Assignment Four

Sybil Melton

October 7, 2014

CONTENTS

LIST OF FIGURES

LISTINGS

1 Task One

From your list of 1000 links, choose 100 and extract all of the links from those 100 pages to other pages.

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).

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

1.1 Solution

I chose to pick the 100 sites randomly. The URI to hash mappings were extracted from "mapping.txt" into lists and I used the random.randint python module to choose 100 numbers between 0 and 1000, which corresponded to the position of the URIs in the list. [2] The already downloaded html was put into BeautifulSoup, which was able to find all of the <a href> tags and extract the links. [1] Each new file with the extracted links was written to a "links" directory. [5] The 100 files are included with this report. Listing 1 is the python program used to open the downloaded html, extract the links with BeautifulSoup, and write them to file.

1.2 Python Code

```
import random
from bs4 import BeautifulSoup

#read the file from the downloads directory
#after reading the file, feed the data into beautiful soup to get the links
#write them to the output file, in the links directory
def getLinks(u,f):
  inFile = 'downloads/'+f
  outFile = 'links/'+f
  with open(inFile, 'r') as file:  #open downloaded file
    data = file.read()
  soup = BeautifulSoup(data)
  if len(soup.find_all('a')) > 0:
    with open(outFile, 'w', 0) as O:  #write the header
      O.write('site:\nhttp://')
      O.write(u)
      O.write('\nlinks:\n')
      for link in soup.find_all('a'):  #http://www.crummy.com/software/BeautifulSoup/bs4/doc/
        l = str(link.get('href'))
        if l.startswith('http'):
          O.write(l)                    #write the extracted links
          O.write('\n')
  else:
    print 'no links found in ' + f

if __name__ == '__main__':
  uri = list()
  out = list()
  ran = list()
  with open('mapping.txt' as mFile:  #open uri to hash mappings
    for line in mFile:               # put them into lists for enumeration
      mapping = line.split('\t')
      uri.append(mapping[0])
      out.append(mapping[1].rstrip())
    i = 0
  while i < 100:  # create a list of random numbers that will be used to chose the files
    ran.append(random.randint(0, len(uri)))
  for r in ran:   #get the links for each
    getLinks(uri[r],out[r])
```

Listing 1: Link Extractor

2 Task Two

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

2.1 Solution

I created a list of all the files in my “links” directory, which was created to hold the files from Question 1. Each file was opened, the original URI was extracted and labelled with the netloc from the urlparse Python module. [4] Then the edges were placed into adjacency lists. [3] The connected URIs were labelled with blanks, because the resulting graph has over 6000 nodes. The graph is found in section 3.2. Originally I tried to use the URIs and then the domain names to label all the nodes, but both proved to be too long and cluttered for the graph. Listing 2 is the python program used to open the links files, parse the lines, and write the “dot” file, which is included with this report.

2.2 Python Code

```
import random
from bs4 import BeautifulSoup

#read the file from the downloads directory
#after reading the file, feed the data into beautiful soup to get the links
#write them to the output file, in the links directory
def getLinks(u,f):
  inFile = 'downloads/'+f
  outFile = 'links/'+f
  with open(inFile, 'r') as file:  #open downloaded file
    data = file.read()
  soup = BeautifulSoup(data)
  if len(soup.find_all('a')) > 0:
    with open(outFile, 'w', 0) as O:  #write the header
      O.write('site:\nhttp://')
      O.write(u)
      O.write('\nlinks:\n')
      for link in soup.find_all('a'):  #http://www.crummy.com/software/BeautifulSoup/bs4/doc/
        l = str(link.get('href'))
        if l.startswith('http'):
          O.write(l)                    #write the extracted links
          O.write('\n')
  else:
    print 'no links found in ' + f

if __name__ == '__main__':
  uri = list()
  out = list()
  ran = list()
  with open('mapping.txt' as mFile:  #open uri to hash mappings
    for line in mFile:               # put them into lists for enumeration
      mapping = line.split('\t')
      uri.append(mapping[0])
      out.append(mapping[1].rstrip())
    i = 0
  while i < 100:  # create a list of random numbers that will be used to chose the files
    ran.append(random.randint(0, len(uri)))
  for r in ran:   #get the links for each
    getLinks(uri[r],out[r])
```

Listing 2: Build Dot File

3 Task Three

Download and install Gephi.

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.

3.1 Solution

I downloaded and used the layout scheme OpenOrd, which is advertised as scalable to up to one million nodes. The network graph was colored by Strongly Connected ID, in order to show the node clusters and edges that connect them. There seemed to be a lot of blog sites, so it was not surprising to see so many sites connected to wordpress.com or blogger.com. I looked up one connection, disqus.com, and found it to be a website building company. So it makes sense that multilple sites would connect to it.
In Gephi, average degree was calculated to be 1.002, network diameter was 1, and connected components was 70. The resulting graphs are in section 3.2.

3.2 GRAPHS

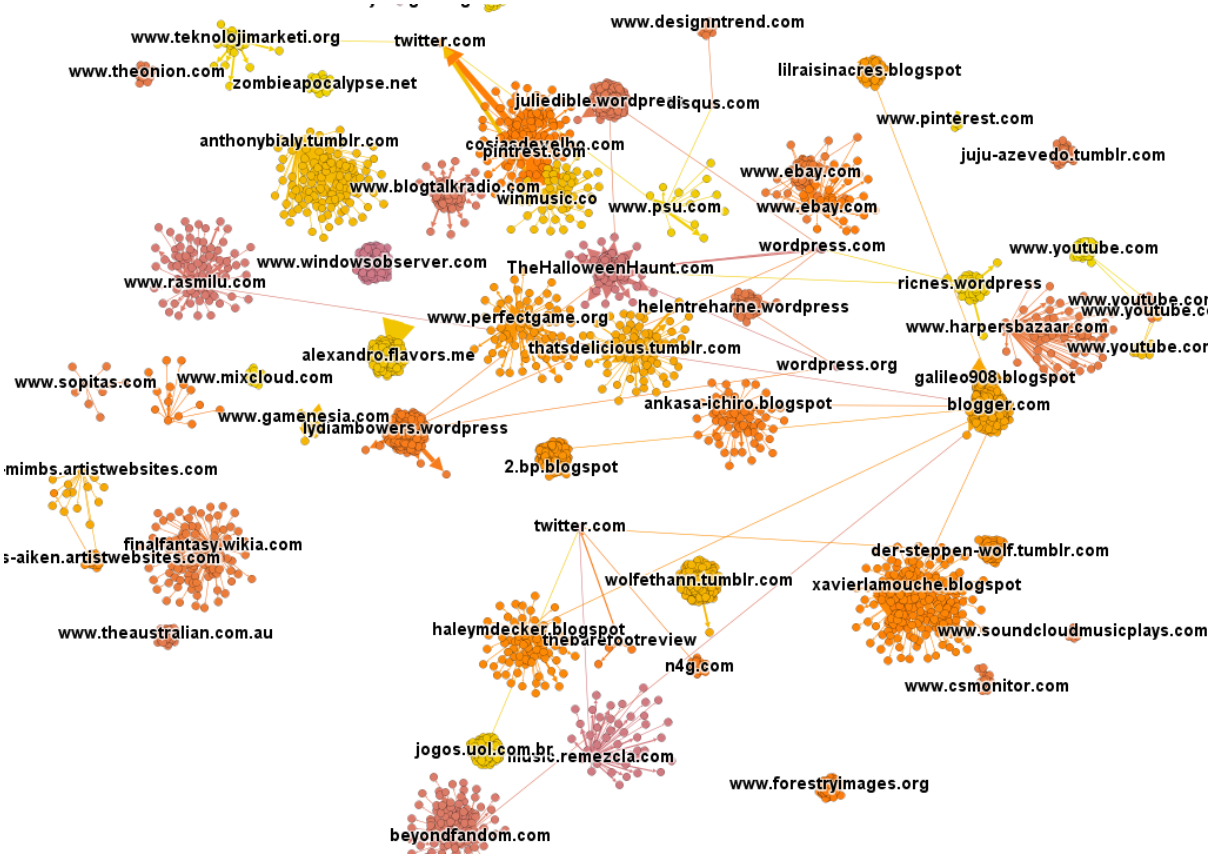

Figure 3.1: Network Graph

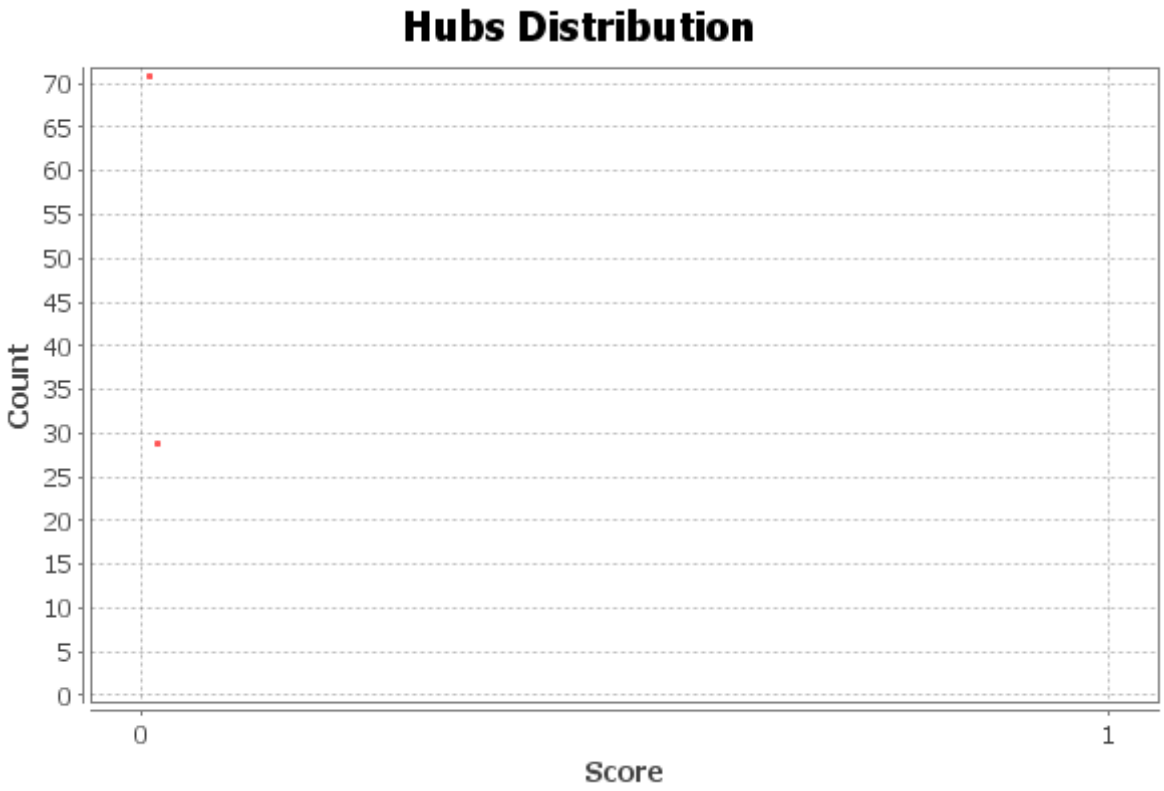

Figure 3.2: HITS HUBS

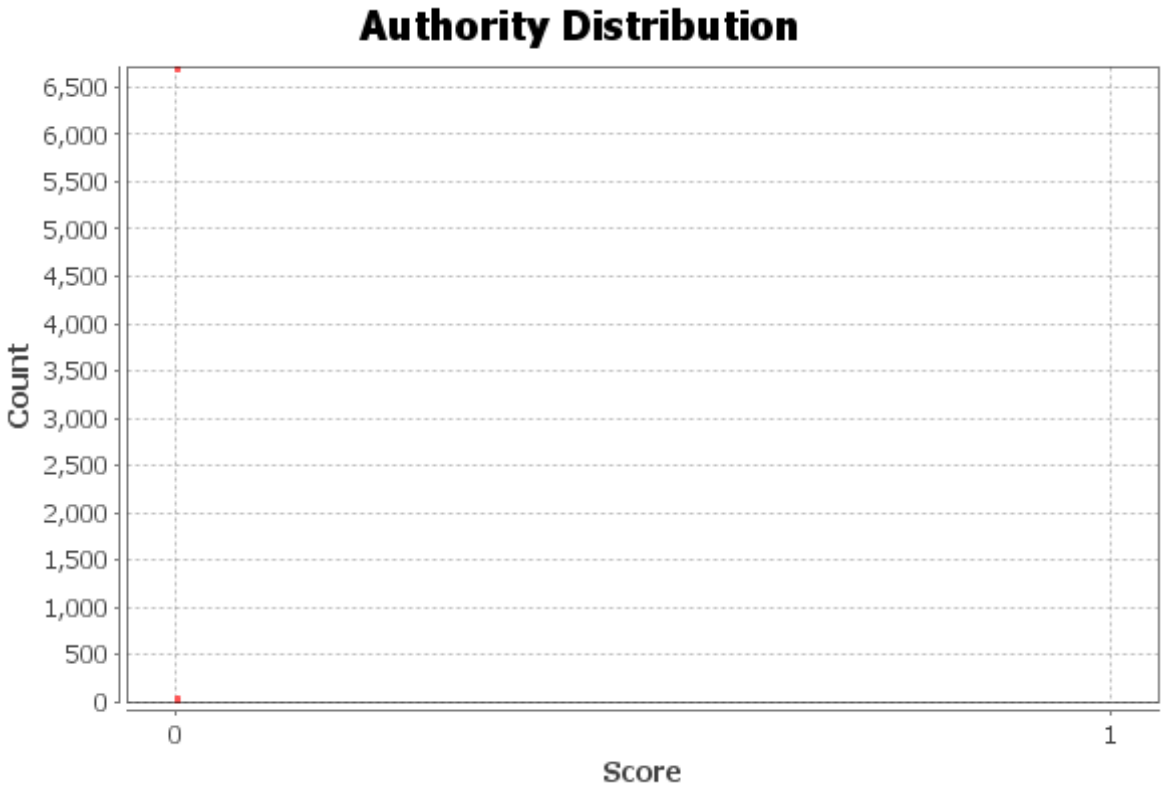

Figure 3.3: Authorities

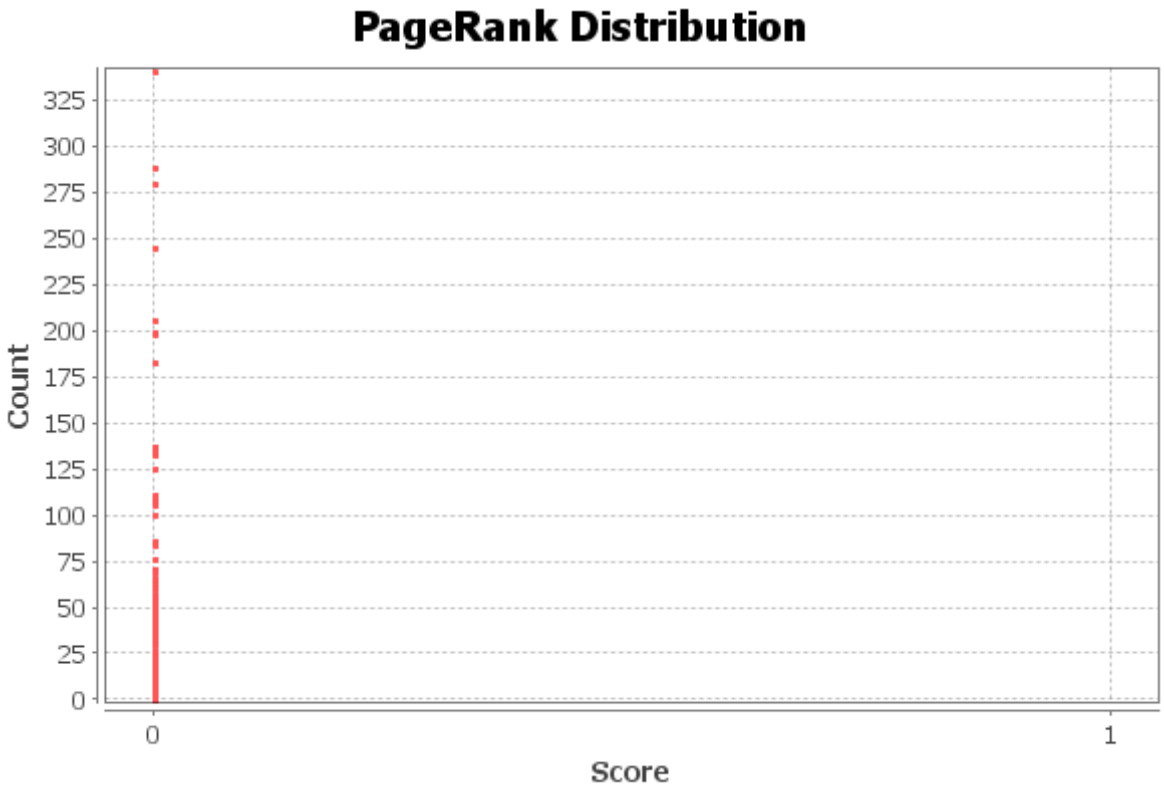

Figure 3.4: Page Ranks

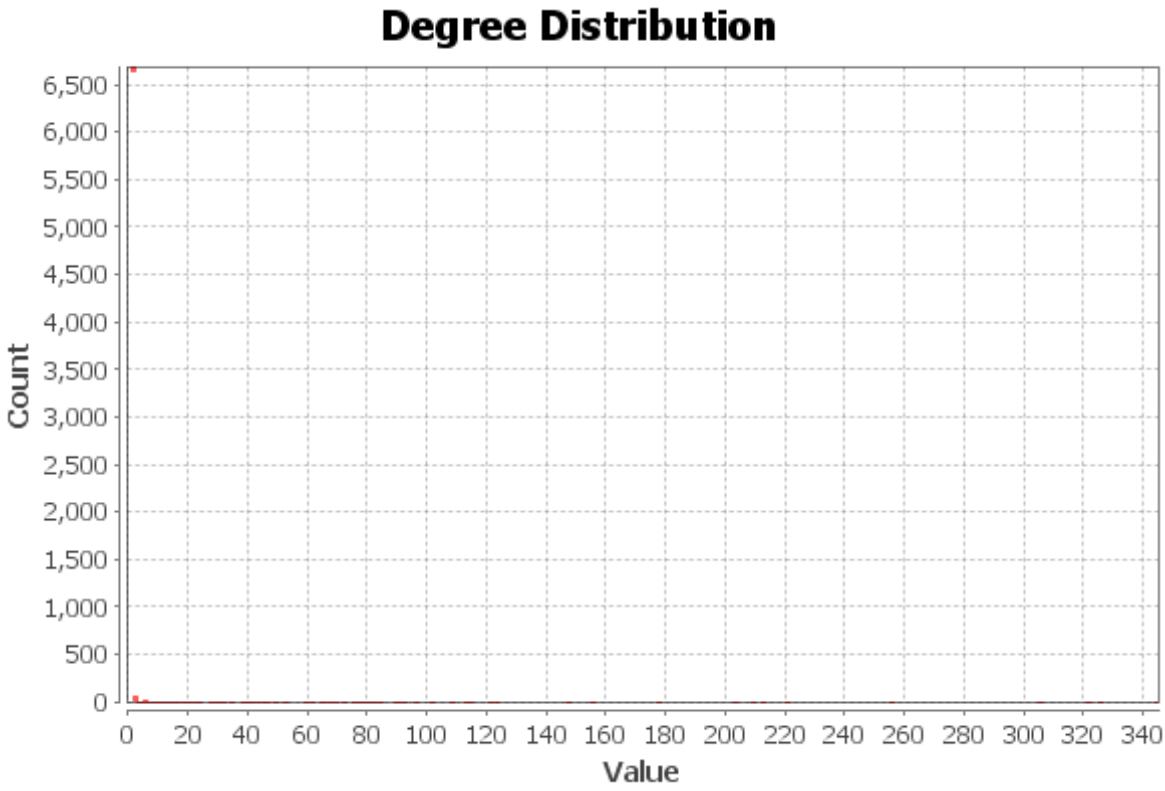

Figure 3.5: Degree Distribution

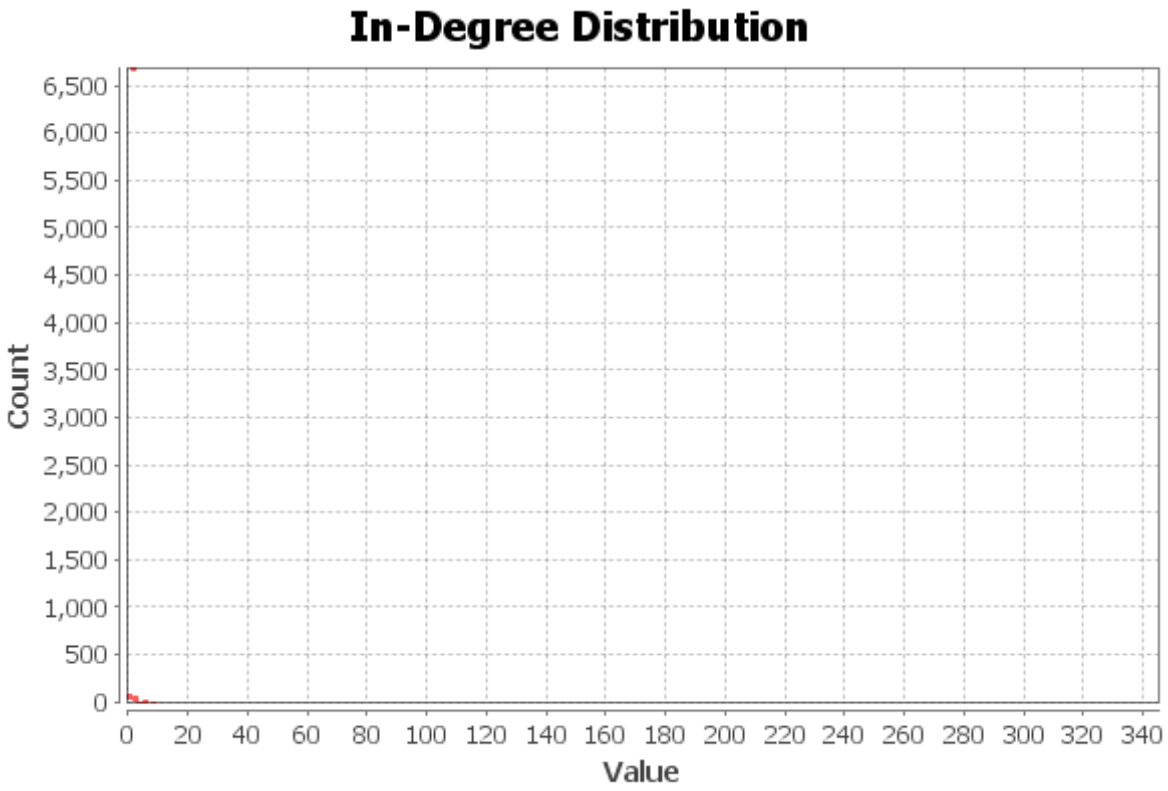

Figure 3.6: In Degree Distribution

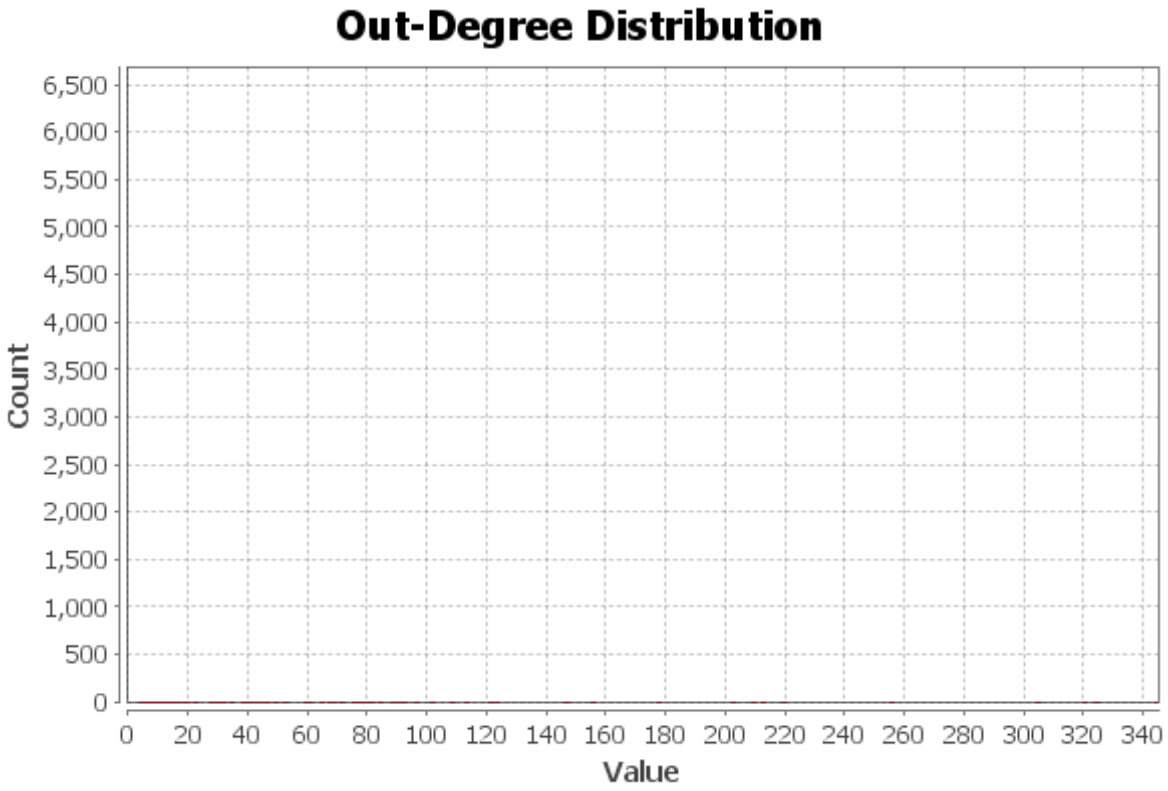

Figure 3.7: Out Degree Distribution

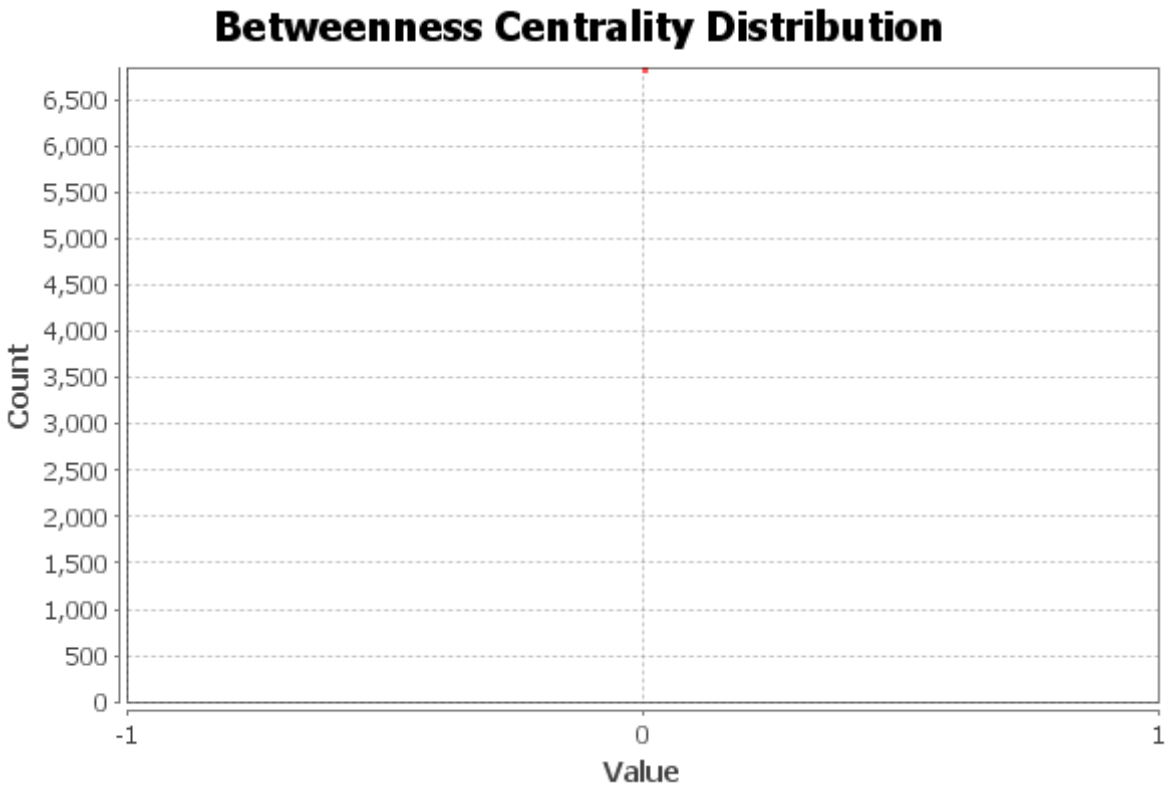

Figure 3.8: Betweenness Centrality Distribution

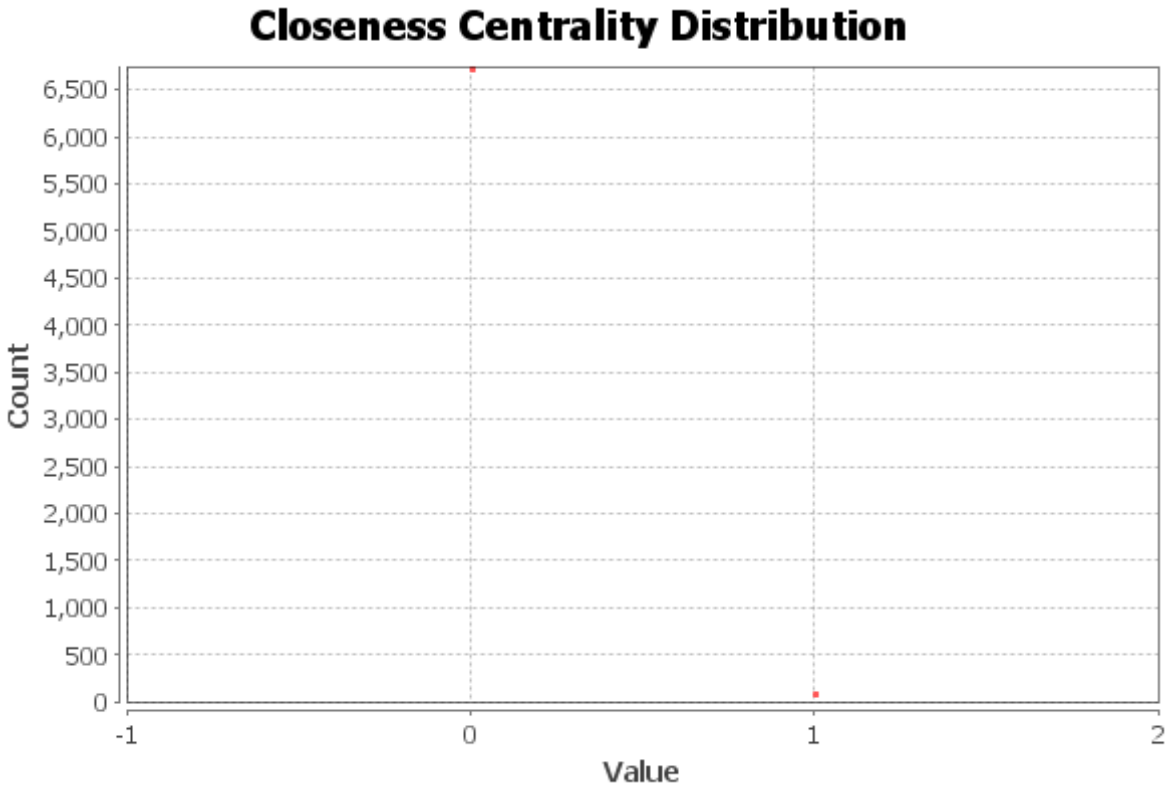

Figure 3.9: Closeness Centrality Distribution

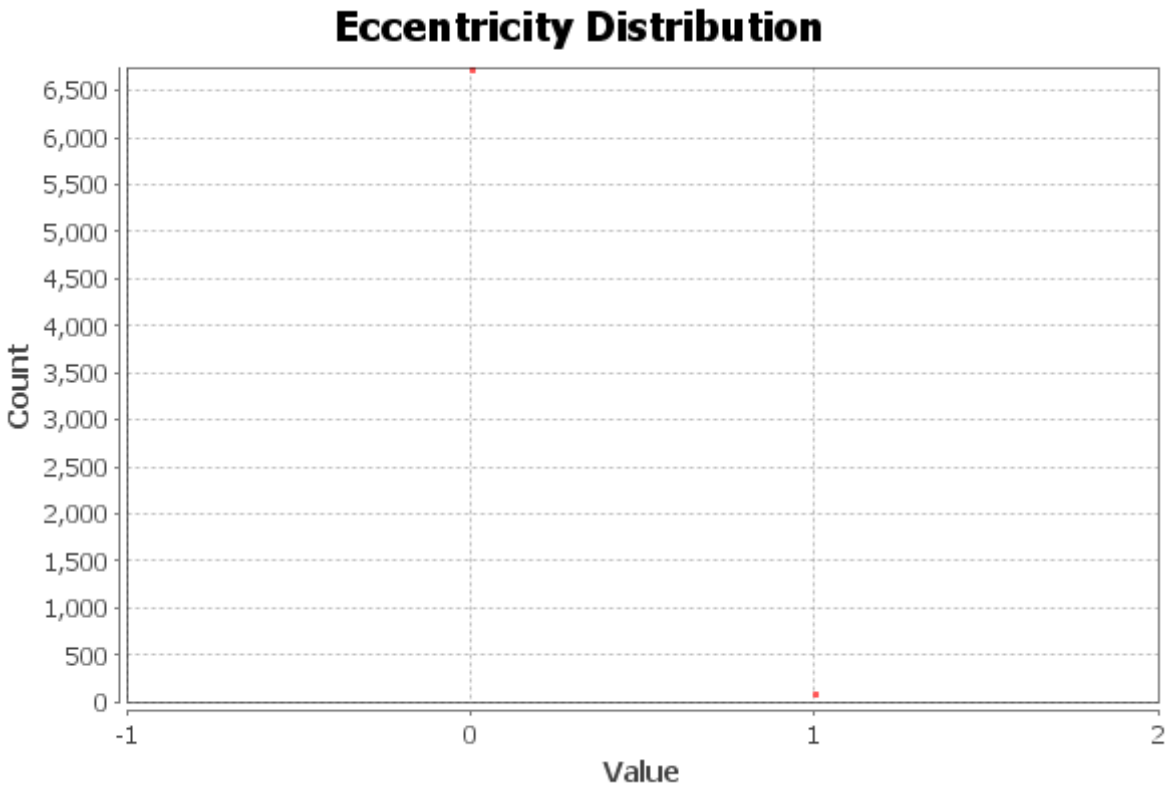

Figure 3.10: Eccentricity Distribution

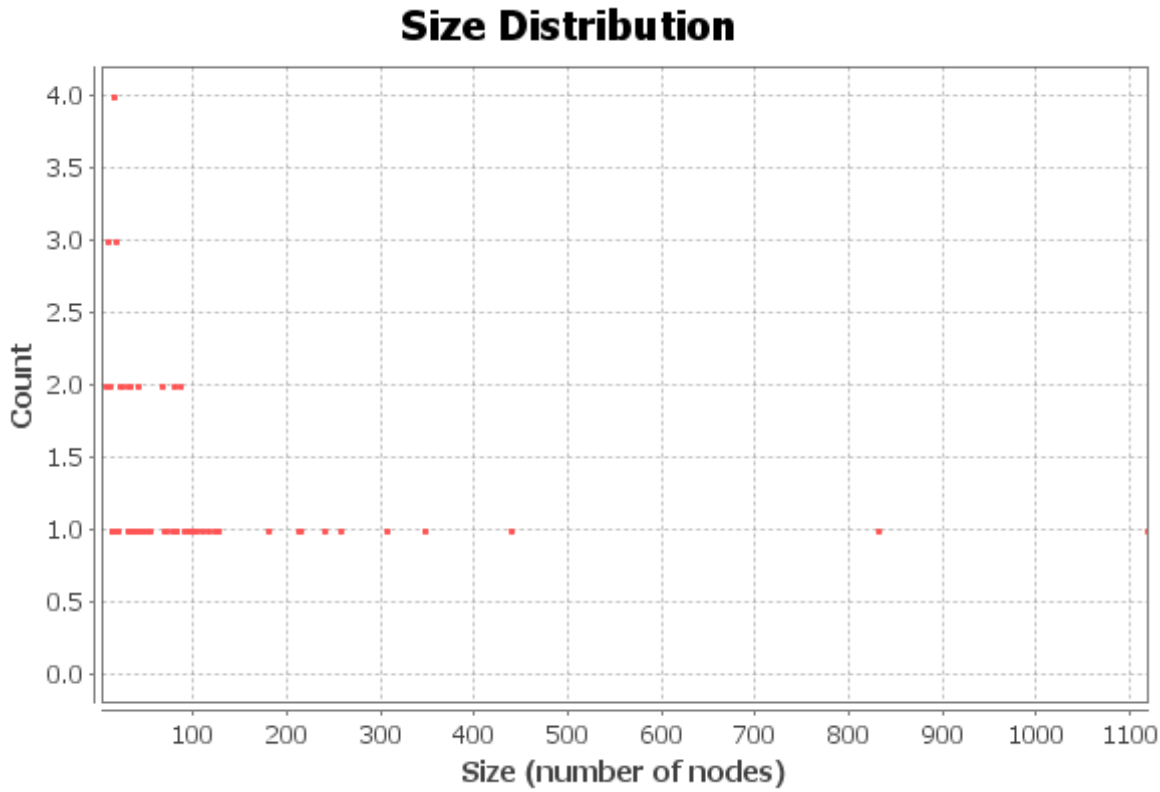

Figure 3.11: Connected Components Size Distribution

REFERENCES

[1] Beautiful soup documentation. http://www.crummy.com/software/BeautifulSoup/bs4/doc/. Accessed: 2014-10-03.

[2] Generate pseudo-random numbers. https://docs.python.org/2/library/random.html. Accessed: 2014-10-03.

[3] Graphviz dot format. https://gephi.github.io/users/supported-graph-formats/graphviz-dot-format/. Accessed: 2014-10-05.

[4] Parse urls into components. https://docs.python.org/2/library/urlparse.html. Accessed: 2014-10-04.

[5] Reading and writing files. https://docs.python.org/2/tutorial/inputoutput.html#reading-and-writing-files. Accessed: 2014-10-03.