Assignment 4

CS 595: Introduction to Web Science

Fall 2013

Shawn M. Jones

Finished on October 10, 2013

1

Question

```
1.  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).
```

Answer

To get the 100 links, I decided it was likely that `cnn.com` would likely link to itself a lot, so I typed the following command to extract 100 of those entries from my `shalist-final.txt` file from Assignment 3.

```
grep "cnn.com" shalist-final.txt | head -n 100 > pages-to-extract.txt
```

The program for extracting the links from the given list of pages is in Listing 1. It is executed as follows

```
./getLinks.py pages-to-extract.txt ../../assignment3/q1/collection linkFiles
```

The first argument is a file containing the list of URIs and SHA hashes. The second argument is the directory to search through for the files with the name corresponding to the SHA hashes contained in the file from the first argument. The last argument is the directory to write the link information out to.

Once this is done, the `linkFiles` directory contains files like `f25db00c9aeb80bb92f465cd7794536d87889958.links.json` which are single key JSON dictionaries containing the URI and the URIs the representation links to.

I chose JSON because it should be relatively easy to parse for Question 2, and also because the text for question 1 states "use any syntax that is easy for you". I wanted to save time and not write my own parser when Python has one built in. On the first pass, I had the script generate one large JSON file, which could easily be read for Question 2, and would work well for our 100 links. But having the script write out a file per URI would scale better when we look at millions of URIs, especially if there is a bug in the script or hardware failure.

The function on line 62 extracts the links using BeautifulSoup, and the first pass at this script stopped there. Once I examined the output, I realized that there were a large number of relative URIs, JavaScript links, URIs missing complete paths, etc. This caused the creation of the function on line 12, which tries to account for that madness.

```
#!/usr/local/bin/python3

import sys
import os
import os.path
import shutil
import json
from urllib.parse import urlparse

from bs4 import BeautifulSoup

def processLinks(uri, links):
    """
        Rules:
            * If it doesn't start with http[s]://, then:
                * if it starts with #, then replace with uri
                * if it starts with 'javascript:', we don't know
                     where it goes,
                    throw it out, as mentioned in class
                * if it starts with /, prepend the scheme and
                    netloc from
                    urlparse to it
            * If it contains ?, strip out everything after ? <--
                not done
    """

    fixedLinks = []

    mainpr = urlparse(uri)
    mainscheme = mainpr[0]
    mainnetloc = mainpr[1]
    mainpath = '/' if not mainpr[2] else mainpr[2]
    mainparams = mainpr[3]
    mainquery = mainpr[4]
    mainfrag = mainpr[5]

    for link in links:

        linkpr = urlparse(link)
        scheme = mainscheme if not linkpr[0] else linkpr[0]
        netloc = mainnetloc if not linkpr[1] else linkpr[1]
        path = '/' if not linkpr[2] else linkpr[2]
        params = linkpr[3]
        query = linkpr[4]
        frag = linkpr[5]

        if scheme != 'javascript':
            if path:
                if path[0] != '/':
                    path = mainpath + path

            fixedLink = scheme + '://' + netloc + path + params

            if query:
                fixedLink = fixedLink + '?' + query

            fixedLinks.append(fixedLink)

    # we only care about the set of links, not how many
    fixedLinks = list(set(fixedLinks))

    return fixedLinks


def extractLinks(data):
    soup = BeautifulSoup(data)
    links = []

    for a in soup.find_all('a', href=True):
        links.append(a['href'])

    return links

if __name__ == "__main__":
    urilist = sys.argv[1]
    searchdir = sys.argv[2]
    outputdir = sys.argv[3]

    if os.path.exists(outputdir):
        if os.path.isdir(outputdir):
            shutil.rmtree(outputdir)
        else:
            sys.stderr.write('cannot remove ' + outputdir)
            sys.exit(212)

    os.mkdir(outputdir)

    f = open(urilist)

    filelist = {}

    for line in f:
        (uri, checksum) = line.split()
        fname = checksum + '.raw'
        filelist[fname] = uri

    f.close()

    for root, _, files in os.walk(searchdir):
        for filename in files:
            if filename in filelist:
                f = open( os.path.join(root, filename) )
                data = f.read()
                f.close()

                linkDict = {}

                uri = filelist[filename]

                links = extractLinks(data)
                links = processLinks(uri, links)
                linkDict[uri] = links

                outputfile = filename.replace('.raw', '.links.
                    json')
                o = open( os.path.join(outputdir, outputfile), '
                    w' )
                o.write(
                    json.dumps(linkDict, sort_keys=True,
                    indent=4, separators=(',', ': ') )
                    )
                o.close()
```
Listing 1: Python program for printing out links from 100 pages

2

Question

```
2.  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.)
```

Answer

Because I stored the data from Question 1 in JSON format, the script for Question 2 is much smaller. It was run like so:

```
./makeDot.py ../q1/linkFiles > linkGraph.dot
```

where the first argument is the directory to search for the JSON files from Question 1. The script prints to `stdout` so a redirect creates the DOT file.

The script is shown in Listing 2. Lines 27-29 print the DOT code for each edge, with labels. Line 10 prints the opening DOT block and line 31 prints the close to the block.

```
 1  #!/usr/local/bin/python3
 2
 3  import sys
 4  import os
 5  import json
 6
 7  if __name__ == "__main__":
 8      searchdir = sys.argv[1]
 9
10      print("digraph linkForest {")
11
12      for root, _, files in os.walk(searchdir):
13          for filename in files:
14              f = open( os.path.join( root, filename ) )
15              data = f.read()
16              f.close()
17
18              info = json.loads(data)
19
20              (uri, links) = info.popitem()
21
22              for link in links:
23
24                  link = link.replace('"', '%22')
25
26                  print(
27                      '"' + uri + '" -> "' + link + '"' +
28                      ' [ label = "' + uri + ' links to ' + link +
                            '" ];'
29                      )
30
31      print("}")
```

Listing 2: Python program for printing converting links from Question 1 into a DOT file

3

Question

```
3.  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.
```

Answer

The graph described by the dot file in Question 2 contains 4276 nodes and 11488 edges.

From Gephi, visualizing the graph produced a horrid monstrosity shown in Figure 1. Choosing all of my links from CNN is what led to this. The shaving on the right and left are not actually unconnected edges. Gephi appears to have cropped the image on export.

After using the *Yifan Hu* layout, the graph became the much better, as shown in Figure 2. It turns out that each of the subgraphs in the CNN graph somewhat correspond to parts of the CNN site (and outbound links). Some of these have been colorized to demonstrate this phenomenon.

- brown-orange corresponds to `religion.blogs.cnn.com`
- green corresponds to URIs with `eatocracy.cnn.com`
- red-orange is `ac360.blogs.cnn.com`
- purple is `crossfire.blogs.cnn.com`
- blue-yellow is `fortune.cnn.com`
- violet-yellow is `money.cnn.com`

This does convey some interesting phenomenon. The nodes in the center are the typical menu links found on each page, but there are 10 nodes shared between the brown-orange group and the green group, which turn out to be menu items shared between `eatocracy.cnn.com` and `religion.blogs.cnn.com`, but not used anywhere else.

Unfortunately, Gephi did not have enough colors to convey all of the subgroups.

All of CNN's pages link to the same template menu links, which explains much of the strongly connected parts. I expected most of the pages to link to pages in the center of the graph, but I did not expect this much "clumping".

Because of the clumping Figure 3 shows low authority scores on all pages, and Figure 4 shows what look to be four hubs, but are likely so many data points crammed into these four scores. Also, the PageRank for all pages approaches 0, as shown in Figure 5 because so many of the pages link to one another.

As shown in Figure 7, few nodes have a high in-degree. These are likely menu options, with those with a small in-degree being actual stories. Surprisingly, Figure 8 shows that a lot of pages do seem to link out, but in an almost horizontal linear pattern.

The graph has a network diameter of 2, with 26508 shortest paths. The average path length is ≈ 1.57. Figure 10 shows that most of the nodes are so connected that their closeness is quite low. As Figure 9 shows, none of the nodes are bridges alone. Looking back at Figure 2, we see many edges connecting each node, so that may explain this value.

Figure 12 shows the 4276 strongly connected components in this graph. Even though the visualization in Figure 2 shows distinct subgraphs, so many of them are strongly connected within a few hops that it is difficult to classify them as weakly connected.

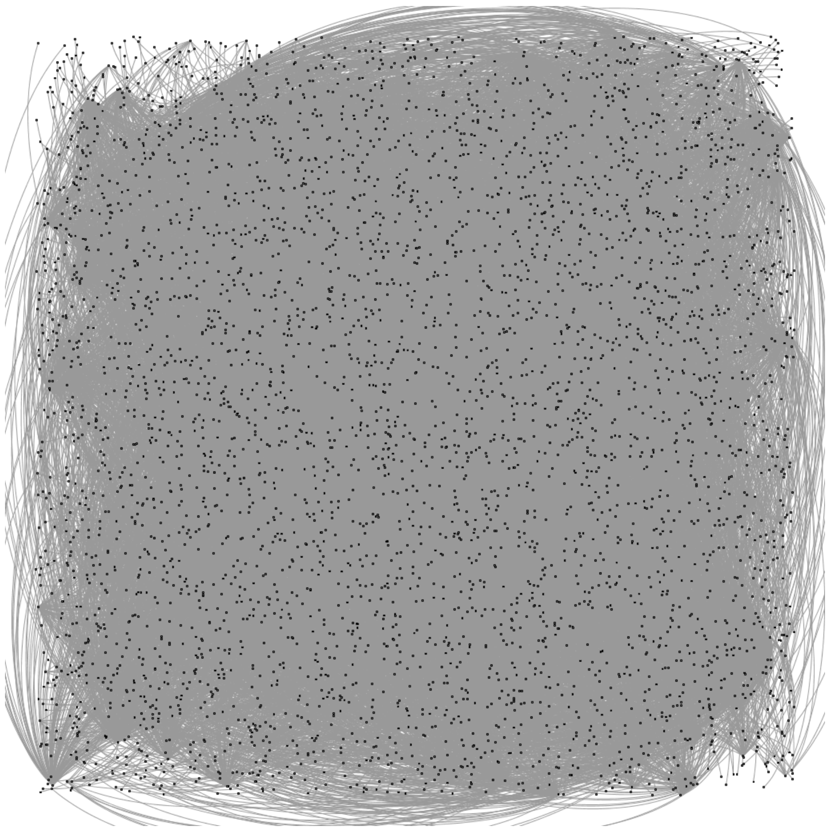

Figure 1: Initial Visualization of the Dot file produced from Question 2

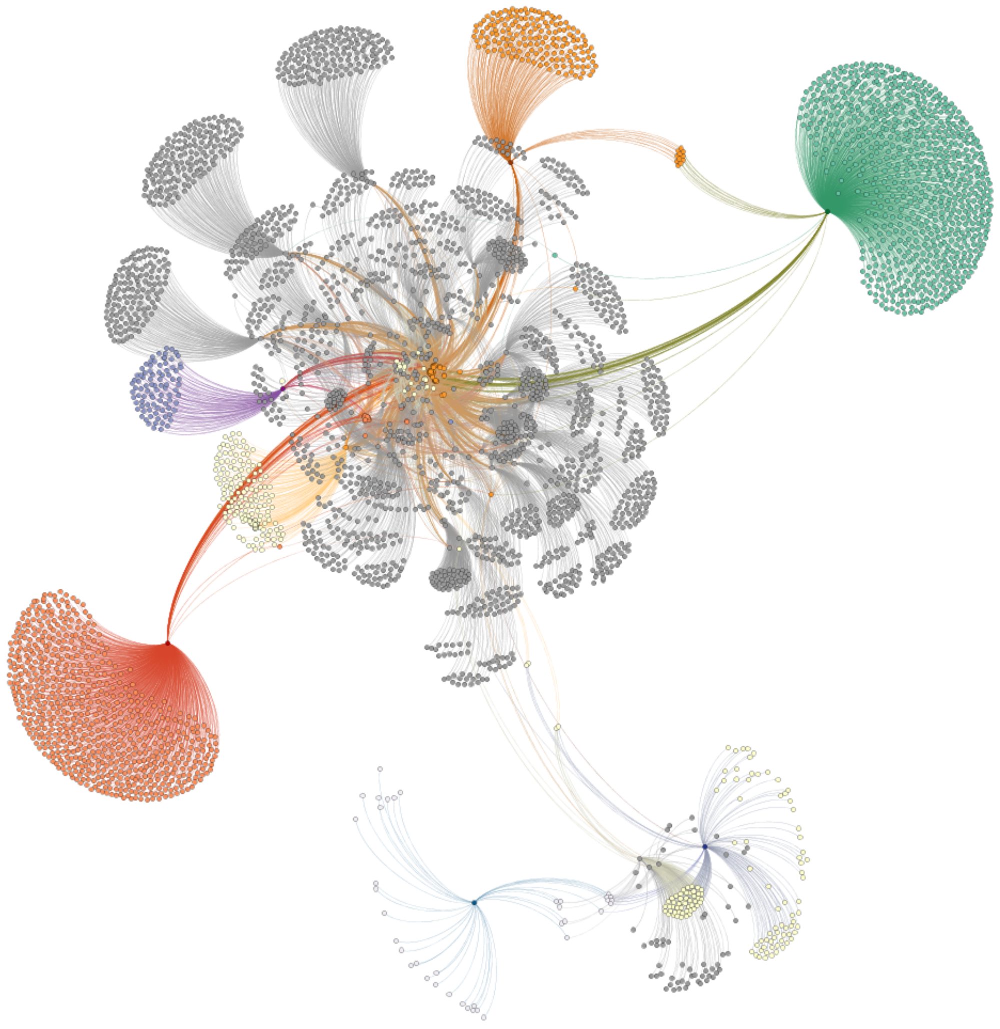

Figure 2: Yifan Hu Visualization of the Dot file produced from Question 2

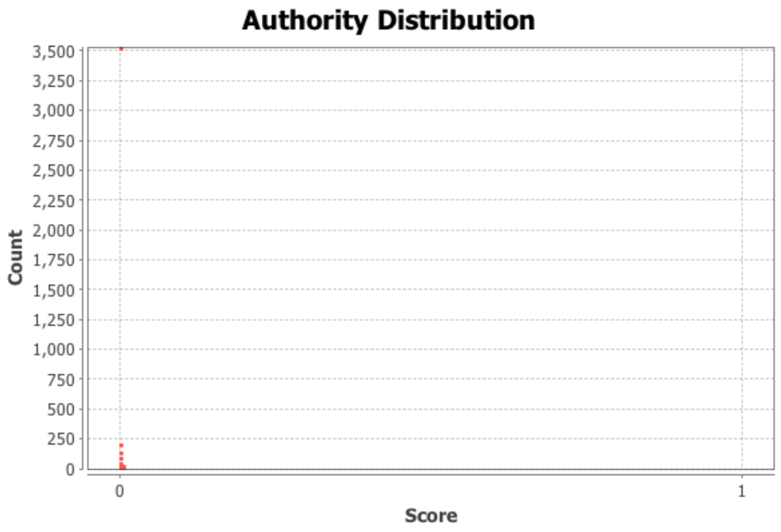

Figure 3: Plot of the HITS authorities from the Dot file produced from Question 2

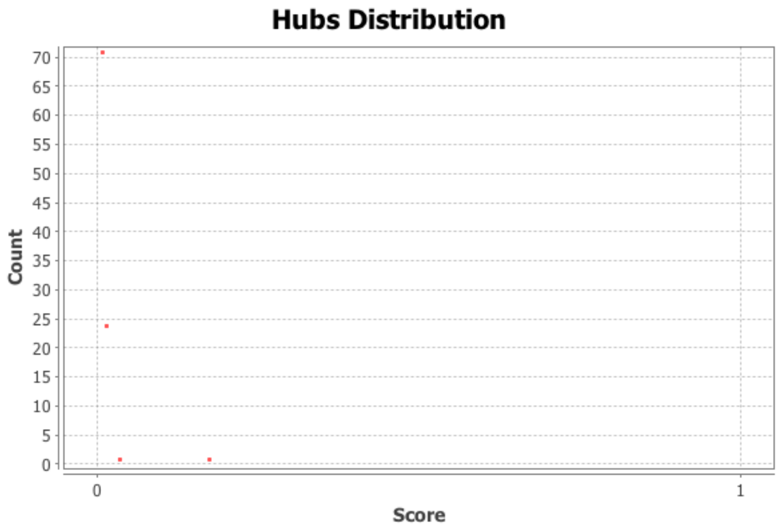

Figure 4: Plot of the HITS hubs from the Dot file produced from Question 2

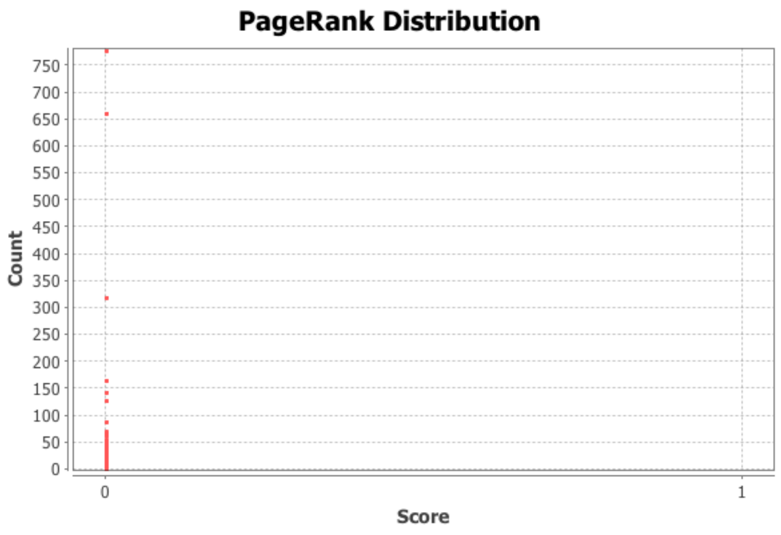

Figure 5: Plot of PageRank from the Dot file produced from Question 2

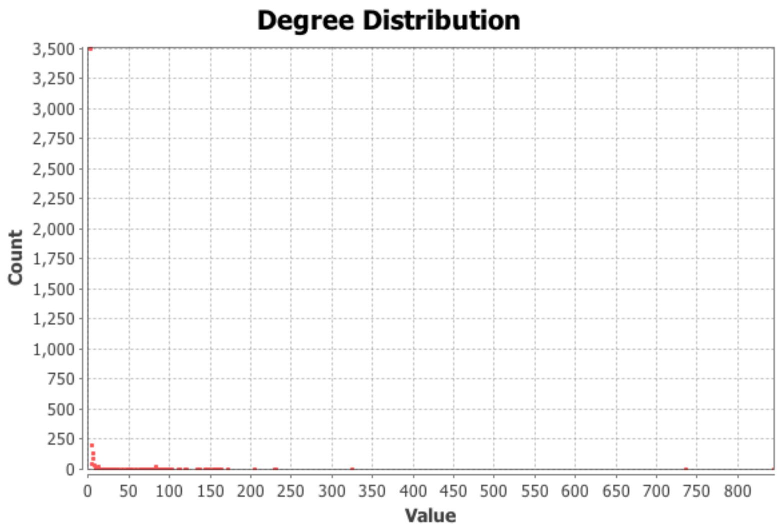

Figure 6: Plot of the Avg Degree Distribution from the Dot file produced from Question 2

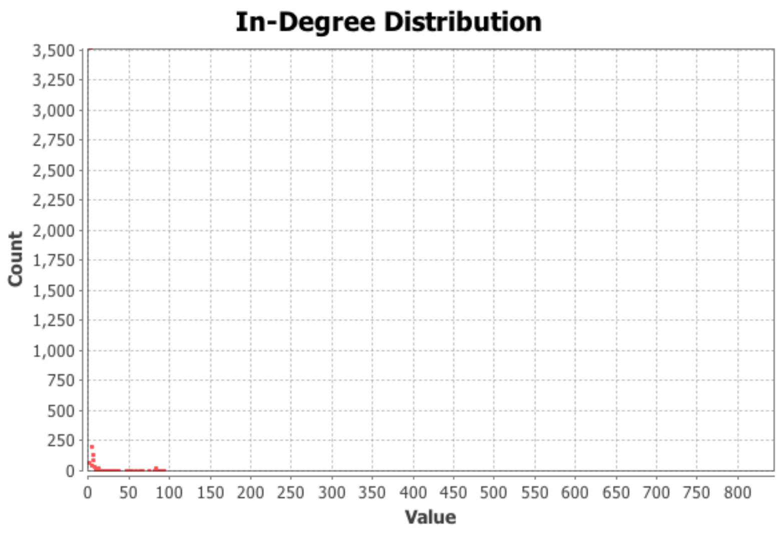

Figure 7: Plot of the Avg Degree Indegree Distribution from the Dot file produced from Question 2

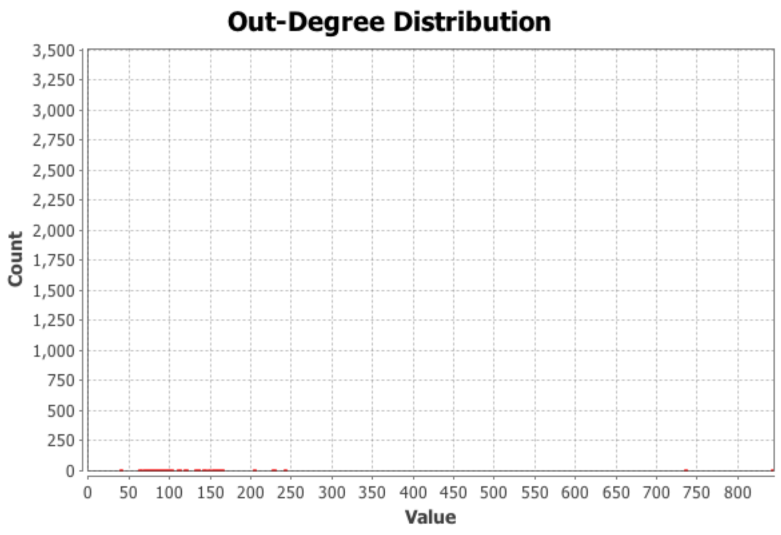

Figure 8: Plot of the Avg Degree Outdegree Distribution from the Dot file produced from Question 2

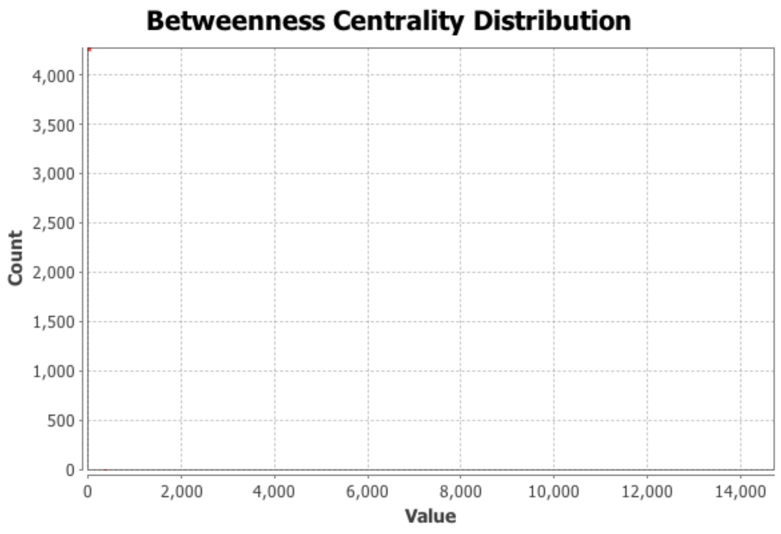

Figure 9: Plot of the Network Diameter Betweenness Centrality Distribution from the Dot file produced from Question 2

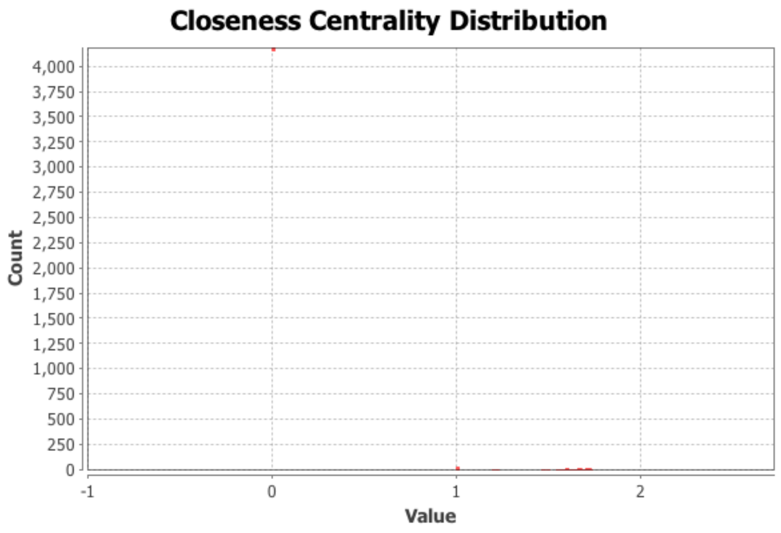

Figure 10: Plot of the Network Diameter Closeness Centrality Distribution from the Dot file produced from Question 2

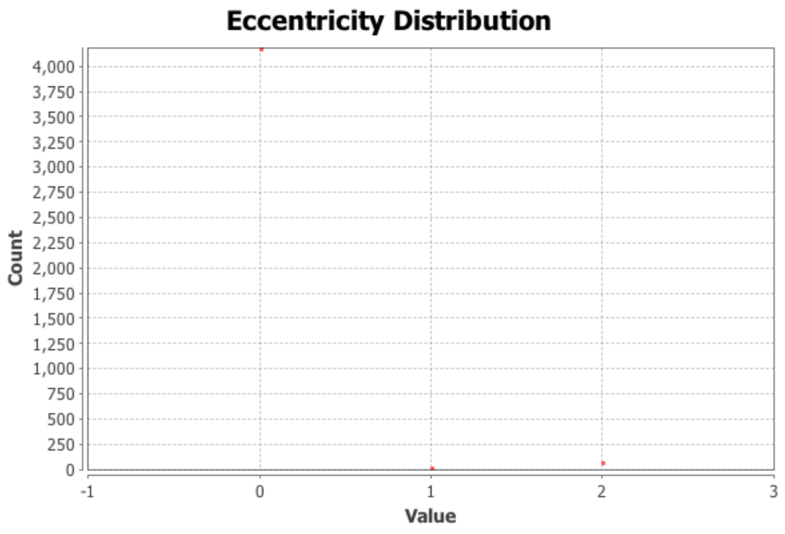

Figure 11: Plot of the Network Diameter Eccentricity Distribution from the Dot file produced from Question 2

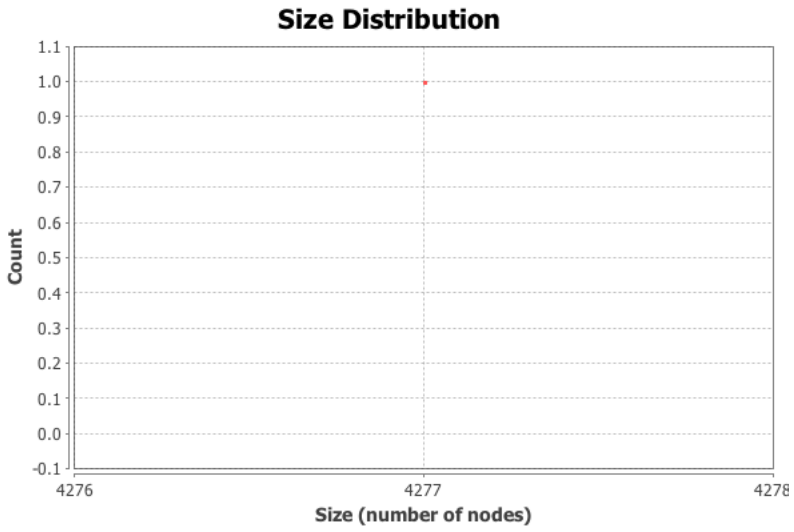

Figure 12: Plot of the Connected Components from the Dot file produced from Question 2