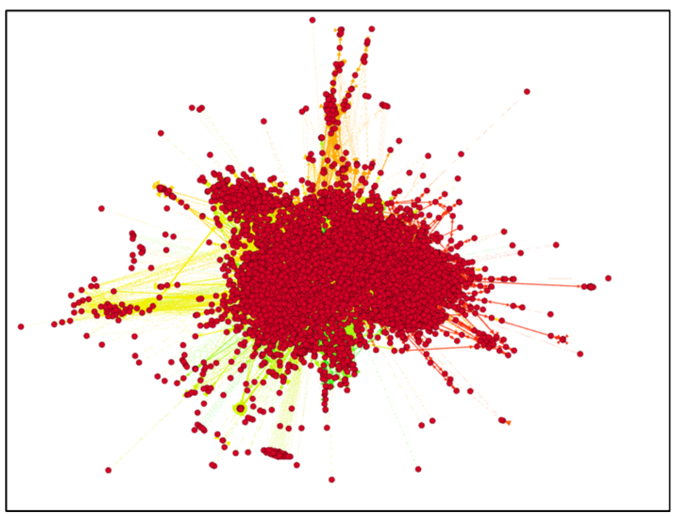
1. **OPEN ORD**



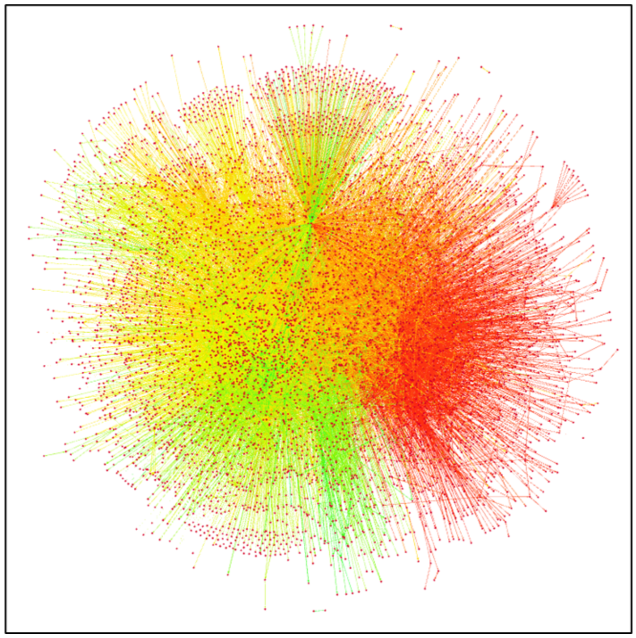
2. FORCED ATLAS



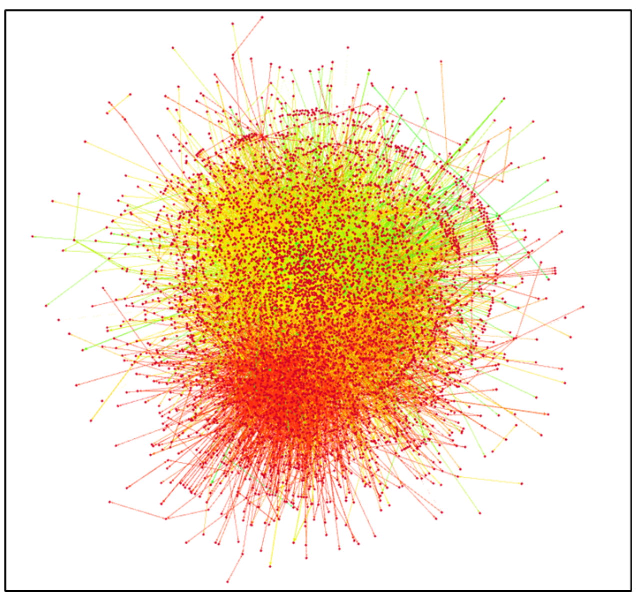
3. FORCED ATLAS 2



4. FRUCHTERMAN REINGOLD



5. YIFAN HU



NETWORK PARAMETERS

Network Parameter	Values
Average Degree	6.052
Average Weighted Degree	72.697
Network Diameter	11
Connected Components	4
Average Clustering Coefficient	0.149
Average Path Length	3.719

PARAMETER VALUES FOR THE NODES IN THE NETWORK

Id	Label	Interval	In-Degree	Out-Degr...	Degree	Weighted In-Degr...	Weighted Out-Deg...	Weighted Degr...	Eccentricity	Closeness Centra...	Harmonic Closeness Centr...	Betweenness Centra...	Component ...	Strongly-Connecte...	Clustering Coeffici...
6			44	40	84	545.0	537.0	1082.0	6.0	0.363682	0.389427	88568.624765	0	1115	0.112458
2			41	45	86	574.0	638.0	1212.0	6.0	0.328706	0.347387	69358.877895	0	1115	0.091437
5			3	3	6	40.0	40.0	80.0	6.0	0.303414	0.315518	0.0	0	1115	1.0
1			226	215	441	3287.0	2798.0	6085.0	5.0	0.418581	0.450476	1555486.335162	0	1115	0.042372
15			13	15	28	163.0	189.0	352.0	6.0	0.309075	0.322033	23555.602727	0	1115	0.073529
4			54	63	117	762.0	868.0	1630.0	6.0	0.357873	0.382743	225907.72811	0	1115	0.087138
3			21	0	21	225.0	0.0	225.0	0.0	0.0	0.0	0.0	0	5	0.140476
13			191	210	401	2442.0	2596.0	5038.0	6.0	0.40552	0.438455	1114323.791117	0	1115	0.038405
16			1	0	1	19.0	0.0	19.0	0.0	0.0	0.0	0.0	0	1111	0.0
10			5	8	13	85.0	144.0	229.0	6.0	0.303982	0.315367	256.490036	0	1115	0.375
7			216	232	448	2990.0	3063.0	6053.0	5.0	0.386364	0.420716	1376579.074592	0	1115	0.024313
21			26	22	48	351.0	332.0	683.0	6.0	0.313566	0.32791	43608.756867	0	1115	0.088624
20			10	0	10	130.0	0.0	130.0	0.0	0.0	0.0	0.0	0	153	0.3
8			3	1	4	50.0	14.0	64.0	6.0	0.295085	0.305292	0.0	0	1115	1.0
17			19	26	45	256.0	278.0	534.0	6.0	0.323612	0.339646	22085.793427	0	1115	0.105911
23			26	18	44	372.0	250.0	622.0	6.0	0.309647	0.323441	67640.29628	0	1115	0.067734
25			113	0	113	1538.0	0.0	1538.0	0.0	0.0	0.0	0.0	0	13	0.055863
26			11	12	23	138.0	140.0	278.0	6.0	0.306821	0.326716	41992.177168	0	1115	0.037879
28			11	7	18	139.0	87.0	226.0	6.0	0.306579	0.327069	12211.273746	0	1115	0.109091
29			35	33	68	457.0	407.0	864.0	6.0	0.325776	0.342567	81092.829012	0	1115	0.066999
31			2	2	4	25.0	27.0	52.0	6.0	0.297094	0.307646	0.0	0	1115	1.0
32			6	6	12	72.0	72.0	144.0	6.0	0.313482	0.32606	1891.519429	0	1115	0.266667
34			3	3	6	36.0	36.0	72.0	6.0	0.31042	0.323276	0.0	0	1115	1.0
35			535	763	1298	6901.0	9267.0	16168.0	6.0	0.414018	0.47793	4912540.070276	0	1115	0.003058
36			33	35	68	425.0	449.0	874.0	6.0	0.336711	0.358641	154022.021714	0	1115	0.0503
37			12	13	25	148.0	160.0	308.0	7.0	0.305602	0.318049	17494.948396	0	1115	0.044872
44			3	2	5	25.0	24.0	49.0	7.0	0.248165	0.25479	69.066862	0	1115	0.333333
39			25	25	50	354.0	349.0	703.0	6.0	0.316845	0.33231	48476.090862	0	1115	0.115385

TOOLS USED

GEPHI

A Java-based visualization programme called Gephi was created. It primarily makes use of raw edge and node graph data to visualize, manipulate, and explore networks and graphs. It is an open-source, free programme. Its visualization engine, OpenGL, is built on top of the Netbeans Platform. It functions on Linux, Mac OS X, and Windows. It is a great resource for data scientists and analysts who are interested in exploring and comprehending graphs. It works with graph data but is similar to Photoshop. In order to uncover hidden patterns, the user manipulates the representation's structures, forms, and colors. The main objective is to make it possible for the user to form an opinion, find hidden patterns, and identify faults and singularities in the structure.

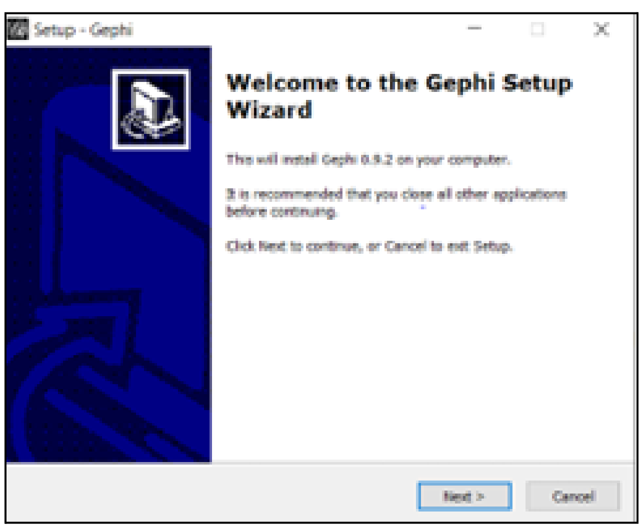
Installing Gephi

Step 1: Ensure that your computer has the most recent Java JRE version installed before downloading Gephi. If not, download it first from this page.

Step 2: Visit the Gephi official website and select "Download Now."

Step 3: Select the right download option for Gephi on either Windows, Mac, or Linux.

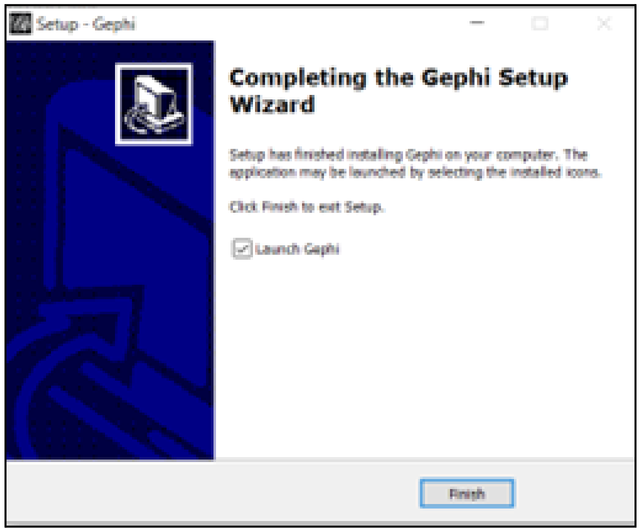
Step 4: After waiting for the download to finish, launch the installer.



Installing initially

Step 5: Continue clicking Next and go through the motions.

Step 6: Click Finish after the installation is finished.



Advantages of Gephi:

1. Extremely Quick: OpenGL is used in its construction. Gephi therefore enables us to operate with very vast networks at very fast speeds. Up to a million elements can be seen in networks. Layout, filter, and drag operations are all real-time.

2. Simple: It is incredibly simple to install and use. Its window interface is its most prominent feature. It is loaded with helpful tools and has a very simple user interface.

3.Modularity: The Gephi programme is divided into various components. Its features are all contained into discrete modules. The fact that each module performs a distinct purpose makes software maintenance much simpler.

4. Simple Data Import: Data import is a fairly simple operation in CSV format.

Disadvantages of Gephi:

1. There is zero connection between the points of view.

2. A few visual hiccups exist.

3. The Graph's navigation could be made easier.

Applications of Gephi:

1. It makes analyses by real-time network manipulation in exploratory data analysis.
2. It is employed to represent biological data patterns.
3. It is used to produce printable, high-quality posters that promote scientific research.

NETWORKX

NetworkX is a set of Python-based tools for building, modifying, and researching the composition, dynamics, and operation of complicated networks. Large complicated networks that are represented as graphs with nodes and edges are studied using this method. We can load and store complex networks using networkx. We are able to create a variety of random and conventional networks, study their structure, create network models, create new network methods, and even sketch them. Setting up the package: network install with pip

PANDAS

Working with "relational" or "labeled" data can be simple and intuitive thanks to the Python package pandas, which offers quick, adaptable, and expressive data structures.It aims to serve as Python's fundamental building block for conducting accurate, real-world data analysis.

SKLEARN

Skearn is the name of the most efficient and dependable Python machine learning package (Skit-Learn). It provides a number of efficient tools for statistical modeling and machine learning. This library was mostly created in Python and is built on NumPy and Matplotlib.

SEABORN

Python's Seaborn package allows you to create statistical visuals. It may examine and comprehend your data with Seaborn. Its charting functions work with dataframes and arrays that include entire datasets, and they internally carry out the semantic mapping and statistical aggregation required to make useful graphs.

CHALLENGES FACED

- The presence of very few features in the data set.
- Access and retrieval of individual values is tough since the data frame is in time series format
- Due to high number of nodes , graph creation was a time consuming way
- The community detection is difficult due to the use of signed weighted graph

CONTRIBUTION

Contribution of Team Members are

Roll no	Name	Contribution
19Z201	Akash A	Link analysis
19Z204	Ashwin R	
19Z205	Bala Bharat Raaj G S	Goodness and fairness of a node
20Z434	Sruthi S	
20Z435	Udhayakumaran H	Creation of data frame representing the relationship between nodes and their goodness and fairness value

ANNEXURE 1

Fairness and goodness of a node

```
import networkx as nx
import math
import pandas as pd
```

```
#function to initialize the fairness and goodness scores of each nodes
def initialize_scores(G):
    fairness = {}
    goodness = {}

    nodes = G.nodes()
    for node in nodes:
        fairness[node] = 1
        try:
            goodness[node] = G.in_degree(node, weight='weight')*1.0/G.in_degree(node)
        except:
            goodness[node] = 0
    return fairness, goodness
```

```
[ ] #function to compute the fairness and Goodness scores
def compute_fairness_goodness(G):
    fairness, goodness = initialize_scores(G)

    nodes = G.nodes()
    iter = 0

    #Running the loop for 100 epochs
    while iter < 100:
        df = 0
        dg = 0

        print('-----------------')
        print("Iteration number", iter)

        print('Updating goodness')

        #updating the goodness score and difference in goodness score
        for node in nodes:
            inedges = G.in_edges(node, data='weight')
            g = 0
            for edge in inedges:
                g += fairness[edge[0]]*edge[2]
```

```
            try:
                dg += abs(g/len(inedges) - goodness[node])
                goodness[node] = g/len(inedges)
            except:
                pass

        print('Updating fairness')

        #updating the fairness score and difference in fairness score
        for node in nodes:
            outedges = G.out_edges(node, data='weight')
            f = 0
            for edge in outedges:
                f += 1.0 - abs(edge[2] - goodness[edge[1]])/2.0
            try:
                df += abs(f/len(outedges) - fairness[node])
                fairness[node] = f/len(outedges)
            except:
                pass

        print('Differences in fairness score and goodness score = %.2f, %.2f' % (df, dg))

        #if both the difference in goodness and fairness score is below a certaing threshold stop the loop
```

```
        if df < math.pow(10, -6) and dg < math.pow(10, -6):     ##threshold here is 10^-6
            break
        iter+=1

    return fairness, goodness
```

```
#Using networkx library for creation and analysis of a bitcoin trust network
G = nx.DiGraph()
df = pd.read_csv("/content/Bitcoin_OTC.csv")
df.head(5)
```

```
#Normalizing the weights between the range of -1 to 1
df["Weight"] = df["Weight"] - 11
df.head(5)
```

```
df["Weight"] = df["Weight"]/10
df.head(5)
```

```
for ind in range(len(df)):
    ## the weight should already be in the range of -1 to 1
    G.add_edge(df["Source"][ind], df["Target"][ind], weight = df["Weight"][ind]) #creation of weighted edges

fairness, goodness = compute_fairness_goodness(G)
print(fairness, goodness)
```

Link Analysis

```
[ ]  G = nx.DiGraph()
```

```
[ ]
```

```
[ ]  with open('./Bitcoin_OTC.csv', 'r') as f:
         data = csv.reader(f)
         headers = next(data)
         for row in tqdm(data):
           G.add_node(int(row[0]))
           G.add_node(int(row[1]))
           if G.has_edge(int(row[0]), int(row[1])):
             print(row[0], row[1])
           G.add_edge(row[0], row[1], weight=int(row[2])-11, timestamp=int(row[3]))
```

```
35592it [00:00, 140809.07it/s]
```

```
[ ]  print(G.number_of_nodes())
     print(G.number_of_edges())
```

```
5881
35592
```

```
!pip install thresholdclustering==1.1
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: thresholdclustering==1.1 in /usr/local/lib/python3.7/dist-packages (1.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from thresholdclustering==1.1) (1.21.6)
Requirement already satisfied: networkx in /usr/local/lib/python3.7/dist-packages (from thresholdclustering==1.1) (2.6.3)
```

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
Adj=nx.to_numpy_matrix(G)
cos_Adj=cosine_similarity(Adj.T)
G=nx.from_numpy_matrix(cos_Adj)
pos = nx.spring_layout(G)
weights = np.array([G[u][v]['weight'] for u,v in G.edges()])*5
```

```
import thresholdclustering
import matplotlib.cm as cm
from community import community_louvain
import matplotlib.pyplot as plt
cluster_function = community_louvain.best_partition
partition, alpha = thresholdclustering.thresholdclustering.best_partition(G, cluster_function=cluster_function)

cmap = cm.get_cmap('viridis', max(partition.values()) + 1)
nx.draw_networkx_nodes(G, pos, partition.keys(), node_size=20,
                       cmap=cmap, node_color=list(partition.values()))
nx.draw_networkx_edges(G, pos, alpha=0.2,width=weights)
plt.show()
```

```
print(set(partition.values()))
```

```
import networkx as nx
```

```
import math
```

```
import pandas as pd
```

```
import random
```

```
#function to initialize the fairness and goodness scores of each nodes
def initialize_scores(G):
    fairness = {}
    goodness = {}

    nodes = G.nodes()
    for node in nodes:
        fairness[node] = 1
        try:
            goodness[node] = G.in_degree(node, weight='weight')*1.0/G.in_degree(node)
        except:
            goodness[node] = 0
    return fairness, goodness
```

```python
#function to compute the fairness and Goodness scores
def compute_fairness_goodness(G):
    fairness, goodness = initialize_scores(G)

    nodes = G.nodes()
    iter = 0
    print(len(nodes))

    #Running the loop for 100 epochs
    while iter < 100:
        df = 0
        dg = 0

        #print('-----------------')
        #print("Iteration number", iter)

        #print('Updating goodness')

        #updating the goodness score and difference in goodness score
        for node in nodes:
            inedges = G.in_edges(node, data='weight')
            g = 0
            for edge in inedges:
                g += fairness[edge[0]]*edge[2]

            try:
                dg += abs(g/len(inedges) - goodness[node])
                goodness[node] = g/len(inedges)
            except:
                pass

        #print('Updating fairness')
```

```python
        #updating the fairness score and difference in fairness score
        for node in nodes:
            outedges = G.out_edges(node, data='weight')
            f = 0
            for edge in outedges:
                f += 1.0 - abs(edge[2] - goodness[edge[1]])/2.0
            try:
                df += abs(f/len(outedges) - fairness[node])
                fairness[node] = f/len(outedges)
            except:
                pass

        print('Differences in fairness score and goodness score = %.2f, %.2f' % (df, dg))

        #if both the difference in goodness and fairness score is below a certaing threshold stop the loop

        if df < math.pow(10, -6) and dg < math.pow(10, -6):     ##threshold here is 10^-6
            break
        iter+=1

    return fairness, goodness
```

```
G = nx.DiGraph()
```

```
df = pd.read_csv('Bitcoin_OTC.csv')
```

```
df.head(5)
```

```
#Normalizing the weights between the range of -1 to 1
df["Weight"] = df["Weight"] - 11
df.head(5)
```

```
df["Weight"] = df["Weight"]/10
df.head(5)
```

```
for ind in range(len(df)):
    ## the weight should already be in the range of -1 to 1
    G.add_edge(df["Source"][ind], df["Target"][ind], weight = df["Weight"][ind]) #creation of weighted edges
```

```
for ind in range(len(df)):
    ## the weight should already be in the range of -1 to 1
    G.add_edge(df["Source"][ind], df["Target"][ind], weight = df["Weight"][ind]) #creation of weighted edges

fairness, goodness = compute_fairness_goodness(G)
dic = {}
#print(list(fairness))
dic['fairness'] = list(fairness)
print('-----------')
#print(list(goodness))
dic['goodness'] = list(goodness)
```

```
dic
```

```
node = df['Source']
```

```
node1 = set(node)
```

```
len(node1)
```

4814

```
list(node1)
```

```
data = pd.read_csv('fair_and_good.csv')
```

```
def convert_to_csv():
    fair_good = pd.DataFrame(columns=['Node', 'Fairness', 'Goodness'])
    nodes = G.nodes()
    for node in nodes:
        dic = {}
        dic['Node'] = int(node)
        dic['Fairness'] = fairness[node]
        dic['Goodness'] = goodness[node]
        temp = pd.Series(dic.copy())
        fair_good = pd.concat([fair_good, pd.DataFrame([temp], columns=temp.index)], axis=0).reset_index(drop=True)
    fair_good["Node"] = fair_good["Node"].astype('int')
    fair_good.to_csv('fair_and_good.csv')
```

```
from collections import defaultdict
```

```
src = list(df['Source'])
dest = list(df['Target'])
```

```python
class Graph:

    def __init__(self) -> None:
        self.graph = defaultdict(list)


    def addEdge(self,u,v,w):
        self.graph[u].append((v,w))
```

```python
g = Graph()
k=0
for i,j in zip(src,dest):
    g.addEdge(str(i), str(j),data[k])
    k+=1
k=0
```

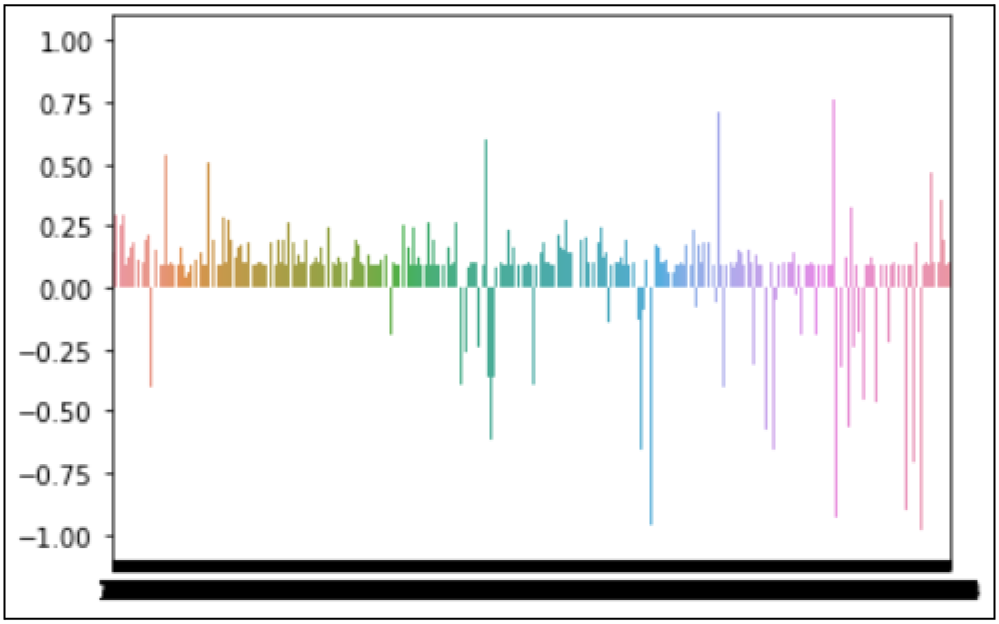
```python
data = pd.read_csv('fair_and_good.csv')
```

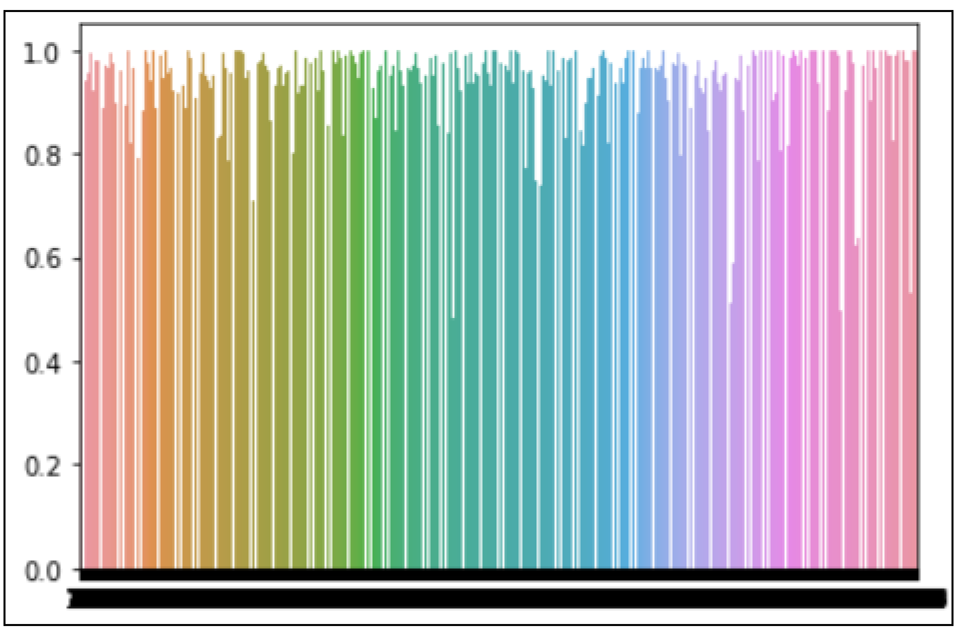
```python
a = 191
```

ANNEXURE 2

Snapshots of the Output

<u>Fairness and goodness of a node</u>



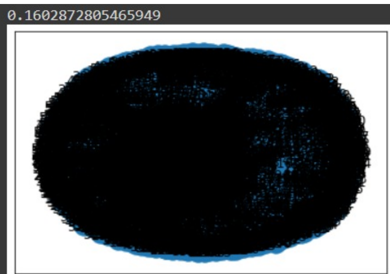


	Node	Fairness	Goodness
0	6	0.895726	0.130176
1	2	0.893744	0.269531
2	5	0.940111	0.214168
3	1	0.922436	0.323933
4	15	0.960489	0.144465

```
The nodes which have negative goodness score (Fault nodes):
[3, 44, 61, 75, 62, 179, 204, 260, 310, 315, 410, 423, 467, 463, 472, 512, 566, 594, 642, 574, 672, 726, 766, 787, 805, 824, 832, 870, 895, 906, 957, 984, 958, 1074,
The number of faulty nodes in the network is:  828
```

Link analysis

```
0.1602872805465949
```



```
5881
Differences in fairness score and goodness score = 382.26, 0.00
Differences in fairness score and goodness score = 34.49, 102.19
Differences in fairness score and goodness score = 2.39, 6.94
Differences in fairness score and goodness score = 0.39, 1.06
Differences in fairness score and goodness score = 0.10, 0.24
Differences in fairness score and goodness score = 0.03, 0.06
Differences in fairness score and goodness score = 0.01, 0.02
```

Plagiarism Report

Similarity Statistics

Similarity Statistics [what is this?]

Total number of documents: 1

Number of documents which can be processed: 1

Number of documents which cannot be processed: 0

Show 10 entries Search:

Entry	Document	Status	Similarity	Action
1	SENA_REPORT_Team_1_.pdf	processed	5/63=7.90%	View details

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

References

https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html

Kumar, S., Spezzano, F., Subrahmanian, V. S., & Faloutsos, C. (2016). Edge weight prediction in weighted signed networks. *2016 IEEE 16th International Conference on Data Mining (ICDM)*. https://doi.org/10.1109/icdm.2016.0033

https://www.sciencedirect.com/topics/computer-science/link-prediction#:~:text=4.7.&text=Link%20prediction%20tells%20about%20the,et%20al.%2C%202014)

https://www.sciencedirect.com/topics/computer-science/community-detection4

https://gephi.org/users/install/

https://networkx.org/

https://www.w3schools.com/python/pandas/default.asp

https://www.tutorialspoint.com/scikit_learn/index.htm

https://www.w3schools.com/python/numpy/numpy_random_seaborn.asp

https://www.analyticsvidhya.com/blog/2020/01/link-prediction-how-to-predict-your-future-connections-on-facebook/