Danmarks Tekniske Universitet

42586 - Decisions under uncertainty

Project: Health Care Reforms

Aksel Buur Christensen - s203947

26. marts 2023

Indhold

1 Decision-making with and without probabilities

Contributions for the assignment:
The first section in the assignment is primarily done by David Holsko whom I was in group with until short before deadline, the remaining of the assignment is done by myself.

Table provided for the different hospitals with travel time and number of visits required:

Hospital	Travel time (minutes)			
	Morning-peak	Off-peak	Afternoon-peak	Nr. visits
North Zealand Hospital	60	30	75	4
Lyngby (check up)	10	10	10	6
Herlev Hospital	40	20	35	4
Gentofte (check up)	20	20	20	6

Tabel 1: *Provided table from the project description for the different hospitals and their respective travel times. Appointment times are assigned stochastically.*

1.a The system, environment, decision-maker, the decision, and outcome

The system in this case is the Hospitals, distances to these hospitals and number of visits that needs to be made to the check-ups and hospitals. Now the environment is the travel time and whether it's at a peak or off-peak.
The decision maker is the pregnant patient and the decision has a cost in hours spent choosing either North Zealand Hospital or Herlev Hospital and once the decision has been made it can't be changed. Lastly the outcome is the time spent by the decision makers decision in terms of cost in minutes.

1.b Maximax, Minimax, Minimax regret

For making a decision on the basis of limited information and not knowing the likely-hood of each outcome in the system the Maximax, Minimax and Minimax regret methods can be used to determine the course of action
Below is a table of the amount of minutes it will take to reach all appointments in the Morning and Afternoon peaks, as well as the Off-peak.

Hospital	Morning-peak [min.]	Off-peak [min.]	Afternoon-peak [min.]
North Zealand	600	360	720
Herlev	560	400	520

The Maximax method is based on finding the output that yields highest output no matter of the risks, so in our case this means that the option where the possibilty of the lowest transport time will be chosen, not considering if one might hit the morning or afternoon peak. Therefore in this case using the maximax strategy the choice falls upon the North Zealand Hospital as the minimum time that can be spent is here is 360 min. which is lower than any possibility at Herlev.

The Minimax method is used to find the choice that yields the highest minimum output that is possible. This way the decision-maker is avoiding the lowest yields and playing it a bit more safe, risk-averse and a bit pessemistic. If this minimax method was used, then the choice would land on Herlev Hospital as the highest amount of time that can be used is 560 min, whereas with

the North Zealand Hospital there is a risk of it taking up to 720 min.

The Minimax Regret method is based on finding the output that yields the lowest amount of regret of choice, which is based on the difference between the values for North Zealand and Herlev. For this a second table is made outlining the regret.

Hospital	Morning-peak [min.] Regret	Off-peak [min.] Regret	Afternoon-peak [min.] Regret
North Zealand	40	0	200
Herlev	0	40	0

In the table it can be seen that the maximum possible regret form choosing North Zealand Hospital amounts to 200 min., whereas this only amounts to 40 min. for Herlev Hospital. Therefor using this method the decision-maker would choose Herlev Hospital.

1.c Probability of appointment falling at specific time

To find the probability of an appointment being at a specific time in relation to one another for the two options, North Zealand Hospital and Herlev Hospital, we are given the criterias that the Estimated Costs (EC) for both hospitals are the same, as well as the probability for an appointment in the afternoon-peak is 10% for both Hospitals.
We now can write the EC up for each hospital

$$\begin{aligned} EC(NZ) &= 600 \cdot \frac{p_1(x)}{100} + 360 \cdot \frac{p_2(x)}{100} + 72 \\ EC(HH) &= 560 \cdot \frac{p_3(x)}{100} + 400 \cdot \frac{p_4(x)}{100} + 52 \end{aligned} \tag{1.1}$$

We can now set values for probabilities for going to North Zealand hospital at the morning-peak and off-peak. We set these to 45% each as 10% has already been applied to the afternoon-peak. Therfore we get the total Estimated cost of going to the North Zealand Hospital to:

$$EC(NZ) = 600 \cdot \frac{45}{100} + 360 \cdot \frac{45}{100} + 72 = 504 \tag{1.2}$$

Since we know that:

$$\begin{aligned} EC(NZ) = EC(HH) &= 504 \\ p_3(x) + p_4(x) &= 90 \end{aligned} \tag{1.3}$$

Then we have 2 equations with 2 unknowns which we can solve, thus giving us the following probabilities:

Hospital	Morning-peak [prob.]	Off-peak [prob.]	Afternoon-peak [prob.]
North Zealand	45%	45%	10%
Herlev	57.5%	32.5%	10%

These probabilites could be changed in ever so many ways where the expected costs of North Zealand and Herlev Hospital are equal.

1.d Relection 1

If the decision maker was now able to better choose the time of the check-ups to suit the decision maker, then in theory we would have to differentiate between the probabilities of going to North Zealand Hospital and Lyngby, and Herlev Hospital and Gentofte Hospital. The probability for Lyngby and Gentofte Hospital would now be dependant on the probability for the decision-maker to get the time that the decision maker wants from the locations. Since the travel times to both Lyngby and Gentofte does not change even though the decision maker could choose between different time slots.

Hospital	Morning-peak [min.]	Off-peak [min.]	Afternoon-peak [min.]
Lyngby	10	10	10
Gentofte	20	20	20

1.e Reflection 2

By comparing the full possible time spent with the other combinations for birth centers and check-ups as seen below:

Hospital	Morning-peak [min.]	Off-peak [min.]	Afternoon-peak [min.]
North Zealand	600	360	720
Herlev	560	400	520
Gentofte	400	400	400

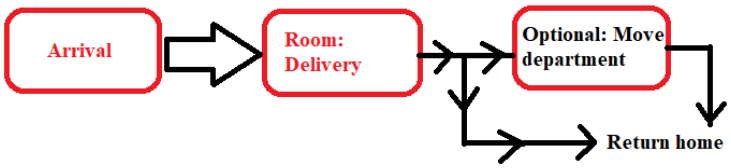
Then we can clearly see that the North Zealand Hospital would still be the choice if the decision maker based her choice on the **maximax method**. But if the **minimax** and the **minimax regret methods** were used, then the choice would land on Gentofte Hospital as there in the minimax method is the lowest maximum amount of time that can be used out of the three (400 min.), and in the minimax regret method it has the lowest amount of regret on 40 min as seen in the table below:

Hospital	Morning-peak [min.] Regret	Off-peak [min.] Regret	Afternoon-peak [min.] Regret
North Zealand	200	0	320
Herlev	160	40	120
Gentofte	0	40	0

So therefor the only way to not choose Gentofte Hospital as the prefered birth center would be if the Maximax method was used.

2 Markov Theory

The Markov property is a type of stochastic process where the probability evolve over time. The Markov property is a stochastic process that only depends on the present state[1]. This obviously means that Markov chains do not depend on past states/actions and to analyze whether the described (and visualised) process is a Markov chain or not, the first state is considered: "Arrival"(see fig. 1). As stated in the introduction of this section, the arrival of people in labour is stochastic. This is a Markov property, moreover that these arrivals are independent of each other. After the people in labour has arrived they will be in a waiting position until a room is available for delivery. This means that the people in labour can be considered in a waiting state until they will move on to the next state: To have a room for delivery. In delivery the people in labour will stay for the amount of time periods required to finish delivery. After the delivery is completed, the patient can either return home and thereby exiting the system or move department in the



Figur 1: *The process visualised*

maternity ward. To sum up, this satisfies a Markov property since this is a stochastic process and the process does not dependent on past states/events.

2.a Single hospital

As scheduled in fig 1 and described above, it is possible to split the process in to an arrival which is stochastic, where the patients entering the system and thereby the queue. Since a single hospital with multiple rooms and the same arrival rate is considered, it is possible to introduce servers into this queue. This can be done since the patients can be served from multiple rooms in the hospital and this can be used as a property in the single queue instead of having the same amount of queues as rooms.
The next part the process can be split into is after the arrival, a room is assigned to the patient meaning that now the service time start (this might not be true in practice if a delivery begins in the queue for rooms, but this is ignored). The queue and the queue length in this context should be taken into consideration from a multiple server perspective. This means that the queue both consists of the number of rooms (servers) and the patients waiting in the queue for these rooms. As long as the amount of persons in the queue is below the amount of rooms available, there will be no queue. The queue length is therefore only formed after an arrival has occurred while all rooms are occupied. The states are: Arrival, room for delivery and leaving the system, where the transition between arrival and a room for delivery is determined by; if there are any vacant rooms or when a room will become vacant. After the delivery is complete the only transition that will be used in the following is the leaving of the maternity ward.

2.b Determine the statistics

The following calculations will be done based on Hillier & Liebermann: Introduction to Operations Research, chapter 17 where multiple server queues is treated.

[1]Hillier & Liebermann: Introduction to Operations Research chapter 16

2.b.1 5 centers

Defining the variables:

$$\begin{aligned}
\lambda &= 0.42 \\
\mu &= 0.08 \\
s &= 8 \\
\rho &= \frac{\lambda}{(s\mu} = 0.65625 \\
P_0 &= \frac{1}{\sum_{n=0}^{s-1} + \frac{\left(\frac{\lambda}{\mu}\right)^s}{s!}\frac{1}{1-\rho}} = 0.00496 \\
L_q &= \frac{P_0\left(\frac{\lambda}{\mu}\right)^s \rho}{s!(1-\rho)^2} = 0.394 \\
W_q &= \frac{L_q}{\lambda} = 56\,\text{min}
\end{aligned}$$

The percentage of time there is one patient in the queue is the time when all rooms are occupied and a patient arrives, forming a queue. To check the probability that at least one patient is in the queue is defined as:

$$p_{1patient} = 1 - \sum_{i=0}^{s} \mathbf{P} \tag{2.1}$$

Where $\mathbf{P}$ is the vector containing all the probabilities that a different amount of patients is in the queue/system. In the same way, the probability that 2 or more people are in the queue will be:

$$p_{2patient} = 1 - \sum_{i=0}^{s+1} \mathbf{P} \tag{2.2}$$

Which can be calculated to be:

$$\begin{aligned}
p_{1patient} &= 0.1411 \approx 14.1\% \\
p_{2patient} &= 0.0991 \approx 9.9\%
\end{aligned}$$

2.b.2 4 centers

$$\begin{aligned}
\lambda &= 0.525 \\
\mu &= 0.08 \\
s &= 10 \\
\rho &= \frac{\lambda}{(s\mu} = 0.65625 \\
P_0 &= \frac{1}{\sum_{n=0}^{s-1} + \frac{\left(\frac{\lambda}{\mu}\right)^s}{s!}\frac{1}{1-\rho}} = 0.00136 \\
L_q &= \frac{P_0\left(\frac{\lambda}{\mu}\right)^s \rho}{s!(1-\rho)^2} = 0.308 \\
W_q &= \frac{L_q}{\lambda} = 35\,\text{min}
\end{aligned}$$

The percentage of time there is one patient in the queue is the time when all rooms are occupied and a patient arrives, forming a queue. To check the probability that at least one patient is in the queue is defined as:

$$p_{1patient} = 1 - \sum_{i=0}^{s} \mathbf{P} \tag{2.3}$$

Where $\mathbf{P}$ is the vector containing all the probabilities that a different amount of patients is in the queue/system. In the same way, the probability that 2 or more people are in the queue will be:

$$p_{2patient} = 1 - \sum_{i=0}^{s+1} \mathbf{P} \tag{2.4}$$

Which can be calculated to be:

$$p_{1patient} = 0.1072 \approx 10.7\%$$
$$p_{2patient} = 0.07414 \approx 7.4\%$$

The julia-code used to obtain all the above answers can be found in appendix.

2.c How many centers?

From the calculated statistics it is clear that it is favored to have 4 centers with more rooms, but there are obvious reasons why this is not favorable in practice. Does a part of the population gain longer travel times to the fewer, but bigger centers. The average waiting time is only 20 minutes different, so their might actually be patients having longer time in the system if transportation would be considered as well. The obtained results matches what is seen in the modern society: New giant ("supersygehuse") hospitals are build making it possible to close many smaller hospitals around in the countryside.

2.d More rooms

The target is to have the amount of time 2 people or more are waiting are less than 5%. To make sure that this target is being reached a criteria can be written:

$$p_{2patients} \leq 0.05 \tag{2.5}$$

Since both centers had quite low probabilities for having 2 or more patients in the queue, it is tested whether an extra server makes the amount of time 2 people or more are waiting in the queue below 5%. The calculations are the same as above, but for an extra server. This gives the following probabilities:

$$\begin{aligned} \text{5 centers - 9 servers} &= \text{45 rooms} \\ p_{1patient} &= 0.0653 \approx 6.5\% \\ p_{2patient} &= 0.0428 \approx 4.3\% \\ \text{4 centers - 11 servers} &= \text{44 rooms} \\ p_{1patient} &= 0.0515 \approx 5.2\% \\ p_{2patient} &= 0.0330 \approx 3.3\% \end{aligned}$$

Where it is also seen that the 4 centers have the best probabilities to make sure that 95% of the time there wont be more than 2 people in the queue.

3 Simulation

3.a Monte-Carlo

To determine how much food should be ordered to the new mothers and fathers a Monte-Carlo simulation is used. The information given is used, and a fixed number meals is chosen to make the first test runs. The chosen amount of meals is 40 meals, since it is the average in the normal distribution. This number of meals should eventually be optimized, but for now it is used to test out the function and calculate confidence intervals.
The Monte-Carlo function is written below in pseudo code, but see appendix for the used code.

Algorithm 1 Monte Carlo - basic

```
function MEALS(NO_MEALS)
    days ←100
    mu ←40
    sigma ←15
    c_left ←30
    c_shortage ←100
    total_cost ←Vector{Float64}()
    for i ←1,days do
        drawi ←rand(Normal(mu,sigma),1)
        if drawi < no_meals then
            short_meal ←no_meals - drawi
            cost_short ←short_meals· c_shortage
            total_cost(i)←cost_short
        else if drawi > no_meals then
            left_meal ←drawi - no_meals
            cost_left ←left_meals· c_left
            total_cost(i)←cost_left
        else
            total_cost(i) ←0
        end if
    end for
end function
```

With this function it is now possible to estimate the number of runs needed to obtain the 95% confidence interval for the average cost of a meal within in ± 5 DKK. This means that the width of the confidence interval should be 10 DKK. The confidence interval :[2]

$$\mathrm{CI} = \left[\bar{x} - \frac{qs}{\sqrt{n}}, \bar{x} + \frac{qs}{\sqrt{n}}\right] \tag{3.1}$$

Where $\bar{x}$ is the mean of the sample here the total cost, s is the standard deviation of the sample, here the total cost and n is the number of observations in the sample. At last, q denotes the 95% quantile from the normal distribution which will be used here since the standard deviation for the sample is known.
It is although seen that when the number of meals is fixed and the function is runned for 100 days, the interval wished is seen to be obtained. This can be a consequence of the choice to have the fixed number of meals to be 40 which is the mean of the distribution. Now the function will be optimized and a loop will be introduced to determine the amount of meals to be ordered every day.

[2] `https://en.wikipedia.org/wiki/Confidence_interval`

3.a.1 Optimize Monte Carlo

This is implemented in the "simulmeals(Nrep)"function that try out the number of meals ordered each day from 1 to 100 where each number of ordered meals is tested for 100 days. It is determined that the optimal amount of meals ordered each day from a cost perspective is 68, this will obviously vary in a Monte Carlo simulation, but the average cost per meal in this case is 13 DKK pr meal. It makes sense that the optimal number of meals is quite higher than the average required amount of meals since it is much cheaper to have leftover than shortages.
In the optimizing only positive number of meals is drawn from the normal distribution which is done by the truncated normal distribution in julia. Moreover it is implemented that the cost for shortages differ for the amount of meals the hospital are short from the required. This probably makes the optimal number of meals to be a little smaller than the number of meals found before since it is cheaper to be below 10 meals short. At last another hospital is introduced with a normal distributed demand with a mean of 80 and a standard deviation of 30. It has the same orignial shortage/leftover costs.
When this additional hospital is introduced it is seen that the minimum costs for the first and second hospital is about the same for the most runs, but when the confidence interval for the two hospitals is compared it is seen that for the first hospital it is possible to get a confidence interval with a width of around ± 10 DKK, but for the second hospital the confidence interval is around ± 50 DKK wide making it a much more uncertain cost for the meals. On the basis of the almost same cost pr meal for the two hospitals and the fact that the second hospital has a much more uncertain cost it will not be advised to merge the two. The results for one of the runs can be seen in results_monte.txt file.

3.b Discrete Event Simulation

I have unfortunately not been able to succesfully implement the discrete event simulation, but below is the used pseducode for the attempt.
We are trying to set up the simulation through the following algorithms.
States:

- x = length of urgent queue
- y = length of normal queue
- z = length of low queue

Events:

- AU = arrival of urgent patient
- AN = arrival of normal patient
- AL = arrival of low patient
- DH = Departure from hospital (do we need three?)

Algorithm 2 Main function

$t \leftarrow 0, x \leftarrow 0, y \leftarrow 0, z \leftarrow 0, T \leftarrow 1008$
Call Schedule Arrival
while *true* **do**
Get the first event
Let t be the time of this event
    if AU **then**
Call ArrivalUrgent
    else if AN **then**
Call ArrivalNormal
    else if AL **then**
Call ArrivalLow
    else if DH **then**
Call DepartureHospital
    end if
end while

3.c Patient arrival events

Algorithm 3 ArrivalUrgent

if $x = 0$ **then**
Schedule DH at $t + \exp\{12.5\}$
end if
$x \leftarrow x + 1$
Call ScheduleArrival

Algorithm 4 ArrivalNormal

if $x = 0$ **then**
    if $y = 0$ **then**
Schedule DH at $t + \exp\{12.5\}$
    else
        $y \leftarrow y + 1$
    end if
end if
$y \leftarrow y + 1$
Call ScheduleArrival

Algorithm 5 ArrivalLow

if $x = 0$ **then**
  if $y = 0$ **then**
    if $z = 0$ **then**
Schedule DH at $t + \exp\{12.5\}$
    else
      $z \leftarrow z + 1$
    end if
  else
    $z \leftarrow z + 1$
  end if
end if
$z \leftarrow z + 1$
Call ScheduleArrival

4 The News Vendor Problem

This is part 4

5 Stochastic Optimization

This is part 5

6 Conclusion

This is the conclusion

7 Appendix

7.a Markov script

```
## PROJECT: MARKOV ##

# 3 a)
# 5 centers.
# Arrival rate of 0.42 pr hour
# Average room occupation time for 12.5 hours = 0.08 patient/hour
# 8 rooms = 8 servers

# Solution a)
# Arrival at 5 centers
lambda = 0.42 # Arrival rate
mu = 0.08 # service rate
s = 8 # Number of servers
rho = 0.42/(8*0.08) # Occupation rate: 0.65625
summ = 0 # Calculating the sum used to determine p0:
for i in 0:(s-1)
    summ += ((lambda/mu)^i)/factorial(i)
end
# The probability that no one is in the rooms or in the queue:
p0 = 1/( summ + (((lambda/mu)^s)/factorial(s)) * (1/(1-(lambda/(s*mu)))))
```

```
# Lq, number of customers in the queue
lq = (p0*((lambda/mu)^s)*rho)/(factorial(s)*((1-rho)^2))

wq = lq/lambda # Waiting time in queue: 56.3 min = 56 min

# Number of patients in queue where from p1 to p8 is the probability that the 8 rooms
are occupied.
# To determine the probability that 1 person is waiting for instance, p9 need to be
calculated.
n = range(1,8)
p_occupied = Vector{Float64}()
for j in n
    p = (((lambda/mu)^j)/factorial(j))*p0
    push!(p_occupied,p)
end
#println("The probability that all rooms are full: ",1-sum(p_occupied))
p9 = (((lambda/mu)^9)/factorial(9))*p0
println("(5 center) The probability that at least one person is waiting for a room
is: ",1-(sum(p_occupied)))
println("(5 center) The probability that at least two persons is waiting for a room
is: ",1-(sum(p_occupied)+p9))

# 3 b)
# 4 centers
# Arrival rate of 0.525 pr hour
# Average room occupation time for 12.5 hours = 0.08 patient/hour
# 10 rooms = 10 servers
lambda1 = 0.525 # Arrival rate
mu = 0.08 # service rate
s1 = 10 # Number of servers
rho1 = lambda1/(s1*mu) # Occupation rate: 0.65625
summ1 = 0 # Calculating the sum used to determine p0:
for i in 0:(s1-1)
    summ1 += ((lambda1/mu)^i)/factorial(i)
end
# The probability that no one is in the rooms or in the queue:
p01 = 1/( summ1 + (((lambda1/mu)^s1)/factorial(s1)) * (1/(1-(lambda1/(s1*mu)))))

# Lq, number of customers in the queue
lq1 = (p01*((lambda1/mu)^s1)*rho1)/(factorial(s1)*((1-rho1)^2))

wq1 = lq1/lambda1 # Waiting time in queue: 35 minuttes

# Number of patients in queue where from p1 to p8 is the probability that the 8 rooms
are occupied.
# To determine the probability that 1 person is waiting for instance, p9 need to be
calculated.
n1 = range(1,10)
p_occupied1 = Vector{Float64}()
for j in n1
    p1 = (((lambda1/mu)^j)/factorial(j))*p01
    push!(p_occupied1,p1)
end
```

```
#println(p_occupied1)
#println("The probability that all rooms are full: ",1-sum(p_occupied))
p11 = (((lambda1/mu)^11)/factorial(11))*p01
println("(4 center) The probability that at least one person is waiting for a room
is: ",1-(sum(p_occupied1)))
println("(4 center) The probability that at least two persons is waiting for a room
is: ",1-(sum(p_occupied1)+p11))




# 3.5
# For 5 centers we have probability for 2 or more people waiting 0.09910387095483297
= 9.9%.
# Determining how much probability that needs to be "cut off" to reach 5% or less:
s = 9
summ = 0 # Calculating the sum used to determine p0:
for i in 0:(s-1)
    summ += ((lambda/mu)^i)/factorial(i)
end
p0 = 1/( summ + (((lambda/mu)^s)/factorial(s)) * (1/(1-(lambda/(s*mu)))))
n = range(1,9)
p_occupied = Vector{Float64}()
for j in n
    p = (((lambda/mu)^j)/factorial(j))*p0
    push!(p_occupied,p)
end

p10 = (((lambda/mu)^10)/factorial(10))*p0
println("(5 center) The probability that at least one person is waiting for a room
is: ",1-(sum(p_occupied)))
println("(5 center) The probability that at least two persons is waiting for a room
is: ",1-(sum(p_occupied)+p10))
# With 9 servers = 9 rooms*5centers = 45 rooms the probability that 2 or more people
are in the queue is
# p = 0.04275151052802617

# For 4 centers we have the probability for 2 or more people waiting 0.0741475302248843
= 7.4 %.
s1 = 11
summ1 = 0 # Calculating the sum used to determine p0:
for i in 0:(s1-1)
    summ1 += ((lambda1/mu)^i)/factorial(i)
end
# p0:
p01 = 1/( summ1 + (((lambda1/mu)^s1)/factorial(s1)) * (1/(1-(lambda1/(s1*mu)))))

# Number of patients in queue where from p1 to p8 is the probability that the 8 rooms
are occupied.
# To determine the probability that 1 person is waiting for instance, p9 need to be
calculated.
n1 = range(1,11)
p_occupied1 = Vector{Float64}()
for j in n1
```

```
    p1 = (((lambda1/mu)^j)/factorial(j))*p01
    push!(p_occupied1,p1)
end
#println(p_occupied1)
#println("The probability that all rooms are full: ",1-sum(p_occupied))
p11 = (((lambda1/mu)^12)/factorial(12))*p01
println("(4 center) The probability that at least one person is waiting for a room
is: ",1-(sum(p_occupied1)))
println("(4 center) The probability that at least two persons is waiting for a room
is: ",1-(sum(p_occupied1)+p11))
# With 11 servers = 11 rooms*4centers = 44 rooms the probability that 2 or more people
are in the queue is
# p = 0.03298138934868655
# AND with 1 person in the queue almost 5 %:
# p = 0.05151752790626385
```

7.b Monte Carlo script

```
    ## PROJECT: 3. SIMULATION ##

# 1, 2, 3, 4: Monte Carlo simulation
# Normal distributed with mean 40 and standard deviation of 15
# Cost of leftovers 30 DKK and cost of shortage 100 DKK
# Objective: Create a function that can evaluate the costs of a fixed number of meals
# ordered every day for 100 days
using Distributions
using HypothesisTests
function meals(no_meals)
    #no_meals = 40
    days = 100
    mu = 40
    sigma = 15
    c_left = 30
    c_shortage = 100
    total_cost_1 = Vector{Float64}()
    #print(total_cost)

    for i in 1:days
        #println("no_meals = ",no_meals)
        normal= Normal(mu,sigma)
        td = truncated(normal,0,Inf)
        drawi = rand(td,1)
        drawi = round.(drawi,digits=0)
        # truncated distribution to get positive values

        #println("draw: ",drawi)
        if drawi[1] > no_meals
            short_meal = drawi[1] - no_meals
            if short_meal < 10              #Comment this if-statement out for 3.1.2
                cost_short = 90*short_meal
            elseif short_meal < 25
                cost_short = 110*short_meal
            else
                cost_short = rand(Normal(150,20),1)*short_meal
```

```
            end
            # cost_short = c_shortage*no_meals #Uncomment this for 3.1.2
            push!(total_cost_1,cost_short[1])
        elseif drawi[1] < no_meals
            left_meal = no_meals - drawi[1]
            cost_left = left_meal*c_left
            push!(total_cost_1,cost_left)
            #println(total_cost)
            #no_meals -= 2
        else
            push!(total_cost_1,0)
        end
    end
    #avegare_meal = total_cost ./ no_meals
    #conf = confint(OneSampleTTest(avegare_meal), level=0.95)
    #println("Conf interval: ", conf, "with mean: ",(mean(total_cost))./no_meals)
    #println("The total cost for 100 days is: ",total_cost)
    #println("The cost per day per meal for in total $no_meals meals is: ",)
    return total_cost_1
end

# Meals function for the second hospital
function meals2(no_meals)
    #no_meals = 40
    days = 100
    mu = 80
    sigma = 30
    c_left = 30
    c_shortage = 100
    total_cost_2 = Vector{Float64}()
    #print(total_cost)

    for i in 1:days
        #println("no_meals = ",no_meals)
        normal= Normal(mu,sigma)
        td = truncated(normal,0,Inf)
        drawi = rand(td,1)
        drawi = round.(drawi,digits=0)
        # truncated distribution to get positive values

        #println("draw: ",drawi)
        if drawi[1] > no_meals
            short_meal = drawi[1] - no_meals
            cost_short = short_meal*c_shortage
            push!(total_cost_2,cost_short)
        elseif drawi[1] < no_meals
            left_meal = no_meals - drawi[1]
            cost_left = left_meal*c_left
            push!(total_cost_2,cost_left)
            #println(total_cost)
            #no_meals -= 2
        else
            push!(total_cost_2,0)
        end
```

```
    end
    #avegare_meal = total_cost ./ no_meals
    #conf = confint(OneSampleTTest(avegare_meal), level=0.95)
    #println("Conf interval: ", conf, "with mean: ",(mean(total_cost))./no_meals)
    #println("The total cost for 100 days is: ",total_cost)
    #println("The cost per day per meal for in total $no_meals meals is: ",)
    return total_cost_2
end

function simulmeals(Nrep)
    average_meal_list = zeros((8,Nrep))
    no_meals = 1
    quant = quantile(Normal(0.0, 1.0),1-(1-0.95)/2)
    for i in 1:Nrep
        #First/original hospital loop, fill in cost pr meal with confidence intervals
        total_cost_1 = meals(no_meals)
        amount = (mean(total_cost_1))/no_meals
        s = std(total_cost_1)
        lowconf = amount - (quant*s)/100 # n = 100 for 100 days simulated
        lowconf = round.(lowconf,digits=1)
        upconf = amount + (quant*s)/100
        upconf = round.(upconf,digits=1)
        amount = round.(amount,digits=1)
        average_meal_list[1,i] = no_meals
        average_meal_list[3,i] = amount
        average_meal_list[2,i] = lowconf
        average_meal_list[4,i] = upconf

        # Second hospital
        total_cost_2 = meals2(no_meals)
        cost_meal = (mean(total_cost_2))/no_meals
        s2 = std(total_cost_2)
        lowconf2 = cost_meal - (quant*s2)/100
        lowconf2 = round.(lowconf2,digits=1)
        upconf2 = cost_meal + (quant*s2)/100
        upconf2 = round.(upconf2,digits=1)
        cost_meal = round.(cost_meal,digits=1)
        average_meal_list[5,i] = no_meals
        average_meal_list[6,i] = lowconf2
        average_meal_list[7,i] = cost_meal
        average_meal_list[8,i] = upconf2

        no_meals += 1
    end
    min_meals1 = findmin(average_meal_list[3,:])
    max_meals1 = findmax(average_meal_list[3,:])
    min_meals2 = findmin(average_meal_list[7,:])
    max_meals2 = findmax(average_meal_list[7,:])
    #println("Meal price list: ", average_meal_list)
    println("Minimum cost pr meal is with the index - hosp 1: ",min_meals1)
    println("Maximum cost pr meal is with the index - hosp 1: ",max_meals1)
    println("Minimum cost pr meal is with the index - hosp 2: ",min_meals2)
    println("Maximum cost pr meal is with the index - hosp 2: ",max_meals2)
    # if Nrep > 1
```

```
    #     conf = confint(OneSampleTTest(average_meal_list),level=0.95)
    #     println("Conf interval: ",conf)
    # else
    #     println("Confidence can't be determined from 1 rep!")
    # end
    return average_meal_list
end
average_meal_list = simulmeals(200)

resultsfile = "results_monte.txt"
open(resultsfile,"w") do f
    for j in average_meal_list
        println(f,j)
    end
end
```

7.c Hospital simulation

```
    #module Hospital
using Random, Distributions, DataStructures, Plots
#You can use print-lines for debugging. When your code is done, you would like to
be able to switch them off easily.
#Therefore, one can include a boolean "_debug" that is switched on when debugging,
and off otherwise.
_debug = true

#############################################################################################
#Environment describes all your important settings
# timeHorizonH: the amount of simulated time in hours
# warmUpTime: the warm-up period time (out of total time)
# base: number of hours per period, e.g. 24 if time is registered in hours, and a
day lasts 24 hours
# arrivalRate_patient: the arrival rate per hour of patients
# probDistrPatientType:
# serviceTime_patient: a matrix [nr of patient types]  x 1 storing the service time
(time room is occupied)
# nrRooms: number of available rooms for delivery
# costWaitingPerHour::Matrix{Float64}
struct Environment
    timeHorizonH::Float64
    warmUpTime::Int64
    base::Int64
    arrivalRate_patient::Float64
    probDistrPatientType:: Array{Float64}
    serviceTime_patient::Float64
    nrRooms::Int64
    costWaitingPerHour::Matrix{Float64}
end


#DO NOT CHANGE: an event  has a name, starting with ":", and a time.
#You can create one by calling e.g: myEvent = (:myEventTypeName, myTimeasAFloat)
struct Event
    type::Symbol
```

```
    time::Float64
end


struct Patient
    type::Symbol
    time::Float64 # of arrival
    #HospitalID::Symbol
end


mutable struct SimStats
    n_served::Int64 #total, and per patient type, so nrPatientTypes+1 long
    waitNormal::Array{Float64}
    waitLow::Array{Float64}
    waitHigh::Array{Float64} #per patient type waiting time + time service start, so
nrPatientTypes vectors, and nr patients p patient type long
    #Store state over time: time, ocupiedRooms, Queuelengths
    overview_usage::Array{Float64,2}
    costsOfWaiting::Float64
end

#Describes the state of your process, similar to states in Markov. You can include
as many variables as you like/need
mutable struct State #for multiple hospitals, change below to matrices
    ocupiedRooms::Int64
    totalRooms::Int64

    # IDs?
    # localQueue?
    events::Vector{PriorityQueue{Patient,Float64}}  #Queue per priority level, priority
level going down
    t::Float64
    #id::Int64

    # Events here?

    #State() = new(0, 7,Vector{PriorityQueue{Patient,Float64}}(),0) # ID ?

end

function schedule_arrival(env::Environment,state::State)
    tt = state.t + rand(Exponential(env.arrivalRate_patient))
    if tt <= env.warmUpTime + env.timeHorizonH
        probT = rand()
        if probT <= 0.6
            patient = Patient(:Normal,tt)
        elseif probT <= 0.9
            patient = Patient(:Low,tt)
        else
            patient = Patient(:High,tt)
        end
        e = Event(:ArrivalPatient)
    end
```

```
end



#TODO create this function
function arrival_patient(patient::Patient, env::Environment, state::State, events::PriorityQueue{
stats::SimStats, t::Float64)
    # MISSING:
    # Use of environment - use it for nr.rooms?
    # Use of Events - enqueue into events instead of the localQueue?
    # patientID = state.id
    # Check if time is over the time horizon.


    println("ArrivalPatient has started at time: $(state.t)")

    tt = state.t + rand(Exponential(env.arrivalRate_patient))

    # Generate next arrival

    if patient.type ==:High
        #push!(stats.waitHigh,state.t) # ?!
        if isempty(state.localQueue[1])
            println("No high patients in queue")
            if state.totalRooms > state.ocupiedRooms
                println("Available rooms")
                e = Event(:serviceStart,tt)
            else
                enqueue!(localQueue[1],patient,tt)
                # stats.overview_usage[:,1] = tt
                # stats.overview_usage[:,3] += 1
                # CORRECT ?
            end
        else
            # stats.overview_usage[:,1] = tt
            # stats.overview_usage[:,3] += 1
            # Queuetimes?

        end

    elseif patient.type ==:Normal
        if isempty(state.localQueue[1])
            println("No high patients in queue")
            if isempty(state.localQueue[2])
                println("No normal patients in queue")
                if state.totalRooms > state.ocupiedRooms
                    println("Available rooms")
                    e = Event(:serviceStart,tt)
                else
                    enqueue!(localQueue[2],patient,tt)
                    # stats.overview_usage[:,1] = tt
                    # stats.overview_usage[:,4] += 1
                    # CORRECT ?
                end
```

```
            else
                enqueue!(state.localQueue[2],patient,tt)
                # stats.overview_usage[:,1] = t
                # stats.overview_usage[:,4] += 1
                # CORRECT ?
            end
            enqueue!(state.localQueue[2],patient,tt)
            # stats.overview_usage[:,1] = t
            # stats.overview_usage[:,4] += 1
            # CORRECT ?
        end

    else
        if isempty(state.localQueue[1])
            println("No high patients in queue")
            if isempty(state.localQueue[2])
                println("No normal patients in queue")
                if isempty(state.localQueue[3])
                    println("No low patients in queue")
                    if state.totalRooms > state.ocupiedRooms
                        println("Available rooms")
                        e = Event(:serviceStart,tt)
                    else
                        enqueue!(localQueue[3],patient,tt)
                        # stats.overview_usage[:,1] = t
                        # stats.overview_usage[:,5] += 1
                        # CORRECT ?
                    end
                else
                    enqueue!(localQueue[3],patient,tt)
                     # stats.overview_usage[:,1] = t
                    # stats.overview_usage[:,5] += 1
                    # CORRECT ?
                end
            else
                enqueue!(localQueue[3],patient,tt)
                # stats.overview_usage[:,1] = tt
                # stats.overview_usage[:,5] += 1
                # CORRECT ?
            end
        else
            enqueue!(localQueue[3],patient,tt)
            # stats.overview_usage[:,1] = t
            # stats.overview_usage[:,5] += 1
            # CORRECT ?
        end
    end
    # New arrival ?
end



#TO DO create this function
function service_complete(env::Environment, state::State, stats::SimStats, events::PriorityQueue{
```

```
t::Float64)
    # Check if there any patients to put in empty rooms
end

#Both after an arrival, or a service completion, a new service can start (someone
is admitted to the room).
#You can create this function for both these cases -- and call it at appropriate places
in your code
function serviceStart(env::Environment, state::State, events::PriorityQueue{Event,Float64},
stats::SimStats,patient::Patient,  t::Float64)

end



function simulate(env::Environment, id::Int64)
    #Initialize the state of our process:
    localQueue = PriorityQueue{Patient,Float64}[] #Vector{PriorityQueue{Patient,Float64}}(Priorit
    for i= 1:length(env.probDistrPatientType)
        push!(localQueue, PriorityQueue{Patient,Float64}())
    end
    state = State(0, 7,localQueue,0)

    #Create an empty statistics field (n_served, waitingTimeHigh[],waitingTimeNormal[],
waitingTimeLow[], overview_usage, costsOfWaiting )
    #overview usage: [current time, nrocupiedRooms, QueueLengthHigh,QueueLengthNormal,QueueLength
    overview_usage = [0 0 0 0 0]
    waitHigh=Float64[]
    waitNormal=Float64[]
    waitLow=Float64[]
    costsOfWaiting = 0.0
    stats = SimStats(0,waitHigh, waitNormal, waitLow, overview_usage, costsOfWaiting)

    #Initialize first event :
    t=0.0;
    events = PriorityQueue{Event,Float64}()
    nextArrival = t +rand(Exponential(env.arrivalRate_patient)) # TODO?
    e1 = Event(:ArrivalPatient,nextArrival)
    enqueue!(events,e1,e1.time)
    #schedule_arrival(env,state)

    warmUp = true;

    while !isempty(events)
        e = dequeue!(events)
        t = e.time

        if t>= env.warmUpTime + env.timeHorizonH
            agregatedStats = printResults(stats, env, id)
            return agregatedStats
        end

        if warmUp && (t >= env.warmUpTime)
            warmUp = false;
```

```
            waitHigh=Float64[]
            waitNormal=Float64[]
            waitLow=Float64[]
            costsOfWaiting = 0.0
            overview_usage = [t state.ocupiedRooms length(state.queue[1]) length(state.queue[2])
length(state.queue[3])]
            println("RESETTING. START OVERVIEW USAGE:", overview_usage)
            stats = SimStats(0,waitHigh, waitNormal, waitLow, overview_usage, costsOfWaiting)
        end

        if e.type==:ArrivalPatient
            # patient = Patient(:High,t) #Prob is 0.1
            # probT = rand()
            # println("Prob: ",probT)
            # if probT <= 0.6 #Prob is 0.6
            #     patient = Patient(:Normal,t)
            # elseif probT <= 0.9 #Prob is 0.3
            #     patient = Patient(:Low,t)
            # end
            arrival_patient(patient, env,state,events,stats,t)
            #ToDo: select priority class for the patient "pPriority, e.g. ":High",
":Normal", or ":Low"
            #patient = Patient(pPriority, t)
            #TODO: program function: arrival_patient(patient, env,state,events,stats,
t)and then execute the function "patient arrival"
        elseif e.type==:ServiceComplete
            #Todo: program the function "service_complete(env, state, stats, events,
t)"
        end
    end
    #Print Results to file
    agregatedStats = printResults(stats, env, id)
    return agregatedStats
end
```

```
function printResults(stats::SimStats, env::Environment, id::Int )
#print some results to screen:
println("Nr served requests: ", stats.n_served)
println("Costs of Waiting: ", stats.costsOfWaiting)

#p = plot(stats.overview_usage[:,1],stats.overview_usage[:,2] , label="Ocupied Rooms")
p = plot(stats.overview_usage[:,1],stats.overview_usage[:,3] , label="QueueHigh")
p = plot!(stats.overview_usage[:,1],stats.overview_usage[:,4] , label="QueueNormal")
p = plot!(stats.overview_usage[:,1],stats.overview_usage[:,5] , label="QueueLow")
display(p)

averageRoomOccupation = timeWeightedAverage(stats.overview_usage[:,1], stats.overview_usage[:,2])
averageQueueLengthHigh = timeWeightedAverage(stats.overview_usage[:,1], stats.overview_usage[:,3]
averageQueueLengthNormal = timeWeightedAverage(stats.overview_usage[:,1], stats.overview_usage[:,
averageQueueLengthLow = timeWeightedAverage(stats.overview_usage[:,1], stats.overview_usage[:,5])
println("Average Room Occupation: " , averageRoomOccupation)
```

```
println("Average Queue Length High: " , averageQueueLengthHigh)
println("Average Queue Length Normal: " , averageQueueLengthNormal)
println("Average Queue Length Low: " , averageQueueLengthLow)


avWHigh = mean(stats.waitHigh)
avWNorm = mean(stats.waitNormal)
avWLow=   mean(stats.waitLow)
println( "Average Waiting Time High", avWHigh)
println( "Average Waiting Time Normal", avWNorm)
println( "Average Waiting Time Low", avWLow)

#print detailed stats results to file, so you can review and analyze also later
open("C:/Users/buurd/OneDrive/Dokumenter/GitHub/Decisions/results_discrete$id.txt","a")
do io
    println(io, "nrServed ", stats.n_served)
    println(io, "Costs of Waiting ", stats.costsOfWaiting)
    println(io, "Average Room Occupation: " , averageRoomOccupation)
    println(io, "Average Queue Length High: " , averageQueueLengthHigh)
    println(io, "Average Queue Length Normal: " , averageQueueLengthNormal)
    println(io, "Average Queue Length Low: " , averageQueueLengthLow)
    println(io, "Average Waiting Time High", avWHigh)
    println(io, "Average Waiting Time Normal", avWNorm)
    println(io, "Average Waiting Time Low", avWLow)
    println(io, stats.overview_usage)
    println(io, "------------------------------")
    println(io, stats.waitHigh)
    println(io, stats.waitNormal)
    println(io, stats.waitLow)
 end

 return [stats.costsOfWaiting   averageRoomOccupation averageQueueLengthHigh averageQueueLengthNo
averageQueueLengthLow]
end


function printToFile(aggregatedStats)
    #print detailed stats results to file, so you can review and analyze also later
open("C:/Users/buurd/OneDrive/Dokumenter/GitHub/Decisions/results_aggregated.txt","a")
do io
    println(io, aggregatedStats)
 end
end

function timeWeightedAverage(time::Array{Float64}, queueLength::Array{Float64})
timeWeightedAverage = 0
for i=2:length(queueLength)
    timeWeightedAverage += queueLength[i]*(time[i]-time[i-1])
end
timeWeightedAverage = timeWeightedAverage / (time[length(time)] - time[1])
end

function main()
    #The below will call the function "simulate" for "totalSimulations", and print
```

```
results to file
    ####################################################
    #Settings
    t = 0.0
    totalRooms = 7
    weeksSimTime = 6
    weeksWarmUp = 2
    base = 24
    arrivalRate = 1/0.42
    serviceRate = 12.5
    probPerPatientType = [0.1, 0.6, 0.3]
    costsPerPatientType =  [10000.0 1000.0 100.0]
    nrPatientTypes = length(probPerPatientType)
    ####################################################

    ####################################################
    #Initialize variables
    #1. Environment
    env = Environment(
        weeksSimTime*24*7, #6 weeks '
        weeksWarmUp*24*7, # 2 weeks warmup
        base,
        arrivalRate,
        probPerPatientType,
        serviceRate,
        totalRooms,
        costsPerPatientType
        )
    totalSimulations = 1;
    curRun = 0
    overallStats=Array{Float64,2}
    while true
        aggregatedStats = simulate(env, curRun)
        println("Completed run $curRun")
        println("AggregatedStats: ", aggregatedStats)
        vcat(overallStats, aggregatedStats)
        curRun+=1;

    #If reached target, end loop
        if  curRun >= totalSimulations
            println("Completed $curRun runs")
            printToFile(overallStats)
            return
        end
    end
end

main()
#end
```