Graph Partitioning , Link Prediction and Most Influential Node Analysis on Twitter Network

SNA Project

Members

- Bokkisam Charansai - AM.EN.U4CSE19314
- Musunuru Varun - AM.EN.U4CSE19336
- Ashwin R - AM.EN.U4CSE19343
- Vasantha Gopikrishna - AM.EN.U4CSE19359

In [2]:

```
!wget https://raw.githubusercontent.com/charansai123/SNA/main/Twitter%20-%20Analysis/graph.csv
```

```
--2022-06-04 23:58:27--  https://raw.githubusercontent.com/charansai123/SNA/main/Twitter%20-%20Analysis/graph.csv
 (https://raw.githubusercontent.com/charansai123/SNA/main/Twitter%20-%20Analysis/graph.csv)
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.108.133, 185.199.109.13
3, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6286011 (6.0M) [text/plain]
Saving to: ‘graph.csv’

graph.csv           100%[===================>]   5.99M  --.-KB/s    in 0.08s

2022-06-04 23:58:27 (76.4 MB/s) - ‘graph.csv’ saved [6286011/6286011]
```

In [3]:

```
!pip install pyvis
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheels/publ
ic/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Collecting pyvis
  Downloading pyvis-0.2.1.tar.gz (21 kB)
Requirement already satisfied: jinja2>=2.9.6 in /usr/local/lib/python3.7/dist-packages (from pyvis) (2.11.3)
Requirement already satisfied: networkx>=1.11 in /usr/local/lib/python3.7/dist-packages (from pyvis) (2.6.3)
Requirement already satisfied: ipython>=5.3.0 in /usr/local/lib/python3.7/dist-packages (from pyvis) (5.5.0)
Collecting jsonpickle>=1.4.1
  Downloading jsonpickle-2.2.0-py2.py3-none-any.whl (39 kB)
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-packages (from ipython>=5.3.0->pyvis) (4.
4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.7/dist-packages (from ipython>=5.3.0->pyvis)
 (0.7.5)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.7/dist-packages (from ipython>=5.3.0->pyv
is) (57.4.0)
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/python3.7/dist-packages (from ipython>=5.3.0->py
vis) (0.8.1)
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in /usr/local/lib/python3.7/dist-packages (from ipython
>=5.3.0->pyvis) (1.0.18)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dist-packages (from ipython>=5.3.0->pyvi
s) (5.1.1)
Requirement already satisfied: pexpect in /usr/local/lib/python3.7/dist-packages (from ipython>=5.3.0->pyvis) (4.8.
0)
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-packages (from ipython>=5.3.0->pyvis) (2.
6.1)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2>=2.9.6->pyvi
s) (2.0.1)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from jsonpickle>=1.4.1
->pyvis) (4.11.4)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-packages (from prompt-toolkit<2.0.0,>=1.
0.4->ipython>=5.3.0->pyvis) (1.15.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packages (from prompt-toolkit<2.0.0,>=1.0.4
->ipython>=5.3.0->pyvis) (0.2.5)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->jsonpi
ckle>=1.4.1->pyvis) (3.8.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3.7/dist-packages (from importlib-m
etadata->jsonpickle>=1.4.1->pyvis) (4.2.0)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.7/dist-packages (from pexpect->ipython>=5.
3.0->pyvis) (0.7.0)
Building wheels for collected packages: pyvis
  Building wheel for pyvis (setup.py) ... done
  Created wheel for pyvis: filename=pyvis-0.2.1-py3-none-any.whl size=23688 sha256=f6be928c221e5246f8f8c25557a760b1
ab54e7414db3f19faa7cbcf84cc8e61e
  Stored in directory: /root/.cache/pip/wheels/2a/8f/04/6340d46afc74f59cc857a594ca1a2a14a1f4cbd4fd6c2e9306
Successfully built pyvis
Installing collected packages: jsonpickle, pyvis
Successfully installed jsonpickle-2.2.0 pyvis-0.2.1
```

In [4]:

```
from pyvis.network import Network
import networkx as nx
import community
import pandas as pd
import numpy
```

In [10]:

```
twi_data = pd.read_csv("graph.csv")
twi_data
```

Out[10]:

	Source	Target
0	1070010671918665728	2329921
1	1070010671918665728	382134761
2	1070010671918665728	814015332
3	1070010671918665728	934041262608584704
4	1070010671918665728	304928205
...	...	...
272004	895773623570636800	837635778117259264
272005	895773623570636800	1186648442
272006	895773623570636800	1066288106
272007	895773623570636800	2789457827
272008	895773623570636800	1115181426

272009 rows × 2 columns

In [11]:

```
twi_data=twi_data.sample(frac=0.0037, replace=False, random_state=1)
twi_data
```

Out[11]:

	Source	Target
184476	2701384105	4217153597
239908	831966756277153793	14131652
40283	18026546	490512649
76593	1154392441346240513	2414056867
199207	1008153102	188204899
...	...	...
97502	2932027868	1315396027
186606	1054502754494939137	531122860
20339	257111136	2679687798
217168	628230869	2329921
61419	258069365	4873826663

1006 rows × 2 columns

Generation of visual network graph with pyvis

In [12]:
```
twi_net = Network(height='750px', width='100%', bgcolor='#222222', font_color='white')
twi_net.barnes_hut()
sources = twi_data['Source']
targets = twi_data['Target']

edge_data = zip(sources, targets)

for e in edge_data:
    src = e[0]
    dst = e[1]

    twi_net.add_node(src, src, title=str(src))
    twi_net.add_node(dst, dst, title=str(dst))
    twi_net.add_edge(src,dst,value=1)


neighbour_map = twi_net.get_adj_list()
twi_net.nodes
```

Out[12]:
```
[{'font': {'color': 'white'},
  'id': 2701384105,
  'label': 2701384105,
  'shape': 'dot',
  'title': '2701384105'},
 {'font': {'color': 'white'},
  'id': 4217153597,
  'label': 4217153597,
  'shape': 'dot',
  'title': '4217153597'},
 {'font': {'color': 'white'},
  'id': 831966756277153793,
  'label': 831966756277153793,
  'shape': 'dot',
  'title': '831966756277153793'},
 {'font': {'color': 'white'},
  'id': 14131652,
  'label': 14131652,
  'shape': 'dot',
  'title': '14131652'},
```

In [13]:
```
# add neighbor data to node hover data
from IPython.core.display import display, HTML

for node in twi_net.nodes:
    node['title'] += ' Neighbors:<br>' + '<br>'.join(str(neighbour_map[node['id']]))
    node['value'] = len(neighbour_map[node['id']])

twi_net.show('twi.html')
```

In [14]:
```
G=[]
for ind in twi_data.index:
  k=(twi_data['Source'][ind],twi_data['Target'][ind])
  G.append(k)
G[150:300]
```

```
 (1167145117452570626, 734866162949967873),
 (1406419332, 437797632),
 (733181629993091073, 3088703037),
 (1058691719175254016, 722001114992926720),
 (314217400, 818907239612379136),
 (699395297118523392, 2375509748),
 (1479301693, 120814510),
 (837770828, 244481174),
 (976105732994228225, 1944845792),
 (1167145117452570626, 33555058),
 (2909490018, 582161546),
 (753497077431369728, 1632913393),
 (156204739, 14538236),
 (1026404288564736000, 701015997537501185),
 (204721301, 90258002),
 (3295496994, 2293415520),
 (50517429, 460489687),
 (18406335, 6144162),
 (115677576, 520778935)]
```

In [15]:
```
g=nx.Graph(G)
```

In [16]:
```
g.number_of_nodes()
```

Out[16]:
```
1337
```

In [17]:
```
g.number_of_edges()
```

Out[17]:
```
1006
```

In [18]:
```
nx.degree_histogram(g)
```

Out[18]:
```
[0, 963, 222, 85, 31, 21, 6, 3, 2, 0, 0, 2, 1, 0, 1]
```

Degree centrality

In [19]:
```
dic=nx.degree_centrality(g)
Keymax = max(zip(dic.values(), dic.keys()))[1]
print("Node with maximum degree centrality:",Keymax)
print("Max degree centrality:",dic[Keymax])
dic
```

```
Node with maximum degree centrality: 56341402
Max degree centrality: 0.010479041916167666
```

Betweenness centrality

In [20]:
```
dic=nx.betweenness_centrality(g, normalized=True, endpoints=False)
Keymax = max(zip(dic.values(), dic.keys()))[1]
print("Node with maximum betweenness centrality:",Keymax)
print("Max betweenness centrality:",dic[Keymax])
dic
```

```
Node with maximum betweenness centrality: 29692085
Max betweenness centrality: 0.008731974253739712
```

Closeness centrality

In [21]:

```
dic=nx.closeness_centrality(g)
Keymax = max(zip(dic.values(), dic.keys()))[1]
print("Node with maximum closeness centrality:",Keymax)
print("Max closeness centrality:",dic[Keymax])
dic
```

```
Node with maximum closeness centrality: 29692085
Max closeness centrality: 0.021973684210526315
```

Eigenvector centrality

In [22]:

```
dic=nx.eigenvector_centrality(g)
Keymax = max(zip(dic.values(), dic.keys()))[1]
print("Node with maximum eigenvector centrality:",Keymax)
print("Max eigenvector centrality:",dic[Keymax])
dic
```

```
Node with maximum eigenvector centrality: 56341402
Max eigenvector centrality: 0.689430038857041
```

Communities Detection

Girvan Newman

In [23]:

```
import matplotlib.pyplot as plt
import networkx as nx
from networkx.algorithms.community.centrality import girvan_newman
```

In [24]:

```
communities = girvan_newman(g)
```

In [25]:

```python
node_groups = []
for com in next(communities):
  node_groups.append(list(com))

print(node_groups)
print(len(node_groups))
```

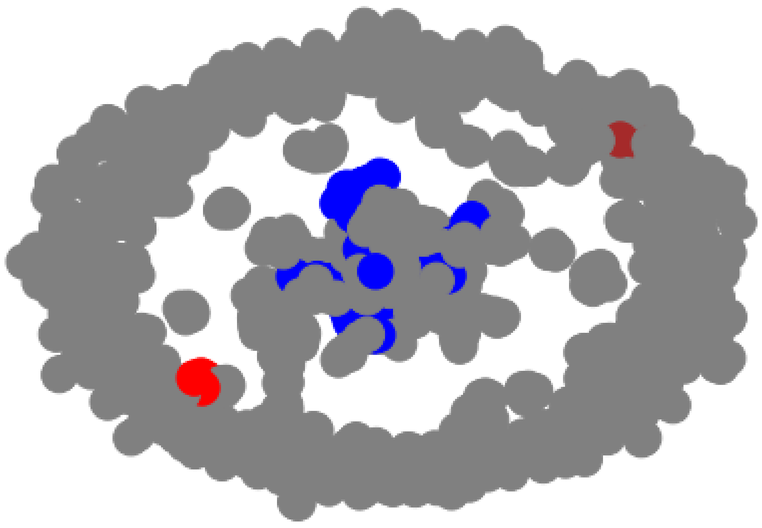
```
506792775680], [948959573205045248, 5815392], [162605310, 468784726], [1026299734116392960, 926506527216865281],
[987199729086820352, 12685362], [307880202, 64698978], [2226551180, 73241590], [62621489, 101824202, 87718038585
5254532, 890710574], [2919801897, 423847851], [60910828, 3696200237], [318335269, 2176606374], [1104234200, 2872
512304], [95509514, 2893971], [371328288, 709767126530265088], [1059202250, 3367784062], [4926409928, 8934337007
00106752], [50796520, 588596248], [1869245306, 44373964], [138098380, 389411405], [260656385, 122464459], [95774
5963753406464, 710165317507530756, 112333725], [14810361, 26730180], [1664996929, 3456615865], [34570530, 948743
33], [50095170, 16428756], [68376272, 2387451451], [4023660472, 746152190], [16191522, 48417486], [34339576, 851
08450], [61001305, 141328091], [911193721936457728, 710244765], [299603744, 2797997959], [1038442496, 77820030
1], [22671862, 17083959], [440645901, 47881149], [978706420999876608, 1069907880441192448], [1454215074, 1506010
765], [504803544, 122724789], [566812856, 9700072, 34653414], [4896916113, 98764925, 77502415], [2269684160, 104
5763448670486529], [2220997760, 1627053481], [871869837315686400, 707433649], [792416615463813120, 770012574494625792], [143751299, 1635900535], [1092012778762747904, 566542654], [31051381, 93476253], [17013577, 102345161],
[4120890970, 146287709, 37243261], [3198332901, 4852816143], [774529682033213440, 2221799833], [1004142841008754
688, 28718742], [92415827, 2374887246], [891677275799908352, 21305051], [3131557785, 392695595], [2819255259, 21
90466679], [2451308594, 23233540], [785851877690867712, 951703932], [61555249, 634216642], [41518369, 59192420
6], [835953569668939776, 868966324751523845], [1002621855046471681, 1616964710], [4654667776, 258069365, 4873826
663], [1856509825, 2726806519], [3671492717, 2797948887], [198389338, 147650011], [995412536, 146129222], [13554
17364, 1019337134120062983], [3486259881, 205618337], [870427994350907394, 262489445]]
334
```

In [26]:

```python
color_map = []
for node in g:
    if node in node_groups[0]:
        color_map.append('blue')
    elif node in node_groups[1]:
        color_map.append('green')
    elif node in node_groups[2]:
        color_map.append('red')
    elif node in node_groups[3]:
        color_map.append('yellow')
    elif node in node_groups[4]:
        color_map.append('brown')
    else:
      color_map.append('grey')

print(len(node_groups[0]))
print(len(node_groups[1]))
print(len(node_groups[2]))
print(len(node_groups[3]))
print(len(node_groups[4]))
nx.draw(g, node_color=color_map)
plt.show()
```

```
103
2
9
2
6
```



Kernighan–Lin algorithm

In [27]:

```python
from networkx.algorithms import community
```

In [28]:

```
ker_lin_community=community.kernighan_lin_bisection(g, partition=None, max_iter=10, weight='weight', seed=None)
list(ker_lin_community)
```

Out[28]:

```
[{743913,
  821962,
  1190041,
  1246421,
  2893971,
  5811092,
  6334772,
  7559192,
  7685632,
  8517882,
  9700072,
  11094912,
  11914552,
  12090952,
  12169762,
  12402812,
  12685362,
  13256982,
  14066472,
```

In [42]:

```
print(len(ker_lin_community))
print(len(list(ker_lin_community)))
k=list(ker_lin_community)
print(len(k[0]))
print(len(k[1]))
```

```
2
2
668
669
```

Clauset-Newman-Moore greedy modularity maximization

In [30]:

```
from networkx.algorithms.community import greedy_modularity_communities
c = greedy_modularity_communities(g)
print(len(c))
print(len(c[0]))
c[:3]
```

```
340
46
```

Louvain Community Detection

In [31]:
```
from community import community_louvain
c = community_louvain.best_partition(g)
c
```

Out[31]:
```
{2701384105: 0,
 4217153597: 0,
 831966756277153793: 1,
 14131652: 1,
 18026546: 2,
 490512649: 2,
 1154392441346240513: 3,
 2414056867: 3,
 1008153102: 4,
 188204899: 4,
 355654498: 5,
 1540384460: 5,
 3001054244: 6,
 2284174986: 6,
 284120528: 7,
 7559192: 7,
 764121685905866752: 90,
 373682248: 90,
 1973058649: 9,
```

Link Prediction

In [5]:
```
twi_data = pd.read_csv("graph.csv")
twi_data
```

Out[5]:

	Source	Target
0	1070010671918665728	2329921
1	1070010671918665728	382134761
2	1070010671918665728	814015332
3	1070010671918665728	934041262608584704
4	1070010671918665728	304928205
...	...	...
272004	895773623570636800	837635778117259264
272005	895773623570636800	1186648442
272006	895773623570636800	1066288106
272007	895773623570636800	2789457827
272008	895773623570636800	1115181426

272009 rows × 2 columns

In [6]:
```
G=[]
for ind in twi_data.index:
  k=(twi_data['Source'][ind],twi_data['Target'][ind])
  G.append(k)
G[150:300]
```

```
 (814015332, 305504617),
 (814015332, 22615177),
 (814015332, 33826996),
 (814015332, 4723010406),
 (814015332, 159565864),
 (814015332, 1070010671918665728),
 (814015332, 1423906682),
 (814015332, 3196161958),
 (814015332, 3001907483),
 (814015332, 822240036280111105),
 (814015332, 76084600),
 (814015332, 973636307774828551),
 (814015332, 567627175),
 (814015332, 1128158782263947264),
 (814015332, 2872512304),
 (814015332, 885237745198747648),
 (814015332, 3216966611),
 (814015332, 727455850521059328),
 (814015332, 1139160970721406976),
 (814015332, 1157733977605603328),
```

In [7]:
```
g=nx.Graph(G)
```

In [8]:
```
g.number_of_nodes()
```

Out[8]:
```
6757
```

In [10]:
```
g.number_of_edges()
```
Out[10]:
```
219977
```

Resource Allocation Index

In [19]:
```
preds = nx.resource_allocation_index(g)
cnt=0
m=0
for u, v, p in preds:
    if p>m:
      m=p
      pair=(u,v)
    print(f"({u}, {v}) -> {p:.8f}")
    cnt=cnt+1
    if cnt==100000:
      break
print(f"Max value : {pair} -> {m:.8f}")
```
```
(637075471, 612534082) -> 0.00158730
(637075471, 2532017986) -> 0.00033400
(637075471, 2607793988) -> 0.00756880
(637075471, 311052101) -> 0.00353357
(637075471, 2267645766) -> 0.00000000
(637075471, 321783625) -> 0.00082589
(637075471, 995133259) -> 0.00318649
(637075471, 860849995) -> 0.00000000
(637075471, 1668630349) -> 0.02671139
(637075471, 857214014618730496) -> 0.00032938
(637075471, 41846608) -> 0.04246780
(637075471, 830375762) -> 0.00097019
(637075471, 2343077714) -> 0.00326685

(637075471, 459294549) -> 0.01260873
(637075471, 274515797) -> 0.00066338
(637075471, 26347355) -> 0.00000000
(637075471, 2283652956) -> 0.00273198
(637075471, 2180630366) -> 0.00381304
Max value : (935785016155570177, 56341402) -> 4.19698064
```

Jaccard Coefficient

In [20]:
```
preds = nx.jaccard_coefficient(g)
cnt=0
m=0
for u, v, p in preds:
    if p>m:
      m=p
      pair=(u,v)
    print(f"({u}, {v}) -> {p:.8f}")
    cnt=cnt+1
    if cnt==100000:
      break
print(f"Max value : {pair} -> {m:.8f}")
```
```
(637075471, 24102720) -> 0.00440529
(637075471, 612534082) -> 0.00364964
(637075471, 2532017986) -> 0.00462963
(637075471, 2607793988) -> 0.02510460
(637075471, 311052101) -> 0.00202840
(637075471, 2267645766) -> 0.00000000
(637075471, 321783625) -> 0.00787402
(637075471, 995133259) -> 0.01587302
(637075471, 860849995) -> 0.00000000
(637075471, 1668630349) -> 0.04899135
(637075471, 857214014618730496) -> 0.00452489
(637075471, 41846608) -> 0.03361345
(637075471, 830375762) -> 0.00888889
(637075471, 2343077714) -> 0.01339286
(637075471, 459294549) -> 0.00970874
(637075471, 274515797) -> 0.00746269
(637075471, 26347355) -> 0.00000000
(637075471, 2283652956) -> 0.02066116
(637075471, 2180630366) -> 0.02631579
Max value : (699296562002919424, 227524663) -> 0.53333333
```

Adamic–Adar index

In [22]:

```
preds = nx.adamic_adar_index(g)
cnt=0
m=0
for u, v, p in preds:
    if p>m:
      m=p
      pair=(u,v)
    print(f"({u}, {v}) -> {p:.8f}")
    cnt=cnt+1
    if cnt==100000:
      break
print(f"Max value : {pair} -> {m:.8f}")
```

```
(637075471, 24102720) -> 0.12493183
(637075471, 612534082) -> 0.15514171
(637075471, 2532017986) -> 0.12493183
(637075471, 2607793988) -> 0.88024610
(637075471, 311052101) -> 0.17713390
(637075471, 2267645766) -> 0.00000000
(637075471, 321783625) -> 0.25621249
(637075471, 995133259) -> 0.54624835
(637075471, 860849995) -> 0.00000000
(637075471, 1668630349) -> 2.52670375
(637075471, 857214014618730496) -> 0.12471478
(637075471, 41846608) -> 1.35268730
(637075471, 830375762) -> 0.26166397
(637075471, 2343077714) -> 0.43002441
(637075471, 459294549) -> 0.54126984
(637075471, 274515797) -> 0.24964660
(637075471, 26347355) -> 0.00000000
(637075471, 2283652956) -> 0.65954215
(637075471, 2180630366) -> 0.80186636
Max value : (935785016155570177, 56341402) -> 78.79761316
```

Preferential attachment

In [23]:

```
preds = nx.preferential_attachment(g)
cnt=0
m=0
for u, v, p in preds:
    if p>m:
      m=p
      pair=(u,v)
    print(f"({u}, {v}) -> {p:.8f}")
    cnt=cnt+1
    if cnt==100000:
      break
print(f"Max value : {pair} -> {m:.8f}")
```

```
(637075471, 612534082) -> 13054.00000000
(637075471, 2532017986) -> 642.00000000
(637075471, 2607793988) -> 6634.00000000
(637075471, 311052101) -> 59920.00000000
(637075471, 2267645766) -> 1284.00000000
(637075471, 321783625) -> 8988.00000000
(637075471, 995133259) -> 8988.00000000
(637075471, 860849995) -> 1070.00000000
(637075471, 1668630349) -> 32100.00000000
(637075471, 857214014618730496) -> 1712.00000000
(637075471, 41846608) -> 6848.00000000
(637075471, 830375762) -> 2782.00000000
(637075471, 2343077714) -> 2782.00000000
(637075471, 459294549) -> 20972.00000000
(637075471, 274515797) -> 11984.00000000
(637075471, 26347355) -> 1284.00000000
(637075471, 2283652956) -> 7062.00000000
(637075471, 2180630366) -> 4280.00000000
Max value : (935785016155570177, 56341402) -> 1320660.00000000
```

Common Neighbor and Centrality based Parameterized Algorithm(CCPA)

In [24]:

```
preds = nx.common_neighbor_centrality(g)
cnt=0
m=0
for u, v, p in preds:
    if p>m:
      m=p
      pair=(u,v)
    print(f"({u}, {v}) -> {p:.8f}")
    cnt=cnt+1
    if cnt==100000:
      break
print(f"Max value : {pair} -> {m:.8f}")
```

```
(637075471, 24102720) -> 676.50000000
(637075471, 612534082) -> 676.50000000
(637075471, 2532017986) -> 676.50000000
(637075471, 2607793988) -> 680.50000000
(637075471, 311052101) -> 676.50000000
(637075471, 2267645766) -> 450.46666667
(637075471, 321783625) -> 677.30000000
(637075471, 995133259) -> 678.90000000
(637075471, 860849995) -> 450.46666667
(637075471, 1668630349) -> 689.30000000
(637075471, 857214014618730496) -> 676.50000000
(637075471, 41846608) -> 682.10000000
(637075471, 830375762) -> 677.30000000
(637075471, 2343077714) -> 678.10000000
(637075471, 459294549) -> 678.10000000
(637075471, 274515797) -> 677.30000000
(637075471, 26347355) -> 450.46666667
(637075471, 2283652956) -> 679.70000000
(637075471, 2180630366) -> 680.50000000
Max value : (935785016155570177, 56341402) -> 974.90000000
```

Page Rank

Most Influential Node through Page rank

In [25]:

```
pgr=nx.pagerank(g)
```

In [26]:

```
pgr_max = max(pgr, key=pgr.get)
print("Maximum page rank node:",pgr_max)
print("Max page rank value:",pgr[pgr_max])
```

```
Maximum page rank node: 579299426
Max page rank value: 0.010002694145263193
```

Voterank

In [27]:

```
vr=nx.voterank(g)
```

In [28]:

```
print(len(vr))
vr[0:5]
```

```
3505
```

Out[28]:

```
[56341402, 579299426, 582161546, 90258002, 229910053]
```