Project 1 Report

Group #9

Zhengyang Tang, Li Zhu, Jiayi Xu, Zhuolin Yang

**3. Loading map**

We load the file and parse it into a map using

#attention: load sna first, then load igraph

#read data,need to set working directory first

edges1 <- read.table('roadNet-CA.txt')

em <- as.matrix(edges1) #save data in matrix

v1 <- em[,1] #Save the first column of data to v1

v2 <- em[,2] #Save the second column of data to v2

relations <- data.frame(from=v1,to=v2)

g<-graph.data.frame(relations,directed=TRUE)

Now g is our graph

**\* Map size reduction**

Firstly, we tried to reduce the map size by the method in the announcement. We deleted vertices with degree lower than or equal to certain number k.

The map size is acceptable when k is 10.

g <- g - V(g)[degree(g)<=10]

There were 2094 vertices and 184 edges left. Apparently, this method of map size reduction is not good since the graph is much too sparse after reduction. We decided to try another method.

Since deleting vertices with certain degree will result in drop of degree of vertices not being deleted and kept most of the remaining vertices connected, we tried to iteratively delete vertices with a very low degree. (here we delete vertices with degree lower than or equals to 4)

i <- 0

while(vcount(g)>5000){

g <- g - V(g)[igraph::degree(g)<=4]

i <- i+1

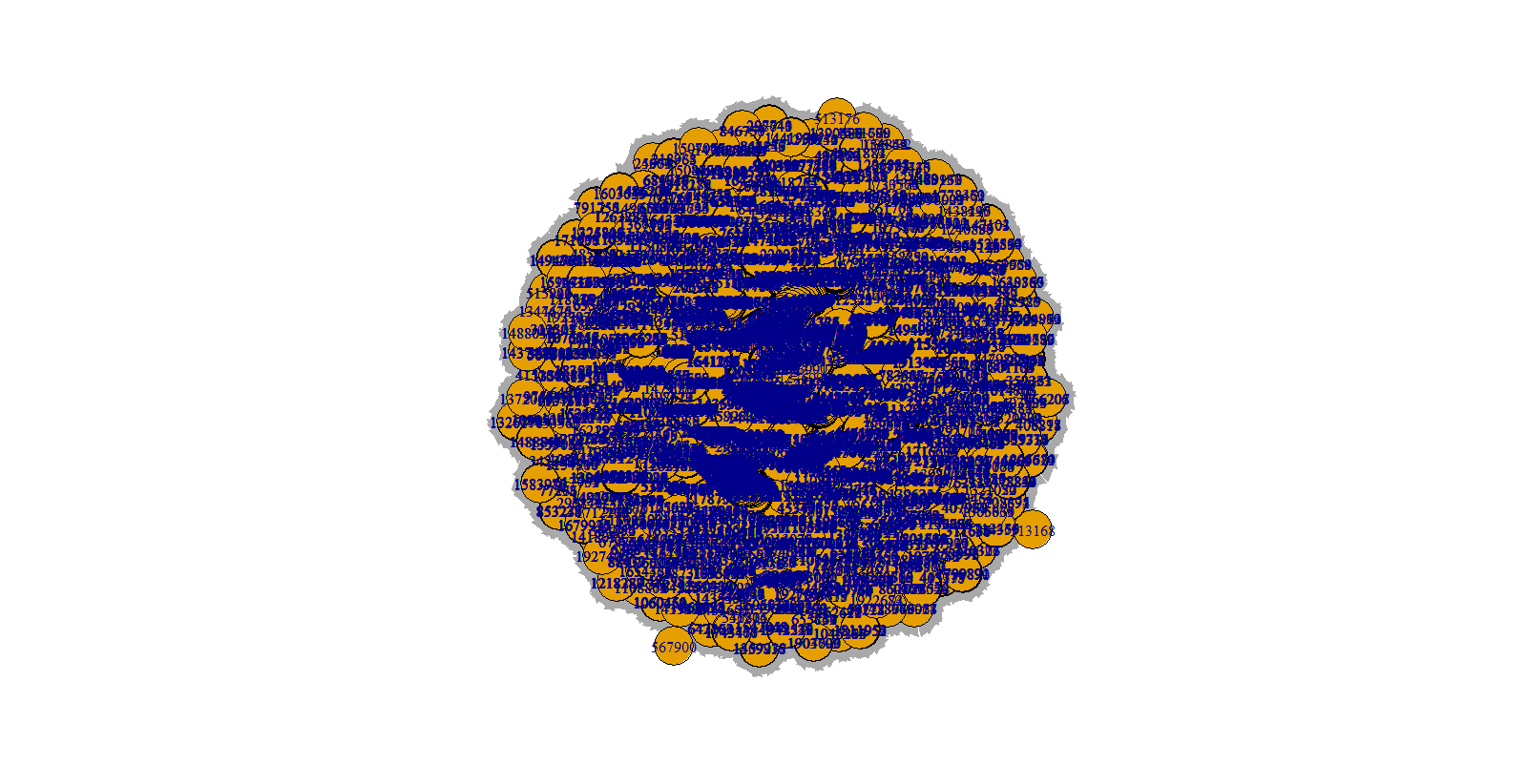
}

We keep deleting vertices whose degree lower or equals to 4 until there are less than 5000 of them remains. (graph with 5000 nodes is properly small for plotting) It takes 56 iterations to reduce the map size. Now the map has 4947 vertices and 16382 edges. This method is obviously better than the previous one.

\* if we keep the loop running, the number of vertices left will converge to a number between 4000 and 5000.

**Plotting**

plot(g)



4.

5.

#15 other functions in igraph

bipartite.mapping(g)

cliques(g)

centralization.betweenness(g)

centralization.closeness(g)

articulation.points(g)

average.path.length(g)

automorphisms(g)

blockGraphs(g)

cohesion(g)

coreness(g)

efficiency(graph.matrix)

gtrans(graph.matrix)

isolates(graph.matrix)

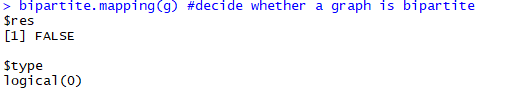
loadcent(graph.matrix)

g.reach<-reachability(graph.matrix)

stresscent(graph.matrix)

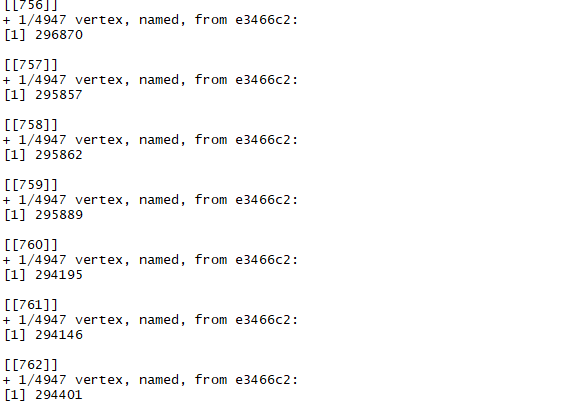
1)bipartite.mapping(g)

decide whether a graph is bipartite



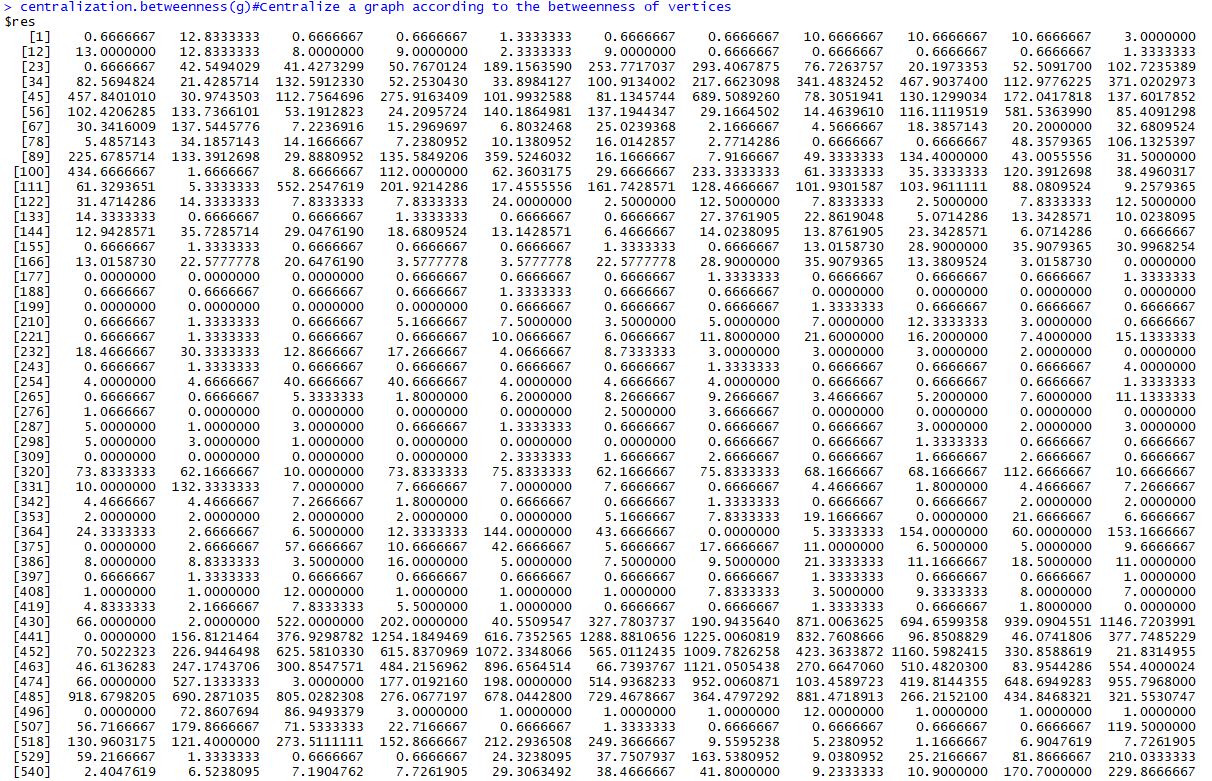
2) cliques(g)

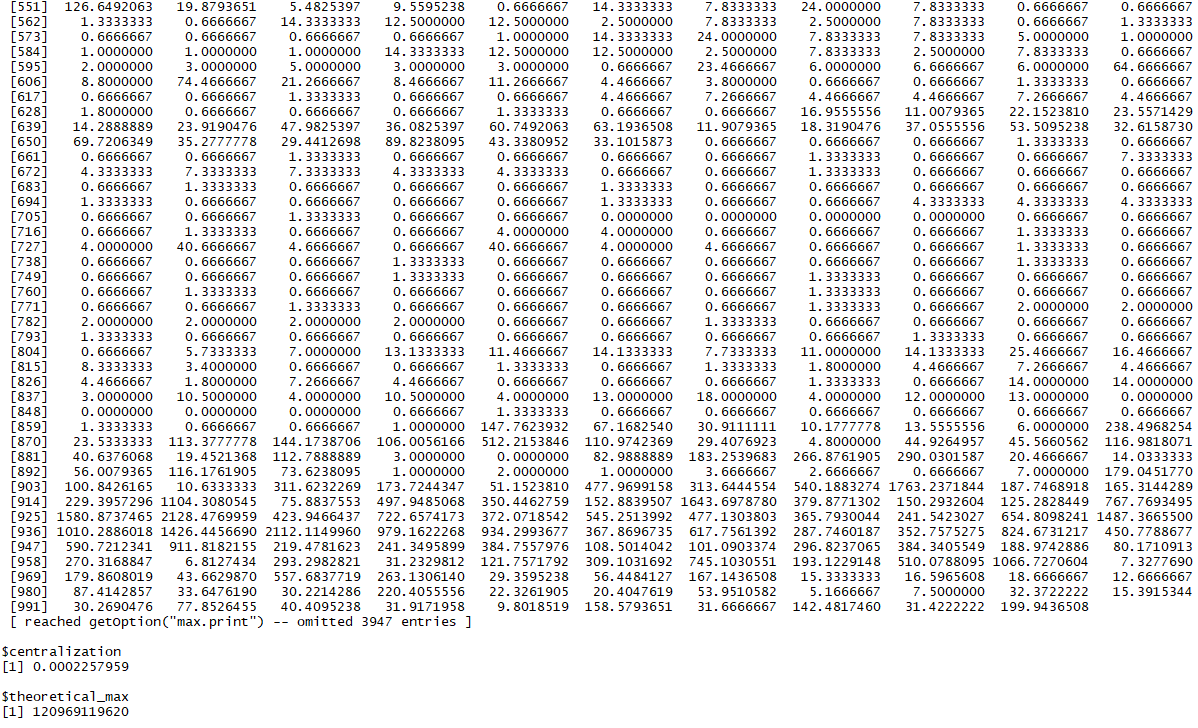
The functions find cliques



3-4) centralization.betweenness(g)

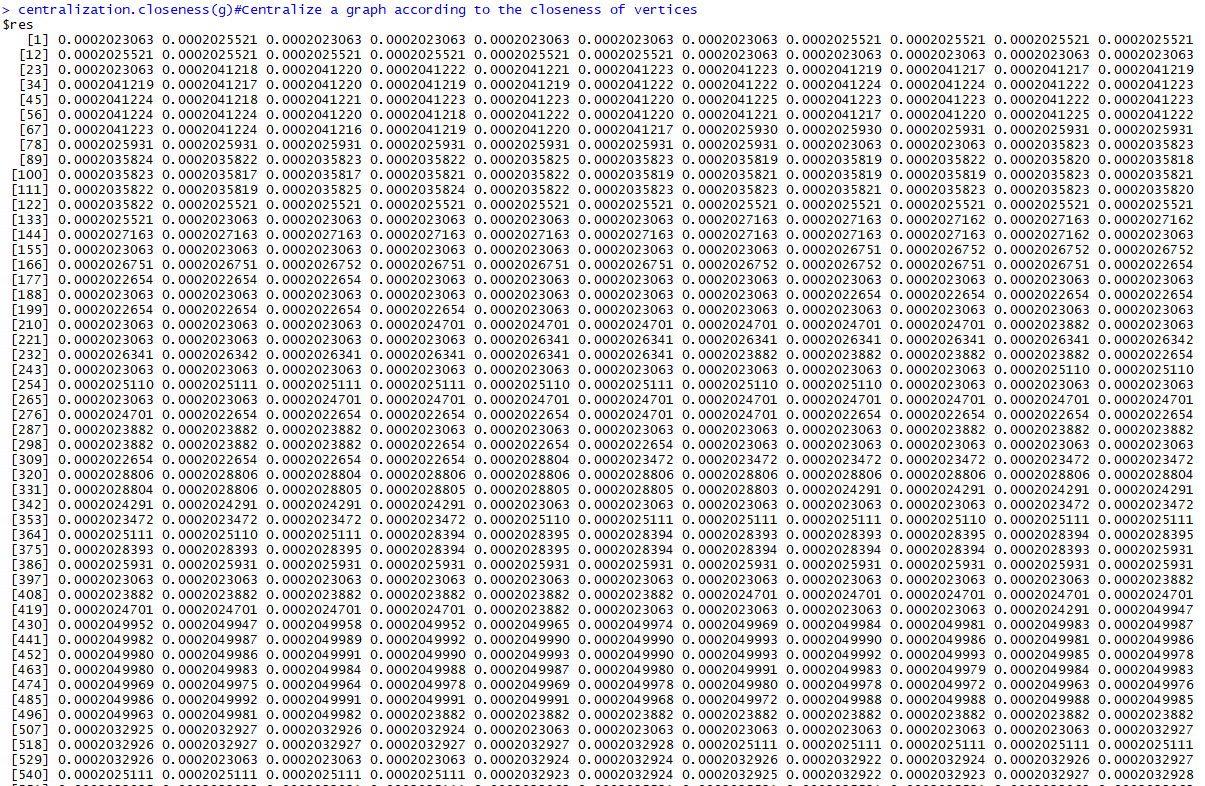
Centralize a graph according to the betweenness of vertices

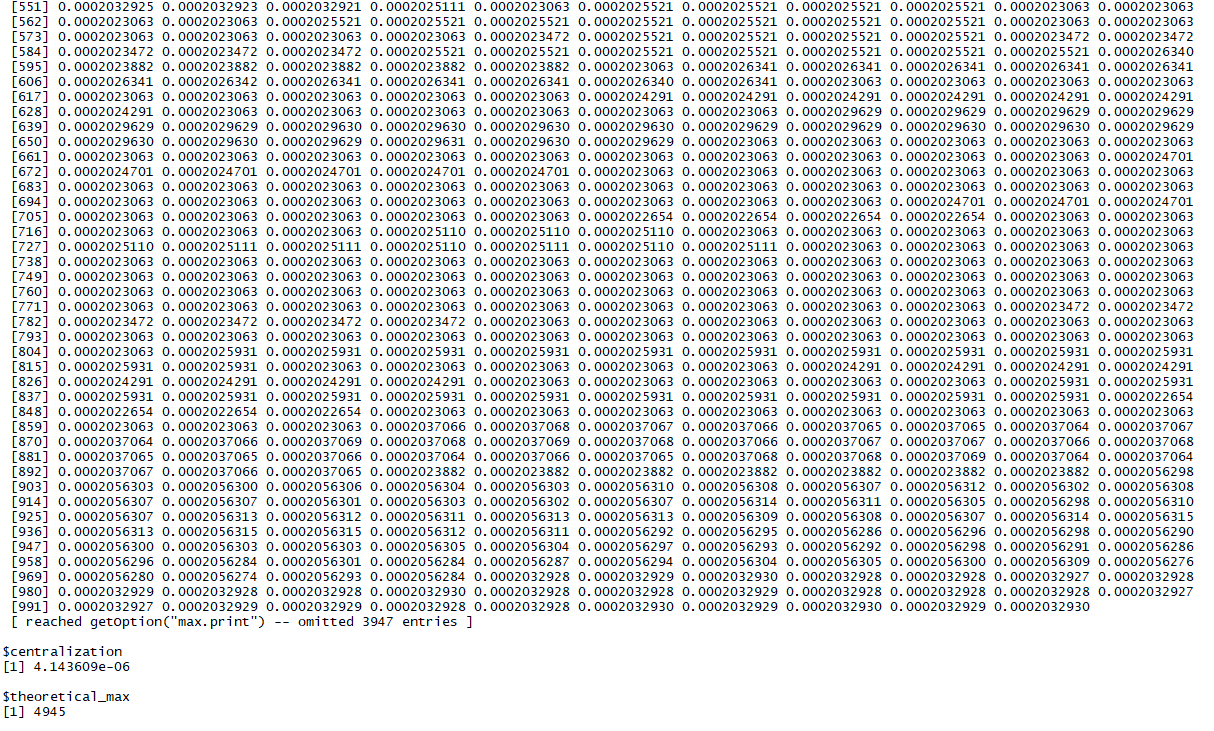




centralization.closeness(g)

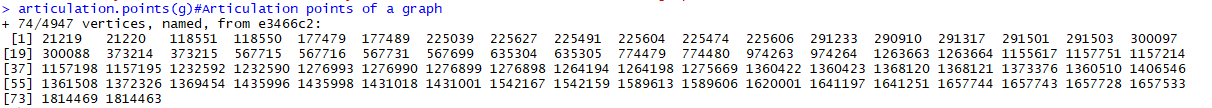
Centralize a graph according to the closeness of vertices





5) articulation.points(g)

Articulation points of a graph



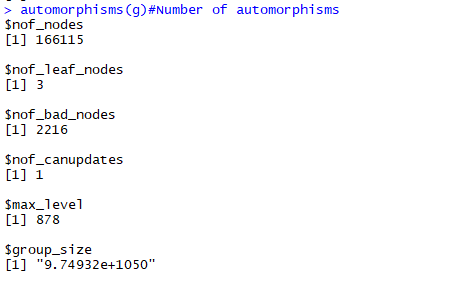
6) average.path.length(g)

Shortest (directed or undirected) paths between vertices



7) automorphisms(g)

Number of automorphisms



8) blockGraphs(g)

Calculate Cohesive Blocks



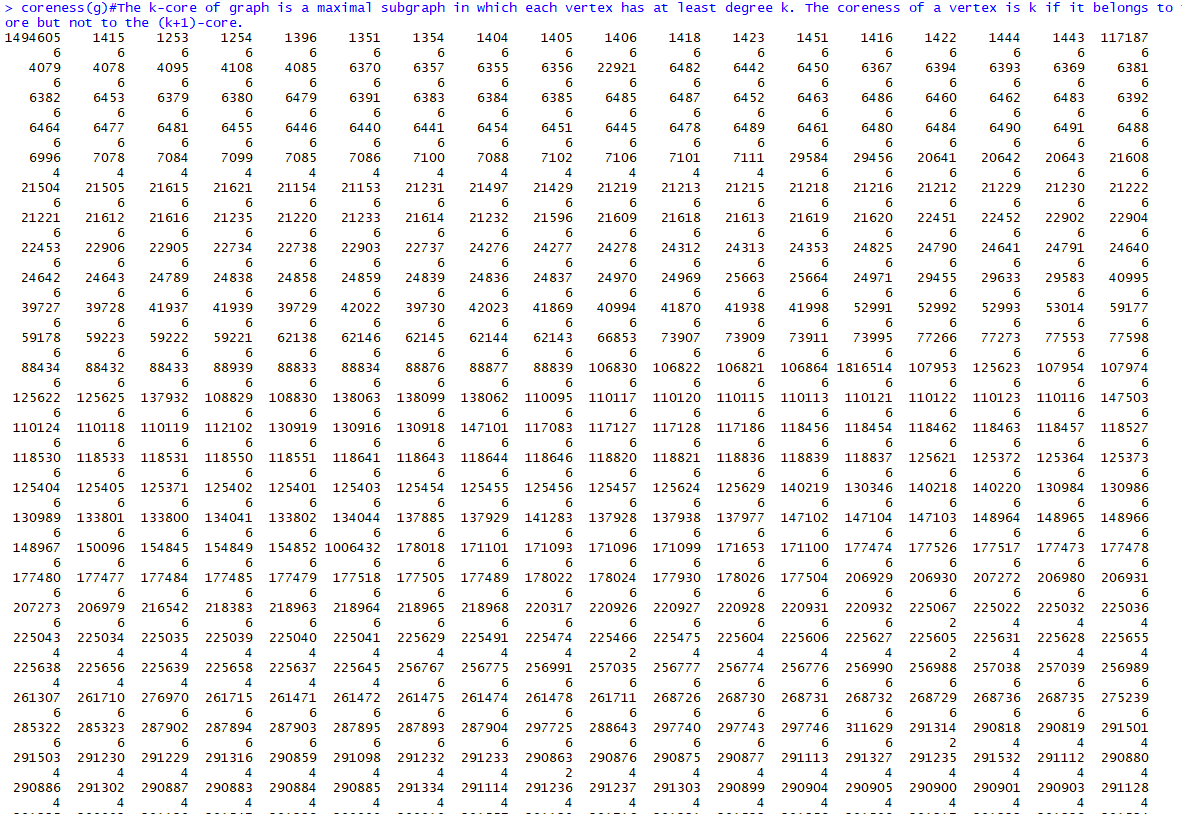
9) cohesion(g)

Vertex connectivity



10) coreness(g)

The k-core of graph is a maximal subgraph in which each vertex has at least degree k. The coreness of a vertex is k if it belongs to the k-core but not to the (k+1)-core.



11) efficiency(graph.matrix)

Compute Graph Efficiency Scores



12) gtrans(graph.matrix)

Compute the Transitivity of an Input Graph or Graph Stack



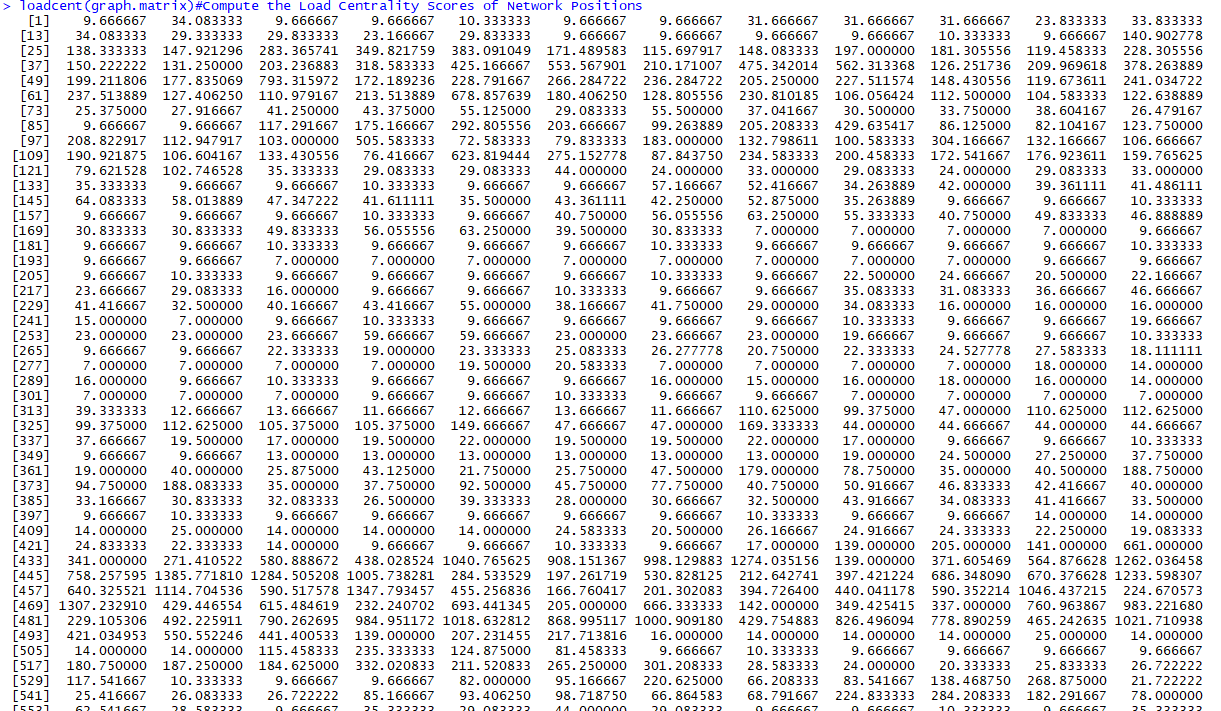
13) isolates(graph.matrix)

List the Isolates in a Graph or Graph Stack



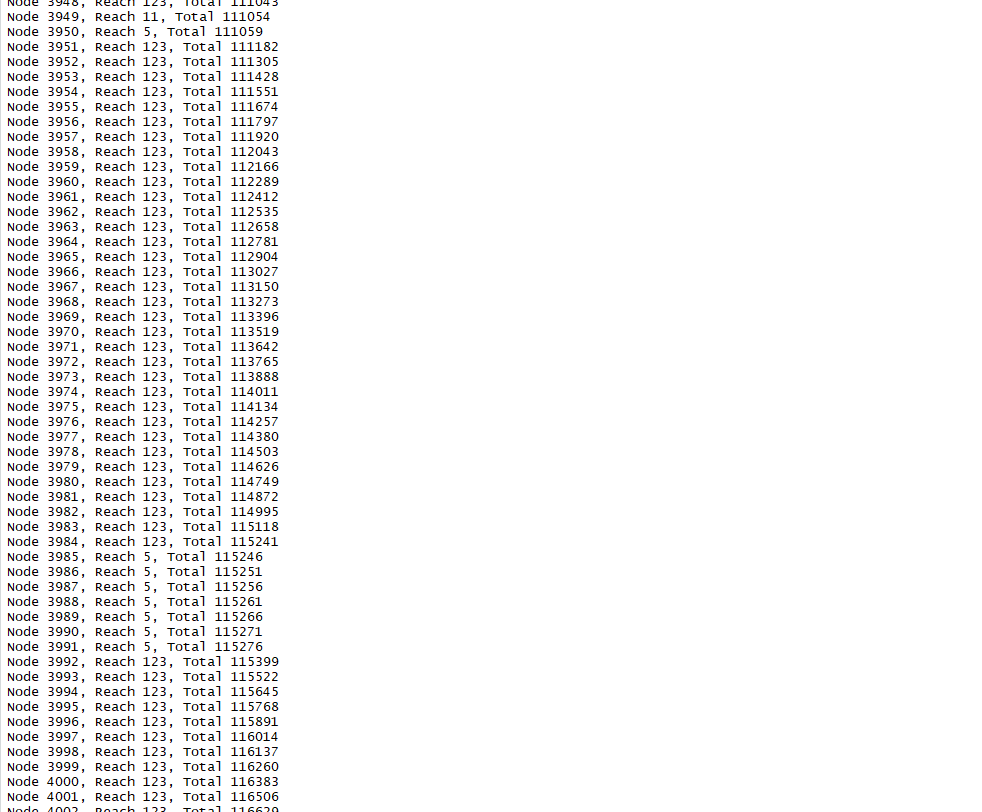
14) loadcent(graph.matrix)

Compute the Load Centrality Scores of Network Positions



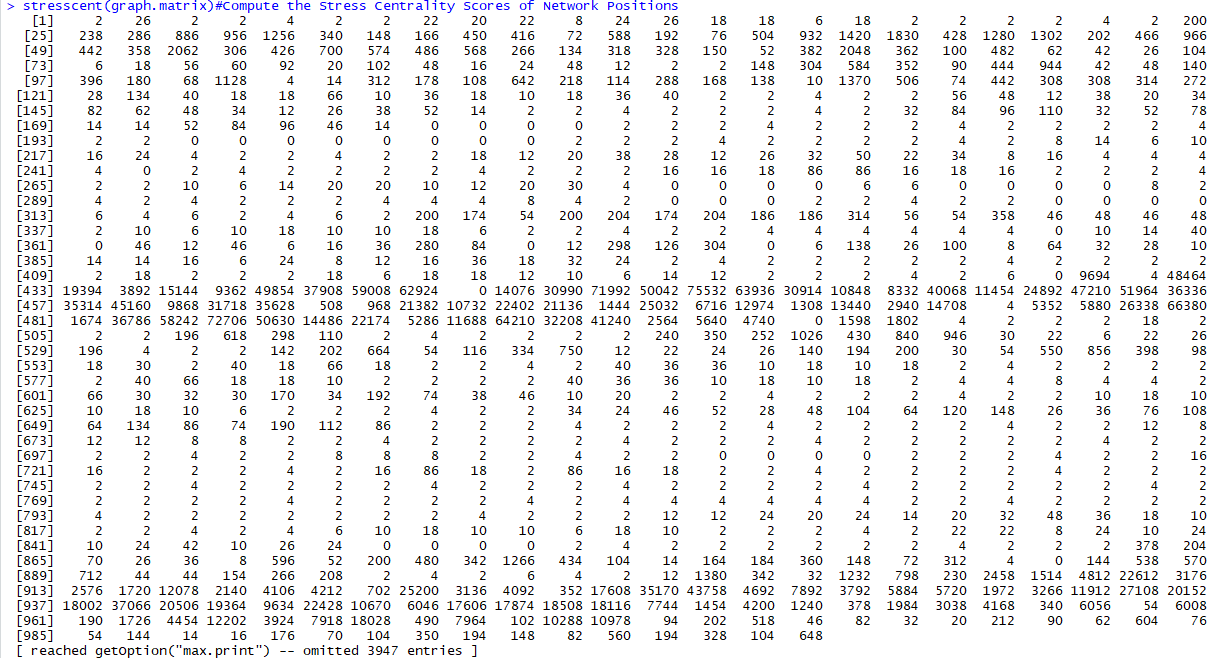
15) g.reach<-reachability(graph.matrix)

producing the associated reachability matrices



16) stresscent(graph.matrix)

Compute the Stress Centrality Scores of Network Positions



6.

7.

Step 1: find number of all nodes of order 0 to 3 for all nodes

vc<-ego\_size(g, 3, V(g),mindist = 0)

step 2: order the number of 3-reachable nodes for each nodes, and select top 20 indexes of them

od are the indexes of target nodes

od<-order(vc,decreasing = TRUE)[1:20]

step 3: create list of 3-reachable nodes of each nodes

V(g)[c(od)] is the vertices of each index in od

topGraphs <- ego(g, 3, V(g)[c(od)],mindist = 0)

step 4: iterate through each pair of list of nodes, and find their intersections

for(i in 1:19){

for(j in i:20){

print(intersect(topGraphs[i], topGraphs[j]))

}

}

The result indicates there are 19 pair of 3-reachable nodes have intersection.

