Arutyunov_source

April 8, 2022

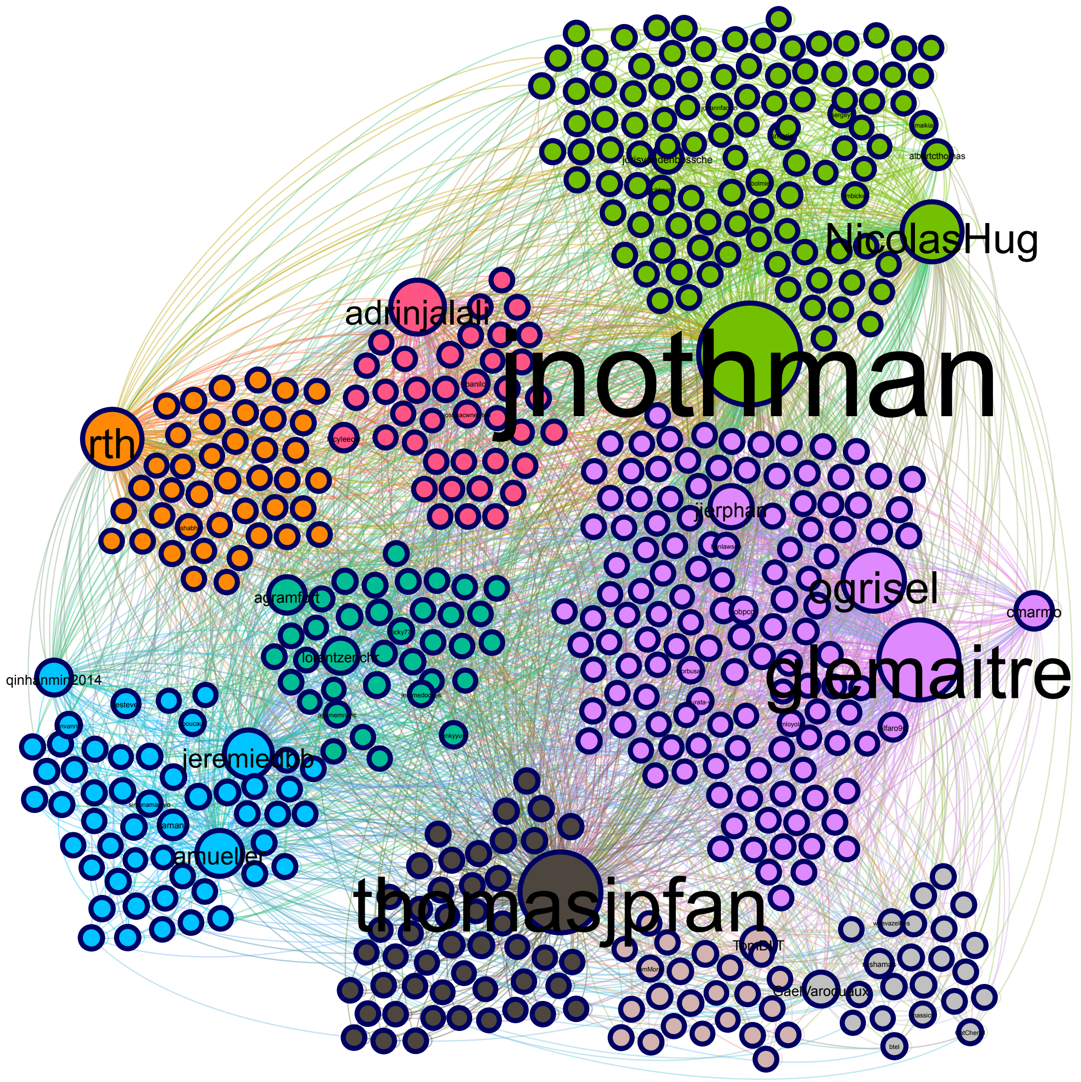
1 Community detection in large open-source projects

```
[3]: import networkx as nx
     import numpy as np
     import pandas as pd
     from IPython.display import SVG, display, Image
     import matplotlib.pyplot as plt
     import os

     from db.session import connect
     from db.models import Request, Review, Comment, User
     from network import get_graph
     from statistics import mean
```

```
[4]: %load_ext dotenv
     %dotenv
```

```
[5]: display(SVG(filename='./scikit_graph.svg'))
```



1.1 Prerequisites

1. Install requirements

[6]:
```
# !pip install -r requirements.txt
```

2. You also need a PostgreSQL database instance if you want to parse your own copy of the dataset
3. And a GitHub account with a GitHub token

1.2 Data

The dataset used in the project is composed of Pull Requests from scikit-learn repository and associated communication between reviewers and authors.

Data was collected from Pull Requests in scikit-learn repository using GitHub GraphQL API.

1.2.1 Database

All data is stored in PostgreSQL database using SQLAlchemy ORM. To initialize database run following command:

```
[7]: # !python -m db -c <your connection string>
```

1.2.2 Parsing

To parse pull requests:

```
[8]: # !python -m parse -c <your connection string> -t <your GitHub token> -o␣
      ↪scikit-learn -r scikit-learn
```

1.2.3 Data overview

```
[9]: Session = connect(os.getenv('PG_CONN'))
```

```
[10]: print('Number of pull requests: {0}'.format(Session.query(Request).count()))
```

```
Number of pull requests: 990
```

```
[11]: print('Number of reviews: {0}'.format(Session.query(Review).count()))
```

```
Number of reviews: 18905
```

```
[12]: print('Number of comments: {0}'.format(Session.query(Comment).count()))
```

```
Number of comments: 29077
```

```
[13]: print('Number of contributors: {0}'.format(Session.query(User).count()))
```

```
Number of contributors: 509
```

1.3 Network analysis

1.3.1 Summary

Since this is a communication network, the resulting graph is a directional homogeneous unweighted graph with self loops. All nodes are collaborators to scikit-learn repository. Edges represent communication between them.

```
[14]: g = get_graph()

      # leave only the gigantic component
```

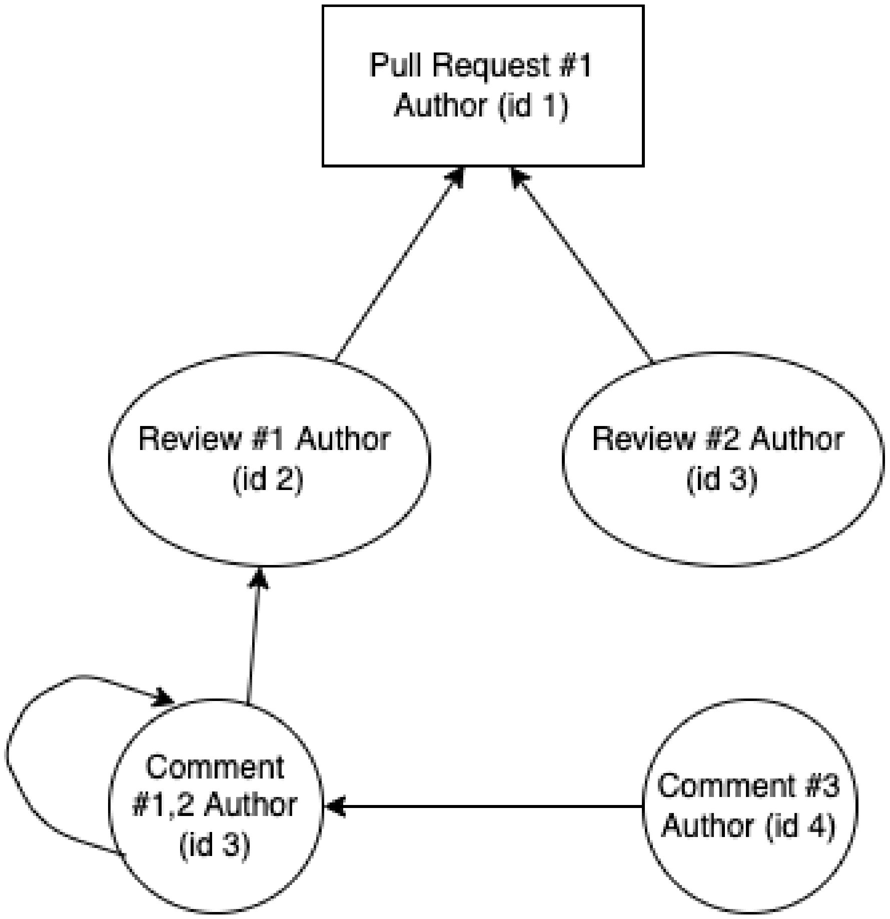
```
gcc = sorted([c for c in nx.connected_components(g.to_undirected())],␣
 ↪key=lambda x: len(x), reverse=True)[0]

g = g.subgraph(gcc)

# Convert to undirected graph for some calculations
g_u = g.to_undirected()
```

[15]:
```
nx.write_gexf(g, 'gephi/scikit_graph.gexf')
```

[16]:
```
display(Image(filename='communication_diagram.png'))
```



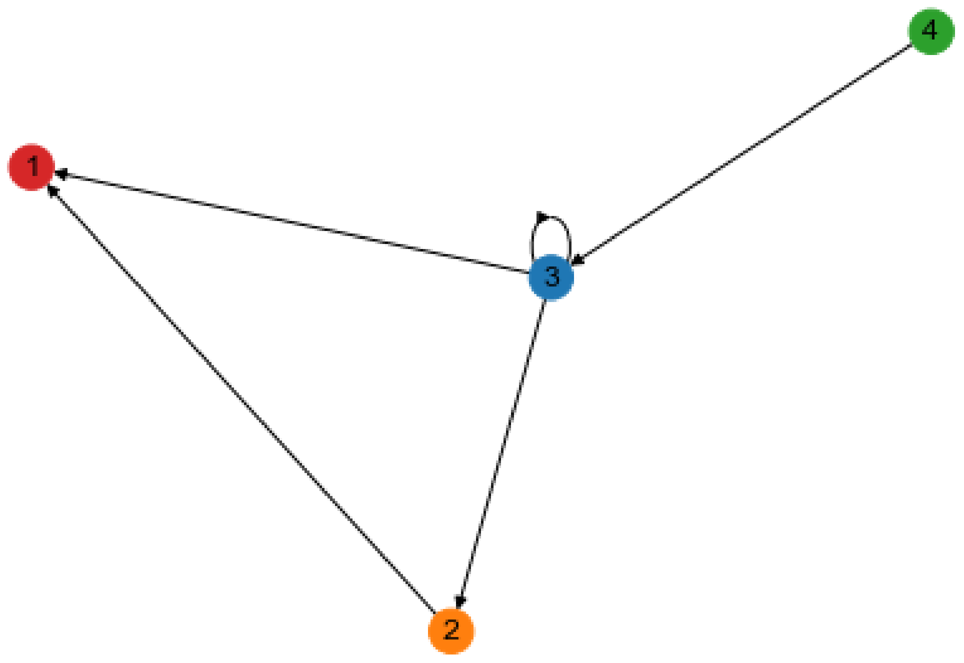
For example, the diagram above will result in following graph:

[17]:
```
dg = nx.DiGraph()

dg.add_nodes_from(range(1, 5))
dg.add_edges_from([(3, 1), (2, 1), (4, 3), (3, 3), (3, 2)])
```

```
node_color = ['tab:red', 'tab:orange', 'tab:blue', 'tab:green']

nx.draw(dg, with_labels=True, node_color=node_color)
```



[18]:
```
print('Number of nodes: ', g.number_of_nodes())
```

```
Number of nodes:  503
```

[19]:
```
print('Number of edges: ', g.number_of_edges())
```

```
Number of edges:  3182
```

[20]:
```
print('Diameter: ', nx.diameter(g_u))
```

```
Diameter:  6
```

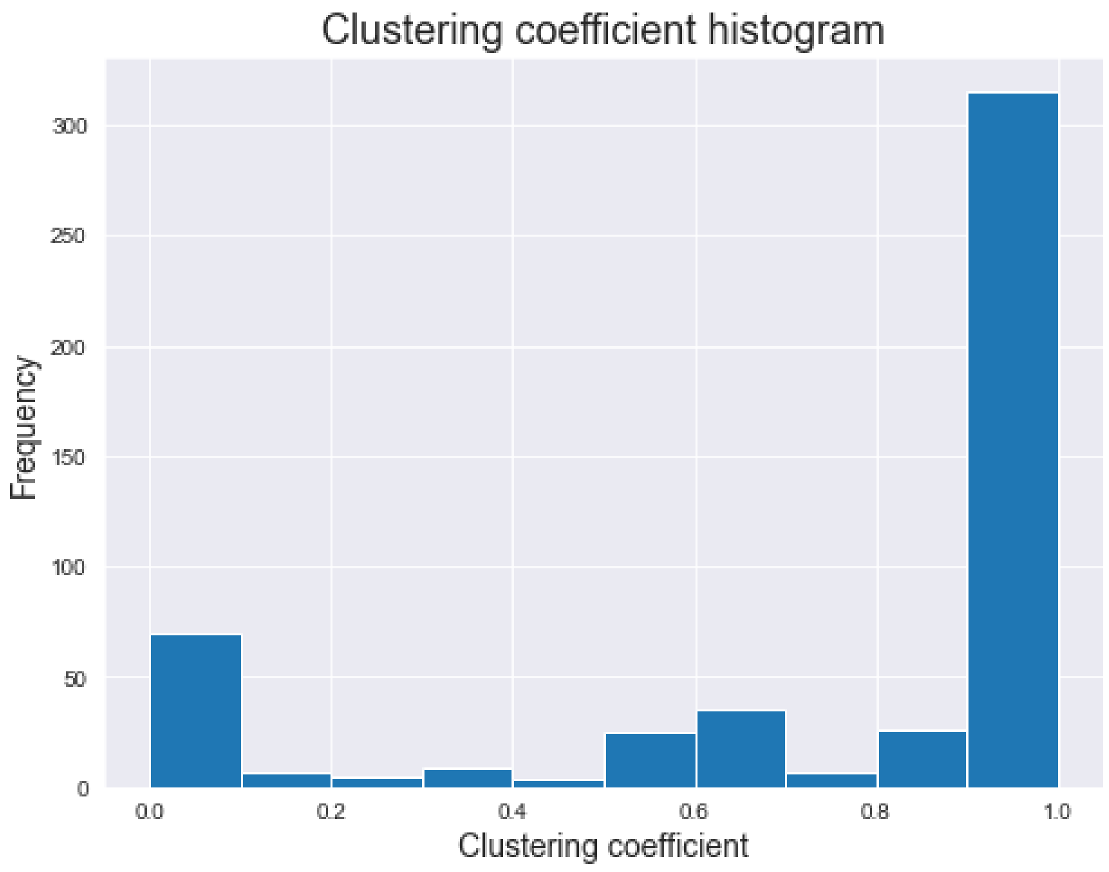
[21]:
```
print('Radius: ', nx.radius(g_u))
```

```
Radius:  3
```

Clustering coefficient Average clustering coefficient for nodes of low degrees is higher than of others. Nodes of lower degrees usually communicate with each others in tighter communities.

```
[22]: c = nx.clustering(g_u)

      plt.hist(c.values())
      plt.title('Clustering coefficient histogram', fontdict={'size': '18'})
      plt.ylabel('Frequency', fontdict={'size': '14'})
      plt.xlabel('Clustering coefficient', fontdict={'size': '14'})
      plt.gcf().set_size_inches(8, 6)
```



```
[23]: print('Average clustering coefficient: %.5f' % nx.average_clustering(g_u))
```

```
Average clustering coefficient: 0.76663
```

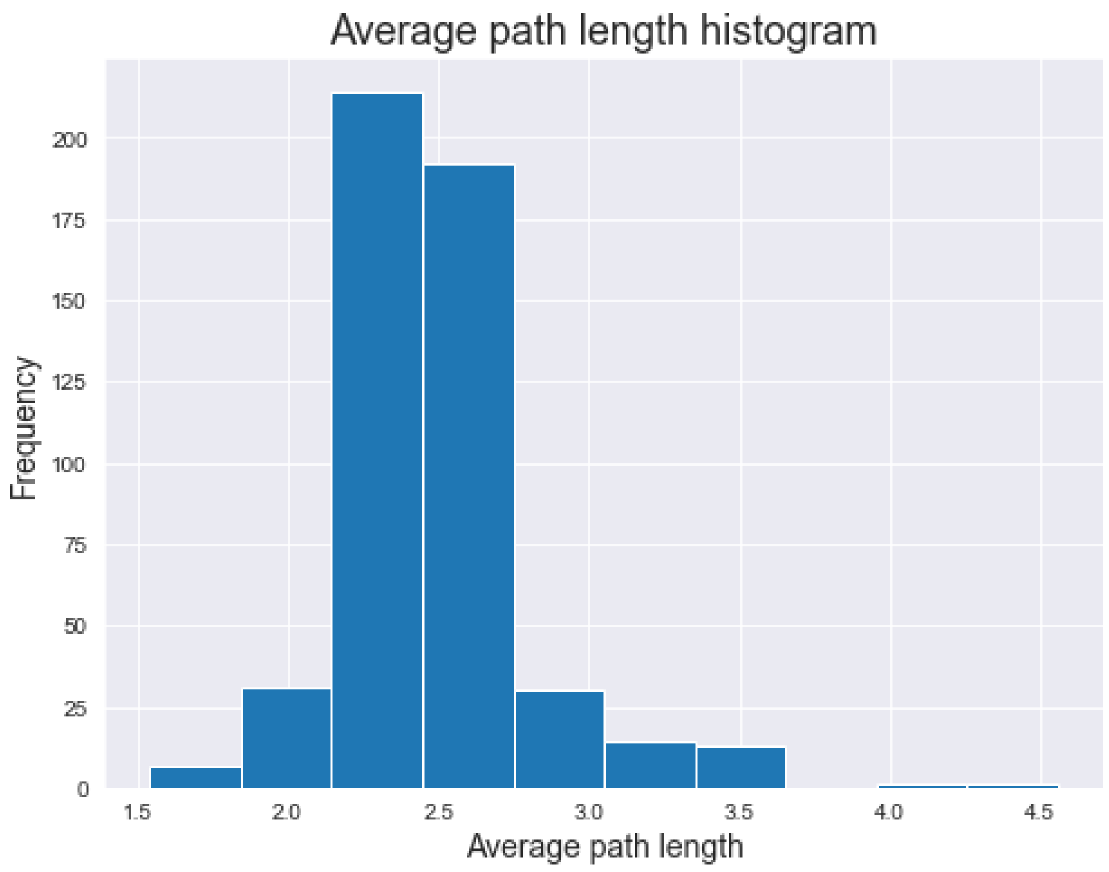
Average path length

```
[24]: spl = dict(nx.shortest_path_length(g_u))

      apl = {}

      for source, targets in spl.items():
          apl[source] = mean(targets.values())
```

```
plt.hist(apl.values())
plt.title('Average path length histogram', fontdict={'size': '18'})
plt.ylabel('Frequency', fontdict={'size': '14'})
plt.xlabel('Average path length', fontdict={'size': '14'})
plt.gcf().set_size_inches(8, 6)
```



[25]:
```
print('Average path length: %.5f' % nx.average_shortest_path_length(g_u))
```

```
Average path length: 2.48334
```

As we can see, average local shortest path us usually low, since there are bridges that connect contributors. Same result will be demonstrated with betweeness centrality.

Degree distribution Fit linear regression model to the node degree distribution.

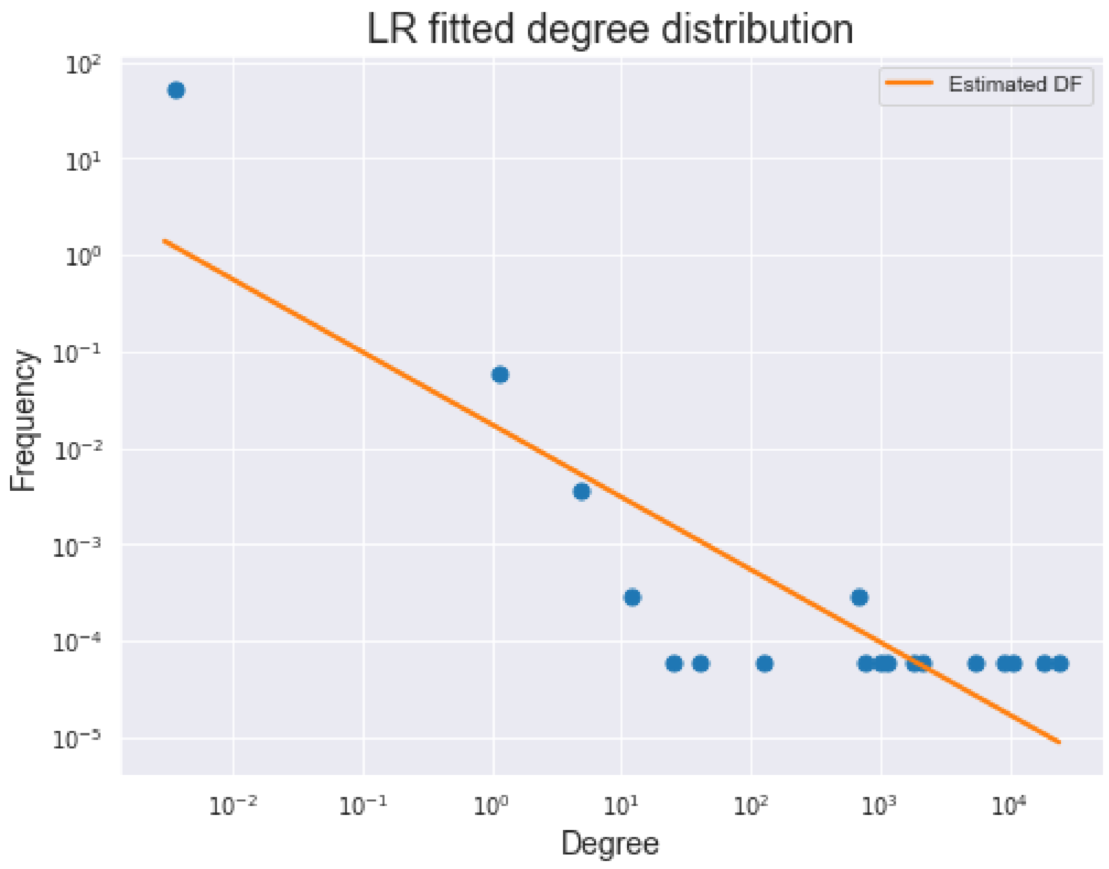
[26]:
```
import seaborn as sns
import matplotlib.ticker as mtick

#create scatterplot with regression line
hist, bin_edges = np.histogram(nx.degree_histogram(g), bins=500, density=True)
```

```
bin_centers = (bin_edges[1:] + bin_edges[:-1]) / 2
bin_centers = bin_centers[hist > 0]
hist = hist[hist > 0]

pl = sns.regplot(x=np.log(bin_centers), y=np.log(hist), ci=None, x_bins=500,␣
 ↪line_kws={'color': 'tab:orange', 'label': 'Estimated DF'})
plt.title('LR fitted degree distribution', fontdict={'size': '18'})
plt.legend()
plt.ylabel('Frequency', fontdict={'size': '14'})
plt.xlabel('Degree', fontdict={'size': '14'})
pl.xaxis.set_major_formatter(mtick.ScalarFormatter(useMathText=True))
pl.set_xticklabels(["$10^{%d}$" % int(l) for l in pl.get_xticks()])
pl.yaxis.set_major_formatter(mtick.ScalarFormatter(useMathText=True))
pl.set_yticklabels(["$10^{%d}$" % int(l) for l in pl.get_yticks()])
plt.gcf().set_size_inches(8, 6)
```

```
/var/folders/5f/tg63xq8x6jl7df71n9v3_g9c0000gn/T/ipykernel_2295/1629655113.py:16
: UserWarning: FixedFormatter should only be used together with FixedLocator
  pl.set_xticklabels(["$10^{%d}$" % int(l) for l in pl.get_xticks()])
/var/folders/5f/tg63xq8x6jl7df71n9v3_g9c0000gn/T/ipykernel_2295/1629655113.py:18
: UserWarning: FixedFormatter should only be used together with FixedLocator
  pl.set_yticklabels(["$10^{%d}$" % int(l) for l in pl.get_yticks()])
```

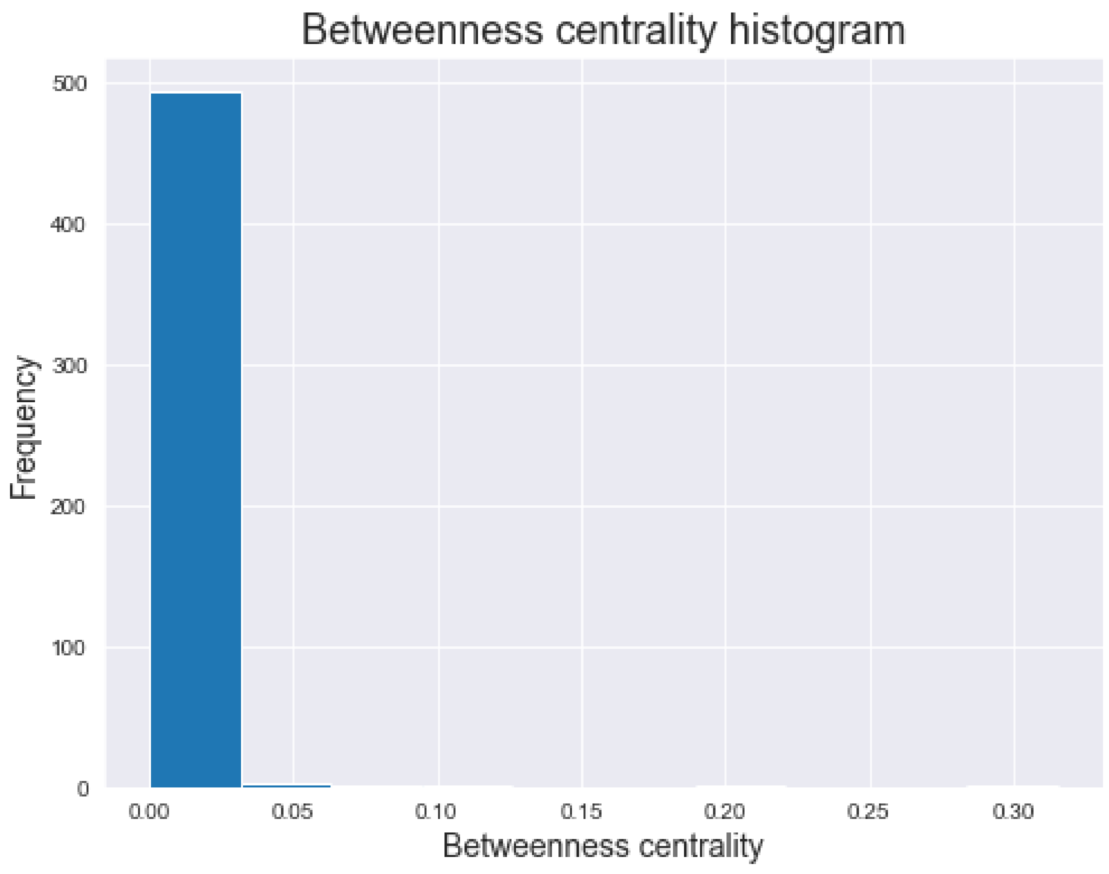


As we can see it is a power-law distribution

1.3.2 Structural Analysis

Betweenness centrality Betweenness centrality is pretty high for hubs. Those hubs a reviewers that connect developers in the community.

[27]:
```python
bc = nx.betweenness_centrality(g_u)
plt.hist(bc.values())
plt.title('Betweenness centrality histogram', fontdict={'size': '18'})
plt.ylabel('Frequency', fontdict={'size': '14'})
plt.xlabel('Betweenness centrality', fontdict={'size': '14'})
plt.gcf().set_size_inches(8, 6)
```



Random models Watts & Strogatz. This is a good model since it was designed for generating small world. And here we have a small world.

[28]:
```
ws = nx.watts_strogatz_graph(503, 12, 0.4)
nx.write_gexf(ws, 'gephi/watts-strogatz.gexf')
```

[29]:
```
print('Number of nodes: ', ws.number_of_nodes())
```

```
Number of nodes:  503
```

[30]:
```
print('Number of edges: ', ws.number_of_edges())
```

```
Number of edges:  3018
```

[31]:
```
print('Average clustering coefficient: %.5f' % nx.average_clustering(ws))
```

```
Average clustering coefficient: 0.16765
```

[32]:
```
print('Average path length: %.5f' % nx.average_shortest_path_length(ws))
```

```
Average path length: 2.92311
```

[33]:
```
print('Diameter: ', nx.diameter(ws))
```

```
Diameter:  4
```

[34]:
```
print('Radius: ', nx.radius(ws))
```

```
Radius:  4
```

Erdos & Renyi

[35]:
```
er = nx.erdos_renyi_graph(503, 0.025, directed=False)
nx.write_gexf(er, 'gephi/erdos-renyi.gexf')
```

[36]:
```
print('Number of nodes: ', er.number_of_nodes())
```

```
Number of nodes:  503
```

[37]:
```
print('Number of edges: ', er.number_of_edges())
```

```
Number of edges:  3099
```

[38]:
```
print('Average clustering coefficient: %.5f' % nx.average_clustering(er))
```

```
Average clustering coefficient: 0.02345
```

[39]:
```
print('Average path length: %.5f' % nx.average_shortest_path_length(er))
```

```
Average path length: 2.74180
```

[40]:
```
print('Diameter: ', nx.diameter(er))
```

```
Diameter:  4
```

[41]:
```
print('Radius: ', nx.radius(er))
```

```
Radius:  3
```

Barabasi & Albert. Has similar properties as other generation algorithms but is better at creating hubs.

```
[42]: ba = nx.extended_barabasi_albert_graph(503, 5, 0.05, 0.05)
      nx.write_gexf(ba, 'gephi/barabasi_albert.gexf')
```

```
[43]: print('Number of nodes: ', er.number_of_nodes())
```

```
Number of nodes:  503
```

```
[44]: print('Number of edges: ', er.number_of_edges())
```

```
Number of edges:  3099
```

```
[45]: print('Average clustering coefficient: %.5f' % nx.average_clustering(er))
```

```
Average clustering coefficient: 0.02345
```

```
[46]: print('Average path length: %.5f' % nx.average_shortest_path_length(er))
```

```
Average path length: 2.74180
```

```
[47]: print('Diameter: ', nx.diameter(er))
```

```
Diameter:  4
```

```
[48]: print('Radius: ', nx.radius(er))
```

```
Radius:  3
```

1.3.3 Community detection

The best model for community detection that is suitable for our case is the Louvain method. We have a tight graph. Girvan-Newman algorithm doesn't work well since we have a lot of nodes with low betweenness centrality. Klique algorithm doesn't detect communities.

```
[49]: from community.community_louvain import best_partition

      communities = best_partition(g_u, randomize=True)
```

```
[50]: pd.DataFrame.from_dict(communities, orient='index', columns=['community']).
       ↪value_counts(['community'])
```

```
[50]: community
      1            118
      5             71
      4             45
      10            39
      0             37
      3             36
```

```
9             36
11            27
2             26
7             23
8             23
6             14
12             8
dtype: int64
```

[51]:
```
cliques = [k for k in nx.find_cliques(g_u)]

len([c for c in nx.community.k_clique_communities(g_u, 5, cliques)])
```

[51]:
```
1
```

[51]:
```

```