Project 1

R and Graph Analytics

Our approach to this project centered around being able to work with/preserve as much of the given data as possible. Computationally, we set up a parallel backend to maximize our computing efficiency, especially during plotting. We also used Base R’s set.seed function to standardize our results across runs. Reading in the provided data was a trivial task, which we accomplished using Base R’s read.table function. Other environment setup tasks centered around writing functions that allowed our work to be readable (lessening repeated code) and easily updated (parameterizing arguments).

Overall, the major problems we had were mechanical: long compute times, as well as coercing the input data into structures accepted by each function. The long compute times improved with our implementation of parallelizing many of the compute tasks, as well as with each successive simplification of the graph. The understanding and use of igraph’s and sna’s provided functions required that we spend time with each library’s respective documentation to understand what the input data should look like, as well as what each parameter did and what the defaults were, if any.

# Part 2 | Plot the unaltered network with edge labels

Here we plotted the given data set, mapping the given edge labels to their corresponding values.

Graphical user interface, text, application

Description automatically generated

We first plotted the given data set with edge labels using the defaults in the igraph plot function, getting the “blue blob” mentioned in lecture. Next, we applied a Fruchterman Reingold layout to better display the nodes and edges, as well as to get a general sense of what the unaltered, pre-simplified, graph would look like as a whole.

|  |  |
| --- | --- |
| A picture containing text  Description automatically generated |  |

# Part 3 | Simplify and apply demonstrated functions

## Simplify the plot

For the following attempts to simplify the plot, we first used igraph’s provided “simplify” function to remove multiple edges and loops.

### Version 1

Our first attempt to simplify the plot was to find the average degree a typical node in this dataset had. We then removed any nodes whose degree was below the average. This resulted in a still very unreadable plot.

|  |  |
| --- | --- |
| A picture containing text  Description automatically generated | # 1 - Keep nodes whose degrees are above the graph's average degree  average.degree <-  mean(igraph::degree(graph = g.simplified.0, mode = "total"))  g.simplified.1 <-  delete.vertices(graph = g.simplified.0, v = degree(g.simplified.0) < average.degree)  plot\_with\_args(g.simplified.1, "Simplify | Test 1") |

### Version 2

Our next attempt at simplifying the graph revolved around calculating the average coreness of the graph. We removed the nodes whose degree was less than the average coreness. This also resulted in an unreadable plot, but – compared to our first attempt – this attempt was an improvement. We are beginning to make out the nodes, edges, their respective labels.

|  |  |
| --- | --- |
|  | # 2 - Keep nodes above average centrality  average.coreness <- mean(coreness(graph = g.simplified.0))  g.simplified.2 <-  delete.vertices(graph = g.simplified.0,  v = coreness(g.simplified.0) < average.coreness)  plot\_with\_args(g.simplified.2, "Simplify | Test 2") |

### Version 3

Our third attempt at simplifying the graph revolved around finding the betweenness and the closeness centralities of the data points. After determining these metrics, we plotted the top 100, 50, 10, and 5 nodes iteratively. Ultimately, we chose to simplify the graph using this approach, maintaining the top 15 nodes because (1) visually, the plot was more readable and (2) the resulting subgraph appears to be a good approximation of the larger graph. Obviously, a small sample cannot be representative of the entire population, but in this first pass, we found this number to be a good place for beginning to establish a baseline for the “major players” in this network.

|  |  |
| --- | --- |
| Chart, bubble chart  Description automatically generated | # 3 - Find maximum betweenness centrality and maximum closeness centrality  # simplified\_3\_top\_  TOP\_N <- 100  betweenness.centrality <- igraph::betweenness(g.simplified.0)  top.betweeness <- find\_max\_n(betweenness.centrality, TOP\_N)  closeness.centrality <- igraph::closeness(g.simplified.0)  top.closeness <- find\_max\_n(closeness.centrality, TOP\_N)  g.simplified.3 <-  igraph::induced\_subgraph(graph = g.simplified.0, v = union(names(top.betweeness), names(top.closeness)))  plot\_with\_args(g.simplified.3, "Simplify | Test 3") |
| Scatter chart  Description automatically generated with medium confidence | TOP\_N <- 50 |
| A picture containing diagram  Description automatically generated | TOP\_N <- 10 |
| Chart, bubble chart  Description automatically generated | TOP\_N <- 5 |

## Demonstrated functions

|  |  |
| --- | --- |
| Function | Demonstration |
| igraph::V(part.3.graph) | A picture containing text  Description automatically generated |
| igraph::E(part.3.graph) | Text  Description automatically generated |
| igraph::get.adjacency(part.3.graph) | Text  Description automatically generated with low confidence |
| igraph::centr\_betw(part.3.graph) | Text  Description automatically generated with medium confidence |
| igraph::centr\_clo(part.3.graph) | Text  Description automatically generated |
| igraph::shortest\_paths(graph = part.3.graph, from = 1) | Graphical user interface, text, application  Description automatically generated |
| part.3.adjacency <- get.adjacency(part.3.graph)  sna::gden(as.matrix(part.3.adjacency)) | Text  Description automatically generated |
| igraph::edge\_density(part.3.graph) | Text  Description automatically generated |
| sna::geodist(as.matrix(part.3.adjacency)) | Text  Description automatically generated with medium confidence |
| igraph::alpha\_centrality(graph = part.3.graph, alpha = 0.99) | Text  Description automatically generated |

# Part 4 | Explore igraph functions

|  |  |
| --- | --- |
| Function | Demonstration |
| igraph::all\_shortest\_paths(graph = part.4.graph, from = 1) | Graphical user interface, text, application  Description automatically generated |
| igraph::reciprocity(part.4.graph) | Text  Description automatically generated |
| igraph::betweenness(graph = part.4.graph) | Text  Description automatically generated |
| igraph::authority\_score(graph = part.4.graph) | Graphical user interface, text  Description automatically generated with medium confidence |
| igraph::bonpow(graph = part.4.graph, exponent = 0.99) | Text  Description automatically generated |
| igraph::components(part.4.graph, c("weak")) |  |
| igraph::cluster\_walktrap(part.4.graph, steps=6) |  |
| igraph::decompose(part.4.graph, "weak") |  |
| dominator\_tree(part.4.graph, as.numeric(V(part.4.graph)['3320']), mode ="out") |  |
|  |  |

# Part 5 | Determine network characteristics

|  |  |  |
| --- | --- | --- |
|  | Function | Demonstration |
| (a) Central node(s) | ## alpha\_centrality  katz.centrality <- alpha\_centrality(part.5.graph, alpha = 0.9)  find\_max\_n(katz.centrality, TOP\_N)  ## degree centrality  degree.centrality <- igraph::degree(part.5.graph)  find\_max\_n(degree.centrality, TOP\_N)  ## betweenness centrality  betweenness.centrality <- igraph::betweenness(part.5.graph)  find\_max\_n(betweenness.centrality, TOP\_N)  ## closeness centrality  closeness.centrality <- igraph::closeness(part.5.graph)  find\_max\_n(closeness.centrality, TOP\_N)  ## eigen centrality  eigen.centrality <- igraph::eigen\_centrality(part.5.graph)  find\_max\_n(eigen.centrality$vector, TOP\_N)  ## power centrality  power.centrality <-  igraph::power\_centrality(part.5.graph, exponent = .9)  find\_max\_n(power.centrality, TOP\_N)  ## PageRank  pagerank.centrality <- igraph::page\_rank(part.5.graph)  find\_max\_n(pagerank.centrality$vector, TOP\_N)  ## eccentricity  eccentricity.centrality <- igraph::eccentricity(part.5.graph)  find\_max\_n(eccentricity.centrality, TOP\_N)  ## hubs and authorities  hubs.authorities <- igraph::hub\_score(part.5.graph)  find\_max\_n(hubs.authorities$vector, TOP\_N)  ## subgraph centrality  subgraph.centrality <- igraph::subgraph\_centrality(part.5.graph)  find\_max\_n(subgraph.centrality, TOP\_N) | Text  Description automatically generated |
| (b) longest path(s) | igraph::diameter(graph = part.5.graph, directed = TRUE) # Length  igraph::get\_diameter(graph = part.5.graph, directed = TRUE) # Nodes | Text  Description automatically generated |
| (c) largest clique(s) | igraph::max\_cliques(graph = part.5.graph) # List  igraph::count\_max\_cliques(graph = part.5.graph) # Count | Graphical user interface, text  Description automatically generated |
| (d) ego(s) | igraph::ego(part.5.graph)  igraph::ego\_size(part.5.graph) | Text  Description automatically generated with medium confidence |
| (e) power centrality | igraph::power\_centrality(graph = part.5.graph, exponent = .9) | Text  Description automatically generated with low confidence |