rdf-network-analysis

May 15, 2020

1 Network Analysis of RDF Graphs

In this notebook we provide basic facilities for performing network analyses of RDF graphs easily with Python rdflib and networkx

We do this in 4 steps: 1. Load an arbitrary RDF graph into rdflib 2. Get a subgraph of relevance (optional) 3. Convert the rdflib Graph into an networkx Graph, as shown here 4. Get an network analysis report by running networkx's algorithms on that data structure

1.1 0. Preparation

[13]:
```
# Install required packages in the current Jupyter kernel
# Uncomment the following lines if you need to install these libraries
# If you run into permission issues, try with the --user option
import sys
# !pip install -q rdflib networkx matplotlib scipy
!{sys.executable} -m pip install rdflib networkx matplotlib scipy --user

# Imports
from rdflib import Graph as RDFGraph
from rdflib.extras.external_graph_libs import rdflib_to_networkx_graph
import networkx as nx
from networkx import Graph as NXGraph
import matplotlib.pyplot as plt
import statistics
import collections
```

```
Requirement already satisfied: rdflib in /home/amp/.local/lib/python3.8/site-
packages (5.0.0)
Requirement already satisfied: networkx in /home/amp/.local/lib/python3.8/site-
packages (2.4)
Requirement already satisfied: matplotlib in
/home/amp/.local/lib/python3.8/site-packages (3.2.1)
Requirement already satisfied: scipy in /home/amp/.local/lib/python3.8/site-
packages (1.4.1)
Requirement already satisfied: pyparsing in /usr/lib/python3/dist-packages (from
rdflib) (2.4.6)
Requirement already satisfied: six in /usr/lib/python3/dist-packages (from
rdflib) (1.14.0)
```

```
Requirement already satisfied: isodate in /home/amp/.local/lib/python3.8/site-
packages (from rdflib) (0.6.0)
Requirement already satisfied: decorator>=4.3.0 in /usr/lib/python3/dist-
packages (from networkx) (4.4.2)
Requirement already satisfied: kiwisolver>=1.0.1 in
/home/amp/.local/lib/python3.8/site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: numpy>=1.11 in /usr/lib/python3/dist-packages
(from matplotlib) (1.17.4)
Requirement already satisfied: cycler>=0.10 in
/home/amp/.local/lib/python3.8/site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/lib/python3/dist-
packages (from matplotlib) (2.7.3)
```

1.2 1. Loading RDF

The first thing to do is to load the RDF graph we want to perform the network analysis on. By executing the next cell, we'll be asked to fill in the path to an RDF graph. This can be any path, local or online, that we can look up.

Any of the Turtle (`ttl.`) files that we include with this notebook will do; for example, `bsbm-sample.ttl`. But any Web location that leads to an RDF file (for example, the GitHub copy of that same file at https://raw.githubusercontent.com/albertmeronyo/rdf-network-analysis/master/bsbm-sample.ttl; or any other RDF file on the Web like https://raw.githubusercontent.com/albertmeronyo/lodapi/master/ghostbusters.ttl) will work too.

[14]:
```
# RDF graph loading
path = input("Path or URI of the RDF graph to load: ")
rg = RDFGraph()
rg.parse(path, format='turtle')
print("rdflib Graph loaded successfully with {} triples".format(len(rg)))
```

```
Path or URI of the RDF graph to load: wechanged-british.ttl
rdflib Graph loaded successfully with 155 triples
```

1.3 2. Get a subgraph out of the loaded RDF graph (optional)

This cell can be skipped altogether without affecting the rest of the notebook; but it will be useful if instead of using the whole RDF grahp of the previous step, we just want to use a subgraph that's included in it.

By executing the next cell, we'll be asked two things:

- The URI of the ''entiy" type we are interested in (e.g. `http://dbpedia.org/ontology/Band`)
- The URI of the ''relation" connecting entities we are interested in (e.g. `http://dbpedia.org/ontology/influencedBy`)

Using these two, the notebook will replace the original graph with the subgraph that's constructed by those entity types and relations only.

```
[ ]: # Subgraph construction (optional)
entity = input("Entity type to build nodes of the subgraph with: ")
relation = input("Relation type to build edges of the subgraph with: ")

# TODO: Use entity and relation as parameters of a CONSTRUCT query
query = """
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
CONSTRUCT {{ ?u a {} . ?u {} ?v }} WHERE {{ ?u a {} . ?u {} ?v }}""".
↪format(entity, relation, entity, relation)
# print(query)
subg = rg.query(query)

rg = subg
```

1.4 3. Converting rdflib.Graph to networkx.Graph

Thanks to the great work done by the rdflib developers this step, which converts the basic graph data structure of rdflib into its equivalent in networkx, is straightforward. Just run the next cell to make our RDF dataset ready for network analysis!

```
[15]: # Conversion of rdflib.Graph to networkx.Graph
G = rdflib_to_networkx_graph(rg)
print("networkx Graph loaded successfully with length {}".format(len(G)))
```

```
networkx Graph loaded successfully with length 174
```

1.5 4. Network analysis

At this point we can run the network analysis on our RDF graph by using the networkx algorithms. Exeucting the next cell will output a full network analysis report, with the following parts:

- General network metrics (network size, pendants, density)
- Node centrality metrics (degree, eigenvector, betwenness). For these, averages, stdevs, maximum, minimum and distribution histograms are given
- Clustering metrics (connected components, clustering)
- Overall network plot

The report can be easily selected and copy-pasted for further use in other tools.

```
[17]: # Analysis

def mean(numbers):
    return float(sum(numbers)) / max(len(numbers), 1)

def number_of_pendants(g):
    """
    Equals the number of nodes with degree 1
    """
    pendants = 0
```

```
    for u in g:
        if g.degree[u] == 1:
            pendants += 1
    return pendants


def histogram(l):
    degree_sequence = sorted([d for n, d in list(l.items())], reverse=True)
    degreeCount = collections.Counter(degree_sequence)
    deg, cnt = zip(*degreeCount.items())
    print(deg, cnt)

    fig, ax = plt.subplots()
    plt.bar(deg, cnt, width=0.80, color='b')

    plt.title("Histogram")
    plt.ylabel("Count")
    plt.xlabel("Value")
    ax.set_xticks([d + 0.4 for d in deg])
    ax.set_xticklabels(deg)

    plt.show()

# Network size
print("NETWORK SIZE")
print("============")
print("The network has {} nodes and {} edges".format(G.number_of_nodes(), G.
 ↪number_of_edges()))
print()

# Network size
print("PENDANTS")
print("============")
print("The network has {} pendants".format(number_of_pendants(G)))
print()

# Density
print("DENSITY")
print("============")
print("The network density is {}".format(nx.density(G)))
print()

# Degree centrality -- mean and stdev
dc = nx.degree_centrality(G)
degrees = []
for k,v in dc.items():
    degrees.append(v)
```

```python
print("DEGREE CENTRALITY")
print("=================")
print("The mean degree centrality is {}, with stdev {}".format(mean(degrees),
 ↪statistics.stdev(degrees)))
print("The maximum node is {}, with value {}".format(max(dc, key=dc.get),
 ↪max(dc.values())))
print("The minimum node is {}, with value {}".format(min(dc, key=dc.get),
 ↪min(dc.values())))
histogram(dc)
print()

# Eigenvector centrality -- mean and stdev
ec = nx.eigenvector_centrality_numpy(G)
degrees = []
for k,v in ec.items():
    degrees.append(v)

print("EIGENVECTOR CENTRALITY")
print("======================")
print("The mean network eigenvector centrality is {}, with stdev {}".
 ↪format(mean(degrees), statistics.stdev(degrees)))
print("The maximum node is {}, with value {}".format(max(ec, key=ec.get),
 ↪max(ec.values())))
print("The minimum node is {}, with value {}".format(min(ec, key=ec.get),
 ↪min(ec.values())))
histogram(ec)
print()

# Betweenness centrality -- mean and stdev
# bc = nx.betweenness_centrality(G)
# degrees = []
# for k,v in bc.items():
#     degrees.append(v)
# print("BETWEENNESS CENTRALITY")
# print("======================")
# print("The mean betwenness centrality is {}, with stdev {}".
 ↪format(mean(degrees), statistics.stdev(degrees)))
# print("The maximum node is {}, with value {}".format(max(bc, key=bc.get),
 ↪max(bc.values())))
# print("The minimum node is {}, with value {}".format(min(bc, key=bc.get),
 ↪min(bc.values())))
# histogram(bc)
# print()
```

```python
# Connected components
cc = list(nx.connected_components(G))
print("CONNECTED COMPONENTS")
print("====================")
print("The graph has {} connected components".format(len(cc)))
for i,c in enumerate(cc):
    print("Connected component {} has {} nodes".format(i,len(c)))
print()

# Clusters
cl = nx.clustering(G)
print("CLUSTERS")
print("========")
print("The graph has {} clusters".format(len(cl)))
for i,c in enumerate(cl):
    print("Cluster {} has {} nodes".format(i,len(c)))
print()

# Plot
print("Visualizing the graph:")
plt.plot()
plt.figure(1)
nx.draw(G, with_labels=False, font_weight='normal', node_size=60, font_size=8)
plt.figure(1,figsize=(120,120))
plt.savefig('example.png', dpi=1000)
```

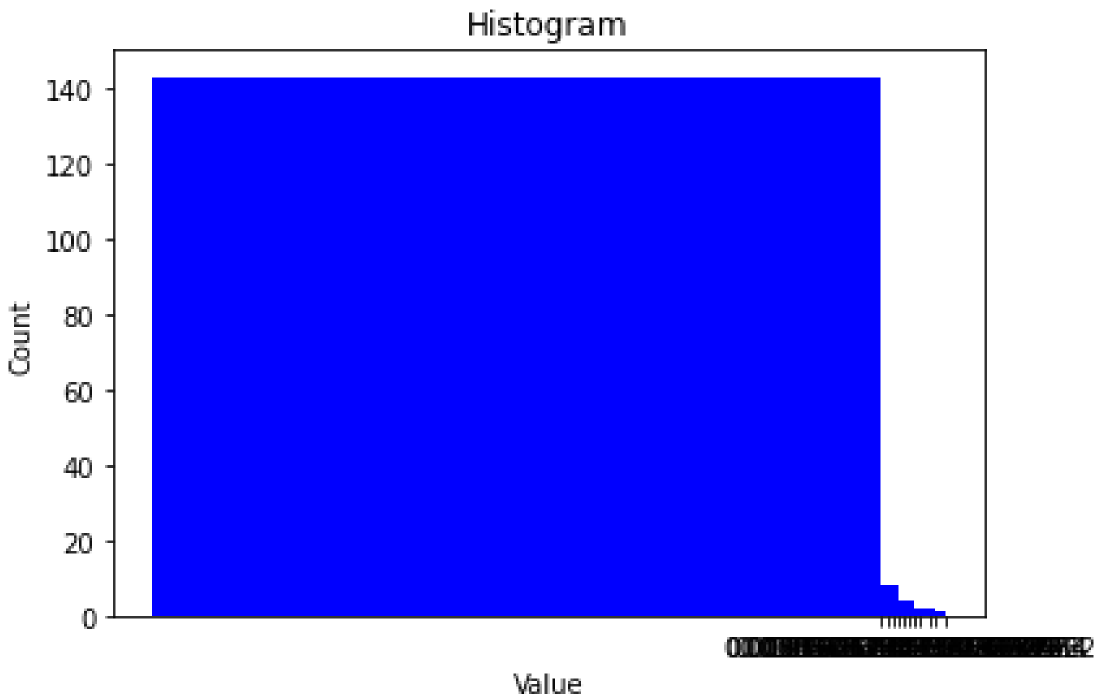
```
NETWORK SIZE
============
The network has 174 nodes and 154 edges

PENDANTS
============
The network has 143 pendants

DENSITY
============
The network density is 0.010231878280512923

DEGREE CENTRALITY
=================
The mean degree centrality is 0.010231878280512965, with stdev
0.011945260908062486
The maximum node is http://www.wikidata.org/entity/Q5373427, with value
0.07514450867052022
The minimum node is wcd_00153_id, with value 0.005780346820809248
(0.07514450867052022, 0.06358381502890173, 0.057803468208092484,
0.046242774566473986, 0.04046242774566474, 0.03468208092485549,
```

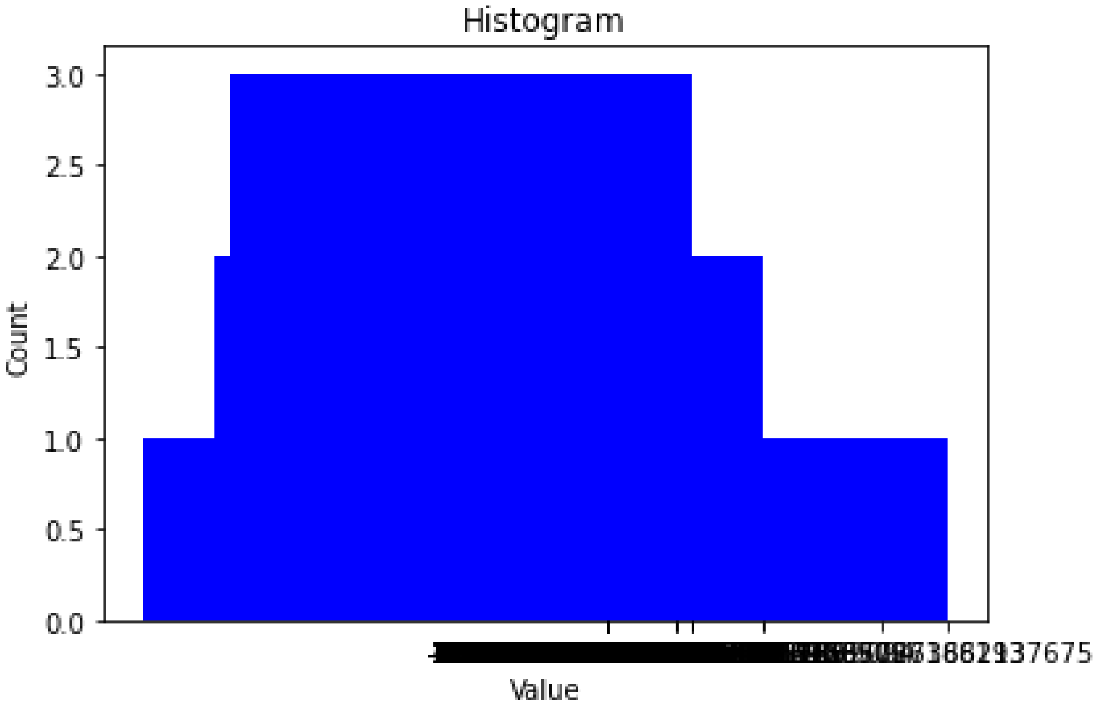
0.028901734104046242, 0.023121387283236993, 0.017341040462427744, 0.011560693641618497, 0.005780346820809248) (1, 2, 1, 2, 4, 2, 2, 8, 4, 5, 143)



EIGENVECTOR CENTRALITY
======================
The mean network eigenvector centrality is 0.01948514200092661, with stdev 0.07347435901965672
The maximum node is http://www.wikidata.org/entity/Q1382113, with value 0.5877661662137675
The minimum node is http://www.wikidata.org/entity/Q850141, with value -4.505481786806643e-16
(0.5877661662137675, 0.4744595027388193, 0.26884388627369094, 0.2688438862736909, 0.1487606117642697, 0.14876061176426966, 0.14876061176426963, 0.1487606117642696, 0.12008327450942122, 0.12008327450942116, 0.12008327450942113, 1.3593413091090835e-16, 9.989522336346594e-17, 8.834850543356192e-17, 8.367196838271632e-17, 8.130062294824438e-17, 8.109920814682827e-17, 7.639250940834377e-17, 6.60755003837558e-17, 6.458331586029222e-17, 6.378391273135149e-17, 5.776875464768082e-17, 5.5230781654597724e-17, 4.632201986965634e-17, 4.5948913688461446e-17, 4.5729664583611263e-17, 4.180583512389912e-17, 3.990563383927098e-17, 3.933785144010534e-17, 3.796689971720141e-17, 3.729655473350136e-17, 3.518282345835072e-17, 3.3858485731042385e-17, 3.116169039624658e-17, 3.0462068968057656e-17, 2.96051565728719e-17, 2.928765390998258e-17, 2.8328372810121837e-17, 2.824097600371789e-17,

2.490232387743002e-17, 2.471227843456779e-17, 2.397858460927947e-17,
2.3693851342977602e-17, 2.353385305828489e-17, 1.96280254858043e-17,
1.9396502611160725e-17, 1.7312280663938313e-17, 1.706705487415864e-17,
1.677162700055526e-17, 1.6009894424159424e-17, 1.3715157481941634e-17,
1.3392917709502256e-17, 1.3282265459074696e-17, 1.2536087619363648e-17,
1.1003381032830206e-17, 1.0367419703382782e-17, 9.690003206518953e-18,
9.119550720730226e-18, 6.669266561467615e-18, 6.2847727103900965e-18,
5.925975685261885e-18, 5.6400307279347755e-18, 4.470552530857102e-18,
4.361772042560186e-18, 3.885819732618177e-18, 2.5991871601432675e-18,
2.293730052186802e-18, 2.216763629558276e-18, 1.5562080570519094e-18,
1.5101135079016543e-18, 8.735088056917535e-19, 5.091946402563726e-19,
-7.380313762569938e-20, -9.145440149184252e-20, -1.6131610381332627e-18,
-2.4637680242200984e-18, -3.6947795051584144e-18, -4.460444437411965e-18,
-4.9426428822043514e-18, -5.0168305851493625e-18, -5.9902170029824135e-18,
-6.478455626071773e-18, -7.657928674497709e-18, -7.796840606083429e-18,
-8.172173875109491e-18, -8.93151464013122e-18, -8.983016759598348e-18,
-1.0032161946479602e-17, -1.0104496019381209e-17, -1.0123396919182274e-17,
-1.1198386672833206e-17, -1.3557292598436024e-17, -1.3730489496385865e-17,
-1.4712533516235855e-17, -1.521153625256868e-17, -1.5629764138743687e-17,
-1.700655190433631e-17, -1.7044920801100344e-17, -1.7493966488533344e-17,
-1.770710757967251e-17, -1.8165666689094258e-17, -1.8892770988936693e-17,
-1.9027407083393462e-17, -2.009564437759557e-17, -2.010810736824598e-17,
-2.0648444074975188e-17, -2.082250898753491e-17, -2.141871505865804e-17,
-2.1858624313601425e-17, -2.273348040287359e-17, -2.2919382920802238e-17,
-2.4286128663675305e-17, -2.4291955939488523e-17, -2.437511230477948e-17,
-2.4619580104228264e-17, -2.5046226881248077e-17, -2.5087553268786578e-17,
-2.5717486384176555e-17, -2.7081292267934938e-17, -3.0491495862458354e-17,
-3.068317981835574e-17, -3.093754708903187e-17, -3.157642045569366e-17,
-3.2272613405298885e-17, -3.3422726711265095e-17, -3.511117111835248e-17,
-3.639011172442013e-17, -3.681622297742476e-17, -3.744625937459483e-17,
-3.7640692960354386e-17, -3.8087987972194085e-17, -3.8272236992131277e-17,
-3.881750104834077e-17, -4.0501962532597664e-17, -4.318125372162919e-17,
-4.3849139175554844e-17, -4.6004154690447276e-17, -4.8885964191348045e-17,
-4.946521968011863e-17, -5.0002431772889196e-17, -5.146258674314064e-17,
-5.800655455856678e-17, -5.925918849422934e-17, -6.154766862396167e-17,
-6.40759874005501e-17, -6.5617533442163e-17, -6.94373540330799e-17,
-7.191715408032303e-17, -7.285838599102591e-17, -7.700483730548221e-17,
-8.117098645054182e-17, -8.185599975814986e-17, -8.879302744879469e-17,
-9.083873672066412e-17, -9.446608991054129e-17, -9.852745480651999e-17,
-9.887923813067803e-17, -1.1231135809944714e-16, -1.1382903973526863e-16,
-1.162264728904461e-16, -1.2143064331837652e-16, -1.290582547627127e-16,
-1.2918140103891835e-16, -1.3250093073348863e-16, -1.446014962661383e-16,
-1.4969791370238877e-16, -1.5439038936193586e-16, -2.0566414218240156e-16,
-4.505481786806643e-16) (1, 1, 1, 2, 1, 3, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
```




```
CONNECTED COMPONENTS
====================
The graph has 23 connected components
Connected component 0 has 16 nodes
Connected component 1 has 14 nodes
Connected component 2 has 5 nodes
Connected component 3 has 15 nodes
Connected component 4 has 5 nodes
Connected component 5 has 12 nodes
Connected component 6 has 11 nodes
Connected component 7 has 4 nodes
Connected component 8 has 5 nodes
Connected component 9 has 8 nodes
Connected component 10 has 4 nodes
Connected component 11 has 8 nodes
Connected component 12 has 7 nodes
Connected component 13 has 5 nodes
Connected component 14 has 11 nodes
Connected component 15 has 8 nodes
Connected component 16 has 5 nodes
```

```
Connected component 17 has 6 nodes
Connected component 18 has 5 nodes
Connected component 19 has 5 nodes
Connected component 20 has 4 nodes
Connected component 21 has 5 nodes
Connected component 22 has 6 nodes

CLUSTERS
========
The graph has 174 clusters
Cluster 0 has 39 nodes
Cluster 1 has 12 nodes
Cluster 2 has 39 nodes
Cluster 3 has 40 nodes
Cluster 4 has 12 nodes
Cluster 5 has 40 nodes
Cluster 6 has 12 nodes
Cluster 7 has 38 nodes
Cluster 8 has 39 nodes
Cluster 9 has 40 nodes
Cluster 10 has 12 nodes
Cluster 11 has 38 nodes
Cluster 12 has 25 nodes
Cluster 13 has 38 nodes
Cluster 14 has 39 nodes
Cluster 15 has 40 nodes
Cluster 16 has 40 nodes
Cluster 17 has 39 nodes
Cluster 18 has 25 nodes
Cluster 19 has 25 nodes
Cluster 20 has 38 nodes
Cluster 21 has 12 nodes
Cluster 22 has 40 nodes
Cluster 23 has 40 nodes
Cluster 24 has 40 nodes
Cluster 25 has 25 nodes
Cluster 26 has 39 nodes
Cluster 27 has 39 nodes
Cluster 28 has 12 nodes
Cluster 29 has 37 nodes
Cluster 30 has 40 nodes
Cluster 31 has 25 nodes
Cluster 32 has 39 nodes
Cluster 33 has 25 nodes
Cluster 34 has 25 nodes
Cluster 35 has 38 nodes
Cluster 36 has 12 nodes
Cluster 37 has 12 nodes
```

```
Cluster 38 has 38 nodes
Cluster 39 has 40 nodes
Cluster 40 has 40 nodes
Cluster 41 has 39 nodes
Cluster 42 has 40 nodes
Cluster 43 has 39 nodes
Cluster 44 has 12 nodes
Cluster 45 has 38 nodes
Cluster 46 has 25 nodes
Cluster 47 has 25 nodes
Cluster 48 has 25 nodes
Cluster 49 has 25 nodes
Cluster 50 has 12 nodes
Cluster 51 has 25 nodes
Cluster 52 has 40 nodes
Cluster 53 has 25 nodes
Cluster 54 has 25 nodes
Cluster 55 has 25 nodes
Cluster 56 has 38 nodes
Cluster 57 has 40 nodes
Cluster 58 has 39 nodes
Cluster 59 has 12 nodes
Cluster 60 has 39 nodes
Cluster 61 has 25 nodes
Cluster 62 has 39 nodes
Cluster 63 has 12 nodes
Cluster 64 has 39 nodes
Cluster 65 has 25 nodes
Cluster 66 has 40 nodes
Cluster 67 has 38 nodes
Cluster 68 has 12 nodes
Cluster 69 has 25 nodes
Cluster 70 has 39 nodes
Cluster 71 has 39 nodes
Cluster 72 has 40 nodes
Cluster 73 has 40 nodes
Cluster 74 has 25 nodes
Cluster 75 has 25 nodes
Cluster 76 has 40 nodes
Cluster 77 has 12 nodes
Cluster 78 has 25 nodes
Cluster 79 has 25 nodes
Cluster 80 has 39 nodes
Cluster 81 has 39 nodes
Cluster 82 has 25 nodes
Cluster 83 has 40 nodes
Cluster 84 has 25 nodes
Cluster 85 has 39 nodes
```

```
Cluster 86 has 25 nodes
Cluster 87 has 25 nodes
Cluster 88 has 39 nodes
Cluster 89 has 25 nodes
Cluster 90 has 25 nodes
Cluster 91 has 25 nodes
Cluster 92 has 39 nodes
Cluster 93 has 40 nodes
Cluster 94 has 25 nodes
Cluster 95 has 39 nodes
Cluster 96 has 39 nodes
Cluster 97 has 12 nodes
Cluster 98 has 25 nodes
Cluster 99 has 25 nodes
Cluster 100 has 25 nodes
Cluster 101 has 39 nodes
Cluster 102 has 39 nodes
Cluster 103 has 12 nodes
Cluster 104 has 25 nodes
Cluster 105 has 25 nodes
Cluster 106 has 25 nodes
Cluster 107 has 25 nodes
Cluster 108 has 38 nodes
Cluster 109 has 40 nodes
Cluster 110 has 40 nodes
Cluster 111 has 25 nodes
Cluster 112 has 39 nodes
Cluster 113 has 40 nodes
Cluster 114 has 40 nodes
Cluster 115 has 25 nodes
Cluster 116 has 12 nodes
Cluster 117 has 25 nodes
Cluster 118 has 39 nodes
Cluster 119 has 39 nodes
Cluster 120 has 40 nodes
Cluster 121 has 39 nodes
Cluster 122 has 25 nodes
Cluster 123 has 12 nodes
Cluster 124 has 40 nodes
Cluster 125 has 25 nodes
Cluster 126 has 40 nodes
Cluster 127 has 39 nodes
Cluster 128 has 40 nodes
Cluster 129 has 40 nodes
Cluster 130 has 39 nodes
Cluster 131 has 25 nodes
Cluster 132 has 25 nodes
Cluster 133 has 12 nodes
```

```
Cluster 134 has 38 nodes
Cluster 135 has 25 nodes
Cluster 136 has 39 nodes
Cluster 137 has 25 nodes
Cluster 138 has 12 nodes
Cluster 139 has 25 nodes
Cluster 140 has 25 nodes
Cluster 141 has 25 nodes
Cluster 142 has 25 nodes
Cluster 143 has 25 nodes
Cluster 144 has 12 nodes
Cluster 145 has 25 nodes
Cluster 146 has 25 nodes
Cluster 147 has 40 nodes
Cluster 148 has 39 nodes
Cluster 149 has 25 nodes
Cluster 150 has 12 nodes
Cluster 151 has 12 nodes
Cluster 152 has 25 nodes
Cluster 153 has 25 nodes
Cluster 154 has 25 nodes
Cluster 155 has 25 nodes
Cluster 156 has 40 nodes
Cluster 157 has 25 nodes
Cluster 158 has 39 nodes
Cluster 159 has 25 nodes
Cluster 160 has 39 nodes
Cluster 161 has 39 nodes
Cluster 162 has 25 nodes
Cluster 163 has 12 nodes
Cluster 164 has 25 nodes
Cluster 165 has 40 nodes
Cluster 166 has 25 nodes
Cluster 167 has 38 nodes
Cluster 168 has 12 nodes
Cluster 169 has 25 nodes
Cluster 170 has 12 nodes
Cluster 171 has 40 nodes
Cluster 172 has 40 nodes
Cluster 173 has 39 nodes

Visualizing the graph:
```