POLITECNICO DI MILANO

School of Industrial and Information Engineering
Master of Science in Mechanical Engineering

A GENERALIZED $c\mu$ RULE FOR SCHEDULING MULTICLASS QUEUEING SYSTEMS WITH DEPENDENT ARRIVAL RATES

Supervisor: Prof. PAOLO TRUCCO
Co-Supervisor: Prof. JOHN J. HASENBEIN

Authors:
PIETRO INVERNIZZI ID. 863978
SIMONE RIPANTI ID. 859218

Academic Year 2016/2017

Acknowledgements

This thesis is the result of a six months research program carried out at the University of Texas at Austin. First of all we would like to thank Prof. John J. Hasenbein who supported and guided us throughout the development of the entire work at UT. Thanks also to our supervisor at Politecnico, Prof. Paolo Trucco, for his advices, his patience and the overseas support. We are very grateful to everyone who contributed in making this experience unforgettable.

Contents

List of Figures

List of Tables

Abstract

Purpose: A service discipline chooses the order in which the customers are served at each station. Examples of service disciplines are the $first\ in\ first\ out$ (FIFO) and $last\ in\ first\ out$ (LIFO), they are the most popular and probably the most intuitive disciplines. Jobs among all classes are ranked according to the arrival time at the queue. When we are dealing with several classes with different arrival rates the whole procedure could become very hard. For this reason $static\ buffer\ priority$ disciplines are sometimes preferred. All the classes at each station have a fixed ranking, the higher the ranking the higher the priority. The $c\mu$ rule is one of these disciplines, it ranks the classes according to the holding cost and the service rate. The goal of this thesis is to determine the reasons why the $c\mu$ rule is not always able to predict the optimal policy that minimizes the total cost when the arrival rates are dependent on the job in service. We started to address the simplest system with only two classes of products/clients, then we moved to a more general n-class problem. We wanted to determine a new formula able to predict the right scheduling sequence with the new set of parameters (dependent arrival rates were added). We applied then the new rule to a real case to test its effectiveness.
Knowledge background: The $c\mu$ rule was introduced for the first time in 1961 in the paper "Queues" written by D.R. Cox and Walker Smith [1]. The $c\mu$ rule defines the optimal scheduling sequence for different classes in the system. The priority level is determined by taking the product of the service rate and the holding cost rate for a given class. The larger the product the higher the priority level. In the following years there were several papers and studies on this topic with different assumptions and parameters used in the model. So far nobody considered the case where the arrival rates were dependent on the product class in service.
Research Questions: Raaijmakers [2] showed that for some sets of parameters the $c\mu$ rule is not able to predict the optimal policy so to minimize the total costs. We moved from this point looking for an explanation of the strange behavior and possibly a solution.
Design/Methodology/Approach: The thesis is divided in two main parts, the first one is purely theoretical while the second one is the application to a real case of the theoretical results. In the theoretical part we run the the fluid model optimization problem several times changing the parameters. The results obtained with the fluid model were compared with the Markov decision process formulation of the problem. Both the methods showed the same conclusions. After the definition of a new relationship for the optimal scheduling thanks to the fluid model and the MDP, we proved it analytically. In the real case application we run a big number of simulations (fixed time increment type) to see the goodness of the new rule applied to the hospital emergency room process.
Main Finding: The main result achieved is the proposal of a new rule able to minimize the total holding cost when scheduling the production of several product/client classes

having different priority level. The rule was defined firstly for a simple two-class system and later it was extended to the general n-class system.

Pratical Implementations: It is possible to apply the new rule to systems/processes in which the arrival rates are dependent on the product/class in service such as hospital emergency rooms and postal offices.

Limitations and future developments: The main limitations of the new rule are the followings: the arrival rates of products/clients depend on what is in service at that time. We considered only the holding cost and we neglect other type of cost such as set up cost and penalties due to non completion of the jobs. We did not consider all the possible parameters, it could be interesting to include the abandon rate in the model. After a certain time spent in the queue the client can decided to abandon the system. For these reasons could be interesting to analyze the effects of these parameters on the rule we obtained and possibly to define even a more general one.

Keywords: Markov Decision Process; Fluid Model; Queuing networks; Scheduling; Optimal policy.

Sommario

Obiettivo: La disciplina di servizio determina l'ordine con cui i prodotti sono processati ad ogni stazione. Alcuni esempi di discipline sono dati da $first\ in\ first\ out$ (FIFO) e $last\ in\ first\ out$ (LIFO), esse sono probabilmente le più conosciute. I prodotti di tutte le classi sono classificati in base all'istante temporale in cui sono entrati nella rispettiva coda. Quando il sistema è caratterizzato da molte classi con diversi tassi di arrivo il processo di classificazione può diventare molto complesso. Per questa ragione a volte vengono preferite politiche di *static buffer priority*. Tutti i prodotti di una singola classe sono descritti da un rango costante, più il rango è alto più il livello di priorità è elevato. La $c\mu$ rule rientra in questa categoria di politiche e ordina le classi basandosi sul costo di attesa in coda e il tasso di processamento. Lo scopo di questo studio è di determinare le ragioni per cui la $c\mu$ rule non è sempre in grado di suggerire la politica ottimale per minimizzare il costo totale del sistema quando i tassi di arrivo dipendono dalla classe sotto processo al momento dell'arrivo in coda. Abbiamo iniziato affrontando il caso più semplice con sole due classi di prodotti/clienti diversi, da qui siamo poi passati al caso generale con n classi diverse. Un ulteriore obiettivo è la definizione di una nuova relazione che permetta di determinare la politica ottimale con i nuovi parametri aggiunti. Infine abbiamo l'obiettivo di applicare questa nuova regola a un caso reale per testarne la sua bontà.
Background letterario: La $c\mu$ rule è stata introdotta per la prima volta nel 1961 nell'articolo scientifico, poi diventato libro, "Queues" da D.R. Cox e Walter Smith [1]. La $c\mu$ rule definisce la sequenza di schedulazione ottimale per diverse classi di prodotto all'interno di un sistema. Il livello di priorità è definito come il prodotto tra il tasso di processamento e il costo di attesa in coda di una classe di prodotti. Più il risultato è grande più il livello di priorità è elevato. Negli anni successivi sono stati condotti diversi studi su questo argomento introducendo diversi parametri e ipotesi nel sistema sotto esame. Ad oggi nessuno ha mai considerato il caso in cui i tassi di arrivo dipendono dalla classe in servizio al momento dell'arrivo in coda.
Domande di ricerca: Raaijmakers [2] ha mostrato che per alcune combinazioni di parametri la $c\mu$ rule non è in grado di suggerire la politica ottimale per minimizzare i costi. Queste osservazioni sono il punto di partenza del nostro studio.
Design/Metodologia/Approccio: Lo studio è diviso in due parti distinte. La prima parte è prettamente teorica mentre la seconda consiste nell'applicazione au un reale caso reale pratico dei risultati teorici ottenuti. Nella parte teorica abbiamo eseguito svariate volte il problema di ottimizzazione basato sull'approssimazione Fluid model del sistema cambiando di volta in volta i parametri in ingresso. I risultati ottenuti sono stati confrontati con la formulazione del Markov Decision Process del problema. Entrambi i metodi hanno restituito risultati concordanti. Da questi risultati abbiamo ricavato una nuova relazione per lo scheduling ottimo e infine abbiamo dimostrato analiticamente la sua

validità. Il caso reale trattato è la gestione dei differenti codici di gravità e il loro smistamento all'interno di un Pronto Soccorso. Abbiamo simulato le dinamiche del Pronto Soccorso attraverso una simulazione a tempi fissi, l'obiettivo era di testare la bontà della regola formulata.

Risultati Principali: Il principale risultato di questa tesi è la definizione di una nuova regola per la valutazione dei livelli di priorità per diverse classi di prodotti/clienti. La regola è stata definita inizialmente per il solo caso semplice con due classi e successivamente è stata estesa al caso generale con n classi.

Implicazioni Pratiche:E' possibile applicare i risultati di questo studio a numerosi casi pratici in cui i tassi di arrivo dipendono dalla classe/prodotto in servizio come nel caso di uffici postali o Pronto soccorsi.

Limitazioni e sviluppi futuri: Le principali limitazioni di questa nuova regola sono le seguenti: è necessario che il tasso di arrivo dei prodotti/clienti dipenda dal tipo di prodotto-cliente sotto processo al momento dell'arrivo in coda. Il modello inoltre considera solamente il costo di permanenza in coda del singolo prodotto, abbiamo trascurato altri tipi di costo come il costo di set up e penalties dovute al mancato completamento del prodotto. Non abbiamo considerato tutti i possibili parametri, può essere utile infatti includere nel modello la possibilità da parte del cliente di abbandonare il sistema dopo un certo lasso di tempo passato in coda. Per queste ragioni sarebbe interessante studiare gli effetti che questi parametri hanno sulla regola da noi ricavata e possibilmente, in futuro, determinarne una di carattere ancor più generale.

Parole chiave: Markov Decision Process; Fluid Model; Queuing networks; Scheduling; Optimal policy.

Chapter 1

Introduction

The $c\mu$ rule is an important rule in the scheduling field. It was introduced for the first time in 1961 by D.R. Cox and Walter Smith [1]. It predicts the optimal job allocation that minimizes the total cost when only the holding cost is considered. According to this rule we can define the scheduling priority between two or more classes of products/clients by considering the product of the service rate and the holding cost of each class. The higher the value obtained the greater is the priority for a given class. The main advantage of this rule is the ease in computing the different levels of priority even with a huge number of product/clients. However the $c\mu$ rule is not always the optimal scheduling rule, it depends on the particular system considered. In a real system also other parameters can affect the outcome of the optimization process. In the following years several studies were made in order to improve, and possibly extend, its validity. In this work we investigate the effects on the optimal scheduling policy of arrival rates dependent on the product in service. We started from the work of Raaijmakers [2] that noticed that the $c\mu$ rule was not always suggesting the true optimal policy for all the sets of parameters. He was not able to fully disclose the reasons why the $c\mu$ rule was not always optimal, but he came up with a practical "rule of thumb" to describe the occurrence of these exceptions. It was formulated by running several times the optimization problem and changing the key parameters. The work is divided in two different parts, one theoretical where we derive structural results, and one practical, where we test the new rule in a real case. We studied a system composed by a single versatile server and two different classes of products with their own queue. The arrival rates depend on the type of product that is currently in service. The decision maker has to know which job class produce first in order to minimize the total holding cost. For the sake of simplicity we started to address the two-class system keeping the costs equal. The simplest system allowed us to understand its basic behavior and the stability conditions required. Once the main concepts were fully understood we added also the cost effects. We used the fluid model optimization for its ease of computation and helpful insights to determine the formula of the new rule. Then we formulated the problem as a Markov decision process looking for confirmations of the obtained results. The Markov decision process is characterized by a bigger computational burden but it is able to give more complete results about the whole state space. The MDP has been solved with a simple linear program (dual simplex solver). The results however are not satisfactory due to the low accuracy of the solver and the strong boundary effects. We decided then to solve the problem also with another approach: the modified policy iteration method. The results in this case are very close to

the fluid model ones, the boundary effects however are still an issue. After the matching of results coming from different methods we decided to prove analytically the validity and the optimality of the new relationship. Once the optimality was proven we followed the same procedure to generalize the results to a n-class system. We applied the rule to the problem of optimizing the service prioritization in an hospital Emergency room. We simulated the behavior of the system by comparing different type of policies looking for the optimal one. We wanted to see if the scheduling optimality could be achieved by using the new rule. Dependent arrival rates do affect the selection of the optimal class, the more one class rates change, the bigger its effect on the overall optimality with respect to the usual $c\mu$ rule. We determined with our work a generalization for the previous $c\mu$ rule in order also to account for the effects of the dependent arrival rates. The results obtained are for the general n-class system. We applied then the theoretical results to an italian emergency room process. The classes represent the different severity codes used to group the patients upon arrival. The goal is to improve the whole scheduling process, depending on the codes, in order to minimize the possibility of losing human lives.

Chapter 2

State of the art

The $c\mu$ rule is a static priority rule that depends on few parameters, service rates and holding cost rates. Service rates are considered constant while the costs are considered linear. It is very simple to implement due to its simplicity. The $c\mu$ rule optimality was first addressed by Smith [3] for a single stage production system and a deterministic, static environment where no dynamic arrivals were considered. Very soon Cox and Smith [1] proved its optimality also for a stochastic, dynamic environment with arbitrary time horizon and a general multi-class $M/G/1$ system. Moving from those studies a lot of extensions were formulated. Klimov [4] extended the $c\mu$ rule validity to a multi-class $M/G/1$ system with feedback. Harrison [5] proved its optimality for a discounted problem in a $M/G/1$ system. The objective is to maximize the expected net present value of service rewards received minus holding costs incurred over an infinite planning horizon. Tcha and Pliska [6] combined both the extensions together, they studied the scheduling problem for a single server system with feedback of the customers so as to maximize the expected discounted reward over an infinite planning horizon. Buyukkoc et al. [7] showed that the $c\mu$ applies with arbitrary arrival processes ($G/M/1$ queues) while Hirayama et al. [8] extended the previous results to a more general $G/G/1$ queue, moreover they considered the $G/DFR/1$ system characterized by a decreasing failure rate service time. Rami et al. [9] included in the model the customer abandonment rate θ, they showed that a routing policy that assigns priority to classes according to their index $c_i\mu_i/\theta_i$ is asymptotically optimal for minimizing the overall long run average holding cost. Sisbot and Hasenbein [10] showed that the $c\mu$ rule is still optimal when scheduling and routing at the same time a multi-class single-server system. Van Mieghem [11] considered a single-server multi-class queuing system that incurs a delay cost for each class residing in the queue for a given amount of time. He considered non linear cost functions and showed that with nondecreasing convex delay costs the $c\mu$ rule is asymptotically optimal if the system operates in heavy traffic regime. Raaijmakers [2] included the possibility to have arrival rates dependent on the class in service, he noticed that the $c\mu$ was not able to predict always the optimal policy. He was not able however to fully disclose the reasons behind these anomalies. Moving from these observations, the goal of this thesis is to determine an extension of the original $c\mu$ able to account for the arrival rates effects and better predict the optimal scheduling policy.

Chapter 3

Model description

In this chapter we present the network studied. The system is characterized by n different job classes labelled with the subscript s. Each job class has its own queue Q_s. Jobs arrive to the system following a Poisson process with rate λ_{ij}. The arrival rates depend on the job class in service, so the general expression for the arrival rate is λ_{ij} that indicates the arrival rate of job class i when job class j is currently in service. The jobs are processed by a single flexible server. It can process only one single job at a time, processing times are assumed independent and identically distributed exponential random variables with rate μ_s where the subscripts s is related to the job class. A class s incurs a holding cost rate c_s (cost per single job per unit time); the goal is to minimize the average total cost rate. The main parameters of the system are then the arrival rates λ_{ij}, the service rates μ_s and the holding cost rates c_s:

$$\Lambda = \begin{bmatrix} \lambda_{11} & \cdots & \lambda_{1n} \\ \vdots & \ddots & \vdots \\ \lambda_{n1} & \cdots & \lambda_{nn} \end{bmatrix} \qquad \mu = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_n \end{bmatrix} \qquad c = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}.$$

The system is considered non-idling, i.e., the server works at full rate whenever there are jobs in the system. When both the queues are empty there is a virtual product under service. For this reason we define extra arrival rates λ_{s0}:

$$\lambda_{s0} = max\{\lambda_{s1}, \ldots, \lambda_{sn}\},$$

the arrival rates in case of empty system are equal to maximum rate possible for a given job class. Throughout all the work presented all the sets of parameters are chosen in a way to meet the stability conditions defined by Ernst et al. [12]. We use unstable parameters only to define the stability region in the two-class system. The simplest system characterized by $n = 2$ is depicted in Figure 3.1.

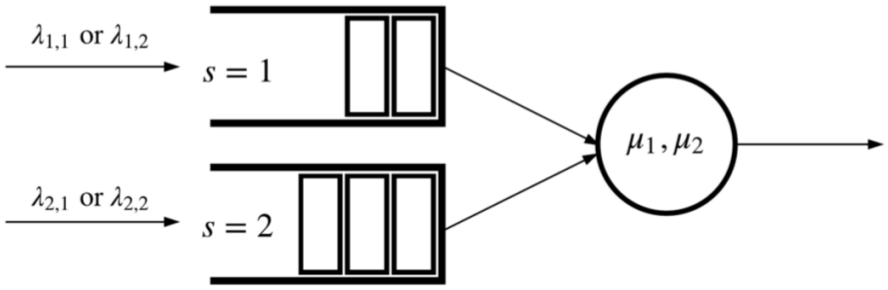

Figure 3.1: *Two-class queuing network model.*

Chapter 4

Two-class system

In this chapter we present the simplest case of the more general model where only two classes are considered. This allows to reduce the complexity of the problem and obtain easily readable graphs and results. The set of parameters reduces to:

$$\Lambda = \begin{bmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \end{bmatrix} \quad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \quad c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}.$$

4.1 Stability

Before starting to address the main problem about the optimality of the $c\mu$ rule, we want to asses the stability of the system. Remembering that for a single-class and single-server system the stability condition is $\rho = \frac{\lambda}{\mu} \leq 1$. For a multi-class and single-server system it is necessary to compute the eigenvalues of the matrix [12]:

$$\begin{bmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \end{bmatrix} \begin{bmatrix} \frac{1}{\mu_1} & 0 \\ 0 & \frac{1}{\mu_2} \end{bmatrix} = \begin{bmatrix} \frac{\lambda_{11}}{\mu_1} & \frac{\lambda_{12}}{\mu_2} \\ \frac{\lambda_{21}}{\mu_1} & \frac{\lambda_{22}}{\mu_2} \end{bmatrix}.$$

The eigenvalues are obtained by putting the determinant equal to 0:

$$det \begin{bmatrix} eig - \frac{\lambda_{11}}{\mu_1} & \frac{\lambda_{12}}{\mu_2} \\ \frac{\lambda_{21}}{\mu_1} & eig - \frac{\lambda_{22}}{\mu_2} \end{bmatrix} = eig^2 - eig\left(\frac{\lambda_{11}}{\mu_1} + \frac{\lambda_{22}}{\mu_2}\right) + \frac{\lambda_{11}\lambda_{22} - \lambda_{12}\lambda_{21}}{\mu_1\mu_2} = 0.$$

The solutions of the quadratic equation are, indeed, the eigenvalues.

$$\begin{aligned} eig_{1,2} &= \frac{\lambda_{11}}{2\mu_1} + \frac{\lambda_{22}}{2\mu_2} \pm \sqrt{\frac{1}{4}\left(\frac{\lambda_{11}}{\mu_1} + \frac{\lambda_{22}}{\mu_2}\right)^2 - \left(\frac{\lambda_{11}\lambda_{22} - \lambda_{12}\lambda_{21}}{\mu_1\mu_2}\right)} \\ &= \frac{\lambda_{11}}{2\mu_1} + \frac{\lambda_{22}}{2\mu_2} \pm \sqrt{\frac{1}{4}\left(\frac{\lambda_{11}}{\mu_1} - \frac{\lambda_{22}}{\mu_2}\right)^2 + \frac{\lambda_{12}\lambda_{21}}{\mu_1\mu_2}}. \end{aligned} \tag{4.1}$$

Numerical results indicate that the maximum eigenvalue is always equal to 1 when the following conditions are satisfied:

$$\begin{aligned} \mu_1 &= \lambda_{11} + \lambda_{21} \\ \mu_2 &= \lambda_{22} + \lambda_{12}. \end{aligned} \tag{4.2}$$

We now apply the conditions (4.2) to (4.1):

$$eig_{1,2} = \frac{1}{2}\left(\frac{\mu_1 - \lambda_{21}}{\mu_1}\right) + \frac{1}{2}\left(\frac{\mu_2 - \lambda_{12}}{\mu_2}\right) \pm \sqrt{\frac{1}{4}\left(\frac{\mu_1 - \lambda_{21}}{\mu_1} - \frac{\mu_2 - \lambda_{12}}{\mu 2}\right)^2 + \frac{\lambda_{21}\lambda_{12}}{\mu_1\mu_2}}$$

$$= \frac{1}{2} - \frac{\lambda_{21}}{2\mu_1} + \frac{1}{2} - \frac{\lambda_{12}}{2\mu_2} \pm \sqrt{\frac{1}{4}\left(1 - \frac{\lambda_{21}}{\mu_1} - 1 + \frac{\lambda_{12}}{\mu_2}\right)^2 + \frac{\lambda_{21}\lambda_{12}}{\mu_1\mu_2}}$$

$$= 1 - \frac{\lambda_{21}}{2\mu_1} - \frac{\lambda_{12}}{2\mu_2} \pm \sqrt{\frac{1}{4}\left(\frac{\lambda_{12}}{\mu_2} - \frac{\lambda_{21}}{\mu_1}\right)^2 + \frac{\lambda_{21}\lambda_{12}}{\mu_1\mu_2}}$$

$$= 1 - \frac{\lambda_{21}}{2\mu_1} - \frac{\lambda_{12}}{2\mu_2} \pm \sqrt{\frac{1}{4}\left(\frac{\lambda_{12}}{\mu_2} + \frac{\lambda_{21}}{\mu_1}\right)^2}$$

$$= 1 - \frac{\lambda_{21}}{2\mu_1} - \frac{\lambda_{12}}{2\mu_2} \pm \frac{1}{2}\left(\frac{\lambda_{12}}{\mu_2} + \frac{\lambda_{21}}{\mu_1}\right).$$

The two eigenvalues are then:

$$eig_{1+} = 1$$
$$eig_{2-} = 1 - \frac{\lambda_{21}}{\mu_1} - \frac{\lambda_{12}}{\mu_2} < 1.$$

Thanks to (4.2) we know with certainty one point laying on the contour of the stability region. This point is also easily obtainable from the system parameters. Starting from this point we want to determine the whole stability region; to do so we need to test different sets of parameters (λ_s are fixed) and compute their eigenvalues.

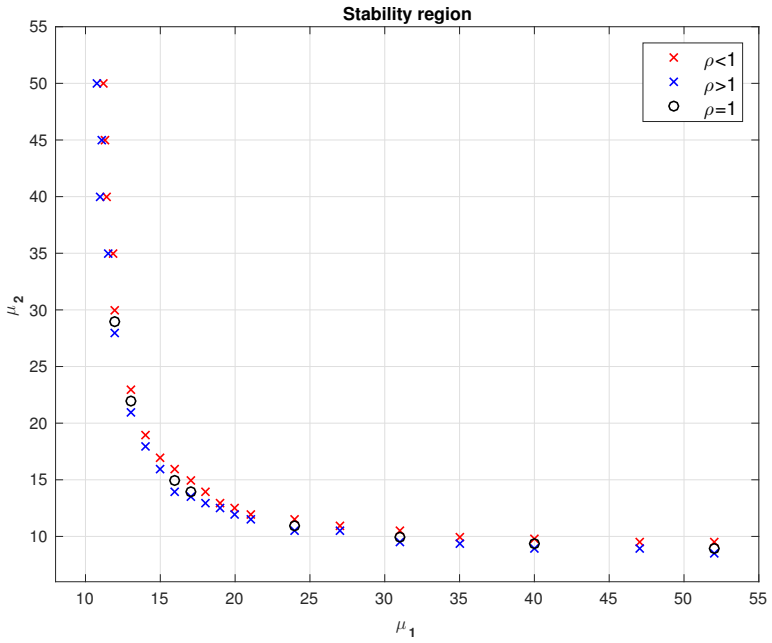

Figure 4.1: *Experimental points to determine the stability region, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 7, \lambda_{21} = 6, \lambda_{22} = 8$.

The points in Figure 4.1 show a pattern that looks like an equilateral hyperbola. Our first guess is that the asymptotes will likely be the diagonal λ_s. When the service rate of one class is much bigger than the other one, the problem reduces to a simple one class system. The slower service rate then needs only to guarantee at least the relative diagonal λ (the extra diagonal λ basically disappears). We have then one known point from (4.2) and two asymptotes, which is enough to fully determine the hyperbola equation:

$$\mu_2 = \frac{a\mu_1 + b}{c\mu_1 + d}. \tag{4.3}$$

The parameters a, b, c, d are defined by using the available data:

$$asym_{\mu_2} = \lambda_{22} = \frac{a}{c} \rightarrow a = \lambda_{22}, c = 1$$
$$asym_{\mu_1} = \lambda_{11} = -\frac{d}{c} \rightarrow d = -\lambda_{11}, c = 1.$$

Using then (4.2):

$$\lambda_{22} + \lambda_{12} = \frac{\lambda_{22}\left(\lambda_{11} + \lambda_{21}\right) + b}{\lambda_{21}} \rightarrow b = \lambda_{21}\lambda_{12} - \lambda_{11}\lambda_{22}.$$

All the parameters are now determined and we replace them in (4.3):

$$\mu_2 = \frac{\lambda_{22}\mu_1 + \lambda_{21}\lambda_{12} - \lambda_{11}\lambda_{22}}{\mu_1 - \lambda_{11}}. \tag{4.4}$$

We compare the hyperbola equation and the previous points. (4.4) does describe exactly the behavior of the stability region (see Figure 4.2).

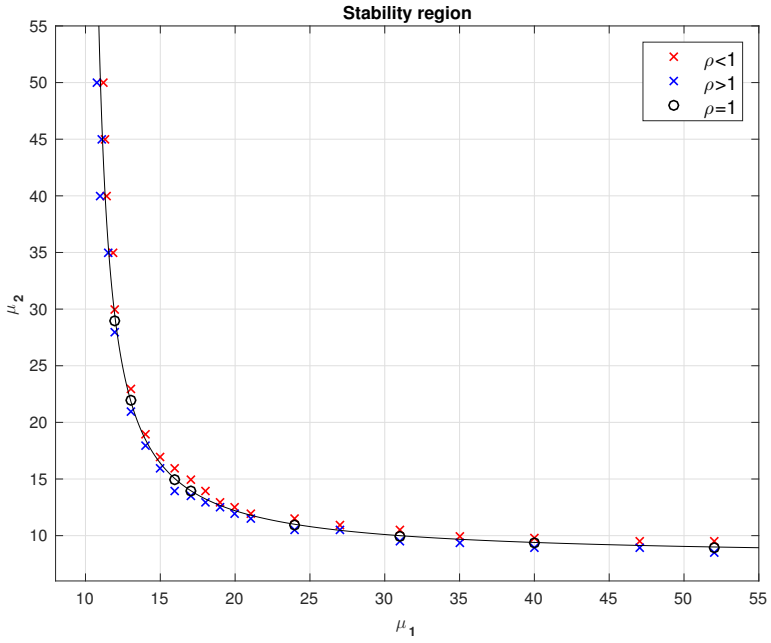

Figure 4.2: *Analytical stability hyperbola obtained from the experimental points, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 7, \lambda_{21} = 6, \lambda_{22} = 8$.

4.2 Fluid approximation

The queueing network is approximated by describing it as a fluid model. The fluid model is the deterministic equivalent of the queuing network under study. Interarrival and service times are no longer exponentially distributed and they are replaced by deterministic parameters λ and μ respectively. The different queues are seen as buckets filled with different fluids, where only one bucket at the time can be drained. The choice about what bucket drain is up to the decision maker. This model is very helpful and useful in case of processes with a very large queue size (computational burden is much lower with respect to other methods), and can provide useful insights about the real behavior of the system.

4.2.1 Fluid dynamics

We define the main quantities of the model: the subscript s represents the job class considered, in this case $s = \{1, 2\}$, $A_s(t)$ is the amount of fluid of type s that arrives in the time span $[0, t]$, $T_s(t)$ is the cumulative time spent by server to process job class s in $[0, t]$, $D_s(t)$ is the number of jobs completed belonging to class s in $[0, t]$, $Q_s(t)$ is the queue size of class s in $[0, t]$ and finally $U_s(t)$ represents the scheduling control vector. The fluid model evolves according to:

$$
\begin{aligned}
Q_s(t) &= Q_s(0) + A_s(t) - D_s(t) \\
A_s(t) &= \begin{bmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \end{bmatrix} \begin{bmatrix} T_1(t) \\ T_2(t) \end{bmatrix} \\
D_s(t) &= \mu_s T_s(t) \\
T(t) &= T_1(t) + T_2(t) = t.
\end{aligned}
\tag{4.5}
$$

Since we are interested in the rate of change of the fluid level, (4.5) are rewritten in terms of derivatives as follows:

$$
\begin{aligned}
\frac{d}{dt}Q_1(t) &= \lambda_{11}u^1(t) + \lambda_{12}u^2(t) - \mu_1 u^1(t) \\
\frac{d}{dt}Q_2(t) &= \lambda_{21}u^1(t) + \lambda_{22}u^2(t) - \mu_2 u^2(t).
\end{aligned}
\tag{4.6}
$$

4.2.2 Optimization problem setup

The goal is to minimize the total holding cost subject to the fluid dynamics constraints in (4.6). The decision making can be modeled as an optimization problem. The exact problem has a quadratic form but by discretizing the total time span in small intervals, it can be transformed into a linear program. As the time interval shrinks the results are closer to the original problem. The problem is then described as:

$$
\begin{array}{lll}
\text{find:} & x = [u_0^1, u_1^1, \ldots, u_N^1, u_0^2, u_1^2, \ldots, u_N^2 & \\
 & \quad Q_1^1, Q_1^2, \ldots, Q_1^{N-1}, Q_2^1, Q_2^2, \ldots, Q_2^{N-1}] & \\
 & & \\
\text{minimize:} & \Delta t \sum_{n=1}^{N-1} c_1^n Q_1^n + c_2^n Q_2^n & \\
 & & \\
\text{subject to:} & Q_1^{n+1} = Q_1^n + (\lambda_{11} - \mu_1)u_n^1 \Delta t + \lambda_{12} u_n^2 \Delta t & n = 0, 1, \ldots, N \\
 & Q_2^{n+1} = Q_2^n + (\lambda_{22} - \mu_2)u_n^2 \Delta t + \lambda_{21} u_n^2 \Delta t & n = 0, 1, \ldots, N \\
 & u_n^1 + u_n^2 \leq 1 & n = 0, 1, \ldots, N \\
 & u_n^1 \geq 0 & n = 0, 1, \ldots, N \\
 & u_n^2 \geq 0 & n = 0, 1, \ldots, N \\
 & Q_1^0 \geq 0 & n = 0, 1, \ldots, N \\
 & Q_2^0 \geq 0 & n = 0, 1, \ldots, N \quad .
\end{array}
$$

We define some boundary conditions: Q_1^0,Q_2^0 are the non null initial fluid levels and Q_1^N,Q_2^N are the final fluid levels that must be set to zero. The total time T should be chosen large enough to ensure the draining of the queues. The number of time increments N should be chosen according to the total time T in order to have time intervals enough small to well approximate the fluid model ($\Delta t = \frac{T}{N}$). The number of equations depends on the discretization level chosen; more time increments lead to more constraints.

4.2.3 Equal costs

When the arrival rates are independent the $c\mu$ rule is the optimal scheduling policy. If the costs are equal it reduces to a simple priority rule. The two optimal regions are separated by the first quadrant angle bisector, as shown in Figure 4.3.

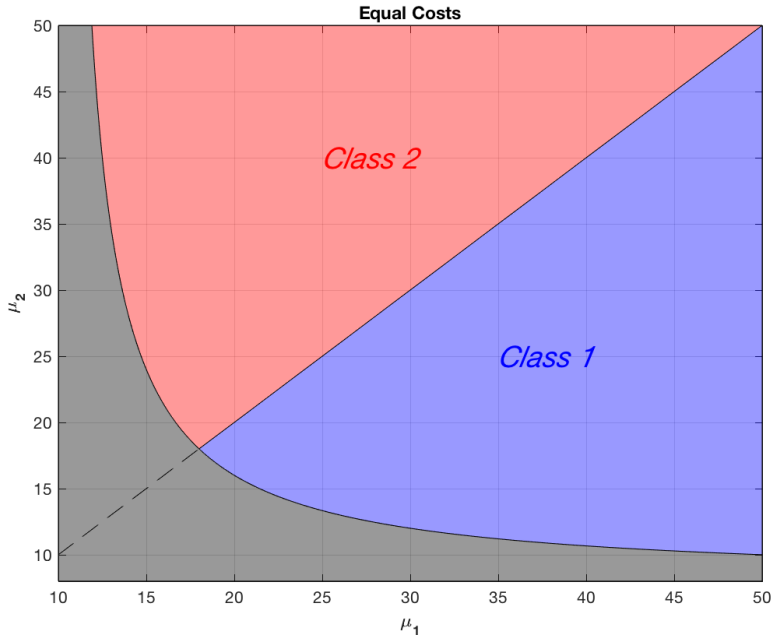

Figure 4.3: *Class optimality regions predicted by the $c\mu$ rule, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 10, \lambda_{21} = 8, \lambda_{22} = 8$.

If we select a point above the bisector, the fluid model cost minimization chooses class 2 as the optimal one to be served (Figure 4.4).

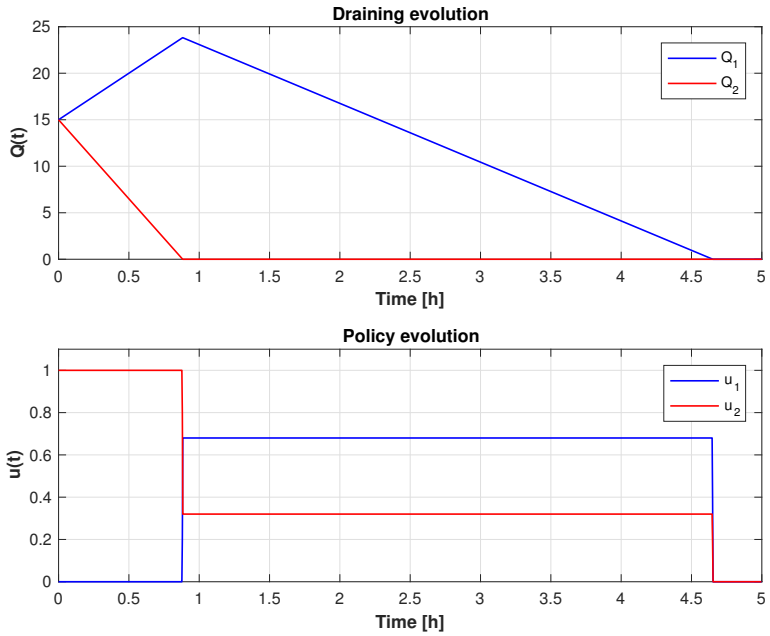

Figure 4.4: *Fluid model results when Class* 2 *is optimal.*

Otherwise if the point is below the bisector class 1 is chosen as the optimal one to be served (Figure 4.5).

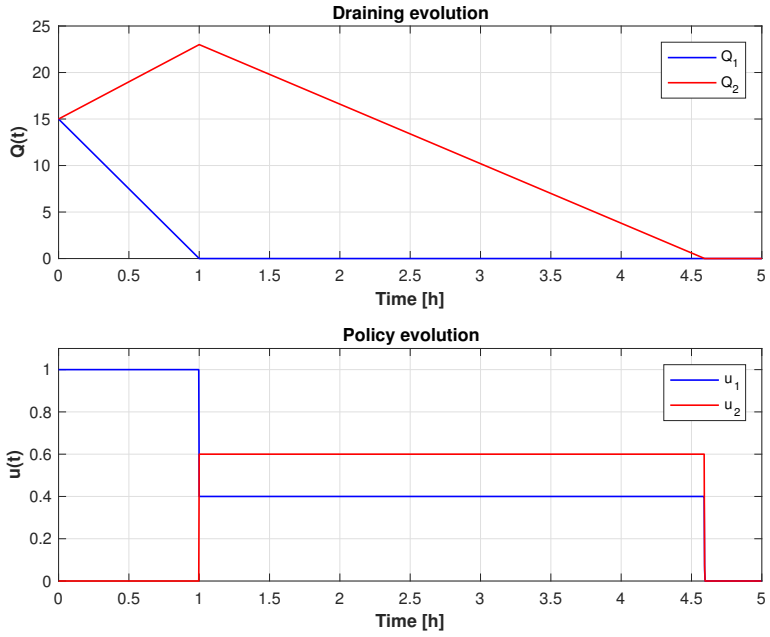

Figure 4.5: *Fluid model results when Class* 1 *is optimal.*

Is the $c\mu$ rule still optimal when the arrival rates are dependent on the class in service? The answer is apparently no. In Figure 4.6 we depict an example in which the $c\mu$ rule

does not hold.

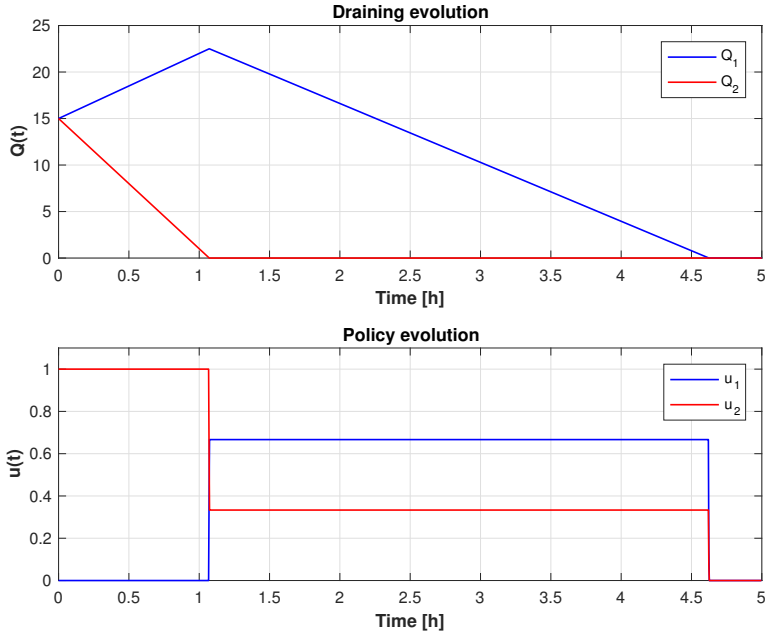

Figure 4.6: *Fluid model results when Class* 2 *is optimal even if cμ rule predicts Class* 1.

As we did to determine the stability region, we test several sets of parameters (λ_s are fixed) to identify the new $equi - priority$ line. The results shown in Figure 4.7 are quite interesting.

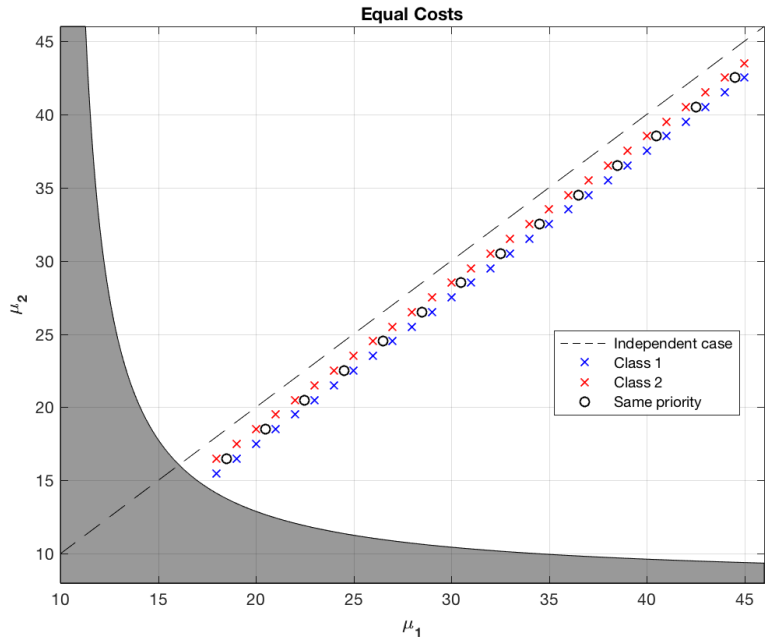

Figure 4.7: *Experimental points to define the new dividing line, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 7, \lambda_{21} = 6, \lambda_{22} = 8$.

All the points $equi - priority$ in Figure 4.7 belong to the same straight line which happens to be parallel to the first quadrant angle bisector (Figure 4.8).

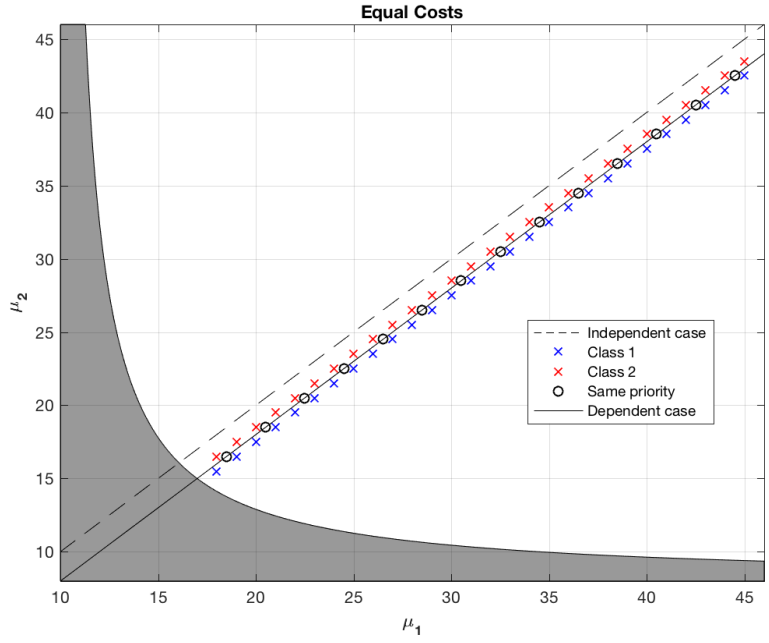

Figure 4.8: *New dividing line obtained from the experimental points, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 7, \lambda_{21} = 6, \lambda_{22} = 8$.

Repeating the same experiment also for different sets of λ_s the results are always the same, the new separating line is parallel to the bisector. The next step is to understand how the new line is affected by the arrival rates. The quantity of which the new line moves is given by the difference between the summation of terms in the same column:

$$\Delta_s = (\lambda_{11} + \lambda_{21}) - (\lambda_{12} + \lambda_{22}).$$

The dividing line, instead of being $\mu_2 = \mu_1$, is:

$$\mu_2 = \mu_1 - (\lambda_{11} + \lambda_{21}) + (\lambda_{12} + \lambda_{22}). \tag{4.7}$$

Three different scenarios can take place:

- Δ_s=0: the dividing line is still the bisector; independent arrival rates belong to this category.

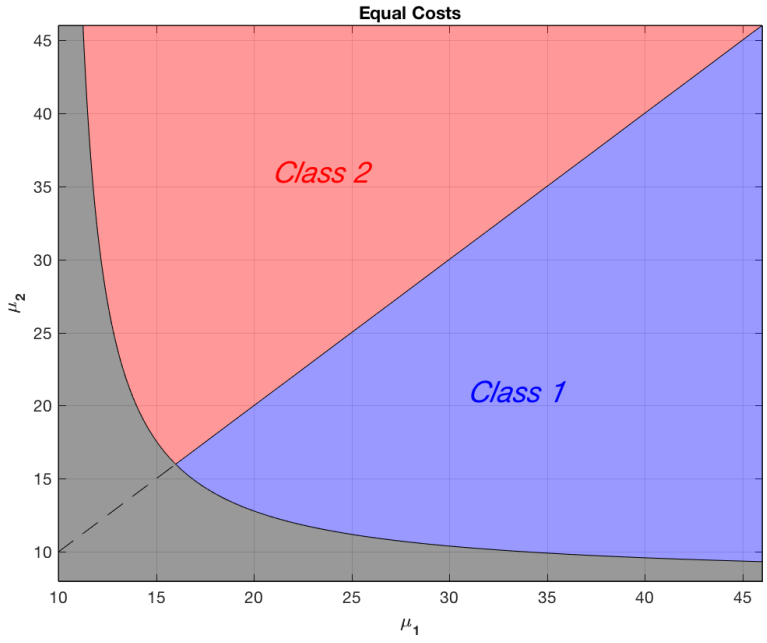

Figure 4.9: *Optimality regions when* $\Delta_s = 0$*, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 8, \lambda_{21} = 6, \lambda_{22} = 8$.

- Δ_s>0: the dividing line moves down i.e., class 2 region increases while class 1 region decreases. The new region between the former line and the new one is characterized by points prioritizing class 1 according to $c\mu$, however the fluid model chooses class 2 as optimal. We call this zone *Reverse* 2.

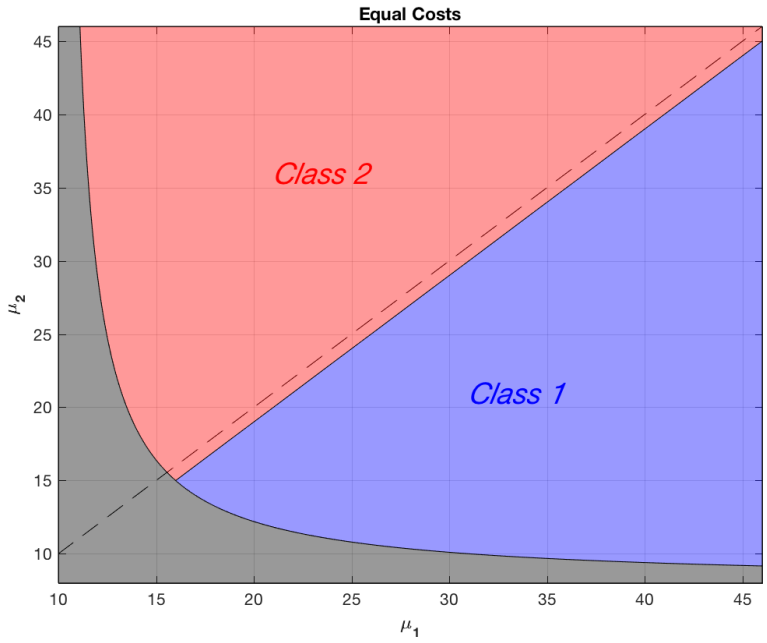

Figure 4.10: *Optimality regions when* $\Delta_s > 0$*, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 7, \lambda_{21} = 6, \lambda_{22} = 8$.

- Δ_s<0: the dividing line moves up, class 1 region increases while class 2 region decreases. The new region between the former line and the new one is characterized by points prioritizing class 2 according to $c\mu$. However the fluid model chooses class 1 as optimal, we call this zone $Reverse$ 1.

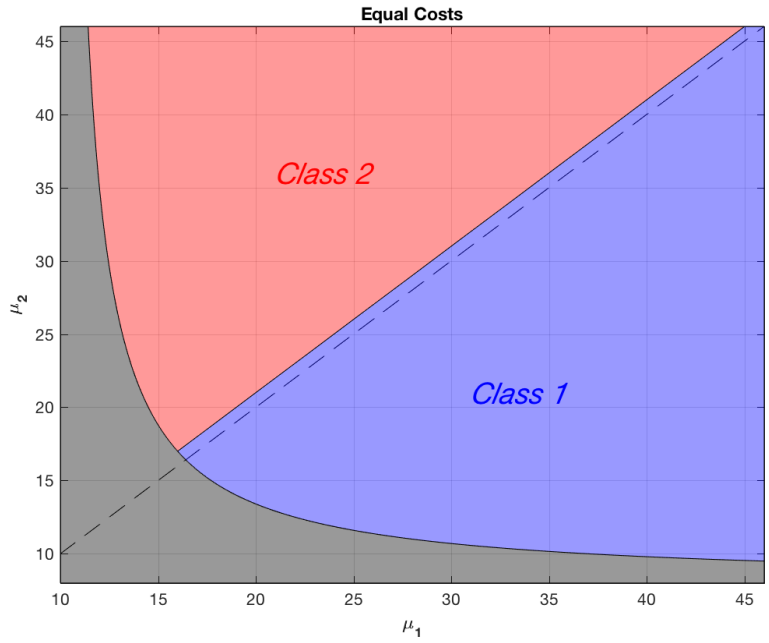

Figure 4.11: *Optimality regions when* $\Delta_s < 0$*, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 9, \lambda_{21} = 6, \lambda_{22} = 8$.

4.2.4 Cost effects

In this section we add the cost effects. We pass from a $\mu_2 - \mu_1$ based graph to a new representation:

$$c_1\mu_1 = c_2\mu_2 \rightarrow \frac{c_1}{c_2} = \frac{\mu_2}{\mu_1}.$$

The dividing line is still the first quadrant angle bisector. The optimal class regions separation is depicted in Figure 4.12.

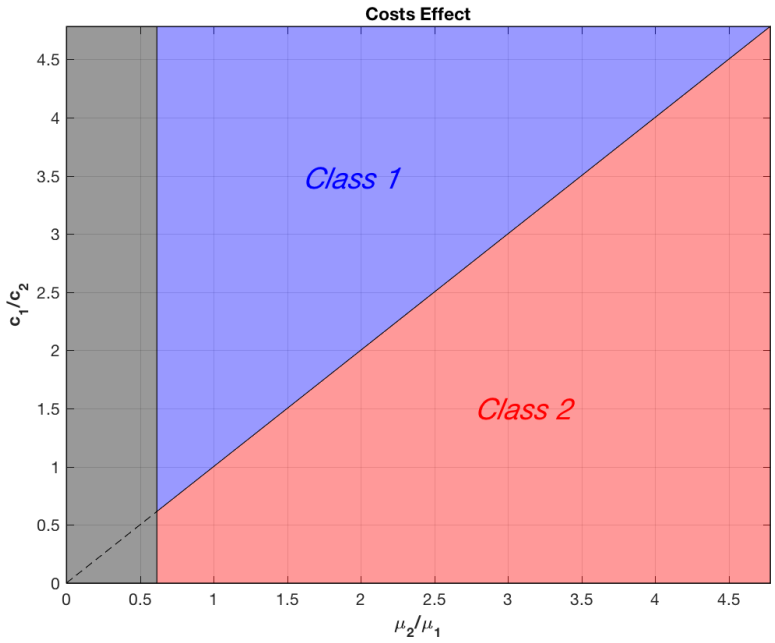

Figure 4.12: *Optimality regions predicted by the cμ rule with cost effects, the parameters are:* $\lambda_{11} = 10, \lambda_{12} = 10, \lambda_{21} = 8, \lambda_{22} = 8$.

As always we need to determine the new separating line. The procedure adopted is the same used for stability and equal costs case: the λ_s and μ_s parameters are kept fixed while $\frac{c_1}{c_2}$ is increased until the change of optimal policy is met, then the procedure is repeated for different values of $\frac{\mu_2}{\mu_1}$.

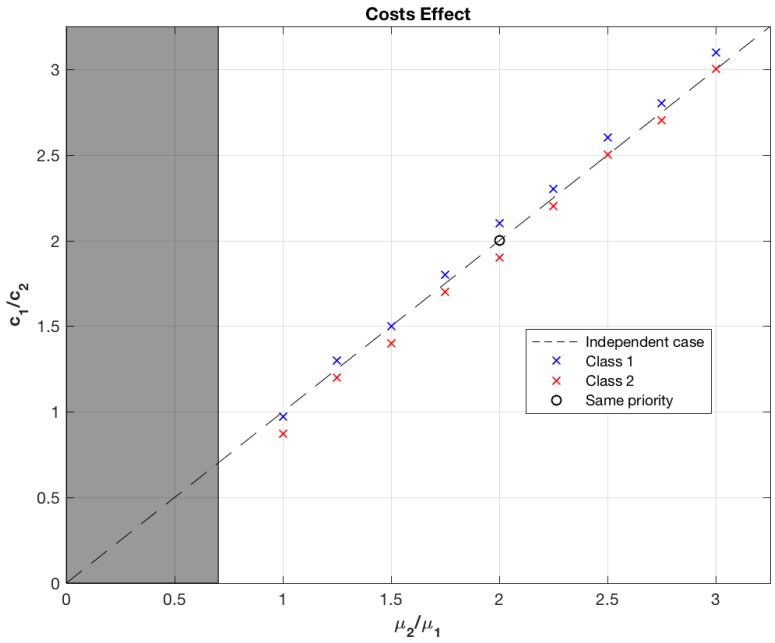

Figure 4.13: *Experimental points to determine the new separating line with cost effects, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 9, \lambda_{21} = 6, \lambda_{22} = 8, \mu_1 = 19.5$.

For any set of parameters there is a particular point that lays on both the former bisector and the new dividing line, we call it the *break even point*, and its coordinates are:

$$\frac{c_1}{c_2} = \frac{\mu_2}{\mu_1} = \frac{\lambda_{22} - \lambda_{21}}{\lambda_{11} - \lambda_{12}}. \tag{4.8}$$

Figure 4.13 shows that the possible new separation line could be a straight line as well with a different slope with respect to the starting bisector. (4.8) and (4.7), from the equal costs case, represent two points belonging to the new line. We can then derive the equation of the straight line using these points. The slope m of the line is:

$$m = \frac{y_{eq.c} - y_{bep}}{x_{eq.c} - x_{bep}}. \tag{4.9}$$

We use (4.7) and (4.8) in (4.9):

$$\begin{aligned} m &= \frac{1-\frac{\lambda_{22}-\lambda_{21}}{\lambda_{11}-\lambda_{12}}}{\frac{\mu_1-(\lambda_{11}+\lambda_{21}-\lambda_{22}-\lambda_{12})}{\mu_1}-\frac{\lambda_{22}-\lambda_{21}}{\lambda_{11}-\lambda_{12}}} \\ &= \frac{\frac{\lambda_{11}-\lambda_{12}-\lambda_{22}+\lambda_{21}}{\lambda_{11}-\lambda_{12}}}{\frac{\mu_1-(\lambda_{11}+\lambda_{21}-\lambda_{22}-\lambda_{12})}{\mu_1}-\frac{\lambda_{22}-\lambda_{21}}{\lambda_{11}-\lambda_{12}}} \\ &= \frac{\frac{\lambda_{11}-\lambda_{12}-\lambda_{22}+\lambda_{21}}{\lambda_{11}-\lambda_{12}}}{\frac{[\mu_1-(\lambda_{11}+\lambda_{21}-\lambda_{22}-\lambda_{12})](\lambda_{11}-\lambda_{12})-\mu_1(\lambda_{22}-\lambda_{21})}{\mu_1(\lambda_{11}-\lambda_{12})}} \\ &= \frac{\mu_1[(\lambda_{11}-\lambda_{12})-(\lambda_{22}-\lambda_{21})]}{\mu_1(\lambda_{11}-\lambda_{12})-(\lambda_{11}-\lambda_{12})[(\lambda_{11}-\lambda_{12})-(\lambda_{22}-\lambda_{21})]-\mu_1(\lambda_{22}-\lambda_{21})}. \end{aligned} \tag{4.10}$$

We define:

$$\begin{aligned} \Delta_1 &= \lambda_{11}-\lambda_{12} \\ \Delta_2 &= \lambda_{22}-\lambda_{21}. \end{aligned}$$

Then we replace them in (4.10):

$$m = \frac{\mu_1-(\Delta_1-\Delta_2)}{\mu_1\Delta_1-\Delta_1(\Delta_1-\Delta_2)-\mu_1\Delta_2} = \frac{\mu_1(\Delta_1-\Delta_2)}{-\Delta_1^2+\Delta_1(\Delta_2-\mu_1)-\mu_1\Delta_2}.$$

The denominator is a quadratic equation in Δ_1 and can be decomposed in $-(\Delta_1-\mu_1)(\Delta_1-\Delta_2)$. (4.10) finally becomes:

$$m = \frac{\mu_1(\Delta_1-\Delta_2)}{-(\Delta_1-\mu_1)(\Delta_1-\Delta_2)} = \frac{\mu_1}{\mu_1-\lambda_{11}+\lambda_{12}}. \tag{4.11}$$

The intercept q is obtained from $y = mx + q$ using (4.7):

$$\begin{aligned} q &= \frac{c_1}{c_2} - m\frac{\mu_2}{\mu_1} \\ &= 1 - m\left(\frac{\mu_1-[(\lambda_{11}+\lambda_{21})-(\lambda_{12}+\lambda_{22})]}{\mu_1}\right). \end{aligned} \tag{4.12}$$

Combining (4.11) and (4.12) together we define the new line equation:

$$\frac{c_1}{c_2} = m\frac{\mu_2}{\mu_1} + q$$

$$= m\frac{\mu_2}{\mu_1} + 1 - m\left(\frac{\mu_1 - [(\lambda_{11} + \lambda_{21}) - (\lambda_{12} + \lambda_{22})]}{\mu_1}\right)$$

$$= \frac{m\mu_2 + \mu_1 - m\mu_1 + m(\lambda_{11} + \lambda_{21} - \lambda_{12} - \lambda_{22})}{\mu_1}$$

$$= \frac{m(\mu_2 - \mu_1) + \mu_1 + m(\lambda_{11} + \lambda_{21} - \lambda_{12} - \lambda_{22})}{\mu_1}$$

$$= \frac{\frac{\mu_1}{\mu_1 - (\lambda_{11} - \lambda_{12})}(\mu_2 - \mu_1) + \mu_1 + \frac{\mu_1}{\mu_1 - (\lambda_{11} - \lambda_{12})}(\lambda_{11} + \lambda_{21} - \lambda_{12} - \lambda_{22})}{\mu_1}$$

$$= \frac{\mu_1}{\mu_1 - (\lambda_{11} - \lambda_{12})}\frac{(\mu_2 - \mu_1)}{\mu_1} + \frac{\mu_1}{\mu_1} + \frac{\mu_1(\lambda_{11} + \lambda_{21} - \lambda_{12} - \lambda_{22})}{\mu_1(\mu_1 - \lambda_{11} + \lambda_{12})}$$

$$= \frac{\mu_2 - \mu_1}{\mu_1 - (\lambda_{11} - \lambda_{12})} + 1 + \frac{(\lambda_{11} + \lambda_{21} - \lambda_{12} - \lambda_{22})}{\mu_1 - \lambda_{11} + \lambda_{12}}$$

$$= \frac{\mu_2 - \mu_1 + \mu_1 - \lambda_{11} + \lambda_{12} + \lambda_{11} + \lambda_{21} - \lambda_{12} - \lambda_{22}}{\mu_1 - \lambda_{11} + \lambda_{12}}$$

$$= \frac{\mu_2 + \lambda_{21} - \lambda_{22}}{\mu_1 - \lambda_{11} + \lambda_{12}}.$$

Eventually we obtain:

$$c_1[\mu_1 - (\lambda_{11} - \lambda_{12})] = c_2[\mu_2 - (\lambda_{22} - \lambda_{21})]. \tag{4.13}$$

The results obtained in (4.13) are depicted in Figure 4.14.

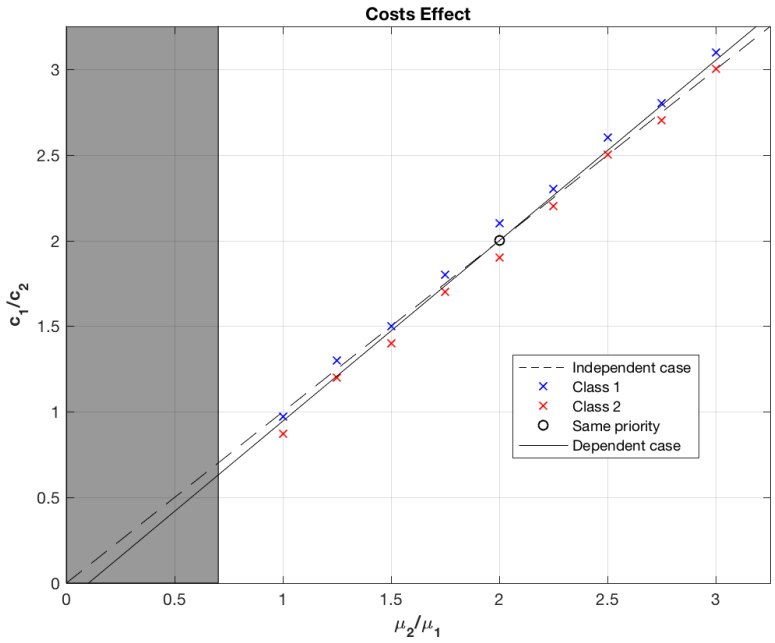

Figure 4.14: *New dividing line obtained from the experimental points with cost effects, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 9, \lambda_{21} = 6, \lambda_{22} = 8, \mu_1 = 19.5$.

(4.13) is an extension of the usual $c\mu$ rule, when the arrival rates are dependent on the product in service. Their variation affects the choice the optimal scheduling rule. Different scenarios are possible.

- $(\lambda_{11} - \lambda_{12}) = 0$ and $(\lambda_{22} - \lambda_{21}) = 0$: (4.13) reduces to the simple $c\mu$ rule. Its representation is given in Figure 4.12.
- $(\lambda_{11} - \lambda_{12}) \neq 0$ and $(\lambda_{22} - \lambda_{21}) \neq 0$: Both classes have dependent arrival rates Figure 4.15.

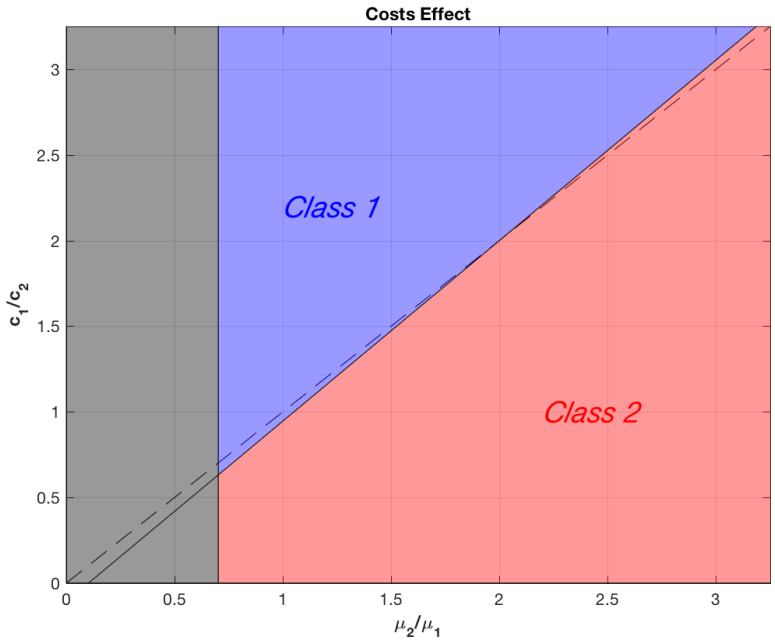

Figure 4.15: *Optimality regions when* $(\lambda_{11} - \lambda_{12}) \neq 0$ *and* $(\lambda_{22} - \lambda_{21}) \neq 0$ *with cost effects, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 9, \lambda_{21} = 6, \lambda_{22} = 8, \mu_1 = 19.5$.

- $(\lambda_{11} - \lambda_{12}) \neq 0$ and $(\lambda_{22} - \lambda_{21}) = 0$: Class 2 has independent arrival rates, then the *break even point* coincides with the origin.

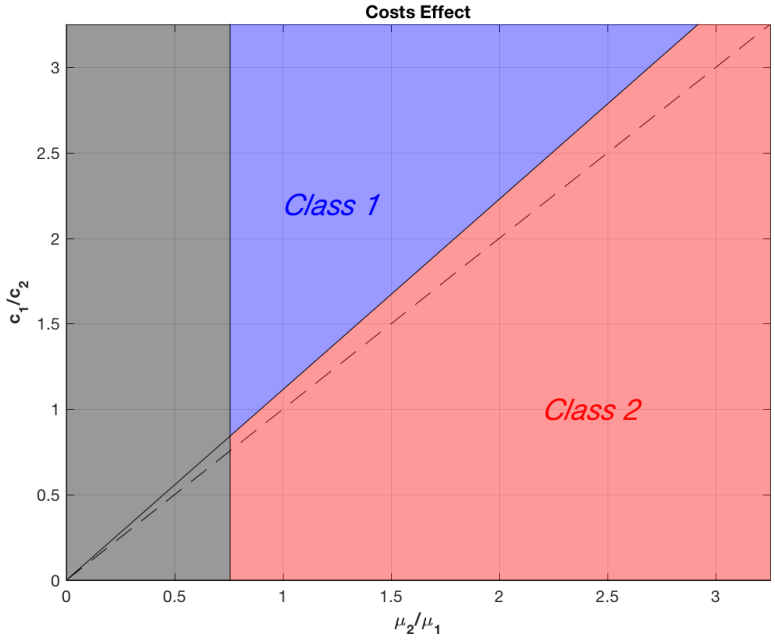

Figure 4.16: *Optimality regions when* $(\lambda_{11} - \lambda_{12}) \neq 0$ *and* $(\lambda_{22} - \lambda_{21}) = 0$ *with cost effects, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 8, \lambda_{21} = 8, \lambda_{22} = 8, \mu_1 = 19.5$.

- $(\lambda_{11} - \lambda_{12}) = 0$ and $(\lambda_{22} - \lambda_{21}) \neq 0$: Class 1 has independent arrival rates, then the *break even point* tends to infinity.

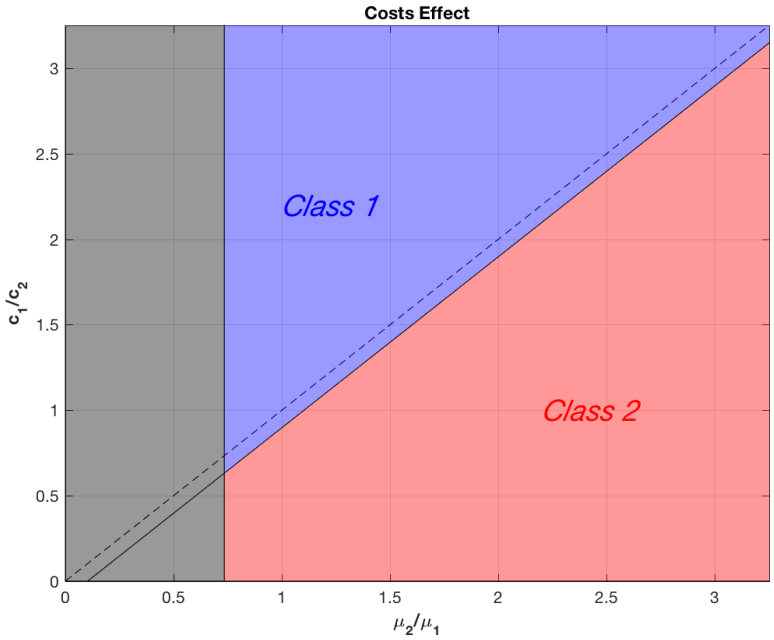

Figure 4.17: *Optimality regions when* $(\lambda_{11} - \lambda_{12}) = 0$ *and* $(\lambda_{22} - \lambda_{21}) \neq 0$ *with cost effects, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 10, \lambda_{21} = 6, \lambda_{22} = 8, \mu_1 = 19.5$.

4.3 Numerical Markov decision process

The problem can be formulated as a Continuous Time Markov Decision Process (CT-MDP), the reason is that the transition probabilities between different states are affected by the product under process and thus by the decision maker's choice. The MDP method under certain conditions allow us to find an optimal policy (set of decision rules for each state and for each time period). More generally it provides the decision maker with a complete set of information about each state, while with the fluid model only some states are described depending on how the problem evolves. The objective is still the minimization of the total holding cost over an infinite time horizon, and it is described as:

$$\limsup_{t\to\infty} \frac{1}{t} E_{(Q_0^1, Q_0^2)} \left[\int_0^t (c_1 Q_1^\pi(s) + c_2 Q_2^\pi(s)) ds \right]. \tag{4.14}$$

The decision maker, starting from the initial state Q_0^1,Q_0^2, selects a set of action to be taken when the actual state changes. This mapping from states to actions characterizes a policy π. Since the subsequent states depend on the decisions taken they are represented as $Q_s^\pi(t)$.

4.3.1 Discretized Markov decision process

The initial problem is converted into a corresponding Discrete Time Markov Decision Process (DTMDP). In the original formulation the time intervals are not the same. To

be able to use numerical methods obtain results constant and discrete times are needed. The total amount of states is given by the combinations of Q^1 and Q^2 and it is infinite if the problem is unbounded. Since the computational burden quickly increases as the size of one single queue increases (the curse of dimensionality) it is necessary to truncate the state space (example in Figure 4.18).

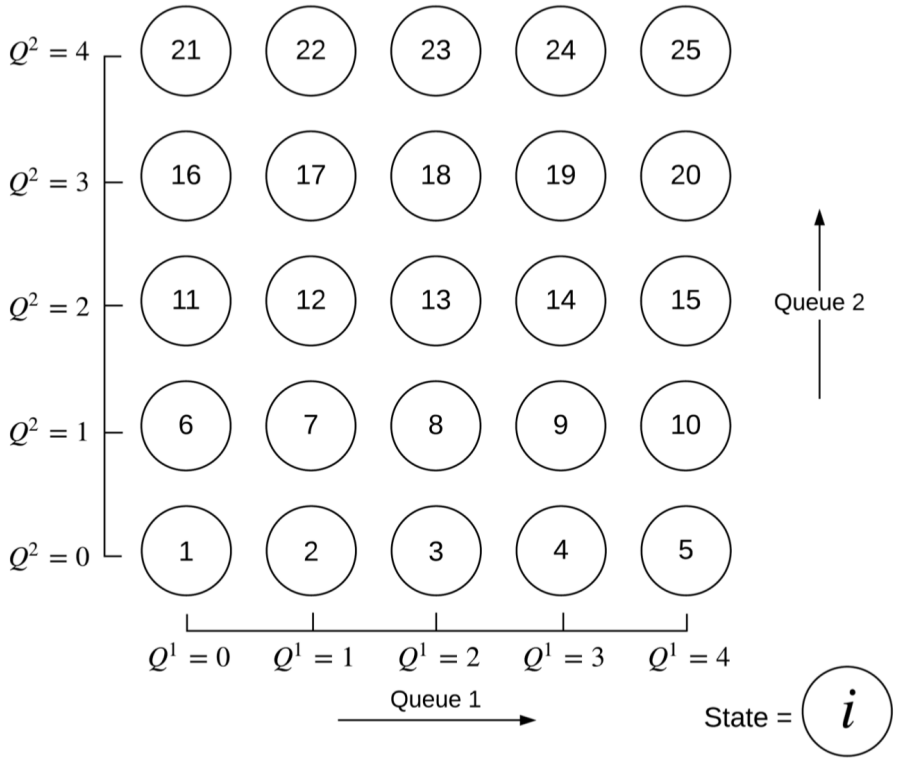

Figure 4.18: *Truncated state space representation.*

The system is described by its states and its state transitions. We assume that both the states and the control remain constant between different state transitions. Transitions from state to state are described by probabilities, for example the probability p_{ij} to go from state i to state j when control u is applied is:

$$p_{ij}(u) = P(x_{k+1} = j | x_k = i, u_k = u) \qquad i, j \in S.Space, u \in U(s). \tag{4.15}$$

The subscript k represents state x and control u at time k. Transition probabilities are computed starting from the transition rates of each state. The probability to go from state i to state j is given by the ratio between the same transition rate and the summation of all the possible transition rates belonging to state i. Transition rates are shown in Figure 4.19.

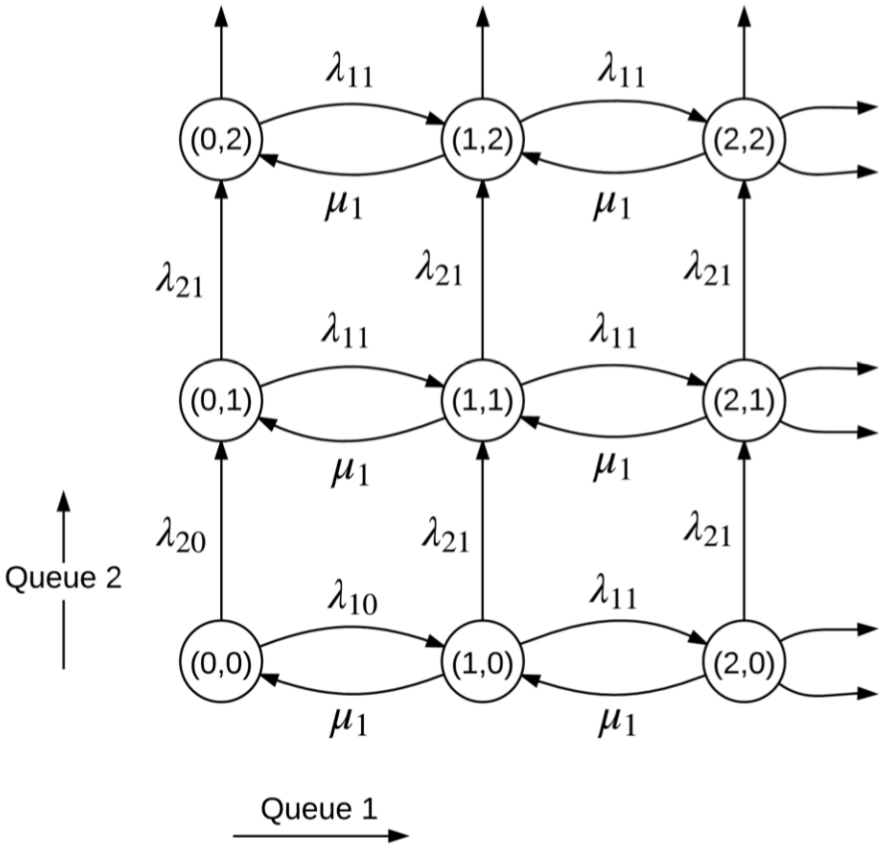

Figure 4.19: *Transition rates when control u_1 is applied.*

The transition rates are not constant because they depend on the states and the control, as already said before we need to make them uniform. This is possible thanks to a simple procedure called uniformization (Bertsekas [13]): the new transition rate ν is set so to be the largest one possible.

$$\nu = max\{\lambda_{11} + \lambda_{21} + \mu_1, \lambda_{22} + \lambda_{12} + \mu_2\}. \tag{4.16}$$

All the probabilities in (4.15) are scaled by the ratio between the former total transition rate $\nu_i(u)$ and the new transition rate in (4.16):

$$\bar{p_{ij}}(u) = \begin{cases} \frac{\nu_i(u)}{\nu} p_{ij} & i \neq j \\ \frac{\nu_i(u)}{\nu} p_{ii} + 1 - \frac{\nu_i(u)}{\nu} & i = j. \end{cases}$$

The conversion creates the possibility to have a fictitious transition from a state to itself. Leaving state i at rate $\nu_i(u)$ in the original process is statistically equivalent to leaving state i at the faster rate ν with a going back rate $\nu - \nu_i(u)$. The new transition rates arrangement is depicted in Figure 4.20.

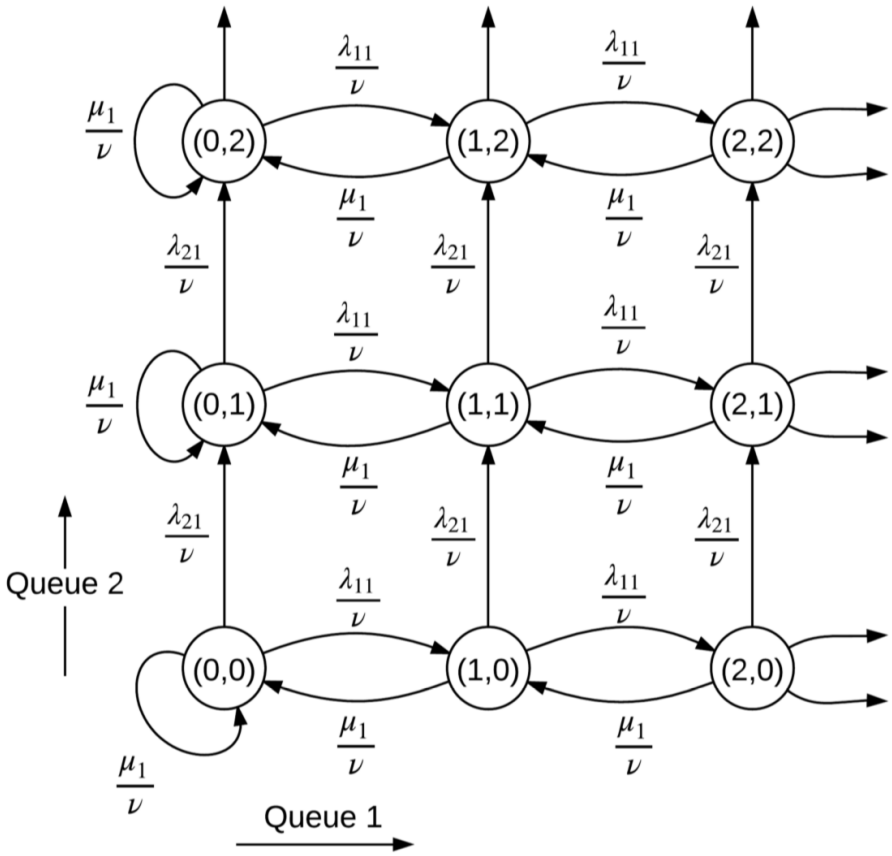

Figure 4.20: *Transition rates after unformization when control u_1 is applied.*

Along with the probabilities also the cost function should be scaled in order to obtain a cost rate per unit time:

$$\bar{g}(i) = \frac{1}{\nu}\left(c_1 Q_1(i) + c_2 Q_2(i)\right).$$

The last step of the discretization process is the discretization of the objective function:

$$J_\pi(x_0) = \limsup_{N\to\infty} \frac{1}{N} E\left[\sum_{k=0}^{N-1} \frac{1}{\nu}\left(c_1 Q_1^{\pi,k} + c_2 Q_2^{\pi,k}\right)\right].$$

$Q_s^{\pi,k}$ represents the queue length of job class s under policy π and after k transitions. The objective is then to find the optimal policy that minimizes the average holding cost. It is convenient if the optimal average cost is the same for all the initial states and the optimal policy is stationary. In this way it is easier to find an optimal policy using numerical methods. A policy is said to be stationary if the same control function is always applied. This happens if the Weak Accessibility (WA) condition holds for the system (Bertsekas [13]). The MDP problem is solved using both linear programming and modified policy iteration algorithm.

4.3.2 Linear programming

The MDP problem can be solved through its linear programming formulation as follows:

$$\text{minimize:} \quad \sum_{i=1}^{n} \sum_{u \in U(i)} y(i,u) g(i,u)$$

$$\text{subject to:} \quad \sum_{u \in U(i)} y(j,u) - \sum_{i=1}^{n} \sum_{u \in U(i)} y(i,u) p_{ij}(u) = 0 \qquad j = 1, \ldots, n$$

$$\sum_{i=1}^{n} \sum_{u \in U(i)} y(i,u) = 1$$

$$y(i,u) \geq 0 \qquad i = 1, \ldots, n \quad u \in U(s).$$

$y(i,u)$ is the long run fraction of time in which the system is in state i and control u is applied (Bello and Riano [14]). They represent basically the steady-state probabilities and then they are independent of the initial state. All the $y(i,u)$ are then normalized:

$$f(i,u) = \frac{y(i,u)}{\sum_{u \in U(i)} y(i,u)} \qquad i = 1, \ldots, n \quad u \in U(s).$$

There exists only one optimal solution, for each state then only one control u can be applied and thus its value is equal to 1 after normalization; the other control must be equal to 0. The set of $f(i,u)$ for all states forms the optimal stationary policy π.

Results

We want to check if the results of the linear programming match what was obtained with the fluid model. The four different zones are analyzed.

- The *Class* 1 zone is where both $c\mu$ rule and (4.13) choose class 1 as optimal. The results obtained with the dual simplex algorithm are depicted in Figure 4.21. Where feasible, the results overlap with the fluid model (and in general with the $c\mu$ rule).

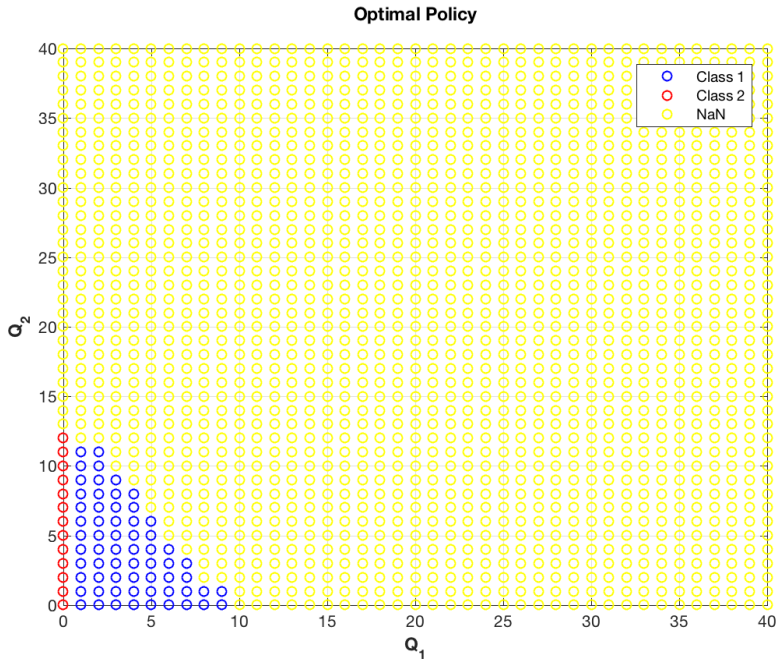

Figure 4.21: *Linear programming results for Class* 1 *zone, the parameters used are:* $\lambda_{11} = 23, \lambda_{12} = 12, \lambda_{21} = 6, \lambda_{22} = 13, \mu_1 = 100, \mu_2 = 147, c_1 = 1.6, c_2 = 1, N = 40$.

- The *Class* 2 zone is where both $c\mu$ rule and (4.13) choose class 2 as optimal, the results obtained with the dual simplex algorithm are depicted in Figure 4.22. Most of the available states follow what is predicted by the fluid model, however some points are not behaving as predicted.

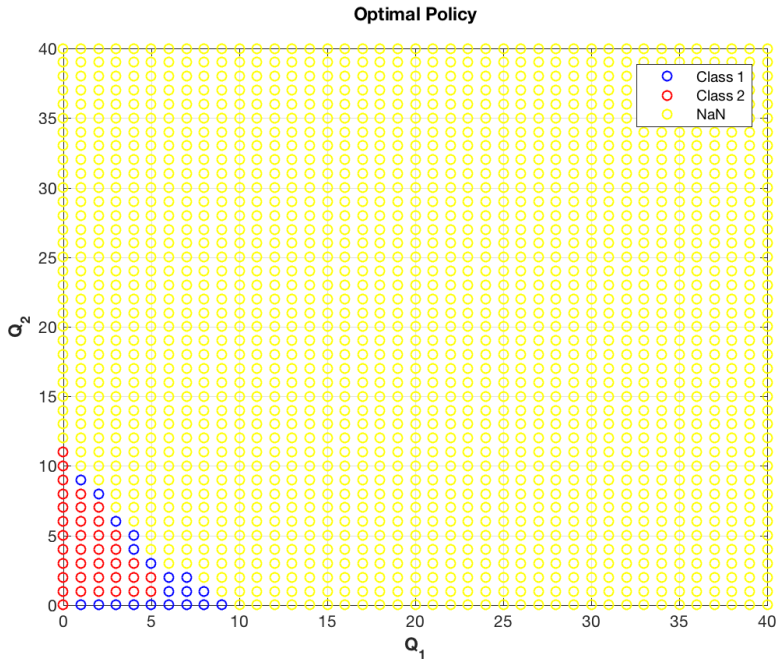

Figure 4.22: *Linear programming results for Class* 2 *zone, the parameters used are:* $\lambda_{11} = 23, \lambda_{12} = 12, \lambda_{21} = 6, \lambda_{22} = 13, \mu_1 = 100, \mu_2 = 147, c_1 = 1.4, c_2 = 1, N = 40$.

- The *Reverse* 2 zone results show in Figure 4.23 a small region where the states prioritize class 2 instead of class 1, however the results are inconclusive since class 1 presence is not negligible.

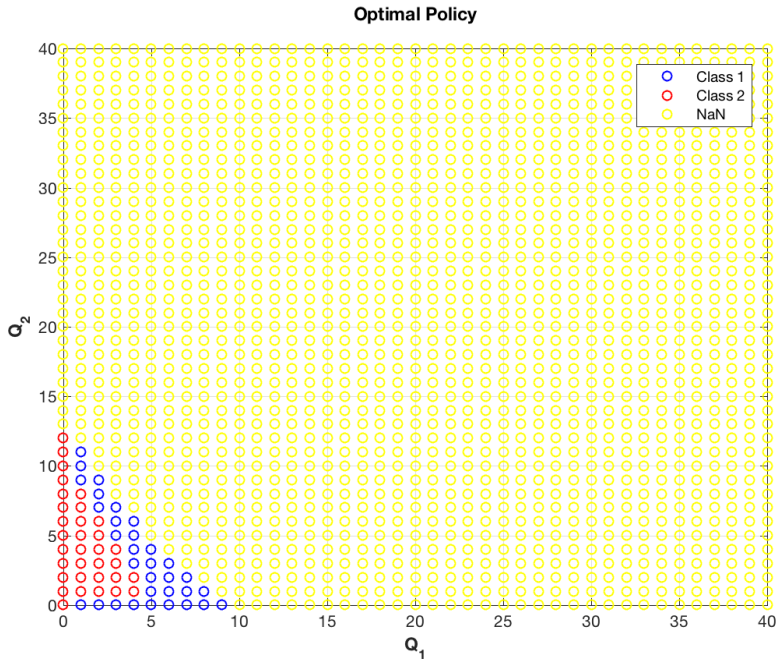

Figure 4.23: *Linear programming results for Reverse* 2 *zone, the parameters used are:* $\lambda_{11} = 23, \lambda_{12} = 12, \lambda_{21} = 6, \lambda_{22} = 13, \mu_1 = 100, \mu_2 = 147, c_1 = 1.53, c_2 = 1, N = 40$.

- The *Reverse* 1 zone results show an improvement with respect to the other results but still not enough to be considered reasonable and comparable to the fluid model results (see Figure 4.24).

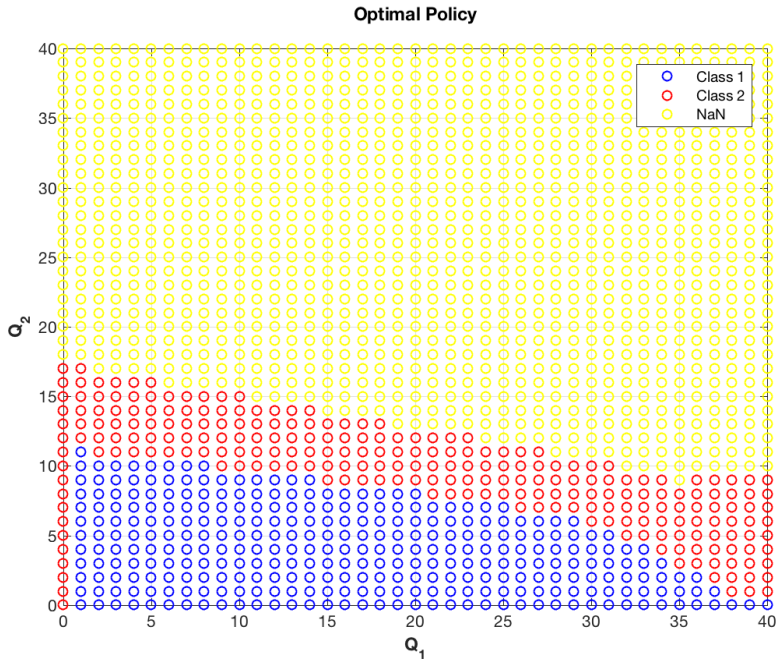

Figure 4.24: *Linear programming results for Reverse* 1 *zone, the parameters used are:* $\lambda_{11} = 23, \lambda_{12} = 12, \lambda_{21} = 6, \lambda_{22} = 13, \mu_1 = 100, \mu_2 = 28, c_1 = 0.25, c_2 = 1, N = 40.$

In all the zones most of the state space is characterized by NaN points, this happens if $y(i,u), \forall u_i$ of a state is null, this leads to a division by zero during the normalization to compute $f(i,u)$, causing an error. We know that if the Markov chain is irreducible all the probabilities are different from zero, but they can be very small, even smaller than the default tolerance of the software, and this is our case. Some probabilities are then considered null because they exceed the tolerances. Truncation is another major contribution to the non-reliability of the results, the state space should be unbounded but, for obvious practical reasons, it is truncated. This causes the results to be affected by strong boundary effects.

4.3.3 Policy Iteration

Policy iteration is a very useful method that can solve problems with a large number of states quickly. Starting from an initial guess (starting policy), at every iteration it computes a new stationary policy that improves the objective function. When two consecutive policies are equal the algorithm stops and the optimal policy is achieved. Usually the convergence happens in very few iterations (Bertsekas [13]).

Optimality

The objective is to minimize the total average cost rate. It can be found using a value iteration algorithm, that chooses the best control over all controls based on the one step costs and all expected future costs. There exist an optimal value that is a fixed point of the procedure. The optimality satisfies the following relationship:

$$\Omega + h(i) = \min_{u \in U(i)} \left[g(i) + \sum_{j=1}^{n} p_{ij}(u) h^k(j) \right] \qquad i = 1, \ldots, n. \tag{4.17}$$

The optimality relationship is known as Bellman's equation, where $h(i)$ is the minimum cost among all the possible policies to reach state n from state $h(i)$ and Ω is the optimal cost per stage. (4.17) states that the left hand side remains the same if for all the state i the control u minimizing the right hand side is applied.

Step 1: Initialization

First stationary policy π^0 has to be guessed, usually it is called greedy policy because it does not account for the future. In our case we choose the standard $c\mu$ rule and the *non-empty product policy* when one of the queues is empty, the second policy is added to meet the Weak Accessibility requirements.

Step 2: New policy

For each iteration step k the cost function corresponding to the stationary policy π^k is evaluated. The cost function is divided in the average cost per stage Ω^k and differential cost $h^k(i)$ (or cost to go). The costs should satisfy the Bellman's equation:

$$\Omega^k + h^k(i) = g(i) + \sum_{j=1}^{n} p_{ij}(\pi^k(i)) h^k(j) \qquad i = 1, \ldots, n. \tag{4.18}$$

(4.18) represents a linear system with n equations and $n+1$ unknowns $(h^k(1), \ldots, h^k(n), \Omega^k)$, to solve it we assigned a value to an arbitrary degree of freedom, for example $h^k(1) = 0$.

Step 3: Policy improvement

A new stationary policy is computed that should satisfy:

$$g(i) + \sum_{j=1}^{n} p_{ij}(\pi^{k+1}(i)) h^k(j) = \min_{u \in U(i)} \left[g(i) + \sum_{j=1}^{n} p_{ij}(u) h^k(j) \right] \qquad i = 1, \ldots, n.$$

If π^{k+1} is equal to π^k the optimal policy is achieved and the algorithm stops, otherwise step 2 is computed again updating the current policy. As already said the method is known to converge very quickly (few iterations).

Value iteration approximation

When the state space is very large the computation in step 2 is very time consuming, since at each step a linear system with state space dimension is solved. In these cases instead of solving the entire system is convenient to approximate the procedure iteratively by using a different method called *Relative value iteration.* We define a new set of iterations l for the approximation of h:

$$h^{l+1}(i) = g(i) + \sum_{j=1}^{n} p_{ij}(\pi^k(i)) h^l(j) \qquad i = 1, \ldots, n.$$

As $l \to \infty$ h^l converges to h^k, however in most of the cases just few iterations are required for the convergence.

Results

We now compare the results of MPI algorithm with the ones obtained with the simple linear program. The parameters chosen are the same.

- *TheClass* 1 zone is depicted in Figure 4.25, class 1 is always chosen as optimal in accordance with what both the rules predict. On the right side there is a small zone characterized by class 2 priority, the results in these state can be reasonably affected by boundary effects due to truncation.

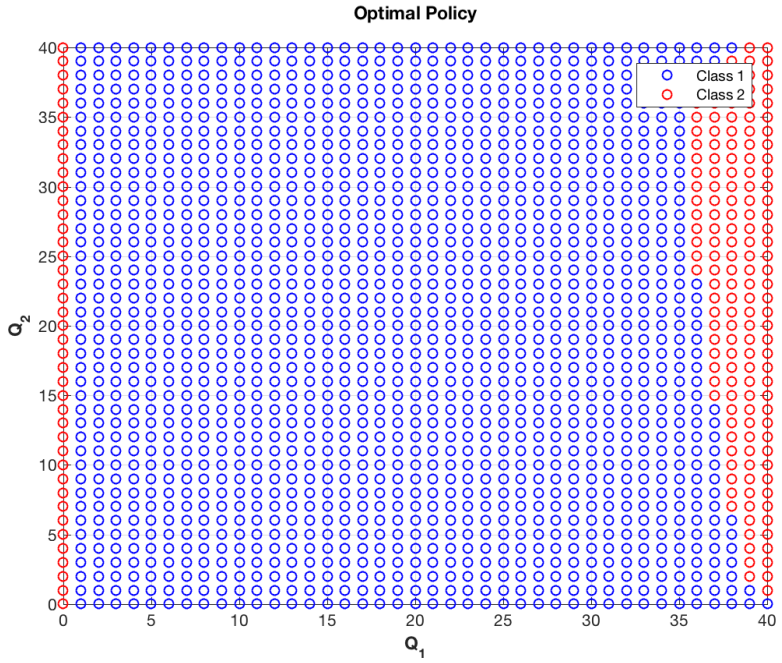

Figure 4.25: *Policy iteration results for Class* 1 *zone, the parameters used are:* $\lambda_{11} = 23, \lambda_{12} = 12, \lambda_{21} = 6, \lambda_{22} = 13, \mu_1 = 100, \mu_2 = 147, c_1 = 1.6, c_2 = 1, N = 40$.

- *TheClass* 2 zone is represented in Figure 4.26, the same considerations made for *Class*1 zone hold in this case, even if with this set of parameters boundary effects are not so apparent.

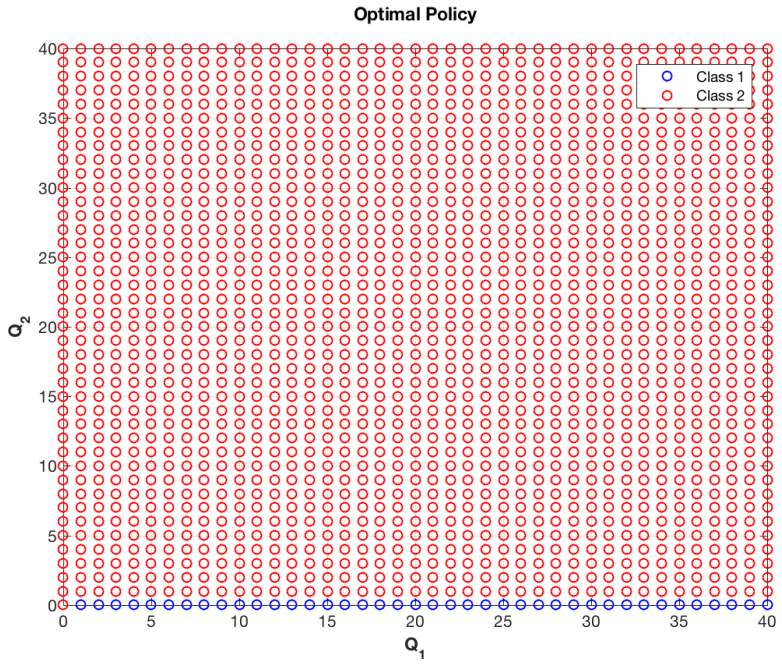

Figure 4.26: *Policy iteration results for Class 2 zone, the parameters used are:* $\lambda_{11} = 23, \lambda_{12} = 12, \lambda_{21} = 6, \lambda_{22} = 13, \mu_1 = 100, \mu_2 = 147, c_1 = 1.4, c_2 = 1, N = 40$.

- *TheReverse* 2 zone is depicted in Figure 4.27, the results confirm that class 2 is the optimal one, even if $c\mu$ rule states the opposite. Boundary effects are almost negligible.

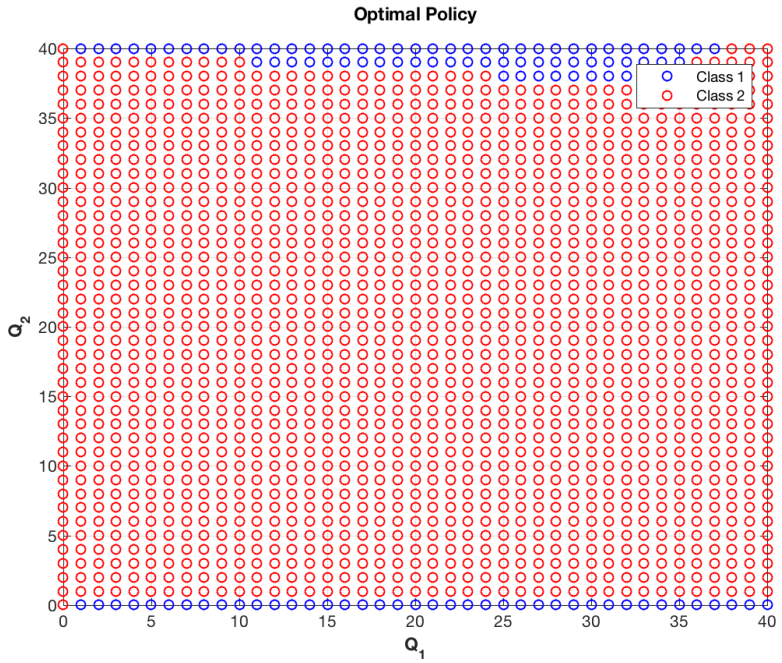

Figure 4.27: *Policy iteration results for Reverse* 2 *zone, the parameters used are:* $\lambda_{11} = 23, \lambda_{12} = 12, \lambda_{21} = 6, \lambda_{22} = 13, \mu_1 = 100, \mu_2 = 147, c_1 = 1.53, c_2 = 1, N = 40$.

- *TheReverse* 1 zone is depicted in Figure 4.28, this is the hardest zone to predict. Class 1 and class 2 have more or less the same occurrence so it is difficult to understand which one should be the correct one. Our first guess is that the boundary effects are strongly affecting the final results because of the strange pattern of the states which do not make sense.

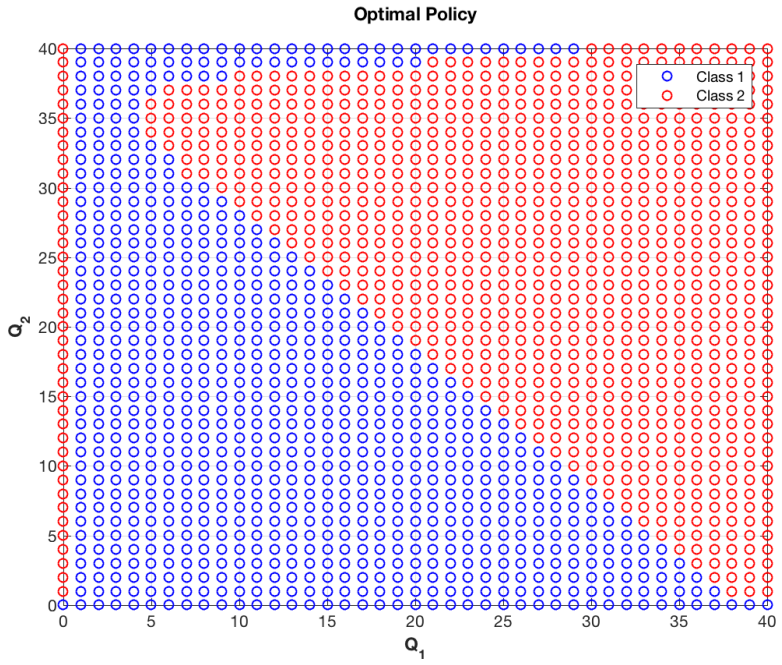

Figure 4.28: *Policy iteration results for Reverse* 1 *zone, the parameters used are:* $\lambda_{11} = 23, \lambda_{12} = 12, \lambda_{21} = 6, \lambda_{22} = 13, \mu_1 = 100, \mu_2 = 28, c_1 = 0.25, c_2 = 1, N = 40$.

We want to focus on *Reverse* 1 zone to understand what is the right behavior we should expect. We presente few cases with different parameters.

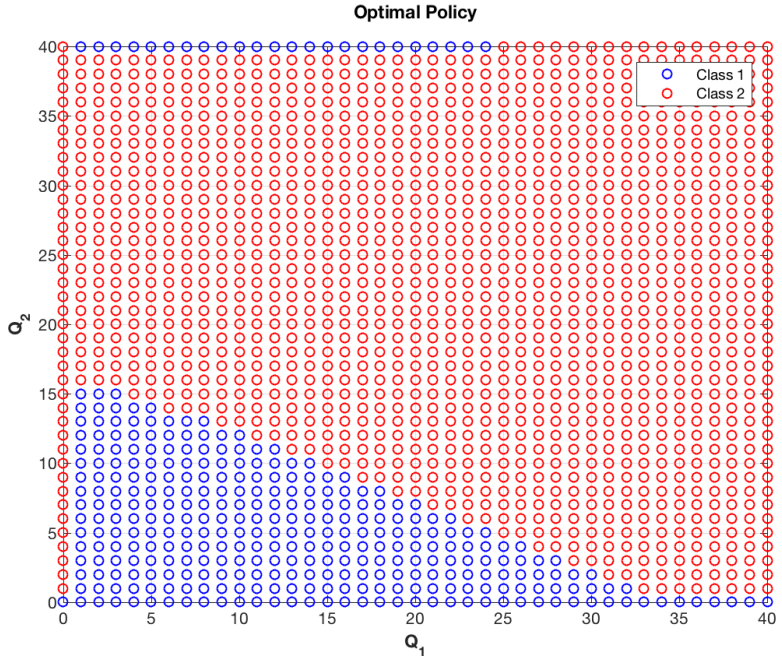

Figure 4.29: *Focus on Reverse* 1 *zone for different sets of parameters, the parameters used are:* $\lambda_{11} = 11, \lambda_{12} = 5, \lambda_{21} = 3, \lambda_{22} = 4, \mu_1 = 70, \mu_2 = 8.4, c_1 = 0.118, c_2 = 1, N = 40$.

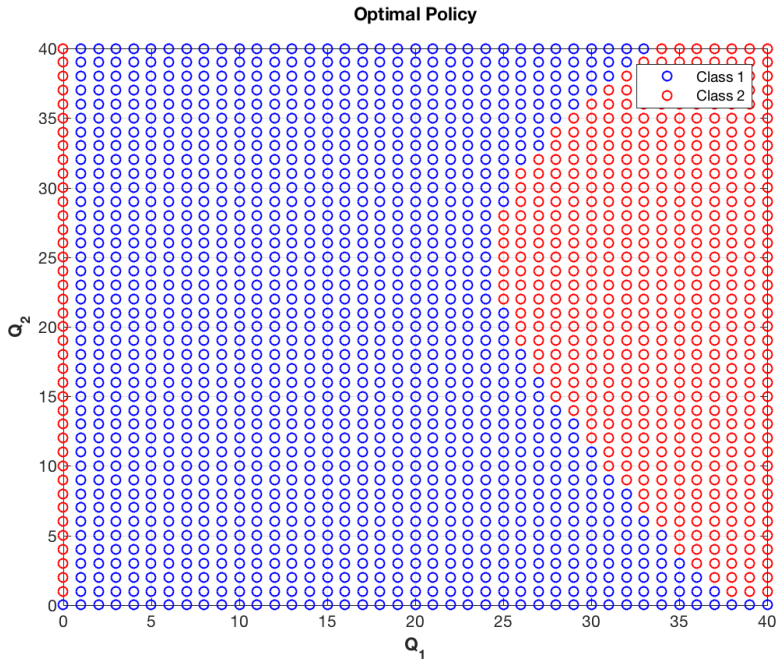

Figure 4.30: *Focus on Reverse* 1 *zone for different sets of parameters, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 6, \lambda_{21} = 3, \lambda_{22} = 14, \mu_1 = 18, \mu_2 = 24, c_1 = 1.1, c_2 = 1, N = 40$.

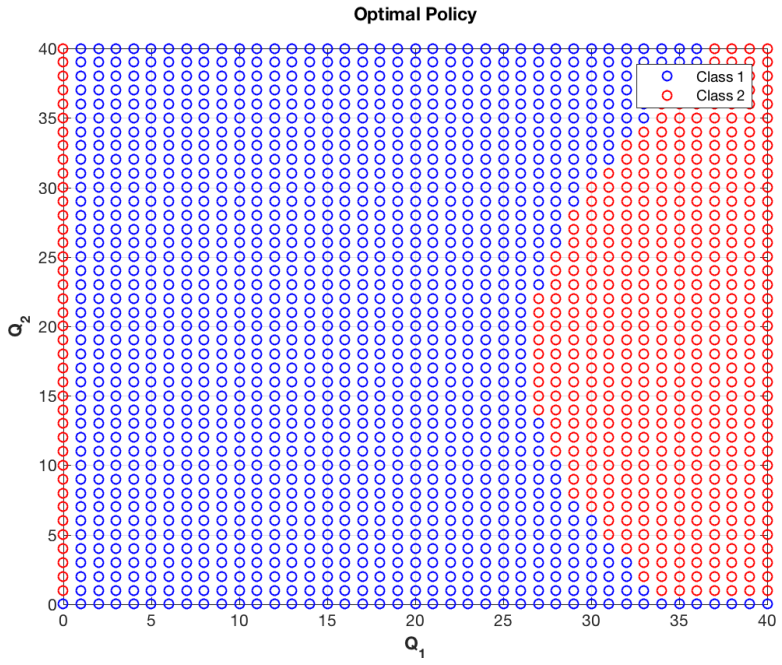

Figure 4.31: *Focus on Reverse* 1 *zone for different sets of parameters, the parameters used are:* $\lambda_{11} = 21, \lambda_{12} = 10, \lambda_{21} = 7, \lambda_{22} = 24, \mu_1 = 30, \mu_2 = 37, c_1 = 1.2, c_2 = 1, N = 40$.

In Figure 4.29 is still not clear enough because it shows a similar behavior to Figure 4.28. Figure 4.30 and Figure 4.31 instead show a strong class 1 presence while class 2 is

probably due to boundary effects. We now change the state space truncation to see if the results improve, the reference case is shown in Figure 4.31.

- Truncation at $N = 20$

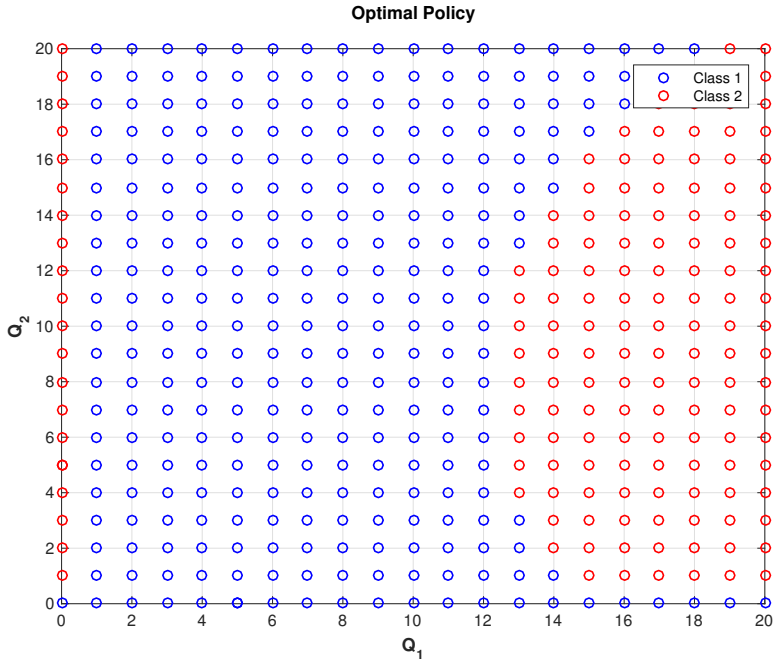

Figure 4.32: *Boundary effects investigation for* $N = 20$*, the parameters used are:* $\lambda_{11} = 11, \lambda_{12} = 5, \lambda_{21} = 3, \lambda_{22} = 4, \mu_1 = 70, \mu_2 = 8.4, c_1 = 0.118, c_2 = 1, N = 20$.

- Truncation at $N = 30$

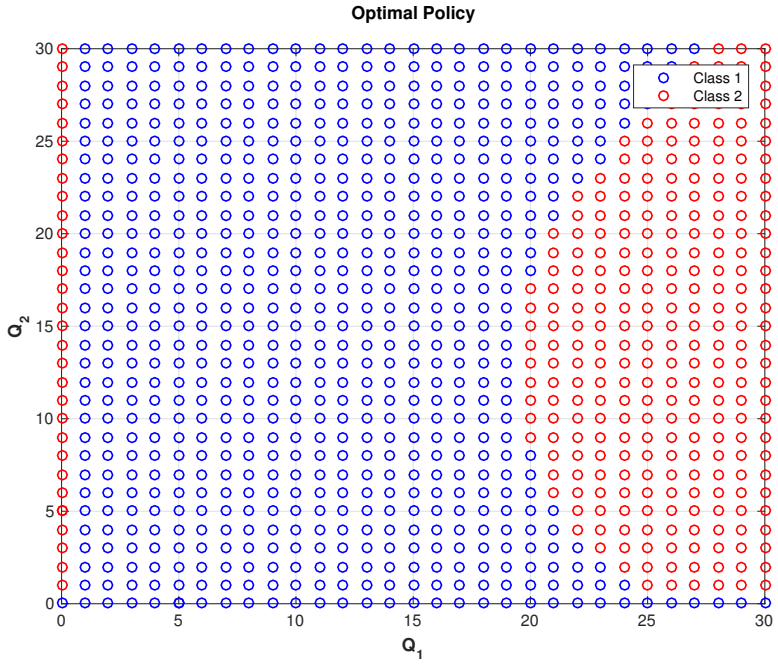

Figure 4.33: *Boundary effects investigation for* $N = 30$*, the parameters used are:* $\lambda_{11} = 11, \lambda_{12} = 5, \lambda_{21} = 3, \lambda_{22} = 4, \mu_1 = 70, \mu_2 = 8.4, c_1 = 0.118, c_2 = 1, N = 30$.

- Truncation at $N = 50$

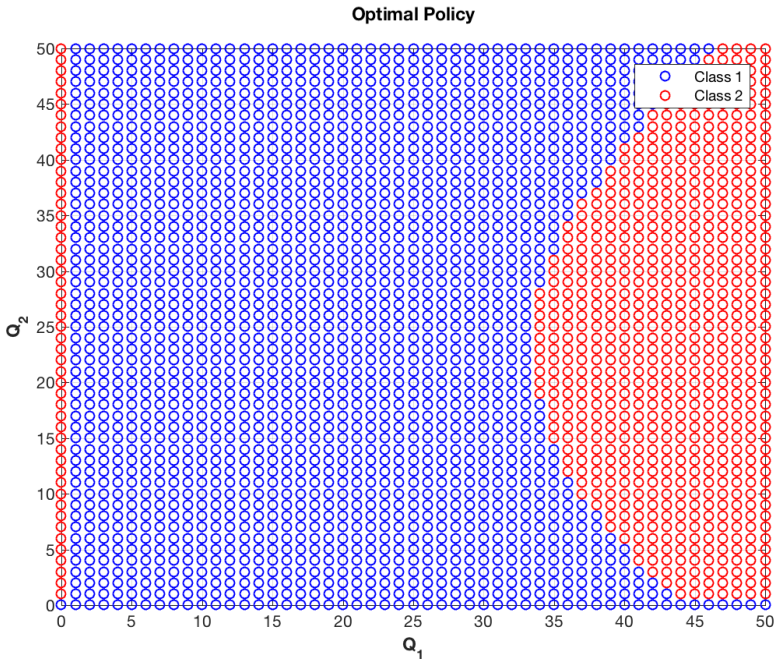

Figure 4.34: *Boundary effects investigation for* $N = 50$*, the parameters used are:* $\lambda_{11} = 11, \lambda_{12} = 5, \lambda_{21} = 3, \lambda_{22} = 4, \mu_1 = 70, \mu_2 = 8.4, c_1 = 0.118, c_2 = 1, N = 50$.

As the level of truncation increases the region of class 2 moves to the right, so some

states that with a low level of truncation were showing class 2 as optimal, with a higher level of truncation prioritize class 1. We can then reasonably assume that the presence of class 2 is due to boundary effects because the true optimal policy is independent of the truncation level. One final remark that could confirm our assumption is that *Reverse* 1 zone is very close to the stability line ($\rho = 1$), the system then is more likely to be in the limit states of the state space leading to very strong boundary effects.

4.4 Analytical Markov decision process

In this section we want to prove analytically the optimality of the policy characterized by (4.13) obtained thanks to approximated methods. The model is still the Markov decision process defined in section 4.3. The goal is to minimize the total average cost rate (4.14). However for structural results purposes we consider the discounted cost case:

$$\limsup_{t\to\infty} \frac{1}{t} E_{(i,j,u)} \left[\int_0^t e^{-\beta s} (c_1 Q_1^\pi(s) + c_2 Q_2^\pi(s)) ds \right],$$

where β is a positive discount factor.

4.4.1 Mapping

As usual starting from the continuous time Markov decision process we derive the corresponding discrete Markov decision process. Since the problem is characterized by transition rates dependent on the state and the control it is necessary to define also a uniform transition rate $\nu = \mu_{max} + \lambda_{max}$ (uniformization process). We define a real valued function f representing the value of each state ($f(i, j, u, t)$ is the value of state i, j given the control u at time instant t). We also define the mapping H of the function as follows (time index t is dropped)(Sisbot and Hasenbein [10]):

$$
\begin{aligned}
H_\delta f(i,j,u) = \delta min &\begin{cases} \lambda_{11} f(i+1,j,u_1) + (\lambda_{max} - \lambda_{11}) f(i,j,u_1) \\ \lambda_{11} f(i+1,j,u_2) + (\lambda_{max} - \lambda_{11}) f(i,j,u_1) \end{cases} \\
+ \delta min &\begin{cases} \lambda_{21} f(i,j+1,u_1) + (\lambda_{max} - \lambda_{21}) f(i,j,u_1) \\ \lambda_{21} f(i,j+1,u_2) + (\lambda_{max} - \lambda_{21}) f(i,j,u_1) \end{cases} \\
+ \delta min &\begin{cases} \mu_1 f(i-1,j,u_1) + (\mu_{max} - \mu_1) f(i,j,u_1) \\ \mu_1 f(i-1,j,u_2) + (\mu_{max} - \mu_1) f(i,j,u_1) \qquad if\ u = u_1 \end{cases} \\
= \delta min &\begin{cases} \lambda_{22} f(i,j+1,u_1) + (\lambda_{max} - \lambda_{22}) f(i,j,u_2) \\ \lambda_{22} f(i,j+1,u_2) + (\lambda_{max} - \lambda_{22}) f(i,j,u_2) \end{cases} \\
+ \delta min &\begin{cases} \lambda_{12} f(i+1,j,u_1) + (\lambda_{max} - \lambda_{12}) f(i,j,u_2) \\ \lambda_{12} f(i+1,j,u_2) + (\lambda_{max} - \lambda_{12}) f(i,j,u_2) \end{cases} \\
+ \delta min &\begin{cases} \mu_2 f(i,j-1,u_1) + (\mu_{max} - \mu_2) f(i,j,u_2) \\ \mu_2 f(i,j-1,u_2) + (\mu_{max} - \mu_2) f(i,j,u_2) \qquad if\ u = u_2, \end{cases}
\end{aligned}
\tag{4.19}
$$

where $\delta = \frac{\nu}{\nu+\beta}$ is the effective discount factor after the uniformization.

4.4.2 Results

We show results for the discounted cost finite horizon problem, then using standard techniques it is possible to prove the same for the infinite horizon average cost case. If there exists an optimal scheduling policy for every single state then Bellman's equation should be satisfied for every single state. In the discounted cost finite horizon problem (4.17) is formulated as follows:

$$
f_\delta^{n+1}(i,j,u) = c_1 i + c_2 j + H_\delta f_\delta^n(i,j,u). \tag{4.20}
$$

The important result is that (4.13) is the optimal scheduling policy, so if it holds it is always optimal to serve class 1 rather than class 2 (if class 1 queue is not empty),

$$
[\mu_1 - (\lambda_{11} - \lambda_{12})] c_1 \geq [\mu_2 - (\lambda_{22} - \lambda_{21})] c_2.
$$

The proof is carried out by induction: we prove the relationships for $n = 0$ and then assuming that the relationships hold for n we prove they hold also for $n + 1$. Since we want to prove class 1 is optimal we assume:

$$
f_\delta^n(i,j,u_1) \leq f_\delta^n(i,j,u_2). \tag{4.21}
$$

Lemma 4.4.1. *$f^n_\delta(i+1,j,u_1) - f^n_\delta(i,j,u_1) \geq 0$ for each fixed j ($f^n_\delta(i,j+1,u_1) - f^n_\delta(i,j,u_1) \geq 0$ for each fixed i), $\forall i \geq 1, \forall j \geq 1$.*

Proof:

- $n = 0$:

$$c_1(i+1) + c_2 j - c_1 i - c_2 j \geq 0$$
$$c_1 \geq 0. \qquad \checkmark$$

- $n+1$: According to (4.20) and (4.19) we expand the terms, using (4.21) we simplify the minima, then we drop the δ terms:

$$\lambda_{11} f(i+2,j,u_1) + (\lambda_{max} - \lambda_{11}) f(i+1,j,u_1) + \lambda_{21} f(i+1,j+1,u_1)$$

$$+(\lambda_{max} - \lambda_{21}) f(i+1,j,u_1) + \mu_1 f(i,j,u_1) + (\mu_{max} - \mu_1) f(i+1,j,u_1)$$

$$+c_1(i+1) + c_2 j$$

$$-\lambda_{11} f(i+1,j,u_1) - (\lambda_{max} - \lambda_{11}) f(i,j,u_1) - \lambda_{21} f(i,j+1,u_1)$$

$$-(\lambda_{max} - \lambda_{21}) f(i,j,u_1) - \mu_1 f(i-1,j,u_1) - (\mu_{max} - \mu_1) f(i,j,u_1)$$

$$-c_1 i - c_2 j \geq 0.$$

Simplifying and collecting the same terms we obtain:

$$\lambda_{11}[f(i+2,j,u_1) - f(i+1,j,u_1)] + (\lambda_{max} - \lambda_{11})[f(i+1,j,u_1) - f(i,j,u_1)]$$

$$+\lambda_{21}[f(i+1,j+1,u_1) - f(i,j+1,u_1)] + (\lambda_{max} - \lambda_{21})[f(i+1,j,u_1) - f(i,j,u_1)]$$

$$+\mu_1[f(i,j,u_1) - f(i-1,j,u_1)] + (\mu_{max} - \mu_1)[f(i+1,j,u_1) - f(i,j,u_1)] + c_1 \geq 0. \qquad \checkmark$$

Each difference inside the brackets is actually what we assumed for n, so the relationship is then proven.

Lemma 4.4.2. *$f^n_\delta(i+1,j,u_1) - f^n_\delta(i,j,u_1) = f^n_\delta(i,j,u_1) - f^n_\delta(i-1,j,u_1)$ for each fixed j (the same applies for j for each fixed i), $\forall i \geq 1, \forall j \geq 1$.*

Proof:

- $n = 0$:

$$c_1(i+1) + c_2 j - c_1 i - c_2 j = c_1 i + c_2 j - c_1(i-1) - c_2 j$$
$$c_1 = c_1. \qquad \checkmark$$

- $n+1$: According to (4.20) and (4.19) we expand the terms, using (4.21) we simplify the minima, then we drop the δ terms:

$$\lambda_{11} f_\delta^n(i+2,j,u_1) + (\lambda_{max} - \lambda_{11}) f_\delta^n(i+1,j,u_1) + \lambda_{21} f_\delta^n(i+1,j+1,u_1)$$

$$+(\lambda_{max} - \lambda_{21}) f_\delta^n(i+1,j,u_1) + \mu_1 f_\delta^n(i,j,u_1) + (\mu_{max} - \mu_1) f_\delta^n(i+1,j,u_1)$$

$$-\lambda_{11} f_\delta^n(i+1,j,u_1) - (\lambda_{max} - \lambda_{11}) f_\delta^n(i,j,u_1) - \lambda_{21} f_\delta^n(i,j+1,u_1)$$

$$-(\lambda_{max} - \lambda_{21}) f_\delta^n(i,j,u_1) - \mu_1 f_\delta^n(i-1,j,u_1) - (\mu_{max} - \mu_1) f_\delta^n(i,j,u_1)$$

$$+c_1(i+1) + c_2 j - c_1 i - c_2 j =$$

$$\lambda_{11} f_\delta^n(i+1,j,u_1) + (\lambda_{max} - \lambda_{11}) f_\delta^n(i,j,u_1) + \lambda_{21} f_\delta^n(i,j+1,u_1)$$

$$+(\lambda_{max} - \lambda_{21}) f_\delta^n(i,j,u_1) + \mu_1 f_\delta^n(i-1,j,u_1) + (\mu_{max} - \mu_1) f_\delta^n(i,j,u_1)$$

$$-\lambda_{11} f_\delta^n(i,j,u_1) - (\lambda_{max} - \lambda_{11}) f_\delta^n(i-1,j,u_1) - \lambda_{21} f_\delta^n(i-1,j+1,u_1)$$

$$-(\lambda_{max} - \lambda_{21}) f_\delta^n(i-1,j,u_1) - \mu_1 f_\delta^n(i-2,j,u_1) - (\mu_{max} - \mu_1) f_\delta^n(i-2,j,u_1)$$

$$+c_1 i + c_2 j - c_1(i-1) - c_2 j.$$

Simplifying and collecting the same terms we obtain:

$$\lambda_{11}[f_\delta^n(i+2,j,u_1) - f_\delta^n(i+1,j,u_1) - f_\delta^n(i+1,j,u_1) + f_\delta^n(i,j,u_1)]$$

$$+(\lambda_{max} - \lambda_{11})[f_\delta^n(i+1,j,u_1) - f_\delta^n(i,j,u_1) - f_\delta^n(i,j,u_1) + f_\delta^n(i-1,j,u_1)]$$

$$+\lambda_{21}[f_\delta^n(i+1,j+1,u_1) - f_\delta^n(i,j+1,u_1) - f_\delta^n(i,j+1,u_1) + f_\delta^n(i-1,j+1,u_1)]$$

$$+(\lambda_{max} - \lambda_{21})[f_\delta^n(i+1,j,u_1) - f_\delta^n(i,j,u_1) - f_\delta^n(i,j,u_1) + f_\delta^n(i-1,j,u_1)]$$

$$+\mu_1[f_\delta^n(i,j,u_1) - f_\delta^n(i-1,j,u_1) - f_\delta^n(i-1,j,u_1) + f_\delta^n(i-2,j,u_1)]$$

$$+(\mu_{max} - \mu_1)[f_\delta^n(i+1,j,u_1) - f_\delta^n(i,j,u_1) - f_\delta^n(i,j,u_1) + f_\delta^n(i-1,j,u_1)] = 0.$$

✓

Each difference inside the brackets is actually what we assumed for n, so the relationship is then proven.

For each j fixed, the difference of $f_\delta^n(i,j,u_1)$ induced by two adjacent states of i is constant (the same applies for j when i is fixed).

Lemma 4.4.3. *$f^n_\delta(i+1,j+1,u_1) - f^n_\delta(i,j+1,u_1) = f^n_\delta(i+1,j,u_1) - f^n_\delta(i,j,u_1)$ for each fixed difference in i and for each fixed j, in the same side of the equivalence(the same applies for j when its difference is fixed), $\forall i \geq 1, \forall j \geq 1$.*

Proof:

- $n = 0$:

$$c_1(i+1) + c_2(j+1) - c_1 i - c_2(j+1) = c_1(i+1) + c_2 j - c_1 i - c_2 j$$
$$c_1 = c_1. \qquad \checkmark$$

- $n+1$: According to (4.20) and (4.19) we expand the terms, using (4.21) we simplify the minima, then we drop the δ terms:

$$\lambda_{11} f^n_\delta(i+2,j+1,u_1) + (\lambda_{max} - \lambda_{11}) f^n_\delta(i+1,j+1,u_1) + \lambda_{21} f^n_\delta(i+1,j+2,u_1)$$

$$+(\lambda_{max} - \lambda_{21}) f^n_\delta(i+1,j+1,u_1) + \mu_1 f^n_\delta(i,j+1,u_1) + (\mu_{max} - \mu_1) f^n_\delta(i+1,j+1,u_1)$$

$$-\lambda_{11} f^n_\delta(i+1,j+1,u_1) - (\lambda_{max} - \lambda_{11}) f^n_\delta(i,j+1,u_1) - \lambda_{21} f^n_\delta(i,j+2,u_1)$$

$$-(\lambda_{max} - \lambda_{21}) f^n_\delta(i,j+1,u_1) - \mu_1 f^n_\delta(i-1,j+1,u_1) - (\mu_{max} - \mu_1) f^n_\delta(i,j+1,u_1)$$

$$+c_1(i+1) + c_2(j+1) - c_1 i - c_2(j+1) =$$

$$\lambda_{11} f^n_\delta(i+2,j,u_1) + (\lambda_{max} - \lambda_{11}) f^n_\delta(i+1,j,u_1) + \lambda_{21} f^n_\delta(i+1,j+1,u_1)$$

$$+(\lambda_{max} - \lambda_{21}) f^n_\delta(i+1,j,u_1) + \mu_1 f^n_\delta(i,j,u_1) + (\mu_{max} - \mu_1) f^n_\delta(i+1,j,u_1)$$

$$-\lambda_{11} f^n_\delta(i+1,j,u_1) - (\lambda_{max} - \lambda_{11}) f^n_\delta(i,j,u_1) - \lambda_{21} f^n_\delta(i,j+1,u_1)$$

$$-(\lambda_{max} - \lambda_{21}) f^n_\delta(i,j,u_1) - \mu_1 f^n_\delta(i-1,j,u_1) - (\mu_{max} - \mu_1) f^n_\delta(i,j,u_1)$$

$$+c_1(i+1) + c_2 j - c_1 i - c_2 j.$$

Simplifying and collecting the same terms we obtain:

$$\lambda_{11}[f_\delta^n(i+2,j+1,u_1) - f_\delta^n(i+1,j+1,u_1) - f_\delta^n(i+2,j,u_1) + f_\delta^n(i+1,j,u_1)]$$

$$+(\lambda_{max} - \lambda_{11})[f_\delta^n(i+1,j+1,u_1) - f_\delta^n(i,j+1,u_1) - f_\delta^n(i+1,j,u_1) + f_\delta^n(i,j,u_1)]$$

$$+\lambda_{21}[f_\delta^n(i+1,j+2,u_1) - f_\delta^n(i,j+2,u_1) - f_\delta^n(i+1,j+1,u_1) + f_\delta^n(i,j+1,u_1)]$$

$$+(\lambda_{max} - \lambda_{21})[f_\delta^n(i+1,j+1,u_1) - f_\delta^n(i,j+1,u_1) - f_\delta^n(i+1,j,u_1) + f_\delta^n(i,j,u_1)]$$

$$+\mu_1[f_\delta^n(i,j+1,u_1) - f_\delta^n(i-1,j+1,u_1) - f_\delta^n(i,j,u_1) + f_\delta^n(i-1,j,u_1)]$$

$$+(\mu_{max} - \mu_1)[f_\delta^n(i+1,j+1,u_1) - f_\delta^n(i,j+1,u_1) - f_\delta^n(i+1,j,u_1) + f_\delta^n(i,j,u_1)] = 0.$$

✓

Each difference inside the brackets is actually what we assumed for n, so the relationship is then proven.

Lemma 4.4.3 is a kind of step forward with respect to Lemma 4.4.2. The difference of $f_\delta^n(i,j,u_1)$ induced by two adjacent states of i is constant even in the case of j is not fixed, j however must be equal in the two adjacent states (the same applies for adjacent states of j).

Lemma 4.4.4. $\frac{f_\delta^n(i,j+1,u_1) - f_\delta^n(i,j,u_1)}{f_\delta^n(i+1,j,u_1) - f_\delta^n(i,j,u_1)} = \frac{c_2}{c_1}$, $\forall i \geq 1, \forall j \geq 1$.

Proof:

- $n = 0$:

$$\frac{c_1 i + c_2(j+1) - c_1 - c_2 j}{c_1(i+1) + c_2 j - c_1 i - c_2 j} = \frac{c_2}{c_1}$$
$$\frac{c_2}{c_1} = \frac{c_2}{c_1}. \qquad \checkmark$$

- $n+1$: According to (4.20) and (4.19) we expand the terms, using (4.21) we simplify the minima, then we drop the δ terms. For the sake of clarity and ease we divide the relationship in numerator and denominator:

$$N = \lambda_{11} f_\delta^n(i+1,j+1,u_1) + (\lambda_{max} - \lambda_{11}) f_\delta^n(i,j+1,u_1) + \lambda_{21} f_\delta^n(i,j+2,u_1)$$

$$+ (\lambda_{max} - \lambda_{21}) f_\delta^n(i,j+1,u_1) + \mu_1 f_\delta^n(i-1,j+1,u_1) + (\mu_{max} - \mu_1) f_\delta^n(i,j+1,u_1)$$

$$- \lambda_{11} f_\delta^n(i+1,j,u_1) - (\lambda_{max} - \lambda_{11}) f_\delta^n(i,j,u_1) - \lambda_{21} f_\delta^n(i,j+1,u_1)$$

$$- (\lambda_{max} - \lambda_{21}) f_\delta^n(i,j,u_1) - \mu_1 f_\delta^n(i-1,j,u_1) - (\mu_{max} - \mu_1) f_\delta^n(i,j,u_1)$$

$$+ c_1 i + c_2(j+1) - c_1 i - c_2 j.$$

Simplifying and collecting the same terms we obtain:

$$N = \lambda_{11}[f_\delta^n(i+1,j+1,u_1) - f_\delta^n(i+1,j,u_1)] + (\lambda_{max} - \lambda_{11})[f_\delta^n(i,j+1,u_1) - f_\delta^n(i,j,u_1)]$$

$$+ \lambda_{21}[f_\delta^n(i,j+2,u_1) - f_\delta^n(i,j+1,u_1)] + (\lambda_{max} - \lambda_{21})[f_\delta^n(i,j+1,u_1) - f_\delta^n(i,j,u_1)]$$

$$+ \mu_1[f_\delta^n(i-1,j+1,u_1) - f_\delta^n(i-1,j,u_1)]$$

$$+(\mu_{max} - \mu_1)[f_\delta^n(i,j+1,u_1) - f_\delta^n(i,j,u_1)] + c_2.$$

Thanks to Lemma 4.4.2 and Lemma 4.4.3 all the differences inside the brackets are equal and thus can be collected together. N then becomes:

$$N = (2\lambda_{max} + \mu_{max})[f_\delta^n(i,j+1,u_1) - f_\delta^n(i,j,u_1)] + c_2.$$

We adopt the same procedure for the denominator:

$$D = \lambda_{11} f_\delta^n(i+2,j,u_1) + (\lambda_{max} - \lambda_{11}) f_\delta^n(i+1,j,u_1) + \lambda_{21} f_\delta^n(i+1,j+1,u_1)$$

$$+ (\lambda_{max} - \lambda_{21}) f_\delta^n(i+1,j,u_1) + \mu_1 f_\delta^n(i,j,u_1) + (\mu_{max} - \mu_1) f_\delta^n(i+1,j,u_1)$$

$$- \lambda_{11} f_\delta^n(i+1,j,u_1) - (\lambda_{max} - \lambda_{11}) f_\delta^n(i,j,u_1) - \lambda_{21} f_\delta^n(i,j+1,u_1)$$

$$- (\lambda_{max} - \lambda_{21}) f_\delta^n(i,j,u_1) - \mu_1 f_\delta^n(i-1,j,u_1) - (\mu_{max} - \mu_1) f_\delta^n(i,j,u_1)$$

$$+ c_1(i+1) + c_2 j - c_1 i - c_2 j.$$

Simplifying and collecting the same terms we obtain:

$$D = \lambda_{11}[f_\delta^n(i+2,j,u_1) - f_\delta^n(i+1,j,u_1)] + (\lambda_{max} - \lambda_{11})[f_\delta^n(i+1,j,u_1) - f_\delta^n(i,j,u_1)]$$

$$+ \lambda_{21}[f_\delta^n(i+1,j+1,u_1) - f_\delta^n(i,j+1,u_1)] + (\lambda_{max} - \lambda_{21})[f_\delta^n(i+1,j,u_1) - f_\delta^n(i,j,u_1)]$$

$$+ \mu_1[f_\delta^n(i,j,u_1) - f_\delta^n(i-1,j,u_1)] + (\mu_{max} - \mu_1)[f_\delta^n(i+1,j,u_1) - f_\delta^n(i,j,u_1)] + c_1.$$

Thanks to Lemma 4.4.2 and Lemma 4.4.3 all the differences inside the brackets are equal and thus can be collected together. D then becomes:

$$D = (2\lambda_{max} + \mu_{max})[f_\delta^n(i+1,j,u_1) - f_\delta^n(i,j,u_1)] + c_1.$$

Now we put back together numerator N and denominator D and we simplify the relationship:

$$\frac{(2\lambda_{max}+\mu_{max})[f_\delta^n(i,j+1,u_1)-f_\delta^n(i,j,u_1)]+c_2}{(2\lambda_{max}+\mu_{max})[f_\delta^n(i+1,j,u_1)-f_\delta^n(i,j,u_1)]+c_1}=\frac{c_2}{c_1}$$

$$(2\lambda_{max}+\mu_{max})[f_\delta^n(i,j+1,u_1)-f_\delta^n(i,j,u_1)]c_1+c_2c_1=$$

$$(2\lambda_{max}+\mu_{max})[f_\delta^n(i+1,j,u_1)-f_\delta^n(i,j,u_1)]c_2+c_1c_2$$

$$(2\lambda_{max}+\mu_{max})[f_\delta^n(i,j+1,u_1)-f_\delta^n(i,j,u_1)]c_1=$$

$$(2\lambda_{max}+\mu_{max})[f_\delta^n(i+1,j,u_1)-f_\delta^n(i,j,u_1)]c_2$$

$$\frac{(2\lambda_{max}+\mu_{max})[f_\delta^n(i,j+1,u_1)-f_\delta^n(i,j,u_1)]}{(2\lambda_{max}+\mu_{max})[f_\delta^n(i+1,j,u_1)-f_\delta^n(i,j,u_1)]}=\frac{c_2}{c_1}$$

$$\frac{f_\delta^n(i,j+1,u_1)-f_\delta^n(i,j,u_1)}{f_\delta^n(i+1,j,u_1)-f_\delta^n(i,j,u_1)}=\frac{c_2}{c_1}. \qquad \checkmark$$

The outcome of $n+1$ is what we assumed for n, the relationship is the proven.

The ratio between the differences of $f_\delta^n(i,j,u_1)$ induced by two adjacent states of j and two adjacent states of i is constant and equal to the ratio of the one-step costs.

Theorem 4.4.1. *If (4.13) holds true then* $f_\delta^n(i,j,u_1)\leq f_\delta^n(i,j,u_2)$, $\forall i\geq 1,\forall j\geq 1$.

Proof:

- $n=0$:
$$c_1i+c_2j\leq c_1i+c_2j. \quad \checkmark$$
- $n+1$: According to (4.20) and (4.19) we expand the terms, using (4.21) we simplify the minima, then we drop the δ terms:

$$\lambda_{11}f_\delta^n(i+1,j,u_1)+(\lambda_{max}-\lambda_{11})f_\delta^n(i,j,u_1)+\lambda_{21}f_\delta^n(i,j+1,u_1)+(\lambda_{max}-\lambda_{21})f_\delta^n(i,j,u_1)$$

$$+\mu_1f_\delta^n(i-1,j,u_1)+(\mu_{max}-\mu_1)f_\delta^n(i,j,u_1)+c_1i+c_2j \quad \leq$$

$$\lambda_{22}f_\delta^n(i,j+1,u_1)+(\lambda_{max}-\lambda_{22})f_\delta^n(i,j,u_2)+\lambda_{12}f_\delta^n(i+1,j,u_1)+(\lambda_{max}-\lambda_{12})f_\delta^n(i,j,u_2)$$

$$+\mu_2f_\delta^n(i,j-1,u_1)+(\mu_{max}-\mu_2)f_\delta^n(i,j,u_2)+c_1i+c_2j.$$

Simplifying and collecting the same terms we obtain:

$$(\lambda_{11}-\lambda_{12})f_\delta^n(i+1,j,u_1)+(\lambda_{max}-\lambda_{11}+\lambda_{max}-\lambda_{21})f_\delta^n(i,j,u_1)+\mu_1 f_\delta^n(i-1,j,u_1)$$

$$+(\mu_{max}-\mu_1)f_\delta^n(i,j,u_1)\leq$$

$$(\lambda_{22}-\lambda_{21})f_\delta^n(i,j+1,u_1)+(\lambda_{max}-\lambda_{22}+\lambda_{max}-\lambda_{12})f_\delta^n(i,j,u_2)+\mu_2 f_\delta^n(i,j-1,u_1)$$

$$+(\mu_{max}-\mu_2)f_\delta^n(i,j,u_2).$$

We now replace $f_\delta^n(i,j,u_2)$ with $f_\delta^n(i,j,u_1)$ using (4.21). By assumption it is a smaller quantity so if we prove the inequality after the replacement it is then also proven for the former inequality. After substitution we can further collect:

$$(\lambda_{11}-\lambda_{12})f_\delta^n(i+1,j,u_1)-(\lambda_{11}-\lambda_{12})f_\delta^n(i,j,u_1)-\mu_1[f_\delta^n(i,j,u_1)-f_\delta^n(i-1,j,u_1)]\leq$$

$$(\lambda_{22}-\lambda_{21})f_\delta^n(i,j+1,u_1)-(\lambda_{22}-\lambda_{21})f_\delta^n(i,j,u_1)-\mu_2[f_\delta^n(i,j,u_1)-f_\delta^n(i,j-1,u_1)].$$

We collect then the λ terms inside the brackets:

$$(\lambda_{11}-\lambda_{12})\underbrace{[f_\delta^n(i+1,j,u_1)-f_\delta^n(i,j,u_1)]}_{(1)}-\mu_1\underbrace{[f_\delta^n(i,j,u_1)-f_\delta^n(i-1,j,u_1)]}_{(2)}\leq$$

$$(\lambda_{22}-\lambda_{21})\underbrace{[f_\delta^n(i,j+1,u_1)-f_\delta^n(i,j,u_1)]}_{(3)}-\mu_2\underbrace{[f_\delta^n(i,j,u_1)-f_\delta^n(i,j-1,u_1)]}_{(4)}.$$

Terms (1) and (2) are equal thanks to Lemma 4.4.2. The same applies for terms (3) and (4):

$$[\mu_1-(\lambda_{11}-\lambda_{12})][f_\delta^n(i+1,j,u_1)-f_\delta^n(i,j,u_1)]\geq$$

$$[\mu_2-(\lambda_{22}-\lambda_{21})][f_\delta^n(i,j+1,u_1)-f_\delta^n(i,j,u_1)].$$

Now using Lemma 4.4.4 we can write:

$$[\mu_1-(\lambda_{11}-\lambda_{12})]c_1\geq[\mu_2-(\lambda_{22}-\lambda_{21})]c_2. \qquad \checkmark$$

We have proven that (4.13) represents the optimal scheduling policy when the arrival rates λ_s depend on the class under process.

Chapter 5

Three-class to n-class system

In this chapter we take one step further. We try to generalize what was obtained for the two-class system in chapter 3. The new analysis follows the same steps. The three-class system is characterized by the parameters:

$$\Lambda = \begin{bmatrix} \lambda_{11} & \lambda_{12} & \lambda_{13} \\ \lambda_{21} & \lambda_{22} & \lambda_{23} \\ \lambda_{31} & \lambda_{32} & \lambda_{33} \end{bmatrix} \quad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{bmatrix} \quad c = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}.$$

The complete n-class system has instead the following set of parameters:

$$\Lambda = \begin{bmatrix} \lambda_{11} & \cdots & \lambda_{1n} \\ \vdots & \ddots & \vdots \\ \lambda_{n1} & \cdots & \lambda_{nn} \end{bmatrix} \quad \mu = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_n \end{bmatrix} \quad c = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}.$$

5.1 Stability

What was derived in the two-class system is a simpler version of the general stability assessment procedure. While in the two-class system it was possible to find a formula for the stability region, in this case it is harder to compute an explicit formula for three classes or more. For stability, the system is said to be stable if the conditions on the eigenvalues are met (Ernst et al. [12]).

5.2 Fluid approximation

The system can be still approximated by formulating the fluid model and then transforming it into a linear program. The problem has the same form but since the number of classes increases also the number of unknowns and constraints increases.

5.2.1 Equal costs

As a first approximation we keep the costs equal and we study only the effects of μ_s and the λ_s. The first step is to try to extend the previous results, obtained for the two-class model, in a simple way as follows:

$$\mu_1 - (\lambda_{11} - \lambda_{12} - \lambda_{13})$$
$$\mu_2 - (\lambda_{22} - \lambda_{21} - \lambda_{23})$$
$$\mu_3 - (\lambda_{33} - \lambda_{31} - \lambda_{32}).$$

The main idea is to sort all the values from the biggest one to the smallest one and then to prioritize the classes in accordance with the ranking (the highest value class is the first one served and so on). Figure 5.1 shows how fluid model results do not match our assumption, meaning that the previous relationships need to be modified (parameters were chosen in order to have 2-1-3 class sequence).

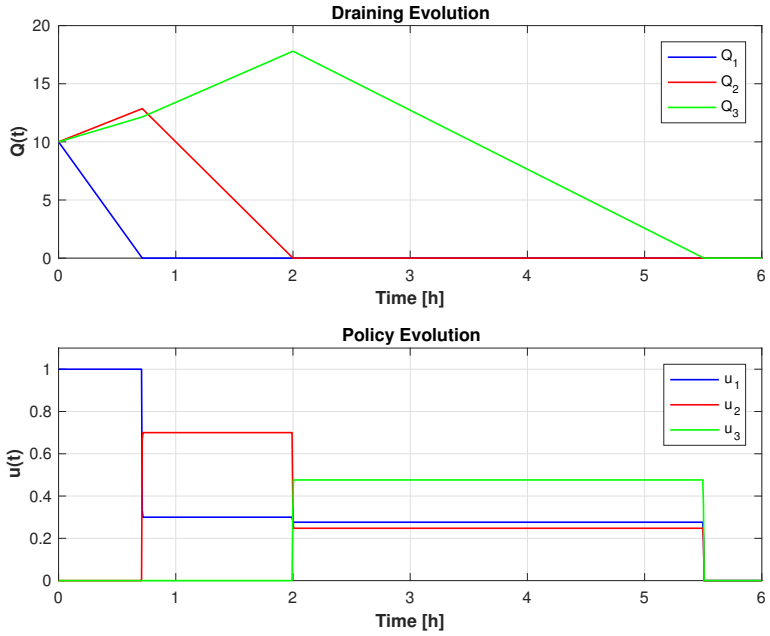

Figure 5.1: *Three-class fluid model results, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 6, \lambda_{13} = 5, \lambda_{21} = 4, \lambda_{22} = 8, \lambda_{23} = 6, \lambda_{31} = 3, \lambda_{32} = 5, \lambda_{33} = 6, \mu_1 = 24, \mu_2 = 24, \mu_3 = 21, c_1 = 1, c_2 = 1, c_3 = 1.$

Since this easy path cannot be pursued we take one step back and try to follow the same procedure used for the two classes problem. Two classes have the same priority when the difference of service rates is equal to the difference of the total arrival rate when one class is served (as in the two-class problem):

$$\mu_i - \mu_j = (\lambda_{ii} + \lambda_{ji} + \lambda_{ki}) - (\lambda_{jj} + \lambda_{ij} + \lambda_{kj}). \tag{5.1}$$

From (5.1) we can write down all the possible relationships between the classes:

$$\begin{aligned} \mu_1 - \mu_2 &= (\lambda_{11} + \lambda_{21} + \lambda_{31}) - (\lambda_{22} + \lambda_{12} + \lambda_{32}) \\ \mu_2 - \mu_3 &= (\lambda_{22} + \lambda_{12} + \lambda_{32}) - (\lambda_{33} + \lambda_{13} + \lambda_{23}) \\ \mu_1 - \mu_3 &= (\lambda_{11} + \lambda_{21} + \lambda_{31}) - (\lambda_{33} + \lambda_{13} + \lambda_{23}). \end{aligned} \tag{5.2}$$

We look for a condition where all the classes have same priority, this takes place when two out of three relationships in (5.2), arbitrarily chosen, hold at the same time (see Figure 5.2). As a matter of fact the procedure is equivalent to defining a straight line in a $\mathbb{R}^3$ space ($\mu_1\mu_2\mu_3$). Only two equations are required to describe the line.

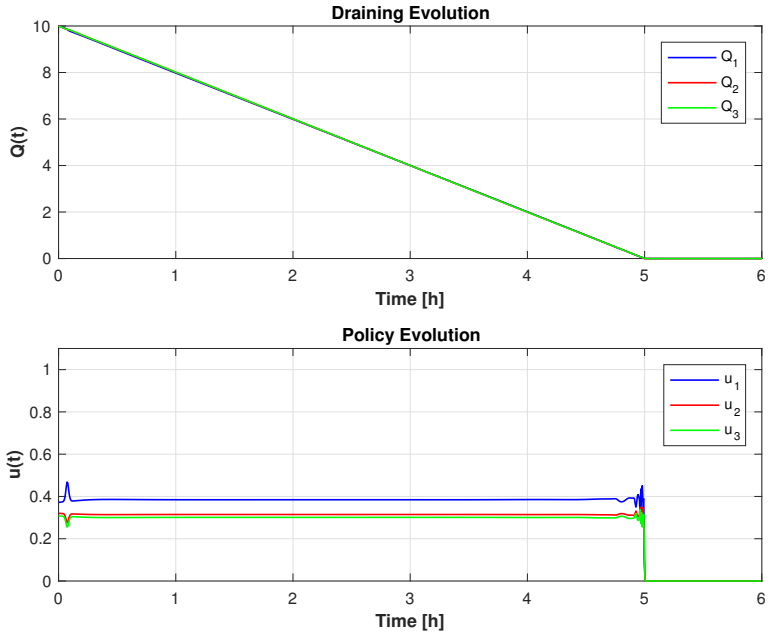

Figure 5.2: *Three-class fluid model results, equi-priority condition, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 6, \lambda_{13} = 5, \lambda_{21} = 4, \lambda_{22} = 8, \lambda_{23} = 6, \lambda_{31} = 4, \lambda_{32} = 5, \lambda_{33} = 6, \mu_1 = 24, \mu_2 = 25, \mu_3 = 23, c_1 = 1, c_2 = 1, c_3 = 1.$

As already said the relationships should hold at the same time, but they are not equal to one another. This means that it is not possible to account for all the classes at the same time and it is not possible to generate an absolute ranking once and for all. The only way to rank the classes is then by using pairwise comparisons among all the possible combinations available. We now rewrite (5.2) in a more suitable way:

$$\begin{aligned} \mu_1 - (\lambda_{11} - \lambda_{12}) - \lambda_{31} &= \mu_2 - (\lambda_{22} - \lambda_{21}) - \lambda_{32} \\ \mu_2 - (\lambda_{22} - \lambda_{21}) - \lambda_{12} &= \mu_3 - (\lambda_{33} - \lambda_{31}) - \lambda_{13} \\ \mu_1 - (\lambda_{11} - \lambda_{12}) - \lambda_{21} &= \mu_3 - (\lambda_{33} - \lambda_{31}) - \lambda_{23}. \end{aligned} \tag{5.3}$$

(5.3) are basically the two-class relationships. What is new are the extra terms λ outside the brackets. The third class not considered in the pairwise comparison does influence the decision making with its arrival rates (the ones dependent on the classes considered).

5.2.2 Cost effects

The next step is to include in (5.3) the cost effects in order to have a more complete understanding of the non-reduced model:

$$[\mu_1 - (\lambda_{11} - \lambda_{12})]c_1 - \lambda_{31}c_3 = [\mu_2 - (\lambda_{22} - \lambda_{21})]c_2 - \lambda_{32}c_3$$

$$[\mu_2 - (\lambda_{22} - \lambda_{21})]c_2 - \lambda_{12}c_1 = [\mu_3 - (\lambda_{33} - \lambda_{31})]c_3 - \lambda_{13}c_1 \tag{5.4}$$

$$[\mu_1 - (\lambda_{11} - \lambda_{12})]c_1 - \lambda_{21}c_2 = [\mu_3 - (\lambda_{33} - \lambda_{31})]c_3 - \lambda_{23}c_2.$$

Now we want to extend the obtained results to a more general case, the n class system. The overall pairwise comparisons work remains the same. Each relationship slightly changes to account for the other classes:

$$[\mu_i - (\lambda_{ii} - \lambda_{ij})]c_i - \sum_{\substack{k=1 \\ k\neq i,j}}^{n} \lambda_{ki}c_k = [\mu_j - (\lambda_{jj} - \lambda_{ji})]c_j - \sum_{\substack{k=1 \\ k\neq i,j}}^{n} \lambda_{kj}c_k. \tag{5.5}$$

The number of (5.5) (PW_{rel}) relationships depends on n:

$$PW_{rel} = \binom{n}{2} = \frac{n!}{(n-2)!2!}.$$

and only $n-1$ are meaningful to establish the ranking of priority among the classes (as stated for (4.4)). One final remark needs to be made regarding how the priority ranking evolves during the scheduling selection. The ranking in fact can be considered dynamic rather than static.

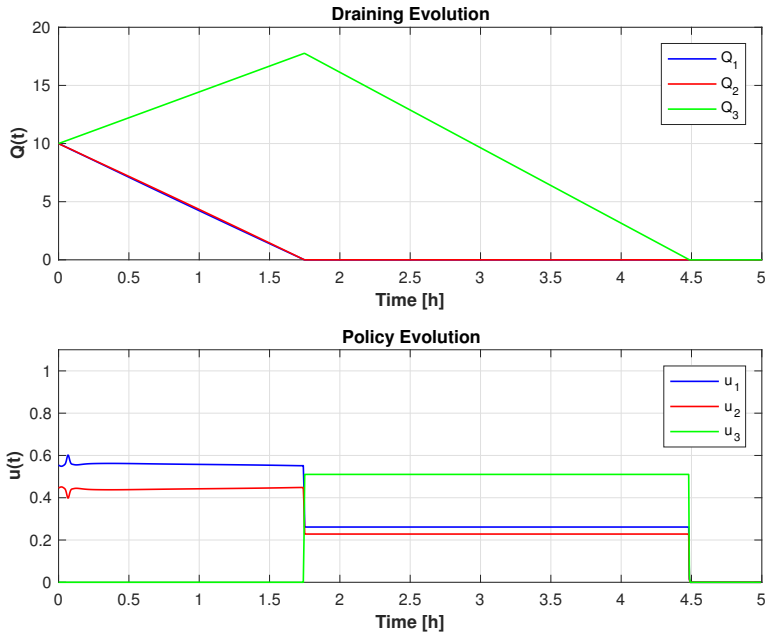

Figure 5.3: *Three-class fluid model results, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 6, \lambda_{13} = 5, \lambda_{21} = 4, \lambda_{22} = 8, \lambda_{23} = 6, \lambda_{31} = 4, \lambda_{32} = 5, \lambda_{33} = 6, \mu_1 = 25, \mu_2 = 26, \mu_3 = 23, c_1 = 1, c_2 = 1, c_3 = 1.$

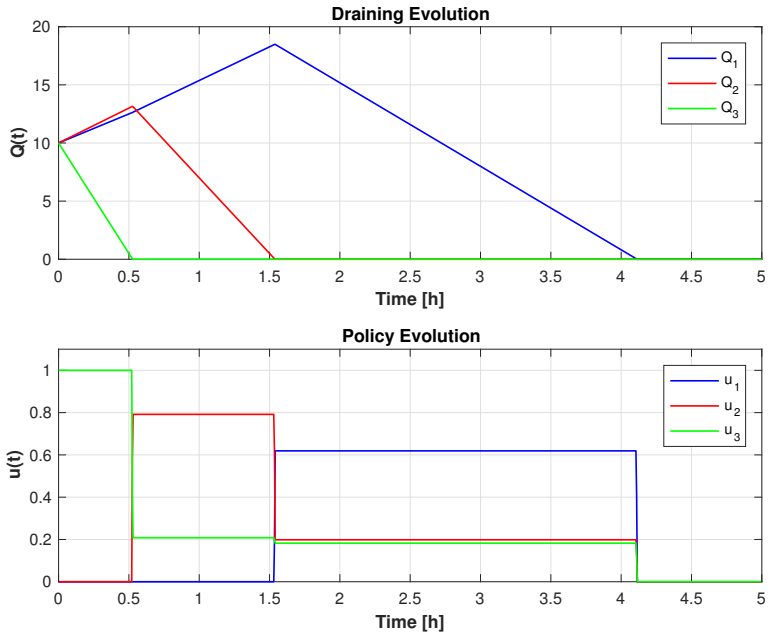

Figure 5.4: *Three-class fluid model results, the parameters used are:* $\lambda_{11} = 10, \lambda_{12} = 6, \lambda_{13} = 5, \lambda_{21} = 4, \lambda_{22} = 8, \lambda_{23} = 6, \lambda_{31} = 4, \lambda_{32} = 5, \lambda_{33} = 6, \mu_1 = 25, \mu_2 = 26, \mu_3 = 25, c_1 = 1, c_2 = 1, c_3 = 1.$

In Figure 5.3 class 1 and class 2 have the same priority level while class 3 has the lowest priority. Now we increase the value of μ_3 so as to make class 3 the highest priority. In Figure 5.4 we have the new ranking due to the different value of μ_3. It is important to notice in this case that class 1 and class 2 no longer have same priority level. The relationship between class 1 and class 2 in (4.4) does not depend on μ_3, so how is such a change possible?

Once class 3 is selected to be the first one to be processed its extra λ terms in the class 1- class 2 relationship vanish, because it no longer affects the decision between the other classes. As a general rule all the linearly independent relationships should be run. Once one class is selected to be processed it is also discharged from the set of the available classes, meaning that all its extra terms λ in the relationships should be discharged. So every time a class is selected all the relationships are updated. The number of comparisons PW_{tot} required to fully rank all the classes is given by:

$$PW_{tot} = \frac{(n-1)n}{2}.$$

5.2.3 Transitivity property

We show that for a three-class model the transitivity property applies. As a consequence the following condition can not happen:

$$[\mu_1 - (\lambda_{11} - \lambda_{12})]c_1 - \lambda_{31}c_3 \geq [\mu_2 - (\lambda_{22} - \lambda_{21})]c_2 - \lambda_{32}c_3$$

$$[\mu_2 - (\lambda_{22} - \lambda_{23})]c_2 - \lambda_{12}c_1 \geq [\mu_3 - (\lambda_{33} - \lambda_{32})]c_3 - \lambda_{13}c_1$$

$$[\mu_1 - (\lambda_{11} - \lambda_{13})]c_1 - \lambda_{21}c_2 \leq [\mu_3 - (\lambda_{33} - \lambda_{31})]c_3 - \lambda_{23}c_2.$$

The proof is carried out by *reduction ad absurdum* by adding three new parameters ϵ, δ, γ as follows:

$$\begin{cases} [\mu_1 - (\lambda_{11} - \lambda_{12})]c_1 - \lambda_{31}c_3 = [\mu_2 - (\lambda_{22} - \lambda_{21})]c_2 - \lambda_{32}c_3 + \epsilon \\ [\mu_2 - (\lambda_{22} - \lambda_{23})]c_2 - \lambda_{12}c_1 = [\mu_3 - (\lambda_{33} - \lambda_{32})]c_3 - \lambda_{13}c_1 + \delta \\ [\mu_1 - (\lambda_{11} - \lambda_{13})]c_1 - \lambda_{21}c_2 = [\mu_3 - (\lambda_{33} - \lambda_{31})]c_3 - \lambda_{23}c_2 - \gamma \\ \epsilon \geq 0 \\ \delta \geq 0 \\ \gamma \geq 0. \end{cases} \tag{5.6}$$

For the sake of simplicity we define:

$$\begin{aligned} \Delta_{11} &= [\mu_1 - (\lambda_{11} - \lambda_{12})] \\ \Delta_{21} &= [\mu_2 - (\lambda_{22} - \lambda_{21})] \\ \Delta_{23} &= [\mu_2 - (\lambda_{22} - \lambda_{23})] \\ \Delta_{32} &= [\mu_3 - (\lambda_{33} - \lambda_{32})] \\ \Delta_{13} &= [\mu_1 - (\lambda_{11} - \lambda_{13})] \\ \Delta_{31} &= [\mu_3 - (\lambda_{33} - \lambda_{31})], \end{aligned}$$

and we replace them in (5.6) obtaining:

$$\begin{cases} \Delta_{12}c_1 - \lambda_{31}c_3 = \Delta_{21}c_2 - \lambda_{32}c_3 + \epsilon \\ \Delta_{23}c_2 - \lambda_{12}c_1 = \Delta_{32}c_3 - \lambda_{13}c_1 + \delta \\ \Delta_{13}c_1 - \lambda_{21}c_2 = \Delta_{31}c_3 - \lambda_{23}c_2 - \gamma. \end{cases} \tag{5.7}$$

From the first equation in (5.7) we determine c_1:

$$c_1 = \frac{\Delta_{21}c_2 + (\lambda_{31} - \lambda_{32})c_3 + \epsilon}{\Delta_{12}}, \tag{5.8}$$

and we substitute it in the second equation in (5.7) to obtain c_2:

$$\Delta_{23}c_2 = \Delta_{32}c_3 + \frac{[\lambda_{12}-\lambda_{13}][\Delta_{21}c_2 + (\lambda_{31}-\lambda_{32})c_3 + \epsilon]}{\Delta_{12}} + \delta$$

$$\Delta_{23}c_2 = \Delta_{32}c_3 + \frac{[\lambda_{12}-\lambda_{13}][\Delta_{21}c_2 + (\lambda_{31}-\lambda_{32})c_3 + \epsilon]}{\Delta_{12}} + \delta$$

$$\left[\Delta_{23} - (\lambda_{12}-\lambda_{13})(\lambda_{31}-\lambda_{32})\frac{\Delta_{21}}{\Delta_{12}}\right]c_2 = c_3\left[\Delta_{32} + \frac{(\lambda_{12}-\lambda_{13})(\lambda_{31}-\lambda_{32})}{\Delta_{12}}\right] + \frac{(\lambda_{12}-\lambda_{13})\epsilon}{\Delta_{12}} + \delta$$

$$c_2 = \frac{\left[\Delta_{32} + \frac{(\lambda_{12}-\lambda_{13})(\lambda_{31}-\lambda_{32})}{\Delta_{12}}\right]c_3 + \frac{(\lambda_{12}-\lambda_{13})}{\Delta_{12}}\epsilon + \delta}{[\Delta_{23} - (\lambda_{12}-\lambda_{13})]\,\frac{\Delta_{21}}{\Delta_{12}}}. \tag{5.9}$$

We now replace (5.8) and (5.9) in the third equation in (5.7):

$$\left\{\left[\frac{\frac{\Delta_{13}\Delta_{21}}{\Delta_{12}} + (\lambda_{23}-\lambda_{21})}{[\Delta_{23}-(\lambda_{12}-\lambda_{13})]\,\frac{\Delta_{21}}{\Delta_{12}}}\right]\left[\Delta_{32} + \frac{(\lambda_{12}-\lambda_{13})(\lambda_{31}-\lambda_{32})}{\Delta_{12}}\right] + \frac{\Delta_{13}(\lambda_{31}-\lambda_{32})}{\Delta_{12}} - \Delta_{31}\right\}c_3$$
$$+\left\{\left[\frac{\frac{\Delta_{13}\Delta_{21}}{\Delta_{12}} + (\lambda_{23}-\lambda_{21})}{[\Delta_{23}-(\lambda_{12}-\lambda_{13})]\,\frac{\Delta_{21}}{\Delta_{12}}}\right]\frac{(\lambda_{12}-\lambda_{13})}{\Delta_{12}} + \frac{\Delta_{13}}{\Delta_{12}}\right\}\epsilon + \left\{\frac{\frac{\Delta_{13}\Delta_{21}}{\Delta_{12}} + (\lambda_{23}-\lambda_{21})}{[\Delta_{23}-(\lambda_{12}-\lambda_{13})]\,\frac{\Delta_{21}}{\Delta_{12}}}\right\}\delta + \gamma = 0. \tag{5.10}$$

We prove that the coefficient of δ in (5.10) is equal to 1:

$$\frac{\frac{\Delta_{13}\Delta_{21}}{\Delta_{12}} + (\lambda_{23}-\lambda_{21})}{[\Delta_{23}-(\lambda_{12}-\lambda_{13})]\,\frac{\Delta_{21}}{\Delta_{12}}} = 1$$

$$\frac{\frac{(\mu_1-\lambda_{11}+\lambda_{13})(\mu_2-\lambda_{22}+\lambda_{21})}{\mu_1-\lambda_{11}+\lambda_{12}} + (\lambda_{23}-\lambda_{21})}{(\mu_2-\lambda_{22}+\lambda_{23}) - (\lambda_{12}-\lambda_{13})\frac{\mu_2-\lambda_{22}+\lambda_{21}}{\mu_1-\lambda_{11}+\lambda_{12}}} = 1$$

$$\frac{(\mu_1-\lambda_{11}+\lambda_{13})(\mu_2-\lambda_{22}+\lambda_{21}) + (\lambda_{23}-\lambda_{21})(\mu_1-\lambda_{11}+\lambda_{12})}{\mu_1-\lambda_{11}+\lambda_{12}} =$$
$$(\mu_2-\lambda_{22}+\lambda_{23}) - (\lambda_{12}-\lambda_{13})\frac{\mu_2-\lambda_{22}+\lambda_{21}}{\mu_1-\lambda_{11}+\lambda_{12}}$$

$$\begin{aligned}\mu_1\mu_2 - \mu_1\lambda_{22} + \mu_1\lambda_{21} - \mu_2\lambda_{11} + \lambda_{11}\lambda_{22} - \lambda_{11}\lambda_{21} + \mu_2\lambda_{13} - \lambda_{13}\lambda_{22} + \lambda_{13}\lambda_{21} \\ +\mu_1\lambda_{23} - \lambda_{12}\lambda_{23} + \lambda_{12}\lambda_{23} - \mu_1\lambda_{21} + \lambda_{11}\lambda_{21} - \lambda_{12}\lambda_{21} = \\ \mu_1\mu_2 - \mu_2\lambda_{11} + \mu_2\lambda_{12} - \mu_2\lambda_{22} + \lambda_{11}\lambda_{22} - \lambda_{12}\lambda_{22} + \mu_1\lambda_{23} - \lambda_{11}\lambda_{23} + \lambda_{12}\lambda_{23} \\ -\mu_2\lambda_{12} + \lambda_{12}\lambda_{22} - \lambda_{12}\lambda_{21} + \mu_2\lambda_{13} - \lambda_{13}\lambda_{22} + \lambda_{13}\lambda_{21}. \qquad \checkmark\end{aligned}$$

Part of the coefficient of ϵ is equal to the coefficient of δ. It is equal to 1 and it allows us to simplify (5.10). We prove then the coefficient of ϵ in (5.10) is equal to 1:

$$\frac{(\lambda_{12} - \lambda_{13})}{\Delta_{12}} + \frac{\Delta_{13}}{\Delta_{12}} = 1$$

$$\lambda_{12} - \lambda_{13} + \mu_1 - \lambda_{11} + \lambda_{13} = \mu_1 - \lambda_{11} + \lambda_{12}. \qquad \checkmark$$

We now prove that the coefficient of c_3 in (5.10) is equal to 0:

$$\left[\Delta_{32} + \frac{(\lambda_{12} - \lambda_{13})(\lambda_{31} - \lambda_{32})}{\Delta_{12}}\right] + \frac{\Delta_{13}(\lambda_{31} - \lambda_{32})}{\Delta_{12}} - \Delta_{31} = 0$$

$$\begin{aligned}(\mu_3 - \lambda_{33} + \lambda_{32})(\mu_1 - \lambda_{11} + \lambda_{12}) + (\lambda_{12} - \lambda_{13})(\lambda_{31} - \lambda_{32}) = \\ -(\mu_1 - \lambda_{11} + \lambda_{13})(\lambda_{31} - \lambda_{32})(\mu_3 - \lambda_{33} + \lambda_{31})(\mu_1 - \lambda_{11} + \lambda_{12})\end{aligned}$$

$$\begin{aligned}&\mu_1\mu_3 - \mu_3\lambda_{11} + \mu_3\lambda_{12} - \mu_1\lambda_{33} + \lambda_{11}\lambda_{33} - \lambda_{12}\lambda_{33} + \mu_1\lambda_{32} - \lambda_{11}\lambda_{32} + \lambda_{12}\lambda_{32} + \lambda_{12}\lambda_{31} \\ &-\lambda_{12}\lambda_{32} - \lambda_{13}\lambda_{31} + \lambda_{13}\lambda_{32} + \mu_1\lambda_{31} - \mu_1\lambda_{32} - \lambda_{11}\lambda_{31} + \lambda_{11}\lambda_{32} + \lambda_{13}\lambda_{31} - \lambda_{13}\lambda_{32} \\ &-\mu_1\mu_3 + \lambda_{11}\mu_3 - \mu_3\lambda_{12} + \mu_1\lambda_{33} - \lambda_{11}\lambda_{33} + \lambda_{12}\lambda_{33} - \mu_1\lambda_{31} + \lambda_{11}\lambda_{31} - \lambda_{12}\lambda_{31} = 0. \qquad \checkmark\end{aligned}$$

(5.10) reduces to the simpler:

$$\epsilon + \delta + \gamma = 0,$$

which is true if and only if all the three parameters ϵ, δ, γ are equal to 0. The equivalences in (5.6) are met only when the new parameters are null at the same time. This means that the transitivity property applies.

5.2.4 Scheduling algorithm

We want to give an example of how the whole scheduling procedure works, the system is characterized by a single flexible server and four different job classes. For sake of completeness we compute all the (4.4) pairwise comparisons (even if it is not needed).

$$\Lambda = \begin{bmatrix} \lambda_{11} & \lambda_{12} & \lambda_{13} & \lambda_{14} \\ \lambda_{21} & \lambda_{22} & \lambda_{23} & \lambda_{24} \\ \lambda_{31} & \lambda_{32} & \lambda_{33} & \lambda_{34} \\ \lambda_{41} & \lambda_{42} & \lambda_{43} & \lambda_{44} \end{bmatrix} = \begin{bmatrix} 10 & 6 & 5 & 7 \\ 7 & 8 & 6 & 7 \\ 4 & 5 & 6 & 5 \\ 7 & 8 & 6 & 9 \end{bmatrix}$$

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \end{bmatrix} = \begin{bmatrix} 41 \\ 44 \\ 25 \\ 30 \end{bmatrix} \qquad c = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 1.1 \\ 1 \\ 1.3 \\ 1.4 \end{bmatrix}$$

- Step 1:

$$[\mu_1 - (\lambda_{11} - \lambda_{12})]c_1 - \lambda_{31}C_3 - \lambda_{41}c_4 = [\mu_2 - (\lambda_{22} - \lambda_{21})]c_2 - \lambda_{32}C_3 - \lambda_{42}c_4$$

$$[\mu_1 - (\lambda_{11} - \lambda_{13})]c_1 - \lambda_{21}C_2 - \lambda_{41}c_4 = [\mu_3 - (\lambda_{33} - \lambda_{31})]c_3 - \lambda_{23}C_2 - \lambda_{43}c_4$$

$$[\mu_1 - (\lambda_{11} - \lambda_{14})]c_1 - \lambda_{21}C_2 - \lambda_{31}c_3 = [\mu_4 - (\lambda_{44} - \lambda_{41})]c_4 - \lambda_{24}C_2 - \lambda_{34}c_3$$

$$[\mu_2 - (\lambda_{22} - \lambda_{23})]c_2 - \lambda_{12}C_1 - \lambda_{42}c_4 = [\mu_3 - (\lambda_{33} - \lambda_{32})]c_3 - \lambda_{13}C_1 - \lambda_{43}c_4$$

$$[\mu_2 - (\lambda_{22} - \lambda_{24})]c_2 - \lambda_{12}C_1 - \lambda_{32}c_3 = [\mu_4 - (\lambda_{44} - \lambda_{42})]c_4 - \lambda_{14}C_1 - \lambda_{34}c_3$$

$$[\mu_3 - (\lambda_{33} - \lambda_{34})]c_3 - \lambda_{13}C_1 - \lambda_{23}c_2 = [\mu_4 - (\lambda_{44} - \lambda_{43})]c_4 - \lambda_{14}C_1 - \lambda_{24}c_2. \tag{5.11}$$

(5.11) represents the set of all the possible combinations with four classes. Now we replace the values inside the relationships and we rank the classes accordingly.

$$\begin{aligned} 25.7 &> 25.3 \qquad 1 > 2 \\ 22.8 &> 15.5 \qquad 1 > 3 \\ 29.6 &> 25.7 \qquad 1 > 4 \\ 24.2 &> 17.3 \qquad 2 > 3 \\ 29.9 &> 26.4 \qquad 2 > 4 \\ 19.7 &< 23.1 \qquad 3 < 4 \end{aligned}$$

The resulting order is $1 > 2 > 4 > 3$ (to be noticed: it is not the final order), class 1 is then chosen to be served as first.

- Step 2: Class 1 has been selected, so it is discharged from the set of available classes and (5.11) are then updated:

$$[\mu_2 - (\lambda_{22} - \lambda_{23})]c_2 - \lambda_{42}c_4 = [\mu_3 - (\lambda_{33} - \lambda_{32})]c_3 - \lambda_{43}c_4$$

$$[\mu_2 - (\lambda_{22} - \lambda_{24})]c_2 - \lambda_{32}c_3 = [\mu_4 - (\lambda_{44} - \lambda_{42})]c_4 - \lambda_{34}c_3 \tag{5.12}$$

$$[\mu_3 - (\lambda_{33} - \lambda_{34})]c_3 - \lambda_{23}c_2 = [\mu_4 - (\lambda_{44} - \lambda_{43})]c_4 - \lambda_{24}c_2.$$

We substitute the values of the parameters inside (5.12) and we create a new ranking.

$$\begin{aligned} 30.8 &> 22.8 \qquad 2 > 3 \\ 36.5 &> 34.1 \qquad 2 > 4 \\ 25.2 &< 30.8 \qquad 3 < 4 \end{aligned}$$

The resulting order is $2 > 4 > 3$ and class 2 is then chosen to be served as second.

- Step 3: Class 2 has been selected, so it is discharged from the set of available classes and (5.12) are then updated:

$$[\mu_3 - (\lambda_{33} - \lambda_{34})]c_3 = [\mu_4 - (\lambda_{44} - \lambda_{43})]c_4. \tag{5.13}$$

We substitute the values of the parameters inside (5.13) and we create a new ranking.

$$31.2 < 37.8 \qquad 3 < 4$$

The resulting order is $4 > 3$ and class 4 is then chosen to be served as third and class 3 as fourth.

The fluid model shows the same results in Figure 5.5. It is a general procedure suitable for all the values assumed by n. As n increases also the number of steps increases.

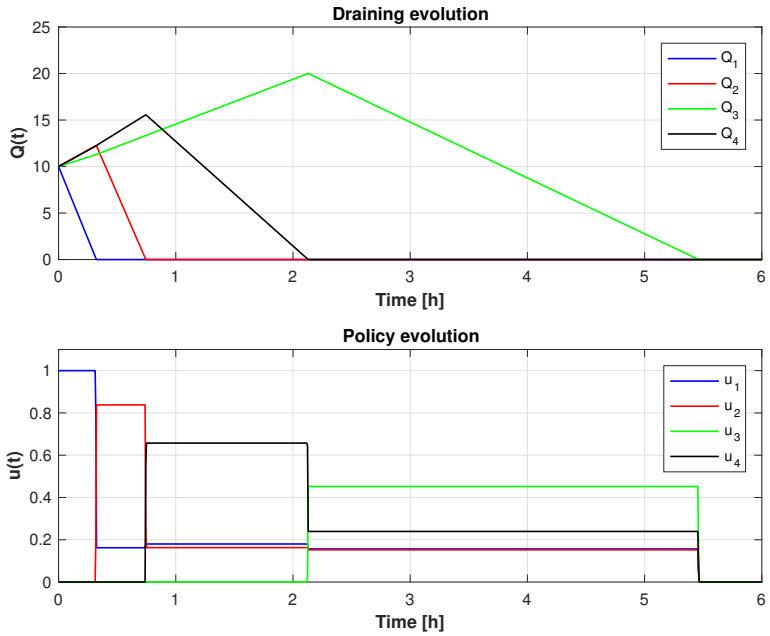

Figure 5.5: *Fluid model results of the scheduling example, the parameters used are given at the beginning of the section.*

5.3 Numerical Markov decision process

While in the two-class model section the numerical Markov decision process was a useful tool to check the results of the fluid model, here in this section we neglect it because we know that the fluid model gives us reliable results. We decided then to address directly the analytical proof following the steps of the simpler system.

5.4 Analytical Markov decision process

We want to prove once again analytically the results given by the fluid model. One major challenge is to account for different relationships at the same time. We saw in the

fluid model section that when the classes are more than two it is not possible to sort them in an absolute ranking but it is necessary to perform several pairwise comparisons.

5.4.1 Mapping

The problem setup is identical to the two classes model, we derive the discrete Markov decision process and then we uniformize the transition rates. The mapping H is expanded to account for the new class and its parameters (Sisbot and Hasenbein [10]):

$$
\begin{aligned}
H_\delta f(i,j,k,u) = \delta min & \begin{cases} \lambda_{11} f(i+1,j,k,u_1) + (\lambda_{max} - \lambda_{11}) f(i,j,k,u_1) \\ \lambda_{11} f(i+1,j,k,u_2) + (\lambda_{max} - \lambda_{11}) f(i,j,k,u_1) \\ \lambda_{11} f(i+1,j,k,u_3) + (\lambda_{max} - \lambda_{11}) f(i,j,k,u_1) \end{cases} \\
+ \delta min & \begin{cases} \lambda_{21} f(i,j+1,k,u_1) + (\lambda_{max} - \lambda_{21}) f(i,j,k,u_1) \\ \lambda_{21} f(i,j+1,k,u_2) + (\lambda_{max} - \lambda_{21}) f(i,j,k,u_1) \\ \lambda_{21} f(i,j+1,k,u_3) + (\lambda_{max} - \lambda_{21}) f(i,j,k,u_1) \end{cases} \\
+ \delta min & \begin{cases} \lambda_{31} f(i,j,k+1,u_1) + (\lambda_{max} - \lambda_{31}) f(i,j,k,u_1) \\ \lambda_{31} f(i,j,k+1,u_2) + (\lambda_{max} - \lambda_{31}) f(i,j,k,u_1) \\ \lambda_{31} f(i,j,k+1,u_3) + (\lambda_{max} - \lambda_{31}) f(i,j,k,u_1) \end{cases} \\
+ \delta min & \begin{cases} \mu_1 f(i-1,j,k,u_1) + (\mu_{max} - \mu_1) f(i,j,k,u_1) \\ \mu_1 f(i-1,j,k,u_2) + (\mu_{max} - \mu_1) f(i,j,k,u_1) \\ \mu_1 f(i-1,j,k,u_3) + (\mu_{max} - \mu_1) f(i,j,k,u_1) \qquad if\ u = u_1 \end{cases}
\end{aligned}
\tag{5.14}
$$

$$H_\delta f(i,j,k,u) = \delta min \begin{cases} \lambda_{22} f(i,j+1,k,u_1) + (\lambda_{max} - \lambda_{22}) f(i,j,k,u_2) \\ \lambda_{22} f(i,j+1,k,u_2) + (\lambda_{max} - \lambda_{22}) f(i,j,k,u_2) \\ \lambda_{22} f(i,j+1,k,u_3) + (\lambda_{max} - \lambda_{22}) f(i,j,k,u_2) \end{cases}$$

$$+ \delta min \begin{cases} \lambda_{12} f(i+1,j,k,u_1) + (\lambda_{max} - \lambda_{12}) f(i,j,k,u_2) \\ \lambda_{12} f(i+1,j,k,u_2) + (\lambda_{max} - \lambda_{12}) f(i,j,k,u_2) \\ \lambda_{12} f(i+1,j,k,u_3) + (\lambda_{max} - \lambda_{12}) f(i,j,k,u_2) \end{cases}$$

$$+ \delta min \begin{cases} \lambda_{32} f(i,j,k+1,u_1) + (\lambda_{max} - \lambda_{32}) f(i,j,k,u_2) \\ \lambda_{32} f(i,j,k+1,u_2) + (\lambda_{max} - \lambda_{32}) f(i,j,k,u_2) \\ \lambda_{32} f(i,j,k+1,u_3) + (\lambda_{max} - \lambda_{32}) f(i,j,k,u_2) \end{cases}$$

$$+ \delta min \begin{cases} \mu_2 f(i,j-1,k,u_1) + (\mu_{max} - \mu_2) f(i,j,k,u_2) \\ \mu_2 f(i,j-1,k,u_2) + (\mu_{max} - \mu_2) f(i,j,k,u_2) \\ \mu_2 f(i,j-1,k,u_3) + (\mu_{max} - \mu_2) f(i,j,k,u_2) \end{cases} \quad if\ u = u_2$$

$$H_\delta f(i,j,k,u) = \delta min \begin{cases} \lambda_{33} f(i,j,k+1,u_1) + (\lambda_{max} - \lambda_{33}) f(i,j,k,u_3) \\ \lambda_{33} f(i,j,k+1,u_2) + (\lambda_{max} - \lambda_{33}) f(i,j,k,u_3) \\ \lambda_{33} f(i,j,k+1,u_3) + (\lambda_{max} - \lambda_{33}) f(i,j,k,u_3) \end{cases}$$

$$+ \delta min \begin{cases} \lambda_{13} f(i+1,j,k,u_1) + (\lambda_{max} - \lambda_{13}) f(i,j,k,u_3) \\ \lambda_{13} f(i+1,j,k,u_2) + (\lambda_{max} - \lambda_{13}) f(i,j,k,u_3) \\ \lambda_{13} f(i+1,j,k,u_3) + (\lambda_{max} - \lambda_{13}) f(i,j,k,u_3) \end{cases}$$

$$+ \delta min \begin{cases} \lambda_{23} f(i,j+1,k,u_1) + (\lambda_{max} - \lambda_{23}) f(i,j,k,u_3) \\ \lambda_{23} f(i,j+1,k,u_2) + (\lambda_{max} - \lambda_{23}) f(i,j,k,u_3) \\ \lambda_{23} f(i,j+1,k,u_3) + (\lambda_{max} - \lambda_{23}) f(i,j,k,u_3) \end{cases}$$

$$+ \delta min \begin{cases} \mu_3 f(i,j,k-1,u_1) + (\mu_{max} - \mu_3) f(i,j,k,u_3) \\ \mu_3 f(i,j,k-1,u_2) + (\mu_{max} - \mu_3) f(i,j,k,u_3) \\ \mu_3 f(i,j,k-1,u_3) + (\mu_{max} - \mu_3) f(i,j,k,u_3) \end{cases} \quad if\ u = u_3.$$

5.4.2 Results

The results are for the discounted cost finite horizon problem. It is then possible to generalize the proof for the infinite horizon average cost case. (5.15) is the optimal scheduling policy, so if both the conditions are met it is always optimal to serve class 1

rather than class 2 and 3 (if class 1 queue is not empty):

$$\begin{aligned}[\mu_1 - (\lambda_{11} - \lambda_{12})]c_1 - \lambda_{31}c_3 \geq [\mu_2 - (\lambda_{22} - \lambda_{21})]c_2 - \lambda_{32}c_3 \\ [\mu_1 - (\lambda_{11} - \lambda_{13})]c_1 - \lambda_{21}c_2 \geq [\mu_3 - (\lambda_{33} - \lambda_{23})]c_3 - \lambda_{23}c_2.\end{aligned} \tag{5.15}$$

Similarly to (4.20) the discounted cost finite horizon problem is formulated as follows:

$$f_\delta^{n+1}(i,j,k,u) = c_1 i + c_2 j + c_3 k + H_\delta f_\delta^n(i,j,k,u). \tag{5.16}$$

Once again the proof is carried out by induction, since we want to prove class 1 is optimal we assume:

$$\begin{aligned}f_\delta^n(i,j,k,u_1) \leq f_\delta^n(i,j,k,u_2) \\ f_\delta^n(i,j,k,u_1) \leq f_\delta^n(i,j,k,u_3).\end{aligned} \tag{5.17}$$

Lemma 5.4.1. *$f_\delta^n(i+1,j,k,u_1) - f_\delta^n(i,j,k,u_1) \geq 0$ for each fixed j, k (the same applies for j and k respectively), $\forall i \geq 1, \forall j \geq 1, \forall k \geq 1$.*

Proof: The proof is carried out by following the same exact procedure as in Lemma 4.4.1. More terms are present but eventually the results are the same.

Lemma 5.4.2. *$f_\delta^n(i+1,j,k,u_1) - f_\delta^n(i,j,k,u_1) = f_\delta^n(i,j,k,u_1) - f_\delta^n(i-1,j,k,u_1)$ for each fixed j, k (the same applies for j and k respectively), $\forall i \geq 1, \forall j \geq 1, \forall k \geq 1$.*

Proof: The proof is carried out by following the same exact procedure as in Lemma 4.4.2. More terms are present but eventually the results are the same.

Lemma 5.4.3. *$f_\delta^n(i+1,j+1,k+1,u_1) - f_\delta^n(i,j+1,k+1,u_1) = f_\delta^n(i+1,j,k,u_1) - f_\delta^n(i,j,k,u_1)$ for each fixed difference in i and for each fixed j, k in the same side of the equivalence (the same applies for j and k respectively), $\forall i \geq 1, \forall j \geq 1, \forall k \geq 1$.*

Proof: The proof is carried out by following the same exact procedure as in Lemma 4.4.3. More terms are present but eventually the results are the same.

So far Lemma 5.4.1 to Lemma 5.4.3 slightly change from the two classes model by adding just the parameter k to account for the third class. Lemma 5.4.4 instead presents major modifications with respect to the one shown for the two classes model.

Lemma 5.4.4.

$$\begin{aligned}\frac{f_\delta^n(i,j+1,k,u_1) - f_\delta^n(i,j,k,u_1)}{f_\delta^n(i+1,j,k,u_1) - f_\delta^n(i,j,k,u_1)} = \frac{c_2}{c_1} \\ \frac{f_\delta^n(i,j,k+1,u_1) - f_\delta^n(i,j,k,u_1)}{f_\delta^n(i+1,j,k,u_1) - f_\delta^n(i,j,k,u_1)} = \frac{c_3}{c_1} \qquad \forall i \geq 1, \forall j \geq 1, \forall k \geq 1.\end{aligned}$$

Proof: The proof is carried out by following the same exact procedure as in Lemma 4.4.4, however it is necessary to prove all the relationships in parallel, the steps are the same while the indexes change.

Theorem 5.4.1. *If (5.15) holds true then* $f_\delta^n(i,j,k,u_1) \leq f_\delta^n(i,j,k,u_2)$ *and* $f_\delta^n(i,j,k,u_1) \leq f_\delta^n(i,j,k,u_3)$, $\forall i \geq 1, \forall j \geq 1, \forall k \geq 1$.

Proof: The proof for both the inequalities evolves in parallel following the same steps. For the sake of compactness we only show one of them.

- $n = 0$:
$$c_1 i + c_2 j + c_3 k \leq c_1 i + c_2 j + c_3 k. \quad \checkmark$$
- $n+1$: According to (5.16) and (5.14) we expand the terms, using (5.17) we simplify the minima, then we drop the δ terms:

$$\lambda_{11} f_\delta^n(i+1,j,k,u_1) + (\lambda_{max} - \lambda_{11}) f_\delta^n(i,j,k,u_1) + \lambda_{21} f_\delta^n(i,j+1,k,u_1)$$

$$+(\lambda_{max} - \lambda_{21}) f_\delta^n(i,j,k,u_1) + \lambda_{31} f(i,j,k+1,u_1) + (\lambda_{max} - \lambda_{31}) f(i,j,k,u_1)$$

$$+\mu_1 f_\delta^n(i-1,j,k,u_1) + (\mu_{max} - \mu_1) f_\delta^n(i,j,k,u_1) + c_1 i + c_2 j + c_3 k \quad \leq$$

$$\lambda_{22} f_\delta^n(i,j+1,k,u_1) + (\lambda_{max} - \lambda_{22}) f_\delta^n(i,j,k,u_2) + \lambda_{12} f_\delta^n(i+1,j,k,u_1)$$

$$+(\lambda_{max} - \lambda_{12}) f_\delta^n(i,j,k,u_2) + \lambda_{32} f(i,j,k+1,u_1) + (\lambda_{max} - \lambda_{32}) f(i,j,k,u_2)$$

$$+\mu_2 f_\delta^n(i,j-1,k,u_1) + (\mu_{max} - \mu_2) f_\delta^n(i,j,k,u_2) + c_1 i + c_2 j + c_3 k.$$

Simplifying and collecting the same terms we obtain:

$$(\lambda_{11} - \lambda_{12}) f_\delta^n(i+1,j,k,u_1) + (\lambda_{max} - \lambda_{11} + \lambda_{max} - \lambda_{21} + \lambda_{max} - \lambda_{31}) f_\delta^n(i,j,k,u_1)$$

$$+\lambda_{31} f(i,j,k+1,u_1) + \mu_1 f_\delta^n(i-1,j,k,u_1) + (\mu_{max} - \mu_1) f_\delta^n(i,j,k,u_1) \leq$$

$$(\lambda_{22} - \lambda_{21}) f_\delta^n(i,j+1,k,u_1) + (\lambda_{max} - \lambda_{22} + \lambda_{max} - \lambda_{12} + \lambda_{max} - \lambda_{32}) f_\delta^n(i,j,k,u_2)$$

$$+\lambda_{32} f(i,j,k+1,u_1) + \mu_2 f_\delta^n(i,j-1,k,u_1) + (\mu_{max} - \mu_2) f_\delta^n(i,j,k,u_2).$$

We now replace $f_\delta^n(i,j,k,u_2)$ with $f_\delta^n(i,j,k,u_1)$ using (5.17). For assumption it is a smaller quantity so if we prove the inequality after the replacement it is then also

proven the former inequality. After substitution we can further collect:

$$(\lambda_{11}-\lambda_{12})f_\delta^n(i+1,j,k,u_1)-(\lambda_{11}-\lambda_{12})f_\delta^n(i,j,k,u_1)$$

$$+\lambda_{31}[f_\delta^n(i,j,k+1,u_1)-f_\delta^n(i,j,k,u_1)]-\mu_1[f_\delta^n(i,j,k,u_1)-f_\delta^n(i-1,j,k,u_1)]\leq$$

$$(\lambda_{22}-\lambda_{21})f_\delta^n(i,j+1,k,u_1)-(\lambda_{22}-\lambda_{21})f_\delta^n(i,j,k,u_1)$$

$$+\lambda_{32}[f_\delta^n(i,j,k+1,u_1)-f_\delta^n(i,j,k,u_1)]-\mu_2[f_\delta^n(i,j,k,u_1)-f_\delta^n(i,j-1,k,u_1)].$$

We collect then the λ terms inside the brackets:

$$(\lambda_{11}-\lambda_{12})\underbrace{[f_\delta^n(i+1,j,k,u_1)-f_\delta^n(i,j,k,u_1)]}_{(5)}-\mu_1\underbrace{[f_\delta^n(i,j,k,u_1)-f_\delta^n(i-1,j,k,u_1)]}_{(6)}$$

$$+\lambda_{31}[f_\delta^n(i,j,k+1,u_1)-f_\delta^n(i,j,k,u_1)]\leq$$

$$(\lambda_{22}-\lambda_{21})\underbrace{[f_\delta^n(i,j+1,k,u_1)-f_\delta^n(i,j,k,u_1)]}_{(7)}-\mu_2\underbrace{[f_\delta^n(i,j,k,u_1)-f_\delta^n(i,j-1,k,u_1)]}_{(8)}$$

$$+\lambda_{32}[f_\delta^n(i,j,k+1,u_1)-f_\delta^n(i,j,k,u_1)].$$

Terms (5) and (6) are equal thanks to Lemma 5.4.2. The same applies for terms (7) and (8):

$$[\mu_1-(\lambda_{11}-\lambda_{12})][f_\delta^n(i+1,j,k,u_1)-f_\delta^n(i,j,k,u_1)]$$

$$-\lambda_{31}[f_\delta^n(i,j,k+1,u_1)-f_\delta^n(i,j,k,u_1)]\geq$$

$$[\mu_2-(\lambda_{22}-\lambda_{21})][f_\delta^n(i,j+1,k,u_1)-f_\delta^n(i,j,k,u_1)]$$

$$-\lambda_{32}[f_\delta^n(i,j,k+1,u_1)-f_\delta^n(i,j,k,u_1)].$$

Now using Lemma 5.4.4 we can write:

$$[\mu_1-(\lambda_{11}-\lambda_{12})]c_1-\lambda_{31}c_3\geq[\mu_2-(\lambda_{22}-\lambda_{21})]c_2-\lambda_{32}c_3. \qquad \checkmark$$

As already said the proof for $f_\delta^n(i,j,k,u_1)<f_\delta^n(i,j,k,u_3)$ follows the same steps, eventually the last step will show the second inequality of (5.17) instead of the first one.

We have proven that (5.15) represents the optimal scheduling policy when the arrival rates λ_s depend on the class under process.

5.4.3 Generalization to n-class system

Once proven the optimality for a three-class system it is easy to extend the results to a general n-class system. The general mapping H is defined in a compact way as follows:

$$
\begin{aligned}
H_\delta f(i,j,\ldots,n,u) = \delta min & \begin{cases} \lambda_{ii} f(i+1,j,\ldots,n,u_i) + (\lambda_{max} - \lambda_{ii}) f(i,j,\ldots,n,u_i) \\ \vdots \\ \lambda_{ii} f(i+1,j,\ldots,n,u_n) + (\lambda_{max} - \lambda_{ii}) f(i,j,\ldots,n,u_i) \end{cases} \\
\vdots & \\
+ \delta min & \begin{cases} \lambda_{ni} f(i,j,\ldots,n+1,u_i) + (\lambda_{max} - \lambda_{ni}) f(i,j,\ldots,n,u_i) \\ \vdots \\ \lambda_{ni} f(i,j,\ldots,n+1,u_n) + (\lambda_{max} - \lambda_{ni}) f(i,j,\ldots,n,u_i) \end{cases} \\
+ \delta min & \begin{cases} \mu_i f(i-1,j,\ldots,n,u_i) + (\mu_{max} - \mu_i) f(i,j,\ldots,n,u_i) \\ \vdots \\ \mu_i f(i-1,j,\ldots,n,u_n) + (\mu_{max} - \mu_i) f(i,j,\ldots,n,u_i) \end{cases} \qquad if\ u = u_i.
\end{aligned}
$$

The relationships representing in a compact way the optimal scheduling are:

$$
[\mu_i - (\lambda_{ii} - \lambda_{ij})]c_i - \sum_{\substack{k=1 \\ k \neq i,j}}^{n} \lambda_{ki} c_k \geq [\mu_j - (\lambda_{jj} - \lambda_{ji})]c_j - \sum_{\substack{k=1 \\ k \neq i,j}}^{n} \lambda_{kj} c_k. \tag{5.18}
$$

To prove class 1 is optimal we assume as usual:

$$
\begin{aligned}
f_\delta^n(i,j,\ldots,n,u_1) &\leq f_\delta^n(i,j,\ldots,n,u_2) \\
&\vdots \\
f_\delta^n(i,j,\ldots,n,u_1) &\leq f_\delta^n(i,j,\ldots,n,u_n).
\end{aligned} \tag{5.19}
$$

Also the Lemmas and the Theorem can be rewritten in a more compact way to describe the general case with n job classes.

Lemma 5.4.5. *$f_\delta^n(i+1,j,\ldots,n,u_1) - f_\delta^n(i,j,\ldots,n,u_1) \geq 0$ for each fixed $j,k,\ldots,n$ (the same applies for $j,\ldots,n$ respectively); $\forall i \geq 1, \forall j \geq 1, \ldots, \forall n \geq 1$.*

Lemma 5.4.6. *$f_\delta^n(i+1,j,\ldots,n,u_1) - f_\delta^n(i,j,\ldots,n,u_1) = f_\delta^n(i,j,\ldots,n,u_1) - f_\delta^n(i-1,j,\ldots,n,u_1)$ for each fixed $j,\ldots,n$ (the same applies for $j,\ldots,n$ respectively); $\forall i \geq 1, \forall j \geq 1, \ldots, \forall n \geq 1$.*

Lemma 5.4.7. *$f_\delta^n(i+1,j+1,\ldots,n+1,u_1) - f_\delta^n(i,j+1,\ldots,n+1,u_1) = f_\delta^n(i+1,j,\ldots,n,u_1) - f_\delta^n(i,j,\ldots,n,u_1)$ for each fixed difference in i and for each fixed $j,\ldots,n$ in the same side of the equivalence (the same applies for $j,\ldots,n$ respectively), $\forall i \geq 1, \forall j \geq 1, \ldots, \forall n \geq 1$.*

Lemma 5.4.8.

$$\frac{f_\delta^n(i,j+1,\ldots,n,u_1)-f_\delta^n(i,j,\ldots,n,u_1)}{f_\delta^n(i+1,j,\ldots,n,u_1)-f_\delta^n(i,j,\ldots,n,u_1)}=\frac{c_j}{c_i}$$

$$\vdots$$

$$\frac{f_\delta^n(i,j,\ldots,n+1,u_1)-f_\delta^n(i,j,\ldots,n,u_1)}{f_\delta^n(i+1,j,\ldots,n,u_1)-f_\delta^n(i,j,\ldots,n,u_1)}=\frac{c_n}{c_i} \qquad \forall i\geq 1,\forall j\geq 1,\ldots,\forall n\geq 1.$$

Theorem 5.4.2. *If (5.18) holds true then (5.19) is also true,* $\forall i\geq 1,\forall j\geq 1,\cdots,\forall n\geq 1$.

Proof: The proof for evolves in parallel following the same steps for each relationship in (5.18), eventually the last step of the proof will show each one of the (5.17) used.

Chapter 6

Practical case

We analyzed the emergency room scheduling process, in particular we considered a medium size emergency room in Lombardy, Italy. The patients in an emergency room are grouped into different classes depending on the severity of their conditions. The severity class is assessed upon arrival. The class is characterized by a color. There are four different colors (see Figure 6.1):

- *Red*: the patient is in critical conditions and needs to be treated immediately,
- *Yellow*: the patient is in potential critical conditions and it needs to be visited within 15 minutes after its arrival,
- *Green*: the patient is not in critical condition, it is visited when is possible,
- *White*: the patient is in normal-life conditions, it is the lowest priority code and it is served only after the other codes.

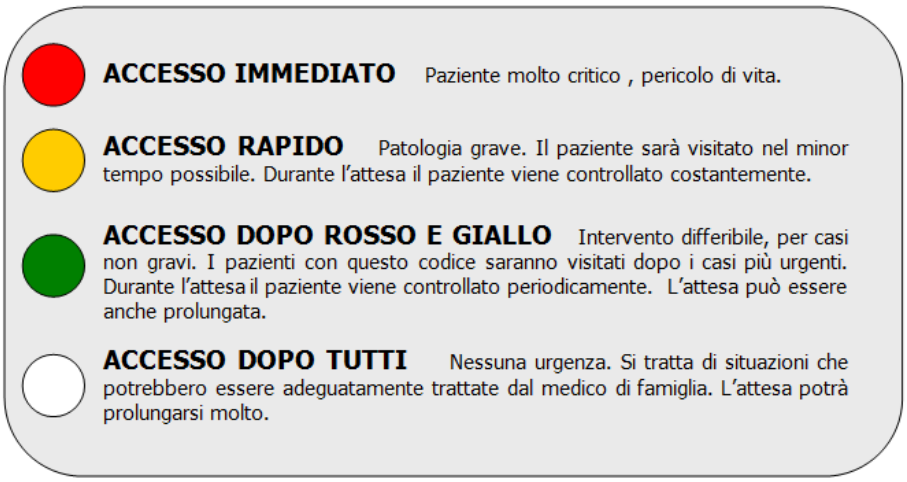

Figure 6.1: *Severity codes description.*

The severity code can change during all the process, after a careful examination the patient life conditions can be judged equal/better/worse than the initial ones. For the sake of simplicity we considered the codes as static, i.e., they are defined once and for all upon

patient arrival. As a matter of fact expert data analysts considered this possibility small enough to be neglected. We didn't consider the white code and we included it in the green code class. For a medium size emergency room the percentage of white code arrivals is considered to be $2-3\%$ and thus negligible. The arrivals are assumed to follow a Poisson distribution with rate λ. For a medium size emergency room is generally impossible to have only one patient "in service" at a time. For this reason it is very hard to determine accurately dependent arrival rates for each code. To handle this problem we analyzed the order of the arrivals looking at the entry time-exit time of each patient.For example we saw that it is unlikely to have two or more red codes in a row; this means that when a red code is "in service" its arrival is lower with respect to the case when other codes are "in service". The service times are exponentially distributed with rate μ. We determined reasonable rates using the data coming from the data analysts and general rules of practice. Generally an emergency room is considered to be in critical conditions when the number of patients is equal to the 91% of the historical data for the same period. Even when an emergency room is saturated the red codes must always be treated immediately. The holding costs c_s represent the cost of non treating the patient in terms of human life. Going from green code to red code the cost increases because the severity increases too. The values given to the holding costs are:

$$c = \begin{bmatrix} 1 \\ 50 \\ 1000 \end{bmatrix}.$$

The first row is referred to the arrival of a green code, the second row is a yellow code arrival and the last one is the red code. The columns indicates what code is "in service": the first column represents the green code "in service" and so on. The arrival rates [patients/h] used are:

$$\Lambda = \begin{bmatrix} 3.454 & 2.01 & 0.269 \\ 1.715 & 1.65 & 0.065 \\ 0.11 & 0.08 & 0.033 \end{bmatrix}.$$

The service rate vector is:

$$\mu = \begin{bmatrix} 7 \\ 5 \\ 2 \end{bmatrix}.$$

There are three severity codes and six possible policies depending on how the classes are sorted:

- Policy 1: Red, Yellow, Green,
- Policy 2: Yellow, Red, Green,
- Policy 3: Green, Red, Yellow,
- Policy 4: Green, Yellow, Red,
- Policy 5: Yellow, Green, Red,
- Policy 6: Red, Green, Yellow.

We simulated the behavior of the emergency room system using a fixed time increment simulation. The state of the system is described at each time increment. The simulation length is 1000 hours and it is repeated ten times in order to derive meaningful statistical parameters and to account for randomness. The simulations use random seeds and are therefore completely independent of each other. The results are shown in Table 6.1.

Policy 1	Policy 2	Policy 3	Policy 4	Policy 5	Policy 6
35.763	44.095	241.957	253.334	305.812	83.94
31.948	41.216	246.138	256.119	309.815	75.761
30.981	45.387	235.289	260.731	311.764	85.895
29.995	48.753	240.783	267.543	299.918	88.771
33.356	42.561	238.613	268.154	310.958	81.454
38.786	48.564	239.78	255.517	308.756	84.018
32.781	50.811	240.118	256.366	305.801	89.598
33.654	51.756	245.761	250.134	303.774	78.854
37.128	48.431	239.118	257.654	301.781	82.123
35.342	45.442	236.541	261.352	303.718	81.915
Mean	Mean	Mean	Mean	Mean	Mean
33.973	46.702	240.41	258.69	306.21	83.233

Table 6.1: *Simulation results for the different policies.*

Policy 1 seems to be the best policy available. However before drawing conclusions, to make sure that the simulation results are statistically significant a hypothesis test was carried out. We used the Tukey's test for means multiple comparisons. The hypothesis is defined as follows:

$$H_0 : \textit{All the policies are equally convenient.}$$
$$H_1 : \textit{All the policies are not equally convenient.}$$

The null hypothesis was rejected confirming what we assumed at the beginning. *Policy 1* is the optimal policy that minimizes the total holding cost (see Figure 6.2 and Figure 6.3).

Comparisons for Cost

Tukey Pairwise Comparisons: Pol

Grouping Information Using the Tukey Method and 99% Confidence

Pol	N	Mean	Grouping
5	10	306,210	A
4	10	258,690	B
3	10	240,410	C
6	10	83,233	D
2	10	46,702	E
1	10	33,973	F

Means that do not share a letter are significantly different.

Figure 6.2: *Tukey's test for means multiple comparisons.*

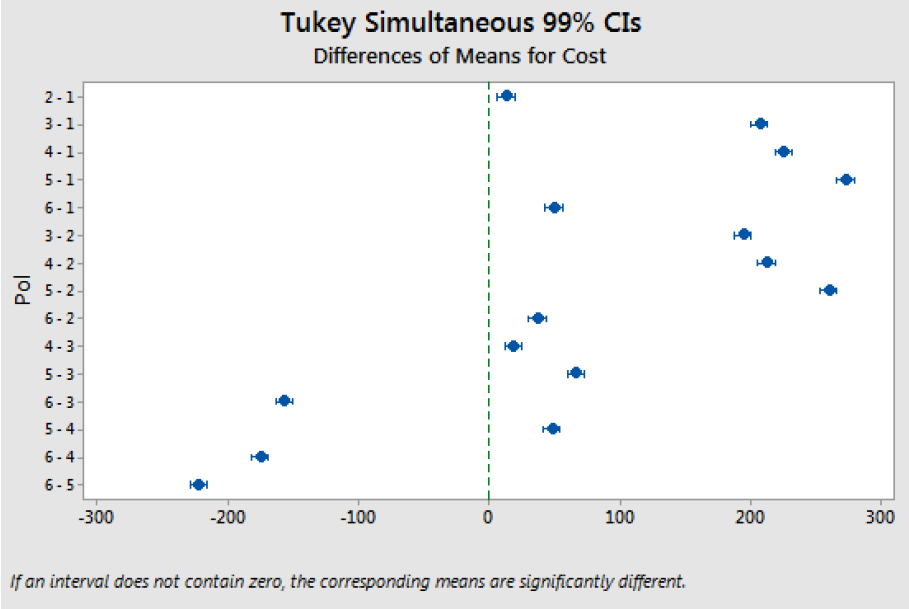

Figure 6.3: *Tukey's test for means multiple comparisons.*

Chapter 7

Conclusions

We now summarize the major contributions obtained for theory and practice, also we mention the possible future developments starting from the results we achieved.

7.1 Contributions to theory

We obtained very important results for the advancement of queuing theory. In fact we formulated a generalized $c\mu$ rule able to predict the optimal scheduling policy when all the assumptions and requirements are met. We obtained results for a two-class products/-clients problem, then moving from the this simpler case we extended the validity to a more general case with n-class. The results are obtained via cost minimization using different methods (such as fluid model and modified policy iteration). They were then analytically proven by formulating the Markov decision process of the problem. We discovered that the previous rule was not enough accurate when the arrival rates are dependent on the job in service because it did not account for their effects. Moreover we defined the stability region and how the optimality regions evolves with different parameters. All of these aspects bring in a big step forward for the knowledge of the $c\mu$ since disclose the root causes of the limitations suffered by the $c\mu$ rule as the optimal scheduling policy, as well as none proposed a better alternative policy.

7.2 Contributions to practice

The general rule we formulated could represent an important scheduling policy in the emergency room process. The results obtained so far are quite interesting. This thesis could be a starting point towards more accurate studies regarding the scheduling optimization in this field.

7.3 Further developments

Given the novelty of the topic studied there are several directions in which it is possible to direct future research. Arrival rates effects were not accounted before, it is then possible to extend the results obtained to more general cases with the inclusion of other parameters. One direction is to consider the abandonment rate of the jobs once they are waiting in the queue, it is a reasonable assumption when considering a post office or a general service

where the abandonment of the clients is a plausible option. Another possibility is to consider more servers and add different kind of costs such as setup costs and delays costs in order to better approximate a real manufacturing system. The data analysts that provided us with the data showed some interest regarding the application of the rule on other aspects of the complex emergency room process. One possible direction is the optimization of the decision whether to transfer the patient to another hospital or to move it to another department. In the last years in the hospital emergency rooms rose the necessity to divide the patients belonging to the same color code in different subgroups characterized by similar health conditions. For example if two yellow codes are waiting in the queue how can we decide which one should be treated first? To do so it is required a precise medical assessment on the severity and a general awareness about the time required to process the patient. With the general rule we formulated it could be possible to go beyond the simple code classification treating each injury as a standalone class. In this way we could even further optimize the whole process in order to achieve better performances.

Bibliography

[1] D. R. Cox and W. Smith. *Queues.* CRC Press, 1991.

[2] H. C. Raaijmakers. Optimal scheduling for a multiclass queue with state dependent arrivals, 2017.

[3] W. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[4] G. P. Klimov. Time-sharing service systems I. *Theory of Probability & Its Applications*, 19(3):532–551, 1975.

[5] J. M. Harrison. Dynamic scheduling of a multiclass queue: discount optimality. *Operations Research*, 23(2):270–282, 1975.

[6] D-W. Tcha and S. R. Pliska. Optimal control of single-server queuing networks and multi-class M/G/1 queues with feedback. *Operations Research*, 25(2):248–258, 1977.

[7] P. Varaiya C. Buyukkoc and J. Walrand. The $c\mu$ rule revisited. *Advances in Applied Probability*, 17(1):237–238, 1985.

[8] M. Kijima T. Hirayama and S. Nishimura. Further results for dynamic scheduling of multiclass G/G/1 queues. *Journal of Applied Probability*, 26(3):595–603, 1989.

[9] C. Giat R. Atar and N. Shimkin. The $c\mu/\theta$ rule for many-server queues with abandonment. *Operations Research*, 58(5):1427–1439, 2010.

[10] E. Arda Sisbot and J. J. Hasenbein. Joint routing and scheduling control in a two-class network with a flexible server. *Queuing Systems*, 88(1):73–97, 2010.

[11] J. A. Van Mieghem. Joint routing and scheduling control in a two-class network with a flexible server. *The Annals of Applied Probability*, 5(3):809–833, 1995.

[12] S. Asmussen P. A. Ernst and J. J. Hasenbein. Stability and busy periods in a multiclass queue with state-dependent arrival rates. 2017.

[13] D. P. Bertsekas. *Dynamic programming and optimal control*, volume 2. Athena Scientific, Belmont, MA, third edition, 2005.

[14] D. Bello and G. Riano. Linear programming solvers for markov decision processes. *2006 IEEE Systems and Information Engineering Design Symposium*, 2006.

Appendix

Two-class system fluid model

```
%% TWO-CLASS SYSTEM, FLUID MODEL LINEAR PROGRAMMING  %%

close all
clear
clc

%% DATA DECLARATION %%

T=10;N=1500;dt=T/N;
c1=1; c2=1; C=[c1;c2]; %cost array

lam11=10;lam12=8;lam21=6;lam22=8;
LAM=[lam11,lam12;lam21,lam22]; %arrival rates

mu1=24;mu2=25;MU=[mu1;mu2]; %service rates

Qin1=15;Qin2=15;Qfin1=0;Qfin2=0; %Queues length

CMU=[c1*mu1;c2*mu2];

%% STABILITY %%

M=LAM/(diag(MU));
EIG=abs(eig(M));  %matrix eigenvalues
L=max(EIG)        %maximum eigenvalue
if max(EIG)>=1
    msg='The set of parameters is not stable';
    error(msg)
end

%% Aeq & beq %%

Aeq=zeros(2*N,2*(2*N-1));
beq=zeros(2*N,1);
```

```
%n=0:
Aeq(1,1)=-(lam11-mu1)*dt;
Aeq(1,N+1)=-lam12*dt;
Aeq(1,2*N+1)=1;

Aeq(N+1,1)=-lam21*dt;
Aeq(N+1,N+1)=-(lam22-mu2)*dt;
Aeq(N+1,3*N)=1;

beq(1)=Qin1;
beq(N+1)=Qin2;

%n=N;
Aeq(N,N)=(lam11-mu1)*dt;
Aeq(N,2*N)=lam12*dt;
Aeq(N,3*N-1)=1;

Aeq(2*N,N)=lam21*dt;
Aeq(2*N,2*N)=(lam22-mu2)*dt;
Aeq(2*N,2*(2*N-1))=1;

%n=1:N-1
for i=2:(N-1)
    Aeq(i,i)=-(lam11-mu1)*dt;
    Aeq(i,N+i)=-lam12*dt;
    Aeq(i,2*N+i)=1;
    Aeq(i,2*N+i-1)=-1;
end
for k=2:(N-1)
    Aeq(N+k,k)=-lam21*dt;
    Aeq(N+k,N+k)=-(lam22-mu2)*dt;
    Aeq(N+k,3*N-1+k)=1;
    Aeq(N+k,3*N-2+k)=-1;
end

%% A & b %%

A=zeros(N,2*(2*N-1));
b=ones(N,1);

for j=1:N
    A(j,j)=1;
    A(j,N+j)=1;
end

%% UPPER AND LOWER BOUNDS %%
```

```
ub=[];
lb=zeros(2*(2*N-1),1);

%% OBJECTIVE FUNCTION %%

f=zeros(2*(2*N-1),1);
f((2*N+1):(3*N-1))=C(1)*dt;
f(3*N:end)=C(2)*dt;

%% SOLUTION %%

[x,fval]=linprog(f,A,b,Aeq,beq,lb,ub);

%% PLOT RESULTS %%

TimeQ=0:dt:T;     %time discretization for Q
TimeU=0:dt:T-dt;  %time discretization for u

figure()
subplot(2,1,1) %queue Q evolution
plot(TimeQ,[Qin1;x(2*N+1:3*N-1);Qfin1],'b',...
     TimeQ,[Qin2;x(3*N:end);Qfin2],'r','linewidth',1);
legend('Q_1','Q_2');
title('Draining evolution');
xlabel('Time [h]','fontweight','bold');
ylabel('Q(t)','fontweight','bold');
grid on

subplot(2,1,2) %control u evolution
plot(TimeU,x(1:N),'b',TimeU,x(N+1:2*N),'r','linewidth',1);
legend('u_1','u_2');
title('Policy evolution')
xlabel('Time [h]','fontweight','bold');
ylabel('u(t)','fontweight','bold');
ylim([0,1.1]);
grid on

%% EQUAL COSTS PLOT %%

MU1=lam11+lam21; MU2=lam12+lam22;
N1=2000;
lim=2*max(mu1,mu2);
x=linspace(lam11,lim,N1);

Bis=@(x) x;
a=lam22;
c=1;
```

```
d=-lam11;
b=MU2*(c*MU1+d)-a*MU1;
Par=@(x) (a.*x+b)./(c.*x+d); %stability hyperbola equation
Nbis=@(x) x-(MU1-MU2);
Pfitt=(b-d*lim)/(lim-a); %fictitious point for the plot

xPar=linspace(Pfitt,lim,N1);
x1=linspace(MU1,lim,N1);
x2=linspace(Pfitt,MU1,N1);

figure(2)
plot(x,Bis(x),'k--');
hold on;
xlabel('\mu_1','fontweight','bold');
ylabel('\mu_2','fontweight','bold');
title('Equal Costs');
xlim([lam11 lim]);
ylim([lam22 lim]);
patch([lam11,lam11,xPar,lim],[lam22,lim,Par(xPar),lam22],...
      [0,0,zeros(1,length(xPar)),0],'facecolor','k','
         FaceAlpha',.4); %unstable zone
patch([x1,x1(end:-1:1)],[Nbis(x1),Par(x1(end:-1:1))],...
      [zeros(1,length(x1)),zeros(1,length(x1))],'facecolor','
         b','FaceAlpha',.4); %class 1 zone
patch([x1,lim,x2],[Nbis(x1),lim,Par(x2)],[zeros(1,length(x1))
   ,0,zeros(1,length(x2))],...
      'facecolor','r','FaceAlpha',.4); %class 2 zone
grid on
text(lim-15,lim-25,'Class 1','Color','b','Fontsize',18,'
   FontAngle','italic');
text(lim-25,lim-10,'Class 2','Color','r','Fontsize',18,'
   FontAngle','italic');

%% COST EFFECT PLOT %%

N2=1000;
lim2=mu2/mu1*5;
rapp=linspace(0,7,lim);
MU11=lam11-lam12; MU22=lam22-lam21;
rappc=MU22/MU11;
P2=(mu1-(MU1-MU2))/mu1;

m=(1-rappc)/(P2-rappc);
q=1-m*P2;
Rev=@(x) m.*x+q;

stab=Par(mu1)/mu1;
```

```
Pfitt2=(lim2-q)/m;

figure(3)
plot(rapp,Bis(rapp),'k--');
hold on;
xlabel('\mu_2/\mu_1','fontweight','bold');
ylabel('c_1/c_2','fontweight','bold');
title('Costs Effect');
xlim([0 3.25]);
ylim([0 3.25]);
grid on;
plot(rapp,Rev(rapp),'k-')


patch([0,stab,stab,0],[0,0,lim2,lim2],[0,0,0,0],...
      'facecolor','k','FaceAlpha',.4); %unstable region
if isnan(rappc)&&isnan(m) %lambda21=lambda22 case
 patch([stab,lim2,lim2,stab],[stab,lim2,0,0],[0,0,0,0],...
      'facecolor','r','FaceAlpha',.4);
 patch([stab,lim2,stab],[stab,lim2,lim2],[0,0,0],...
      'facecolor','b','FaceAlpha',.4);
 text(stab+1,lim2-1.3,'Class 1','Color','b','Fontsize',18,'
    FontAngle','italic');
 text(lim2-2,1.5,'Class 2','Color','r','Fontsize',18,'
    FontAngle','italic');

elseif isnan(m) %dependent case and lambda11=lambda12 case
    m=1;
    q=1-P2;
    plot(rapp,rapp+q,'k-');
    patch([stab,lim2,lim2,stab],[stab+q,lim2+q
       ,0,0],[0,0,0,0],...
      'facecolor','r','FaceAlpha',.4);
    patch([stab,lim2,stab],[stab+q,lim2+q,lim2],[0,0,0],...
      'facecolor','b','FaceAlpha',.4);
    text(1,2.2,'Class 1','Color','b','Fontsize',18,'FontAngle
       ','italic');
    text(2.2,1,'Class 2','Color','r','Fontsize',18,'FontAngle
       ','italic')

else %lambda11\neqlambda12 and lambda21\neqlambda22 case
 patch([stab,Pfitt2,lim2,lim2,stab],[Rev(stab),lim2,lim2
    ,0,0],[0,0,0,0,0],...
      'facecolor','r','FaceAlpha',.4);
 patch([stab,Pfitt2,stab],[Rev(stab),lim2,lim2],[0,0,0],...
      'facecolor','b','FaceAlpha',.4);
 text(1,2.2,'Class 1','Color','b','Fontsize',18,'FontAngle','
```

```
    italic');
 text(2.2,1,'Class 2','Color','r','Fontsize',18,'FontAngle','
    italic');
end
```

Three-class system fluid model

```
%% THREE-CLASS SYSTEM, FLUID MODEL LINEAR PROGRAMMING %%

close all
clear
clc

%% DATA DECLARATION %%

T=5;N=1000;dt=T/N;
c1=1;c2=1;c3=1;C=[c1;c2;c3]; %cost array

lam11=10;lam12=6;lam13=5;lam21=4;lam22=8;lam23=6;
lam31=4;lam32=5;lam33=6;
LAM=[lam11,lam12,lam13;lam21,lam22,lam23;
     lam31,lam32,lam33]; %arrival rates

mu1=25;mu2=26;mu3=25;MU=[mu1;mu2;mu3]; %service rates

Qin1=10;Qin2=10;Qin3=10;Qfin1=0;Qfin2=0;Qfin3=0; %Queues
   length

CMU=[c1*mu1;c2*mu2;c3*mu3];

%% STABILITY %%

M=LAM/(diag(MU));
EIG=abs(eig(M));  %matrix eigenvalues
L=max(EIG)        %maximum eigenvalue
if max(EIG)>=1
    msg='The set of parameters is not stable';
    error(msg)
end

%% Aeq & beq %%

Aeq=zeros(3*N,3*(2*N-1));
beq=zeros(3*N,1);

%n=0:
Aeq(1,1)=-(lam11-mu1)*dt;
Aeq(1,N+1)=-lam12*dt;
Aeq(1,2*N+1)=-lam13*dt;
Aeq(1,3*N+1)=1;

Aeq(N+1,1)=-lam21*dt;
```

```
Aeq(N+1,N+1)=-(lam22-mu2)*dt;
Aeq(N+1,2*N+1)=-lam23*dt;
Aeq(N+1,4*N)=1;

Aeq(2*N+1,1)=-lam31*dt;
Aeq(2*N+1,N+1)=-lam32*dt;
Aeq(2*N+1,2*N+1)=-(lam33-mu3)*dt;
Aeq(2*N+1,5*N-1)=1;

beq(1)=Qin1;
beq(N+1)=Qin2;
beq(2*N+1)=Qin3;

%n=N;
Aeq(N,N)=(lam11-mu1)*dt;
Aeq(N,2*N)=lam12*dt;
Aeq(N,3*N)=lam13*dt;
Aeq(N,4*N-1)=1;

Aeq(2*N,N)=lam21*dt;
Aeq(2*N,2*N)=(lam22-mu2)*dt;
Aeq(2*N,3*N)=lam23*dt;
Aeq(2*N,5*N-2)=1;

Aeq(3*N,N)=lam31*dt;
Aeq(3*N,2*N)=lam32*dt;
Aeq(3*N,3*N)=(lam33-mu3)*dt;
Aeq(3*N,6*N-3)=1;

%n=1:N-1
for i=2:(N-1)
    Aeq(i,i)=-(lam11-mu1)*dt;
    Aeq(i,N+i)=-lam12*dt;
    Aeq(i,2*N+i)=-lam13*dt;
    Aeq(i,3*N+i)=1;
    Aeq(i,3*N+i-1)=-1;
end
for k=2:(N-1)
    Aeq(N+k,k)=-lam21*dt;
    Aeq(N+k,N+k)=-(lam22-mu2)*dt;
    Aeq(N+k,2*N+k)=-lam23*dt;
    Aeq(N+k,4*N-1+k)=1;
    Aeq(N+k,4*N-2+k)=-1;
end

for j=2:(N-1)
    Aeq(2*N+j,j)=-lam31*dt;
```

```
    Aeq(2*N+j,N+j)=-lam32*dt;
    Aeq(2*N+j,2*N+j)=-(lam33-mu3)*dt;
    Aeq(2*N+j,5*N-2+j)=1;
    Aeq(2*N+j,5*N-3+j)=-1;
end

%% A & b %%

A=zeros(N,3*(2*N-1));
b=ones(N,1);

for l=1:N
    A(l,l)=1;
    A(l,N+l)=1;
    A(l,2*N+l)=1;
end

%% UPPER AND LOWER BOUNDS %%

ub=[];
lb=zeros(3*(2*N-1),1);

%% OBJECTIVE FUNCTION %%

f=zeros(3*(2*N-1),1);
f((3*N+1):(4*N-1))=C(1)*dt;
f((4*N):(5*N-2))=C(2)*dt;
f((5*N-1):end)=C(3)*dt;

%% SOLUTION %%

[x,fval]=linprog(f,A,b,Aeq,beq,lb,ub);

%% PLOT RESULTS %%

TimeQ=0:dt:T;     %time discretization for Q
TimeU=0:dt:T-dt; %time discretization for u

figure()
subplot(2,1,1) %queue Q evolution
plot(TimeQ,[Qin1;x(3*N+1:4*N-1);Qfin1],'b',TimeQ,[Qin2;x(4*N
   :5*N-2);Qfin2],'r',...
      TimeQ,[Qin3;x(5*N-1:end);Qfin3],'g','linewidth',1);
legend('Q_1','Q_2','Q_3');
title('Draining Evolution');
xlabel('Time [h]','fontweight','bold');
ylabel('Q(t)','fontweight','bold');
```

```
grid on

subplot(2,1,2) %control u evolution
plot(TimeU,x(1:N),'b',TimeU,x(N+1:2*N),'r',TimeU,x(2*N+1:3*N)
   ,'g','linewidth',1);
legend('u_1','u_2','u_3');
title('Policy Evolution')
xlabel('Time [h]','fontweight','bold');
ylabel('u(t)','fontweight','bold');
ylim([0,1.1]);
grid on
```

Linear programming Markov decision process

```
%% SETUP MARKOV DECISION PROBLEM %%

function [ prob_discr_u1 , prob_discr_u2 ,g_d ,g] =
   MDP_problem_setup_SIMO (N, Lambda0 , Lambda ,Mu ,C)

N_states = (N +1)^2; % Number of states of system
pos_a = (N +1)*N; % n.o. possible arrivals--------------forse
    va +1

% DEFINE TRANSTION RATES

% Define ( continous ) transition times
v_u1 = zeros(N_states ,1) ; % Column to store transition
   rates
v_u2 = zeros(N_states ,1) ; % Column to store transition
   rates

% control : u = 1
v_u1(: ,1) = Lambda(1 ,1)+ Lambda(2 ,1)+Mu(1) ; % state
   matrix where the roows are Q2 and columns Q1
%starting from (0,0) in left part

for i = 1:N+1:N_states % positions  1-6-11-16 (N=4)
    v_u1(i,1)= v_u1(i,1)-Mu(1);
end
v_u1(N_states-N,1)= v_u1(N_states-N,1)-Lambda(2,1); %
   position 21 (N=4)
for i = N+1:N+1:N_states
    v_u1(i,1)= v_u1(i,1)-Lambda(1,1);% positions 5-10-15-20 (
       N=4)
end
v_u1(N_states,1) = v_u1(N_states,1)-Lambda(2,1);% position 25
    (N=4)
for i = 1:N-1
    v_u1(N_states-N+i,1) = v_u1(N_states-N+i,1)- Lambda(2,1);
        %positions 22-23-24 (N=4)
end

% control : u = 2
v_u2(: ,1) = Lambda(2 ,2)+ Lambda(1,2)+Mu(2) ;
for i = 1:N
    v_u2(i,1)= v_u2(i,1)-Mu(2); %positions 1-2-3-4 (N=4)
end
v_u2(N+1,1) = v_u2(N+1,1) - Lambda(1,2) - Mu(2); % position 5
    (N=4)
```

```
passo = N+1;
for i = (N+1)*2:passo:N_states
    v_u2(i,1)= v_u2(i,1)-Lambda(1,2);% positions 10-15-20 (N
       =4)
end
v_u2(N_states) = v_u2(N_states,1)-Lambda(2,2);% position 25 (
   N=4)
for i = 1:N
    v_u2(N_states-N-1+i,1) = v_u2(N_states-N+i,1)- Lambda
       (2,2); % positions 21-22-23-24 (N=4)
end
v_u2(N_states-1) = v_u2(N_states-1)+Lambda(2,2)+Lambda(1,2);
% Correct for state (0 ,0)
v_u1(1,1) = Lambda0(1) + Lambda0(2) ;
v_u2(1,1) = Lambda0(1) + Lambda0(2) ;
% Define the largest possible transition time
v = max( max( v_u1 ),max( v_u2 ));

% DEFINE CTMDP PROBABILITIES
trans_u1 = zeros(N_states , N_states); % Empty transition
   matrix u1
trans_u2 = zeros(N_states , N_states); % Empty transition
   matrix u2
prob_u1 = zeros(N_states , N_states);  % Empty probability
   matrix u1
prob_u2 = zeros(N_states , N_states);  % Empty probability
   matrix u2

% TRANSITION MATRIX FOR  U1
%Starting from (0,0)
trans_u1(1,2) = Lambda0(1);
trans_u1(1,N+1+1) = Lambda0(2);

%I put the values for Mu_1
for i = 1:N
    trans_u1(i+1,i) = Mu(1);                    % from 2,1 to
       5,4 (N=4)
   for j = 1:N
       trans_u1(j*(N+1)+i+1,j*(N+1)+i) = Mu(1); % from 7,6 to
           10,9 from 12,11 to 15,14 from 17,16 to 20,19 (N=4)
   end
end

% Values Lambda(1,1)
for i = 1:N-1
    trans_u1(i+1,i+2) = Lambda(1,1); % from 2,3 to 4,5 (N=4)
end
```

```
for j = 1:N
    for i = 1:N
    trans_u1(j*(N+1)+i,j*(N+1)+i+1) = Lambda(1,1); % from
       6,7-9,10 11,12-14,15-16,17-19,20-21,22-24,25
    end
end

% Values Lambda(2,1)
trans_u1(1,N+2) = Lambda0(2);
for i = 2:(N+1)*N-1
    trans_u1(i,i+(N+1)) = Lambda(2,1);
end



% TRANSITION MATRIX FOR U2

% Values Lambda(1,2)
V_L_12 = Lambda(1,2)*ones(N_states-1,1);
for i = 1:N
    V_L_12(i*N+i) = 0;
end
trans_u2 = diag(V_L_12,1);


for i = 1:(N+1)*N
    trans_u2(N+1+i,i) = Mu(2);  %  Mu2
end
% Values Lambda (2,2)
V_L_22 = Lambda(2,2)*ones(N_states-N-1,1);
m22 = diag(V_L_22,N+1);
trans_u2 = trans_u2 + m22;

% values Lambda 0
trans_u2(1,2) = Lambda0(1);
trans_u2(1,N+1+1) = Lambda0(2);

% Convert arrival / competion matrices to probability
   matrices .
for i = 1: N_states
prob_u1(i ,:) = trans_u1(i ,:) ./ v_u1(i);
prob_u2(i ,:) = trans_u2(i ,:) ./ v_u2(i);
end
```

```
% CTDMP COST FUNCTION

g = zeros(N_states ,1) ; % Matrix to store costs
Q1 = zeros(N_states,1);
Q2 = zeros(N_states,1);
r = [];
for i = 0:N
    r(i+1)= i;
end
ve = r';
for j = 0:N
    for i = 1:N+1
    Q1(j*(N+1)+i) = ve(i);
    end
end
for j = 0:N
    for i = 1:N+1
    Q2(j*(N+1)+i) = j;
    end
end

g = [C(1)*Q1+C(2)*Q2,C(1)*Q1+C(2)*Q2];

% APPLY UNIFORIMZATION TO CREATE DTMDP

% Cost function
g_d = g./v;

% Transition probabilities
prob_discr_u1 = zeros(size( prob_u1 ));
prob_discr_u2 = zeros(size( prob_u2 ));
for i = 1: N_states
for j = 1: N_states
if i == j
prob_discr_u1(i,j) = ( v_u1(i)./v) * prob_u1(i,i) + 1 - (
   v_u1(i)./v);
prob_discr_u2(i,j) = ( v_u2(i)./v) * prob_u2(i,i) + 1 - (
   v_u2(i)./v);
else
prob_discr_u1(i,j) = ( v_u1(i)./v) * prob_u1(i,j);
prob_discr_u2(i,j) = ( v_u2(i)./v) * prob_u2(i,j);
end
end
end
end

%CHOOSE THE PARAMETERS
```

```
% Maximal buffer size ( state space truncation )
N = 40;
% System parameters
% Dependent arrival rates [ lots / hour ]
Lambda = [23 12;
6 13];

% Arrival rates when server is off
Lambda0(1) = max( Lambda(1 ,1) ,Lambda(1 ,2));
Lambda0(2) = max( Lambda(2 ,1) ,Lambda(2 ,2));

% Processing rates [ lots / hour ]
Mu(1) = 100;
Mu(2) = 147;

% Holding costs [ dollars / lot / hour ]
C(1) = 1.6;
C(2) = 1;

% CHECK THE STABILITY OF SYSTEM (abs(Lambda/Mu) < 1)
rho = Lambda * inv(diag(Mu));
EIG = abs(eig(rho));
if max(EIG) < 1
    disp('the system is stable')
else
    disp('the system is unstable please change parameters')
end

% Print theoretical results
max_EIG = max(EIG );
disp ([ 'The traffic intensity , or spectral density of the
   system is: ' ...
num2str( max_EIG )])
fprintf ('\n')

% CHECK THE CU RULE SUGGESTION
CU =[C(1)*Mu(1) C(2)*Mu(2)] ;
a = max(CU);
dim = size(CU);
n = 0;
for i= 1:dim(2)
    n= n+1;
    if CU(i) == a
        break
    end
end
```

```
disp(['the product that must be served for first in according
    to CU rule would be ',num2str(n)])

%% CREATE MODEL STRUCTURE

N_states = (N +1) ^2; % Number of states ( queue length
   combinations )
pos_a = (N +1) *N; % n.o. possible arrivals

% Call to problem setup function
[ prob_discr_u1 , prob_discr_u2 ,g_d ,g] =
   MDP_problem_setup_SIMO(N, Lambda0 , Lambda ,Mu ,C);

%% FORMULATE LINEAR PROGRAM

% Parameters
% g_d = discrete time cost function
% prob_discr_u1 = discrete time probability matrix u1
% prob_discr_u2 = discrete time probability matrix u2


% OBJECTIVE FUNCTION

OB = [ g_d(: ,1)' g_d(: ,2)'];

% EQUALITY CONSTRAINTS
Aeq = zeros (N_states -1 ,2* N_states ); % Empty matrix ,
   except for last constraint
% Everything on the right of equals sign ( probabilities )
for i = 1: N_states
for j = 1: N_states
Aeq (i,j) = -prob_discr_u1 (j,i);
Aeq (i,j+ N_states ) = -prob_discr_u2 (j,i);
end
Aeq (i,i) = Aeq (i,i) +1;
Aeq (i,i+ N_states ) = Aeq (i,i+ N_states ) +1;
end
beq = zeros (N_states ,1) ;

% Add final equality constraint
Aeq_f = zeros (1, N_states *2) ;
for i = 1: N_states
Aeq_f (1,i) = 1;
Aeq_f (1,i+ N_states ) = 1;
end
beq_f = 1;
```

```
Aeq = [ Aeq ; Aeq_f ];
beq = [ beq ; beq_f ];

% INEQUALITY CONSTRAINTS
A = [];
b = [];

% UPPER AND LOWER BOUNDS
% All larger than zero
lb = zeros (1, N_states *2) ;
ub = [];


% Optimization problem
[q,fval , exitflag , output ] = linprog (OB,A,b,Aeq ,beq ,lb
    ,ub);


% Write results to usuable c_frac format          % y_class
    would be y(i,u) divided between class 1 and 2
y_class = zeros (N_states ,2) ;
y_class (: ,1) = q(1: N_states ); % vector for class 1
y_class (: ,2) = q( N_states +1:2*N_states ); % vector for
    class 2


%% ANALYZING RESUTLS

% Stationary Distribution
PI = zeros (N_states ,1) ;
for i = 1: N_states
PI(i) = y_class(i ,1) + y_class(i ,2) ;
end

% Deterministic rule
% Can be used because our MDP is irreducible1____________It
    is f(i,u)
effe = zeros (N_states ,2) ;
for i = 1: N_states
effe(i ,1) = y_class(i ,1) /PI(i);
effe(i ,2) = y_class(i ,2) /PI(i);
end
% Generate control matrix in queue form
F1 = zeros (N+1,N +1) ;
F2 = zeros (N+1,N +1) ;
Y1_frac = zeros (N+1,N +1);
Y2_frac = zeros (N+1,N +1);
```

```
PI = zeros (N+1,N +1) ;
g_dist = zeros (N+1,N +1) ;

for i = 1:N+1
% Optimal control
F1(i ,:) = effe(1+(i -1) *(N +1) :i*(N +1) ,1); % for class 1
F2(i ,:) = effe(1+(i -1) *(N +1) :i*(N +1) ,2); % for class 2
% Fractions
Y1_frac(i ,:) = y_class(1+(i -1) *(N +1) :i*(N +1) ,1);
Y2_frac(i ,:) = y_class(1+(i -1) *(N +1) :i*(N +1) ,2);
% Deterministic Distribution
PI(i ,:) = PI(1+(i -1) *(N+1) :i*(N+1) );
% Holding cost distribution
g_dist(i ,:) = g(1+(i -1) *(N+1) :i*(N+1) ,1);
end



%% PLOTTING RESULTS

% Control routing rule
figure()
axis ([0 N+1 0 N+1]) ;
xlabel ('Queue 1');
ylabel ('Queue 2');
hold on
grid on
% Control 1
for i = 1:N+1
for j = 1:N+1
if F1(i,j) >= 0.99
plot (j,i,'sb ')
elseif isnan (F1(i,j)) % Error value due to truncation
plot (j,i,'sy ')
legend('product 1')
end
end
end
% Control 2
for i = 1:N+1
for j = 1:N+1
if F2(i,j) >= 0.99
plot (j,i,'sr ')
elseif isnan (F2(i,j))
plot (j,i,'sy ')
end
end
```

```
end

figure()
x = [1:1:N_states];
f1 =[];
f2 =[];
for i = 1:N_states
    f1(i) = effe(i,1);
    f2(i) = effe(i,2);
end
for i = 1:N_states
    if f1(i) > 1
        f1(i) =1;
    end
    if f2(i) < 0
        f2(i) = 1;
    end
end

%% LINEAR PROGRAMMING %%

close all
clear all
clc

%CHOOSE THE PARAMETERS

% Maximal buffer size ( state space truncation )
N = 40;
% System parameters
% Dependent arrival rates [ lots / hour ]
Lambda = [23 12;
6 13];

% Arrival rates when server is off
Lambda0(1) = max( Lambda(1 ,1) ,Lambda(1 ,2));
Lambda0(2) = max( Lambda(2 ,1) ,Lambda(2 ,2));

% Processing rates [ lots / hour ]
Mu(1) = 100;
Mu(2) = 147;

% Holding costs [ dollars / lot / hour ]
C(1) = 1.6;
C(2) = 1;

% CHECK THE STABILITY OF SYSTEM (abs(Lambda/Mu) < 1)
rho = Lambda * inv(diag(Mu));
```

```
EIG = abs(eig(rho));
if max(EIG) < 1
    disp('the system is stable')
else
    disp('the system is unstable please change parameters')
end

% Print theoretical results
max_EIG = max(EIG );
disp ([ 'The traffic intensity , or spectral density of the
   system is: ' ...
num2str( max_EIG )])
fprintf ('\n')

% CHECK THE CU RULE SUGGESTION
CU =[C(1)*Mu(1) C(2)*Mu(2)] ;
a = max(CU);
dim = size(CU);
n = 0;
for i= 1:dim(2)
    n= n+1;
    if CU(i) == a
        break
    end
end
disp(['the product that must be served for first in according
    to CU rule would be ',num2str(n)])

%% CREATE MODEL STRUCTURE

N_states = (N +1) ^2; % Number of states ( queue length
   combinations )
pos_a = (N +1) *N; % n.o. possible arrivals

% Call to problem setup function
[ prob_discr_u1 , prob_discr_u2 ,g_d ,g] =
   MDP_problem_setup_SIMO(N, Lambda0 , Lambda ,Mu ,C);

%% FORMULATE LINEAR PROGRAM

% Parameters
% g_d = discrete time cost function
% prob_discr_u1 = discrete time probability matrix u1
% prob_discr_u2 = discrete time probability matrix u2


% OBJECTIVE FUNCTION
```

```
OB = [ g_d(: ,1)' g_d(: ,2)'];

% EQUALITY CONSTRAINTS
Aeq = zeros (N_states -1 ,2* N_states ); % Empty matrix ,
   except for last constraint
% Everything on the right of equals sign ( probabilities )
for i = 1: N_states
for j = 1: N_states
Aeq (i,j) = -prob_discr_u1 (j,i);
Aeq (i,j+ N_states ) = -prob_discr_u2 (j,i);
end
Aeq (i,i) = Aeq (i,i) +1;
Aeq (i,i+ N_states ) = Aeq (i,i+ N_states ) +1;
end
beq = zeros (N_states ,1) ;

% Add final equality constraint
Aeq_f = zeros (1, N_states *2) ;
for i = 1: N_states
Aeq_f (1,i) = 1;
Aeq_f (1,i+ N_states ) = 1;
end
beq_f = 1;

Aeq = [ Aeq ; Aeq_f ];
beq = [ beq ; beq_f ];

% INEQUALITY CONSTRAINTS
A = [];
b = [];

% UPPER AND LOWER BOUNDS
% All larger than zero
lb = zeros (1, N_states *2) ;
ub = [];


% Optimization problem
[q,fval , exitflag , output ] = linprog (OB,A,b,Aeq ,beq ,lb
   ,ub);


% Write results to usuable c_frac format          % y_class
   would be y(i,u) divided between class 1 and 2
y_class = zeros (N_states ,2) ;
y_class (: ,1) = q(1: N_states ); % vector for class 1
```

```
y_class (: ,2) = q( N_states +1:2*N_states ); % vector for
    class 2


%% ANALYZING RESUTLS

% Stationary Distribution
PI = zeros (N_states ,1) ;
for i = 1: N_states
PI(i) = y_class(i ,1) + y_class(i ,2) ;
end

% Deterministic rule
% Can be used because our MDP is irreducible1___________It
    is f(i,u)
effe = zeros (N_states ,2) ;
for i = 1: N_states
effe(i ,1) = y_class(i ,1) /PI(i);
effe(i ,2) = y_class(i ,2) /PI(i);
end
% Generate control matrix in queue form
F1 = zeros (N+1,N +1) ;
F2 = zeros (N+1,N +1) ;
Y1_frac = zeros (N+1,N +1);
Y2_frac = zeros (N+1,N +1);
PI = zeros (N+1,N +1) ;
g_dist = zeros (N+1,N +1) ;

for i = 1:N+1
% Optimal control
F1(i ,:) = effe(1+(i -1) *(N +1) :i*(N +1) ,1); % for class 1
F2(i ,:) = effe(1+(i -1) *(N +1) :i*(N +1) ,2); % for class 2
% Fractions
Y1_frac(i ,:) = y_class(1+(i -1) *(N +1) :i*(N +1) ,1);
Y2_frac(i ,:) = y_class(1+(i -1) *(N +1) :i*(N +1) ,2);
% Deterministic Distribution
PI(i ,:) = PI(1+(i -1) *(N+1) :i*(N+1) );
% Holding cost distribution
g_dist(i ,:) = g(1+(i -1) *(N+1) :i*(N+1) ,1);
end



%% PLOTTING RESULTS

% Control routing rule
figure()
```

```
axis ([0 N+1 0 N+1]) ;
xlabel ('Queue 1');
ylabel ('Queue 2');
hold on
grid on
% Control 1
for i = 1:N+1
for j = 1:N+1
if F1(i,j) >= 0.99
plot (j,i,'sb ')
elseif isnan (F1(i,j)) % Error value due to truncation
plot (j,i,'sy ')
legend('product 1')
end
end
end
% Control 2
for i = 1:N+1
for j = 1:N+1
if F2(i,j) >= 0.99
plot (j,i,'sr ')
elseif isnan (F2(i,j))
plot (j,i,'sy ')
end
end
end

figure()
x = [1:1:N_states];
f1 =[];
f2 =[];
for i = 1:N_states
    f1(i) = effe(i,1);
    f2(i) = effe(i,2);
end
for i = 1:N_states
    if f1(i) > 1
        f1(i) =1;
    end
    if f2(i) < 0
        f2(i) = 1;
    end
end
```

Modified policy iteration Markov decision process

```
%% MARKOV DECISION PROCESS, MODIFIED POLICY ITERATION %%

close all
clear all
clc

%% DATA DECLARATION %%

N=50;  %truncation
LAM=[21,10; 7,24]; %arrival rates
mu1=30; %service rates
mu2=37;
C=[1.2,1]; %cost array

Stati=(N+1)^2; %total states number

%% TOTAL TRANSITION RATE PER STATE %%

%control 1
MAT1=(LAM(1,1)+LAM(2,1)+mu1)*ones(N+1,N+1);
MAT1(:,1)=LAM(1,1)+LAM(2,1);
MAT1(:,N+1)=LAM(2,1)+mu1;
MAT1(N+1,:)=LAM(1,1)+mu1;
MAT1(N+1,1)=LAM(1,1);
MAT1(N+1,N+1)=mu1;

trate1=[];
for i=1:N+1
    trate1=[trate1,MAT1(i,:)];
end
trate1=trate1';

%control 2
MAT2=(LAM(1,2)+LAM(2,2)+mu2)*ones(N+1,N+1);
MAT2(1,:)=LAM(1,2)+LAM(2,2);
MAT2(N+1,:)=LAM(1,2)+mu2;
MAT2(:,N+1)=LAM(2,2)+mu2;
MAT2(1,N+1)=LAM(2,2);
MAT2(N+1,N+1)=mu2;

trate2=[];
for i=1:N+1
    trate2=[trate2,MAT2(i,:)];
end
trate2=trate2';
```

```
trate=max(max(trate1),max(trate2));

%% TRANSITION RATE MATRIX %%

U1=zeros(Stati,Stati);
U2=zeros(Stati,Stati);
P1=zeros(Stati,Stati);
P2=zeros(Stati,Stati);

%control 1
for i=1:Stati-(N+1)
    U1(i,i+(N+1))=LAM(2,1);
end

for j=1:Stati-1
    if(mod(j,N+1)==0)
        U1(j,j+1)=0;
        U1(j+1,j)=0;
    else
        U1(j,j+1)=LAM(1,1);
        U1(j+1,j)=mu1;
    end
end

%control 2
for i=1:Stati-(N+1)
    U2(i,i+(N+1))=LAM(2,2);
    U2(i+(N+1),i)=mu2;
end

for j=1:Stati-1
    if(mod(j,N+1)==0)
        U2(j,j+1)=0;
    else
        U2(j,j+1)=LAM(1,2);
    end
end

%% PROBABILITIES TRANSITION MATRIX %%

for i=1:Stati
    P1(i,:)=U1(i,:)/trate1(i);
    P2(i,:)=U2(i,:)/trate2(i);
end

%% COST FUNCTION %%
```

```
CFunc=zeros(Stati,2);

a={0:N,0:N};
comb=allcomb(a{:});
CFunc(:,1)=C(1)*comb(:,2)+C(2)*comb(:,1);
CFunc(:,2)=CFunc(:,1); %Costi degli stati indipendenti dalla
   scelta

%% DISCRETIZZATION %%

Cfunc_d=CFunc./trate;

P1_d=U1/trate;
P2_d=U2/trate;
for i=1:Stati
    P1_d(i,i)=1-trate1(i)/trate;
    P2_d(i,i)=1-trate2(i)/trate;
end

%% MODIFIED POLICY

CMU=[C(1)*mu1 C(2)*mu2]
CMUnew=[C(1)*(mu1-(LAM(1,1)-LAM(1,2))),C(2)*(mu2-(LAM(2,2)-
   LAM(2,1)))]


maxiter = 20; %maximum number of iterations
maxiterVI = 1500; %maximum relative value iterations to find
   h_mu
toll = 1e-15; %tolerance for stopping value iterations to
   find h_mu


[~,mx]=max(CMU);
psi0=mx*ones(Stati,1); %greedy policy, cmu policy
psi0(2:N+1)=1; %Q2 empty, class 1 optimal
psi0(N+2:N+1:end)=2; %Q1 empty, class 2 optimal
psi=psi0';

hv0=ones(Stati,1)'; %step 0 initialization
hv0(1)=0;
e=ones(Stati,1);
hv=hv0;

%Modified policy iteration with value iteration
k=0;
```

```
while true
   for l=1:maxiterVI

    %Bellman's equations
     for i=1:Stati
        sum1=0;
        sum2=0;
        for j=1:Stati
            sum1=sum1+P1_d(i,j).*hv(j);
            sum2=sum2+P2_d(i,j).*hv(j);
        end
        Th(i,1)=Cfunc_d(i,1)+sum1;
        Th(i,2)=Cfunc_d(i,2)+sum2;

        %minimization over the controls
        [val,id]=min(Th(i,:));
        Thmin(i)=val;
     end
      hv1=Thmin-(Thmin(1)*e');
     if abs(max(hv1-hv))<toll
        break
     else
        hv=hv1;
     end
   end %end step of value iteration

 h=hv1;

%Policy ymprovement

  for i=1:Stati
        sum1=0;
        sum2=0;
        for j=1:Stati
            sum1=sum1+P1_d(i,j).*h(j);
            sum2=sum2+P2_d(i,j).*h(j);
        end
        THimp(i,1)=Cfunc_d(i,1)+sum1;
        THimp(i,2)=Cfunc_d(i,2)+sum2;

        %minimization over the controls
        [VAL,ID]=min(THimp(i,:));
        psi1(i)=ID;
  end
        if psi1==psi %the new policy is equal to the previous
            one
        break
```

```
        else
        psi=psi1;
        end
        if k==maxiter
           break
        else
           k=k+1 ;
        end
end

%% RESULTS %%

GRID=zeros(N+1,N+1);
for i=1:N+1
    GRID(:,i)=psi1(i:N+1:end);
end

figure(1)
plot(5,0,'bo');
hold on;
plot(0,5,'ro');
legend('Class 1','Class 2');
grid on;
axis([0 N 0 N]);
for i=1:N+1
    for j=1:N+1
        if GRID(i,j)==1;
            plot(j-1,i-1,'bo');
        elseif GRID(i,j)==2;
            plot(j-1,i-1,'ro');
        else isnan(GRID(i,j))
            plot(j-1,i-1,'yo'),
        end
    end
end
xlabel('Q_1','fontweight','bold');
ylabel('Q_2','fontweight','bold');
title({'Optimal Policy',''})
```

Fixed time increment simulation

```
%% FIXED TIME INCREMENT SIMULATION %%

% Simulates arrivals according to Poisson process and
   exponential service
% times . Simultanously runs three paralles simulations , for
    a system
% following the cmu policy and a system following the
   reversed cmu policy and another reversed .
% The systems use the same exponential arrivals and service
   times .
% pol1 = cmu policy
% pol2 = reversed cmu policy
% pol3 = reversed reversed

clear all ;
clc ;
close all

disp ('Running the simulation .');
fprintf ('\n')

%% PARAMETERS

% Simulation
simrep = 5; % Times to repeat simulation
samp = 0.01; % Simulation time sample length [ hour ]
simlen = 1000; % Simulation length [ hours ]
time = 0: samp : simlen ; % Time vector
tlen = simlen / samp +1; % Time vector length

% System parameters
% Initial queue lengths
Q_0(1) = 0; % Product 1
Q_0(2) = 0; % Product 2
Q_0(3) = 0; % Product 3
% Dependent arrival rates [ lots / hour ]
Lambda = [3.454 2.01 0.269;
1.715 1.65 0.065;
0.11 0.08 0.033];

% Arrival rates when server is off
Lambda0(1) = max(Lambda(1,:));
Lambda0(2) = max(Lambda(2,:));
Lambda0(3) = max(Lambda(3,:));
```

```
% Processing rates [ lots / hour ]
Mu(1) = 7;
Mu(2) = 5;
Mu(3) = 2;

% Holding costs [ dollars / lot / hour ]
C(1) = 1;
C(2) = 50;
C(3) = 1000;

% Stability check
m = inv ( diag(Mu));
M = Lambda *m;
EIG = abs(eig(M));
if max(EIG) >= 1
msg = ['This choice of parameters does not guarantee
   stability ,'...
' please choose different parameters .'];
error( msg );
end

% Checking theoretical results
A_1 = C(1)*(Mu(1)-(Lambda(1,1)-Lambda(1,2)))-C(3)*Lambda(3,1)
   ;
A_2 = C(2)*(Mu(2)-(Lambda(2,2)-Lambda(2,1)))-C(3)*Lambda(3,2)
   ;
A_3 = C(2)*(Mu(2)-(Lambda(2,2)-Lambda(2,3)))-C(1)*Lambda(1,2)
   ;
A_4 = C(3)*(Mu(3)-(Lambda(3,3)-Lambda(3,2)))-C(1)*Lambda(1,3)
   ;
A_5 = C(1)*(Mu(1)-(Lambda(1,1)-Lambda(1,3)))-C(2)*Lambda(2,1)
   ;
A_6 = C(3)*(Mu(3)-(Lambda(3,3)-Lambda(3,1)))-C(2)*Lambda(2,3)
   ;
% in case 1 wins
if A_1 >= A_2 && A_5 >= A_6
    cmu = 1;
    if (C(2)*(Mu(2)-(Lambda(2,2)-Lambda(2,3)))) > (C(3)*(Mu
       (3)-(Lambda(3,3)-Lambda(3,2))))
        rev = 2;
        revve = 3;
        To = [1;2;3];
        disp(' in accordance with cmu rule the order of
           priority will be 1 2 3')
    else
        rev = 3;
        revve = 2;
```

```
        To = [1;3;2];
        disp(' in accordance with cmu rule the order of
           priority will be 1 3 2')
    end
    % 2 wins
elseif A_2 > A_1 && A_3 > A_4
    cmu = 2;
    if (C(1)*(Mu(1)-(Lambda(1,1)-Lambda(1,3)))) > (C(3)*(Mu
       (3)-(Lambda(3,3)-Lambda(3,1))))
        rev = 1;
        revve = 3;
        To = [2;1;3];
        disp(' in accordance with cmu rule the order of
           priority will be 2 1 3')
    else
        rev = 3;
        revve = 1;
        To = [2;3;1];
        disp(' in accordance with cmu rule the order of
           priority will be 2 3 1')
    end
    % 3 wins
elseif A_4 > A_3 && A_6 > A_5
    cmu = 3;
if (C(2)*(Mu(2)-(Lambda(2,2)-Lambda(2,1)))) > (C(1)*(Mu(1)-(
   Lambda(1,1)-Lambda(1,2))))
   rev = 2;
   revve = 1;
   To = [3;2;1];
   disp(' in accordance with cmu rule the order of priority
      will be 3 2 1')
else
    rev = 1;
    revve = 2;
    To = [3;1;2];
    disp(' in accordance with cmu rule the order of priority
       will be 3 1 2')
end
end

%% SIMULATION

L = Lambda .* samp ; % Scale lambda to simulation parameters
L0 = Lambda0 * samp ;
Mu_inv = 1./ Mu; % Inverse of Mu

arr = zeros(3, tlen ); % Matrix to store arrivals
```

```
Q_pol1 = zeros(3, tlen ); % Matrix representing queue length
   for policy 1 (new cmu rule)
Q_pol2 = zeros(3, tlen ); % Matrix representing queue length
   for policy 2
Q_pol3 = zeros(3,tlen);   % Matrix representing queue length
   for policy 3
Q_pol4 = zeros(3,tlen);   % Matrix representing queue length
   for policy 4
Q_pol5 = zeros(3,tlen);   % Matrix representing queue length
   for policy 5
Q_pol6 = zeros(3,tlen);   % Matrix representing queue length
   for policy 6
Q_pol1(: ,1) = Q_0(:) ; % Initialize intial queue lengths
Q_pol2(: ,1) = Q_0(:) ; % Initialize intial queue lengths
Q_pol3(:,1) = Q_0(:);   % Initialize intial queue lengths
Q_pol4(:,1) = Q_0(:);   % Initialize intial queue lengths
Q_pol5(:,1) = Q_0(:);   % Initialize intial queue lengths
Q_pol6(:,1) = Q_0(:);   % Initialize intial queue lengths

Q_pol1_arr = zeros(3, tlen );
Q_pol2_arr = zeros(3, tlen );
Q_pol3_arr = zeros(3,tlen);
Q_pol4_arr = zeros(3,tlen);
Q_pol5_arr = zeros(3,tlen);
Q_pol6_arr = zeros(3,tlen);
Q_pol1_proc_times = [];
Q_pol2_proc_times = [];
Q_pol3_proc_times = [];
Q_pol4_proc_times = [];
Q_pol5_proc_times = [];
Q_pol6_proc_times = [];

% Product server statusses
server_pol1 = 4; % 1 = product #1, 2 = product #2, 3 =
   product #3 , 4 = idle
server_pol2 = 4; % 1 = product #1, 2 = product #2, 3 =
   product #3 , 4 = idle
server_pol3 = 4; % 1 = product #1, 2 = product #2, 3 =
   product #3 , 4 = idle
server_pol4 = 4; % 1 = product #1, 2 = product #2, 3 =
   product #3 , 4 = idle
server_pol5 = 4; % 1 = product #1, 2 = product #2, 3 =
   product #3 , 4 = idle
server_pol6 = 4; % 1 = product #1, 2 = product #2, 3 =
   product #3 , 4 = idle
```

```
TC_pol1 = zeros(1,simrep);  % Creation Total cost vector for
   policy 1
TC_pol2 = zeros(1,simrep);  % Creation Total cost vector for
   policy 2
TC_pol3 = zeros(1,simrep);  % Creation Total cost vector for
   policy 3
TC_pol4 = zeros(1,simrep);  % Creation Total cost vector for
   policy 4
TC_pol5 = zeros(1,simrep);  % Creation Total cost vector for
   policy 5
TC_pol6 = zeros(1,simrep);  % Creation Total cost vector for
   policy 6
AC_pol1 = zeros(1,simrep);  % Creation Average cost vector
   for policy 1
AC_pol2 = zeros(1,simrep);  % Creation Average cost vector
   for policy 2
AC_pol3 = zeros(1,simrep);  % Creation Average cost vector
   for policy 3
AC_pol4 = zeros(1,simrep);  % Creation Average cost vector
   for policy 4
AC_pol5 = zeros(1,simrep);  % Creation Average cost vector
   for policy 5
AC_pol6 = zeros(1,simrep);  % Creation Average cost vector
   for policy 6

cmu = To(1); % Priority product
rev = To(2); % Non Priority products
revve = To(3); % Non Priority products
disp ('Simulating ... ')
fprintf ('\n')
for r = 1: simrep

fprintf ('Simulation repetition number %d ... \n',r)

c = 2; % Counter
serv_t_pol1 = 0; % Current production time remaining pol1
   simulation
serv_t_pol2 = 0; % Current production time remaining pol2
   simulation
serv_t_pol3 = 0; % Current production time remaining pol3
   simulation
serv_t_pol4 = 0; % Current production time remaining pol4
   simulation
serv_t_pol5 = 0; % Current production time remaining pol5
   simulation
serv_t_pol6 = 0; % Current production time remaining pol6
```

```
   simulation

% Start simulation
for t = 0: samp : simlen - samp

if serv_t_pol1 <= 0 % No product being serviced in pol1 sim
server_pol1 = 4;
serv_t_pol1 = 0;
end

if serv_t_pol2 <= 0 % No product being serviced in pol2 sim
server_pol2 = 4;
serv_t_pol2 = 0;
end

if serv_t_pol3 <= 0 % No product being serviced in pol3 sim
    server_pol3 = 4;
    serv_t_pol3 = 0;
end

if serv_t_pol4 <= 0 % No product being serviced in pol4 sim
    server_pol4 = 4;
    serv_t_pol4 = 0;
end

if serv_t_pol5 <= 0 % No product being serviced in pol5 sim
    server_pol5 = 4;
    serv_t_pol5 = 0;
end

if serv_t_pol6 <= 0 % No product being serviced in pol6 sim
    server_pol6 = 4;
    serv_t_pol6 = 0;
end


% Simulate queue lengths pol1 simulation queue
if server_pol1 == 1
Q_pol1(1,c) = Q_pol1(1,c -1) + poissrnd(L(1 ,1));
Q_pol1(2,c) = Q_pol1(2,c -1) + poissrnd(L(2 ,1));
Q_pol1(3,c) = Q_pol1(3,c-1)  + poissrnd(L(3,1));
elseif server_pol1 == 2
Q_pol1(1,c) = Q_pol1(1,c -1) + poissrnd(L(1 ,2));
Q_pol1(2,c) = Q_pol1(2,c -1) + poissrnd(L(2 ,2));
Q_pol1(3,c) = Q_pol1(3,c-1) + poissrnd(L(3,2));
elseif server_pol1 == 3
Q_pol1(1,c) = Q_pol1(1,c -1) + poissrnd(L(1 ,3));
```

```
Q_pol1(2,c) = Q_pol1(2,c -1) + poissrnd(L(2 ,3));
Q_pol1(3,c) = Q_pol1(3,c-1) + poissrnd(L(3,3));
else % server_cmu == 0
Q_pol1(1,c) = Q_pol1(1,c -1) + poissrnd(L0(1));
Q_pol1(2,c) = Q_pol1(2,c -1) + poissrnd(L0(2));
Q_pol1(3,c) = Q_pol1(3,c -1) + poissrnd(L0(3));
end
Q_pol1_arr(:,c) = Q_pol1(:,c) - Q_pol1(:,c -1) ;

% Simulate queue lengths pol2 simulation queue
if server_pol2 == 1
Q_pol2(1,c) = Q_pol2(1,c -1) + poissrnd(L(1 ,1));
Q_pol2(2,c) = Q_pol2(2,c -1) + poissrnd(L(2 ,1));
Q_pol2(3,c) = Q_pol2(3,c -1) + poissrnd(L(3 ,1));
elseif server_pol2 == 2
Q_pol2(1,c) = Q_pol2(1,c -1) + poissrnd(L(1 ,2));
Q_pol2(2,c) = Q_pol2(2,c -1) + poissrnd(L(2 ,2));
Q_pol2(3,c) = Q_pol2(3,c -1) + poissrnd(L(3 ,2));
elseif server_pol2 == 3
Q_pol2(1,c) = Q_pol2(1,c -1) + poissrnd(L(1 ,3));
Q_pol2(2,c) = Q_pol2(2,c -1) + poissrnd(L(2 ,3));
Q_pol2(3,c) = Q_pol2(3,c -1) + poissrnd(L(3 ,3));
else % server_rev == 0
Q_pol2(1,c) = Q_pol2(1,c -1) + poissrnd(L0(1));
Q_pol2(2,c) = Q_pol2(2,c -1) + poissrnd(L0(2));
Q_pol2(3,c) = Q_pol2(3,c -1) + poissrnd(L0(3));
end
Q_pol2_arr(:,c) = Q_pol2(:,c) - Q_pol2(:,c -1) ;

% Simulate queue lengths pol3 simulation queue
if server_pol3 == 1
Q_pol3(1,c) = Q_pol3(1,c -1) + poissrnd(L(1 ,1));
Q_pol3(2,c) = Q_pol3(2,c -1) + poissrnd(L(2 ,1));
Q_pol3(3,c) = Q_pol3(3,c -1) + poissrnd(L(3 ,1));
elseif server_pol2 == 2
Q_pol3(1,c) = Q_pol3(1,c -1) + poissrnd(L(1 ,2));
Q_pol3(2,c) = Q_pol3(2,c -1) + poissrnd(L(2 ,2));
Q_pol3(3,c) = Q_pol3(3,c -1) + poissrnd(L(3 ,2));
elseif server_pol2 == 3
Q_pol3(1,c) = Q_pol3(1,c -1) + poissrnd(L(1 ,3));
Q_pol3(2,c) = Q_pol3(2,c -1) + poissrnd(L(2 ,3));
Q_pol3(3,c) = Q_pol3(3,c -1) + poissrnd(L(3 ,3));
else % server_rev == 0
Q_pol3(1,c) = Q_pol3(1,c -1) + poissrnd(L0(1));
Q_pol3(2,c) = Q_pol3(2,c -1) + poissrnd(L0(2));
Q_pol3(3,c) = Q_pol3(3,c -1) + poissrnd(L0(3));
end
```

```
Q_pol3_arr(:,c) = Q_pol3(:,c) - Q_pol3(:,c -1) ;

% Simulate queue lengths pol4 simulation queue
if server_pol4 == 1
Q_pol4(1,c) = Q_pol4(1,c -1) + poissrnd(L(1 ,1));
Q_pol4(2,c) = Q_pol4(2,c -1) + poissrnd(L(2 ,1));
Q_pol4(3,c) = Q_pol4(3,c -1) + poissrnd(L(3 ,1));
elseif server_pol4 == 2
Q_pol4(1,c) = Q_pol4(1,c -1) + poissrnd(L(1 ,2));
Q_pol4(2,c) = Q_pol4(2,c -1) + poissrnd(L(2 ,2));
Q_pol4(3,c) = Q_pol4(3,c -1) + poissrnd(L(3 ,2));
elseif server_pol4 == 3
Q_pol4(1,c) = Q_pol4(1,c -1) + poissrnd(L(1 ,3));
Q_pol4(2,c) = Q_pol4(2,c -1) + poissrnd(L(2 ,3));
Q_pol4(3,c) = Q_pol4(3,c -1) + poissrnd(L(3 ,3));
else % server_rev == 0
Q_pol4(1,c) = Q_pol4(1,c -1) + poissrnd(L0(1));
Q_pol4(2,c) = Q_pol4(2,c -1) + poissrnd(L0(2));
Q_pol4(3,c) = Q_pol4(3,c -1) + poissrnd(L0(3));
end
Q_pol4_arr(:,c) = Q_pol4(:,c) - Q_pol4(:,c -1) ;



% Simulate queue lengths pol5 simulation queue
if server_pol5 == 1
Q_pol5(1,c) = Q_pol5(1,c -1) + poissrnd(L(1 ,1));
Q_pol5(2,c) = Q_pol5(2,c -1) + poissrnd(L(2 ,1));
Q_pol5(3,c) = Q_pol5(3,c -1) + poissrnd(L(3 ,1));
elseif server_pol5 == 2
Q_pol5(1,c) = Q_pol5(1,c -1) + poissrnd(L(1 ,2));
Q_pol5(2,c) = Q_pol5(2,c -1) + poissrnd(L(2 ,2));
Q_pol5(3,c) = Q_pol5(3,c -1) + poissrnd(L(3 ,2));
elseif server_pol2 == 3
Q_pol5(1,c) = Q_pol5(1,c -1) + poissrnd(L(1 ,3));
Q_pol5(2,c) = Q_pol5(2,c -1) + poissrnd(L(2 ,3));
Q_pol5(3,c) = Q_pol5(3,c -1) + poissrnd(L(3 ,3));
else % server_rev == 0
Q_pol5(1,c) = Q_pol5(1,c -1) + poissrnd(L0(1));
Q_pol5(2,c) = Q_pol5(2,c -1) + poissrnd(L0(2));
Q_pol5(3,c) = Q_pol5(3,c -1) + poissrnd(L0(3));
end
Q_pol5_arr(:,c) = Q_pol5(:,c) - Q_pol5(:,c -1) ;


% Simulate queue lengths pol6 simulation queue
if server_pol6 == 1
```

```
Q_pol6(1,c) = Q_pol6(1,c -1) + poissrnd(L(1 ,1));
Q_pol6(2,c) = Q_pol6(2,c -1) + poissrnd(L(2 ,1));
Q_pol6(3,c) = Q_pol6(3,c -1) + poissrnd(L(3 ,1));
elseif server_pol6 == 2
Q_pol6(1,c) = Q_pol6(1,c -1) + poissrnd(L(1 ,2));
Q_pol6(2,c) = Q_pol6(2,c -1) + poissrnd(L(2 ,2));
Q_pol6(3,c) = Q_pol6(3,c -1) + poissrnd(L(3 ,2));
elseif server_pol6 == 3
Q_pol6(1,c) = Q_pol6(1,c -1) + poissrnd(L(1 ,3));
Q_pol6(2,c) = Q_pol6(2,c -1) + poissrnd(L(2 ,3));
Q_pol6(3,c) = Q_pol6(3,c -1) + poissrnd(L(3 ,3));
else % server_rev == 0
Q_pol6(1,c) = Q_pol6(1,c -1) + poissrnd(L0(1));
Q_pol6(2,c) = Q_pol6(2,c -1) + poissrnd(L0(2));
Q_pol6(3,c) = Q_pol6(3,c -1) + poissrnd(L0(3));
end
Q_pol6_arr(:,c) = Q_pol6(:,c) - Q_pol6(:,c -1) ;
```

```
% Production process for pol1 simulation
if server_pol1 == 4 % Server is idle
num1 = size( Q_pol1_proc_times ,2) ; % N.o. products so far
if Q_pol1(cmu ,c) >= 1 % If queue of cmu product is non -
   empty
% Simulate processing time cmu prod and adapt server time
serv_t_pol1 = exprnd( Mu_inv (cmu ));
Q_pol1_proc_times(cmu , num1 +1) = serv_t_pol1 ;
% Decrease qeueu length for cmu product
Q_pol1(cmu ,c) = Q_pol1(cmu ,c) -1;
% Set server status to cmu producttype in service
server_pol1 = cmu ;
elseif Q_pol1(rev ,c) >= 1 % Otherwise , non - preffered
   product
% Simulate processing time rev prod and adapt server time
serv_t_pol1 = exprnd( Mu_inv(rev ));
Q_pol1_proc_times(rev , num1 +1) = serv_t_pol1 ;
% Decrease qeueu length for rev product
Q_pol1(rev ,c) = Q_pol1(rev ,c) -1;
% Set server status to rev producttype in service
server_pol1 = rev ;
elseif Q_pol1(revve,c) >= 1
    serv_t_pol1 = exprnd(Mu_inv(revve));
    Q_pol1_proc_times(revve,num1+1) = serv_t_pol1;
    Q_pol1(revve,c) = Q_pol1(revve,c) -1;
    server_pol1 = revve;
```

```
    end
else % Server is busy
serv_t_pol1 = serv_t_pol1 - samp ;
end



% Production process for pol2 simulation
if server_pol2 == 4 % Server is idle
num2 = size( Q_pol2_proc_times ,2) ;
if Q_pol2(rev ,c) >= 1 % If queue of rev product is non -
   empty
% Simulate processing time cmu prod and adapt server time
serv_t_pol2 = exprnd( Mu_inv (rev ));
Q_pol2_proc_times(rev , num2 +1) = serv_t_pol2 ;
% Decrease qeueu length for rev product
Q_pol2(rev ,c) = Q_pol2(rev ,c) -1;
% Set server status to rev producttype in service
server_pol2 = rev ;
elseif Q_pol2(cmu ,c) >= 1 % Otherwise , non - preffered
   product
% Simulate processing time cmu prod and adapt server time
serv_t_pol2 = exprnd( Mu_inv (cmu ));
Q_pol2_proc_times(cmu , num2 +1) = serv_t_pol2 ;
% Decrease qeueu length for cmu product
Q_pol2(cmu ,c) = Q_pol2(cmu ,c) -1;
% Set server status to cmu producttype in service
server_pol2 = cmu ;
elseif Q_pol2(revve ,c) >= 1 % Otherwise , non - preffered
   product
% Simulate processing time cmu prod and adapt server time
serv_t_pol2 = exprnd( Mu_inv (revve ));
Q_pol2_proc_times(revve , num2 +1) = serv_t_pol2 ;
% Decrease qeueu length for cmu product
Q_pol2(revve ,c) = Q_pol2(revve ,c) -1;
% Set server status to cmu producttype in service
server_pol2 = revve ;

end
else % Server is busy
serv_t_pol2 = serv_t_pol2 - samp ;
end


% Production process for pol3 simulation
if server_pol3 == 4 % Server is idle
num3 = size( Q_pol3_proc_times ,2) ;
```

```
if Q_pol3(revve ,c) >= 1 % If queue of rev product is non -
   empty
% Simulate processing time cmu prod and adapt server time
serv_t_pol3 = exprnd( Mu_inv (revve ));
Q_pol3_proc_times(revve , num3 +1) = serv_t_pol3 ;
% Decrease qeueu length for rev product
Q_pol3(revve ,c) = Q_pol3(revve ,c) -1;
% Set server status to rev producttype in service
server_pol3 = revve ;
elseif Q_pol3(cmu ,c) >= 1 % Otherwise , non - preffered
   product
% Simulate processing time cmu prod and adapt server time
serv_t_pol3 = exprnd( Mu_inv (cmu ));
Q_pol3_proc_times(cmu , num3 +1) = serv_t_pol3 ;
% Decrease qeueu length for cmu product
Q_pol3(cmu ,c) = Q_pol3(cmu ,c) -1;
% Set server status to cmu producttype in service
server_pol3 = cmu ;
elseif Q_pol3(rev ,c) >= 1 % Otherwise , non - preffered
   product
% Simulate processing time cmu prod and adapt server time
serv_t_pol3 = exprnd( Mu_inv (rev ));
Q_pol3_proc_times(rev , num3 +1) = serv_t_pol3 ;
% Decrease qeueu length for cmu product
Q_pol3(rev ,c) = Q_pol3(rev ,c) -1;
% Set server status to cmu producttype in service
server_pol3 = rev ;

end
else % Server is busy
serv_t_pol3 = serv_t_pol3 - samp ;
end


% Production process for pol4 simulation
if server_pol4 == 4 % Server is idle
num4 = size( Q_pol4_proc_times ,2) ;
if Q_pol4(revve ,c) >= 1 % If queue of rev product is non -
   empty
% Simulate processing time cmu prod and adapt server time
serv_t_pol4 = exprnd( Mu_inv (revve ));
Q_pol4_proc_times(revve , num4 +1) = serv_t_pol4 ;
% Decrease qeueu length for rev product
Q_pol4(revve ,c) = Q_pol4(revve ,c) -1;
% Set server status to rev producttype in service
server_pol4 = revve ;
elseif Q_pol4(rev ,c) >= 1 % Otherwise , non - preffered
```

```
   product
% Simulate processing time cmu prod and adapt server time
serv_t_pol4 = exprnd( Mu_inv (rev ));
Q_pol4_proc_times(rev , num4 +1) = serv_t_pol4 ;
% Decrease qeueu length for cmu product
Q_pol4(rev ,c) = Q_pol4(rev ,c) -1;
% Set server status to cmu producttype in service
server_pol4 = rev ;
elseif Q_pol4(cmu ,c) >= 1 % Otherwise , non - preffered
   product
% Simulate processing time cmu prod and adapt server time
serv_t_pol4 = exprnd( Mu_inv (cmu ));
Q_pol4_proc_times(cmu , num4 +1) = serv_t_pol4 ;
% Decrease qeueu length for cmu product
Q_pol4(cmu ,c) = Q_pol4(cmu ,c) -1;
% Set server status to cmu producttype in service
server_pol4 = cmu ;

end
else % Server is busy
serv_t_pol4 = serv_t_pol4 - samp ;
end

% Production process for pol5 simulation
if server_pol5 == 4 % Server is idle
num5 = size( Q_pol5_proc_times ,2) ;
if Q_pol5(rev ,c) >= 1 % If queue of rev product is non -
   empty
% Simulate processing time cmu prod and adapt server time
serv_t_pol5 = exprnd( Mu_inv (rev ));
Q_pol5_proc_times(rev , num5 +1) = serv_t_pol5 ;
% Decrease qeueu length for rev product
Q_pol5(rev ,c) = Q_pol5(rev ,c) -1;
% Set server status to rev producttype in service
server_pol5 = rev ;
elseif Q_pol5(revve ,c) >= 1 % Otherwise , non - preffered
   product
% Simulate processing time cmu prod and adapt server time
serv_t_pol5 = exprnd( Mu_inv (revve ));
Q_pol5_proc_times(revve , num5 +1) = serv_t_pol5 ;
% Decrease qeueu length for cmu product
Q_pol5(revve ,c) = Q_pol5(revve ,c) -1;
% Set server status to cmu producttype in service
server_pol5 = revve ;
elseif Q_pol5(cmu ,c) >= 1 % Otherwise , non - preffered
   product
% Simulate processing time cmu prod and adapt server time
```

```
serv_t_pol5 = exprnd( Mu_inv (cmu ));
Q_pol5_proc_times(cmu , num5 +1) = serv_t_pol5 ;
% Decrease qeueu length for cmu product
Q_pol5(cmu ,c) = Q_pol5(cmu ,c) -1;
% Set server status to cmu producttype in service
server_pol5 = cmu ;

end
else % Server is busy
serv_t_pol5 = serv_t_pol5 - samp ;
end


% Production process for pol6 simulation
if server_pol6 == 4 % Server is idle
num6 = size( Q_pol6_proc_times ,2) ;
if Q_pol6(cmu ,c) >= 1 % If queue of rev product is non -
   empty
% Simulate processing time cmu prod and adapt server time
serv_t_pol6 = exprnd( Mu_inv (cmu ));
Q_pol6_proc_times(cmu , num6 +1) = serv_t_pol6 ;
% Decrease qeueu length for rev product
Q_pol6(cmu ,c) = Q_pol6(cmu ,c) -1;
% Set server status to rev producttype in service
server_pol6 = cmu ;
elseif Q_pol6(revve ,c) >= 1 % Otherwise , non - preffered
   product
% Simulate processing time cmu prod and adapt server time
serv_t_pol6 = exprnd( Mu_inv (revve ));
Q_pol6_proc_times(revve , num6 +1) = serv_t_pol6 ;
% Decrease qeueu length for cmu product
Q_pol6(revve ,c) = Q_pol6(revve ,c) -1;
% Set server status to cmu producttype in service
server_pol6 = revve ;
elseif Q_pol6(rev ,c) >= 1 % Otherwise , non - preffered
   product
% Simulate processing time cmu prod and adapt server time
serv_t_pol6 = exprnd( Mu_inv (rev ));
Q_pol6_proc_times(rev , num6 +1) = serv_t_pol6 ;
% Decrease qeueu length for cmu product
Q_pol6(rev ,c) = Q_pol6(rev ,c) -1;
% Set server status to cmu producttype in service
server_pol6 = rev ;

end
else % Server is busy
serv_t_pol6 = serv_t_pol6 - samp ;
```

```
end


c = c +1;
end

% Compensate for potential warmup effects
wa = 0.1* simlen / samp ; % Warmup part
Q_pol1(: ,1: wa) = []; % Remove data
Q_pol2(: ,1: wa) = []; % Remove data
Q_pol3(: ,1: wa) = []; % Remove data
Q_pol4(: ,1: wa) = []; % Remove data
Q_pol5(: ,1: wa) = []; % Remove data
Q_pol6(: ,1: wa) = []; % Remove data
% Compute total costs
TC_pol1(r) = sum( Q_pol1(1 ,:))*(C(1) * samp )+sum( Q_pol1(2
   ,:))*(C(2) * samp) + sum( Q_pol1(3 ,:))*(C(3)*samp);
TC_pol2(r) = sum( Q_pol2(1 ,:))*(C(1) * samp )+sum( Q_pol2(2
   ,:))*(C(2) * samp) + sum( Q_pol2(3 ,:))*(C(3)*samp);
TC_pol3(r) = sum( Q_pol3(1 ,:))*(C(1) * samp )+sum( Q_pol3(2
   ,:))*(C(2) * samp) + sum( Q_pol3(3 ,:))*(C(3)*samp);
TC_pol4(r) = sum( Q_pol4(1 ,:))*(C(1) * samp )+sum( Q_pol4(2
   ,:))*(C(2) * samp) + sum( Q_pol4(3 ,:))*(C(3)*samp);
TC_pol5(r) = sum( Q_pol5(1 ,:))*(C(1) * samp )+sum( Q_pol5(2
   ,:))*(C(2) * samp) + sum( Q_pol5(3 ,:))*(C(3)*samp);
TC_pol6(r) = sum( Q_pol6(1 ,:))*(C(1) * samp )+sum( Q_pol6(2
   ,:))*(C(2) * samp) + sum( Q_pol6(3 ,:))*(C(3)*samp);
% Compute average costs ( with warmup compsensation )
AC_pol1(r) = TC_pol1(r) / ( simlen *0.9) ;
AC_pol2(r) = TC_pol2(r) / ( simlen *0.9) ;
AC_pol3(r) = TC_pol3(r) / ( simlen *0.9) ;
AC_pol4(r) = TC_pol4(r) / ( simlen *0.9) ;
AC_pol5(r) = TC_pol5(r) / ( simlen *0.9) ;
AC_pol6(r) = TC_pol6(r) / ( simlen *0.9) ;


AC_pol1(r);
AC_pol2(r);
AC_pol3(r);
AC_pol4(r);
AC_pol5(r);
AC_pol6(r);



 end
 %% ANALYSING RESULTS
```

```
TC_pol1_tot = mean( TC_pol1 );
TC_pol2_tot = mean( TC_pol2 );
TC_pol3_tot = mean( TC_pol3 );
TC_pol4_tot = mean( TC_pol4 );
TC_pol5_tot = mean( TC_pol5 );
TC_pol6_tot = mean( TC_pol6 );

AC_pol1_tot = mean( AC_pol1 )
AC_pol2_tot = mean( AC_pol2 )
AC_pol3_tot = mean( AC_pol3 )
AC_pol4_tot = mean( AC_pol4 )
AC_pol5_tot = mean( AC_pol5 )
AC_pol6_tot = mean( AC_pol6 )

 disp ('Simulation finished .')
```