Bachelor Thesis
Econometrics and operations research

The Influence of Non-Strategic Customers in Service Networks

Zoutendijk, R.W. 458391

Supervisor: Oosterom, C.D. van

Second assessor: Vester, J.S.

July 7, 2019

Abstract

I study the behaviour of strategic customers in the presence of non-strategic customers in an open routing service network. This network consists of two stations, which customers have to visit both. When a customer arrives, he chooses his route through the system. Strategic customers want to minimise their system time, non-strategic customers choose their route randomly. I compare two systems in a simulation study, one with strategic customers only and one with a combination of strategic and non-strategic customers. I find that strategic customers herd, which means they all choose the same route. If non-strategic customers are present in the system, this behaviour is even stronger.

Erasmus School of Economics

Contents

1 Introduction

A long wait in a queue is something that almost every customer tries to avoid. The same holds in many service networks in which services are provided at multiple stations and where customers are free to decide in which order they visit the stations. For example, think about an amusement park. There are a few attractions you want to visit, but when you are at the entrance of the park you do not know how long the queue at each attraction is. At that point, you got to decide which route you are going to follow. Many visitors try to avoid long waiting times and base their routing choice on past experiences or their expectations of waiting times. Those visitors are called strategic. However, some visitors follow the directions of an employee of the park or they just visit the stations in random order, regardless of their expectations and past experiences.

The situation stated above is called open routing in service networks. Open routing means that customers can choose in which order they want to visit the stations. This is a complex situation because all customers can have an impact on each other's waiting times. Whenever they choose a different route they can still end up at the same station at the same time. Arlotto et al. (2019) studied the behaviour of strategic customers in an open routing service network with two stations. They find that strategic customers show herding behaviour, which means they all choose the same route.

However, in many real-life service networks, not all customers are strategic. That is where my research comes in. I look at the behaviour of strategic customers when non-strategic customers are present in the system. I do this by performing a simulation study with two different cases. In the first case, I use a system with strategic customers only, just like the simulation of Arlotto et al. (2019). In the second case, the customers can be strategic or non-strategic. For these non-strategic customers, their route is determined exogenously, as in the example in the first paragraph. The strategic customers do not have any knowledge about the routes of non-strategic customers. Do the strategic customers still show herding behaviour in this situation with uncertainty about the routes of non-strategic customers or does this behaviour vanish in the presence of non-strategic customers?

The paper is structured in the following way: In Section 2 I review the literature. Section 3 describes the model that I use for this research. Section 4 explains the simulation study with the two different cases. Section 5 analyses the output from the simulation study and compares the output with the results from Arlotto et al. (2019). Finally, Section 6 concludes the paper. The code from the simulation study is provided in the appendix.

2 Literature review

In this section, I review the existing literature on open routing in service networks. Hassin and Haviv (2003) provide a general overview of the queueing literature and more recent work about this topic is covered in Hassin (2016).

Since I replicate and extend a part of Arlotto et al. (2019) this paper gives some very useful information because it analyses the same situation. At first, they come up with some analytical results concerning herding behaviour in a setting with deterministic service times and a random arrival order. They show that if the number of customers is big enough the routing game has a Nash equilibrium in which all players herd at one of the stations. Furthermore, if one of the stations serves twice (or more) as slow as the other station, then attending the slow station first is a dominant strategy. These Nash equilibria are proved to be the only existing Nash equilibria.

Arlotto et al. (2019) also show some analytical results for a case with strategic and non-strategic customers. The implementation of non-strategic customers is a bit different compared to my implementation: a non-strategic customer always chooses the same route and strategic customers know the route choices of the non-strategic customers. The results show that in this case strategic customers still herd. However, the non-strategic customers play a role in which herding profile is a Nash equilibrium.

Another closely related paper is from Parlaktürk and Kumar (2004). They analyse a stochastic system with two stations and customers who have to visit both stations to perform two tasks, which can be completed at both stations. They show the existence of unstable Nash equilibria. In this system, the stations got two queues, one for each task, and a system manager determines which queue is served at a certain time. That is the main difference with my system, where both stations have one queue and customers are served in the same order as they arrive.

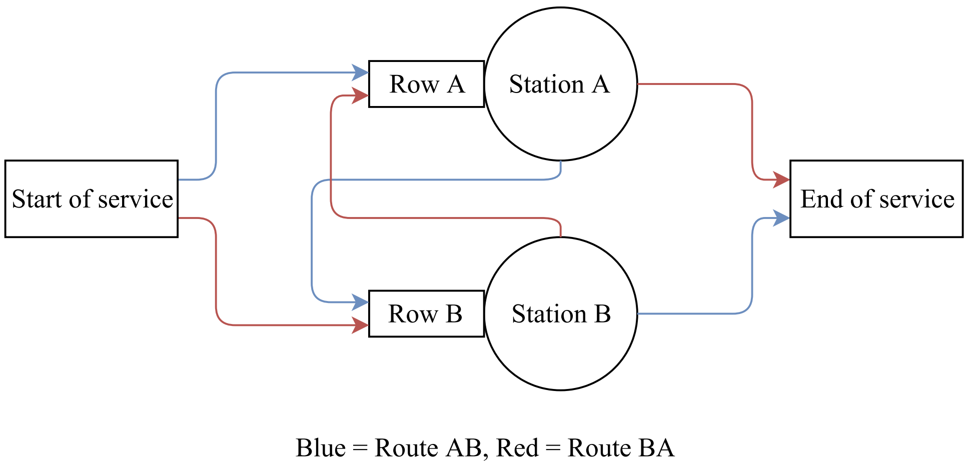
If the addressed routing choice situation is more closely examined, it can be explained as a game with incomplete information. All customers make a decision and their choices influence each other. It is called a game with incomplete information since all customers make their decision without any knowledge about the current state of the system, their position in it or the decisions of other customers. This results in higher uncertainty for customers in making the best choice. The interested reader is referred to Tadelis (2013) for more information about this game.

3 Service network with two stations

In this section, I describe the model that is used in the simulation study, which is based on the model from Arlotto et al. (2019). A visual explanation is provided in Figure 1. The model consists of a service network with two stations, A and B. Both stations have their own service rates, μ_A and μ_B respectively, and serve customers based on the FCFS rule. This means that the customers are served in the same order as they arrive. Service rate μ_A is assumed to be smaller than μ_B, so the expected service time at station A is greater than at station B. The network serves N customers who all have to visit both stations once. At arrival, a customer can decide which route, AB or BA, he wants to follow. If a customer chooses route AB he attends station A and subsequently station B, for route BA the opposite holds.

To implement the non-strategic customers I add N_S to the model. This is the total number of strategic customers, which is smaller or equal to N. The other $N - N_S$ customers are non-strategic. The route of non-strategic customers is determined exogenously. They choose, independently of each other, route AB with probability p_{AB} and route BA with probability p_{BA}.

Figure 1: The service network



4 Simulation study

For my research, I use a simulation study. I choose this method because it requires less restrictive conditions and assumptions. Compared to the analytical results from Arlotto et al. (2019), discussed in the literature section, a simulation requires almost no conditions on modelling the arrival and service times. Furthermore, it allows for modeling a wide range of learning rules to describe the dynamics of customers.

The simulation study consists of two cases. The simulation set-up is based on the set-up from Arlotto et al. (2019). In the first case, I analyse a system with only strategic customers, which is the same situation as studied by Arlotto et al. (2019). In the second case, I implement non-strategic customers in the system.

4.1 Set-up

In the simulation study, the behaviour of the customers is analysed. In both cases $N = 50$ customers. To let them learn from previous experiences 250 rounds are simulated and the results from the last round are analysed. All customers choose their route at the beginning of each round, without any knowledge about the choices of other customers. At the end of each round, the customers know their system time, but also observe which system time they would have had if they had chosen the opposite route when the choices of all other customers remain the same.

In every round customer i's arrival time is uniformly distributed on the interval $[i\gamma - \phi, i\gamma + \phi]$. Parameters γ and ϕ are used to adjust the mean and the variance, respectively. The services times at both stations are exponentially distributed. The service rate at station B, μ_B, is constant and equal to 1, the service rate μ_A is a parameter which can take multiple values. The values for the parameters mentioned above are as follows: $\mu_A \in \{0.1, 0.25, 0.5, 0.75\}$, $\gamma \in \{0.001, 0.1, 0.25, 0.5, 0.75, 1\}$ and $\phi \in \{0, 0.25, 0.5, 0.75, 1\}$. This leads to 120 different combinations of parameters and for every combination the simulation is carried out 100 times.

4.2 Strategic Customers

In the first case, all customers are strategic, which means they want to minimise their system time. That is why they use the historical system times in their route choice by taking the average of all historical system times and choosing the route which has the smallest average historical system time. In the first round, the choice is completely random because there is no historical information yet.

4.3 Non-strategic Customers

In the second case, non-strategic customers are present in the system. I simulate the system for $N_S \in \{30, 40\}$. Before the start of the simulation for all rounds I randomly take $N - N_S$ customer indices to assign which $N - N_S$ customers are non-strategic that round. Since they are non-strategic their route is not based on historical system times. They choose, independently of each other, route AB with probability $p_{AB} = 0.5$ and route BA with probability $p_{BA} = 1 - p_{AB} = 0.5$. After every round, I use the system times of customers who had a strategic role that round to update their historical average system times, on which they base their route choice. The historical average system times of the customers who had a non-strategic role that round remains the same. I use this set-up to keep the number of customers consistent with case 1. However, the results from case 1 are coming from 50 customers which are strategic in 250 rounds. That is why I adjust the number of rounds in the following way:

$$\#Rounds = 250 * \frac{N}{N_S} \tag{1}$$

To make a good comparison I increase the number of rounds so every customer in case 2 has the expectation to be strategic in 250 rounds. After the last round, I use the historical average waiting times to see how many customers would now choose route AB and how many would choose route BA.

4.4 Comparison

In their paper, Arlotto et al. (2019) only present a selection of sample means, without providing any information about the standard deviation. To be able to compare the results from the first part of my research with the outcomes from Arlotto et al. (2019) I use a one-sample t-test, which tests whether my sample mean is significantly different compared to their sample mean and only uses the standard deviation of my sample. I do this for every parameter set to get to the general conclusion whether the distributions of the number of AB customers are the same. However, due to the lack of information I now assume the mean of Arlotto et al. (2019) to be the true mean. This means that I underestimate the variance of the difference between both samples and that I reject the null hypothesis faster.

The test statistic is determined in the following way:

$$t = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}} \tag{2}$$

in which $\bar{X}$ is my sample mean, μ equals the sample mean to compare with, σ is the standard deviation of my sample and n is the number of observations in my sample. This test statistic assumes that the sample mean is normally distributed. This is not exactly the case here. However, the central limit theory states that if the sample size is large enough the distribution of the mean is approximately normal. Kwak and Kim (2017) show that this holds for a sample size of 30 or more. Since I deal with a sample size of 50, using this t-test is still reasonable.

5 Results

5.1 Case 1

Table 1 shows some sample means and ranges of the number of customers who choose AB in the last round for a situation in which the arrival times are deterministic and the arrival order is constant for every round, since $\phi = 0$ and $\gamma = 0.001$. This situation is related to an analytical result from Arlotto et al. (2019) since they also analysed a situation with deterministic service times and a fixed arrival order. The main difference here is that customers can not observe moves from customers who are in the system already. The present an equilibrium in which the first 49 customers choose AB and the last one chooses BA. All results from Table 1 are close to this value. However, possibly due to the fact that I now deal with stochastic service times, not all outcomes are 49, but there is a range in which all outcomes are situated, as shown in the table.

Table 1: Sample means and ranges of the number of AB customers in the last round with $\gamma = 0.001$ and $\phi = 0$

μ_A	0.1	0.25	0.5	0.75
Mean	49.30	49.99	48.92	47.71
Range	49 - 50	49 - 50	48 - 50	45 - 50

When all simulations are compared, Arlotto et al. (2019) notice that herding at BA only happens when $\gamma = 0.001$. The same holds in my simulations. Table 2 and 3 only show results with $\gamma \geq 0.1$, so the means are a good indicator for herding behaviour. The closer the mean is to 50, the stronger the herding behaviour is.

All results in Table 3 are at least 47.16 and are greater than the corresponding values in Table 2. This supports the analytical results from Arlotto et al. (2019) and shows that if the amount of time you have to wait behind another customer at station A increases, more people tend to choose route AB. Furthermore, we can see that, in general, if γ increases the number of AB customers decreases. This can be explained by the fact that if γ increases, arrivals occur less frequently and people influence each other less. For ϕ the general effect is a bit harder to observe, because we need to compare it with γ. If ϕ is greater than γ the arrival intervals of individuals overlap, so there is more uncertainty about the arrival order. If γ increases, the influence of ϕ decreases. As an example, I look at Table 2 with $\gamma = 0.1$. If $\phi = 1$, customer 25 his arrival position can be between sixth en forty-forth, but when $\phi = 0$ his arrival position is twenty-fifth. This difference in uncertainty about positions leads to the difference in means, which is clearly visible in this situation, but not for all values of γ.

Table 2: Sample means of the number of customers who choose AB in the last round with $\mu_A = 0.75$. My results are on the left, results from Arlotto et al. (2019) are on the right.

	ϕ						ϕ				
γ	0	0.25	0.5	0.75	1	γ	0	0.25	0.5	0.75	1
0.1	46.53	46.50	45.66	47.54	48.99*	0.1	46.22	46.61	45.42	47.45	49.20
0.25	46.38	45.95	45.87	46.23	45.94	0.25	46.55	45.83	46.22	46.07	45.72
0.5	44.29*	44.21*	45.06	44.53	44.64	0.5	45.20	45.27	45.00	44.64	44.32
0.75	41.91	41.25*	42.30	41.94	40.98*	0.75	41.68	42.70	41.59	41.73	42.46
1	36.78	36.46*	36.67	37.05	37.62	1	35.90	37.38	36.47	36.23	38.17

Notes: * denotes a significant difference in means at the 5% level.

Table 3: Sample means of the number of customers who choose AB in the last round with $\mu_A = 0.5$. My results are on the left, results from Arlotto et al. (2019) are on the right.

	ϕ						ϕ				
γ	0	0.25	0.5	0.75	1	γ	0	0.25	0.5	0.75	1
0.1	48.62	48.47	49.99	50.00	50.00	0.1	48.47	48.53	49.98	50.00	50.00
0.25	48.53*	48.49	47.83	48.60	49.25	0.25	48.34	48.39	47.81	48.61	49.18
0.5	48.13	47.97	48.01*	48.13	47.67	0.5	48.05	48.07	48.30	48.10	47.82
0.75	47.75	47.84	47.71	47.67	47.81	0.75	47.88	47.9	47.81	47.80	47.76
1	47.43	47.16	47.22*	47.40	47.35	1	47.48	47.42	47.47	47.52	47.33

Notes: * denotes a significant difference in means at the 5% level.

5.2 Case 2

The results for the case with non-strategic customers can be found in Tables 4 and 5. Compared to the results from case 1 all outcomes are higher. Furthermore, if we compare the situation of $N_S = 30$ with $N_S = 40$ we see the outcomes of $N_S = 30$ are higher. The addition of non-strategic customers does not seem to lower the herding behaviour of strategic customers. A possible explanation for the higher number of AB customers in the presence of non-strategic customers could be the threat for a strategic customer that the large majority of customers that enter the system after him, but choose route BA, arrives before him at station B if he chooses route AB. This threat is lower in the situation with non-strategic customers, because they choose routes AB and BA with equal probabilities so a large majority is very rare here. This thread is one of the reasons why customers can choose BA and since this thread is lower in a situation with more non-strategic customers, it could be an explanation why we see more AB customers in a setting with non-strategic customers.

Table 4: Sample means of the number of customers who choose AB in the last round with $\mu_A = 0.75$. Results with $N_S = 30$ are on the left, those with $N_S = 40$ are on the right.

	ϕ						ϕ				
γ	0	0.25	0.5	0.75	1	γ	0	0.25	0.5	0.75	1
0.1	50.00	50.00	50.00	50.00	50.00	0.1	49.84	50.00	50.00	50.00	50.00
0.25	50.00	50.00	50.00	50.00	50.00	0.25	47.77	47.88	48.19	48.84	49.39
0.5	49.23	49.15	49.42	49.58	49.75	0.5	46.95	46.78	46.93	46.83	46.73
0.75	46.56	46.43	46.38	46.63	46.72	0.75	44.63	44.64	44.96	45.04	44.95
1	38.64	38.90	39.34	39.60	40.60	1	37.86	37.79	38.57	38.79	39.14

Table 5: Sample means of the number of customers who choose AB in the last round with $\mu_A = 0.50$. Results with $N_S = 30$ are on the left, those with $N_S = 40$ are on the right.

	ϕ						ϕ				
γ	0	0.25	0.5	0.75	1	γ	0	0.25	0.5	0.75	1
0.1	50.00	50.00	50.00	50.00	50.00	0.1	50.00	50.00	50.00	50.00	50.00
0.25	50.00	50.00	50.00	50.00	50.00	0.25	49.99	50.00	50.00	50.00	50.00
0.5	50.00	50.00	50.00	50.00	50.00	0.5	48.59	48.75	48.70	49.08	49.27
0.75	49.25	49.19	49.33	49.45	49.70	0.75	48.32	48.29	48.35	48.17	48.14
1	48.07	48.33	48.23	48.39	48.55	1	47.93	47.95	47.98	47.75	47.77

6 Conclusion

I use the model from Arlotto et al. (2019) about a service network with two stations to analyse the behaviour of strategic customers in the presence of non-strategic customers. All customers need service at both stations and are free to choose their route through the system. If all customers are strategic they show herding behaviour, which means that all customers choose the same route. They do this to avoid long waiting times at the congested station. A bigger difference between the service rates increases the tendency to visit the slow station first. If the arrival times of customers are further apart, which means the system is less congested, the herding behaviour decreases. Apparently, there are some differences between my implication and the implication of Arlotto et al. (2019). Further research could explain these differences.

The presence of non-strategic customers even strengthens the herding behaviour of strategic customers. The uncertainty for a strategic customer about the choices of customers who arrive after him can play a role in this effect. The implementation of non-strategic customers can be developed in further research. I choose to keep the number of customers consistent with the case with strategic customers only and every round I randomly select $N - N_S$ customer indices to give them a non-strategic role. Another option could be to keep the number of strategic customers consistent and to increase the total number of customers. The difficulty of this implication is the influence of the customer index. This index is a factor in the arrival interval, so the allocation of indices over strategic and non-strategic customers is important here.

In this service network is assumed that customers can not observe the status of the system when they arrive. It could be interesting to analyse a system in which customers can base their routing choice on historical average system times and on the current status of system. Besides, it could also be interesting to adjust the utility function. Now customers only want to minimise their system time, but perhaps there are other preferences which should be incorporated too.

References

Arlotto, A., Frazelle, A. E., and Weib, Y. (2019). Strategic open routing in service networks. *Management Science*, 65(2):735–750.

Hassin, R. (2016). *Rational queueing*. Chapman and Hall/CRC.

Hassin, R. and Haviv, M. (2003). *To queue or not to queue: Equilibrium behavior in queueing systems*, volume 59. Springer Science & Business Media.

Kwak, S. G. and Kim, J. H. (2017). Central limit theorem: the cornerstone of modern statistics. *Korean journal of anesthesiology*, 70(2):144.

Parlaktürk, A. K. and Kumar, S. (2004). Self-interested routing in queueing networks. *Management Science*, 50(7):949–966.

Tadelis, S. (2013). *Game theory: an introduction*. Princeton University Press.

7 Appendix

```
###########################################
# Simulation study for all parameter sets #
###########################################

#A time function to see how long the program runs
startTime <- Sys.time()

#Set the parameter values
gammas <- c(0.001, 0.1, 0.25, 0.5, 0.75, 1)
phis <- c(0, 0.25, 0.5, 0.75, 1)
muAs <- c(0.1, 0.25, 0.5, 0.75)
muB <- 1

#Set the amount of customers, rounds and trials
N <- 50
R <- 250
A <- 100
G <- 6
P <- 5
M <- 4

#Create a matrix to save the results
results <- matrix(0, nrow = A, ncol = G*P*M)

#Select a seed for the main program
set.seed(200)

for (g in 1:G){
  gamma <- gammas[g]
  for (p in 1:P) {
    phi <- phis[p]
    for (m in 1:M) {
      muA <- muAs[m]
      for (a in 1:A){
```

```r
#Create matrices for the expected system times, the chosen
   strategies and the arrivals
expectedSystemTimes <- matrix(NA, nrow = R, ncol = 2*N)
strategies <- matrix(NA, nrow = R, ncol = N)
arrivalOrder <- matrix(NA, nrow = R, ncol = N)

#Generate the first strategies random
strategies[1,] <- rbinom(N, 1, 0.5)

#Start simulating a round
for (r in 1:R){
  serviceTimes <- matrix(NA, nrow = N, ncol = 2)
  serviceTimes[,1] <- rexp(n = N, rate = muA)
  serviceTimes[,2] <- rexp(n = N, rate = muB)
  arrivalTimes <- matrix(NA, nrow = N, ncol = 2)
  arrivalTimes[,1] <- c(1:50)
  for (i in 1:N){
    #Choose the strategy based on the expected system times
    if (r!=1) {
      if (expectedSystemTimes[r-1, 2*i-1] <= expectedSystemTimes[r
         -1, 2*i]){
        strategies[r,i] <- 1
      }
      else{
        strategies[r,i] <- 0
      }
    }
    #Generate the arrival times
    arrivalTimes[i,2] <- runif(1, min = i*gamma - phi, max = i*
       gamma + phi)
  }
  #Order the arrivals
  arrivalOrder[r,] <- arrivalTimes[order(arrivalTimes[,2])]
```

```
#Create a matrix to store the system times
systemTimes <- matrix(NA, nrow = N, ncol = 2)

#Initialise some variables
servedCustomersA <- 0
servedCustomersB <- 0
arrivedCustomers <- 0
arrivedCustomersA <- 0
arrivedCustomersB <- 0
queuelengthA <- 0
queuelengthB <- 0

#Create matrices to keep track of order in which the customers
   visit the stations
queueA <- matrix(NA, nrow = N, ncol = 1)
queueB <- matrix(NA, nrow = N, ncol = 1)

#The times for the next events
arrival <- Inf
departureA <- Inf
departureB <- Inf


#Simulate the first arrival
arrivedCustomers <- arrivedCustomers+1
time <- arrivalTimes[arrivalOrder[r, arrivedCustomers],2]
arrival <- arrivalTimes[arrivalOrder[r, arrivedCustomers+1],2]
if(strategies[r, arrivalOrder[r, arrivedCustomers]]==1){
  departureA <- time + serviceTimes[arrivalOrder[r,
     arrivedCustomers],1]
  queuelengthA <- queuelengthA+1
  arrivedCustomersA <- arrivedCustomersA+1
  queueA[arrivedCustomersA] <- arrivalOrder[r, arrivedCustomers]
}
else{
  departureB <- time + serviceTimes[arrivalOrder[r,
      arrivedCustomers],2]
  queuelengthB <- queuelengthB+1
  arrivedCustomersB <- arrivedCustomersB+1
  queueB[arrivedCustomersB] <- arrivalOrder[r, arrivedCustomers]
}

#This keeps running until everybody is out of the system
while(servedCustomersA<N | servedCustomersB<N){

  #The steps that need to be taken if the next event is an
      arrival
  if(min(arrival, departureA, departureB)==arrival){
    arrivedCustomers <- arrivedCustomers+1
    time <- arrival
    if(arrivedCustomers<50){
      arrival <- arrivalTimes[arrivalOrder[r, arrivedCustomers
          +1],2]
    }
    else{
      arrival <- Inf
    }
    if(strategies[r, arrivalOrder[r, arrivedCustomers]]==1){
      arrivedCustomersA <- arrivedCustomersA+1
      queueA[arrivedCustomersA] <- arrivalOrder[r,
          arrivedCustomers]
      if(queuelengthA==0){
        departureA <- time + serviceTimes[queueA[
            arrivedCustomersA],1]
      }
      queuelengthA <- queuelengthA+1
    }
    else{
      arrivedCustomersB <- arrivedCustomersB+1
      queueB[arrivedCustomersB] <- arrivalOrder[r,
         arrivedCustomers]
      if(queuelengthB==0){
        departureB <- time + serviceTimes[queueB[
           arrivedCustomersB],2]
      }
      queuelengthB <- queuelengthB+1
    }
  }

  #The steps that need to be taken if the next event is a
      departure at station A
  else if(min(arrival, departureA, departureB)==departureA){
    servedCustomersA <- servedCustomersA+1
    time <- departureA
    queuelengthA <- queuelengthA-1
    if(queuelengthA!=0){
      departureA <- time + serviceTimes[queueA[servedCustomersA
         +1],1]
    }
    else{
      departureA <- Inf
    }
    if(strategies[r, queueA[servedCustomersA]]==1){
      if(queuelengthB==0){
        departureB <- time + serviceTimes[queueA[servedCustomersA
           ],2]
      }
      queuelengthB <- queuelengthB+1
      arrivedCustomersB <- arrivedCustomersB+1
      queueB[arrivedCustomersB] <- queueA[servedCustomersA]
    }
    else{
      systemTimes[queueA[servedCustomersA],2] <- time -
         arrivalTimes[queueA[servedCustomersA],2]
    }
  }

  #The steps that need to be taken if the next event is a
     departure at station B
  else if(min(arrival, departureA, departureB)==departureB){
    servedCustomersB <- servedCustomersB+1
    time <- departureB
    queuelengthB <- queuelengthB-1
    if(queuelengthB!=0){
      departureB <- time + serviceTimes[queueB[servedCustomersB
         +1],2]
    }
    else{
      departureB <- Inf
    }
    if(strategies[r, queueB[servedCustomersB]] == 0){
      if(queuelengthA==0){
        departureA <- time + serviceTimes[queueB[servedCustomersB
           ],1]
      }
      queuelengthA <- queuelengthA+1
      arrivedCustomersA <- arrivedCustomersA+1
      queueA[arrivedCustomersA] <- queueB[servedCustomersB]
    }
    else{
      systemTimes[queueB[servedCustomersB],1] <- time -
         arrivalTimes[queueB[servedCustomersB],2]
    }
  }
}

#Run the system another 50 times, each time change the strategy
   of one customer to obtain his system time if the other
   strategy was chosen
for(i in 1:N){
```

```r
if(strategies[r,i]==1){
  strategies[r,i] <- 0
}
else{
  strategies[r,i] <- 1
}

#Initialise a new variable, stop the system if customer i is
    served at both stations
customerServed <- 0

#Initialise some variables
servedCustomersA <- 0
servedCustomersB <- 0
arrivedCustomers <- 0
arrivedCustomersA <- 0
arrivedCustomersB <- 0
queuelengthA <- 0
queuelengthB <- 0

#Create matrices to keep track of order in which the customers
     visit the stations
queueA <- matrix(NA, nrow = N, ncol = 1)
queueB <- matrix(NA, nrow = N, ncol = 1)

#The times for the next events
arrival <- Inf
departureA <- Inf
departureB <- Inf


#Simulate the first arrival
arrivedCustomers <- arrivedCustomers+1
time <- arrivalTimes[arrivalOrder[r, arrivedCustomers],2]
arrival <- arrivalTimes[arrivalOrder[r, arrivedCustomers+1],2]
```

```
if (strategies[r, arrivalOrder[r, arrivedCustomers]]==1){
  departureA <- time + serviceTimes[arrivalOrder[r,
      arrivedCustomers],1]
  queuelengthA <- queuelengthA+1
  arrivedCustomersA <- arrivedCustomersA+1
  queueA[arrivedCustomersA] <- arrivalOrder[r,
      arrivedCustomers]
}
else{
  departureB <- time + serviceTimes[arrivalOrder[r,
      arrivedCustomers],2]
  queuelengthB <- queuelengthB+1
  arrivedCustomersB <- arrivedCustomersB+1
  queueB[arrivedCustomersB] <- arrivalOrder[r,
      arrivedCustomers]
}

#This keeps running until customer i is out of the system
while(customerServed < 1){

  #The steps that need to be taken if the next event is an
      arrival
  if(min(arrival, departureA, departureB)==arrival){
    arrivedCustomers <- arrivedCustomers+1
    time <- arrival
    if(arrivedCustomers<50){
      arrival <- arrivalTimes[arrivalOrder[r, arrivedCustomers
          +1],2]
    }
    else{
      arrival <- Inf
    }
    if (strategies[r, arrivalOrder[r, arrivedCustomers]]==1){
      arrivedCustomersA <- arrivedCustomersA+1
      queueA[arrivedCustomersA] <- arrivalOrder[r,
        arrivedCustomers]
      if(queuelengthA==0){
        departureA <- time + serviceTimes[queueA[
            arrivedCustomersA],1]
      }
      queuelengthA <- queuelengthA+1
    }
    else{
      arrivedCustomersB <- arrivedCustomersB+1
      queueB[arrivedCustomersB] <- arrivalOrder[r,
        arrivedCustomers]
      if(queuelengthB==0){
        departureB <- time + serviceTimes[queueB[
            arrivedCustomersB],2]
      }
      queuelengthB <- queuelengthB+1
    }
  }

  #The steps that need to be taken if the next event is a
      departure at station A
  else if(min(arrival, departureA, departureB)==departureA){
    servedCustomersA <- servedCustomersA+1
    time <- departureA
    queuelengthA <- queuelengthA-1
    if(queuelengthA!=0){
      departureA <- time + serviceTimes[queueA[servedCustomersA
        +1],1]
    }
    else{
      departureA <- Inf
    }
    if(strategies[r, queueA[servedCustomersA]]==1){
      if(queuelengthB==0){
        departureB <- time + serviceTimes[queueA[
          servedCustomersA],2]
    }
    queuelengthB <- queuelengthB+1
    arrivedCustomersB <- arrivedCustomersB+1
    queueB[arrivedCustomersB] <- queueA[servedCustomersA]
  }
  else{
    if(queueA[servedCustomersA]==i){
      systemTimes[queueA[servedCustomersA],2] <- time -
          arrivalTimes[queueA[servedCustomersA],2]
      customerServed <- 1
    }
  }
}

#The steps that need to be taken if the next event is a
    departure at station B
else if(min(arrival, departureA, departureB)==departureB){
  servedCustomersB <- servedCustomersB+1
  time <- departureB
  queuelengthB <- queuelengthB-1
  if(queuelengthB!=0){
    departureB <- time + serviceTimes[queueB[servedCustomersB
        +1],2]
  }
  else{
    departureB <- Inf
  }
  if(strategies[r, queueB[servedCustomersB]]==0){
    if(queuelengthA==0){
      departureA <- time + serviceTimes[queueB[
          servedCustomersB],1]
    }
    queuelengthA <- queuelengthA+1
    arrivedCustomersA <- arrivedCustomersA+1
```

```
          queueA[arrivedCustomersA] <- queueB[servedCustomersB]
        }
        else{
          if(queueB[servedCustomersB]==i){
            systemTimes[queueB[servedCustomersB],1] <- time -
               arrivalTimes[queueB[servedCustomersB],2]
            customerServed <- 1
          }
        }
      }
    }
    #Change the strategy of player i to its initial choice
    if(strategies[r,i]==1){
      strategies[r,i] <- 0
    }
    else{
      strategies[r,i] <- 1
    }
  }
  #update expected system times
  if(r==1){
    for(i in 1:N){
      expectedSystemTimes[r, 2*i-1] <- systemTimes[i,1]
      expectedSystemTimes[r, 2*i] <- systemTimes[i,2]
    }
  }
  else{
    for(i in 1:N){
      expectedSystemTimes[r, 2*i-1] <- ((r-1)*expectedSystemTimes[
         r-1, 2*i-1] + systemTimes[i,1])/r
      expectedSystemTimes[r, 2*i] <- ((r-1)*expectedSystemTimes[r
         -1, 2*i] + systemTimes[i,2])/r
    }
  }
}
```

```
        #Check how many times strategy 1 is chosen
        for (i in 1:N){
          if (strategies[R,i] == 1){
            results[a, m + (p-1)*M + (g-1)*P*M] <- results[a, m + (p-1)*M
                + (g-1)*P*M]+1
          }
        }
      }
    }
  }
}

count <- matrix(0, nrow = N, ncol = G*P*M)
gemiddeld <- matrix(NA, nrow = G*P*M, ncol = 1)

for (j in 1:(G*P*M)){
  for (i in 1:A){
    count[results[i,j],j] <- count[results[i,j],j] + 1
  }
}
for (j in 1:(G*P*M)){
  teller <- 0
  for (i in 1:N){
    teller <- teller + count[i,j]*i
  }
  gemiddeld[j] <- teller/A
}

#Check how long the program has run
endTime <- Sys.time()
timeTaken <- endTime - startTime
beep(sound=3, expr=NULL)
```

```
##################################################################
# Simulation study for all parameter sets with strategic and non-
   strategic customers
##################################################################

#A time function to see how long the program runs
startTime <- Sys.time()

#Set the parameter values
gammas <- c(0.001, 0.1, 0.25, 0.5, 0.75, 1)
phis <- c(0, 0.25, 0.5, 0.75, 1)
muAs <- c(0.1, 0.25, 0.5, 0.75)
muB <- 1
pAB <- 0.5

#Set the amount of customers, rounds and trials
N <- 50
N_S <- 40
R <- 312
A <- 100

#Set the length of parameter vectors
G <- 6
P <- 5
M <- 4

#Create a matrix to save the results
results <- matrix(0, nrow = A, ncol = G*P*M)

#Select a seed for the selection of customers
set.seed(100)

#Randomly select which customers are non-strategic for every round in
   every trial
types <- matrix(NA, nrow = (R*A), ncol = N)
```

```
typesVector <- matrix(NA, nrow = N, ncol = 1)
for(i in 1:N){
  if(i<=(N-N_S)){
    typesVector[i] <- 1
  }
  else{
    typesVector[i] <- 0
  }
}
for(i in 1:(R*A)){
  types[i,] <- typesVector[sample(nrow(typesVector)),]
}

#Select a seed for the main program
set.seed(200)

for (g in 1:G){
  gamma <- gammas[g]
  for (p in 1:P) {
    phi <- phis[p]
    for (m in 1:M) {
      muA <- muAs[m]
      for (a in 1:A){

        #Create matrices for the expected system times, the chosen
           strategies and the arrivals
        expectedSystemTimes <- matrix(NA, nrow = R, ncol = 2*N)
        strategies <- matrix(NA, nrow = R, ncol = N)
        arrivalOrder <- matrix(NA, nrow = R, ncol = N)

        #Generate the first strategies
        for(i in 1:N){
          if(types[(1+((a-1)*R)),i]==1){
            strategies[1,i] <- rbinom(1, 1, pAB)
          }
          else{
            strategies[1,i] <- rbinom(1, 1, 0.5)
          }
        }

        #Start simulating a round
        for (r in 1:R){
          serviceTimes <- matrix(NA, nrow = N, ncol = 2)
          serviceTimes[,1] <- rexp(n = N, rate = muA)
          serviceTimes[,2] <- rexp(n = N, rate = muB)
          arrivalTimes <- matrix(NA, nrow = N, ncol = 2)
          arrivalTimes[,1] <- c(1:50)
          for (i in 1:N){
            #Choose the strategy based on the expected system times
            if(r!=1){
              if(types[(r+((a-1)*R)),i]!=1){
                if (expectedSystemTimes[r-1, 2*i-1] < expectedSystemTimes[r
                   -1, 2*i]){
                  strategies[r,i] <- 1
                }
                else if(expectedSystemTimes[r-1, 2*i-1] >
                   expectedSystemTimes[r-1, 2*i]){
                  strategies[r,i] <- 0
                }
                else if(expectedSystemTimes[r-1, 2*i-1] ==
                   expectedSystemTimes[r-1, 2*i]){
                  strategies[r,i] <- rbinom(1, 1, 0.5)
                }
              }
              else{
                strategies[r,i] <- rbinom(1, 1, pAB)
              }
            }

            #Generate the arrival times
          arrivalTimes[i,2] <- runif(1, min = i*gamma - phi, max = i*
              gamma + phi)
        }
        #Order the arrivals
        arrivalOrder[r,] <- arrivalTimes[order(arrivalTimes[,2])]

        #Create a matrix to store the system times
        systemTimes <- matrix(NA, nrow = N, ncol = 2)

        #Initialise some variables
        servedCustomersA <- 0
        servedCustomersB <- 0
        arrivedCustomers <- 0
        arrivedCustomersA <- 0
        arrivedCustomersB <- 0
        queuelengthA <- 0
        queuelengthB <- 0

        #Create matrices to keep track of order in which the customers
            visit the stations
        queueA <- matrix(NA, nrow = N, ncol = 1)
        queueB <- matrix(NA, nrow = N, ncol = 1)

        #The times for the next events
        arrival <- Inf
        departureA <- Inf
        departureB <- Inf



        #Simulate the first arrival
        arrivedCustomers <- arrivedCustomers+1
        time <- arrivalTimes[arrivalOrder[r, arrivedCustomers],2]
        arrival <- arrivalTimes[arrivalOrder[r, arrivedCustomers+1],2]
        if(strategies[r, arrivalOrder[r, arrivedCustomers]]==1){
          departureA <- time + serviceTimes[arrivalOrder[r,
                  arrivedCustomers],1]
              queuelengthA <- queuelengthA+1
              arrivedCustomersA <- arrivedCustomersA+1
              queueA[arrivedCustomersA] <- arrivalOrder[r, arrivedCustomers]
            }
            else{
              departureB <- time + serviceTimes[arrivalOrder[r,
                  arrivedCustomers],2]
              queuelengthB <- queuelengthB+1
              arrivedCustomersB <- arrivedCustomersB+1
              queueB[arrivedCustomersB] <- arrivalOrder[r, arrivedCustomers]
            }

            #This keeps running until everybody is out of the system
            while(servedCustomersA<N | servedCustomersB<N){

              #The steps that need to be taken if the next event is an
                  arrival
              if(min(arrival, departureA, departureB)==arrival){
                arrivedCustomers <- arrivedCustomers+1
                time <- arrival
                if(arrivedCustomers<50){
                  arrival <- arrivalTimes[arrivalOrder[r, arrivedCustomers
                      +1],2]
                }
                else{
                  arrival <- Inf
                }
                if(strategies[r, arrivalOrder[r, arrivedCustomers]]==1){
                  arrivedCustomersA <- arrivedCustomersA+1
                  queueA[arrivedCustomersA] <- arrivalOrder[r,
                      arrivedCustomers]
                  if(queuelengthA==0){
                    departureA <- time + serviceTimes[queueA[
                        arrivedCustomersA],1]
                  }
                  queuelengthA <- queuelengthA+1
                }
                else{
                  arrivedCustomersB <- arrivedCustomersB+1
                  queueB[arrivedCustomersB] <- arrivalOrder[r,
                      arrivedCustomers]
                  if(queuelengthB==0){
                    departureB <- time + serviceTimes[queueB[
                        arrivedCustomersB],2]
                  }
                  queuelengthB <- queuelengthB+1
                }
              }

              #The steps that need to be taken if the next event is a
                  departure at station A
              else if(min(arrival, departureA, departureB)==departureA){
                servedCustomersA <- servedCustomersA+1
                time <- departureA
                queuelengthA <- queuelengthA-1
                if(queuelengthA!=0){
                  departureA <- time + serviceTimes[queueA[servedCustomersA
                      +1],1]
                }
                else{
                  departureA <- Inf
                }
                if(strategies[r, queueA[servedCustomersA]]==1){
                  if(queuelengthB==0){
                    departureB <- time + serviceTimes[queueA[servedCustomersA
                        ],2]
                  }
                  queuelengthB <- queuelengthB+1
                  arrivedCustomersB <- arrivedCustomersB+1
                  queueB[arrivedCustomersB] <- queueA[servedCustomersA]
                }
                else{
                  systemTimes[queueA[servedCustomersA],2] <- time -
                      arrivalTimes[queueA[servedCustomersA],2]
                }
              }

              #The steps that need to be taken if the next event is a
                  departure at station B
              else if(min(arrival, departureA, departureB)==departureB){
                servedCustomersB <- servedCustomersB+1
                time <- departureB
                queuelengthB <- queuelengthB-1
                if(queuelengthB!=0){
                  departureB <- time + serviceTimes[queueB[servedCustomersB
                     +1],2]
                }
                else{
                  departureB <- Inf
                }
                if(strategies[r, queueB[servedCustomersB]] == 0){
                  if(queuelengthA==0){
                    departureA <- time + serviceTimes[queueB[servedCustomersB
                       ],1]
                  }
                  queuelengthA <- queuelengthA+1
                  arrivedCustomersA <- arrivedCustomersA+1
                  queueA[arrivedCustomersA] <- queueB[servedCustomersB]
                }
                else{
                  systemTimes[queueB[servedCustomersB],1] <- time -
                     arrivalTimes[queueB[servedCustomersB],2]
                }
              }
```

```
}

#Run the system another N-N_S times, each time change the
   strategy of one customer to obtain his system time if the
   other strategy was chosen
for(i in 1:N){
  if(types[(r+((a-1)*R)),i]!=1){
    if(strategies[r,i]==1){
      strategies[r,i] <- 0
    }
    else{
      strategies[r,i] <- 1
    }



    #Initialise a new variable, stop the system if customer i is
       served at both stations
    customerServed <- 0

    #Initialise some variables
    servedCustomersA <- 0
    servedCustomersB <- 0
    arrivedCustomers <- 0
    arrivedCustomersA <- 0
    arrivedCustomersB <- 0
    queuelengthA <- 0
    queuelengthB <- 0

    #Create matrices to keep track of order in which the
       customers visit the stations
    queueA <- matrix(NA, nrow = N, ncol = 1)
    queueB <- matrix(NA, nrow = N, ncol = 1)

    #The times for the next events
    arrival <- Inf
```

```
departureA <- Inf
departureB <- Inf


#Simulate the first arrival
arrivedCustomers <- arrivedCustomers+1
time <- arrivalTimes[arrivalOrder[r, arrivedCustomers],2]
arrival <- arrivalTimes[arrivalOrder[r, arrivedCustomers
   +1],2]
if (strategies[r, arrivalOrder[r, arrivedCustomers]]==1){
  departureA <- time + serviceTimes[arrivalOrder[r,
     arrivedCustomers],1]
  queuelengthA <- queuelengthA+1
  arrivedCustomersA <- arrivedCustomersA+1
  queueA[arrivedCustomersA] <- arrivalOrder[r,
     arrivedCustomers]
}
else{
  departureB <- time + serviceTimes[arrivalOrder[r,
     arrivedCustomers],2]
  queuelengthB <- queuelengthB+1
  arrivedCustomersB <- arrivedCustomersB+1
  queueB[arrivedCustomersB] <- arrivalOrder[r,
     arrivedCustomers]
}

#This keeps running until customer i is out of the system
while(customerServed < 1){

  #The steps that need to be taken if the next event is an
     arrival
  if(min(arrival, departureA, departureB)==arrival){
    arrivedCustomers <- arrivedCustomers+1
    time <- arrival
    if(arrivedCustomers<50){
```

```
    arrival <- arrivalTimes[arrivalOrder[r,
        arrivedCustomers+1],2]
  }
  else{
    arrival <- Inf
  }
  if (strategies[r, arrivalOrder[r, arrivedCustomers]]==1){
    arrivedCustomersA <- arrivedCustomersA+1
    queueA[arrivedCustomersA] <- arrivalOrder[r,
        arrivedCustomers]
    if(queuelengthA==0){
      departureA <- time + serviceTimes[queueA[
          arrivedCustomersA],1]
    }
    queuelengthA <- queuelengthA+1
  }
  else{
    arrivedCustomersB <- arrivedCustomersB+1
    queueB[arrivedCustomersB] <- arrivalOrder[r,
        arrivedCustomers]
    if(queuelengthB==0){
      departureB <- time + serviceTimes[queueB[
          arrivedCustomersB],2]
    }
    queuelengthB <- queuelengthB+1
  }
}

#The steps that need to be taken if the next event is a
    departure at station A
else if(min(arrival, departureA, departureB)==departureA){
  servedCustomersA <- servedCustomersA+1
  time <- departureA
  queuelengthA <- queuelengthA-1
  if(queuelengthA!=0){
    departureA <- time + serviceTimes[queueA[
        servedCustomersA+1],1]
  }
  else{
    departureA <- Inf
  }
  if(strategies[r, queueA[servedCustomersA]]==1){
    if(queuelengthB==0){
      departureB <- time + serviceTimes[queueA[
          servedCustomersA],2]
    }
    queuelengthB <- queuelengthB+1
    arrivedCustomersB <- arrivedCustomersB+1
    queueB[arrivedCustomersB] <- queueA[servedCustomersA]
  }
  else{
    if(queueA[servedCustomersA]==i){
      systemTimes[queueA[servedCustomersA],2] <- time -
          arrivalTimes[queueA[servedCustomersA],2]
      customerServed <- 1
    }
  }
}

#The steps that need to be taken if the next event is a
    departure at station B
else if(min(arrival, departureA, departureB)==departureB){
  servedCustomersB <- servedCustomersB+1
  time <- departureB
  queuelengthB <- queuelengthB-1
  if(queuelengthB!=0){
    departureB <- time + serviceTimes[queueB[
        servedCustomersB+1],2]
  }
  else{
          departureB <- Inf
        }
        if(strategies[r, queueB[servedCustomersB]]==0){
          if(queuelengthA==0){
            departureA <- time + serviceTimes[queueB[
                servedCustomersB],1]
          }
          queuelengthA <- queuelengthA+1
          arrivedCustomersA <- arrivedCustomersA+1
          queueA[arrivedCustomersA] <- queueB[servedCustomersB]
        }
        else{
          if(queueB[servedCustomersB]==i){
            systemTimes[queueB[servedCustomersB],1] <- time -
                arrivalTimes[queueB[servedCustomersB],2]
            customerServed <- 1
          }
        }
      }
    }
    #Change the strategy of player i to its initial choice
    if(strategies[r,i]==1){
      strategies[r,i] <- 0
    }
    else{
      strategies[r,i] <- 1
    }
  }
}
#update expected system times
if(r==1){
  for(i in 1:N){
    if(types[(r+((a-1)*R)),i]!=1){
      expectedSystemTimes[r, 2*i-1] <- systemTimes[i,1]
      expectedSystemTimes[r, 2*i] <- systemTimes[i,2]
```

```
        }
        else{
          expectedSystemTimes[r, 2*i-1] <- 0
          expectedSystemTimes[r, 2*i] <- 0
        }
      }
    }
    else{
      for(i in 1:N){
        if(types[(r+((a-1)*R)),i]!=1 & expectedSystemTimes[r-1, 2*i
            -1]!=0){
          expectedSystemTimes[r, 2*i-1] <- ((r-1)*
              expectedSystemTimes[r-1, 2*i-1] + systemTimes[i,1])/r
          expectedSystemTimes[r, 2*i] <- ((r-1)*expectedSystemTimes[r
              -1, 2*i] + systemTimes[i,2])/r
        }
        else if(types[(r+((a-1)*R)),i]!=1){
          expectedSystemTimes[r, 2*i-1] <- systemTimes[i,1]
          expectedSystemTimes[r, 2*i] <- systemTimes[i,2]
        }
        else{
          expectedSystemTimes[r, 2*i-1] <- expectedSystemTimes[r-1,
              2*i-1]
          expectedSystemTimes[r, 2*i] <- expectedSystemTimes[r-1, 2*
              i]
        }
      }
    }
  }
  #Check how many times strategy 1 is the best strategy
  for (i in 1:N){
    if (expectedSystemTimes[R, 2*i-1] <= expectedSystemTimes[R, 2*i
        ]){
      results[a, m + (p-1)*M + (g-1)*P*M] <- results[a, m + (p-1)*M
          + (g-1)*P*M]+1
          }
        }
      }
    }
  }
}

count <- matrix(0, nrow = N, ncol = G*P*M)
gemiddeld <- matrix(NA, nrow = G*P*M, ncol = 1)

for (j in 1:(G*P*M)){
  for (i in 1:A){
    count[results[i,j],j] <- count[results[i,j],j] + 1
  }
}
for (j in 1:(G*P*M)){
  teller <- 0
  for (i in 1:N){
    teller <- teller + count[i,j]*i
  }
  gemiddeld[j] <- teller/A
}

#Check how long the program has run
endTime <- Sys.time()
timeTaken <- endTime - startTime
beep(sound=3, expr=NULL)
```