The GitHub Network

Limingrui Wan, Danyu Zhang & Daniel Alonso

May 29th, 2021

Part 1

Description of the network

Our dataset corresponds to the GitHub.com network of developers. This data was collected from the public API in June 2019. Each node is a developer with at least 10 repositories starred and each edge is a mutual follower relationship between them. The vertex features are extracted based on the location, repositories starred, employer and e-mail address.

Source

The dataset was optained from the Stanford University SNAP website.. This dataset originally come from a paper published the 28th of september, 2019 called Multi-scale Attributed Node Embedding. The github repository for that project can be found here.

Characteristics of the network

Vertices/Nodes

Our network has the following amount of vertices/nodes:

```
#> [1] 37700
```

Degrees of vertices

Top vertex degrees

The vertices with the largest degrees (top 5) are:

```
#>      dalinhuang99             nfultz        addyosmani            Bunlong
#>              9458               7085               3324               2958
#> gabrielpconceicao
#>              2468
```

Who are they?

As these are individuals, we could peak inside GitHub.com and check the public profile of these individuals.

- Top user: **dalinhuang99**

The reason why this user might have so many followers could be the fact that he's followed a very large amount of users (160k as of May 4th 2021).

The user also seems to be a top 4% Stack overflow participant. However, the user has had no activity since July 17th 2018.

- Second top user: **nfultz**

This next user seems to have several useful tutorials/content in his github pages hosted static site. Also seems to have several active repositories where the user does some collaborative work.

There is no immediate apparent reason as to why the user has garnered such a large following.

- Third top user: **addyosmani**

This user has actually garnered a larger following than shown on the dataset. The user is an engineer at Google, working specifically on Google Chrome and the user has a significantly larger following on other social media (259.5k on twitter).

- Fourth top user: **Bunlong**

The user seems to be very active on github, committing code basically every day. The user also follows about 24.4k other users, and has created a couple projects that seem to be somewhat public as well.

- Fifth top user: **gabrielpconceicao**

The user also seems to be following a very large amount of users (31.8k following), not as active as the rest in the list, so the amount of followers could've come from following a large amount as well.

Bottom vertex degrees

The following amount of users have a degree of exactly 1 (one follower):

```
#> [1] 5045
```

Sum of vertex degrees

Calculated as follows:

$$\sum_{v=1}^{N} d_v = 2L$$

Corresponds to twice the size of the graph:

```
#> [1] 578006
```

Average degree

Calculated as follows:

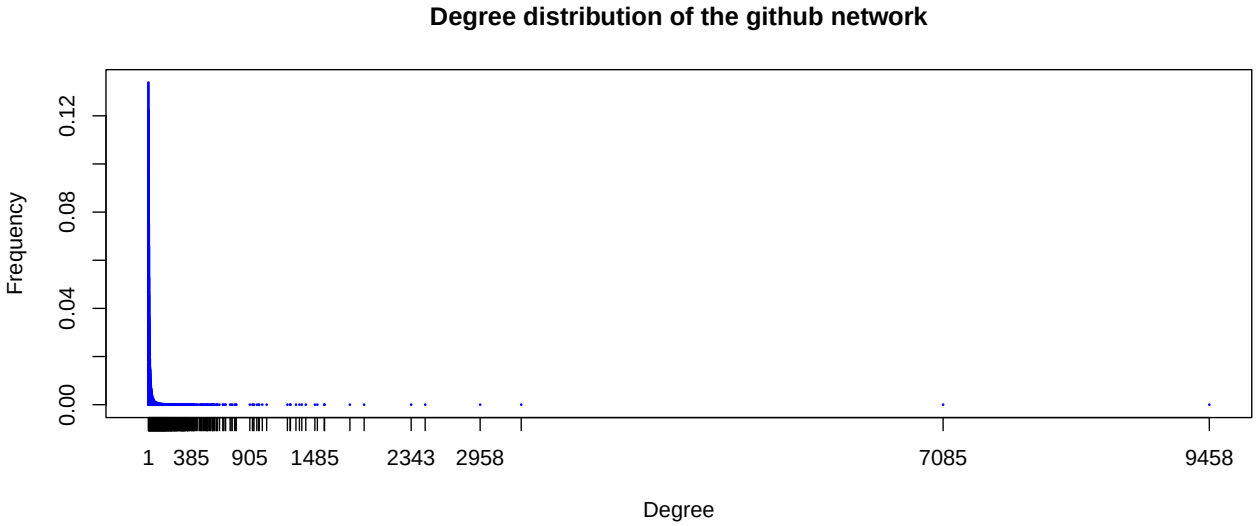
$$\frac{1}{N} * \sum_{v=1}^{N} d_v = 2\frac{L}{N}$$

Represents the average amount of mutual followers among all users in the network (nodes):

```
#> [1] 15.33172
```

Degree distribution

We can see the degree distribution of our graph as follows:



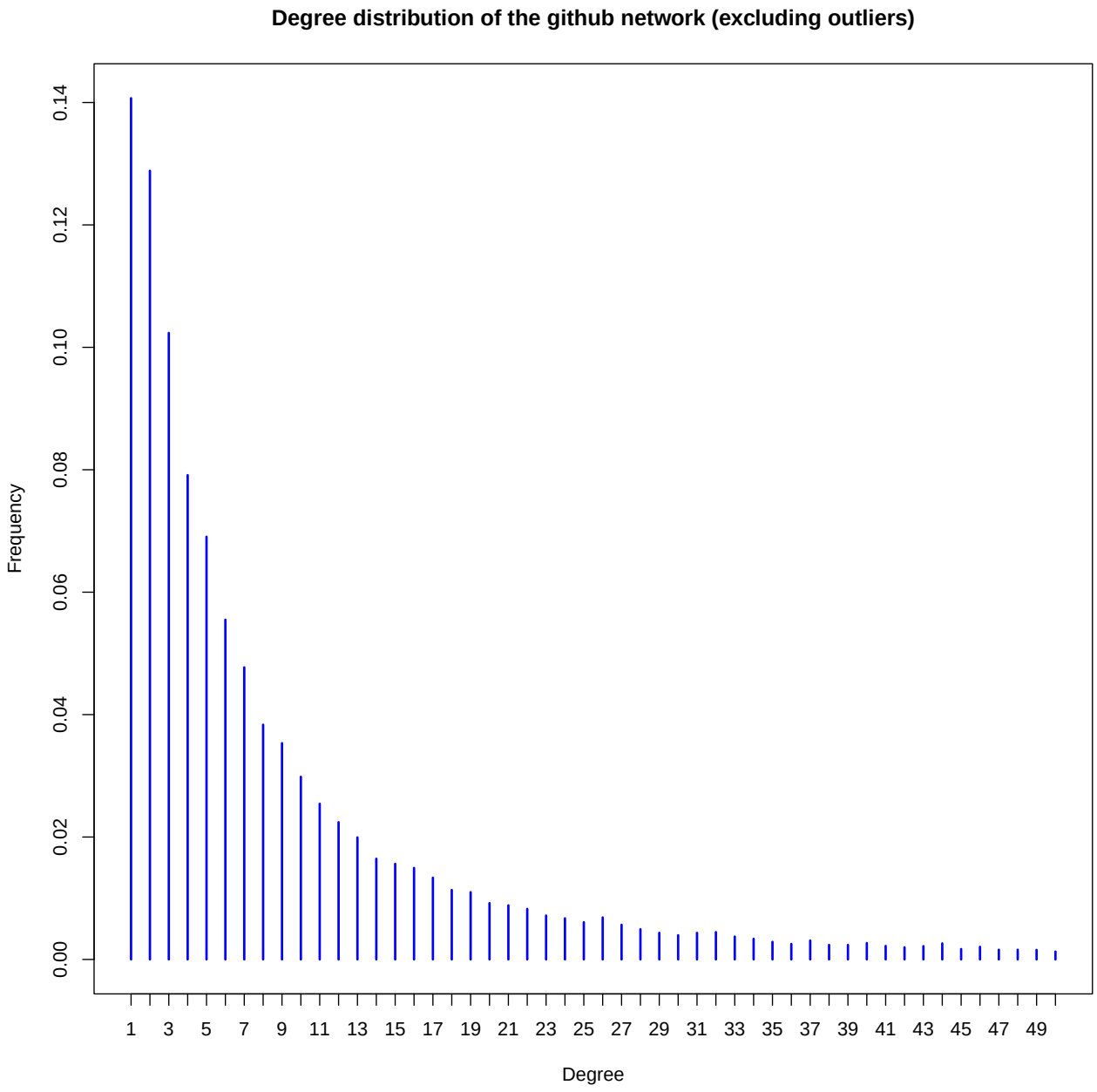
We can notice that our degree distribution is extremely right-skewed.

The reason for this we can clearly see by looking at a table of our degree frequencies (top 10):

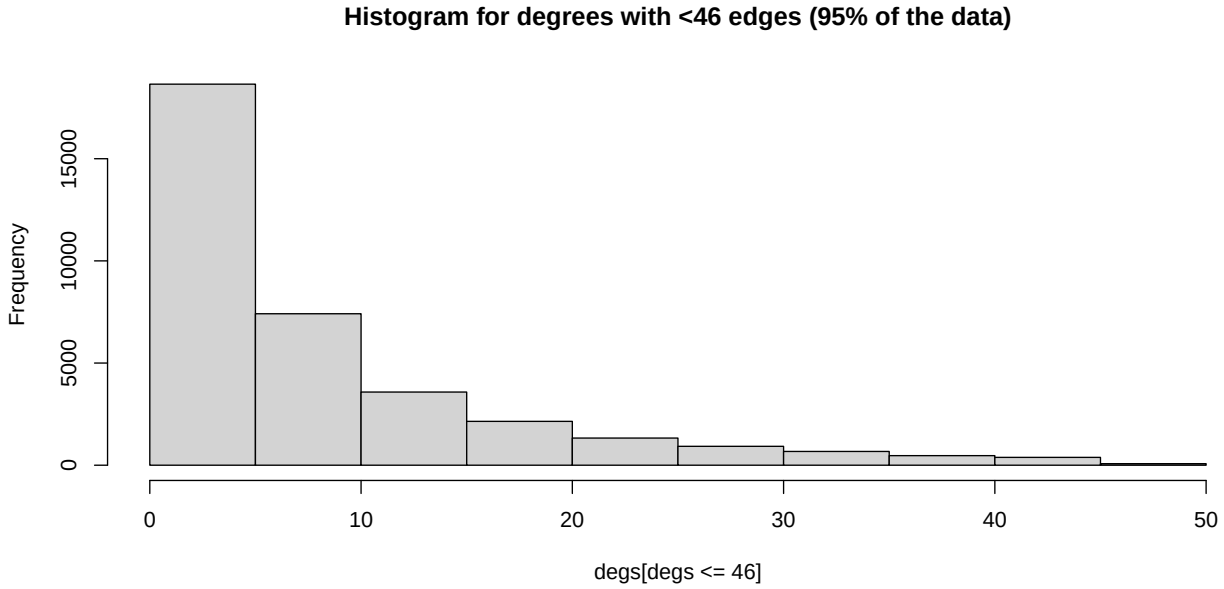
```
#>
#>    1    2    3    4    5    6    7    8    9   10
#> 5045 4620 3670 2837 2476 1990 1711 1375 1267 1070
```

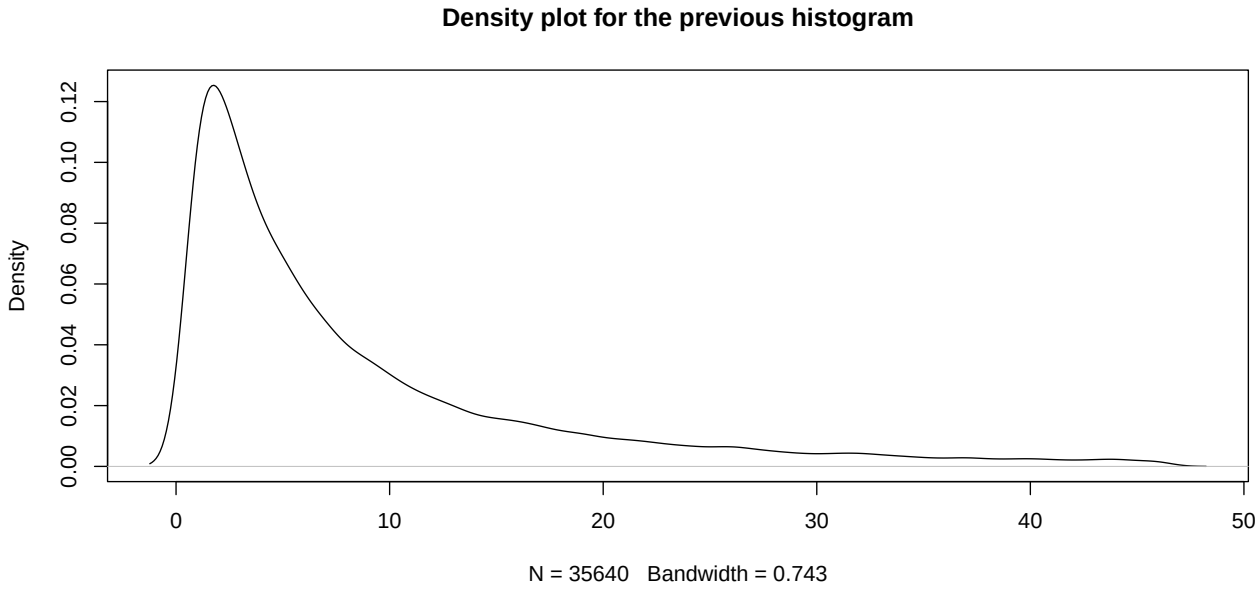
We can see the rightmost tail of our plot (top 50 frequencies), excluding the biggest outliers.

Here we can see a bit over 95% of the data points in our graph:



We can also create a histogram and density of our degree distributions, but for practicality and visual purposes, we avoid plotting any frequency above 46 edges.





Edges/Links

Our network has the following amount of edges (the size of the graph):

```
#> [1] 289003
```

The edges of the GitHub network are not weighted, because the edges represent a mutual follower relationship on GitHub, which would essentially always have a weight of 1.

Additionally, our network does not have any loop, therefore it's not a multigraph.

Connectedness

We can notice that our graph is connected, therefore every vertex is reachable from every other:

```
#> [1] TRUE
```

Diameter

Diameter is the shortest distance between the two most distant nodes in the network and with the *diameter* function we can obtain the value of the longest geodesic distance in the network.

This diameter corresponds to the following set of 12 nodes:

haochenli –> **ChiuMungZitAlexander** –> **IrvingZha0** –> **harryworld** –> **NigelEarle** –> **getify** –> **indrajithbandara** –> **kaizenagility** –> **artificialsoph** –> **kemacdonald** –> **jpriniski** –> **SOUMAJYOTI**

With a diameter of 11 (longest geodesic distance).

Farthest vertices

The farthest vertices corresponds to the previously mentioned first and last node, which are:

- Start: **haochenli**
- End: **SOUMAJYOTI**

Adjacency Matrix

For size purposes, it makes no sense to show the adjancency matrix of this network, as it would turn out to be a matrix of size 37,700 by 37,700.

Graphical representations

We have learned several methods to generate graphical representations of graphs, but none of them can be used directly for a big network. Unfortunately, our graph has 37.7k nodes and 298k edges, which is a rather large graph.

So we can only make a subgraph of the first 400 nodes and those edges with these nodes. If we choose more nodes, the graph will be too crowded, if we choose less nodes, the graph will possibly contains too few edges.

But we can hardly find useful information from these plots.

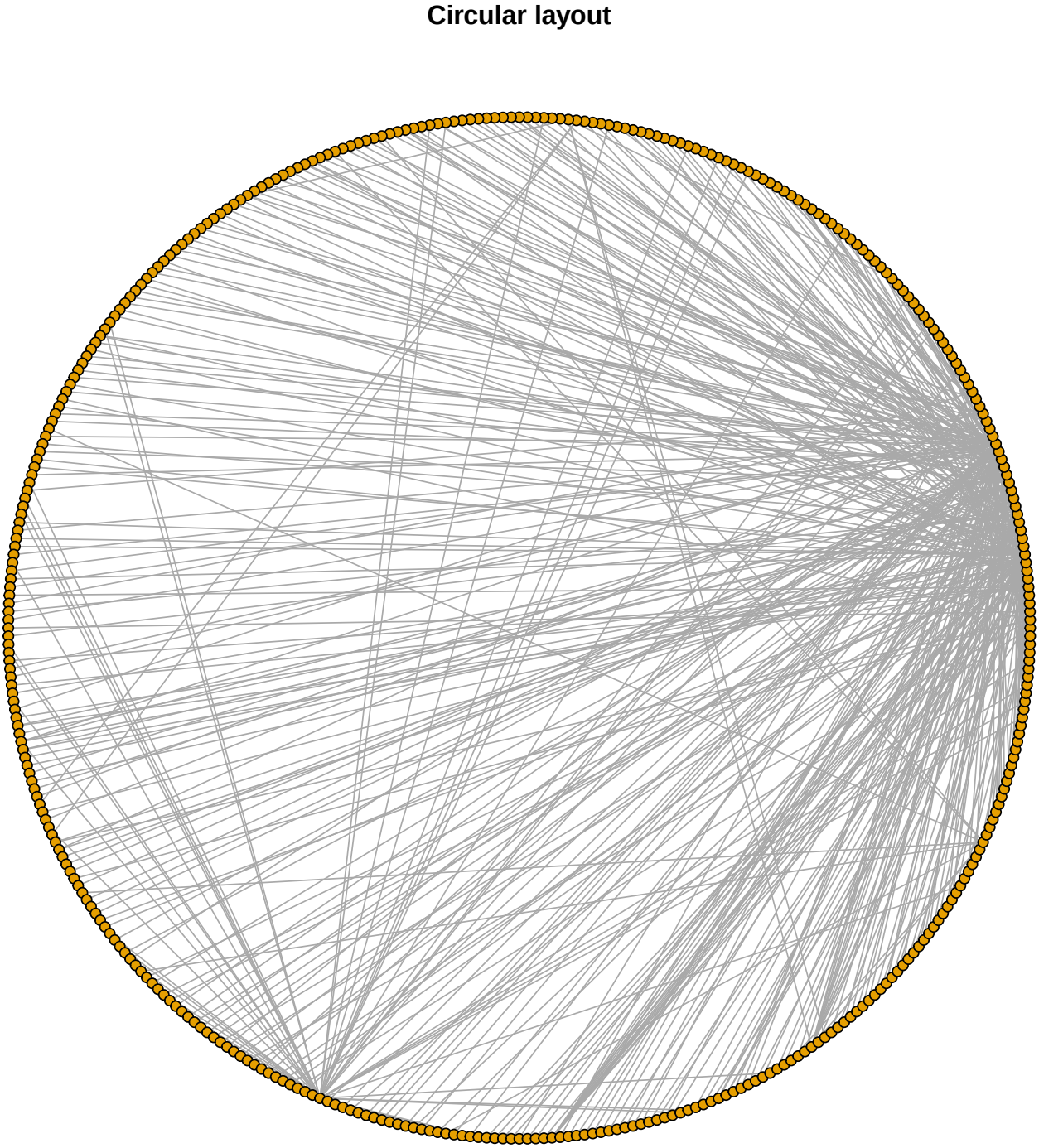
So the analysis is only attached to the subgraph itself but not the original big network.

We have taken the liberty to plot a slightly more complex graph than the one shown in the presentation, as we had more time for processing.

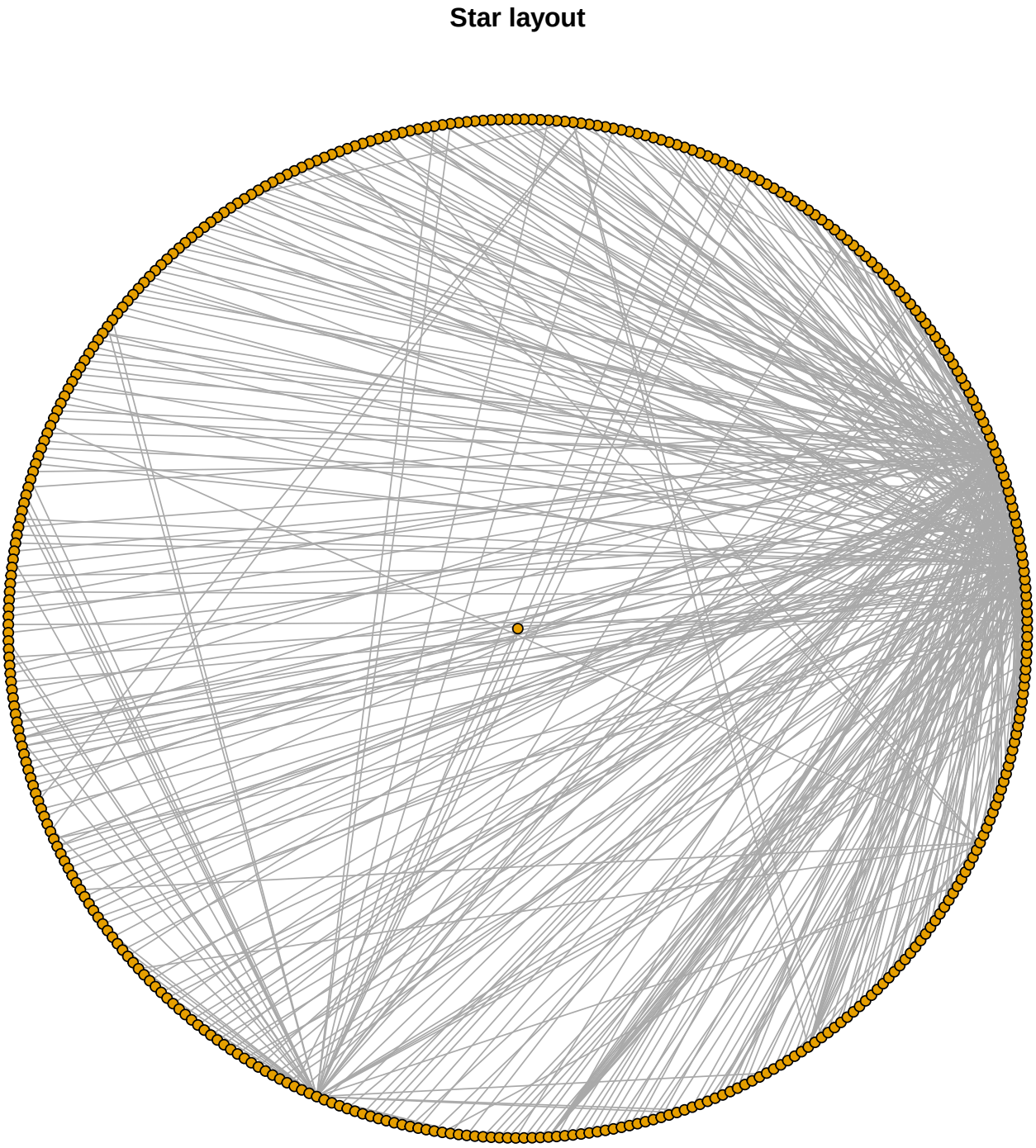
The graph's connectedness is unfortunately not perfectly showcased for this subgraph.

Circular layout

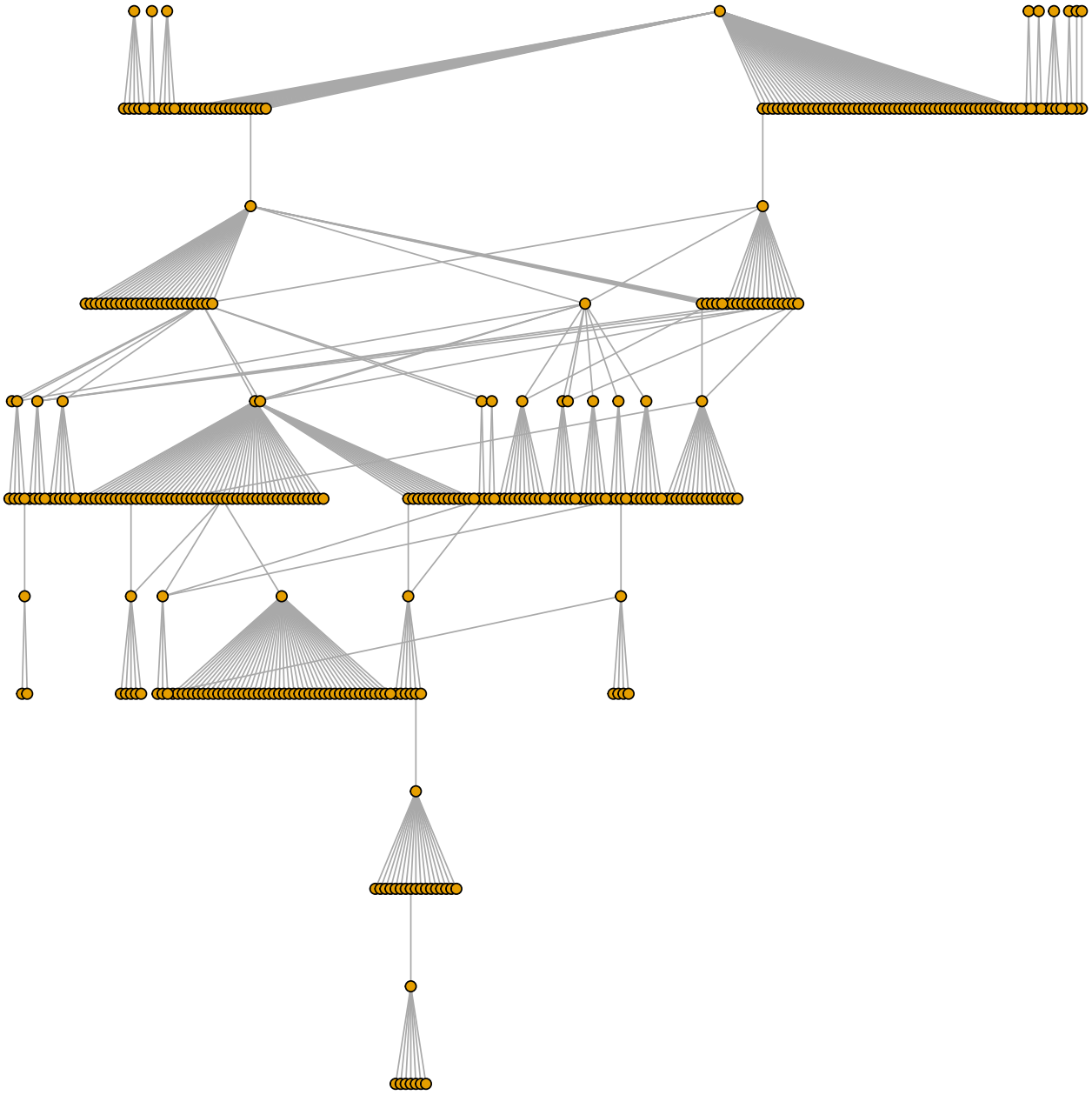
The circular and star layouts appear to confirm that the number of edges is not very large. This is a **sparse network**,relatively small compared with the number of vertices.



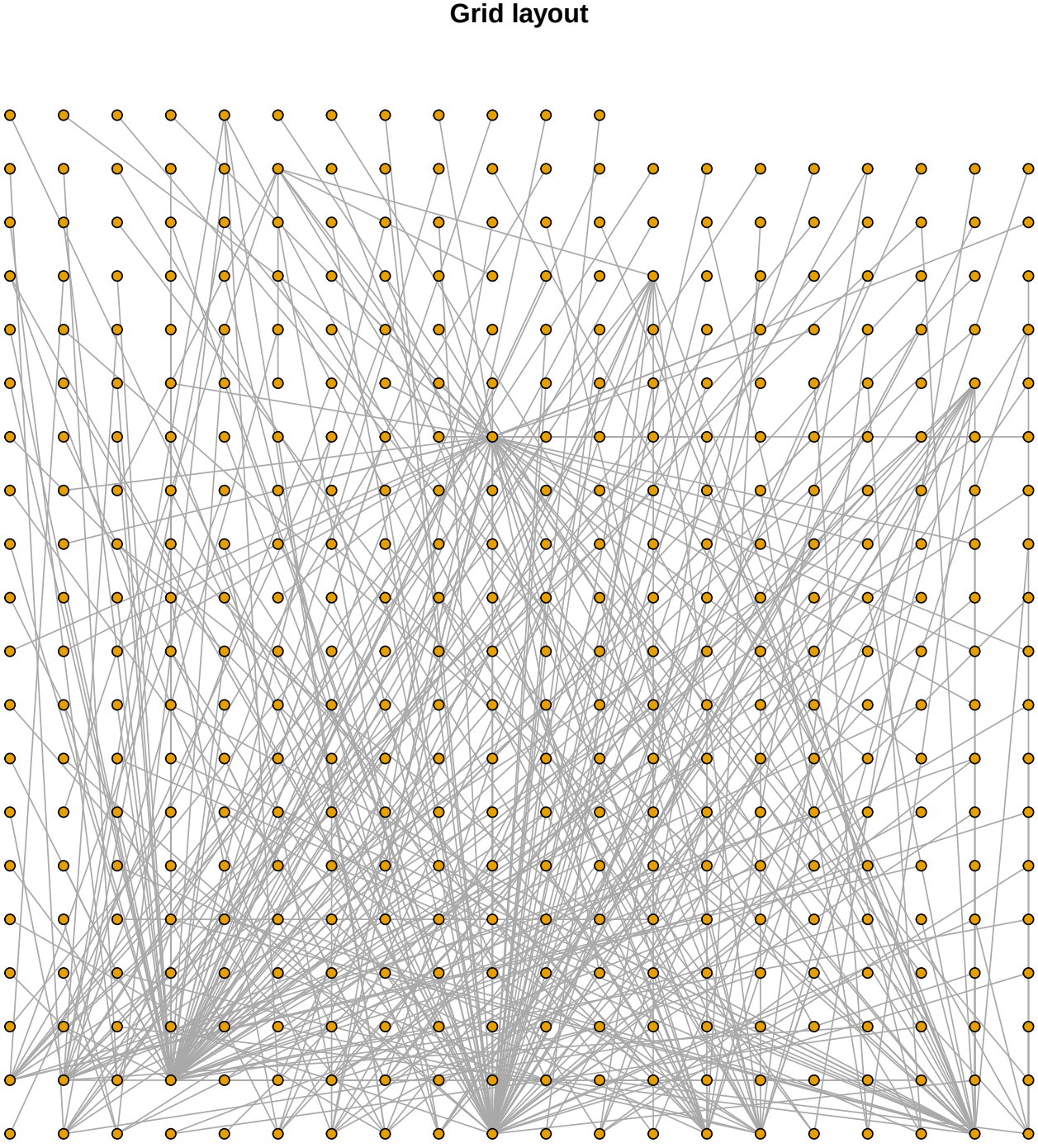
Star layout



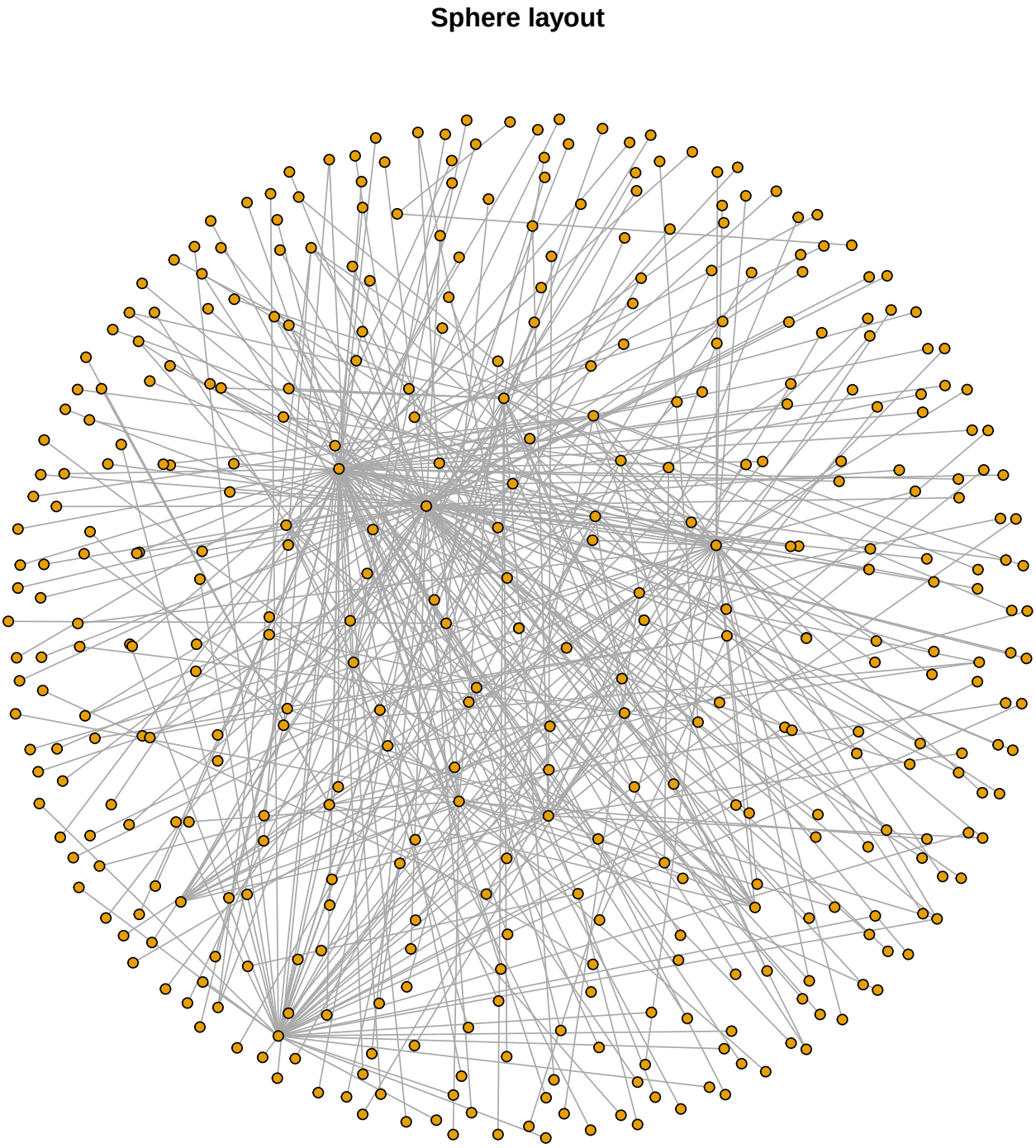
Tree layout



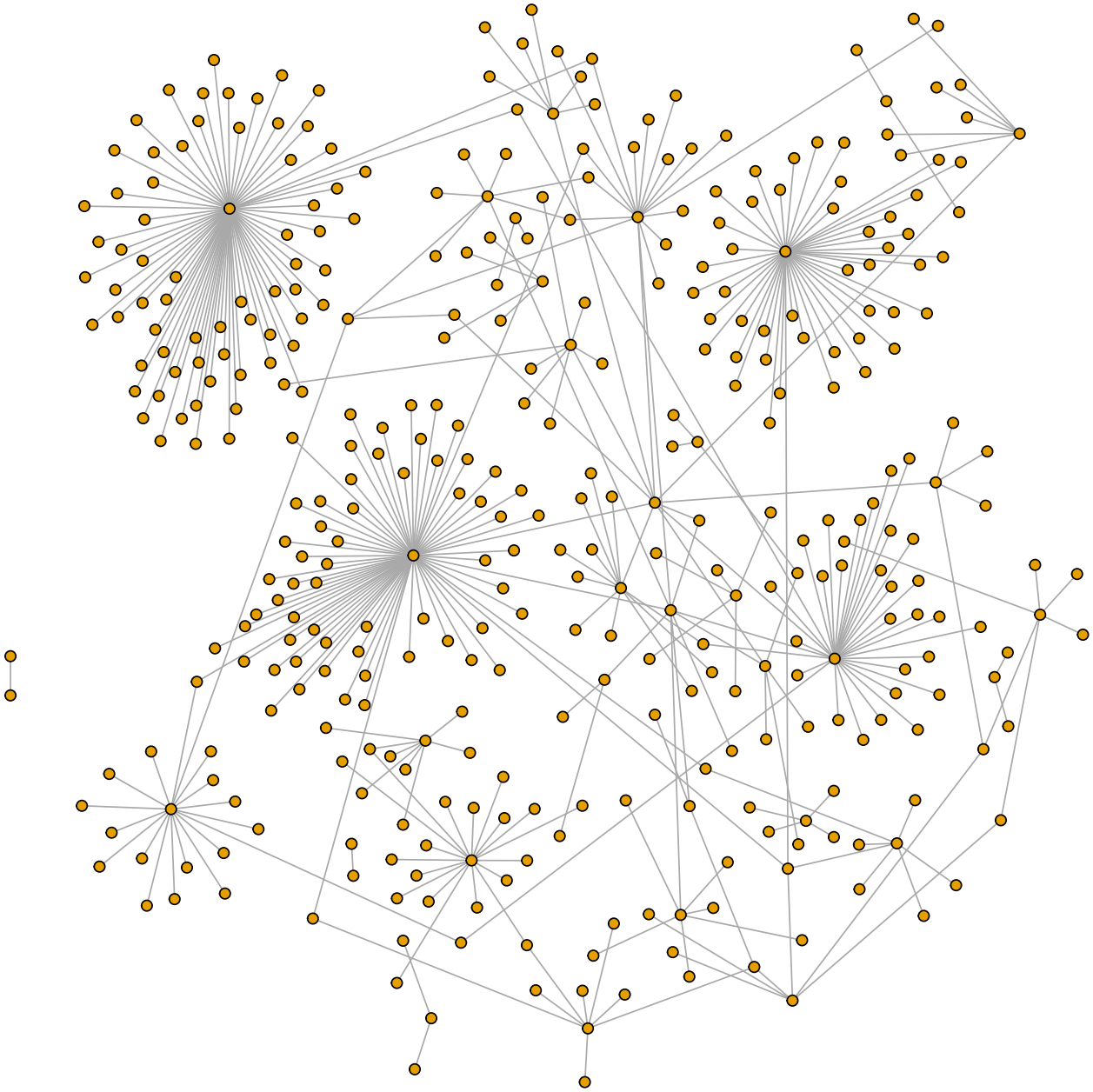
Grid layout



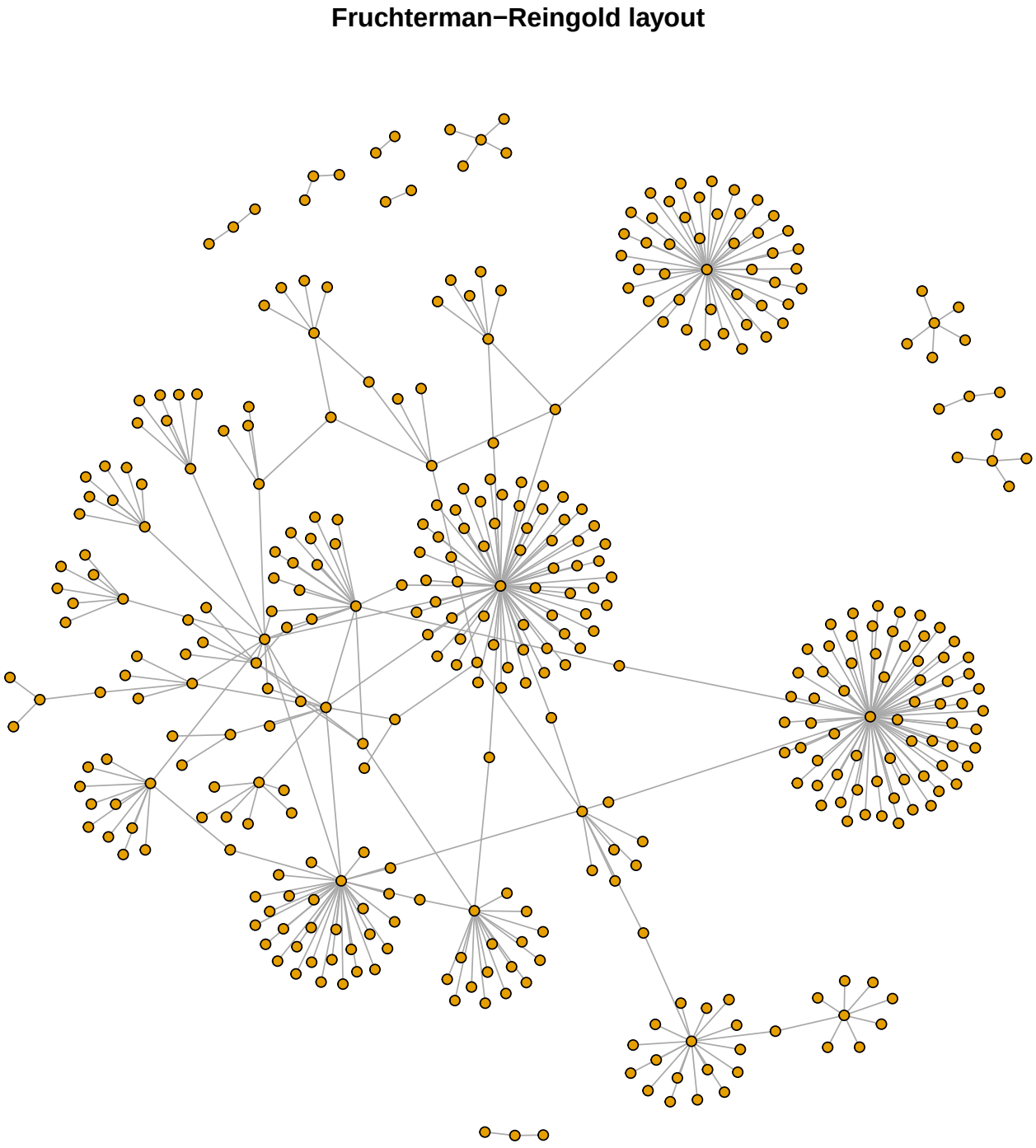
Sphere layout



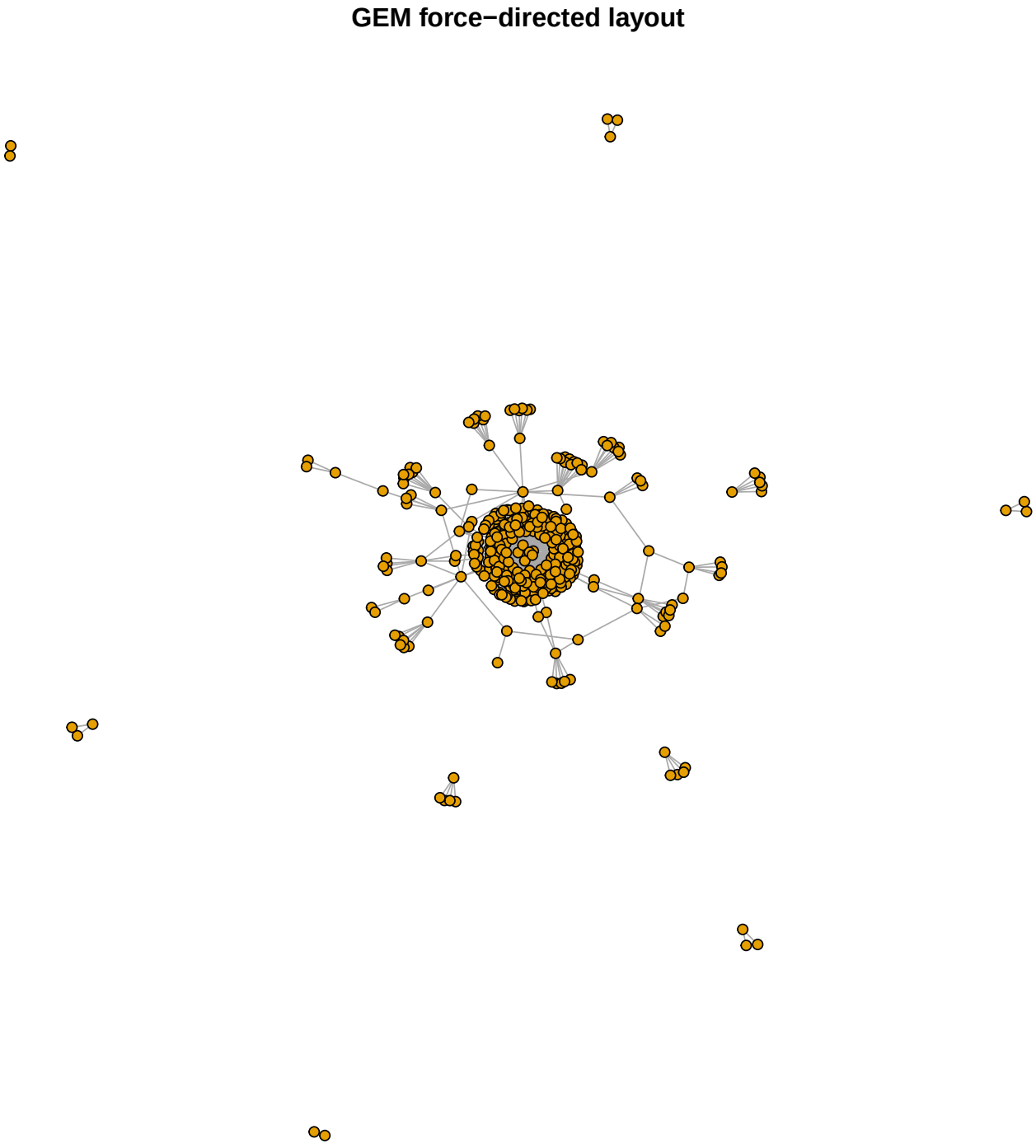
Davidson-Harel layout



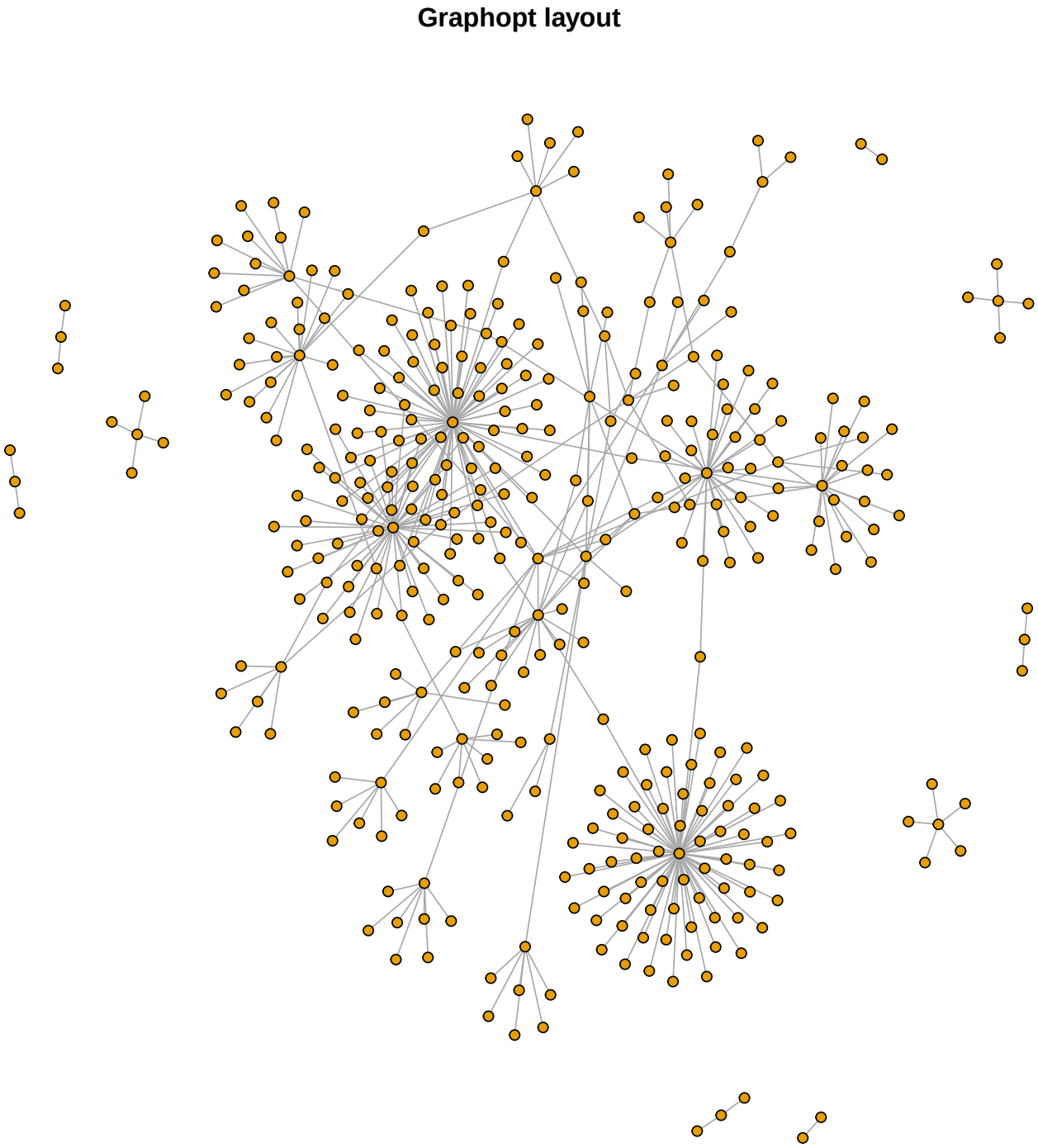
Fruchtermann-Reingold layout



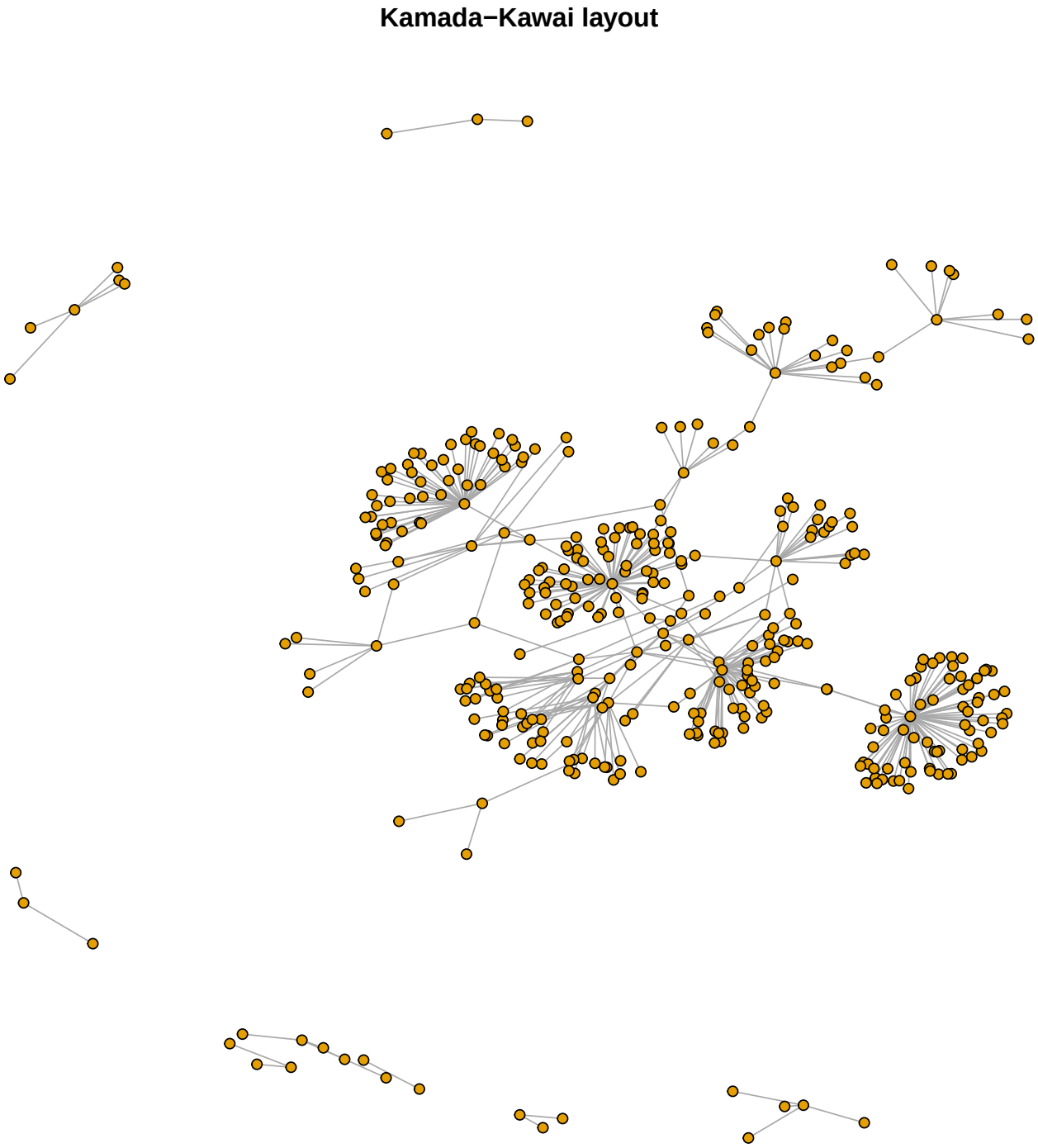
GEM force-directed layout



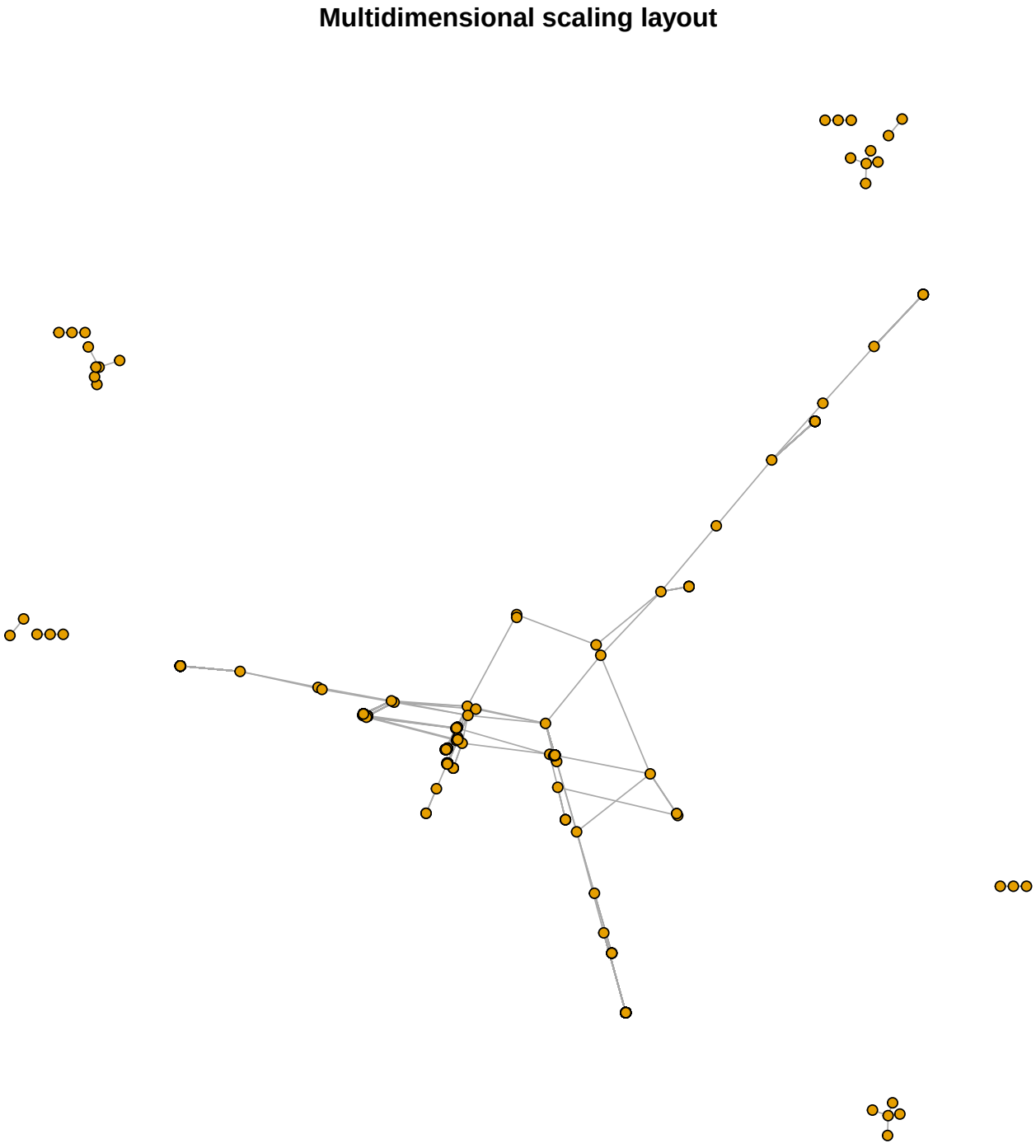
Graphopt layout



Kamada-Kawai layout

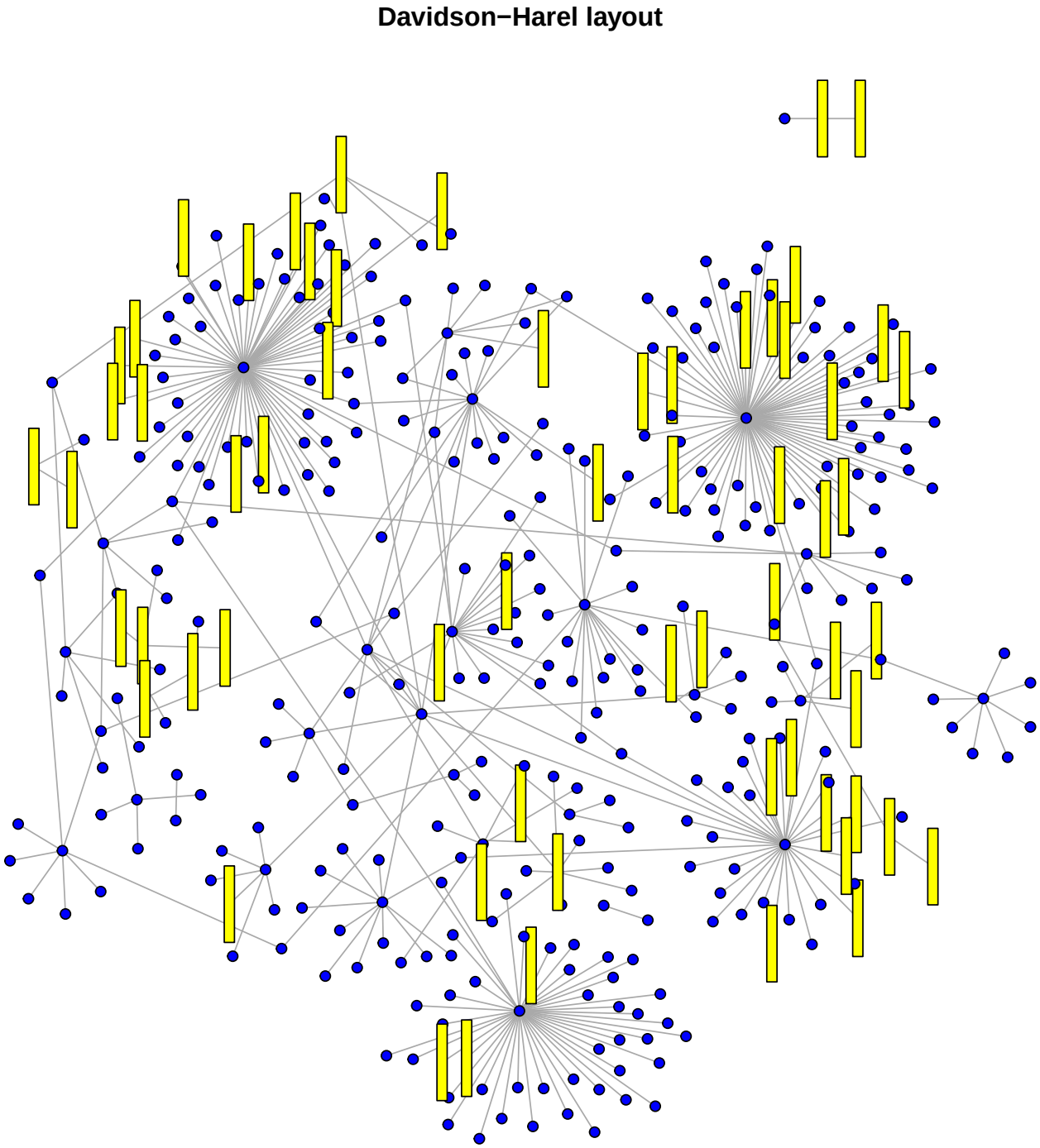


Multidimensional Scaling layout



Our choice to represent the graph

We choose to use Davidson-Harel layout, since there and only two type of nodes, the user is a web or machine learning developer, we wish to identify them with shapes and colors. And we adjust the size of nodes and edges to make the plot clearer.



Part 2

Section 1: Centrality measures

Centrality measures

The first thing we shall do in part 2 is obtain the centrality measures for our network.

Vertex degree

As we saw before, according to vertex degree, which is technically a very simple but still quite informative centrality measure. We see that the top vertices are as follows:

```
#>      dalinhuang99            nfultz        addyosmani            Bunlong
#>              9458              7085              3324              2958
#> gabrielpconceicao
#>              2468
```

Given that the network is not directed or weighted, the measures utilizing *strength* will be equal to those of using *degree*, therefore our top 5 remains the same from either perspective.

Closeness centrality

Given the size of our graph and the computational intensity of the calculation of certain centrality measures like closeness centrality, we will take a subgraph of our graph using *subgraph.edges*, which preserves a given amount of edges along with vertices.

```
#>       nfultz dalinhuang99     popomore     dexteryy  denysdovhan
#>  0.004035861  0.004035534  0.004031521  0.004031484  0.004031138
```

We can see the largest values for closeness centrality correspond to some of the top people we saw before with the highest vertex degree. Here we see that there might be variation given that we took a subset of the graph.

Computing the vertex degree again for our subset returns the following:

```
#>       chokcoco       popomore   dalinhuang99 tamimibrahim17       dexteryy
#>            372            239            215            209            155
```

Which shows that yes, there might be a correlation between what the measure of vertex degree considers more central and what closeness centrality considers more central. We notice that there are some repeated individuals, but not necessarily the same.

Betweenness centrality

As for the betweenness centrality, we will also use the same subset of the graph as we used for closeness centrality, and this yields the following resulting top 5 individuals:

```
#>   dalinhuang99         nfultz       chokcoco tamimibrahim17       popomore
#>     0.23834076     0.23590782     0.10052484     0.06385892     0.06277038
```

This shows that *dalinhuang99*, for example, hasn't just amassed a large amount of followers/following, but also represents a central position for movement among nodes within the graph. He also shares a similar position to *nfultz*, which given his own following (although smaller than *dalinhuang99*'s), still boasts an important position according to his betweenness centrality score. So much so, that the scores are nearly identical, even though the amount of connections is significantly higher for *dalinhuang99* (about 33% higher).

Eigenvector centrality

Obtaining the eigenvector centrality measures for the nodes we see something interesting:

```
#>     chokcoco dalinhuang99     popomore       nfultz     dexteryy
#>    0.4179368    0.2948025    0.2603379    0.2175484    0.1689700
```

We can see that eigenvector centrality clearly uses different considerations, where our most central users according to the previous two centralities somewhat coincided in the two top users in vertex degree, here we see others that, while present in previous' centralities top, are now at the top. *dalinhuang99* still retains the 2nd-highest position according to this measure, however, there's a significant difference with the top user *chokcoco*. Also trailing behind is *nfultz*, which was top 2 in vertex degree.

Pagerank centrality

As for pagerank centrality we obtain the following top:

```
#>       chokcoco       popomore tamimibrahim17   dalinhuang99       dexteryy
#>    0.019111077    0.010140520    0.010014577    0.008050142    0.006462591
```

Pagerank centrality brings a top that is also somewhat different, but coincides more strongly with eigenvector centrality than many of the others.

We see that both share the top top user, which is *chokcoco*, with nearly double the score of *dalinhuang99* (the top user by vertex degree). Also this type of centrality introduces *tamimibrahim17*, which does not even belong in the top 10 users by vertex degree. This could mean that his positioning within the graph could be reasonably better than most others in the top along with *popomore*, which also shows as a top user according to eigenvector centrality.

Section 2: Characterizing network cohesion

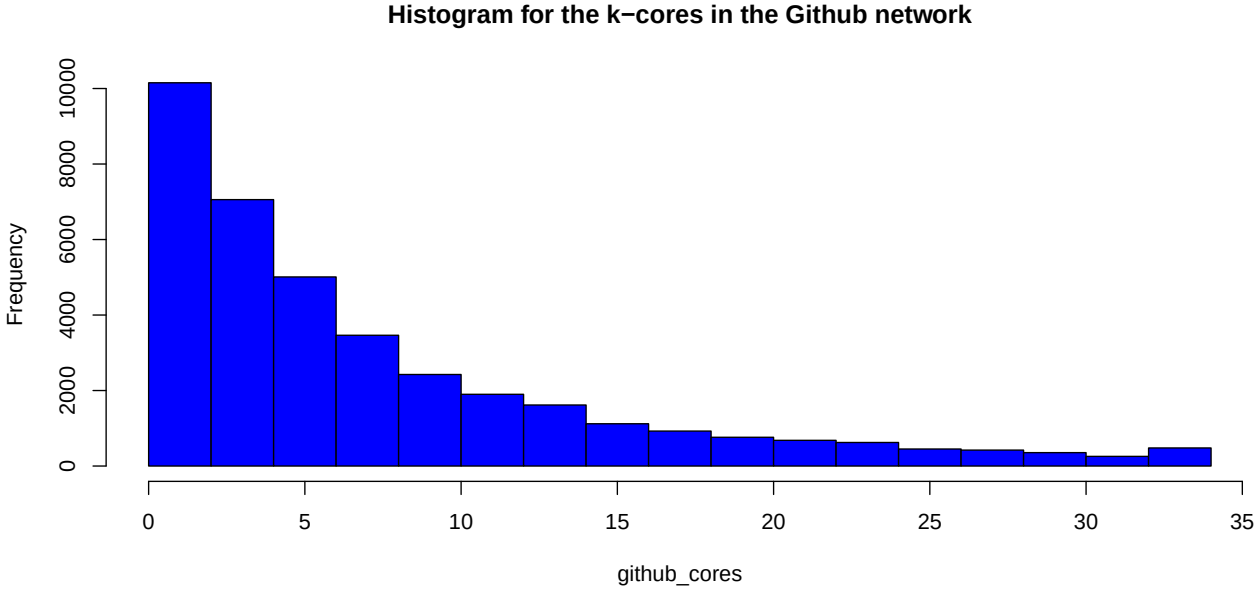
Local density

When dealing with such large graphs, we should compute coreness for local density instead of cliques.

Coreness

```
#>       Eiryyy   shawflying  JpMCarrilho    SuhwanCha sunilangadi2    j6montoya
#>            1            6            1            4            2            1
```

```
#> [1] 360
```



We can see that the max core value is 34, and basically higher the core value the fewer the nodes.

```
#> [1] 34
```

```
#> [1] 0.0004066878
```

We can see the density of this graph is very low, thus the nodes are considered to be scattered.

We choose two nodes with the most connected edges to compare their density levels:

```
#> [1] 0.001624422
```

```
#> [1] 0.003480618
```

User *dalinhuang99* has higher a degree value but a lower coreness value. it shows that more edges attached to him doesn't necessarily imply a high density level.

Connectivity

```
#> [1] 1
```

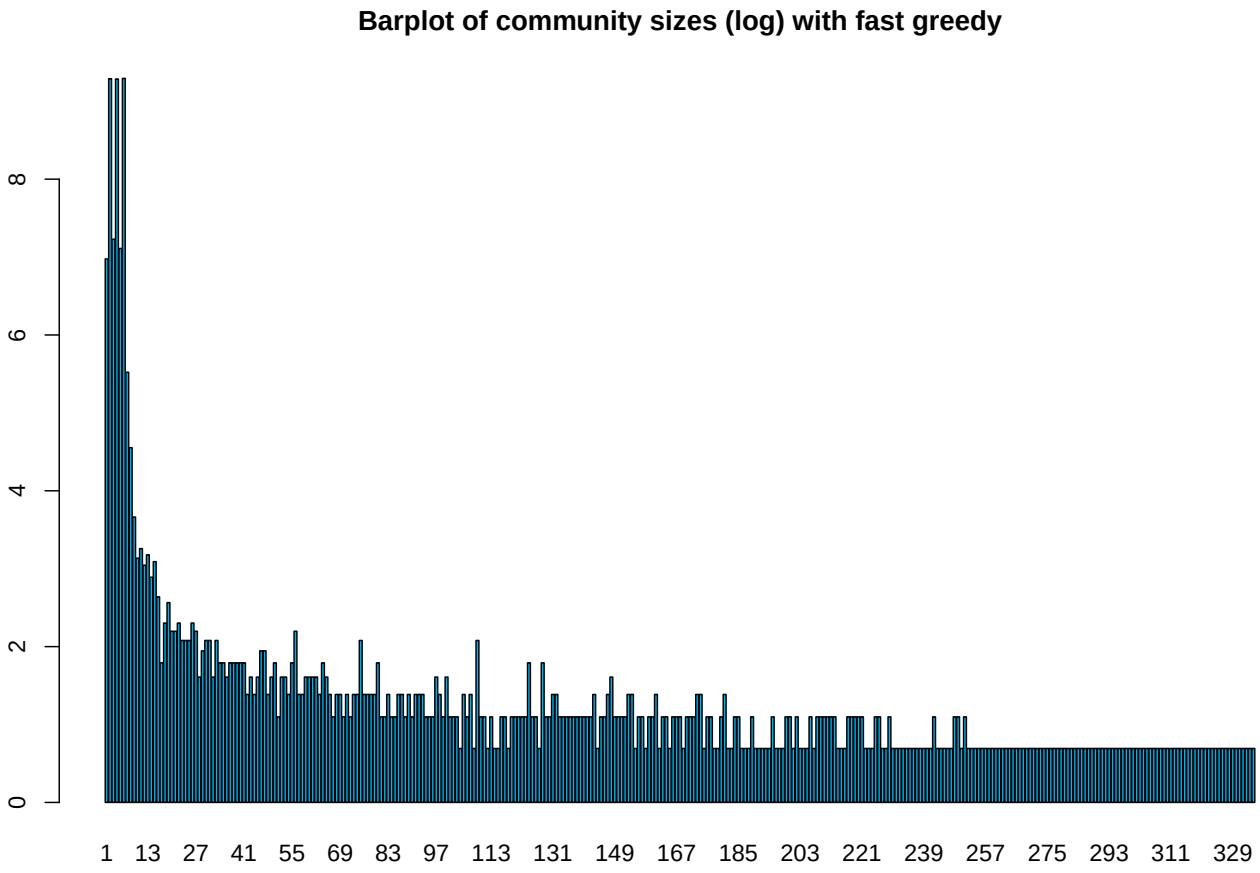
There are no isolated nodes in this graph, it's a connected graph.

Community detection

In this section we intend to split the graph into proper parts (communities), notice that not every method mentioned during the course is appropriate for a graph of this size.

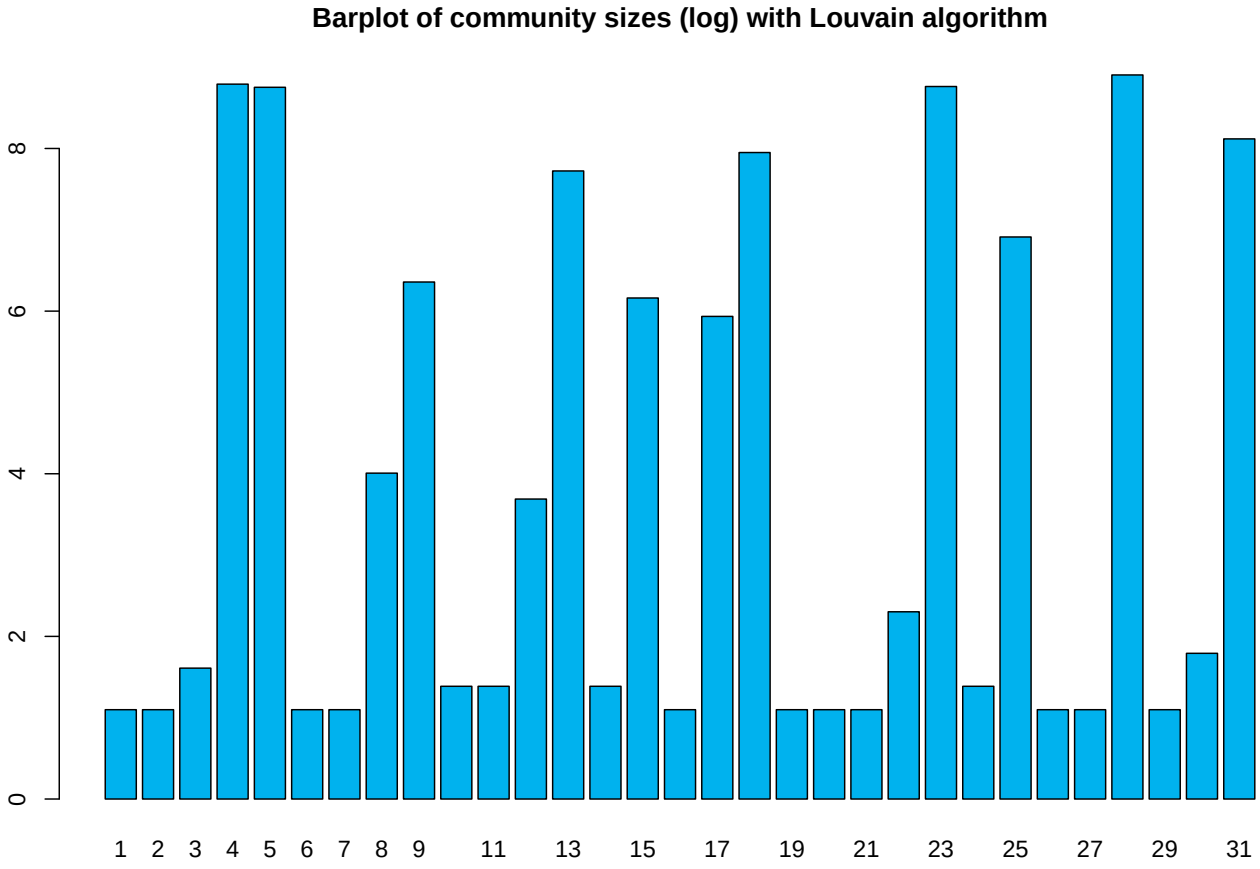
Fast greedy method

```
#> [1] "the number of clusters is: 335"
#> [1] "the modularity value is: 0.395688421680964"
```



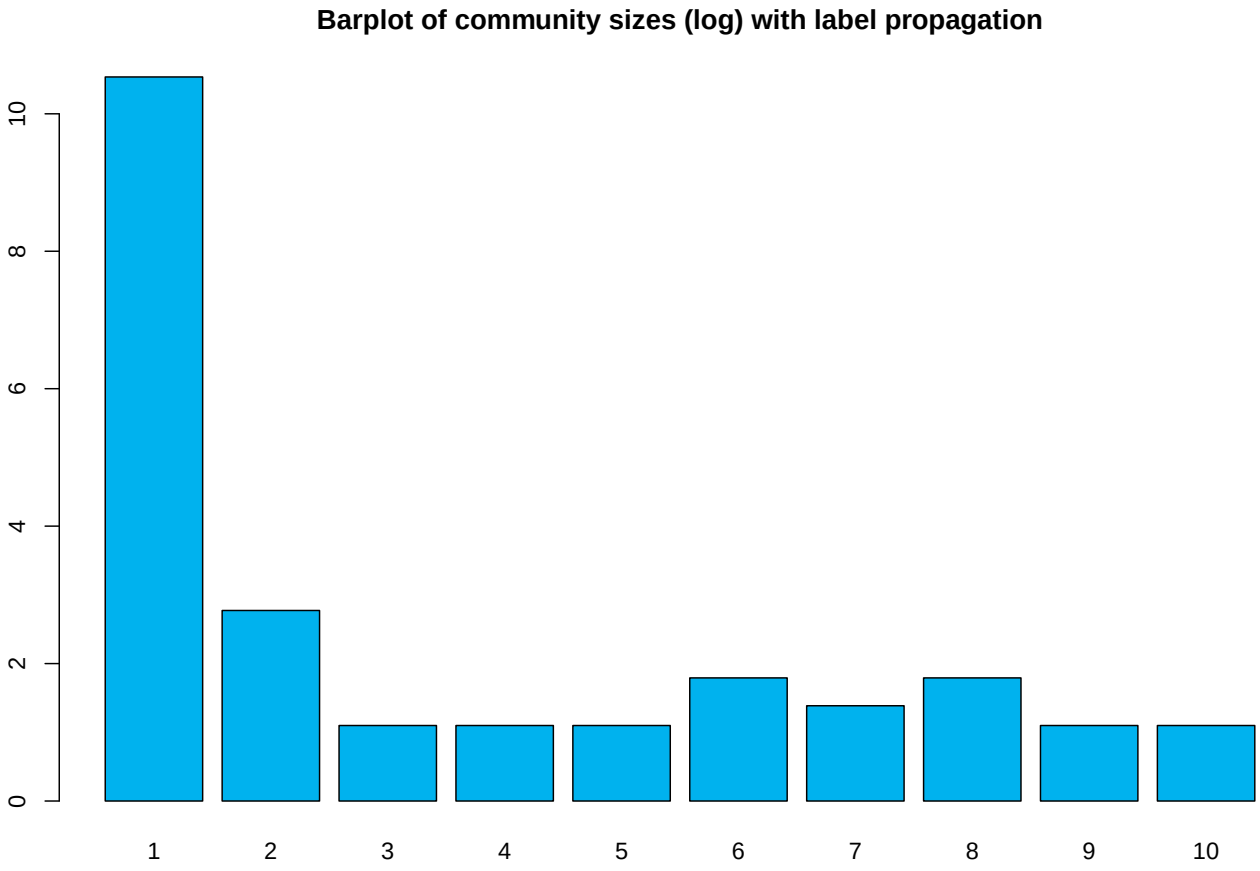
Louvain algorithm

```
#> [1] "the number of clusters is: 31"
#> [1] "the modularity value is: 0.455036075212824"
```



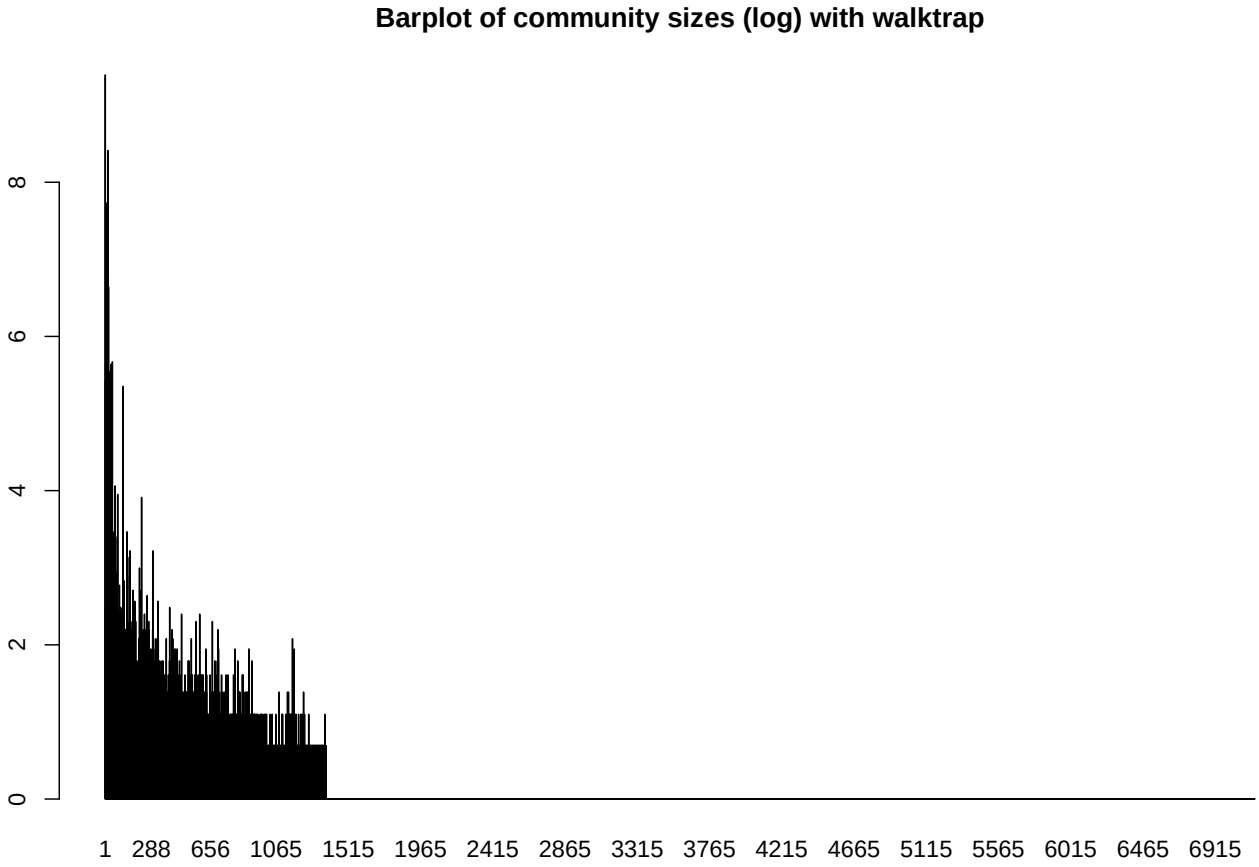
Label propagation method

```
#> [1] "the number of clusters is: 10"
#> [1] "the modularity value is: 0.00029753477654749"
```



Walktrap algorithm

```
#> [1] "the number of clusters is: 7161"
#> [1] "the modularity value is: 0.356498032808304"
```



Among these methods, we consider Louvain algorithm the best.

Fast greedy and walktrap methods provide too many clusters, and the modularity of the label propagation method is too close to 0.

Plotting the clustered network (using louvain clustering)

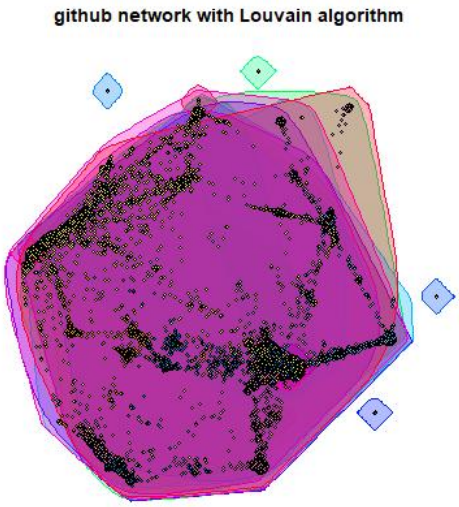
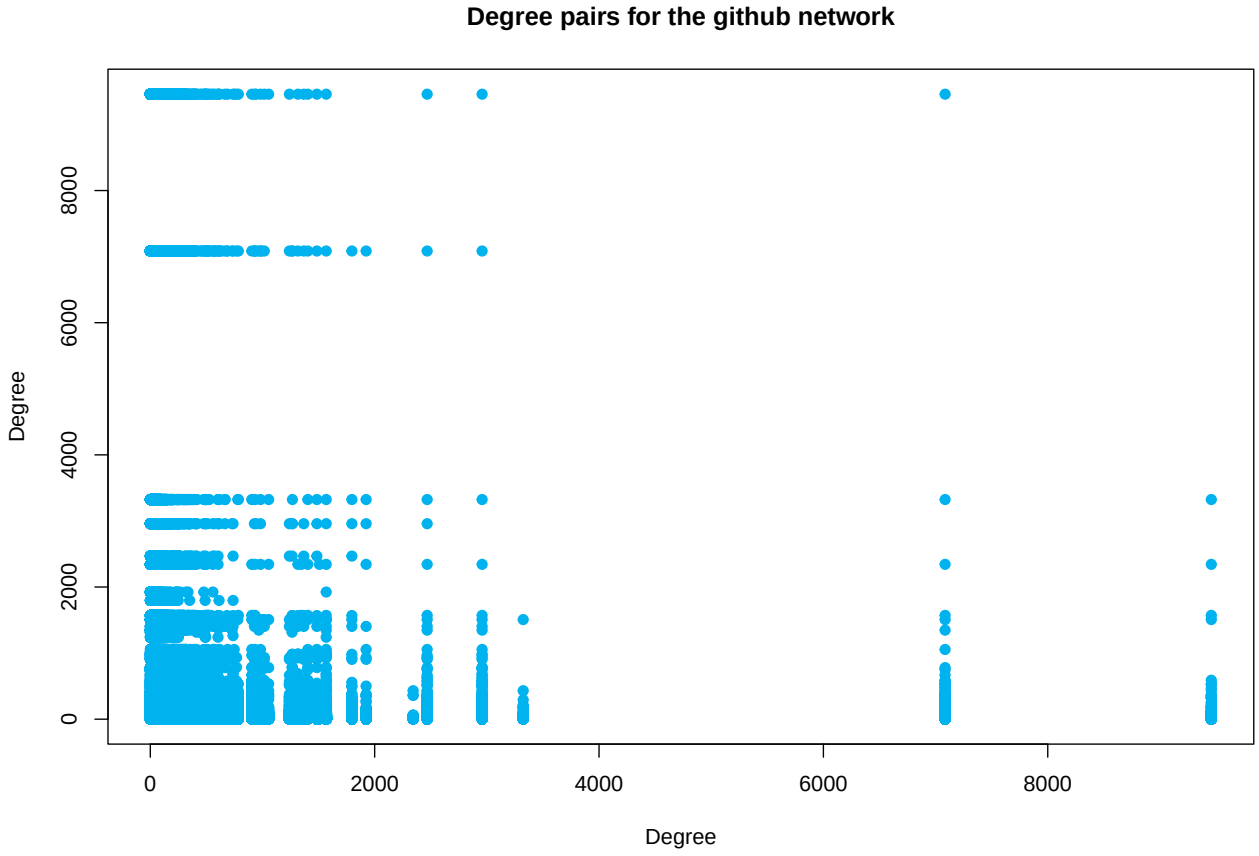


Figure 1: Clustering

Assortativity




```
#> [1] -0.07521713
```

The the assortativity coefficient is -0.075. Consequently, the network looks neutral.

Section 3: Assessing the number of communities / small world models

Number of communities

```
#> [1] 24
```

```
#> [1] 21
```

```
#> [1] 21
```

```
#> [1] 23
```

```
#> [1] 26
```

For assessing the number of communities given by these solutions, we consider the Erdös and Rényi model and the generalized random graph model after fixing the degree sequence of the our network.

```
#> [1] 470
#> [1] 500
```

We will start with the Erdös and Rényi model.

We first generate 1000 networks from the Erdös and Rényi model with N=470 vertices and L=500 edges. Then, for each network, we run the five community detection algorithms and select the number of communities detected by each of them.

Now, we use the degree sequence and generate 1000 networks from the generalized random graph model after fixing the degree sequence of the network with N=470 vertices.

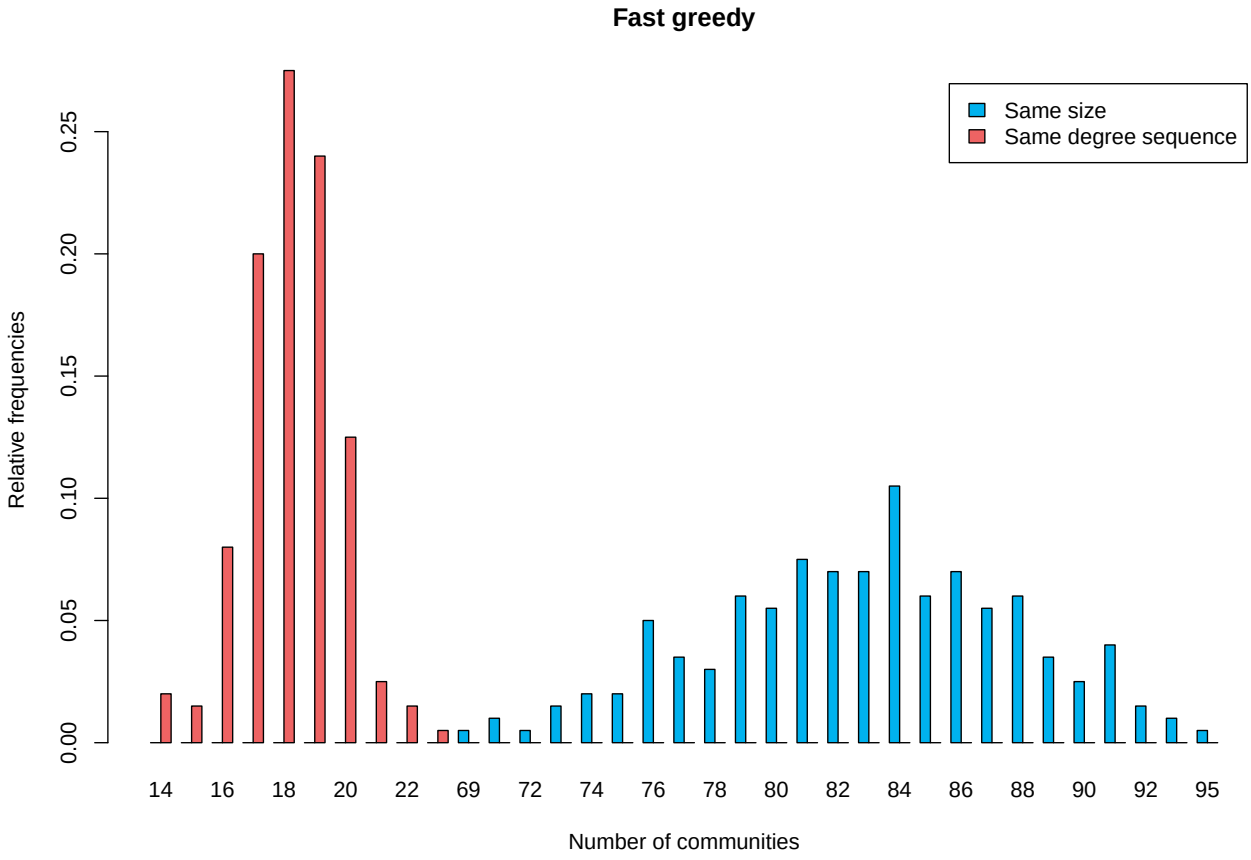
Then, for each network, we run the six community detection algorithms and select the number of communities detected by each of them.

Once the simulations have finished, we can compare the different results obtained with each community detection algorithm with barplots:

Fast greedy Method

The table shows the probability of obtaining different number of communities:

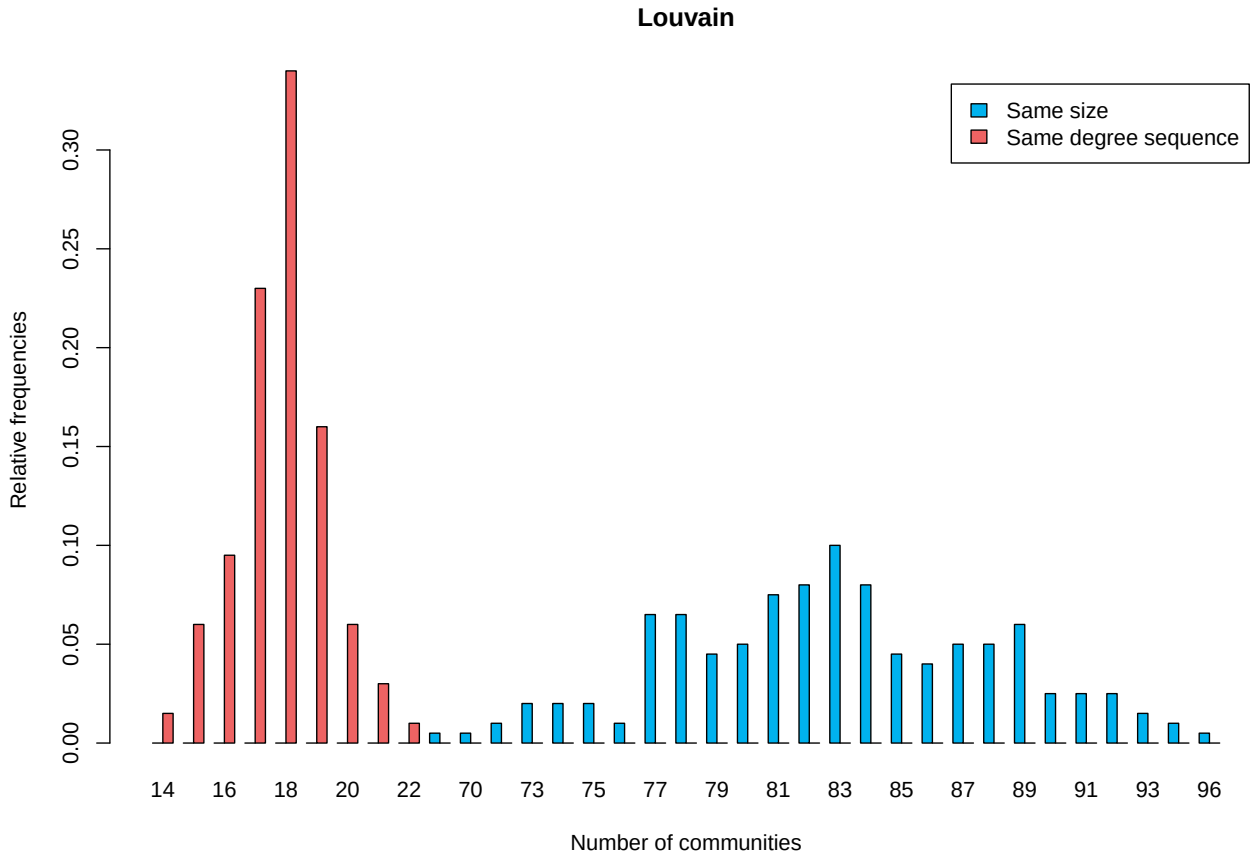
```
#>                 comm_ER_G_fg
#> ind_comm_ER_G_fg    14    15    16    17    18    19    20    21    22    23
#>                1 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>                2 0.020 0.015 0.080 0.200 0.275 0.240 0.125 0.025 0.015 0.005
#>                 comm_ER_G_fg
#> ind_comm_ER_G_fg    69    71    72    73    74    75    76    77    78    79
#>                1 0.005 0.010 0.005 0.015 0.020 0.020 0.050 0.035 0.030 0.060
#>                2 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>                 comm_ER_G_fg
#> ind_comm_ER_G_fg    80    81    82    83    84    85    86    87    88    89
#>                1 0.055 0.075 0.070 0.070 0.105 0.060 0.070 0.055 0.060 0.035
#>                2 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>                 comm_ER_G_fg
#> ind_comm_ER_G_fg    90    91    92    94    95
#>                1 0.025 0.040 0.015 0.010 0.005
#>                2 0.000 0.000 0.000 0.000 0.000
```



Louvain Method

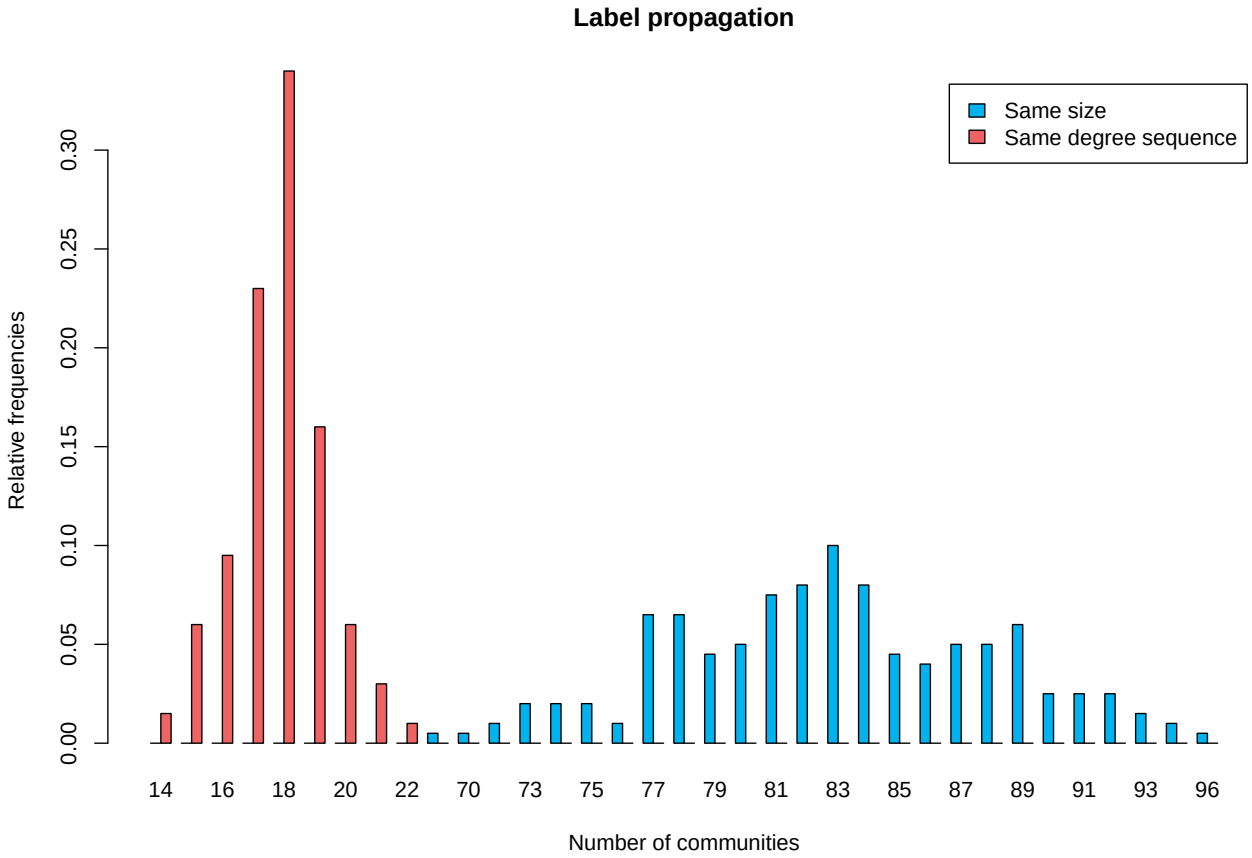
```
#>                 comm_ER_G_lo
#> ind_comm_ER_G_lo    14    15    16    17    18    19    20    21    22    69
#>                1 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.005
#>                2 0.015 0.060 0.095 0.230 0.340 0.160 0.060 0.030 0.010 0.000
#>                 comm_ER_G_lo
#> ind_comm_ER_G_lo    70    71    73    74    75    76    77    78    79    80
#>                1 0.005 0.010 0.020 0.020 0.020 0.010 0.065 0.065 0.045 0.050
#>                2 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>                 comm_ER_G_lo
#> ind_comm_ER_G_lo    81    82    83    84    85    86    87    88    89    90
```

```
#>               1 0.075 0.080 0.100 0.080 0.045 0.040 0.050 0.050 0.060 0.025
#>               2 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>                comm_ER_G_lo
#> ind_comm_ER_G_lo    91    92    93    94    96
#>               1 0.025 0.025 0.015 0.010 0.005
#>               2 0.000 0.000 0.000 0.000 0.000
```



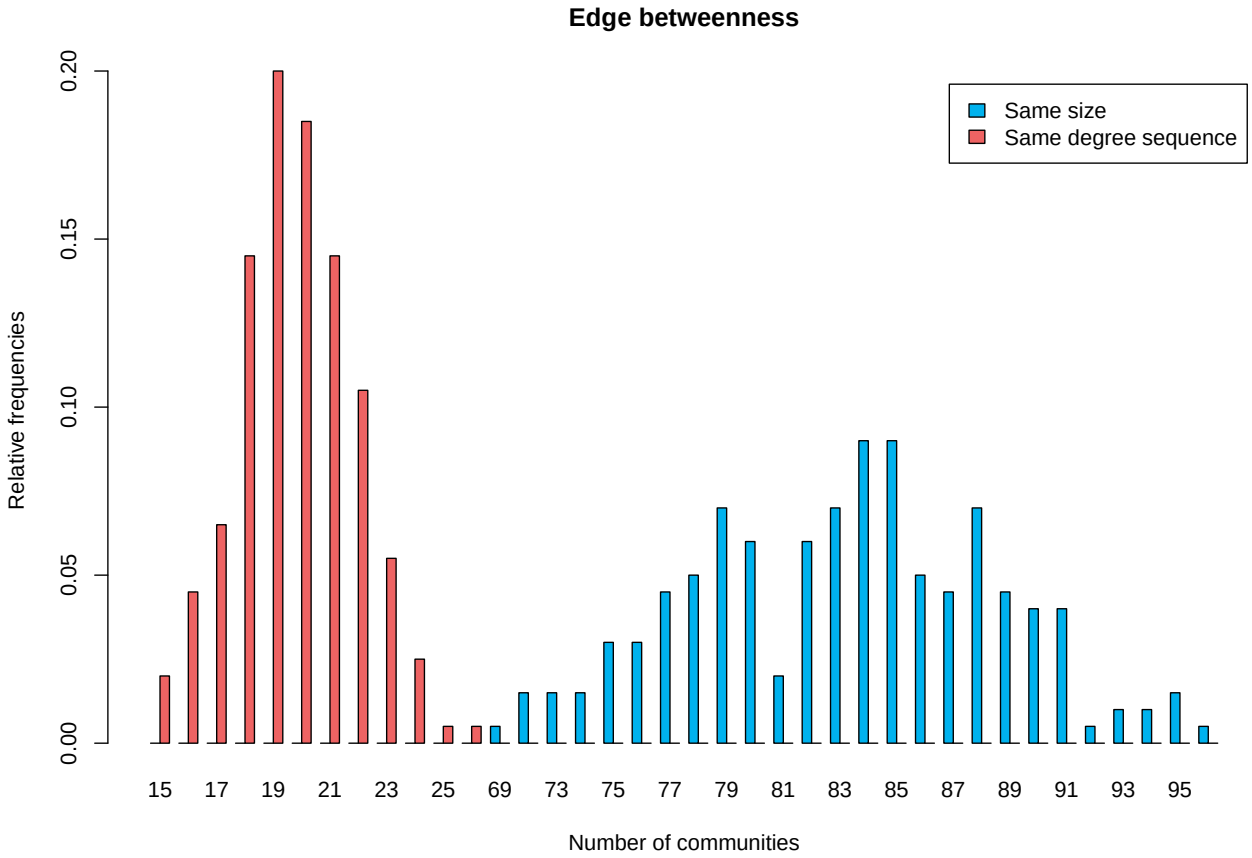
Label propagation Method

```
#>                comm_ER_G_lp
#> ind_comm_ER_G_lp    14    15    16    17    18    19    20    21    22    69
#>               1 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.005
#>               2 0.015 0.060 0.095 0.230 0.340 0.160 0.060 0.030 0.010 0.000
#>                comm_ER_G_lp
#> ind_comm_ER_G_lp    70    71    73    74    75    76    77    78    79    80
#>               1 0.005 0.010 0.020 0.020 0.020 0.010 0.065 0.065 0.045 0.050
#>               2 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>                comm_ER_G_lp
#> ind_comm_ER_G_lp    81    82    83    84    85    86    87    88    89    90
#>               1 0.075 0.080 0.100 0.080 0.045 0.040 0.050 0.050 0.060 0.025
#>               2 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>                comm_ER_G_lp
#> ind_comm_ER_G_lp    91    92    93    94    96
#>               1 0.025 0.025 0.015 0.010 0.005
#>               2 0.000 0.000 0.000 0.000 0.000
```



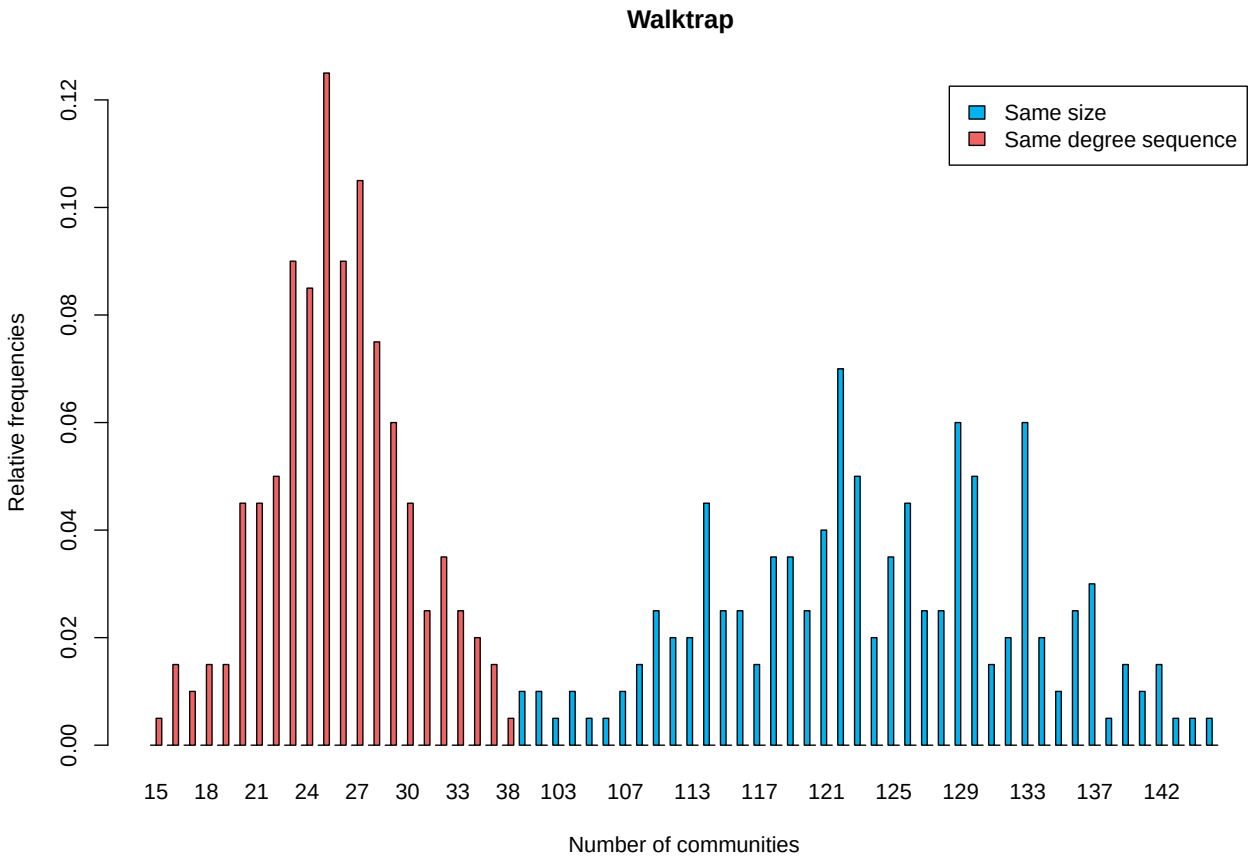
Edge betweeness Method

```
#>                comm_ER_G_eb
#> ind_comm_ER_G_eb    15    16    17    18    19    20    21    22    23    24
#>               1 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>               2 0.020 0.045 0.065 0.145 0.200 0.185 0.145 0.105 0.055 0.025
#>                comm_ER_G_eb
#> ind_comm_ER_G_eb    25    26    69    72    73    74    75    76    77    78
#>               1 0.000 0.000 0.005 0.015 0.015 0.015 0.030 0.030 0.045 0.050
#>               2 0.005 0.005 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>                comm_ER_G_eb
#> ind_comm_ER_G_eb    79    80    81    82    83    84    85    86    87    88
#>               1 0.070 0.060 0.020 0.060 0.070 0.090 0.090 0.050 0.045 0.070
#>               2 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>                comm_ER_G_eb
#> ind_comm_ER_G_eb    89    90    91    92    93    94    95    98
#>               1 0.045 0.040 0.040 0.005 0.010 0.010 0.015 0.005
#>               2 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
```



Walktrap Method

```
#>                comm_ER_G_wa
#> ind_comm_ER_G_wa    15    16    17    18    19    20    21    22    23    24
#>               1 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>               2 0.005 0.015 0.010 0.015 0.015 0.045 0.045 0.050 0.090 0.085
#>                comm_ER_G_wa
#> ind_comm_ER_G_wa    25    26    27    28    29    30    31    32    33    34
#>               1 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>               2 0.125 0.090 0.105 0.075 0.060 0.045 0.025 0.035 0.025 0.020
#>                comm_ER_G_wa
#> ind_comm_ER_G_wa    35    38    96   102   103   104   105   106   107   110
#>               1 0.000 0.000 0.010 0.010 0.005 0.010 0.005 0.005 0.010 0.015
#>               2 0.015 0.005 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>                comm_ER_G_wa
#> ind_comm_ER_G_wa   111   112   113   114   115   116   117   118   119   120
#>               1 0.025 0.020 0.020 0.045 0.025 0.025 0.015 0.035 0.035 0.025
#>               2 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>                comm_ER_G_wa
#> ind_comm_ER_G_wa   121   122   123   124   125   126   127   128   129   130
#>               1 0.040 0.070 0.050 0.020 0.035 0.045 0.025 0.025 0.060 0.050
#>               2 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>                comm_ER_G_wa
#> ind_comm_ER_G_wa   131   132   133   134   135   136   137   138   140   141
#>               1 0.015 0.020 0.060 0.020 0.010 0.025 0.030 0.005 0.015 0.010
#>               2 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
#>                comm_ER_G_wa
#> ind_comm_ER_G_wa   142   143   146   148
#>               1 0.015 0.005 0.005 0.005
#>               2 0.000 0.000 0.000 0.000
```



From the barplots we can obtain the following conclusions:

1. Fast greedy method has selected 24 communities, however the barplot suggests that it has a very low frequency, so it seems that the solution is not appropriate.
2. Louvain has selected 21 communities, the barplot suggests a lower number with the networks of same size and larger number with the networks of same degree of sequences, so it is probably not appropriate number of communities to select.
3. Label propagation has selected 21 communities, the barplot suggests that 21 has very low frequency, so the solution might not be correct.
4. Edge betweeness has selected 23 communities, the barplot suggests that it might be a good option for the networks with same size but not with same degree sequences.
5. Walktrap selected 26 communities and the barplot suggests that it might be appropriate for the same sized networks, but not appropriate for the same degree sequences.

Small world models

In this part of the work, we will use the method to assess whether our github network has the small world property. For that, we generate a large number of random networks with the same number of vertices and edges than those of the github network. Then, for each network we obtain the clustering coefficient and the average path length.

```
#> [1] 0
```

```
#> [1] Inf
```

Now we compare the results obtained with those of our own network. The average path length of the the subset of github network is around 7 which is in the middle of the histogram. However, the clustering

coefficient is 0. Which suggests that our network may have the small world property.