Report

December 22, 2017

```
In [16]: import json
         import networkx as nx
         import Modules
         import matplotlib.pyplot as plt
         import seaborn as sb
         sb.set_style('darkgrid')
```

1 - Graph In this section we loaded the data from the file `full_dblp.json` and we created a new dictionary called `authors` in which we keep only the authors' name and id and the set of thier publications.

```
In [10]: with open('full_dblp.json', 'r') as f:
             data = json.load(f)

         data = Modules.convert_names(data)

         #each element of data represents a different publication,
         #with all the authors that collaborated
         data[0]
```

```
Out[10]: {'authors': [{'author': 'pierre seimandi', 'author_id': 1},
           {'author': 'guillaume dufour', 'author_id': 2},
           {'author': 'françois rogier', 'author_id': 3}],
          'id_conference': 'conf/iccs/2010',
          'id_conference_int': 1,
          'id_publication': 'journals/procedia/seimandidr10',
          'id_publication_int': 1,
          'title': 'a two scale model of air corona discharges.'}
```

```
In [11]: #new dictionary
         authors = Modules.create_dict(data)
```

We created a graph in which each node represents an author and two nodes are linked if the two authors share, at least, one publication. Moreover, the nodes have as attributes the author name and his publications and the edges are weighted through the Jaccard similarity.

It turns out that the nodes are 904664 and the edges are 3679331.

```
In [12]: G = Modules.create_graph(data, authors)
         nx.info(G)
```

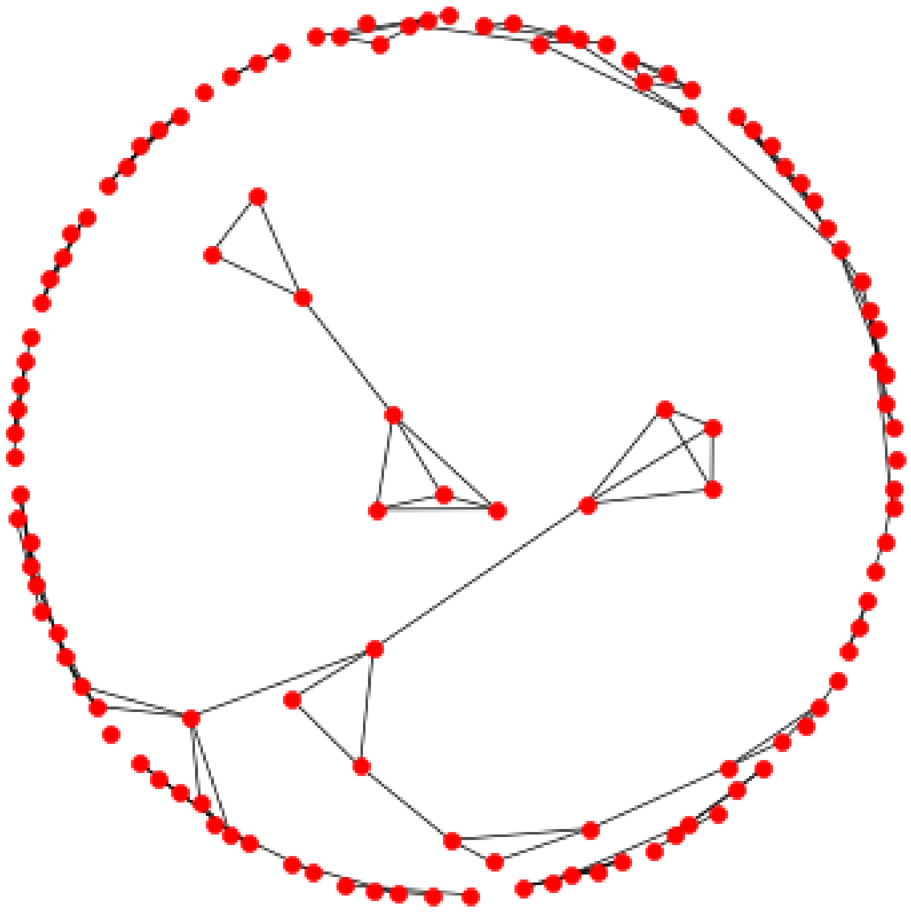
```
Out[12]: 'Name: \nType: Graph\nNumber of nodes: 904664\nNumber of edges: 3679331\nAverage degre
```

2 - Subgraph of a conference. Given a conference id we derived from the original graph G a subgraph in which the nodes are the authors who published at the input conference at least once and then we computed some centralities measures like the degree, closeness and betweeness.

For example, given in input the conference id 3052 corresponding to the conference whose title is *'is there a best quality metric for graph clusters?'* the resulting subgraph has 120 nodes and 129 edges and the plot of the graph is shown below. Analyzing the plot is clear that not all the nodes are linked to each others so, even if the authors joined the same conference, they did not partecipate at the same publication.

```
In [13]: author_sublist= Modules.sub_conference_nodes(3052, data)
         #create a subgraph
         conf_G = G.subgraph(author_sublist)
         nx.info(conf_G)

Out[13]: 'Name: \nType: Graph\nNumber of nodes: 120\nNumber of edges: 129\nAverage degree:   2

In [21]: plt.clf()
         plt.figure(num=None, figsize=(8,8), dpi=50)
         nx.draw(conf_G, node_shape= '.')
         plt.show()
```

```
/Users/silvia/anaconda3/lib/python3.6/site-packages/networkx/drawing/nx_pylab.py:126: Matplotli
    Future behavior will be consistent with the long-time default:
    plot commands add elements without first clearing the
    Axes and/or Figure.
  b = plt.ishold()
/Users/silvia/anaconda3/lib/python3.6/site-packages/networkx/drawing/nx_pylab.py:138: Matplotli
    Future behavior will be consistent with the long-time default:
    plot commands add elements without first clearing the
    Axes and/or Figure.
  plt.hold(b)
/Users/silvia/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py:917: UserWarning: a
  warnings.warn(self.msg_depr_set % key)
/Users/silvia/anaconda3/lib/python3.6/site-packages/matplotlib/rcsetup.py:152: UserWarning: axe
  warnings.warn("axes.hold is deprecated, will be removed in 3.0")


<matplotlib.figure.Figure at 0x1a1241e080>
```

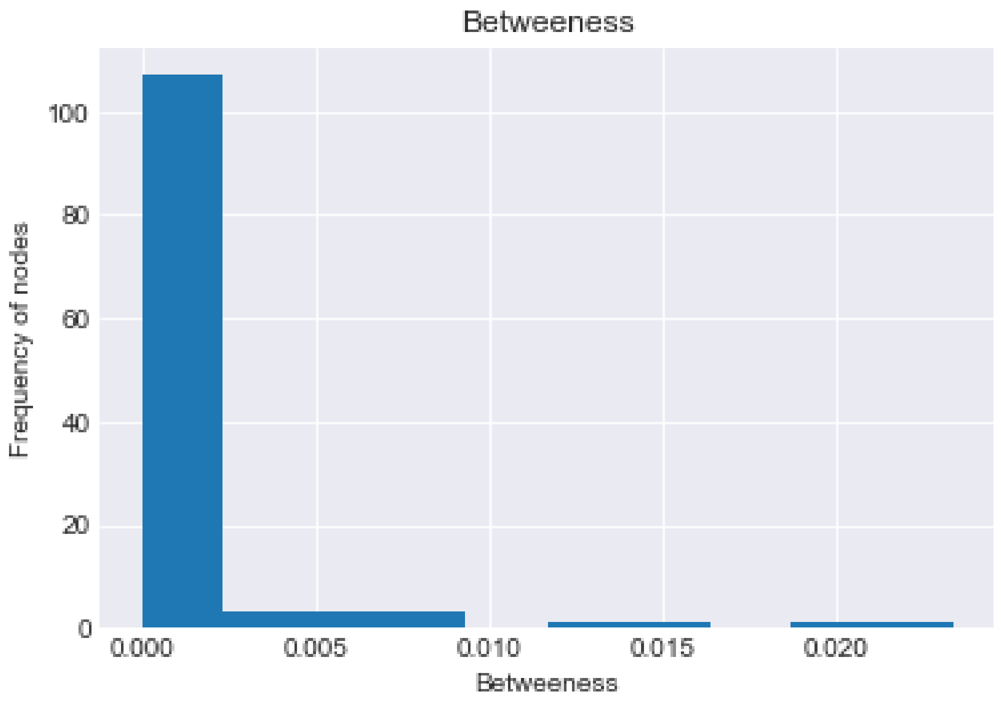


```
In [22]: #betweeness
         betweenness = nx.betweenness_centrality(conf_G)
         #closeness
         closeness = nx.closeness_centrality(conf_G)
         #degree
         degree = nx.degree(conf_G)
```

Betweeness

The betweenness centrality is a measure of centrality in a graph based on shortest paths. In this case, the betweenness centrality of a node v is the sum of the fraction of all-pairs shortest paths that pass through v. The networkx function returns a dictionary and we realized the histogram of the dictionary values. As we can see from the plot below there are more than 100 nodes with betweeness centrality close to 0 and this means that there are a lot of shortest paths passing through two nodes but only few of them pass through a specific node v.

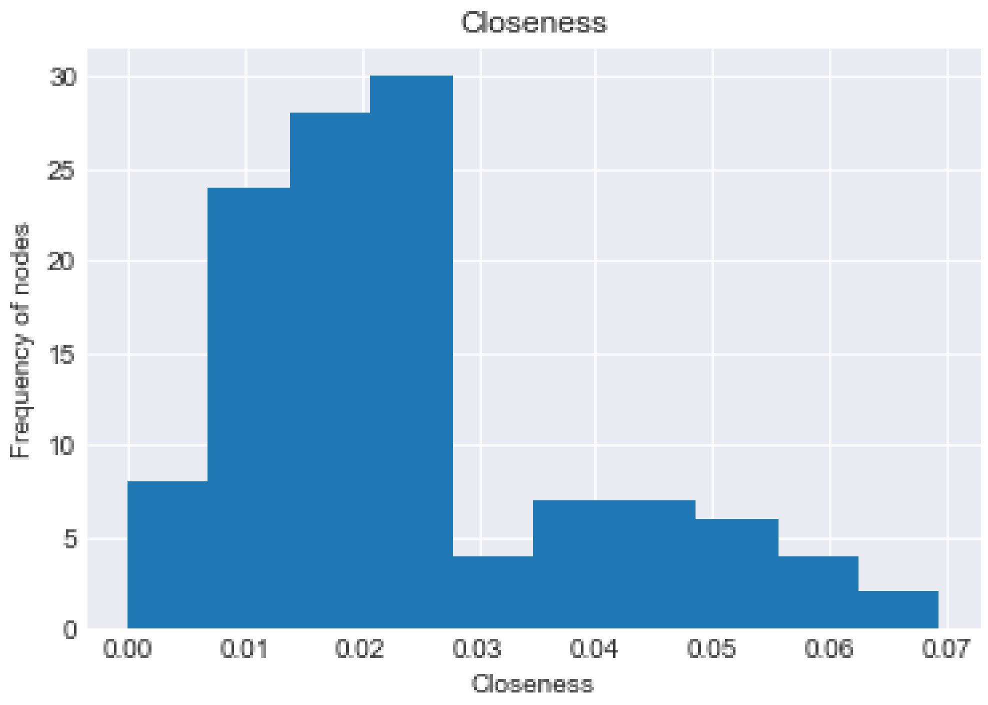
```
In [29]: values=[]
         for x in betweenness.keys():
             values.append(betweenness[x])
         plt.hist(values)
         plt.xlabel('Betweeness')
         plt.ylabel('Frequency of nodes')
         plt.title('Betweeness')
         plt.show()
```



Closeness

The closeness centrality of a node u is the reciprocal of the sum of the shortest path distances from u to all $n - 1$ other nodes. The histogram below shows that most of the nodes have a closeness centrality between 0.01 and 0.03 and this means that the sum of the shortest path distances from a node u to all $n - 1$ other nodes should be a large values.

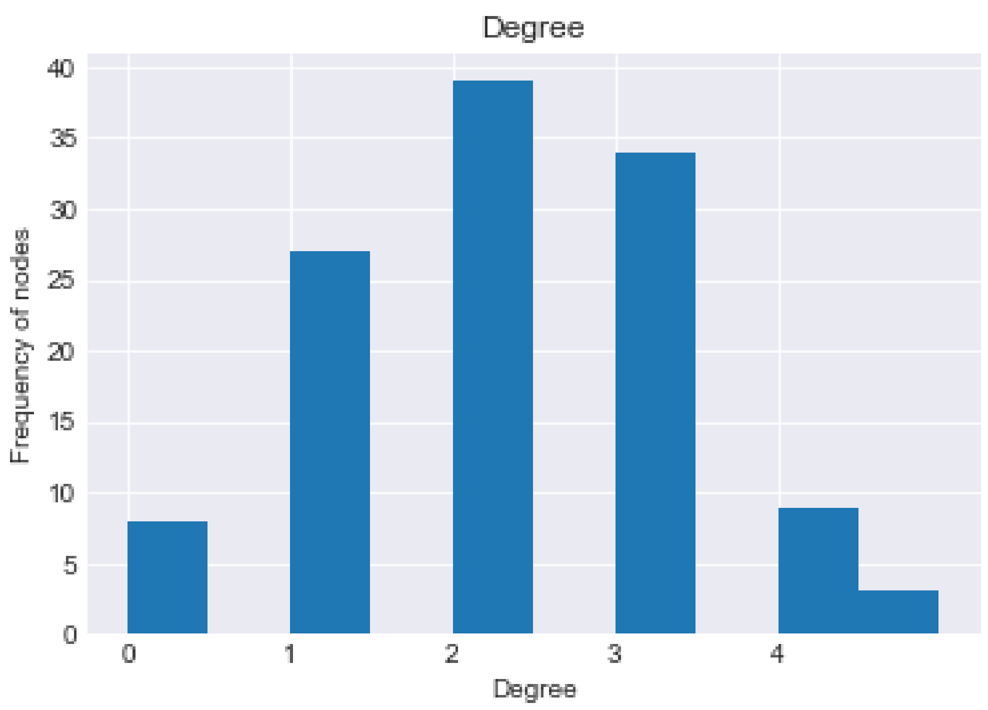
```
In [30]: values=[]
         for x in closeness.keys():
             values.append(closeness[x])
         plt.hist(values)
         plt.xlabel('Closeness')
         plt.ylabel('Frequency of nodes')
         plt.title('Closeness')
         plt.show()
```



Degree

The degree of a node v is the number of edges adjacent to that node. The histogram below shows that more than an half of nodes have 2 or 3 adjacent nodes whereas only 5 nodes have more than 4 adjacent nodes.

```
In [31]: values=[]
         for x in degree.keys():
             values.append(degree[x])
         plt.hist(values)
         plt.xlabel('Degree')
         plt.ylabel('Frequency of nodes')
         plt.title('Degree')
         plt.xticks(range(min(degree.values()), max(degree.values())))
         plt.show()
```



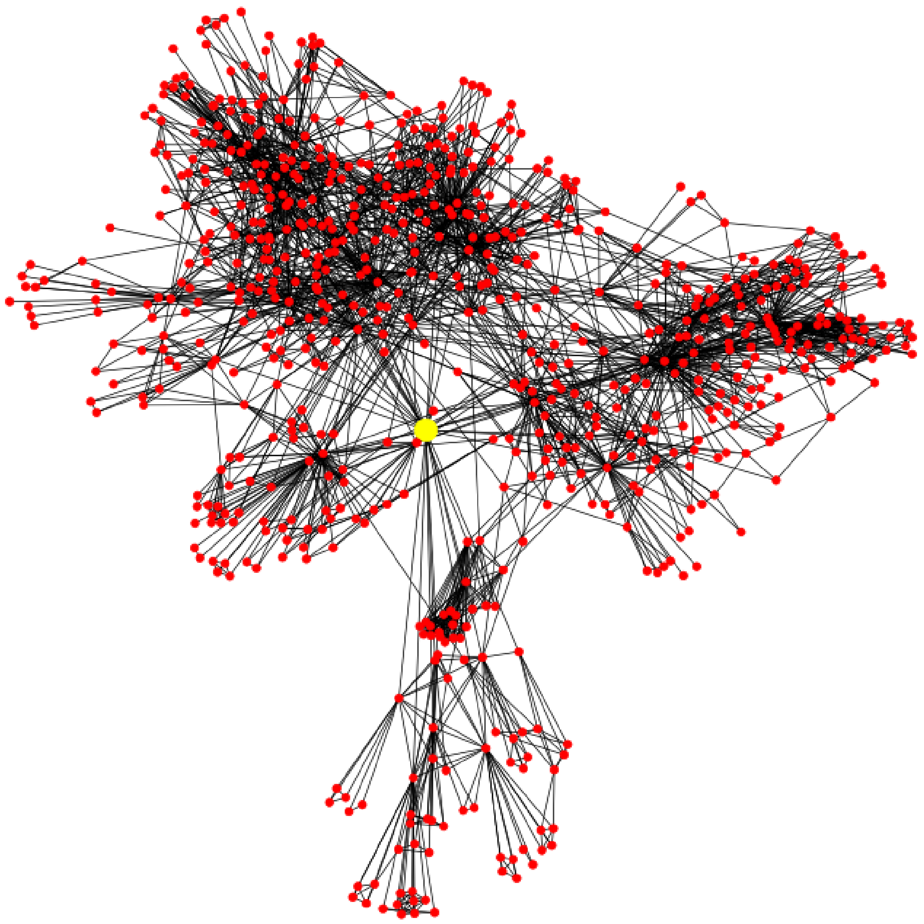
2 - Subgraph given an author id In this section, starting from a given node (author id) we made a subgraph in which there are all the nodes that are at hop distance at most equal to a specific distance (number of edges) given in input.

For example, if we consider *Aris* and we want all the authors linked to him that are at hop distance 2 we obtain a subgraph with 705 nodes and 2826 edges as we can see below. All the node are marked in red instead the node representing Aris is in yellow.

```
In [32]: # Aris ID: 256176
         aris_id = 256176
         author_id = aris_id
         distance = 2
         nodes_list = Modules.sub_authors_nodes(G, author_id, distance)
         #create a subgraph
         author_G = G.subgraph(nodes_list)
         nx.info(author_G)

Out[32]: 'Name: \nType: Graph\nNumber of nodes: 705\nNumber of edges: 2826\nAverage degree:   8

In [42]: for node in author_G.nodes():
             if (node == aris_id):
                 color = 'yellow'
                 node_size=2000
             else:
                 color = 'red'
```

```
                node_size=250
            author_G.node[node]['color'] = color
            author_G.node[node]['node_size']= node_size

        plt.clf()
        plt.figure(num=None, figsize=(15,15), dpi=50)
        nx.draw(author_G, node_shape= '.', node_color = list(nx.get_node_attributes(author_G,
               node_size=list(nx.get_node_attributes(author_G,'node_size').values()))
        plt.show()
```

```
/Users/silvia/anaconda3/lib/python3.6/site-packages/networkx/drawing/nx_pylab.py:126: Matplotli
    Future behavior will be consistent with the long-time default:
    plot commands add elements without first clearing the
    Axes and/or Figure.
  b = plt.ishold()
/Users/silvia/anaconda3/lib/python3.6/site-packages/networkx/drawing/nx_pylab.py:138: Matplotli
    Future behavior will be consistent with the long-time default:
    plot commands add elements without first clearing the
    Axes and/or Figure.
  plt.hold(b)
/Users/silvia/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py:917: UserWarning: a
  warnings.warn(self.msg_depr_set % key)
/Users/silvia/anaconda3/lib/python3.6/site-packages/matplotlib/rcsetup.py:152: UserWarning: axe
  warnings.warn("axes.hold is deprecated, will be removed in 3.0")


<matplotlib.figure.Figure at 0x1a341ca630>
```



3 - Dijkstra Here we evaluated the Dijkstra algorithm to get the shortest path between Aris and another author given in input.

For example, if we are interested in the shortest path between Aris (whose id is 256176) and the author with id 16837 we obtain the following results:

```
In [43]: shortest_path = Modules.dijkstra(G, aris_id, 16837)
         shortest_path

Out[43]: (2.8776876177538426, [256176, 365027, 114966, 16837])
```

The shortest path we got is a tuple in which the first element (2.87) is the distance of the shortest path and the second element $[256176, 365027, 114966, 16837]$ is a list of all nodes involved to go from the Aris to the author given in input.

3 - Group Number In this last section we built up a code that, given a subset of nodes *I* returns for each node of the graph *G* its group number. We tested the code on a very small subset and we provided just one result in order to explain the output:

```
   20407 : (6.630122655122655, [20407, 18262, 16244, 17009, 271907, 272230,
364807, 21324, 287])
```

This means that the group number of the node 20407 is 6.63 that is the value of the shortest path from this node to the node 287 belonging to the subset *I* given in input.

```
In [45]: I = {aris_id, 19211, 287}
         group = Modules.group_number(G,I)
```