# IS609 - Networked Game Theory - Final Project

*J. Hamski, S. Hong, H. Berk*

*April 9, 2016*

## Introduction

Game theory is applied in economics, political science, biology and a range of other discriplines to study how rational decision makers interact. The canonical teaching example in game theory is the Prisoner's Dilemma (PD). In this game, two players are separated and cannot communicate. They are told that if they give up the other player (defect) they will get a reduced sentence, but if they don't give up the other player (cooperate) they may serve even less time. However, if they cooperate and the other player defects, they will go to jail for an even greater amount of time than if they would have defected. This scenario is often extended to multiple games (iterated Prisoner's Dilemma).

Two things stick out when making the leap from a toy example like the Prisoner's Dilemma and a real-world example like a market:
(A) Most market participants don't play multiple games against the same opponent, they play only one or perhaps several over time.
(B) Most market participants don't play against just one opponent, they play against several.

These "n-person" games can utilize Graph Theory to allow for games to be played on networks that reflect realistic human relationships. This project has two sections. First, I developed functions to play the Prisoner's Dilemma on any graph where the nodes have two degrees. Second, I developed an game that can be played on any graph, where low-degree players can defect, influencing their neighboring nodes to be more likely to defect.

## Prisoner's Dilemma on a Graph.

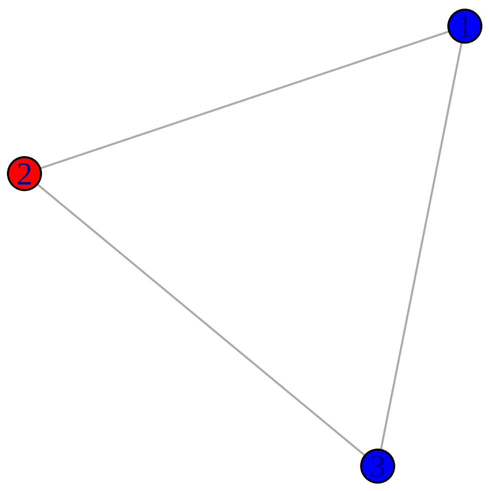
The two player PD may be modeled as two nodes connected by a single edge. Extending to three players, we can model it as three nodes connected by three edges. Each node in this arrangement has a degree of two. In this Prisoner's Dilemma on a Graph demonstration, the penalties depending on player actions are:
Player Defects {
1) All opponents defect -> 8 years (equilibrium)
2) One opponent defects, one cooperates -> 4
3) All opponents cooperate -> 0
Player Cooperates {
4) All opponents defect -> 12
5) One opponend defects, one cooperates -> 6
6) All opponents cooperate -> 1

First, a graph is created and each node is randomly given a 'strategy' parameter with a value of 1 for defect and 2 for cooperate.

```r
set.seed(945)
nodes <- 3
degree <- 2
g.1 <- degree.sequence.game(rep(degree, nodes), method =  "vl")

#randomply select strategy
# 1 = defect, 2 = cooperate
```

```r
V(g.1)$strategy <- sample(c(1,2), size = length(V(g.1)), replace = T)

V(g.1)[strategy == 1]$color <- 'red' # defect
V(g.1)[strategy == 2]$color <- 'blue' # cooperate
```

```r
plot(g.1)
```



In order to generalize the evaluation function to other graphs, there is a function to calculate the maximum degree in the graph. In the future, this should be re-factored to store the value for the graph instead of calculating it each time the network evaluation function runs.

```r
max.degree <- function(g){

  degree.list <- NULL

  for(i in V(g)){
    degree.list <- c(degree.list, degree(g = g, v = i))
  }

  return(max(degree.list))
}
```

Network evaluation function: returns a dataframe with each row representing the strategy of the opposing players for a given node.

```r
network.evaluation <- function(g, nodes){
  network.results <- NULL

  max.degree <- max.degree(g)

  for(i in 1:nodes){

    opponents <- neighbors(g, i, mode="all")
    opponent.results <- array(NA, max.degree)

    for(j in 1:length(opponents)){
      opponent.results[j] <- opponents[j]$strategy
    }
    network.results <- rbind(network.results, opponent.results)
  }
  return(network.results)
}
```

Game evaluation: here the resulting ‘sentence’ in years (game outcome) is calculated for each node using the player’s strategy and the strategies of connected nodes.

```
game.evaluation <- function(results, nodes, g){
  row.names(results) <- 1:nodes
  results <- cbind(results, V(g)$strategy)

  sentence.list <- NULL

  for(i in 1:length(results[,1])){
    if(results[i,1] == 1){ #row defects
        opponent.sum <- sum(results[i,-1])
        if(opponent.sum == 2){sentence.list = c(sentence.list, 0)}
        if(opponent.sum == 3){sentence.list = c(sentence.list, 6)}
        if(opponent.sum == 4){sentence.list = c(sentence.list, 12)}
    }
    else{
        opponent.sum <- sum(results[i,-1])
        if(opponent.sum == 2){sentence.list = c(sentence.list, 1)}
        if(opponent.sum == 3){sentence.list = c(sentence.list, 4)}
        if(opponent.sum == 4){sentence.list = c(sentence.list, 8)}
    }
  }
  return(sentence.list)
}
```

```
r <- network.evaluation(g.1, nodes)
r.list <- game.evaluation(r, nodes, g.1)
```

Results graph: the results are moved from list form back onto the graph.

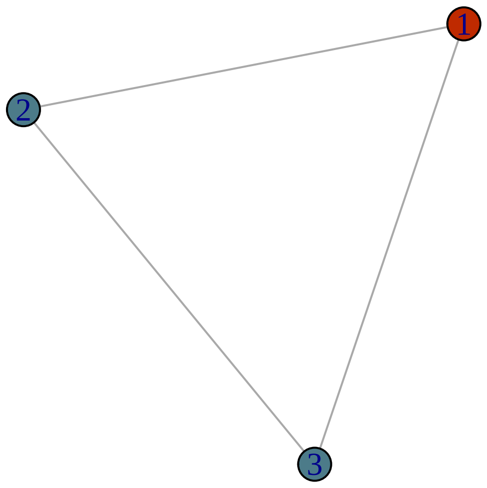
```
results.graph <- function(results.list, nodes, g){
  for(i in 1:nodes){
    V(g)[i]$result <- results.list[i]
  }
  return(g)
}
```

```
g.results <- results.graph(r.list, nodes, g.1)

results <- cbind(1:nodes, V(g.results)$result)
colnames(results) <-c("Node", "Years Sentenced")
kable(results)
```

| Node | Years Sentenced |
|---:|---:|
| 1 | 12 |
| 2 | 4 |
| 3 | 4 |

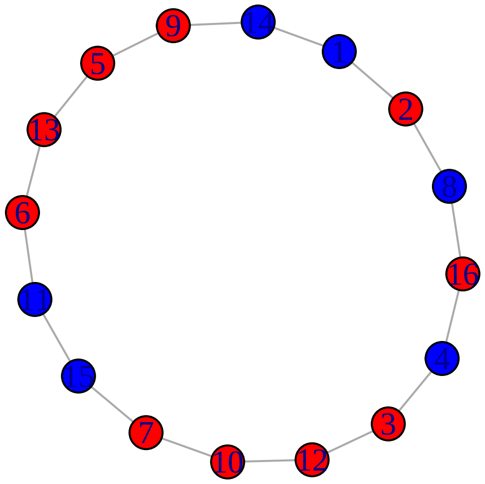
Color coding each of the possible outcomes:

```r
V(g.results)[result == 0]$color <- '#00B2E5'
V(g.results)[result == 1]$color <- '#2696B7'
V(g.results)[result == 4]$color <- '#4C7B89'
V(g.results)[result == 6]$color <- '#72605B'
V(g.results)[result == 8]$color <- '#98452D'
V(g.results)[result == 12]$color <- '#BF2A00'

plot(g.results)
```



This ‘Prisoner’s Dilemma on a Graph’ simulation may be extended to n players all of degree 2, creating a ring graph.

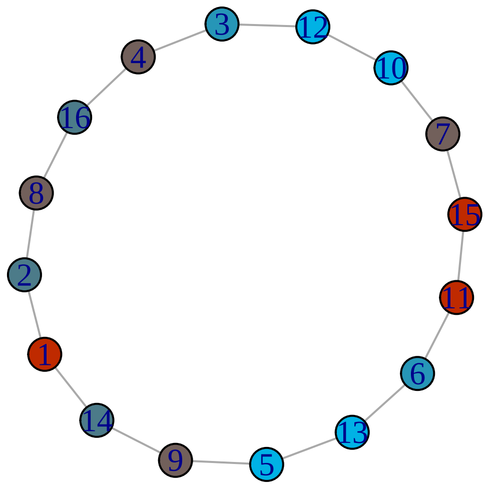
```r
nodes <- 16
degree <- 2
g.2 <- degree.sequence.game(rep(degree, nodes), method =  "vl")

#randomply select strategy
# 1 = defect, 2 = cooperate
V(g.2)$strategy <- sample(c(1,2), size = length(V(g.2)), replace = T)

V(g.2)[strategy == 1]$color <- 'red' # defect
V(g.2)[strategy == 2]$color <- 'blue' # cooperate
plot(g.2)
```



```r
r.2 <- network.evaluation(g.2, nodes)
r.2.list <- game.evaluation(r.2, nodes, g.2)

g.2.results <- results.graph(r.2.list, nodes, g.2)

results.2 <- cbind(1:nodes, V(g.2.results)$result)
colnames(results.2) <-c("Node", "Years Sentenced")
kable(results.2)
```

| Node | Years Sentenced |
|---:|---:|
| 1 | 12 |
| 2 | 4 |
| 3 | 1 |
| 4 | 6 |
| 5 | 0 |
| 6 | 1 |
| 7 | 6 |
| 8 | 6 |
| 9 | 6 |
| 10 | 0 |
| 11 | 12 |
| 12 | 0 |
| 13 | 0 |
| 14 | 4 |
| 15 | 12 |
| 16 | 4 |

```r
V(g.2.results)[result == 0]$color <- '#00B2E5'
V(g.2.results)[result == 1]$color <- '#2696B7'
V(g.2.results)[result == 4]$color <- '#4C7B89'
V(g.2.results)[result == 6]$color <- '#72605B'
V(g.2.results)[result == 8]$color <- '#98452D'
V(g.2.results)[result == 12]$color <- '#BF2A00'

plot(g.2.results)
```

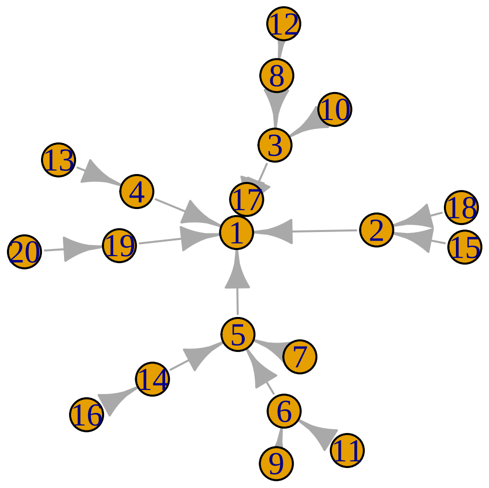


## Game Theory on Complex Networks

Most human networks are not well modeled by a ring where every person has two connections. Instead, they may resemble a Barabási–Albert model - a random, scale-free graph that commonly arises in human networks. Here, I use the example of a crime syndicate to explore how games can be played on a graph.

The model here is applying game theory to a network diffusion model that follows these steps:
1) All 1st degree nodes ("Footsoldiers") are given some probability of defection. 2) Each greater than 1 degree node ("Wiseguys") is given a probability that depends on whether their Footsoldiers defected or not. This step is created until all except the central node ("Godfather") is either defected upon or the Wiseguys connected to him stay loyal.

```
size = 20
set.seed(2913)
g <- barabasi.game(size, power = 0.6, m = 1)

plot(g)
```



one.degree: finds the 'Footsoldiers' - nodes with degree of one. If there are no one-degree nodes, 25% of the network is randomly assigned as a starting point.

```r
one.degree <- function(g){
  degree.list <- NULL

  for(i in V(g)){
    if(degree(g = g, v = i) == 1){
      degree.list <- c(degree.list, i)
    }
  }

  if(length(degree.list) == 0){
    num.samples <- round(length(V(g))*0.25, 0)
    degree.list <-sample(V(g), size = num.samples)}

  return(degree.list)
}
```

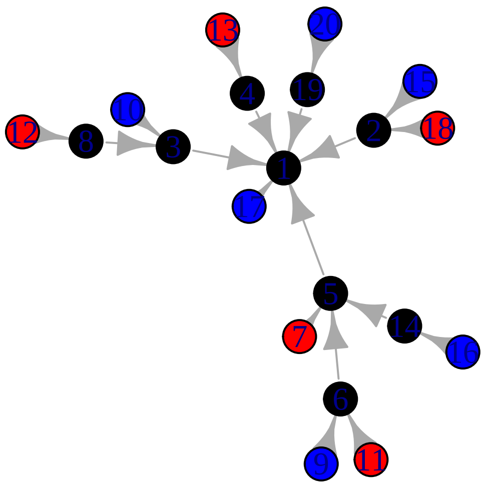
```r
footsoldiers <- one.degree(g)
```

decide.defection: randomly give 40% of Footsoldiers the defect strategy.

```r
decide.defection <- function(g, size, footsoldiers){
  for(i in 1:size){
    if(V(g)[i] %in% footsoldiers){
      V(g)[i]$strategy <- sample(c(1,2), size = 1, prob = c(0.4, 0.6))
    }
    else{V(g)[i]$strategy <- NA}
  }

  for(i in 1:size){
    if(is.na(V(g)$strategy[i]) == TRUE){
      V(g)[i]$color <- 'black'
    }
    else{
      if(V(g)$strategy[i] == 1){
        V(g)[i]$color <- 'red'
      }
      if(V(g)$strategy[i] == 2){
        V(g)[i]$color <- 'blue'
      }
    }
  }
  return(g)
}
```

At this point, only Footsoldiers have played the game.

```r
g <- decide.defection(g, size, footsoldiers)
plot(g)
```



get.next.list: obtains a list of nodes which do not have a strategy yet.

```r
get.next.list <- function(g, size){
  next.list <- NULL
  for(i in 1:size){
    opponents <- neighbors(g, i, mode="all")
    if(length(opponents$strategy[is.na(opponents$strategy) == TRUE]) ==
       length(opponents$strategy)){next}
    else{
      #get index for the next round of evaluation
      next.list <- c(next.list, i)
    }
  }
  return(next.list)
}
```

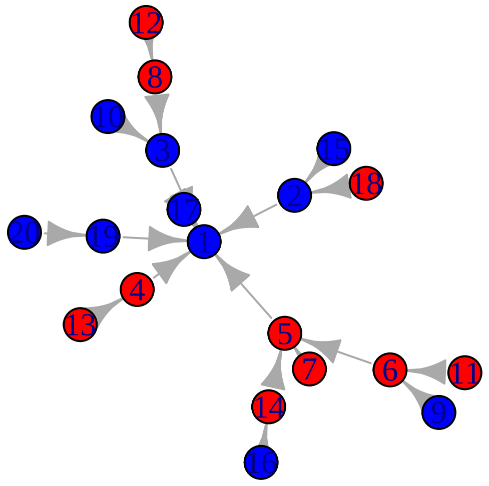
```r
next.list <- get.next.list(g, size)
```

Now, the players who are connected to a Footsoldier (the "Wiseguys") are given a defect or cooperate strategy. If they are connected to someone who has defected, they will defect 50% of the time. If they are connected to someone who stayed loyal, they will defect only 10% of the time.

```r
decide.remaining.players <- function(g, next.list){
  for(i in next.list){
    opponents <- neighbors(g, i, mode="all")

    results <- plyr::count(opponents$strategy)

    defectors <- results$freq[results$x == 1][1]
    loyals <- results$freq[results$x == 2][1]

    if(is.na(defectors) == TRUE){defectors = 0}
    if(is.na(loyals) == TRUE){loyals = 0}


    if(defectors >= loyals){
      V(g)[i]$strategy <- sample(c(1,2), size = 1, prob = c(0.5, 0.5))
      V(g)[i]$color <- "red"
    }
    else{
      V(g)[i]$strategy <- sample(c(1,2), size = 1, prob = c(0.9, 0.1))
      V(g)[i]$color <- "blue"
      }
  }
  return(g)
}
```

On this relatively small graph it only takes one round to decide the strategy of all players. The central node (#1) is the Godfather. In this case, two of three of the Godfather's connections defected. The Godfather stayed loyal.

```
g <- decide.remaining.players(g, next.list)
plot(g)
```



```
length(V(g)[strategy == 1]) / length(V(g))
```

```
## [1] 0.55
```

55% of the players defected.

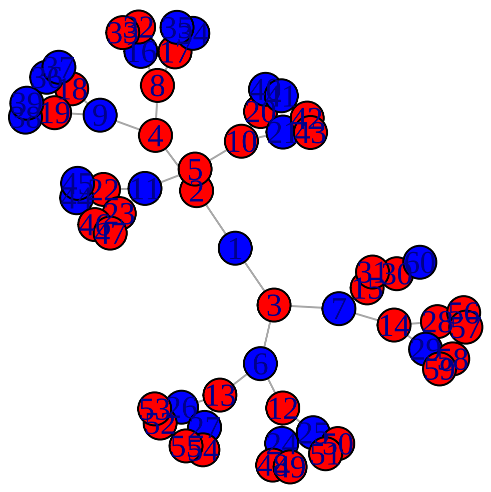
### Tree Graph

In this scenario, I use the above game on a larger network. It is necissary to play the game multiple times in order for each level to play.

```
size = 60
tree <- make_tree(n = size, children = 2,  mode = "undirected")

footsoldiers <- one.degree(tree)
tree <- decide.defection(tree, size, footsoldiers)

next.list <- get.next.list(tree, size)
tree <- decide.remaining.players(tree, next.list)
```

```r
while(is.na(V(tree)$strategy[1]) == TRUE){
  next.list <- get.next.list(tree, size)
  tree <- decide.remaining.players(tree, next.list)
}
```

```r
plot(tree)
```



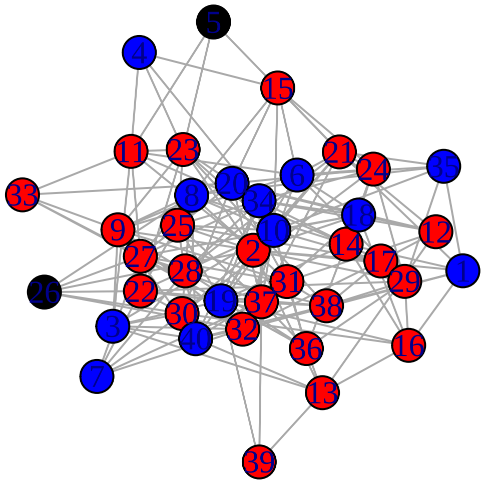
In this tree network, 65% of the players defected.

### Random Graph

For a random graph:

```r
size = 40
random <- erdos.renyi.game(n = size, 0.25)

footsoldiers <- one.degree(random)
random <- decide.defection(random, size, footsoldiers)

while(is.na(V(random)$strategy[1]) == TRUE){
  next.list <- get.next.list(random, size)
  random <- decide.remaining.players(random, next.list)
}
```

```r
plot(random)
```

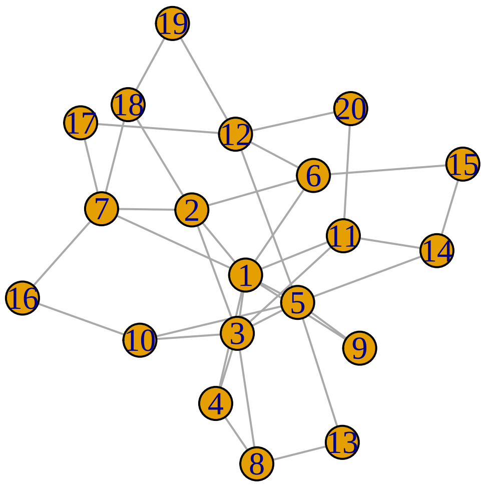


In this highly connected graph, 65% of the players defected.

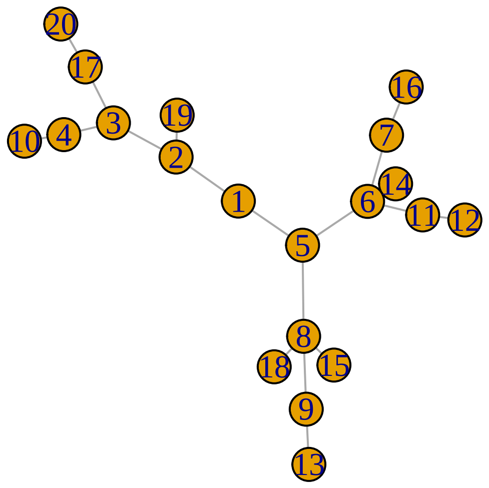
## Does connectivity change game results?

Here, we will run a game on two different graphs multiple times. The percentage of defectors will be calculated and returned.

```r
size = 20
more.connected <- barabasi.game(size, power = 0.5, m = 2, directed = FALSE)
plot(more.connected)
```



```r
size = 20
less.connected <- barabasi.game(size, power = 0.15, m = 1, directed = FALSE)
plot(less.connected)
```



```
iterate.game <- function(graph){
  footsoldiers <- one.degree(graph)
  graph <- decide.defection(graph, size, footsoldiers)

  while(is.na(V(graph)$strategy[1]) == TRUE){
    next.list <- get.next.list(graph, size)
    graph <- decide.remaining.players(graph, next.list)
  }

  defect.proportion.iter <- length(which(V(graph)$strategy == 1)) /
    length(V(graph))

  return(defect.proportion.iter)
}
```

```
iters <- 500
```

```
more.connected.percent.defectors <- replicate(iters, iterate.game(more.connected))
summary(more.connected.percent.defectors)
```

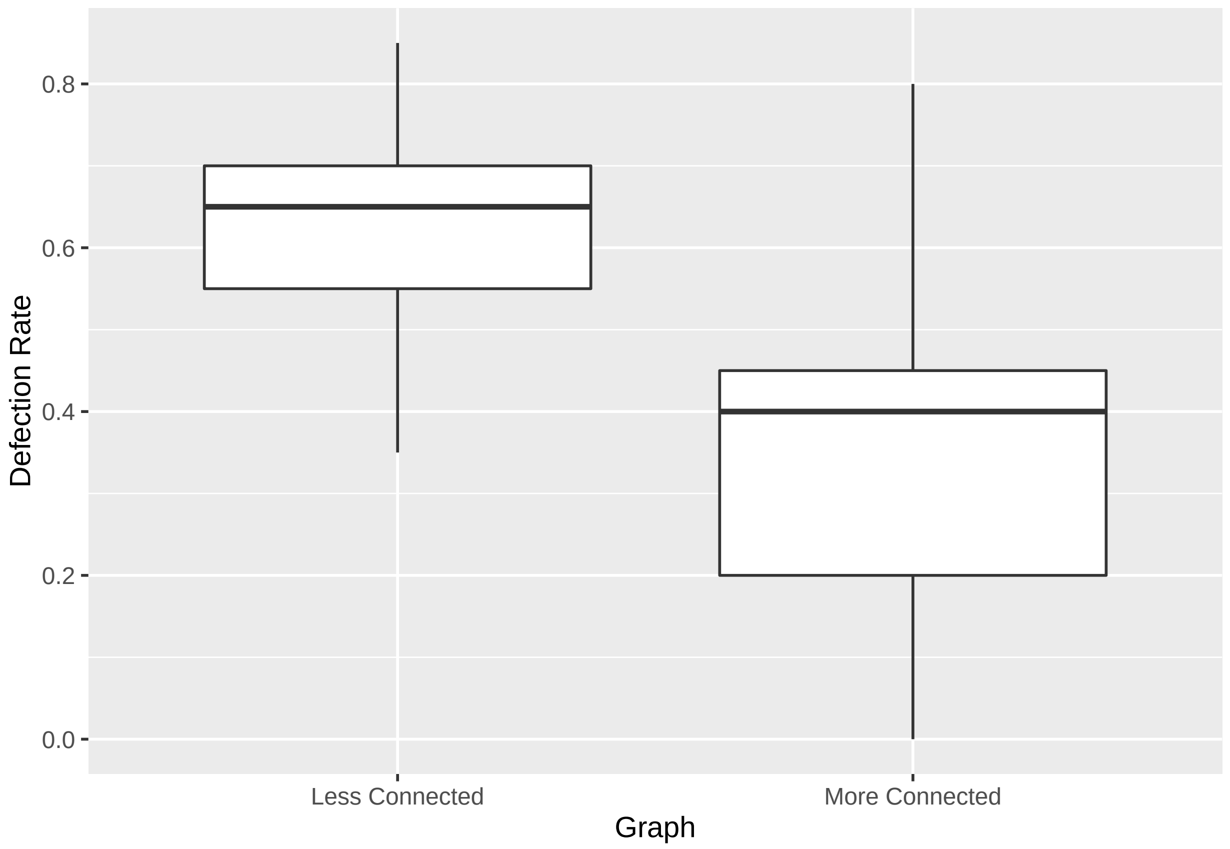
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.2000  0.4000  0.3516  0.4500  0.8000
```

```
less.connected.percent.defectors <- replicate(iters, iterate.game(less.connected))
summary(less.connected.percent.defectors)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.3500  0.5500  0.6500  0.6299  0.7000  0.8500
```

```
iteration.results <- cbind(more.connected.percent.defectors, less.connected.percent.defectors)
iteration.results <- as.data.frame(iteration.results)
colnames(iteration.results) <- c("More Connected", "Less Connected")

iteration.results <- gather(iteration.results, "Graph", "Defections", 1:2)
```

```r
ggplot(iteration.results, aes(Graph, Defections)) +
  geom_boxplot(stat = "boxplot") +
  ylab("Defection Rate")
```



The more connected graph shows a lower average and higher variation in defection rate.