REPORT

December 21, 2017

```
In [25]: import json
         from os.path import expanduser
         import itertools
         from collections import defaultdict
         import math
         import networkx as nx
         import heapq
         import matplotlib.pyplot as plt
         import Modules as m
```

0.0.1 Load the data

```
In [2]: db = expanduser("reduced_dblp.json")
        database = json.loads(open(db, 'r').read())
```

DATABASE = List of PUBBLICATION
PUBBLICATION -> 6 DICT= authors, id_conference, id_conference_int, id_publication, id_publication_int, title
AUTHORS -> List of DICT (About 10500 in reduced json) -> {(author:Name , author_id:ID)}

0.0.2 1.

Below we created the dictionary of authors and add in key as a node in a graph

```
In [38]: G = nx.Graph()
         dictAutor = m.pubblicationDictionary(database)
         for j in dictAutor.keys():
             G.add_node(j[1], id = j[1], author=j[0])
         print(nx.info(G))
```

```
Name:
Type: Graph
Number of nodes: 7406
Number of edges: 0
Average degree:   0.0000
```

Below we created the dictionary of publications with parsing the author's name beacause there were some names unreadable

```
In [39]: dict_publ = {}
         for elem in range(len(database)):
             d = database[elem]['authors']
             for author in range(len(d)):
                 if "&" in d[author]['author']:
                     pass
                 else:
                     try:
                         dict_publ[database[elem]['id_publication']].append((d[author]['author
                     except:
                         dict_publ[database[elem]['id_publication']] = [(d[author]['author'], c

         dew = defaultdict(list)
         for keys in dictAutor.keys():
             for i in range(len(dictAutor[keys])):
                 try:
                     dew[keys].append(dictAutor[keys][i][0])
                 except:
                     dew[keys]=dictAutor[keys][i][0]
```

We added the edges

```
In [40]: for k,v in dict_publ.items():
             for i in itertools.combinations(v,2):
                 G.add_edge(i[0][1],i[1][1], pubblication=k[0], pubblication_int = k[1], weight

         print(nx.info(G))
```

```
Name:
Type: Graph
Number of nodes: 7406
Number of edges: 15368
Average degree:   4.1501
```

0.0.3 2(a)

- In the code below we created the subgraph of conference

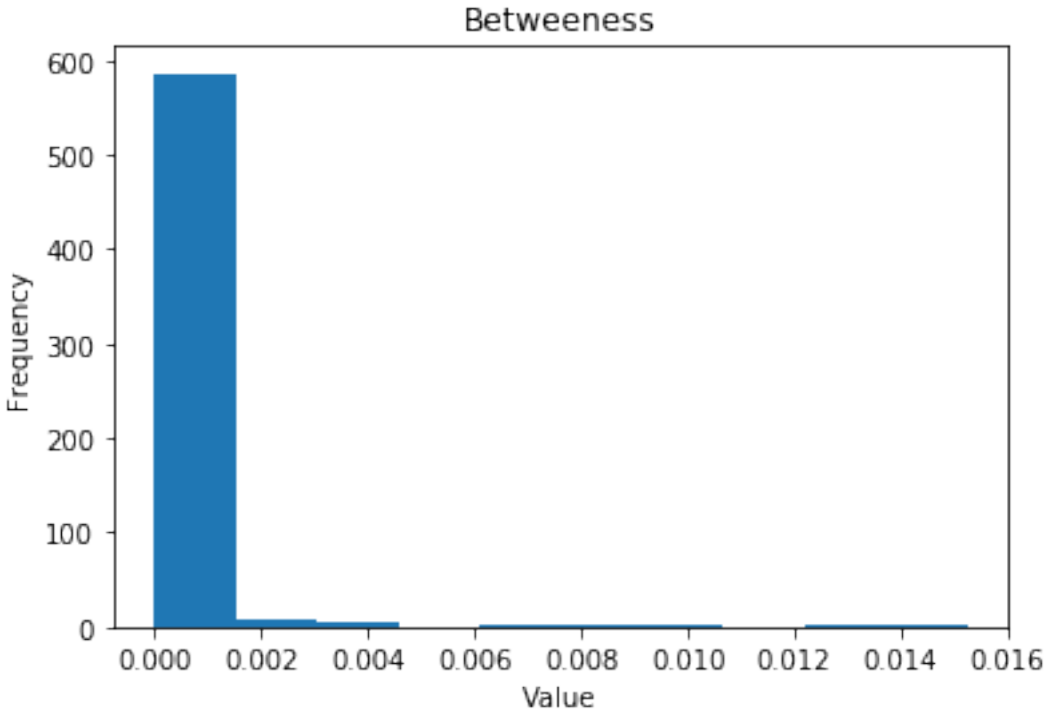
```
In [45]: h = m.graphConference(database, 3345)
```

- Centralities measures

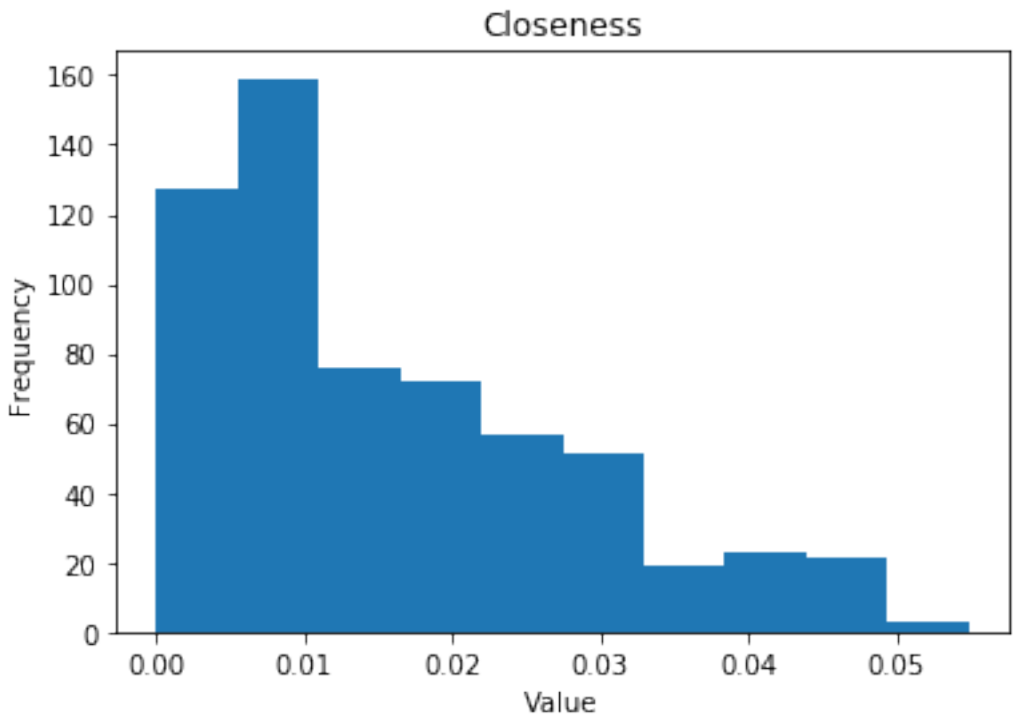
```
In [46]: betweness = nx.betweenness_centrality(h)
         closeness = nx.closeness_centrality(h)
         degree = nx.degree(h)
```

```
In [42]: betwn_val = []
         for k in betweness.keys():
```

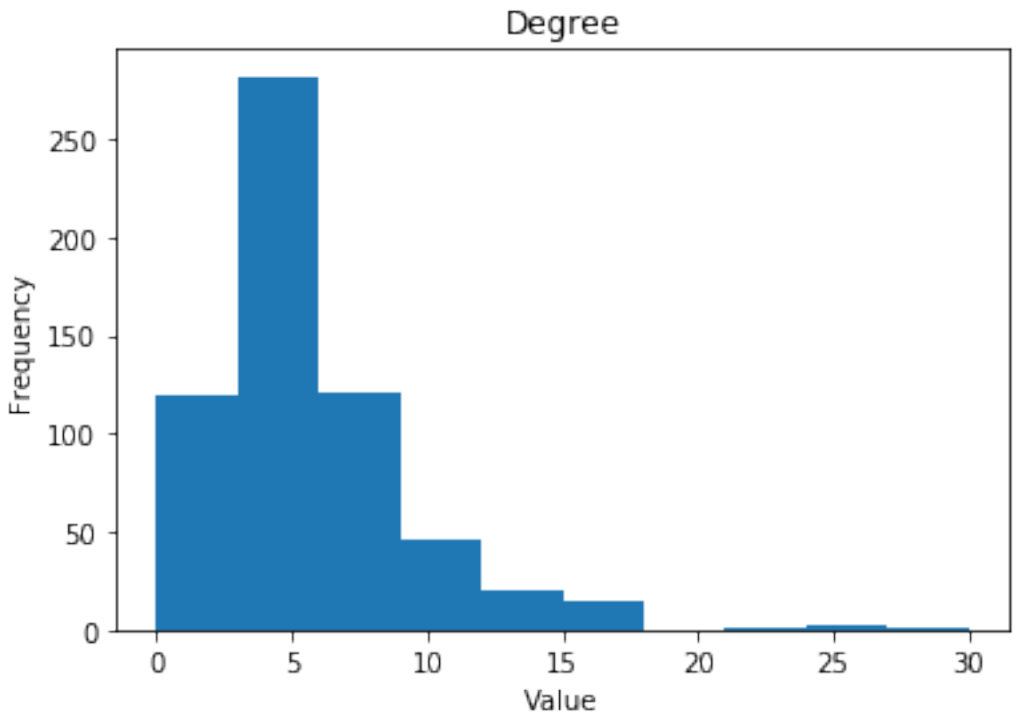
```
    betwn_val.append(betweness[k])
plt.hist(betwn_val)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Betweeness')
plt.show()
```



In [43]:
```
closeness_val = []
for k in closeness.keys():
    closeness_val.append(closeness[k])
plt.hist(closeness_val)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Closeness')
plt.show()
```




```
In [44]: degree_val = []
         for k in degree.keys():
             degree_val.append(degree[k])
         plt.hist(degree_val)
         plt.xlabel('Value')
         plt.ylabel('Frequency')
         plt.title('Degree')
         plt.show()
```

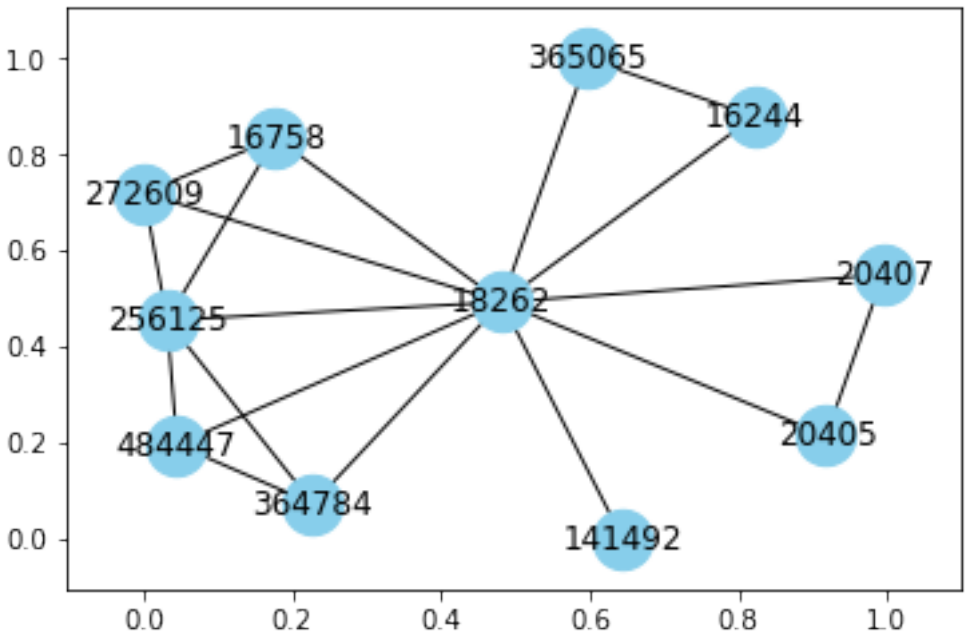


0.0.4 2(b)

```
In [33]: G_sub=m.author_dist("nicola barbieri",2)
```

Below we plotted the subgraph induced by the nodes thathave hop distance at most equal to 2 from the input node "nicola barbieri"

```
In [34]: import matplotlib.pyplot as plt
         nx.draw_networkx(G_sub,with_labels=True, node_size=500, node_color="skyblue", pos=nx.s
         plt.show()
```



0.0.5 3(a)

we wrote a function that computes the minimum weight from one input node to Aris

```
In [35]: m.distance_to_aris(18262)
```

```
Out[35]: 5.914316239316239
```

0.0.6 3(b)

This function is too slow, so we decide to write another function that compute the minimum distance between a set of nodes and another set of nodes (not for alle the nodes of the full graph)

```
In [47]: #m.Group_number([18262,256176,141492,256125])
```

```
In [55]: def Group_number(s, list_nodes):
             G_n={}

             for node in  s:
                 gr=[]
                 for i in list_nodes:
                     if node!=i:
                         gr.append( m.Dijkstra(G, node,i))

                 G_n[node]=min(gr)

             return(G_n)
```

```
In [56]: Group_number([18262,256176],[141492,256125])
```

```
Out[56]: {18262: 0.6, 256176: 6.514316239316239}
```