---

title: "Final code"

author: "Sergei and Tino"

date: "2023-04-09"

output: word\_document

---

{r setup, include=FALSE}

knitr::opts\_chunk$set(echo = TRUE)

{r, include=FALSE}

#call library

library(igraph)

library(RColorBrewer)

library(visNetwork)

library(ggplot2)

library(tidyverse)

library(ggpubr)

library(dplyr)

library("GGally")

#datainput

highschool\_edge<-read.csv("Highschool\_network\_edge.csv",header=FALSE)

highschool\_att<-read.csv("Highschool\_network\_att.csv",header = TRUE)

facebook\_edge<-read.csv("Facebook\_network\_edge.csv",header=FALSE)

facebook\_att<-read.csv("Facebook\_network\_att.csv",header = TRUE)

EXERCISE 1

#build high school network

highschool\_nodes<-data.frame(name=as.character(highschool\_att$NodeID),

gender=as.character(highschool\_att$Gender),

hall=as.character(highschool\_att$Hall))

highschool\_edges<-data.frame(from=c(as.character(highschool\_edge[,1])),

to=c(as.character(highschool\_edge[,2])))

Highschool<-graph\_from\_data\_frame(highschool\_edges,directed = FALSE,vertices =

highschool\_nodes)

co <- components(Highschool)

Highschool <- induced.subgraph(Highschool, which(co$membership == which.max(co$csize)))

#use only the largest component for analysis

summary(Highschool)

#build facebook network

facebook\_nodes<-data.frame(name=as.character(facebook\_att$NodeID))

facebook\_edges<-data.frame(from=c(as.character(facebook\_edge[,1])),

to=c(as.character(facebook\_edge[,2])))

Facebook<-graph\_from\_data\_frame(facebook\_edges,directed = FALSE,vertices = facebook\_nodes)

summary(Facebook)

{r, message=FALSE,}

#build high school network

highschool\_nodes<-data.frame(name=as.character(highschool\_att$NodeID),

gender=as.character(highschool\_att$Gender),

hall=as.character(highschool\_att$Hall))

highschool\_edges<-data.frame(from=c(as.character(highschool\_edge[,1])),

to=c(as.character(highschool\_edge[,2])))

Highschool<-graph\_from\_data\_frame(highschool\_edges,directed = FALSE,vertices = highschool\_nodes)

co <- components(Highschool)

Highschool <- induced.subgraph(Highschool, which(co$membership == which.max(co$csize))) #use only the largest component for analysis

summary(Highschool)

## Question 1 (3 points):

degree\_HS <- degree(Highschool, mode = "all") %>% sort(T)

closeness\_HS <- closeness(Highschool, normalized = TRUE) %>% sort(T)

betweenness\_HS <- betweenness(Highschool, directed = FALSE, normalized = TRUE) %>% sort(T)

eigen\_HS <- eigen\_centrality(Highschool)$vector %>% sort(T)

centralities\_HS <- data.frame(degree\_HS, closeness\_HS, betweenness\_HS, eigen\_HS)

# degree[1]

# betweenness[1]

# closeness[1]

# eigen[1]

degree\_FB <- degree(Facebook, mode = "all") %>% sort(T)

closeness\_FB <- closeness(Facebook, normalized = TRUE) %>% sort(T)

betweenness\_FB <- betweenness(Facebook, directed = FALSE, normalized = TRUE) %>% sort(T)

eigen\_FB<- eigen\_centrality(Facebook)$vector %>% sort(T)

centralities\_FB <- data.frame(degree\_FB, closeness\_FB, betweenness\_FB, eigen\_FB)

{r pressure, echo=FALSE}

set.seed(100)

Highschool\_interactive\_layout<-visNetwork(data.frame(id=V(Highschool)$name), highschool\_edges, main = "Highschool",submain="Can zoom in/out to check the IDs and ties") %>%

visIgraphLayout(layout = "layout\_nicely",smooth = FALSE) %>%

visNodes(shape="circle",label = TRUE) %>%

visOptions(highlightNearest = list(enabled = T, hover = T), nodesIdSelection = T)

Highschool\_interactive\_layout

## Question 2

{r, message=F}

ggpairs(centralities\_HS, aes(alpha = 0.5),

lower = list(continuous = "smooth", colour = "blue"))+

labs(title = "Centralities for Highschool data")+

theme\_minimal()

{r, message=F}

ggpairs(centralities\_FB, aes(alpha = 0.5), lower = list(continuous = "smooth"))+

labs(title = "Centralities for Facebook data")+

theme\_minimal()

## Question 3 (5 points):

h\_distance <- distances(

Highschool,

v = V(Highschool),

to = V(Highschool),

mode = c("all", "out", "in"),

weights = NULL) %>% sort(decreasing = T)#Shortest path lengths between every pair of two nodes in the network

# Creating distances vectors

h\_distance =

distances(

Highschool,

v = V(Highschool),

to = V(Highschool),

mode = c("all", "out", "in"),

weights = NULL)

fb\_distance =

distances(

Facebook,

v = V(Facebook),

to = V(Facebook),

mode = c("all", "out", "in"),

weights = NULL)

dd\_facebook<- degree\_distribution(Facebook)

# Convert degree distribution to data frame

degree\_distribution <- data.frame(degree = seq(1:length(dd\_facebook)),

frequency = dd\_facebook)

FB\_count <- as.data.frame(table(fb\_distance))

FB\_count$fraction <- FB\_count$Freq /sum(FB\_count$Freq) # sum by the number of shortest paths

probs = c(0.5, 0.9)

quantiles <- quantile(degree\_FB, prob=probs)

# Plot degree distribution using ggplot2

ggplot(degree\_distribution, aes(x = degree, y = frequency)) +

geom\_bar(stat = "identity") +

xlab("Degree") +

ylab("% of total count")+

geom\_vline(xintercept = quantiles[1], colour="red", linetype="dotdash") +

geom\_vline(xintercept = quantiles[2], colour="red", linetype="dotdash" ) +

scale\_y\_continuous(labels = scales::percent)+

geom\_label(aes(x = quantiles[1], y = 0.025, label ="q: 0.5")) +

geom\_label(aes(x = quantiles[2], y = 0.025, label = "q: 0.9 ")) +

labs(title = "Degree distribution for Facebook")+

theme\_minimal()

quantiles <- quantile(fb\_distance, prob=probs) + 2

FB\_count %>%

ggplot(aes(x = fb\_distance, y = fraction))+

geom\_bar(stat = "identity") +

geom\_vline(xintercept = quantiles[1] , colour="red", linetype="dotdash") +

geom\_vline(xintercept = quantiles[2] , colour="red", linetype="dotdash" ) +

geom\_label(aes(x = quantiles[1] , y = 0.025, label ="q: 0.5")) +

geom\_label(aes(x = quantiles[2] , y = 0.025, label = "q: 0.9 ")) +

scale\_y\_continuous(labels = scales::percent)+

xlab("Shortest path length") +

ylab("% of total count by group")+

labs(title = "Shortest path distribution for Facebook")+

theme\_minimal()

dd\_highschool<- degree\_distribution(Highschool)

# Convert degree distribution to data frame

degree\_distribution <- data.frame(degree = seq(1:length(dd\_highschool)),

frequency = dd\_highschool)

HS\_count <- as.data.frame(table(h\_distance))

HS\_count$fraction <- HS\_count$Freq /sum(HS\_count$Freq) # sum by the number of shortest paths

probs = c(0.5, 0.9)

quantiles <- quantile(degree\_HS, prob=probs)

# Plot degree distribution using ggplot2

ggplot(degree\_distribution, aes(x = degree, y = frequency)) +

geom\_bar(stat = "identity") +

xlab("Degree") +

ylab("% of total count")+

geom\_vline(xintercept = quantiles[1], colour="red", linetype="dotdash")+

geom\_vline(xintercept = quantiles[2], colour="red", linetype="dotdash" )+

scale\_y\_continuous(labels = scales::percent)+

geom\_label(aes(x = quantiles[1], y = 0.025, label ="q: 0.5"))+

geom\_label(aes(x = quantiles[2], y = 0.025, label = "q: 0.9 "))+

labs(title = "Degree distribution for High school")+

theme\_minimal()

quantiles = quantile(h\_distance, prob=probs) +2

HS\_count %>%

ggplot(aes(x = h\_distance, y = fraction))+

geom\_bar(stat = "identity") +

geom\_vline(xintercept = quantiles[1] , colour="red", linetype="dotdash")+

geom\_vline(xintercept = quantiles[2] , colour="red", linetype="dotdash" )+

geom\_label(aes(x = quantiles[1], y = 0.025, label ="q: 0.5"))+

geom\_label(aes(x = quantiles[2] , y = 0.025, label = "q: 0.9 "))+

scale\_y\_continuous(labels = scales::percent)+

xlab("Shortest path length")+

ylab("% of total count by group")+

labs(title = "Shortest path distribution for High school")+

theme\_minimal()

## Question 4

#Codes to visualize the network and calculate subgraph density:

## visualize the network by gender###

library(RColorBrewer)

coul <- brewer.pal(length(unique( V(Highschool)$gender)), "Set2")

my\_color <- coul[as.numeric(as.factor(V(Highschool)$gender))]

set.seed(10)

plot(Highschool, vertex.color = my\_color,

vertex.size=5,

layout=layout\_nicely(Highschool),vertex.label=NA,

main="Highschool network by gender")

legend("bottomleft", legend=levels(as.factor(V(Highschool)$gender)) ,col = coul , bty =

"n", pch=20 , pt.cex = 1.5, cex = 1.5, horiz = FALSE, inset = c(0.1, 0.1))

#introduce subgraph by gender, calculate their edge densities

group\_gender <- as.factor(unique(V(Highschool)$gender))

sapply(levels(group\_gender), function(x) {

y <- induced\_subgraph(Highschool, which(V(Highschool)$gender==x))

cat(paste0("Density for ", x, " friends is ", edge\_density(y), '\n'))

})

par(mfrow = c(1, 3))

for (i in 1:length(group\_gender)) {

subgraph <- subgraph(Highschool, V(Highschool)$gender == group\_gender[i])

plot(subgraph, vertex.size=5,

layout=layout\_nicely(subgraph),vertex.label=NA, xlab =paste(group\_gender[i]) )

}

#introduce subgraph by gender, calculate their edge densities

group\_hall <- as.factor(unique(V(Highschool)$hall))

sapply(levels(group\_hall), function(x) {

y <- induced\_subgraph(Highschool, which(V(Highschool)$hall==x))

cat(paste0("Density for ", x, " friends is ", edge\_density(y), '\n'))

})

par(mfrow = c(2, 3))

for (i in 1:length(group\_hall)) {

subgraph <- subgraph(Highschool, V(Highschool)$hall == group\_hall[i])

plot(subgraph, vertex.size=5,

layout=layout\_nicely(subgraph),vertex.label=NA, xlab = paste("Network of", group\_hall[i]))

}

dev.off()

paste0('Density for the whole network is ', edge\_density(Highschool))

unknown = induced\_subgraph(Highschool, V(Highschool)$gender=='unknown')

Highschool\_interactive\_layout<-visNetwork(data.frame(id=V(unknown)$name),

highschool\_edges, main = "Highschool",submain="Can zoom in/out to check the IDs and

ties") %>%

visIgraphLayout(layout = "layout\_nicely",smooth = FALSE) %>%

visNodes(shape="circle",label = TRUE) %>%

visOptions(highlightNearest = list(enabled = T, hover = T), nodesIdSelection = T)

Highschool\_interactive\_layout

library(network)

library("intergraph")

library('ergm')

Highschool<-graph\_from\_data\_frame(highschool\_edges,directed = FALSE,vertices = highschool\_nodes)

net\_Highschool<-asNetwork(Highschool)

net\_Highschool %e% "gender" <- highschool\_att$Gender

net\_Highschool %v% "hall" <- highschool\_att$Hall

model <- ergm(net\_Highschool ~ edges + nodematch("gender") + nodematch("hall"))

summary(model)

detach("package:ergm", unload=TRUE)

detach("package:network", unload=TRUE)

## Question 5

#Codes to customize community and calculate modularity:

### customize community by gender ###

genderCommunity<-V(Highschool)$gender

genderCommunity<-replace(genderCommunity, genderCommunity=="female",1)

genderCommunity<-replace(genderCommunity, genderCommunity=="male",2)

genderCommunity<-replace(genderCommunity, genderCommunity=="unknown",3)

genderCommunity<-as.numeric(genderCommunity)

hallCommunity<-V(Highschool)$hall

hallCommunity<-replace(hallCommunity,hallCommunity=="1501",1)

hallCommunity<-replace(hallCommunity,hallCommunity=="1502",2)

hallCommunity<-replace(hallCommunity,hallCommunity=="1503",3)

hallCommunity<-replace(hallCommunity,hallCommunity=="1504",4)

hallCommunity<-replace(hallCommunity,hallCommunity=="1505",5)

hallCommunity<-as.numeric(hallCommunity)

#clustering

gender.clustering <- make\_clusters(Highschool, membership=genderCommunity)

hall.clustering <- make\_clusters(Highschool, membership=hallCommunity)

#modularity

paste0('gender modularity ', modularity(gender.clustering))

paste0('hall modularity ', modularity(hall.clustering))

### Louvain algorithm ###

Louv<-cluster\_louvain(Highschool)

modularity(Louv)

#Codes to visualize the network and calculate subgraph density:

## visualize the network by gender###

library(RColorBrewer)

coul <- brewer.pal(length(unique( Louv$membership)), "Set2")

my\_color <- coul[as.numeric(Louv$membership)]

set.seed(10)

plot(Highschool, vertex.color = my\_color,

vertex.size=5,

layout=layout\_nicely(Highschool), vertex.label=NA,

main="Highschool network by moduarity")

legend("bottomleft", legend= unique(Louv$membership) ,col = coul , bty =

"n", pch=20 , pt.cex = 1.5, cex = 1.5, horiz = FALSE, inset = c(0.1, 0.1))

#Codes to visualize the network and calculate subgraph density:

## visualize the network by gender###

library(RColorBrewer)

coul <- brewer.pal(length(unique( V(Highschool)$hall)), "Set2")

my\_color <- coul[as.numeric(as.factor(V(Highschool)$hall))]

set.seed(10)

plot(Highschool, vertex.color = my\_color,

vertex.size=5,

layout=layout\_nicely(Highschool),vertex.label=NA,

main="Highschool network by hall")

legend("bottomleft", legend=levels(as.factor(V(Highschool)$hall)) ,col = coul , bty =

"n", pch=20 , pt.cex = 1.5, cex = 1.5, horiz = FALSE, inset = c(0.1, 0.1))

EXERCISE 2

## Question 6

n=40

ER1 <- sample\_gnp(n, 0.05, directed = FALSE, loops = FALSE)

ER2 <- sample\_gnp(n, 0.1, directed = FALSE, loops = FALSE)

ER3 <- sample\_gnp(n, 0.2, directed = FALSE, loops = FALSE)

ER1000 <- sample\_gnp(1000, 0.2, directed = FALSE, loops= FALSE)

#use only the largest component for analysis

#plot(ER1, layout = layout.circle(ER1), vertex.label = NA)

#plot(ER2, layout = layout.circle(ER2), vertex.label = NA)

#plot(ER3, layout = layout.circle(ER3), vertex.label = NA)

#plot(ER1000)

transitivity(ER1000)

## Question 7

Run the following code and discuss with your group mate how the rewiring probability changes the network structure:

Regular<-watts.strogatz.game(dim=1,size=300,nei=6, p=0)

plot(Regular, layout=layout.circle, vertex.label=NA, vertex.size=5,

main = "Network with zero rewiring probability ")

SW1 <- watts.strogatz.game(dim=1,size=300,nei=6, p=0.001)

plot(SW1, layout=layout.circle, vertex.label=NA, vertex.size=5,

main= "Network with 0.001 rewiring probability ")

SW2 <-watts.strogatz.game(dim=1,size=300,nei=6, p=0.01)

plot(SW2, layout=layout.circle, vertex.label=NA, vertex.size=5,

main = "Network with 0.01 rewiring probability ")

SW3 <- watts.strogatz.game(dim=1,size=300,nei=6, p=0.1)

plot(SW3, layout=layout.circle, vertex.label=NA, vertex.size=5,

main = "Network with 0.1 rewiring probability ")

transitivity(Regular)

average.path.length(Regular)

transitivity(SW1)

average.path.length(SW1)

transitivity(SW2)

average.path.length(SW2)

transitivity(SW3)

average.path.length(SW3)

## Question 8

shortest\_paths <- distances(Regular, v = V(Regular), to = V(Regular))

L0 <- mean(shortest\_paths[shortest\_paths != Inf])

C0 <- transitivity(Regular)

vecC <- character()

vecL <- character()

vecP <- c(0, 0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0,1, 0.5, 1)

for (val in vecP) {

network<-watts.strogatz.game(dim=1,size=300,nei=6, p=val)

shortest\_paths <- distances(network, v = V(network), to = V(network))

L <- (mean(shortest\_paths[shortest\_paths != Inf]))/L0

C <- transitivity(network)/C0

vecL <- append(vecL, L)

vecC <- append(vecC, C)

}

vecL <- as.double(vecL)

vecC <- as.double(vecC)

data <- data.frame(vecL, vecC, vecP)

# Plot the data

ggplot(data) +

geom\_point(aes(vecP, vecL)) +

geom\_point(aes(vecP, vecC), colour='black', shape=0) +

scale\_x\_log10() +

labs(x = 'p', y = 'C and L') +

geom\_rect(mapping=aes(xmin=0.0004, xmax=0.012, ymin=0.3, ymax=1.02), color="red", alpha=0.00001) +

theme\_minimal()

## Question 9

g0 <- barabasi.game(100, power = 0.5 , m = NULL, out.dist = NULL, out.seq = NULL, out.pref = FALSE, zero.appeal = 1, directed = FALSE,algorithm ="psumtree", start.graph = NULL)

g1 <- barabasi.game(100, power = 1 , m = NULL, out.dist = NULL, out.seq = NULL, out.pref = FALSE, zero.appeal = 1, directed = FALSE,algorithm ="psumtree", start.graph = NULL)

g1\_5 <- barabasi.game(100, power = 1.5 , m = NULL, out.dist = NULL, out.seq = NULL, out.pref = FALSE, zero.appeal = 1, directed = FALSE,algorithm ="psumtree", start.graph = NULL)

g2 <- barabasi.game(100, power = 2 , m = NULL, out.dist = NULL, out.seq = NULL, out.pref = FALSE, zero.appeal = 1, directed = FALSE,algorithm ="psumtree", start.graph = NULL)

plot(g0, vertex.label= NA, edge.arrow.size=2.9, vertex.size = 4 ,layout = layout\_nicely(g0), main = "Scale-free network model, power=0.5")

plot(g1, vertex.label= NA, edge.arrow.size=2.9, vertex.size = 4 ,layout = layout\_nicely(g1), main = "Scale-free network model, power=1")

plot(g1\_5, vertex.label= NA, edge.arrow.size=2.9, vertex.size = 4 ,layout = layout\_nicely(g1\_5), main = "Scale-free network model, power=1.5")

plot(g2, vertex.label= NA, edge.arrow.size=2.9, vertex.size = 4 ,layout = layout\_nicely(g2), main = "Scale-free network model, power=2")

library(igraph)

set.seed(100)

random\_n <- sample(1:100, 1)

subset\_random\_05 <- igraph::delete.vertices(g0, V(g0) == random\_n)

subset\_random\_15 <- igraph::delete.vertices(g1\_5, V(g1\_5) == random\_n)

degree\_FB <- degree(subset\_random\_05, mode = "all") %>% sort(T)

node\_with\_max05 <- degree(g0) == max(degree(g0))

node\_with\_max15 <- max(degree(g1\_5))

subset\_target\_05 <- igraph::delete.vertices(g0, V(g0) == match(max(degree(g0)), degree(g0)))

subset\_target\_15 <- igraph::delete.vertices(g1\_5, V(g1\_5) == match(max(degree(g1\_5)), degree(g1\_5)))

diameter(g0)

diameter(g1\_5)

diameter(subset\_random\_05)

diameter(subset\_random\_15)

diameter(subset\_target\_05)

diameter(subset\_target\_15, unconnected = T)

count\_components(subset\_random\_05)

count\_components(subset\_random\_15)

count\_components(subset\_target\_05)

count\_components(subset\_target\_15)

plot(subset\_random\_05, vertex.label= NA, edge.arrow.size=2.9,xlim = c(-1, 1), ylim = c(-1, 1) , vertex.size = 4 ,layout = layout\_nicely(subset\_random\_05), xlab = "Random attack on network model, power=0.5")

plot(subset\_random\_15, vertex.label= NA, edge.arrow.size=2.9, vertex.size = 4 ,xlim = c(-1, 1), ylim = c(-1, 1),layout = layout\_nicely(subset\_random\_15), xlab = "Random attack on network model, power=1.5")

plot(subset\_target\_05, vertex.label= NA, edge.arrow.size=2.9, vertex.size = 4 ,xlim = c(-1, 1), ylim = c(-1, 1),layout = layout\_nicely(subset\_target\_05), xlab = "Target attack on network model, power=0.5")

plot(subset\_target\_15, vertex.label= NA, edge.arrow.size=2.9, vertex.size = 4 ,xlim = c(-1, 1), ylim = c(-1, 1),layout = layout\_nicely(subset\_target\_15), xlab = "Target attack on network model, power=1.5")

EXERCISE 3

# Question 10

{r, echo=FALSE}

#build high school network

highschool\_nodes<-data.frame(name=as.character(highschool\_att$NodeID),

gender=as.character(highschool\_att$Gender),

hall=as.character(highschool\_att$Hall))

highschool\_edges<-data.frame(from=c(as.character(highschool\_edge[,1])),

to=c(as.character(highschool\_edge[,2])))

Highschool<-graph\_from\_data\_frame(highschool\_edges,directed = FALSE,vertices =

highschool\_nodes)

co <- components(Highschool)

Highschool <- induced.subgraph(Highschool, which(co$membership == which.max(co$csize)))

#use only the largest component for analysis

summary(Highschool)

Highschool\_interactive\_layout<-visNetwork(data.frame(id=V(Highschool)$name),

highschool\_edges, main = "Highschool",submain="Can zoom in/out to check the IDs and

ties") %>%

visIgraphLayout(layout = "layout\_nicely",smooth = FALSE) %>%

visNodes(shape="circle",label = TRUE) %>%

visOptions(highlightNearest = list(enabled = T, hover = T), nodesIdSelection = T)

Highschool\_interactive\_layout

print('transitivity:')

transitivity(Highschool)

print('average path length:')

average.path.length(Highschool)

avg\_path\_length <- average.path.length(Highschool)

print(paste0('avg path length:', avg\_path\_length))

candidates <- c()

for (i in 1:nrow((highschool\_edges))) {

# remove an edge

graph2 <- delete\_edges(Highschool, i)

# Calculate the new average path length

new\_avg\_path\_length <- average.path.length(graph2)

# Check if the new average path length is greater than the current one

if (new\_avg\_path\_length > avg\_path\_length\*1.0125) {

# Print out the IDs of the source and target nodes for the edge that was removed

print(paste('deleted edge:', highschool\_edges[i,1], "-", highschool\_edges[i,2]))

print(paste0('new\_avg\_path\_length:' ,new\_avg\_path\_length))

candidates <- append(candidates, i)

}

}

graph <- delete\_edges(Highschool, candidates)

average.path.length(graph)

my\_subset <- data.frame(

from = highschool\_edges$from[candidates],

to = highschool\_edges$to[candidates])

## Question 11

{r, message=FALSE}

stopifnot(require(data.table))

stopifnot(require(Matrix))

calculate\_value <- function(node, each\_neighbors,Pprob){

return(each\_neighbors[[node]][ which(runif(length(each\_neighbors[[node]]), 0, 1)<=Pprob)])

#'runif' is a function to generate random number in R

}

#This function:

#1) searches the neighbors of contagious node;

#2) To those who are connected to a contagious node, generates a random number and compare to the

#probability of p, if random number<p, this node will be infected and return the value of 1

IC<-function(node\_seed,network,Pprob){

#prepare input for the 'calculate\_value' function#

adj\_matrix <- igraph::as\_adjacency\_matrix(network, type = 'both')

each\_neighbors <- which(adj\_matrix > 0, arr.ind = TRUE)

each\_neighbors <- split(each\_neighbors[, 2], each\_neighbors[, 1]) #get the neigbhour list of each node

nNode<-vcount(network)

node\_status <- rep.int(0, nNode) #start from a healthy population

day\_infected<-vector()#Total number of infected population

new\_infected <- list() # Record the ID of person getting infected at each time step

day<-1

node\_status[as.numeric(node\_seed)] <- 1 # infected(value=1) health(value=0)

day\_infected[day] <- sum(node\_status )

new\_infected[[day]]<-node\_seed #The ID of the person infected in Day 1 (Patient Zero)

#simulate the spread of virus within 4 weeks##

for (day in c(2:28)){

ContagiousID<-which(node\_status == 1)

infectedID<-unlist(lapply(ContagiousID,calculate\_value,each\_neighbors,Pprob))

newinfectedID<- setdiff(infectedID, which(node\_status == 1))

#Update the node status and other variables

node\_status[newinfectedID] <- 1

day\_infected[day] <- length(newinfectedID)

new\_infected[[day]]<-newinfectedID

day=day+1

}

return(day\_infected) #return the number of newly infected people by day

#return(list(day\_infected,new\_infected)) #if you want to see the ID of infected ppl in each day,use this command instead

}

tel = 0

for (i in 0:99){

tel = tel + IC("5", Highschool, 0.15)

}

as.integer(tel/100)

## Question 12

li = list()

avg\_path\_length <- average.path.length(Highschool)

print(paste0('avg path length:', avg\_path\_length))

for (i in 1:nrow(highschool\_edges)) {

# remove an edge

graph2 <- delete\_edges(Highschool, i)

# Calculate the new average path length

new\_avg\_path\_length <- average.path.length(graph2)

# Check if the new average path length is greater than the current one

if (new\_avg\_path\_length > avg\_path\_length\*1.0125) {

li <- append(li, i)

# Print out the IDs of the source and target nodes for the edge that was removed

print(paste('deleted edge:', highschool\_edges[i,1], "-", highschool\_edges[i,2]))

print(paste0('new\_avg\_path\_length:', new\_avg\_path\_length))

}

}

# Delete edges that reduce average path length

Highschool2 <- delete\_edges(Highschool, li)

# Identify strong ties using k-core algorithm

k\_cores <- coreness(Highschool2)

# Extract the nodes that belong to the strongest k-core subgraph

strong\_ties <- V(Highschool2)[which(k\_cores == max(k\_cores))]

# Print the nodes in the stronges

print(strong\_ties)

list1 <- list('S20|S112','S112|S1', 'S110|S52', 'S107|S28', 'S64|S58')

Highschool3 <- Highschool2

for (edge in list1){

Highschool3<-delete\_edges(Highschool3, edge)

}

Highschool\_interactive\_layout2<-visNetwork(data.frame(id=V(Highschool2)$name),

highschool\_edges, main = "Highschool2",submain="Can zoom in/out to check the IDs and

ties") %>%

visIgraphLayout(layout = "layout\_nicely",smooth = FALSE) %>%

visNodes(shape="circle",label = TRUE) %>%

visOptions(highlightNearest = list(enabled = T, hover = T), nodesIdSelection = T)

Highschool\_interactive\_layout2

infected <- function(network1, network2, network3, p=0.15){

tel1 = 0

tel2 = 0

tel3 = 0

for (i in 0:99){

tel1 = tel1 + IC("5", network1, p)

tel2 = tel2 + IC("5", network2, p)

tel3 = tel3 + IC("5", network3, p)

}

tel1 = as.integer(tel1/100)

tel2 = as.integer(tel2/100)

tel3 = as.integer(tel3/100)

len <- 1:length(tel1)-1

print(tel1)

print(tel2)

print(tel3)

ggplot() + geom\_smooth(aes(len, tel1, color='red')) +

geom\_smooth(aes(len, tel2, color='green')) +

geom\_smooth(aes(len, tel3, color='blue')) +

scale\_color\_discrete(labels=c('Highschool', 'Highschool2', 'Highschool3')) +

xlab('day') + ylab('newly infected')

}

#build high school network

highschool\_nodes<-data.frame(name=as.character(highschool\_att$NodeID),

gender=as.character(highschool\_att$Gender),

hall=as.character(highschool\_att$Hall))

highschool\_edges<-data.frame(from=c(as.character(highschool\_edge[,1])),

to=c(as.character(highschool\_edge[,2])))

Highschool<-graph\_from\_data\_frame(highschool\_edges,directed = FALSE,vertices =

highschool\_nodes)

co <- components(Highschool)

Highschool <- induced.subgraph(Highschool, which(co$membership == which.max(co$csize)))

infected(Highschool3, Highschool2, Highschool)

## Question 13

infected(Highschool3, Highschool2, Highschool, 0.075)

infected(Highschool3, Highschool2, Highschool, 0.9)

networked\_sir <- function(X0, G, beta, gamma, tmax) {

# Simulate the networked SIR model with arbitrary network G (igraph)

# using the Gillespie stochastic simulation algorithm (SSA).

# Parameters:

# X0 : Initial configuration of state vector X0 in {0,1,2}^N,

# where {0,1,2} ~ {S,I,R} and N is the number of individuals

# G : Network (igraph object) consisting of N nodes (individuals)

# beta : per-edge infection rate

# gamma : Recovery rate

# tmax : Maximum time duration of simulation

# Returns:

# List with data frame [t,s(t),i(t),r(t)], network G,

# initial state vector X0, final state vector X, and parameters beta, gamma

N <- length(X0) # Number of individuals

if(N != vcount(G)) {

error("Length of X0 has to equal number of vertices!")

}

lambda <- numeric(length = N) # Allocate rate vector

num\_inf\_neighbours <- numeric(length = N) # Allocate number of infected neighbors vector

# Set state X to initial state and time t to zero

X <- X0;

t <- 0

# Keep track of number of individuals in each compartment.

st <- length(which(X == 0))

it <- length(which(X == 1))

rt <- length(which(X == 2))

new\_inf <- list()

# Initialize data matrix with observations

df <- data.frame(t = 0, s = st, i = it, r = rt)

dm <- as.matrix(df) # Matrix is faster than data frame to update

# Get adjacency list of G. This appears to be the fastest way

# to find the neighbors of a particular individual

adj\_list <- igraph::as\_adj\_list(G)

days\_infected <- X0

# Compute rates for all individuals

for(i in 1:N) {

if(X[i] == 0) { # If susceptible

neighbour\_indices <- adj\_list[[i]]

num\_inf\_neighbours[i] <- length(which(X[neighbour\_indices] == 1))

lambda[i] <- beta

} else if(X[i] == 1) { # If infected

lambda[i] <- gamma

} else {

lambda[i] <- 0

}

}

while(t < tmax) {

lambda0 <- sum(lambda)

newly\_infected <- 0

t <- t + 1

pos\_j <- which(X==1)

for(infect in pos\_j){

neighbour\_indices <- adj\_list[[infect]] # Get neighbor indices of individual j

for(j in neighbour\_indices){ # get neighbours

if(X[j]==0){ # check if they are susceptible

if(lambda[j]>runif(1)){ # check if they get infected

X[j]<-1 #infection event

days\_infected[j] <- 1 # count days of being infected

lambda[j] <- gamma # Update rate for individual j

# Increment number of infected neighbors count for neighborhood

num\_inf\_neighbours[neighbour\_indices] <- num\_inf\_neighbours[neighbour\_indices] + 1

# Bookkeeping

st <- st - 1

it <- it + 1

newly\_infected <- newly\_infected+1

}

}

}

if(X[infect]==1) { # Recovery event

days\_infected[infect] <- days\_infected[infect]+1

if(days\_infected[infect]>3){# check if person is 3 days infected

X[infect] <- 2 # Set individual j to recovered

lambda[infect] <- 0 # Set individual j rate to zero

# Decrement number of infected neighbors count

num\_inf\_neighbours[neighbour\_indices] <- num\_inf\_neighbours[neighbour\_indices] - 1

# Bookkeeping

it <- it - 1

rt <- rt + 1

}

}

}

# Update rates for susceptible neighbours of individual j

update\_indices <- neighbour\_indices[X[neighbour\_indices] == 0]

lambda[update\_indices] <- beta\*num\_inf\_neighbours[update\_indices]

# Update data matrix

dm <- rbind(dm,c(t,st,it,rt))

print(paste0('day:',t))

print(paste0('newly\_infected:',newly\_infected))

print(paste0('infected:',length(which(X==1))))

print(paste0('recovered:',length(which(X==2))))

new\_inf <- append(new\_inf, newly\_infected)

} # end while

# Return fractions instead of counts

dm[,c("s","i","r")] <- dm[,c("s","i","r")]/N

df <- as.data.frame(dm)

row.names(df) <- NULL # Remove row names

return(new\_inf)

}

# Population size

N <- vcount(Highschool)

# Parameters

X0 <- rep.int(0, N)

seed <- 5 # patient zero

beta <- 0.15 # infection rate

gamma <- 1

tmax <- 28

X0[seed]<-1

sim <- networked\_sir(X0 = X0, G = Highschool, beta = beta, gamma = gamma, tmax = tmax)

dfm <- melt(sim$df, id.vars = "t")

ggplot(data = dfm, mapping = aes(t, value, color=variable)) +

geom\_line() +

theme\_light()

sir\_inf <- function(network1, network2, network3, X0, beta, gamma, tmax){

tel1 = 0

tel2 = 0

tel3 = 0

for (i in 0:99){

tel1 = tel1 + (networked\_sir(X0 = X0, G = network1, beta = beta, gamma = gamma, tmax = tmax))

tel2 = tel2 + (networked\_sir(X0 = X0, G = network2, beta = beta, gamma = gamma, tmax = tmax))

tel3 = tel3 + (networked\_sir(X0 = X0, G = network3, beta = beta, gamma = gamma, tmax = tmax))

}

tel1 = as.integer(tel1/100)

tel2 = as.integer(tel2/100)

tel3 = as.integer(tel3/100)

len <- 1:length(tel1)-1

print(tel1)

print(tel2)

print(tel3)

ggplot() + geom\_smooth(aes(len, tel1, color='red')) +

geom\_smooth(aes(len, tel2, color='green')) +

geom\_smooth(aes(len, tel3, color='blue')) +

scale\_color\_discrete(labels=c('Highschool', 'Highschool2', 'Highschool3')) +

xlab('day') + ylab('newly infected')

}

sir\_inf(Highschool, Highschool2, Highschool3, X0, 0.15, gamma, tmax)

sir\_inf(Highschool, Highschool2, Highschool3, X0, 0.1, gamma, tmax)

sir\_inf(Highschool, Highschool2, Highschool3, X0, 0.9, gamma, tmax)

## Question 14

threshold\_model <- function(network, node\_seed, threshold, n\_day){

nNode <- vcount(network)

node\_status <- rep.int(0, nNode) # start from a healthy population

adj\_matrix <- igraph::as\_adjacency\_matrix(network, type = 'both')

each\_neighbors <- which(adj\_matrix > 0, arr.ind = TRUE)

each\_neighbors <- split(each\_neighbors[, 2], each\_neighbors[, 1]) # get the neighbour list of each node

day <- 1

for (seed in node\_seed){

node\_status[as.numeric(node\_seed)] <- 1 # adopt(value=1) dont adopt(value=0)

}

for (day in 2:n\_day) {

for (node in 1:nNode) {

if (node\_status[node] == 0) {

neighbours <- each\_neighbors[[node]]

n\_neighbors <- length(neighbours)

n\_adopters <- sum(node\_status[neighbours] == 1)

if(n\_adopters > n\_neighbors \* threshold[node]){

node\_status[node] <- 1

}

}

}

}

return(sum(node\_status==1))

}

threshold\_model(Highschool, list('59', '63', '91', '92', '99'), highschool\_att$Threshold, 10000)

Highschool\_interactive\_layout

stopifnot(require(data.table))

stopifnot(require(Matrix))

calculate\_adoptedNei <- function(node, node\_status, each\_neighbors){

return(mean(node\_status[each\_neighbors[[node]]] == 1)) ### to calculate the percentage of adopted

neigbhours

}

ThModel<-function(node\_seed,network,threshold){

#prepare input for the 'calculate\_value' function#

adj\_matrix <- igraph::as\_adjacency\_matrix(network, type = 'both')

each\_neighbors <- which(adj\_matrix > 0, arr.ind = TRUE)

each\_neighbors <- split(each\_neighbors[, 2], each\_neighbors[, 1]) #get the neigbhour list of each node

nNode<-vcount(network)

node\_status <- rep.int(0, nNode)

neighbour\_status<-rep.int(0, nNode) ##percentage of adopted neighbours

new\_infected <- list()

day\_total\_infected <- rep(0,28) ### Total number of active people by end of each day

### Day 1 ####

day <- 1

node\_status[as.numeric((node\_seed))] <- 1

new\_infected[[day]] <-node\_seed

day\_total\_infected[day]=sum(node\_status == 1)

####

for (day in c(2:28)){

NotAdopted <- which(node\_status == 0)

Adopted <- which(node\_status == 1)

neighbour\_status[NotAdopted] <- unlist(lapply(NotAdopted, calculate\_adoptedNei, node\_status,each\_neighbors))

new\_infected[[day]] <- setdiff(which(neighbour\_status > threshold), Adopted)

node\_status[new\_infected[[day]]] <- 1 #update the staus to 1 for those newly adopted

day\_total\_infected[day] <- sum(node\_status)

day <- day + 1

}

#return(day\_total\_infected)

return(list(day\_total\_infected,new\_infected))

}

ThModel(list('59', '63', '91', '92', '99'), Highschool, highschool\_att$Threshold)

## Question 15

x <- highschool\_att$Threshold

# create a sample dataset

my\_list <- c(rep(0, 15), rep(0.1, 5), rep(0.2, 5), 0.3, rep(0.4, 2), rep(0.5, 2), 0.6, 0.7, 0.8, rep(0.9, 4), rep(1, 85))

# plot the empirical distribution

plot(my\_list, main="Empirical Distribution")

# plot the uni distribution

plot(sort(x), main="Original Threshold Distribution")

highschool\_att$Emp <- sample(my\_list)

threshold\_model(Highschool, list('59', '63', '91', '92', '99'), highschool\_att$Emp, 28)

#ThModel(list('59', '63', '91', '92', '99'), Highschool, highschool\_att$Emp)

# Run the simulation n times and store the results in a vector

n <- 1000

sum <- replicate(n, threshold\_model(Highschool, list('59', '63', '91', '92', '99'), highschool\_att$Emp, 28))

# Calculate the mean of the results

mean(sum)

EXERCISE 4

## Question 17

{r, message=F}

stopifnot(require(data.table))

stopifnot(require(Matrix))

calculate\_value <- function(node, each\_neighbors,Pprob){

return(each\_neighbors[[node]][ which(runif(length(each\_neighbors[[node]]), 0, 1)<=Pprob)])

#'runif' is a function to generate random number in R

}

#This function:

#1) searches the neighbors of contagious node;

#2) To those who are connected to a contagious node, generates a random number and compare to the

#probability of p, if random number<p, this node will be infected and return the value of 1

IC\_immunized<-function(node\_seed,network,Pprob, immunized = c(), n\_days = 28 ){

#prepare input for the 'calculate\_value' function#

adj\_matrix <- igraph::as\_adjacency\_matrix(network, type = 'both')

each\_neighbors <- which(adj\_matrix > 0, arr.ind = TRUE)

each\_neighbors <- split(each\_neighbors[, 2], each\_neighbors[, 1]) #get the neigbhour list of each node

nNode<-vcount(network)

node\_status <- rep.int(0, nNode) #start from a healthy population

day\_infected<-vector()#Total number of infected population

new\_infected <- list() # Record the ID of person getting infected at each time step

day<-1

node\_status[as.numeric(node\_seed)] <- 1 # infected(value=1) health(value=0)

day\_infected[day] <- sum(node\_status)

new\_infected[[day]] <- node\_seed #The ID of the person infected in Day 1 (Patient Zero)

infected\_number = 0

number\_day <- c()

#simulate the spread of virus within 4 weeks##

for (day in c(2:n\_days)){

ContagiousID<-which(node\_status == 1)

infectedID<-unlist(lapply(ContagiousID,calculate\_value,each\_neighbors,Pprob))

infectedID <- setdiff(infectedID, immunized)

newinfectedID<- setdiff(infectedID, which(node\_status == 1))

#Update the node status and other variables

node\_status[newinfectedID] <- 1

day\_infected[day] <- length(newinfectedID)

infected\_number = infected\_number + length(newinfectedID)

new\_infected[[day]]<-newinfectedID

number\_day <- append(number\_day, infected\_number)

day = day+1

}

return(day\_infected ) #return the number of newly infected people by day

#return(list(day\_infected,new\_infected)) #if you want to see the ID of infected ppl in each day,use this command instead

}

degree\_n = (degree(Highschool, mode = "all")) %>% sort(T)

degree\_n[1:3]

betweenness\_n = (betweenness(Highschool)) %>% sort(T)

betweenness\_n[1:3]

no\_immune <- IC\_immunized("107", Highschool, 0.1)

degree\_immune <- IC\_immunized("107", Highschool, 0.1, c(54,20, 110))

betweenness\_immune <- IC\_immunized("107", Highschool, 0.1, c(37,4, 96 ))

days <- seq(1:28)

immune\_df <- data.frame(days= days,

not\_immune = no\_immune,

degree\_immuned = degree\_immune,

betweenness\_immuned = betweenness\_immune)

data\_long <- melt(immune\_df, id = "days")

sum(no\_immune)

sum(degree\_immune)

sum(betweenness\_immune)

final\_number <- function(network1, network2, network3, p=0.8){

sum1 = 0

sum2= 0

sum3= 0

node\_number <- paste(sample(1:300, 1))

for (i in 0:99){

no\_immune <- IC\_immunized(node\_number, network1, p)

degree\_immune <- IC\_immunized(node\_number, network2, p, c(54,20, 110))

betweenness\_immune <- IC\_immunized(node\_number, network3, p, c(37,4, 96 ))

sum1= sum1 + sum(no\_immune)

sum2= sum2 + sum(degree\_immune)

sum3= sum3 + sum(betweenness\_immune)

}

sum1 = as.integer(sum1/100)

sum2 = as.integer(sum2/100)

sum3 = as.integer(sum3/100)

len <- 1:length(sum1)-1

return(c(sum1,sum2,sum3))}

infected <- function(network1, network2, network3,network4, p=0.1){

tel1 = 0

tel2 = 0

tel3 = 0

tel4 = 0

for (i in 0:99){

tel1 = tel1 + IC\_immunized("107", network1, p,c(), 28)

tel2 = tel2 + IC\_immunized("107", network2, p, c(54,20, 110), 28)

tel3 = tel3 + IC\_immunized("107", network3, p, c(37,4, 96 ), 28)

tel4 = tel4 + IC\_immunized("107", network4, p, c(36, 118, 97), 28)

}

tel1 = as.integer(tel1/100)

tel2 = as.integer(tel2/100)

tel3 = as.integer(tel3/100)

tel4 = as.integer(tel4/100)

len <- 1:length(tel1)-1

ggplot() + geom\_smooth(aes(len, tel1, colour='No Immunity'), se =F) +

geom\_smooth(aes(len, tel2, colour='Degree heuristic'), se =F) +

geom\_smooth(aes(len, tel3, colour='Betweenness heuristic'), se =F) +

scale\_fill\_discrete(labels=c('Highschool', 'Highschool2', 'Highschool3'))+

labs(x = "Days", y = "Average Number of infected nodes", title = paste("Average Number of infected nodes in Barabasi model")

)+

theme\_minimal()

}

infected(Highschool, Highschool, Highschool, 0.15)

SW3<-watts.strogatz.game(dim=1,size=300,nei=6, p=0.01)

plot(SW3, layout=layout.circle, vertex.label=NA, vertex.size=5,

main= "Network with 0.1 rewiring probability ")

g0 <- barabasi.game(300, power = 0.5 , m = NULL, out.dist = NULL, out.seq = NULL, out.pref = FALSE, zero.appeal = 1, directed = FALSE,algorithm ="psumtree", start.graph = NULL)

g1 <- barabasi.game(300, power = 1 , m = NULL, out.dist = NULL, out.seq = NULL, out.pref = FALSE, zero.appeal = 1, directed = FALSE,algorithm ="psumtree", start.graph = NULL)

g1\_5 <- barabasi.game(300, power = 1.5 , m = NULL, out.dist = NULL, out.seq = NULL, out.pref = FALSE, zero.appeal = 1, directed = FALSE,algorithm ="psumtree", start.graph = NULL)

g2 <- barabasi.game(300, power = 2 , m = NULL, out.dist = NULL, out.seq = NULL, out.pref = FALSE, zero.appeal = 1, directed = FALSE,algorithm ="psumtree", start.graph = NULL)

infected\_random <- function(network1, network2, network3, p=0.8){

tel1 = 0

tel2 = 0

tel3 = 0

node\_number <- paste(sample(1:300, 1))

for (i in 0:99){

tel1 = tel1 + IC\_immunized(node\_number, network1, p,c(), 40)

tel2 = tel2 + IC\_immunized(node\_number, network2, p, c(54,20, 110), 40)

tel3 = tel3 + IC\_immunized(node\_number, network3, p, c(37,4, 96 ), 40)

}

tel1 = as.integer(tel1/100)

tel2 = as.integer(tel2/100)

tel3 = as.integer(tel3/100)

len <- 1:length(tel1)-1

ggplot() + geom\_smooth(aes(len, tel1, colour='No Immunity'), se =F) +

geom\_smooth(aes(len, tel2, colour='Degree heuristic'), se =F) +

geom\_smooth(aes(len, tel3, colour='Betweenness heuristic'), se =F) +

scale\_fill\_discrete(labels=c('Highschool', 'Highschool2', 'Highschool3'))+

labs(x = "Days", y = "Average Number of infected nodes", title = paste("Average Number of infected nodes in Barabasi model")

)+

theme\_minimal()

}c

c <- c()

for (i in seq(0, 0.5, 0.1)){

c <-append(c, final\_number(SW3,SW3, SW3,p=i))

}

c

infected\_random(SW3, SW3, SW3, 0.06)

# Question 18

# Checking the greedy

final\_number\_greedy <- function(node\_number, network1, network2, network3, network4, p=0.15){

sum1 = 0

sum2 = 0

sum3 = 0

sum4 = 0

# node\_number <- paste(sample(1:300, 1))

for (i in 0:99){

no\_immune <- IC\_immunized(node\_number, network1, p)

degree\_immune <- IC\_immunized(node\_number, network2, p, c(54,20, 110))

betweenness\_immune <- IC\_immunized(node\_number, network3, p, c(37,4, 96 ))

greedy = IC\_immunized(node\_number, network4, p, c(37, 21, 9), 28)

sum1= sum1 + sum(no\_immune)

sum2= sum2 + sum(degree\_immune)

sum3= sum3 + sum(betweenness\_immune)

sum4= sum4 + sum(betweenness\_immune)

}

sum1 = as.integer(sum1/100)

sum2 = as.integer(sum2/100)

sum3 = as.integer(sum3/100)

sum4 = as.integer(sum4/100)

len <- 1:length(sum1)-1

return(c(sum1,sum2,sum3, sum4))

}

infected\_greedy <- function(node\_number, network1, network2, network3, network4, p=0.15){

tel1 = 0

tel2 = 0

tel3 = 0

tel4 = 0

# node\_number <- paste(sample(1:122, 1))

for (i in 0:100){

tel1 = tel1 + IC\_immunized(node\_number, network1, p, c(), 28)

tel2 = tel2 + IC\_immunized(node\_number, network2, p, c(54,20, 110), 28)

tel3 = tel3 + IC\_immunized(node\_number, network3, p, c(37, 4, 96 ), 28)

tel4 = tel4 + IC\_immunized(node\_number, network4, p, c(96, 37, 70), 28)

}

tel1 = as.integer(tel1/100)

tel2 = as.integer(tel2/100)

tel3 = as.integer(tel3/100)

tel4 = as.integer(tel4/100)

len <- 1:length(tel1)-1

ggplot() + geom\_smooth(aes(len, tel1, colour='No Immunity'), se =F) +

geom\_smooth(aes(len, tel2, colour='Degree heuristic'), se =F) +

geom\_smooth(aes(len, tel3, colour='Betweenness heuristic'), se =F) +

geom\_smooth(aes(len, tel4, colour='Greedy'), se =F) +

scale\_fill\_discrete(labels=c('Highschool', 'Highschool2', 'Highschool3'))+

labs(x = "Days", y = "Average Number of infected nodes", title = paste("Average Number of infected nodes of 100 MC simulations in Highschool network")

)+

theme\_minimal()

}

final\_number\_greedy(107, Highschool, Highschool, Highschool, Highschool,p = 0.06)

infected\_greedy(107, Highschool, Highschool, Highschool, Highschool, p = 0.15)

From now we start with Python

First come the functions then the code

import matplotlib.pyplot as plt

from random import uniform, seed

import numpy as np

import pandas as pd

import csv

import time

from igraph import \*

def compute\_mean\_by\_index(list\_of\_lists):

# Get the number of inner lists

num\_lists = len(list\_of\_lists)

# Get the length of each inner list

list\_length = len(list\_of\_lists[0])

# Initialize a list to store the means

means = []

# Iterate through the indices

for i in range(list\_length):

# Initialize a sum variable for the current index

sum\_for\_index = 0

# Iterate through the inner lists

for j in range(num\_lists):

# Add the value at the current index to the sum

sum\_for\_index += list\_of\_lists[j][i]

# Compute the mean for the current index

mean\_for\_index = sum\_for\_index / num\_lists

# Append the mean to the result list

means.append(mean\_for\_index)

return means

def IC(g,S,p=0.5,mc=1000, timestamps = 28, Monte\_Carlo = True):

"""

Input:

g - graph object,

S - set of seed nodes(dtype list)

p - propagation probability

mc - the number of Monte-Carlo simulations

timestamps - the number of timestamps

Monte\_Carlo - Boolean, determines whether to make to random seed or not

Output:

- average number of nodes activated in each Monte-Carlo simulation

- average number of nodes influenced by the seed nodes in each timestamp

"""

# Loop over the Monte-Carlo Simulations

spread, sum\_spread = [], []

for i in range(mc):

# Simulate propagation process

Active = S[:]

sum = 0

for \_ in range(timestamps):

new\_ones = []

# For each active node, find its neighbors that become activated

for node in Active:

if Monte\_Carlo == True:

np.random.seed(i)

# Determine neighbors that become infected

success = np.random.uniform(0,1,len(g.neighbors(node,mode="out"))) < p

new\_ones += list(np.extract(success, g.neighbors(node,mode="out")))

new\_active = list(set(new\_ones) - set(Active))

sum += len(new\_active)

# in case the network is fully activated

if new\_active == []:

break

# Add newly activated nodes to the set of activated nodes

Active = list(set(Active).union(set(new\_active)))

spread.append(len(Active))

sum\_spread.append(sum)

return(np.mean(spread), np.mean(sum\_spread))

def IC\_immunized(g, S, immunized, p=0.15, mc=1000, timestamps = 28, full = False, Monte\_Carlo = True):

"""

Input:

g - graph object,

S - set of seed nodes(dtype list)

immunized - set of immunized nodes(dtype list)

p - propagation probability

mc - the number of Monte-Carlo simulations

timestamps - the number of timestamps

Monte\_Carlo - Boolean, determines whether to make to random seed or not

Output:

- average number of nodes activated in each Monte-Carlo simulation

- average number of nodes influenced by the seed nodes in each timestamp

"""

# Loop over the Monte-Carlo Simulations

spread, sum\_spread, infected\_a\_day = [], [], []

for i in range(mc):

S = list(set(S).difference(set(immunized)))

# Simulate propagation process

Active = S[:]

sum = 0

infected\_per\_day = []

for \_ in range(timestamps):

new\_ones = []

# For each active node, find its neighbors that become activated

for node in Active:

if Monte\_Carlo == True:

np.random.seed(i)

# Determine neighbors that become infected

success = np.random.uniform(0,1, len(g.neighbors(node,mode="out"))) <= p

new\_ones += list(np.extract(success, g.neighbors(node,mode="out")))

new\_ones = list(set(new\_ones).difference(set(immunized)))

new\_active = list(set(new\_ones).difference(set(Active)))

sum += len(Active)

if new\_active == [] and full == True:

break

infected\_per\_day.append(len(new\_active))

# Add newly activated nodes to the set of activated nodes

Active = list(set(Active).union(set(new\_active)))

spread.append(len(Active))

sum\_spread.append(sum)

infected\_a\_day.append(infected\_per\_day)

return(np.mean(spread), np.mean(sum\_spread), compute\_mean\_by\_index(infected\_a\_day))

def greedy(g, k, p=0.15, mc=1000, timestamps = 28):

"""

Input:

g - graph object

k - number of seed nodes

p - propagation probability

mc - the number of Monte-Carlo simulations

timestamps - the number of timestamps

Monte\_Carlo - Boolean, determines whether to make to random seed or not

Output: optimal seed set, resulting spread, time for each iteration

"""

S, spread, timelapse, start\_time = [], [], [], time.time()

# setting list of nodes and making the node 107 the first node in odrer to compare with other methods

l = [i for i in range(g.vcount())]

l.remove(106)

l.insert(0, 106)

# Find k nodes with largest marginal gain

for \_ in range(k):

# Loop over nodes that are not yet in seed set to find biggest marginal gain

best\_spread = 0

#for j in set(range(g.vcount())) - set(S):

for j in set(l) - set(S):

# Get the spread

s = IC(g, S + [j], p, mc, timestamps, Monte\_Carlo = True)

# Update the winning node and spread so far

if s[1] > best\_spread:

best\_spread, node = s[1], j

# Add the selected node to the seed set

S.append(node)

# Add estimated spread and elapsed time

spread.append([best\_spread, s[1]])

timelapse.append(time.time() - start\_time)

return(S,spread, timelapse)

def greedy\_immunized(g,k,p=0.1,mc=1000):

"""

g - graph object

k - number of seed nodes

p - propagation probability

mc - the number of Monte-Carlo simulations

timestamps - the number of timestamps

Monte\_Carlo - Boolean, determines whether to make to random seed or not

Output: optimal seed set, resulting spread, time for each iteration

"""

immunaized\_list, spread, timelapse, start\_time = [],[], [], time.time()

# Find k nodes with largest marginal gain

for \_ in range(k):

# Loop over nodes that are not yet in seed set to find biggest marginal gain

best\_spread = IC\_immunized(g, [106],[], p, mc, timestamps = 28, full = True, Monte\_Carlo = True)[1]

for j in set(range(g.vcount())) - set(immunaized\_list):

# Get the spread

s = IC\_immunized(g, [106], immunaized\_list + [j], p, mc, timestamps=28, full = True, Monte\_Carlo = True)

# Update the winning node and spread so far

if s[1] < best\_spread and s[1] !=0 :

best\_spread, node = s[1], j

# Add the selected node to the seed set

if 'node' in locals():

immunaized\_list.append(node)

# Add estimated spread and elapsed time

spread.append(best\_spread)

timelapse.append(time.time() - start\_time)

return(immunaized\_list, spread, timelapse)

def threshold\_model(network, node\_seed, threshold, n\_day, mc=100, Monte\_Carlo=False):

spread, sum\_spread, infected\_per\_day, infected\_per\_iteration = [], [], [], []

for i in range(mc):

if Monte\_Carlo == True:

np.random.seed(i)

nNode = network.vcount()

node\_status = np.zeros(nNode, dtype=int) # start from a healthy population

adj\_matrix = np.array(network.get\_adjacency().data)

each\_neighbors = {node: np.where(adj\_matrix[node, :] > 0)[0] for node in range(nNode)} # get the neighbor list of each node

infected\_a\_day,infected\_List = [] , []

for seed in node\_seed:

node\_status[seed] = 1 # adopt (value=1), don't adopt (value=0)

sum\_of\_ifected = 0

for day in range(1, 29): #n\_day + 1):

n\_infected = 0

for node in range(nNode):

if node\_status[node] == 0:

neighbours = each\_neighbors[node]

n\_neighbors = len(neighbours)

n\_adopters = np.sum(node\_status[neighbours] == 1)

#if n\_adopters/n\_neighbors > threshold[node]:

if n\_adopters > threshold[node]:

node\_status[node] = 1

n\_infected += 1

# if n\_infected == 0:

# break

infected\_at\_all = np.sum(node\_status == 1)

sum\_of\_ifected += np.sum(node\_status == 1)

infected\_a\_day.append(n\_infected)

infected\_List.append(infected\_at\_all)

# Collect all of the necessary information

spread\_per\_simulation = np.sum(node\_status == 1)

spread.append(spread\_per\_simulation)

sum\_spread.append(sum\_of\_ifected)

infected\_per\_day.append(infected\_a\_day)

infected\_per\_iteration.append(infected\_List)

return np.mean(spread), np.mean(sum\_spread), compute\_mean\_by\_index(infected\_per\_day), compute\_mean\_by\_index(infected\_per\_iteration)

def greedy\_Th(g, k, threshold, mc=1000, timestamps = 28):

"""

Input:

g - graph object

k - number of seed nodes

p - propagation probability

mc - the number of Monte-Carlo simulations

timestamps - the number of timestamps

Monte\_Carlo - Boolean, determines whether to make to random seed or not

Output: optimal seed set, resulting spread, time for each iteration

"""

# Threshold\_list = [0] \* 15 + [0.1] \* 5 + [0.2] \* 5 + [0.3] + [0.4] \* 2 + [0.5] \* 2 + [0.6, 0.7, 0.8] + [0.9] \* 4 + [1] \* 85

# # shuffle the list

# np.random.shuffle(Threshold\_list)

S, spread, timelapse, start\_time = [], [], [], time.time()

# setting list of nodes and making the node 107 the first node in odrer to compare with other methods

# l = [i for i in range(g.vcount())]

# l.remove(106)

# l.insert(0, 106)

# Find k nodes with largest marginal gain

for \_ in range(k):

# Loop over nodes that are not yet in seed set to find biggest marginal gain

best\_spread = 0

for j in set(range(g.vcount())) - set(S):

#for j in set(l) - set(S):

# Get the spread

s = threshold\_model(g, S + [j], threshold, 28, mc, Monte\_Carlo = True)

# Update the winning node and spread so far

if s[1] > best\_spread:

best\_spread, node = s[1], j

# Add the selected node to the seed set

S.append(node)

# Add estimated spread and elapsed time

spread.append(best\_spread)

timelapse.append(time.time() - start\_time)

return(S,spread, timelapse)

def greedy\_immunized\_Th(g, k, threshold=0.1 ,mc=1000):

"""

g - graph object

k - number of seed nodes

p - propagation probability

mc - the number of Monte-Carlo simulations

timestamps - the number of timestamps

Monte\_Carlo - Boolean, determines whether to make to random seed or not

Output: optimal seed set, resulting spread, time for each iteration

"""

immunaized\_list, spread, timelapse, start\_time = [],[], [], time.time()

# Find k nodes with largest marginal gain

for \_ in range(k):

# Loop over nodes that are not yet in seed set to find biggest marginal gain

best\_spread = threshold\_model([106], g, threshold, [], mc, Monte\_Carlo = True)[1]

for j in set(range(g.vcount())) - set(immunaized\_list):

# Get the spread

s = threshold\_model([106], g, threshold, immunaized\_list + [j], mc, Monte\_Carlo = True)

# Update the winning node and spread so far

if s[1] < best\_spread and s[1] !=0 :

best\_spread, node = s[1], j

# Add the selected node to the seed set

if 'node' in locals():

immunaized\_list.append(node)

# Add estimated spread and elapsed time

spread.append(best\_spread)

timelapse.append(time.time() - start\_time)

return(immunaized\_list, spread, timelapse)

def celf(g,k,threshold,mc):

"""

Input: graph object, number of seed nodes

Output: optimal seed set, resulting spread, time for each iteration

"""

# --------------------

# Find the first node with greedy algorithm

# --------------------

# Calculate the first iteration sorted list

start\_time = time.time()

marg\_gain = [threshold\_model(g,[node], threshold, mc)[0] for node in range(g.vcount())]

# Create the sorted list of nodes and their marginal gain

Q = sorted(zip(range(g.vcount()), marg\_gain), key=lambda x: x[1],reverse=True)

# Select the first node and remove from candidate list

S, spread, SPREAD = [Q[0][0]], Q[0][1], [Q[0][1]]

Q, timelapse = Q[1:], [time.time()-start\_time]

# LOOKUPS = [g.vcount()]

# --------------------

# Find the next k-1 nodes using the list-sorting procedure

# --------------------

for \_ in range(k-1):

check, node\_lookup = False, 0

while not check:

# Count the number of times the spread is computed

node\_lookup += 1

# Recalculate spread of top node

current = Q[0][0]

# Evaluate the spread function and store the marginal gain in the list

Q[0] = (current, threshold\_model(g, S + [current], threshold, mc)[0] - spread)

# print(Q)

# Re-sort the list

Q = sorted(Q, key = lambda x: x[1], reverse = True)

# Check if previous top node stayed on top after the sort

check = (Q[0][0] == current)

# Select the next node

spread += Q[0][1]

S.append(Q[0][0])

SPREAD.append(spread)

# LOOKUPS.append(node\_lookup)

timelapse.append(time.time() - start\_time)

# Remove the selected node from the list

Q = Q[1:]

return(S,SPREAD,timelapse) #,LOOKUPS)

# Import packages

%matplotlib inline

import matplotlib.pyplot as plt

from random import uniform, seed

import numpy as np

import pandas as pd

import csv

import time

from igraph import \*

from functions\_EX4 import \*

with open('Highschool\_network\_edge.csv') as csvfile:

reader = csv.reader(csvfile)

edges = [(int(row[0][1:]), int(row[1][1:])) for row in reader]

NodeID = []

Gender = []

Hall = []

Threshold = []

with open('Highschool\_network\_att.csv') as csvfile:

reader = csv.reader(csvfile)

next(reader)

for row in reader:

NodeID.append(int(row[0][1:]))

Gender.append(row[1])

Hall.append(row[2])

Threshold.append(row[3])

# g.vs["Gender"] = Gender

# g.vs["Hall"] = Hall

# g.vs["Threshold"] = Threshold

g = Graph(edges, directed=False)

g.delete\_vertices(0)

# Number of components

g.vs["NodeID"] = NodeID

print("Number of components: ", g.vcount())

plot(g, bbox = (300,300), margin = 20, layout = g.layout("auto"), vertex\_size = 1, vertex\_color = "white", vertex\_label = g.vs["NodeID"])

greedy\_immunized(g, 3, 0.15, mc=600)

greedy(g, 3, 0.15, mc=200) # works to slow

# creating the list of infected nodes per day

No\_immunity = IC\_immunized(g, [106], [], p = 0.15, mc = 100, timestamps = 28, Monte\_Carlo=False)

Degree = IC\_immunized(g, [106], [53, 19, 109], p = 0.15, mc = 100, timestamps = 28, Monte\_Carlo=False)

Betweenness = IC\_immunized(g, [106], [36, 3, 95], p = 0.15, mc = 100, timestamps = 28, Monte\_Carlo=False)

Greedy = IC\_immunized(g, [106], [53, 91, 72], p = 0.15, mc = 100, timestamps = 28, Monte\_Carlo=False)

Greedy2 = IC\_immunized(g, [106], [95, 36, 69], p = 0.15, mc = 100, timestamps = 28 , Monte\_Carlo=False)

# Creating a list of days as x-axis

days = list(range(1, len(Greedy[2]) + 1))

# Plotting the three lists

plt.plot(days, No\_immunity[2], label='No\_immunity')

plt.plot(days, Degree[2], label='Degree heuristic')

plt.plot(days, Betweenness[2], label='Betweenness heuristic')

plt.plot(days, Greedy[2], label='Greedy with Influence Maximization')

plt.plot(days, Greedy2[2], label='Greedy with Influence Minimization')

# Adding labels and title

plt.xlabel('Days')

plt.ylabel('Number of Infected')

plt.title('Daily Infected Cases')

# Adding legend

plt.legend()

# Displaying the plot

plt.show()

# Question 19

# Original dictionary with values

original\_dict = {0:15, 1:5, 2:5, 3:1, 4:2, 5:2, 6:1, 7:1, 8:1, 9:0, 10:4}

# Calculate total sum of original values

total\_sum = sum(original\_dict.values())

# Desired total sum

desired\_sum = 122

# Scale factor

scale\_factor = desired\_sum / total\_sum

# Create a new dictionary to store the scaled values

scaled\_dict = {}

# Scale up each value proportionally and round to nearest integer

for key, value in original\_dict.items():

scaled\_value = round(value \* scale\_factor)

scaled\_dict[key] = scaled\_value

# Check if the total sum of scaled values is less than the desired sum

# If so, add the difference to the highest key (">=10")

total\_scaled\_sum = sum(scaled\_dict.values())

if total\_scaled\_sum < desired\_sum:

diff = desired\_sum - total\_scaled\_sum

scaled\_dict[10] += diff

Threshold\_list = []

for key, value in scaled\_dict.items():

Threshold\_list.extend([key] \* value)

# Shuffle the list to randomize the distribution

np.random.shuffle(Threshold\_list)

greedy\_Th(g, 7, Threshold\_list, mc=100)

degree\_centrality = g.degree()

sorted\_nodes = sorted(range(len(degree\_centrality)), key=degree\_centrality.\_\_getitem\_\_, reverse=True)

top\_7\_nodes = sorted\_nodes[:7]

print("Top 7 Nodes based on Degree Centrality: ", top\_7\_nodes)

betweenness\_centrality = g.betweenness()

sorted\_nodes = sorted(range(len(degree\_centrality)), key=betweenness\_centrality.\_\_getitem\_\_, reverse=True)

top\_7\_nodes = sorted\_nodes[:7]

print("Top 7 Nodes based on Betweenness Centrality: ", top\_7\_nodes)

Degree = threshold\_model(g, [53, 19, 109, 63, 95, 27, 36], Threshold\_list,28, mc=100, Monte\_Carlo = True)

Betweenness = threshold\_model(g, [36, 3, 95, 23, 69, 76, 16], Threshold\_list, 28, mc=100, Monte\_Carlo = True)

Greedy = threshold\_model(g, [41, 107, 60, 21, 30, 45, 42], Threshold\_list, 28, mc=100, Monte\_Carlo = True)

Celf = threshold\_model(g, [3, 67, 102, 96, 2, 55, 75], Threshold\_list, 28, mc=100, Monte\_Carlo = True)

Page\_rank = threshold\_model(g, [53, 19, 63, 95, 27, 23, 109], Threshold\_list, 28, mc=100, Monte\_Carlo = True)

# Creating a list of days as x-axis

days = list(range(1, len(Greedy[3]) + 1))

# Plotting the three list

plt.plot(days, Degree[3], label='Degree heuristic')

plt.plot(days, Betweenness[3], label='Betweenness heuristic')

plt.plot(days, Greedy[3], label='Greedy with Influence Maximization')

plt.plot(days, Page\_rank[3], label='Page\_rank')

# Adding labels and title

plt.xlabel('Days')

plt.ylabel('Number of Infected')

plt.title('Infected Cases')

# Setting x-axis limit to show only 0 to 5 days

plt.xlim(0, 10)

# Adding legend

plt.legend()

# Displaying the plot

plt.show()

# Question 20

# Run algorithms

celf\_output = celf(g,7,Threshold\_list,mc = 50)

greedy\_output = greedy\_Th(g,7,Threshold\_list, mc = 50)

# Print resulting seed sets

print("celf output: " + str(celf\_output[0]))

print("greedy output: " + str(greedy\_output[0]))

# Plot settings

plt.rcParams['figure.figsize'] = (9,6)

plt.rcParams['lines.linewidth'] = 4

plt.rcParams['xtick.bottom'] = False

plt.rcParams['ytick.left'] = False

# Plot Computation Time

plt.plot(range(1,len(greedy\_output[2])+1),greedy\_output[2],label="Greedy",color="#FBB4AE")

plt.plot(range(1,len(celf\_output[2])+1),celf\_output[2],label="CELF",color="#B3CDE3")

plt.ylabel('Computation Time (Seconds)'); plt.xlabel('Size of Seed Set')

plt.title('Computation Time'); plt.legend(loc=2)

Degree = threshold\_model(g, [53, 19, 109, 63, 95, 27, 36], Threshold\_list,28, mc=100, Monte\_Carlo = True)

Betweenness = threshold\_model(g, [36, 3, 95, 23, 69, 76, 16], Threshold\_list, 28, mc=100, Monte\_Carlo = True)

Greedy = threshold\_model(g, [41, 107, 60, 21, 30, 45, 42], Threshold\_list, 28, mc=100, Monte\_Carlo = True)

Celf = threshold\_model(g, [3, 67, 102, 96, 2, 55, 75], Threshold\_list, 28, mc=100, Monte\_Carlo = True)

Page\_rank = threshold\_model(g, [53, 19, 63, 95, 27, 23, 109], Threshold\_list, 28, mc=100, Monte\_Carlo = True)

# Creating a list of days as x-axis

days = list(range(1, len(Greedy[3]) + 1))

# Plotting the three list

plt.plot(days, Degree[3], label='Degree heuristic')

plt.plot(days, Betweenness[3], label='Betweenness heuristic')

plt.plot(days, Greedy[3], label='Greedy with Influence Maximization')

plt.plot(days, Page\_rank[3], label='Page\_rank')

plt.plot(days, Celf[3], label='Celf')

# Adding labels and title

plt.xlabel('Days')

plt.ylabel('Number of Infected')

plt.title('Infected Cases')

# Setting x-axis limit to show only 0 to 5 days

plt.xlim(0, 10)

# Adding legend

plt.legend()

# Displaying the plot

plt.show()

with open('soc-sign-bitcoinalpha.csv') as csvfile:

reader = csv.reader(csvfile)

edges = [(int(row[0]), int(row[1])) for row in reader]

# NodeID = []

# Gender = []

# Hall = []

# Threshold = []

# with open('Highschool\_network\_att.csv') as csvfile:

# reader = csv.reader(csvfile)

# next(reader)

# for row in reader:

# NodeID.append(int(row[0][1:]))

# Gender.append(row[1])

# Hall.append(row[2])

# Threshold.append(row[3])

bitcoinalpha = Graph(edges, directed=False)

# create a connected subset of the graph with 1001 nodes

bitcoinalpha = bitcoinalpha.subgraph(range(1010))

# count the nuber of edges in the subgraph

bitcoinalpha.ecount()

# delete those nodes without an edge

bitcoinalpha.delete\_vertices(bitcoinalpha.vs.select(\_degree\_eq=0))

# count the number of edges in the subgraph

bitcoinalpha.vcount()

# Desired total sum

desired\_sum = 999

# Scale factor

scale\_factor = desired\_sum / total\_sum

# Create a new dictionary to store the scaled values

scaled\_dict = {}

# Scale up each value proportionally and round to nearest integer

for key, value in original\_dict.items():

scaled\_value = round(value \* scale\_factor)

scaled\_dict[key] = scaled\_value

# Check if the total sum of scaled values is less than the desired sum

# If so, add the difference to the highest key (">=10")

total\_scaled\_sum = sum(scaled\_dict.values())

if total\_scaled\_sum < desired\_sum:

diff = desired\_sum - total\_scaled\_sum

scaled\_dict[10] += diff

Threshold\_for\_bitcoing = []

for key, value in scaled\_dict.items():

Threshold\_for\_bitcoing.extend([key] \* value)

# Shuffle the list to randomize the distribution

np.random.shuffle(Threshold\_for\_bitcoing)

bitcoinalpha

# Plot Graph

bitcoinalpha.es["color"], bitcoinalpha.vs["color"], bitcoinalpha.vs["label"] = "#B3CDE3", "#FBB4AE", ""

plot(bitcoinalpha, bbox = (300, 300),margin = 11,layout = bitcoinalpha.layout("kk"))

# Run algorithms

celf\_output = celf(bitcoinalpha,7,Threshold\_for\_bitcoing,mc = 1)

greedy\_output = greedy\_Th(bitcoinalpha,7,Threshold\_for\_bitcoing, mc = 1)

# Print resulting seed sets

print("celf output: " + str(celf\_output[0]))

#print("greedy output: " + str(greedy\_output[0]))

# 137, 431, 487, 479, 260, 238, 485,

# Plot settings

plt.rcParams['figure.figsize'] = (9,6)

plt.rcParams['lines.linewidth'] = 4

plt.rcParams['xtick.bottom'] = False

plt.rcParams['ytick.left'] = False

# Plot Computation Time

plt.plot(range(1,len(greedy\_output[2])+1),greedy\_output[2],label="Greedy",color="#FBB4AE")

plt.plot(range(1,len(celf\_output[2])+1),celf\_output[2],label="CELF",color="#B3CDE3")

plt.ylabel('Computation Time (Seconds)'); plt.xlabel('Size of Seed Set')

plt.title('Computation Time'); plt.legend(loc=2)

pagerank\_centrality = bitcoinalpha.pagerank()

sorted\_nodes = sorted(range(len(pagerank\_centrality)), key=pagerank\_centrality.\_\_getitem\_\_, reverse=True)

top\_7\_nodes = sorted\_nodes[:7]

print("Top 7 Nodes based on Betweenness Centrality: ", top\_7\_nodes)

# bitcoinalpha.pagerank()

#53, 19, 63, 95, 27, 23, 109

# 137, 431, 487, 479, 260, 238, 485

degree\_centrality = bitcoinalpha.degree()

sorted\_nodes = sorted(range(len(degree\_centrality)), key=degree\_centrality.\_\_getitem\_\_, reverse=True)

top\_7\_nodes = sorted\_nodes[:7]

print("Top 7 Nodes based on Degree Centrality: ", top\_7\_nodes)

betweenness\_centrality = bitcoinalpha.betweenness()

sorted\_nodes = sorted(range(len(degree\_centrality)), key=betweenness\_centrality.\_\_getitem\_\_, reverse=True)

top\_7\_nodes = sorted\_nodes[:7]

print("Top 7 Nodes based on Betweenness Centrality: ", top\_7\_nodes)

Degree = threshold\_model(bitcoinalpha, [53, 19, 109, 63, 95, 27, 36], Threshold\_list,28, mc=100, Monte\_Carlo = True)

Betweenness = threshold\_model(bitcoinalpha, [36, 3, 95, 23, 69, 76, 16], Threshold\_list, 28, mc=100, Monte\_Carlo = True)

Greedy = threshold\_model(bitcoinalpha, [41, 107, 60, 21, 30, 45, 42], Threshold\_list, 28, mc=100, Monte\_Carlo = True)

Celf = threshold\_model(bitcoinalpha, [3, 67, 102, 96, 2, 55, 75], Threshold\_list, 28, mc=100, Monte\_Carlo = True)

Page\_rank = threshold\_model(bitcoinalpha, [53, 19, 63, 95, 27, 23, 109], Threshold\_list, 28, mc=100, Monte\_Carlo = True)

# Creating a list of days as x-axis

days = list(range(1, len(Greedy[3]) + 1))

# Plotting the three list

plt.plot(days, Degree[3], label='Degree heuristic')

plt.plot(days, Betweenness[3], label='Betweenness heuristic')

plt.plot(days, Greedy[3], label='Greedy with Influence Maximization')

plt.plot(days, Page\_rank[3], label='Page\_rank')

plt.plot(days, Celf[3], label='Celf')

# Adding labels and title

plt.xlabel('Days')

plt.ylabel('Number of Infected')

plt.title('Infected Cases')

# Setting x-axis limit to show only 0 to 5 days

plt.xlim(0, 10)

# Adding legend

plt.legend()

# Displaying the plot

plt.show()