Discrete Event Simulation Analysis on Multiple Class Single Server Queueing System with Abandonments and Promotions

Discrete Event Simulation Analysis on Multiple Class Single Server Queueing System with Abandonments and Promotions

by

Shaochen Wang

Report

Presented to the Faculty of the Graduate School
of the University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree(s) of

Master of Science in Engineering

The University of Texas at Austin

December 2018

Discrete Event Simulation Analysis on Multiple Class Single Server Queueing System with Abandonments and Promotions

by

Shaochen Wang, MSE

The University of Texas at Austin, 2018

SUPERVISOR: John Hasenbein

Abstract

In the United States, the Model for End-Stage Liver Disease (MELD) scoring system is implemented to rank patients that are on the liver transplantation waiting list. However, this system is too complex to be directly analyzed. Thus, a multiple class single server queueing model with abandonments and promotions is developed as a simplified version of the original system. The queueing model is analyzed using discrete event simulation. In general, the simulation results indicate that the $\rho\alpha$ rule and the c/α rule seem to hold for minimizing the system abandonment rate and the total holding cost respectively. While the rα rule does not seem to hold for maximizing the system reward.

Contents

1. Introduction

Liver transplantation is an organ transplantation surgery that uses healthy livers to replace failing livers for diseased patients. Liver transplantation is the only way to save a patient with severe liver failure [1]. In the first ten months of 2018, nearly 7000 livers were transplanted in the United States. However, more than 13000 patients are still waiting for liver donation [2].

The United Network for Organ Sharing (UNOS) governs the Organ Procurement and Transplantation Network (OPTN) which determines the policy of allocation of each liver donation. The severity of a candidate’s disease condition is evaluated by the Model for End-Stage Liver Disease (MELD) scoring system, which has a range from 6 to 40. The score of 6 represents the least ill situation and the score of 40 represents the most ill case. A patient's MELD score may change over time depending on their disease condition [3]. Routine blood tests are conducted to evaluate the level of disease severity and then a MELD score is assigned to the patient. The frequency to conduct blood tests varies for patients with different levels of disease severity. Essentially, a more severe situation requires more frequent blood tests. For patients having the same MELD score, the waiting time is the measure to decide who has a higher priority. The waiting time is not simply the total time a patient spends in the system, but rather the time of staying at the current severity level and worse [4].

The MELD allocation system is so complicated that it is almost impossible to find an analytical method to evaluate the model’s behavior directly. Philip Ernst, John Hasenbein and Andrew Schaefer proposed a research to develop new models that have similar features as the MELD model to examine the model’s behavior [5]. Dirk van de Sande investigated the simplified fluid model and a Markov decision process model that are motivated by the MELD model. Some insight into the system dynamics is gained with numerical analysis [6]. This report will provide some insights about the simplified MELD model using the discrete event simulation method.

2. Model description

2.1 UNOS queueing model

A queueing model is used to simulate the liver transplantation waiting system. The “jobs” in the queueing model represent the patients that are waiting for liver donation. The system has multiple priority classes corresponding to different levels of disease condition. The waiting process includes abandonments (patients die before receiving the donation) and promotions (patients’ priority class changes).

The model contains K classes of patients, labeled $k = 1, \dots, K$. Patients of class k arrive according to a Poisson process with rate λ_k that is independent of the other classes. The queueing system has a single server, where the “service” represents the arrival of a liver donation. Essentially, the liver must be transplanted to a patient immediately, or it is wasted. The donated liver is assumed to be acceptable for patients of all classes. Thus, there is only one type of the liver and the interarrival time between donations is exponentially distributed with rate μ. This is a simplification of the real-world problem since livers are not always accepted by patients. Patients on the top of the waiting list may reject transplantation surgery due to other physical weakness [7]. Thus, they must wait for longer time and their service rate is effectively lower. Assuming the service rates to be equal is equivalent to assuming that all patients have the same physical weakness. Hence, this assumption yields a waiting time that is not shorter than the actual situation. Thus, this simplification in some sense gives an upper bound (in terms of waiting time) as compared to the actual system.

Patients of class k each have an exponential clock with rate α_k. When the clock expires, the patient jumps to another class, including the death class. The probability a patient jumps from class k to class l is given by p_{kl}, with $l \in \{0, 1, \dots, K\}$. A jump to any class except 0 represents a change in the patient’s health class. If the patient jumps to class 0, it represents the death of the patient while on the waiting list.

The service priority is determined by the severity of the illness. Lower MELD score indicates higher priority. The rank of the MELD score will not change but patients’ situation could change over time. The patients in the same class are ranked by the total waiting time in the current class, or worse. The patient on the top of the queueing list is served when a liver donation arrives.

2.1 Simulation model

Discrete event simulation is the method of simulating the behavior of a complicated system that cannot be easily analyzed using analytical methods. The simulation model is programmed using Python because it is a strongly and dynamically typed programming language and it conveniently provides well-defined statistical functions. A Python class named “patient” is defined to represent the real patients. The “patient” class has constant and dynamic attributes to store information of the patients. The constant attributes don’t change once they are assigned. On the contrary, the dynamic attributes keep updating while the simulation goes on. When the simulation terminates, statistics of interest can be calculated from these attributes.

The input parameters for the simulation include:

K - The number of patient classes

λ - The array of arrival rates of each class. The length of λ is $K+1$ and the first element is 0, which represents the class of deceased patients

μ - The service rate

ρ - The K by $K+1$ matrix of class transition probabilities

α - The array of clock rates of each class. The length of α is $K+1$ and the first element is 0, which represents the class of deceased patients.

The simulation process is divided into short time steps of length t. At each time step, the algorithm generates new patient arrivals, updates patient status and execute services. For patient’s arrival process, the arrival of patients of class k is a Poisson process with rate λ_k. Thus, the number of patients arrived in t follows a Poisson distribution with rate $\lambda_k t$. For the service process, since the surgery time is ignored, the service time is the interarrival time of the liver donation which is exponentially distributed with rate μ. Thus, the number of livers arrived in t also follows Poisson distribution but with rate μt. The clock time of patient of class k is an exponential random variable with rate α_k. Since t is small enough, the clock time rate can be scaled accordingly. So that the clock time is large enough and can be rounded up to the closest integer to comply with the discrete-time setting. For all the random events, if the time step t is set to be 1 unit, then the number of arrivals and services at each time step can be directly simulated by generating random variates from their corresponding distributions. At each time step, the class time of patient's current class increases by 1 and the clock time decreases by 1. When a patient's clock time decreases to 0, a new class is assigned for the patient. The class assignment is also a random event that is based on the transition probabilities. If the new class is 0, then the patient has died. For each liver donation, a patient is selected to be cured by the determined policy. The class of cured patient is set to be -1. Figure 2.1 illustrates the class transitions of a 3 classes case. The cured and deceased patients are not removed from the simulation system so that their information can be used for future analysis. However, the class time and clock time are no longer updated when a patient is cured or deceased. The simulation terminates when the predetermined simulation time is reached. Figure 2.2 illustrates the work flow of the simulation process.

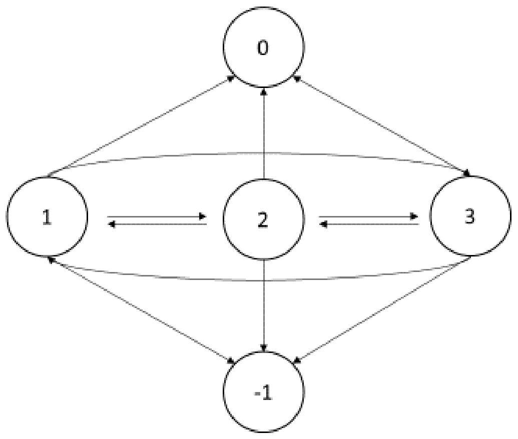

Figure 2.1 Illustration of patient's class change with 3 diseased classes

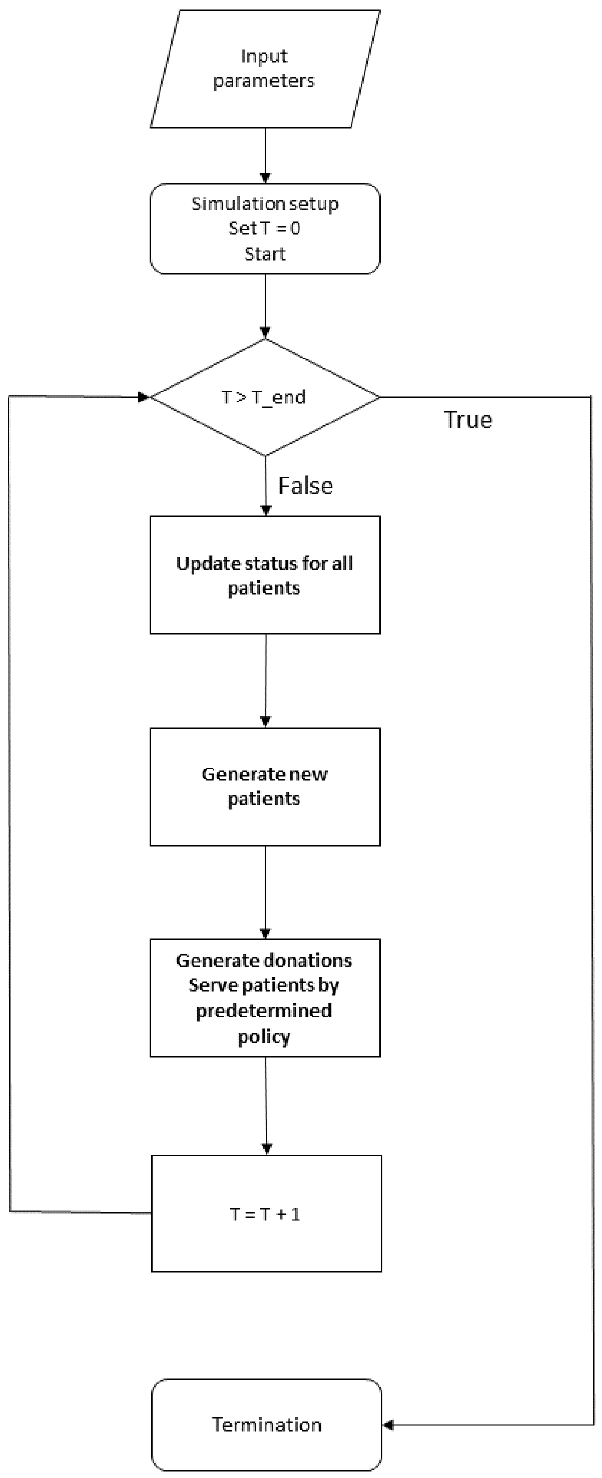

Figure 2.2 Flowchart of the simulation process

3. Problem description

3.1 Abandonment problem

Minimizing the abandonment rate of all patients is one of the most important problems to solve. The simplest policy would be prioritizing on classes that have higher abandonment probability ρ_{k0}. However, since the clock time rate is different for patients of different classes, it is possible for a patient with a lower abandonment probability but higher clock time rate to have a higher actual abandonment rate. Therefore, the parameter α_k should also be taken into consideration, which leads to a conjecture that the $\rho\alpha$ rule should hold for minimizing the abandonment rate of all patients. The $\rho\alpha$ rule essentially means that class k should have higher priority if the $\rho_{k0}\alpha_k$ value is higher. Since the clock time is exponentially distributed, $\frac{1}{\alpha_k}$ is the mean clock time of patients of class k. Hence $\frac{\rho_{k0}}{\frac{1}{\alpha_k}} = \rho_{k0}\alpha_k$ can be interpreted as the abandonment rate of patients of class k. If the conjecture is true, then this policy should yield the lowest abandonment rate.

3.2 Reward problem

Since different classes of customers may have different post-transplantation survival rates, we may want to assign different reward levels for curing patients. Suppose the reward value for a patient being cured in class k is r_k. The simplest policy to maximize the total reward of the system would be just prioritizing on the class with the highest reward value. But for a similar reason as in the abandonment problem, the parameter α_k should also be taken into consideration. Assume there are two patients in the system. Patient A is of class 1 and patient B is of class 2. Suppose $r_1 < r_2$ and $\alpha_1 > \alpha_2$, so that patient B is expected to stay longer in the current class than patient A. If the policy prioritizes on class 2, then patient B will be served first, and patient A would likely be deceased when the second liver arrives. However, if patient A is served first, then the probability that both patients will be cured is higher. Therefore, $r\alpha$ should be the decision metric instead of r. This policy is defined as the $r\alpha$ rule. If the conjecture is true, then this policy should yield the highest system reward.

3.3 Holding cost problem

Suppose the cost of staying in the queue per unit time for a patient of class k is c_k. To minimize the system holding cost, patients with higher holding cost should have higher priority. However, a patient with lower holding cost and higher clock time rate could cost more than a patient with higher holding cost and lower clock time rate. $c_k/\alpha_k = c_k \frac{1}{\alpha_k}$ is the average cost of staying in class k. Therefore, a patient with higher c/α value should have higher priority. c/α should be the decision metric of the holding cost problem. If the conjecture is true, then this policy should yield the lowest system holding cost.

4. Numerical Analysis

4.1 Simulation setup

In this report 3 classes are used to in the simulation. Hence, there are six possible policies:

Policy No.	1	2	3	4	5	6
Priority order	1>2>3	1>3>2	2>1>3	2>3>1	3>1>2	3>2>1

The arrival rates for all the classes are set to be equal: $\lambda_1 = \lambda_2 = \lambda_3 = 0.5$. The clock rate for each class is $\alpha_1 = 0.1, \alpha_2 = 0.025, \alpha_3 = 0.0125$. The reward per patient in each class is: $r_1 = 1, r_2 = 3, r_3 = 4$. The holding cost per patient of each class for 1 time step is: $c_1 = 5, c_2 = 3, c_3 = 2$. The transition matrix is shown below:

$$\rho_{kl} = \begin{bmatrix} 0.8 & 0 & 0.18 & 0.02 \\ 0.3 & 0.6 & 0 & 0.1 \\ 0.1 & 0.2 & 0.7 & 0 \end{bmatrix}, \ k = 1,2,3, \ l = 0,1,2,3.$$

To minimize the uncertainty, the arrival rates are assumed to be equal. The arrival rates are assumed to be 0.5 so that at each time step there will be 1.5 patients arrived as expectation. This value is neither too small to cause many “empty” time steps nor too large to increase the computational load. The clock time rates are assumed as mentioned to accommodate the service rates. Class 1 is assumed to be the severest condition and class 3 is the least severe situation. Patients in a more severe condition are more unstable, thus have higher clock time rate. Patients in a more severe condition also cost more to keep alive (higher holding cost, c_k) and have less reward value if cured (lower reward value, r_k). The transition probabilities are assumed based on the idea that it is the easiest to jump to one level severer class for each patient.

The policy decision metrics are then calculated:

class k	1	2	3
$\rho_{k0}\alpha_k$	0.08	0.008	0.001
$r_k\alpha_k$	0.1	0.075	0.05
c_k/α_k	50	120	160

For the abandonment problem, the expected optimal policy is policy 1. For the reward problem, the expected optimal policy is policy 1. For the holding cost problem, the expected optimal policy is policy 6.

Since the simulation behavior under overload and underload conditions could be various. Various service rates are used to simulate the system under different traffic intensities:

μ	0.15	0.75	1.2	1.5	1.8	3	7.5	15
Traffic intensity	0.1	0.5	0.8	1	1.2	2	5	10

The simulation starts without any patients in the system, so there is a warm-up period for the system to reach steady state. The survival rate under $\mu = 1$ condition is selected to be the testing metric. Figure 4.1 presents the means and standard deviations of 10 independent simulations. Figure 4.2 presents the 95% confidence intervals of the means. After 5000 time-steps, the system appears to have reached steady state. Thus, 5000 is used as the total simulation length in the following analysis.

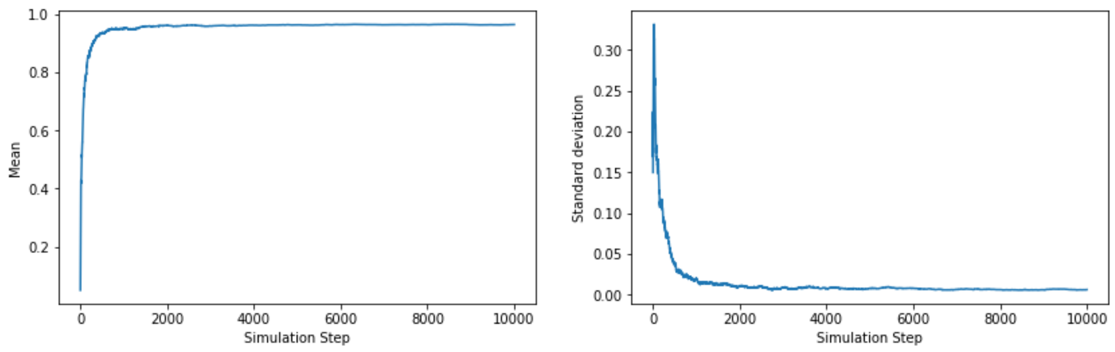

Figure 4.1 Mean and standard deviation changes over simulation steps

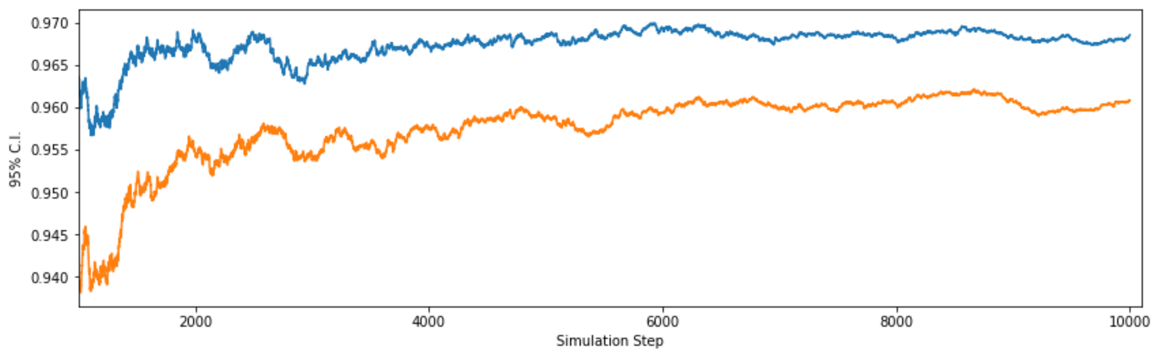

Figure 4.2 95% confidence interval changes over simulation steps

4.2 Abandonment problem

10 simulations are performed for each policy under different traffic intensities. Denote $\overline{D}_i$ to be the average system abandonment rate under policy i. The average system abandonment rate under the expected optimal policy is $\overline{D}_1$. $\overline{D}_1$ is compared with the average system abandonment rate under other policies. The following statistical tests are conducted for each pair of simulations:

$$H_o:\ \overline{D}_1 - \overline{D}_i > 0, \quad H_1:\ \overline{D}_1 - \overline{D}_i \leq 0 \quad (i \neq 1)$$

Table 4.1 - 1-tailed p-values of statistical tests of abandonment problem

Traffic intensity	vs. policy 2	vs. policy 3	vs. policy 4	vs. policy 5	vs. policy 6
0.1	0.2751	0.3772	0.0982	0.0476	0.1301
0.5	0.0841	0.0497	0.0458	0.0032	0.0448
0.8	0.0920	0.0801	6.00E-05	0.1265	1.88E-05
1	2.18E-08	1.46E-07	2.07E-17	9.77E-12	6.14E-16
1.2	9.86E-06	2.46E-05	1.63E-10	5.63E-08	7.07E-13
2	0.0591	2.30E-05	1.24E-07	0.0010	3.66E-05
5	0.4707	0.3601	0.0264	0.3398	0.1240
10	0.3196	0.1577	0.0630	0.3526	0.0198

The 1-tailed p-values of the statistical tests are shown in table 4.1. It is hard to conclude any strict patterns from the p-values. However, the p-values appear to be relatively higher in lower and higher traffic intensity cases. For common significance level 0.05, if we treat statistical tests under the same traffic intensity as a “family”, then the significance level is adjusted to 0.01 by Bonferroni correction. Thus, we cannot reject the null hypothesis when the traffic intensity is extremely high or extremely low. The reason could be that when the system is extremely overloaded, most patients will die waiting for the liver donation. The abandonment rate is highly restricted by the service rate. When the system is extremely underloaded, almost all the patients will be saved. Thus, the difference between the priority policies are harder to detect.

For moderate traffic intensity cases, the evidence to reject null hypothesis is very strong. We may conclude the $\rho\alpha$ rule holds for these cases.

4.3 Reward problem

10 simulations are performed for each policy under different traffic intensities. Denote $\bar{R}_i$ to be the average system reward under policy i. The average system reward under the expected optimal policy is $\bar{R}_1$. $\bar{R}_1$ should be larger than average system reward received under other policies. The following statistical tests are conducted for each pair of simulations:

$$H_o:\ \bar{R}_i - \bar{R}_1 > 0, \quad H_1:\ \bar{R}_i - \bar{R}_1 \leq 0 \quad (i \neq 1)$$

Table 4.2 - 1-tailed p-values of statistical tests of reward problem

Traffic intensity	vs. policy 2	vs. policy 3	vs. policy 4	vs. policy 5	vs. policy 6
0.1	0.4309	1.0000	1.0000	1.0000	1.0000
0.5	0.7801	1.0000	1.0000	1.0000	1.0000
0.8	1.0000	1.0000	1.0000	1.0000	1.0000
1	0.5684	1.0000	1.0000	0.9994	1.0000
1.2	0.0046	0.4978	0.9029	0.0667	0.7971
2	0.1768	0.5754	0.5936	0.6806	0.2896
5	0.5023	0.8529	0.7396	0.5983	0.5886
10	0.6984	0.7362	0.8408	0.7582	0.7002

The p-values are much larger than 0.01 which is the adjusted significance level by Bonferroni correction. There is very strong evidence to not reject the null hypothesis. Therefore, the $r\alpha$ rule does not seem to hold.

When the system is extremely overloaded, the survival rates of different policies are the same and are limited by the service rate. Thus, the policy that prioritizes on the class with the highest reward value yields the highest system reward value and α is irrelevant to the result in this case.

When the system is extremely underloaded, almost all patients are cured. Thus, the system rewards are expected to be the same.

4.4 Holding cost problem

10 simulations are performed for each policy under different traffic intensities. Denote $\bar{C_i}$ to be the average system holding cost under policy i. The average system holding cost under the expected optimal policy is $\bar{C_6}$. $\bar{C_6}$ should be less than average holding cost simulated under other policies. The following statistical tests are conducted for each pair of simulations:

$$H_o\colon\ \bar{C_i} - \bar{C_6} > 0, \quad H_1\colon\ \bar{C_i} - \bar{C_6} \le 0 \quad (i \ne 6)$$

Table 4.3 - 1-tailed p-values of statistical tests of reward problem

Traffic intensity	vs. policy 1	vs. policy 2	vs. policy 3	vs. policy 4	vs. policy 5
0.1	5.38E-09	9.18E-13	6.12E-10	1.61E-10	0.3090
0.5	3.09E-18	1.04E-17	3.29E-12	3.34E-11	4.58E-11
0.8	4.23E-15	5.42E-11	3.77E-15	0.2650	3.94E-10
1	4.94E-07	8.02E-08	7.91E-06	0.2815	3.62E-06
1.2	0.9504	0.2903	0.4974	0.8559	0.3828
2	0.6106	0.5674	0.9236	0.8648	0.9163
5	0.5143	0.6898	0.7225	0.4028	0.7292
10	0.8594	0.9303	0.7461	0.6337	0.9728

For an adjusted significance level of 0.01, most p-values are significant when the system is overloaded. Thus, the c/α rule seems to hold for overloaded cases. When the system is underloaded, the p-values are relatively large. There is strong evidence that we should accept the null hypothesis. When the traffic intensity is high, almost all the patients are cured as soon as they arrive in the system. Thus, the system holding cost approaches to 0 for all the policies. That may be the reason why the c/α rule is not effective.

5. Conclusion

Liver donation for transplantation surgery is in considerable shortage in the United States. The MELD score system is implemented to determine the priority of patients on the liver transplantation queueing list. However, this system is too complicated to be analyzed directly. As a result, the UNOS queueing model that has similar features is developed to analyze the original system. In this report, the discrete time simulation method is used to bring some insights of a simplified version of the UNOS queueing model.

The traffic intensity plays an important role in the queueing model. When the system is extremely overloaded, the main limitation is the service rate. In this case, the clock time is too short compared to the interarrival time of liver donations. Most patients will die while waiting in the list. Different policies can't affect the abandonment rate significantly. Saving patients with the highest reward value will maximize the system reward. However, the clock time is important when minimizing the holding cost. Since most patients will stay in the queue for the entire clock time, prioritizing on class with higher c/α value is the optimal policy for minimization of system holding cost. When the traffic intensity is extremely high, almost all the patient will be cured immediately while they are still in their initial class. Thus, the holding cost and the abandonment rate will approach 0 for all policies. Also, the number of patients cured per class is approximately equal to the number of patients arrived per class, which depends only on the arrival rate. Therefore, the system reward will be the same for different priority orders. When the traffic intensity is at a moderate level, the problem is rather complicated. The simulation results indicate that the $\rho\alpha$ rule and the c/α rule hold for the abandonment problem and the holding cost problem respectively. However, the rα rule doesn't hold for the reward problem. Still, this result is true only for the parameters assumed in this report. If the magnitude of the parameters or the ratio between the parameters change, the result could be completely different.

6. Recommendations

In this report, a simplified model with only 3 classes was analyzed. The results from the simulations might only be true for the specific setting in this report. In further research, simulation of the model with more classes and different parameter combinations should be conducted to gain more accurate insights. The simulation code will need to be streamlined to adapt to a large increase of computation. A different data structure can be used to keep track the status of patients of different classes. The UNOS queueing model is rather an ideal model for the liver transplantation problem in real life. The livers donated are assumed to be the same and suitable for all patients. However, patients in a severe condition could reject a less qualified liver. This problem will not only affect the death rate of the patients, but also causes problems of maximizing the usage of livers.

Appendix

Simulation code

```
### This code is created for simulate multiple class queueing system with abandonments and
promotions
### The code is created under Anaconda environment
### "numpy" and "scipy" package is required to run the code

### Import packages
import random
import math
import copy
import scipy as sp
import numpy as np
from scipy.stats import poisson as poi
from scipy.stats import expon


### Define type "patient". The "patient" objects store all the imformation generated of each
patient during the simulation
class patient():
    def __init__(self, initClass, arrTime, currentClass, classTime, clockTime, curedClass,
diedClass, endTime):
        # The class of the patient when the patient arrives in the system. This attribute
doesm't change
        self.initClass = initClass

        # The arrival time of the patient. This attribute doesm't change
        self.arrTime = arrTime

        # The current class of the patient. This attribure will be updated
        self.currentClass = currentClass

        # The time that a ptient spend in each class. It is a list of length numClass+1. This
attribure will be updated
        self.classTime = classTime

        # The clock time of the patient. This attribure will be updated
        self.clockTime = clockTime

        # The class in which the patient is cured. This attribute doesm't change
        self.curedClass = curedClass

        # The class in which the patient is died. This attribute doesm't change
        self.diedClass = diedClass

        # The time that the patient is died or cured. This attribute doesm't change
        self.endTime = endTime

    # Calculate the time a patient spend in the cuurent and worse class
    def worseTime(self):
        i = 1
        worseT = 0
        while i <= self.currentClass:
            worseT += self.classTime[i]
            i += 1
        return worseT


### Transfer the input parameters into desired form
```

```
def inputProcess(numClass, rho):
    classes = [i for i in range(numClass+1)]
    patientsList = []

    # Create the cumulative transition probability matrix
    rhoC = copy.deepcopy(rho)
    for row in rhoC[1:]:
        for j in range(1, len(row)):
            row[j] = row[j] + row[j-1]

    return classes, patientsList, rhoC


### Generate a new patient
def newPatient(inputClass, T):
    initClass = inputClass
    currentClass = initClass
    diedClass = currentClass
    classTime = [0 for i in range(numClass + 1)]
    clockTime = math.ceil(expon.rvs(scale = 1/alpha[inputClass]))
    return patient(initClass, T, currentClass, classTime, clockTime, 0, diedClass, 0)


### Generate new arrivals
def newArr(classes, patientsList, lambdas, alpha, T):
    for i in classes[1:]:
        # The number of arrivals at each time step
        rnum = poi.rvs(lambdas[i])

        for j in range(rnum):
            patientsList.append(newPatient(i, T))
    return patientsList


### Update patient Status
def pUpdate(classes, patientsList, alpha, rhoC, T):
    if len(patientsList) > 0:
        for p in patientsList:
            if p.currentClass != -1 and p.currentClass != 0:
                p.classTime[p.currentClass] +=1
                p.clockTime -=1

                # The class of the patient changes when the clock time is 0
                if p.clockTime == 0:
                    rnum = random.random()
                    for j in range(len(rhoC[p.currentClass])):
                        if rnum <= rhoC[p.currentClass][j]:
                            p.currentClass = j
                            break

                    if p.currentClass != 0:
                        p.clockTime = math.ceil(expon.rvs(1/alpha[p.currentClass]))
                        p.diedClass = p.currentClass
                    if p.currentClass == 0:
                        p.endTime = T

    return patientsList

### The queueing policy of the simulation. The function returns the index of the patient to
serve in "patientList"
def policy(patientsList, classOrder):
    minClass = len(classOrder)
    maxWorseT = 0
    c = False
    for i in range(len(patientsList)):
        if patientsList[i].currentClass > 0:
            if classOrder.index(patientsList[i].currentClass) < minClass:
                minClass = classOrder.index(patientsList[i].currentClass)
                c = i
            if classOrder.index(patientsList[i].currentClass) <= minClass and
patientsList[i].worseTime() > maxWorseT:
                maxWorseT = patientsList[i].worseTime()
                c = i


### The service procedure
def serv(classes, patientsList, mu, T, classOrder):
    # Number of donations arrivaled in time T
    rnum = poi.rvs(mu)

    for j in range(rnum):
        c = policy(patientsList, classOrder)
        if c:
            patientsList[c].curedClass = patientsList[c].currentClass
            patientsList[c].currentClass = -1
            patientsList[c].endTime = T
    return patientsList

### The main simulation function
### This function is called to run the simulation
### Inputs:
### T_end: int. Total duration of the simulation
### numClass: int. The number of classes excluding the died class and cured class
### lambdas: list. len(lambdas) == numClass. The arrival rates of each class
### mu: int. The service rate.
### rho: list[list]. numClass by numClass+1. The transition probability
### alpha: list. len(alpha) == numClass. The clock rate of each class
### classOrder: list. The class priority order of the policy. example: if class 3 > class 2 >
class 1, then [3, 2, 1]
### The function returns a list containing all the "patient" type object
def mainSim(T_end, numClass, lambdas, mu, rho, alpha, classOrder):
    classes, patientsList, rhoC = inputProcess(numClass, rho)
    T = 0
    while T <= T_end:
        # update patients status
        patientsList = pUpdate(classes, patientsList, alpha, rhoC, T)

        # check new arrivals
        patientsList = newArr(classes, patientsList, lambdas, alpha, T)

        # check service
        patientsList = serv(classes, patientsList, mu, T, classOrder)

        T += 1
    return patientsList
```

References

[1] UCSF. Liver Transplant. https://transplant.surgery.ucsf.edu/conditions--procedures/liver-transplant.aspx

[2] United Network for Organ Sharing. Transplant trends. https://unos.org/data/%20transplant-trends/

[3] United Network for Organ Sharing. Questsion and Answers for Transplant Candidates About MELD and PELD. https://www.unos.org/wp-content/uploads/unos/MELD_PELD.pdf.

[4] Penn Medicine. Understanding the MELD Score. https://www.pennmedicine.org/updates/blogs/transplant-update/2013/september/understanding-the-meld-score.

[5] P. Ernst, J. J. Hasenbein and A. J. Schaefer. (2018). Retrieved May 23, 2018, Research proposal.

[6] D.T.J. van de Sande. Multi-class Single Server Queues with Abandonments and Promotions. 2018

[7] Oren K. Fix, M.D.. (August 12, 2015). Why are some people turned down for a liver transplant. https://www.swedish.org/blog/2015/08/why-are-some-people-turned-down-for-a-liver-transplant