

R and Arxiv GR-QC Graph Analysis

Ran Wei

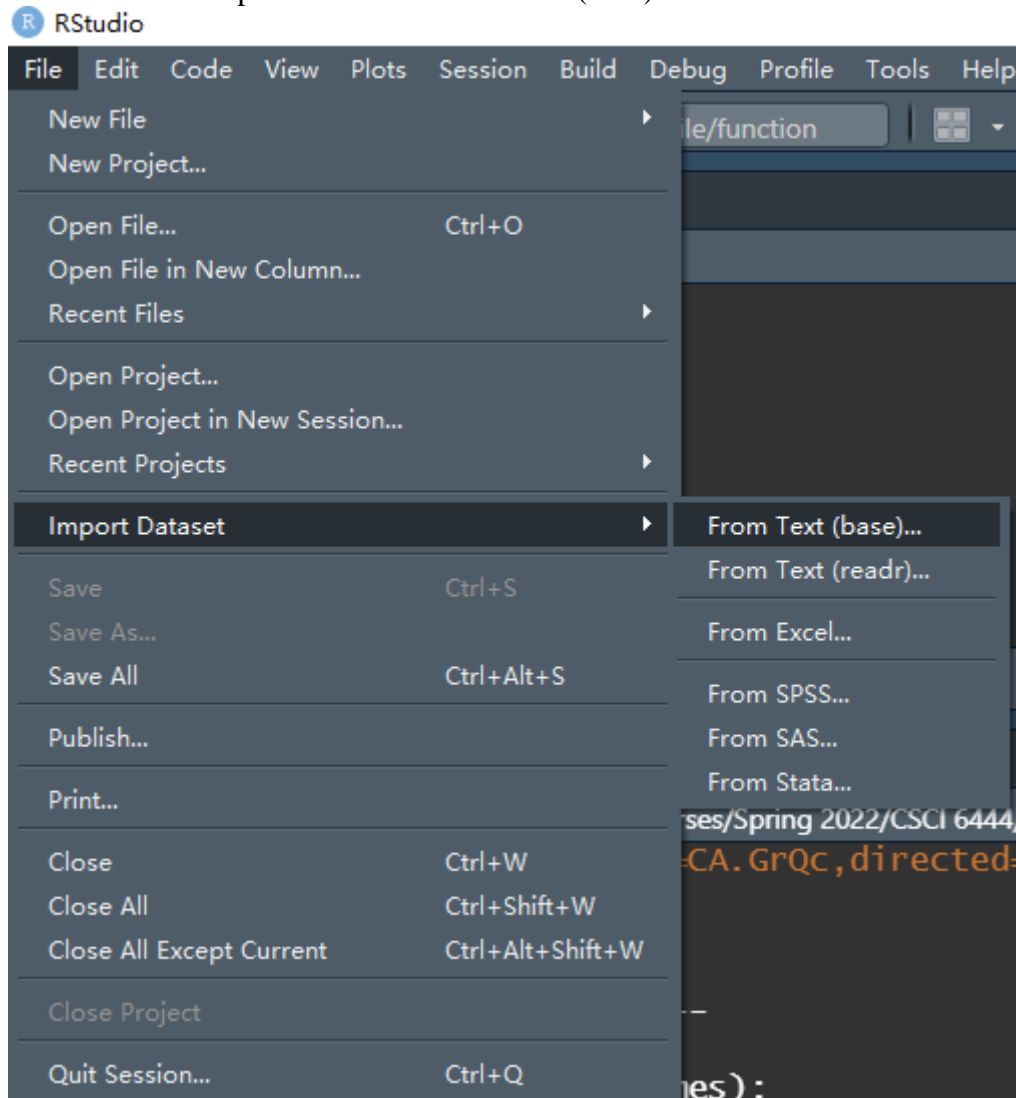
Feb 10, 2022

Part 1 Dataset Loading

In order to learn how to use RStudio for graph analysis, I used the dataset of Arxiv GR-QC (General Relativity and Quantum Cosmology) collaboration network. This dataset contains 5242 nodes and 14496 edges in total and covers papers in the period from January 1993 to April 2003 (124 months).

a. Loading Approach 1: Import

File → Import Dataset → From Text (base)



b. Loading Approach 2: read.table() function

```
CA.GrQc = read.table(file="CA-GrQc.txt", header = TRUE, comment.char="#")
```

Then I got a dataframe variable named 'CA.GrQc'.

Part 2 Initial Graph Plotting

I installed the igraph package from CRAN mirrors using the `install.package()` function and loaded the package by `library()` function. Before continuing, I was required to convert the dataframe into an igraph format using `graph_from_data_frame()`. I named this object 'graph' and then used `plot.igraph()` to create the plot for the entire dataset as shown in Figure 1.

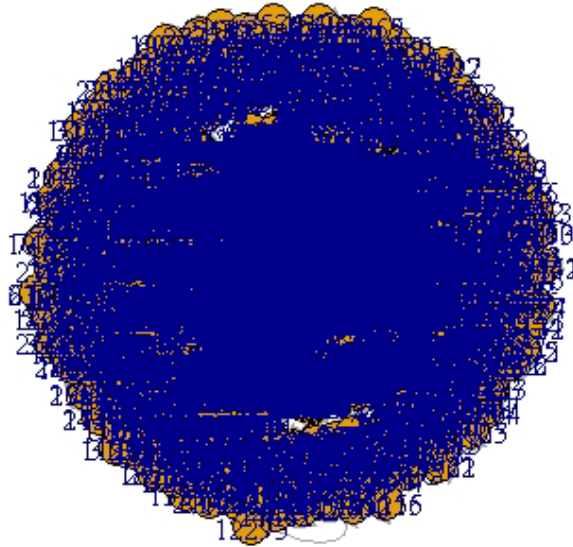


Figure 1. Raw Dataset

Due to the large amount of data, I can hardly observe the data structure, even though I set parameters like 'vertex.label.cex' and 'vertex.size', to view the graph more clearly (Figure 2).

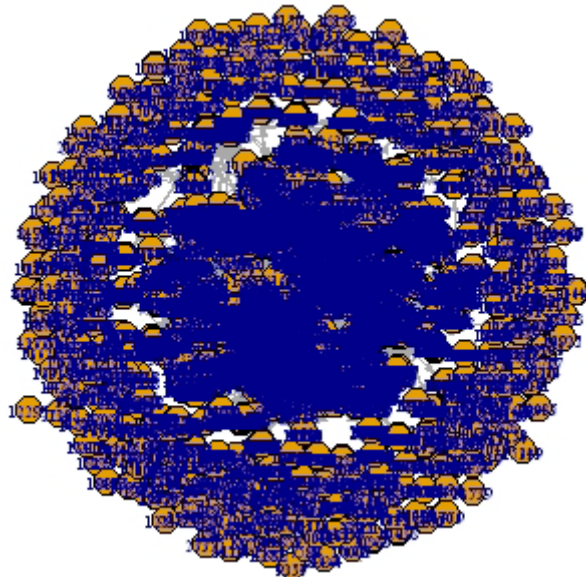


Figure 2. Raw Dataset (Clearer Version)

Part 3 Dataset Simplifying and Plotting

To solve the problem described above, I took the following steps towards graph simplification:

- Step 1. Choose the nodes with degree greater than or equal to 20 and plot the graph.

Code Implementation:

```
vertices_lt20 = V(graph)[igraph::degree(graph)<20]  
graph1 = igraph::delete.vertices(graph,vertices_lt20) #remove nodes (degree <=20)  
plot.igraph(graph1, edge.arrow.size=0.3, vertex.size=10, vertex.label.cex=0.5)
```

Plot Result as Figure 3:

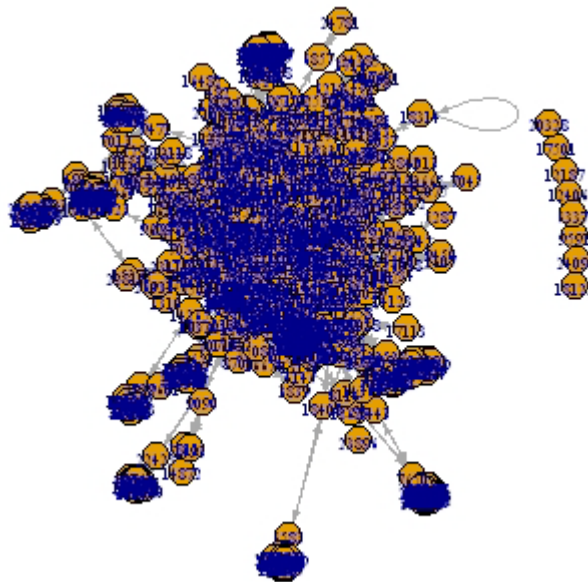


Figure 3. Simplify-Step 1

It is still hard to see the internal structure, so I continue to step 2.

- Step 2. Choose the first 200 edges and plot the graph.

Code Implementation:

```
graph1_df = as_data_frame(graph1, what="edges") #convert igraph object to dataframe  
graph2_df = graph1_df[1:200,] #choose the first 200 edges  
graph2 = graph_from_data_frame(graph2_df)  
plot.igraph(graph2, edge.arrow.size=0.3, vertex.size=10, vertex.label.cex=0.5)
```

Plot Result as Figure 4:

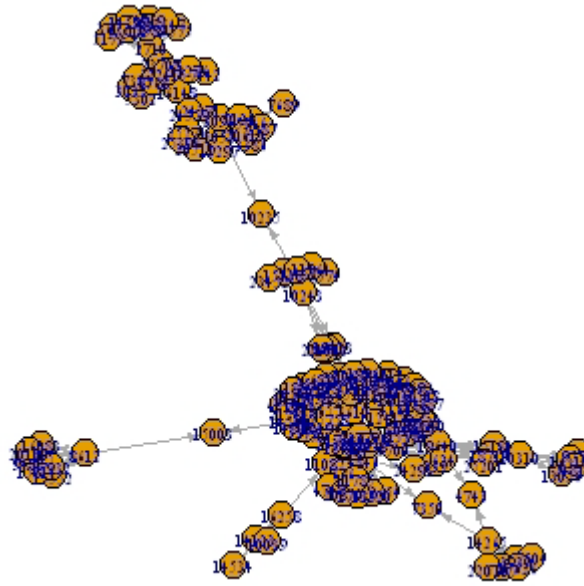


Figure 4. Simplify-Step 2

Now the structure is become clearer, but I can still simplify some more.

- Step 3. Choose the nodes with degree larger than 2 and plot the graph.

Code Implementation:

```
vertices_lt2 <- V(graph2)[igraph::degree(graph2)<2]
graph.sim = igraph::delete.vertices(graph2, vertices_lt2)
plot.igraph(graph.sim, edge.arrow.size=0.3, vertex.size=10, vertex.label.cex=0.5)
```

Plot Result as Figure 5:

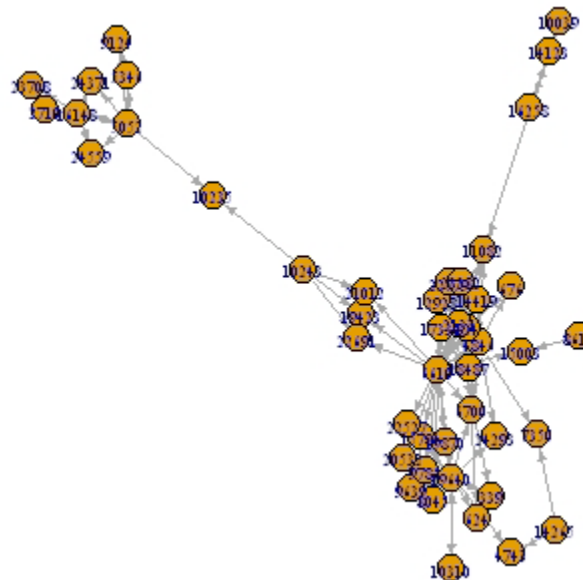


Figure 5. Simplify-Step 3

Finally, I arrive at a three-cluster igraph object for deeper analysis and plotting.

- Vertices of the Graph

- Edges of the Graph

- Adjacency Matrix: Indicates which vertices are connected by edges

[illegible]

	24371	24559	339	624	4743	8045	9639	9785	15784	19870	20532	22527	24293	19423	21012	22691	7350
10310	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5052	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5346	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19640	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
10243	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
14265	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
8612	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16258	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14123	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2710	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16148	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4846	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
824	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2133	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6610	0	0	0	0	0	1	1	1	1	1	1	1	0	1	1	1	0
6700	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
9124	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10235	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24371	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24559	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
339	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15003	10039	11082	23708	676	12928	14419	17330	18487	22779	23382
10310	0	0	0	0	0	0	0	0	0	0	0
5052	0	0	0	0	0	0	0	0	0	0	0
5346	0	0	0	0	0	0	0	0	0	0	0
19640	0	0	0	0	0	0	0	0	0	0	0
10243	0	0	0	0	0	0	0	0	0	0	0
14265	0	0	0	0	0	0	0	0	0	0	0
8612	1	0	0	0	0	0	0	0	0	0	0
16258	0	1	1	0	0	0	0	0	0	0	0
14123	0	1	0	0	0	0	0	0	0	0	0
2710	0	0	0	1	0	0	0	0	0	0	0
16148	0	0	0	1	0	0	0	0	0	0	0
4846	0	0	1	0	1	1	1	1	1	1	1
824	0	0	1	0	1	1	1	1	1	1	1
2133	0	0	1	0	0	1	1	1	1	1	1
6610	1	0	1	0	0	1	1	1	1	1	1
6700	0	0	0	0	0	0	0	0	0	0	0
9124	0	0	0	0	0	0	0	0	0	0	0
10235	0	0	0	0	0	0	0	0	0	0	0
24371	0	0	0	0	0	0	0	0	0	0	0
24559	0	0	0	0	0	0	0	0	0	0	0
339	0	0	0	0	0	0	0	0	0	0	0

[reached getOption("max.print") -- omitted 25 rows]

- Density: Gives us the ratio of edges in the graph vs the maximum number of possible edges in a graph. When loops = T this means that the maximum counts self-loops.
Note: I use as.matrix() for compatibility with the sna functions.

```
> graph.sim.adj = as.matrix(get.adjacency(graph.sim))
> graph.sim.density = gden(graph.sim.adj)
> graph.sim.density
[1] 0.05072464
> edge_density(graph.sim)
[1] 0.05072464
> edge_density(graph.sim, loops=T)
[1] 0.04962193
```

- Egocentric Network and Degrees: Creates a subgraph of a given node and its immediate neighbors. At the bottom there is the number of degrees per node, which is a measurement of the number of edges connected to it

```
> graph.sim.ego = ego.extract(graph.sim.adj)
> graph.sim.ego["8612"]
$`8612`
      8612 15003
8612      0      1
15003      0      0
> graph.sim.ego["6610"]
$`6610`
      6610 19640 4846 824 2133 6700 8045 9639 9785 15784 19870 20532 22527 19423 21012 22691 15003
6610      0      1      1      1      1      1      1      1      1      1      1      1      1      1      1      1
19640      1      0      0      0      0      1      1      1      1      1      1      1      0      0      0      0
4846      1      0      0      1      1      1      0      0      0      0      0      0      0      0      0      0
824      1      0      1      0      1      1      0      0      0      0      0      0      0      0      0      0
2133      1      0      1      1      0      1      0      0      0      0      0      0      0      0      0      0
6700      0      0      0      1      0      0      0      0      0      0      0      0      0      0      0      0
8045      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
9639      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
9785      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
15784      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
19870      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
20532      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
22527      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
19423      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
21012      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
22691      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
15003      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
11082      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
12928      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
14419      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
17330      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
18487      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
22779      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
23382      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0

      11082 12928 14419 17330 18487 22779 23382
6610      1      1      1      1      1      1      1
19640      0      0      0      0      0      0      0
4846      1      1      1      1      1      1      1
824      1      1      1      1      1      1      1
2133      1      1      1      1      1      1      1
6700      0      0      0      0      0      0      0
8045      0      0      0      0      0      0      0
9639      0      0      0      0      0      0      0
9785      0      0      0      0      0      0      0
15784      0      0      0      0      0      0      0
19870      0      0      0      0      0      0      0
20532      0      0      0      0      0      0      0
22527      0      0      0      0      0      0      0
19423      0      0      0      0      0      0      0
21012      0      0      0      0      0      0      0
22691      0      0      0      0      0      0      0
15003      0      0      0      0      0      0      0
11082      0      0      0      0      0      0      0
12928      0      0      0      0      0      0      0
14419      0      0      0      0      0      0      0
17330      0      0      0      0      0      0      0
18487      0      0      0      0      0      0      0
22779      0      0      0      0      0      0      0
23382      0      0      0      0      0      0      0

> igraph::degree(graph.sim)
10310 5052 5346 19640 10243 14265 8612 16258 14123 2710 16148 4846 824 2133 6610 6700 9124
      2      8      3      16      4      2      1      4      3      3      7      17      16      14      27      8      2
10235 24371 24559 339 624 4743 8045 9639 9785 15784 19870 20532 22527 24293 19423 21012 22691
      2      2      2      2      2      2      2      2      2      2      2      2      2      2      2      2      2
7350 15003 10039 11082 23708 676 12928 14419 17330 18487 22779 23382
      2      2      2      5      2      2      4      4      4      4      4      4
```


- Betweenness Centrality: The betweenness centrality is measured through the number of times a node is present in the shortest path. The centralization is the graph level centrality, and the theoretical max is the maximum possible centrality for a graph with these parameters

```
> igraph::centr_betw(graph.sim)
$res
[1] 0.000000 12.000000 0.000000 40.500000 0.000000 0.000000 0.000000 1.000000 0.000000 0.000000
[11] 10.000000 11.166667 29.333333 0.000000 86.333333 8.666667 0.000000 0.000000 0.000000 0.000000
[21] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
[31] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
[41] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

$centralization
[1] 0.0423382

$theoretical_max
[1] 89100
```

- Closeness Centrality: The closeness centrality is similar to betweenness but is calculated using the average shortest path.

```
> igraph::centr_clo(graph.sim)
$res
[1] 0.05867014 0.02628505 0.02619325 0.06097561 0.02380952 0.02272727 0.02222222 0.02325581 0.02324380
[10] 0.02617801 0.02626970 0.06097561 0.06081081 0.06072874 0.06189821 0.05882353 0.02173913 0.02173913
[19] 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913
[28] 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913
[37] 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913 0.02173913
[46] 0.02173913

$centralization
[1] 0.03525705

$theoretical_max
[1] 44.02174
```

- Shortest Paths: The length of the shortest path between any 2 nodes in the graph. I also show all the shortest paths for node 6610.

```
> igraph::shortest.paths(graph.sim)
      10310 5052 5346 19640 10243 14265 8612 16258 14123 2710 16148 4846 824 2133 6610 6700 9124 10235
10310      0      6      7      1      4      3      4      4      5      8      7      3      3      3      2      2      7      5
5052      6      0      1      5      2      7      6      6      7      2      1      5      5      5      4      5      1      1
5346      7      1      0      6      3      8      7      7      8      3      2      6      6      6      5      6      1      2
19640     1      5      6      0      3      2      3      3      4      7      6      2      2      2      1      1      6      4
10243     4      2      3      3      0      5      4      4      5      4      3      3      3      3      2      3      3      1
14265     3      7      8      2      5      0      5      4      5      9      8      2      3      3      3      3      8      6
8612      4      6      7      3      4      5      0      4      5      8      7      3      3      3      2      3      7      5
16258     4      6      7      3      4      4      4      0      1      8      7      2      2      2      2      3      7      5
14123     5      7      8      4      5      5      5      1      0      9      8      3      3      3      3      4      8      6
2710      8      2      3      7      4      9      8      8      9      0      1      7      7      7      6      7      3      3
16148     7      1      2      6      3      8      7      7      8      1      0      6      6      6      5      6      2      2
4846      3      5      6      2      3      2      3      2      3      7      6      0      1      1      1      1      6      4
824       3      5      6      2      3      3      3      2      3      7      6      1      0      1      1      1      6      4
2133      3      5      6      2      3      3      3      2      3      7      6      1      1      0      1      1      6      4
6610      2      4      5      1      2      3      2      2      3      6      5      1      1      1      0      1      5      3
6700      2      5      6      1      3      3      3      3      4      7      6      1      1      1      1      0      6      4
9124      7      1      1      6      3      8      7      7      8      3      2      6      6      6      5      6      0      2
10235     5      1      2      4      1      6      5      5      6      3      2      4      4      4      3      4      2      0
24371     7      1      2      6      3      8      7      7      8      2      1      6      6      6      5      6      2      2
24559     7      1      2      6      3      8      7      7      8      2      1      6      6      6      5      6      2      2
339       2      6      7      1      4      3      4      4      5      8      7      2      2      2      2      1      7      5
```

	24371	24559	339	624	4743	8045	9639	9785	15784	19870	20532	22527	24293	19423	21012	22691	7350
10310	7	7	2	2	2	2	2	2	2	2	2	2	2	3	3	3	4
5052	1	1	6	6	6	5	5	5	5	5	5	5	6	3	3	3	6
5346	2	2	7	7	7	6	6	6	6	6	6	6	7	4	4	4	7
19640	6	6	1	1	1	1	1	1	1	1	1	1	1	2	2	2	3
10243	3	3	4	4	4	3	3	3	3	3	3	3	4	1	1	1	4
14265	8	8	3	3	1	3	3	3	3	3	3	3	3	4	4	4	1
8612	7	7	4	4	4	3	3	3	3	3	3	3	4	3	3	3	4
16258	7	7	4	4	4	3	3	3	3	3	3	3	3	3	3	3	3
14123	8	8	5	5	5	4	4	4	4	4	4	4	4	4	4	4	4
2710	2	2	8	8	8	7	7	7	7	7	7	7	8	5	5	5	8
16148	1	1	7	7	7	6	6	6	6	6	6	6	7	4	4	4	7
4846	6	6	2	2	3	2	2	2	2	2	2	2	2	1	2	2	1
824	6	6	2	2	3	2	2	2	2	2	2	2	2	2	2	2	2
2133	6	6	2	2	3	2	2	2	2	2	2	2	2	2	2	2	2
6610	5	5	2	2	2	1	1	1	1	1	1	1	2	1	1	1	2
6700	6	6	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2
9124	2	2	7	7	7	6	6	6	6	6	6	6	7	4	4	4	7
10235	2	2	5	5	5	4	4	4	4	4	4	4	5	2	2	2	5
24371	0	2	7	7	7	6	6	6	6	6	6	6	7	4	4	4	7
24559	2	0	7	7	7	6	6	6	6	6	6	6	7	4	4	4	7
339	7	7	0	2	2	2	2	2	2	2	2	2	2	3	3	3	3
	15003	10039	11082	23708	676	12928	14419	17330	18487	22779	23382						
10310	3	5	3	8	4	3	3	3	3	3	3						
5052	5	7	5	2	6	5	5	5	5	5	5						
5346	6	8	6	3	7	6	6	6	6	6	6						
19640	2	4	2	7	3	2	2	2	2	2	2						
10243	3	5	3	4	4	3	3	3	3	3	3						
14265	4	5	3	9	3	3	3	3	3	3	3						
8612	1	5	3	8	4	3	3	3	3	3	3						
16258	3	1	1	8	3	3	3	3	3	3	3						
14123	4	1	2	9	4	4	4	4	4	4	4						
2710	7	9	7	1	8	7	7	7	7	7	7						
16148	6	8	6	1	7	6	6	6	6	6	6						
4846	2	3	1	7	1	1	1	1	1	1	1						
824	2	3	1	7	1	1	1	1	1	1	1						
2133	2	3	1	7	2	1	1	1	1	1	1						
6610	1	3	1	6	2	1	1	1	1	1	1						
6700	2	4	2	7	2	2	2	2	2	2	2						
9124	6	8	6	3	7	6	6	6	6	6	6						
10235	4	6	4	3	5	4	4	4	4	4	4						
24371	6	8	6	2	7	6	6	6	6	6	6						
24559	6	8	6	2	7	6	6	6	6	6	6						
339	3	5	3	8	3	3	3	3	3	3	3						

```
> igraph::get.shortest.paths(graph.sim,"6610")
$vpath
$vpath[[1]]
+ 3/46 vertices, named, from 4df0ba4:
[1] 6610 19640 10310

$vpath[[2]]
+ 0/46 vertices, named, from 4df0ba4:

$vpath[[3]]
+ 0/46 vertices, named, from 4df0ba4:

$vpath[[4]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 19640

$vpath[[5]]
+ 0/46 vertices, named, from 4df0ba4:
```

```
$vpath[[6]]
+ 0/46 vertices, named, from 4df0ba4:

$vpath[[7]]
+ 0/46 vertices, named, from 4df0ba4:

$vpath[[8]]
+ 0/46 vertices, named, from 4df0ba4:

$vpath[[9]]
+ 0/46 vertices, named, from 4df0ba4:

$vpath[[10]]
+ 0/46 vertices, named, from 4df0ba4:

$vpath[[11]]
+ 0/46 vertices, named, from 4df0ba4:

$vpath[[12]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 4846

$vpath[[13]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 824

$vpath[[14]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 2133

$vpath[[15]]
+ 1/46 vertex, named, from 4df0ba4:
[1] 6610

$vpath[[16]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 6700

$vpath[[17]]
+ 0/46 vertices, named, from 4df0ba4:

$vpath[[18]]
+ 0/46 vertices, named, from 4df0ba4:

$vpath[[19]]
+ 0/46 vertices, named, from 4df0ba4:
```

```
$vpath[[20]]
+ 0/46 vertices, named, from 4df0ba4:

$vpath[[21]]
+ 3/46 vertices, named, from 4df0ba4:
[1] 6610 19640 339

$vpath[[22]]
+ 3/46 vertices, named, from 4df0ba4:
[1] 6610 19640 624

$vpath[[23]]
+ 3/46 vertices, named, from 4df0ba4:
[1] 6610 19640 4743

$vpath[[24]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 8045

$vpath[[25]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 9639

$vpath[[26]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 9785

$vpath[[27]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 15784

$vpath[[28]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 19870

$vpath[[29]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 20532

$vpath[[30]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 22527

$vpath[[31]]
+ 3/46 vertices, named, from 4df0ba4:
[1] 6610 19640 24293
```

```
$vpath[[32]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 19423

$vpath[[33]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 21012

$vpath[[34]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 22691

$vpath[[35]]
+ 3/46 vertices, named, from 4df0ba4:
[1] 6610 4846 7350

$vpath[[36]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 15003

$vpath[[37]]
+ 0/46 vertices, named, from 4df0ba4:

$vpath[[38]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 11082

$vpath[[39]]
+ 0/46 vertices, named, from 4df0ba4:

$vpath[[40]]
+ 3/46 vertices, named, from 4df0ba4:
[1] 6610 4846 676
```

```
$vpath[[41]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 12928

$vpath[[42]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 14419

$vpath[[43]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 17330

$vpath[[44]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 18487

$vpath[[45]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 22779

$vpath[[46]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 6610 23382

$epath
NULL

$predecessors
NULL

$inbound_edges
NULL
```

- Geodesic: The shortest path between any two nodes in the network

[illegible]

	[,18]	[,19]	[,20]	[,21]	[,22]	[,23]	[,24]	[,25]	[,26]	[,27]	[,28]	[,29]	[,30]	[,31]	[,32]	[,33]	
[1,]	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	
[2,]	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
[3,]	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
[4,]	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	
[5,]	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	
[6,]	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
[7,]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
[8,]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
[9,]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
[10,]	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
[11,]	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
[12,]	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	
[13,]	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	
[14,]	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	
[15,]	0	0	0	2	2	1	1	1	1	1	1	1	1	2	1	1	
[16,]	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	
[17,]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
[18,]	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
[19,]	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
[20,]	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
[21,]	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
	[,34]	[,35]	[,36]	[,37]	[,38]	[,39]	[,40]	[,41]	[,42]	[,43]	[,44]	[,45]	[,46]				
[1,]	1	1	1	0	1	0	3	1	1	1	1	1	1				
[2,]	0	0	0	0	0	1	0	0	0	0	0	0	0				
[3,]	0	0	0	0	0	1	0	0	0	0	0	0	0				
[4,]	1	1	1	0	1	0	3	1	1	1	1	1	1				
[5,]	1	0	0	0	0	0	0	0	0	0	0	0	0				
[6,]	0	1	0	0	0	0	0	0	0	0	0	0	0				
[7,]	0	0	1	0	0	0	0	0	0	0	0	0	0				
[8,]	0	0	0	1	1	0	0	0	0	0	0	0	0				
[9,]	0	0	0	1	1	0	0	0	0	0	0	0	0				
[10,]	0	0	0	0	0	1	0	0	0	0	0	0	0				
[11,]	0	0	0	0	0	1	0	0	0	0	0	0	0				
[12,]	1	1	1	0	1	0	1	1	1	1	1	1	1				
[13,]	1	1	1	0	1	0	1	1	1	1	1	1	1				
[14,]	1	1	1	0	1	0	2	1	1	1	1	1	1				
[15,]	1	1	1	0	1	0	2	1	1	1	1	1	1				
[16,]	1	1	1	0	1	0	1	1	1	1	1	1	1				
[17,]	0	0	0	0	0	0	0	0	0	0	0	0	0				
[18,]	0	0	0	0	0	0	0	0	0	0	0	0	0				
[19,]	0	0	0	0	0	0	0	0	0	0	0	0	0				
[20,]	0	0	0	0	0	0	0	0	0	0	0	0	0				
[21,]	0	0	0	0	0	0	0	0	0	0	0	0	0				
[reached getOption("max.print") -- omitted 25 rows]																	
\$gdist																	
	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]	[,11]	[,12]	[,13]	[,14]	[,15]	[,16]	[,17]
[1,]	0	Inf	Inf	1	Inf	Inf	Inf	Inf	Inf	Inf	Inf	3	3	3	2	2	Inf
[2,]	Inf	0	1	Inf	Inf	Inf	Inf	Inf	Inf	2	1	Inf	Inf	Inf	Inf	Inf	1
[3,]	Inf	1	0	Inf	Inf	Inf	Inf	Inf	Inf	3	2	Inf	Inf	Inf	Inf	Inf	1
[4,]	1	Inf	Inf	0	Inf	Inf	Inf	Inf	Inf	Inf	Inf	2	2	2	1	1	Inf
[5,]	Inf	Inf	Inf	Inf	0	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[6,]	Inf	Inf	Inf	Inf	Inf	0	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[7,]	Inf	Inf	Inf	Inf	Inf	Inf	0	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[8,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	0	1	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[9,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	1	0	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[10,]	Inf	2	3	Inf	Inf	Inf	Inf	Inf	Inf	0	1	Inf	Inf	Inf	Inf	Inf	3
[11,]	Inf	1	2	Inf	Inf	Inf	Inf	Inf	Inf	1	0	Inf	Inf	Inf	Inf	Inf	2
[12,]	3	Inf	Inf	2	Inf	Inf	Inf	Inf	Inf	Inf	Inf	0	1	1	1	1	Inf
[13,]	3	Inf	Inf	2	Inf	Inf	Inf	Inf	Inf	Inf	Inf	1	0	1	1	1	Inf
[14,]	3	Inf	Inf	2	Inf	Inf	Inf	Inf	Inf	Inf	Inf	1	1	0	1	1	Inf
[15,]	2	Inf	Inf	1	Inf	Inf	Inf	Inf	Inf	Inf	Inf	1	1	1	0	1	Inf
[16,]	4	Inf	Inf	3	Inf	Inf	Inf	Inf	Inf	Inf	Inf	2	1	2	2	0	Inf
[17,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	0
[18,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[19,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[20,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[21,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf

	[,18]	[,19]	[,20]	[,21]	[,22]	[,23]	[,24]	[,25]	[,26]	[,27]	[,28]	[,29]	[,30]	[,31]	[,32]	[,33]
[1,]	Inf	Inf	Inf	2	2	2	2	2	2	2	2	2	2	2	3	3
[2,]	1	1	1	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[3,]	2	2	2	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[4,]	Inf	Inf	Inf	1	1	1	1	1	1	1	1	1	1	1	2	2
[5,]	1	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	1	1
[6,]	Inf	Inf	Inf	Inf	Inf	1	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[7,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[8,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[9,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[10,]	3	2	2	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[11,]	2	1	1	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[12,]	Inf	Inf	Inf	2	2	3	2	2	2	2	2	2	2	2	1	2
[13,]	Inf	Inf	Inf	2	2	3	2	2	2	2	2	2	2	2	2	2
[14,]	Inf	Inf	Inf	2	2	3	2	2	2	2	2	2	2	2	2	2
[15,]	Inf	Inf	Inf	2	2	2	1	1	1	1	1	1	1	1	2	1
[16,]	Inf	Inf	Inf	1	1	4	3	3	3	3	3	3	3	3	3	3
[17,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[18,]	0	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[19,]	Inf	0	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[20,]	Inf	Inf	0	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[21,]	Inf	Inf	Inf	0	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf

	[,34]	[,35]	[,36]	[,37]	[,38]	[,39]	[,40]	[,41]	[,42]	[,43]	[,44]	[,45]	[,46]
[1,]	3	4	3	Inf	3	Inf	4	3	3	3	3	3	3
[2,]	Inf	Inf	Inf	Inf	Inf	2	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[3,]	Inf	Inf	Inf	Inf	Inf	3	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[4,]	2	3	2	Inf	2	Inf	3	2	2	2	2	2	2
[5,]	1	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[6,]	Inf	1	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[7,]	Inf	Inf	1	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[8,]	Inf	Inf	Inf	1	1	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[9,]	Inf	Inf	Inf	1	2	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[10,]	Inf	Inf	Inf	Inf	Inf	1	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[11,]	Inf	Inf	Inf	Inf	Inf	1	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[12,]	2	1	2	Inf	1	Inf	1	1	1	1	1	1	1
[13,]	2	2	2	Inf	1	Inf	1	1	1	1	1	1	1
[14,]	2	2	2	Inf	1	Inf	2	1	1	1	1	1	1
[15,]	1	2	1	Inf	1	Inf	2	1	1	1	1	1	1
[16,]	3	3	3	Inf	2	Inf	2	2	2	2	2	2	2
[17,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[18,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[19,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[20,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf
[21,]	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf

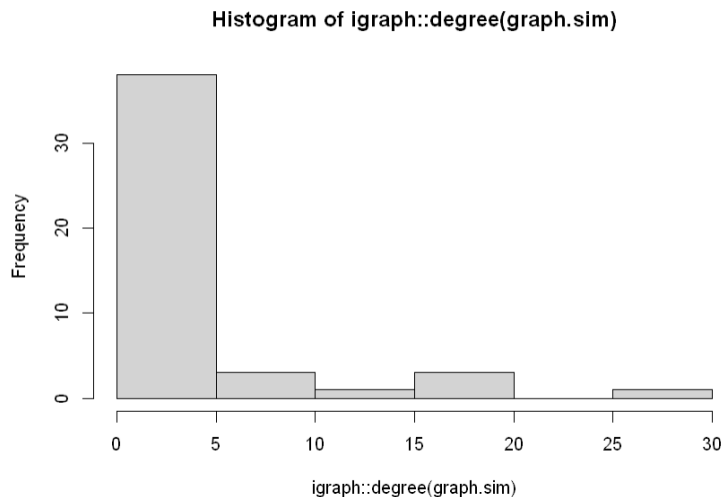
[reached getOption("max.print") -- omitted 25 rows]

- Number of Paths Between Nodes: Gives the total number of possible paths between nodes

```
> graph.sim.adj%%graph.sim.adj
10310 10310 5052 5346 19640 10243 14265 8612 16258 14123 2710 16148 4846 824 2133 6610 6700 9124 10235 24371
10310 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
5052 0 2 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1
5346 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1
19640 0 0 0 2 0 0 0 0 0 0 0 0 1 2 1 0 1 0 0 0
10243 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
14265 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
8612 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
16258 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
14123 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
2710 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1
16148 0 0 1 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 1 1
4846 0 0 0 1 0 0 0 0 0 0 0 0 3 3 2 2 3 0 0 0
824 0 0 0 1 0 0 0 0 0 0 0 0 2 4 2 2 3 0 0 0
2133 0 0 0 1 0 0 0 0 0 0 0 0 2 3 3 2 3 0 0 0
6610 1 0 0 0 0 0 0 0 0 0 0 0 2 3 2 4 4 0 0 0
6700 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0
9124 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
10235 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
24371 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
24559 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
339 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
24559 339 624 4743 8045 9639 9785 15784 19870 20532 22527 24293 19423 21012 22691 7350 15003 10039 11082
10310 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
5052 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5346 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
19640 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
10243 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
14265 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
8612 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
16258 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
14123 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
2710 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
16148 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4846 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0
824 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 3
2133 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 3
6610 0 2 2 1 1 1 1 1 1 1 1 2 0 0 0 1 0 0 3
6700 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
9124 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
10235 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
24371 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
24559 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
339 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
23708 676 12928 14419 17330 18487 22779 23382
10310 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5052 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5346 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
19640 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
10243 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
14265 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
8612 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
16258 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
14123 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2710 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
16148 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4846 0 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
824 0 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
2133 0 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
6610 0 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
6700 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
9124 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
10235 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
24371 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
24559 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
339 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[ reached getOption("max.print") -- omitted 25 rows ]
```

- Histogram: Shows the distribution of the number of degrees per node

```
> hist(igraph::degree(graph.sim))
```



- Diameter: The greatest distance between any pair of vertices

```
> igraph::diameter(graph.sim)
[1] 4
```

- Max Cliques: Returns all the cliques for a given node in the graph

```
> node = c(1)
> graph.sim.1clique = igraph::max_cliques(graph.sim, min=NULL, max=NULL, subset=node)
Warning message:
In igraph::max_cliques(graph.sim, min = NULL, max = NULL, subset = node) :
  At maximal_cliques_template.h:261 :Edge directions are ignored for maximal clique calculation
> graph.sim.1clique
[[1]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 15003 6610

[[2]]
+ 2/46 vertices, named, from 4df0ba4:
[1] 15003 8612
```

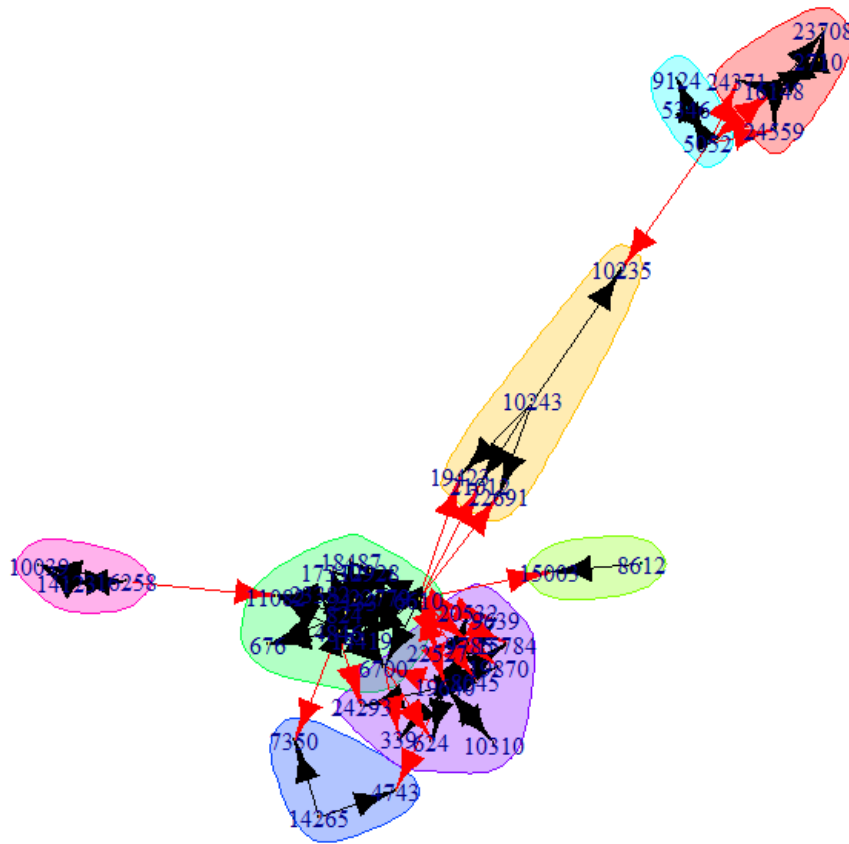
- Largest Cliques: Returns the largest clique in the graph

```
> igraph::clique_num(graph.sim)
[1] 5
```

- Alpha Centrality: Another measurement of centrality which is calculated similarly to eigenvector centrality

```
> alpha centrality(graph.sim, alpha = 0.9)
      10310      5052      5346      19640      10243      14265      8612
26.6324435 -2.6760563 -1.4084507 28.4804928 1.0000000 1.0000000 1.0000000
      16258      14123      2710      16148      4846      824      2133
10.0000000 10.0000000 -1.4084507 -2.6760563 -9.5893224 -6.0780287 -9.5893224
      6610      6700      9124      10235      24371      24559      339
 3.9014374 7.4127310 -2.6760563 -0.5084507 -3.8169014 -3.8169014 33.3039014
      624      4743      8045      9639      9785      15784      19870
33.3039014 27.5324435 30.1437372 30.1437372 30.1437372 30.1437372 30.1437372
      20532      22527      24293      19423      21012      22691      7350
30.1437372 30.1437372 18.0020534 5.4112936 5.4112936 5.4112936 -6.7303901
      15003      10039      11082      23708      676      12928      14419
 5.4112936 19.0000000 -9.2197125 -2.6760563 -13.1006160 -18.2197125 -18.2197125
      17330      18487      22779      23382
-18.2197125 -18.2197125 -18.2197125 -18.2197125
```


- Walktrap Communities: Automatically clusters nodes in order to determine a structure to the graph

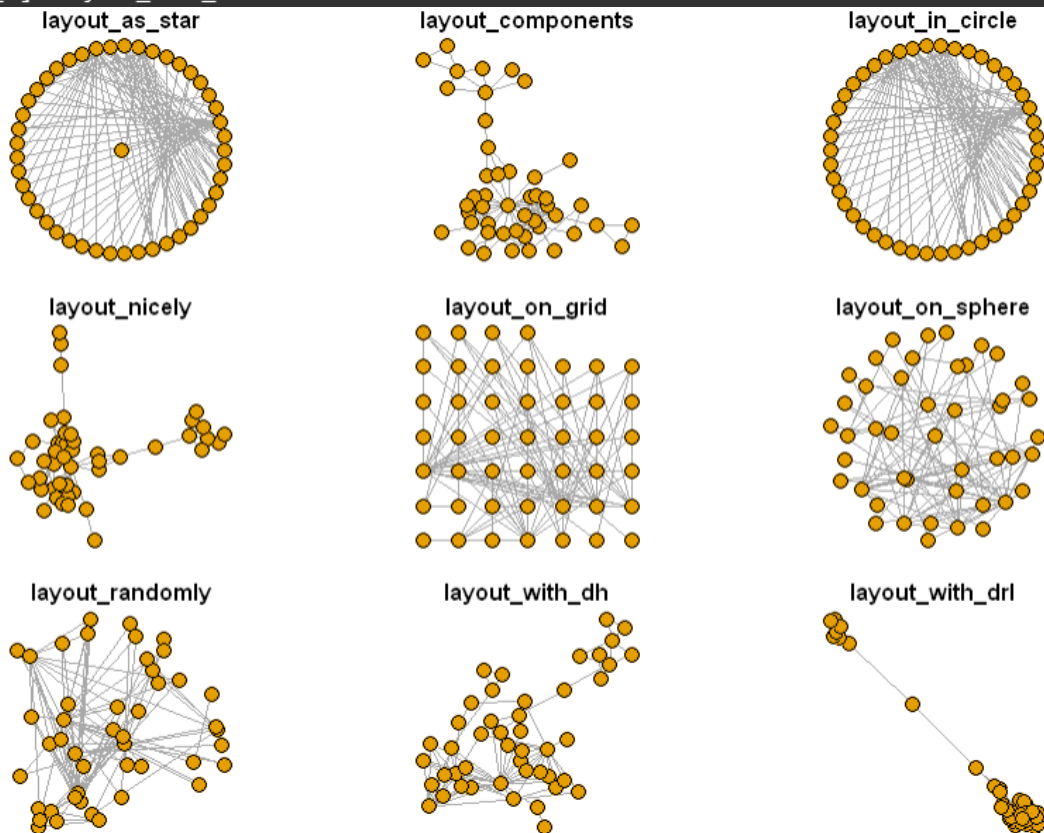


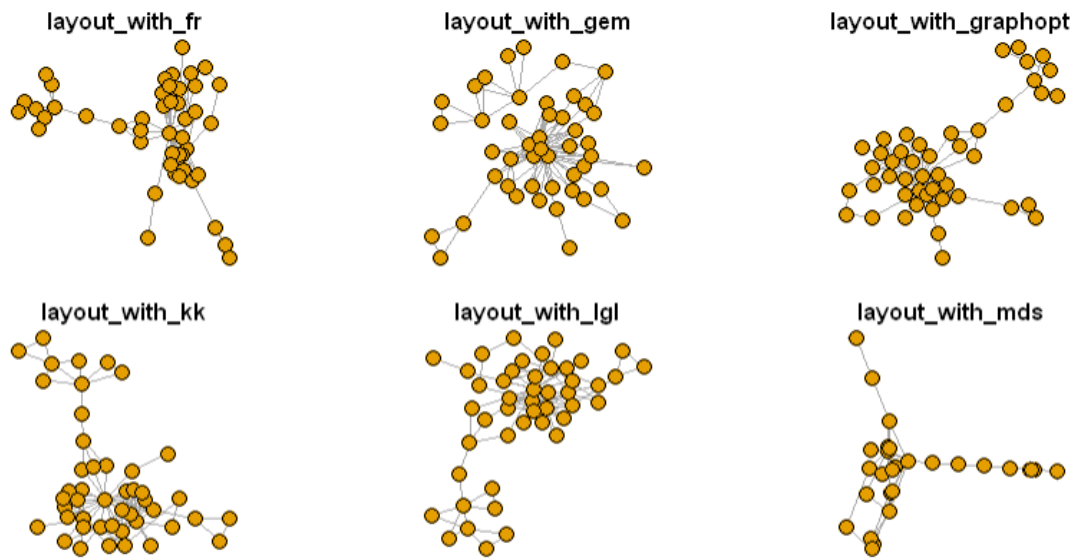
Part 4 Explore Functions

Igraph is a powerful package which contains over 700 functions for graph plotting. I chose to explore and exercise by 11 of them.

- Look at more available layouts in igraph

```
> layouts = grep("^layout_", ls("package:igraph"), value=TRUE)[-1]
> # Remove layouts that do not apply to our graph
> layouts = layouts[!grepl("bipartite|merge|norm|sugiyama|tree", layouts)]
> par(mfrow=c(3,3), mar=c(1,1,1))
> for (layout in layouts) {
+   print(layout)
+   l = do.call(layout, list(graph.sim))
+   plot(graph.sim, edge.arrow.mode=0, layout=l, main=layout, vertex.label = NA) }
[1] "layout_as_star"
[1] "layout_components"
[1] "layout_in_circle"
[1] "layout_nicely"
[1] "layout_on_grid"
[1] "layout_on_sphere"
[1] "layout_randomly"
[1] "layout_with_dh"
[1] "layout_with_drl"
[1] "layout_with_fr"
[1] "layout_with_gem"
[1] "layout_with_graphopt"
[1] "layout_with_kk"
[1] "layout_with_lgl"
[1] "layout_with_mds"
```



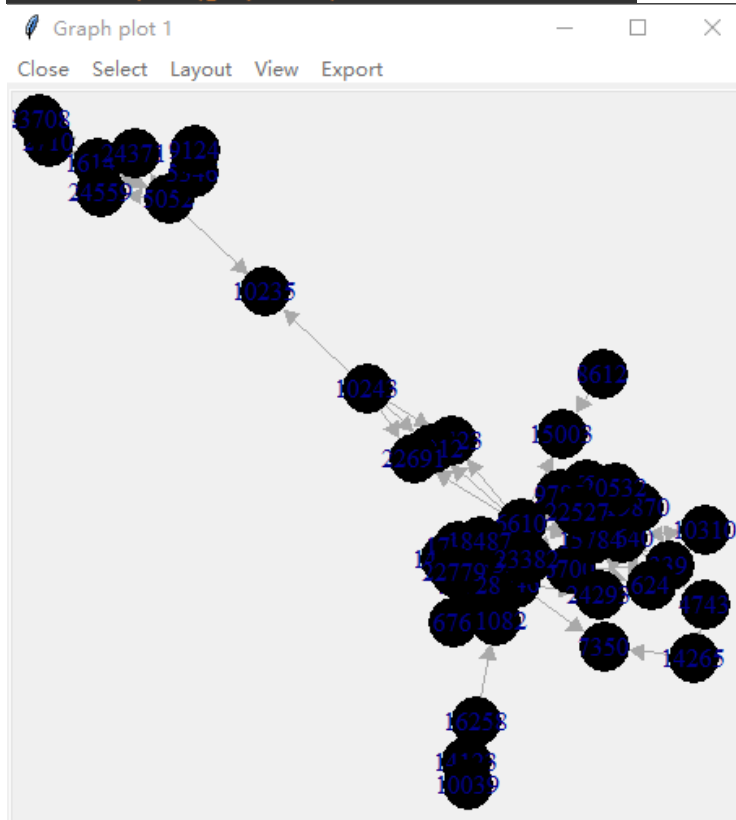


- Reset the graphical parameters

```
> dev.off()
null device
      1
> par(mfrow=c(1,1))
```

- Interact with the plotting of networks

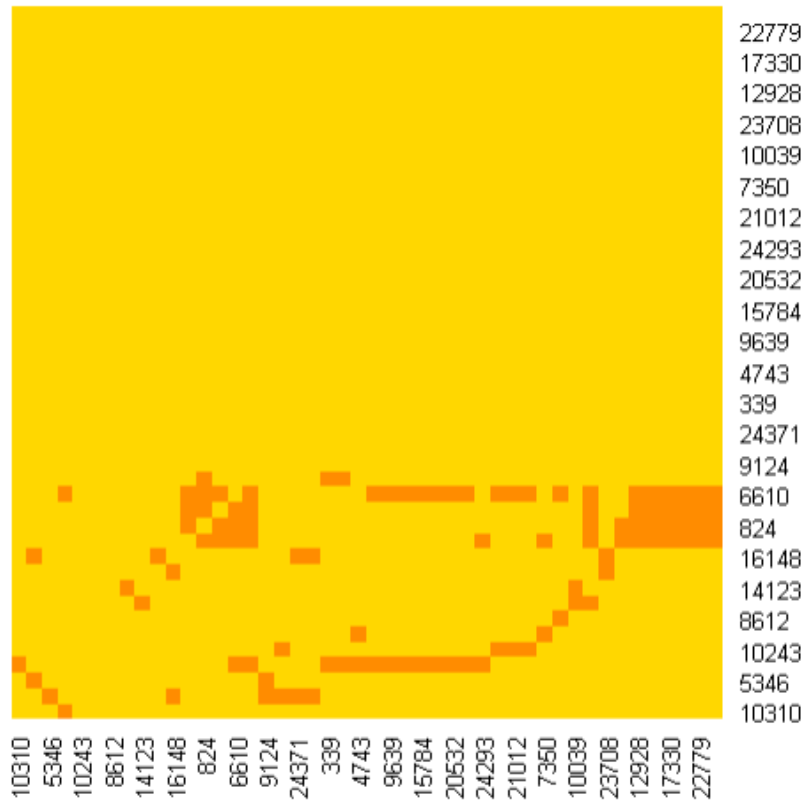
```
#tkid is the id of the tkplot that will open
tkid = tkplot(graph.sim)
```



```
> # grab the coordinates from tkplot
> tkplot.getcoords(tkid)
      [,1]      [,2]
[1,] 430.00000 157.28386
[2,] 100.22101 360.74600
[3,] 114.70852 376.62594
[4,] 379.40899 151.92319
[5,] 221.79991 243.92975
[6,] 423.12877  77.64722
[7,] 366.50639 252.82990
[8,] 289.30982  38.90435
[9,] 283.20887  13.70336
[10,]  25.59058 395.64810
[11,]  55.52754 383.49962
[12,] 312.59872 124.28955
[13,] 301.66005 129.76949
[14,] 300.08468 136.47132
[15,] 316.92797 160.52278
[16,] 347.15917 132.89946
[17,] 116.24348 391.18442
[18,] 159.49564 303.56805
[19,]  78.85337 388.80169
[20,]  58.26132 364.65567
[21,] 407.83971 135.08751
[22,] 396.73725 123.49188
[23,] 429.86180 111.19825
[24,] 371.38786 170.42204
[25,] 356.86588 185.18401
[26,] 340.65845 179.46412
[27,] 359.64920 150.96960
[28,] 388.56324 170.52054
[29,] 375.16029 183.08621
[30,] 351.97997 167.55718
[31,] 365.03762 116.66337
[32,] 273.70866 212.07237
[33,] 261.86309 206.77228
[34,] 250.71904 200.88233
[35,] 367.82546  85.38216
[36,] 342.43621 215.95485
[37,] 283.85098  0.00000
[38,] 301.05211 101.36903
[39,]  20.00000 410.00000
[40,] 274.73991  99.81228
[41,] 284.69661 122.63150
[42,] 270.30168 139.39674
[43,] 277.76006 147.24872
[44,] 291.98137 150.30532
[45,] 275.90481 131.27425
[46,] 320.62322 138.50694
```

- Create a heatmap of the network matrix

```
> pal = colorRampPalette(c("gold", "dark orange"))
> heatmap(graph.sim.adj, Rowv = NA, Colv = NA, col = pal(5), scale="none", margins=c(10,10))
```



- Reciprocity: the proportion of reciprocated ties

```
> reciprocity(graph.sim)
[1] 0.247619
```

- Transitivity: which is calculated according to two different ways: global and local

```
> #global - ratio of triangles (direction disregarded) to connected triples.
> transitivity(graph.sim, type="global")
[1] 0.2982456
>
> #local - ratio of triangles to connected triples each vertex is part of.
> transitivity(graph.sim, type="local")
Transitivity works on undirected graphs only. The result might be incorrect if the graph includes mutual edge pairs or multiple edges between the same pair of vertices. igraph 1.3.0 and later will treat this as an error.
[1] 0.00000000 0.10714286 0.33333333 0.08333333 0.00000000 0.00000000 NaN 0.16666667 0.33333333
[10] 0.33333333 0.14285714 0.20588235 0.23333333 0.29670330 0.09971510 0.32142857 1.00000000 0.00000000
[19] 1.00000000 1.00000000 1.00000000 1.00000000 0.00000000 1.00000000 1.00000000 1.00000000 1.00000000
[28] 1.00000000 1.00000000 1.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
[37] 1.00000000 0.60000000 1.00000000 1.00000000 1.00000000 1.00000000 1.00000000 1.00000000 1.00000000
[46] 1.00000000
```

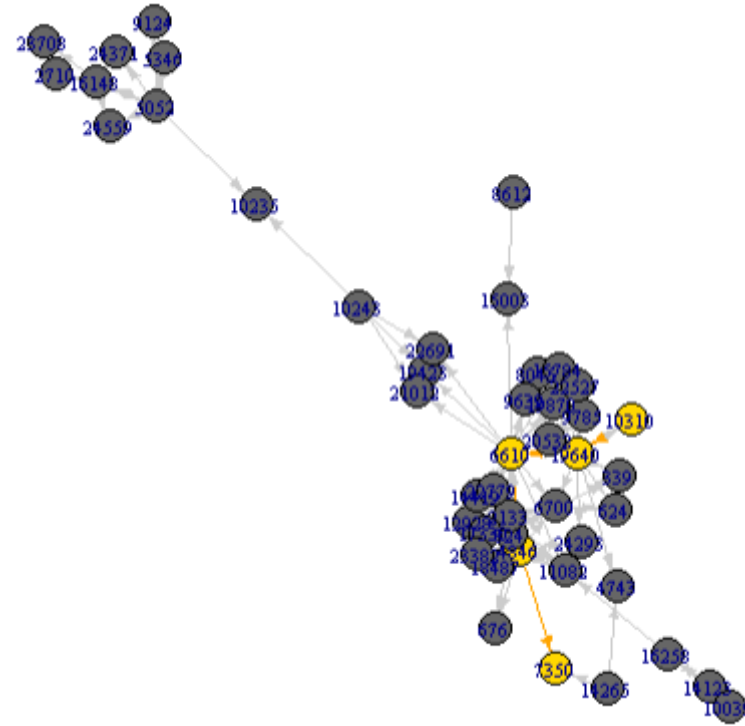
- get_diameter()

#In igraph, diameter() returns the distance, while get_diameter() returns the nodes along the first found path of that distance.

```
> #In igraph, diameter() returns the distance, while get_diameter() returns the nodes along the first found path of that distance.
> diam = get_diameter(graph.sim, directed=T)
> diam
+ 5/46 vertices, named, from 4df0ba4:
[1] 10310 19640 6610 4846 7350
```

- Color nodes along the diameter:

```
> vcol = rep("gray40", vcount(graph.sim))
> vcol[diam] = "gold"
> ecol = rep("gray80", ecount(graph.sim))
> ecol[E(graph.sim, path=diam)] = "orange"
> ew = rep(2, ecount(graph.sim))
> ew[E(graph.sim, path=diam)] = 4
> plot(graph.sim, vertex.color=vcol, edge.color=ecol, edge.arrow.size=0.3, vertex.size=10, vertex.label.cex=0.5)
```



- Values of the first eigenvector of the graph matrix.

```
> centr_eigen(graph.sim, directed=T, normalized=T)
$vector
[1] 0.07331048 0.00000000 0.00000000 0.24902065 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
[10] 0.00000000 0.00000000 0.71592474 0.94336314 0.71592474 0.77256160 1.00000000 0.00000000 0.00000000
[19] 0.00000000 0.00000000 0.36770565 0.36770565 0.07331048 0.30074888 0.30074888 0.30074888 0.30074888
[28] 0.30074888 0.30074888 0.30074888 0.28407526 0.22743840 0.22743840 0.22743840 0.21076478 0.22743840
[37] 0.00000000 0.92668952 0.00000000 0.48848634 0.92668952 0.92668952 0.92668952 0.92668952 0.92668952
[46] 0.92668952

$value
[1] 3.396795
```

<code>\$value</code> [1] 3.396795	<code>\$options\$start</code> [1] 1
<code>\$options\$bmatt</code> [1] "I"	<code>\$options\$sigma</code> [1] 0
<code>\$options\$n</code> [1] 46	<code>\$options\$sigma1</code> [1] 0
<code>\$options\$which</code> [1] "LR"	<code>\$options\$info</code> [1] 0
<code>\$options\$nev</code> [1] 1	<code>\$options\$iter</code> [1] 1
<code>\$options\$tol</code> [1] 0	<code>\$options\$nconv</code> [1] 1
<code>\$options\$ncv</code> [1] 0	<code>\$options\$numop</code> [1] 20
<code>\$options\$ldv</code> [1] 0	<code>\$options\$numopb</code> [1] 0
<code>\$options\$ishift</code> [1] 1	<code>\$options\$numreo</code> [1] 18
<code>\$options\$maxiter</code> [1] 1000	
<code>\$options\$nb</code> [1] 1	<code>\$centralization</code> [1] 0.6719116
<code>\$options\$mode</code> [1] 1	<code>\$theoretical_max</code> [1] 45

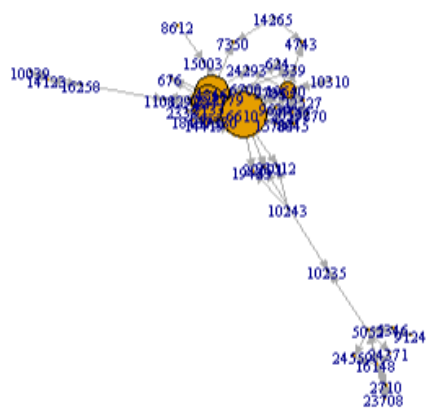
- Average path length

```
> mean_distance(graph.sim, directed=T)
[1] 1.780392
```

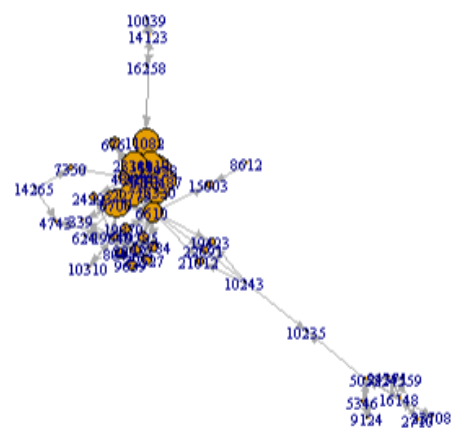
- Hubs Score (contain a large number of outgoing links) and Authorities Score (contain many incoming links from hubs)

```
> plot(graph.sim, vertex.size=hs*25, main="Hubs", edge.arrow.size=0.3, vertex.label.cex=0.5)
> plot(graph.sim, vertex.size=as*15, main="Authorities", edge.arrow.size=0.3, vertex.label.cex=0.5)
> hs <- hub_score(graph.sim, weights=NA)$vector
> as <- authority_score(graph.sim, weights=NA)$vector
> par(mfrow=c(1,2))
> plot(graph.sim, vertex.size=hs*25, main="Hubs", edge.arrow.size=0.3, vertex.label.cex=0.5)
> plot(graph.sim, vertex.size=as*15, main="Authorities", edge.arrow.size=0.3, vertex.label.cex=0.5)
```

Hubs



Authorities



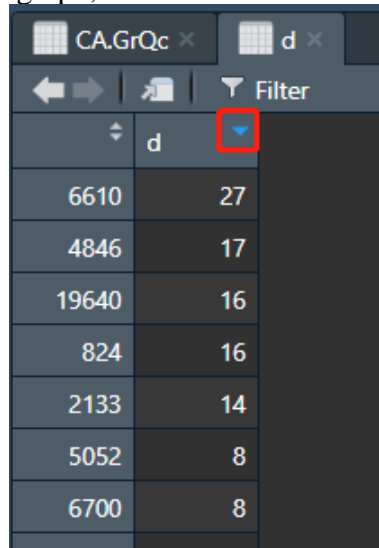
Part 5 Determine Several Technical Values

(a) central node(s)

Code Implementation:

```
d = igraph::degree(graph.sim)
d = as.data.frame(d)
```

The central node is the nodes with the high number of degrees in the graph. For this question, I took the degrees of each node and used the sorting tool within R to find the largest value. For our graph, the central node is 6610.



	d
6610	27
4846	17
19640	16
824	16
2133	14
5052	8
6700	8

Figure 6. Dataframe of Degree

(b) longest path(s)

Code Implementation:

```
igraph::diameter(graph.sim)
diam = get_diameter(graph.sim, directed=T)
diam
```

I can get the length of the longest path simply by using the diameter function in igraph. I can also use get_diameter to get the nodes that follow this longest path. For this graph, I have a longest path of length 4 using the following sequence of nodes shown in Figure 7.

```
> igraph::diameter(graph.sim)
[1] 4
> diam = get_diameter(graph.sim, directed=T)
> diam
+ 5/46 vertices, named, from ca05217:
[1] 10310 19640 6610 4846 7350
```

Figure 7: Longest Path Calculation

(c) largest clique(s)

Code Implementation:

```
vcol = rep("grey80", vcount(graph.sim))
vcol[unlist(largest_cliques(graph.sim))] = "gold"
plot(as.undirected(graph.sim), vertex.color=vcol, vertex.label.cex=0.5, vertex.size=15)
```

Plotting Result:

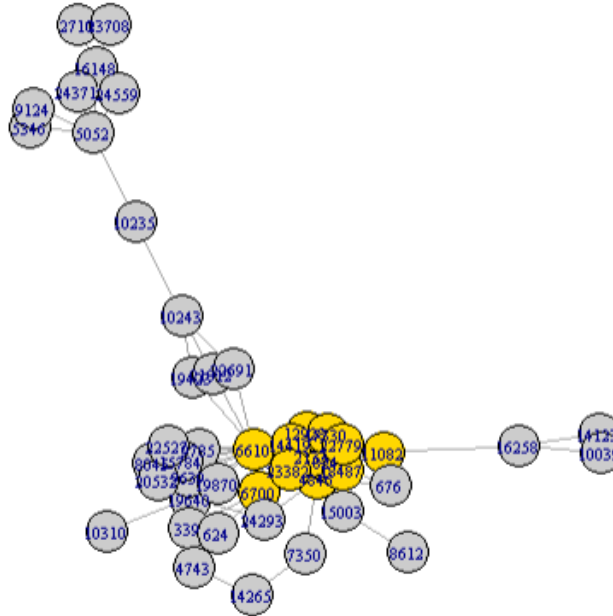


Figure 8: Largest Clique (Gold)

(d) ego(s)

I can find the biggest immediate neighborhood for all nodes by ego() function, here I only show the result for our central node: 6610.

```
> ego(graph.sim, nodes=V(graph.sim)["6610"])
[[1]]
+ 24/46 vertices, named, from 4df0ba4:
 [1] 6610 19640 4846 824 2133 6700 8045 9639 9785 15784 19870 20532 22527 19423 21012 22691
[17] 15003 11082 12928 14419 17330 18487 22779 23382
```

Figure 9: Ego for node 6610

(e) power centrality

power centrality takes a graph (data) and returns the Boncich power centralities of positions (selected by nodes).

Code Implementation:

```
pc = power_centrality(graph.sim, exponent = 0.9)
pc_df = as.data.frame(pc)
pc_df <- cbind(node = rownames(pc_df), pc_df)
rownames(pc_df) <- 1:nrow(pc_df)
pc_df[which.max(pc_df$pc),]
```

Using `power_centrality()` from the `igraph` package, I am able to calculate the power centrality for each node in the graph, as well as the central node. Note: I must set `exponent = 0.9` to avoid an error

```
> power_centrality(graph.sim, exponent = 0.9)
10310      5052      5346      19640      10243      14265
4.202730e+00 -2.763556e-01 -1.924528e-01 4.638441e+00 1.125344e-01 5.626719e-02
8612      16258      14123      2710      16148      4846
2.813360e-02 7.107435e-01 6.959363e-01 -2.156824e-01 -3.021662e-01 -9.866892e-01
824      2133      6610      6700      9124      10235
-1.016303e+00 -1.031111e+00 1.343731e+00 -8.302723e-01 0.000000e+00 0.000000e+00
24371      24559      339      624      4743      8045
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
9639      9785      15784      19870      20532      22527
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
24293      19423      21012      22691      7350      15003
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
10039      11082      23708      676      12928      14419
0.000000e+00 0.000000e+00 2.922532e-17 0.000000e+00 0.000000e+00 0.000000e+00
17330      18487      22779      23382
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
```

Figure 10a: Power Centrality for All Nodes

```
> pc_df[which.max(pc_df$pc),]
      node      pc
4 19640 4.638441
```

Figure 10b: Central Node using Power Centrality

Part 6 Discussion and Conclusion

For this project, I loaded a large data set of authors and co-authors into a graph using the igraph package in R. Since this dataset contained over 5000 nodes and 14000 edges, it was clear I would need to implement some graph simplification strategies.

I applied multiple strategies before successfully completing our simplification. Firstly, I ordered my nodes based on frequency and selected the top 300 authors. This strategy helped simplify our graph, but I am concerned this would not have enough randomness within the sample. Instead, I selected 400 nodes randomly, and then reduced this number by mandating the nodes have at least one degree. This strategy performed better, but resulted in a graph containing two subgraphs, and I preferred to work with just one. The final strategy had me performing degree removal first, dropping all nodes with less than 20 degrees, then I took the first 200 nodes of the resulting graph and dropped the remaining nodes with less than 2 degrees. The result was the graph shown in Figure 5, which contains 46 vertices, 105 edges, and no subgraphs, this is the graph I used for the remainder of my analysis

The rest of our analysis includes practicing with the igraph functions mentioned in the rubric, as well as some others. This exploration helped me become much more comfortable working with, visualizing, and understanding graph. The functions made it easier for me to identify and label cliques that occurred within the dataset. I also learned how to use different measurements for centrality, including closeness, betweenness, alpha, and power centrality. Finally, I learned the difference between directional measurements, such as hubs and authorities, and how to represent them in my graph.