Estimating In-Group Bias With Fuzzy Identities

Evidence from Regional Bias in Mexico

R-Scripts

Mauricio Fernández-Duque*

March 13, 2024

1 R-Scripts for Generating Network Regionalizations

1.1 Master Script

```
# This script provides the instructions for the commands and code
    that should be run. The following ten commands should be run
    sequentially.

# You should have an adjacency matrix called "Adjacency Matrix.
    xlsx".
```

*CIDE, Carretera México-Toluca 3655, Col. Lomas de Santa Fe, Del. Cuajimalpa, 01210, Mexico City, Mexico
Email: mauricio.fernandez@cide.edu.

```
#First, set the working directory to where you have the adjacency
    matrix.

setwd("D:/Documents/Regional Bias")
library(readxl)



# Second, define the maximum number of states in a region, and
   call the first R-script.

max_states <-7
source("SetOfRegionStrategies.R")

# Third, define the list "strategies", which captures the output
   of the first R-script. As the name suggests, "strategies"
   lists all possible strategies for each node, and provides some
    additional information.

strategies <-SetOfRegionStrategies("Adjacency_Matrix.xlsx",
   max_states)
```

```
# Fourth, from "strategies" generate the variables and lists you
    need in the data frame.

NumberOfStatesInCountry<-strategies[[1]]

for (i in 1:NumberOfStatesInCountry){
  placeholder<-strategies[1+NumberOfStatesInCountry+2+
      NumberOfStatesInCountry+i]
  assign(paste0("list_of_strategies",i),setNames(placeholder,
      seq_along(placeholder)))[[1]]
  rm(placeholder)
}


for (i in 1:NumberOfStatesInCountry){
  placeholder<-strategies[1+NumberOfStatesInCountry+2+
      NumberOfStatesInCountry+NumberOfStatesInCountry+2+i]
  assign(paste0("length_of_strategies",i),setNames(placeholder,
      seq_along(placeholder)))[[1]]
  rm(placeholder)
}
```

```
for (i in 1:NumberOfStatesInCountry){
  placeholder<-strategies[1+NumberOfStatesInCountry+2+
     NumberOfStatesInCountry+NumberOfStatesInCountry+1]
  strategies_per_state<-setNames(placeholder,seq_along(
     placeholder))[[1]]
  rm(placeholder)
}

# This step also defines the multiplicands of the nodes' utility
   funcions. alpha refers to the weight on the matching utility,
   beta to the weight on the poularity utility, and gamma to the
   cost.

alpha<-1
beta<-1
gamma<-0.3

# repetitions is a parameter that indicates how many equilibriums
    the program must find before stopping. In the computers I was
    working with, the computers would slow down after 20
   equilibriums, so I needed to shut R down and rerun.
```

```
repetitions <-20


# Sixth, create a "repetitions" amount of strategy profiles.

strategyprofile <-list()
index_strategies <-list()
for (j in 1:repetitions){
  print(paste("Creating sample strategy profiles",j))
  index_strategiesj <-c()
  strategy_profilej <-list()
  for (x in 1:NumberOfStatesInCountry) {
    index_strategiesj[x]<-sample(1:strategies_per_state[x],1)
    strategy_profilej[[x]]<-get(paste0("list_of_strategies",x))
       [[1]][[index_strategiesj[x]]]
  }
  strategyprofile[[j]]<-strategy_profilej
  index_strategies[[j]]<-index_strategiesj
  rm(strategy_profilej,index_strategiesj)
}
```

```
# Seventh, parameter n is a starting point for a for loop defined
    for EqSearchStep.R, the R-script that is run in the next step
   .

n<-1

# Eighth, run the algorithm for finding the equilibrium. The
   algorithm is going to proceed from the  strategy profile
   number "n" to the strategy profiles number "repetitions". For
   each strateg profile, it will calculate utilities from
   following and from deviating from the strategy profile for
   each player, and substituing in  the best profitable deviation
    until it finds a Nash equilibrium.

source("EqSearchStep.R")
```

1.2 Listing the Strategies per State

```
#This program lists each possible strategy that each node has.

#Once this program has run, you will end up with a long list
   called "strategies". strategies[[1]] is the number of states
```

in the country. Say that number is N. strategies[[1+i]] is equal to the possible strategies for i between 1 and the number of states in the country. strategies[[1+N+1]] is a vector with the xth entry equal to the number of total strategies state x has. strategies[[1+N+2]] is equal to the total number of strategy profiles (a multiplication of the strategies of each state). Each of strategies[[i]] for i between 1+N+2+1 and 1+N+2+N is a vector equal to the length of the strategies for each strategy of state i. The rest of the values of strategies[[i]] (from 1+N+2+N+1 to 1+N+2+N+N+2+N) are analogous to the values we have just described, but for strategies restricted to be at most of length max_states.

```
SetOfRegionStrategies<-function(file_path,max_states){

  AdjacencyMatrix<- read_excel(file_path)

  # Now that we've uploaded the network that corresponds to the
     state, we need to run two embedded for loops to obtain all
     the strategies.
  # Need a counter s per state.
```

```
# For each s
#   Generate the set of neighbors of state s.
#   Add the state, and the state plus each of the combinations
   of neighbors, to  the list of strategies.
#   Generate a counter c equal to the number of combinations of
    neighbors of s.
#     For each c, generate the set of neighbors of each state
   in c that are  neither the state c nor the neighbors of c.
#     Add the state plus each c plus each of the combinations
   of neighbors of the state in c to the list of strategies.


NumberOfStatesInCountry<-length(AdjacencyMatrix)

for(s in 1:NumberOfStatesInCountry){
  # The for loop fixes state s. Will use row s in the adjacency
      matrix.

  # Create a logical vector indicating where the elements in
     the fixed row are equal to 1

  # Use the logical vector to subset the variable names. The
```

```
    new variable is the set of neighbors of the state.

# The neighbors' names are numbers encoded as strings, so
    have to turn into numbers.
print(paste(s,"First loop"))

assign(paste("Neighbors",s,sep=""),as.numeric(colnames(
    AdjacencyMatrix)[AdjacencyMatrix[s, ] == 1]))

# Create a list of combinations of the neighbors of the state
     in question.

# Here is how to obtain all possible combinations of zeros
    and ones in a  vector of size n. This can be used to get
    all possible combinations of  states that are d degrees
    away from the state in question.

assign(paste("Number_of_Neighbors",s,sep=""),length(get(paste
    ("Neighbors",s,sep="")))))

l<-rep(list(0:1),get(paste("Number_of_Neighbors",s,sep="")))
```

```
Sequence<-1:get(paste("Number_of_Neighbors",s,sep=""))

generate_vector<-function(x){
  return(expand.grid(1)[[x]]*get(paste("Neighbors",s,sep=""))
    [x])
}

vector_list <-lapply(Sequence,generate_vector)

assign(paste("CombinationsOfNeighbors",s,sep=""),t(as.data.
  frame(do.call(rbind,vector_list))))

# Now I'm going to create the first elements of the set of
  strategies.
list_of_strategies<-list()

# Add to the list of strategies the strategies what include
  all combinations of at most neighbors that are one state
  away

assign(paste("NumberOfCombinationsOfNeighbors",s,sep=""),2**
  get(paste("Number_of_Neighbors",s,sep="")))
```

```
for (x in 1:get(paste("NumberOfCombinationsOfNeighbors",s,sep
  =""))) {
  list_of_strategies[[x]]<-c(s,get(paste("
    CombinationsOfNeighbors",s,sep=""))[x,][get(paste("
    CombinationsOfNeighbors",s,sep=""))[x,] != 0])
}


# Now we need to create a list of the neighbors of each of
  the neighbors of the state in question
# Start at c=2 because c=1 is the combination of neighbors of
   neighbors that does not include any neighbors of
  neighbors. Therefore, it yields strategies that are
  already included in the list.

for(c in 2:get(paste("NumberOfCombinationsOfNeighbors",s,sep
  =""))){
  assign(paste("Combination",c,"OfNeighborsOf",s,sep=""),get(
    paste("CombinationsOfNeighbors",s,sep=""))[c,])
  assign(paste("Combination",c,"OfNeighborsOf",s,sep=""),get(
    paste("Combination",c,"OfNeighborsOf",s,sep=""))[get(
```

```
paste("Combination",c,"OfNeighborsOf",s,sep="")) !=0])

#I take the rows that correspond to the neighbors in
   question ...
assign(paste("subset_Network_",s,"_",c,sep=""),
   AdjacencyMatrix[get(paste("Combination",c,"OfNeighborsOf
   ",s,sep="")),])

# ... I create a row that sums the subset of rows, and I
   keep as TRUE those variables in the row that are
   positive (where at least one of the  neighbor states in
   question has it as a neighbor)



assign(paste("logical_vector_subset_network_",s,"_",c,sep
   =""),colSums(get(paste("subset_Network_",s,"_",c,sep=""))
   ))>0)



assign(paste("NeighborsOfTheNeighbors",c,"OfState",s,sep
   =""),as.numeric(colnames(AdjacencyMatrix)[get(paste("
   logical_vector_subset_network_",s,"_",c,sep=""))]))
```

```
# I need to remove from the list of neighbors of the
    neighbor of state s both the neighbors of state s and
    state s itself.

assign(paste("NeighborsOfTheNeighbors",c,"OfState",s,sep
    =""),setdiff(get(paste("NeighborsOfTheNeighbors",c,"
    OfState",s,sep="")),s))
assign(paste("NeighborsOfTheNeighbors",c,"OfState",s,sep
    =""),setdiff(get(paste("NeighborsOfTheNeighbors",c,"
    OfState",s,sep="")),get(paste("Combination",c,"
    OfNeighborsOf",s,sep=""))))


# Now I have to find all possible combinations of states
    within the filtered set of neighbors of the neighbor of
    state s.

assign(paste("NumberOfNeighborsOfTheNeighbors",c,"OfState",
    s,sep=""),length(get(paste("NeighborsOfTheNeighbors",c,"
    OfState",s,sep=""))))
```

```
l<-rep(list(0:1),get(paste("NumberOfNeighborsOfTheNeighbors
    ",c,"OfState",s,sep="")))

gridnotallzeros<-expand.grid(l)[-1,]

# This next variable is useful for the upcoming if operator
Number_of_strategies_so_far<-length(list_of_strategies)

# Need an if loop because there are three separate cases.
    If a neighbor n has no neighbors, then we already
    included the corresponding strategy of the state s plus
    the neighbor n. If a neighbor n has one neighbor nn, we
    only need to add the strategy of s plus n plus nn.

if (get(paste("NumberOfNeighborsOfTheNeighbors",c,"OfState
    ",s,sep=""))>1){

  Sequencec <- 1:get(paste("NumberOfNeighborsOfTheNeighbors
      ",c,"OfState",s,sep=""))

  generate_vectorc<-function(x){
    return(gridnotallzeros[[x]]*get(paste("
```

```
    NeighborsOfTheNeighbors",c,"OfState",s,sep=""))[x])
}

vector_listc <-lapply(Sequencec,generate_vectorc)

assign(paste("CombinationsOfNeighborsOfNeighbors",c,"
   OfState",s,sep=""),t(as.data.frame(do.call(rbind,
   vector_listc))))

assign(paste("NumberOfCombinationsOfNeighborsOfNeighbors
   ",c,"OfState",s,sep=""),nrow(get(paste("
   CombinationsOfNeighborsOfNeighbors",c,"OfState",s,sep
   =""))))

# Now I'm going to continue to compile the set of
   strategies into a vector

for (x in 1:get(paste("
   NumberOfCombinationsOfNeighborsOfNeighbors",c,"OfState
   ",s,sep=""))) {
  list_of_strategies[[x+Number_of_strategies_so_far]]<-c(
    s,get(paste("Combination",c,"OfNeighborsOf",s,sep
```

```
          =""),get(paste("CombinationsOfNeighborsOfNeighbors
          ",c,"OfState",s,sep=""))[x,][get(paste("
          CombinationsOfNeighborsOfNeighbors",c,"OfState",s,
          sep=""))[x,] != 0])
    }

  } else {
    if (get(paste("NumberOfNeighborsOfTheNeighbors",c,"
        OfState",s,sep=""))==1){
      list_of_strategies[[1+Number_of_strategies_so_far]]<-c(
          s,unlist(unname(get(paste("Combination",c,"
          OfNeighborsOf",s,sep="")))),unlist(unname(get(paste
          ("NeighborsOfTheNeighbors",c,"OfState",s,sep=""))))))
    }
  }
}

# The following is cleanup. Don't want to keep all the
   variables created in the for loop.
# But don't want to delete the adjacency matrix just yet!

assign(paste("list_of_strategies",s,sep=""),
```

```
    list_of_strategies)

  all_objects <- ls()

  list_of_list_of_strategies <- c(all_objects[grep(paste0("^",
    list_of_strategies"), all_objects)],"AdjacencyMatrix","
    max_states")

  objects_to_remove<-setdiff(all_objects,
    list_of_list_of_strategies)

  rm(list=objects_to_remove,all_objects,
    list_of_list_of_strategies)

  }
NumberOfStatesInCountry<-length(AdjacencyMatrix)



# Now we construct a vector that specifies how many strategies
   each state has.

strategies_per_state<-numeric(length=length(ls(pattern="
```

```
    list_of_strategies[0-9]+")))

for(i in 1:length(strategies_per_state)){
   strategies_per_state[i]<-length(get(paste0("
      list_of_strategies",i)))
}

# We would like to compute the total number of strategy
    profiles.
number_of_strategy_profiles<-strategies_per_state[1]

for(i in 2:length(strategies_per_state)){
   number_of_strategy_profiles<-number_of_strategy_profiles*
      strategies_per_state[i]
}




# Finally, we would like to know the length of each strategy of
     each state.
```

```
for(i in 1:NumberOfStatesInCountry){
  length<-list()
  for(x in 1:strategies_per_state[i]){
    print(i)
    print(x)
    length[[x]]<-length(get(paste0("list_of_strategies",i))[[x
       ]])
  }
  assign(paste0("length_of_strategies_state",i),length)
  rm(length)
}


# Now we will cull the strategies so that they  contain at most
   max_states states, and then provide the same statistics as for
    the full sets of strategies


for(i in 1:NumberOfStatesInCountry){
  placeholder<-get(paste0("list_of_strategies",i))[get(paste0("
     length_of_strategies_state",i))<=max_states]
```

```
  assign(paste0("list_of_strategies_MaxStates",i),placeholder)
  rm(placeholder)
}
strategies_per_state_MaxStates<-numeric(length=length(ls(
   pattern="list_of_strategies[0-9]+")))

for(i in 1:length(strategies_per_state_MaxStates)){
  strategies_per_state_MaxStates[i]<-length(get(paste0("
     list_of_strategies_MaxStates",i)))
}

number_of_strategy_profiles_MaxStates<-
   strategies_per_state_MaxStates[1]

for(i in 2:length(strategies_per_state_MaxStates)){
  number_of_strategy_profiles_MaxStates<-
     number_of_strategy_profiles_MaxStates*
     strategies_per_state_MaxStates[i]
}

for(i in 1:NumberOfStatesInCountry){
  length<-list()
```

```
  for(x in 1:strategies_per_state_MaxStates[i]){
    print(i)
    print(x)
    length[[x]]<-length(get(paste0("
        list_of_strategies_MaxStates",i))[[x]])
  }
  assign(paste0("length_of_strategies_state_MaxStates",i),
     length)
  rm(length)
}




# Now we build the list of variables to return.

variables_to_return<-list()

variables_to_return[[1]]<-NumberOfStatesInCountry



for(i in 1:NumberOfStatesInCountry){
  variables_to_return[[1+i]]<-get(paste0("list_of_strategies",i
```

```
    ))
}

variables_to_return[[1+NumberOfStatesInCountry+1]]<-
  strategies_per_state
variables_to_return[[1+NumberOfStatesInCountry+2]]<-
  number_of_strategy_profiles

for(i in 1:NumberOfStatesInCountry){
variables_to_return[[1+NumberOfStatesInCountry+2+i]]<-get(
  paste0("length_of_strategies_state",i))
 }



for(i in 1:NumberOfStatesInCountry){
  variables_to_return[[1+NumberOfStatesInCountry+2+
    NumberOfStatesInCountry+i]]<-get(paste0("
    list_of_strategies_MaxStates",i))
}
```

```
  variables_to_return[[1+NumberOfStatesInCountry+2+
    NumberOfStatesInCountry+NumberOfStatesInCountry+1]]<-
    strategies_per_state_MaxStates
  variables_to_return[[1+NumberOfStatesInCountry+2+
    NumberOfStatesInCountry+NumberOfStatesInCountry+2]]<-
    number_of_strategy_profiles_MaxStates

  for(i in 1:NumberOfStatesInCountry){
    variables_to_return[[1+NumberOfStatesInCountry+2+
      NumberOfStatesInCountry+NumberOfStatesInCountry+2+i]]<-get
      (paste0("length_of_strategies_state_MaxStates",i))

  }


return(variables_to_return)



}
```

1.3 Finding the Equilibria

```
# This R-script takes as a starting point a strategy profile j (
   starting with j=n and ending with j=repetitions), and
   calculates the utilities from following and deviating from the
    strategy profile for each node. It then stops if no one wants
    to deviate, and registers  the result as an equilibrium. If
   someone wants to deviate, it substitutes the best deviation
   into the strategy profile, and starts again until it finds an
   equilibrium.

for(j in n:repetitions){

  EquilibriumCondition<-0
  counter<-1

  strategyprofilej<-strategyprofile[[j]]
  index_strategiesj<-index_strategies[[j]]

  # The program has defined its strategy profile starting point.
     The while loop below will continue to run until an
     equilibrium is found.
```

```
while (EquilibriumCondition==0){
  print(paste("This is iteration",counter,"of strategy profile
     ",j))
  counter<-counter+1

  length<-list()
  for(i in 1:NumberOfStatesInCountry){
    strategyprofileji<-unlist(strategyprofilej[i])
    length[[i]]<-length(strategyprofileji)
    rm(strategyprofileji)
  }
  assign(paste0("lengths_strategyprofile",j),length)
  rm(length)

  # Calculating the utility for each possible strategy of each
     possible state given the strategy profile (that is,
     keeping others' strategies fixed).

  # For the strategy profile j



  utilities_strategyprofilej<-list()
```

```
# For each state i
for (i in 1:NumberOfStatesInCountry){
  print(paste(i))

  utilities_strategyprofilej_statei <-list()

  # For each strategy of state i
  for (k in 1:strategies_per_state[i]){
    utilities_strategyprofilejstateistrategyk <-list()

    assign(paste0("matching_utility_profile",j,"state_",i,"
       strategy_",k),0)

    assign(paste0("popularity_utility_profile",j,"state_",i,"
       strategy_",k),0)

    # Consider each state x (other than i) in the kth
       strategy of i

    if (get(paste0("length_of_strategies",i))[[1]][[k]]>1){
```

```
for (x in 2:get(paste0("length_of_strategies",i))
    [[1]][[k]]) {

  xthStateInStrategyOfi <- get(paste0("
      list_of_strategies",i))[[1]][[k]][x]

  # Consider each state y in x's strategy
  if(get(paste0("lengths_strategyprofile",j))[[
      xthStateInStrategyOfi]]>1){

    for (y in 2:get(paste0("lengths_strategyprofile",j)
        )[[xthStateInStrategyOfi]]){

      # Give i a point if x includes i in the strategy
          is following according to strategy profile j

      if (strategyprofilej[[xthStateInStrategyOfi]][y
          ]==i){

        assign(paste0("matching_utility_profile",j,"
            state_",i,"strategy_",k),get(paste0("
            matching_utility_profile",j,"state_",i,"
```

```
        strategy_",k))+1)

      }
    }
  }
}

# Also give i a point for every state z (other than i)
   ...



for(z in 1:NumberOfStatesInCountry){

  if (z!=i){
    # ... that has state x as part of its strategy. To
       do this, have to consider each state b that is
       part of z's strategy in profile j.

    for (b in 1:get(paste0("lengths_strategyprofile",j)
       )[[z]]){
      bthStateInStrategyOfz<-strategyprofilej[[z]][b]
```

```
                if(bthStateInStrategyOfz==xthStateInStrategyOfi){

                  assign(paste0("popularity_utility_profile",j,"
                     state_",i,"strategy_",k),get(paste0("
                     popularity_utility_profile",j,"state_",i,"
                     strategy_",k))+1)
                }

              }
            }
          }
        }

# In the paper I consider several parametrizations of the
   popularity utility. The code below shows how to program the
   bound on the popularity utility I used for the second
   parametrization.

        if (get(paste0("popularity_utility_profile",j,"state_",i
           ,"strategy_",k))<3) {
          bounded_popularity<-get(paste0("
             popularity_utility_profile",j,"state_",i,"strategy_
```

```
    ",k))
  }
  else {
  bounded_popularity <-2
  }

# This is where the utility function is calculated. The
   parameters alpha and beta are set in the Master script
   , although the piece of code below could be changed to
    modify the utility function more significantly.
assign(paste0("utility",i,"strategyprofile",j,"strategy_
   ",k),alpha*get(paste0("matching_utility_profile",j,"
   state_",i,"strategy_",k))+beta*bounded_popularity -
   gamma*(1/2)*(get(paste0("length_of_strategies",i))
   [[1]][[k]]-1)*(get(paste0("length_of_strategies",i))
   [[1]][[k]]+1-1))
utilities_strategyprofilej_statei[k]<-get(paste0("utility
   ",i,"strategyprofile",j,"strategy_",k))

objects_to_remove <-c(paste0("matching_utility_profile",j
   ,"state_",i,"strategy_",k),paste0("
   popularity_utility_profile",j,"state_",i,"strategy_",k
```

```
        ),paste0("utility",i,"strategyprofile",j,"strategy_",k
        ))

      rm(list=objects_to_remove)
      rm(objects_to_remove)
    }
    utilities_strategyprofilej[[i]]<-
      utilities_strategyprofilej_statei

    objects_to_remove<-c("xthStateInStrategyOfi","
      utilities_strategyprofilej_statei","
      bthStateInStrategyOfz","x","i","z","y","b","k")

    rm(list=objects_to_remove)
    rm(objects_to_remove)

}




# For the strategy profile j, calculate the utility of each
   actor when everybody is following the strategies
```

```
   prescribed by the strategy profile

utility_from_strategy_profile_j <-c()
for (i in 1:NumberOfStatesInCountry){
  index_strategies_profilej_statei <-index_strategiesj[i]
  utility_from_strategy_profile_j[i]<-
     utilities_strategyprofilej[[i]][[
     index_strategies_profilej_statei]]
}
rm(list="index_strategies_profilej_statei")

# For the strategy profile j, calculates the sum of payoffs

TotalPayoff<-sum(utility_from_strategy_profile_j)
print(paste("Total payoff in round",counter,"strategy profile
   ",j,"is",TotalPayoff))

# For each strategy profile j, calculates the difference
   between the utility a state gets from following the
   strategy prescription (along with everybody else), and the
    utility a state gets from each possible strategy.
```

```
Differencej<-list()
for (i in 1:NumberOfStatesInCountry){
  Differenceji<-c()
  for (k in 1:strategies_per_state[i]) {
    Differenceji[k]<-utility_from_strategy_profile_j[i]-
       utilities_strategyprofilej[[i]][[k]]
  }
  Differencej[[i]]<-Differenceji
}
rm(list="Differenceji")



# Calculates the best possible deviation of each player,
   providing
# a) the difference in  utilities between everybody following
    the candidate strategy profile and everybody but the
   state in question following the candidate strategy profile
# b) An indicator value for each state equal to one if they
   have a profitable deviation
# c) the label of the strategy that provides the best
   deviation.
```

```
IsThereAProfitableDeviationj <-c()
BestDeviationPayoffj <-c()
BestDeviationStrategyj <-c()
for(i in 1:NumberOfStatesInCountry){
  BestDeviationPayoffj[i]<-min(Differencej[[i]])
  BestDeviationStrategyj[i]<-which.min(Differencej[[i]])
  IsThereAProfitableDeviationj[i]<-ifelse(min(Differencej[[i
     ]]) <0,1,0)
}

print(paste("The states with a profitable deviation (those
   with a 1), are the following:",
   IsThereAProfitableDeviationj))
print(paste("The  vector of best deviation payoffs is:",
   BestDeviationPayoffj))
print(paste("The vector of best deviation strategies is:",
   BestDeviationStrategyj))


# Provides the number of actors who would deviate from
   strategy profile j,  as well as the sum of the differences
    of the deviators between the utility  they get from
```

```
   following the strategy prescription and the utility they
   would get from their best deviation.

NumberOfActorsWhoWouldDeviatej<-sum(
   IsThereAProfitableDeviationj)
# Although I want to sum only over the individuals who would
   like to deviate, the vector BestDeviationPayoff is at most
    zero (for those who cannot do better from deviating).
SumOfDeviationPotentialj<-sum(BestDeviationPayoffj)

print(paste("The number of actors who would deviate is given
   by:",NumberOfActorsWhoWouldDeviatej))
print(paste("The sum of the deviation potential is given by
   :",SumOfDeviationPotentialj))
ScrambledSubsetOfActorsWhoWouldDeviate<-which(
   BestDeviationPayoffj<0)
if (length(ScrambledSubsetOfActorsWhoWouldDeviate)>1){
  ScrambledSubsetOfActorsWhoWouldDeviate<-sample(
     ScrambledSubsetOfActorsWhoWouldDeviate)
}
FirstElementofScrambledSubset<-
   ScrambledSubsetOfActorsWhoWouldDeviate[1]
```

```
if (NumberOfActorsWhoWouldDeviatej==0){
  # Here have to record the relevant variables, because if we
       are in this condition it means that an equilibrium has
     been found.

  EquilibriumCondition<-1
  save(strategyprofilej,utility_from_strategy_profile_j,
     TotalPayoff,file=paste0("equilibriu1_1_30_Bounded_1_",j
     ,".RData"))
  print("An equilibrium has been found! Habemus equilibrium")
}
else{
  BestDeviationStrategyjFirstElementofScrambledSubset<-
     BestDeviationStrategyj[FirstElementofScrambledSubset]
  strategyprofilej[[FirstElementofScrambledSubset]]<-get(
     paste0("list_of_strategies",
     FirstElementofScrambledSubset))[[1]][[
     BestDeviationStrategyjFirstElementofScrambledSubset]]
  index_strategiesj[FirstElementofScrambledSubset]<-
     BestDeviationStrategyjFirstElementofScrambledSubset
```

```
            print("Black smoke")
        }



    }


}
```