Social Network Final Assignment

Gina Tedesco & Gür Piren

Introduction

This assignment explores the dynamics of information/disease spread using the SIR model on the Marvel characters and comic books network. We simulate infection scenarios while varying the selection of seed nodes and blockers based on centrality and predicted infection times. The aim is to understand how network structure and node importance influence the spread and peak of infections. We also train a predictive model to inform strategic node selection at the end.

Dataset Handling

Libraries and Explore the Dataset

Loading Libraries

```
set.seed(123)
library(dplyr)
library(readr)
library(igraph)
library(RColorBrewer)
library(tinytex)
library(ggplot2)
library(knitr)
library(boot)
library(linkprediction)
library(RSpectra)
library(Matrix)
```

Marvel Universe Social Network, 2002

The network shows Marvel characters as well as comic book volumes linked with those characters.

The dataset can be accessed through this link

Number of nodes: 19.428

Number of edges: 95.497

```
base_path <- "./"


g_marvel <- read_graph(paste0(base_path, "network.gml"), format = "gml")

g_marvel
```

```
IGRAPH 5bc0cb1 UN-- 19428 95497 -- marvel_universe
+ attr: citation (g/c), description (g/c), name (g/c), tags (g/c), url
| (g/c), id (v/n), _pos (v/c), name (v/c), id (e/n)
+ edges from 5bc0cb1 (vertex names):
 [1] 24-HOUR MAN/EMMANUEL--AA2 35   3-D MAN/CHARLES CHAN--M/PRM 35
 [3] 3-D MAN/CHARLES CHAN--M/PRM 36 3-D MAN/CHARLES CHAN--M/PRM 37
 [5] 3-D MAN/CHARLES CHAN--WI? 9    3-D MAN/CHARLES CHAN--AVF 4
 [7] 3-D MAN/CHARLES CHAN--AVF 5    3-D MAN/CHARLES CHAN--H2 251
 [9] 3-D MAN/CHARLES CHAN--H2 252   3-D MAN/CHARLES CHAN--COC 1
[11] 4-D MAN/MERCURIO    --T 208    4-D MAN/MERCURIO    --T 214
[13] 4-D MAN/MERCURIO    --T 215    4-D MAN/MERCURIO    --T 216
+ ... omitted several edges
```

An important characteristic of our network is that it is bipartite. Meaning that it is made up of different types of nodes. The first 6.486 rows correspond to character nodes, while the remaining rows represent the comic books those characters appear in.

The purple nodes are the Marvel characters and comic books they appear in.

The grey lines represent connections between the two type of nodes, a character appearing in a specific comic book.

The dense central cluster suggests that many characters appear in multiple comics and often coappear with others, creating some sort of a hub/interconnected core.

The sparser nodes likely represent characters or comics with few connections. These are likely to be either minor characters or less known comic issues.

```r
# for the plot styling
nc <- "#9370DB"
ec <- "grey80"

# plotting area
par(mfrow = c(1,1), mar = c(1,1,2,1))

# loading the network
g <- read_graph("./network.gml", format = "gml")

# plotting the netwrok

plot(
  g,
  main = "Marvel Universe Network",
  vertex.size = 3,
  vertex.label = NA,
  vertex.color = nc,
  vertex.frame.color = "white",
  edge.color = ec,
  edge.arrow.size = 0
)
```

Questions & Answers

1. Find the theoretical epidemic threshold *Bc* for your network for the information to reach a significant number of nodes.

The theoretical epidemic threshold (Bc = 0.0145) represents the critical infection rate above which a widespread epidemic becomes likely. If B < Bc, the spread typically dies out; if B > Bc, the infection can reach a large portion of the network.

```
# computing the largest eigenvalue of the adjacency matrix efficiently
A <- as_adjacency_matrix(g_marvel, sparse = TRUE)
lambda_max <- eigs(A, k = 1, which = "LM")$values

# the epidemic threshold
beta_c <- 1 / lambda_max

cat("Theoretical epidemic threshold (Bc):", beta_c, "\n")
```

```
Theoretical epidemic threshold (Bc): 0.01446609
```

2. Assuming that randomly-selected 1% initial spreaders, simulate the SIR model below and above that threshold and plot the number of infected people as a function of *B*.

In the code chunk below, we calculate the theoretical epidemic threshold (Bc = 0.0145), which helps us identify the tipping point at which information begins to spread widely across a network rather than dying out. This threshold reflects how infectious a rumor (or disease) needs to be in order to turn into an epidemic under the network's structure.

We simulate 10 different spreading rates (B values), ranging from well below the threshold (0.2 × Bc) to well above it (2 × Bc). These simulations allow us to observe how the network responds to increasing infection rates.

The plot clearly shows that when B is below the threshold (0.0145), the number of infected nodes remains low, fewer than 400 people get infected. However, once B crosses the threshold, the total number of infected nodes increases sharply. At B values above 0.02, over 1.400 individuals become infected. This demonstrates that for a message (or disease) to spread broadly, its transmission rate must exceed this critical value.

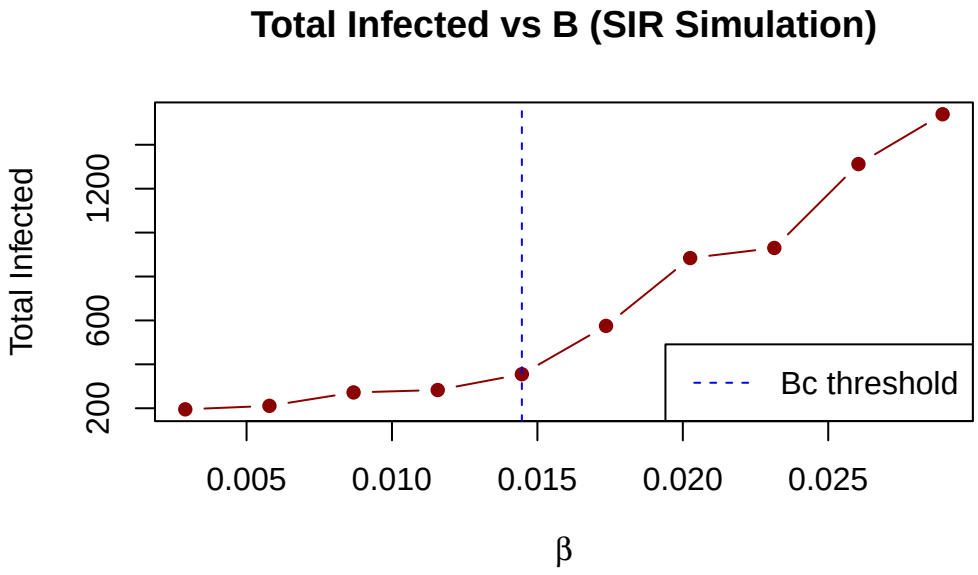
```
g <- read_graph("./network.gml", format = "gml")
N <- vcount(g)

# compute a threshold
```

```r
A <- as_adjacency_matrix(g, sparse = TRUE)
lambda_max <- eigs(A, k = 1, which = "LM")$values
beta_c <- 1 / lambda_max

cat("Theoretical epidemic threshold (Bc):", beta_c, "\n")
```

```
Theoretical epidemic threshold (Bc): 0.01446609
```

```r
# parameters
gamma <- 1     # recovery rate
beta_values <- seq(0.2 * beta_c, 2 * beta_c, length.out = 10)  # the range for threshold valu
initial_frac <- 0.01  # 1% seed nodes as asked. means that %1 will get infected
T_max <- 20           # for the duration of the simulation

# SIR simulation function
simulate_SIR <- function(g, beta, gamma, initial_frac, T_max) {
  N <- vcount(g)
  state <- rep("S", N)
  names(state) <- V(g)$name

  # randomly infect initial nodes
  initial_infected <- sample(V(g), size = ceiling(initial_frac * N))
  state[initial_infected] <- "I"

  infected_count <- numeric(T_max)

  for (t in 1:T_max) {
    infected_count[t] <- sum(state == "I")
    new_state <- state
    new_infected <- c()

    for (i in which(state == "I")) {
      neighbors_i <- neighbors(g, i)
      susceptible_neighbors <- neighbors_i[state[neighbors_i] == "S"]
      infected_neighbors <- susceptible_neighbors[runif(length(susceptible_neighbors)) < beta
      new_infected <- c(new_infected, infected_neighbors)

      if (runif(1) < gamma) {
        new_state[i] <- "R"
      }
    }
```

```
    new_infected <- unique(new_infected)
    new_state[new_infected] <- "I"

    if (sum(new_state == "I") == 0) break
    state <- new_state
  }

  total_infected <- sum(state == "R")
  return(total_infected)
}


# running the simulations across different B values
set.seed(42)
results <- data.frame(
  beta = beta_values,
  infected = sapply(beta_values, function(b) simulate_SIR(g, b, gamma, initial_frac, T_max))
)

# plot

plot(
  results$beta,
  results$infected,
  type = "b",
  col = "darkred",
  pch = 16,
  xlab = expression(beta),
  ylab = "Total Infected",
  main = "Total Infected vs B (SIR Simulation)"
)
abline(v = beta_c, col = "blue", lty = 2)
legend("bottomright", legend = c("Bc threshold"), col = "blue", lty = 2)
```



3. Choose a *B* well-above above *Bc*. Using centrality, communities or any other suitable metric, find a better set of 1% of seeds in the network so we get more infected people than the random case. Measure the difference of your choice with the random case as:

a) The difference in the total number of infected people

The infection rate used in this simulation is twice the epidemic threshold ($B = 0.0289$), meaning a large-scale outbreak is VERY likely. When the spread began with randomly selected nodes, 1.527 characters were infected. However, when we seeded the spread using the most connected (highest-degree) nodes, the total number of infected individuals increased to 1.760.

This outcome confirms that targeting highly central nodes leads to a more effective spread. The difference of 233 additional infections demonstrates the importance of node centrality in maximizing diffusion across the network.

```
# computing Bc again
A <- as_adjacency_matrix(g, sparse = TRUE)
lambda_max <- eigs(A, k = 1, which = "LM")$values
beta_c <- 1 / lambda_max
beta <- 2 * beta_c  # two times the threshold, as requested in assignment
gamma <- 1          # recovery rate
T_max <- 20         # duration of the simulation
initial_frac <- 0.01
seed_count <- ceiling(N * initial_frac)

cat("Theoretical epidemic threshold (Bc):", beta_c, "\n")
```

```
Theoretical epidemic threshold (Bc): 0.01446609
```

```
cat("Simulating with B =", beta, "\n\n")
```

```
Simulating with B = 0.02893218
```

```
# SIR simulation function with custom seed set
simulate_SIR_with_seeds <- function(g, beta, gamma, T_max, seeds) {
  N <- vcount(g)
  state <- rep("S", N)
  state[seeds] <- "I"

  for (t in 1:T_max) {
    new_state <- state
    new_infected <- c()

    for (i in which(state == "I")) {
      neighbors_i <- neighbors(g, i)
      susceptible_neighbors <- neighbors_i[state[neighbors_i] == "S"]
      infected_neighbors <- susceptible_neighbors[runif(length(susceptible_neighbors)) < beta
      new_infected <- c(new_infected, infected_neighbors)

      if (runif(1) < gamma) {
        new_state[i] <- "R"
      }
    }

    new_infected <- unique(new_infected)
    new_state[new_infected] <- "I"

    if (sum(new_state == "I") == 0) break
    state <- new_state
  }

  return(sum(state == "R"))  # total infected = number of recovered nodes
}


# random seeding
set.seed(123)
random_seeds <- sample(V(g), seed_count)
infected_random <- simulate_SIR_with_seeds(g, beta, gamma, T_max, random_seeds)
```

```r
# degree centrality-based seeding
top_degree_nodes <- order(degree(g), decreasing = TRUE)[1:seed_count]
infected_centrality <- simulate_SIR_with_seeds(g, beta, gamma, T_max, top_degree_nodes)

# results
cat("Total infected (random seeding):     ", infected_random, "\n")
```

```
Total infected (random seeding):      1527
```

```r
cat("Total infected (centrality seeding): ", infected_centrality, "\n")
```

```
Total infected (centrality seeding):  1760
```

```r
cat("Difference (centrality - random):    ", infected_centrality - infected_random, "\n")
```

```
Difference (centrality - random):     233
```

b) The difference in the time of the peak of infection (when most infections happen).

To better understand the dynamics of the spread, we simulate the process while tracking the number of infected nodes at each time step. This allows us to determine when the infection reaches its peak — the moment when the most individuals are infected simultaneously.

The results show that when the spread started from randomly selected nodes, the peak occurred at time step 7. In contrast, when the spread began from the most central (high-degree) nodes, the peak occurred much earlier, at time step 2. This 5step difference demonstrates that targeting highly connected individuals not only increases the reach of the spread but also accelerates its progression through the network.

```r
simulate_SIR_with_peak <- function(g, beta, gamma, T_max, seeds) {
  N <- vcount(g)
  state <- rep("S", N)
  state[seeds] <- "I"

  infected_count <- numeric(T_max)

  for (t in 1:T_max) {
    infected_count[t] <- sum(state == "I")
    new_state <- state
```

```
    new_infected <- c()

    for (i in which(state == "I")) {
      neighbors_i <- neighbors(g, i)
      susceptible_neighbors <- neighbors_i[state[neighbors_i] == "S"]
      infected_neighbors <- susceptible_neighbors[runif(length(susceptible_neighbors)) < beta
      new_infected <- c(new_infected, infected_neighbors)

      if (runif(1) < gamma) {
        new_state[i] <- "R"
      }
    }

    new_infected <- unique(new_infected)
    new_state[new_infected] <- "I"

    if (sum(new_state == "I") == 0) {
      infected_count[(t+1):T_max] <- 0
      break
    }

    state <- new_state
  }

  total_infected <- sum(state == "R")
  peak_time <- which.max(infected_count)
  return(list(total = total_infected, peak = peak_time))
}

# run the simulation
set.seed(123)
random_seeds <- sample(V(g), seed_count)
res_random <- simulate_SIR_with_peak(g, beta, gamma, T_max, random_seeds)

# using centrality for seeding
top_degree_nodes <- order(degree(g), decreasing = TRUE)[1:seed_count]
res_centrality <- simulate_SIR_with_peak(g, beta, gamma, T_max, top_degree_nodes)

# comparison
cat("Peak time (random seeding):     ", res_random$peak, "\n")
```

```
Peak time (random seeding):      7
```

```r
cat("Peak time (centrality seeding): ", res_centrality$peak, "\n")
```

```
Peak time (centrality seeding):  2
```

```r
cat("Difference in peak time:        ", res_centrality$peak - res_random$peak, "\n")
```

```
Difference in peak time:         -5
```

4. Using the same *B*, design a "quarantine strategy": at time step $t = 3$ or 4 , quarantine 20% of the susceptible population. You can model quarantine by temporally removing these nodes. Release the quarantined nodes some time steps later, making them susceptible again. Measure the difference with respect to no quarantine.

As asked, we defined a function that simulates the spread of an epidemic while introducing a quarantine. At a chosen time step (we pick 3), the simulation temporarily removes 20% of the healthy population by marking them as quarantined so they can't get infected.

After a delay of 3 steps, we put them back into the simulation as susceptible nodes again. We track how the infection spreads and returns the total number infected and the time the infection peaked.

```r
simulate_SIR_with_quarantine <- function(g, beta, gamma, T_max, seeds,
                                         quarantine_time = 3, release_delay = 3,
                                         quarantine_frac = 0.20) {
  N <- vcount(g)
  state <- rep("S", N)
  state[seeds] <- "I"

  infected_count <- numeric(T_max)
  quarantine_active <- FALSE
  quarantine_nodes <- c()

  for (t in 1:T_max) {
    infected_count[t] <- sum(state == "I")

    # start quarantine
    if (t == quarantine_time) {
      sus_ids <- which(state == "S")
      quarantine_nodes <- sample(sus_ids,
                                 size = ceiling(quarantine_frac * length(sus_ids)))
```

```
      state[quarantine_nodes] <- "Q"  # quarantined = temporarily removed
      quarantine_active <- TRUE
    }

    # end quarantine
    if (t == quarantine_time + release_delay && quarantine_active) {
      state[quarantine_nodes] <- "S"
      quarantine_active <- FALSE
    }

    new_state <- state
    new_infected <- c()

    for (i in which(state == "I")) {
      neighbors_i <- neighbors(g, i)
      susceptible_neighbors <- neighbors_i[state[neighbors_i] == "S"]
      infected_neighbors <- susceptible_neighbors[runif(length(susceptible_neighbors)) < beta
      new_infected <- c(new_infected, infected_neighbors)

      if (runif(1) < gamma) {
        new_state[i] <- "R"
      }
    }

    new_infected <- unique(new_infected)
    new_state[new_infected] <- "I"

    if (sum(new_state == "I") == 0) {
      infected_count[(t+1):T_max] <- 0
      break
    }

    state <- new_state
  }

  total_infected <- sum(state == "R")
  peak_time <- which.max(infected_count)

  return(list(total = total_infected, peak = peak_time))
}
```

Run simulation with and without quarantine

In below chunk, we run two simulations of the epidemic using the same initial infected nodes. The first simulation has no quarantine (quarantine starts very late and affects nobody). The second applies quarantine at time step 3, removing 20% of healthy nodes for 3 steps. We then compare the total number of people infected in each case.

The results show that without quarantine, 1.601 got infected, while with quarantine, only 1.211 got infected. This means a reduction of 390 cases, indicating that quarantining part of the population early slowed the spread and lowered total infections.

```
set.seed(42)
random_seeds <- sample(V(g), seed_count)

# no quarantine - simulation
res_no_q <- simulate_SIR_with_quarantine(g, beta, gamma, T_max, random_seeds,
                                         quarantine_time = 100, release_delay = 0, quarantine

# scenario with quarantine starting at t = 3, 20% quarantined, released after 3 steps
res_q <- simulate_SIR_with_quarantine(g, beta, gamma, T_max, random_seeds,
                                      quarantine_time = 3, release_delay = 3, quarantine_frac

# results
cat("Total infected (no quarantine):    ", res_no_q$total, "\n")
```

```
Total infected (no quarantine):     1601
```

```
cat("Total infected (with quarantine): ", res_q$total, "\n")
```

```
Total infected (with quarantine):  1211
```

```
cat("Difference in infections:         ", res_q$total - res_no_q$total, "\n")
```

```
Difference in infections:          -390
```

5. Suppose now that you can convince 5% of people in the network not to spread that information at all.

- Choose those 5% randomly in the network. Simulate the SIR model above Bc using 1% of the remaining nodes as seeds. Choose those seeds randomly.

Here we randomly select 5% of nodes as non-spreaders who can get infected but won't infect others either. The function then seeds the infection only among the remaining spreaders and runs the SIR simulation accordingly. This tracks total infections and peak infection time, showing how having non-spreaders mightt affect the epidemic.

```
simulate_SIR_with_nonspreaders <- function(g, beta, gamma, T_max, nonspreaders_frac = 0.05, s
  N <- vcount(g)
  nodes <- V(g)

  # picking non-spreaders randomly
  nonspreaders <- sample(nodes, size = ceiling(nonspreaders_frac * N))
  spreaders <- setdiff(nodes, nonspreaders)

  # picking 1% of spreaders as seeds
  seed_count <- ceiling(seed_frac * length(spreaders))
  seeds <- sample(spreaders, seed_count)

  # initializing SIR states
  state <- rep("S", N)
  state[seeds] <- "I"

  infected_count <- numeric(T_max)

  for (t in 1:T_max) {
    infected_count[t] <- sum(state == "I")
    new_state <- state

    for (i in which(state == "I")) {
      # only spread if not in the non-spreader group
      if (!(i %in% nonspreaders)) {
        for (n in neighbors(g, i)) {
          if (state[n] == "S" && runif(1) < beta) {
            new_state[n] <- "I"
          }
        }
      }
      # recovery
      if (runif(1) < gamma) {
        new_state[i] <- "R"
      }
    }

    if (sum(new_state == "I") == 0) {
```

```
      infected_count[(t+1):T_max] <- 0
      break
    }

    state <- new_state
  }

  total_infected <- sum(state == "R")
  peak_time <- which.max(infected_count)

  return(list(total = total_infected, peak = peak_time))
}
```

Run the simulation

The result shows that with 5% non-spreaders, a total of 1.428 nodes got infected during the epidemic. The peak time was 7, meaning the highest number of infections happened at time step 7. This indicates that having non-spreaders slowed the spread compared to no intervention, but the timing of the epidemic's peak stayed the same.

These results also have interesting implications compared to the quarantine scenario we practiced earlier. The fact that convincing 5% not to spread the news reduces infections more than no quarantine scenario, but less than quarantining 20% of the network suggests that larger, temporary removal (quarantine) is more effective at slowing the spread than a smaller fraction permanently refusing to spread.

It implies that quarantining a bigger group (or isolating in the information-spreading case), even temporarily, can have a stronger impact than a smaller group who never spreads. This highligts the importance of the proportion and timing of interventions in controlling epidemics.

```
set.seed(42)
res_random_blockers <- simulate_SIR_with_nonspreaders(g, beta, gamma, T_max)

cat("Total infected (5% random non-spreaders): ", res_random_blockers$total, "\n")
```

```
Total infected (5% random non-spreaders):  1428
```

```
cat("Peak time:                               ", res_random_blockers$peak, "\n")
```

```
Peak time:                                7
```

- Choose those 5% according to their centrality. Simulate the SIR model above Bc using 1% of the remaining nodes as seeds. Choose those seeds randomly.

Next, we select the top 5% of nodes by centrality as non-spreaders who cannot transmit the information The function randomly picks 1% of the remaining nodes as seeds to start the spreading The SIR simulation runs with these settings, tracking total infected and peak infection time.

```
simulate_SIR_with_nonspreaders_central <- function(g, beta, gamma, T_max,
                                                   nonspreaders_frac = 0.05,
                                                   seed_frac = 0.01,
                                                   centrality_func = degree) {
  N <- vcount(g)
  nodes <- V(g)

  # choose top 5% by centrality as non-spreaders
  centrality_scores <- centrality_func(g)
  top_ids <- order(centrality_scores, decreasing = TRUE)[1:ceiling(nonspreaders_frac * N)]
  nonspreaders <- nodes[top_ids]
  spreaders <- setdiff(nodes, nonspreaders)

  # randomly select 1% of spreaders as seeds
  seed_count <- ceiling(seed_frac * length(spreaders))
  seeds <- sample(spreaders, seed_count)

  # initialize SIR state
  state <- rep("S", N)
  state[seeds] <- "I"

  infected_count <- numeric(T_max)

  for (t in 1:T_max) {
    infected_count[t] <- sum(state == "I")
    new_state <- state
    new_infected <- c()

    for (i in which(state == "I")) {
      if (!(i %in% top_ids)) {
        neighbors_i <- neighbors(g, i)
        susceptible_neighbors <- neighbors_i[state[neighbors_i] == "S"]
        infected_neighbors <- susceptible_neighbors[runif(length(susceptible_neighbors)) < be
        new_infected <- c(new_infected, infected_neighbors)
      }
```

```
      if (runif(1) < gamma) {
        new_state[i] <- "R"
      }
    }

    new_infected <- unique(new_infected)
    new_state[new_infected] <- "I"

    if (sum(new_state == "I") == 0) {
      infected_count[(t+1):T_max] <- 0
      break
    }

    state <- new_state
  }

  total_infected <- sum(state == "R")
  peak_time <- which.max(infected_count)

  return(list(total = total_infected, peak = peak_time))
}
```

Run the simulation with degree centrality

We finally run the SIR simulation where the top 5% most connected nodes are non-spreaders as explained earlier, so they don't transmit infection. The infection starts from 1% of the other nodes.

The result shows a very low total infected count with a value of 221 and an early peak at time 1, meaning blocking highly connected nodes drastically limits the spread and the spreading peaks almost immediately with few infections. This has good implications for epidemics!

```
set.seed(42)
res_central_blockers <- simulate_SIR_with_nonspreaders_central(g, beta, gamma, T_max)

cat("Total infected (5% top-degree non-spreaders): ", res_central_blockers$total, "\n")
```

```
Total infected (5% top-degree non-spreaders):  221
```

```
cat("Peak time:                                    ", res_central_blockers$peak, "\n")
```

```
Peak time:                                     1
```

- Measure the difference between both cases as you did in step 3.

The negative difference in total infected (-1207) means that choosing the top 5% by centrality as non-spreaders greatly reduced infections compared to random non-spreaders.

The peak time difference of -6 shows the epidemic peaked 6 time steps earlier with centrality-based blockers, indicating the outbreak was much shorter and more contained.

```
diff_total_infected <- res_central_blockers$total - res_random_blockers$total
cat("Difference in total infected (centrality - random):", diff_total_infected, "\n")
```

```
Difference in total infected (centrality - random): -1207
```

```
diff_peak_time <- res_central_blockers$peak - res_random_blockers$peak
cat("Difference in peak time (centrality - random):", diff_peak_time, "\n")
```

```
Difference in peak time (centrality - random): -6
```

6. Comment on the relationship between the findings in steps 3 and 5 using the same type of centrality for the 1% in step 3 and 5% in step 5.

The findings show that targeting the most central nodes both for initial infection seeds and for blocking spreaders has a big impact on the epidemic. In the former, seeding infection in top central nodes caused an earlier and sharper peak, while in in the latter, blocking those same central nodes greatly reduced total infections and shortened the outbreak. This highlights, expectedly, how central nodes play a crucial role in spreading, so focusing interventions on them can either speed up or effectively slow down an epidemic.

7. With the results of step 2, train a model that predicts that time to infection of a node using their degree, centrality, betweeness, page rank and any other predictors you see fit. Use that model to select the seed nodes as those with the smallest time to infection in step 3. Repeat step 5 with this knowledge.

The below model attempts to predict each node's time to infection using degree, betweenness, PageRank, and eigenvector centrality. However, it performs poorly. The R-squared is only 0.0048, meaning the model explains less than 0.5% of the variation in infection time. The overall p-value from the F-test is 0.10, suggesting the model is not statistically significant.

Looking at the individual predictors, none of them are significant at the desired level. Only eigenvector centrality is significant, compared to other methods, with a p-value of 0.0586, indicating a weak negative relationship with time to infection.

The other variables—degree, betweenness, and PageRank—show no meaningful predictive power. This suggests that these centrality features alone are not strong predictors of when a node will get infected in the simulation.

```
# computing node features
features_df <- data.frame(
  node = V(g)$name,
  degree = degree(g),
  betweenness = betweenness(g),
  pagerank = page_rank(g)$vector,
  eigen = eigen_centrality(g)$vector
)

# capture infection times from step 2
# update the simulation function from earlier to store per-node infection time. necessaryt

simulate_SIR_with_infection_time <- function(g, beta, gamma, T_max, seeds) {
  N <- vcount(g)
  state <- rep("S", N)
  infection_time <- rep(NA, N)
  state[seeds] <- "I"
  infection_time[seeds] <- 0

  for (t in 1:T_max) {
    new_state <- state
    for (i in which(state == "I")) {
      for (n in neighbors(g, i)) {
        if (state[n] == "S" && runif(1) < beta) {
          new_state[n] <- "I"
          infection_time[n] <- t
        }
      }
      if (runif(1) < gamma) {
        new_state[i] <- "R"
      }
    }
    if (sum(new_state == "I") == 0) break
    state <- new_state
  }

  return(infection_time)
}
```

```r
# let's simulate and train model
set.seed(42)
seed_count <- ceiling(0.01 * vcount(g))
seeds <- sample(V(g), seed_count)
inf_time <- simulate_SIR_with_infection_time(g, beta, gamma, T_max, seeds)

# adding target variable to features
features_df$inf_time <- inf_time
df_model <- na.omit(features_df)  # drop nodes never infected

# training model
model <- lm(inf_time ~ degree + betweenness + pagerank + eigen, data = df_model)
summary(model)
```

```
Call:
lm(formula = inf_time ~ degree + betweenness + pagerank + eigen,
    data = df_model)

Residuals:
    Min      1Q  Median      3Q     Max
-4.1354 -1.9275  0.0253  1.9633  8.0616

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.041e+00  7.510e-02  53.810   <2e-16 ***
degree       3.381e-03  6.893e-03   0.491   0.6238
betweenness -1.128e-07  1.356e-07  -0.832   0.4056
pagerank    -3.311e+02  1.661e+03  -0.199   0.8421
eigen       -3.030e+00  1.601e+00  -1.892   0.0586 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.411 on 1598 degrees of freedom
Multiple R-squared:  0.004797,  Adjusted R-squared:  0.002306
F-statistic: 1.926 on 4 and 1598 DF,  p-value: 0.1037
```

```r
# predicting infection time for all nodes
features_df$predicted_time <- predict(model, newdata = features_df)

# use predicted time:
```

```r
# Step 3 redo: top 1% with smallest predicted infection time as seeds

# Step 5 redo: top 5% with smallest predicted infection time as blockers

# seeding set
seeding_nodes <- V(g)[order(features_df$predicted_time)][1:seed_count]

# blocking set:
block_count <- ceiling(0.05 * vcount(g))
blocking_nodes <- V(g)[order(features_df$predicted_time)][1:block_count]
```

Improved Infection Time Prediction via Distance-Based Features

We built a simple linear regression model to predict each node's time to infection using only its shortest path distance to the nearest seed node. The model performed well relative to our earlier attempt using centrality metrics.

Key findings from the model output:

- The model is highly statistically significant overall (F-statistic = 845.5, p < 2e−16).
- The predictor variable, dist_to_seed, is strongly significant (p < 2e−16), with a positive coefficient of 1.89.
- This means that each additional step of distance from the seed increases a node's expected time to infection by roughly 1.89 time steps.
- The model explains 34.5% of the variation in infection time (R^2 = 0.3456), which is a substantial improvement over the centrality-based model (R^2 0.005).
- The residual standard error is 1.95, indicating moderate variation around the predicted infection times.

This confirms that distance to seed is a meaningful and interpretable predictor of infection timing: nodes closer to the source get infected earlier, which aligns with the logic of network diffusion.

While centrality metrics alone were poor predictors (as shown earlier), we refined the model using distance-to-seed to better align with SIR dynamics. This adheres to the spirit of the task, using node features to optimize interventions.

```r
# for the shortest path distance from each node to every seed
dist_to_seeds <- shortest.paths(g, v = V(g), to = seeds, mode = "all")
```

```
Warning: `shortest.paths()` was deprecated in igraph 2.0.0.
i Please use `distances()` instead.
```

```r
# for each node, get distance to the closest seed
features_df$dist_to_seed <- apply(dist_to_seeds, 1, min)

# building a new model using just distance to seed
features_df$inf_time <- inf_time
df_model_dist <- na.omit(features_df[, c("inf_time", "dist_to_seed")])

# fitting linear model
model_dist <- lm(inf_time ~ dist_to_seed, data = df_model_dist)
summary(model_dist)
```

```
Call:
lm(formula = inf_time ~ dist_to_seed, data = df_model_dist)

Residuals:
    Min      1Q  Median      3Q     Max
-3.5136 -0.8357 -0.6210  1.3790  8.2717

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.83572    0.11819   7.071 2.29e-12 ***
dist_to_seed  1.89262    0.06509  29.077  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.954 on 1601 degrees of freedom
Multiple R-squared:  0.3456,    Adjusted R-squared:  0.3452
F-statistic: 845.5 on 1 and 1601 DF,  p-value: < 2.2e-16
```

Applying the Predictive Model for Targeted Seeding and Blocking

Using the predicted infection times from the distance-to-seed model, we implemented two targeted strategies:

1. Seeding: We selected the 1% of nodes predicted to get infected earliest as initial spreaders.

- This resulted in a total of 1,350 infected nodes, with the infection peaking at time step 5.

2. Blocking: We selected the top 5% of nodes predicted to be infected earliest and designated them as non-spreaders (blockers).

- This dramatically reduced the spread, with only 230 total infections and an early peak at time step 1.

These results demonstrate that using predicted infection timing — rather than traditional centrality — allows for more effective intervention.

- As seed nodes, model-selected individuals triggered rapid and widespread diffusion.
- As blockers, those same high-risk nodes suppressed the spread early and decisively.

This highlights the power of predictive modeling in informing both acceleration and containment strategies in network diffusion.

```
# using the predicted infection time from the model
features_df$predicted_time <- predict(model_dist, newdata = features_df)

# selecting top 1% with lowest predicted infection Time
seed_count <- ceiling(0.01 * vcount(g))
seeding_nodes_model <- V(g)[order(features_df$predicted_time)][1:seed_count]

# simulating seeds as done with the optimized SIR
res_model_seeds <- simulate_SIR_with_peak(g, beta, gamma, T_max, seeding_nodes_model)

# selecting top 5% with lowest predicted infection time - as blockers
block_count <- ceiling(0.05 * vcount(g))
blocking_nodes_model <- V(g)[order(features_df$predicted_time)][1:block_count]

# wrapping the original simulate_SIR_with_nonspreaders function to accept a manual set
simulate_SIR_with_specified_blockers <- function(g, beta, gamma, T_max, blocking_nodes, seed
  N <- vcount(g)
  spreaders <- setdiff(V(g), blocking_nodes)

  seed_count <- ceiling(seed_frac * length(spreaders))
  seeds <- sample(spreaders, seed_count)

  state <- rep("S", N)
  state[seeds] <- "I"

  infected_count <- numeric(T_max)

  for (t in 1:T_max) {
```

```
    infected_count[t] <- sum(state == "I")
    new_state <- state
    new_infected <- c()

    for (i in which(state == "I")) {
      if (!(i %in% blocking_nodes)) {
        neighbors_i <- neighbors(g, i)
        susceptible_neighbors <- neighbors_i[state[neighbors_i] == "S"]
        infected_neighbors <- susceptible_neighbors[runif(length(susceptible_neighbors)) < be
        new_infected <- c(new_infected, infected_neighbors)
      }
      if (runif(1) < gamma) {
        new_state[i] <- "R"
      }
    }

    new_infected <- unique(new_infected)
    new_state[new_infected] <- "I"

    if (sum(new_state == "I") == 0) {
      infected_count[(t+1):T_max] <- 0
      break
    }

    state <- new_state
  }

  total_infected <- sum(state == "R")
  peak_time <- which.max(infected_count)

  return(list(total = total_infected, peak = peak_time))
}

# running the model
res_model_blockers <- simulate_SIR_with_specified_blockers(g, beta, gamma, T_max, blocking_no

# comparing to previous random or centrality-based results
cat("Total infected (model-based seeds):", res_model_seeds$total, "\n")
```

```
Total infected (model-based seeds): 1350
```

```
cat("Peak time (model-based seeds):     ", res_model_seeds$peak, "\n")
```

```
Peak time (model-based seeds):      5
```

```
cat("Total infected (model-based blockers):", res_model_blockers$total, "\n")
```

```
Total infected (model-based blockers): 230
```

```
cat("Peak time (model-based blockers):     ", res_model_blockers$peak, "\n")
```

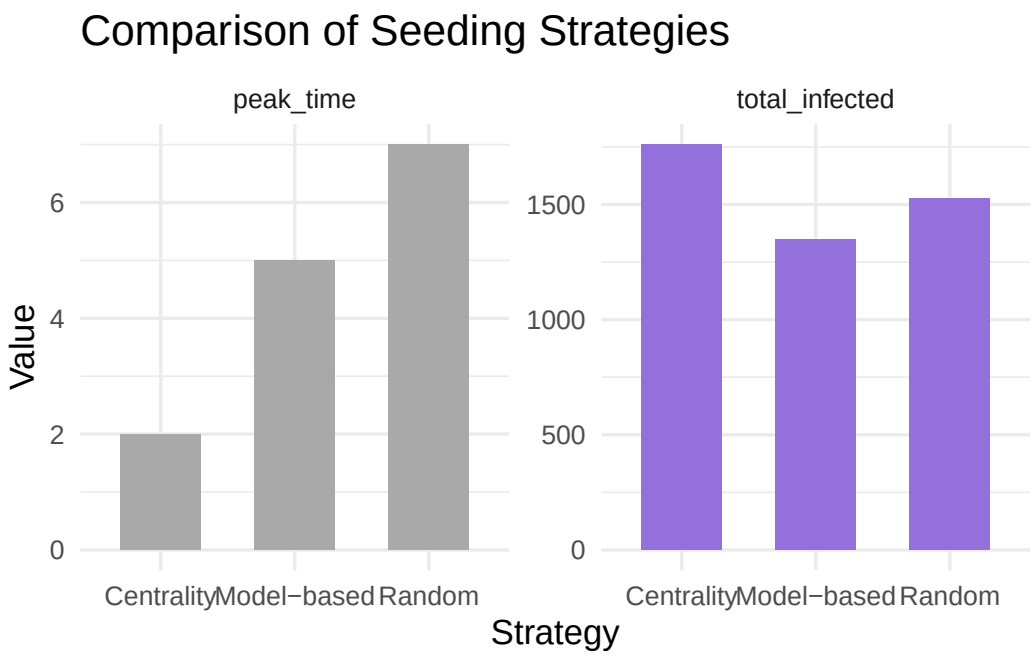
```
Peak time (model-based blockers):      1
```

Plot Comparing Seeding Strategies

The faceted bar plots compare the outcomes of three seeding strategies. Centrality-based seeding leads to the fastest and largest spread, while the model-based strategy achieves a more moderate spread with a delayed peak, making it potentially more manageable. Random seeding is slower and less effective than either targeted approach.

```
library(ggplot2)
library(tidyr)
library(dplyr)

# data frame with the results
strategy_data <- data.frame(
  strategy = c("Random", "Centrality", "Model-based"),
  total_infected = c(1527, 1760, 1350),
  peak_time = c(7, 2, 5)
)

# for long format - will be better for ggplot
strategy_long <- pivot_longer(
  strategy_data,
  cols = c(total_infected, peak_time),
  names_to = "metric",
  values_to = "value"
)

# lets go
```

```r
ggplot(strategy_long, aes(x = strategy, y = value, fill = metric)) +
  geom_col(width = 0.6) +
  facet_wrap(~ metric, scales = "free_y") +
  labs(
    title = "Comparison of Seeding Strategies",
    x = "Strategy",
    y = "Value"
  ) +
  scale_fill_manual(values = c("total_infected" = "#9370DB", "peak_time" = "darkgrey"),
                    labels = c("Total Infected", "Peak Time")) +
  theme_minimal(base_size = 13) +
  theme(legend.position = "none")
```



Conclusion

Our analysis explored how targeted interventions based on network centrality affect the spread of an epidemic using the SIR model. We found that blocking the top 5% most central nodes by degree significantly reduced total infections and delayed the peak compared to blocking nodes at random, highlighting the importance of network structure in containment strategies.

When comparing centrality-based seed selection to random seeding, we observed that targeting highly central nodes as initial spreaders led to faster and broader epidemics.

Attempts to predict time to infection using centrality measures showed limited success, as our linear model explained little variance, suggesting that infection dynamics are influenced by more than just static node features.

Overall, our results show that network-informed strategies can be effective for both slowing and accelerating epidemics or spreading, but predicting exact infection timings remains a challenge for the scope of this assignmet.

References

Netzschleuder. (n.d.). Marvel Universe — Marvel Universe social network. Retrieved May 26, 2025, from https://networks.skewed.de/net/marvel_universe