

Assignment 4

Fall 2014

CS595 Web Science

Dr. Michael Nelson

Mathew Chaney

October 9, 2014

Contents

1	Question 1	3
1.1	Question	3
1.2	Answer	3
2	Question 2	5
2.1	Question	5
2.2	Answer	5
3	Question 3	7
3.1	Question	7
3.2	Answer	7
4	References	14

List of Figures

1	Graph Overview	7
2	Strongly Connected Core	8
3	Core Close-up	8
4	HITS Authorities Scores	9
5	HITS Hubs Scores	9
6	Page Rank Scores	10
7	Average Degree Distributions	10
8	In-Degree Distributions	11
9	Out-Degree Distributions	11
10	Betweenness Centrality Distribution	12
11	Closeness Centrality Distribution	12
12	Eccentricity Distribution	13
13	Connected Components	13

Listings

1	get_links.py	4
2	dot.py	6

1 Question 1

1.1 Question

From your list of 1000 links, choose 100 and extract all of the links from those 100 pages to other pages. We're looking for user navigable links, that is in the form of:

```
<A href="foo">bar</a>
```

We're not looking for embedded images, scripts, `<link>` elements, etc. You'll probably want to use BeautifulSoup for this.

For each URI, create a text file of all of the outbound links from that page to other URIs (use any syntax that is easy for you). For example:

```
site:
http://www.cs.odu.edu/~mln/
links:
http://www.cs.odu.edu/
http://www.odu.edu/
http://www.cs.odu.edu/~mln/research/
http://www.cs.odu.edu/~mln/pubs/
http://ws-dl.blogspot.com/
http://ws-dl.blogspot.com/2013/09/2013-09-09-ms-thesis-http-mailbox.html
etc.
```

Upload these 100 files to github (they don't have to be in your report).

1.2 Answer

To obtain the outbound links of 100 URIs for use in building a network graph, a subset of 1000 URIs from a previous assignment was selected semi-randomly. The original list was sorted lexicographically and then a contiguous section was taken from it for processing. This was done to increase the likelihood that a portion of the resulting list came from the same domain since it is probable that of 1000 randomly selected URIs from Twitter, some pointed to resources on the same domain.

The HTML of each URI was previously downloaded from another assignment and organized by creating a mapping from the URI to the filename of its contents, called `uri_map`. This data structure is a pickled^[1] python dictionary object that maps from a URI to a filename. This file contains the URI as the first line and the HTML contents of that URI when it was dereferenced. This mapping was created to save time when looking for the contents of a particular URI because only some of the URIs were successfully downloaded at the time the files were created.

After the list was chosen, each of the files was parsed using the BeautifulSoup^[2] module to extract all of the out-links. The python code that was used to parse the content is displayed in Listing 1.

```

1  #! /usr/bin/python
2
3  import os
4  import random
5  import pickle
6  from bs4 import BeautifulSoup
7
8  LINKS_DIR = 'links' + os.sep
9  HTML_DIR = 'html' + os.sep
10 uri_map = pickle.load(open('uri_map', 'rb'))
11
12 def get_links(filename):
13     '''Parses HTML contents and returns tuple of (filename, uri, list of links)'''
14     with open(HTML_DIR + filename) as infile:
15         uri = infile.readline().rstrip('\n')
16         soup = BeautifulSoup(infile.read())
17         links = [link['href'].encode('utf-8') for link in soup.find_all('a') if link.
18                 has_attr('href')]
19         return filename, uri, links
20
21 def write_links(filename, uri, links):
22     '''Writes URI and outlinks to file'''
23     print('Writing {}'.format(uri))
24     with open(LINKS_DIR + filename, 'w') as outfile:
25         outfile.write('{}\n'.format(uri))
26         for link in links:
27             outfile.write('{}\n'.format(link))
28
29 if __name__ == '__main__':
30     # random selection of 100 URIs in series from sorted list
31     start = random.randint(0, 899)
32     keys = uri_map.keys()
33     keys.sort()
34     for i in xrange(start, start + 100):
35         write_links(*get_links(uri_map[keys[i]]))

```

Listing 1: get_links.py

2 Question 2

2.1 Question

Using these 100 files, create a single GraphViz "dot" file of the resulting graph. Learn about dot at:

Examples:

<http://www.graphviz.org/content/unix>

<http://www.graphviz.org/Gallery/directed/unix.gv.txt>

Manual:

<http://www.graphviz.org/Documentation/dotguide.pdf>

Reference:

<http://www.graphviz.org/content/dot-language>

<http://www.graphviz.org/Documentation.php>

Note: you'll have to put explicit labels on the graph, see:

<https://gephi.org/users/supported-graph-formats/graphviz-dot-format/>

(note: actually, I'll allow any of the formats listed here:

<https://gephi.org/users/supported-graph-formats/>

but "dot" is probably the simplest.)

2.2 Answer

Using python to parse each of the 100 files from Question 1, all of the links that each URI contained were extracted from the source html, converted to the GraphViz[3] dot format and stored in a file named `links.gv`. To save space in the graph the `urlparse`[4] python module was used to shorten the URIs used as labels. The code used is in Listing 2.

```

1  #!/usr/bin/python
2
3  import os
4  import random
5  from urlparse import urlparse
6
7  LINKS_DIR = 'links' + os.sep
8
9  def extract(uri):
10     '''Strip the path from the URI and return the domain name'''
11     parsed = urlparse(uri.strip())
12     if parsed.netloc:
13         return parsed.netloc + parsed.path[:20]
14     else:
15         return ''
16
17  def write_dot(uri, links, outfile):
18     '''Write the uri and all of its links (with labels) to the file "outfile" in dot format
19     '''
20     for link, label in links.iteritems():
21         outfile.write('\t"{}" -> "{}";\n'.format(uri, link))
22         outfile.write('\t"{}" [label="{}"]\n'.format(uri, extract(uri)))
23         outfile.write('\t"{}" [label="{}"]\n'.format(link, label))
24
25  if __name__ == '__main__':
26     with open('links.gv', 'w') as outfile:
27         outfile.write('digraph unix {\n')
28         for filename in os.listdir('links'):
29             with open(LINKS_DIR + filename, 'r') as infile:
30                 uri = infile.readline().rstrip('\n')
31                 links = {link.rstrip('\n'): extract(link) for link in infile if extract(link)}
32                 write_dot(uri, links, outfile)
33         outfile.write('}')

```

Listing 2: dot.py

3 Question 3

3.1 Question

Download and install Gephi:

<https://gephi.org/>

Load the dot file created in #2 and use Gephi to:

- visualize the graph (you'll have to turn on labels)
- calculate HITS and PageRank
- avg degree
- network diameter
- connected components

Put the resulting graphs in your report.

You might need to choose the 100 sites with an eye toward creating a graph with at least one component that is nicely connected. You can probably do this by selecting some portion of your links (e.g., 25, 50) from the same site.

3.2 Answer

To give an idea of the overall layout of the graph refer to Figure 1. All graphs were produced using Gephi^[5].

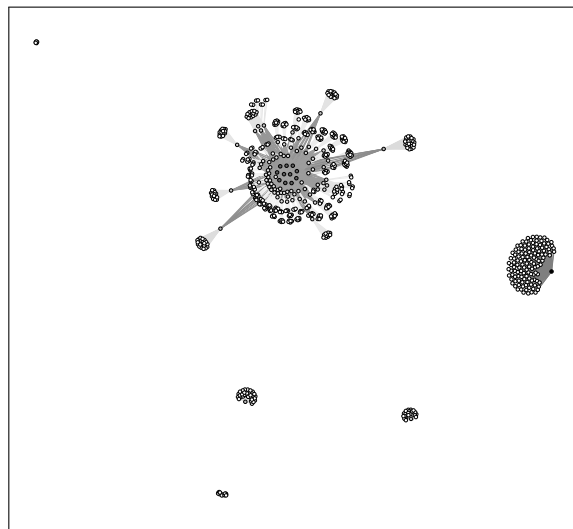


Figure 1: Graph Overview

The graph has one strongly connected core component with a few chunked islands of links not connected to the core. The strongly connected component is shown closer in Figure 2.

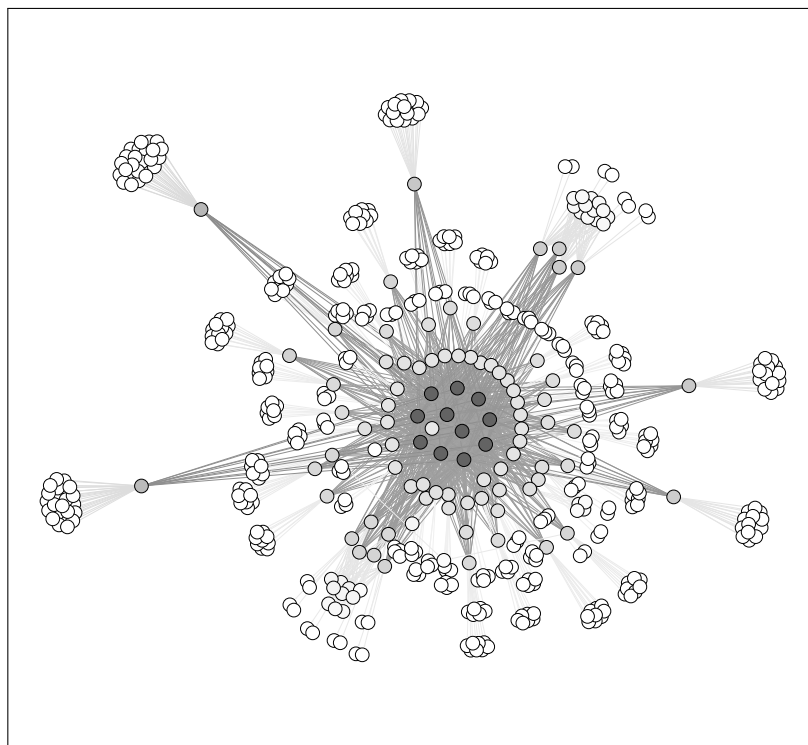


Figure 2: Strongly Connected Core

The nodes in the center are mostly from google domains – YouTube.com, various google.com sub-domains, like plus.google.com, accounts.google.com, etc. This core is illustrated with node labels in Figure 3. The core is well connected, presumably because they are domains controlled by a single company that is involved in connecting their users with a multitude of the companies other resources, which leads to having many connections among their own domains.

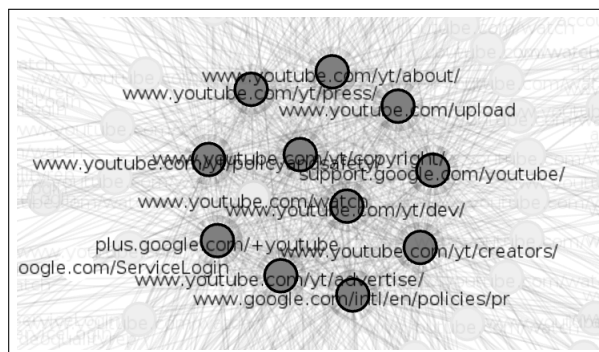


Figure 3: Core Close-up

The Hyperlink-Induced Topic Search (HITS) is a graph analysis algorithm that ranks nodes in a connected graph based on two categories: authorities and hubs. A high authority score means that the page is pointed to by many hubs and a high hub score means the page points to many authorities. The algorithm was run on the sample graph created and the results are displayed in Figures 4 and 5.

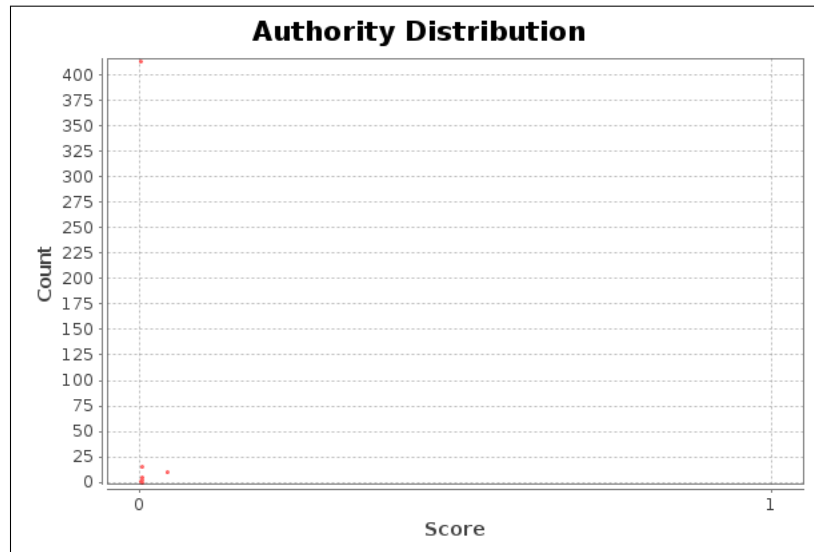


Figure 4: HITS Authorities Scores

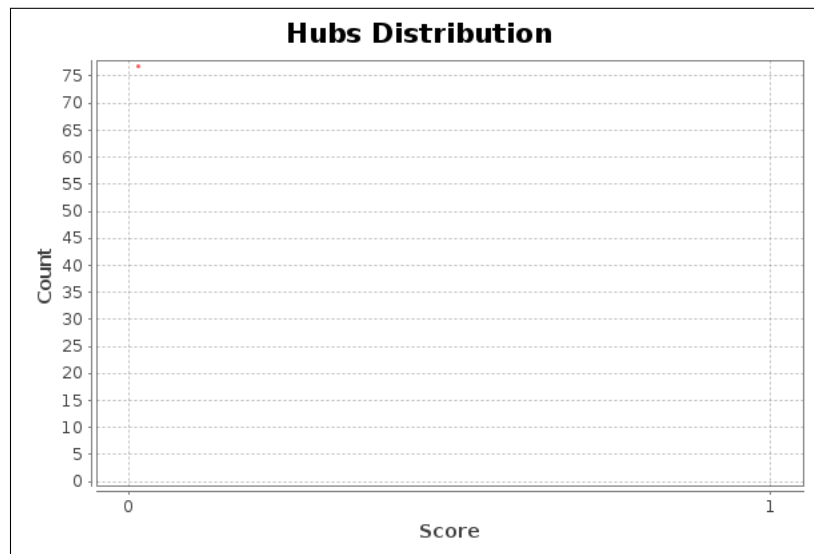


Figure 5: HITS Hubs Scores

Page Rank is an algorithm that is used to measure the importance of a node in a connected graph, used to rank web pages in an order search result. It was developed by Google for use in its search engine of the same name. It is basically a valuation measure of each page found by adding up the degree for each page and normalizing it against the rest of the pages in the set. The algorithm was run on the sample graph created and the results are displayed in Figure 6

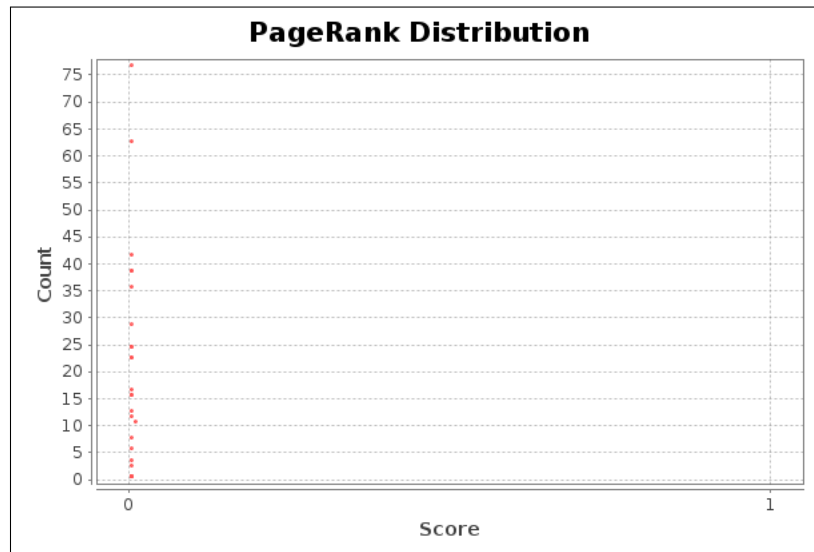


Figure 6: Page Rank Scores

The average degree of the graph, or links into or out of each node, was also calculated. The results are found in Figures 7, 8 and 9.

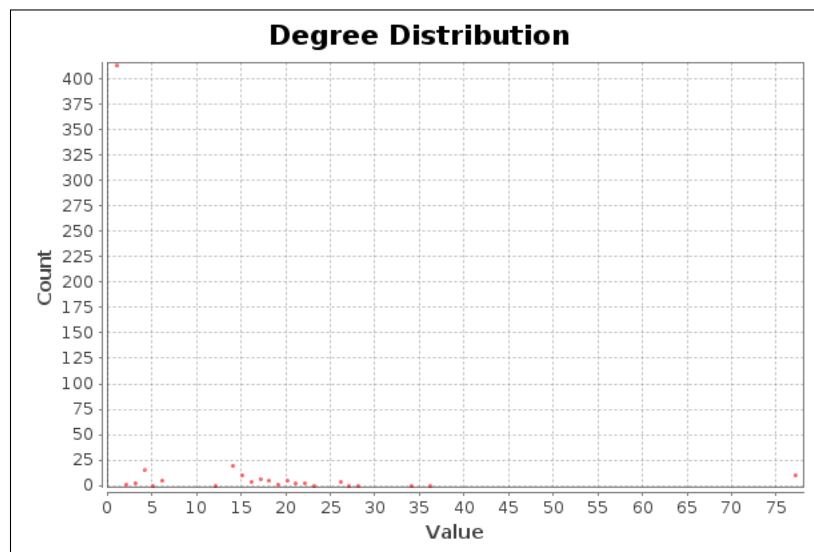


Figure 7: Average Degree Distributions

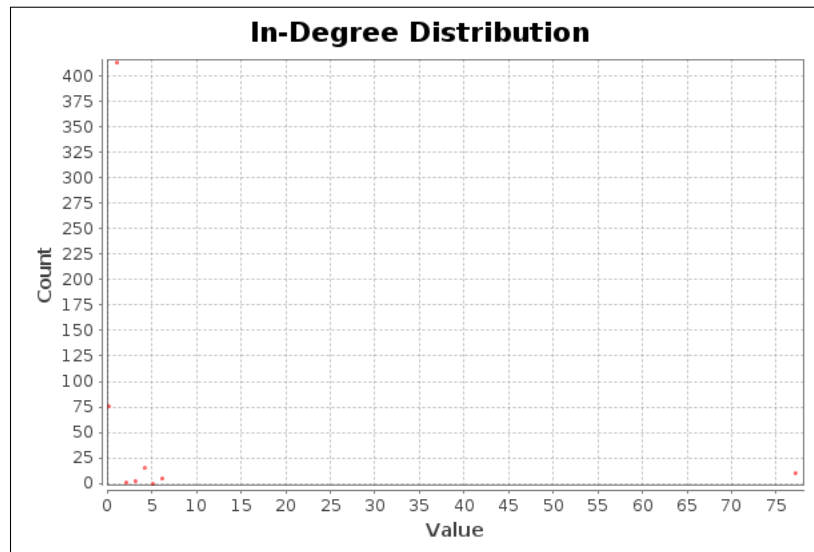


Figure 8: In-Degree Distributions

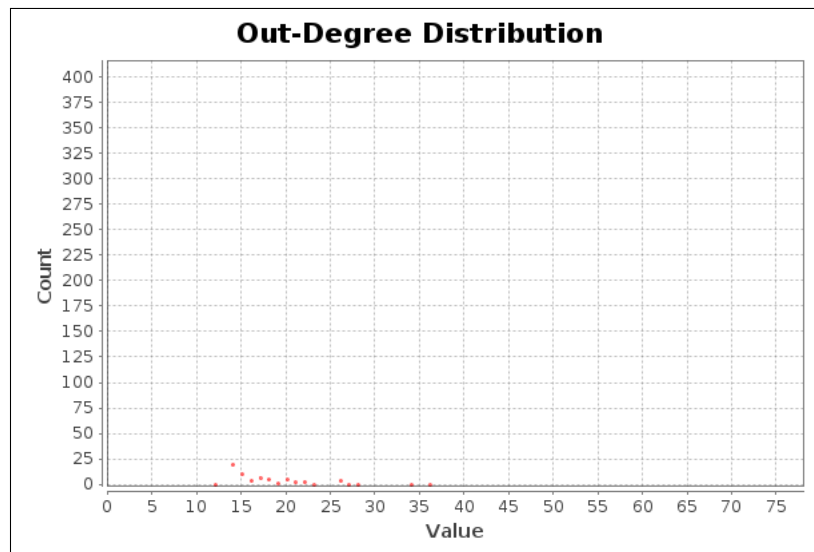


Figure 9: Out-Degree Distributions

The diameter of a network is defined as the maximum of the all the shortest paths between any two nodes in the network. The network diameter was calculated for the example graph and the results are found in Figures 10, 11 and 12.

The “betweenness centrality” of a graph measures how often a node appears on the shortest paths between nodes in the network.

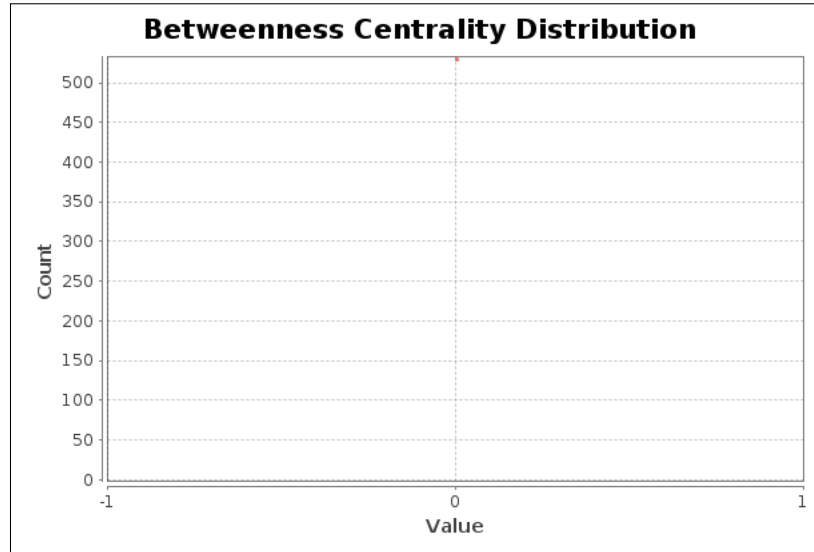


Figure 10: Betweenness Centrality Distribution

The “closeness centrality” is the average distance from a given starting node to all other nodes in the network.

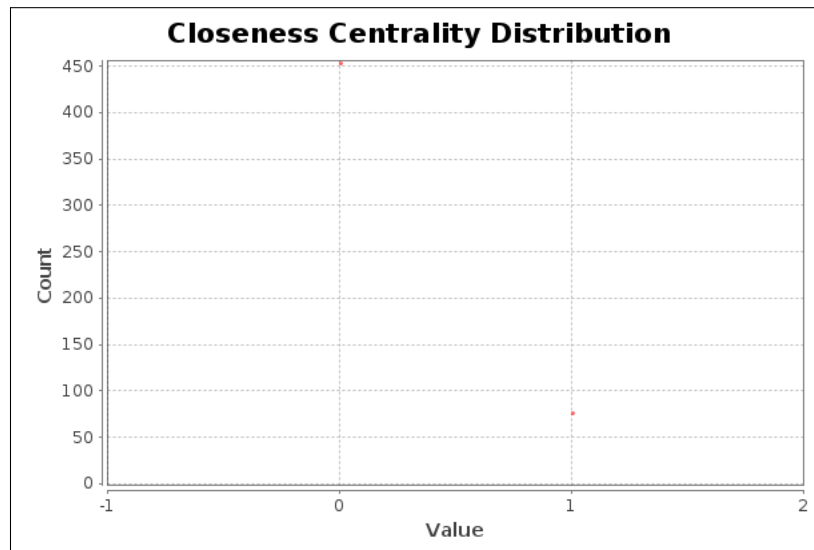


Figure 11: Closeness Centrality Distribution

The “eccentricity” is the distance from a given starting node to the farthest node from it in the network.

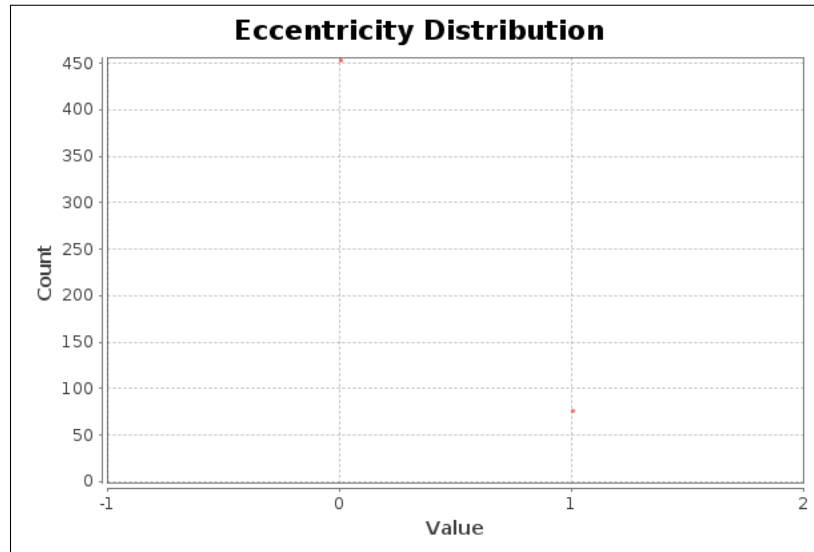


Figure 12: Eccentricity Distribution

The number of strongly connected components were found to be six in total and the number of weakly connected components was 738. The size distribution is shown in the graph in Figure 13

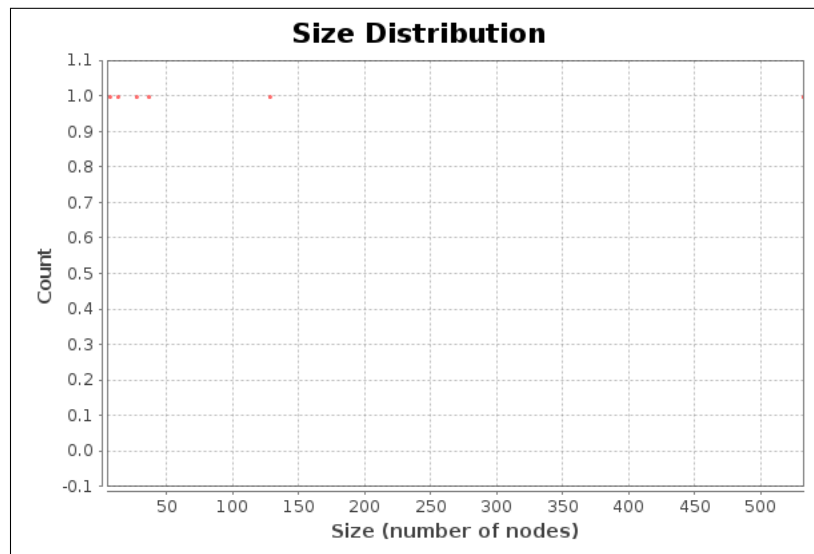


Figure 13: Connected Components

4 References

- [1] The Python Software Foundation. Python pickle module. <https://docs.python.org/2/library/pickle.html>, 2014.
- [2] Leonard Richardson. Beautiful Soup. <http://www.crummy.com/software/BeautifulSoup/>, 2014.
- [3] AT&T Research Labs. GraphViz. <http://www.graphviz.org/>, 2014.
- [4] The Python Software Foundation. Python urlparse module. <https://docs.python.org/2/library/urlparse.html>, 2014.
- [5] The Gephi Team. Gephi. <https://gephi.github.io/>, 13 Sep 2014.