Network Science Homework 6

Arghya Kannadaguli (ak5357)

2025-03-06

Assignment Description: Select a real-world network, test random node-removal and degree-based node removal to break down the network, compare which method works best.

```
library(igraph)
library(tidyverse)
library(ggplot2)
```

Iceland Network Dataset

“This is a network of sexual contacts of male homosexuals in Iceland, collected in 1992.”

Data Source: The Konnect Project (http://konect.cc/networks/moreno_iceland/)

Data Import

Find a real-world network.

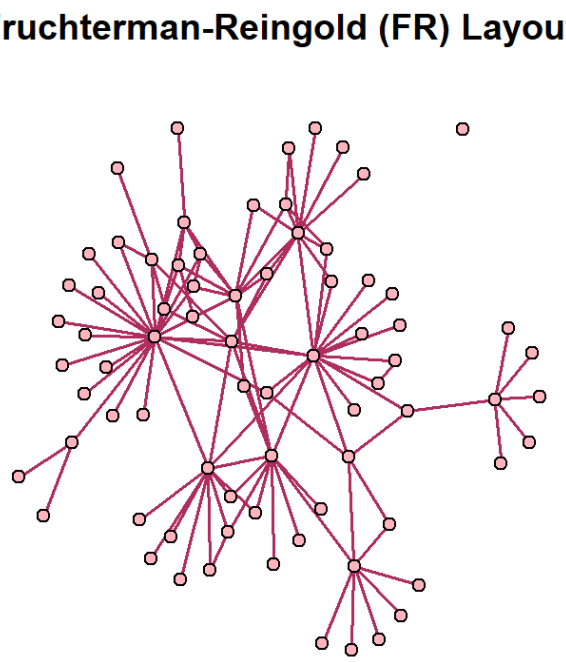
Here we are pulling in the physicians network described above.

```
iceland =
  readr::read_delim("data/iceland/out.iceland", skip = 1) |>
  janitor::remove_empty(which = "cols") |>
  janitor::clean_names() |>
  as.matrix() |>
  graph_from_edgelist()

# Save original network
og_iceland = iceland
```

Basic Analysis/Visualization

There are **75 nodes** and **113 edges** in this network. Here is a visualization of the network using the FR layout. I tried the KK layout as well, but ultimately felt FR yielded a more visually organized result.

```
plot(iceland,
     layout=layout_with_fr,
     vertex.label = NA,
     vertex.size = 5,
     vertex.color = "lightpink",
     edge.width = 1.5,
     edge.color = "maroon",
     edge.arrow.size = 0,
     main = "Fruchterman-Reingold (FR) Layout")
```



Random Node Removal

Define a function to remove nodes randomly from the graph.

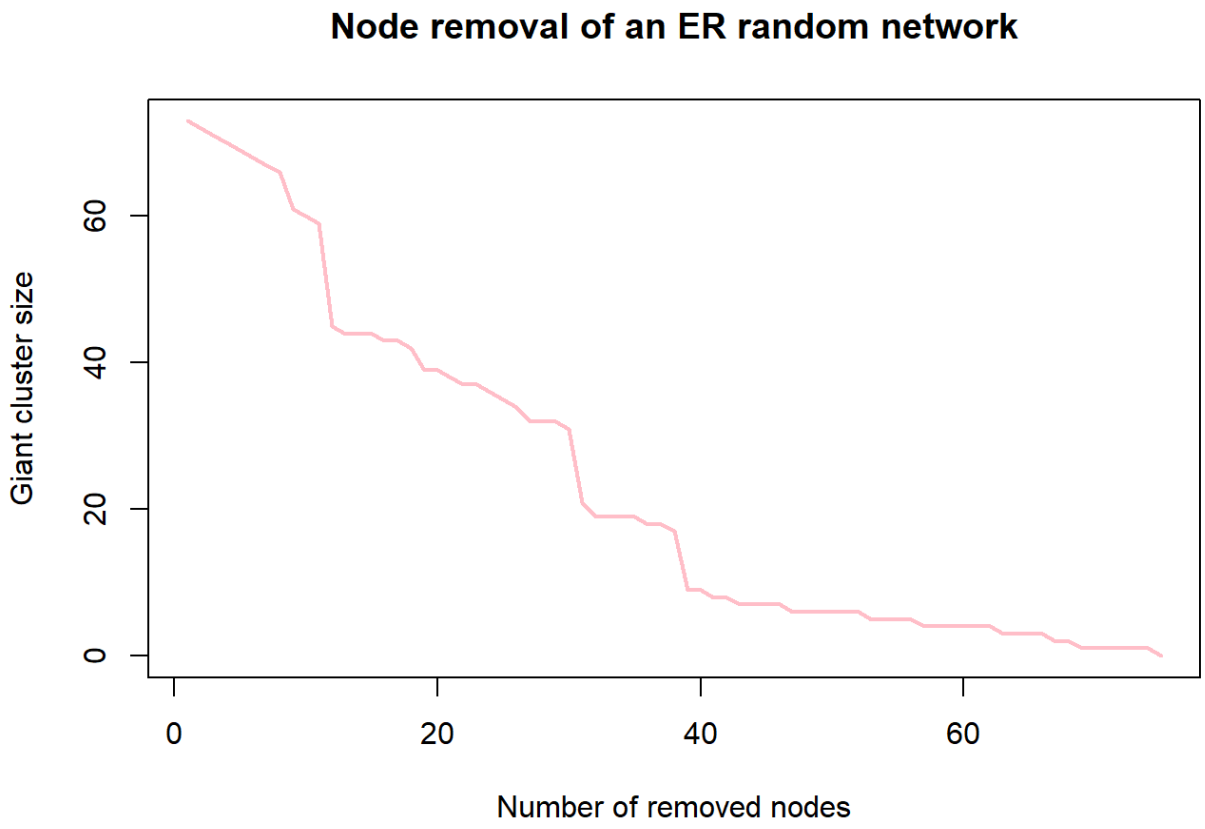
```
remove_random_nodes = function(graph, random_nodes) {
  remaining_nodes = setdiff(V(graph), random_nodes)
  induced_subgraph(graph, remaining_nodes)
}
```

Iteratively remove nodes using random removal function.

```
# Restore network to original state if needed
iceland = og_iceland

# Number of nodes in iceland network
n_nodes = vcount(iceland)

# Create empty vectors with length n_nodes
removed_nodes = numeric(n_nodes)
gc_size = numeric(n_nodes)

# Iteratively remove nodes
for(i in (1:n_nodes)){
  removed_node_count = sample(V(iceland), 1)
  removed_nodes[i] = removed_node_count
  iceland = remove_random_nodes(iceland, removed_node_count)
  comps = components(iceland)

  if(i < n_nodes){
    gc_size[i] = max(comps$csize)
  } else {
    gc_size[i] = 0
  }
}
```

Visualize how size of giant cluster changes after node removal.

```
plot(1:n_nodes, gc_size,
     type = "l", lwd = 2,
     col = "pink",
     xlab = c("Number of removed nodes"),
     ylab = c("Giant cluster size"),
     main = "Node removal of an ER random network"
     )
```



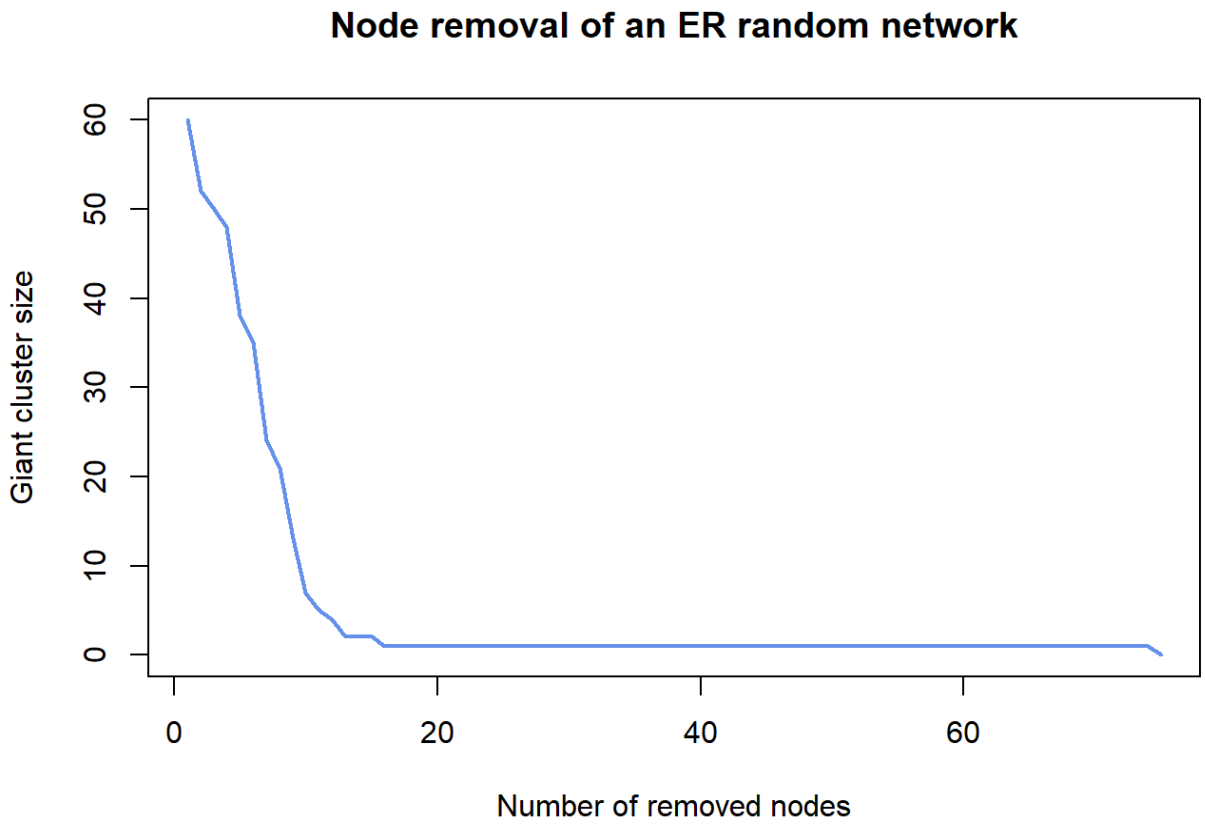
Degree-based Node Removal

Now let's try running target attacks based on degree in the same network.

```
# Restore original network
iceland = og_iceland

# Sort degree
gc_size_hd = numeric(n_nodes)
degrees = degree(iceland)
removed_nodes_hd = order(-degrees)

# Iteratively remove nodes
for(i in 1:n_nodes){
  removed_node_count = removed_nodes_hd[1:i]
  iceland = remove_random_nodes(iceland, removed_node_count)

  comps = components(iceland)
  if(i < n_nodes){
    gc_size_hd[i] = max(comps$csize)
  } else {
    gc_size_hd[i] = 0
  }
}
```

Visualize how size of giant cluster changes after node removal.

```
plot(1:n_nodes, gc_size_hd,
     type = "l", lwd = 2,
     col = "cornflowerblue",
     xlab = c("Number of removed nodes"),
     ylab = c("Giant cluster size"),
     main = "Node removal of an ER random network"
     )
```



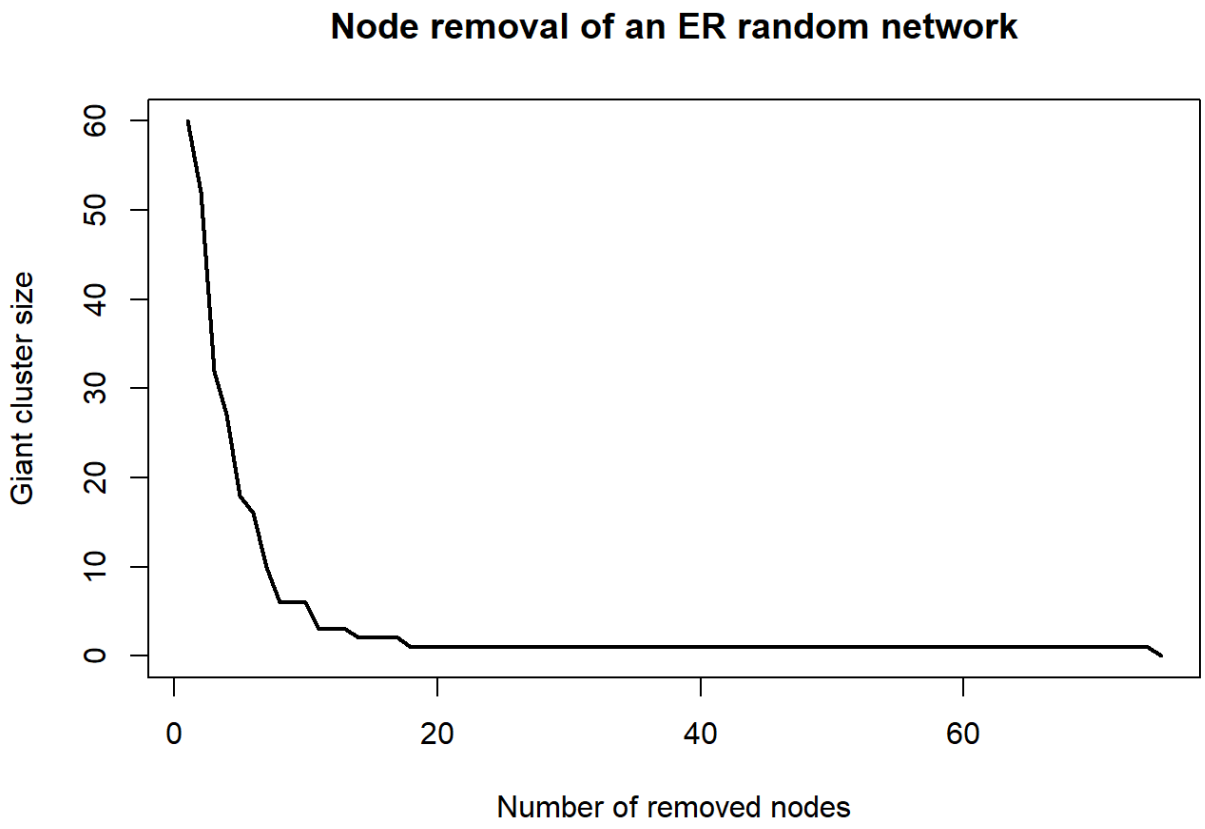
Adaptive High Degree Node Removal

Now let's try another degree-based method, where we will recalculate the degree after each removal and remove the node with the highest degree among the remaining nodes.

```
# Restore original network
iceland = og_iceland

# Create two empty vectors
gc_size_hda = numeric(n_nodes)
removed_nodes_hda = numeric(n_nodes)

# Iteratively remove nodes
for(i in 1:n_nodes){
  degrees = degree(iceland)
  removed_node_count = order(-degrees)[1]
  removed_nodes_hda[i] = removed_node_count

  iceland = remove_random_nodes(iceland, removed_node_count)

  comps = components(iceland)
  if(i < n_nodes){
    gc_size_hda[i] = max(comps$csize)
  } else {
    gc_size_hda[i] = 0
  }
}
```

Visualize how size of giant cluster changes after node removal.

```
plot(1:n_nodes, gc_size_hda,
     type = "l", lwd = 2,
     col = "black",
     xlab = c("Number of removed nodes"),
     ylab = c("Giant cluster size"),
     main = "Node removal of an ER random network"
     )
```

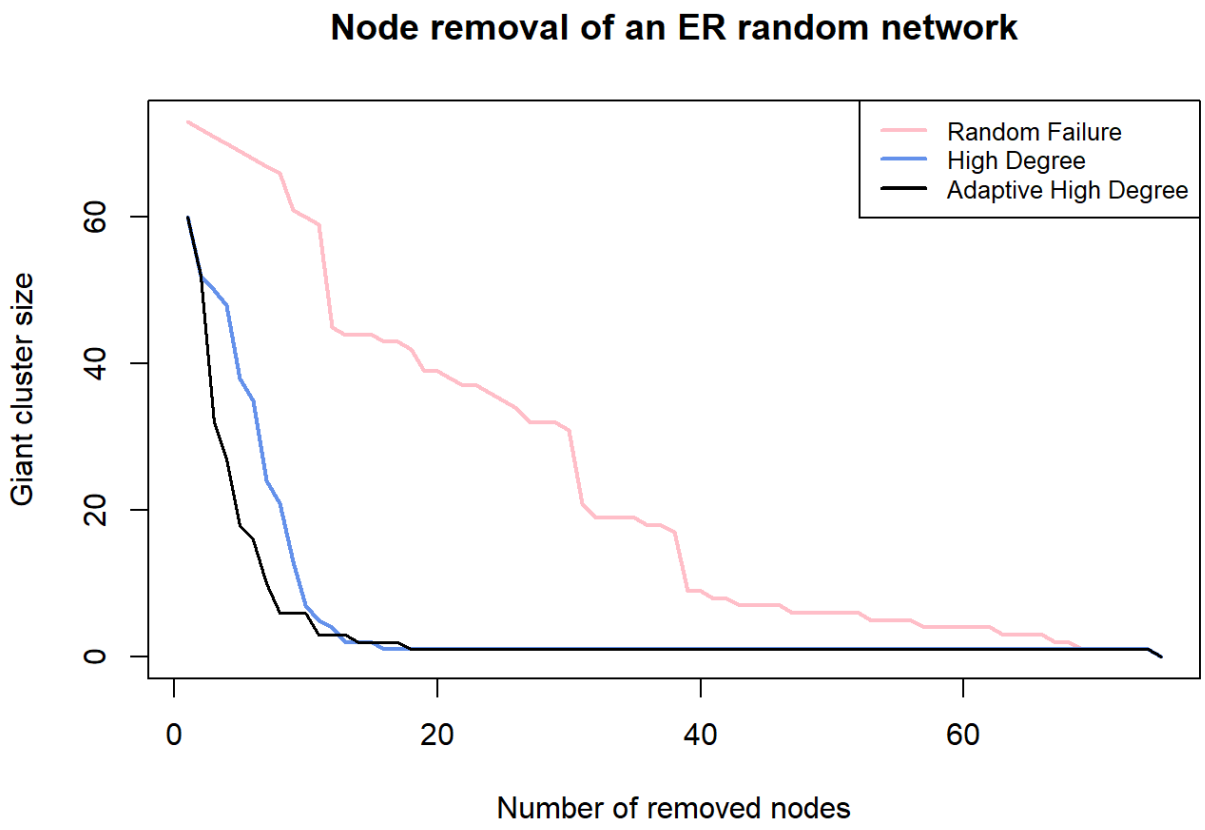


Comparison

Now let’s compare the three methods

Visualize how size of giant cluster changes after node removal.

```
# Random
plot(1:n_nodes, gc_size,
     type = "l", lwd = 2,
     col = "pink",
     xlab = c("Number of removed nodes"),
     ylab = c("Giant cluster size"),
     main = "Node removal of an ER random network"
     )

# High Degree
lines(1:n_nodes, gc_size_hd,
      lwd = 2, col = "cornflowerblue")

# Adaptive
lines(1:n_nodes, gc_size_hda,
      lwd = 1.5, col = "black")

# Legend
legend("topright",
       legend = c("Random Failure", "High Degree", "Adaptive High Degree"),
       col = c("pink", "cornflowerblue", "black"),
       lwd = 2, cex = 0.8)
```



From the plot above, we can see that the adaptive high degree node removal method is most efficient in reducing the size of the giant component after removing fewer nodes. The high degree removal method is not too far behind in terms of efficiency. The random failure method is the least efficient; you have to remove most of the nodes in the

network before the giant component’s size becomes close to 0.