Assignment 4

Fall 2014
CS595 Web Science
Dr. Michael Nelson

Mathew Chaney

October 8, 2014

Contents

List of Figures

Listings

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.

```python
#! /usr/bin/python

import os
import random
import pickle
from bs4 import BeautifulSoup

LINKS_DIR = 'links' + os.sep
HTML_DIR = 'html' + os.sep

uri_map = pickle.load(open('uri_map', 'rb'))

def get_links(filename):
    '''Parses HTML contents and returns tuple of (filename, uri, list of links)'''
    with open(HTML_DIR + filename) as infile:
        uri = infile.readline().rstrip('\n')
        soup = BeautifulSoup(infile.read())
        links = [link['href'].encode('utf-8') for link in soup.find_all('a') if link.has_attr('href')]
        return filename, uri, links

def write_links(filename, uri, links):
    '''Writes URI and outlinks to file'''
    print('Writing {}'.format(uri))
    with open(LINKS_DIR + filename, 'w') as outfile:
        outfile.write('{}\n'.format(uri))
        for link in links:
            outfile.write('{}\n'.format(link))

if __name__ == '__main__':
    # random selection of 100 URIs in series from sorted list
    start = random.randint(0, 899)
    keys = uri_map.keys()
    keys.sort()
    for i in xrange(start, start + 100):
        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.

```python
#! /usr/bin/python

import os
import random
from urlparse import urlparse

LINKS_DIR = 'links' + os.sep

def extract(uri):
    '''Strip the path from the URI and return the domain name'''
    parsed = urlparse(uri.strip())
    if parsed.netloc:
        return parsed.netloc + parsed.path[:20]
    else:
        return ''

def write_dot(uri, links, outfile):
    '''Write the uri and all of its links (with labels) to the file "outfile" in dot format'''
    for link, label in links.iteritems():
        outfile.write('\t"{}" -> "{}";\n'.format(uri, link))
        outfile.write('\t"{}" [label="{}"]\n'.format(uri, extract(uri)))
        outfile.write('\t"{}" [label="{}"]\n'.format(link, label))

if __name__ == '__main__':
    with open('links.gv', 'w') as outfile:
        outfile.write('digraph unix {\n')
        for filename in os.listdir('links'):
            with open(LINKS_DIR + filename, 'r') as infile:
                uri = infile.readline().rstrip('\n')
                links = {link.rstrip('\n'): extract(link) for link in infile if extract(link)}
                write_dot(uri, links, outfile)
        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].

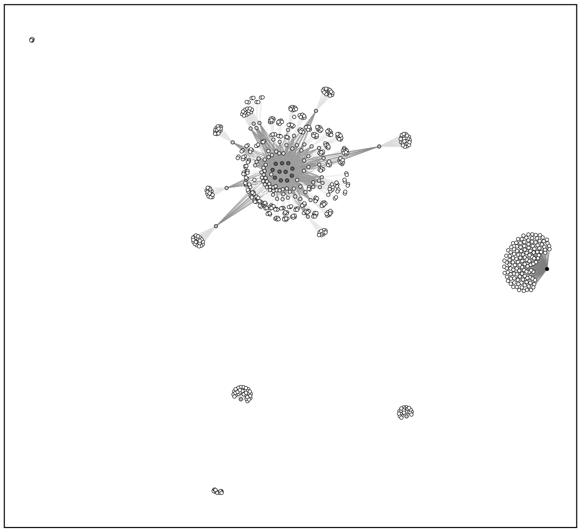

Figure 1: Graph Overview

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.

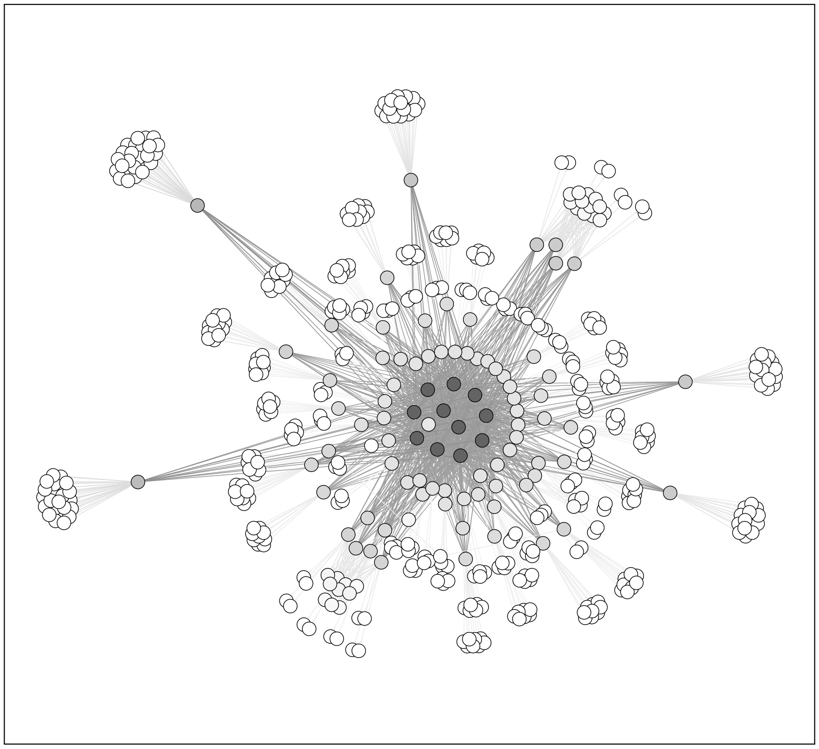

Figure 2: Strongly Connected Core

The nodes in the center are mostly from google domains – YouTube.com, various google.com subdomains, 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 online resources.

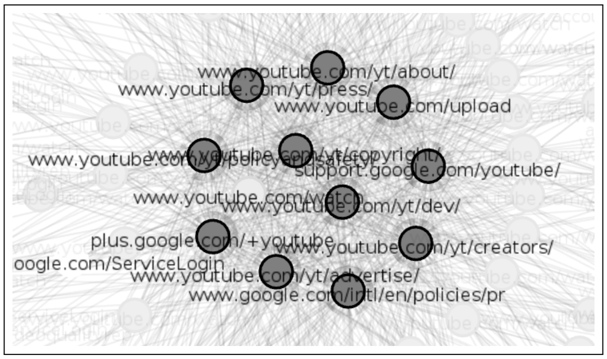
www.youtube.com/yt/about/
www.youtube.com/yt/press/
www.youtube.com/upload
www.youtube.com/yt/policyandsafety/
www.youtube.com/yt/copyright/
support.google.com/youtube/
www.youtube.com/watch
www.youtube.com/yt/dev/
plus.google.com/+youtube
www.youtube.com/yt/creators/
oogle.com/ServiceLogin
www.youtube.com/yt/advertise/
www.google.com/intl/en/policies/pr

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.

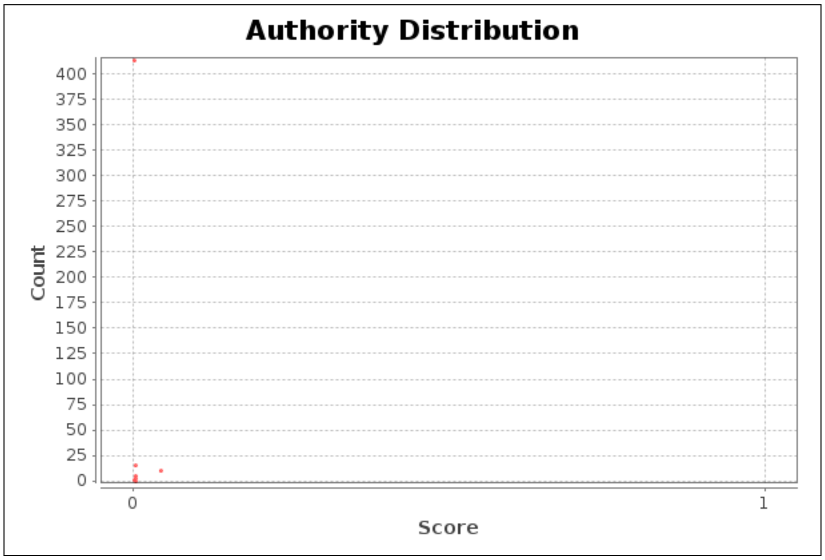

Figure 4: HITS Authorities Scores

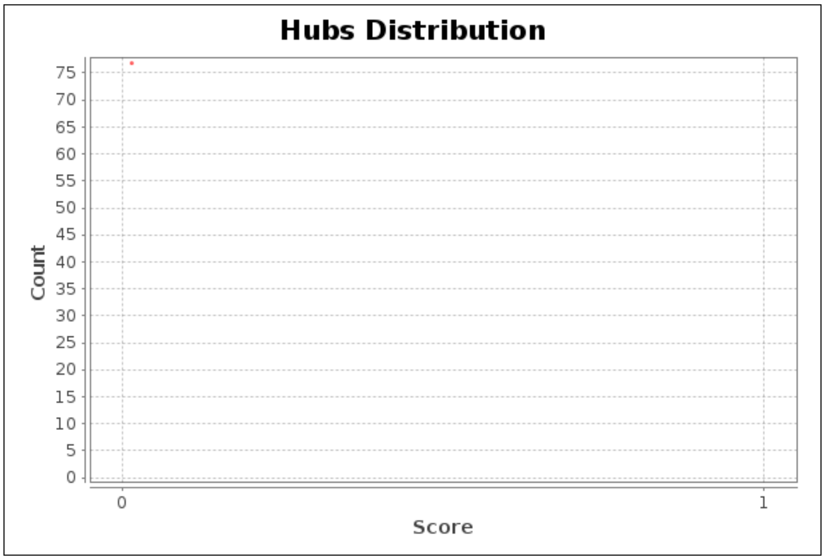

Figure 5: HITS Hubs Scores

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

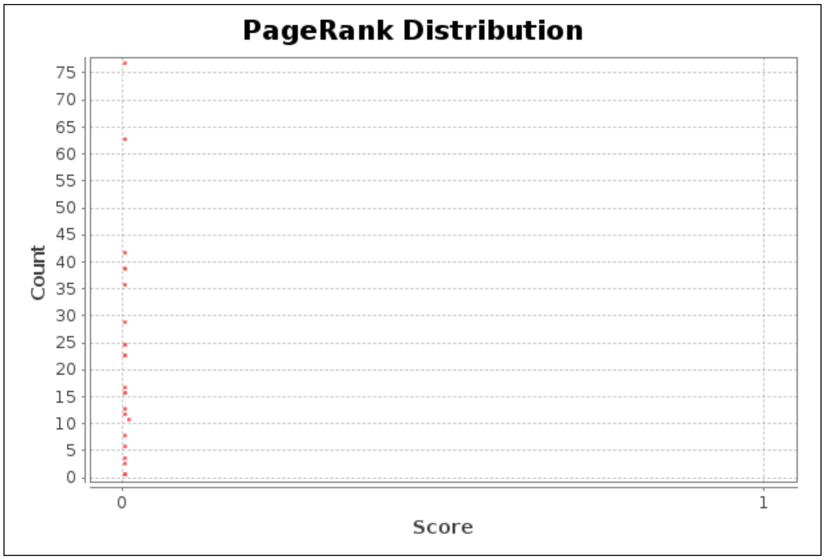

Figure 6: Page Rank Scores

The average degree of the graph was also calculated. The results are found in Figures 7, 8 and 9.

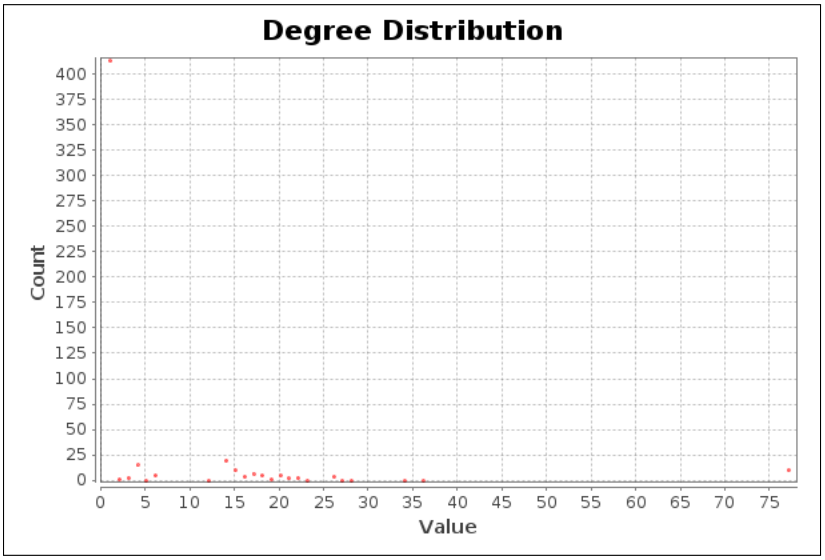

Figure 7: Average Degree Distributions

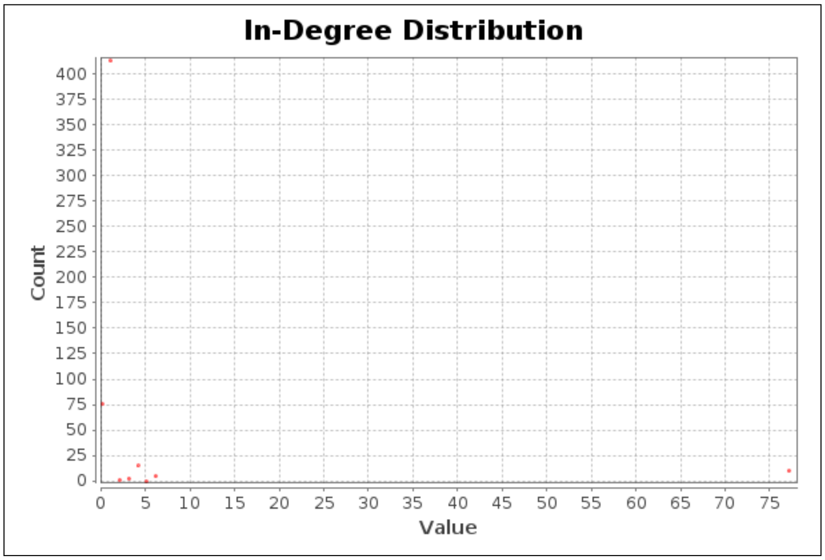

Figure 8: In-Degree Distributions

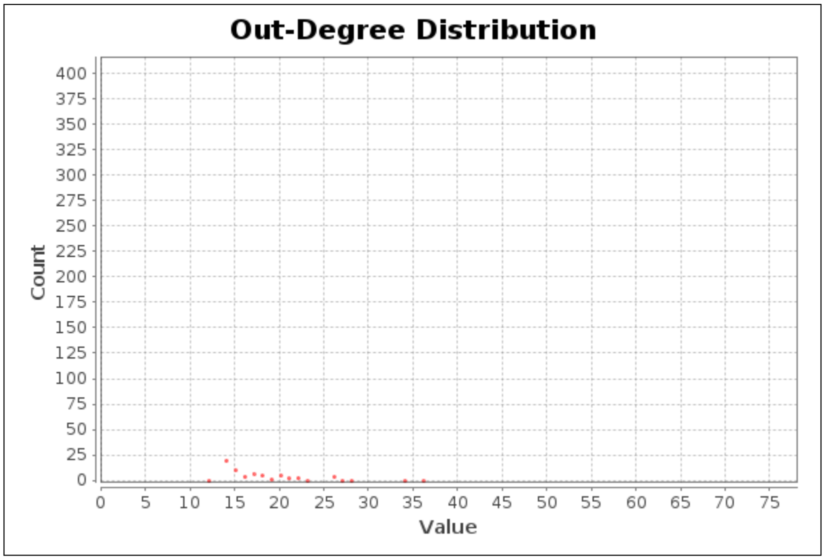

Figure 9: Out-Degree Distributions

The network diameter was calculated and the results are found in Figures 10, 11 and 12.

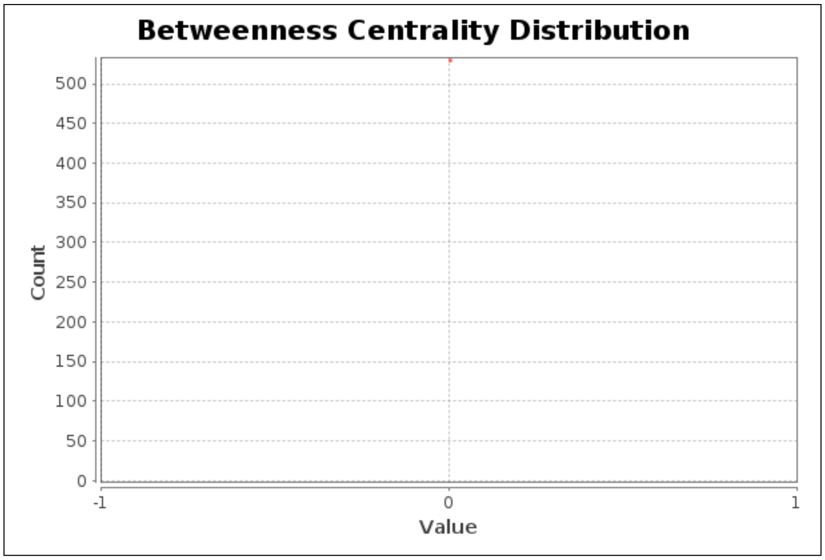

Figure 10: Betweenness Centrality Distribution

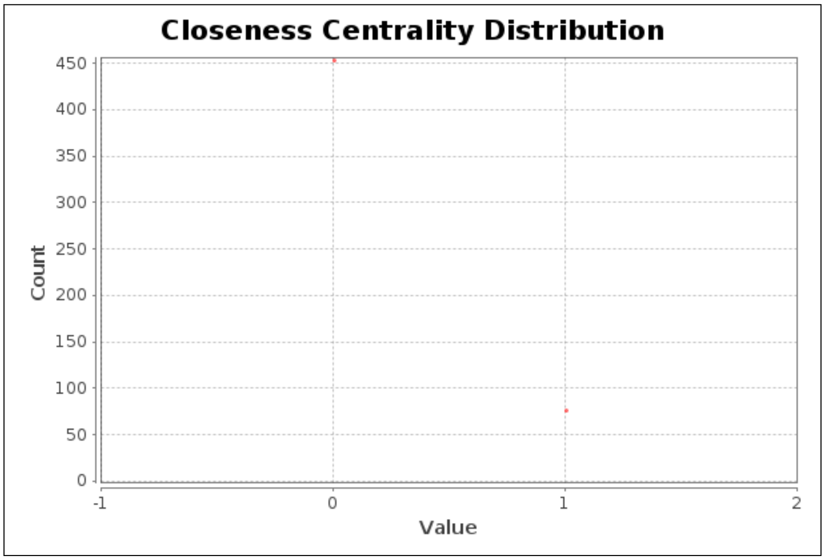

Figure 11: Closeness Centrality Distribution

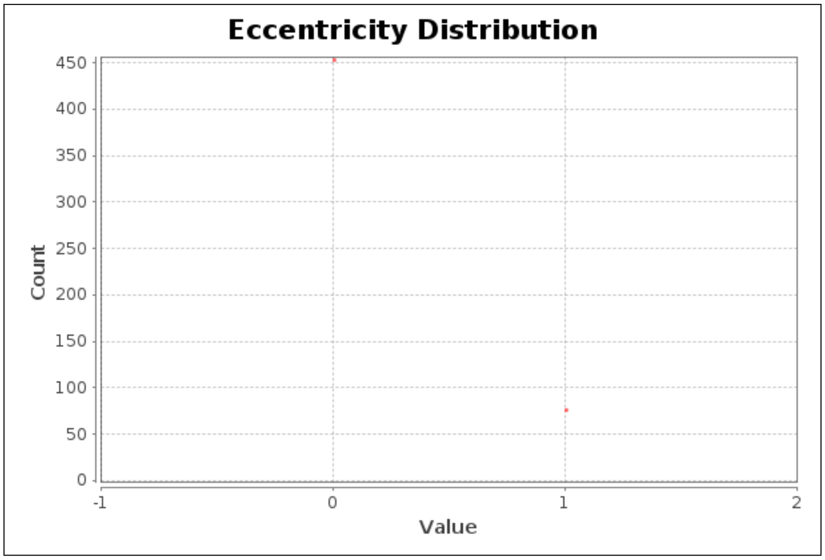

Figure 12: Eccentricity Distribution

The connected components were found. The results are in Figure 13

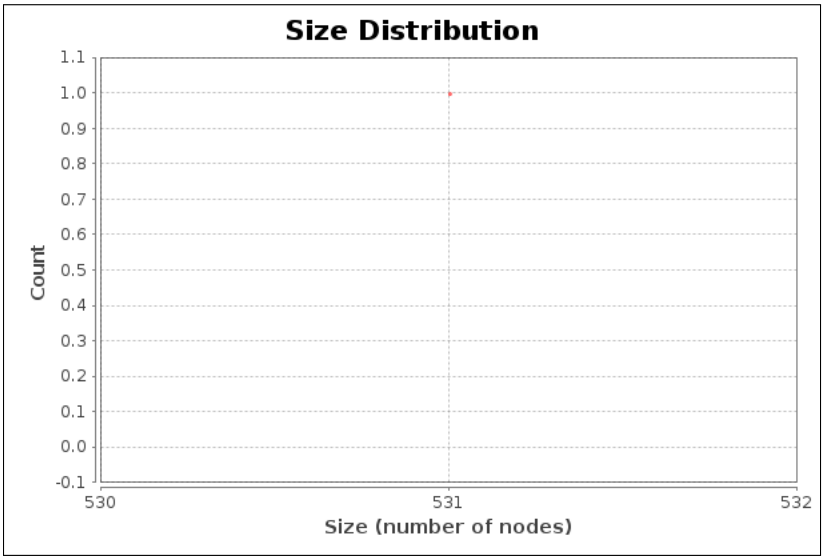

Figure 13: Connected Components

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.