University of Auckland

Doctoral Thesis

User Equilibria in Systems of Parallel Processor Sharing Queues

Niffe Hermansson

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Applied Probability Group
Department of Statistics

November 28, 2018

Declaration of Authorship

I, Niffe Hermansson, declare that this thesis titled, "User Equilibria in Systems of Parallel Processor Sharing Queues" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Abstract

User Equilibria in Systems of Parallel Processor Sharing Queues

by Niffe Hermansson

This thesis considers a system of processor sharing queues under selfish routing, where the users choice of queue depends on their expected waiting times in each queue – in the simplest case they will seek just to minimize their expected waiting time. We assume that users have full knowledge of the state of the system when choosing their route. We introduce several update and routing rules, and seek to characterize the state dependent user equilibria induced by them. We discuss properties of both the system itself and the user equilibrium policies.

We employ stochastic comparison and coupling arguments to show various monotonicity properties possessed by this system. Some of these theorems, such as that describing the monotonic relationship between service rate in a queue and the probability that a user will join that queue, hold only for systems of two queues. Other theorems, such as that describing the monotonic relationship between the service rate in a queue and the expected waiting time for a user joining that queue, hold in the context of a system of arbitrarily many queues in parallel.

We also present extensive numerical work exploring the structure of fixed points of policy iteration in the system with two queues. We confirm previous findings of complex behaviour arising from policy iteration in this system. In particular, we see that policy iteration is not guaranteed to converge to a fixed point, and may instead converge to a periodic orbit. When it does converge, the fixed point policy is sometimes non-monotonic in occupancy. We implement novel routing and update rules, investigating the effect of boundary conditions, and of routing rules which are not necessarily 0-1 rules. We find that none of the boundary conditions we employ eliminates the nonmonotonicity or periodicity. Each routing rule gives rise to non-monotonic policies, but for one of them all our numerical examples converge to a fixed point, with no periodic behaviour. We go on to use a method related to backwards induction to show that with this routing rule, policy iteration in the system is indeed guaranteed to have at least one fixed point.

Acknowledgements

I want to express my sincere and deep gratitude to all those who made this thesis possible, and all those who helped and supported me along the way.

First of all I want to thank my supervisor Ilze Ziedins. I could not have asked for a more knowledgeable, competent, warm and understanding person to guide me on this journey. I would also like to extend my sincerest thanks to Mark Holmes for his invaluable help, in particular in the early stages of this work.

I also want to thank my colleagues in the Department of Statistics (Room 224 and otherwise). I am fairly certain that this work would have been done significantly faster if it wasn't for all of them, but I wouldn't have missed any of it for the world.

This work would not have been possible without a generous grant from the Marsden Fund, for which I am very grateful.

I want to thank my mother, Ann, for the support, input and inspiration she has provided throughout my life in general, and this work in particular. A warm thank you to my siblings as well, for their interest and support. In particular I want to thank my sister, Mia; having someone as awesome as her in my life always pushes me to try to be better and work harder.

I also want to thank my dear friends. Friends on water, friends on rock, friends on snow, even friends on flat, dry, land. Friends in New Zealand, California, Sweden and around the world. They keep me happy, thinking and sane, and I can't thank them enough for that.

Finally, there is no way to overstate the gratitude I feel for and towards my partner, Lisett. She has supported me through this work in every way imaginable; emotionally, intellectually, economically and physically. She has spent more time than is reasonable talking to me about breakthroughs and obstacles. She even read an early version of the thesis. Lisett, I wish to one day come up with a way of repaying you for all that you have done for me, until then I will just try to live up to you.

For Inga Petersson, my Momma, you taught me to ask questions.

Contents

Declaration of Authorship iii

Abstract v

Acknowledgements vii

1 Introduction 1
- 1.1 Background and motivation . . . 3
  - 1.1.1 Queuing networks . . . 4
  - 1.1.2 Service disciplines, processor sharing . . . 7
  - 1.1.3 Routing . . . 8
    - State dependent routing . . . 8
    - Selfish routing . . . 10
  - 1.1.4 Methods . . . 12
    - Stochastic coupling . . . 13
    - Markov Decision Processes . . . 15
- 1.2 Literature review . . . 19
- 1.3 Thesis outline . . . 23

2 System definition 25
- 2.1 System structure . . . 25
- 2.2 Assumptions . . . 28
- 2.3 Decision rule and policy . . . 31
  - 2.3.1 Decision policy definition . . . 31
  - 2.3.2 Expected waiting times . . . 34
  - 2.3.3 User equilibrium policy . . . 35
  - 2.3.4 Decision policy monotonicity . . . 37

3 Monotonicity results **43**
3.1 Expected waiting time monotonicities . . . 44
3.1.1 Monotonicity of expected waiting time in occupancy . . . 44
3.1.2 Monotonicity of expected waiting time in λ . . . 52
3.1.3 Monotonicity of expected waiting time in σ_i . . . 61
3.1.4 Monotonicity of expected waiting time in μ_i . . . 65
3.1.5 k-asymmetry in expected waiting time for $\mathbf{n}+\mathbf{e}_k$ . . . 68
3.2 Stationary joining probabilities . . . 72
3.2.1 Monotonicity of stationary joining probability in μ_i . . . 72
3.2.2 Monotonicity of stationary joining probability in σ_i . . . 80
3.3 Extension to infinite capacity systems . . . 81
3.3.1 Extension to infinite capacity queues . . . 81
3.3.2 k-asymmetry in expected waiting time for $\mathbf{n}+\mathbf{e}_k$, infinite queues . . . 81
3.4 Summary . . . 83

4 Policy iteration **85**
4.1 Definitions . . . 86
4.1.1 Fundamental description . . . 86
4.1.2 Policy iteration algorithm . . . 87
4.1.3 Update rules . . . 88
$\{0,1,\tilde{p}\}$-update rule . . . 89
$(0,1)$-decision rules . . . 91
$[0,1]$-decision rules . . . 91
4.1.4 Lack of policy convergence . . . 93
4.1.5 Boundary conditions . . . 95
Automatically routing boundaries . . . 95
Rejection boundaries . . . 96
$\hat{z}$-routing boundaries . . . 96
4.1.6 Expected waiting time . . . 98
Calculating expected waiting times . . . 98
Hypothetical waiting times . . . 99
4.2 Numerical results . . . 100
4.2.1 Spatial distribution of policy properties - large parameter space region . . . 101
Validation of spatial distribution of policy properties . . . 102

Spatial distribution of policy properties with rejection boundaries . . . 105
Spatial distribution of policy properties with $\hat{z}$-routing boundaries . . . 107
Conclusions from varying boundary conditions . . . 108
Spatial distribution of policy properties with $[0, 1]$-updates . . . 108
Policy property patterns with $(0, 1)$-updates . . . 111
Conclusions from reproduction and extensions . . . 113
4.2.2 Policy property spatial distribution observations in the service rate plane . . . 113
Non-monotonicities and periodicity with $\{0, 1, \tilde{p}\}$-updates . . . 114
Periodicities and effect of δ with $[0, 1]$-updates . . . 117
Non-monotonic structure with $(0, 1)$-updates . . . 122
4.2.3 Stationary joining probabilities . . . 136
$p^*(\mathbf{\Gamma})$ and service rates . . . 137
Stationary joining probabilities by μs, $\{0, 1, \tilde{p}\}$-updates . . . 139
Stationary joining probabilities by μs, $[0, 1]$-updates . . . 142
Stationary joining probabilities by μs, $(0, 1)$-updates . . . 147
Conclusions on stationary joining probabilities and service rates . . . 149
$p^*(\mathbf{\Gamma})$ and intrinsic arrival rates . . . 150
Stationary joining probabilities by σs, $\{0, 1, \tilde{p}\}$-updates . . . 154
Stationary joining probabilities by σs, $[0, 1]$-updates . . . 155
Stationary joining probabilities by σs, $(0, 1)$-updates . . . 158
4.3 Conclusions from numerics . . . 160
4.3.1 Parameters and problematic policies . . . 161
4.3.2 Boundary conditions, update rules, and policy iteration outcomes . . . 162

5 Modified policy iteration **165**
5.1 Motivation . . . 165
5.2 Definitions . . . 166
5.2.1 Modified policy iteration algorithm . . . 167
5.2.2 Policy iteration and modified policy iteration fixed point identity . . . 171
5.2.3 Existence of fixed points . . . 176

5.3 Remarks . . . 180
5.3.1 Applicability of result . . . 180
5.3.2 Relationship to results from Chapter 4 . . . 181
5.4 Attempted extensions . . . 181
5.4.1 Fixed Point Uniqueness by Contraction . . . 182
The Project . . . 182
The obstacle . . . 184
The interpretation . . . 184
5.4.2 Explicit backwards induction . . . 185
The project . . . 185
The formulation . . . 186
The outcome . . . 188
The obstacle . . . 191
The interpretation . . . 191

6 Conclusion **195**
6.1 Selfish optimisation as a method for job allocation . . . 195
6.2 System and process properties . . . 200

7 Epilogue **207**

A MATLAB code **209**
A.1 Policy iteration algorithm . . . 209
A.2 Calculating stationary joining probabilities . . . 235
A.3 Plotting mechanism . . . 238

Bibliography **249**

List of Figures

1.1 Common visual representation of a single queue with a single server. 4
1.2 Visual representation of queues in parallel without explicit interaction. 5
1.3 Visual representation of queues in tandem with general onward routing. 6

3.1 All types of transitions that can occur at positive rate in each state of the part of the l_1l_2-plane such that $l_i \geq 0, \forall i \in \{1, 2\}$. 74

4.1 Parameter space distribution of policy iteration outcomes with $\{0, 1, \tilde{p}\}$-updates and automatically routing boundaries. 103
4.2 Magnified region of Figure 4.1. 105
4.3 Parameter space distribution of policy iteration outcomes with $\{0, 1, \tilde{p}\}$-updates and rejection boundaries. 106
4.4 Parameter space distribution of policy iteration outcomes with $\{0, 1, \tilde{p}\}$-updates and $\hat{z}$-routing boundaries. 107
4.5 Parameter space distribution of policy iteration outcomes with $[0, 1]$-updates and automatically routing boundaries. 109
4.6 Points of difference between Figure 4.5 and Figure 4.1. 110
4.7 Reproduction of Figure 4.5 for three different values of δ. 111
4.8 Parameter space distribution of policy iteration outcomes with $(0, 1)$-updates and automatically routing boundaries. 112
4.9 Parameter space distribution of types of resulting policies with $\{0, 1, \tilde{p}\}$-updates and automatically routing boundaries for two different symmetrical values of capacity. 115
4.10 Parameter space distribution of types of resulting policies with $\{0, 1, \tilde{p}\}$-updates and $\hat{z}$-routing for two different symmetrical values of capacity. 116
4.11 Parameter space distribution of types of resulting policies with $\{0, 1, \tilde{p}\}$-updates and rejection routing for two different symmetrical values of capacity. 117

4.12 Parameter space distribution of types of resulting policies with $[0,1]$-updates and rejection boundaries for two different symmetrical values of capacity. . . . 118
4.13 Decision rule development in increasing δ. . . . 120
4.14 Parameter space distribution of types of resulting policies with $(0,1)$-updates and automatically routing boundaries. . . . 123
4.15 Types of non-monotonic policies in Figure 4.14. . . . 126
4.16 Parameter space distribution of types of non-monotonic policies with $(0,1)$-updates, $N_1 = 5, N_2 = 6$. . . . 127
4.17 Parameter space distribution of types of non-monotonic policies with $(0,1)$-updates, $N_1 = N_2 = 8$. . . . 131
4.18 $p_1^*(\mathbf{\Gamma})$ as a function of μ_1 and μ_2, $\{0,1,\tilde{p}\}$-updates. . . . 140
4.19 $p_1^*(\mathbf{\Gamma})$ as a function of μ_1 and μ_2, $\{0,1,\tilde{p}\}$-updates, asymmetric parameters. . . . 141
4.20 $p_1^*(\mathbf{\Gamma})$ as a function of μ_1 and μ_2, $[0,1]$-updates. . . . 142
4.21 $p_1^*(\mathbf{\Gamma})$ as a function of μ_1 and μ_2, $[0,1]$-updates, asymmetric parameters. . . . 143
4.22 Figure 4.20 for three different values of δ. . . . 144
4.23 Comparison between Figures 4.18 and 4.20. . . . 145
4.24 Comparison between fixed point policies in Figures 4.18 and 4.20. . . . 146
4.25 $p_1^*(\mathbf{\Gamma})$ as a function of μ_1 and μ_2, $(0,1)$-updates. . . . 148
4.26 $p_1^*(\mathbf{\Gamma})$ as a function of μ_2, $(0,1)$-updates. . . . 149
4.27 $p_1^*(\mathbf{\Gamma})$ as a function of μ_2, $\{0,1,\tilde{p}\}$- and $[0,1]$-updates. . . . 150
4.28 $p_1^*(\mathbf{\Gamma})$ as a function of μ_1 and μ_2, $(0,1)$-updates, asymmetric parameters. . . . 151
4.29 $p_1^*(\mathbf{\Gamma})$ as a function of σ_1 and σ_2 for $\{0,1,\tilde{p}\}$-updates. . . . 154
4.30 $p_1^*(\mathbf{\Gamma})$ as a function of σ_1 and σ_2 for $\{0,1,\tilde{p}\}$-updates, asymmetric parameters. . . . 155
4.31 $p_1^*(\mathbf{\Gamma})$ as a function of σ_1 and σ_2 for $[0,1]$-updates. . . . 156
4.32 Figure 4.31 for three different values of δ. . . . 157
4.33 $p_1^*(\mathbf{\Gamma})$ as a function of σ_1 and σ_2 for $[0,1]$-updates, asymmetric parameters. . . . 158
4.34 $p_1^*(\mathbf{\Gamma})$ as a function of σ_1 and σ_2 for $(0,1)$-updates. . . . 159
4.35 $p_1^*(\mathbf{\Gamma})$ as a function of σ_1 and σ_2 for $(0,1)$-updates, asymmetric parameters. . . . 160

5.1 Outcomes of backwards induction over the region depicted in Figure 4.2. 190

List of Notation

Q_i	the ith queue
μ	without subscript in general used to denote some service rate
μ_i	service rate in Q_i
$\mu^e(n)$	effective service rate
σ_i	arrival rate for intrinsic arrivals to Q_i
λ	general arrival rate
$\mathbf{\Gamma}$	paramater vector
$\mathcal{G}$	set of all parameter vector
n	without subscript used to denote a number of customers
n_i	number of customers in Q_i
N_i	capacity of the ith queue
M	number of queues
$\mathcal{M} = \{1, ..., M\}$	index set describing a set of queues
$\mathbf{n} = (n_1, ..., n_M)$	occupancy vector, identical to state
$\mathcal{S}$	state space of the queueing system
$\mathcal{S}^e$	the set of edge states, states such that at least one queue is full
$\mathcal{N}_i^e$	set of states where $n_i = N_i$ and $n_j < N_j$
$\mathcal{S}^-$	the set of internal states, $\mathcal{S}^- = \mathcal{S} \setminus \mathcal{S}^e$
$D_i(\mathbf{n})$	decision rule for state $\mathbf{n}$
$D_0(\mathbf{n})$	balking probability for state $\mathbf{n}$
$D(\mathbf{n})$	the set of decision ruled for state $\mathbf{n}$
$\mathbf{D}$	decision policy
$\mathbf{D}^*$	fixed point or user equilibrium decision policy
$\mathbf{D}^t_{\mathbf{\Gamma}}$	decision policy after t steps of policy iteration with parameter vector $\mathbf{\Gamma}$
$\mathcal{D}$	the set of all decision policies.
$\mathbf{r}^{n_j}_{i,j}$	decision rules for Q_i with the occupancy of Q_j fixed to n_j
$z_i(\mathbf{n})$	expected waiting time for Q_i when the system is in state $\mathbf{n}$
$\hat{z}_i(\mathbf{n})$	hypothetical waiting time
$\mathbf{Z}$	expected waiting times for each queue and each state of some system

$\zeta_i(\mathbf{n})$	pseudo waiting time for Q_i when the system is in state $\mathbf{n}$
$\hat{\zeta}_j(\mathbf{n})$	hypothetical pseudo waiting time
$\boldsymbol{\zeta}$	pseudo waiting times for each queue and each state of some system
$\boldsymbol{\zeta}^t$	pseudo waiting times after t steps of modified policy iteration
$\boldsymbol{\zeta}^*$	fixed point pseudo waiting times
$\mathbf{f}$	routing/update rule
$R_j(\mathbf{n}, \boldsymbol{\zeta}(\mathbf{n}))$	update rule analogue in Modified Policy Iteration
$c_{i,T}(\mathbf{n})$	terminal cost for state $\mathbf{n}$ (Chapter 5)
t	iterative step or time as measured on a clock depending on context
$X_i(t)$	Q_i occupancy process
$\mathbf{X}(t)$	system occupancy process
C_k	decision of the kth customer
$c_{k,i}$	the kth general arrival who chooses to join Q_i (Chapter 2)
τ	time of specific event, modified with sub and super- scripts
$\mathbf{J}$	square matrix where every entry is 1
S_i	Server i (Section 2.1)
P_i	Pump i (Section 2.1)
$\{a, b, c\}$	Set of available files/liquids (Section 2.1)
μ_j	for $j \in \{a, b, c\}$ flow rate of liquid j (Section 2.1)
$\mathbf{e}_i$	unit vector; all entries are 0 except the ith entry which is 1
$\pi_{\boldsymbol{\Gamma},\mathbf{D}}(\mathbf{n})$	probability, at stationarity, of finding system in state $\mathbf{n}$
$p_i(\boldsymbol{\Gamma}, \mathbf{D})$	stationary joining probability
$p^*(\boldsymbol{\Gamma})$	fixed point stationary joining probability, $p_i(\boldsymbol{\Gamma}, \mathbf{D}^*_{\boldsymbol{\Gamma},\mathbf{D}^0}) = p^*_{i,\mathbf{D}^0}(\boldsymbol{\Gamma})$
$\tilde{p}$	tie breaker probability in $\{0, 1, \tilde{p}\}$-updates
δ	half the range of interpolation in $[0, 1]$-updates

List of Abbreviations

PS	**P**rocessor **S**haring
EPS	**E**galitarian **P**rocessor **S**haring
RS	**R**andom **S**ervice
FIFO/FCFS	**F**irst **I**n **F**irst **O**ut/**F**irst **C**ome **F**irst **S**erved
LIFO/LCFS	**L**ast **I**n **F**irst **O**ut/**L**ast **C**ome **F**irst **S**erved
MDP	**M**arkov **D**ecision **P**rocess
JSQ	**J**oin the **S**hortest **Q**ueue
IID	**I**ndependent and **I**dentically **D**istributed
a.r.	**a**t **r**ate

1

Introduction

> *A further attractive feature of the theory [of queues] is the quite astonishing range of its application.*
>
> – David G. Kendall, Kendall [1].

The last few rays of the sun catch the heavy drift of snow teetering precariously on the roof's edge of a big, gray, windowless building deep in the arctic. Outside, the lack of snow on a few parked four wheel drive cars is the sole evidence of ongoing human activity. The only life to be seen is a timid white fox peeking out of the surrounding forest only to quickly slink back into its deepening shadows. It's early afternoon, and a scene that couldn't be further from urban modernity is already being plunged into darkness.

Inside the building the air is heavy with a complex mixture of hums. Narrow corridor upon narrow corridor stretches off, each into its own vanishing point. The slightly more than head high walls of the corridors are covered in madly blinking LEDs. It feels like the transition from outside to inside was, even more than a shift from tranquility to noise, a step straight into the center of our modern, digital, world.

Acclimatisation happens slowly, but after a while it starts to become possible to tease out the different strands of noise filling the vast volume to the brim. It is not warm in this room but there is a strange sense of furious activity, and somehow it feels like it should be. That's what the strongest, deepest, hum is, air conditioning on an industrial scale. With the hum of the massive fans mentally filtered out the remaining noise, issuing from the blinking walls, becomes recognisable as the sounds of computers steadily at work. Thousands of them.

If we went looking for the physical components of a cloud computing system something like this place is what we might find. In a world where the ability for

everyone, from private individuals all the way to conglomerations of governments, to gather data is increasing day by day so does the need for the ability to store and treat that data. The desktop computer, once replacing the mainframe terminal as the standard method of computing for most applications, is diminishing in importance as the locus of computation. The increasing importance of cloud computing could be viewed as a step back towards the old familiar mainframe computing, but there are interesting differences between the two paradigms. Certain aspects of those differences lead to facilities like our arctic data center being built in remote and seemingly strange locations while other aspects of the differences can help to put the theoretical work in this thesis into a real world context.

Cloud computing is the situation when some entity offers access on some form of ad hoc basis to very powerful data processing capability Voorsluys *et al.* [2]. One of the results of this is that there is less need for individual people, companies et cetera to own and maintain their own capacity for powerful computing. This in turn means that the computing can be located in the optimal place for the computation itself, rather than physically close to the user, which accounts for why many such facilities are built in locations chosen for their readily available free energy, or access to cooling.

However, the decoupling of owning and using the facility also means that in many cases the computational facility is being utilised largely by users who are independent of each other as well as of the entity maintaining the facility. So, in contrast to main frame computing it is difficult, and presumably competitionally disadvantageous to subject the users of the cloud computing services to scheduling. Clearly the flexibility that allows users to post jobs for the system at short notice is important. Likewise there is no reason to expect users of a cloud computing system to have aligned interests, so that they can solve scheduling problems in a co-operative manner. In fact, it is reasonable to assume that sometimes users on the same cloud computing service, potentially having their computations carried out on the same rack, have opposing interests. Maybe we are crunching the same numbers as a competing team, and whoever finishes the calculation first will be able to make an investment, or update a block chain, and the other won't.

This all boils down to a situation where a very high volume of time critical jobs are being carried out in parallel by servers who might be indifferent to the actual distribution of jobs within certain parameters. This is the context in which we ask a number of questions related to routing and system behaviour. What happens to the time a job takes under certain changes of parameters, such as the number

of jobs in the system, when we have some understanding of the structure of the decisions that the users make? What will the strategies that a user, particularly a user who is a computer, comes up with to optimise the use of the system look like? Does the system design affect the ability for a set of idealised users to formulate and implement optimal strategies for utilising the system? We come to the conclusion that while systems of this kind are well behaved under certain relatively simple assumptions, it is a non-trivial matter in terms of system design to ensure that those assumptions hold in the context of users making selfish routing decisions in the system.

1.1 Background and motivation

Consider a set of queues in parallel. Each queue is attended by a server who works in such a way that as soon as a user joins a queue they start receiving service, and as soon as a user's service requirements are met they leave the system. In each queue the server divides a constant total amount of service per unit time among the users currently in the queue they are serving. So each user gets a smaller amount of service per unit time the more other users there are in the same queue. When a user arrives to the system they know everything there is to know about how the system works; how fast the server in each queue works, how arrivals are distributed over time, and the limit on the number of users that the queue can accommodate at any one time if there is one. They also instantaneously find out which state each queue is in by finding out what amount of work is already present in each queue. Based on all of this information the arriving user decides which queue to join in such a way that they themselves will have the cheapest possible queuing experience, without considering how their decision will affect the system performance in general.

In other words we consider a set of parallel *processor sharing*, *PS*, queues where users employ selfish state-dependent routing. We give a more rigorous definition of the system under consideration in Chapter 2.

While the work in this thesis is theoretical in nature, and the system under consideration is highly idealised, it is clear that systems that can be modelled in this way are important in the real world today, and that their importance is constantly growing.

1.1.1 Queuing networks

The quote at the start of this chapter is from the introduction to Kendall [1], among the first papers to explicitly discuss the "Theory of Queues". We shall see that if this was true in 1951 then it is certainly true now. The theory of queues, or queuing theory as it is commonly called today, and its extension to networks of queues is central to this thesis. Therefore briefly discussing what queuing theory is, and considering some of the "astonishing range of applications" seems like a natural starting point for this text.

Queuing theory deals with the situation when a set of agents (potentially with individual properties) seek access to some resource which can only be acquired by interaction with some entity (also potentially with individual properties) that dispatches the resource in such a way that agents might have to wait an amount of time with some dependence on the presence of other agents before being granted access to the resource. It is common when first coming into contact with the subject to think of the agents as customers in a store, the entity granting the resource as a check-out clerk and the resource they are seeking access to as the attention of a check-out clerk. Here the dependence on other agents is obvious as the limited number of clerks means that the amount of time that a user has to spend waiting for the attention of a clerk depends on the number of other users in the system. It is common to represent queues graphically by figures like the one given in Figure 1.1.

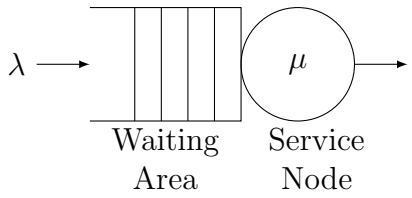


FIGURE 1.1: Common visual representation of a single queue with a single server. λ denotes the rate of arrivals to the system and μ denotes the rate at which each server completes jobs.

Before considering applications we add another level of complexity to the discussion and define the concept of a queuing network. In this definition as well as the notation used in this text we largely conform to conventions employed in for example Walrand [3] and Bolch [4]. A queuing network is any system consisting

of several service stations. The distinction between a single queue and a queuing network is not all that important, and therefore not completely clear, but it is common to think of two servers dealing with a single set of waiting customers as one queue, but two servers dealing with two distinct sets of queuing customers as a queuing network.

So the fundamental building block of a queuing network is the individual queue, with it's attendant service station. The next step in complexity is to combine two queues in some way. There are two natural structures.

The first is for the two queues to be in parallel, where some users go to one queue and others to the other, but without any particular order between the two, or intrinsic assumption that some or all of the users will visit both queues. A visual representation of the most general version of a queuing system consisting of parallel queues is given in Figure 1.2.

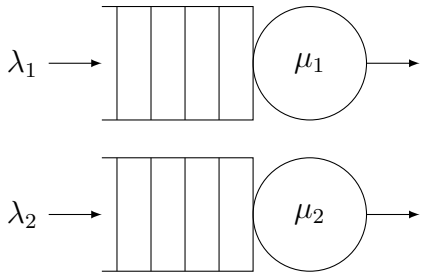


FIGURE 1.2: Visual representation of queues in parallel without explicit interaction. λ_i denotes the rate of arrivals to queue i and μ_i denotes the rate at which each server in queue i completes jobs.

In the interest of generality the queues here are not explicitly connected in any particular way. Each queue is shown with an associated stream of arrivals, and upon leaving the queue the users leave the system. Commonly, and particularly in this thesis, the two arrival streams, here shown with rates λ_i, will be related in such a way that they emanate from some common stream of users. In these cases an individual user ending up in one queue or the other is the outcome of a decision based on some property of the system.

In this thesis all of the objects of interest are versions of this queuing system. For completeness, though, we also mention the alternate arrangement; two queues in tandem. We give a visual representation of a general system of two queues in tandem in Figure 1.3

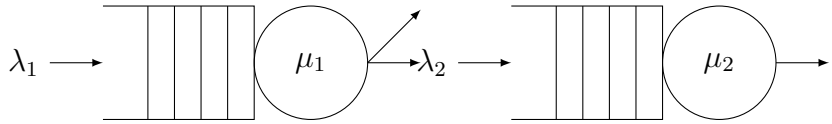


FIGURE 1.3: Visual representation of queues in tandem with general onward routing. λ_i denotes the rate of arrivals to queue i and μ_i denotes the rate at which each server in queue i completes jobs.

Queues in tandem are arranged in such a way that some proportion of the users who have passed through the first queue continue on to the second one. So that λ_2, the rate at which users arrive at queue 2, in Figure 1.3 is related in some way to λ_1, the rate at which users arrive at queue 1. Again, in the interest of generality no specific structure of this dependence is explicitly included in the figure.

With these basic building blocks, along with some other concepts such as feedback, it is possible to create queuing systems of essentially arbitrary complexity. Since the properties of these atomic structures under a variety of conditions are well understood, the analysis of properties of large systems that might otherwise be intractable falls within reach.

For a much deeper introduction to the concept of queuing networks see for example Walrand [5] and the references therein.

While this theory can be fruitfully employed to deal with a variety of physical systems where people queue up for goods and services, the true strength of the theory arises from the fact that a vast variety of physical as well as abstract situations can be thought of as users arriving to some system to seek some sort of scarce service. One important such system to model from the very beginning of the field of queuing (going back to when it was only a loose collection of problems related to 'congestion'), as is clear from the citations given in Kendall [1], is telecommunications which has clearly not lost any currency in the time since 1951. However, while the importance of telecommunications has only increased, it has arguably been overtaken by computing which is an ever growing area of interest for queuing theory. When we add that queuing theory is now commonly employed in everything from biology (see for example Evstigneev *et al.* [6]) to finance (see for example Kelly & Yudovina [7]) it is easy to see what Kendall meant by the "astonishing range of applications".

1.1.2 Service disciplines, processor sharing

The processor sharing discipline works in such a way that the server splits their attention among all of the users currently present in the queue they serve. One result of this is that later arrivals affect earlier ones. This is clear if you consider the situation that someone arrives to the queue in which we are waiting. Regardless of the number of other users who were already present in the system, the addition of the new user means that the server will spend some of the service they would otherwise have given to us to the new user.

It is fruitful to consider the Processor Sharing Discipline in the context of other service disciplines with the same complicating factor that the arrival of a new user reduces the service rate, or otherwise increases the expected waiting time, of users already in the system. We note here that there are many other service disciplines besides processor sharing which have this property. One relatively well understood such discipline is the *Last In First Out*, *LIFO*, (see for example Kelly-Bootle & Lutek [8], Robert [9]) also known in the context of queuing theory as the *Last Come First Served*, *LCFS*, service discipline and in computer science where it is referred to as a *stack*. This service discipline comes in two different versions, which share the negative effect of later arrivals with Processor Sharing to different degrees. The version that most clearly expresses the property is the situation when a user cannot be served unless it was the last arrival to the system, so that a new arrival replaces the user currently in service. LIFO is however also frequently used to refer to the case when no interruption of ongoing service occurs, but the next user to start service after a service completion is the latest one to arrive to the system. While the two cases lead to different system behaviour, particularly from the point of view of the individual user, in either case we see that a system working under LIFO shares some relevant properties with a system working under processor sharing. In particular we note that in all these cases the time that a particular user has to spend in the system depends greatly on the number of other users who arrive *after* the user's arrival.

We note as well that there are other service disciplines in which the time that we have to spend in the system is affected by users who arrive to the system after us, but where the relationship is inverted as compared to that in processor sharing. One example of that is the Batch Service discipline, which is another service discipline name that encompasses several distinct but related service disciplines. In general it can be thought to refer to any situation where a server deals with users in

bulk, or as a batch, rather than individually. Simple examples of this are systems such as elevators or public transport. For the purposes of this discussion we follow Bell & Stidham Jr. [10] and use the second definition on page 186 of Medhi [11]. In this context the Batch Service service discipline is such that service does not commence until a sufficient number of users have arrived, but when the requisite number of users are present in the queue the service commences immediately and everyone is simultaneously served.

Intuitively we can see a difference between the LIFO and PS service disciplines on the one hand and the Batch service discipline on the other. In particular we note that our sojourn time as users waiting in the queue in each case depends on the users who arrive after us, but the direction of the dependence varies. In the case of processor sharing and LIFO we always hope that no one else will arrive during the time we spend in the system, as that is the situation that will minimise the amount of time in the system, while in the case of the Batch Service queue we are either the person who fills up the batch, and later arrivals do not matter to us, or there is room left in the batch after we arrive, in which case we hope for more people to arrive as soon as possible so that service can commence.

This difference turns out to be of huge importance when we consider users who try to optimise their own experience of the system in an adaptive way.

1.1.3 Routing

As mentioned above this thesis focuses on a particular situation when it comes to the way that users pick their way through the system; selfish state-dependent routing. We briefly discuss the two constituent parts of this concept separately here.

State dependent routing

When considering the routing decisions made for, or by, a user who is traversing a system of queues the amount of information available to the router matters greatly to the type of strategies that can be employed. In particular knowledge of the moment-to-moment state of the system in terms of amount and distribution of unprocessed work present in the system can be used to great, and interesting, effect. We call the situation when the current state of the system affects the routing through the system *state-dependent routing*, as opposed to *probabilistic routing*.

Note that state-dependent routing need not be deterministic as a function of state, we would refer to this as state-dependent routing with *pure policies*, as opposed to state-dependent routing with *mixed policies*, which is the situation when each state is associated with a probability distribution over all possible decisions. State-dependent routing is also referred to, for example in Altman *et al.* [12] and some of its references, as *dynamic routing* as opposed to *static routing*.

Again we consider the concept by contrasting it to its alternative.

Consider the case of a driver considering which of two routes to take.

If the driver is familiar with the two routes they might have an idea of which route is usually faster, and they might have some inkling of how the time of day affects the relative merit of the two routes. As an extreme case we can even convince ourselves that the driver will make the same decision every time they are faced with the choice, as long as none of the parameters that the driver has access to, such as season and time of day, changes. This would represent a case of state-independent routing.

By contrast we can consider the case when the driver is given instantaneous access, perhaps via a personal hand held device, to information about congestion on the two potential routes. In this case the knowledge that the driver has about road conditions, the way that congestion usually develops over time on the different routes et cetera, will still enter into the decision process, but so will the current state of the road network. Now the extreme case would instead be that the driver makes the same choice each time they come to the decision point when the external factors from the previous situation are all the same *and* the distribution of traffic on the two routes is the same.

Certain aspects of the relationship are nearly trivial, such as the superiority of state-dependent routing in cases when there might be large random fluctuations in usage of the different routes, and the ability of state-dependent routing to deal with extremely rare events, such as traffic accidents. This type of adaptivity is treated for example in Kelly [13]. Other aspects are more subtle, but can be understood without any particularly deep discussion. In this category we note particularly the fact that it is always possible for state dependent routing, correctly implemented, to deliver outcomes, in terms of for example the average delay, of the same quality with respect to the operation of the system as a whole as state-independent routing, while the converse is not true.

However, state-dependent routing, particularly as coupled with selfish routing, can give rise to problematic system behaviour, including some well known

paradoxes. We discuss this further below.

It is interesting to note that in every day life more and more situations that used to be close to ideally state-independent are becoming more state-dependent, in some cases in a gradual and increasing fashion. As an example we can stay in the realm of private traffic and compare the current situation for a person driving home from work with the same situation in the probabilistic days of yore. Gradually a set of technologies have been introduced to make the choice of when to leave work, and what route to take more and more dependent on the actual state of the road network. As just a few examples we mention real-time information boards specifying time between motorway off ramps, web cams showing the current state of certain bottlenecks and real time updates on the congestion state of the road network as a whole available as map-overlays in hand held devices.

One can think of this tendency as being based on the fact, mentioned above, that any routing policy available to a probabilistic router is also available to a state-dependent one. However, as we shall see, while the same strategies are indeed available in theory, the added information in the wrong hands might mean that the policies that would be preferable both from the perspective of the individual users and the system as a whole will never be utilised in practice.

Selfish routing

The final piece of the routing puzzle relates to who the routing is optimised for. The distinction is now between the case when the route is chosen in such a way as to optimise system performance in total, and the situation when it is chosen in such a way that it optimises the outcome for the individual user. We refer to the former as the *socially optimal* route and the latter as the *user optimal* route. In this thesis we focus on the structure and availability of user optima, and do not explicitly discuss social optima. For the same reason we also do not dwell on another common subject in the context of user optima; discussing the *price of anarchy*, which is the commonly given to the difference in cost between user optimal and socially optimal routing. In order to explain why we have chosen to focus on selfish routing we will, however, consider a well-known result related to these concepts, *Braess' Paradox* first described in Braess [14] (English translation Braess *et al.* [15]), which nicely illustrates the type of problems and counter-intuitive situations that can arise when selfish, state-dependent, routing is employed.

In short the paradox describes the initially counter-intuitive case when adding an edge to a road network increases the travel time for all users on the network. The paradox arises due to the fact that each user can find a way to lower their own travel time by using the added edge, but by doing so they increase the travel time of all of the other users. So when all the users ignore the outcome for the system as a whole, they are also making the outcome worse for themselves. The problem is that from the point of view of every individual user forgoing the short cut when everybody else is using it, would increase their individual travel time, and only when a significant number of users chose the path without the short cut could any real improvements be made to the total travel time in the system.

This situation is an example of a Nash Equilibrium, first described in Nash [16] and more recently in the context of queues in Hassin & Haviv [17]. Somewhat informally a Nash Equilibrium is the situation when no user can improve their own outcome by unilaterally changing their decision. The keyword here is *unilaterally*, and we consider Braess' Paradox again to see exactly what this means.

As is illustrated by Braess' Paradox a central planner could have gotten everybody to use the routes they would have used had the shortcut not existed, and just route particularly important traffic along the shortcut. Thereby keeping the previous, preferable, situation for the average user while having the opportunity to utilise the short cut if need be. The trouble is that this solution cannot be found by users seeking to optimise their own travel through the system. We'll see why by once again thinking of ourselves as a user in the system. If everybody else utilises the short cut, it is marginally optimal for us to do so too, as it lowers my travel time, if only slightly. So we will use the shortcut.

As a side note we also note that in the situation when no one else uses the short cut, so that we have a chance of being the only users utilising it, we will gain even more from using the short cut. So we will use the shortcut, and so will every other user in the system faced with the same decision, showing that not only is the situation when everybody uses the short cut an equilibrium, it is also attractive.

The point to note here is that selfish state-dependent routing leads to a situation when users seeking to minimise the time spent in a system in fact achieve the opposite not just for the system as a whole, but even for themselves. However, the problem goes further than that, the users are actually unable to find the better solution without communicating with each other, or being centrally directed.

Thus, while it might seem as though the problem lies with the greedy drivers who can't just stay off the short cut, we would argue that Braess' Paradox, and

many other instances of selfish routing leading to paradoxical or problematic outcomes is actually a result of poor system design. In fact, we can even formulate this as a heuristic: When designing a network through which users will make their own routing choices it should be done in such a way that the users can find the socially optimal route through the system.

However, we note that the reason that we can easily tell that the Nash Equilibrium in the canonical Braess' paradox is suboptimal is that we are aware that the solution from before the addition of the short cut is still available to the users, even if they can't find it. In more complex situations it might be computationally heavy, or even intractable, to find a global optimum, both socially and selfishly. In this thesis we focus on equilibria with respect to certain algorithms of optimisation, and the structures of these equilibria, rather than comparing equilibria to optimal outcomes or considering the price of anarchy.

With this in mind the potential problems that we alluded to in the description of the processor sharing service discipline, Section 1.1.2, start to become obvious. In particular, as we have noted before, the fact that later arrivals negatively effect the service rates of users already in the system is a complicating factor. We think it is reasonable to expect that this property, coupled with the strong state dependence of the amount of service received by each user in the system, means that systems including processor sharing queues are perfect set-ups for problems of the type exemplified by Braess' paradox.

1.1.4 Methods

All the work presented in this thesis takes place within the context of applied probability, and in particular stochastic processes. An excellent, comprehensive, introduction to these field can be found in Grimmett [18]. Part of the work, in particular Chapter 4, has an important numerical component. In the majority of this work the numerical work itself is not novel, as opposed to the analytical formulations and the interpretations of the outcomes. Nevertheless, numerical linear algebra is important to the work, and a comprehensive introduction to this field can be found in Trefethen [19].

Two concepts end up being of particular relevance to this work, *Stochastic Coupling* and *Markov Decision Processes*, so we take a brief look at each of these individually.

Stochastic coupling

The proofs of the majority of Theorems we present, in particular in Chapter 3, rest on stochastic coupling arguments.

In essence the method of stochastic coupling is based on the idea that it is possible to compare properties of some set of random variables by defining a new set of random variables with the same marginal laws as the random variables of interest but with known joint distributions. The most common approach is to examine the properties of interest by letting the "same randomness" govern all of the variables or processes of interest.

We can see how this works by considering a trivial example.

Example 1.1.1 (A crooked coin). *Let's assume that we have two unfair coins, coin 1 and coin 2, and let H_i and T_i be the events that coin i shows heads or tails respectively. Let the coins be unfair in such a way that $\mathbb{P}[H_1] \geq \mathbb{P}[H_2]$. We then define the random variables X_1 and X_2 such that X_i is the number of Heads out of n throws of coin i. We can now show, for example, that $\mathbb{E}[X_1] \geq \mathbb{E}[X_2]$, by using a coupling argument.*

We do that by constructing a random variable Y by taking the set of outcomes for X_2 and change each Tails to Heads with probability $\mathbb{P}[H_1] - \mathbb{P}[H_2]$. We then note that Y has the same law as X_1.

Thus, we have shown that we can construct a random variable with the same distribution as X_1 by taking the random variable X_2 and adding a non-negative random amount to it. Thus we can conclude that the expected value of X_1 is greater than or equal to the expected value of X_2.

The extremely simple Example 1.1.1 does show what is meant by comparing some property of two random variables by considering how they would behave had they been based on some common source of randomness. In this context the word coupling simply refers to how the random variables are coupled to each other by deriving their randomness from the same source. However, coupling also commonly has another meaning, particularly in the context of coupling arguments related to stochastic processes.

In the context of stochastic processes a common, but not guaranteed, feature of coupling arguments is a *coupling event*. This concept occurs frequently in the proofs in this thesis, so we endeavour to explain it here. The concept of a coupling event is somewhat linguistically confusing, because while a coupling event is

certainly intimately related to the fact that the random processes under consideration are founded on some common source of randomness, which is what we have come to understand the word coupling to mean, it is not identical to that concept. Instead a coupling event is the situation when a set of stochastic processes end up in such a relationship that that relationship is maintained for all future states of the joint process. The most commonly seen type of coupling event is when two processes end up in the same state, and will remain in the same state for all eternity after that.

We give a trivial example of how a coupling event can enter into a coupling argument as well.

EXAMPLE 1.1.2 (Random walking in lockstep). *Let $X_1(t)$ and $X_2(t)$ be two stochastic processes on the non-negative integers. Both processes are such that at each integer time point $t > 0$ a fair coin is tossed, if it shows heads $X_i(t) = X_i(t-1)+1$ and if it shows tails $X_i(t) = X_i(t-1)-1$ except if $X_i(t-1) = 0$ in which case $X_i(t) = 0$. Now let $X_1(0) = 2$ and $X_2(0) = 1$.*

We can now again use a coupling argument to show that

$$\mathbb{E}[X_1(\tau)] \geq \mathbb{E}[X_2(\tau)], \forall \tau \geq 0.$$

As long as we assume that we never flip a tails when $X_2(t) = 0$ we note that a stochastic process $Y(t) = X_2(t) + 1$ has the same law as $X_1(t)$. So why did we need to exclude the case when we flip tails in $X_2(t) = 0$? Because in that case the outcome of the flip did not in fact change $Y(t)$, but it would have decreased $X_1(t)$. And what it should have done to $X_1(t)$ in this case is to lower it from 1 to 0, thereby putting it in the same state as $X_2(t)$. However we notice that after such an event we can again construct a process with the law of $X_1(t)$ by just taking the values at each time point of $X_2(t)$.

So we see that we can construct a random process Y with the law of $X_1(t)$ letting $Y(t) = X_2(t) + 1$ until the event at t' that $X_2(t') = 0$ and the coin flip at t' shows tails, after which $Y(t) = X_2(t), \forall t > t'$. And again we note that we constructed a process with the law of $X_1(t)$ from $X_2(t)$ by adding, but never subtracting, something. This again shows that $\mathbb{E}[X_1(t)] \geq \mathbb{E}[X_2(t)]$.

So in Example 1.1.2 we call the event taking place at t', the event that takes place when the Y process goes from being one greater than the X_2 process to having the same value, the coupling event. The coupling event is often important

to coupling arguments as the processes under consideration maintain whatever relationship they have from the coupling event on, so only that relationship needs to be examined with respect to the property of interest. A common outcome is, as is the case in Example 1.1.2, that the coupling events lead to the processes being identical, which in many cases mean that only behaviour up until the coupling event needs to be examined, as no difference between the processes exists from the coupling event on.

While stochastic coupling is a powerful and complex method, these simple examples are sufficient as background for the coupling arguments employed in this thesis. For a deeper discussion regarding the theory of stochastic coupling we cannot recommend Thorisson [20] enough.

Markov Decision Processes

The object of interest in this thesis is the structure and existence, of optimal decision policies in stochastically developing systems. Due to system assumptions, discussed in detail in Chapter 2, this system develops according to a Markov Process, which lends itself to time discretisation and thus reduction to a Markov Chain. For this kind of situation a rich body of theory exists under the name Markov Decision Processes, *MDP*s. In this introduction as well as the work in this thesis in general we largely follow Puterman [21]. We also found parts of Tijms [22], De Leve *et al.* [23], Guo [24] (which deals only with the continuous time case), White [25] and Feinberg & Shwartz [26] (in particular the introduction and the first chapter) useful. It is also interesting to note that MDPs find use outside of decision theory, for an example of this see for example Littman [27].

The following description of MDPs concern the aspects of the theory relevant to the current text, and is not meant as a comprehensive description of the theory. We do mention, however, that the main aspect left out of this description is discounts. In the context of MDPs discounts are used to reflect the difference in perceived value between a reward now, and a reward some time in the future. We have not taken discounts into account in any of the studies presented here, so we do not dwell on them further.

For our purposes an MDP can be viewed as a set of related sets. First of all a set of states, $\mathcal{S} = \{s_1, ..., s_k\}$ describing all the states that the system can be in. To this we add a set of actions, $\mathcal{A}_s = \{a_{s,1}, ..., a_{s,n_s}\}$ available in each state, associated with a set of transitions probabilities, $\mathcal{P} = \{\mathbb{P}(s_{t+1} = s_1 | s_t =$

$s_j, a_t = a_{s_j,1}), ..., \mathbb{P}(s_{t+1} = s_k | s_t = s_j, a_t = a_{s_j,n_{s_j}})\}$ giving the probability of each state transition resulting from each action. Finally there is a set of rewards $\mathcal{R}_{a,t}(s) = \{r_{a,t}(s_1, s_1), ..., r_{a,t}(s_k, s_k)\}$ where $r_{a,t}(s_i, s_j)$ represents the reward, or expected reward, for taking action a at time t resulting in the system transitioning from state s_i to state s_j.

To this we also add the concept of a decision epoch. A decision epoch is simply the time between one decision and the next. In some applications the decision epochs are completely deterministic both in number and duration, so that a decision maker knows both the number of decisions to be made, and the amount of time between consecutive decisions. In many applications, though, there is some stochasticity in one or both of the properties. So a decision maker might know the number of decisions that need to be made, but not the time between consecutive decisions. Alternatively the decision maker knows the amount of time a decision is valid for, but not the number of decisions they will be making; this is in some sense the case in Example 1.1.3. Finally there is the case when both the number of decisions and the timing of them, are stochastic, which is the case we consider in this thesis. While we do, in Chapter 5, comment further on the nature of the events that require decisions, as opposed to those that don't, we do not explicitly discuss decision epochs further.

So these basic building blocks form the framework in which some decision maker considers the value of a particular action in each state of the system.

While the thinking of Markov Decision processes permeate the whole thesis, we in particular consider two concepts explicitly, *backwards induction* and *policy iteration* in order to consider the structures of optimal policies.

The method of policy iteration refers to the method of starting from some *policy*, or decision making scheme, which we can call a and then finding one, let's call it b that does better than a when everyone else uses a. Then a policy c is found that does better than b when everyone uses b and so on. This method is described in detail in the Chapter 4. We just mention here that this method is well understood, but still to some extent an area of active research; both the 2005 paper Chang *et al.* [28] and the 2008 paper Zhu [29] deal explicitly with policy iteration. Note that the name policy iteration is used for a variety of closely related but non identical concepts, so the versions of policy iteration discussed in the 2 citations is not identical to that used in the present text.

Backwards induction is the technique of finding optimal policies for some decision process by starting out by considering the possible terminal states of the

system and considering what the optimal decision would be for a user making a choice just before this terminal state. This decision is made with the knowledge that no more decisions will be made, but will instead result in a payoff of whatever the value is of being in the state that the decision leads to. Thus, we can be certain that this decision is optimal, at least inasmuch as its optimality will not be counteracted by future decisions of other users. So with the knowledge that the final decision in the system is being made optimally, we consider the previous decision, the one that leads up to the state in which the final decision is made. We now know what the probability of each available decision in each available state of the system is in the final step, so the situation is similar to the case with the last step. There is a difference, though, in that now instead of knowing that there will be no more decisions made, we know that there will be exactly one more decision made in the process, but we know what that decision is for each state, so we can plan accordingly. Thus we can find the optimal decision for each state in the situation that there is a total of two more decisions to be made in the system. Then we consider the decision in the step before that, and then the in the step before that, and so on.

When we find an optimal strategy in this way it has the special, and powerful, property that it is optimal against all future optimal moves. In a game theoretical framework it is useful to think of each of the decision epochs as a separate game, thus dividing the full game represented by the decision process into discrete *subgames*. So the equilibrium policy that is such that it represents a Nash equilibrium in each subgame is called a *subgame perfect* equilibrium. Equilibrium policies found by backwards induction are subgame perfect equilibria. In Example 1.1.3 we give a toy example of a game with a optimal strategy that can be found by backwards induction.

EXAMPLE 1.1.3 (Competitive Counting). *Consider a game with 2 players. Let X_t be the state of the game when there are t moves left before the game ends and let $X_\tau = 0$, where τ is the number of moves left when the game begins. You and your opponent take turns picking an a such that $a \in \{1, ..., 10\}$ and thereby defining the next state which is $X_{t-1} = X_t + a$. The person who picks an a at X_t such that $X_{t-1} = 50$ wins. Now we use backwards induction to find an unbeatable, or in game theoretical language 'dominant', strategy for the player who goes first, and thereby prove that there is no dominant strategy for the player who goes second.*

To find a strategy that always wins we start by assuming that we make the final move, which means that we picked an a so that $X_0 = 50$ which means that it has to be the case that $X_1 \in \{40, ..., 49\}$. In order for it to be the case that $X_1 \in \{40, ..., 49\}$ our opponent must have chosen an a to get it there, and since we have to assume that our opponent understands the game, the only reason that they would have chosen such an a is if they had to. The only X_2 in which they are forced to pick such an a is $X_2 = 39$. So while looking for the optimal strategy for the whole game, we have shown that any player who picks an a so that $X_t = 39$ is guaranteed to win.

Now, how do we make sure that we get to pick such an a? Well, if $X_3 \in \{29, ..., 38\}$ we can pick such an a, and we note that $X_4 = 28 \Rightarrow X_3 \in \{29, ..., 38\}$ so we now know that setting $X_t = 28$ guarantees the win.

The argument is becoming repetitive by now, so we can just state the last few steps without much comment. Since we want to be able to pick $X_4 = 28$ we need $X_5 \in \{18, ..., 27\}$, which is guaranteed by $X_6 = 17$.

Then $X_6 = 17$ is guaranteed by $X_7 \in \{7, ..., 16\}$, which in turn is guaranteed by $X_8 = 6$.

So we have shown that whoever picks an a so that $X_t = 6$ has a way of winning the game in 8 moves, which means that the game can be won by the first mover by simply picking $a = 6$, after which playing the rest of the game is just a case of going through the motions.

In its most straightforward form the method can be used to identify strategies for games with a limited or stochastic number of moves, which we show in the trivial Example 1.1.3, but is perhaps at its most powerful when used to identify optimal policies in games with infinite time horizons. In this case there are at least two common proof strategies. The first is, as the name of the method suggests, to use inductive reasoning to show that some strategy is optimal in the base case, and that any strategy that is optimal at t is optimal at $t - 1$ as well. Obviously this kind of argument can include more base cases, and inductive steps. The alternative is to show that as the number of steps increases the optimal policy converges to some policy.

For a more comprehensive discussion of backwards induction and subgame perfect equilibria we recommend Nisan [30].

1.2 Literature review

While the field of Queueing Theory can be said to have David Kendall's 1951 paper Kendall [1] as its starting point, the processor sharing service discipline was not defined until almost 16 years later in Leonard Kleinrock's 1967 paper Kleinrock [31]. In this paper the processor sharing service discipline was proposed as the limiting behaviour of the *Round Robin* service discipline in the case when the length of each service instance goes to 0 It was initially proposed as a means of modelling congestion problems in communications networks, telephony in particular.

Since the publication of that paper the area of telecommunications has only grown in importance and has gotten entangled with, in some cases through budding parts off and in other cases by absorption, the area of computing. At the time of writing the two fields are becoming all but indistinguishable. Through this development the relationship between telecommunications and processor sharing remained strong, and many of the publications in the area relate to either of the two fields of computing and telecommunications, or both. We give a few recent examples of how processor sharing is still a concept of interest in this field, both as an object of study and as a model. The 2014 article Rosberg *et al.* [32] studies a problem similar to ours in the explicit context of server farms, where socially optimal routing policies with respect to throughput and energy consumption are studied. Similarly, Mukhopadhyay & Mazumdar [33] considers a system of many parallel processor sharing queues, and the ability of a particular combination of random routing and the *Join the Shortest Queue*, *JSQ*, policy to reduce the mean sojourn time in such a system. Similarly, routing in the context of a system of parallel processor sharing queues, explicitly modelling a server farm, is considered in Altman *et al.* [34], in which results about the price of anarchy, and other properties of optimal selfish and social routing, are discussed. The system under consideration in this paper is quite similar to the one we are interested in with the major distinction that they are considering what they call *open-loop* rather than state dependent decisions. On the more applied end of the field we find publications such as Lee *et al.* [35] where simulation of processor sharing queues is used to model certain properties of bandwith usage on the internet.

These publications have a more applied focus than this thesis, and in order to find work that is more closely related to our work in terms of its mainly theoretical nature we start by looking somewhat further back in time. In particular the 1990 paper Bonomi [36] provides a solid foundation for the work we present

in this thesis. In this paper job assignment in a system of parallel queues with servers working according to the processor sharing service discipline is considered. In particular it is shown that in the trivial case, when all inter-arrival and service times are exponential, Independent and Identically distributed. *IID*, the JSQ routing method is optimal under the criterion of minimising the mean time spent by a user in the system. They then go on to show, by example, that with heterogeneous service requirements JSQ routing can be improved upon. It is important to note that there are small but non-trivial differences between the situation under consideration in this paper and the one we have studied. In particular the heterogeneity present in the system under consideration in this paper is related to the service requirements of the customers, rather than the properties of the queues. These differences do not invalidate the applicability to the system we study of the result in the homogeneous case. This article has also been referenced in some interesting applied work, among which we mention a few. The article Gupta *et al.* [37] deals with the quality of the JSQ policy in server farms, and Gupta *et al.* [38] describes the low sensitivity to variance in job size distribution in PS server farms under the JSQ policy. Other interesting examples are Fu *et al.* [39] and Fu *et al.* [40] both dealing with energy efficiency in data centers, as well as Lefebvre *et al.* [41] and Yi *et al.* [42] dealing more directly with system performance.

The work in Bonomi [36] is then extended upon in the 2011 paper Hyytiä *et al.* [43] which goes further in considering the relative merits of a variety of policies. The main result in this paper is arguably a closed form expression for the relative values, based on sojourn times, of different states of a single processor sharing queue. This expression is then used to carry out comparisons of a variety of decision policies for a system of 2 parallel M/M/1-PS queues. This analysis includes a version of, partial, policy iteration, but a fundamentally different version than the one presented in this thesis. In particular the convergence issues discussed in Chapter 4 would not be identifiable in the version of policy iteration utilised in this paper due to the fact that they don't run the iteration to convergence. The main thrust of this analysis, though, is the consideration of policies that take not only the occupancy, but the remaining service requirement present in the system, into account. It is again concluded, now with higher specificity to a variety of situations with regards to available information and heterogeneity of the network, that JSQ can be improved upon.

Another interesting application based paper in the area of parallel processor sharing queues is Hoekstra *et al.* [44], in which the problem of routing fragmented

jobs over a network consisting of processor sharing queues in parallel is considered. While the system under consideration is similar to the one we consider, and the results, which are mainly numerical in nature, are interesting, the problem of job splitting is not particularly relevant to the questions we are considering in this thesis.

The papers mentioned above provide the background for the subject matter under consideration as the systems under consideration in them are very similar to those under consideration in this thesis. In terms of method, and research questions, though the closest comparison, except for the 2012 thesis Chen [45] which this work is a direct continuation of, the 1998 paper Altman & Shimkin [46] is the closest background to our work. In this paper a system of two queues is considered, one of them is a single server queue with service according to the processor sharing discipline the other queue is an $M/M/\infty$ queue. The system is described as the situation when users who want to perform some operation on data has the option of either posting the operation to a mainframe which is accessed by many other users, or to perform it on a desktop PC. Since this situation is one dimensional in state space, only the processor sharing queue has states in any meaningful way in this model, the concept of a monotonic policy reduces to the concept of a threshold policy (both concepts described in detail in Section 2.3). It is shown in a variety of situations that the policies that selfish users use in this system are all threshold policies, a construction to find such policies is given and the system performance under selfish routing is compared to that under socially optimal policies. It is interesting to note that this work was published at a time when mainframe computing was relatively rare, and cloud computing had not yet started to be of any relevance. As we have already noted, though, from the perspective of a routing or work allocation problem there are many similarities between mainframe and cloud computing and so the results from this paper have recently seen use in analysis of routing schemes related to cloud computing, see for example Nahir *et al.* [47] and Simhon & Starobinski [48].

We also mention that a similar analysis was carried out with a selfish choice between a processor sharing queue with general service distribution and a constant time server in the 1991 paper Haviv [49]. This analysis differs from the previously mentioned ones in particular in that it more explicitly focuses on the cost of what is here viewed as balking from the processor sharing queue. We also note that in this treatment the choice between processor sharing and constant expected

time processing is viewed as the choice between some local and non-local mainframe computer, once again illustrating the interesting drift from one computing paradigm to another, with a retained relevance of the processor sharing discipline as a model.

Slightly tangentially, but still clearly related to our work, is the work presented in Hassin [50] which studies, mainly numerically, the choice between a First Come First Served queue and a queue in which the server works according to the Random Service discipline. They find some interesting results of sensitivity to choices in the case when the system is empty. This might seem unrelated to our work, but we mention it here partially in order to point out that the processor sharing discipline in the case of exponentially distributed service requirements is identical to random service, except in a few states. However, the states in which the two differ are exactly the case of an empty system, and in the case of high traffic surrounding states. So while interesting, these results are not immediately relevant to our work. Another only superficially related, but interesting, paper is Borst [51], in which socially optimal allocation to a system of m queues in parallel is considered.

Since our main focus is on the routing issue, in particular as it relates to systems of parallel queues containing processor sharing queues, the focus of this review is on work in that area. In terms of the properties of single egalitarian processor sharing queue the state of the art is excellently described in the 2007 paper Yashkov & Yashkova [52] and its references. The same is true for the behaviour in terms of job numbers and transit time in the context of Discriminatory processor sharing, see particularly, Altman *et al.* [53] and its references. While these two publications are recent, the interest in properties for queuing systems with some processor sharing component is ongoing. In particular we mention the 2014 paper Yu *et al.* [54], which derives both individually and socially optimal rules for whether to join a single egalitarian processor sharing queue, and the 2013 ArXiv upload Abramov [55], which finds some further properties of Discriminatory processor sharing queues, and interesting relations between Egalitarian and Discriminatory processor sharing.

Conversely there is ongoing interest in other situations where users make a selfish choice based on sojourn time. One interesting example of recent work in this field is the 2016 article Selen *et al.* [56], in which various properties of a system of two First Come First Served Queues under this kind of routing are derived.

A central component of the work presented in this thesis is the fact that selfish routing has the capacity to lead to equilibria that are less than desirable from

a social perspective. Above we discuss the fact that it has been known since at least the 60s that selfish routing has the capacity to lead to interesting paradoxes. Identifying such paradoxes is still an area of active interest, see for example Ziedins [57]. We also want to mention that the study of such paradoxes as well as generally studying the price of anarchy, the difference in some cost between social and selfish optima, in various systems has remained of interest. We mention a few examples of publications dealing with these subjects. While somewhat tangential to the subject of this thesis we still want to mention the seminal paper Naor [58] where the possibility of aligning social and selfish optima through tolls is introduced. As an example of a paper slightly more closely related to the work presented here we mention the 1983 paper Bell & Stidham Jr. [10] which deals with the relationship between social and individual optima in a system of parallel preemptive queues, in this case queues where the server works according to the Last In First Out discipline. More recently we have the 2002 paper Roughgarden & Tardos [59], the 2004 paper Parlaktürk & Kumar [60], as well as the 2007 paper Haviv & Roughgarden [61] each of which deal explicitly with the additional system costs that arise as a result of selfish routing and derive interesting results regarding what to expect of it, and how to counteract it.

For most of the work presented in this thesis it is assumed that any arriving user for whom there is room in the queuing system will join it, in other words we mostly assume that no user who can join the system chooses to balk. Likewise for all the work in this thesis we assume that a user who has picked a queue will stay in it until they have been served, which in the language of queuing is often expressed as the process we are considering being free of jockeying and/or reneging. The question of whether to join a queue, and whether to stay in a queue one has joined, while not an object of explicit interest for us is an interesting and important one and for a comprehensive discussion on this subject as well as a survey of the literature we highly recommend the book Hassin [62], and the references therein. In particular the chapters on decision making with relation to Egalitarian Processor Sharing is relevant as background to the present work.

1.3 Thesis outline

In the next chapter, Chapter 2, we give a rigorous description of the system that is the central object of interest in this thesis. In this chapter we give definitions of

the main concepts, as well as present the central assumptions, that run through this thesis.

In the following chapter, Chapter 3, we present the first results in this thesis. These consist of a set of theorems related to system properties in the context of varying system parameters, which we present here together with their stochastic coupling based proofs.

By contrast to the analytical nature of that chapter the following chapter, Chapter 4, deals largely with numerical exploration of user decisions in the system. In this chapter we take as our starting point the counter-intuitive, and in some sense problematic, outcomes of policy iteration presented briefly in Chen [45]. By extensive exploration of parameter space under a variety of user properties we can present, in the form of theorems largely proved by example, interesting conclusions about both the lack and presence of structure in this system.

After that, in Chapter 5, we consider a different approach to essentially the same set of equations as those underlying the numeric work in Chapter 4. In this chapter we show, by a value iteration like analytical approach, that it is possible to define user behaviour in such a way that there are at least guaranteed to be fixed points. In this chapter we also discuss two smaller studies which provided valuable insight into the system.

Finally, in Chapter 6, we provide a unified discussion of the results presented in the thesis.

2

System definition

It has been said that a specialist is a barbarian whose ignorance is not well-rounded.

– Stanisław Lem, His Master's Voice

In the majority of the following the same system of queues will be studied. In this chapter the structure of that system is presented, the fundamental assumptions on the system are laid out and the mathematical notation is defined. The first results from studying the system, along with proofs, are also presented in this chapter. In Section 2.1 the structure of the system is described and a mental model to keep in mind is suggested. In Section 2.2 the process assumptions that are used throughout are presented and similarities to some other systems due to those assumptions are discussed. In Section 2.3 the, in some sense, central feature of this work - the decision rules and decision policies - are introduced.

2.1 System structure

In general we study a system of M queues in parallel, but for the most part the specific system in question is made up of only two queues so this is the system structure that we present here.

We consider a queuing system made up of two queues in parallel. We call them queue 1, Q_1, and queue 2, Q_2. To emphasise the symmetry of the system we will often call the queues queue i, Q_i, and queue j, Q_j. The customers are served by a single server per queue, each of whom work under the processor sharing service discipline which, as touched upon in Chapter 1 means that the servers are simultaneously serving everyone who is currently in the system. In particular the

servers work according to *Egalitarian Processor Sharing* or *EPS*, see Kleinrock [31]. Under this service discipline all users in the system get the same amount of service per unit time. Consider a server working according to the EPS discipline providing service at a total rate of μ. If there are n customers in the system the effective rate of service that a customer actually receives is $\frac{\mu}{n}$. As we only consider Egalitarian processor sharing we simply refer to it as processor sharing or PS.

The only way that a user can leave the system, once they have joined it, is by completing service. Furthermore users are not allowed to switch queues once they have joined a queue. In other words, customers are not allowed to balk once they have joined the system, neither from the system or from the queue they have joined to the other one.

In general the users arrive to the system in three streams. One stream consists of designated arrivals to queue 1 and another of designated arrivals to queue 2. The third stream is our main focus of interest, and the only one that will always have a non-zero rate. These are the general arrivals who have a choice of joining either queue 1 or queue 2.

We consider both limited and infinitely large queuing systems; it is made clear in each situation which case is under consideration. In either case the occupancy of Q_i is referred to as n_i. We often refer to the occupancy of the whole system as a vector $\mathbf{n} = (n_1, ..., n_M)$, in the case of M queues. Here we also introduce $\mathcal{M} = \{1, ..., M\}$, the index set of the M queues. In the cases when the capacity is finite we let N_i denote the capacity of Q_i, $i \in \mathcal{M}$.

We are sometimes interested in the occupancy of the system at a specific time. However, we almost always uniformise time so that we don't work in the continuous time domain but rather in discrete time defined by events in the system. We nevertheless generally refer to this simply as time, and call it t. When time is a factor we refer to the occupancy of Q_i at time t as $X_i(t)$, and to the occupancy of the whole system at time t as $X(t) = (X_1(t), ..., X_M(t))$. We note here that time is also used in some parts of the following to denote steps in iterative processes, but this should not lead to any ambiguity. These different uses of time are discussed further in Sections 2.2 and 2.3.

For reasons further discussed in Section 2.2 the state of the system is mostly identical to occupancy. The main exception to this are situations when specific states are added to account for, for example, service completion for a marked customer. Therefore we also call system states $\mathbf{n} = (n_1, ..., n_M)$. This should not lead to confusion as in the situations when time is of no importance we will always

be interested in the system only in terms of its states and in the situation when time is a factor it is clear whether we are talking about states or occupancy. In these cases we might for example write: $X(t) = \mathbf{n}$ which should be taken to mean that the occupancy at time t is such that the system is in state $\mathbf{n}$.

The possibilities for a mental image that would correspond to this system are of course essentially endless but we suggest two here.

The first suggested mental image conforms to the idea of processor sharing as the limit of the *Round Robin* service discipline. Round Robin is the service discipline where the server, who works at rate μ, deals with each of the n customers currently queuing in turn either until they have received the amount of service that fulfils their needs or until they have been served for a specified time t, see Kleinrock [31]. Then the server moves on to the next customer. The process is repeated until the server has dealt with each customer currently queuing after which they return to the first customer in the queue and repeat the process. This means that a marked customer receives at most $t\mu$ units of service per round, after which they have to wait, at most, $(n-1)t$ units of time until they receive another $t\mu$ units of service. In the limit as t goes to 0 the round robin service discipline we obtain the processor sharing discipline. With this in mind an idealised version of parallel computing in computers certainly fits the idea of processor sharing. Therefore one mental model to keep in mind when thinking about the following is a set of idealised file servers. There could be any number, but let's consider two for now. We call the servers S_1 and S_2. The servers each hold two files of equal size. S_1 holds files called a and b while S_2 holds the same file a and in addition to that a file called c. There are three types of users who access the file system. One type needs file b, and can therefore only be served by S_1. These users represent the intrinsic arrivals to queue 1. Another type of user needs file c, and can thus only be served by S_2. These users represent the intrinsic arrivals to queue 2. The third type of user needs file a. These users can get what they need from either of the two servers, so they choose which server to connect to based on which one will finish their request quicker. These users represent the general arrivals.

Another way of more directly picturing processor sharing is to think of continuous flows of some substance. To this end consider two pumps dispensing liquids. Each pump works at a constant rate to feed a set of outlets in such a way that a total constant amount of each liquid is available in the system per unit time. Pump 1, P_1, dispenses liquid a with a flow of μ_a units of liquid per unit time, Pump 2, P_2 dispenses liquid b with a flow of μ_b units of liquid per unit time. All users

simultaneously hooked up to outlets connected to the same pump share the flow equally so that when there are n users connected to P_i each of the users receive a flow of $\frac{\mu_i}{n}$ units of volume of the liquid per unit time. Now we just need to define the types of users to complete the picture. One type of users are only interested in liquid a and will thus always connect to P_1, these are the designated arrivals to Q_1. Another type of users will only be interested in liquid b and they will thus always connect to P_2, these are the designated arrivals to Q_2. The final type of users can use either liquid a or liquid b, and can therefore pick whichever pump will minimise their time in the system. These are obviously the general arrivals.

These two mental images illustrate some different ways of looking at the system. The liquid-scenario emphasises the pure processor sharing behaviour, the situation where we have a limited, but truly continuous, flow that is divided up between all the users queuing concurrently. The idealised file server emphasises the interpretation of processor sharing as the limit behaviour in systems of discrete time sharing. Likewise the two mental images offer up different interpretations of the three types of users. In the case of the file servers the general arrivals are general because they want one thing that both of the servers offer up, and they can therefore go to either server to get their needs met. In the pump example one type of user is indifferent to the liquid they receive, and they are thus indifferent to which pump they connect to. The fact that all types of users in the mental image based on the idealised file server are discriminating is incidental. We could just as well have imagined that the general arrivals were indifferent between file b and c to achieve the same effect. Equivalently we could imagine a continuous scenario with three discerning types of customers.

2.2 Assumptions

Several assumptions run through this study, some quite intuitive, and some a little more subtle. We explain, examine and to some extent motivate them in this section. For completeness, and since it doesn't add substantially to notational complexity, these assumptions are stated in the case of M queues.

In all of the following we assume that arrival as well as service events happen according to independent Poisson processes. The following definitions and notation are used throughout this thesis. The designated arrivals to queue i occur according to a Poisson process at rate σ_i. The general arrivals to the system occur according

to a Poisson process at rate λ. Likewise the potential service completion events at queue i occur according to a Poisson process at rate μ_i. When a service completion event occurs while the relevant queue is empty it is a null event and does not change the state of the system.

It is often convenient to collect the system parameters in a parameter vector $\mathbf{\Gamma} = (\lambda, \mu_1, ..., \mu_M, \sigma_1, ..., \sigma_M, N_1, ..., N_M)$, where it is possible that any, or all, $N_i = \infty$. The $\mathbf{\Gamma}$s are taken from $\mathcal{G}$, the set of all parameter vectors.

In general, without any knowledge about the distributions governing the arrivals and services, the state of the system would be quite a complex object. In particular it would need to include the arrival times of each of the customers currently queuing, as well as the time since the last arrival of each type.

However, the memoryless property of the exponential distribution means that the occupancy of the system is a Markov process with state space $\mathcal{S}$. In the case with limited capacity we have $\mathcal{S} = \{\mathbf{n} = (n_1, ..., n_M) | 0 \leq n_j \leq N_j, j \in \mathcal{M}\}$ and for the infinite system $\mathcal{S} = \{\mathbf{n} = (n_1, ..., n_M) | 0 \leq n_j, j \in \mathcal{M}\}$.

Since the state of the system changes only based on a set of Poisson processes we can also define an overarching *event process*, which describes the process of events that potentially change the state of the system. As this process is a sum of Poisson processes it will also be a Poisson process. The event process has rate $\lambda + \sum_{i \in \mathcal{M}} \sigma_i + \sum_{i \in \mathcal{M}} \mu_i$. This process is relevant in itself in certain situations but most often comes up in the many cases when the queuing process is studied in terms of the embedded Markov chain that it induces.

When we study the finite system in practice there are two sets of states that are often treated as distinct, the *internal states* and *edge states*. We call the set of edge states $\mathcal{S}^e$ and the set of internal states $\mathcal{S}^-$ and define them as follows

Definition 2.2.1. $\mathbf{n} = (n_1, ..., n_M) \in \mathcal{S}^e \Leftrightarrow \exists\ i \in \mathcal{M}$ s.t $n_i = N_i$,
$\mathcal{S}^- = \mathcal{S} \setminus \mathcal{S}^e$.

These distinct sets are particularly relevant in Chapters 4 and 5.

The system lends itself well to a simple, and fruitful, way of thinking that is fundamental to, among other things, the coupling arguments in Chapter 3. Consider the server in queue i. The general way of thinking about how they provide service to the customers in queue i is, as described above, that they continuously provide service at rate $\frac{\mu_i}{n_i}$ to each of the n_i users currently in the system. After an amount of time with expected value $\frac{1}{\mu_i}$ one of the users will have received sufficient service to be able to leave the system, satisfied. The alternate way to think of the

system is that the server does not interact with the users at all during the service time, but instead spends an amount of time with expected value $\frac{1}{\mu_i}$ producing one complete service. Once the server is done creating the service they instantly pick one of the n_i users currently queuing uniformly at random, so that each user is picked with probability $\frac{1}{n_i}$, and awards that user the service. The user then leaves the system, satisfied.

This alternate way of thinking about the processor sharing discipline makes the process very similar to another service discipline, the *random service*, or *RS* discipline. In this service discipline the server picks a user currently queuing at random and provides them with service up to their service requirement, then picks a new user among the ones who are now in the queue and fully serves them, and so on. Clearly the general version of this discipline differs substantially from the general version of the processor sharing discipline. But in the case when all events occur according to Poisson processes the two disciplines might at first glance seem interchangeable. There is, however, at least one key difference. Consider two queues, in one the server works under the PS discipline and in the other the server works under the RS discipline. Initially both queues are empty, so both servers are idle, then a user arrives to each queue. The sole user obviously starts receiving service in both cases, but while the first user is being served in each queue a second user might enter each system. In the PS queue this user now also starts receiving service, and in fact is equally likely as the first user, or any other users who might arrive before the first service completion event, to receive the completed service. Meanwhile, in the RS queue, the user who arrived first is guaranteed to be the one to receive the first completed service after which a new user is picked for service among the users who have arrived during the first user's service time. This difference is not trivial when it comes to making the optimal decision of which queue to join, particularly in low occupancy states.

We assume that all users have complete knowledge of the system. This means that the decision that a user makes is based on knowing the system parameters, the state of the system upon their arrival as well as the decision policies of all other general arrivals, as they all use the same decision policy. This is discussed further in Section 2.3.

A smaller, but relevant, assumption is that decisions by general arrivals are made instantaneously upon joining the system. Given the Poisson arrivals and services, the only effect this has is that there is no chance that an event takes place while a general arrival makes a decision. In other words the state that a

general arrival observes is also the state in which they will be joining the system.

With all these assumptions in place we can state that each queue is a $\cdot/\mathrm{M}/1-\mathrm{PS}$ queue in Kendall notation. The initial $\cdot$, rather than M comes from the fact that even though the inter-arrival times to the system are in fact exponentially distributed this is not necessarily the case for the arrivals to an individual queue due to the state dependent choices of the general arrivals.

2.3 Decision rule and policy

The two most central concepts running through this thesis are the *decision rules* and the closely related *decision policies*. We start by defining the decision rules.

2.3.1 Decision policy definition

Consider a system of M queues. Let $X_i(t)$ be the number of customers in queue i at time t, and let $X(t) = (X_1(t), ..., X_M(t))$. Hence let $\{X(t), t \geq 0\}$ be a process on the state space $\mathcal{S}$ for a queuing system of the kind described in Section 2.2. Let the arrival time of the kth general arrival to the system be τ_k. Upon arrival they immediately make a decision, C_k, of which queue to join. The decision $C_k \in \{Q_0, Q_1, ..., Q_M\}$ where $C_k = Q_i$ for $i \in \{1, ..., M\}$ represents the event that the user chooses to join queue i and $C_k = Q_0$ is the event that they choose to balk instead of joining any of the queues. Balking is discussed briefly below. Now we define the probability, $D_{j,k}(\mathbf{n}) = \mathbb{P}[C_k = Q_j | X(\tau_k) = \mathbf{n}]$. So for $j \in \{1, ..., M\}$ the probability $D_{j,k}(\mathbf{n})$ is the probability that the kth general arrival chooses to join queue j given that the process is in state $\mathbf{n}$ at the moment of their arrival, while the probability $D_{0,k}(\mathbf{n})$ is the probability that the kth general arrival decides to balk under the same circumstance. Since the only options open to any general arrival to the system is to either join one of the queues or balk we have

$$\sum_{j=0}^{M} D_{j,k}(\mathbf{n}) = 1, \forall\ \mathbf{n} \in \mathcal{S},\ k \in \mathbb{N}.$$

Now, we can collect all of the $D_{j,k}(\mathbf{n})$, $0 \leq j \leq M$, into the set

$$D_k(\mathbf{n}) = \{D_{0,k}(\mathbf{n}), D_{1,k}(\mathbf{n}), ..., D_{M,k}(\mathbf{n})\}.$$

We call this set the decision rule of state $\mathbf{n}$ for the kth arrival. In Section 2.2 it was stated that we assume that all the users in the system use the same set of decision rules. We refer to this as *symmetric decision rules*. Then we can write that $D_{i,k}(\mathbf{n}) = D_i(\mathbf{n}), \forall \mathbf{n} \in \mathcal{S}, i \in \{0, 1, ..., M\}$. Thus the index referring to the identity of the general arrival will be dropped from decision rules from now on.

In the following there are situations when decision rules are compared based on other criteria. For example it will be useful to be able to compare the decision rules for the same state $\mathbf{n}$ under different conditions in which case we add subscripts to the notation as needed. So, for example, when we are interested in what the decision rule is for some specific state $\mathbf{n}$ under two different sets of system parameters, $\mathbf{\Gamma}_k$ and $\mathbf{\Gamma}_l$, we work with decision rules $D_{\mathbf{\Gamma}_k}(\mathbf{n})$ and $D_{\mathbf{\Gamma}_l}(\mathbf{n})$. To reduce notational complexity we always endeavour to use the least amount of subscripts possible, without causing confusion, in every particular situation. Unfortunately this parsimony does not fully remedy the fact that notation occasionally becomes quite unwieldy.

This general definition of decision rules always holds so $D_j(\mathbf{n}) \in [0, 1]$, $\forall \mathbf{n} \in \mathcal{S}, j \in \{0, 1, ..., M\}$. However, the whole interval $[0, 1]$ will not always be accessible for decision rules. For example it is natural to study pure policies so that only $D_j(\mathbf{n}) \in \{0, 1\}$ are actually available. Conversely we will also deal with totally mixed strategies where $D_i(\mathbf{n}) \in (0, 1), \forall i \in \mathcal{M}$.

Now to the second important concept, the decision policy. The decision policy is the collection of decision rules for all states of the system. We denote the full decision policy, the probability of each decision in each possible state of the system, by $\mathbf{D} = \{D(\mathbf{n}) : \mathbf{n} \in \mathcal{S}\}$. It is often interesting, or sufficient, to look at the decision rules pertaining to either a specific queue, or to balking. When it comes to the probability of a general arrival making decision C_i we consider the marginal decision policies $\mathbf{D}_i = \{D_i(\mathbf{n}) : \mathbf{n} \in \mathcal{S}\}$. It is sometimes useful to talk about the set of all possible decision policies $\mathcal{D}$, so that $\mathbf{D} \in \mathcal{D}$. We also define the set of all marginal decision policies for decision C_i, $\mathcal{D}_i$.

We are particularly interested in policies that represent user-optima for the general arrivals under some conditions such as specific parameter vectors, update rules or starting policies. A user optimal policy is a Nash-equilibrium in terms of the cost of participating in the system for an individual user. Explicitly, this means that a user optimal policy is a decision policy from which no general arrival can unilaterally deviate in order to improve their own situation. In general improvement means lowering their expected waiting time in the system, but we also

utilise slightly more involved cost functions.

As we use a variety of iterative approaches to arrive at equilibrium policies it is frequently necessary to clearly state the iterative step that has given rise to a specific decision policy, therefore we also often use a "time" superscript on the policies thus; $\mathbf{D}^t$. This should never be taken to be a time as measured on a clock, but always refers to the generation of the iterative scheme to which the policy belongs.

In the special case, which we usually deal with, when $M = 2$ it is useful to use matrix representation for the decision policies. In the case of limited capacity we have a set of $(N_1 + 1) \times (N_2 + 1)$-matrices, while in the case of potentially infinite queues the matrices are infinite in size. In this case we follow standard matrix notation so that the marginal decision policy for queue 1 in a system with $N_1 = N_2 = 3$ has the following matrix representation

$$\mathbf{D}_1 = \begin{bmatrix} D_1(0,0) & D_1(0,1) & D_1(0,2) & D_1(0,3) \\ D_1(1,0) & D_1(1,1) & D_1(1,2) & D_1(1,3) \\ D_1(2,0) & D_1(2,1) & D_1(2,2) & D_1(2,3) \\ D_1(3,0) & D_1(3,1) & D_1(3,2) & D_1(3,3) \end{bmatrix} \tag{2.1}$$

In any system of M queues it has to be the case for each state, $\mathbf{n}$, that $\sum_{i=0}^{M} D_i(\mathbf{n}) = 1$. This means that in general we only need to know M of the $M + 1$ marginal policies in order to have full knowledge of the policy as a whole. For the special case $M = 2$ this means that using matrix representation we have

$$\mathbf{D}_0 + \mathbf{D}_1 + \mathbf{D}_2 = \mathbf{J} \tag{2.2}$$

where we introduce the notation $\mathbf{J}$ for a matrix consisting of only ones.

This becomes particularly relevant in the cases, which we deal with frequently, when the probability of balking is always zero as long as there is another choice. In these cases

$$\mathbf{D}_0 + \mathbf{D}_1 + \mathbf{D}_2 = \mathbf{D}_1 + \mathbf{D}_2 \tag{2.3}$$

so that

$$\mathbf{D}_j = \mathbf{J} - \mathbf{D}_i. \tag{2.4}$$

In this way we can, in these special but frequently occurring cases, fully represent the decision policy by a single matrix. In these cases we, as a convention, represent the full decision policy of the system by the marginal policy $\mathbf{D}_1$; the marginal decision policy pertaining to queue 1.

Note here that, as alluded to above, the situation when $\mathbf{D}_0 = \mathbf{0}$ is never fully realised in a finite queuing system, as a general arrival who finds the system in state (N_1, N_2) will always balk with probability 1. This is a forced move, though, and one that we can always take for granted, so in these cases we can think of it as a property of the system rather than a property of the decision policy.

2.3.2 Expected waiting times

The expected waiting time for a marked customer is an important quantity in much of the work presented here. Partially it is important as the basis for considering the relative merit of various decision policies, and partially, as in Chapter 3, we discuss the expected waiting time explicitly. Thus we state a formal definition of, as well as an expression for, the quantity here.

Let $\tau^A_{i,k,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n})$ be the arrival time of the kth general arrival who chooses to join queue i, $c_{k,i}$. Note that the argument $\mathbf{n}$ is the state of the system immediately before the arrival of the marked user $c_{k,i}$. This means that with $\mathbf{n} = (n_1, ..., n_M)$ there will be a total of $1 + \sum_{j=1}^{M} n_j$ users in the system, and after $c_{k,i}$ joins queue i there will be $n_i + 1$ users in that queue. The user, $c_{k,i}$ uses decision policy $\mathbf{D}$. Let $\tau^S_{i,k,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n})$ be the service completion time for the user $c_{k,i}$. We now use the difference between the arrival and the service time to define this user's sojourn time in the system, $\tau_{i,k,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n}) = \tau^S_{i,k,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n}) - \tau^A_{i,k,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n})$. We note that the fact that the process governing occupancy in the system is a Markov process means that the specific time of arrival doesn't affect the distribution of transitions that occur after the instant of arrival. So that for two such customers $c_{k,i}$ and $c_{l,i}$ we see that $\tau_{i,k,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n}) \sim \tau_{i,l,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n})$. Therefore we can simply drop the subscript that refers to the specific customer under consideration and let $\tau_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n})$ be the sojourn time of a user who joins queue i when the system which operates under parameters $\mathbf{\Gamma}$ is in state $\mathbf{n}$ and all general arrivals employ decision policy $\mathbf{D}$. We are now free to take the expectation of this well defined random variable, which we state as

Definition 2.3.1. With $\tau_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n})$ defined as above $z_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n}) = \mathbb{E}[\tau_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n})]$

We find the expected waiting time, $z_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n})$ as in Definition 2.3.1, by first step analysis, which requires the solution of two systems of equations, one per queue, defined by the expression given in Equation 2.5.

$$z_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n}) =$$
$$\Big(\lambda \sum_{j\in\mathcal{M}} D_j(\mathbf{n}+\mathbf{e}_i) + \sum_{j\in\mathcal{M}} \big(\mu_j I_{\{n_j>0\}} + \sigma_j I_{\{n_j+I_{\{j=i\}}<N_j\}}\big)\Big)^{-1}$$
$$\times \Big(1 + \sum_{j\in\mathcal{M}} I_{\{n_j+I_{\{j=i\}}<N_j\}}[\lambda D_j(\mathbf{n}+\mathbf{e}_i) + \sigma_j] z_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n}+\mathbf{e}_j)$$
$$+ \mu_i \frac{n_i}{n_i+1} z_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n}-\mathbf{e}_i) + \sum_{j\in\mathcal{M}\setminus i} \mu_j I_{\{n_j>0\}} z_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n}-\mathbf{e}_j)\Big),$$
$$i \in \mathcal{M}, \mathbf{n} \in \mathcal{S} \quad (2.5)$$

This expression is largely self-explanatory, but some aspects, or choices, in the formulation warrant closer inspection. The main thing to keep in mind when looking at this expression is that it is the expected waiting time of a marked customer who joins queue i upon finding the system with $n_1 + n_2$ *other* customers already in the system, distributed as $\mathbf{n}$.

This accounts for the asymmetry in the indicator functions associated with the rates with which the process goes to states with increased occupancy in either queue. The next event after the marked customers arrival in the process can only be an arrival to queue i as long as $n_i < N_i - 1$, as this means that, including the marked customer, the queue is full. Meanwhile it is perfectly fine for the next event in the process to be an arrival to queue j as long as $n_j < N_j$.

For the same reason the decision rule that is used in determining which queue a general arrival will join is $D_i(\mathbf{n} + \mathbf{e}_i)$, the decision rule for the state that the system is in after the marked customer has joined it.

2.3.3 User equilibrium policy

We now consider how each individual user arrives at the decision policy that they employ in the system based on the expected waiting times, and how this relates to the *User Equilibria* that we are particularly concerned with. This relationship depends crucially on the expected cost for joining the system in each state. However, rather than consider a general cost, we carry out this description in terms of expected waiting times.

Since the general arrivals have full knowledge of the system, including the decision policies of the other general arrivals they can calculate the expected waiting time that they would face joining each available queue. It is frequently useful to collect the expected waiting times for each state of the system into a set $\mathbf{Z} = \{z_i(\mathbf{n}) : \mathbf{n} \in \mathbf{S}, i \in \mathcal{M}\}$. In the following it is also often useful to explicitly state the decision policy employed by the general arrivals that give rise to a particular set of expected waiting times. In these cases we add the decision policy in question as a subscript so that the expected waiting times that arise from general arrivals using the decision policy $\mathbf{D}$ is denoted by $\mathbf{Z_D}$.

In order to move from expected waiting times to decision policies the general arrivals apply a routing rule, $\mathbf{f}$, where

$$\mathbf{f} : \mathbb{R}^{M \prod_{i \in \mathcal{M}} (N_i+1)} \to [0,1]^{M \prod_{i \in \mathcal{M}} (N_i+1)}.$$

The most obvious routing rule is the one employed in Chen [45] where the users join the queue with the lowest expected waiting time with probability 1. It is important to note, though, that even this simple routing rule has non-trivial components. In particular a non-trivial choice has to be made about how users deal with the case when for some state the expected waiting time is equal in several queues. Two possible choices are to let the users in this case have a hierarchy of preferences with regards to the queues so that in the situation of equal expected waiting times one queue is always joined with probability 1, or to let the users join each of the queues with equal expected waiting times with equal probabilities. In Chen [45] a generalisation of these alternatives is used, so that in this situation, with two queues, a tie breaker probability $\tilde{p}$ is employed so that a user faced with a choice between two queues with equal expected waiting times join queue 1 with probability $\tilde{p}$ and the other queue with probability $1 - \tilde{p}$. Here, in particular in Chapter 4, we consider other routing rules, which all crucially share the property that a user always has a higher probability of joining a queue with a lower expected waiting time than one with a higher expected waiting time.

A special kind of policy is the situation when the policy is such that under the routing rule employed by the general arrivals no user deviates from the policy. This is an example of a Nash equilibrium and we refer to them, depending on context, as *User Equilibrium Policies*, *User Optimal Policies* or *Fixed Point Policies*. We denote user equilibrium policies by $\mathbf{D}^*$. The user equilibrium that corresponds to a particular set of parameters depends on the routing rule employed, but as it is

always unambiguous in the following which routing rule is employed to arrive at a user optimal policy we do not explicitly include this in the notation.

In the case of a given routing rule $\mathbf{f}$ a user equilibrium policy $\mathbf{D}$ is defined as in Definition 2.3.2.

Definition 2.3.2. $\mathbf{D}^*$ is a user equilibrium policy iff

$$\mathbf{f}(\mathbf{Z}_{\mathbf{D}^*}) = \mathbf{D}^*.$$

2.3.4 Decision policy monotonicity

The concept of *monotonic policies* is important for large parts of this thesis. For that reason we will define what this means as well as to some extent motivate the importance of this particular type of decision policy here. We start by defining the set $\mathcal{D}^M$, the set of all monotonic policies.

Definition 2.3.3. Let $\mathbf{D} \in \mathcal{D}$ then,

$$\mathbf{D} \in \mathcal{D}^M \Leftrightarrow \begin{cases} D_i(\mathbf{n} + \mathbf{e}_i) \leq D_i(\mathbf{n}) \\ D_i(\mathbf{n} + \mathbf{e}_j) \geq D_i(\mathbf{n}) \end{cases}, \ \forall i \in \mathcal{M}, j \in \mathcal{M} \setminus i, \mathbf{n} \in \mathcal{S}$$

Here, as throughout the following, $\mathbf{e}_i$ represents a unit base vector in the direction i, a vector that has value 0 in all places except the ith where it has value 1. For example, with $M = 2$, when $\mathbf{n} = (n_1, n_2)$ the two basis vectors are $\mathbf{e}_1 = (1, 0)$ and $\mathbf{e}_2 = (0, 1)$.

Definition 2.3.3 states that a decision policy is monotonic if and only if it is such that the probability of a marked customer joining queue i is non-increasing in n_i, the occupancy of queue i, and non-decreasing in n_j for any $j \neq i$, the occupancy of any queue other than i. Since this is a central concept in the following we give some examples.

In the context of pure decision rules a monotonic policy is equivalent to a set of *threshold policies*. To describe a threshold policy we limit ourselves to $M = 2$ to reduce complexity, but it is trivial to extend it to more dimensions. We then express the set of decision rules with regards to queue 1, $D_1(\mathbf{n})$, for a set occupancy of queue 2, n_2, as a vector-valued function $\mathbf{r}_{1,2}^{n_2} = (D_1(0, n_2), D_1(1, n_2), ...)$ where the first index of the function describes which queue's decision rule the vector contains, the second index describes which occupancy is fixed, and the superscript represents the occupancy of the queue with fixed occupancy. In other words $\mathbf{r}_{i,j}^{n_j}$ is

row, or column, n_j+1 for the marginal decision policy D_i, and whether it is a row or a column depends on the choice of j – if $j=1$ it is a row, otherwise a column. So $\mathbf{r}_{i,j}^{n_j}(n_k) = D_i(\mathbf{n})$ where $\mathbf{n} = (n_j, n_k)$ or $\mathbf{n} = (n_k, n_j)$ depending on the values of i and j. We can now use the $\mathbf{r}_{i,j}^{n_j}$s to define threshold policies as in Definition 2.3.4.

Definition 2.3.4. With $M = 2$, $\mathbf{D}$ is a threshold policy if for all $i, j \in \{1,2\}$ $\exists\, r_{i,j}^{n_j} \in \{0, ..., N_i\}$ s.t

$$\mathbf{r}_{i,j}^{n_j}(k) = \begin{cases} \mathbf{r}_{i,j}^{n_j}(0) & k \leq r_{i,j}^{n_j} \\ 1 - \mathbf{r}_{i,j}^{n_j}(0) & k > r_{i,j}^{n_j} \end{cases} .$$

This means that to be part of a threshold policy the vector that the $\mathbf{r}_{i,j}^{n_j}$ function returns has to have the form $\mathbf{r}_{1,2}^{n_2} = (1, 1, ...1, 0, 0, ...)$ which means that $\mathbf{r}_{2,2}^{n_2} = (0, 0, ...0, 1, 1, ...)$. So, as the name implies, there is a *threshold occupancy*, $r_{i,j}^{n_j}$ in Definition 2.3.4, of queue 1 at which the choice switches from queue 1 to queue 2. Clearly both $\mathbf{r}_{1,2}^{n_2} = (1, 1, ..., 1)$ and $\mathbf{r}_{1,2}^{n_2} = (0, 0, ..., 0)$ also qualify as columns in threshold policies. In these cases we can simply view it as the threshold being "higher than N_1" and "lower than 0" respectively. The cases that do not qualify as threshold policies are when the choice switches, and then switches back, for example $\mathbf{r}_{1,2}^{n_2} = (1, 1, ..., 1, 0, ..., 1, ...)$.

Obviously it would be possible to construct something that looks like a threshold policy, but would not conform to the definition of monotonicity given in Definition 2.3.3, to wit $\mathbf{r}_{1,2}^{n_2} = (0, 0, ..., 0, 1, 1...)$. We simply state that the monotonicity, and thus the threshold policies, that we are interested in is in some sense the "natural" monotonicity where the increased occupancy of a queue makes the queue less appealing. The type of threshold policy we are interested in could be viewed as monotonic threshold policies by Definition 2.3.3, but we simply refer to them as threshold policies. Clearly threshold policies are the natural state of affairs for many intuitive service disciplines. As an example, the case of optimal pure policies in the context of two M/M/1−FIFO queues with equal service rates in parallel. In this case we know that the optimal policy with respect to expected waiting times for joining queue 1 with occupancy n_2 of queue 2 is a threshold policy. We also know that the state in which the probability of joining queue 1 goes from 1 to 0 is when $n_1 = n_2$, or potentially when $n_1 = n_2 + 1$, depending on how equal expected waiting times are handled.

So for pure policies a decision policy is monotonic if, and only if, all one dimensional projections analogous to the function $\mathbf{r}_{i,j}(\mathbf{n}_k)$ with appropriate dimensionality, are threshold policies.

We give a few examples, and counter examples, of what this means in terms of pure decision policies.

EXAMPLE 2.3.1. $N_1 = N_2 = 3$.

$$\mathbf{D}_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{D}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}, \quad \mathbf{D}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example 2.3.1 represents a straightforward monotonic decision policy. As this is the first time we write out decision policies explicitly we have given the policy for balking as well as joining the two queues, for completeness. As discussed above, the probability of balking in state (N_1, N_2) is a forced move, and does not have to be viewed as a decision. From now on we will not be giving the balking policy except in the few special cases when it is actually relevant. Unless otherwise stated the balking policy for finite systems has exactly the form $D_0(N_1, N_2) = 1$ and $D_0(n_1, n_2) = 0$ otherwise, as in Example 2.3.1. In the case of infinite state systems we have $\mathbf{D}_0 = \mathbf{0}$ unless otherwise stated.

These policies clearly conform to Definition 2.3.3 as the probability of joining queue i never increases as n_i increases or decreases as n_j increases. In fact, as these are pure policies we can actually state that $D_i(\mathbf{n}) - D_i(\mathbf{n} + \mathbf{e}_i) \in \{0, 1\}$, while in a non-monotonic policy we would instead find that there exists some i and $\mathbf{n}$ such that $D_i(\mathbf{n}) - D_i(\mathbf{n} + \mathbf{e}_i) = -1$.

This example also makes the connection between monotonicity and threshold policies clear. We note that every row and column in each of the three matrices in Example 2.3.1 satisfies the threshold condition.

We now give two examples of non-monotonic pure policies. Again we let $M = 2$ and $N_1 = N_2 = 3$ First

EXAMPLE 2.3.2. $\mathbf{D}_1 = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{D}_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}.$

Let us look at this from the perspective of a marked customer's probability of joining queue 1 when queue 1 is empty. The marked customer will chose to

join queue 1 with probability one both when the occupancy of queue 2 is $n_2 = 1$ and $n_2 = 3$, but when $n_2 = 2$ the marked customer will instead choose to join queue 2 with probability 1. So here we have an example of the choice switching, and then switching back. This means that $D_1(0,2) - D_1(1,2) = -1$, and also $D_2(0,1) - D_2(0,2) = -1$, both of which clearly violate definition 2.3.3. As stated above, this obviously means that $D_i(\mathbf{n}) - D_i(\mathbf{n}+\mathbf{e}_i) \in \{-1,0,1\}$. We see here that the first row, and third column in each of the matrices do not in fact correspond to threshold policies.

We also, for completeness, give this example, where the capacity in both queues is infinite,

EXAMPLE 2.3.3. $\mathbf{D}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & \dots \\ 1 & 0 & 0 & 0 & \\ 1 & 1 & 0 & 0 & \\ 1 & 1 & 1 & 0 & \\ \vdots & & & & \ddots \end{bmatrix}$, $\mathbf{D}_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots \\ 0 & 1 & 1 & 1 & \\ 0 & 0 & 1 & 1 & \\ 0 & 0 & 0 & 0 & \\ \vdots & & & & \ddots \end{bmatrix}$

which one could be forgiven for mistaking for a monotonic policy at first glance. However, this is in fact the very opposite of the behaviour proscribed for monotonicity in Definition 2.3.3. By this we mean that instead of $D_i(\mathbf{n}) - D_i(\mathbf{n}+\mathbf{e}_i) \in \{0,1\}$ we have $D_i(\mathbf{n}) - D_i(\mathbf{n}+\mathbf{e}_i) \in \{-1,0\}$. The probability of a marked customer choosing to join queue i is here non-decreasing in n_i. This illustrates the point touched upon above; each row and column does indeed have threshold structure, but not in the correct direction. So, once again, threshold structure is a necessary condition for policy monotonicity, but it is not a sufficient one.

The concept of monotonicity is most clearly described in terms of pure policies, as above, but as this is a central concept throughout the rest we give examples in terms of mixed policies as well. First an example of a monotonic decision policy.

EXAMPLE 2.3.4. $\mathbf{D}_1 = \begin{bmatrix} 0.5 & 0.6 & 0.7 & 1 \\ 0.4 & 0.5 & 0.6 & 1 \\ 0.3 & 0.4 & 0.5 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, $\mathbf{D}_2 = \begin{bmatrix} 0.5 & 0.4 & 0.3 & 0 \\ 0.6 & 0.5 & 0.4 & 0 \\ 0.7 & 0.6 & 0.5 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$.

This policy is analogous to the one in Example 2.3.1 so we don't need to dwell on it too much. We point out a few properties, though. We note that, as required, $D_i(\mathbf{n}) - D_i(\mathbf{n}+\mathbf{e}_i) \in [0,1]$. We also note that the concept of threshold policies

does not have an have an analogue in the mixed case in general. Only when the probability of joining either queue is in the set $\{0, 1\}$ for some state do we see the threshold structure in this case. Since this is a special case we can not rely on threshold structure to distinguish monotonic from non-monotonic policies in the mixed case. So the concept of threshold structure is not important for mixed policies in the following.

So why do we care about monotonicity? There are at least three answers to this question, we can call them intuition, tractability and numerics. We discuss intuition in some depth here while the numerics and tractability answers are merely touched upon and elucidated further later on, particularly in Chapters 4 and 5 respectively.

So, first we look at why we intuitively would like equilibrium policies to be monotonic. We start this discussion by noting again that the effective service rate for a single customer, $\mu^e(n)$, in a processor sharing queue is inversely proportional to the occupancy of the queue. If we study a single processor sharing queue in isolation we find that the expected waiting time for a customer joining the queue in state n is non-decreasing in n. This is not quite as obvious in the PS case as it is in the FIFO case, but is well known. See for example Berg [63] or Kleinrock [64]. We can also intuitively see that this is the case by considering the fact that a user who joins a processor sharing queue in which there are already n users in the queue would expect to spend a longer time in the system than a user who joins when there are n' with $n' < n$. This is because the user joining in the case of n other users in the system simply has to compete with more, or at least as many, users for the attention of the server for their entire stay in the system. Here we essentially apply the reasoning presented in Example 1.1.2.

Thus we know that in a processor sharing queue in isolation the expected waiting time is non-decreasing in occupancy. We might expect this to hold also in the case when the queue is embedded in a system of other PS-queues in parallel with the queue under consideration. However, it is clear that the situation changes in some ways when the choice of later arrivals comes into play. As examples of non-monotonic policies have previously been observed, as discussed in Chen [45], and reproduced and extended upon in Chapter 4 in the present text, we are challenged to understand this counter-intuitive behaviour.

However, as previously stated, the intuitive nature of monotonicity is not the only, or even necessarily the dominant, reason why we focus a lot of attention

on studying this property. Tractability is in some sense as important in terms of motivating this focus, and has two components, one intrinsic and one emergent.

The intrinsic reason can be viewed as slightly outside of the scope of this research but, we think, still merits consideration. We consider the situation when the customers in the system are simply humans, rather than computers or computer aided humans. In this context we can easily imagine that the customers might make decisions about which queue to join by considering the service rate, occupancy and the rate of other arrivals in the system, and it is easy to imagine that they might arrive at some type of monotonic policy. Particularly, perhaps, a threshold type policy. As a concrete example; it is much harder to imagine them adhering to a non-monotonic policy, where their choice would be queue 2 when there are 6 people in queue 1, but queue 1 when the only thing that has changed is that there are now 7 people in queue 1.

This is obviously not put forth as an argument for why we would expect fixed point policies to be monotonic, quite the contrary. Rather, since we are studying a system that demonstrably allows for non-monotonic fixed point policies under certain conditions this consideration motivates us to find out what parameters in the system have the potential to lead to non-monotonic equilibrium policies. So this part of the motivation for focusing on monotonicity has more to do in the end with system design than it has to do with discovery. To wit, working out the conditions that need to be placed on a system of parallel processor sharing queues in order to ensure that the optimal policy is one that humans can arrive at, or at least approximate, is a worthwhile endeavour from a system design perspective.

The emergent reason has to do with the tractability of proofs of other system properties, and is discussed in detail in Chapter 5. We also find in Chapter 3 that many seemingly obvious results can be obtained by straightforward methods under the assumption that the general arrivals are operating under a monotonic decision policy, but not otherwise.

The reason based on numerics is discussed in detail in the Chapter 4. Suffice it to say here that numeric exploration makes it reasonable to conjecture that fixed point policies under certain, reasonable, system conditions are indeed guaranteed to be monotonic.

3

Monotonicity results

> *'Oh, obvious,' said Granny. 'I'll grant you it's obvious. Trouble is, just because things are obvious doesn't mean they're true.'*
>
> – Terry Pratchett, Wyrd Sisters

In this Chapter we present a number of results related to system properties in the context of systems of the kind defined in the previous chapters.

The Theorems, all of which are proven by coupling argument under the assumption of decision policy monotonicity, concern two different properties of the system; the expected waiting time and the stationary joining probability, defined in Sections 2.3.2 and 3.2 respectively. All of the results are monotonicity results, making statements about the monotonic relationship between the changes in some system parameter and the system properties of interest.

One overarching observation concerning these results relates to the considerations mentioned in 2.3.4, namely the emergent reason for policy monotonicity. It is generally easy to see in the following proofs why they would not work if the assumption of decision policy monotonicity was not in place. While this is not an indication that the monotonicity results in question would necessarily fail to hold had the decision policies been non-monotonic, it is interesting to consider that the lack of decision policy monotonicity disallows certain, seemingly self-evident, arguments about system behaviour.

We also note that while we were able to prove a variety of monotonicity results concerning expected waiting times with the full state description presented in Chapter 2, and in the case of M queues, when it came to the results related to asymmetrical increases in expected waiting times with respect to which queue we added users to we had to not only restrict ourselves to 2 queues but also put

other, harsher, constraints on the system. We discuss this further at the end of this chapter.

In general we are considering M queues in parallel and general arrivals always join the system when there is room in some queue. Theorems 3.1.1, 3.1.2, 3.1.3 and 3.1.4 relate to the situation when M is arbitrary while Theorems 3.1.5, 3.2.1, 3.2.2 and 3.3.1 are only applicable when $M = 2$.

The results presented here are also stated to hold under two different general conditions on system parameters. Theorems 3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.1.5, 3.2.1 and 3.2.2 are all stated in the context of limited capacity system, but otherwise for arbitrary choices of parameters and monotonic decision policies. Corollary 3.3.1 states that each of these Theorems hold also in the context of infinite capacity queues, as long as the system parameters and decision policies are chosen in such a way that the system is stable.

3.1 Expected waiting time monotonicities

We start by considering the monotonicity results related to expected waiting times.

3.1.1 Monotonicity of expected waiting time in occupancy

We have already touched upon the idea that it is reasonable to expect that the expected waiting time of a marked customer in queue i is non-decreasing in occupancy n_i, even when the queue is embedded in a system of parallel queues, and one set of arrivals makes a state based choice of which queue to join upon arrival to the system. Beyond that it also seems reasonable that the expected waiting time would be non-decreasing in occupancy of the other queue in the system, as there is only so much service to go around per unit time. Theorem 3.1.1 tells us that this is indeed the case.

Theorem 3.1.1. Let $z_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n})$, be the expected waiting time as defined in Section 2.3.2, Γ a parameter vector as defined in Section 2.2 with $N_i \in \mathbb{N}, \forall i \in \mathcal{M}$, and $\mathbf{D}$ a monotonic policy. Then:

$$z_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n}) \leq z_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n}+\mathbf{e}_j), \forall \mathbf{n} \in \mathcal{S} \text{ s.t } \mathbf{n}+\mathbf{e}_j \in \mathcal{S}, \text{ for } i,j \in \mathcal{M}.$$

Proof. We prove this theorem using a stochastic coupling argument. Due to the arbitrary naming of the queues we can, without loss of generality, write the proof in

the case when $i = 1$ in Theorem 3.1.1. This means that Q_1 contains an added user not explicitly shown in the state description; the marked customer. Furthermore, as there is only one set of system parameters, and one decision policy in play we drop the subscripts $\mathbf{D}$ and $\mathbf{\Gamma}$ from the expected waiting times, so that in this proof we write $z_i(\mathbf{n}) = z_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n})$.

We start by defining the joint process $\mathbf{X} = \{(X^{(1)}(t), X^{(2)}(t)), t \geq 0\}$ on $(\mathcal{S} \times \mathcal{S}) \cup \{g_1, g_2, g_S\}$. $X^{(1)}$ and $X^{(2)}$ both follow the law of X, as defined in Section 2.1, and are coupled in such a way that they make the same transitions whenever possible. The coupling is, crucially, also such that the joint process never transitions to g_2 at a positive rate. We define the states g_1, g_2 and g_S below. The marginal processes $X^{(1)}$ and $X^{(2)}$ are processes, on $\mathcal{S}$, in which a marked customer has joined queue 1. We let $X^{(1)}(0) = \mathbf{n}$ and $X^{(2)}(0) = \mathbf{n} + \mathbf{e}_j$.

The states g_1, g_2 and g_S are absorbing states of the joint process and we let the process stop immediately when it hits one of the absorbing states. g_1 is the state in which the marked customer has been served in the $X^{(1)}$-process, but not in the $X^{(2)}$-process, g_2 is the state in which the marked customer has been served in the $X^{(2)}$-process but not in the $X^{(1)}$-process and g_S is the state in which the marked customer has been served in both of the marginal processes. We also define the sets $G_i = \{g_i, g_S\}$ to be the states in which the marked customer in the X_i-process has been served. Now, let $T_i = \inf\{t \colon \mathbf{X}(t) \in G_i \,|\mathbf{X}(0)\}$ be the hitting times of the joint process to the sets G_1 and G_2. We let τ be the time when the process hits an absorbing state, so that $\tau = \min(T_1, T_2)$. If the joint process hits g_1 then $T_1 = \tau$ and $T_2 = \infty$ so $T_1 < T_2$. If instead it hits the absorbing state g_2 then $T_2 = \tau$ and $T_1 = \infty$ so $T_2 < T_1$ and, finally, if the absorbing state is g_S then $T_1 = T_2 = \tau$. This means that if $\mathbb{P}[\mathbf{X}(t) = g_2 | \mathbf{X}(0) = (\mathbf{n}, \mathbf{n} + \mathbf{e}_j)] = 0,\ \forall t > 0$ then $T_1 \leq T_2$ a.s., which implies that $z_1(\mathbf{n}) \leq z_1(\mathbf{n} + \mathbf{e}_j)$.

Note that the states discussed and listed below are, as above, what the marked customer observes. So that the total number of customers in Q_1 after the marked customer has joined the system is in fact $n_1 + 1$. This means that, if the first event after the arrival of the marked customer is a general arrival, the rule that is applied in the $X^{(1)}$-process is $D(\mathbf{n} + \mathbf{e}_1)$ and the decision rule in the $X^{(2)}$-process is $D(\mathbf{n} + \mathbf{e}_j + \mathbf{e}_1)$. In the context of decision policies in the following we often use $\mathbf{n}' = \mathbf{n} + \mathbf{e}_1$ in order to improve legibility.

We start by examining the transitions with $j = 1$ and when $\mathbf{n} + 2\mathbf{e}_1 \in \mathcal{S}^-$, the situation when the joint process is in a state in which there is room left in all queues. Then the transitions shown in Display 3.1 occur at positive rate (remembering

that the marked customer in Q_1 is not included in the state description). The abbreviation *a.r.* in Display 3.1 as well as elsewhere means *at rate*.

$$(\mathbf{n}, \mathbf{n}+\mathbf{e}_1) \to \begin{cases} g_S & \text{a.r.} \\ \quad \mu_1 \frac{1}{n_1+2} & \\ g_1 & \text{a.r.} \\ \quad \mu_1(\frac{1}{n_1+1} - \frac{1}{n_1+2}) & \\ (\mathbf{n}-\mathbf{e}_1, \mathbf{n}) & \text{a.r.} \\ \quad \mu_1(1 - \frac{1}{n_1+1}) & \\ (\mathbf{n}-\mathbf{e}_j I_{\{n_j>0\}}, \mathbf{n}+\mathbf{e}_1-\mathbf{e}_j I_{\{n_j>0\}}) & \text{a.r.} \\ \quad \mu_j, 1<j\le M & \\ (\mathbf{n}+\mathbf{e}_1, \mathbf{n}+2\mathbf{e}_1) & \text{a.r.} \\ \quad \sigma_1 + \lambda D_1(\mathbf{n}'+\mathbf{e}_1) & \\ (\mathbf{n}+\mathbf{e}_j, \mathbf{n}+\mathbf{e}_1+\mathbf{e}_j) & \text{a.r.} \\ \quad \big(\sigma_j + \lambda D_j(\mathbf{n}')\big), 1<j\le M & \\ (\mathbf{n}+\mathbf{e}_1, \mathbf{n}+\mathbf{e}_1+\mathbf{e}_j) & \text{a.r.} \\ \quad \lambda\big(D_j(\mathbf{n}'+\mathbf{e}_1) - D_j(\mathbf{n}')\big), 1<j\le M & \end{cases} \tag{3.1}$$

The first three transitions are the possible outcomes when the next event is a service completion in Q_1.

The first transition is the one to the graveyard state g_S, the event that the marked customer is served in both $X^{(1)}$ and $X^{(2)}$. The probability that the marked customer receives the service when a service completion occurs in Q_1 is lower in state $\mathbf{n}+\mathbf{e}_1$ than in $\mathbf{n}$ the rate of this transition is given by the former rate, and is $\frac{\mu_1}{n_1+2}$.

The second transition is the one to the graveyard state g_1, a service completion of the marked customer in $X^{(1)}$ but not in $X^{(2)}$. The rate of the transition to this graveyard state is the difference between the rate at which the marked customer is

served in $X^{(1)}$ and the rate at which the marked customer is served in $X^{(2)}$. Thus the rate is $\mu_1(\frac{1}{n_1+1} - \frac{1}{n_1+2})$.

The third transition is the one that occurs when there is a service completion in Q_1, but someone other than the marked customer is served in both marginal processes. Note that in the situation when the marked customer is the only customer in Q_1 in $X^{(1)}$ we have $n_1 = 0$ so that the rate of this transition is also 0. This is the first transition that changes the state of the joint process without being a graveyard state. However, with $\hat{\mathbf{n}} = \mathbf{n} - \mathbf{e}_1$ we can write the resulting state as $(\hat{\mathbf{n}}, \hat{\mathbf{n}} + \mathbf{e}_1)$, which has the same *structure* as $(\mathbf{n}, \mathbf{n} + \mathbf{e}_1)$.

The next transition represents a service in Q_j when $j \neq 1$. The occupancy of Q_j is equal in both marginal processes. So either $n_j = 0$ in which case the transition is a null transition, and the resulting state of the joint process is still $(\mathbf{n}, \mathbf{n} + \mathbf{e}_1)$. Otherwise the state of the joint process after the transition is $(\mathbf{n} - \mathbf{e}_j, \mathbf{n} + \mathbf{e}_1 - \mathbf{e}_j)$, which preserves the structure of the state.

The three remaining entries in Display 3.1 are the transitions that occur when the event is an arrival to the system.

First of these we have the transition that is a result of a customer joining Q_1 in both processes. Since we are considering the situation when $\mathbf{n} + 2\mathbf{e}_1 \in \mathcal{S}^-$ this transition is always possible. While this transition does preserve the structure of the state of the joint process we can not be sure that the resulting state of $X^{(2)}$ is in $\mathcal{S}^-$. We discuss this situation below. The fact that the coupling is such that the marginal processes make the same transition whenever possible gives the rate of this transition too, and since $D_1(\mathbf{n}') \geq D_1(\mathbf{n}' + \mathbf{e}_1)$ by monotonicity, the rate is given by $\sigma_1 + \lambda D_1(\mathbf{n}' + \mathbf{e}_1)$.

The next entry in Display 3.1 is a set of transitions, each of which is very similar to the previous one. These are the transition that occurs when a customer joins Q_j, $j \neq 1$, in both processes. While these transitions do preserve the structure of the state of the joint process they can result in states in $\mathcal{S}^e$ for both joint processes, which will be discussed below. By monotonicity we have that $D_j(\mathbf{n}') \leq D_j(\mathbf{n}' + \mathbf{e}_1)$ for $j \neq 1$, so now the rate is given by the rate at which customers join Q_j in $X^{(1)}$, to wit $\sigma_j + \lambda D_j(\mathbf{n}')$.

The final set of transitions are the ones that occur as a result of a general arrival joining Q_1 in $X^{(1)}$ and Q_j, $j \neq 1$, in $X^{(2)}$. This transition occurs because by monotonicity the rate at which general arrivals joins Q_1 in state $\mathbf{n}$ is greater than or equal to the rate at which a customer joins Q_1 in state $\mathbf{n} + \mathbf{e}_1$ while the rate at which general arrivals join Q_j for any $j \neq 1$ in state $\mathbf{n} + \mathbf{e}_1$ is greater than

or equal to the rate at which this occurs in state $\mathbf{n}$. So $D_j(\mathbf{n}') \leq D_j(\mathbf{n}' + \mathbf{e}_1)$ for all $j \neq 1$, while $D_1(\mathbf{n}') \geq D_1(\mathbf{n}' + \mathbf{e}_1)$, so that $D_1(\mathbf{n}') - D_1(\mathbf{n}' + \mathbf{e}_1) \geq 0$ and since $\sum_{j=1}^{M} D_j(\mathbf{n}) = 1$ for all $\mathbf{n} \in \mathcal{S}^-$ we have

$$\sum_{j=2}^{M} \big(D_j(\mathbf{n}' + \mathbf{e}_1) - D_j(\mathbf{n}')\big) = D_1(\mathbf{n}') - D_1(\mathbf{n}' + \mathbf{e}_1). \tag{3.2}$$

We check that the transition rates due to arrivals sum to $\lambda + \sum_{j=1}^{M} \sigma_j$. The total rate of the last three entries in Display 3.1 is

$$\sigma_1 + \lambda D_1(\mathbf{n}' + \mathbf{e}_1) + \sum_{j=2}^{M} \Big(\sigma_j + \lambda\big(D_j(\mathbf{n}') + D_j(\mathbf{n}' + \mathbf{e}_1) - D_j(\mathbf{n}')\big)\Big) = \sum_{j=1}^{M} \big(\sigma_j + \lambda D_j(\mathbf{n}' + \mathbf{e}_1)\big) = \lambda + \sum_{j=1}^{M} \sigma_j.$$

Here the last equality following since $\sum_{j=1}^{M} D_j(\mathbf{n} + \mathbf{e}_1) = 1$.

We now look at what changes when we allow the starting states for $X^{(1)}$ and $X^{(2)}$ to be in $\mathcal{S}^e$, the set of states in which at least one queue is full. We first note that none of the transitions related to services, the first four entries in Display 3.1 change by this generalisation of the initial state. Any service that could occur when starting from a state in $\mathcal{S}^-$ can still occur and any event that leads to a null transition when starting from a state in $\mathcal{S}^-$ still does so when starting from a state in $\mathcal{S}^e$. The rates of events also remain the same. Thus we focus on the remaining three entries in the table.

We write the potential transitions resulting from arrivals in this, more general, state.

$$(\mathbf{n}, \mathbf{n} + \mathbf{e}_1) \to \begin{cases} (\mathbf{n} + \mathbf{e}_1, \mathbf{n} + (1 + I_{\{n_1+2<N_1\}})\mathbf{e}_1) & \text{a.r.} \\ \quad \sigma_1 + \lambda\big(D_1(\mathbf{n}' + \mathbf{e}_1) + D_0(\mathbf{n}' + \mathbf{e}_1)\big) & \\ (\mathbf{n} + \mathbf{e}_j I_{\{n_j<N_j\}}, \mathbf{n} + \mathbf{e}_1 + \mathbf{e}_j I_{\{n_j<N_j\}}) & \text{a.r.} \\ \quad \sigma_j + \lambda D_j(\mathbf{n}'), 1 < j \leq M & \\ (\mathbf{n} + \mathbf{e}_1, \mathbf{n} + \mathbf{e}_1 + \mathbf{e}_j) & \text{a.r.} \\ \quad \lambda\big(D_j(\mathbf{n}' + \mathbf{e}_1) - D_j(\mathbf{n}')\big), 1 < j \leq M & \end{cases} \tag{3.3}$$

We start by examining the set of possible transitions resulting from an arrival to Queue 1 in both processes. We first note that in a state of the form $(\mathbf{n}, \mathbf{n}+\mathbf{e}_1)$ a customer will always be able to join Q_1 in $X^{(1)}$; if nothing else the space taken up by the extra customer in $X^{(2)}$ is available. However, Q_1 might be full in $X^{(2)}$ in which case the customer is turned away, and the arrival is a null arrival. So when all queues are full in $X^{(2)}$, so that $D_0(\mathbf{n}' + \mathbf{e}_1) = 1$, there is still exactly one spot left in $X^{(1)}$, and that spot is in Q_1 so that $D_1(\mathbf{n}') = 1$. We thus focus on the case when $n_1 + 2 = N_1$, the case when Q_1 is full in $X^{(2)}$. If there is room in other queues in $X^{(2)}$ this transition only occurs as a result of an intrinsic arrival and the transition is a coupling event. Otherwise the entire system is full in $X^{(2)}$ in which case either a general or intrinsic arrival can lead to this transition. In this situation the transition fills up the system in $X^{(1)}$ as well, again leading to a coupling event. So regardless of the occupancy of the other queues, starting from a state in which Q_1 is full in $X^{(2)}$ this transition is a coupling event implying that $T_1 = T_2$.

The next set of transitions in this list are the ones that result from a customer joining Q_j, $j \neq 1$ in both of the marginal processes. The only difference between these transitions when $\mathbf{n}, \mathbf{n}' \in \mathcal{S}^-$ initially and when we also allow $\mathbf{n}, \mathbf{n}' \in \mathcal{S}^e$ initially is the case when $n_j = N_j$, in which case $D_j(\mathbf{n}) = D_j(\mathbf{n}') = 0$, so that no general arrival will chose to join Q_j in either process. An intrinsic arrival to Q_j can still occur, but in this case it is a null event, which doesn't change the state of the process at all.

The final set of transitions do not change at all, either in outcome or in the rate at which they occur, as a result of allowing $\mathbf{X}(0) \in \mathcal{S}^e$. The reason is that this transition occurs as long as there is room in some queue in $X^{(2)}$ and that it only affects queues with room left. When all queues are full in $X^{(2)}$ all of these transition occur at rate 0. Thus they still occur at a total rate

$$\sum_{j=2}^{M} \lambda\big(D_j(\mathbf{n}' + \mathbf{e}_1) - D_j(\mathbf{n}')\big).$$

This gives us a total rate of events related to arrivals of

$$\lambda \sum_{j=0}^{M} D_j(\mathbf{n}' + \mathbf{e}_1) + \sum_{j=1}^{M} \sigma_j.$$

We finally note that when $\mathbf{X}(0) \in \mathcal{S}$ rather than $\mathbf{X}(0) \in \mathcal{S}^-$ we need to include the possibility of balking in order to ensure that the decision rules sum to unity, so now $\sum_{j=1}^{M} D_j(\mathbf{n}' + \mathbf{e}_1) \leq 1$ while $\sum_{j=0}^{M} D_j(\mathbf{n}' + \mathbf{e}_1) = 1$. As we indeed have the latter sum we can rewrite the expression as

$$\lambda + \sum_{j=1}^{M} \sigma_j.$$

So we have seen that starting from a state of the form $(\mathbf{n}, \mathbf{n} + \mathbf{e}_1)$ a transition might leave the joint process in a state of the same structure, a state of the form $(\mathbf{n}, \mathbf{n} + \mathbf{e}_j)$ with $j \in \mathcal{M}$ or a coupling event occurs. Regardless of where in $\mathcal{S}$ the joint process starts it never transitions from a state of the form $(\mathbf{n}, \mathbf{n} + \mathbf{e}_1)$ to the graveyard state g_2 at positive rate. Now it remains to show that this is still true when the joint process is in a state $(\mathbf{n}, \mathbf{n} + \mathbf{e}_j)$ for arbitrary j and anywhere in $\mathcal{S}$.

We consider the case when the joint process is in a state of the form $(\mathbf{n}, \mathbf{n} + \mathbf{e}_j)$ for arbitrary j such that $\mathbf{n} + \mathbf{e}_1 + \mathbf{e}_j \in \mathcal{S}$. We start by considering the transitions that occur as a result of service completions.

$$(\mathbf{n}, \mathbf{n} + \mathbf{e}_j) \to \begin{cases} g_S & \text{a.r.} \\ \quad \mu_1 \frac{1}{n_1 + 1 + I_{\{j=1\}}} & \\ g_1 & \text{a.r.} \\ \quad \mu_1 (\frac{1}{n_1+1} - \frac{1}{n_1+1+I_{\{j=1\}}}) & \\ (\mathbf{n} - \mathbf{e}_1, \mathbf{n} + \mathbf{e}_j - \mathbf{e}_1) & \text{a.r.} \\ \quad \mu_1 (1 - \frac{1}{n_1+1}) & \\ (\mathbf{n} - \mathbf{e}_k I_{\{n_k > 0\}}, \mathbf{n} + \mathbf{e}_j - \mathbf{e}_k I_{\{n_k + I_{\{k=j\}} > 0\}}) & \text{a.r.} \\ \quad \mu_k, 1 < k \leq M & \end{cases} \tag{3.4}$$

The first small difference occurs in the transitions to the graveyard states g_S and g_1 where the rate changes when $j \neq 1$. We note that without the extra customer in Q_1 in $X^{(2)}$ the probability that the marked customer receives the service completion is identical in the marginal processes, so the rate at which the joint process transitions to state g_1 is zero.

We also see that the joint process does not transition to g_2 at positive rate. Finally, taking the services in Q_1, except for the marked customer, into account

we conclude that these transition either take the joint process to a graveyard state in G_1, or a state with structure $(\mathbf{n}, \mathbf{n} + \mathbf{e}_j)$.

Now, the next set of transitions differ somewhat in potential outcome. In the event that $n_k > 0$, or when $k \neq j$ these transitions are either service completions in both marginal processes, or null events in both processes, in either case preserving the structure of the state of the joint process. In the case when $k = j$, when the service completion occurs in the queue with the added customer in $X^{(2)}$, and $n_j = 0$ so that Q_j is empty in $X^{(1)}$, however, this transition is a null event in $X^{(1)}$ but a service of the added customer in $X^{(2)}$. This is a coupling event so that $T_1 = T_2$.

We then consider the remaining transitions, the ones that occur as a result of arrivals to the system.

$$(\mathbf{n}, \mathbf{n} + \mathbf{e}_j) \to \begin{cases} (\mathbf{n} + \mathbf{e}_j, \mathbf{n} + (1 + I_{\{n_j+1<N_j\}})\mathbf{e}_j) & \text{a.r.} \\ \quad \sigma_j + \lambda\big(D_j(\mathbf{n}' + \mathbf{e}_j) + D_0(\mathbf{n}' + \mathbf{e}_j)\big) & \\ (\mathbf{n} + \mathbf{e}_k I_{\{n_k<N_k\}}, \mathbf{n} + \mathbf{e}_j + \mathbf{e}_k I_{\{n_k<N_k\}}) & \text{a.r.} \\ \quad \big(\sigma_k + \lambda D_k(\mathbf{n}')\big), k \in \mathcal{M} \setminus j & \\ (\mathbf{n} + \mathbf{e}_j, \mathbf{n} + \mathbf{e}_j + \mathbf{e}_k) & \text{a.r.} \\ \quad \lambda\big(D_k(\mathbf{n}' + \mathbf{e}_j) - D_k(\mathbf{n}')\big), k \in \mathcal{M} \setminus j & \end{cases} \tag{3.5}$$

When the added customer goes from being in Q_1 to being in Q_j for arbitrary j these transitions only change inasmuch as the label of the queues under consideration, but we go through them briefly regardless.

The first transition is the transition resulting from an arrival seeking to join Q_j in both processes, which is equivalent to an arrival seeking to join Q_1 in both processes when the joint process is in a state of the form $(\mathbf{n}, \mathbf{n} + \mathbf{e}_1)$. We recognise the transition rate as well as the outcome of the transition from that case.

The fact that $j = 1$ had no impact on either the outcome or the rate of the second transition in Display 3.3, the case when a customer seeks to join one of the queues with equal occupancy in the marginal processes. So the rates and outcomes of the same situation with general j, the second transition in Display 3.5, needs no further explanation.

The final set of transitions are the ones that occur as a result of a general arrival deciding to join Q_j in $X^{(1)}$, as that queue is shorter in $X^{(1)}$ than it is in $X^{(2)}$, while

a simultaneous general arrival occurs in $X^{(2)}$ but in that process instead chooses to join Q_k. This was the only transition in the case with the process in a state of the form $(\mathbf{n}, \mathbf{n}+\mathbf{e}_1)$ that resulted in a state of a different structure, but was not a coupling event. In fact it resulted in a state of form $(\mathbf{n}, \mathbf{n}+\mathbf{e}_j)$, with $j \in \mathcal{M} \setminus 1$. Now we let the joint process start from a state of that structure but with $j \in \mathcal{M}$, and we see that now this transition preserves the structure of the joint process.

The total rate of this set of transitions can be stated as

$$\lambda \sum_{k \in \mathcal{M} \setminus j} \big(D_k(\mathbf{n}' + \mathbf{e}_j) - D_k(\mathbf{n}')\big) = \lambda\big(D_j(\mathbf{n}') - D_j(\mathbf{n}' + \mathbf{e}_j)\big).$$

So we can state that the total event rate for the joint process, and thus each of the marginal processes, is

$$\lambda + \sum_{k=1}^{M} \sigma_k + \sum_{k=1}^{M} \mu_k$$

as required.

Thus we have explored all the transitions that occur at positive rate with this coupling starting from state $(\mathbf{n}, \mathbf{n}+\mathbf{e}_j)$ with $j \in \mathcal{M}$. An arrival at time t either induces a transition to a state of the same structure so that $\mathbf{X}(t) = (\mathbf{n}, \mathbf{n}+\mathbf{e}_k)$ for some $k \in \mathcal{M}$, or it is a coupling event so that $\mathbf{X}(t) = (\mathbf{n}, \mathbf{n})$ which implies that $X^{(1)}(t') = X^{(2)}(t'), \forall t' \geq t$. In addition to inducing transitions to states with the same structure and coupling events, services also induce the joint process to transitions to graveyard states. When the joint process is in a state of the form $(\mathbf{n}, \mathbf{n}+\mathbf{e}_1)$ the joint process transitions at positive rate to both g_1, so that $T_1 < T_2$, and g_S, in which case $T_1 = T_2$. When the joint process is instead in a state of the form $(\mathbf{n}, \mathbf{n}+\mathbf{e}_j)$ with $j \neq 1$, or if a coupling event has occurred so that the joint process is in state $(\mathbf{n}, \mathbf{n})$, the only graveyard state the joint process transitions to at positive rate is g_S, so that $T_1 = T_2$. As these are the only states the joint process visits with positive probability starting from a state $(\mathbf{n}, \mathbf{n}+\mathbf{e}_j)$ we can conclude that this coupling is such that $T_1 \leq T_2$ a.s., which implies that $z_1(\mathbf{n}) \leq z_1(\mathbf{n}+\mathbf{e}_j)$. □

3.1.2 Monotonicity of expected waiting time in λ

Just as it is reasonable to expect that an increase in the occupancy will lead to an increase in expected waiting time for a marked customer, it is reasonable to expect

that an increased rate of general arrivals would never decrease the expected waiting time. Theorem 3.1.2 states that expected waiting times, $z_{i,\mathbf{D},\mathbf{\Gamma}}(\mathbf{n})$ are indeed non-decreasing in λ.

Theorem 3.1.2. Let $\mathbf{\Gamma}_l = (\mu_1, ..., \mu_M, \sigma_1, ..., \sigma_M, \lambda_l, N_1, ..., N_M)$ and $\mathbf{\Gamma}_h = (\mu_1, ..., \mu_M, \sigma_1, ..., \sigma_M, \lambda_h, N_1, ..., N_M)$, where $\lambda_l \leq \lambda_h$, and $N_i \in \mathbb{N}, \forall i \in \mathcal{M}$, be parameter vectors of two processes as described above. Let the general arrivals in both processes follow the same monotonic policy $\mathbf{D}$.
Then

$$z_{k,\mathbf{D},\Gamma_l}(\mathbf{n}) \leq z_{k,\mathbf{D},\Gamma_h}(\mathbf{n} + \sum_{i=1}^{M} a_i \mathbf{e}_i) \text{ for } a_i \in \mathbb{N}_0 \text{ and } k \in \mathcal{M}.$$

Proof. This proof largely follows the same structure and steps as the proof of Theorem 3.1.1. We again work with the joint process $\mathbf{X} = \{(X^{(1)}(t), X^{(2)}(t)), t \geq 0\}$ on $(\mathcal{S} \times \mathcal{S}) \cup \{g_1, g_2, g_S\}$ where the state space is defined as in the previous proof but the marginal processes $X^{(1)}$ and $X^{(2)}$ differ subtly. The main differences stem from the fact that the marginal processes now experience general arrivals at different rates. In particular $X^{(1)}$ has general arrival rate λ_l while $X^{(2)}$ has general arrival rate λ_h with $\lambda_l \leq \lambda_h$. The processes are again coupled in such a way that they make the same transition whenever possible, which with these differing general arrival rates means that $X^{(2)}$ will experience a general arrival whenever $X^{(1)}$ does. In addition $X^{(2)}$ will, at rate $\lambda_h - \lambda_l$, experience a general arrival while no event takes place in $X^{(1)}$.

We also let the joint process start in a more general state than in the previous proof in that the marginal process $X^{(2)}$ contains an arbitrary number of added customers as compared to $X^{(1)}$, distributed in an arbitrary way. We let $\mathbf{a} = (a_1, ..., a_M)$ with $a_i \in \mathbb{N}_0$ for $i \in \mathcal{M}$ so that the joint process is in a state of the form $(\mathbf{n}, \mathbf{n} + \mathbf{a})$. As above the proof proceeds by demonstrating that there is a coupling, as described above, such that both marginal processes follow the required laws, and with which the joint process never transitions to the graveyard state g_2 at positive rate. This is largely done by showing that the joint process does not transition to g_2 at positive rate from the initial state while all transitions that do occur preserve the structure of the state of the joint process. This means that the resulting state of the transition is also of the form $(\mathbf{n}, \mathbf{n} + \mathbf{a})$ with $a_i \in \mathbb{N}_0$ for all $i \in \mathcal{M}$ for some $\mathbf{n}$ and $\mathbf{a}$ such that $n_i + a_i < N_i, \ \forall i \in \mathcal{M}$.

Just as in the previous proof we assume without loss of generality that the marked customer is in Q_1, and the states of the joint processes are the states as

perceived by the marked customer so that a general arrival to the state that is given as $\mathbf{n}$ will use the decision rule $D(\mathbf{n}+\mathbf{e}_1) = D(\mathbf{n}')$.

One important difference between the proof of Theorem 3.1.1 and the current proof is that coupling events, situations when $X^{(1)}(t) = X^{(2)}(t)$, are of no real relevance in this proof. In the proof of Theorem 3.1.1 $X^{(1)}(t) = X^{(2)}(t) \Rightarrow X^{(1)}(s) = X^{(2)}(s), s > t$, which in turn implied that $T_1 = T_2$. This is not the case in the process employed in this proof since the arrival rates differ.

We start by examining the transitions that occur as a result of service completions.

$$(\mathbf{n}, \mathbf{n}+\mathbf{a}) \to \begin{cases} g_1 & \text{a.r.} \\ \quad \mu_1(\frac{1}{n_1+1} - \frac{1}{n_1+1+a_1}) & \\ g_S & \text{a.r.} \\ \quad \mu_1 \frac{1}{n_1+1+a_1} & \\ (\mathbf{n}-\mathbf{e}_1, \mathbf{n}+\mathbf{a}-\mathbf{e}_1) & \text{a.r.} \\ \quad \mu_1(1 - \frac{1}{n_1+1}) & \\ (\mathbf{n}-\mathbf{e}_j I_{\{n_j>0\}}, \mathbf{n}+\mathbf{a}-\mathbf{e}_j I_{\{n_j+a_j>0\}}) & \text{a.r.} \\ \quad \mu_j, 1 < j \leq M & \end{cases} \tag{3.6}$$

These transitions are almost identical to those in Display 3.4. The fact that there might be several customers in the $X^{(2)}$ process that are not present in the $X^{(1)}$ process leads to a few subtle differences, though.

We first note that in the first two transitions, the transitions to the graveyard states g_S and g_1 the rate is a function of a_1, the number of added customers in Q_1 in the $X^{(2)}$-process. This is because when a service completion occurs in Q_1 at time t the probability that the completed service goes to the marked customer in process $X^{(k)}$ is $\frac{1}{X_1^{(k)}(t)+1}$, and the number of customers in Q_1 in the $X^{(2)}$-process is $n_1 + a_1$.

Here we see why we require $a_i \geq 0$ particularly in the case of $i = 1$, as the fact that $a_1 \geq 0$ is what guarantees that the joint process does not transition to g_2 at positive rate with this coupling, thus guaranteeing that $T_1 \leq T_2$.

Next we see that a service in Q_j that does not induce a transition of the joint process to a graveyard state causes a change of state which preserves the structure

of the initial state when $n_j > 0$ or $a_j \geq 1$, and it is a null event when $n_j = 0$ and $a_j = 0$.

It is trivial to extend the discussion about the rates of the relevant transitions in the proof of Theorem 3.1.1 to this case, and to come to the conclusion that the total rate of these transitions in the joint process, as well as the rate of events induced by service completions in each of the marginal processes, is

$$\sum_{j=1}^{M} \mu_j. \tag{3.7}$$

We then turn to the transitions that occur as a result of arrivals to the system. In Display 3.8, and the following, $\big(D_j(\mathbf{n}') - D_j(\mathbf{n}'+\mathbf{a})\big)^+ = \max\{D_j(\mathbf{n}') - D_j(\mathbf{n}'+\mathbf{a}), 0\}$.

$$(\mathbf{n}, \mathbf{n}+\mathbf{a}) \rightarrow \begin{cases} (\mathbf{n} + \mathbf{e}_j I_{\{n_j + I_{\{j=1\}} < N_j\}}, \mathbf{n} + \mathbf{a} + I_{\{n_j + I_{\{j=1\}} + a_j < N_j\}} \mathbf{e}_j) & \text{a.r.} \\ \quad \sigma_j + \lambda_l \big(\min\{D_j(\mathbf{n}'), D_j(\mathbf{n}'+\mathbf{a})\} & \\ \qquad\qquad + D_j(\mathbf{n}') D_0(\mathbf{n}'+\mathbf{a}) \big), 1 \leq j \leq M & \\ (\mathbf{n} + \mathbf{e}_j, \mathbf{n} + \mathbf{a} + \mathbf{e}_k) & \text{a.r.} \\ \quad \lambda_l \big(D_j(\mathbf{n}') - D_j(\mathbf{n}'+\mathbf{a}) \big)^+ & \\ \quad \times \Big(\dfrac{\big(D_k(\mathbf{n}'+\mathbf{a}) - D_k(\mathbf{n}')\big)^+}{\sum_{l=1}^{M} \big(D_l(\mathbf{n}'+\mathbf{a}) - D_l(\mathbf{n}')\big)^+} \Big), 1 \leq j \leq M & \\ (\mathbf{n}, \mathbf{n} + \mathbf{a} + \mathbf{e}_j), 1 \leq j \leq M & \text{a.r.} \\ \quad (\lambda_h - \lambda_l) D_j(\mathbf{n}'+\mathbf{a}) & \end{cases} \tag{3.8}$$

Note that the transitions in Display 3.8 are the transitions of the joint process; the rates given do not necessarily apply to both marginal processes. We untangle this below.

We start by showing that all the transitions occur at the correct rates, and result in states of the form $(\mathbf{n}, \mathbf{n}+\mathbf{a})$ with all $a_i \geq 0$ guaranteeing that service completions do indeed result in the transitions given in Display 3.6 so that $T_1 \leq T_2$.

The first set of transitions are the ones that result from a customer seeking to join Queue j in at least one of the processes. When this transition is the result

of an intrinsic arrival, and queue j is full in both marginal processes it is a null arrival in both processes. When $n_i + a_i + I_{\{i=1\}} = N_i, \forall i \in \mathcal{M}$ and at least one of the $a_i > 0$ this transition is an arrival to queue j in the $X^{(1)}$-process and a null event in the $X^{(2)}$-process. Finally, when $n_i + a_i + I_{\{i=1\}} < N_i$ for some i, and the transition is due to either a general arrival or an intrinsic arrival to queue Q_i, this transition is an arrival in both processes. Regardless, this transition preserves the structure of the state of the joint process.

We can write the total rate of these transitions as

$$\sum_{j=1}^{M} \sigma_j + \lambda_l \Big(\sum_{j=1}^{M} \min\{D_j(\mathbf{n}'), D_j(\mathbf{n}' + \mathbf{a})\} + \sum_{j=1}^{M} D_j(\mathbf{n}') D_0(\mathbf{n}' + \mathbf{a}) \Big). \tag{3.9}$$

The next set of transitions are the ones in which a general arrival occurs in both of the marginal processes, and the arriving customer chooses to join Q_j in $X^{(1)}$ while they join Q_k in $X^{(2)}$. This transition is the same sort of event as the third transition in Display 3.5, but now, as there are potentially more differences between the two processes, the transition is a lot more complex.

The main thing to note, and in many ways the crux of this proof, is that this transition can never make Q_j in $X^{(1)}$ longer than the corresponding queue in $X^{(2)}$ which would mean that $a_j < 0$ and thus change the structure of the joint process. At most this transition can equalise occupancy in the two marginal processes. To see this we start by noting that the only way that the joint process could end up in a state where $a_j < 0$ is to start in a state where $a_j = 0$ and have a customer join Q_j in the $X^{(1)}$-process but not in the $X^{(2)}$-process. With this coupling the marginal processes make the same transitions whenever possible, so at time t a general arrival can only join Q_j in the $X^{(1)}$-process without also joining Q_j in $X^{(2)}$-process when $D_j(X^{(1)}(t)) \geq D_j(X^{(2)}(t))$. This means that when $a_j = 0$ we have two cases; either $a_i = 0$ for all other $j \neq i$ too, or $a_i > 0$ for some j. In the first case, when $a_j = 0$ for all j then $D_i(X^{(1)}(t)) = D_i(X^{(2)}(t))$ for all i, including $i = j$, so a general arrival will always join the same queue in both processes. For the case when $a_i > 0$ for some i we remind ourselves that decision policy monotonicity gives that $D_i(\mathbf{n}) \leq D_i(\mathbf{n} + \mathbf{e}_j)$ for $j \neq i$, which means that with $a_j = 0$ we have $D_j(\mathbf{n}) \leq D_j(\mathbf{n} + \mathbf{a})$ so that a user might join Q_j in the $X^{(2)}$ process and not the $X^{(1)}$-process, but not the other way around. Thus these transitions do preserve the structure of the state of the joint process.

Since these transitions affect different queues in the marginal processes we have to sum over the set of queues twice in order to find the total rate of these transitions. We start by writing down the total rate as

$$\sum_{j=1}^{M} \sum_{k \in \mathcal{M} \setminus j} \lambda_l \big(D_j(\mathbf{n}') - D_j(\mathbf{n}' + \mathbf{a})\big)^+ \frac{\big(D_k(\mathbf{n}' + \mathbf{a}) - D_k(\mathbf{n}')\big)^+}{\sum_{l=1}^{M} \big(D_l(\mathbf{n}' + \mathbf{a}) - D_l(\mathbf{n}')\big)^+}$$

which we rewrite as

$$\lambda_l \sum_{j=1}^{M} \big(D_j(\mathbf{n}') - D_j(\mathbf{n}' + \mathbf{a})\big)^+ \frac{\sum_{k \in \mathcal{M} \setminus j} \big(D_k(\mathbf{n}' + \mathbf{a}) - D_k(\mathbf{n}')\big)^+}{\sum_{l=1}^{M} \big(D_l(\mathbf{n}' + \mathbf{a}) - D_l(\mathbf{n}')\big)^+}. \tag{3.10}$$

We then consider the fraction in that expression in isolation

$$\frac{\sum_{k \in \mathcal{M} \setminus j} \big(D_k(\mathbf{n}' + \mathbf{a}) - D_k(\mathbf{n}')\big)^+}{\sum_{l=1}^{M} \big(D_l(\mathbf{n}' + \mathbf{a}) - D_l(\mathbf{n}')\big)^+}$$

and note that we can rewrite it as

$$\frac{\sum_{k=1}^{M} \big(D_k(\mathbf{n}' + \mathbf{a}) - D_k(\mathbf{n}')\big)^+}{\sum_{l=1}^{M} \big(D_l(\mathbf{n}' + \mathbf{a}) - D_l(\mathbf{n}')\big)^+} - \frac{\big(D_j(\mathbf{n}' + \mathbf{a}) - D_j(\mathbf{n}')\big)^+}{\sum_{l=1}^{M} \big(D_l(\mathbf{n}' + \mathbf{a} - D_l(\mathbf{n}')\big)^+}. \tag{3.11}$$

where we in particular consider the second part,

$$\frac{\big(D_j(\mathbf{n}' + \mathbf{a}) - D_j(\mathbf{n}')\big)^+}{\sum_{l=1}^{M} \big(D_l(\mathbf{n}' + \mathbf{a}) - D_l(\mathbf{n}')\big)^+} \tag{3.12}$$

which we note is equal to 0 except when $D_j(\mathbf{n}' + \mathbf{a}) - D_j(\mathbf{n}') > 0$. However, when we look back at the full expression in Equation 3.10 we note that this whole expression is equal to 0 except when $D_j(\mathbf{n}') - D_j(\mathbf{n}' + \mathbf{a}) > 0$. In other words, the expression in Equation 3.12 is only non-zero when the whole summand in Equation 3.10 is zero. So for all cases when the summand is non-zero we can write Equation 3.11 as

$$\frac{\sum_{k=1}^{M} \big(D_k(\mathbf{n}' + \mathbf{a}) - D_k(\mathbf{n}')\big)^+}{\sum_{l=1}^{M} \big(D_l(\mathbf{n}' + \mathbf{a}) - D_l(\mathbf{n}')\big)^+}. \tag{3.13}$$

Now we note that the expression in Equation 3.13 is equal to 1, except in the situation when $D_k(\mathbf{n}' + \mathbf{a}) = 0$ for all k. The only situation that this can occur in

is when all queues are full in $X^{(2)}$. While this leads to Equation 3.13 taking the form $\frac{0}{0}$, we easily see that this outcome should be interpreted as 0, due to the fact that this is in effect a decision rule for joining one of the queues. This situation only occurs when all queues in $X^{(2)}$ are full, so that $D_0(\mathbf{n}' + \mathbf{a}) = 1$. Therefore we can write

$$\frac{\sum_{k=1}^{M} \left(D_k(\mathbf{n}' + \mathbf{a}) - D_k(\mathbf{n}')\right)^+}{\sum_{l=1}^{M} \left(D_l(\mathbf{n}' + \mathbf{a}) - D_l(\mathbf{n}')\right)^+} = 1 - D_0(\mathbf{n}' + \mathbf{a}).$$

Thus we can now write Equation 3.10 as

$$\lambda_l \sum_{j=1}^{M} \left(D_j(\mathbf{n}') - D_j(\mathbf{n}' + \mathbf{a})\right)^+ \left(1 - D_0(\mathbf{n}' + \mathbf{a})\right) \tag{3.14}$$

which is the total rate of this set of transitions.

The third set of transitions in Display 3.8 are general arrivals to the $X^{(2)}$-process and are not events at all in the marginal process $X^{(1)}$. These transitions trivially preserve the structure of the state of the joint process. These transitions occur at a total rate of

$$(\lambda_h - \lambda_l) \sum_{j=1}^{M} D_j(\mathbf{n}' + \mathbf{a}) \tag{3.15}$$

In addition to the transitions stated in Display 3.8 the joint process also experiences global null-events due to balking general arrivals. This occurs when a general arrival, either in both marginal processes, or just in the $X^{(2)}$-process arrives to find that the system they seek to join is full so that they have to balk. The rate of this null-transition is given by the facts that there is an arrival in $X^{(2)}$ whenever there is an arrival in $X^{(1)}$, but general arrivals in $X^{(2)}$ without a simultaneous general arrival in $X^{(1)}$ occurs at rate $\lambda_h - \lambda_l$, and the fact that $D_0(\mathbf{n}') = 1 \Rightarrow D_0(\mathbf{n}' + \mathbf{a}) = 1$. Thus the total rate of this null-event is given by

$$\lambda_l D_0(\mathbf{n}') + (\lambda_h - \lambda_l) D_0(\mathbf{n}' + \mathbf{a}) \tag{3.16}$$

We now find the total rate of transitions induced by arrivals to the system by summing up the rates given in Equations 3.9, 3.14, 3.15, and 3.16

$$\sum_{j=1}^{M}\sigma_j + \lambda_l\Big(\sum_{j=1}^{M}\min\{D_j(\mathbf{n}'), D_j(\mathbf{n}'+\mathbf{a})\} + \sum_{j=1}^{M} D_j(\mathbf{n}')D_0(\mathbf{n}'+\mathbf{a})\Big) + \lambda_l\sum_{j=1}^{M}\big(D_j(\mathbf{n}') - D_j(\mathbf{n}'+\mathbf{a})\big)^+\big(1 - D_0(\mathbf{n}'+\mathbf{a})\big) + (\lambda_h - \lambda_l)\sum_{j=1}^{M} D_j(\mathbf{n}'+\mathbf{a}) + \lambda_l D_0(\mathbf{n}') + (\lambda_h - \lambda_l)D_0(\mathbf{n}'+\mathbf{a}) \quad (3.17)$$

which we can rewrite as

$$\sum_{j=1}^{M}\sigma_j + (\lambda_h - \lambda_l)\big(D_0(\mathbf{n}'+\mathbf{a}) + \sum_{j=1}^{M} D_j(\mathbf{n}'+\mathbf{a})\big) + \lambda_l\big(D_0(\mathbf{n}') + \sum_{j=1}^{M}\big[\min\{D_j(\mathbf{n}'), D_j(\mathbf{n}'+\mathbf{a})\} + D_j(\mathbf{n}')D_0(\mathbf{n}'+\mathbf{a}) + \big(D_j(\mathbf{n}') - D_j(\mathbf{n}'+\mathbf{a})\big)^+\big(1 - D_0(\mathbf{n}'+\mathbf{a})\big)\big]\big) \quad (3.18)$$

To understand this expression we start by focusing on the factor that is multiplied by λ_l

$$D_0(\mathbf{n}') + \sum_{j=1}^{M}\big[\min\{D_j(\mathbf{n}'), D_j(\mathbf{n}'+\mathbf{a})\} + D_j(\mathbf{n}')D_0(\mathbf{n}'+\mathbf{a}) + \big(D_j(\mathbf{n}') - D_j(\mathbf{n}'+\mathbf{a})\big)^+\big(1 - D_0(\mathbf{n}'+\mathbf{a})\big)\big] \quad (3.19)$$

which we rewrite as

$$D_0(\mathbf{n}') + \sum_{j=1}^{M}\big[\min\{D_j(\mathbf{n}'), D_j(\mathbf{n}'+\mathbf{a})\} + \big(D_j(\mathbf{n}') - D_j(\mathbf{n}'+\mathbf{a})\big)^+ + D_0(\mathbf{n}'+\mathbf{a})\big(D_j(\mathbf{n}') - \big(D_j(\mathbf{n}') - D_j(\mathbf{n}'+\mathbf{a})\big)^+\big)\big] \quad (3.20)$$

We note that

$$D_0(\mathbf{n}'+\mathbf{a})\big(D_j(\mathbf{n}')-\big(D_j(\mathbf{n}')-D_j(\mathbf{n}'+\mathbf{a})\big)^+\big)=0$$

and rewrite the expression in Equation 3.20 as

$$D_0(\mathbf{n}')+\sum_{j=1}^{M}\big[\min\{D_j(\mathbf{n}'),D_j(\mathbf{n}'+\mathbf{a})\}+\big(D_j(\mathbf{n}')-D_j(\mathbf{n}'+\mathbf{a})\big)^+\big] \tag{3.21}$$

If j is such that $D_j(\mathbf{n}')\leq D_j(\mathbf{n}'+\mathbf{a})$, then

$$\min\{D_j(\mathbf{n}'),D_j(\mathbf{n}'+\mathbf{a})\}+\big(D_j(\mathbf{n}')-D_j(\mathbf{n}'+\mathbf{a})\big)^+=D_j(\mathbf{n}')$$

while in the complementary situation, when $D_j(\mathbf{n}')>D_j(\mathbf{n}'+\mathbf{a})$, we have

$$\min\{D_j(\mathbf{n}'),D_j(\mathbf{n}'+\mathbf{a})\}+\big(D_j(\mathbf{n}')-D_j(\mathbf{n}'+\mathbf{a})\big)^+= D_j(\mathbf{n}'+\mathbf{a})+D_j(\mathbf{n}')-D_j(\mathbf{n}'+\mathbf{a})=D_j(\mathbf{n}').$$

Thus we can rewrite Equation 3.19 as

$$D_0(\mathbf{n}')+\sum_{j=1}^{M}D_j(\mathbf{n}')=\sum_{j=0}^{M}D_j(\mathbf{n}')=1$$

and Equation 3.18 as

$$\sum_{j=1}^{M}\sigma_j+\lambda_l+(\lambda_h-\lambda_l)\big(D_0(\mathbf{n}'+\mathbf{a})+\sum_{j=1}^{M}D_j(\mathbf{n}'+\mathbf{a})\big) \tag{3.22}$$

where we see that

$$D_0(\mathbf{n}'+\mathbf{a})+\sum_{j=1}^{M}D_j(\mathbf{n}'+\mathbf{a})=\sum_{j=0}^{M}D_j(\mathbf{n}'+\mathbf{a})=1$$

so that Equation 3.22, the total event rate induced by arrivals to the system, can be written

$$\sum_{j=1}^{M}\sigma_j+\lambda_l+(\lambda_h-\lambda_l)=\lambda_h+\sum_{j=1}^{M}\sigma_j \tag{3.23}$$

giving a total event rate for the joint process as well as the marginal $X^{(2)}$-process of

$$\lambda_h + \sum_{j=1}^{M} \sigma_j + \sum_{j=1}^{M} \mu_j$$

and a total event rate of

$$\lambda_l + \sum_{j=1}^{M} \sigma_j + \sum_{j=1}^{M} \mu_j$$

in the $X^{(1)}$-process, as required.

Thus, we have seen that the marginal processes have the required properties. Together with the same argument as in the conclusion of the proof of Theorem 3.1.1, which states that this process is such that $T_1 \leq T_2$, this proves that $z_{k,\mathbf{D},\Gamma_l}(\mathbf{n}) \leq z_{k,\mathbf{D},\Gamma_h}(\mathbf{n} + \sum_{i=1}^{M} a_i \mathbf{e}_i)$ for $a_i \in \mathbb{N}_0$, as required.

□

3.1.3 Monotonicity of expected waiting time in σ_i

We can use the same reasoning that was briefly employed in relation to Theorem 3.1.2 to develop the intuition that an increase in intrinsic arrival rate should never lead to a decrease in expected waiting time. That is to say, it is reasonable to expect that any increase in intrinsic arrival rate will lead to increased occupancy in the system as a whole. We then know from 3.1.1 that expected waiting times are non-decreasing in occupancy. The fact that expected waiting times are non-decreasing in intrinsic arrival rates assuming monotonic decision policies is stated as Theorem 3.1.3.

Theorem 3.1.3. Let $\mathbf{\Gamma}_s = (\mu_1, ..., \mu_M, \sigma_{1,s}, ..., \sigma_{M,s}, \lambda, N_1, ..., N_M)$ for $s \in \{l, h\}$, where $\sigma_{i,l} \leq \sigma_{i,h}$, and $N_i \in \mathbb{N}$, $\forall i \in \mathcal{M}$ be the parameter vectors of two processes as described above. Let the general arrivals in both processes follow the same monotonic policy $\mathbf{D}$.
Then $z_{k,\mathbf{D},\Gamma_l}(\mathbf{n}) \leq z_{k,\mathbf{D},\Gamma_h}(\mathbf{n} + \sum_{j=1}^{M} a_j \mathbf{e}_j)$ for $a_j \in \mathbb{N}_0$ and $k \in \mathcal{M}$.

Proof. This proof proceeds along the same lines as the proof of Theorem 3.1.2, with only minor differences. We again let $X^{(1)}$ and $X^{(2)}$ be processes on $\mathcal{S}$, and define the joint process $\mathbf{X} = (X^{(1)}, X^{(2)})$ as a process on $(\mathcal{S} \times \mathcal{S}) \cup \{g_1, g_2, g_S\}$,

as defined above. The nomenclature, state description and notation is also, as applicable, the same as in the proof of Theorem 3.1.2.

As in the proof of Theorem 3.1.2 the marginal processes $X^{(1)}$ and $X^{(2)}$ differ in arrival rates, but in the present proof the difference lies in some intrinsic arrival rate, or rates, rather than in the general arrival rate. So $X^{(1)}$ follows the law of a queuing process as described above with parameter vector $\mathbf{\Gamma}_l$ while $X^{(2)}$ follows the law of another such process but with parameter vector $\mathbf{\Gamma}_h$, and the processes are again coupled in such a way that they make the same transition whenever possible. This means that general arrivals and service completions occur simultaneously in the marginal processes, and that every intrinsic arrival in process $X^{(1)}$ is matched by an identical intrinsic arrival in $X^{(2)}$, while in $X^{(2)}$ unmatched intrinsic arrivals to Q_i occur at rate $\sigma_{i,h} - \sigma_{i,l}$.

The proof again works by showing that a coupling of this kind such that the joint process never transitions to g_2 at positive rate, implying $T_1 \leq T_2$ a.s, exists. This is done by showing that the marginal processes follow the laws of the processes they represent.

We start by considering the transitions of the joint process that occur as a result of service completions, these can be found in Display 3.24.

$$(\mathbf{n}, \mathbf{n}+\mathbf{a}) \to \begin{cases} g_1 & \text{a.r.} \\ \quad \mu_1(\frac{1}{n_1+1} - \frac{1}{n_1+1+a_1}) & \\ g_S & \text{a.r.} \\ \quad \mu_1 \frac{1}{n_1+1+a_1} & \\ (\mathbf{n}-\mathbf{e}_1, \mathbf{n}+\mathbf{a}-\mathbf{e}_1) & \text{a.r.} \\ \quad \mu_1(1 - \frac{1}{n_1+1}) & \\ (\mathbf{n}-\mathbf{e}_j I_{\{n_j>0\}}, \mathbf{n}+\mathbf{a}-I_{\{n_j+a_j>0\}}\mathbf{e}_j) & \text{a.r.} \\ \quad \mu_j, 1 < j \leq M & \end{cases} \tag{3.24}$$

We note that the transitions in Display 3.24 are identical in rates and resulting states to the transitions in Display 3.6, so the arguments regarding service completions in the proof of Theorem 3.1.2 apply to this case as well. Thus we can state that, as long as the structure of the state of the joint process is preserved, service completions either result in absorption in one of the graveyard states in G_1, so that $T_1 \leq T_2$, or preserve the structure of the state of the joint process.

We thus go on to consider transitions that occur as a result of arrivals, these can be found in Display 3.25.

$$
\begin{aligned}
&(\mathbf{n}, \mathbf{n}+\mathbf{a}) \rightarrow \\
&\begin{cases}
(\mathbf{n}+\mathbf{e}_j I_{\{n_j+I_{\{j=1\}}<N_j\}}, \mathbf{n}+\mathbf{a}+I_{\{n_j+a_j+I_{\{j=1\}}<N_j\}}\mathbf{e}_j) & \text{a.r.} \\
\quad \sigma_{j,l} + \lambda\big(\min\{D_j(\mathbf{n}'), D_j(\mathbf{n}'+\mathbf{a})\} & \\
\qquad\qquad +D_j(\mathbf{n}')D_0(\mathbf{n}'+\mathbf{a})\big), 1 \le j \le M & \\
(\mathbf{n}+\mathbf{e}_j, \mathbf{n}+\mathbf{a}+\mathbf{e}_k) & \text{a.r.} \\
\quad \lambda\big(D_j(\mathbf{n}') - D_j(\mathbf{n}'+\mathbf{a})\big)^+ & \\
\quad \times\Big(\frac{\big(D_k(\mathbf{n}'+\mathbf{a}) - D_k(\mathbf{n}')\big)^+}{\sum_{l=1}^M \big(D_l(\mathbf{n}'+\mathbf{a}) - D_l(\mathbf{n}')\big)^+}\Big), 1 \le j,k \le M & \\
(\mathbf{n}, \mathbf{n}+\mathbf{a}+I_{\{n_j+a_j+I_{\{j=1\}}<N_j\}}\mathbf{e}_j) & \text{a.r.} \\
\quad \sigma_{j,h} - \sigma_{j,l}, 1 \le j \le M &
\end{cases}
\end{aligned}
\tag{3.25}
$$

With $\lambda = \lambda_l$ and $\sigma_{i,l} = \sigma_i$ we recognise the first two transitions exactly from the proof of Theorem 3.1.2 and can without further argument conclude that they preserve the structure of the state of the joint process, and we can state that the total rate of the first set of transitions is

$$
\sum_{j=1}^M \sigma_{j,l} + \lambda\Big(\sum_{j=1}^M \min\{D_j(\mathbf{n}'), D_j(\mathbf{n}'+\sum_{i=1}^M a_i\mathbf{e}_i)\} + \sum_{j=1}^M D_j(\mathbf{n}')D_0(\mathbf{n}'+\sum_{i=1}^M a_i\mathbf{e}_i)\Big) \tag{3.26}
$$

and that, as in the proof of the previous Theorem, the total rate of the second set of transitions is

$$
\lambda\sum_{j=1}^M \big(\{D_j(\mathbf{n}') - D_j(\mathbf{n}'+\sum_{i=1}^M a_i\mathbf{e}_i)\big)^+\big(1 - D_0(\mathbf{n}'+\sum_{i=1}^M a_i\mathbf{e}_i)\big) \tag{3.27}
$$

The following set of transitions in Display 3.25 represents the situations when an intrinsic arrival seeks to join Q_j in $X^{(2)}$while no event takes place in $X^{(1)}$. This transition preserves the structure of the state of the joint process. These events occur at a total rate of

$$
\sum_{j=1}^M (\sigma_{j,h} - \sigma_{j,l}) \tag{3.28}
$$

In addition to the transitions in Display 3.25 there is also the null event that occurs when a general arrival seeks to join the system in both of the marginal processes, but the systems are full in both cases. This null-event does not change the state of the joint process and occurs at rate

$$\lambda D_0(\mathbf{n}').$$

To find the total event rate we sum over the individual rates to get

$$\begin{aligned}
&\sum_{j=1}^{M}\sigma_{j,l}+\lambda\Big(\sum_{j=1}^{M}\min\{D_j(\mathbf{n}'),D_j(\mathbf{n}'+\sum_{i=1}^{M}a_i\mathbf{e}_i)\}\\
&\qquad\qquad+\sum_{j=1}^{M}D_j(\mathbf{n}')D_0(\mathbf{n}'+\sum_{i=1}^{M}a_i\mathbf{e}_i)\Big)\\
&\quad+\lambda\sum_{j=1}^{M}\big(\{D_j(\mathbf{n}')-D_j(\mathbf{n}'+\sum_{i=1}^{M}a_i\mathbf{e}_i)\big)^{+}\big(1-D_0(\mathbf{n}'+\sum_{i=1}^{M}a_i\mathbf{e}_i)\big)\\
&\qquad\qquad+\sum_{j=1}^{M}(\sigma_{j,h}-\sigma_{j,l})+\lambda D_0(\mathbf{n}')\\
&\qquad\qquad=\sum_{j=1}^{M}\big(\sigma_{j,l}-\sigma_{j,l}+\sigma_{j,h})+\lambda\Big(D_0(\mathbf{n}')\\
&\quad+\sum_{j=1}^{M}\big(\min\{D_j(\mathbf{n}'),D_j(\mathbf{n}'+\sum_{i=1}^{M}a_i\mathbf{e}_i)\}+D_j(\mathbf{n}')D_0(\mathbf{n}'+\sum_{i=1}^{M}a_i\mathbf{e}_i)\\
&\qquad+\big(D_j(\mathbf{n}')-D_j(\mathbf{n}'+\sum_{i=1}^{M}a_i\mathbf{e}_i)\big)^{+}\big(1-D_0(\mathbf{n}'+\sum_{i=1}^{M}a_i\mathbf{e}_i)\big)\Big)\\
&\qquad\qquad=\sum_{j=1}^{M}\sigma_{j,h}+\lambda\sum_{j=0}^{M}D_j(\mathbf{n}')=\sum_{j=1}^{M}\sigma_{j,h}+\lambda. \qquad (3.29)
\end{aligned}$$

We add this to the total rate of events induced by service completions, given by Equation 3.7. We find that the total event rate in the joint process and the $X^{(2)}$ process is

$$\sum_{j=1}^{M}\mu_j+\sum_{j=1}^{M}\sigma_{j,h}+\lambda \qquad (3.30)$$

and we get the total event rate for the $X^{(1)}$ process by subtracting the rate of events that only occurred in the $X^{(2)}$-process, $\sum_{j=1}^{M}(\sigma_{j,h} - \sigma_{j,l})$, which leaves

$$\sum_{j=1}^{M} \mu_j + \sum_{j=1}^{M} \sigma_{j,l} + \lambda \tag{3.31}$$

as required.

So the marginal processes have the required properties. We again employ the argument from the conclusion of the proof of Theorem 3.1.1, which states that this process is such that $T_1 \leq T_2$, which proves that$z_{k,\mathbf{D},\Gamma_l}(\mathbf{n}) \leq z_{k,\mathbf{D},\Gamma_h}(\mathbf{n} + \sum_{i=1}^{M} a_i \mathbf{e}_i)$ for $a_i \in \mathbb{N}_0$, as required. □

3.1.4 Monotonicity of expected waiting time in μ_i

One change that can be made to the system that can reasonably be assumed to not have the effect of increasing the expected waiting time for a customer joining queue k is to increase the service rate in that queue. That this is in fact the case as long as all general arrivals follow a monotonic decision policy is expressed in Theorem 3.1.4. However, slightly less obvious, the theorem also states that the expected waiting time is non-increasing in the service rate in the other queues.

Theorem 3.1.4. Consider $\Gamma_s = (\mu_{1,s}, ..., \mu_{M,s}, \sigma_1, ..., \sigma_M, \lambda, N_1, ..., N_M)$ for $s \in \{l, h\}$, where $\mu_{i,l} \leq \mu_{i,h}$ and $N_i \in \mathbb{N}\ \forall i \in \mathcal{M}$. Let $\mathbf{\Gamma}_l$ and $\mathbf{\Gamma}_h$ be the parameter vectors of two processes, as described above, in which all general arrivals follow the monotonic decision policy, $\mathbf{D}$.
Then $z_{k,\mathbf{D},\mathbf{\Gamma}_h}(\mathbf{n}) \leq z_{k,\mathbf{D},\mathbf{\Gamma}_l}(\mathbf{n} + \sum_{j=1}^{M} a_j \mathbf{e}_j)$ for $a_j \in \mathbb{N}_0$ and $k \in \mathcal{M}$.

Proof. We again define a joint process $\mathbf{X} = (X^{(1)}, X^{(2)})$ on $(\mathcal{S} \times \mathcal{S}) \cup \{g_1, g_2, g_S\}$, in the same way as the previous proofs. In this case the difference between the marginal processes is in service rate in such a way that the server in Q_j works at rate $\mu_{j,h}$ in $X^{(1)}$ but $\mu_{j,l}$ in $X^{(2)}$ where $\mu_{j,h} \geq \mu_{j,l}$. With a coupling such that the marginal processes experience the same transition when possible this means that there will be a potential service completion in Q_j in $X^{(2)}$ whenever there is one in $X^{(1)}$, but service completions occur in $X^{(1)}$ without a matching event in $X^{(2)}$ at rate $\mu_{j,h} - \mu_{j,l}$.

We use the same notation as above, so that $\mathbf{n}' = \mathbf{n} + \mathbf{e}_1$ and we let $\mathbf{a} = (a_1, ..., a_M)$ with $a_i \in \mathbb{N}_0$ for all $i \in \mathcal{M}$. We show that this coupling exists, and

that it is such that starting from a state of the form $(\mathbf{n}, \mathbf{n}+\mathbf{a})$ the joint process is eventually absorbed in one of the graveyard states in G_1 with probability 1 so that $T_1 \leq T_2$ a.s.

We start by examining the transitions related to service completions, which can be found in Display 3.32.

$$(\mathbf{n}, \mathbf{n}+\mathbf{a}) \rightarrow \begin{cases} g_1 & \text{a.r.} \\ \quad \mu_{1,l}(\frac{1}{n_1+1} - \frac{1}{n_1+1+a_1}) + (\mu_{1,h} - \mu_{1,l})\frac{1}{n_1+1} & \\ g_S & \text{a.r.} \\ \quad \mu_{1,l}\frac{1}{n_1+1+a_1} & \\ (\mathbf{n} - \mathbf{e}_1, \mathbf{n} + \mathbf{a} - \mathbf{e}_1) & \text{a.r.} \\ \quad \mu_{1,l}(1 - \frac{1}{n_1+1}) & \\ (\mathbf{n} - \mathbf{e}_1, \mathbf{n} + \mathbf{a}) & \text{a.r.} \\ \quad (\mu_{1,h} - \mu_{1,l})(1 - \frac{1}{n_1+1}) & \\ (\mathbf{n} - I_{\{n_j>0\}}\mathbf{e}_j, \mathbf{n} + \mathbf{a} - I_{\{n_j+a_j>0\}}\mathbf{e}_j) & \text{a.r.} \\ \quad \mu_{j,l}, 1 < j \leq M & \\ (\mathbf{n} - I_{\{n_j>0\}}\mathbf{e}_j, \mathbf{n} + \mathbf{a}) & \text{a.r.} \\ \quad (\mu_{j,h} - \mu_{j,l}), 1 < j \leq M & \end{cases} \tag{3.32}$$

We see that if we were to replace $\mu_{j,l}$ with μ_j we recognise the majority of these transitions from Display 3.6 in the proof of Theorem 3.1.2. We therefore focus on the difference between these transitions and the transitions in that case.

The transition to graveyard state g_1, where the marked customer has been served in $X^{(1)}$ but not in $X^{(2)}$, occurs when there is a service completion in Q_1 in both processes at the familiar rate that is a function of a_1 and $\mu_{1,l}$. In addition to that it also occurs at a rate that is a function of the difference in service rates in Q_1 in both processes $(\mu_{1,h} - \mu_{1,l})$ and the occupancy of Q_1 in $X^{(1)}$.

The transition to graveyard state g_S occurs at a rate that is identical to previous proofs with $\mu_1 = \mu_{1,l}$.

The same is true for the next transition in Display 3.32, the service completion of someone other than the marked customer in Q_1 in both processes.

The next transition is one that has not occurred in previous proofs. This is the transition that occurs as a result of a service completion other than the marked customer in Q_1 in $X^{(1)}$ while no event takes place in $X^{(2)}$.

So far all of the transitions discussed have been single events, as opposed to sets of events, so the full rates are given in Display 3.32.

With $\mu_{j,l} = \mu_j$ the next set of transitions, service completion in Q_j in both marginal processes, is also familiar from previous proofs.

The final set of transitions occur as the result of a service completion in Q_j in $X^{(1)}$ that is unmatched in $X^{(2)}$. As this transition lowers the number of customers in Q_j in $X^{(1)}$ without affecting the occupancy of Q_j in $X^{(2)}$ it preserves the structure of the state of the joint process. If $n_j = 0$ this is a null event in $X^{(1)}$. The total rate of these transitions is

$$\sum_{j=2}^{M}(\mu_{j,h} - \mu_{j,l}).$$

The total rate of events in the joint process induced by services is thus

$$\begin{aligned}\mu_{1,l}(\frac{1}{n_1+1} - \frac{1}{n_1+1+a_1}) + \mu_{1,l}\frac{1}{n_1+1+a_1} + \mu_{1,l}(1 - \frac{1}{n_1+1}) \\ + (\mu_{1,h} - \mu_{1,l})\frac{1}{n_1+1} + (\mu_{1,h} - \mu_{1,l})(1 - \frac{1}{n_1+1}) + \sum_{j=2}^{M}\mu_{j,l} + \sum_{j=2}^{M}(\mu_{j,h} - \mu_{j,l}) \\ = \sum_{j=1}^{M}\mu_{j,h} \quad (3.33)\end{aligned}$$

which is also the rate of events induced by service completions in $X^{(1)}$, while the rate of service completion in $X^{(2)}$ is

$$\sum_{j=1}^{M}\mu_{j,l}.$$

We turn to the transitions that occur as a result of arrivals to the system, these can be found in Display 3.34.

$$
\begin{aligned}
&(\mathbf{n}, \mathbf{n}+\mathbf{a}) \rightarrow \\
&\begin{cases}
(\mathbf{n}+\mathbf{e}_j I_{\{n_j+I_{\{j=1\}}<N_j\}}, \mathbf{n}+\mathbf{a}+I_{\{n_j+a_j+I_{\{j=1\}}<N_j\}}\mathbf{e}_j) & \text{a.r.} \\
\quad \sigma_j + \lambda\big(\min\{D_j(\mathbf{n}'), D_j(\mathbf{n}'+\mathbf{a})\} & \\
\qquad\qquad +D_j(\mathbf{n}')D_0(\mathbf{n}'+\mathbf{a})\big), 1 \le j \le M & \\
(\mathbf{n}+\mathbf{e}_j, \mathbf{n}+\mathbf{a}+\mathbf{e}_k) & \text{a.r.} \\
\quad \lambda\big(D_j(\mathbf{n}') - D_j(\mathbf{n}'+\mathbf{a})\big)^+ & \\
\quad \times\Big(\frac{\big(D_k(\mathbf{n}'+\mathbf{a}) - D_k(\mathbf{n}')\big)^+}{\sum_{l=1}^{M}\big(D_l(\mathbf{n}'+\mathbf{a}) - D_l(\mathbf{n}')\big)^+}\Big), 1 \le j,k \le M &
\end{cases}
\end{aligned}
\tag{3.34}
$$

We see that with $\lambda_l = \lambda_h = \lambda$ the transitions and the associated rates are identical to those in table 3.8, so we know that these transitions preserve the structure of the state of the joint process, and that the event rate of these transitions is

$$\lambda + \sum_{i=1}^{M} \sigma_i.$$

We see that this coupling is such that $T_1 \le T_2$ a.s. and that the marginal processes $X^{(1)}$ and $X^{(2)}$ have total event rates $\sum_{j=1}^{M}\mu_{j,h} + \lambda + \sum_{i=1}^{M}\sigma_i$ and $\sum_{j=1}^{M}\mu_{j,l} + \lambda + \sum_{i=1}^{M}\sigma_i$, respectively. Since we have shown that $T_1 \le T_2$ a.s. this concludes the proof.

□

3.1.5 k-asymmetry in expected waiting time for $\mathbf{n}+\mathbf{e}_k$

We have already shown that the expected waiting time is higher for a marked customer joining the system in state $\mathbf{n}+\mathbf{e}_i$, than it is for a marked customer joining the system in state $\mathbf{n}$ regardless of where the extra customer is, and which queue the marked customer chooses to join. This is Theorem 3.1.1. Theorem 3.1.5 instead deals with comparing the expected waiting time of a marked customer in two processes that have the same total occupancy upon the marked customer's moment of arrival. The two processes differ, though, in that in one process there

is an added user in the same queue as the marked customer, and in the other the added user is in the other queue. Intuitively the marked customer in the process with the added customer in the other queue should have the lowest expected waiting time. Theorem 3.1.5 states this for the situation with $M = 2$ when all general arrivals operate under the same monotonic decision policy and there are no intrinsic arrivals to the system.

Theorem 3.1.5. Let a queuing system, as defined above, with $M = 2$ operate under parameter vector $\Gamma = (\mu_1, \mu_2, \sigma_1 = 0, \sigma_2 = 0, \lambda, N_1, N_2)$ where $N_1, N_2 \in \mathbb{N}$, and let all general arrivals to the system follow the monotonic decision policy $\mathbf{D}$. Then $z_{i,\mathbf{D},\Gamma}(\mathbf{n} + \mathbf{e}_j) \leq z_{i,\mathbf{D},\Gamma}(\mathbf{n} + \mathbf{e}_i)$, with $(i, j) \in \{(1, 2), (2, 1)\}$.

Proof. This proof utilises the same type of coupling argument as in the previous proofs. So we again define $\mathbf{X} = (X^{(1)}, X^{(2)})$ on $(\mathcal{S} \times \mathcal{S}) \cup \{g_1, g_2, g_S\}$. The state space as well as the stochastic variables T_1 and T_2 are defined as in previous proofs. We now proceed to define a coupling such that the marginal processes make the same transition whenever possible and follow the law of a process arising from a queuing system of the type under consideration.

Importantly the proof also shows that starting from a state of form $(\mathbf{n}+\mathbf{e}_2, \mathbf{n}+\mathbf{e}_1)$ with a marked customer in queue 1 the coupling is such that the joint process never transitions to g_2 at positive rate. As before this means that $T_1 \leq T_2$ a.s. which thus implies that $z_1(\mathbf{n} + \mathbf{e}_2) \leq z_1(\mathbf{n} + \mathbf{e}_1)$, and by symmetry $z_i(\mathbf{n} + \mathbf{e}_j) \leq z_i(\mathbf{n} + \mathbf{e}_i)$ for $(i, j) \in \{(1, 2), (2, 1)\}$.

All the transitions that occur in the joint process at positive rate are listed in Display 3.35.

$$(\mathbf{n}+\mathbf{e}_2, \mathbf{n}+\mathbf{e}_1) \to \begin{cases} (\mathbf{n}+\mathbf{e}_2-\mathbf{e}_1, \mathbf{n}) & \text{a.r.} \\ \quad \mu_1(\frac{1}{n_1+1}-\frac{1}{n_1+2}) & \\ (\mathbf{n}+\mathbf{e}_2, \mathbf{n}) & \text{a.r.} \\ \quad \mu_1 \frac{1}{n_1+2} & \\ (\mathbf{n}+\mathbf{e}_2-\mathbf{e}_1, \mathbf{n}) & \text{a.r.} \\ \quad \mu_1 \frac{n_1}{n_1+1} & \\ (\mathbf{n}, \mathbf{n}+\mathbf{e}_1-\mathbf{e}_2 I_{\{n_2\geq 1\}}) & \text{a.r.} \\ \quad \mu_2 & \\ (\mathbf{n}+\mathbf{e}_2+\mathbf{e}_i, \mathbf{n}+\mathbf{e}_1+\mathbf{e}_i) & \text{a.r.} \\ \quad \lambda D_i(\mathbf{n}'+\mathbf{e}_i) & \\ (\mathbf{n}+\mathbf{e}_2+\mathbf{e}_1, \mathbf{n}+\mathbf{e}_1+\mathbf{e}_2) & \text{a.r.} \\ \quad \lambda(D_1(\mathbf{n}'+\mathbf{e}_2)-D_1(\mathbf{n}'+\mathbf{e}_1)). & \end{cases} \tag{3.35}$$

We first note that the lack of intrinsic arrivals together with the fact that the initial state guarantees room in both marginal processes, result in a situation in which the transitions due to arrivals are very straightforward. We nevertheless start by considering the transitions that result from service completions.

We see that with this initial state the rates to both g_1 and g_S are always positive. We also see that the third transition only occurs at positive rate when $n_1 > 0$ and thus it always preserves the structure of the joint process.

The fourth transition, the transition that occurs as a result of a service completion in Q_2 in both the marginal processes, preserves the structure of the joint process as long as $n_2 > 0$. However, with $n_2 = 0$ this transition is a null event in $X^{(2)}$ and a service completion event in $X^{(1)}$ so that the resulting state is of the form $(\mathbf{n}, \mathbf{n}+\mathbf{e}_1)$. This transition changes the state of the joint process. The resulting state is a familiar one, though, and in particular a special case of the starting state in the proof of Theorem 3.1.1. We see in that proof that this coupling and starting point gives us $T_1 \leq T_2$ a.s.

The total rate of events induced by service completions is

$$\mu_1(\frac{1}{n_1+1}-\frac{1}{n_1+2})+\mu_1\frac{1}{n_1+2}+\mu_1\frac{n_1}{n_1+1}+\mu_2=\mu_1+\mu_2$$

We go on to consider the transitions induced by arrivals to the system. The first of these is the transition that occurs when a general arrival decides to join Q_i in both of the marginal processes. As this transition can only occur at positive rate when $n_i+1<N_i$ it preserves the structure of the state of the joint process.

Due to decision policy monotonicity the complementary case, the transition in which a general arrival chooses to join different queues, only occurs at positive rate for one particular transition. This is the transition in which a general arrival joins Q_1 in $X^{(1)}$ and Q_2 in $X^{(2)}$. The resulting state of this transition can be written as $(\hat{\mathbf{n}},\hat{\mathbf{n}})$, so this is a coupling event. This means that when this transition has occurred $T_1=T_2$. We note that we could have written the rate for this transition as $(D_2(\mathbf{n}'+\mathbf{e}_1)-D_2(\mathbf{n}'+\mathbf{e}_2))$ since $(D_1(\mathbf{n}'+\mathbf{e}_2)-D_1(\mathbf{n}'+\mathbf{e}_1))=(D_2(\mathbf{n}'+\mathbf{e}_1)-D_2(\mathbf{n}'+\mathbf{e}_2))$ by symmetry.

The event rate induced by arrivals in the joint process as well as both marginal processes is

$$\begin{aligned}\lambda\sum_{i=1}^{2}D_i(\mathbf{n}+\mathbf{e}_i)+\lambda(D_1(\mathbf{n}'+\mathbf{e}_2)-D_1(\mathbf{n}'+\mathbf{e}_1))\\ =\lambda\big(D_1(\mathbf{n}+\mathbf{e}_1)+D_2(\mathbf{n}+\mathbf{e}_2)+D_1(\mathbf{n}'+\mathbf{e}_2)-D_1(\mathbf{n}'+\mathbf{e}_1\big)\\ =\lambda\big((1-D_1(\mathbf{n}+\mathbf{e}_2))+D_1(\mathbf{n}'+\mathbf{e}_2)\big)=\lambda\end{aligned}$$

where we could use $D_1(\mathbf{n})=1-D_2(\mathbf{n})$ because we know that the process is in a state in $\mathcal{S}^-$ thereby guaranteeing $D_0(\mathbf{n})=0$.

Thus we see that the total event rate in the joint process, as well as both marginal processes, is

$$\mu_1+\mu_2+\lambda$$

as required and we have seen that starting from a state of the form $(\mathbf{n}+\mathbf{e}_2,\mathbf{n}+\mathbf{e}_1)$ the joint process never transitions to the graveyard state g_2 at positive rate which means that $T_1\leq T_2$ a.s. which in turn implies $z_1(\mathbf{n}+\mathbf{e}_2)\leq z_1(\mathbf{n}+\mathbf{e}_1)$. □

3.2 Stationary joining probabilities

The second set of results we consider relate to a quantity we call the *stationary joining probability* for queue i or $p_i(\mathbf{\Gamma}, \mathbf{D})$. This is the probability that, when the system has reached stationarity, a general arrival to the system who does not balk will choose to join queue i. This quantity captures, in one dimension, properties of the parameter vector as well as the decision policy of a system. We come back to this in Chapter 4. In the present chapter we just consider certain basic properties of the quantity.

We define the stationary joining probability by letting $\pi_{\mathbf{\Gamma},\mathbf{D}}(\mathbf{n})$ be the probability that a general arrival finds a system which has reached stationarity in state $\mathbf{n}$ when the system operates under parameter vector $\mathbf{\Gamma}$, and all general arrivals follow the policy $\mathbf{D}$. Then

$$p_i(\mathbf{\Gamma}, \mathbf{D}) = \frac{\sum_{\mathbf{n}\in\mathcal{S}} \pi_{\mathbf{\Gamma},\mathbf{D}}(\mathbf{n}) D_i(\mathbf{n})}{\sum_{\mathbf{n}\in\mathcal{S}: D_0(\mathbf{n})\neq 1} \pi_{\mathbf{\Gamma},\mathbf{D}}(\mathbf{n})}, \mathbf{\Gamma} \in \mathcal{G}, \mathbf{D} \in \mathcal{D}, i \in \mathcal{M}.$$

So the stationary joining probability of queue i is the probability that a general arrival who joins a system that has reached stationarity joins queue i.

3.2.1 Monotonicity of stationary joining probability in μ_i

One property that is reasonable to expect $p_i(\mathbf{\Gamma}, \mathbf{D})$, the stationary joining probability in queue i, to have in the case of a fixed, monotonic decision policy $\mathbf{D}$ is to be non-decreasing in μ_i, the service rate in queue i. We show this in the following Theorem. We consider the case of also updating the decision policy to reflect the change in service rate by example and numerics in Chapter 4.

Theorem 3.2.1. Let $\mathbf{\Gamma} = (\lambda, \mu_1, \mu_2, \sigma_1, \sigma_2, N_1, N_2)$ with $N_1, N_2 \in \mathbb{N}$, and let $\mathbf{\Gamma}_i^+$ be identical to $\mathbf{\Gamma}$ with the exception that μ_i has been replaced with μ_i^+ such that $\mu_i^+ \geq \mu_i$. Let $\mathbf{\Gamma}$ and $\mathbf{\Gamma}_i^+$ be parameter vectors of processes such as those described above, in which all general arrivals follow the monotonic policy $\mathbf{D}$. Then

$$p_i(\mathbf{\Gamma}, \mathbf{D}) \leq p_i(\mathbf{\Gamma}_i^+, \mathbf{D}).$$

Proof. We prove this Theorem by a coupling argument in the case of $i = 1$, but it holds for the case when $i = 2$ by symmetry. We start by defining the joint

process $\mathbf{X}(t) = (\mathbf{X}^1(t), \mathbf{X}^2(t))$ such that $\mathbf{X}^i(t) = (X_1^i(t), X_2^i(t))$. Each $\mathbf{X}^i$ is an occupancy process on a system of two parallel processor sharing queues so that $X_j^i(t)$ represents the occupancy of queue j in process i at time t.

We let the $\mathbf{X}^2$ process operate under the parameter vector $\mathbf{\Gamma} = (\lambda, \mu_1, \mu_2, \sigma_1, \sigma_2, N_1, N_2)$ and the $\mathbf{X}^1$ process operate under the parameter vector $\mathbf{\Gamma}^+ = (\lambda, \mu_1^+, \mu_2, \sigma_1, \sigma_2, N_1, N_2)$ where $\mu_1^+ \geq \mu_1$. All general arrivals follow the monotonic decision policy $\mathbf{D}$.

We construct the marginal processes by sums of elementary Poisson processes. We let the marginal process $\mathbf{X}^2$ experience general arrivals according to a Poisson process with rate λ, intrinsic arrivals to queue i according to a Poisson process with rate σ_i and potential services in queue i according to a Poisson process with rate μ_i. We let $\mathbf{X}^1$ be governed by the same set of Poisson processes in addition to which it experiences potential service events in queue 1 according to a Poisson process with rate $\mu_1^+ - \mu_1$.

We construct the decisions of a general arrival in such a way that when a general arrival enters the system at time t a random number $u(t) \sim U(0,1)$ is drawn and if $D_1(\mathbf{X}^i(t)) \leq u(t)$ the general arrival joins queue 1 in the $\mathbf{X}^i$-process, otherwise the user joins queue 2. This construction has the effect of ensuring that the general arrivals make the same choice in all events when they follow the same decision policy while asymmetries in the decisions of general arrivals are always such that they reflect the asymmetries in the decision rules related to the states that the marginal processes are in.

With these constructions the marginal processes $\mathbf{X}^i$ follow the laws of X, as defined in Section 2.1.

Without loss of generality $\mathbf{X}^2(t)$ can be expressed as $\mathbf{X}^1(t) + l_1(t)\mathbf{e}_1 + l_2(t)\mathbf{e}_2$ where $l_i(t) = X_i^2(t) - X_i^1(t)$, so that a general state of the joint process has the form $\mathbf{X}(t) = (\mathbf{X}^1(t), \mathbf{X}^1(t) + l_1(t)\mathbf{e}_1 + l_2(t)\mathbf{e}_2)$.

We denote state space transitions as transition process $\mathbf{v}(t) = (\mathbf{v}_1(t), \mathbf{v}_2(t))$ where $\mathbf{v}_i(t)$ represents the change to the $\mathbf{X}^i(t)$ process at time t. We also let t^- denote the moment just before a transition that occurs at time t. With this notation $\mathbf{X}(t) = \mathbf{X}(t^-) + \mathbf{v}(t)$.

The first step of this proof consists of showing that starting from $\mathbf{X}(t) = (\mathbf{X}^1(t), \mathbf{X}^1(t))$, that is any state in which $l_1(t) = l_2(t) = 0$, then $l_i(t') \geq 0$ a.s., $\forall i \in (1,2), t' \geq t$. In other words, we show that Figure 3.1 depicts all the transitions that can occur at positive rate in the $l_1 l_2$-plane starting from any state such that $l_i \geq 0, \forall i \in \{1,2\}$.

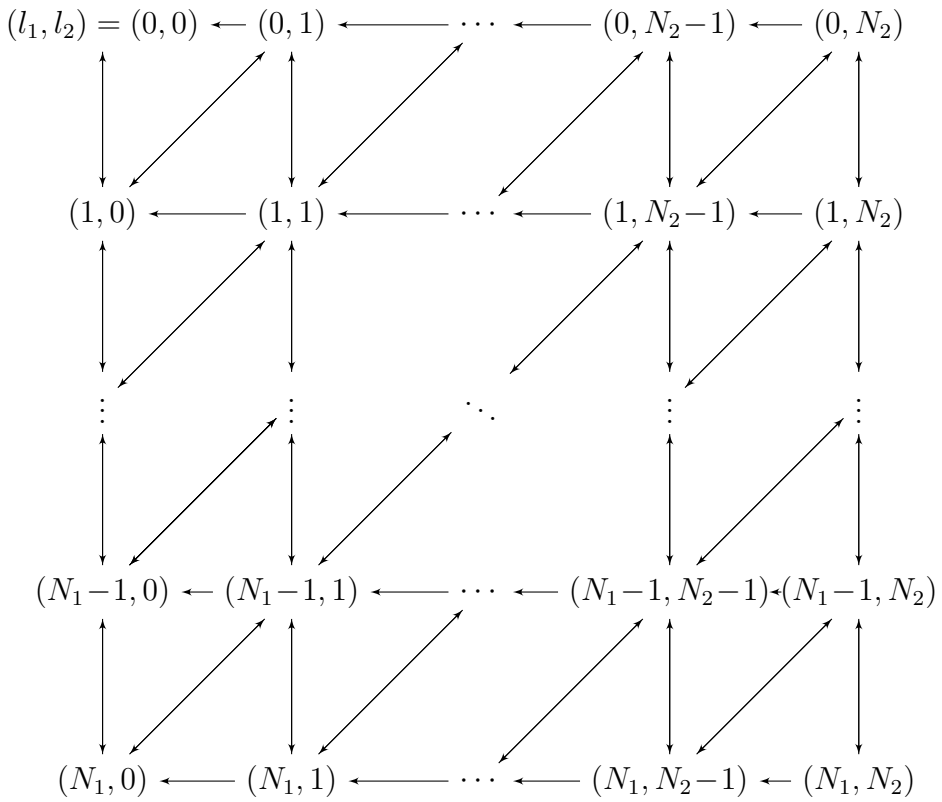


FIGURE 3.1: All types of transitions that can occur at positive rate in each state of the part of the $l_1 l_2$-plane such that $l_i \geq 0, \forall i \in \{1, 2\}$.

We first note that the only asymmetric transition that occurs in a state of form $(\mathbf{X}^1(t), \mathbf{X}^1(t))$ is the one that occurs as a result of the higher service rate in queue 1 in $\mathbf{X}^1$, the transition $(-\mathbf{e}_1, 0)$. This transition occurs at positive rate when $X_1^1(t) > 0$ and leads to a state of the form $(\mathbf{X}^1(t), \mathbf{X}^1(t) + \mathbf{e_1})$ where $l_1(t) = 1$ and $l_2(t) = 0$.

Therefore we need to consider the transitions that occur at positive rate when $l_1(t) > 0$ and $l_2(t) = 0$. Again, the transition that results from an unmatched service in queue 1 in the $\mathbf{X}^1(t)$ process occurs at positive rate when $\mathbf{X}_1^1(t) > 0$ when this transition occurs at t the result is that $l_1(t) = l_1(t^-) + 1$ preserving $l_1(t) > 0$. When $X_1^2(t) = N_1$ both general and intrinsic arrivals join queue 1

in $\mathbf{X}^1$ process unmatched at positive rates, so that the joint process experiences the transition $(\mathbf{e}_1, 0)$. When $X_1^1(t) = 0$ a service event in queue 1 affecting both marginal processes leads to a null service in the $\mathbf{X}^1$ process so that the transition is of the form $(0, -\mathbf{e_1})$. When either of these two transitions occur at time t the effect is that $l_1(t) = l_1(t^-) - 1$ so that $l_1(t) \geq 0$. The final asymmetric transition that occurs at positive rate in states where $l_1(t) > 0$ and $l_2(t) = 0$ is the transition resulting from a general arrival joining queue 1 in $\mathbf{X}^1$ and queue 2 in $\mathbf{X}^2$, where queue 1 is longer. This transition occurs at positive rate when $X_2^1(t) = X_2^2(t) < N_2$, and has the effects $l_1(t) = l_1(t^-) - 1$ and $l_2(t) = l_2(t^-) + 1$ so that $l_1(t) \geq 0$ and $l_2(t) = 1 > 0$. So all transitions that occur at positive rate at time t when $l_1(t^-) > 0$ and $l_2(t^-) = 0$ are such that $l_i(t) \geq 0, \forall i \in \{1, 2\}$.

This prompts us to consider the transitions that occur from states where $l_1(t) = 0$ and $l_2(t) > 0$. Four types of asymmetric transitions occur at positive rate at time t when $l_1(t) = 0$ and $l_2(t) > 0$. The transition that occurs as a result of the difference in service rate in queue 1 between the two processes, $(-\mathbf{e}_1, 0)$ occurs at positive rate at time t when $X_1^1(t) > 0$ and has the effect $l_1(t) = l_1(t^-) + 1 = 1$. When $X_2^2(t) = N_2$ the transition $(\mathbf{e}_2, 0)$ occurs at positive rate as a result of arrivals, general or intrinsic, to queue 2. When $X_2^1(t) = 0$ the transition $(0, -\mathbf{e}_2)$ occurs at positive rate as a result of service in queue 2. When either of these transitions occurs at time t it has the effect $l_2(t) = l_2(t^-) - 1 \geq 0$. Finally the process in states of this type also experiences an asymmetric transition due to general arrivals joining different queues in the different marginal processes. In this case this transition, which only occurs when $X_1^1(t) < N_1$, has the form $(\mathbf{e}_2, \mathbf{e}_1)$ as a user joins queue 2 in $\mathbf{X}^1$ but queue 1 in $\mathbf{X}^2$, where queue 2 is longer. This transition has the effects $l_2(t) = l_2(t^-) - 1$ and $l_1(t) = l_1(t^-) + 1$ so that $l_2(t) \geq 0$ and $l_1(t) = 1 > 0$. So all transitions that occur at positive rate at time t when $l_2(t^-) > 0$ and $l_1(t^-) = 0$ are such that $l_i(t) \geq 0, \forall i \in \{1, 2\}$.

When a transition occurs at time t when the joint process is in a state where $l_i(t^-) > 0, \forall i \in \{1, 2\}$ then trivially $l_i(t) \geq 0$ as a transition changes $l_i(t^-)$ by at most 1.

Thus we can state that when a joint process is in state $\mathbf{X}(0) = (\mathbf{X}^1(0), \mathbf{X}^1(0))$ then at any $t > 0$ the state of the joint process can be expressed as $\mathbf{X}(t) = (\mathbf{X}^1(t), \mathbf{X}^1(t) + l_1(t)\mathbf{e}_1 + l_2(t)\mathbf{e}_2)$ with $l_i(t) \geq 0, \forall i \in \{1, 2\}$.

When $l_i(t) \geq 0, \forall i \in \{1, 2\}$ 7 types of asymmetric transitions occur at positive rate. We state them with rates in Display 3.36.

$$\begin{cases} (-\mathbf{e}_1, 0) & \text{a.r.} \\ \quad I_{\{X_1^1(t)>0\}}(\mu_1^+ - \mu_1) + I_{\{X_1^1(t)>0\}}\mu_1 & \\ (\mathbf{e}_1, 0) & \text{a.r.} \\ \quad I_{\{X_1^1(t)<N_1 \cap X_1^2(t)=N_1\}}(\sigma_1 + \lambda D_1(\mathbf{X}^1)) & \\ (0, -\mathbf{e}_1) & \text{a.r.} \\ \quad I_{\{X_1^1(t)=0 \cap l_1(t)>0\}}\mu_1 & \\ (\mathbf{e}_1, \mathbf{e}_2) & \text{a.r.} \\ \quad \lambda(D_1(\mathbf{X}^1) - D_1(\mathbf{X}^2))^+ & \\ (\mathbf{e}_2, \mathbf{e}_1) & \text{a.r.} \\ \quad \lambda(D_2(\mathbf{X}^1) - D_2(\mathbf{X}^2))^+ & \\ (0, -\mathbf{e}_2) & \text{a.r.} \\ \quad I_{\{X_2^1(t)=0 \cap l_2(t)>0\}}\mu_2 & \\ (\mathbf{e}_2, 0) & \text{a.r.} \\ \quad I_{\{X_2^1(t)<N_2 \cap X_2^2(t)=N_2\}}(\sigma_2 + \lambda D_2(\mathbf{X}^1)) & \end{cases} \tag{3.36}$$

We now use these transitions to express a general state of the process at time t. To this end we assume that at time t the process has experienced $k_1(t)$ events of $(-\mathbf{e}_1, 0)$, $k_2(t) = (k_2^I(t) + k_2^G(t))$ of $(\mathbf{e}_1, 0)$ where $k_2^I(t)$ is the number of such transitions that occur due to intrinsic arrivals and $k_2^G(t)$ are the number due to general arrivals, $k_3(t)$ of $(0, -\mathbf{e}_1)$, $k_4(t)$ of $(\mathbf{e}_1, \mathbf{e}_2)$, $k_5(t)$ of $(\mathbf{e}_2, \mathbf{e}_1)$, $k_6(t)$ of $(0, -\mathbf{e}_2)$ and $k_7(t) = (k_7^I(t) + k_7^G(t))$ of $(\mathbf{e}_2, 0)$ where $k_7^I(t)$ and $k_7^G(t)$ are defined as in the case $k_2(t)$. Thus the state of a process that starts in state $\mathbf{X}(0) = (\mathbf{X}^1(0), \mathbf{X}^1(0))$ can be expressed as

$$\begin{aligned} \mathbf{X}(t) = \big(\mathbf{X}^1(t), \mathbf{X}^1(t)\big) &+ k_1(t)(-\mathbf{e}_1, 0) + k_2(t)(\mathbf{e}_1, 0) + k_3(t)(0, -\mathbf{e}_1) \\ &+ k_4(t)(\mathbf{e}_1, \mathbf{e}_2) + k_5(t)(\mathbf{e}_2, \mathbf{e}_1) + k_6(t)(0, -\mathbf{e}_2) + k_7(t)(\mathbf{e}_2, 0) \end{aligned}$$

which we can rewrite as

$$\mathbf{X}(t) = \Big(\mathbf{X}^1(t), \mathbf{X}^1(t) + \big(k_1(t) + k_5(t) - k_2(t) - k_3(t) - k_4(t)\big)\mathbf{e}_1 + \big(k_4(t) - k_5(t) - k_6(t) - k_7(t)\big)\mathbf{e}_2\Big)$$

So we can identify

$$l_1(t) = k_1(t) + k_5(t) - k_2(t) - k_3(t) - k_4(t)$$

and

$$l_2(t) = k_4(t) - k_5(t) - k_6(t) - k_7(t)$$

and then conclude, due to $l_i(t) > 0$ that

$$k_1(t) + k_5(t) \geq k_2(t) + k_3(t) + k_4(t)$$

and

$$k_4(t) \geq k_5(t) + k_6(t) + k_7(t)$$

which implies that

$$k_1(t) + k_5(t) \geq k_2^G(t) + k_3(t) + k_4(t) \tag{3.37}$$

and

$$k_4(t) \geq k_5(t) + k_6(t) + k_7^G(t) \tag{3.38}$$

Now we define a set of counting processes, $J_1^i(t)$ and $J^i(t)$ for $i \in \{1, 2\}$. The process $J_1^i(t)$ counts the number of general arrivals who have joined queue 1 up to and including time t in the $\mathbf{X}^i$-process. The process $J^i(t)$ counts the total number of general arrivals who have joined the system, rather than balked, up to and including time t in the $\mathbf{X}^i$-process. We also define the two counting processes $J^S(t)$ counting the number of general arrivals that have occurred simultaneously and chosen to join the same queue in the marginal processes up to and including time t and likewise $J_1^S(t)$ denoting the number of users who have joined queue 1 simultaneously in the marginal processes up to and including time t.

Now let $\rho^i(t) = \frac{J_1^i(t)}{J^i(t)}$. Since the occupancy processes are irreducible and finite we can state by ergodicity that

$$\text{as } t \to \infty, \rho^1(t) \to p_1(\mathbf{\Gamma}^+, \mathbf{D}) \text{ a.s.}$$

and

$$\text{as } t \to \infty, \rho^2(t) \to p_1(\mathbf{\Gamma}, \mathbf{D}) \text{ a.s..}$$

Thus the final part of the proof consists of showing that

$$\rho^1(t) \geq \rho^2(t), a.s, \forall t > 0.$$

Of the transitions that occur at positive rate at time $t \geq 0$ only 4 change the values of the counting processes, J^i and J_1^i, at all. We explicitly state how these changes occur. We see that $J^1(t) = J^1(t^-) + 1$ when either of $(\mathbf{e}_1, 0)$, $(\mathbf{e}_1, \mathbf{e}_2)$, $(\mathbf{e}_2, \mathbf{e}_1)$, or $(\mathbf{e}_2, 0)$ occur at time t. A subset of these transitions also leads to changes in $J_1^1(t)$, in particular $J_1^1(t) = J_1^1(t^-) + 1$ when $(\mathbf{e}_1, 0)$ or $(\mathbf{e}_1, \mathbf{e}_2)$ occur at time t. When it comes to the other process; $J^2(t) = J^2(t^-) + 1$ when $(\mathbf{e}_1, \mathbf{e}_2)$ or $(\mathbf{e}_2, \mathbf{e}_1)$ occur at t and $J_1^2(t) = J_1^2(t^-) + 1$ when $(\mathbf{e}_2, \mathbf{e}_1)$ occurs at time t. So we can write the process counting the total number of general arrivals who join the system in the $\mathbf{X}^1(t)$-process as

$$J^1(t) = J^S(t) + k_2^G(t) + k_4(t) + k_5(t) + k_7^G(t),$$

the total number of arrivals who join the system in the $\mathbf{X}^2(t)$-process as

$$J^2(t) = J^S(t) + k_4(t) + k_5(t),$$

the number of general arrivals who join queue 1 in the $\mathbf{X}^1$ process as

$$J_1^1(t) = J_1^S(t) + k_2^G(t) + k_4(t)$$

and finally the number of general arrivals who join queue 1 in the $\mathbf{X}^2$ process as

$$J_1^2(t) = J_1^S(t) + k_5(t).$$

So now we are ready to show that $\rho_1(t) \geq \rho_2(t)$ a.s. by evaluating $\rho^1(t) - \rho^2(t)$ in terms of the transition processes. So we first write the expression we are interested in

$$\rho^1(t) - \rho^2(t) = \frac{J_1^1(t)}{J^1(t)} - \frac{J_1^2(t)}{J^2(t)}$$

We rewrite this as

$$\rho^1(t) - \rho^2(t) = \frac{J_1^1(t)J^2(t) - J_1^2(t)J^1(t)}{J^1(t)J^2(t)}$$

where we see that the denominator is positive, so that to show that the whole expression is positive we only have to consider the numerator. We expand the numerator

$$(J_1^S(t) + k_2^G(t) + k_4(t))(J^S(t) + k_4(t) + k_5(t)) \\ - (J_1^S(t) + k_5(t))(J^S(t) + k_2^G(t) + k_4(t) + k_5(t) + k_7^G(t))$$

which we rewrite as

$$J^S(t)(k_2^G(t) + k_4(t) - k_5(t)) - J_1^S(t)(k_2^G(t) + k_7^G(t)) \\ + k_4(t)(k_2^G(t) + k_4(t)) - k_5(t)(k_5(t) + k_7^G(t)). \quad (3.39)$$

We start by considering the factors related to the simultaneous counting processes

$$J^S(t)(k_2^G(t) + k_4(t) - k_5(t)) - J_1^S(t)(k_2^G(t) + k_7^G(t)) \\ \geq J^S(t)(k_2^G(t) + k_4(t) - k_5(t)) - J^S(t)(k_2^G(t) + k_7^G(t)) \\ = J^S(t)(k_4(t) - k_5(t) - k_7^G(t))$$

Were the inequality follows since $J^S(t) \geq J_1^S(t)$. By Equation 3.38 we have $k_4(t) \geq k_5(t) + k_6(t) + k_7^G(t)$ which implies $k_4(t) \geq k_5(t) + k_7^G(t)$ so

$$J^S(t)(k_2^G(t) + k_4(t) - k_5(t)) - J_1^S(t)(k_2^G(t) + k_7^G(t)) \geq 0 \quad (3.40)$$

We then consider the remaining terms of Equation 3.39

$$k_4(t)(k_2^G(t) + k_4(t)) - k_5(t)(k_5(t) + k_7^G(t)) \geq k_4(t)(k_2^G(t) + k_4(t)) - k_4(t)k_4(t) \\ = k_4(t)k_2^G(t) \geq 0$$

where we have again used Equation 3.38 which implies that $k_4(t) \geq k_5(t)$ and $k_4(t) \geq (k_5(t) + k_7^G(t))$. Thus

$$k_4(t)(k_2^G(t) + k_4(t)) - k_5(t)(k_5(t) + k_7^G(t)) \geq 0. \quad (3.41)$$

By Inequalities 3.40 and 3.41 we can conclude that the expression in Display 3.39 is positive a.s.. which implies that

$$\frac{J_1^1(t)J^2(t) - J_1^2(t)J^1(t)}{J^1(t)J^2(t)} \geq 0, \text{ a.s.},$$

which further implies that

$$\rho^1(t) \geq \rho^2(t), \text{ a.s..}$$

Hence, by ergodicity, we obtain

$$p_1(\mathbf{\Gamma}^+, \mathbf{D}) \geq p_1(\mathbf{\Gamma}, \mathbf{D})$$

as required. □

3.2.2 Monotonicity of stationary joining probability in σ_i

Another property that we would expect the system to have when all general arrivals follow a monotonic decision policy is that we would not expect that increasing the intrinsic arrival rate to queue i increases the probability that a general arrival joins queue i. Again, this can intuitively be understood by considering that an increased intrinsic arrival rate to queue i would lead to queue i being more likely to be in a state with more users in it, which in the case of a monotonic decision policy means a lower, or at least not a higher, probability that a general arrival joins queue i. We state this as Theorem 3.2.2.

Theorem 3.2.2. Let $\mathbf{\Gamma} = (\lambda, \mu_1, \mu_2, \sigma_1, \sigma_2, N_1, N_2)$ with $N_1, N_2 \in \mathbb{N}$, and let $\mathbf{\Gamma}_i^+$ be identical to $\mathbf{\Gamma}$ with the exception that σ_i has been replaced with σ_i^+ such that $\sigma_i^+ \geq \sigma_i$. Let $\mathbf{\Gamma}$ and $\mathbf{\Gamma}_i^+$ be parameter vectors of processes such as described above, in which all general arrivals follow the monotonic policy $\mathbf{D}$. Then

$$p_i(\mathbf{\Gamma}, \mathbf{D}) \geq p_i(\mathbf{\Gamma}_i^+, \mathbf{D}).$$

Proof. We prove this Theorem by a small modification of the proof of Theorem 3.2.1. We again do this in the context of $i = 1$, but as before it holds for the case when $i = 2$ by symmetry.

We start by defining joint process $\mathbf{X}(t) = (\mathbf{X}^1(t), \mathbf{X}^2(t))$ in the same way as in the proof of Theorem 3.2.1.

We let the $\mathbf{X}^1$ process operate under the parameter vector $\mathbf{\Gamma} = (\lambda, \mu_1, \mu_2, \sigma_1, \sigma_2, N_1, N_2)$ and the $\mathbf{X}^2$ process operate under the parameter vector $\mathbf{\Gamma}^+ = (\lambda, \mu_1, \mu_2, \sigma_1^+, \sigma_2, N_1, N_2)$ where $\sigma_1^+ \geq \sigma_1$. All general arrivals follow the monotonic decision policy $\mathbf{D}$.

With these process definitions the extension from the proof of Theorem 3.2.1 is trivial. □

3.3 Extension to infinite capacity systems

We are also interested in the case of infinite capacity queues and the theorems stated above can be extended to that case with some adjustment to the conditions. The extension to infinite capacity also allows us to make a statement similar to that in Theorem 3.1.5, but for different conditions on the system parameters.

3.3.1 Extension to infinite capacity queues

All the Theorems stated so far in this chapters are stated in the context of limited capacity system, and otherwise arbitrary system parameters, except in the case of Theorem 3.1.5 where intrinsic arrivals are exluded. We note that in each case the capacity limitation induces system stability, guaranteeing that the Theorems have well defined interpretations, but otherwise is not required for the proofs to work. Thus if we require the choice of system parameters and decision policy to be such that the system is stable even in the infinite capacity case, the same properties are hold based on trivially reduced versions of the same proofs. We state this as corollary.

Corollary 3.3.1. The statements in Theorems 3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.1.5, 3.2.1 and 3.2.2 hold in the case that $\exists i$ s.t. $N_i = \infty$ when remaining parameters and the decision policy are chosen in such a way that system stability is guaranteed.

Proof. In the proof associated with each of the theorems the adjustment to the case with infinite capacity in some queue or queues represents, for stable systems, a trivial reduction in the complexity of the coupling. □

3.3.2 k-asymmetry in expected waiting time for $\mathbf{n} + \mathbf{e}_k$, infinite queues

With infinite capacity queues we also find an alternate condition under which we can prove the expected waiting time asymmetry with respect to occupancy in Theorem 3.1.5.

Theorem 3.1.5 states that in a finite system a marked customer who joins queue i in state $\mathbf{n}+\mathbf{e}_j$ will not have a higher expected waiting time than a marked customer who joins queue i in state $\mathbf{n}+\mathbf{e}_i$ with arbitrary queue capacities as long as no intrinsic arrivals occur. Corollary 3.3.1 extends the statement, in the case of

stable systems, to the case when one or both queues has infinite capacity. We can now show the same property to hold under somewhat complementary conditions; when intrinsic arrival rates are arbitrary but the queues' capacities aren't limited. So the system behaviour described in Theorem 3.3.1 is identical to that in Theorem 3.1.5, the difference is the conditions on the system.

Theorem 3.3.1. Let a queuing system, as defined above, with $M = 2$ operate under the parameter vector $\Gamma = (\mu_1, \mu_2, \sigma_1, \sigma_2, \lambda, N_1 = \infty, N_2 = \infty)$ and let all general arrivals to the system follow the monotonic decision policy $\mathbf{D}$. Let Γ and $\mathbf{D}$ be such that the queueing system is stable.
Then, with $(i, j) \in \{(1, 2), (2, 1)\}$, $z_{i,\Gamma}(\mathbf{n} + \mathbf{e}_j) \leq z_{i,\Gamma}(\mathbf{n} + \mathbf{e}_i)$.

Proof. This theorem and Theorem 3.1.5 only differ in the constraints that are imposed on the system parameters in order for the theorem to hold. The proofs too only differ on a few small points. We define the joint process $\mathbf{X}$ exactly as in the proof of Theorem 3.1.5, and again consider the system when it is in state $(\mathbf{n} + \mathbf{e}_2, \mathbf{n} + \mathbf{e}_1)$ with a marked customer in queue 1.

The coupling such that the marginal processes make the same transitions whenever possible includes exactly the same set of transitions induced by service completions at the exact same rate here as in the proof of Theorem 3.1.5. We can thus, based on the proof of that Theorem as well as Corollary 3.3.1, state that transitions induced by service completions either lead to graveyard states g_1 or g_S so that $T_1 \leq T_2$ a.s., preserve the structure of the state of the joint process, or lead to other states for which we have already showed that this coupling results in $T_1 \leq T_2$ a.s.. We also know that the total rate of these transitions is

$$\mu_1 + \mu_2.$$

The transitions related to services in the system change slightly from the proof of Theorem 3.1.5, due to the change in parameters. The transitions that occur as a result of arrivals are listed in Display 3.42.

$$(\mathbf{n} + \mathbf{e}_2, \mathbf{n} + \mathbf{e}_1) \to \begin{cases} (\mathbf{n} + \mathbf{e}_2 + \mathbf{e}_i, \mathbf{n} + \mathbf{e}_1 + \mathbf{e}_i) & \text{a.r.} \\ \quad \lambda D_i(\mathbf{n}' + \mathbf{e}_i) + \sigma_i & \\ & \\ (\mathbf{n} + \mathbf{e}_2 + \mathbf{e}_1, \mathbf{n} + \mathbf{e}_1 + \mathbf{e}_2) & \text{a.r.} \\ \quad \lambda(D_1(\mathbf{n}' + \mathbf{e}_2) - D_1(\mathbf{n}' + \mathbf{e}_1)). & \end{cases} \tag{3.42}$$

The first set of transition in Display 3.42 consist of the transitions that occur when a customer seeks to join Q_j in both marginal processes. Here this occurs as a result of both general and intrinsic arrivals. As both queues are infinite, though, this transition always preserves the structure of the state of the joint process.

The final transition is identical to the final transition in Display 3.35 and as such is always a coupling event so that when this transition occurs $T_1 = T_2$.

The total rate of transitions of the joint process, as well as both marginal processes, that occur as a result of arrivals is

$$\lambda + \sum_{i=1}^{2} \sigma_i.$$

Thus the total event rate is

$$\lambda + \sum_{i=1}^{2} \sigma_i + \sum_{i=1}^{2} \mu_i$$

as required. Thus, as in previous proofs, $T_1 \leq T_2$ a.s. which implies that with $(i, j) \in \{(1, 2), (2, 1)\}$, $z_{i,\Gamma}(\mathbf{n} + \mathbf{e}_j) \leq z_{i,\Gamma}(\mathbf{n} + \mathbf{e}_i)$ as required. □

3.4 Summary

The majority of the results we see in this chapter conform to well known results from simpler queuing systems. We find them interesting partially due to the fact that these are simply new results, which give an indication of regular behaviour. This is particularly interesting since we have shown properties for systems with general M, while it has been previously found that the system has the capacity to exhibit counter-intuitive non-monotonicities even with $M = 2$. These results also indicate that there are other, potentially more complex, properties of this system that could yield to analysis with this method.

Theorems 3.1.1 and 3.1.4 relate to the intuitive fact that we never expect to have a shorter waiting time when there are more people in the system when we arrive, or when the servers work more slowly. Similarly Theorems 3.2.1 and 3.2.2 relate to the fact that we would expect general arrivals to have a higher probability of choosing to join queues with a higher service rate, or lower intrinsic arrival rates.

Slightly more specific to this particular situation, but still unsurprising are the statements in Theorems 3.1.2, 3.1.3 which state that we expect our waiting time to be non-decreasing in all of the arrival rates to the system. While this conforms to the behaviour of simpler systems, such as the single FCFS queue, the statement here has slightly more relevance than in that case. In that case there is simply no dependence between arrival rates and the expected waiting time of an arrival who finds $\mathbf{n}$ users already in the system when they arrive, while in the case presented here such a dependence exists, making these theorems that much more relevant. The fact of this dependence can be clearly seen in the expressions governing the state based expected waiting time; Equation 2.5 in Chapter 2.

We also note that there is some possible insight to be gleaned from the way in which we need to restrict the system in order to be able to prove the property stated in Theorems 3.1.5 and 3.3.1. We can make these observations due to proving these theorems by this straightforward coupling which proceeds by considering the possible transitions of the system. These properties are also ones that we would have thought of as trivial in the case of a system of FCFS queues for the reason that the occupancy in queues other than the one that the marked customer joins is simply irrelevant to their waiting time. In this case, however, the waiting time property is non-trivial, and in fact requires additional assumptions for this proof method to work. In particular, the additional assumptions needed are either that there are no intrinsic arrivals to the system, or that the system has room for infinitely many users. These two conditions both amount to ensuring the relationship between the occupancy of the two queues doesn't get inverted by an intrinsic arrival to the system when the marked customer is in a full queue in one of the processes but not in the other. So while this particular outcome does not offer a solution to the counter-intuitive behaviours presented in Chapter 4 it does show us ways in which the system behaviour might defy basic intuitions of customers.

4

Policy iteration

> *And it happened all the time that the compromise between two perfectly rational alternatives was something that made no sense at all.*
>
> – Neal Stephenson, Anathem

In this chapter the system under consideration is a set of two parallel queues of limited maximum length, each queue with a single server working under the processor sharing discipline. So in other words we are considering a system of the kind described in Chapter 2 with $M = 2$, $N_1 \in \mathbb{N}$, and $N_2 \in \mathbb{N}$.

The concept of policy iteration in this exact system has been briefly studied numerically before, and results can be found in Chen [45]. This chapter briefly describes those results and then extends them in several directions. We reach the same conclusions as in Chen [45] - that the user behaviour in this system does not lend itself to easy description. However, in our extended treatment we find reasons for optimism as well as a way forward.

In Section 4.1 we present the definitions related to the method of policy iteration. The treatment of the method and system follows Chen [45] where applicable, but the notation differs in some regards, notably in relation to the explicit notation for decision policies. Most of the properties under consideration here have not previously been considered.

In Section 4.2 we present the results of numerical exploration of this method. In particular certain choices of system parameters are shown to lead to counter intuitive, or potentially undesirable, results. We also present the system design choices that have had the most useful outcomes in this section.

In Section 4.3 we discuss the conclusions we draw from these numerical explorations, some of which are further explored in this thesis, and others which are left unanswered.

4.1 Definitions

4.1.1 Fundamental description

Policy iteration is a straightforward method of finding equilibrium policies, as described in Chapter 2. As suggested by the name the method relies on iteratively updating policies in such a way that in each step of the iteration the probability of making either decision is updated by state by state comparisons of the expected waiting times as calculated based on the decision policy from the previous step of the iteration. The goal of the iterative process is to find a policy that is optimal against itself so that the best behaviour of a marked customer is simply to use the same decision policy as all other general arrivals in the system. This is clearly an example of a Nash equilibrium as it is a situation in which no user can unilaterally improve their experience of the process by defecting.

As discussed implicitly in the previous chapters we have chosen to work with expected waiting times as the user's measure of system performance. Users seek to minimise the amount of time that they spend in the system, but as that amount of time is stochastic they instead have to work in expectation based on their complete information of the system. Note that in the context of processor sharing without a waiting area, *waiting time*, *service time* and *sojourn time* can all be taken to refer to the same quantity, so we use the terms essentially interchangeably. This is because unlike many other service disciplines, including the closely related Random Service-discipline and Round Robin-discipline described in Chapter 2, every user starts receiving service the moment they enter the system. This makes it obvious why the sojourn time and service time are identical, but how about the waiting time? Based on the easily understood identity between service time and sojourn time it would be reasonable to view it as though there is no waiting time in the system at all. However, the alternative way to think of how the system works discussed in Section 2.1 in Chapter 2 shows that it can sometimes be useful to think of the whole time a user spends in the system as waiting time while the actual completed service is bestowed upon the lucky user in an instant. From this perspective there is clearly identity between waiting time and sojourn time, while the service time is 0.

Working with expected waiting times is not the only possible choice for this method. Some simple alternatives would be some cost function based on expected waiting time, or the probability of being served in less than some specified amount

of time. While more exotic choices might include things like peak effective service rate or more complex quality of service criteria that differ from customer to customer. As we only deal with one class of customers, and we don't include reneging after initiated service there seemed to be little reason to use something more complex than expected waiting time. In later chapters we will work in slightly different quantities, though. In particular in the main project described in Chapter 5 as well as the other projects described in that chapter, where we use cost functions that are not identical to the expected waiting times, even if they are closely related.

4.1.2 Policy iteration algorithm

We introduce the policy iteration algorithm here with some concepts left implicit, but hopefully comprehensible, to be explicitly described in the following sections. In the description of the algorithm we assume that the system operates under some parameter vector $\mathbf{\Gamma}$, the specifics of which is not relevant to this discussion. In the description of the algorithm we use $t = 0, 1, 2, ...$ to refer to steps of the iteration, and we let $\mathbf{D}^t = \{\mathbf{D}_1^t, \mathbf{D}_2^t, \mathbf{D}_0^t\}$ refer to the decision policy at iterative step number t.

Algorithm 1 The policy iteration algorithm

$t = 0$
Choose a $t_{\max}$
Choose an initial decision policy $\mathbf{D}^0$
while $\max\{|\mathbf{D}^t - \mathbf{D}^{t-1}|\} > \varepsilon$ AND $t \leq t_{\max}$ **do**
    Using $\mathbf{D}^t$ find the expected waiting times $z_{1,\mathbf{D}^t}(\mathbf{n})$ and $z_{2,\mathbf{D}^t}(\mathbf{n})$ for every $\mathbf{n} \in \mathcal{S}$.
    Find $D_i^{t+1} = f_i(z_{1,\mathbf{D}^t}(\mathbf{n}), z_{2,\mathbf{D}^t}(\mathbf{n}))$ for each $\mathbf{n} \in \mathcal{S}$, and thus find $\mathbf{D}^{t+1}$
    $t = t + 1$
end while
if $t < t_{\max}$ **then**
    $D^* = D^t$
else
    The iteration has failed to produce a fixed point policy.
end if

The first important step of the algorithm, line 3, is to pick an initial decision policy, we don't discuss sensitivity to initial condition for the iteration here, but think of the choice as arbitrary while trying to make it in sensible, intuitive ways. Then, at line 5, the expected waiting times for both queues in each state are calculated on the basis of the initial decision policy. At the following line, line 6, an update function, $\mathbf{f} = \{f_1, f_2\}$, operates on the expected waiting times to determine the next decision policy. We use several different update functions, $\mathbf{f}$, through this study and the concept, as well as the specific functions, are explicitly defined in Section 4.1.3. For the purposes of the explanation of the iterative algorithm itself it is sufficient to state that update functions might return values in $\{0, 1\}$ but also in some cases the decision rules returned are taken from the open set $(0, 1)$, this is relevant when we look at the condition for the while loop. Once the first decision policy is calculated a comparison takes place between the $\mathbf{D}^0$ and $\mathbf{D}^1$ to see if the initial choice of decision policy was already a fixed point policy. For $\mathbf{D}^0$ to be a fixed point policy we require that $\mathbf{D}^0 = \mathbf{D}^1$, and this is indeed what we expect in the cases when the decision rules take values in $\mathbb{N}$. Since this is a numerical algorithm, though, we instead examine whether the difference between the two policies is lower than some tolerance. This is done by comparing $|D_i^0(\mathbf{n}) - D_i^1(\mathbf{n})|$ for each $\mathbf{n} \in \mathcal{S}$ and $i \in \{1, 2\}$ to some small tolerance $\varepsilon > 0$. If the policy passes that test they are considered identical, and the algorithm moves on to line 9.

If the test fails the algorithm returns to line 5, a new set of expected waiting times is calculated, this time based on $\mathbf{D}^1$. Based on these expected waiting times $\mathbf{D}^2$ is found using $\mathbf{f}$, and $|D_i^1(\mathbf{n}) - D_i^2(\mathbf{n})|$ is compared to ε and so on.

The iteration continues either until the ε-test passes, or until some maximum number of iterations, $t_{\max}$, is reached. As stated on line 10 if the test is passed by the t-th policy then $\mathbf{D}^* = \mathbf{D}^t$, the policy is taken to be a fixed point policy. If instead $t_{\max}$ is reached the policy iteration is taken to have failed to find a fixed point policy. We discuss this possibility in some detail in Section 4.1.4, when we are armed with a full understanding of the different update rules employed in this study.

4.1.3 Update rules

Recall Section 2.3.3 where we described the routing rules as the way in which the general arrivals translate the expected waiting times into probabilities of joining

each queue. In the context of the policy iteration algorithm we think of the routing rules as update rules and express them as the set of functions

$$\mathbf{f}(z_{1,\mathbf{D}}(\mathbf{n}), z_{2,\mathbf{D}}(\mathbf{n})) = \{f_1(z_{1,\mathbf{D}}(\mathbf{n}), z_{2,\mathbf{D}}(\mathbf{n})), f_2(z_{1,\mathbf{D}}(\mathbf{n}), z_{2,\mathbf{D}}(\mathbf{n}))\}$$

referred to in line 6 of Algorithm 1. This set of functions is in some sense the crux of the calculation as the functions are responsible for setting up the condition for the convergence test, as well as for the next step of the iteration in case the convergence test fails.

The update rule is a set of functions that take the expected waiting times for a marked customer joining either queue in state $\mathbf{n}$, which are real numbers, as inputs and returns decision rules for joining either queue, or balking, in that state. So

$$\mathbf{f} : \mathbb{R}^2 \to [0, 1]^3.$$

The reason why each of the different rules were employed is discussed further in Section 4.2, we simply present the functions here. We denote each update rule by the subset of $[0, 1]$ which they take decision rules from. In the following Sections we explain the update rules and use the same expected waiting times to give an example of an update step for each update rule. To avoid confusion here we ignore the boundary states of the system and just assume that the expected waiting times we are looking at are the expected waiting times of internal states. For the same reason we don't give the decision policy for balking as balking only ever happens with non-zero probability on the boundary in our models. As these are just examples we round the expected waiting times to two significant digits.

$\{0, 1, \tilde{p}\}$-update rule

This is in many ways the most natural choice of update rule, and is the one that was employed in the initial study of this system in Chen [45]. This rule is such that a marked customer will choose to join the queue with the lowest expected waiting time with probability 1. In the event that the expected waiting times are exactly equal the marked customer joins queue i with probability $\tilde{p}_i$, and consequently queue j with probability $\tilde{p}_j = 1 - \tilde{p}$. By convention we sometimes let $\tilde{p} = \tilde{p}_1$. The variable $\tilde{p}$ is arbitrarily chosen, but some natural choices would be $\tilde{p} = 0.5$, or either $\tilde{p} = 0$ or $\tilde{p} = 1$. With these two last choices the $\{0, 1, \tilde{p}\}$-update rule reduces to something we could justifiably call a $\{0, 1\}$-update rule, but we do not make this distinction.

Thus the update rule can be expressed as in Definition 4.1.1

Definition 4.1.1. When policy iteration uses $\{0, 1, \tilde{p}\}$-update rules the update function $\mathbf{f} = \{f_1, f_2\}$ has the components

$$f_i(z_{1,\mathbf{D}}(\mathbf{n}), z_{2,\mathbf{D}}(\mathbf{n})) = I_{\{z_{i,\mathbf{D}}(\mathbf{n})<z_{j,\mathbf{D}}(\mathbf{n})\}} + \tilde{p}_i I_{\{z_{i,\mathbf{D}}(\mathbf{n})=z_{j,\mathbf{D}}(\mathbf{n})\}}$$

for $i, j \in \{1, 2\}$.

We now introduce the method of giving sets of expected waiting times in matrix form in such a way that the expected waiting time for a user who joins Queue i when they find the queue in state (k, l) and all general arrivals use decision policy $\mathbf{D}^t$, can be found at position $(k-1, l-1)$ in the matrix $\mathbf{Z}_{i,\mathbf{D}^t}$. We state this as Definition 4.1.2.

Definition 4.1.2.

$$\mathbf{Z}_{i,\mathbf{D}^t} = \begin{bmatrix} z_{i,\mathbf{D}}(0,0) & z_{i,\mathbf{D}}(0,1) & z_{i,\mathbf{D}}(0,2) & \cdots \\ z_{i,\mathbf{D}}(1,0) & z_{i,\mathbf{D}}(1,1) & z_{i,\mathbf{D}}(1,2) & \cdots \\ z_{i,\mathbf{D}}(2,0) & z_{i,\mathbf{D}}(2,1) & z_{i,\mathbf{D}}(2,2) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

In Example 4.1.1 we give a set of expected waiting times. We use this same set of expected waiting times in the description of each update rule to illustrate how the different update rules deal with the same set of expected waiting times.

Example 4.1.1.

$$\mathbf{Z}_{1,\mathbf{D}^t} = \begin{bmatrix} 1.0 & 1.5 & 1.8 & \cdots \\ 2.0 & 2.4 & 2.5 & \cdots \\ 3.0 & 3.3 & 3.4 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \quad \mathbf{Z}_{2,\mathbf{D}^t} = \begin{bmatrix} 1.0 & 1.6 & 2.1 & \cdots \\ 1.5 & 1.9 & 2.6 & \cdots \\ 1.9 & 2.2 & 2.7 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

When the $\{0, 1, \tilde{p}\}$-update rule is applied to the expected waiting times given in Example 4.1.1 we find the updated decision policies in Example 4.1.2

Example 4.1.2.

$$\mathbf{D}_1^{t+1} = \begin{bmatrix} \tilde{p} & 1 & 1 & \cdots \\ 0 & 0 & 1 & \cdots \\ 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \quad \mathbf{D}_2^{t+1} = \begin{bmatrix} (1-\tilde{p}) & 0 & 0 & \cdots \\ 1 & 1 & 0 & \cdots \\ 1 & 1 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

$(0,1)$-decision rules

This update rule is such that the probability of a user joining each queue is inversely proportional to the waiting time of that queue. Thus, in state $\mathbf{n} \in \mathcal{S}^-$ by Definition 2.2.1, a general arrival never joins either queue with probability 1 but instead always randomises their choice. The reason this only holds for internal states is that the decision rules in boundary states, $\mathbf{n} \in \mathcal{S}^e$, are governed first by the boundary conditions, discussed in Section 4.1.5, and in some situations secondarily by the update rules.

Definition 4.1.3. When policy iteration uses $(0,1)$-update rules the update function $\mathbf{f} = \{f_1, f_2\}$ has the components

$$f_i(z_{1,\mathbf{D}}(\mathbf{n}), z_{2,\mathbf{D}}(\mathbf{n})) = \frac{z_{j,\mathbf{D}}(\mathbf{n})}{z_{i,\mathbf{D}}(\mathbf{n}) + z_{j,\mathbf{D}}(\mathbf{n})}$$

for $i, j \in \{1, 2\}$.

This, together with the expected waiting times in Example 4.1.1 gives the decision rules in Example 4.1.3, where the decision rules are rounded to 3 significant digits.

Example 4.1.3.

$$\mathbf{D}_1^{t+1} = \begin{bmatrix} 0.500 & 0.516 & 0.538 & \cdots \\ 0.429 & 0.442 & 0.510 & \cdots \\ 0.388 & 0.400 & 0.443 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix},$$

$$\mathbf{D}_2^{t+1} = \begin{bmatrix} 0.500 & 0.484 & 0.462 & \cdots \\ 0.571 & 0.558 & 0.490 & \cdots \\ 0.612 & 0.600 & 0.557 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

$[0,1]$-decision rules

The final update rule under consideration is intermediate between the previous two. With this update rule a user faced with a sufficiently large difference between the expected waiting times of the two queues will chose to join the queue with the

lower expected waiting time with probability 1, but for smaller differences a linear function on $(0, 1)$ governs the probability of the user joining either queue.

Unlike the other update rules this one requires the choice of a parameter, δ, defining the difference between expected waiting times that triggers the pure policies. How this parameter choice affects system behaviour is explored to some extent in Section 4.2.2.

Definition 4.1.4. When policy iteration uses $[0, 1]$-update rules the update function $\mathbf{f} = \{f_1, f_2\}$ has the components

$$f_i(z_{1,\mathbf{D}}(\mathbf{n}), z_{2,\mathbf{D}}(\mathbf{n})) =$$
$$I_{\{z_{j,\mathbf{D}}(\mathbf{n}) - z_{i,\mathbf{D}}(\mathbf{n}) > \delta\}} + I_{\{|z_{i,\mathbf{D}}(\mathbf{n}) - z_{j,\mathbf{D}}(\mathbf{n})\}| \leq \delta\}}(0.5 - \frac{z_{i,\mathbf{D}}(\mathbf{n}) - z_{j,\mathbf{D}}(\mathbf{n})}{2\delta}),$$
$$\text{for } i, j \in \{1, 2\}.$$

The central idea behind this update rule is to create what is, in some sense, a continuous version of the $\{0, 1, \tilde{p}\}$-update rules, where instead of a specific mixed update rule being employed for identical waiting times there is instead a continuum of mixed rules for sufficiently close expected waiting times, while sufficiently different waiting times still result in pure rules. The choice of function between the two extremes is of course arbitrary, and the reason for the choice we have made above is simply that a linear function in the difference is the most obvious choice. For example the function could have been chosen in such a way that as $\delta \to \infty$ this update rule approaches the $(0, 1)$-update rule, just as it goes to the $\{0, 1, \tilde{p}\}$-update rule with $\tilde{p} = 0.5$ when $\delta \to 0$ now.

As with the previous update rules we use the expected waiting times from Example 4.1.1 to show what decision policies look like with this update rule. This can be found in Example 4.1.4 where we use $\delta = 0.5$.

$$\mathbf{Z}_{1,\mathbf{D}^t} = \begin{bmatrix} 0.5 & 0.6 & 0.8 & \cdots \\ 0 & 0 & 0.6 & \cdots \\ 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix},$$

Example 4.1.4.

$$\mathbf{Z}_{2,\mathbf{D}^t} = \begin{bmatrix} 0.5 & 0.4 & 0.2 & \cdots \\ 1 & 1 & 0.4 & \cdots \\ 1 & 1 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

4.1.4 Lack of policy convergence

Before we start examining the ways in which policy iteration can fail to converge we remind ourselves that all of this takes place in the context of finite state spaces, that is $N_i < \infty$. For the following discussion on the ways in which the policy iteration can fail to converge we also assume that $t_{\max}$ in Algorithm 1 is arbitrarily large. Failure of the policy iteration to converge can in theory take a number of forms, and the set of possibilities depends on the specific subset of $[0, 1]$ from which the decision rules are taken. In our case this depends primarily on the form of the update rule, $\mathbf{f}$. To understand this we remind ourselves that each of the three update rules presented in 4.1.3 are sets of function $\mathbf{f} : \mathbb{R}^2 \to [0, 1]^3$, but that the subsets of $[0, 1]^3$ that the update rule actually returns values in vary significantly from update rule to update rule. The relevant difference is whether the subset in question is countable or not. Both the $(0, 1)$-update rule and the $[0, 1]$-update rule return values from an uncountable subset of $[0, 1]^3$, while the $\{0, 1, \tilde{p}\}$-update rule returns results from a countable, and in fact finite, subset of $[0, 1]^3$.

The first type of non-convergence relates to the situation when the range of the update function is a finite set. In this situation we are dealing with a finite set of outcomes on a finite set of states, so the full set of possible decision policies is also finite. We note that the iterative algorithm is deterministic. This means that if we consider two decision policies $\mathbf{D}^A$ and $\mathbf{D}^B$ that are such that, with parameter vector $\mathbf{\Gamma}$, $\mathbf{D}^0 = \mathbf{D}^A \Rightarrow \mathbf{D}^1 = \mathbf{D}^B$, then we know that $\mathbf{D}^t = \mathbf{D}^A \Rightarrow \mathbf{D}^{t+1} = \mathbf{D}^B$. Thus, if we denote the set of policies that have been visited by the iteration at the tth step $\mathcal{D}^t$ then we see that either

$$\mathbf{D}^{t+1} \in \mathcal{D}^t$$

or

$$\mathbf{D}^{t+1} \notin \mathcal{D}^t.$$

When $\mathbf{D}^{t+1} \notin \mathcal{D}^t$ the updated set of visited policies is $\mathcal{D}^{t+1} = \mathbf{D}^{t+1} \cup \mathcal{D}^t$. Otherwise, in the case that $\mathbf{D}^{t+1} \in \mathcal{D}^t$, either $\mathbf{D}^{t+1} = \mathbf{D}^t$ or $\mathbf{D}^{t+1} = \mathbf{D}^\tau$ where $\tau \neq t$. In the case when $\mathbf{D}^{t+1} = \mathbf{D}^t$ we don't have a problem, the iteration will simply stop as a fixed point policy has been found. In the other case, however, we now have full knowledge of what the following policies will be, as $\mathbf{D}^\tau$ will be followed by the same $\mathbf{D}^{\tau+1}$ that followed it last time, all the way until we again reach $\mathbf{D}^{t+1}$, which takes us right back to $\mathbf{D}^\tau$. We refer to the subset of $\mathcal{D}$ that is visited by a periodic orbit as $\mathcal{D}^p$, and note that if $\mathbf{D}^0 \in \mathcal{D}^p$ then $\mathbf{D}^t \in \mathcal{D}^p, \forall t \geq 0$. Both the set of achievable policies, and the set of policies in periodic orbits depend on the update rule as well as the parameter vector. To reduce notational complexity we can suppress indices referring to the update rules and parameter vector without causing problems.

When the iteration gets locked in a periodic orbit we refer to the policies that are part of that orbit as *periodic policies* and we refer to $|\mathcal{D}^p|$ as the *period* of the periodic policy.

We then note that we have no a priori reason to expect at most one periodic orbit to exist for each parameter vector and update rule. In fact $\mathcal{D}^p$ can in theory consist of several disjoint sets of periodic orbits. As a way to ensure that we can refer to a specific periodic orbit we add a subscript to the description of a set of policies in a periodic orbits, and let $\mathcal{D}^p_{\mathbf{D}^A}$ refer to the set of policies that are visited by policy iteration starting from the decision policy $\mathbf{D}^A$, in the case when $\mathbf{D}^A$ is itself part of a periodic orbit. Note that if $\mathbf{D}^A$ and $\mathbf{D}^B$ are both part of the same periodic orbit $\mathcal{D}^p_{\mathbf{D}^A}$ and $\mathcal{D}^p_{\mathbf{D}^B}$ refer to the same set.

It is easy to see that in general $2 \leq |\mathcal{D}^p_{\mathbf{D}_A}| \leq |\mathcal{D}|$ where $|\mathcal{D}^p_{\mathbf{D}_A}| = 2$ is the simple situation when

$$\mathbf{D}^A \to \mathbf{D}^B \to \mathbf{D}^A$$

while $|\mathcal{D}^p_{\mathbf{D}_A}| = |\mathcal{D}|$ is the situation when the iteration goes through each and every possible policy before returning to the starting point and going through the whole set again.

We now return to the set $\mathcal{D}^t$ from earlier in this section and note that as $t \to |\mathcal{D}|$ then $|\mathcal{D}^t| \to |\mathcal{D}|$, assuming that periodicity or convergence have not occured. So the iteration either finds a fixed point policy or gets stuck in a periodic orbit.

Clearly it is theoretically possible for periodic policies to occur in the case of the update rules that return decision rules in the uncountable sets as well, but it is much less likely. In this situation another, much more straightforward, sort of non-convergence can occur instead. Since $|\mathcal{D}| = \infty$ in this case it is, at least in theory, possible that every step in the iteration leads to a completely new policy, such that

$$\exists \mathbf{n}, \text{ s.t } |D_i^{t+1}(\mathbf{n}) - D_i^t(\mathbf{n})| > \varepsilon$$

forever. This type of non-convergence has not been observed numerically at all with this model, so it is not discussed in the numerical work.

4.1.5 Boundary conditions

An issue that we explore in this chapter is the affect of customer behaviour on the edges of the system, or what we refer to as the system's *boundary conditions*. The edges of the system are the states $\mathbf{n} \in \mathcal{S}^e$ by the notation from Section 2.2 in Chapter 2. By customer behaviour in this situation we refer to the range of possible actions for a general arrival who finds the system in one of these states. This might seem quite trivial - clearly they simply join the other queue. This is indeed one of the boundary conditions that we employ, called *Automatically Routing Boundaries* below. We are, however, also interested in boundary conditions that might more closely resemble an infinite system, which can obviously not be realised numerically. With this in mind we define *Rejection Boundaries* and *z-Routing Boundaries*.

For the purposes of this discussion it is useful to define the set of states for which one queue is full, but the other has room left. This is done in Definition 4.1.5.

Definition 4.1.5. Let $\mathcal{N}_i^e$ be the set of states where $n_i = N_i$ and $n_j < N_j$.

Automatically routing boundaries

Automatically routing boundaries, or *Natural Boundaries* refer to the situation when a general arrival who finds the system in a state in $\mathcal{N}_i^e$ joins queue j with probability one, we state this as Definition 4.1.6.

Definition 4.1.6. A system is operating under automatically routing boundaries if

$$\mathbf{n} \in \mathcal{N}_i^e \Rightarrow D_j(\mathbf{n}) = 1$$

This behaviour doesn't need any justification as it conforms to our intuitive idea about how users behave.

Rejection boundaries

When the system operates under Rejection Boundaries it works in such a way that a general arrival that finds the system in a state where either queue is full balks with probability one. We state this as Definition 4.1.7.

Definition 4.1.7. A system is operating under Rejection boundaries if

$$\mathbf{n} \in \mathcal{S}^e \Rightarrow D_0(\mathbf{n}) = 1$$

This behaviour, in contrast to the automatically routing boundaries clearly needs some justification. We first note that this boundary condition creates a system in which the general arrivals never join the system without having made a non-forced choice of which queue to join. The relevance of this can perhaps best be understood by considering the system behaviour in the edge states in the case of automatically routing boundaries. We consider the case when $n_i = N_i$. In this situation the system ceases to have three separate arrival streams, and instead can be viewed as a system with only one arrival stream, a stream of designated arrivals to queue j, with effective rate that we can call σ_j^e of magnitude $\sigma_j^e = \sigma_j + \lambda$. This is clearly a situation when the dynamics of the system changes completely from what is the case with $\mathbf{n} \in \mathcal{S}^-$, and thus a situation that has no analogy in an infinite system.

$\hat{z}$-routing boundaries

As described in Section 4.1.1, in the context of policy iteration decision rules are functions of expected waiting times, so as discussed in Section 4.1.2 we can write $D_i(\mathbf{n}) = f_i(z_1(\mathbf{n}), z_2(\mathbf{n}))$. The quantity $z_i(\mathbf{n})$ when $n_i = N_i$, the expected waiting time for a customer who chooses to join queue i when that queue is full, is undefined. For the purposes of $\hat{z}$-routing boundaries we do define a very similar quantity; the expected waiting time a customer would experience if they joined queue i in a state where $n_i = N_i$, so that from the perspective of the arriving customer the system would be of size $N_j \times (N_i + 1)$. We call this quantity hypothetical waiting time and denote it by $\hat{z}_i(\mathbf{n})$. When the system operates under $\hat{z}$-routing

boundaries the decision rule for states in $\mathcal{S}^e$ are calculated with the same function as the decision rule for other states but with $\hat{z}_i$ substituted for the waiting time of queue i.

Definition 4.1.8. Let a queuing system as the one described above operate under $\hat{z}$-routing boundaries, and let the decision rules for states $\mathbf{n} \in \mathcal{S}^-$ be $D_i(\mathbf{n}) = f_i(z_i(\mathbf{n}), z_j(\mathbf{n}))$ and let $\mathbf{n}' \in \mathcal{N}_i^e$ then

$$D_j(\mathbf{n}') = f_j(\hat{z}_i(\mathbf{n}'), z_j(\mathbf{n}'))$$

Clearly $\hat{z}$-routing boundaries is a middle ground between automatically routing boundaries and rejection boundaries in that joining the system in the edge states is still possible for the general arrivals, but they are no longer obliged to do so, and are given the opportunity to balk instead. In Example 4.1.5 we explicitly state the form that the decision policies take under $\hat{z}$-routing boundaries.

Example 4.1.5.

$$\mathbf{D}_1 = \begin{bmatrix} D_1(0,0) & D_1(0,1) & \cdots & D_1(0,N_2) \\ D_1(1,0) & \ddots & & \vdots \\ \vdots & & & \\ D_1(N_1-1,0) & \cdots & & D_1(N_1-1,N_2) \\ D_1(N_1-1,0) & \cdots & & D_1(N_1-1,N_2) \\ 0 & \cdots & & 0 \end{bmatrix},$$

$$\mathbf{D}_2 = \begin{bmatrix} D_2(0,0) & D_2(0,1) & \cdots & D_2(0,N_2-1) & 0 \\ D_2(1,0) & \ddots & & \vdots & \vdots \\ \vdots & & & & \\ D_2(N_1,0) & \cdots & & D_2(N_1,N_2-1) & 0 \end{bmatrix},$$

$$\mathbf{D}_0 = \begin{bmatrix} 0 & \cdots & 0 & D_0(0,N_2) \\ \vdots & \ddots & \vdots & \vdots \\ 0 & & 0 & D_0(N_1-1,N_2) \\ D_0(N_1,0) & \cdots & D_0(N_1,N_2-1) & 1 \end{bmatrix}.$$

4.1.6 Expected waiting time

As we saw already in example 4.1.1 it is sometimes useful to arrange expected waiting times for the system in matrices just as we do with the decision rules. These matrices are slightly different, though, and warrant a few words of description. The main point of difference between the decision policy matrices and the expected waiting time matrices is that the matrices containing the expected waiting times for queue 1 and queue 2 are not of the same dimensions. We see this by looking at examples of such matrices in a system of two queues, where $N_1 = 3$ and $N_2 = 4$, in Example 4.1.6, where we have dropped the indices referring to parameter vector and decision policy.

EXAMPLE 4.1.6.

$$\mathbf{Z}_{1,\mathbf{D}^t} = \begin{bmatrix} z_1(0,0) & z_1(0,1) & z_1(0,2) & z_1(0,3) & z_1(0,4) \\ z_1(1,0) & z_1(1,1) & z_1(1,2) & z_1(1,3) & z_1(1,4) \\ z_1(2,0) & z_1(2,1) & z_1(2,2) & z_1(2,3) & z_1(2,4) \end{bmatrix}$$

$$\mathbf{Z}_{2,\mathbf{D}^t} = \begin{bmatrix} z_2(0,0) & z_2(0,1) & z_2(0,2) & z_2(0,3) \\ z_2(1,0) & z_2(1,1) & z_2(1,2) & z_2(1,3) \\ z_2(2,0) & z_2(2,1) & z_2(2,2) & z_2(2,3) \\ z_2(3,0) & z_2(3,1) & z_2(3,2) & z_2(3,3) \end{bmatrix}.$$

The matrix containing the expected waiting times for queue 1 is of size 3×5, or $N_1 \times (N_2+1)$, while the one containing the waiting times for queue 2 is of size 4×4, or $(N_1+1) \times N_2$. The reason for this is that $z_i(n_1, n_2)$ is the expected waiting time for a marked customer who joins queue i when there are already n_1 users in queue 1 and n_2 users in queue 2. Therefore the quantities $z_1(N_1, n_2)$ and $z_2(n_1, N_2)$ are undefined, as there simply is no expected waiting time for a user who joins queue i when queue i is already full. However, in the context of $\hat{z}$-routing boundaries we extend the matrices with hypothetical waiting times, which makes them equal in size, this is discussed further in Section 4.1.6.

Calculating expected waiting times

Now the only part of the policy iteration algorithm we have yet to explain is step 5 in Algorithm 1, the calculation of expected waiting times. This step consists of

employing equation 2.5, and in the case of $\hat{z}$-routing boundaries as discussed below Equation 4.1.

We note that the expression in Equation 2.5 is valid regardless of the boundary condition employed, as the behaviour of general arrivals on the system boundaries is fully expressed in the decision policies. To see this we can for example consider the case when $\hat{z}$-routing boundaries are employed, so that some general arrivals do not in fact join the system, but instead balk, even though the system is in a state that would allow for additional users. This is reflected in Equation 2.5 by the factor $\sum_{j\in\mathcal{M}} D_j(\mathbf{n}+\mathbf{e}_i)$ which for $M = 2$ reduces to $D_1(\mathbf{n}+\mathbf{e}_i)+D_2(\mathbf{n}+\mathbf{e}_i)$ and adds to 1 when all general arrivals are accepted to the system, and 0 when all general arrivals are turned away or balk. In the case of a system with $\hat{z}$-routing boundaries in a boundary state this factor can take values in $(0, 1)$ and thus moderate the arrival rate. In the same way the boundary conditions are expressed by the decision rules in the terms representing arrivals to the system.

Hypothetical waiting times

When the system operates under $\hat{z}$-routing boundaries a marked customer is offered the chance to balk in the cases when they would under automatically routing boundaries be forced to join the only queue with room left in it. The decision to balk or not is based on the update function which takes as input two expected waiting times. However, as pointed out in the first part of this section, in a situation when $\mathbf{n}$ is such that $n_i = N_i$ but $n_j < N_j$ there is no expected waiting time for queue i to compare $z_j(\mathbf{n})$ to. This is where $\hat{z}$ comes into the picture. Consider Equation 4.1 which describes the *hypothetical waiting times* for a user joining queue i in a situation when $n_i = N_i$, under the assumption that the system will allow them, but no one else, to join queue i even though it is full.

$$\begin{aligned}\hat{z}_{i,\mathbf{D}}(\mathbf{n}) = \frac{1}{\sigma_j I_{\{n_j<N_j\}} + \lambda D_j(\mathbf{n}) + \mu_i + \mu_j I_{\{n_j>0\}}} \\ \times \Big(1 + I_{\{n_j<N_j\}}[\lambda D_j(\mathbf{n}) + \sigma_j]\hat{z}_{i,\mathbf{D}}(\mathbf{n}+\mathbf{e}_j) \\ + \mu_i \frac{N_i}{N_i+1} z_{i,\mathbf{D}}(\mathbf{n}-\mathbf{e}_i) + \mu_j I_{\{n_j>0\}}\hat{z}_{i,\mathbf{D}}(\mathbf{n}-\mathbf{e}_j)\Big), \\ \mathbf{n}\in\mathcal{S}^e, i\in\{1,2\}. \quad (4.1)\end{aligned}$$

This equation takes some explaining, even though it is very similar to the straightforward Equation 2.5.

We first note that we have made the assumption that the hypothetical extra room that has been made in queue i is not perceived by future general arrivals, so they follow decision rule $D_j(\mathbf{n})$ rather than $D_j(\mathbf{n}+\mathbf{e}_i)$. The pragmatic reason for this is obviously that $D_j(\mathbf{n}+\mathbf{e}_i)$ does not exist, as $\mathbf{n}+\mathbf{e}_i$ is not within of $\mathcal{S}$ in this case. However, we see that in the factor related to services in queue i, we take all the users, including the hypothetical one, into account.

We can understand this apparent discrepancy by considering more precisely which behaviours in the system the marked customer is considering. To do so we first note explicitly that this calculation is specific to $\hat{z}$-routing boundaries, and only takes place in the boundary state, so the factor $D_j(\mathbf{n})$ is not the probability of choosing queue j over queue i but the probability of choosing queue j over balking. So it describes the effective rate at which users join the system in the real edge state closest to the hypothetical state. As a contrast it is not clear what $D_j(\mathbf{n}+\mathbf{e}_i)$ would mean in a state when queue i is already full, or how this probability should be calculated without leading to infinite regress. Meanwhile, the factor $\frac{N_i}{N_i+1}$ associated with the μ_i-factor can be viewed as the reciprocal of the rate at which the marked customer is served. As the marked customer is considering the time that they would have to spend in the system had there been a spot for them in queue i it is reasonable that the rate at which they themselves would be served is based on there being $N_i + 1$ users in the system. So the discrepancy is largely pragmatically motivated in that the resulting expression conforms well to the idea behind the hypothetical waiting times, without making unwieldy assumptions about the behaviour of general arrivals in the hypothetical state of the system.

4.2 Numerical results

In this section we discuss certain system properties, and their relationship to system parameters and boundary conditions. We present these system properties by example, generally in the form of figures representing policy properties in various regions of state space. The figures presented here with relation to each property discussed are small, representative, subsets of what we explored in the course of this work. The choices we make with regards to what to include here have been

made on various grounds, and we try to be clear about our motivation. As a general rule the examples are chosen due to being particularly clear examples of some general tendency or structure.

With this work we sought to answer a set of interrelated questions about equilibria in policy iteration in this system. The fundamental question is whether any combination of update rule and boundary conditions leads, for any choice of system parameters, to equilibrium policies that conform to the intuitive properties of a decision policy in terms of monotonicity and lack of periodicity. For the situations in which fixed point policies either do not exist, or are non-monotonic, we are interested in understanding this behaviour. In particular we are interested in delineating the regions of parameter space that lead to the counter-intuitive results.

Instead of tackling the question of whether certain combinations of update rules and boundary conditions give rise to periodic and non-monotonic policies, and then examining the distribution of those policies in parameter space we start by exploring how different types of policy iteration outcomes are distributed in parameter space, and discuss the presence of problematic iteration outcomes from that starting point. The reason we chose to do it in that order is that we need to establish some properties of the parameter space distributions in order to discuss our conclusions with regards to the presence of periodicities and non-monotonicities.

4.2.1 Spatial distribution of policy properties - large parameter space region

The motivation behind these studies come largely from previous observations of the counter-intuitive nature of the outcomes of the policy iteration algorithm in this system, particularly the fact that both non-monotonic fixed point policies and periodic policies have been observed. So our first step in understanding this behaviour is to make sure it is reproducible. Furthermore it has previously been observed that the distribution of the potentially problematic policies in parameter space is not easily described. Therefore we first reproduce the previously obtained results, that were limited to $\{0, 1, \tilde{p}\}$-updates and automatically routing boundaries, and then go on to see if the same situation persists in the context of different update rules and boundary conditions.

In the following we often look at policy properties in terms of parameter space representations. This is partially due to convenience - looking at a two dimensional

cut through parameter space is an easy way to get an idea of the outcomes of policy iteration - and partially because it gives interesting results. While the combination of any two parameters would be possible as basis for a parameter space visualisation, we do in practice focus on certain parameters. In particular we focus on the parameters that are intrinsic to the system rather than parameters related particularly to the iterative process. So we are frequently interested in the distribution of the types of resulting policies in the μ_1-μ_2 plane but do not focus particularly on $\tilde{p}$ as a basis.

For practical reasons we also do not look at the distribution of types of resulting policies in the N_1-N_2 plane. It would be interesting to get an overview of how queue size affects monotonicity and periodicity for otherwise constant parameter sets. Unfortunately the computational complexity of the policy iteration grows in such a way with queue size that it is not practical to do this kind of calculation for the number of states that would be required in order to give an interesting parameter space picture in this case. However, we do look at some examples with increased, as well as varying, N_1 and N_2.

Validation of spatial distribution of policy properties

In Chen [45] a parameter space representation for a system of the kind discussed here under $\{0, 1, \tilde{p}\}$-updates with automatically routing boundaries is presented in Chapter 5, in particular in Figure 5.3. As a starting point for state space representations here we reproduce that figure, and examine the convergence and monotonicity properties for the same region of parameter space with different update rules and boundary conditions.

The parameters are $\mu_1 = 1, \sigma_1 = \sigma_2 = 0$, the queues are of equal size with $N_1 = N_2 = 3$, and the tie breaker probability is $\tilde{p} = 1$. We let λ vary from 0.083 to 50 and μ_2 vary from 0.0037 to 2.2 both in 600 steps of equal length. Each policy iteration is started from a policy of Join the Shortest Queue kind, with 0.5 tie-breakers and appropriate boundary conditions. This is an example of an initial policy with automatically routing boundaries.

$$\mathbf{D}_1^0 = \begin{bmatrix} 0.5 & 1 & 1 & 1 \\ 0 & 0.5 & 1 & 1 \\ 0 & 0 & 0.5 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

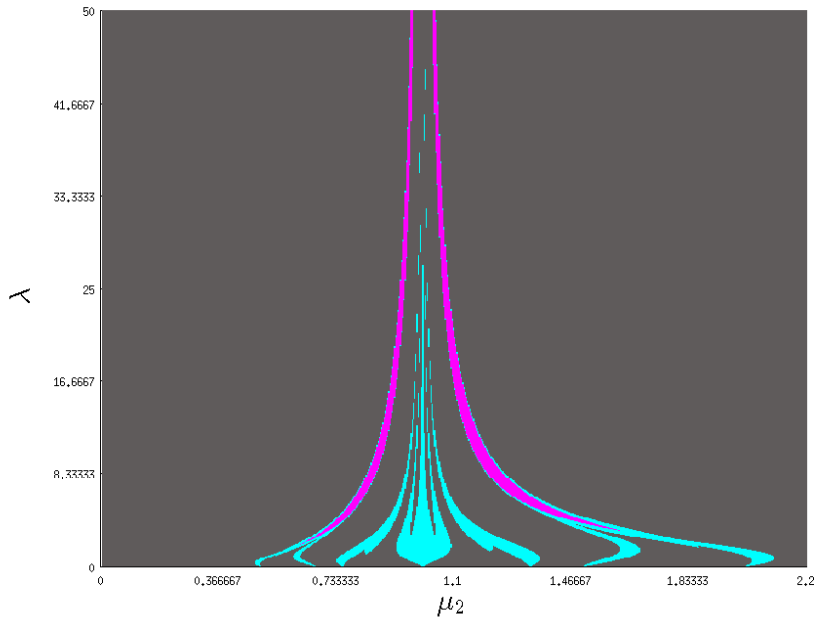


FIGURE 4.1: Distribution in parameter space of policy properties of resulting policies with $\mu_1 = 1, \sigma_1 = \sigma_2 = 0, N_1 = N_2 = 3, \tilde{p} = 1$, $\{0, 1, \tilde{p}\}$-updates and automatically routing boundaries. Parameter sets that give rise to monotonic aperiodic policies are represented as gray points, sets that give rise to non-monotonic fixed point policies are represented as pink and the sets of parameters that give rise to periodic policies are represented as cyan.

Figure 4.1 completely conforms to the findings in Chen [45] where it is discussed in some detail. The main thing to note is related to the shapes described by the different kind of iteration outcomes in parameter space, which were also noted in Chen [45]. First we note that the transition from one type of resulting policy to another is not a threshold type transition. That is, if we fix a particular value of λ, say 25, and let μ_2 increase from 0.0037 to 2.2 the outcome of the iteration will go from monotonic fixed point, to periodic policy, to non-monotonic fixed point, and then back to a monotonic policy again via another swathe of periodic policies. It is clear from Figure 4.1 that this behaviour is far less easily characterised even than

that description makes it seem. In fact the boundaries of the regions representing parameter sets with different outcomes of the iteration defies easy characterisation.

When considering this outcome it is tempting to at least consider that the outcome presented in Figure 4.1 on the preceding page is a result of numerical instabilities or other problems commonly associated with numerics. In that case it would be interesting only in as much as it related to the code that produced it, rather than the underlying theory that the code sought to represent. This potential objection is an important reason for the reproduction of this particular, striking, image. The implementation in this case is based on the theory described in Chen [45], but in no way on the code presented there. As an example we mention that the implementation was even done in different languages. The numerical exploration in Chen [45] was carried out in R while the exploration presented here took place in MATLAB. The fact that this striking image occurs with both implementations is an indication that the effect is inherent in the iterative algorithm, rather than a numerical artefact. This observation of successful reproduction by re-implementation from description of theory motivates us to also view the outcomes described in the following not as results of numerical instability but as actual properties of the system under consideration.

We note, before we go on to look at this same region of parameter space with different combinations of update rules and boundary conditions, that the choice of parameters for creating this figure is in general somewhat atypical from a system design perspective. In the majority of this region the system is severely overloaded, and will almost constantly be in a boundary state. However much of the interesting structure, in particular with regards to periodicity, takes place in the critical region where the system is either underloaded or just somewhat overloaded. To make this clear we take a look at a region of less extreme parameter choices, which can be found in Figure 4.2 on the next page, where we see that the interesting behaviour persists at a finer scale.

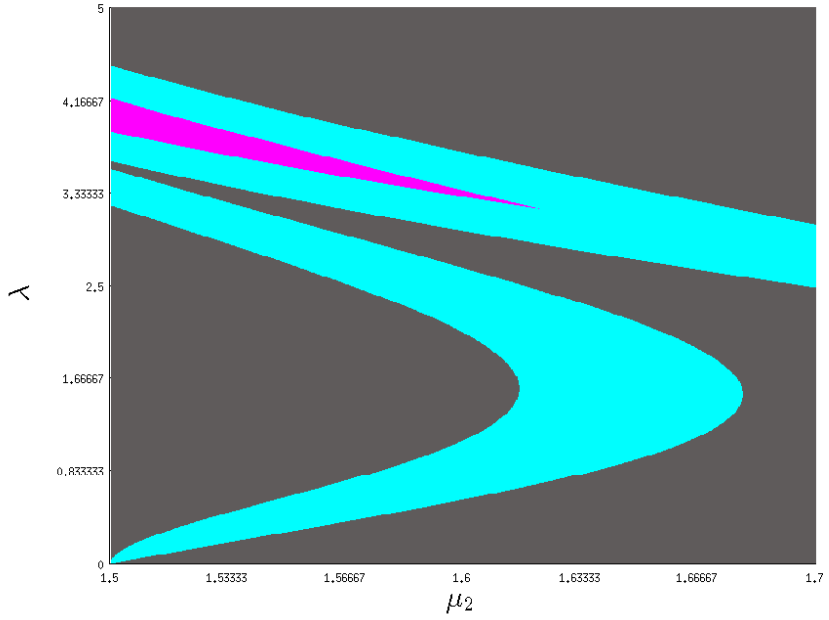


FIGURE 4.2: Magnified region of Figure 4.1 on page 103, with same total resolution as in that figure. The update rule and boundary condition is the same and the system parameters remain $\mu_1 = 1, \sigma_1 = \sigma_2 = 0, N_1 = N_2 = 3, \tilde{p} = 1$. Parameter sets that give rise to monotonic aperiodic policies are represented as gray points, sets that give rise to non-monotonic fixed point policies are represented as pink and the sets of parameters that give rise to periodic policies are represented as cyan.

Spatial distribution of policy properties with rejection boundaries

The question now is whether this behaviour persists, and if so in what form, when we change the boundary conditions or update rules.

We start by seeing what the same region of parameter space looks like if we change the boundary conditions from automatically routing boundaries to rejection boundaries, this is depicted in Figure 4.3 on the following page.

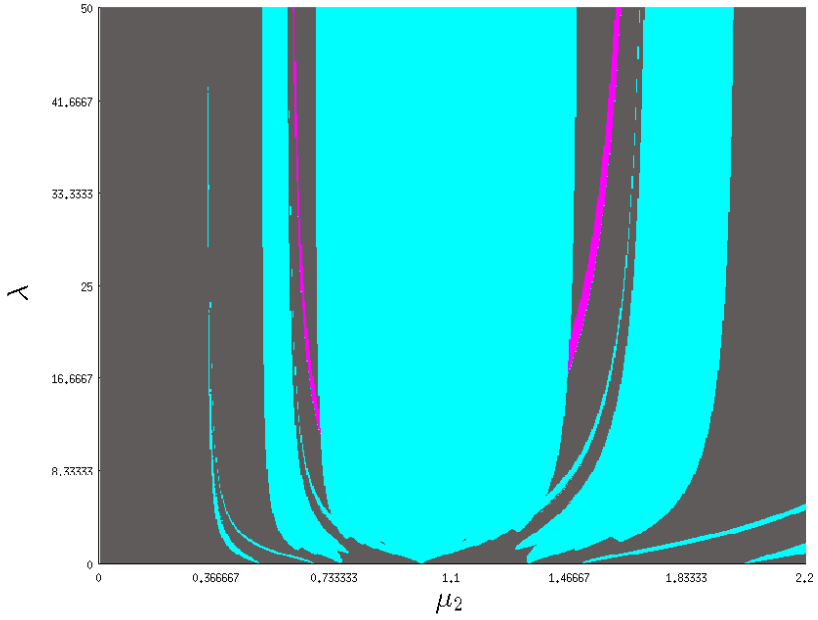


FIGURE 4.3: Parameter space distribution of policy properties of resulting policies with $\{0, 1, \tilde{p}\}$-updates and rejection boundaries. Parameter sets that give rise to monotonic aperiodic policies are represented as gray points, sets that give rise to non-monotonic fixed point policies are represented as pink and the sets of parameters that give rise to periodic policies are represented as cyan.

In essence Figure 4.3 and Figure 4.1 on page 103 have the same properties. In both cases there are instances of periodic policies as well as non-monotonic ones. Furthermore, in both cases the distribution of the parameter sets that lead to the different kinds of iterative outcomes make complex patterns on the $\mu_2\lambda$-plane. We can also see that there are some similarities between the patterns themselves in both cases, but the rejection boundaries, far from solving the problem of periodicity seems to give rise to more of it in this particular volume of state space and with this update rule.

Spatial distribution of policy properties with $\hat{z}$-routing boundaries

We stay in the same region of parameter space and again run policy iteration for the same set of parameter sets still with $\{0, 1, \tilde{p}\}$-updates but utilising $\hat{z}$-routing boundaries which leaves us with Figure 4.4.

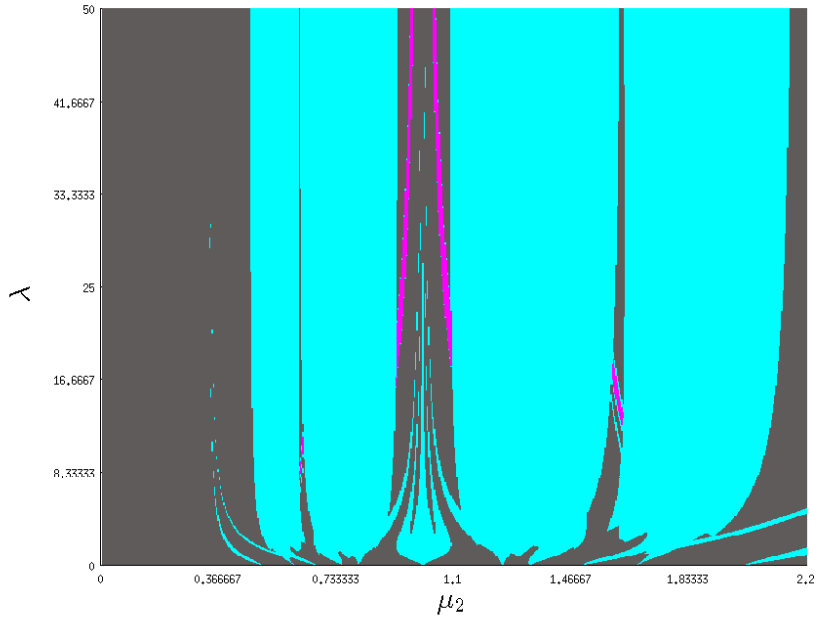


FIGURE 4.4: Parameter space distribution of policy properties of resulting policies with $\{0, 1, \tilde{p}\}$-updates and $\hat{z}$-routing boundaries. Parameter sets that give rise to monotonic aperiodic policies are represented as gray points, sets that give rise to non-monotonic fixed point policies are represented as pink and the sets of parameters that give rise to periodic policies are represented as cyan.

We again see the same basic behaviour as in Figure 4.1 on page 103, but with a slightly different overall pattern. As an aside we note that the $\hat{z}$-routing boundaries give rise to a pattern that visually has more in common with the pattern that emerges with automatically routing boundaries than the rejection boundaries did. In particular we see similar structures in the periodic policies around $\mu_2 = 1.1$ in

Figures 4.4 and 4.1. Likewise we note visual similarities between these two figures in terms of the non-monotonic fixed point policies around $\mu_2 = 1$ and $\lambda = 30$. We have not been able to formalise, or generalise, this affinity over other parts of parameter space, and it seems to be more an effect of these particular parameter choices than an actual effect of the boundary conditions themselves. In the same spirit we also note that the $\hat{z}$-routing boundaries and the rejection boundaries seem to produce outputs that are more visually similar to each other than either is to the output with automatically routing boundaries. Again, we haven't been able to formalise this or make any more general observations of that nature, and therefore have to hypothesise that it is an artefact of the particular part of parameter space we are looking at.

Conclusions from varying boundary conditions

From just these visualisations we can conclude that changing boundary conditions to one of the alternatives considered here is not sufficient to eliminate any of the problematic system behaviours that have previously been identified. With both of the new boundary conditions we still observed both non-monotonic and periodic policies and the types of policies are still distributed in a complex fashion in parameter space.

We now keep the automatically routing boundaries and change the update rule.

Spatial distribution of policy properties with $[0, 1]$-updates

We begin by using the $[0, 1]$-update rule with $\delta = 0.0001$, with the result that can be observed in Figure 4.5 on the next page.

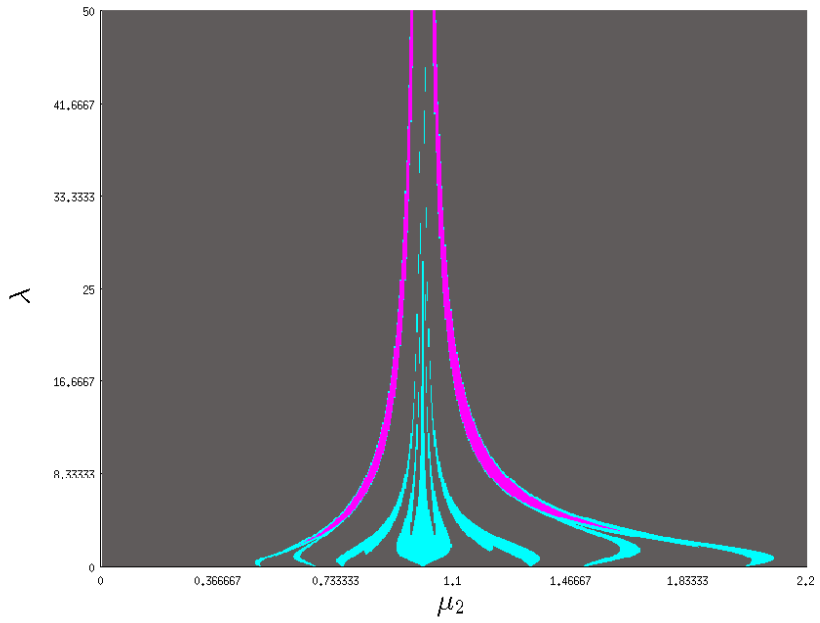


FIGURE 4.5: Parameter space distribution of policy properties of resulting policies with $[0, 1]$-updates with $\delta = 0.0001$ and automatically routing boundaries. Parameter sets that give rise to monotonic aperiodic policies are represented as gray points, sets that give rise to non-monotonic fixed point policies are represented as pink and the sets of parameters that give rise to periodic policies are represented as cyan.

Before considering the actual plot in Figure 4.5 we need to consider the effect of the choice of δ-parameter. This choice is not completely arbitrary, and does have an impact on the outcome. We have already noticed that as $\delta \to 0$ the $[0, 1]$-update goes to $\{0, 1, \tilde{p}\}$-updates with $\tilde{p} = 0.5$, and the greater δ becomes the more different the two update rules become. Unsurprisingly a greater δ also gives rise to greatly increased computation times, due to an increased number of steps until convergence. The number of steps until convergence also has a strong dependence on the other parameters in question, and the difference between number of steps until convergence for similar sets of parameters can be quite large. Therefore the

choice of δ in this case was to some extent based on practicality of executing the relevant calculations.

The choice of δ here is such that the outcomes with $[0, 1]$-updates and $\{0, 1, \tilde{p}\}$-updates look identical, and in fact the difference is quite small. What is relevant for our purposes, though, is whether the difference between the outcomes is in the direction of improving the situation, in terms of eliminating potentially undesirable resulting policies. We explore this by looking at the difference between Figure 4.5 on the preceding page and Figure 4.1 on page 103, which is shown in Figure 4.6 on Page 110. The only differences between the two are that some are periodic in one, but not the other and, which is crucial here, vice versa. There was no difference in terms of non-monotonicities.

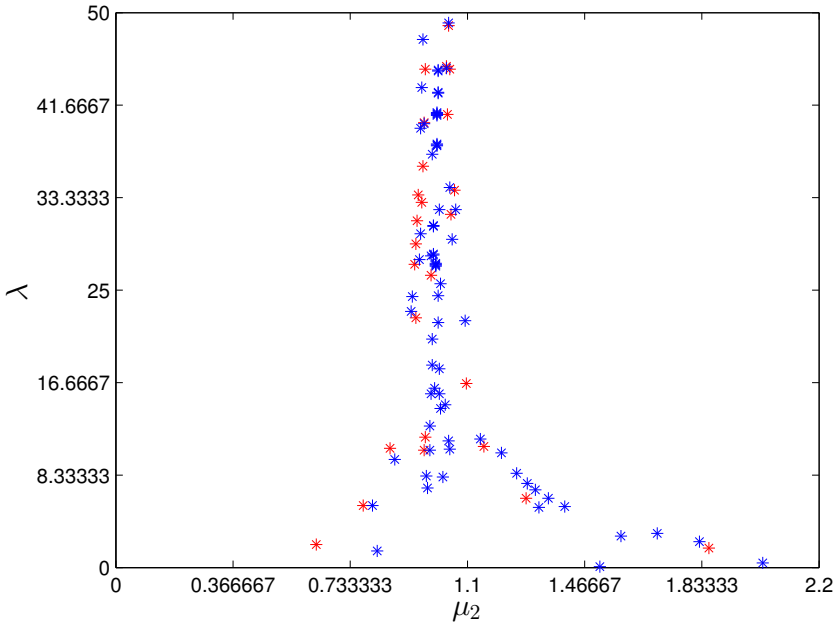
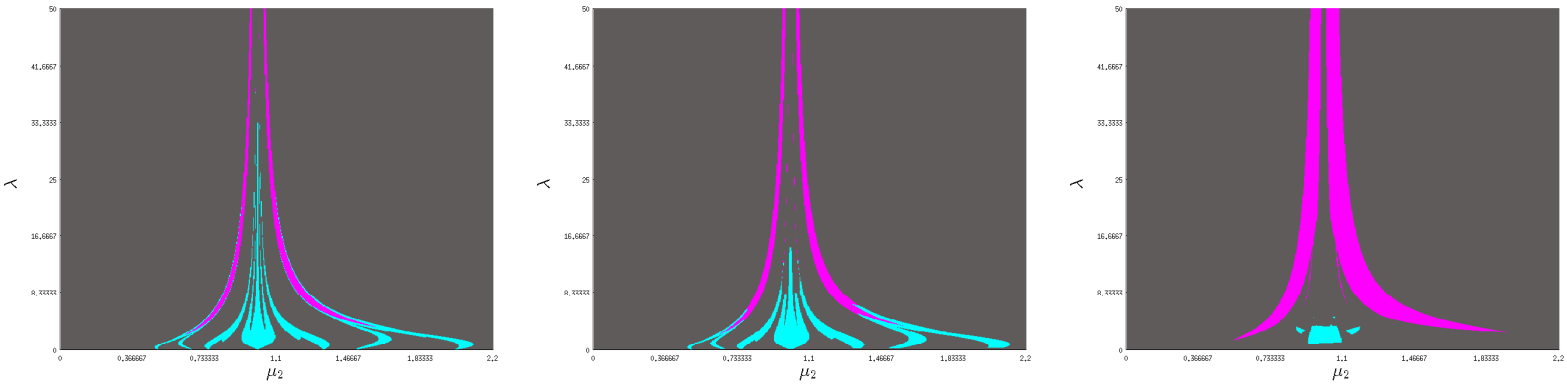


FIGURE 4.6: Points of difference between Figure 4.5 on the preceding page and Figure 4.1 on page 103. A red point marks a set of parameters that results in periodic behaviour with $\{0, 1, \tilde{p}\}$-updates but not with $[0, 1]$-updates, a blue point marks a set of parameters that result in periodic behaviour with $[0, 1]$-updates but not with $\{0, 1, \tilde{p}\}$-updates,

The point of looking at Figure 4.6 on the facing page is that we have no indication that $[0, 1]$-updates monotonically eliminate periodicities, in fact we see that it does not at a δ level that clearly does produce differences.

It is of course interesting to see what happens as we let δ increase progressively, so we produce a visualisation of the policy outcomes for the same parameter sets for three progressively larger values of δ.



(A) $\delta = 0.001$ (B) $\delta = 0.01$ (C) $\delta = 0.1$

FIGURE 4.7: Reproduction of Figure 4.5 on page 109 for three different values of δ.

We see in Figure 4.7 that as δ increases the distribution of resulting policies becomes increasingly different from what we saw in Figure 4.1 on page 103. We still see that, just as illustrated by Figure 4.6 on the preceding page, the departure from the structure we saw with automatic routing boundaries and $\{0, 1, \tilde{p}\}$-updates is not purely, or even mainly, in the direction of unproblematic policies and policy distributions in parameter space. There is, however, a clear tendency in these realisations for a decreased incidence of periodicity and an increased propensity for non-monotonic policies as δ increases.

Policy property patterns with $(0, 1)$-updates

We repeat the procedure with $(0, 1)$-updates and automatically routing boundaries. The, somewhat pointless, figure resulting from this operation can be found in

Figure 4.8.

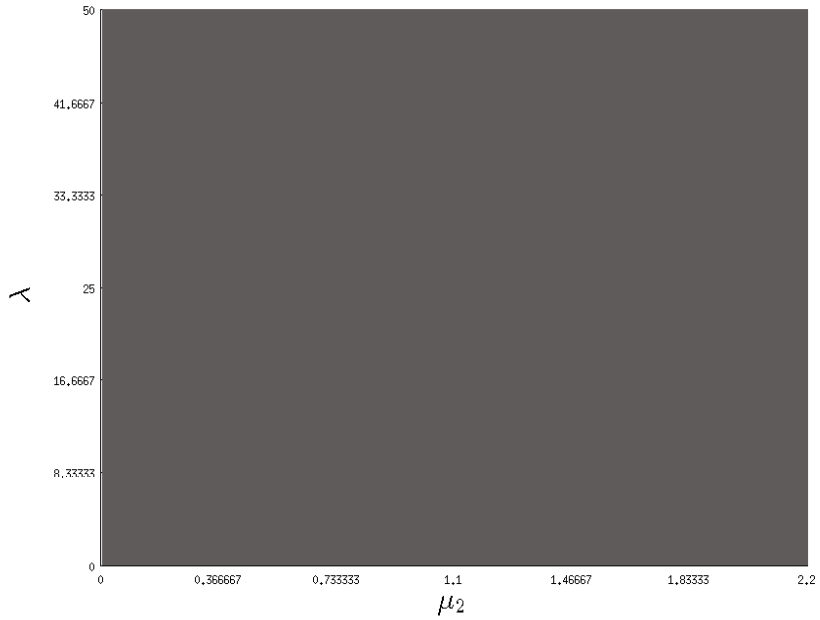


FIGURE 4.8: Parameter space distribution of policy properties of resulting policies with $(0, 1)$-updates and automatically routing boundaries. Parameter sets that give rise to monotonic aperiodic policies are represented as gray points.

This figure makes it clear that none of the problematic behaviour from previous experiences is present. Policy iteration with any of the sets of parameters in question lead to monotonic fixed point policies, so there can obviously not be any complex patterns in parameter space either. There is no need to show the outcome for experiments with $(0, 1)$-updates and the two other boundary conditions, as the pictures are identical to that shown in Figure 4.8. The $(0, 1)$-update rule does not result in any periodic or non-monotonic behaviour for this set of parameters, regardless of the choice of boundary condition.

It is worth stating clearly here that given what we already know about the potentially complex patterns of problematic sets of parameters this outcome merely

shows that this update rule does not produce any non-monotonic fixed point policies or any periodic behaviour in the iteration for the particular sets of parameters tested here.

Conclusions from reproduction and extensions

We can draw some conclusions from these tests. We have shown, by counter-example, that none of the boundary conditions we set out to test guarantee monotonic fixed point policies, or indeed fixed point policies at all, for arbitrary update rules. By counter-example we also showed that policy iteration with the $[0, 1]$-update rule with $\delta > 0$ and arbitrary boundary condition is not guaranteed to result in fixed point policies, and that the fixed point policies that it does lead to are not guaranteed to be monotonic.

We also have no indication so far that policy iteration with $(0, 1)$-updates gives rise to non-monotonic fixed point policies, or fails to converge. This indication is particularly strong as we have tested it in a volume of parameter space that does give rise to problems in varied ways with other update rules, regardless of boundary conditions. The fact that we ran the same calculation for the same region with the other boundary conditions with the same result adds to the strength of the indication.

Though less precise data, we also found in making these plots that in the context of these three update rules $[0, 1]$-updates are prohibitively difficult to work with. In particular; for this region of parameter space it offered, as we saw, no appreciable advantages over $\{0, 1, \tilde{p}\}$-updates as we could not easily pick a δ such that the iteration converged for all 360000 parameter vectors in a reasonable numbers of steps. This is not to say that we found examples of sets of parameters that did not converge, but rather to say that the relationship between parameter set, choice of δ and time to convergence was too complex on the face of it to merit further study when the $(0, 1)$-update rule offered a much more promising path forward.

4.2.2 Policy property spatial distribution observations in the service rate plane

Due to the large number of parameters that could possibly have an impact on system behaviour, and the awareness that system behaviour changes in complex

ways in parameter space, it is not possible to do anything like an exhaustive study of possible combinations of parameters, update rules and boundary conditions. Instead we have to be satisfied with exploring a few especially interesting sets of parameters and rules for the system.

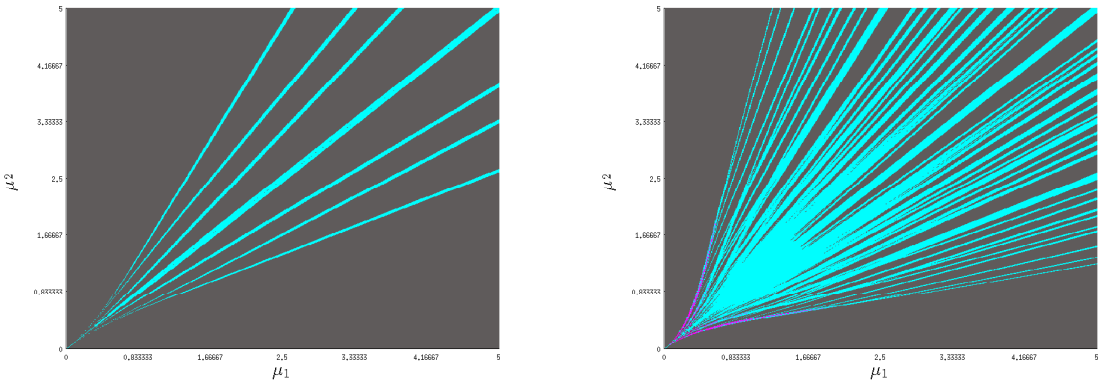
We focus on regions of parameter space spanned by either service rates or intrinsic arrival rates. The reason for this is twofold. On the one hand it is easy to form an intuitive understanding of what traversing such a region would mean. On the other hand it is easy to let it span over regions of parameter space that are intrinsically interesting, such as underloaded-overloaded or symmetrical-asymmetrical.

The fact that we are working with a maximum queue length is largely due to the fact that it makes it easy to solve the expected waiting time equations exactly numerically, even if the results regarding limited maximum length queues are interesting and relevant in and of themselves. However, it is interesting to see how maximum queue length affects the distribution of types of resulting policies. It is not practical to do something similar to the above with maximum queue lengths on the x- and y-axes. We can however look at the distribution of resulting policies for some region of parameter space for different values of maximum queue length to get an idea of whether these parameters affect the type of resulting policy.

Non-monotonicities and periodicity with $\{0, 1, \tilde{p}\}$-updates

In Figure 4.9 on the facing page we see the resulting types of policies for two different regions of parameter space. In both cases μ_1 varies from 0.0083 to 5 in 600 steps of equal length on the x-axis and μ_2 does likewise on the y-axis. In both cases $\lambda = 1$, $\sigma_1 = \sigma_2 = 0.45$ and $\tilde{p} = 0.5$. The difference between the two representations is that on the left $N_1 = N_2 = 3$ while on the right $N_1 = N_2 = 8$.

There are very clear differences between the two representations in Figure 4.9 on the next page, the most striking difference perhaps being the severely increased presence of periodic policies in the case of the situation with increased availability of spots in the queue. We also note that when there is only room for a maximum of three people in each queue we don't see any evidence of non-monotonic policies for any of these parameter sets, while the system with eight spots in each queue gives rise to non-monotonic resulting policies, in particular in the overloaded region close to the origin. On the other hand we see that the basic structure is maintained through the increase in capacity. In both cases the image shows a background of



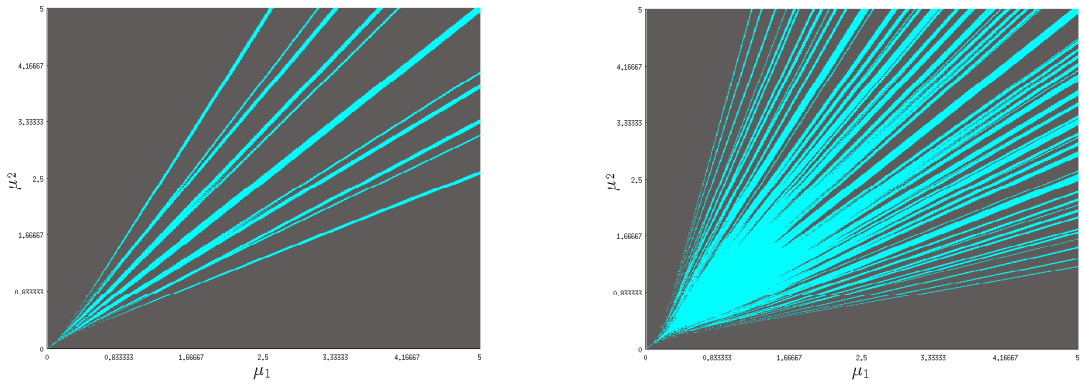
(A) $N_1 = N_2 = 3$ (B) $N_1 = N_2 = 8$

FIGURE 4.9: Parameter space distribution of types of resulting policies with automatically routing boundaries and $\{0, 1, \tilde{p}\}$-updates for two different symmetrical values of capacity. In both cases $\lambda = 1$, $\sigma_1 = \sigma_2 = 0.45$ and $\tilde{p} = 0.5$. Parameter sets that give rise to monotonic aperiodic policies are represented as gray points, sets that give rise to non-monotonic fixed point policies are represented as pink and the sets of parameters that give rise to periodic policies are represented as cyan.

monotonic fixed point policies broken up by swathes of periodic policies seemingly projecting out from the origin.

When we repeat the same experiment, with $\hat{z}$-routing instead of automatically routing boundaries we get the result we can see in Figure 4.10 on the following page.

The results in Figure Figure 4.10 on the next page are remarkably similar to those in Figure 4.9, but also have some noticeable differences. The similarities are most obvious in the case of $N_1 = N_2 = 3$ where the two visualisations only differ in four additional spokes of periodicity in the $\hat{z}$-routing case compared to the automatically routing case. In terms of differences we notice that the patterns of periodicities are very similar, but the non-monotonicies are absent in the case of $\hat{z}$-routing boundaries.



(A) $N_1 = N_2 = 3$ (B) $N_1 = N_2 = 8$

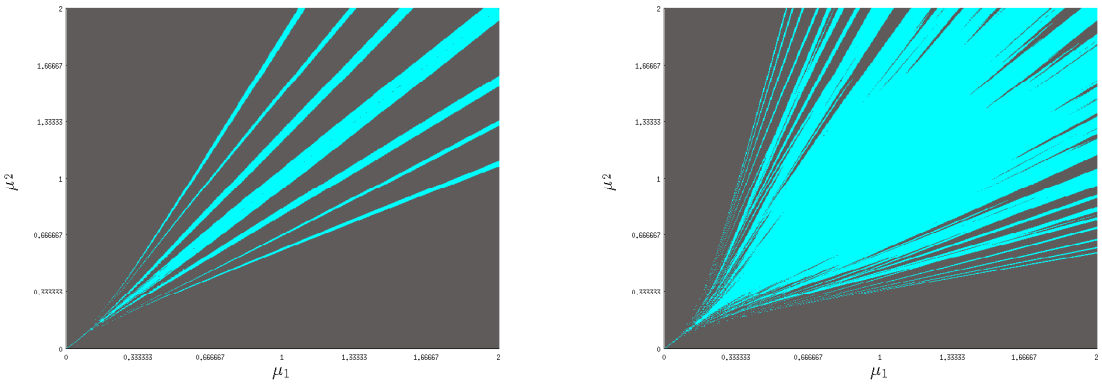
FIGURE 4.10: Parameter space distribution of types of resulting policies with $\hat{z}$-routing and $\{0, 1, \tilde{p}\}$-updates for two different symmetrical values of capacity. In both cases $\lambda = 1$, $\sigma_1 = \sigma_2 = 0.45$ and $\tilde{p} = 0.5$. Parameter sets that give rise to monotonic aperiodic policies are represented as gray points and the sets of parameters that give rise to periodic policies are represented as cyan.

In the interest of completeness we look at the same situation in the rejection boundary case as well, this can be found in Figure 4.11 on the next page.

In Figure 4.11 we see a pattern of the same kind as the two previous cases. However, we see here that as in the case with $\hat{z}$-routing we don't see any non-monotonic policies. On the other hand we see more instances of iterations that fail to converge except to a periodic orbit than we did in that case.

Just as in Section 4.2.1 we note here that while the boundary conditions do have an impact on the type of resulting policy we find for a set of parameters, none of the boundary conditions utilised here can guarantee monotonic fixed point policies for all parameter sets.

We can, however, make some kind of distinction between the boundary conditions by invoking parsimony. In particular we note that $\hat{z}$-routing boundaries are in every sense more complex than the other two boundary conditions. Not only is the complexity higher in terms of the decisions that have to be made in order to



(A) $N_1 = N_2 = 3$ (B) $N_1 = N_2 = 8$

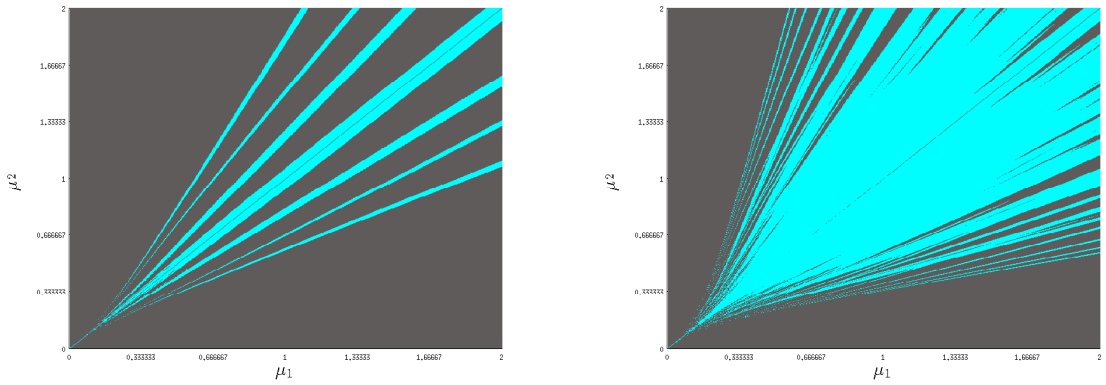
FIGURE 4.11: Parameter space distribution of types of resulting policies with rejection routing and $\{0, 1, \tilde{p}\}$-updates for two different symmetrical values of capacity. In the left figure $N_1 = N_2 = 3$ and to the right $N_1 = N_2 = 8$. In both cases $\lambda = 1$, $\sigma_1 = \sigma_2 = 0.45$ and $\tilde{p} = 0.5$. Parameter sets that give rise to monotonic aperiodic policies are represented as gray points and sets of parameters that give rise to periodic policies are represented as cyan.

just define the method, but also in terms of computational complexity it far outweighs the other types of boundary conditions. Had these, or other, experiments with the policy iteration method shown that the increased complexity resulted in a more robust system, it would be worth exploring this kind of boundary condition further. We, however, draw the conclusion here that the added complexity is not balanced by improved system performance.

Periodicities and effect of δ with $[0, 1]$-updates

We are also interested in how the other update rules behave under a change of maximum queue length. We start by looking at $[0, 1]$-updates with $\delta = 0.001$ with rejection boundaries in Figure 4.12 on the following page.

This figure is essentially identical to Figure 4.11. The main difference of interest can be found along the $\mu_1 = \mu_2$ line in both the $N_1 = N_2 = 3$ and $N_1 = N_2 = 8$ case. With $\{0, 1, \tilde{p}\}$-updates the policies along this line are generally periodic,



(A) $N_1 = N_2 = 3$ (B) $N_1 = N_2 = 8$

FIGURE 4.12: Parameter space distribution of types of resulting policies with rejection boundaries and $[0, 1]$-updates with $\delta = 0.001$ for two different symmetrical values of capacity. In both cases $\lambda = 1$ and $\sigma_1 = \sigma_2 = 0.45$. Parameter sets that give rise to monotonic aperiodic policies are represented as gray points, sets of parameters that give rise to periodic policies are represented as cyan.

while in the case of $[0, 1]$-updates the increased flexibility of the update rule means that we do in fact see policy convergence along this line. Other than this small difference the two outcomes are essentially identical.

To get a fuller understanding of what a periodic policy means in this case, we take a look at a specific realisation of policy iteration leading to a periodic orbit in the region of parameter space depicted in Figure 4.12. We show this in Example 4.2.1. We look at the situation when $N_1 = N_2 = 3$ as it is much easier to get an overview of the policies in that situation.

EXAMPLE 4.2.1. *Let $\lambda = 1$, $\sigma_1 = \sigma_2 = 0.45$, $N_1 = N_2 = 3$ so that the parameter vector is somewhere in the region shown in the left hand side plot in Figure 4.12. Also let $\mu_1 = 1.5$ and $\mu_2 = 1$ so that the parameter vector in question is taken from the cyan coloured field second furthest to the right and bottom in that figure. Then start policy iteration with $[0, 1]$ updates with $\delta = 0.001$, and rejection boundaries*

starting from

$$\mathbf{D}_1^0 = \begin{bmatrix} 0.5 & 1 & 1 & 0 \\ 0 & 0.5 & 1 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Then the policy iteration leads to the following sequence of policies

$$\mathbf{D}_1^1 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{D}_1^2 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{D}_1^3 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{D}_1^4 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

We can see clearly here that $\mathbf{D}_1^2 = \mathbf{D}_1^4$ *which means that without doing any calculations we can state that* $\mathbf{D}_1^5 = \mathbf{D}_1^3$ *and* $\mathbf{D}_1^6 = \mathbf{D}_1^2$ *et cetera, and that the iteration will never converge.*

It is tempting to take Example 4.2.1 to mean that with a sufficiently high value of δ we would find an aperiodic decision policy which lies between the two policies in the periodic orbits. To some extent this assumption holds, and we explore it in Example 4.2.2.

EXAMPLE 4.2.2. *Let the parameters except δ be the same as in Example 4.2.1, let $\delta = 0.1$. Now the policy is no longer periodic, but we instead find the monotonic fixed point policy*

$$\mathbf{D}_1^* = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0.46 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

this does indeed fall between the policies in the periodic orbit. It is now interesting to ask what happens to the decision rules if we let δ increase from the current value. In order to examine this we fix the parameters, except for δ and let δ go from 0.1 to 6 in 100 steps of equal size, and plot how each decision rule develops as δ changes. We find the outcome of this experiment in Figure 4.13.

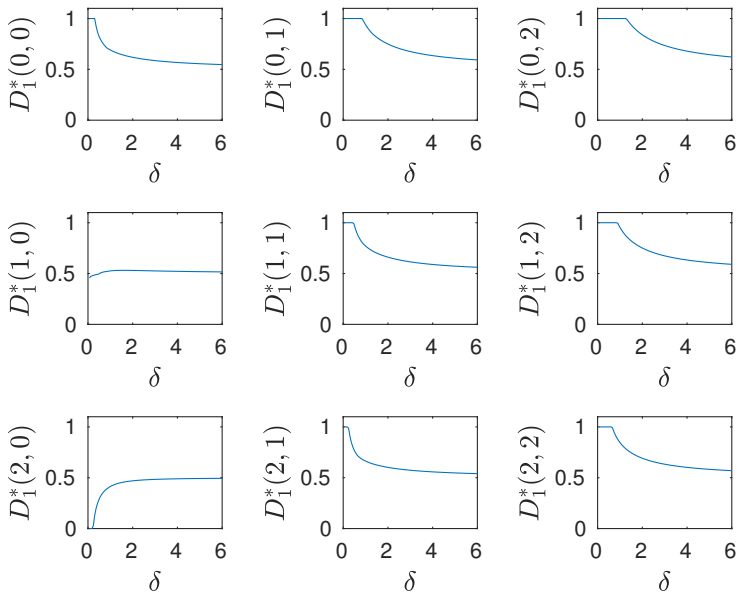


FIGURE 4.13: Each subplot represents the development of the decision rule in the designated state as δ varies from 0.1 to 6 in 100 steps of equal length. The remaining parameters are $\lambda = 1$, $\sigma_1 = \sigma_2 = 0.45$, $N_1 = N_2 = 3$, $\mu_1 = 2.9$ and $\mu_2 = 1.55$.

It is interesting to note in Figure 4.13 on the preceding page the strong effect that the value of δ has on the outcome of policy iteration. In particular we remind ourselves that $[0,1]$-updates were defined to act as a version of $\{0,1,\tilde{p}\}$-updates with increased flexibility in dealing with similar expected waiting times. We note that for values close to $\delta = 0.1$ this indeed seems to be the behaviour.

As δ increases we see that, as expected, the policy eventually becomes totally mixed. Somewhat less obviously at first glance, all of the decision rules cluster close to equal probability of joining either queue rather than remaining closer to the pure policy of $\delta = 0.1$.

It seems that the decision policies do converge to some value as we increase the value of delta. For $\delta = 6$ the decision policy is

$$\mathbf{D}_1^* = \begin{bmatrix} 0.547 & 0.592 & 0.620 & 0 \\ 0.516 & 0.561 & 0.590 & 0 \\ 0.494 & 0.540 & 0.569 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

For an even higher value of δ, $\delta = 100$ the policy is given by

$$\mathbf{D}_1^* = \begin{bmatrix} 0.502 & 0.504 & 0.505 & 0 \\ 0.501 & 0.503 & 0.504 & 0 \\ 0.501 & 0.502 & 0.504 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

while at this level of precision with $\delta = 10000$ we find that all decision rules are equal to 0.5. The choice of δ affects the outcome substantially, and it is not trivially obvious how an optimal δ should be chosen. For comparison we look at the decision policy for the same set of parameters with $\{0,1,\tilde{p}\}$-updates and $(0,1)$-updates.

With $\{0,1,\tilde{p}\}$-updates the iteration does not converge for any of the 100000 values of $\tilde{p}$ equally spaced over $[0,1]$ that we tried. With $(0,1)$-updates we found the following decision policy

$$\mathbf{D}_1^* = \begin{bmatrix} 0.61 & 0.68 & 0.72 & 0 \\ 0.53 & 0.61 & 0.65 & 0 \\ 0.48 & 0.56 & 0.61 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Non-monotonic structure with $(0,1)$-updates

As mentioned above we haven't observed any instances of non-monotonic fixed point policies or failures to converge for the policy iteration algorithm in the context of $(0,1)$-updates when keeping capacity constant at $N_1 = N_2 = 3$ and considering a variety of parts of parameter space in terms of service and arrival rates. There is one type of parameter we have yet to explore in terms of its relation to the outcome of policy iteration – system capacity.

In the context of capacity we have found it impractical to consider policies in the $N_1 - N_2$-plane, so we are instead left with the option of considering some familiar part of parameter space in the context of varying capacity. We have found that there are interesting behaviours to be gleaned from considering the $\mu_1 - \mu_2$-plane. As opposed to the situation with the other update rules we have studied here we have not observed any occurrences of periodicities in the context of $(0,1)$-updates, but it turns out that it is possible to find non-monotonic policies by increasing the capacity of just one of the queues by 1 to $N_i = 4$. We show this in, by now familiar, state space representation in Figure 4.14.

We note that in this case the non-monotonic fixed point policies occur deep in the overloaded region, and can justifiably be thought of as occupying a small region of parameter space. However, we see below that as we increase the capacity of the system, the non-monotonic fixed point policies start to fill up the underloaded region, and move out beyond it. Before we go on to consider these volumes of parameter space, we introduce the concept of non-monotonic structure.

Using Figure 4.14 as our starting point we go on to discuss non-monotonic structure. We have chosen to carry out this discussion in the context of the $(0,1)$-policies even though it can, due to the mixed rules, be somewhat less easy to immediately see the non-monotonicities in these than other types of policies. The reason for this choice is that the observed guaranteed convergence makes them the most interesting case to continue studying, and thus the most interesting case to try to understand in detail.

We start by considering Figure 4.14 on the facing page and in particular three of the non-monotonic policies. In Examples 4.2.3, 4.2.4 and 4.2.5 we represent the

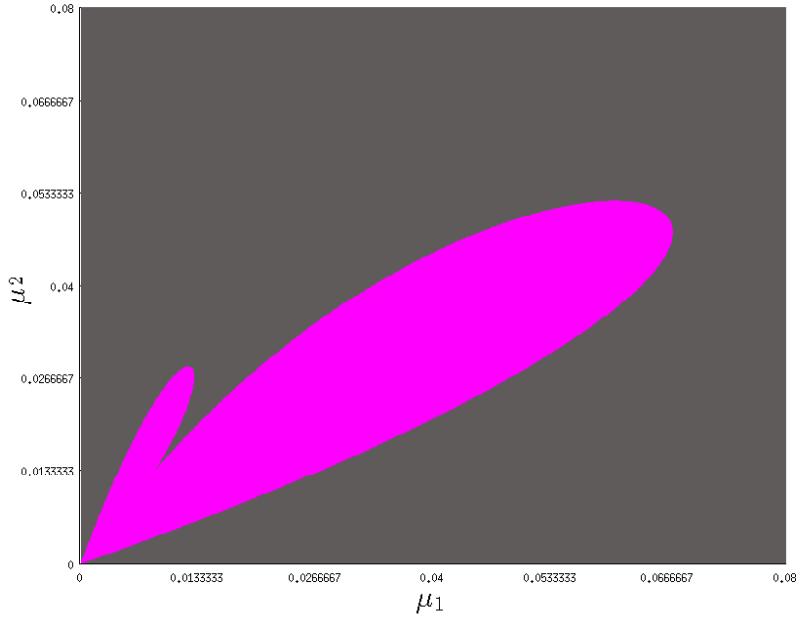


FIGURE 4.14: Parameter space distribution of types of resulting policies with automatically routing boundaries and $(0,1)$-updates. The parameters are $\lambda = 1$, $\sigma_1 = \sigma_2 = 0$, $N_1 = 3, N_2 = 4$. Parameter sets that give rise to monotonic aperiodic policies are represented as gray points, sets that give rise to non-monotonic fixed point policies are represented as pink.

fixed point policies for policy iteration with $(0,1)$-updates and automatically routing boundaries for specified parameter vectors. Without losing any information we characterise each decision policy by the decision rules for Q_1.

EXAMPLE 4.2.3. *For parameter vector*

$$\mathbf{\Gamma_1} = (\lambda = 1, \mu_1 = 9.2 \cdot 10^{-3}, \mu_2 = 2 \cdot 10^{-2}, \sigma_1 = 0, \sigma_2 = 0, N_1 = 3, N_2 = 4),$$

we have

$$\mathbf{D}^*_{1,\boldsymbol{\Gamma_1}} = \begin{bmatrix} 0.376031 & 0.381065 & 0.382509 & \mathbf{0.382497} & 1 \\ 0.373122 & 0.378130 & 0.379826 & 0.380299 & 1 \\ 0.373002 & 0.377555 & 0.379150 & 0.379704 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

EXAMPLE 4.2.4. *For parameter vector*

$$\boldsymbol{\Gamma_2} = (\lambda = 1, \mu_1 = 1.36 \cdot 10^{-2}, \mu_2 = 1.08 \cdot 10^{-2}, \sigma_1 = 0, \sigma_2 = 0, N_1 = 3, N_2 = 4),$$

we have

$$\mathbf{D}^*_{1,\boldsymbol{\Gamma_2}} = \begin{bmatrix} 0.625587 & 0.629382 & 0.630577 & 0.630687 & 1 \\ 0.622965 & 0.626380 & 0.627542 & 0.627897 & 1 \\ \mathbf{0.623060} & 0.625852 & 0.626812 & 0.627159 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

EXAMPLE 4.2.5. *For parameter vector*

$$\boldsymbol{\Gamma_3} = (\lambda = 1, \mu_1 = 2.4 \cdot 10^{-3}, \mu_2 = 4 \cdot 10^{-3}, \sigma_1 = 0, \sigma_2 = 0, N_1 = 3, N_2 = 4),$$

we have

$$\mathbf{D}^*_{1,\boldsymbol{\Gamma_3}} = \begin{bmatrix} 0.443627 & 0.444808 & 0.445137 & \mathbf{0.445132} & 1 \\ 0.442923 & 0.444068 & 0.444444 & 0.444549 & 1 \\ \mathbf{0.442926} & 0.443934 & 0.444275 & 0.444395 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

We have chosen the three parameter vectors from different regions of Figure 4.14 on the previous page. $\boldsymbol{\Gamma_1}$ denotes a point in the upper, smaller, of the two lobes represented by the region of non-monotonic fixed point policies, $\boldsymbol{\Gamma_2}$ is a point in the lower, larger, lobe and $\boldsymbol{\Gamma_3}$ denotes a point in the part of the non-monotonic region closer to the origin than the bi-lobed structure. In each of the policies we have marked the decision rules that violate our expectations of policy monotonicity as we proceed from lower to higher occupancy by making it bold. For example, by monotonicity we would expect $D^*_{1,\boldsymbol{\Gamma_3}}(0,2) > D^*_{1,\boldsymbol{\Gamma_3}}(0,1)$ and $D^*_{1,\boldsymbol{\Gamma_3}}(0,3) > D^*_{1,\boldsymbol{\Gamma_3}}(0,2)$, and while it is indeed the case that $D^*_{1,\boldsymbol{\Gamma_3}}(0,2) > D^*_{1,\boldsymbol{\Gamma_3}}(0,1)$ we find that $D^*_{1,\boldsymbol{\Gamma_3}}(0,3) < D^*_{1,\boldsymbol{\Gamma_3}}(0,2)$ thus we can justifiably think of

$D^*_{1,\mathbf{\Gamma_3}}(0,3)$ as the offending rule, which we indicate by writing it in bold in the representation of the policy.

The reason for considering these particular policies is that they together represent all the types of policies that we have identified in the part of parameter space depicted in Figure 4.14. In all fixed point policies for parameter sets contained in the upper lobe the non-monotonicity occurs in exactly the same way as in Example 4.2.3 in that the offending rule is $D^*_{1,\mathbf{\Gamma_1}}(0,3)$. In all the cases we have considered where the parameter vector falls somewhere in the lower lobe we see the same type of monotonicity as that shown in Example 4.2.4 in that the offending rule is $D^*_{1,\mathbf{\Gamma_2}}(2,0)$. Finally we have noted that in the region where the lobes intersect we find a set of intermediate policies which have the form seen in Example 4.2.5. In this case we have two offending rules, one in either direction, $D^*_{1,\mathbf{\Gamma_3}}(0,3)$ as well as $D^*_{1,\mathbf{\Gamma_3}}(2,0)$.

In fact, we can illustrate this in a plot that shows how the structure of non-monotonicity relates to the service rates. In Figure 4.15 on the following page we have given each type of non-monotonicity a different colour, and we can clearly see the three different regions represented by Examples 4.2.3, 4.2.4 and 4.2.5 in purple, blue and red respectively.

These examples, particularly in conjunction with the plot, is yet another clear indication that the non-monotonicities are more than computational anomalies. They indicate that there is a level of structure beyond the fact that they only occur in certain regions.

As we let the capacity of the system increase so do the types of non-monotonicities that the system exhibits. If we let the capacity of each queue increase by 2 so that $N_1 = 5$ and $N_2 = 6$ we obtain the plot in Figure 4.16 on page 127.

We note that while this plot is quite different from Figure 4.15 on the following page there are some clear similarities, so we explore these. We could describe Figure 4.15 as consisting of two regions with distinct non-monotonicity structures that overlapped in a region where the non-monotonicity was of a type that could justifiably be described as a combination, or intermediate, between the two others. An obvious question is whether Figure 4.16 has the same structure. We explore this question in Example 4.2.6.

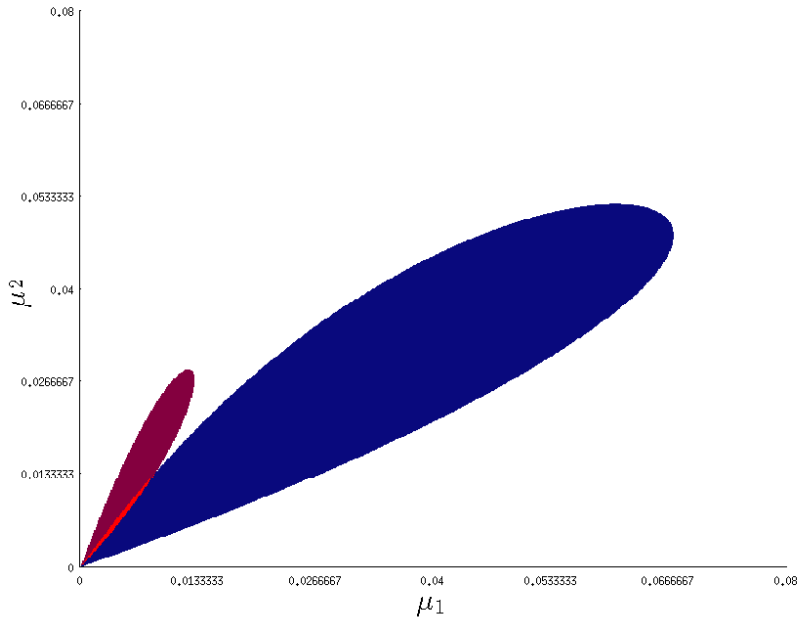


FIGURE 4.15: Parameter space distribution of types of non-monotonic policies with automatically routing boundaries and $(0,1)$-updates. The parameters are $\lambda = 1$, $\sigma_1 = 0, \sigma_2 = 0$, $N_1 = 3, N_2 = 4$. Each μ_i varies from $1.3 \cdot 10^{-4}$ to 0.08 in 600 steps of equal length. Parameter sets that give rise to monotonic policies are represented as white points, the parameter sets that give rise to non-monotonic policies with the same structure are represented by the same colour.

EXAMPLE 4.2.6. *Consider a vertical line in Figure 4.16 at* $\mu_1 = 0.21$ *and sample one non-monotonic policy for each region we traverse as we follow this line in the positive* μ_2 *direction.*

The first region of non-monotonic policies is the dark blue one in which we find the parameter vector $\mathbf{\Gamma_1} = (\lambda = 1, \mu_1 = 0.21, \mu_2 = 0.03, \sigma_1 = 0, \sigma_2 = 0, N_1 =$

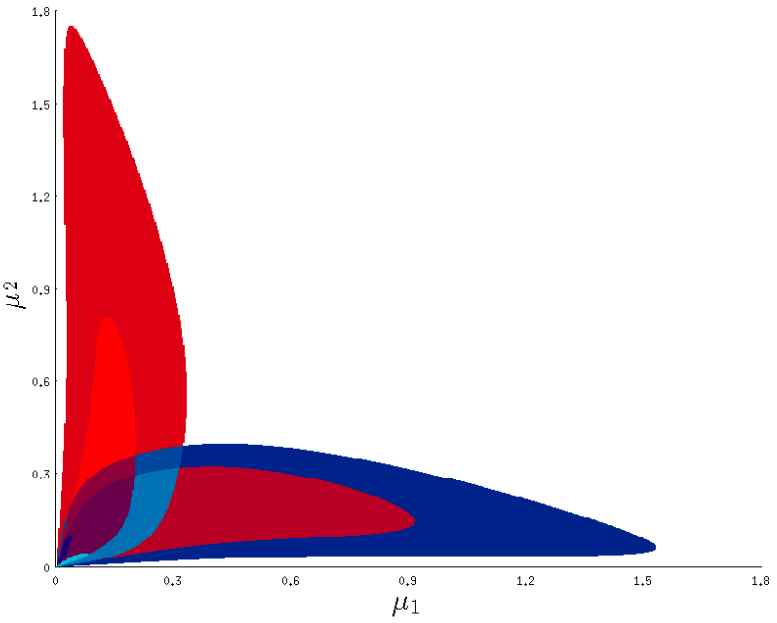


FIGURE 4.16: Parameter space distribution of types of non-monotonic policies with automatically routing boundaries and $(0,1)$-updates. The parameters are $\lambda = 1$, $\sigma_1 = 0.2, \sigma_2 = 0.3$, $N_1 = 5, N_2 = 6$. Each μ_i goes from $3 \cdot 10^{-3}$ to 1.8 in 600 steps of equal length. Parameter sets that give rise to monotonic policies are represented as white points, the parameter sets that give rise to non-monotonic policies with the same structure are represented by the same colour.

$5, N_2 = 6$) *which gives rise to the fixed point policy*

$$\mathbf{D}^*_{1,\boldsymbol{\Gamma_1}} = \begin{bmatrix} 0.9078 & 0.9150 & 0.9179 & 0.9194 & 0.9201 & 0.9203 & 1 \\ 0.8955 & 0.9029 & 0.9059 & 0.9073 & 0.9081 & 0.9084 & 1 \\ 0.8904 & 0.8971 & 0.8998 & 0.9011 & 0.9018 & 0.9021 & 1 \\ 0.8890 & 0.8947 & 0.8969 & 0.8980 & 0.8986 & 0.8989 & 1 \\ \mathbf{0.8900} & 0.8944 & 0.8961 & 0.8969 & 0.8974 & 0.8976 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Next we enter the subdued red region where we find the parameter vector $\boldsymbol{\Gamma_2} =$

$(\lambda = 1, \mu_1 = 0.21, \mu_2 = 0.06, \sigma_1 = 0, \sigma_2 = 0, N_1 = 5, N_2 = 6)$ *and the fixed point policy*

$$\mathbf{D}^*_{1,\boldsymbol{\Gamma_2}} = \begin{bmatrix} 0.8182 & 0.8381 & 0.8468 & 0.8513 & 0.8535 & 0.8536 & 1 \\ 0.7955 & 0.8160 & 0.8250 & 0.8296 & 0.8319 & 0.8324 & 1 \\ 0.7866 & 0.8058 & 0.8141 & 0.8183 & 0.8205 & 0.8213 & 1 \\ 0.7854 & 0.8021 & 0.8093 & 0.8129 & 0.8148 & 0.8156 & 1 \\ \mathbf{0.7894} & \mathbf{0.8027} & 0.8083 & 0.8111 & 0.8126 & 0.8132 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

As we keep moving up along the same vertical line we find the light blue region and the parameter vector $\boldsymbol{\Gamma_3} = (\lambda = 1, \mu_1 = 0.21, \mu_2 = 0.18, \sigma_1 = 0, \sigma_2 = 0, N_1 = 5, N_2 = 6)$ *which is associated with the fixed point policy*

$$\mathbf{D}^*_{1,\boldsymbol{\Gamma_3}} = \begin{bmatrix} 0.5623 & 0.6162 & 0.6446 & 0.6602 & 0.6670 & \mathbf{0.6649} & 1 \\ 0.5142 & 0.5689 & 0.5984 & 0.6148 & 0.6228 & 0.6232 & 1 \\ 0.4925 & 0.5461 & 0.5749 & 0.5911 & 0.5995 & 0.6017 & 1 \\ 0.4865 & 0.5376 & 0.5645 & 0.5796 & 0.5877 & 0.5908 & 1 \\ \mathbf{0.4942} & \mathbf{0.5400} & 0.5634 & 0.5763 & 0.5832 & 0.5862 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The next region our vertical line comes across is a slightly darker blue region from which we pick the parameter vector $\boldsymbol{\Gamma_4} = (\lambda = 1, \mu_1 = 0.21, \mu_2 = 0.33, \sigma_1 = 0, \sigma_2 = 0, N_1 = 5, N_2 = 6)$ *and hence the fixed point policy*

$$\mathbf{D}^*_{1,\boldsymbol{\Gamma_4}} = \begin{bmatrix} 0.3941 & 0.4596 & 0.4976 & 0.5195 & 0.5285 & \mathbf{0.5239} & 1 \\ 0.3381 & 0.4010 & 0.4391 & 0.4620 & 0.4732 & 0.4735 & 1 \\ 0.3100 & 0.3707 & 0.4083 & 0.4316 & 0.4443 & 0.4477 & 1 \\ 0.2974 & 0.3564 & 0.3932 & 0.4161 & 0.4293 & 0.4346 & 1 \\ \mathbf{0.2988} & 0.3556 & 0.3902 & 0.4113 & 0.4236 & 0.4292 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Finally the vertical line gets into the large red region where we chose the parameter vector $\boldsymbol{\Gamma_5} = (\lambda = 1, \mu_1 = 0.21, \mu_2 = 0.42, \sigma_1 = 0, \sigma_2 = 0, N_1 = 5, N_2 = 6)$ *which*

gives us the fixed point policy

$$\mathbf{D}^*_{1,\mathbf{r_5}} = \begin{bmatrix} 0.3326 & 0.3991 & 0.4393 & 0.4628 & 0.4726 & \mathbf{0.4672} & 1 \\ 0.2768 & 0.3387 & 0.3780 & 0.4023 & 0.4145 & 0.4147 & 1 \\ 0.2482 & 0.3068 & 0.3452 & 0.3700 & 0.3839 & 0.3879 & 1 \\ 0.2338 & 0.2905 & 0.3283 & 0.3531 & 0.3679 & 0.3741 & 1 \\ 0.2320 & 0.2874 & 0.3238 & 0.3474 & 0.3616 & 0.3684 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The similarities between the structure in 4.15 and that in 4.16 is shown clearly by Example 4.2.6. The difference between the two plots is not really one of structure, but one of complexity of structure. We can see this more clearly if we consider that Examples 4.2.3, 4.2.4 and 4.2.5 show us that a traversal of the nonmonotonic region in Figure 4.15 in such a way that all three regions are intersected would start with fixed point policies with this type of non-monotonicity

$$\mathbf{D}^*_1 = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 \\ + & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

where $+$ denotes a decision rule that is greater than we would have expected it to be by monotonicity, $\cdot$ denotes decision rules that conform to the expectations set by monotonicity, and entries taken from $\mathbb{Z}$ are trivially identical in all policies due to boundary conditions. The traversal would eventually lead to policies that exhibit the following type of non-monotonicity

$$\mathbf{D}^*_1 = \begin{bmatrix} \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

where $-$ denotes a decision rule that is less than we would have expected it to be by monotonicity. In between these types of policies we would have found policies with non-monotonicities of the type

$$\mathbf{D}^*_1 = \begin{bmatrix} \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 \\ + & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

We see that the vertical line in Example 4.2.6 is chosen in such a way that it traverses both the large red and the large blue region without passing any region twice and without passing through any region of parameter vectors associated with monotonic fixed point policies. Along this line we find that we again progress from policies with this non-monotonicity structure

$$\mathbf{D}_1^* = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

to policies with this structure

$$\mathbf{D}_1^* = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Between the two extremes we again find intermediates. In the case of the line described in 4.2.6 the intermediates are

$$\mathbf{D}_1^* = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{D}_1^* = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{D}_1^* = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

So for this particular choice of line the intermediates are obviously similar, in particular the last intermediate policy type is structurally identical to the only intermediate policy in the previous case. However, any choice of vertical line of this kind that we have followed have lead to the same outcome, just with different types of intermediate policies.

We finish this section by looking at the distributions of non-monotonicity types in a region of parameter space with even higher capacity, the case when $N_1 =$

$N_2 = 8$. Here the volume of parameter space in which we have observed non-monotonic policies exhibits a complexity in the the spatial distribution of types of non-monotonic policies that is substantially higher than in the previous examples. We show this in Figure 4.17. Again we get an understanding of the structure of this plot by considering the types of non-monotonicities we encounter along a particular vertical line in Figure 4.17, we do so in Example 4.2.7. In this example we compress the information contained in the policies like above, and only show

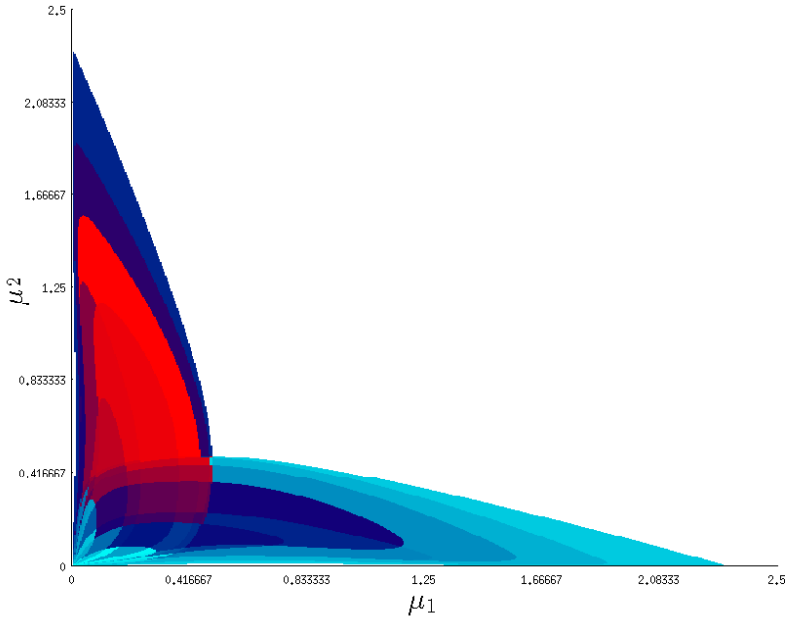


FIGURE 4.17: Parameter space distribution of types of non-monotonic policies with automatically routing boundaries and $(0,1)$-updates. The parameters are $\lambda = 1$, $\sigma_1 = \sigma_2 = 0$, $N_1 = N_2 = 8$. Each μ_i goes from $4.2 \cdot 10^{-3}$ to 2.5 in 600 steps of equal length. Parameter sets that give rise to monotonic policies are represented as white points, the parameter sets that give rise to non-monotonic policies with the same structure are represented by the same colour.

the non-monotonicities. So, for example, instead of writing

$$\mathbf{D}_1^* = \begin{bmatrix} 0.6856 & 0.7447 & 0.7778 & 0.7990 & 0.8133 & 0.8227 & 0.8277 & \mathbf{0.8268} & 1 \\ 0.6189 & 0.6842 & 0.7220 & 0.7465 & 0.7631 & 0.7740 & 0.7799 & \mathbf{0.7793} & 1 \\ 0.5772 & 0.6445 & 0.6841 & 0.7100 & 0.7275 & 0.7391 & 0.7453 & 0.7453 & 1 \\ 0.5499 & 0.6177 & 0.6579 & 0.6842 & 0.7019 & 0.7135 & 0.7200 & 0.7208 & 1 \\ 0.5328 & 0.6003 & 0.6404 & 0.6664 & 0.6838 & 0.6952 & 0.7017 & 0.7032 & 1 \\ 0.5248 & 0.5913 & 0.6304 & 0.6554 & 0.6719 & 0.6827 & 0.6890 & 0.6912 & 1 \\ \mathbf{0.5268} & 0.5906 & 0.6275 & 0.6506 & 0.6657 & 0.6755 & 0.6812 & 0.6836 & 1 \\ \mathbf{0.5413} & \mathbf{0.5993} & \mathbf{0.6319} & \mathbf{0.6521} & 0.6649 & 0.6730 & 0.6778 & 0.6798 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

we simply write

$$\mathbf{D}_1^* = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & + & + & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

EXAMPLE 4.2.7. *Consider a vertical line in Figure 4.17 at* $\mu_1 = 0.0944$ *and sample one non-monotonic policy for each region we traverse as we follow this line in the positive* μ_2 *direction. Since* μ_2 *is the only parameter that differs, we denote the policies by* $\mathbf{D}_i^{*,k}(\hat{\mu_2})$ *which should be interpreted as the set of decision rules for queue* i *under the parameter vector* $\mathbf{\Gamma}_k = (\lambda = 1, \mu_1 = 0.429, \mu_2 = \hat{\mu_2}, \sigma_1 = 0, \sigma_2 = 0, N_1 = 8, N_2 = 8)$*, where* k *is the sequential number of the policy as encountered in Figure 4.17.*

$$\mathbf{D}_1^{*,1}(0.0127) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{D}_1^{*,2}(0.025) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{D}_1^{*,3}(0.05) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & + & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{D}_1^{*,4}(0.0709) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & + & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{D}_1^{*,5}(0.0959) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & + & + & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{D}_1^{*,6}(0.1127) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & + & + & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{D}_1^{*,7}(0.1377) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & + & + & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{D}_1^{*,8}(0.2082) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & + & + & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{D}_1^{*,9}(0.2123) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & + & + & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{D}_1^{*,10}(0.25) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & + & + & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{D}_1^{*,11}(0.4168) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & + & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{D}_1^{*,12}(0.4668) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{D}_1^{*,13}(0.4877) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{D}_1^{*,14}(0.5418) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{D}_1^{*,15}(0.7082) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{D}_1^{*,16}(0.8541) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & - & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The first thing to note about the policies in Example 4.2.7 is that we again see an orderly progression from less non-monotonicity to more non-monotonicity and back again. We do note one anomalous transition in this case, though, the one between the $D_1^{*,1}$ and $D_1^{*,2}$ which we find to be an effect of the sampling resolution of μ_2 rather than lack of structure. We see this by considering the region surrounding this transition in the same resolution as is used for the whole plot, that is let μ_1 go from 0.4167 to 0.4375 and let μ_2 go from $4.167 \cdot 10^{-3}$ to $8.33 \cdot 10^{-2}$ both in 600 steps of equal length. In this resolution we find exactly one more type of non-monotonic policy in between the two already shown in example 4.2.7. One example of this type of policy can be found at $\mu_2 = 3.15 \cdot 10^{-2}$ and it

has the expected form

$$\mathbf{D}_1^*(0.0315) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ + & + & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

While it is not possible to do a numerically exhaustive search of even a limited volume of parameters space, we have examined several other regions of interest along this vertical line and not found any other missing policy types, which indicates that the orderliness of the structures of the non-monotonic policies is maintained as the system complexity is increased by for example adding capacity.

These examples of the structure contained in the regions of parameter space characterised by non-monotonic polices give us some indication of how the non-monotonicities occur. In particular we see that the non-monotonicities do not occur in a randomly distributed fashion, but that there seems to be some structure to it, at least in as much as policies with complex non-monotonic structures occur embedded in regions of parameter space with lower levels of non-monotonicity complexity. We also see that this observation is robust over at least the low levels of capacity that we we have considered here, so that while the complexity of the patterns of non-monotonicity increases, it still increases in the organised manner of an increase of at most one locus of non-monotonicity at a time.

4.2.3 Stationary joining probabilities

Up to this point the resulting decision policies have only really been classified in terms of whether they produce monotonic, aperiodic policies or not. These are obviously very coarse classifications that do nothing to help us understand the system behaviour in the normal situation - when the fixed point policies are monotonic and aperiodic. One reasonable question to ask about the fixed point policies is how parameter changes affect the stationary joining probabilities defined in Section 3.2.

We are naturally particularly interested in the joining probability with respect to fixed point policies, so we extend the definition somewhat here. Following our notation with respect to decision policies we denote the stationary joining probability for the fixed point policy associated with $\mathbf{\Gamma}$ starting the iteration from decision policy $\mathbf{D}^0$ by $p^*_{i,\mathbf{D}^0}(\mathbf{\Gamma})$. If we let $\mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}$ be the fixed point policy associated with $\mathbf{\Gamma}$ starting from $\mathbf{D}^0$, then

$$p_i(\mathbf{\Gamma}, \mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}) = p^*_{i,\mathbf{D}^0}(\mathbf{\Gamma}).$$

In Chapter 3 we show certain basic monotonicity properties of stationary joining probabilities with regards to varying system parameters. Here we go on to consider how this relates to fixed point policies of the system, as well as how these relationships are affected by the specific update rules used to find the fixed point policies.

$p^*(\mathbf{\Gamma})$ and service rates

We consider varying the service rate in queue i, μ_i, while allowing the service rate in queue j, μ_j, to remain fixed and consider how this will affect the stationary joining probability for queue i, $p^*_{i,\mathbf{D}^0}(\mathbf{\Gamma})$. In addition to $\mathbf{\Gamma}$, $\mathbf{D}^0$ and $\mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}$ defined above we also define a parameter vector $\mathbf{\Gamma}^+$ which differs from $\mathbf{\Gamma}$ only in that μ_i in $\mathbf{\Gamma}$ is replaced with $\mu_i^+ > \mu_i$ in $\mathbf{\Gamma}^+$. As a starting point we consider what happens when we compare the stationary joining probability of $\mathbf{\Gamma}$ under its fixed point policy, $p^*_{i,\mathbf{D}^0}(\mathbf{\Gamma})$ to the stationary joining probability of $\mathbf{\Gamma}^+$ assuming all the customers still use $\mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}$, $p_i(\mathbf{\Gamma}^+, \mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0})$.

First we consider how the change in service rates affects the probability of the system being in different states. Increasing μ_i, the service rate in queue i, while keeping all other parameters constant, means that the system spends more time in states with a lower occupancy in queue i. As an aside, we note that this is true for any set of parameters, even in the severely overloaded case. As we keep the decision policy associated with the lower service rate, this change in time spent in lower n_i states is the only first order change in the system. To see how this affects the stationary joining probability we must however consider the structure of $\mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}$. In terms of structure we consider three cases; either $\mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}$ is constant, non-monotonic or monotonic. First we note that if the decision policy is constant in $\mathbf{n}$ the fact that the system has a higher probability of being in a state with low occupancy in queue i will not affect the stationary joining probability

at all. In fact, if the decision policy $\mathbf{D}$ is constant with $D_i(\mathbf{n}) = q, \forall\ \mathbf{n} \in \mathcal{S}$ then $p_i(\mathbf{\Gamma}, \mathbf{D}) = q$. If instead $\mathbf{D}$ is non-monotonic, we will not be able to make any statement regarding the afffect of the increased time spent in the low n_i states. This is because without monotonicity we are unable to say anything, in general, about the relationship between occupancy and decision rules. However, in the case when $\mathbf{D}$ is monotonic our situation conforms to Theorem 3.2.1 in Chapter 3 which, with the notation employed here with regards to fixed point policies, can be stated as

$$p^*_{1,\mathbf{D}^0}(\mathbf{\Gamma}) \leq p_1(\mathbf{\Gamma}^+, \mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}).$$

We go on to consider the implications of not only changing the service rates, but to also change the decision policies accordingly. As previously discussed it is reasonable to expect that $D^*_i(\mathbf{n})$, the probability at stationarity of a general arrival joining queue i when they find the system in state $\mathbf{n}$, is non-decreasing in μ_i. Thus we might expect that the effect discussed above would be maintained, or in fact exacerbated, by not only increasing the service rate, but also finding the fixed point policy associated with that increased service rate. Which is to say, again assuming decision policy monotonicity, we might expect $p^*_{i,\mathbf{D}^0}(\mathbf{\Gamma}) \leq p^*_{i,\mathbf{D}^0}(\mathbf{\Gamma}^+)$. We give a numeric example of this in Example 4.2.8.

EXAMPLE 4.2.8. *Let* $\lambda = 1,\ \sigma_1 = 0.1,\ \sigma_2 = 0.2,\ \mu_1 = 2,\ \mu_2 = 3$ *be the system parameters of a system with* $N_1 = N_2 = 3$ *and let* $\mathbf{\Gamma} = (\lambda, \mu_1, \mu_2, \sigma_1, \sigma_2, N_1, N_2)$. *We then run policy iteration using the* $[0, 1]$*-update rules and use decision policies with rejection boundaries. Starting from a constant* $\mathbf{D}^0$ *where* $D_i(\mathbf{n}) = 0.5, \forall \mathbf{n} \in \mathcal{S}, i \in \{1, 2\}$ *we find the fixed point policy*

$$\mathbf{D}^*_{1,\mathbf{\Gamma}} = \begin{bmatrix} 0 & 0.62 & 1 & 0 \\ 0 & 0 & 0.16 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

and we can compute the stationary joining probability to be

$$p^*_{1,\mathbf{D}^0}(\mathbf{\Gamma}) = 0.18.$$

Now, we create our $\mathbf{\Gamma}^+$ *by replacing* μ_1 *with* $\mu_1^+ = 6$ *and find that*

$$p_1(\mathbf{\Gamma}^+, \mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}) = 0.20.$$

So, as we know from Theorem 3.2.1,

$$p^*_{1,\mathbf{D}^0}(\mathbf{\Gamma}) \leq p_1(\mathbf{\Gamma}^+, \mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0})$$

We now we run policy iteration with $\mathbf{\Gamma}^+$ *to find that*

$$\mathbf{D}^*_{1,\mathbf{\Gamma}^+} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0.9 & 1 & 1 & 0 \\ 0.51 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

which we use to compute the stationary joining probability

$$p^*_{1,\mathbf{D}^0}(\mathbf{\Gamma}^+) = 0.97.$$

So we see that, as anticipated

$$p^*_{1,\mathbf{D}^0}(\mathbf{\Gamma}) < p_1(\mathbf{\Gamma}^+, \mathbf{D}^*_{\mathbf{\Gamma}}) < p^*_{1,\mathbf{D}^0}(\mathbf{\Gamma}^+).$$

So as indicated above we might expect the stationary joining probability for queue i to be non-decreasing in μ_i, and in the same way we might expect it to be non-increasing in μ_j. We explore this idea further by looking at ranges of values, update rule by update rule. We obviously only expect this to hold in the cases when we actually see policy convergence, so in the following plots, where each point corresponds to the stationary joining probability for a specific set of parameters, the values for the parameter sets that result in periodic policies are set to -0.1 to clearly separate it from actual probabilities, and are depicted as black in the pictures.

Stationary joining probabilities by μs, $\{0, 1, \tilde{p}\}$-updates

We start by considering the stationary joining probabilities in the context of $\{0, 1, \tilde{p}\}$-updates, with the symmetric parameters $\lambda = 1$, $\sigma_1 = 0.45$, $\sigma_2 = 0.45$, $N_1 = 3$, $N_2 = 3$ and $\tilde{p} = 0.5$. Then we let μ_i vary from 0.01 to 2 in 200 steps of equal length. This produces the following plot, Figure 4.18, where we again see an interesting pattern emerge.

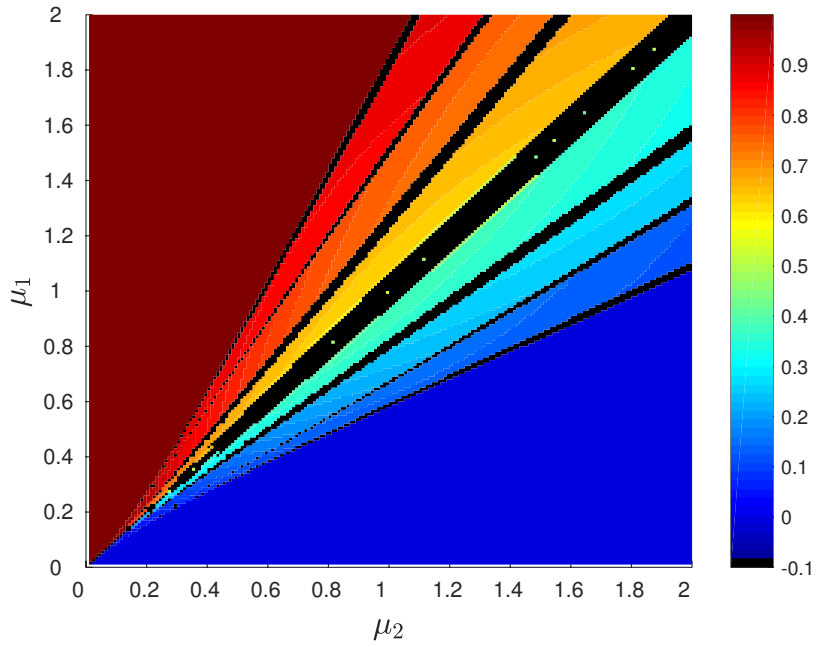


FIGURE 4.18: Stationary joining probability $p_1^*(\mathbf{\Gamma})$ as a function of μ_1 and μ_2 with remaining parameters $\lambda = 1$, $\sigma_1 = 0.45$, $\sigma_2 = 0.45$, $N_1 = 3$, $N_2 = 3$, $\tilde{p} = 0.5$. The points representing periodic policies have been assigned the value -0.1 and are depicted as black in the picture.

Two things are striking about this plot. First of all we note the conspicuous pattern of periodic policies, we will not focus on them here as this has been discussed in Section 4.2.1. The second striking feature is the orderly structure of the probabilities and how well they conform to what we would expect, something that we have learned to be cautious about expecting in this system.

In the interest of getting some understanding for the patterns in figure 4.18 we also look at the figure produced by a less symmetric set of parameters in Figure 4.19 on the facing page. For this figure we have used the same general arrival rate, system size, tie breaker probability, and range of the service rates as for Figure 4.18. The point of difference is that we have changed the intrinsic arrival rates to $\sigma_1 = 0$ and $\sigma_2 = 0.9$, thus preserving the total intrinsic arrival rate, but making it asymmetric between the two queues.

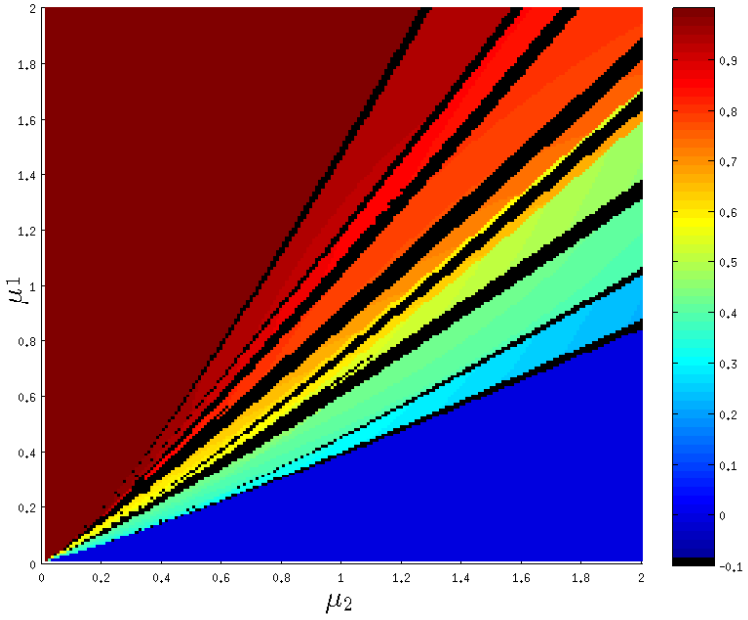


FIGURE 4.19: Stationary joining probability $p_1^*(\mathbf{\Gamma})$ as a function of μ_1 and μ_2 with remaining parameters $\lambda = 1$, $\sigma_1 = 0$, $\sigma_2 = 0.9$, $N_1 = 3$, $N_2 = 3$, $\tilde{p} = 0.5$. The points representing periodic policies have been assigned the value -0.1 and are depicted as black in the picture.

We can see that the pattern changes roughly in the way we would expect. The increase of the intrinsic arrival rate in queue 2 and decrease in queue 1 mean that there are going to be more sets of parameters for which queue 1 is more attractive. Along the μ_2-axis, where the service rate in queue 1 is very low, the asymmetry of intrinsic arrival rates is not sufficient to make queue 1 attractive, and almost all general arrivals who enter the system will join queue 2. However, the red region along the μ_1-axis of the picture, the region where almost all general arrivals join queue 1 is much larger. Between the two extremes we see a similar structure to that in the case of symmetric parameters, but with a generally much higher probability of general arrivals joining queue 1.

Stationary joining probabilities by μs, $[0,1]$-updates

We wish to compare this with how the stationary joining probabilities depend on service rates in the context of $[0,1]$-updates, and start by utilising the same symmetric set of parameters as in the case of $\{0,1,\tilde{p}\}$-updates as far as applicable, with $\lambda = 1,\ \sigma_1 = 0.45,\ \sigma_2 = 0.45,\ N_1 = 3,\ N_2 = 3$. With this update rule we also need to chose a δ to specify the range that gives rise to mixed policies, somewhat arbitrarily we pick $\delta = 0.1$. The result can be found in Figure 4.20.

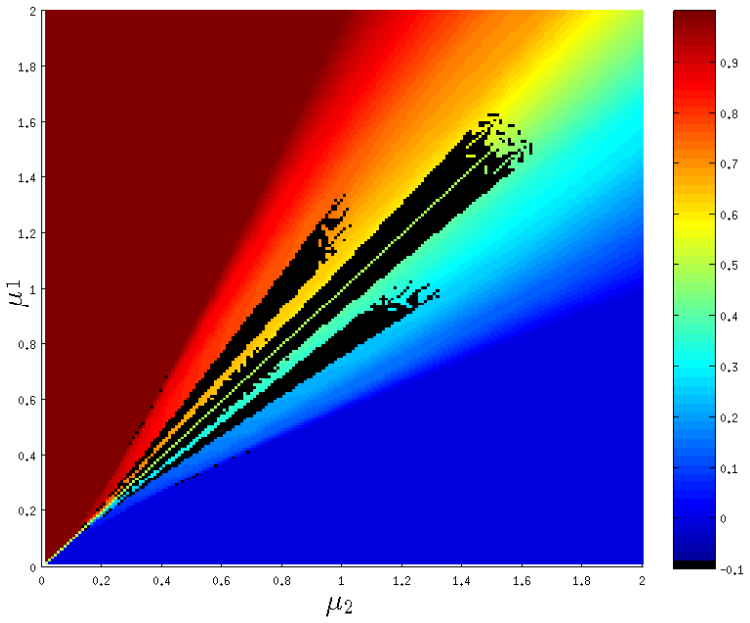


FIGURE 4.20: Stationary joining probability $p_1^*(\mathbf{\Gamma})$ for fixed point policies under $[0,1]$-update policy iteration as a function of μ_1 and μ_2 with remaining parameters $\lambda = 1,\ \sigma_1 = 0.45,\ \sigma_2 = 0.45,\ N_1 = 3,\ N_2 = 3,\ \delta = 0.1$. The points representing periodic policies have been assigned the value -0.1 and are depicted as black in the picture.

Again we note the interesting structure in the figure, in particular we note that the pattern bears some similarities to the pattern in Figure 4.18. This is clearly

not surprising, given the by now well documented similarities between the output from the two update rules.

We again also look at an asymmetric case to see if the structure changes in the expected way. Just as before we maintain the total intrinsic arrival rate, but let all of the intrinsic arrivals join queue 2 instead of having the same rate at both queues. The resulting plot can be found in Figure 4.21.

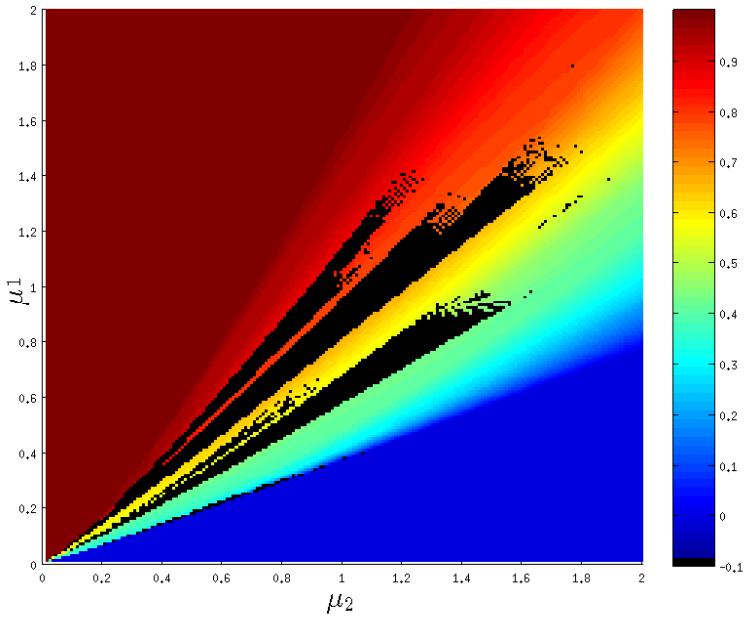
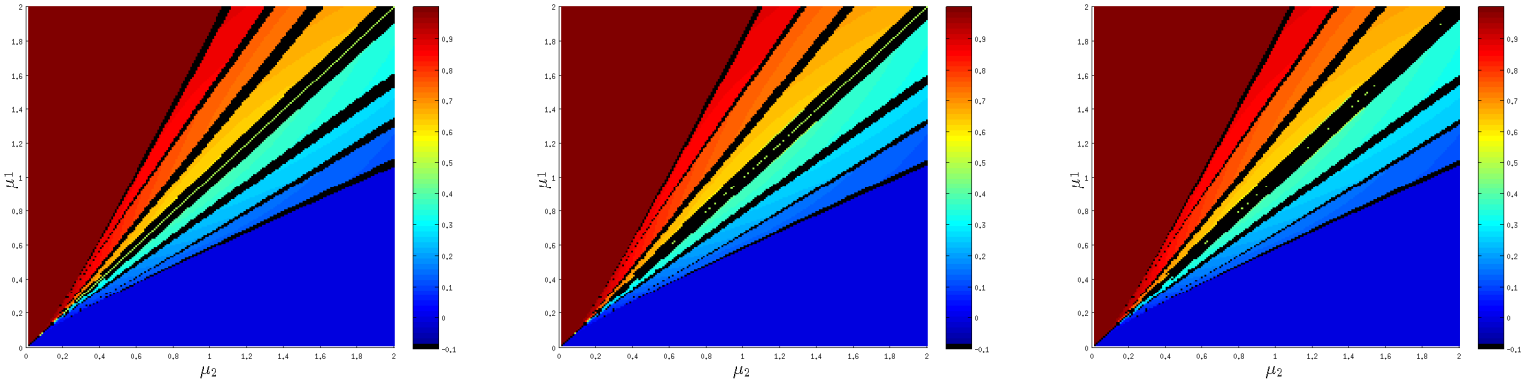


FIGURE 4.21: Stationary joining probability $p_1^*(\mathbf{\Gamma})$ as a function of μ_1 and μ_2 with remaining parameters $\lambda = 1$, $\sigma_1 = 0$, $\sigma_2 = 0.9$, $N_1 = 3$, $N_2 = 3$, $\delta = 0.1$. The points representing periodic policies have been assigned the value -0.1 and are depicted as black in the picture.

We again note that the region of parameter space in which almost all of the general arrivals go to queue 1, the red region along the vertical axis, grows, while the region in which almost all of the general arrivals go to queue 2, the blue part region the horizontal axis, shrinks. It is also clear that in between the two extremes

the stationary joining probability has increased in general, so that the region where the $p_1^* < 0.5$ has shrunk considerably compared to that found in Figure 4.20.

To further understand the affect of δ on the outcome in terms of stationary joining probability we also look at the symmetric case in three different cases, for δs at three different orders of magnitude.



(A) $\delta = 0.01$ (B) $\delta = 0.001$ (C) $\delta = 0.0001$

FIGURE 4.22: Stationary joining probability $p_1^*(\mathbf{\Gamma})$ as a function of μ_1 and μ_2 for the same parameters as in Figure 4.20 on page 142 but with varying values of δ.

As previously discussed and numerically examined in some detail in Section 4.2.1 the $[0, 1]$-updates approach $\{0, 1, \tilde{p}\}$-updates as δ approaches 0. Comparing decision policies directly does not lend itself well to visualisation. However, the stationary joining probabilities serve as a sort of proxy for the decision policies. As the stationary joining probabilities represent a compression, we can't say that equal stationary joining probabilities mean equal decision policies, although when we are looking at the same parameter sets, and we know that all policies are monotonic, equal stationary joining probabilities is an indication, but not a guarantee, that the decision policies themselves are close, subject to the precision of both calculation and graphic representation.

With that in mind Figure 4.22 on the facing page is one way to gain some insight into how this convergence works over parameter space. We can see that the regions of parameter space that give rise to periodic policies grow as δ shrinks but not much changes in the stationary joining probabilities. In fact, when we look at the difference between the distribution of stationary joining probabilities in parameter space for $[0, 1]$-updates when $\delta = 0.01$ and $\{0, 1, \tilde{p}\}$-updates with $\tilde{p} = 0.5$ we find that the only substantial difference between the two sets of joining probabilities do in fact fall in, or near, the regions that correspond to periodic policies in one situation or the other. This is the situation shown in Figure 4.23.

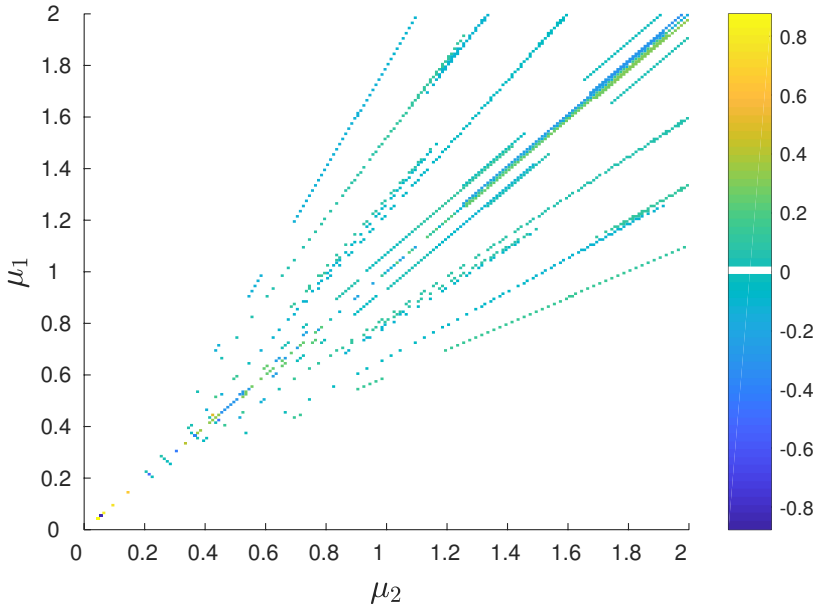


FIGURE 4.23: Difference in stationary joining probabilities $p_1^*(\mathbf{\Gamma})$ between stationary joining probabilities in the case of $[0, 1]$-updates with $\delta = 0.01$ and $\{0, 1, \tilde{p}\}$-updates with $\tilde{p} = 0.5$ for the same set of parameters used to create Figures 4.18 and 4.20.

We see clearly here that for the majority of parameter sets the difference between the stationary joining probability from the two kinds of update rules is 0, depicted

as white in Figure 4.23 on the previous page. We remind the reader that due to the finite number of decision policies available to $\{0, 1, \tilde{p}\}$-updates, the fact that the outcome is exactly 0 is highly suggestive of policy identity between the two update rules for these parameter sets. We can also go one step further and only take into account the states that fall outside the parameter sets that give rise to periodic policies with either update rule, this is depicted in Figure 4.24.

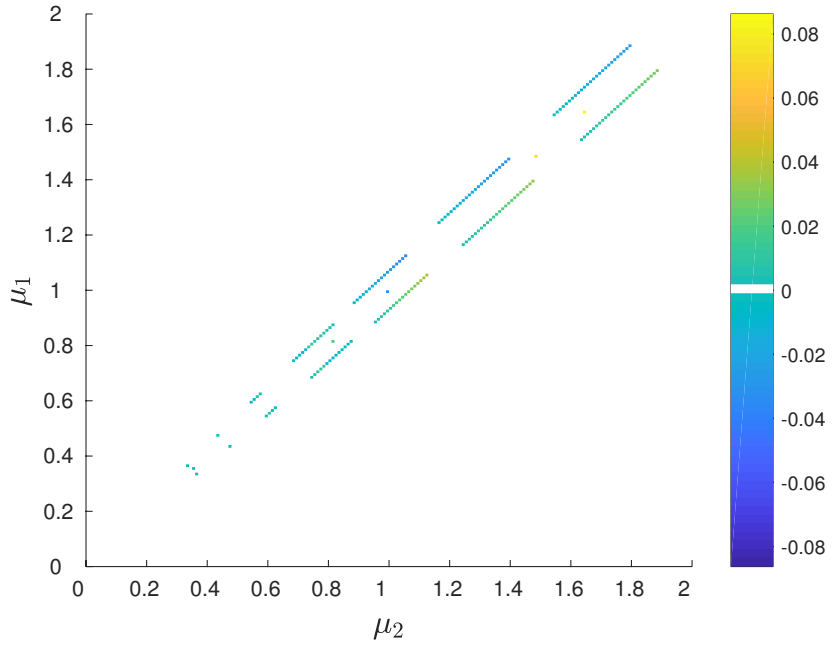


FIGURE 4.24: The same information as in Figure 4.23 on the preceding page, but with all sets of parameters that result in periodic policies with either update rule forced to be 0.

In this figure we have artificially set the value for all parameter sets that correspond to periodic policies with either update rule to 0, depicted as white in the figure. This means that all the coloured parts of the image correspond to sets of parameters that give rise to fixed point policies with both update rules, but

where the resulting stationary joining probabilities differ. We also note that the maximum absolute value of the difference between the stationary joining probabilities is much lower when the parameter sets giving rise to the periodic policies are blocked out, further suggesting that in the cases when a fixed point policy has been found, there are already strong similarities between the results from the two types of update rules.

Stationary joining probabilities by μs, $(0,1)$-updates

We finally consider how the service rates affect the stationary joining probability when the policy iteration is carried out with the $(0,1)$-update rule. So we again let the parameter vector consist of $\lambda = 1,\ \sigma_1 = 0.45,\ \sigma_2 = 0.45,\ N_1 = 3,\ N_2 = 3$, and again let μ_1 and μ_2 go from 0.01 to 2 in 200 steps of equal length. The resulting stationary joining probabilities are depicted in Figure 4.25 on the next page. As expected no parameter sets lead to periodicities with this update rule, so there is no need to block any points out. In addition to all parameter sets leading to aperiodic monotonic policies it is also clear that this update rule gives rise to exactly the sort of transition in stationary joining probabilities that we might expect in practice for the system. Explicitly we see here that for any fixed value of either service rate, μ_i, we see a smooth decline in stationary joining probability, $p_i^*(\mathbf{\Gamma})$, in that queue when the other service rate, μ_j, is increased from 0.01 to 2. The shape of this change is shown explicitly for $\mu_1 = 1$ in Figure 4.26 on page 149. For comparison we show the same figure but for $[0,1]$- and $\{0,1,\tilde{p}\}$-updates in Figure 4.27 on page 150. There are several points of interest in Figure 4.27, but we restrict ourselves to briefly discussing the most conspicuous properties. First we note that, as we might have expected looking at Figure 4.18 on page 140 and Figure 4.20 on page 142, there are clear similarities between the two images, but also some interesting differences. In particular, this figure makes it clear that the differences in outcome in terms of stationary joining probabilities between $\{0,1,\tilde{p}\}$-updates and $[0,1]$-updates are both intuitive and complex. Intuitive in that there is a sequence of identical stationary joining probabilities from the two methods, and that these identities occur in the interior of the stable regions. Complex in that while we note that $[0,1]$-updates manages to bridge many of the gaps created by periodicities in the case of $\{0,1,\tilde{p}\}$-updates, as we saw in Figure 4.6 on page 110, there are sets of parameters that result in periodicities for $[0,1]$-updates and fixed point policies with $\{0,1,\tilde{p}\}$-updates.

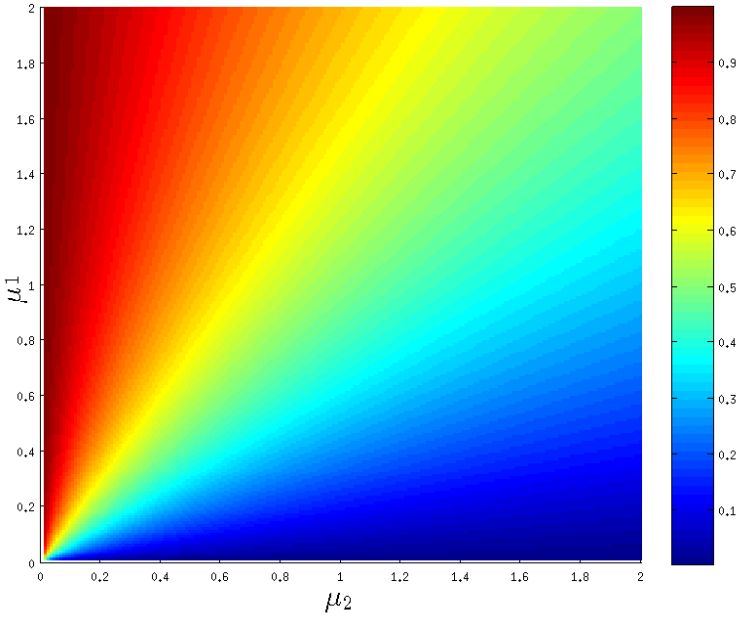


FIGURE 4.25: Stationary joining probabilities for fixed point policies under $(0,1)$-update policy iteration as a function of μ_1 and μ_2 with remaining parameters $\lambda = 1,\ \sigma_1 = 0.45,\ \sigma_2 = 0.45,\ N_1 = 3,\ N_2 = 3$.

However, the main point of interest is the structure of the transition from high $p_1^*(\mathbf{\Gamma})$ to low $p_1^*(\mathbf{\Gamma})$. For both update rules represented in Figure 4.27 on page 150 the transition happens in something close to piecewise linear fashion with discontinuities at the gaps induced by the parameter sets that give rise to periodic policies. In Figure 4.26 on the facing page by contrast we see a continuous transition all the way from high to low stationary joining probability.

As for the two other update rules we are interested to see what the stationary joining probabilities look like for an asymmetric set of parameters as well, so in Figure 4.28 on page 151 we see $p_1^*(\mathbf{\Gamma})$ for $0.01 \leq \mu_i \leq 2$ for $i \in \{1,2\}$ with remaining parameters $\lambda = 1,\ \sigma_1 = 0,\ \sigma_2 = 0.9,\ N_1 = 3,\ N_2 = 3$. With the smooth transition that this update rule gives rise to the skewing is less obvious than in the two other cases, but comparing Figure 4.25 to Figure 4.28 we see the same kind of skewing

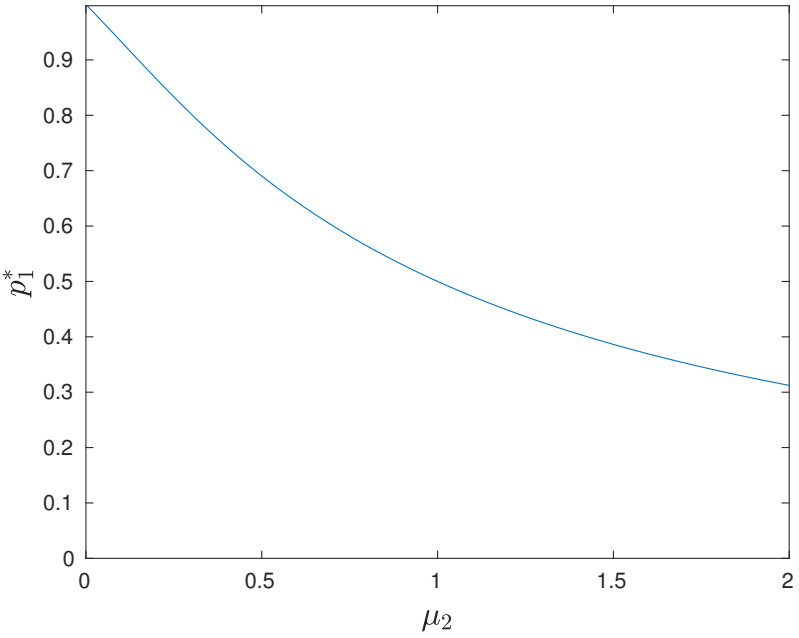


FIGURE 4.26: Stationary joining probability $p_1^*(\mathbf{\Gamma})$ for fixed point policies under $(0,1)$-update policy iteration as a function of μ_2 with remaining parameters $\mu_1 = 1,\ \lambda = 1,\ \sigma_1 = 0.45,\ \sigma_2 = 0.45,\ N_1 = 3,\ N_2 = 3$.

as we saw between Figure 4.20 and Figure 4.21 as well as between Figure 4.18 and Figure 4.19.

Conclusions on stationary joining probabilities and service rates

We see that in these examples the stationary joining probability for queue i is non-decreasing in μ_i and non-increasing in μ_j. We have not observed any departure from this behaviour despite extensive search in parameter space for each update rule.

On the other hand the way that the stationary joining probability depends on the service rates differs significantly depending on update rules. In the case of the $(0,1)$-update rules we see a smooth transition, while the two other update rules

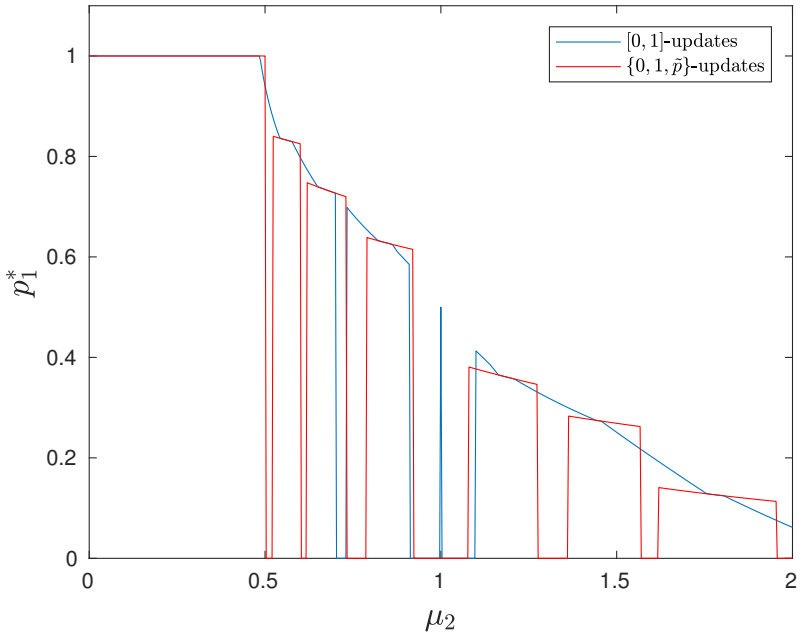


FIGURE 4.27: Stationary joining probability $p_1^*(\mathbf{\Gamma})$ for fixed point policies under $\{0, 1, \tilde{p}\}$-update policy iteration in red and $[0, 1]$-update policy iteration in blue as a function of μ_2 with $\mu_1 = 1$ and remaining parameters as in Figure 4.18 on page 140 and Figure 4.20 on page 142 respectively.

give rise to piecewise linear transitions with discontinuities marked out by regions of state space giving rise to periodic policies.

$p^*(\mathbf{\Gamma})$ and intrinsic arrival rates

As in Chapter 3 we also consider how the stationary joining probabilities and the intrinsic arrival rates interact. We have already touched upon this in the preceding sections where we used the intrinsic arrival rates to skew the set of parameters. There are also strong similarities between the effect on stationary joining probabilities of changes in intrinsic arrival rates and the affect of changing service rates. For these two reasons this discussion is somewhat more cursory than the preceding one.

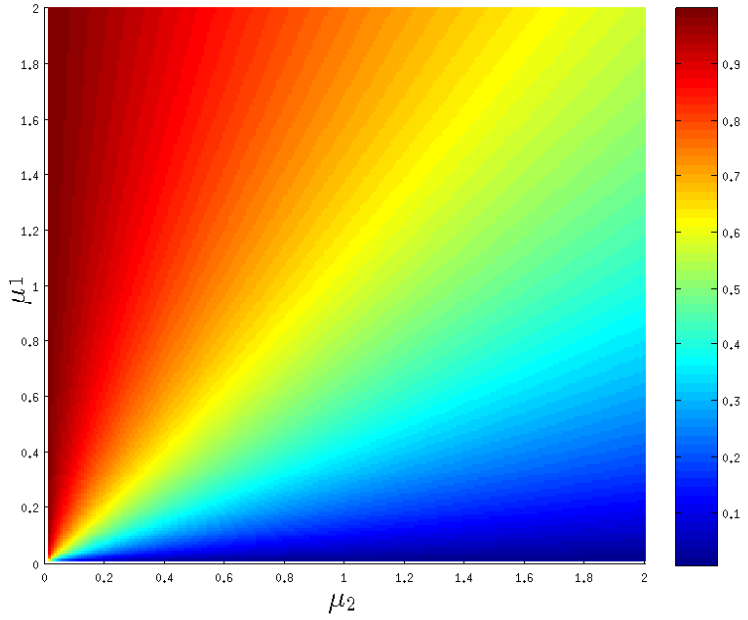


FIGURE 4.28: Stationary joining probabilities for fixed point policies under $(0,1)$-update policy iteration as a function of μ_1 and μ_2 with remaining parameters $\lambda = 1,\ \sigma_1 = 0,\ \sigma_2 = 0.9,\ N_1 = 3,\ N_2 = 3,\ \delta = 0.1$.

Just as in the discussion related to service rates we start by defining the parameter vectors and decision policies we need for the analysis. We again carry out the discussion in the context of $[0,1]$-updates and rejection boundaries. Let $\mathbf{\Gamma}$ be a parameter vector and let $\mathbf{D}^0$ be an initial policy and $\mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}$ the fixed point policy associated with this parameter vector . Let $\mathbf{\Gamma}^+$ be a parameter vector which differs from $\mathbf{\Gamma}$ only in that σ_i in $\mathbf{\Gamma}$ is replaced by $\sigma_i^+ > \sigma_i$ in $\mathbf{\Gamma}^+$. We again start by considering what we expect to find when we compare the stationary joining probability associated with $\mathbf{\Gamma}$ under its fixed point policy, $p^*_{i,\mathbf{D}^0}(\mathbf{\Gamma})$, to the stationary joining probability associated with $\mathbf{\Gamma}^+$ but assuming all the customers still use $\mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}$, $p_i(\mathbf{\Gamma}^+, \mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0})$.

So, compared to the situation when the parameter vector is $\mathbf{\Gamma}$, the process operating under the parameter vector $\mathbf{\Gamma}$ is more likely to be in a state in which

the occupancy in queue i is higher which, again assuming monotonicity of the decision policy, directly translates to a lower probability that an intrinsic arrival joins queue i. This is another, intuitive, way of reaching the statement in Theorem 3.2.2 which we can state, with the notation used in the present chapter, as

$$p_i(\mathbf{\Gamma}^+, \mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}) \leq p^*_{i,\mathbf{D}^0}(\mathbf{\Gamma}).$$

As in the case of service rates we then go on to consider what happens to the stationary joining probability when we not just replace the intrinsic arrival rate, but also find the corresponding fixed point policy. It is reasonable to expect that the increased intrinsic arrival rate in queue i will make queue i less attractive, so that the decision rule $D^*_i(\mathbf{n})$ is non-increasing in σ_i for all $\mathbf{n} \in \mathcal{S}$. Thus, as in the case with increased service rates explored in Section 4.2.3, we expect the effect of an increase in intrinsic arrival rates to not only remain but be exacerbated in this situation. So we expect

$$p^*_{i,\mathbf{D}^0}(\mathbf{\Gamma}^+) \leq p_i(\mathbf{\Gamma}^+, \mathbf{D}^*_{\mathbf{\Gamma}}).$$

This situation is realised numerically in Example 4.2.9.

EXAMPLE 4.2.9. *Let* $\lambda = 1,\ \sigma_1 = 0.1,\ \sigma_2 = 0.1,\ \mu_1 = 3.5,\ \mu_2 = 3$ *be the system parameters of a system with* $N_1 = N_2 = 3$ *and let* $\mathbf{\Gamma} = (\lambda, \mu_1, \mu_2, \sigma_1, \sigma_2, N_1, N_2)$. *We run policy iteration using the* $[0, 1]$*-update rules and use decision policies with rejection boundaries. Starting from a constant* $\mathbf{D}^0$ *where* $D_i(\mathbf{n}) = 0.5, \forall \mathbf{n} \in \mathcal{S}, i \in \{1, 2\}$ *we find the fixed point policy*

$$\mathbf{D}^*_{1,\mathbf{\Gamma}} = \begin{bmatrix} 0.73 & 1 & 1 & 0 \\ 0.033 & 0.82 & 1 & 0 \\ 0 & 0.17 & 0.93 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

and we can compute the stationary joining probability to be

$$p^*_{1,\mathbf{D}^0}(\mathbf{\Gamma}) = 0.64.$$

Now, we create our $\mathbf{\Gamma}^+$ *by replacing* $\sigma_1 = 0.1$ *with* $\sigma_1^+ = 1$ *and find that*

$$p_1(\mathbf{\Gamma}^+, \mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}) = 0.53.$$

So, as expected based on Theorem 3.2.2, in this case

$$p_1(\mathbf{\Gamma}^+, \mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}) < p^*_{1,\mathbf{D}^0}(\mathbf{\Gamma}).$$

We go on to run policy iteration with $[0,1]$*-updates starting from* $\mathbf{D}^0$ *to find the fixed point policy for the high intrinsic arrival rates parameters*

$$\mathbf{D}^*_{1,\mathbf{\Gamma}^+} = \begin{bmatrix} 0.56 & 1 & 1 & 0 \\ 0 & 0.61 & 1 & 0 \\ 0 & 0.048 & 0.8 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

We use that fixed point policy to find stationary joining probability

$$p^*_{1,\mathbf{D}^0}(\mathbf{\Gamma}^+) = 0.45.$$

So we see that that the relationship between the stationary joining probabilities exhibit the relationships we expected;

$$p^*_{1,\mathbf{D}^0}(\mathbf{\Gamma}^+) \leq p_1(\mathbf{\Gamma}^+, \mathbf{D}^*_{\mathbf{\Gamma},\mathbf{D}^0}) \leq p^*_{1,\mathbf{D}^0}(\mathbf{\Gamma}).$$

This reasoning explains why we expect the stationary joining probabilities in the context of fixed point policies to be non-decreasing in intrinsic arrival rates.

Just as with the service rate above we now go on to numerically consider the stationary joining probabilities swathe of parameter vectors under the different update rules through visual representations. Due to previously discussed relationship in terms of system performance between service rates and intrinsic arrival rates, we spend less time on analysis of these figures as which all conform to what we expect based Example 4.2.9 as well as the numerical exploration of the relationship between stationary joining probabilities and service rate.

In the following we consider the same two sets of parameter vectors with each of the update rules. In both sets $\lambda = 2$ and $N_1 = N_2 = 3$ and σ_i varies from 0.025 to 5 in 200 steps. The difference between the cases is the fact that in one set of parameter vectors the service rates are identical in the queues at $\mu_1 = \mu_2 = 6$ while in the other case $\mu_1 = 5.9$ and $\mu_2 = 6.1$. This means that all of the figures below contain outcomes for underloaded systems only.

We also note that the colour scale is constant over the following figures. This has the effect of facilitating comparison between images but does have the effect that in some cases, particularly Figure 4.34 on page 159 and Figure 4.35 on page 160 the figures are fairly monotonous.

Stationary joining probabilities by σs, $\{0, 1, \tilde{p}\}$-updates

We start by considering the stationary joining probabilities $p^*_{i,\mathbf{D}^0}(\mathbf{\Gamma})$ in the case of $\{0, 1, \tilde{p}\}$, where we let the tie breaker probability be $\tilde{p} = 0.5$. The outcomes for the set of parameter vectors with equal service rates be found in Figure 4.29 and the outcomes for the set with different service rates can be found in Figure 4.30 on the next page.

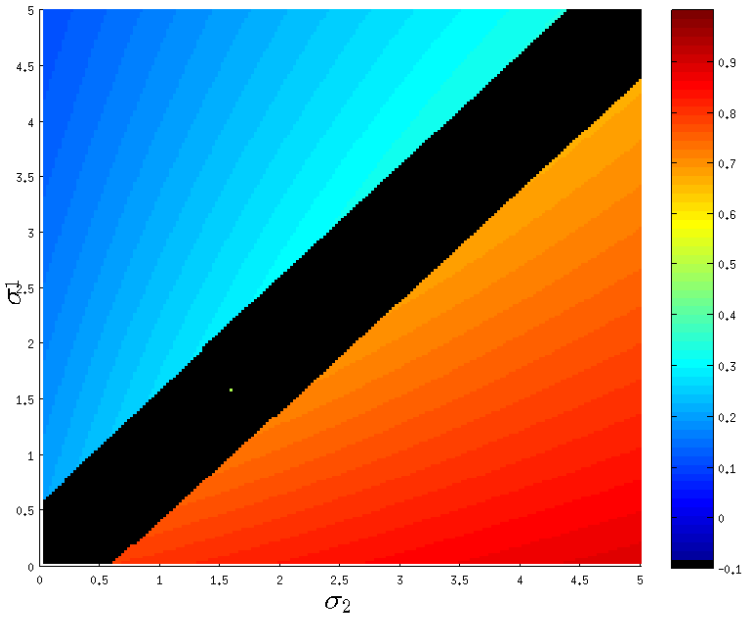


FIGURE 4.29: Stationary joining probability $p^*_1(\mathbf{\Gamma})$ as a function of σ_1 and σ_2 with remaining parameters $\lambda = 2,\ \mu_1 = 6,\ \mu_2 = 6,\ N_1 = 3,\ N_2 = 3,\ \tilde{p} = 0.5$. The points representing periodic policies have been assigned the value -0.1 and are depicted as black in the picture.

While showing interesting structure, the parameter space distribution conforms to our expectations.

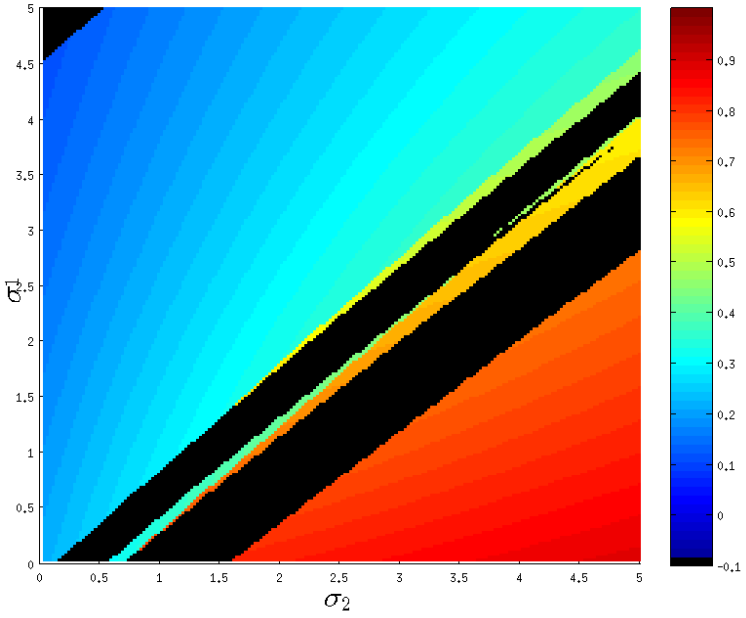


FIGURE 4.30: Stationary joining probability $p_1^*(\mathbf{\Gamma})$ as a function of σ_1 and σ_2 with remaining parameters $\lambda = 2$, $\mu_1 = 5.9$, $\mu_2 = 6.1$, $N_1 = 3$, $N_2 = 3$, $\tilde{p} = 0.5$. The points representing periodic policies have been assigned the value -0.1 and are depicted as black in the picture.

The change from symmetry to asymmetry of the service rates leads not only to a much more complex picture in parameter space, but also to the changes we would expect.

Stationary joining probabilities by σs, $[0, 1]$-updates

For the $[0, 1]$-updates we also have to chose a δ parameter, and we show the outcomes in the case of the symmetric service rates for three different values here. In Figure 4.31 on the following page we set $\delta = 0.001$.

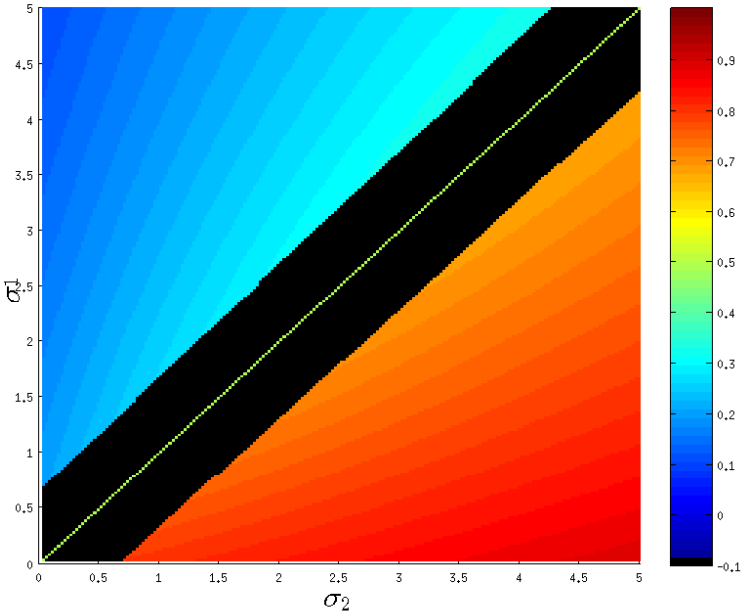
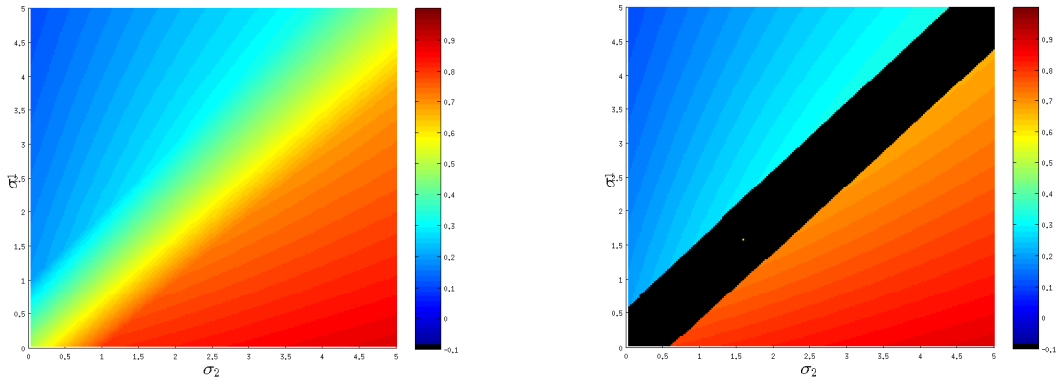


FIGURE 4.31: Stationary joining probability $p_1^*(\mathbf{\Gamma})$ as a function of σ_1 and σ_2 with remaining parameters $\lambda = 2,\ \mu_1 = 6,\ \mu_2 = 6,\ N_1 = 3,\ N_2 = 3,\ \delta = 0.001$. The points representing periodic policies have been assigned the value -0.1 and are depicted as black in the picture.

Figure 4.31 demonstrates one important aspect of the relationship between the $[0, 1]$- and the $\{0, 1, \tilde{p}\}$-updates; the fact that policies resulting from $[0, 1]$-updates can be fruitfully thought of as policies resulting from $\{0, 1, \tilde{p}\}$-updates with some added flexibility. In Figure 4.29 on page 154 there is a single point in amongst the blocked out periodic policies, hinting at the presence of a set of stationary policies that could be accessible with a sufficiently fine grid. In Figure 4.31 a line of fixed point policies is realised in the corresponding region despite the fact that the grid is no finer. This is not solely an effect of the difference between the two types of update rules, but the presence of that line also depends on the choice of δ. We chose the particular value of $\delta = 0.001$ in order to be able to show this intermediate stage between situations that even more closely correspond to the case with $\{0, 1, \tilde{p}\}$-updates for smaller δ, and the situation when we don't

observe any periodic policies in this particular parameter range for higher δ. For completeness we show both of these situations in Figure 4.32.



(A) $\delta = 0.01$ (B) $\delta = 0.00001$

FIGURE 4.32: Stationary joining probability $p_1^*(\mathbf{\Gamma})$ as a function of σ_1 and σ_2 for the same parameters as in Figure 4.31 on the preceding page but with different values of δ.

In the case of the asymmetric service rates we only consider the case of $\delta = 0.001$, and the outcomes can be found in Figure 4.33 on the following page.

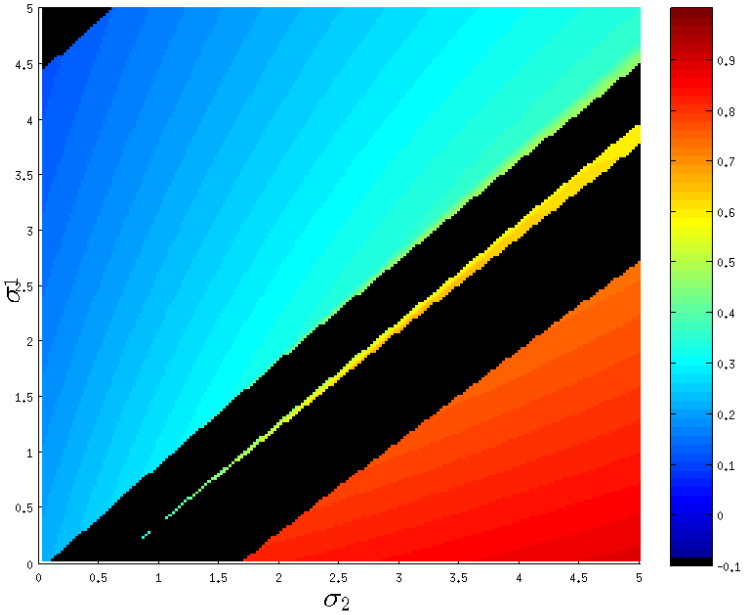


FIGURE 4.33: Stationary joining probability $p_1^*(\mathbf{\Gamma})$ as a function of σ_1 and σ_2 with remaining parameters $\lambda = 2,\ \mu_1 = 5.9,\ \mu_2 = 6.1,\ N_1 = 3,\ N_2 = 3,\ \delta = 0.001$. The points representing periodic policies have been assigned the value -0.1 and are depicted as black in the picture.

The changes in this figure compared to Figure 4.32 on the previous page again correspond to what we expect, but we note the similarities between this realisation and the corresponding one in the case of $\{0, 1, \tilde{p}\}$-updates, Figure 4.30 on page 155.

Stationary joining probabilities by σs, $(0, 1)$-updates

In the case of $(0, 1)$-updates we are not required to make any additional parameter choices. We show the outcomes in the case of symmetric service rates in Figure 4.34 on the next page and in the case of asymmetric service rates in Figure 4.35 on page 160.

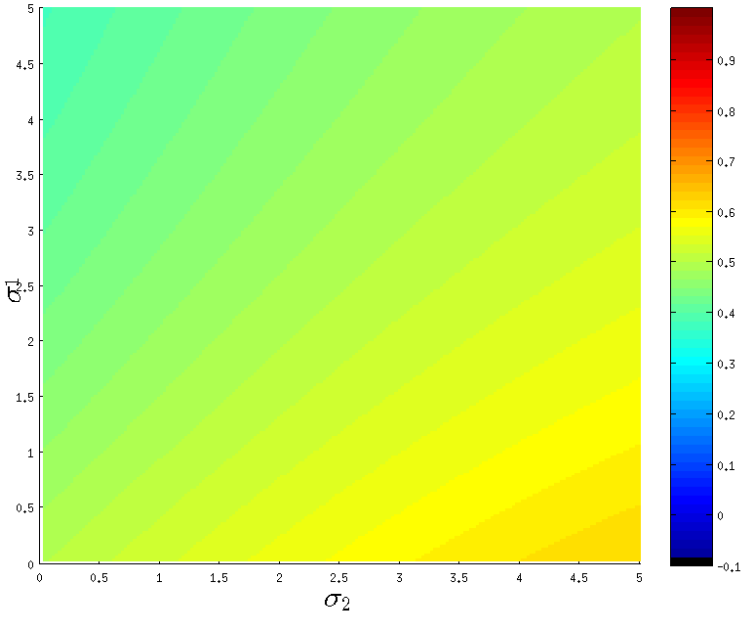


FIGURE 4.34: Stationary joining probability $p_1^*(\mathbf{\Gamma})$ as a function of σ_1 and σ_2 with remaining parameters $\lambda = 2$, $\mu_1 = 6$, $\mu_2 = 6$, $N_1 = 3$, $N_2 = 3$. All points in this image correspond to aperiodic monotonic policies.

Using the same colour scheme for all the realisations in this section leads to this figure looking quite monotonous. The stationary joining probability does however vary from $p_1^*(\mathbf{\Gamma}) = 0.38$ to $p_1^*(\mathbf{\Gamma}) = 0.62$, in a remarkably smooth manner.

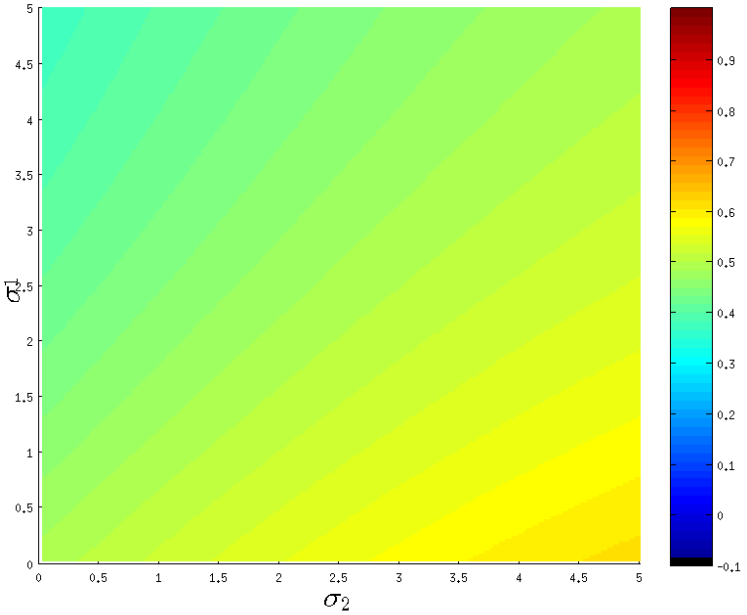


FIGURE 4.35: Stationary joining probability $p_1^*(\mathbf{\Gamma})$ as a function of σ_1 and σ_2 with remaining parameters $\lambda = 2$, $\mu_1 = 5.9$, $\mu_2 = 6.1$, $N_1 = 3$, $N_2 = 3$. All points in this image correspond to aperiodic monotonic policies.

The variation can again be hard to make out but in this case we see a smooth transition from $p_1^*(\mathbf{\Gamma}) = 0.31$ to $p_1^*(\mathbf{\Gamma}) = 0.62$. So while the introduction of the asymmetry only had a small effect, the effect is in the expected direction.

4.3 Conclusions from numerics

With this numeric work we set out to increase our understanding of the system behaviour in general, reproduce previous counter-intuitive results, to find out if those results could be mitigated by restrictions on parameter choices or changes in update rules and/or boundary conditions. We discuss these outcomes separately here.

4.3.1 Parameters and problematic policies

The reproduction of the complex pattern in Figure 4.1 on page 103 confirms the previously described behaviour. That we got the same kind of results for some of the changed update rules and boundary conditions indicates that careful consideration is required in the design of systems of this sort if one wishes to ensure that it is possible for human customers to find and utilise optimal policies. In particular we could not formulate any straightforward prescriptions in terms of parameter relationships to always ensure convergence to a monotonic fixed point policy.

The one, clear, tendency that ran as a common theme through the illustrations in 4.2.1 is that overloaded systems are more prone to producing non-monotonic fixed point policies, compared to underloaded systems. We have seen examples of non-monotonic policies in underloaded system, this can for example be found in the case with high capacity in Figure 4.9 on page 115 where some of the non-monotonic tendrils do extend into underloaded territory and Figure 4.17 where the non-monotonic regions also extend into the sub-critical region along the axis. However, we can see a clear tendency for non-monotonicity to coexist with overloading.

Restricting our study of these systems to the underloaded case is not a strong restriction, in terms of seeking to avoid problematic equilibrium policies for reasons we discuss below. A more disturbing tendency, that is again best exemplified through Figure 4.9 on page 115 and Figure 4.17 on page 131, but that has been observed in other situations as well, is that adding more capacity to the queues can apparently increase the incidence of problematic outcomes of policy iteration.

It is not reasonable within the context of this study to limit ourselves to systems below a particular size. In fact, part of the reason why we are open to restricting future study of the system to the underloaded region is that this would have to be a prerequisite for studying infinite versions of this system anyway. As we want to be able to study systems of arbitrary size, at least analytically, we simply have to take the indication that higher capacity tends to lead to more problematic policy iteration outcomes as a warning in relation to system design rather than a limitation on the parameter choices for future studies.

We also note that while increased capacity has been shown to lead to non-monotonic behaviour with $(0, 1)$-updates, we have seen no examples of periodicities with this update rule.

4.3.2 Boundary conditions, update rules, and policy iteration outcomes

While the system design aspect of this study yielded indications on how to design the system to minimise problems, as discussed in the preceding section, but no rules that can be expected to carry through to analytical work, the user behaviour aspect yielded much stronger results.

Most strikingly we found that the update rule employed in the policy iteration strongly affected system behaviour. The most important observation is the lack of periodicity as a result of policy iteration with $(0, 1)$-updates, regardless of the choice of parameters and boundary condition. We state this observation as Conjecture 4.3.1.

Conjecture 4.3.1. Policy iteration as defined in Algorithm 1 with the $(0, 1)$-updates as in Definition 4.1.3 is guaranteed to converge to a decision policy.

Meanwhile, we have shown by example that both of the other decision rules that we have tested can fail to converge, which we state as Theorem 4.3.1, proved by example above in Figure 4.1 on page 103 and Figure 4.5 on page 109.

Theorem 4.3.1. Let $\mathbf{D}^\tau$ and $\mathbf{D}^{\tau+1}$ be two adjacent policies in a realisation of policy iteration with parameters $\mathbf{\Gamma} \in \mathcal{G}$ operating under $\{0, 1, \tilde{p}\}$- or $[0, 1]$-updates with $\delta > 0$ and any boundary conditions defined above.
Then there exists a $\mathbf{\Gamma}$ such that $\mathbf{D}^\tau \neq \mathbf{D}^{\tau+1}, \forall \tau \in \mathbb{N}$.

Theorem 4.3.1 holds in the case when $\delta = 0$ as well, but as this is equivalent to $\{0, 1, \tilde{p}\}$ with $\tilde{p} = 0.5$ it is in some sense not really an example of $[0, 1]$-updates.

We can state Theorem 4.3.1 exactly without reference to policy iteration tolerance even in the case of $[0, 1]$-updates as we have observed examples of periodic policies in which each decision rule is pure, indicating that we are looking at an exact periodic orbit rather than non-monotonic convergence. This is shown in Example 4.2.1.

We have observed non-monotonic policies with all update rules, but not with all combinations of update rules and boundary conditions. For $\{0, 1, \tilde{p}\}$-updates we have seen non-monotonicities with all types of boundary conditions so Theorem 4.3.2 is proven by example in Figure 4.1 on page 103, Figure 4.4 on page 107 and Figure 4.3 on page 106. Again,

Theorem 4.3.2. Let $\mathbf{\Gamma} \in \mathcal{G}$ be a parameter vector and let $\mathbf{D}^*$ be the fixed point policy resulting from policy iteration with parameter vector $\mathbf{\Gamma}$ under $\{0, 1, \tilde{p}\}$-updates and any of the boundary conditions discussed above.
Then there exists a $\mathbf{\Gamma}$ such that $\mathbf{D}^*$ is not monotonic in the sense of Definition 2.3.3.

We have also shown for both $(0, 1)$-updates and $[0, 1]$-updates that they give rise to non-monotonic fixed point policies, but only with certain boundary conditions. In particular Figure 4.5 on page 109 and Figure 4.17 prove that with automatically routing boundaries we do indeed see non-monotonic policies with all update rules, which we state as Theorem 4.3.3.

Theorem 4.3.3. Let $\mathbf{\Gamma} \in \mathcal{G}$ be a parameter vector and let $\mathbf{D}^*$ be a fixed point policy resulting from policy iteration with any of the update rules discussed above and automatically routing boundaries.
Then there exists a $\mathbf{\Gamma}$ such that $\mathbf{D}^*$ is not monotonic in the sense of Definition 2.3.3.

We do note, however, that we did not manage to provoke any non-monotonicities in the case of $(0, 1)$-updates as long as we restricted the search to the part of parameter space where the capacity was less than or equal to 3 in the underloaded volume of parameter space. We find this quite interesting, since it is a clear cut boundary between desirable and undesirable system behaviour. This type of clear cut boundary has generally been rare in the work we present here. Thus we state the observation as a conjecture, Conjecture 4.3.2.

Conjecture 4.3.2. Let $\mathbf{D}^*$ be a fixed point policy found by policy iteration as defined in Algorithm 1 with the $(0, 1)$-updates as in Definition 4.1.3. Let the parameters of the system be such that $N_1 \leq 3, N_2 \leq 3$ and $\mu_1 + \mu_2 > \lambda + \sigma_1 + \sigma_2$. Then $\mathbf{D}^*$ is guaranteed to be monotonic in the sense of Definition 2.3.3.

Note that we have not seen any examples of non-monotonic policies with these updates and these capacities in the overloaded region either, but we are not comfortable making conjectures about the full parameter space, due to our experiences with the counter intuitive complexities inherent in this system.

Theorem 4.3.1 in conjunction with Conjecture 4.3.1 motivate us to proceed with the $(0, 1)$-update rule as our basis for the project described in Chapter 5.

This decision is also supported by the outcome of the calculations in Section 4.2.3. In this section we studied the probability that a general arrival who could join the system under rejection boundaries would join a specific queue. We found that regardless of the update rule that was employed for the policy iteration the stationary joining probability depended on service rates as well as intrinsic arrival rates in exactly the way that we would expect. However, in the cases of $\{0, 1, \tilde{p}\}$-updates as well as $[0, 1]$-updates the transition from low to high stationary joining probabilities occurred in a piecewise fashion in parameter space, broken up by swathes of periodic policies. In the case of the $(0, 1)$-updates this transition was obviously not broken up by periodic polices, as we have yet to see this update rule produce such, but beyond that the transition from low to high joining probabilities followed a smooth curve.

In Example 4.2.2 we saw further problematic aspects of the $[0, 1]$-update rules. In particular we saw the great impact that the choice of δ had on the resulting decision policy. Too low δ resulted in periodic behaviour. Too high δ on the other hand resulted in policies that had little to do with the parameters in question, as the probability of joining either queue approached 0.5 as δ got large. In a situation when we were without a simpler solution, or in a situation when $[0, 1]$-updates had other compelling properties as compared to the other proposed update rules, this might motivate us to explore how to optimally pick δ. In the situation when we are comparing the $[0, 1]$-update rule to the $(0, 1)$-update rule, the fact of the difficulty of choosing δ instead speaks in favour of $(0, 1)$-updates.

These facts, the intuitive relationship between stationary joining probabilities and system parameters, as the smooth transition in the case of $(0, 1)$-updates, the problem of choosing δ as well as the fact that the $[0, 1]$-updates are computationally much heavier than the other update rules added further motivation to proceed with the study of policy iteration in general and the $(0, 1)$-update rule in particular.

5

Modified policy iteration

You could find out most things, if you knew the right questions to ask. Even if you didn't, you could still find out a lot.

– Iain M. Banks, The Player of Games

In Chapter 4 we explored the policy iteration algorithm and found that in general it can give rise to periodic sets of policies and, slightly less problematic, non-monotonic policies. In this chapter we focus on analytically exploring the form of policy iteration that we identified in that chapter as the type of policy iteration that seemed most promising in terms of guaranteeing a fixed point policy. This means that when exploring whether policy iteration is guaranteed to have a fixed point we proceed with $(0, 1)$-updates.

We can relate the reformulation of the policy iteration from Chapter 4 to policy iteration as described in any of Puterman [21], White [25] or Feinberg & Shwartz [26]. In particular the formulation we use here draws on the concept of value iteration with respect to average expected reward over infinite time horizon as discussed in the first chapter of Feinberg & Shwartz [26]. Since our main interest in this method is related to what we can discern about the policy iteration based on it, we choose to think of it as *Modified Policy Iteration*.

5.1 Motivation

We consider the policy iteration algorithm, Algorithm 1 in Chapter 4, once again and particularly note that each iterative step includes the solution of a set of equations. This carries with it two problematic effects; one computational, and one related to analytical study of the process.

The computational problem is that the solution of the systems of equations requires two matrix inversions for each iterative step. Matrix inversion is a relatively computationally complex operation, the complexity of which grows rapidly with matrix size, for a square $n \times n$ matrix the complexity is $\mathcal{O}(n^3)$ Demmel [65]. Matrix size in this case is directly proportional to the size of the state space, $|\mathcal{S}|$, which is $((N_1 + 1) \times (N_2 + 1))$. This is thus a compound problem – the matrices grow rapidly with capacity, and the computational complexity grows rapidly in matrix size. This means that the complexity grows roughly at $\mathcal{O}\Big(\big((N_1+1)\cdot(N_2+1)\big)^3\Big)$. In practice this means that anything but the smallest systems become intractable for anything but the calculation of individual policies, as opposed to more comprehensive explorations of parameter space such as the interesting parameter space diagrams presented in Chapter 4.

The analytical problem is related to the fact that each expected waiting time depends on current decision policies and all other current expected waiting times. This means that the relationship between iterative steps is quite weak compared to the relationship within one iterative step. This limits the ability to make strong arguments based on the relationship between the outcomes of iterative steps. In this chapter we propose an alternative iterative update scheme.

While the computational issue motivates consideration of this algorithm, we do not return explicitly to computation in the following.

5.2 Definitions

We now define an iterative algorithm in which optimisation is carried out with respect to the expected outcome of a value function for a marked customer. In terms of its role in the updates of the iterative algorithm the value function is closely related to the expected waiting time from policy iteration. We therefore refer to the values as pseudo waiting times. In this case we do not solve for the exact expected waiting times at each step, though, but instead start out from a guessed set of *pseudo waiting times*, and then at each step use the pseudo waiting times from the previous step for our updates. We go on to show that this algorithm shares fixed points with policy iteration.

5.2.1 Modified policy iteration algorithm

Our algorithm is the following:

Algorithm 2 The policy iteration algorithm

1: Choose a $t_{\max}$
2: Guess a set of initial pseudo waiting times $\boldsymbol{\zeta}^0$.
3: Calculate $\mathbf{D}^0$
4: $t = 15$
5: **while** $\max\{|\boldsymbol{\zeta}^t - \boldsymbol{\zeta}^{t+1}|\} > \varepsilon$ AND $t \leq t_{\max}$ **do**
6: Using $\boldsymbol{\zeta}^{t-1}$ calculate the updated pseudo waiting times $\zeta^t_{1,\mathbf{D}^t}(\mathbf{n})$ and $\zeta^t_{2,\mathbf{D}^t}(\mathbf{n})$ for every $\mathbf{n} \in \mathcal{S}$.
7: $t = t + 1$
8: **end while**
9: **if** $t < t_{\max}$ **then**
10: $\boldsymbol{\zeta}^* = \boldsymbol{\zeta}^t$
11: Using $\boldsymbol{\zeta}^*$ calculate $\mathbf{D}^*$
12: **else**
13: The iteration has failed to produce a fixed point policy.
14: **end if**

We explain Algorithm 2 by focusing on the differences between this and Algorithm 1. The main difference can arguably be found in step 6 where the pseudo waiting times from the previous steps are used to find the updated pseudo waiting times, while in the previous algorithm only the decision policy of the previous step was taken as input when the updated expected waiting times were calculated via matrix inversion. This is the concession we make in order to avoid solving a system of equations at each step. For the same reason we also now need to guess an initial set of pseudo waiting times, $\boldsymbol{\zeta}^0$, as opposed to just an initial decision policy.

We mention at this stage that the reason that there is no need to explicitly calculate decision policies at each step is related to the fact that we know that we are using $(0, 1)$-updates. Thus we have chosen to explicitly include the calculation of the relevant decision rule into the function representing the update rule. This is discussed further below, in the context of the explicit form of pseudo waiting time update rule, Equation 5.1.

We explicitly state the update rule for pseudo waiting times, step 6 in Algorithm 2, as Equation 5.1. In this formulation we employ null events, so we allow services to occur in empty, as well as arrivals to full, queues. These null events do not change the state of the queue, though. So with null events $\Lambda = \lambda + \sum_{j\in\mathcal{M}}(\mu_j + \sigma_j)$ is the expected time between events in the process, regardless of the state the system is in.

$$\begin{aligned}\zeta_{i,t}(\mathbf{n}) = \\ \Lambda^{-1}\Big(1 + \sum_{j\in\mathcal{M}} I_{\{n_j+I_{\{j=i\}}<N_j\}}\big[\lambda R_j(\mathbf{n}, \boldsymbol{\zeta}_{t-1}(\mathbf{n}+\mathbf{e}_i)) + \sigma_j\big]\zeta_{i,t-1}(\mathbf{n}+\mathbf{e}_j) \\ + \big[\lambda\big(1 - \sum_{j\in\mathcal{M}} I_{\{n_j+I_{\{j=i\}}<N_j\}} R_j(\mathbf{n}, \boldsymbol{\zeta}_{t-1}(\mathbf{n}+\mathbf{e}_i))\big) + \sum_{j\in\mathcal{M}} I_{\{n_j+I_{\{j=i\}}=N_j\}}\sigma_j\big]\zeta_{i,t-1}(\mathbf{n}) \\ + \mu_i \frac{n_i}{n_i+1}\zeta_{i,t-1}(\mathbf{n}-\mathbf{e}_i) + \sum_{j\in\mathcal{M}\setminus i} \mu_j \zeta_{i,t-1}(\mathbf{n} - I_{\{n_j>0\}}\mathbf{e}_j)\Big), \\ \mathbf{n}\in\mathbf{S}, i\in\mathcal{M}, t\geq 0. \quad (5.1)\end{aligned}$$

In Equation 5.1 we find the new quantity $R_j(\mathbf{n}, \boldsymbol{\zeta}_{t-1}(\mathbf{n}+\mathbf{e}_i))$, given in Definition 5.2.1, which is related to the decision rule in Chapter 4. We return to this quantity below.

We also give the equations for the boundary state in the case of $\hat{z}$-routing boundaries, which we here refer to as $\hat{\zeta}$-routing boundaries. This is the natural extension from $M = 2$ to general M dimensions. As previously users are allowed to balk in the states in which at least one queue is full; now the probability of balking is based on all of the pseudo waiting times from the states with room left, as well as all of the pseudo waiting times from the ones that do not. In this formulation most of that structure is contained within the function $R_j(\mathbf{n}, \zeta_{t-1}(\mathbf{n}))$, discussed in detail below.

$$\hat{\zeta}_{i,t}(\mathbf{n}) = \Lambda^{-1}\Big(1 + \sum_{j\in\mathcal{M}} I_{\{n_j<N_j\}}[\lambda R_j(\mathbf{n},\boldsymbol{\zeta}_{t-1}(\mathbf{n})) + \sigma_j]\hat{\zeta}_{i,t-1}(\mathbf{n}+\mathbf{e}_j) + \big[\lambda\big(1 - \sum_{j\in\mathcal{M}} I_{\{n_j<N_j\}}R_j(\mathbf{n},\boldsymbol{\zeta}_{t-1}(\mathbf{n}))\big) + \sum_{j\in\mathcal{M}} \sigma_j I_{\{n_j+I_{\{j=i\}}=N_j\}}\big]\hat{\zeta}_{i,t-1}(\mathbf{n}) + \mu_i\frac{N_i}{N_i+1}\zeta_{i,t-1}(\mathbf{n}-\mathbf{e}_i) + \sum_{j\in\mathcal{M}\setminus i}\mu_j\hat{\zeta}_{1,t-1}(\mathbf{n}-\mathbf{e}_j I_{\{n_j>0\}})\Big), \quad \mathbf{n}\in\mathbf{S}, i\in\mathcal{M}, t\geq 0. \quad (5.2)$$

When $M = 2$ Function 5.1 is very similar to Equation 2.5. There are a few differences, though. The most notable difference between the formulations, except for the above mentioned restructuring in terms of null-steps, is the fact that we have here replaced the statement of externally calculated decision rules with explicit functions for calculating $R_j(\mathbf{n},\boldsymbol{\zeta}_{t-1}(\mathbf{n}+\mathbf{e}_i))$ in terms of the outcomes from the previous steps of the iteration. In the present case we consider only $(0,1)$-policies, in which case $R_j(\mathbf{n},\boldsymbol{\zeta}_{t-1}(\mathbf{n}+\mathbf{e}_i))$ has the explicit form given in Definition 5.2.1.

Definition 5.2.1.

$$R_j(\mathbf{n},\boldsymbol{\zeta}_{t-1}(\mathbf{n})) = \frac{\zeta_{i,t-1}(\mathbf{n})^{-1}}{\sum_{j\in\mathcal{M}}\bar{\zeta}_{j,t-1}(\mathbf{n})^{-1}}, j\in\mathcal{M}, \mathbf{n}\in\mathcal{S}, \boldsymbol{\zeta}\in\mathbb{R}^{M\prod_{i\in\mathcal{M}}N_i}$$

where $\bar{\zeta}_{j,t-1}(\mathbf{n})$ indicates that the explicit calculation of the pseudo waiting time in question depends on boundary conditions and state. If the state $\mathbf{n}$ is such that $\mathbf{n}\in\mathcal{S}^-$, or the system is operating according to automatically routing boundaries, then $\bar{\zeta}_{j,t-1}(\mathbf{n}) = \zeta_{j,t-1}(\mathbf{n})I_{\{n_j<N_j\}}$. If the system is operating under $\hat{\zeta}$-routing boundaries, and the state $\mathbf{n}$ is such that $n_i = N_i$ then $\bar{\zeta}_{j,t-1}(\mathbf{n}) = \zeta_{j,t-1}(\mathbf{n})I_{\{n_j<N_j\}} + \hat{\zeta}_{j,t-1}(\mathbf{n})I_{\{n_j=N_j\}}$.

We also note that the definition of $R_j(\mathbf{n},\boldsymbol{\zeta}_{t-1}(\mathbf{n}))$ given in Definition 5.2.1 is such that we do not explicitly define the probability of balking for a general arrival finding the system in state $\mathbf{n}$, $R_0(\mathbf{n},\boldsymbol{\zeta}_{t-1}(\mathbf{n}))$, following the notation from previous chapters. Instead, by letting the numerator in that expression be a function of $\zeta_{i,t-1}(\mathbf{n})^{-1}$ rather than $\bar{\zeta}_{j,t-1}(\mathbf{n})$ we have, in the case of internal states, Equation

5.1, $R_0(\mathbf{n}, \boldsymbol{\zeta}_{t-1}(\mathbf{n})) = \left(1 - \sum_{j \in \mathcal{M}} I_{\{n_j + I_{\{j=i\}} < N_j\}} R_j(\mathbf{n}, \boldsymbol{\zeta}_{t-1}(\mathbf{n} + \mathbf{e}_i))\right)$. We see the same construction, appropriately adjusted, in the expression related to the edge states in Equation 5.2.

However, the main difference between the present formulation and policy iteration, which we have touched upon, is that this is an expression in the pseudo waiting times from the previous step, rather than the expected waiting times from the current one. Thus the role of the explicit waiting times from the same iterative step, $z_{i,\mathbf{D_t}}(\mathbf{n})$, in Equation 2.5 is instead performed by pseudo waiting times from the previous step, $\zeta_{i,t-1}(\mathbf{n} + \mathbf{e}_i)$, in equation 5.1.

One way to think of this, is that this method is a kind of backwards induction, in which case we think of the time parameter t as the number of events left until the process ends. In particular this perspective becomes clear in the case when we let the initial pseudo waiting times be constant and $\zeta_{i,0}(\mathbf{n}) = \varepsilon, \forall i \in \mathcal{M}, \mathbf{n} \in \mathcal{S}$ where ε is some small positive quantity. This represents the situation that a marked customer who is in the system when there are 0 events left in the process will be served essentially immediately. The iterative updates then represent progressively adding more events before the situation that the marked customer is guaranteed to be served almost immediately arises. In this view convergence of the iteration represents finding the expected waiting time for a user who with high probability has been served before the system reaches the special situation at $t = 0$.

We however note that despite the slight reformulation the overall structure of the expression as well as the component parts are essentially identical.

With this formulation our criterion for convergence changes. In the case of policy iteration there is, for each parameter set $\boldsymbol{\Gamma}$, a function

$$\mathbf{f} : [0,1]^{M \cdot \prod_{i \in \mathcal{M}}(N_i+1)} \to \mathbb{R}_+^{M \cdot \prod_{i \in \mathcal{M}}(N_i+1)}$$

which gives a set of expected waiting times for each decision policy, this is Equation 2.5, while the policy updates represent a function

$$\mathbf{g} : \mathbb{R}_+^{M \prod_{i \in \mathcal{M}}(N_i+1)} \to [0,1]^{M \prod_{i \in \mathcal{M}}(N_i+1)}$$

in the other direction. In the case of policy iteration this is not an issue as each update relies only on the policies from the previous step so that

$$\mathbf{D}^t = \mathbf{D}^{t+1} \Rightarrow \mathbf{Z}^{t+1} = \mathbf{Z}^{t+2}.$$

By contrast the updates in Modified policy iteration depend on pseudo waiting times from the previous step as well, so $\mathbf{D}^t = \mathbf{D}^{t+1}$ in decision policies is not sufficient to guarantee that the iterative process has also converged in terms of pseudo waiting times. This can be understood by considering the fact that with $(0, 1)$-updates, see Definition 4.1.3, $\mathbf{g}(\mathbf{Z}) = \mathbf{g}(k \cdot \mathbf{Z}) \; \forall \; k \neq 0$. Thus we consider the case when a step in Algorithm 2 is such that, for some set of pseudo waiting times $\boldsymbol{\zeta}$ and real number k, we found that $\boldsymbol{\zeta}^t = \boldsymbol{\zeta}$ while $\boldsymbol{\zeta}^{t+1} = k\boldsymbol{\zeta}$. This would lead to $\mathbf{D}^t = \mathbf{D}^{t+1}$ even though the iteration clearly hasn't reached a fixed point. On the other hand convergence in pseudo waiting times implies convergence in decision policy,

$$\boldsymbol{\zeta}^t = \boldsymbol{\zeta}^{t+1} \Rightarrow \mathbf{D}^t = \mathbf{D}^{t+1},$$

so we use the pseudo waiting times as our criteria for convergence.

For completeness and clarity we mention that there are theoretical problems associated with the fact that we have situations when $g(\mathbf{Z}_1) = g(\mathbf{Z}_2)$ despite $\mathbf{Z}_1 \neq \mathbf{Z}_2$ in policy iteration as well. However, in the case of policy iteration this leads to periodic behaviours rather than false instances of apparent convergence.

5.2.2 Policy iteration and modified policy iteration fixed point identity

In order for us to be able to use any fixed point results found by the modified policy iteration algorithm when making statements about fixed points in policy iteration we need to ensure that the two algorithms do in fact share fixed points. This is indeed the case, which we state as Theorem 5.2.1

Theorem 5.2.1. Let $\mathcal{F}_p(\boldsymbol{\Gamma})$ be the set of all sets of waiting times for fixed points of policy iteration with parameter vector $\boldsymbol{\Gamma}$, and let $\mathcal{F}_m(\boldsymbol{\Gamma})$ be the set of fixed point pseudo waiting times for modified policy iteration, then,

$$\mathbf{X} \in \mathcal{F}_p(\boldsymbol{\Gamma}) \Leftrightarrow \mathbf{X} \in \mathcal{F}_m(\boldsymbol{\Gamma}).$$

For completeness we mention that a shared fixed point in waiting times or pseudo waiting times implies a fixed point in decision policy, assuming identical policy update rules. We prove this in the context of $\hat{z}$-routing boundaries with $(0, 1)$-updates. As we can view automatically routing boundaries as a special case of $\hat{z}$-routing boundaries the proof applies to the case of automatically routing

boundaries as well. It does, however, not give us identity between the two methods in the case of other policy update rules.

Proof. In order to see that the modified policy iteration has the same fixed points as the policy iteration we show that while the two schemes rely on different functions when not at a fixed point both iterative schemes actually conform to the same set of equations at a fixed point, and thus have the same fixed points.

We first note that each step in the policy iteration algorithm can be viewed as a vector valued function of the current expected waiting times. So the policy iteration can be expressed as $\mathbf{g}(\mathbf{Z}^t) = \mathbf{Z}^{t+1}$, until convergence when $\mathbf{g}(\mathbf{Z}^*) = \mathbf{Z}^*$. The function $\mathbf{g}$ consists of multiple distinct steps. First the decision policy $\mathbf{D}^t$ is found by application of the policy update rule to the expected waiting times $\mathbf{Z}^t$. Then the decision policy $\mathbf{D}^t$ is used as input to Equation 2.5 and the updated expected waiting times are found by solution of the system of equations.

It is relevant in the following to note, again, that $\sum_{i=0}^{M} D_i^t(\mathbf{n}) = 1$ for each $\mathbf{n} \in \mathcal{S}$, which means that the specifics of finding $D_0(\mathbf{n})$ does not matter in this comparison. This is due to the fact that we can work in M, rather than the full $M+1$, marginal policies, thereby letting the balking probability enter into calculations only implicitly.

The updated decision policies are calculated such that $D_i(\mathbf{n}) > 0$ for $\mathbf{n}$ such that $n_i \neq N_i$ and $D_i(\mathbf{n}) = 0$ for $\mathbf{n}$ such that $n_i = N_i$ while $D_0(\mathbf{n}) = 0$ for $\mathbf{n}$ such that $n_i \neq N_i$, $\forall i \in \mathcal{M}$ and $D_0(\mathbf{n}) \geq 0$ for $\mathbf{n}$ such that $\exists i \in \mathcal{M}$ s.t. $n_i = N_i$. Thus:

$$\mathbf{g} : \mathbb{R}_+^{M \cdot \prod_{i \in \mathcal{M}}(N_i+1)} \to \mathbb{R}_+^{M \cdot \prod_{i \in \mathcal{M}}(N_i+1)}. \tag{5.3}$$

Likewise we can express the value iteration as a vector valued function, but in this case of the previous step's pseudo waiting times, $\boldsymbol{\zeta}^{t-1}$. We express this as $\mathbf{h}(\boldsymbol{\zeta}^t) = \boldsymbol{\zeta}^{t+1}$ until convergence, when $\mathbf{h}(\boldsymbol{\zeta}^*) = \boldsymbol{\zeta}^*$. The function $\mathbf{h}$ is a lot more straightforward than $\mathbf{g}$ as it simply consists of evaluating Equations 5.1 and 5.2 at $\boldsymbol{\zeta}^t$. So we can conclude that:

$$\mathbf{h} : \mathbb{R}_+^{M \cdot \prod_{i \in \mathcal{M}}(N_i+1)} \to \mathbb{R}_+^{M \cdot \prod_{i \in \mathcal{M}}(N_i+1)}. \tag{5.4}$$

Now, in order to construct the function, $\mathbf{f}$, that is equivalent to $\mathbf{g}$ and $\mathbf{h}$ at stationarity we first note that we need to let $\mathbf{f}$ be a vector valued function such that:

$$\mathbf{f} : \mathbb{R}_+^{M \cdot \prod_{i \in \mathcal{M}}(N_i+1)} \to \mathbb{R}_+^{M \cdot \prod_{i \in \mathcal{M}}(N_i+1)} \tag{5.5}$$

Equivalent to the difference in calculation for the pseudo waiting times, Equation 5.1, and the hypothetical pseudo waiting times, Equation 5.2, as well as the corresponding equations for policy iteration, each $\mathbf{f}_{i,\mathbf{n}}$ also consists of one set of equations for when $\mathbf{n}$ is such that $n_i \neq N_i$ and another set of equations for when $n_i = N_i$. The two sets of equations are presented in Equations 5.6 and 5.7. Here the input $\mathbf{x}$ consists of M matrices $x_1, ..., x_M$ each of which has dimensions $N_1 \times ... \times N_M$ so that $x_i(\mathbf{n})$ is the entry at position $\mathbf{n} = (n_1, ..., n_M)$ in matrix number i. So $f_{i,\mathbf{n}}(\mathbf{x})$ should be interpreted as the evaluation of function number i at state $\mathbf{n}$ for the values in $\mathbf{x}$.

$$
\begin{aligned}
x_i(\mathbf{n}) = f_{i,\mathbf{n}}(\mathbf{x}) = \\
&\Big(\mu_i + \sum_{j \in \mathcal{M} \setminus i} \mu_j I_{\{n_j > 0\}} + \sigma_i I_{\{n_i+1 < N_i\}} + \sum_{j \in \mathcal{M} \setminus i} \sigma_j I_{\{n_j < N_j\}} \\
&+ \lambda\big(D_i(\mathbf{n} + \mathbf{e_i}) I_{\{n_i+1 < N_i\}} + \sum_{j \in \mathcal{M} \setminus i} D_j(\mathbf{n} + \mathbf{e}_i) I_{\{n_j < N_j\}}\big)\Big)^{-1} \\
&\cdot \Big(1 + \big(\sigma_i + \lambda D_i(\mathbf{n} + \mathbf{e}_i)\big) I_{\{n_i+1 < N_i\}} x_i(\mathbf{n} + \mathbf{e}_i) \\
&+ \sum_{j \in \mathcal{M} \setminus i} (\sigma_j + \lambda D_j(\mathbf{n} + \mathbf{e}_i)) I_{\{n_j < N_j\}} x_i(\mathbf{n} + \mathbf{e}_j) \\
&+ \mu_i \cdot \frac{n_i}{n_i + 1} \cdot x_i(\mathbf{n} - \mathbf{e}_i) + \sum_{j \in \mathcal{M} \setminus i} \mu_j I_{\{n_j > 0\}} x_i(\mathbf{n} - \mathbf{e}_j)\Big) \quad (5.6)
\end{aligned}
$$

We also write a set of equations that correspond to boundary calculations in the case of $\hat{z}$-routing boundaries. This is given in Equation 5.7.

$$\begin{aligned}\hat{x}_i(\mathbf{n}) = f_{i,\mathbf{n}}(\mathbf{x}) = \\ &\Big(\sum_{j\in\mathcal{M}\setminus i} \sigma_j I_{\{n_j<N_j\}} + \lambda \sum_{j\in\mathcal{M}\setminus i} D_i(\mathbf{n}+\mathbf{e}_i) I_{\{n_j<N_j\}} \\ &\qquad + \mu_i + \sum_{j\in\mathcal{M}\setminus i} \mu_j \cdot I_{\{n_j>0\}} \Big)^{-1} \\ &\cdot \Big(1 + \sum_{j\in\mathcal{M}\setminus i} (\sigma_j + \lambda D_i(\mathbf{n}+\mathbf{e}_i) I_{\{n_j<N_j\}} x_i(\mathbf{n}+\mathbf{e}_j) \\ &\qquad + \mu_i \frac{N_i}{N_i+1} x_i(\mathbf{n}-\mathbf{e}_i) + \sum_{j\in\mathcal{M}\setminus i} \mu_j I_{\{n_j>0\}} x_i(\mathbf{n}-\mathbf{e}_j) \Big) \end{aligned} \tag{5.7}$$

Here $D_j(\mathbf{n})$ represents the policy update function employed, so that for example in the case of $(0,1)$-updates we have

$$D_j(\mathbf{n}) = \frac{x_i(\mathbf{n})^{-1}}{\sum_{j\in\mathcal{M}} \bar{x}_j(\mathbf{n})^{-1}},$$

where $\bar{x}_j(\mathbf{n})$ indicates that the explicit calculation of the denominator depends on boundary conditions and state. If the state $\mathbf{n}$ is such that $\mathbf{n} \in \mathcal{S}^-$, or the system is operating according to automatically routing boundaries, then $\bar{x}_j(\mathbf{n}) = x_j(\mathbf{n}) I_{\{n_j<N_j\}}$. If the system is operating under $\hat{x}$-routing boundaries, and the state $\mathbf{n}$ is such that $n_i = N_i$ then $\bar{x}_j(\mathbf{n}) = x_j(\mathbf{n}) I_{\{n_j<N_j\}} + \hat{x}_j(\mathbf{n}) I_{\{n_j=N_j\}}$.

It is clear that at a fixed point these functions 5.6 and 5.7 correspond to Equations 2.5 and 4.1. So now what remains is to restate these equations into forms that are obviously identical to 5.1 and 5.2 at a fixed point. The manipulation is

trivial so we simply state the results. Here, again, $\Lambda = \lambda + \sum_{j\in\mathcal{M}}(\mu_j + \sigma_j)$.

$$\begin{aligned} x_i(\mathbf{n}) = f_{i,\mathbf{n}}(\mathbf{x}) = \\ \Lambda^{-1}\Big(1 + \sum_{j\in\mathcal{M}} I_{\{n_j+I_{\{j=i\}}<N_j\}}\big[\lambda R_j(\mathbf{n},\mathbf{x}(\mathbf{n}+\mathbf{e}_i)) + \sigma_j\big]x_i(\mathbf{n}+\mathbf{e}_j) \\ + \big[\lambda\big(1 - I_{\{n_j+I_{\{j=i\}}<N_j\}}\sum_{j\in\mathcal{M}} R_j(\mathbf{n},\mathbf{x}(\mathbf{n}+\mathbf{e}_i))\big) + \sum_{j\in\mathcal{M}} I_{\{n_j+I_{\{j=i\}}=N_j\}}\sigma_j\big]x_i(\mathbf{n}) \\ + \mu_i\frac{n_i}{n_i+1}xi(\mathbf{n}-\mathbf{e}_i) + \sum_{j\in\mathcal{M}\setminus i}\mu_j x_i(\mathbf{n} - I_{\{n_j>0\}}\mathbf{e}_j)\Big) \end{aligned} \tag{5.8}$$

We also write a set of equations that correspond to boundary calculations in the case of $\hat{z}$-routing boundaries. This is given in Equation 5.9.

$$\begin{aligned} \hat{x}_i(\mathbf{n}) = f_{i,\mathbf{n}}(\mathbf{x}) = \\ \Lambda^{-1}\Big(1 + \sum_{j\in\mathcal{M}} I_{\{n_j<N_j\}}[\lambda R_j(\mathbf{n},\mathbf{x}(\mathbf{n})) + \sigma_j]\hat{x}_i(\mathbf{n}+\mathbf{e}_j) \\ + \big[\lambda\big(1 - \sum_{j\in\mathcal{M}} I_{\{n_j<N_j\}}R_j(\mathbf{n},\mathbf{x}(\mathbf{n}))\big) + \sum_{j\in\mathcal{M}}\sigma_j I_{\{n_j+I_{\{j=i\}}=N_j\}}\big]\hat{x}_i(\mathbf{n}) \\ + \mu_i\frac{N_i}{N_i+1}x_{i,t-1}(\mathbf{n}-\mathbf{e}_i) + \sum_{j\in\mathcal{M}\setminus i}\mu_j\hat{x}_{1,t-1}(\mathbf{n} - \mathbf{e}_j I_{\{n_j>0\}})\Big) \end{aligned} \tag{5.9}$$

where $R_j(\mathbf{n},\mathbf{x}(\mathbf{n}+\mathbf{e}_i))$ represents the policy update and so in the case of $(0,1)$-updates conforms to Definition 5.2.1 with suitable adjustment to the lack of a time dimension.

We note that in the case that $\boldsymbol{\zeta}_t = \boldsymbol{\zeta}_{t-1}$ Equations 5.1 and 5.2 are identical to Equations 5.8 and 5.9, which are just reformulations of Equations 5.6 and 5.7, which are in turn, at a fixed point, identical to Equations 2.5 and 5.2.

Thus, as $\mathbf{g}$, $\mathbf{h}$ and $\mathbf{f}$ are completely equivalent at a fixed point, the three functions necessarily have the same fixed points. This proves that a fixed point of the value iteration is also a fixed point of the policy iteration, and vice versa. □

Theorem 5.2.1 states that if we let the pseudo waiting times $\boldsymbol{\zeta}^*$ be the fixed point of modified policy iteration with parameter vector $\boldsymbol{\Gamma}$ then we can be certain that the set of expected waiting times $\mathbf{Z}^*$ that is such that $\mathbf{Z}^* = \boldsymbol{\zeta}^*$ is a fixed point

of policy iteration with parameters $\mathbf{\Gamma}$. This means that any statement we can make about the properties of fixed points of modified policy iteration also holds for fixed points of policy iteration.

Theorem 5.2.1 does not tell us anything else about the relationship between policy iteration and modified policy iteration. In particular it does not mean that the transients leading up to convergence will be in any way related between the two algorithms.

It does however mean that inasmuch as we can use direct manipulation to make statements about fixed points of modified policy iteration, whatever conclusions we come to will be applicable to fixed points of policy iteration as well.

5.2.3 Existence of fixed points

We now go on to study the existence of fixed points in modified policy iteration by direct manipulation of the pseudo waiting time update rule. The proof consists of showing that the modified policy iteration algorithm is a continuous function mapping a closed subset of $\mathbb{R}^n$ into itself, which by Brouwer's fixed point theorem implies there is at least one fixed point.

Statements of Brouwer's fixed point theorem can be found in a variety of forms in many places. We give a definition that is equivalent to the aspects of the statement in Chapter 5 of Granas [66] that are relevant to our work, and in different notation.

Theorem 5.2.2 (Brouwer's Fixed Point Theorem)**.** Let $\mathcal{K}^n$ be a closed convex subset of $\mathbb{R}^n$, then every continuous function $\mathbf{f}$ such that

$$\mathbf{f} : \mathcal{K}^n \to \mathcal{K}^n$$

has at least one fixed point.

We then show that $\mathbf{g}$ does in fact map a closed subset of $\mathbb{R}^n$ into itself. We state this as Lemma 5.2.1.

Lemma 5.2.1. Let $\mathbf{g}$ represent the update step in modified policy iteration as defined in Algorithm 2 and Equations 5.1 and 5.2 with rejection boundaries and parameter vector $\mathbf{\Gamma}$. Let $\mathcal{A} \subset \mathbb{R}^M$such that each point $\mathbf{a} = (a_1, ..., a_m) \in \mathcal{A}$ satisfies

$$\frac{1}{\mu_i} \leq a_i \leq \frac{N_i}{\mu_i}, 1 \leq i \leq M,$$

then

$$\mathbf{a} \in \mathcal{A} \Rightarrow \mathbf{g}(\mathbf{a}) \in \mathcal{A}$$

Proof. The proof of Lemma 5.2.1 is straightforward. It consists of finding the values $\mathbf{a}^+$ and $\mathbf{a}^-$ in $\mathcal{A}$ that correspond to the largest and smallest image with respect to the function $\mathbf{g}$, and then showing that these images also fall in $\mathcal{A}$.

The first step here is to note that it is clear that in order for the iterative process to approximate policy iteration $\boldsymbol{\zeta}^0$ should be chosen in such a way that the pseudo waiting times could be expected waiting times for a processor sharing system with the parameters in question. A user in queue i never gets through such a system quicker than when they are alone for the entirety of their service, and the conditional expected sojourn time in this case is $\frac{1}{\mu_i}$. Conversely, it never takes longer than it does when a user has to share the service with a full queue for the whole time that they are in the system, in which case the conditional expected sojourn time is $\frac{N_i}{\mu_i}$.

We will show that for any $a \in \mathcal{A} = \prod_{i\in\mathcal{M}}[\frac{1}{\mu_i}, \frac{N_i}{\mu_i}]$, $\mathbf{g}(\mathbf{a}) \in \mathcal{A}$. Both Equations 5.1 and 5.2 have pseudo waiting times only in the numerators, so the largest image is associated with the largest pre-image and the smallest image with the smallest pre-image.

We start by considering the lower bound by letting $\zeta_i^0(\mathbf{n}) = \frac{1}{\mu_i}, \forall \mathbf{n} \in \mathcal{S}$ in Equation 5.10.

$$\begin{aligned}\zeta_{i,t}(\mathbf{n}) = \Lambda^{-1}\Big(1 + \sum_{j\in\mathcal{M}} I_{\{n_j+I_{\{j=i\}}<N_j\}}\big[\lambda R_j(\mathbf{n}, \boldsymbol{\zeta}_{t-1}(\mathbf{n}+\mathbf{e}_i)) + \sigma_j\big]\frac{1}{\mu_i} \\ + \big[\lambda\big(1 - \sum_{j\in\mathcal{M}} I_{\{n_j+I_{\{j=i\}}<N_j\}} R_j(\mathbf{n}, \boldsymbol{\zeta}_{t-1}(\mathbf{n}+\mathbf{e}_i))\big) + \sum_{j\in\mathcal{M}} I_{\{n_j+I_{\{j=i\}}=N_j\}}\sigma_j\big]\frac{1}{\mu_i} \\ + \mu_i\frac{n_i}{n_i+1}\frac{1}{\mu_i} + \sum_{j\in\mathcal{M}\setminus i}\mu_j\frac{1}{\mu_i}\Big) \quad (5.10)\end{aligned}$$

which we rearrange into

$$\begin{aligned}\zeta_{i,t}(\mathbf{n}) = \Lambda^{-1}\Big(1 + \frac{1}{\mu_i}\sum_{j\in\mathcal{M}}\sigma_j + \frac{n_i}{n_i+1} + \frac{1}{\mu_i}\sum_{j\in\mathcal{M}\setminus i}\mu_j \\ + \lambda\frac{1}{\mu_i}\big[\sum_{j\in\mathcal{M}} I_{\{n_j+I_{\{j=i\}}<N_j\}}R_j(\mathbf{n},\boldsymbol{\zeta}_{t-1}(\mathbf{n}+\mathbf{e}_i)) \\ + \big(1 - \sum_{j\in\mathcal{M}} I_{\{n_j+I_{\{j=i\}}<N_j\}}R_j(\mathbf{n},\boldsymbol{\zeta}_{t-1}(\mathbf{n}+\mathbf{e}_i))\big)\big]\Big) \quad (5.11)\end{aligned}$$

and further to

$$\begin{aligned}\zeta_{i,t}(\mathbf{n}) &= \Lambda^{-1}\Big(1 + \frac{n_i}{n_i+1} + \frac{1}{\mu_i}\sum_{j\in\mathcal{M}}\sigma_j + \lambda\frac{1}{\mu_i} + \frac{1}{\mu_i}\sum_{j\in\mathcal{M}\setminus i}\mu_j\Big) \\ &= \Lambda^{-1}\frac{1}{\mu_i}\Big(\mu_i + \mu_i\frac{n_i}{n_i+1} + \sum_{j\in\mathcal{M}}\sigma_j + \lambda + \sum_{j\in\mathcal{M}\setminus i}\mu_j\Big) \\ &= \Lambda^{-1}\frac{1}{\mu_i}\Big(\mu_i\frac{n_i}{n_i+1} + \lambda + \sum_{j\in\mathcal{M}}(\sigma_j+\mu_j)\Big) \geq \frac{1}{\mu_i} \quad (5.12)\end{aligned}$$

where the last inequality holds due to the fact that $\Lambda = \Big(\lambda + \sum_{j\in\mathcal{M}}(\sigma_j+\mu_j)\Big)$ and $\mu_i\frac{n_i}{n_i+1} \geq 0$ with equality only for $n_i = 0$. Thus $\Lambda^{-1}\Big(\mu_i\frac{n_i}{n_i+1} + \lambda + \sum_{j\in\mathcal{M}}(\sigma_j+\mu_j)\Big) \geq 1$.

The strategy for the upper bound is the same. We start by writing Equation 5.1 in the context of $\zeta_{i,t-1}(\mathbf{n}) = \frac{N_i}{\mu_i}, \forall\mathbf{n}\in\mathcal{S}$ and rearranging it following the same strategy as above as far as possible. This gives us Equation 5.13.

$$\zeta_{i,t}(\mathbf{n}) = \Lambda^{-1}\frac{N_i}{\mu_i}\Big(\frac{\mu_i}{N_i} + \mu_i\frac{n_i}{n_i+1} - \mu_i + \lambda + \sum_{j\in\mathcal{M}}\mu_j + \sum_{j\in\mathcal{M}}\sigma_j\Big) \quad (5.13)$$

which implies the final result stated as Equation 5.14.

$$\zeta_{i,t}(\mathbf{n}) = \Lambda^{-1}\frac{N_i}{\mu_i}\Big(\mu_i\frac{n_i+1-N_i}{N_i(n_i+1)} + \lambda + \sum_{j\in\mathcal{M}}\mu_j + \sum_{j\in\mathcal{M}}\sigma_j\Big) \leq \frac{N_i}{\mu_i} \quad (5.14)$$

where the final inequality stems from the fact that $n_i + 1 \leq N_i$, with equality for $n_i = N_i - 1$, which we remind ourselves is the highest possible number of users in the system together with the marked customer.

□

When we replace the automatically routing boundaries with $\hat{z}$-routing boundaries it is still possible to define a subset of $\mathbb{R}^M$ which the algorithm maps back into itself, but in this case we need to consider a slightly larger subset of the real numbers, due to the fact that the hypothetical waiting times conform to the situation when the system can accommodate an additional customer. We state this as Corollary 5.2.1.

Corollary 5.2.1. If we replace the region $\mathcal{A}$ in Lemma 5.2.1 with a region $\mathcal{B} \subset \mathbb{R}^M$ such that $\mathbf{a} = (a_1, ..., a_m) \in \mathcal{B}$ iff

$$\frac{1}{\mu_i} \leq a_i \leq \frac{N_i + 1}{\mu_i}$$

then Lemma 5.2.1 holds when the automatically routing boundaries are replaced with $\hat{\zeta}$-routing boundaries.

Proof. This proof consists of the same direct manipulations as in the proof of Lemma 5.2.1. So Equation 5.15 is Equation 5.2 where we have let $\hat{\zeta}_{i,t-1}(\mathbf{n}) = \frac{1}{\mu_i}, \forall \mathbf{n} \in \mathcal{S}^e$ as well as $\zeta_{i,t-1}(\mathbf{n}) = \frac{1}{\mu_i}, \forall \mathbf{n} \in \mathcal{S}^-$, and rearranged in the same way as in equation 5.12.

$$\hat{\zeta}_{i,t}(\mathbf{n}) = \frac{1}{\mu_i} \Lambda^{-1} \Big(\lambda + \sum_{j \in \mathcal{M}} (\mu_j + \sigma_j) + \mu_i \frac{N_i}{N_i + 1} \Big) > \frac{1}{\mu_i} \tag{5.15}$$

The strict inequality stems from $\frac{N_i}{N_i+1} > 0$.

Finally we do the same for the edge calculations in the case of $\hat{\zeta}$-routing boundaries. We simply state the final step as Equation 5.16

$$\hat{\zeta}_{i,t}(\mathbf{n}) = \Lambda^{-1} \frac{N_i + 1}{\mu_i} \Big(\mu_i \frac{N_i + 1 - (N_i + 1)}{N_i + 1} + \lambda + \sum_{j \in \mathcal{M}} (\sigma_j + \mu_j) \Big) = \frac{N_i + 1}{\mu_i}. \tag{5.16}$$

So with $\hat{\zeta}$-routing boundaries we have the special, but irrelevant, situation that the boundary of the subset of the space of real numbers in which the fixed points are found maps back into itself. □

Note that Lemma 5.2.1 and Corollary 5.2.1 state that the modified policy iteration algorithm is guaranteed to map a subset of $\mathbb{R}^M$ into itself with any decision rule update function $R_j(\mathbf{n}, \boldsymbol{\zeta}_{t-1}(\mathbf{n}))$ such that $R_j(\mathbf{n}, \boldsymbol{\zeta}_{t-1}(\mathbf{n})) \in [0, 1]$.To satisfy the criteria of Theorem 5.2.2 the function $\mathbf{g}$ must also be continuous. With $(0, 1)$-updates, so $R_j(\mathbf{n}, \boldsymbol{\zeta}_{t-1}(\mathbf{n}))$ as in Definition 5.2.1, this is trivially the case. The same is true for any other continuous update function.

So we conclude that when $R_j(\mathbf{n}, \boldsymbol{\zeta}_{t-1}(\mathbf{n}))$ is a continuous function $\mathbf{g}$ fulfils Brouwer's fixed point Theorem, Theorem 5.2.2, and can therefore state Theorem 5.2.3

Theorem 5.2.3. Modified policy iteration, as defined in Algorithm 2, with $R_j(\mathbf{n}, \boldsymbol{\zeta}_{t-1}(\mathbf{n}))$ continuous, and either automatic or $\hat{z}$-routing boundaries, is guaranteed to have at least one fixed point.

Proof. Theorem 5.2.2 and Lemma 5.2.1 and Corollary 5.2.1. □

We can therefore conclude that policy iteration with $(0, 1)$-updates is also guaranteed to have at least one fixed point. We state this as Theorem 5.2.4.

Theorem 5.2.4. Policy iteration as defined in Algorithm 1, with $(0, 1)$-updates and either automatic or $\hat{z}$-routing boundaries, is guaranteed to have at least one fixed point.

Proof. Theorem 5.2.1 and Theorem 5.2.3. □

5.3 Remarks

5.3.1 Applicability of result

We briefly consider what Theorem 5.2.4 means in terms of the $\{0, 1, \tilde{p}\}$-update rules. Consider the operation of writing a version of modified policy iteration in the case of $\{0, 1, \tilde{p}\}$-updates. We note that the expression would include a generalisation of the concept of $\{0, 1, \tilde{p}\}$-policy to M queues. This generalisation would necessarily be such that it would consist of terms in which an arbitrarily

small change in pseudo waiting times can lead to the activation or deactivation of an indicator function, and thus discontinuities. This means that the criteria in Theorem 5.2.2 are not fulfilled, and thus the results of Theorem 5.2.3 do not apply.

5.3.2 Relationship to results from Chapter 4

Theorem 5.2.4 is actually not a resolution of Conjecture 4.3.1, although it does partially deal with the assumptions underlying that conjecture. The thing that would remain to fully resolve the conjecture is to show that at least one of the fixed points is attractive, which we have not been able to show, but which we still conjecture to be the case, based on the numerics.

Conversely we note that the above comment on the fact that this result is not applicable to $\{0, 1, \tilde{p}\}$-updates should not be taken as the statement that the reason for the lack of convergence for certain choices of parameters is necessarily a result of the fact that these discontinuities induce a lack of fixed points. It is also possible that the reason for the lack of convergence to a fixed point is related to the fact that the fixed points are not attractive, or that we have not been able to find their basins of attraction in the numerical studies.

We did attempt to use similar methods to show the guaranteed existence of exactly one fixed point, as well as convergence, in the case of $(0, 1)$-updates, in order to resolve Conjecture 4.3.1, but turned out to not be able to. In the interest of saving others from this cumbersome and ultimately fruitless work we give a short description of this project in Section 5.4.

5.4 Attempted extensions

We conclude this chapter by briefly describing two projects aimed at extending the work presented here. Both of the projects failed to reach satisfying conclusions. We undertake a succinct discussion of each of these projects for slightly different reasons, though. In the case of the first of the projects, Section 5.4.1, which was a bid to show identity and attraction of fixed points in policy iteration, we largely discuss it here in order to save others from the cumbersome work. In the case of the second project, which is a more rigorous, and slightly different, implementation of backwards induction than the one we have considered here, we present the work partially for the same reason as above, to save someone else the frustration, but

also to comment on the information we did glean from the project before the ultimately unsatisfying outcome.

5.4.1 Fixed Point Uniqueness by Contraction

The Project

As mentioned in Section 5.3.2, Theorem 5.2.4 is not a full resolution of conjecture 4.3.1. With the groundwork represented by Theorem 5.2.1 laid it was attractive to try to get closer to a full resolution of that conjecture by some other version of direct manipulation of 5.1. A possible extension in the direction of full resolution would be to show that the modified policy iteration represents a contraction. Since we had seen, by Lemma 5.2.1, that modified policy iteration does map a closed subset of $\mathbb{R}^M$ back into itself showing that it is also a contraction would, by the Banach principle described for example in the very beginning of Granas [66], show that it has exactly one fixed point. For this project we again restricted ourselves to $M = 2$, the system from Chapter 4.

In order to avoid some of the complexity that comes with direct manipulation of this expression we decided to initially carry this project out in the context of the process without intrinsic arrivals. We state the explicit expression in question here for clarity.

$$\begin{aligned}\zeta_{i,t}(\mathbf{n}) = \Big(\lambda\big(&\frac{\zeta_{i,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}I_{\{n_i+1<N_i\}}}{\zeta_{i,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}+\zeta_{j,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}}\\ &+\frac{\zeta_{j,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}I_{\{n_j<N_i\}}}{\zeta_{i,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}+\zeta_{j,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}}\big)+\mu_i+\mu_j I_{\{n_j>0\}}\Big)^{-1}\\ \times\Big(1+I_{\{n_i+1<N_i\}}&[\lambda\frac{\zeta_{i,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}}{\zeta_{i,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}+\zeta_{j,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}}]\zeta_{i,t-1}(\mathbf{n}+\mathbf{e}_i)\\ +\sum_{j\in\mathcal{M}\setminus i}&I_{\{n_j<N_j\}}[\lambda\frac{\zeta_{j,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}}{\zeta_{i,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}+\zeta_{j,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}}]\zeta_{i,t-1}(\mathbf{n}+\mathbf{e}_j)\\ &+\mu_i\frac{n_i}{n_i+1}\zeta_{i,t-1}(\mathbf{n}-\mathbf{e}_i)+\sum_{j\in\mathcal{M}\setminus i}\mu_j I_{\{n_j>0\}}\zeta_{i,t-1}(\mathbf{n}-\mathbf{e}_j)\Big)\end{aligned}\tag{5.17}$$

Just as in the formulation in Chapter 5 we have left the boundary conditions implicit in order to reduce notational complexity. This means that in the terms

that are equivalent to the decision rules, those of the form $\frac{\zeta_{i,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}I_{\{n_i+1<N_i\}}}{\zeta_{i,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}+\zeta_{j,t-1}(\mathbf{n}+\mathbf{e}_i)^{-1}}$, we leave it as implicit that the values in the denominator depend on whether the state in question is an edge state, and if so what happens in such a state. As before we chose to carry the work out in the context of $\hat{\zeta}$-routing boundaries.

A contraction is a function f which, in the context of real numbers, has the property that for two points, x and y in the domain of f this relationship holds: $|x-y| > |f(x)-f(y)|$. We note that this is equivalent to $||J(\mathbf{f}_i)|| < 1$, where $J(f)$ refers to the Jacobian of the function f and $||\cdot||$ refers to any norm. We chose to work in the max-norm so that

$$||J(\mathbf{f}_i)|| = \max_{\mathbf{n}\in\mathcal{S}}\{|\frac{\partial f_i(\mathbf{n})}{\partial \zeta_j(\mathbf{n}^*)}|\}.$$

So, if the absolute value of each element of the Jacobian of Equation 5.1 is less than or equal to 1, then Equation 5.1 is a contraction on a closed subset of $\mathbb{R}^2$ and thus modified policy iteration has exactly one fixed point. This in turn would have meant that policy iteration with $(0,1)$-updates also has exactly one fixed point.

Due to the cumbersome nature of these differentials we do not explicitly state any of them here, but instead just make some observations.

First we note that $\frac{\partial f_i(\mathbf{n})}{\partial \zeta_j(\mathbf{n}^*)} = 0$ when $\mathbf{n}^* = \mathbf{n} + k\mathbf{e_l}$ for all $|k| > 1, l, i, j \in \{1,2\}$, as well as the special case of $k = 1, i \neq j$ and $i \neq l$. Similarly $\frac{\partial f_i(\mathbf{n})}{\partial \zeta_i(\mathbf{n}^*)}$ reduces to simple expressions that can be relatively easily manipulated and shown to be less than 1 for $\mathbf{n}^* \in \{\mathbf{n}+\mathbf{e}_j, \mathbf{n}-\mathbf{e}_i, \mathbf{n}-\mathbf{e}_j\}$.

By contrast the expressions for $\frac{\partial f_i(\mathbf{n})}{\partial \zeta_j(\mathbf{n}^*)}$ when $\mathbf{n}^* = \mathbf{n}+\mathbf{e}_i$ are quite complex, for the purpose of legibility, essentially prohibitively so. In particular these expression consist of large collections of products of the form $\zeta_i(\mathbf{n}) \cdot \zeta_j(\mathbf{n}^*) \cdot ... \cdot \zeta_k(\mathbf{n}^{**})$. Discerning whether these differentials are guaranteed to be less than or equal to 1 becomes a question of comparing the relative sizes of these products.

Mainly these comparisons consisted of showing, for each possible type of state, a particular relationship held between some pseudo waiting times. For example showing that $\zeta_i(\mathbf{n}+\mathbf{e}_j) \leq \zeta_i(\mathbf{n}+\mathbf{e}_i)$ for $\mathbf{n}$ s.t. $n_i+1 < N_i$ and $n_j < N_j$. We see that the condition is similar to the statement in Theorem 3.1.5. This was frequently the case, that some property that needed to hold had already been shown to hold in the fixed point case, and in several of these cases we managed to show that the same held for each step of modified policy iteration. These proofs were frequently algebraically involved, but theoretically straightforward.

The obstacle

Eventually it became clear that we would not be able to show that the cumbersome expressions were less than 1 without showing Equation 5.18 to be guaranteed to hold.

$$\zeta_i(\mathbf{n}+\mathbf{e}_i)-\zeta_i(\mathbf{n}) \geq \zeta_j(\mathbf{n}+\mathbf{e}_i)-\zeta_j(\mathbf{n}), i \neq j, \mathbf{n} \in \mathcal{S}. \tag{5.18}$$

This initially seemed like a rather uncomplicated task, surely it is trivially true that adding a user to queue i will increase the expected waiting time at least as much in queue i as in queue j? It soon became clear that not only is this not trivially true, we could also not show it to be true under assumptions of decision policy monotonicity. So then what remained was to identify the criteria we could put on either the system parameters, or the starting point for the iteration to make the inequality hold. Regardless of what we tried, we did in fact not manage to show the inequality to hold under any non-trivial assumptions.

Since showing this inequality to hold was a prerequisite for showing the contraction itself by this method, the inability to find any conditions under which this inequality holds means that we were unable to show the modified policy iteration to be a contraction at all. We note that we haven't ruled out that there could be other issues with this method, even if we showed the required inequality to be guaranteed to hold.

The interpretation

The condition that we were unable to show to hold, given in Equation 5.18, is a version of the statement that the queue that is subjected to a change in occupancy is also the queue in which the change in expected waiting time is the greatest. It still seems to us as if this would be either true under relatively simple assumptions, or can be made to hold under more demanding conditions. In order to show that the conditions holds, or that it can be made to hold we tried to find some condition that could be put on the parameters or the initial set of pseudo waiting time. We had initially intended to restrict the study to the underloaded region, so $\sum_{i\in\{1,2\}} \mu_i > \lambda + \sum_{i\in\{1,2\}} \sigma_i$ was assumed to hold from the beginning.

Among the conditions we tried to put on the parameters in order to induce the property of interest to hold was decision policy monotonicity, which we also had to use as a condition for a few of the other properties. In this case we were very reluctant to use this condition as we are aware that it is in fact not guaranteed

for all choices of parameters. We also mention that even if we had never seen any examples of a non-monotonic fixed point policy in the context of these update rules, we would still not be justified in assuming that all the iterates prior to the fixed point would be monotonic. Since the assumption of decision policy monotonicity did not solve our issue we did not expend much energy trying to justify the assumption of decision policy monotonicity in this context. One viable option to justify this assumption could be to show that under certain conditions the property propagates from step to step in the iteration. We did however endeavour, with limited success, to do exactly that in the context of more explicit backwards induction, so we describe what such an argument might look like, but in a more general context, in Section 5.4.2.

In conclusion we were unable to show that modified policy iteration in two dimensions without intrinsic arrivals is a contraction due to being unable to find any conditions under which we could show that the expected waiting time is affected more in the queue that is subjected to change than in the other queue.

5.4.2 Explicit backwards induction

The project

The final project we present, briefly, here is a version of backwards induction in the context of pure policies. In this context we considered system behaviour in both the infinite and the finite system, analytically as well as numerically. With the exception of the update rule, and the fact of the consideration of the infinite capacity system the full formulation we consider here is very similar to the formulation discussed in the rest of this chapter. We therefore want to mention that this project differs significantly from the previously discussed ones due to an explicit focus on transients on the way to fixed point policies.

The results that we got from the analytical portion of this project are not sufficiently strong to merit detailed treatment here, so we describe the formulation of the process under consideration at a fairly high level. However, we think the nature of the obstacles we faced, as well as one of the numerical outcomes justify the inclusion here.

The formulation

For this project we sought to formulate a version of backwards induction in terms of pure decision policies for general arrivals to a system consisting of two processor sharing queues in parallel. From our study of policy iteration in the context of pure policies in this queuing system, presented in Chapter 4, we were aware that in limited capacity systems certain parameter sets lead to non-monotonic fixed point policies while others don't lead to convergence at all. By formulating a backwards induction on the same system we attempted to understand certain aspects of this outcome. In particular we present two different questions that we sought to answer in this formulation. First of all, fixed points of policy iteration are not guaranteed to represent subgame perfect equilibria, might this be the reason for the non-monotonicities or even the failures to converge? Related to this question, but also to the fact that we have so far only considered iteration in limited capacity systems: can we analytically prove or disprove the existence of non-monotonicities in subgame perfect equilibria in the infinite capacity system? We present the answers to these questions in the Outcome and Obstacle sections below, respectively.

The main quantity of interest in this project is closely related to the pseudo waiting times that we consider earlier in this chapter. In this formulation it is more fruitful to think of the same quantity as a cost consisting of two separate parts, the cost of waiting and the terminal cost. For simplicity we let the cost of waiting be linear in time with unity differential, so that a user who waits t units of time incurs a cost t. The terminal cost, which we denote by $c_{i,T}(\mathbf{n})$, is the amount that a user who is left in queue i when the process ends pays for the remainder of their service in lieu of waiting. The terminal cost is a function of the state, $\mathbf{n}$, of the system when the process ends. As observed in Section 5.2, a terminal cost is in certain cases a useful interpretation of the initial pseudo waiting times that are used to initiate modified policy iteration as well.

Now the process simply consist of, step by step, updating the process so that a general arrival always chooses to join, with probability 1, the queue in which their expected total cost of participating in the process is minimised. We note that the fact that each decision is based on full knowledge of the decisions of future arrivals in any possible future states of the system until the process ends means that any equilibrium found will be subgame perfect.

For simplicity we let the general arrivals prefer queue 1, so that with equal expected costs associated with joining either queue they join queue 1 with probability 1. This is similar to using $\tilde{p} = 1$ in policy iteration with $\{0, 1, \tilde{p}\}$-updates.

As a minor point we note that when we think of the process as policy iteration with a limited number of steps and a terminal cost it is natural to invert the time indices of the iteration as compared to our previous formulations, so that the terminal cost occurs at the final step of the process at time T and that the first policy that is calculated based on the terminal costs is $\mathbf{D}^{T-1}$ rather than $\mathbf{D}^1$.

We state without discussion the expression, with familiar notation, for the expected cost, $z_{i,\tau}(\mathbf{n})$, for a user who joins the system, which is in state $\mathbf{n}$ before they join queue i, as the τth event of the iterative process when $\tau < T$ in Equation 5.19

$$\begin{aligned} z_{i,\tau}(\mathbf{n}) = \frac{1}{\lambda + \mu_1 + \mu_2}\Big(1 + \mu_j z_{i,\tau+1}(\mathbf{n} - \mathbf{e}_j I_{\{n_j>0\}}) \\ + \lambda\big(z_{i,\tau+1}(\mathbf{n}+\mathbf{e}_1)I_{\{z_{1,\tau+1}(\mathbf{n}+\mathbf{e_i}) \le z_{2,\tau+1}(\mathbf{n}+\mathbf{e_i})\}} \\ + z_{i,\tau+1}(\mathbf{n}+\mathbf{e}_2)I_{\{z_{2,\tau+1}(\mathbf{n}+\mathbf{e_i}) < z_{1,\tau+1}(\mathbf{n}+\mathbf{e_i})\}}\big) \\ + \mu_i z_{i,\tau+1}(\mathbf{n}-\mathbf{e}_i)\frac{n_i}{n_i+1}\Big), i \in \{1,2\}, 0 \le \tau < T, \mathbf{n} \in \mathbb{N}_0^2 \quad (5.19) \end{aligned}$$

and also note that the expected cost for a user who's arrival is the final event in the system is simply the terminal cost associated with the state that they find the system in, so that $z_{i,T}(\mathbf{n}) = c_{i,T}(\mathbf{n})$. This is the formula in the context of an infinite capacity system, the adjustment to the limited capacity system with automatically routing boundaries that we considered numerically is trivial.

The analytical work on this process focused to a substantial degree on considering the effect of employing various terminal cost functions. We did however limit the study to what we think of as sensible terminal cost functions. In particular this means any terminal cost function that preserves our intuitions about monotonicities in the system, so that the terminal cost in queue i is non-decreasing in system occupancy as well as occupancy in queue i for example. For the numerical part we chose to let the terminal cost function be such that the cost incurred by a user left in queue i with the system in state $\mathbf{n}$ when the process ends is identical to the expected time that they would have had to spend were they the n_ith user in an M/M/1-FIFO queue. So

$$c_{i,T}(\mathbf{n}) = \frac{n_i}{\mu_i}.$$

The outcome

We start by considering the outcomes of the numeric explorations of this process. We don't discuss the numerical implementation of this process in any great detail except to just mention what distinguishes this formulation from the policy iteration in Chapter 4 and to a lesser extent the modified policy iteration earlier in this chapter. The most noticeable difference is that in this formulation only convergence of the expected cost, rather than policy convergence, signifies process convergence. This is immediately clear if we consider the fact that Equation 5.19 is a function of only system parameters and expected costs. Thus the process has not converged when it has the same decision policy for two iterative steps as long as the expected costs change. We had a similar situation in the case of modified policy iteration, but the continuous nature of $(0, 1)$-updates makes this effect significantly less pronounced. This means that the concept of policy periodicity loses its meaning to be replaced by the concept of expected cost periodicity.

It is also worth reiterating here that while the omission of the solution of a system of equations leads to backwards induction being much less computationally heavy than policy iteration at each step it also means that the convergence is much slower. However, this had a stronger effect than we had expected. In particular this is interesting for the sets of parameters which we knew to lead to periodicities in policy iteration. In each case of these parameter sets we considered with this process we did not identify periodicities; instead we found that there were many sets of parameters for which the iteration failed to converge despite running for several orders of magnitude longer than the average time to convergence for the parameter sets for which the process converged. Regardless, we didn't find a single example that triggered our criterion for periodicity, which we state as Definition 5.4.1, where δ is the criterion that we used to determine convergence.

Definition 5.4.1. We call an instance of backwards induction periodic if the following property holds for τ_1 and τ_2 such that $|\tau_1 - \tau_2| > 1$ but not for $|\tau_1 - \tau_2| = 1$:

$$|z_{i,\tau_1}(\mathbf{n}) - z_{i,\tau_2}(\mathbf{n})| < \delta$$

We make a few comments on this Definition. First we note that in the case of long iterations these checks are themselves quite computationally expensive. This limited the resolution and size of the areas of state space we could explore. We also note that this is a fairly weak definition of periodicity, and if this criterion

ended up triggering we would have to consider more stringent criteria. We would also have to look directly at the particular iterations that triggered the criterion to make sure that we were actually looking at periodicities. Since we have not seen this criterion trigger, despite some effort, we are tempted to rule periodicity out as a dominant source of non-convergence with this process. Otherwise the lack of observed periodicity of this sort might seem as though it could have something to do with the expected costs being on $\mathbb{R}$ as opposed to the policies in $\{0, 1\}$. We bear in mind, though, that this definition was sufficient to identify periodicities in policies on $\mathbb{R}$ in Section 4.2.1 in the case of $[0, 1]$-updates. See for example Figure 4.7 where we have applied a much more stringent definition of periodicity and still find examples of periodic policies on $\mathbb{R}$. In that case we first looked at many examples of the policies that were judged periodic by Definition 5.4.1 to see that they were genuinely periodic in nature rather than, for example, just converging in a bimodal fashion.

The most interesting outcome of the small numerical component of this project is that when we consider the same parts of parameter space with backwards induction as we did with policy iteration we find very similar results. As an example Figure 5.1 on the next page show the outcome of backwards induction over the region depicted in Figure 4.2 on page 105. We note that the figures are strikingly similar, but not identical. We consider the reason for the differences briefly, and get deeper into the similarities and the implications of them, below in the discussion of the interpretation of this project as a whole. We note that the exact outcome of this sort of parameter space exploration is much more sensitive to inputs and programming choices in this case than in policy iteration. In particular the outcome depends to some extent on the choices of δ and the limiting number of iterative steps before the iteration for a particular parameter set is deemed to be non-converging. In the case of Figure 5.1 $\delta = 0.00001$ and the maximum number of steps before we assumed that we were looking at non-converging iteration was 100. These numbers might seem, and indeed are, somewhat arbitrary, as they are partially based on something akin to trial and error. It's worth noting however, that in numerous samples of parameter sets we took from the non-converging regions, we could make none converge with this choice of δ regardless of the number of steps we allowed. We did similar studies for the choice of δ and found that the choices we use here strike a good balance between computational speed and precision. We also note that while we saw some connection between choice of δ and the outcomes of the calculations, we could not discern any particular connection between choice

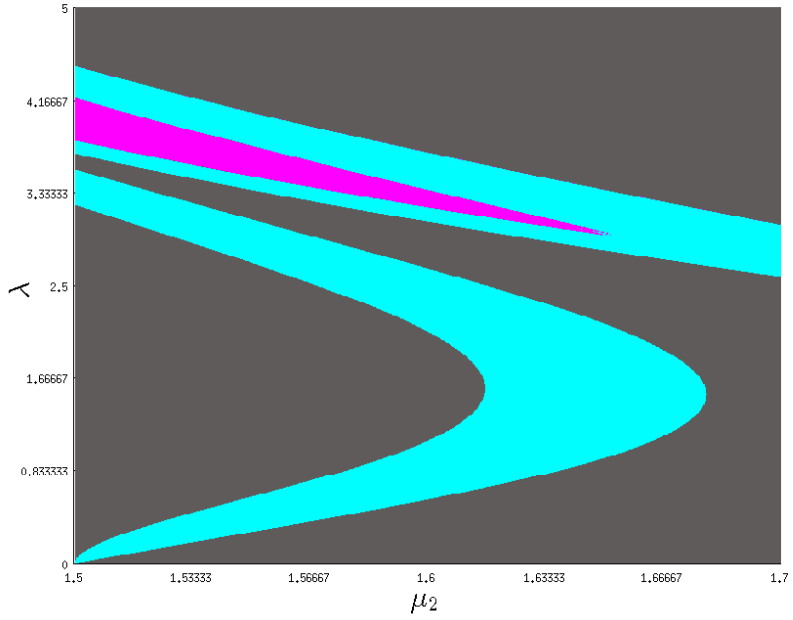


FIGURE 5.1: Outcomes of backwards induction over the region depicted in Figure 4.2. The system parameters are $\mu_1 = 1, \sigma_1 = \sigma_2 = 0, N_1 = N_2 = 3$. Parameter sets that give rise to monotonic fixed-point policies are represented as gray points, sets that give rise to non-monotonic fixed point policies are represented as pink and the sets of parameters for which the iteration failed to converge are represented as cyan.

of δ and the presence or lack of non-monotonic policies. However, due to the fact that the main point of this part of the project was to get an idea of the outcomes in backwards induction for the sets of parameters that lead to non-monotonicities and periodicities in policy iteration we did not focus on this optimisation. This surely accounts for some of the differences between the two figures. Since we do not propose this as a method for identification of optimal policies we did not find it relevant to focus more on the differences in these outcomes.

The obstacle

We knew that in the limited capacity system there are parameter sets that give rise to non-monotonic policies. Could we show this to be true, for sensible choices of terminal cost functions, in the infinite capacity system as well? We explored this question by considering how monotonicity properties of the terminal cost function propagated through updates in the process. In particular we tried, based on knowledge from the numerical work mentioned above as well as numerical explorations of the infinite system, to find sensible terminal cost functions which nevertheless gave rise to non-monotonic policies at some point in the inductive process. We were unable to find any such terminal cost functions.

Since we were unable to find examples of non-monotonicities we instead turned our attention to the possibility of some property, **A**, that could be required of terminal cost functions that would, potentially with certain limitations placed on admissible parameter sets, give rise not only to monotonic decision policies in the previous step, but also to the expected costs in that step having property **A**. Showing monotonicity of any fixed point policies for the relevant parameter sets would in that case then just be a case of showing that property **A** in a general step τ induces property **A** in $\tau - 1$ as well.

We considered many candidates for such a property, and found that while showing that some property induces decision policy monotonicity is very straightforward, showing that some property **A** propagates through the system is in general much more difficult. As an example, a promising candidate for property **A** is the property stated in the context of pseudo waiting times in Equation 5.18. Despite some considerable effort we were unable to prove this property to propagate from step to step in this process. Unfortunately we were likewise unable to find examples of parameter choices for which the property does not propagate.

The interpretation

We found that direct manipulation of the expression describing the cost in each state of the queuing process in this version of the system is rather cumbersome. We also found that it is surprisingly difficult to simplify the formulation in any meaningful way. Among the approaches we tried were several compressions of the expression of the expected costs, none of which were helpful.

However, the project did provide some insight about the system. First of all we want to point out that making a rigorous formulation of backwards induction in this system gave great insight into the structure of the process. In particular, the strategy of thinking about the decision policy at each time step as a reaction to the policies that we know to be optimal in following time steps is instructive in terms of visualising the process of finding equilibria as well as customer behaviour in a physical instantiation of a system of this kind. Simply put it provided a way of thinking clearly about the policy updates in particular, and thus about decision making in general, that is not afforded by the other formulations in this thesis.

Furthermore, what little numerics we did in this formulation increased our understanding of the non-monotonic fixed point policies, as well as non-convergent policy iteration, that were the focus of Chapter 4. For example, the fact that the numerical implementations of this algorithm have the same problems as the policy iteration algorithm rules out certain possible reasons for the non-monotonicities. In particular, it allows us to rule out the possibility that the non-monotonic fixed points are non subgame perfect fixed points which happen to be stable under policy iteration, or have large basins of attraction. The fact that we have evidence of identity between parameter sets that lead to periodic pairs of policies in policy iteration and parameter sets that lead to lack of convergence with backwards induction also motivates us to accept the periodicities as outcomes providing some relevant information, rather than just a pathology of the policy iteration.

While we didn't make this formulation as an alternative to policy iteration as a method for finding fixed point policies in practice, it is interesting to compare the two in this context. The outcome of this comparison can best be expressed as the conclusion that to whatever extent selfish, state dependent, routing with a cost function based on (or identical to) the expected sojourn time for a marked customer is a viable method for routing in a system of this sort, policy iteration is in general a more tractable method for identifying fixed point policies than backwards induction. We also note that we found no examples of parameter sets that had sensible fixed point policies with backwards induction but which didn't with policy iteration. So since the outcomes of backwards induction seem to be at most equally useful as those of policy iteration, the criterion for evaluating the two methods has to be which one is easier to work with. This makes policy iteration the clear choice.

Finally we comment on the outcome of the main focus of this project, properties and existence of fixed point policies for backwards induction in systems of parallel

processor sharing queues with unlimited capacity. To us, the most interesting outcome was our inability to provoke non-monotonicities in this case. This is particularly interesting given how easy we found it to identify parameter sets that give rise to non-monotonic policies in the case of limited capacity.

6

Conclusion

> *When you want to know how things really work, study them when they're coming apart.*
>
> – William Gibson, Zero History

We finally provide some general conclusions that we draw from the work presented here. We group those conclusions into two broad categories. First we discuss, in Section 6.1, the insights the work provides with regards to system behaviour, and the outcome for users, in the context of selfish optimisation. We then, in Section 6.2, go on to consider the implications of the observations we have made about system behaviour in the context of specific assumptions with regards to the behaviour of the general arrivals.

6.1 Selfish optimisation over expected waiting time as a method for job allocation

The main thrust of the work presented in this thesis can best be viewed in the context of articles such as Bonomi [36] and the related work mentioned in Section 1.2. That paper provided the foundation for our work by stating the result that the Join the Shortest Queue policy is suboptimal from a user perspective in all but trivial cases in a system of the kind we study here.

In Chapter 3 we laid the ground work for this study by showing that the system does indeed behave in an intuitive and tractable way under fairly simple assumptions. The theorems in that chapter are all presented with the knowledge, from our own work as well as from the work presented in Chen [45], that some of the assumptions, particularly the assumption of decision policy monotonicity,

is not guaranteed to hold for user optimal policies. Nevertheless those theorems show us that the system under consideration, while exhibiting complex behaviour under certain circumstances, behaves as we would expect a queuing system to, as long as we can make sure that the system is constructed in such a way that the policies that the users employ to make decisions are monotonic. We return to this aspect of the study in Section 6.2.

Next, in Chapter 4, we explored the most straightforward of the, closely related, methods we consider for finding fixed point policies in the system, policy iteration.

In this study we employed a single iterative method but with variations in terms of the exact nature of the updates between iterative steps, and the calculations in certain states, the edge states of the system. We considered three different update rules. The $\{0, 1, \tilde{p}\}$-rule employed in Chen [45], as well as the novel $[0, 1]$, or inclusive, and $(0, 1)$, or exclusive, update rules. In terms of user behaviour in the situation when one queue is full, or boundary conditions, for the system we considered three of those as well. First the natural automatically routing boundaries, where users simply join the system as long as there is room. To this we added the Rejection boundaries, which is the situation when users balk if and only if at least one queue is full, and $\hat{z}$-routing boundaries which is a natural version of the situation when users are given the option of balking if at least one queue is full and choosing to balk with a probability on $(0, 1]$.

We knew from previous work that this method had the capacity to result in complex and counter intuitive outcomes in the form of non-converging iterations and non-monotonic policies. In our study we found limited regions of parameter space where we could not provoke any problematic outcomes, for example for sufficiently low-capacity systems with $(0, 1)$-updates. More interesting than the successes in managing to find small regions of tranquillity by scouring parameter space with a variety of update rules and boundary conditions, was the fact of how hard it was to identify these regions of relative calm. So the main outcome of this chapter, besides just the thorough exploration of these problematic outcomes, is the clear conclusion that policy iteration of this form, regardless of update rules and other considerations, such as boundary conditions, as a method for finding user optimal policies in this system is genuinely problematic. Since we view policy iteration, and related methods, as natural ways that users make decisions, we think these problems are important to take into account from a system design perspective.

To see why we draw this conclusion we start by considering why we find lack

of iteration convergence as well as fixed point policy monotonicity to be so problematic in this case. The obstacle to optimal user performance posed by the lack of policy convergence is somewhat obvious, but we state it explicitly. When the iteration does not converge it means that no optimal policy can be found with this method for certain sets of parameters which are nevertheless admissible as parameters for a queuing system. This simply precludes this method from finding fixed point policies for these sets of parameters. The complex structures of the regions of parameter space in which this occurs makes this problem quite difficult to take into account from a design perspective.

The non-monotonicities are slightly more subtle. We think of it as highly unlikely that any selfish, human, user would work out and employ a non-monotonic policy in any circumstance. Human users are much more likely to rely on heuristics which are almost by definition monotonic. This means that in a system with non-monotonic fixed point policies a selfish user will actually not be able to employ the equilibrium policy. This is part of the reason we have chosen to think of the users as computers for the majority of this; they could at least find and follow the policies, even if they are absurd. The other problem we see with non-monotonic policies is that we have not been able to show why they come into being, and how they are actually optimal. So while numerics tell us that fixed point policies are under certain circumstances non-monotonic, we still view them as an indication of pathology in the system and method under consideration, due to their previously described strangeness.

With this in mind we consider the sequence of theorems and conjectures in this chapter. Theorem 4.3.1 states that policy iteration can fail to converge for certain choices of parameter sets both in the case of $\{0, 1, \tilde{p}\}$- and $[0, 1]$-updates, regardless of boundary condition. By the reasoning above, this suggests that these update rules should be used with caution by users employing policy iteration to find fixed point policies in this system. However, we did not identify, despite rigorous searching, examples of non-convergence in the case of $(0, 1)$ updates. This motivated us to tentatively conclude that policy iteration with this update rule is in fact guaranteed to converge, which we stated as Conjecture 4.3.1.

With regards to the occurrence of non-monotonicity our observations were of a similar nature. In the case of $\{0, 1, \tilde{p}\}$ updates we looked for non-monotonicities, and found them, with all boundary conditions. Thus Theorem 4.3.2 not only states the existence of non-monotonic policies in the case of $\{0, 1, \tilde{p}\}$-updates, but also that this fact is insensitive to choice of boundary conditions among the ones studied

here. Partially due to the lack of affect of boundary conditions both in this case, and with regards to periodicities, we limited the search for non-monotonicities for the two remaining update rules to the natural, automatic routing, boundaries. We observed the problematic behaviour in both cases. So Theorem 4.3.3 states that regardless of which update rule is used, automatically routing boundaries always have the capacity to lead to non-monotonic fixed point policies for certain sets of parameters. Note that this statement is less about boundary conditions than it is about the fact that we have observed non-monotonicities for each update rule when employing perhaps the simplest boundary conditions imaginable, the one that is equivalent to non-balking customers. We return to the lack of observed non-monotonicities in the low-capacity case for $(0, 1)$-updates in Section 6.2.

In summary we came to the conclusion from this part of the study that if the policy iteration method were to be used for finding fixed point policies in a system like this, there are clear advantages to using a version of the method with $(0, 1)$-updates. We also concluded that the boundary conditions did affect the outcomes, but not in any particularly clear way in terms of usefulness of the fixed point policies produced by the method. So the numerics did not give us any clear indication about the relative merits of the different boundary conditions. One sensible reaction to this would be to simply continue on with the simplest version, which in this case would be the automatically routing boundaries. While this is essentially the strategy we followed we concluded that there were still potentially some observations of interest to be made with relation to boundary conditions. So as we built on the findings of this project we did so in the context of $(0, 1)$-updates and mainly automatically routing boundaries. We do, however, in the following chapter come back briefly to the version of boundary conditions that was the most different to the automatically routing boundaries - the $\hat{z}$-routing boundaries.

Then, in Chapter 5 we analytically consider the iterative method implemented numerically in Chapter 4. However, the iterative method we had previously used does not lend itself well to direct manipulation. In particular, the method includes a solution of a system of equations in order to calculate exact expected waiting times at each step. So we defined a different iterative process, the modified policy iteration, and showed that any fixed point in policy iteration is also a fixed point of modified policy iteration and vice versa. We stated this as Theorem 5.2.1.

We then went on to show that the modified policy iteration is in fact guaranteed to have at least one fixed point. We stated this as Theorem 5.2.3, and used the

two Theorems 5.2.1 and 5.2.3 as proof of the fact that policy iteration with $(0, 1)$-updates is guaranteed to have at least one fixed point. We then note that this is in fact not a resolution to Conjecture 4.3.1, since we have not shown the fixed point to be attractive.

Trying to show that policy iteration is guaranteed to have exactly one, unique, fixed point was the goal of the ultimately unsatisfactory work presented in Section 5.4.1. So while the method we attempted in that project, along with other considered but not discussed here, failed to provide evidence for uniqueness and attractivity of the fixed point we nevertheless still believe Conjecture 4.3.1 to hold.

We then finally returned to the study of non-monotonicity, in the final project discussed briefly in Section 5.4.2. This is essentially the purest version of backwards induction of the formulations discussed in Chapter 5. Here we saw that the non-monotonicities that had been shown numerically to occur in fixed points of policy iteration already in Chen [45], still occur when we find the fixed point by explicit backwards induction. This shows that the non-monotonicities are not a pathology of policy iteration, but occur with a rigorous backwards induction formulation, and thus as subgame perfect equilibria, as well, provided the capacity of the system is subject to the same limitations.

When we considered the situation with unlimited capacity, we were unable to provoke non-monotonicities as long as our choice of parameters were reasonable and terminal cost function well behaved. In that context we also made some small headway in showing monotonicity over one process step. While both of these observations might indicate that decision policies in this type of system, parallel processor sharing queues with unlimited capacity, are monotonic we don't think it's indicative enough that we formulate a conjecture about it. The reason we are hesitant to do so is twofold. First of all part of this observation is based on numerics, which can of course only be limited to a trivially small region of state space. The second reason is that we are aware that several of the manipulations carried out in the proof of first step monotonicity were possible due to properties that we know not to propagate to the earlier steps of the process in all but trivial cases.

So we recall the work in Altman & Shimkin [46], that selfish routing based on a cost that is proportional to expected waiting time leads to threshold policies when users make a choice between joining a processor sharing queue and constant expected service time processor. We note that this does in fact not extend to the case when the constant expected waiting time processor is replaced with another

processor sharing queue in the variety of variations on policy and value iterations we have considered. We draw two closely related final conclusions with relation to system design from this. First, when designing systems containing processor sharing queues in parallel in which users make their own choices, all but the simplest systems have the possibility to provoke problematic user behaviour. Second, if the system cannot be designed in a way that is so simple as to be essentially trivial, there is good reason for implementing centralised control of some sort. We reiterate that several recent papers show good performance of size aware task management, for example Hyytiä *et al.* [43] and in the server farm application Gupta *et al.* [37].

We conclude that when designing systems containing processor sharing queues in parallel, it is important to be aware of the potential for these complex user optima. In some cases it may be possible to overcome this difficulty by implementing a central control (whether socially optimal or not). In other cases, the addition of external costs to the system might induce the choice of a more intuitive policy. This remains to be explored.

6.2 System and process properties

The other set of results that we want to summarise here are those related to system properties. Clearly the two sets of results have some overlap, as evidenced by the mention of system properties in the summary in the previous section. However, some of the system properties we showed did not directly have an impact on the validity of selfish routing or the possibility of identifying optimal policies in systems like this. We nevertheless find these results interesting. Thus we think it worthwhile to briefly summarise, separately, the results that relate explicitly to properties of the system. These outcomes can best be viewed in the context of other publications dealing with system properties for systems including processor sharing queues. So we add these results for parallel egalitarian processor sharing queues with exponential service and inter arrival times to the literature which for egalitarian processor sharing is largely covered by the survey article Yashkov & Yashkova [52] and its references. We also reiterate that the relevant literature also includes publications on system properties of closely related systems, such as Altman *et al.* [53], also a survey, but in this case of discriminatory processor sharing.

The majority of results related to system properties can be found in Chapter 3. In this chapter we show that under the assumption of decision policy monotonicity changes of parameters will affect system behaviour in the way that is intuitively natural. We were motivated to consider these properties by the previously shown counter-intuitive structure of fixed point policies in the system. In particular we hypothesised that the decision policy non-monotonicities as outcome from policy iteration starting from monotonic policies indicated that we are lacking some intuition about system behaviour. Thus, in addition to considering what we managed to show we also find it interesting to consider the properties we could not show to hold.

We begin by considering the main results of Chapter 3, though, which is a sequence of theorems related to the properties of a system of M parallel egalitarian processor sharing queues. All of the theorems are statements about the expected waiting time for a marked customer joining the system when it is in some state, and they all share the assumption of decision policy monotonicity.

We start with Theorem 3.1.1 which is the very basic result that increasing the occupancy in the system does not decrease the expected waiting time. This theorem does not cover the case when one user is removed from the queue that the marked customer is in and two users are added to other queues. We note that this is a property that we are familiar with in the case of simple queuing disciplines such as FIFO. We point out again, however, that it's an interesting result in this context due to the possibilities for process behaviour induced by the negative effect that the decisions of later arrivals to the system can have on the service rate for earlier arrivals.

With this in mind, Theorem 3.1.1 is where our inability to show certain properties became interesting. An intuitively attractive conjecture is that the variations in expected waiting time as a result of varying occupation is greatest in the queue in which the change of occupancy takes place. In other words, it would be natural to think that my expected waiting time in the system increases more when another user joins the queue I am in, than when another user joins one of the other queues. No matter how intuitive this statement seems we needed to make several added assumptions in order to be able to show this. First we had to restrict ourselves to the case of two queues, rather than general $M \geq 2$ queues. But even that wasn't sufficient, we had to make at least one out of two other assumptions. We state as Theorem 3.1.5 the result that the asymmetry in expected waiting time increases with occupancy change holds as long as there are no intrinsic arrivals to

the system. Similarly Theorem 3.3.1 states that the same asymmetry holds when the system capacity is unlimited.

We found it interesting that we could not prove this property in greater generality. We note that this property is similar to the obstacle described in 5.4.1. While our inability to show this property obviously doesn't show that it is guaranteed to fail to hold under some circumstance, we note that this property failing to hold despite decision policy monotonicity is a clear way that decision policy non-monotonicity could creep in to the iterative process. In addition to that, the method we use to prove these monotonicity properties, the trivial coupling, is very transparent. So for example in the case of trying to extend the proof of the asymmetry to include the case with limited capacity we note that the coupling breaks down in the edge states because the marked customer can, due to a combination of automatic routing and intrinsic arrivals, end up in a situation when they have a lower probability of service in the process that has to have the higher probability of service for the proof to work. So while this does not show that the property under consideration can fail to hold, it does tell us that interactions between system boundaries and intrinsic arrivals can cause the system to behave in counter intuitive ways.

The remaining results in Chapter 3 relate to monotonicities with respect to system parameters. These theorems are all similar, as are their proofs. Taken together these theorems show that while the system has some complex behaviour with respect to occupancy, expected waiting times in the system are affected in the way that we would expect by changes to system parameters. The remainder of these Theorems are stated in the case of an arbitrary number of queues in parallel.

Theorem 3.1.2 relates to the situation when the rate at which general arrivals arrive at the system is changed. We would expect that the expected waiting time for a marked customer would never decrease due to an increase in the rate at which other users who seek to minimise their own time spent in the system arrive to the system. Indeed this is exactly what the theorem says.

Similarly Theorem 3.1.3 is a statement about how expected waiting times are affected by the rate at which intrinsic arrivals join the system. Intuitively this should be similar to changing the general arrival rate, but have a less pronounced effect as the intrinsic arrivals do not actively seek to game the system, and will therefore for example have a higher probability of being turned away from the system. Regardless, we would expect the expected waiting time to not decrease

as a result of increasing the rate at which intrinsic arrivals join the system. Again this is what the theorem states.

The final theorem related to parameters and expected waiting time is Theorem 3.1.4, which deals with the case of changing service rates. We discuss the interesting aspects of this theorem by considering how and why certain trivial facts about a system of parallel queues under First In First Out service does not carry over to the situation with processor sharing service. In particular we consider the relationship between expected waiting time in queue i given state upon arrival, $z_i(\mathbf{n})$, in these two cases. We note, for clarity, that in the case of FIFO servers $z_i(\mathbf{n})$ is a function only of the occupancy in queue i, n_i, and the service rate in queue i, μ_i.

In the case of processor sharing, as opposed to FIFO, the effect on $z_i(\mathbf{n})$ of changing the service rate is roughly the inverse to changing one of the arrival rates. This is a direct consequence of the fact that the service rate is shared among all currently queuing users. Therefore increasing arrival rates is equivalent in some sense to increasing the expected occupancy which in turn is equivalent to decreased service rate over time. So,we are unsurprised to find that Theorem 3.1.4 states that expected waiting times given system occupancy are non-decreasing in service rate. This is what we would have expected for a service discipline where later arrivals don't negatively impact the service rate of earlier ones as well. We find it interesting in this case to see that the effect is more closely related to the other parameters than it would have been in the case without this relationship between service rate and later arrivals.

We also note here that in the case of FIFO queues we can easily see asymmetries with respect to parameter increases. For example, increasing the service rate in queue i affects the expected waiting time in queue i more than increasing the service rate in queue $j \neq i$ does. In fact we note that in the case of FIFO the expected waiting time in queue i is independent of the service rate in queue $j \neq i$, conditional on the state of the system upon joining. In the case of PS we were unable to show such asymmetry, let alone independence, so all of these theorems simply state that changing parameters affect the expected waiting times in the system in the way that we would expect, but in general rather than with the specificity we are used to from service disciplines in which an arrival negatively impacts the rate of service of users who are already in the system.

In the exploration of the properties of the outcomes of policy iteration in Chapter 4 we find in Section 4.2.3 that while monotonicity is not guaranteed as an outcome of policy iteration, in the cases when a monotonic fixed point policy is found the system exhibits further intuitively reasonable monotonicities. The property in question is the probability that a general arrival will choose to join Queue i when the system has reached stationarity. We call this property the stationary joining probability. Explicitly we would expect that increasing the service rate in queue i would make queue i more attractive, and thus increase the stationary joining probability to queue i. Likewise we would expect an increase in the intrinsic arrival rate in queue i to make queue i less attractive and thus lead to decreased stationary joining probability. Note that this asymmetry is in some sense the type of property that we have found it hard to show to hold in this system, or which we have only been able to show to hold under quite restrictive assumptions.

In Section 4.2.3 we show numerically, and in some cases for the whole underloaded region of parameter space, though obviously not genuinely exhaustively due to the limitations in terms of resolution inherent in numerics, that the stationary joining probability behaves the way we would expect with respect to changes in intrinsic arrival rates and service rates.

As discussed in Section 6.1 policy iteration on this system does not lend itself well to direct manipulation so our ability to make analytical arguments about the outcomes of the process is limited. In Chapter 3 we do, however, make statements about a closely related property, namely the impact of parameter changes on the probability that a general arrival joins queue i. These results, the effect on the joining probability of varying service rates and general arrival rates, can be found in Theorems 3.2.1 and 3.2.2, respectively. The distinction between the property that we mainly explore numerically, and the one that we make analytical statements about is subtle, but clear. We explain this distinction by considering the case of making changes to the service rate. In the numerical case, illustrated well by for example Figure 4.25, we see that when we let the process run to convergence twice with parameter vectors that are such that they differ only in terms of the service rate in queue i, we notice that the stationary joining probability to queue i is no less in the process with the higher rate than it is in the process with the lower service rate. In the analytical case we make a similar statement, but without thinking about it in the context of a process finding a fixed point. Instead we state in Theorem 3.2.1 that given any specified monotonic decision policy, the stationary joining probability to Queue i is non-decreasing in service rate in Queue i.

So the two outcomes are related, if non-identical. The analytical statement can in this case be seen as partial explanation of the numerical outcome, inasmuch as we can view it as saying that even if nothing happened to the decision policies we would expect the stationary joining probabilities to behave in the observed way. Conversely, we can view the numerical work as an extension of the analytical statement. In this case we would think of the analytical result as stating that the joining probability is monotonic in service rates, and the numerical results to add that this property is not counter-acted by policy iteration.

As we mention above, there is a duality between service rates and intrinsic arrival rates in queues where the servers work according to the processor sharing discipline. So we are unsurprised to see that the relationship between stationary joining probabilities and intrinsic arrival rates is the inverse to that with service rates. So, here we see numerically that stationary joining probabilities are non-increasing in intrinsic arrival rates, even if the effect is less pronounced than in the case of service rate. In a similar fashion to the case of service rates we also show analytically that for monotonic decision policies the probability that a general arrival chooses to join Queue i is inversely proportional to the intrinsic arrival rate to i.

Finally we also want to make a comment about the outcome in terms of system properties that we mentioned briefly in Section 5.4.2. We note in the description of that project that the equilibria we find in that case are subgame perfect, as discussed in Section 1.1.4. This makes it all the more interesting to see that even in this case we find that the fixed point policies for certain parameter sets are non-monotonic. We note that this is a numerical result in a method that includes some choices that might be viewed as 'more art than science', such as the choice of δ, but nevertheless we view it as a clear indication that the possibility of non-monotonic pure fixed point policies should be viewed as a property of processor sharing queues in parallel, rather than a property of policy iteration to convergence in such a system. In that way we tentatively put this work into the same context as other work dealing with paradoxical results in congestion games, such as Braess [14] and Afimeimounga *et al.* [67].

So we can summarise the outcome in terms of system properties as a picture of a system that is well behaved in only the simplest of ways. You do indeed expect to get through the system quicker if services occur more frequently, and you do indeed expect to be stuck there longer if there are more people in the system when you arrive. So while we have managed to tease out a number of such properties, which

are useful for heuristics, we have also seen that proving a number of properties that we would intuitively have expected the system to have has eluded us. This in turn tells us that the system does not lend itself particularly well to heuristics, and is thus a system where both users and designers must take great care to avoid unforeseen congestion and other undesirable system states.

In this light we would also like to suggest that future work in this area focus on finding under which circumstances the system does lend itself to heuristics. In particular it would be desirable to understand what distinguishes the regions of parameter space in which we find monotonic pure fixed point policies from the rest. We also note that some of the results here might be extendable, for example to the wider class of queues in which new arrivals negatively impact system performance for users already in the system. This type of generalisation might be desirable, as the problems discussed here might apply to other systems as well.

7

Epilogue

'Oh, figures!' answered Ned. 'You can make figures do whatever you want.'

– Jules Verne, 20,000 Leagues Under The Sea

Towards the end of a beautiful summer's evening a noise like trucks colliding rings out over the Tasman sea. As disturbing as the noise is, in many ways the silence that follows is worse. It soon becomes clear that there will be no re-starting the engines aboard the M/S Bogeserella, fully loaded and en route from New Plymouth to Shanghai. The shipment will be delayed by days, if not more.

Even as the severity of the problem is becoming clear information about the situation is being transmitted to land via satellite internet. The data rapidly trickles out to various databases where it can be retrieved by whoever cares to try to find out, and has the right access privileges.

In small, specialized, commodities trading firms in London, Berlin, Stockholm and Saint Petersburg scripts are waking up in response to very specific shifts in the global shipping patterns. Each of the scripts are different, completely independent from all of the others, but in a sense they all start doing the same thing. Databases are queried, positions ascertained, huge sets of information on various commodities are compiled and compressed. Together with certain well guarded algorithms all of the information is uploaded for processing.

The jobs arrive almost, but not quite, simultaneously to the cloud computing center. By contract with each of the trading firms the time that the data treatment takes is required to meet certain strict tolerances. By considering the size of the incoming jobs, as well as the current state of the computing center, a small piece of software makes the decision of where the jobs go to ensure that these tolerances are met. The decision is unilateral.

As chance would have it, two of the jobs prompted by the engine failure in the Tasman sea end up on the same machine, on the other side of the world. The others are all spread out over distinct machines, even separate racks. It makes no difference, in the end. To everyone's satisfaction the time that the jobs take depend only on the complexity and quality of the code, and the size of the data sets under consideration. Over the following minutes the calculations finish one by one, the results are returned to the appropriate places. Meanwhile new jobs have been started, and the state of the global economy has changed. Slightly.

Outside, the sun seems to struggle to climb above the horizon. It has snowed over night, lending the scene an even deeper sense of otherworldly calm. The little white fox scurries around the corner of the building. It abruptly stops for a moment, cautiously sniffs the air, turns once on the spot and, seemingly content that all is well in the world, continues on its way. The fox knows that, even as a new day is dawning, nightfall is only a few hours away, and it is determined to make the most of the light.

Appendix A

MATLAB code

A.1 Policy iteration algorithm

```
function [resultingD1 resultingD2 allD1s allD2s periodic monotonic...
    ]=policyIteration(mu1,mu2,lambda,sigma1,sigma2,pTilde,delta,N1,...
    N2,D_0,updateRule,boundaryCondition)
%[resultingD1 resultingD2 allD1s allD2s periodic monotonic]=...
    policyIteration(mu1,mu2,lambda,sigma1,sigma2,pTilde,delta,N1,N2...
    ,D_0,updateRule,boundaryCondition)
%
%D_0: is the policy that initiates the policy iteration. In the ...
    case of
%automatically Routing, or rejection boundaries D_0 is expected to...
     be a
%matrix of dimension (N1+1,N2+1) while in the case of z-hat ...
    routing
%boundaries it is expected to be of dimension (N1+1,N2+1,2) where ...
    (:,:,1)
%represents D1, and (:,:,2) represents D2.
%
%monotonic is a single value for autoRouting and rejectionRouting
%boundaries and a vector [monotonicD1 monotonicD2] monotonic=1 if ...
    the
%policy is monotonic

%setting the input D_0 as the seed for the iteration for the ...
    automatically
%routing and rejection routing boundaries
D=D_0(:,:,1);
```

```
%maximum number of iterations before iteration is deemed to not ...
   converge
%and the iteration is broken wuth an error message
maxIterations=1000;
%Initializing the iteration counter
counter=1;

%Defining the acceptable maximum difference between two policies ...
   that is
%still viewed as equal.
policyDiffTolerance=1000*eps;
%The difference to accept peiodicity is set as different (lower) ...
   to guard
%against premature acceptance of periodicity due to non-monotonic
%convergence
periodicityTolerance=policyDiffTolerance;

%Initializing the policy vector for periodicity checks as well as ...
   for
%output. The preallocation is done in such a way that all of the
%preallocated policies can not be reached by actual policy ...
   iteration to
%avoid false positives.
allD1s=2*ones(N1+1,N2+1,maxIterations);
allD2s=2*ones(N1+1,N2+1,maxIterations);

%Intitializing periodic so that it can be returned even when not ...
   examined.
periodic=0;

switch boundaryCondition
    case 'autoRouting'
        while counter<=maxIterations
            allD1s(:,:,counter)=D;

            %Finding the expected waiting times for both queues
            Z_1=Q1ExpWaitTimeAutoRouting(mu1,mu2,lambda,sigma1,...
               sigma2,N1,N2,D);
            Z_2=Q2ExpWaitTimeAutoRouting(mu1,mu2,lambda,sigma1,...
               sigma2,N1,N2,D);

            %Finding the updated policy based on those expected ...
               waiting time
```

```
        newD=policyUpdate(Z_1,Z_2,N1,N2,pTilde,delta,...
            updateRule,boundaryCondition);

        %Control if there is a reason to break the iteration, ...
            either
        %due to convergence or periodicity. In either case the...
             latest D
        %is set as the resulting policy. If convergence has ...
            not been
        %reached the vector containing all previous policies ...
            is
        %checked for previous instances of the current policy....
             If such
        %an instance is found, periodicity is set to 1 and the
        %iteration is broken.
        if all(all(abs(D-newD)<=policyDiffTolerance))
             resultingD1=newD;
             break
        elseif periodicityTest(newD,allD1s,...
            periodicityTolerance,counter)
             periodic=1;
             resultingD1=newD;
             break
        end

        %since newD has passed periodicity and convergence ...
            tests, it is
        %set as the current policy
        D=newD;

        counter=counter+1;
    end

    %Break the function and return if the number of iterative ...
        steps
    %exceeds the maximum number of iterations.
    if counter >= maxIterations
        display('Iteration Timed Out');
        return;
    end

    %Check monotonicity for the resulting policy
```

```
        monotonic=monotonicityTest(resultingD1, boundaryCondition)...
            ;

        %Fill out, then truncate, the policy vector
        allD1s(:,:,counter+1)=resultingD1;
        allD1s=allD1s(:,:,1:counter+1);

        %Create the D2-matrix
        resultingD2=ones(size(resultingD1))-resultingD1;
        resultingD2(end,end)=0;

        %Creating the full set of D2-matrices in the policy ...
            iteration. The
        %D2-matrix is reallocated here, rather than a truncation ...
            of the
        %previously defined one.
        allD2s=ones(size(allD1s));
        for ii=1:size(allD1s,3)
            allD2s(:,:,ii)=allD2s(:,:,ii)-allD1s(:,:,ii);
            allD2s(end,end,ii)=0;
        end

    case 'rejectionRouting'
        while counter<=maxIterations
            allD1s(:,:,counter)=D;

            %Finding the expected waiting times for both queues
            Z_1=Q1ExpWaitTimeRejectionRouting(mu1,mu2,lambda,...
                sigma1,sigma2,N1,N2,D);
            Z_2=Q2ExpWaitTimeRejectionRouting(mu1,mu2,lambda,...
                sigma1,sigma2,N1,N2,D);

            %Finding the updated policy based on those expected ...
                waiting time
            newD=policyUpdate(Z_1,Z_2,N1,N2,pTilde,delta,...
                updateRule,boundaryCondition);

            %Control if there is a reason to break the iteration, ...
                either
            %due to convergence or periodicity. In either case the...
                 latest D
            %is set as the resulting policy. If convergence has ...
                not been
```

```
    %reached the vector containing all previous policies ...
       is
    %checked for previous instances of the current policy....
        If such
    %an instance is found, periodicity is set to 1 and the
    %iteration is broken.
    if all(all(abs(D-newD)<=policyDiffTolerance))
        resultingD1=newD;
        break
    elseif periodicityTest(newD,allD1s,...
       periodicityTolerance,counter)
        periodic=1;
        resultingD1=newD;
        break
    end
    %since newD has passed periodicity and convergence ...
       tests, it is
    %set as the current policy
    D=newD;

    counter=counter+1;
end

%Break the function and return if the number of iterative ...
   steps
%exceeds the maximum number of iterations.
if counter >= maxIterations
    display('Iteration Timed Out');
    return;
end

%Check monotonicity for the resulting policy
monotonic=monotonicityTest(resultingD1, boundaryCondition)...
   ;

%Fill out, then truncate, the policy vector
allD1s(:,:,counter+1)=resultingD1;
allD1s=allD1s(:,:,1:counter+1);

%Create the D2-matrix
resultingD2=ones(size(resultingD1))-resultingD1;
resultingD2(:,end)=0;
resultingD2(end,:)=0;
```

```
        %Creating the full set of D2-matrices in the policy ...
           iteration
        allD2s=ones(size(allD1s));
        for ii=1:size(allD1s,3)
            allD2s(:,:,ii)=allD2s(:,:,ii)-allD1s(:,:,ii);
            allD2s(:,end,ii)=0;
            allD2s(end,:,ii)=0;
        end

    case 'zHatRouting'
        %The initial policy is explicitly turned into a D1 and a ...
           D2.
        D1=D_0(:,:,1);
        D2=D_0(:,:,2);
        while counter<=maxIterations
            allD1s(:,:,counter)=D1;
            allD2s(:,:,counter)=D2;

            %Finding the expected waiting times for both queues
            Z_1 = Q1ExpTimeszHatRouting(mu1,mu2,lambda,sigma1,...
               sigma2,N1,N2,D1,D2);
            Z_2 = Q2ExpTimeszHatRouting(mu1,mu2,lambda,sigma1,...
               sigma2,N1,N2,D1,D2);

            %Finding the updated policy based on those expected ...
               waiting time
            [newD1,newD2]=policyUpdatezHatRouting(Z_1,Z_2,N1,N2,...
               pTilde,delta,updateRule);

            %Control if there is a reason to break the iteration, ...
               either
            %due to convergence or periodicity. In either case the...
                latest D
            %is set as the resulting policy. If convergence has ...
               not been
            %reached the vector containing all previous policies ...
               is
            %checked for previous instances of the current policy....
                If such
            %an instance is found, periodicity is set to 1 and the
            %iteration is broken.
```

```
            if all(all(abs(D1-newD1)<=policyDiffTolerance)) && all...
                (all(abs(D2-newD2)<=policyDiffTolerance))
                 resultingD1=newD1;
                 resultingD2=newD2;
                 break
            elseif periodicityTestzHatRouting(newD1, newD2, allD1s...
                , allD2s, periodicityTolerance,counter);
                 resultingD1=newD1;
                 resultingD2=newD2;
                 periodic=1;
                 break
            end

            %since the newD matrices have passed periodicity and
            %convergence tests, they are set as the current policy
            D1=newD1;
            D2=newD2;

            counter = counter+1;
        end

        %Break the function and return if the number of iterative ...
           steps
        %exceeds the maximum number of iterations.
        if counter >= maxIterations
            display('Iteration Timed Out');
            return;
        end

        %Check monotonicity for the resulting policies.
        resultingDs=resultingD1;
        resultingDs(:,:,2)=resultingD2;
        monotonic=monotonicityTest(resultingDs, boundaryCondition)...
           ;

        %Fill out, then truncate, the policy vectors
        allD1s(:,:,counter+1)=resultingD1;
        allD1s=allD1s(:,:,1:counter+1);
        allD2s(:,:,counter+1)=resultingD2;
        allD2s=allD2s(:,:,1:counter+1);

end
end
```

```
function Z_1=Q1ExpWaitTimeAutoRouting(mu1,mu2,lambda,sigma1,sigma2...
    ,N1,N2,D)
%Returns the expected waiting times for queue 1 when the general ...
   arrivals
%employ automatically routing boundaries.

%Preallocating the matrix that will contain the parameters for the...
    system
%of equations
coefficientMatrx=zeros(N1*N2,5);

%Padding the decision policy matrix with  an extra column and row,...
    so
%that the "neighbouring state" can be examined, even in situations...
    when
%there is no such state. Saves the need for an if-statement in the
%following for-loop.
kludgecol=zeros(N1+1,1);
IndicatorD=[D kludgecol];
kludgerow=ones(1,N2+2);
IndicatorD=[IndicatorD; kludgerow];

for n2=0:N2
    for n1=0:N1-1
        %Setting up the Indicator functions, roughly corresponding...
            to
        %indicators in the written version of the function.
        I=[n1<(N1-1) n2<N2 1-(sum([n1 n2]>=[N1-1 N2])>=2) n2>0];
        %Calculating the coefficients for the state in question
        A=-(sigma1*I(1)+sigma2*I(2)+lambda*I(3)+mu1+mu2*I(4));
        B=(lambda*IndicatorD(n1+2,n2+1)+sigma1)*I(1);
        C=(lambda*(1-IndicatorD(n1+2,n2+1))+sigma2)*I(2);
        E=mu1*(n1/(n1+1));
        F=mu2*I(4);
        %Making sure the coefficients end up in the right order ...
           despite preallocation of the matrix
        coefficientMatrx(n2*N1+n1+1,:)=[A B C E F];
    end
end

%mtrxbuild1 puts together the system matrix for the system of ...
   equations
```

```
systemmtrx=MtrxBuild1(N1*(N2+1), N1, N2, coefficientMatrx);
%By the formulation of the equations all the left hand sides are ...
   -1
solutionvctr=-ones(1,length(systemmtrx));
%Solving the system
resultvector=solutionvctr/systemmtrx;

%As the solution results in a vector it is remapped into a matrix ...
   here. As
%this function deals particularly with the situation with ...
   automatically
%routing boundaries it is sufficient to remap the parts of the ...
   vector that
%corresponds to states in which a marked customer could join bothe...
    queues,
%there is no need to be able to compare the expected waiting times...
    for the
%other states.
Z_1=inf(N1, N2);

for i=1:(N1*N2)
    Z_1(i)=resultvector(i);
end

end

function Z_2=Q2ExpWaitTimeAutoRouting(mu1,mu2,lambda,sigma1,sigma2...
   ,N1,N2,D)
%Returns the expected waiting times for queue 2 when the general ...
   arrivals
%employ automatically routing boundaries.

%Preallocating the matrix that will contain the parameters for the...
    system
%of equations
coefficientMtrx=zeros((N1+1)*N2,5);


%Padding the decision policy matrix with  an extra column and row,...
    so
%that the "neighbouring state" can be examined, even in situations...
    when
%there is no such state. Saves the need for an if-statement in the
```

```
%following for-loop.
kludgecol=zeros(N1+1,1);
IndicatorD=[D kludgecol];
kludgerow=ones(1,N2+2);
IndicatorD=[IndicatorD; kludgerow];

for n2=0:N2-1
    for n1=0:N1
        %Setting up the Indicator functions, roughly corresponding...
            to
        %indicators in the written version of the function.
        I=[n1<N1 n2<(N2-1) 1-(sum([n1 n2]>=[N1 N2-1])>=2) n1>0];
        %Calculating the coefficients for the state in question
        A=-(sigma1*I(1)+sigma2*I(2)+lambda*I(3)+mu2+mu1*I(4));
        B=(lambda*IndicatorD(n1+1,n2+2)+sigma1)*I(1);
        C=(lambda*(1-IndicatorD(n1+1,n2+2))+sigma2)*I(2);
        E=mu1*I(4);
        F=mu2*(n2/(n2+1));
        %Making sure the coefficients end up in the right order ...
           despite
        %preallocation of the matrix
        coefficientMtrx(n2*(N1+1)+n1+1,:)=[A B C E F];
    end
end

%Coef2=coefficientmatrx %For debugging

%mtrxbuild2 puts together the system matrix for the system of ...
   equations
systemmtrx=MtrxBuild2((N1+1)*N2, N1, N2, coefficientMtrx);
%By the formulation of the equations all the left hand sides are ...
   -1
solutionvctr=-ones(1,length(systemmtrx));
%Solving the system
resultvector=solutionvctr/systemmtrx;

%As the solution results in a vector it is remapped into a matrix ...
   here.
Z2Temp=inf(N1+1,N2);

for i=1:N2*(N1+1)
    Z2Temp(i)=resultvector(i);
end
```

```
Z_2=Z2Temp(1:N1,1:N2);
end

function Z_1=Q1ExpWaitTimeRejectionRouting(mu1,mu2,lambda,sigma1,...
    sigma2,N1,N2,D)
%Returns the expected waiting times for queue 1 when the general ...
    arrivals
%employ rejection boundaries.

%Setting up a matrix with the coefficients for the system of ...
    equations.
%There are five coefficients and as many equations (and variables)...
     as the
%number of accessible states in the system.
coefficientMtrx=zeros(N1*N2,5);

%Padding the decision policy matrix with  an extra column and row,...
     so
%that the "neighbouring state" can be examined, even in situations...
     when
%there is no such state. Saves the need for an if-statement in the
%following for-loop.
kludgecol=zeros(N1+1,1);
IndicatorD=[D kludgecol];
kludgerow=ones(1,N2+2);
IndicatorD=[IndicatorD; kludgerow];

for n2=0:N2
    for n1=0:N1-1
        %Setting up the Indicator functions. The third entry in ...
            this vector
        %reflects the fact that no general arrival can enter the ...
            system
        %when either queue is full, either previously or after the...
             marked
        %customer has entered.
        I=[n1<(N1-1) n2<N2 1-(sum([n1 n2]>=[N1-1 N2])>=1) n2>0];
        %Calculating the coefficients for the state in question
        A=-(sigma1*I(1)+sigma2*I(2)+lambda*I(3)+mu1+mu2*I(4));
        B=(lambda*IndicatorD(n1+2,n2+1)*I(2)+sigma1)*I(1);
        C=(lambda*(1-IndicatorD(n1+2,n2+1))*I(1)+sigma2)*I(2);
        E=mu1*(n1/(n1+1));
```

```
        F=mu2*I(4);
        %Making sure the coefficients end up in the right order ...
           despite
        %preallocation of the matrix
        coefficientMtrx(n2*N1+n1+1,:)=[A B C E F];
    end
end

%coefficientmatrx

%mtrxbuild1 puts together the system matrix for the system of ...
   equations
systemMtrx=MtrxBuild1(N1*(N2+1), N1, N2, coefficientMtrx);
%By the formulation of the equations all the left hand sides are ...
   -1
solutionVctr=-ones(1,length(systemMtrx));
%Solving the system
resultVctor=solutionVctr/systemMtrx;

%As the solution results in a vector it is remapped into a matrix ...
   here.
Z_1=inf(N1, N2);

for i=1:(N1*N2)
    Z_1(i)=resultVctor(i);
end

end

function Z_2=Q2ExpWaitTimeRejectionRouting(mu1,mu2,lambda,sigma1,...
   sigma2,N1,N2,D)
%Returns the expected waiting times for queue 2 when the general ...
   arrivals
%employ rejection boundaries.

%Setting up a matrix with the coefficients for the system of ...
   equations.
%There are five coefficients and as many equations (and variables)...
    as the
%number of accessible states in the system.
coefficientMtrx=zeros((N1+1)*N2,5);
```

```
%Padding the decision policy matrix with  an extra column and row,...
    so
%that the "neighbouring state" can be examined, even in situations...
    when
%there is no such state. Saves the need for an if-statement in the
%following for-loop.
kludgecol=zeros(N1+1,1);
IndicatorD=[D kludgecol];
kludgerow=ones(1,N2+2);
IndicatorD=[IndicatorD; kludgerow];

for n2=0:N2-1
    for n1=0:N1
        %Setting up the Indicator functions. The third entry in ...
           this vector
        %reflects the fact that no one can enter the queue when ...
           either
        %queue is full, either previously or after the marked ...
           customer has
        %entered.
        I=[n1<N1 n2<(N2-1) 1-(sum([n1 n2]>=[N1 N2-1])>=1) n1>0];
        %Calculating the coefficients for the state in question
        A=-(sigma1*I(1)+sigma2*I(2)+lambda*I(3)+mu2+mu1*I(4));
        B=(lambda*IndicatorD(n1+1,n2+2)*I(2)+sigma1)*I(1);
        C=(lambda*(1-IndicatorD(n1+1,n2+2))*I(1)+sigma2)*I(2);
        E=mu1*I(4);
        F=mu2*(n2/(n2+1));
        %Making sure the coefficients end up in the right order ...
           despite
        %preallocation of the matrix
        coefficientMtrx(n2*(N1+1)+n1+1,:)=[A B C E F];
    end
end

%mtrxbuild2 puts together the system matrix for the system of ...
   equations
systemMtrx=MtrxBuild2((N1+1)*N2, N1, N2, coefficientMtrx);
%By the formulation of the equations all the left hand sides are ...
   -1
solutionVctr=-ones(1,length(systemMtrx));
%Solving the system
resultVctor=solutionVctr/systemMtrx;
```

```
%As the solution results in a vector it is remapped into a matrix ...
   here.
Z2Temp=inf(N1+1,N2);

for i=1:N2*(N1+1)
    Z2Temp(i)=resultVctor(i);
end

Z_2=Z2Temp(1:N1,1:N2);
end

function Z_1 = Q1ExpTimeszHatRouting(mu1,mu2,lambda,sigma1,sigma2,...
   N1,N2,D1,D2)
%Calculating the matrix of waiting times for queue 1.
%Updated for z-dependent boundary routing.This includes an update ...
   of the
%coefficient matrix calculation so as to use the D1+D2-matrix to ...
   find the
%lambda rate, and the writing of
%Q1outerExpTimes, and finally the additions of the outerExpTimes ...
   to the
%Z1-matrix. NB: this is the implementation that first solves the ...
   normal
%system and then treats the waiting time from the normal queues as
%constants while finding the outer expected times.

%Preallocating a matrix with the coefficients for the system of
%equations. There are five coefficients and as many equations
%(and variables) as the number of accessible states in the system.
coefficientmatrx=zeros(N1*(N2+1),5);

%The loops below serve to calculate all the coefficients needed ...
   for the
%system of equations. The output is a matrix that contain the ...
   coefficients
%for one equation per row.
for n2=0:N2
    for n1=0:N1-1
        %Setting up the throttling factors for this particular ...
           state
        I=[n1<(N1-1) n2<N2 (D1(n1+2,n2+1)+D2(n1+2,n2+1)) n2>0];
        %Calculating the coefficients for the state in question
        A=-(sigma1*I(1)+sigma2*I(2)+lambda*I(3)+mu1+mu2*I(4));
```

```
        B=(lambda*D1(n1+2,n2+1)+sigma1)*I(1);
        C=(lambda*D2(n1+2,n2+1)+sigma2)*I(2);
        E=mu1*(n1/(n1+1));
        F=mu2*I(4);
        coefficientmatrx(n2*N1+n1+1,:)=[A B C E F]; %Making sure ...
           the coefficients end up in the right order despite ...
           preallocation of the matrix
    end
end

%mtrxbuild2 puts together the system matrix for the system of ...
   equations
systemmtrx=MtrxBuild1(N1*(N2+1), N1, N2, coefficientmatrx);
%By the formulation of the equations all the left hand sides are ...
   -1
solutionvctr=-ones(1,length(systemmtrx));
%Solving the system
resultvector=solutionvctr/systemmtrx;

%As the solution results in a vector it is remapped into a matrix ...
   here.
Z1=inf(N1, N2+1);

for i=1:(N1)*(N2+1)
    Z1(i)=resultvector(i);
end

Z_1=[Z1; Q1OuterExpTimes(mu1,mu2,sigma2,lambda,N1,N2,D1,D2,Z1)];
end

function outerZ1=Q1OuterExpTimes(mu1,mu2,sigma2,lambda,N1,N2,D1,D2...
   ,Z1)
%Calculating the waiting times that would be experienced by a user...
    joining
%Q1 in a state that is actually full. That is: Assuming that a ...
   user could
%join the system in an added position, this is the waiting time ...
   that they
%would experience.

%Preallocating the system matrix; one row/column per spot in queue...
    2
systemmtrx=zeros(N2+1);
```

```
%Preallocating the solution vector. The solution vector also ...
   contains
%one solution per spot in Queue 2
solutionvector=zeros(1,N2+1); %

for n2=0:N2
    %Representing the throttling of rates due to availability in ...
       the
    %queue. I(3) represents the multiplicator of Lambda as a ...
       result of
    %possible balking when either queue is full.
    I=[n2>0 n2<N2 (D1(N1+1,n2+1)+D2(N1+1,n2+1))];

    %Diagonal elements consisting of the  rates at which the ...
       current state
    %is left.
    systemmtrx(n2+1,n2+1)=-(mu2*I(1)+mu1+I(3)*lambda+sigma2*I(2));

    if I(2)
        %Unless Queue 2 is full too a general arrival can take the...
            system
        %to a state with one more customer in Queue2. As can ...
           Lambda,
        %subject to balking.
        systemmtrx(n2+2,n2+1)=(lambda*D2(N1+1,n2+1)+sigma2);
    end

    %Multiplier distinguishing between a service in Queue 1 ...
       changing the
    %state of the system and a service in queue 1 going to the ...
       marked
    %customer.
    C=(N1/(N1+1))*mu1;

    if I(1)
        %Unless Queue 2 is empty a service there takes the system ...
           to the
        %state in which there is one customer less in Queue 2.
        systemmtrx(n2,n2+1)=mu2;
    end
```

```
    %As the expected waiting times in the regular states of the ...
       system are
    %treated as constants the expected waiting time after a ...
       service (other
    %than the marked customer) in queue 2 ends up at the solution ...
       side.
    solutionvector(n2+1)=-(1+C*Z1(N1,n2+1));
end

%The system is solved here
outerZ1=solutionvector/systemmtrx;
end

function Z_2 = Q2ExpTimeszHatRouting(mu1,mu2,lambda,sigma1,sigma2,...
   N1,N2,D1,D2)
%Updated for z-dependent boundary routing.This includes an update ...
   of the
%coefficient matrix calculation so as to use the D1+D2-matrix to ...
   find the
%lambda rate and the writing of
%Q2outerExpTimes, and finally the additions of the outerExpTimes ...
   to the
%Z2-matrix. NB: this is the implementation that first solves the ...
   normal
%system and then treats the waiting time from the normal queues as
%constants while finding the outer expected times.

%Setting up a matrix with the coefficients for the system of ...
   equations.
%There are five coefficients and as many equations (and variables)...
    as the
%number of accessible states in the system.
coefficientmatrx=zeros((N1+1)*N2,5);

%The loops below serve to calculate all the coefficients needed ...
   for the
%system of equations. The output is a matrix that contain the ...
   coefficients
%for one equation per row.
for n2=0:N2-1
    for n1=0:N1
        %Setting up the throttling factors for this particular ...
           state
```

```
        I=[n1<(N1) n2<N2-1 (D1(n1+1,n2+2)+D2(n1+1,n2+2)) n1>0 n1<...
           N1 n2<(N2-1) n1>0];
        %Calculating the coefficients for the state in question
        A=-(sigma1*I(1)+sigma2*I(2)+lambda*I(3)+mu2+mu1*I(4));
        B=(lambda*D1(n1+1,n2+2)+sigma1)*I(5);
        C=(lambda*D2(n1+1,n2+2)+sigma2)*I(6);
        E=mu1*I(7);
        F=mu2*(n2/(n2+1));
        %Making sure the coefficients end up in the right order ...
           despite
        %preallocation of the matrix
        coefficientmatrx(n2*(N1+1)+n1+1,:)=[A B C E F];
    end
end

%mtrxbuild2 puts together the system matrix for the system of ...
   equations
systemmtrx=MtrxBuild2((N1+1)*N2, N1, N2, coefficientmatrx);
%By the formulation of the equations all the left hand sides are ...
   -1
solutionvctr=-ones(1,length(systemmtrx));
%Solving the system
resultvector=solutionvctr/systemmtrx;

%As the solution results in a vector it is remapped into a matrix ...
   here.
Z_2=realmax*ones(N1+1,N2+1);

for i=1:N2*(N1+1)
    Z_2(i)=resultvector(i);
end

Z_2(:,end)=Q2OuterExpTimes(mu1,mu2,sigma1,lambda,N1,N2,D1,D2,Z_2);

end

function outerZ2=Q2OuterExpTimes(mu1,mu2,sigma1,lambda,N1,N2,D1,D2...
   ,Z2)
%Calculating the waiting times that would be experienced by a user...
    joining
%Q2 in a state that is actually full. That is: Assuming that a ...
   user could
```

```
%join the system in an added position, this is the waiting time ...
   that they
%would experience.

%Preallocating the system matrix. One row/column per spot in queue...
    1.
systemmtrx=zeros(N1+1);

%Preallocating the solution vector. The solutionvector contains ...
   one
%solution per spot in Queue 1
solutionvector=zeros(1,N1+1);

for n1=0:N1
    %Representing the throttling of rates due to availability in ...
       the
    %queue. I(3) represents the multiplicator of Lambda as a ...
       result of
    %possible balking when either queue is full.
    I=[n1>0 n1<N1 (D1(n1+1,N2+1)+D2(n1+1,N2+1))];

    %Diagonal elements consisting of the  rates at which the ...
       current state
    %is left.
    systemmtrx(n1+1,n1+1)=-(mu1*I(1)+mu2+I(3)*lambda+sigma1*I(2));

    %Unless Queue 1 is full too a general arrivals can take the ...
       system
    %to a state with one more customer in Queue1. As can Lambda, ...
       subject
    %to balking.
    if I(2)
        systemmtrx(n1+2,n1+1)=(lambda*D1(n1+1,N2+1)+sigma1);
    end

    %Multiplier distinguishing between a service in Queue 2 ...
       changing the
    %state of the system and a service in queue 2 going to the ...
       marked
    %customer.
    C=(N2/(N2+1))*mu2;

    if I(1)
```

```
        %Unless Queue 1 is empty a service there takes the system ...
           to the
        %state in which there is one customer less in Queue 1.
        systemmtrx(n1,n1+1)=mu1;
    end

    %As the expected waiting times in the regular states of the ...
       system are
    %treated as constants the expected waiting time after a ...
       service (other
    %than the marked customer) in queue 2 ends up at the solution ...
       side.
    solutionvector(n1+1)=-(1+C*Z2(n1+1,N2));
end

%The system is solved.
outerZ2=solutionvector/systemmtrx;
end

function A=MtrxBuild1(mtrxsize, N1, N2, coefficientmatrx)
%Creating the System matrix for the solution of the system of ...
   equations
%that give the expected waiting times in Queue 1

A=zeros(mtrxsize);

%(Diagonal coeffmtrx 1) The coefficient related to the expected ...
   waiting
%time of the state itself. Diagonal element.
for i=1:mtrxsize
    A(i,i)=coefficientmatrx(i,1);
end

%(1 step down: coeffmtrx 2) The coefficient related to the ...
   expected waiting
%time of the state with one more customer in queue 2. 1 step sub ...
   diagonal
%element.
for i=0:N2
    for j=1:N1-1
        A(N1*i+j+1,N1*i+j)=coefficientmatrx(N1*i+j,2);
    end
end
```

```
%(1 step up coeffmtrx 4). The coefficient related to the expected ...
   waiting
%time of the state with one fewer customer in queue 2. 1 step ...
   super diagonal
%element.
for i=0:N2
    for j=1:N1-1
        A(N1*i+j,N1*i+j+1)=coefficientmatrx(N1*i+j+1,4);
    end
end

%(N steps down coeffmtrx 3) The coefficient related to the ...
   expected waiting
%time of the state with one more customer in queue 1. N+1 step sub...
    diagonal
%element.
for i=1:mtrxsize-N1
    A(i+N1,i)=coefficientmatrx(i,3);
end

%(N steps up coeffmtrx 5). The coefficient related to the expected...
    waiting
%time of the state with one fewer customer in queue 1. N+1 step ...
   super diagonal
%element.
for i=N1+1:mtrxsize
    A(i-N1,i)=coefficientmatrx(i,5);
end
end

function A=MtrxBuild2(mtrxsize, N1, N2, coefficientmatrx)
%Creating the System matrix for the solution of the system of ...
   equations
%that give the expected waiting times in Queue 2

A=zeros(mtrxsize);

%coefficientmatrx contains the coefficients for equation 5.2 where...
    each row
%corresponds to one equation.
```

```
%(Diagonal, coeffmtrx 1) The coefficient related to the expected ...
   waiting
%time of the state itself. Diagonal element.
for i=1:mtrxsize
    A(i,i)=coefficientmatrx(i,1);
end

%(1 down, coeffmtrx 2) The coefficient related to the expected ...
   waiting time
%of the state with one more customer in cue 2. 1 step sub diagonal...
    element.
for i=0:N2-1
    for j=1:N1
        A((N1+1)*i+j+1,(N1+1)*i+j)=coefficientmatrx((N1+1)*i+j,2);
    end
end

%(1 up, coeffmtrx 4). The coefficient related to the expected ...
   waiting time
%of the state with one fewer customer in cue 2. 1 step super ...
   diagonal
%element.
for i=0:N2-1
    for j=1:N1
        A((N1+1)*i+j,(N1+1)*i+j+1)=coefficientmatrx((N1+1)*i+j...
            +1,4);
    end
end

%(N down, coeffmtrx 3) The coefficient related to the expected ...
   waiting time
%of the state with one more customer in cue 1. N+1 step sub ...
   diagonal
%element.
for i=1:mtrxsize-(N1+1)
    A(i+(N1+1),i)=coefficientmatrx(i,3);
end

%(N up, coeffmtrx 5). The coefficient related to the expected ...
   waiting time
%of the state with one fewer customer in cue 1. N+1 step super ...
   diagonal
%element.
```

```
for i=N1+2:mtrxsize
    A(i-(N1+1),i)=coefficientmatrx(i,5);
end
end

function newD=policyUpdate(Z_1,Z_2,N1,N2,pTilde,delta,updateRule,...
    boundaryCondition)
%newD=policyUpdateAutoRouting(Z_1,Z_2,pTilde,delta,updateRule)
%
%Updates decision policies based on expected waiting times. Z_1 ...
    and Z_2 are
%matrices of size N1xN2 (neither including states where the other ...
    queue is
%full, as with Automatically routing boundaries all these states ...
    will have
%decision rules 0/1 regardless.
%
%updateRule has to be 'pTilde', 'inclusive' or 'exclusive'.

%The new decision policy is initialized. This step, rather than
%substitution in the old policy, is carried out to add some ...
    protection
%aginst badly formulated initial policies.
newD=zeros(size(Z_1)+1);
%With automatically routing boundaries the states where queue 2 is...
     full,
%and queue 1 isn't are preset to 1
if strcmp(boundaryCondition,'autoRouting')
    newD(1:end-1,end)=1;
end

%This switch-statement defines the update function that is used to
%calculate the decision rules.
switch updateRule
    case 'pTilde'
        updateFunction=@(z_1,z_2) (z_1<z_2)+pTilde*(z_1==z_2);
    case 'inclusive'
        updateFunction=@(z_1,z_2) (z_1-z_2<=-delta)+(0.5-((z_1-z_2...
            )/(2*delta)))*(abs(z_1-z_2)<delta);
    case 'exclusive'
        updateFunction=@(z_1,z_2) z_2/(z_1+z_2);
end
```

```
%The update rule chosen above is used for each state to find the ...
    new
%decision rule.
for n1=1:N1
    for n2=1:N2
        newD(n1,n2)=updateFunction(Z_1(n1,n2),Z_2(n1,n2));
    end
end

end

function [newD1,newD2]=policyUpdatezHatRouting(Z_1,Z_2,N1,N2,...
    pTilde,delta,updateRule)
%[NewD1,NewD2]=policyUpdatezHatRouting(Z_1,Z_2,N1,N2,pTilde,delta,...
    updateRule)
%
%Returns the decision policies NewD1 and NewD2 based on the ...
    current version
%of the expected waiting times as well as applicable parameters, ...
    depending
%on the current update rule.

%Preallocating the decision policies based on the size of Z1, this...
     choice
%is arbitrary as Z1 and Z2 are the same sizes.
newD1=zeros(size(Z_1));
newD2=zeros(size(Z_1));

%This switch-statement defines the update function that is used to
%calculate the decision rules.
switch updateRule
    case 'pTilde'
        updateFunction1=@(z_1,z_2) (z_1<z_2)+pTilde*(z_1==z_2);
        updateFunction2=@(z_1,z_2) (z_2<z_1)+(1-pTilde)*(z_1==z_2)...
            ;
    case 'inclusive'
        updateFunction1=@(z_1,z_2) (z_1-z_2<=-delta)+(0.5-((z_1-...
            z_2)/(2*delta)))*(abs(z_1-z_2)<delta);
        updateFunction2=@(z_1,z_2) (z_2-z_1<=-delta)+(0.5+((z_1-...
            z_2)/(2*delta)))*(abs(z_2-z_1)<delta);
    case 'exclusive'
        updateFunction1=@(z_1,z_2) z_2/(z_1+z_2);
        updateFunction2=@(z_1,z_2) z_1/(z_1+z_2);
```

```
end

%The decision rule for each state is calculated in this loop
for n1=1:N1+1
    for n2=1:N2+1
        newD1(n1,n2)=updateFunction1(Z_1(n1,n2),Z_2(n1,n2));
        newD2(n1,n2)=updateFunction2(Z_1(n1,n2),Z_2(n1,n2));
    end
end

%The calculation above has stored the balking probabilities in the
%inaccessible states of each matrix. Those probabilities are ...
   artificially
%removed here.
newD1(end,:)=0;
newD2(:,end)=0;

end

function periodic=periodicityTest(newD,allDs,periodicityTolerance,...
   counter)
%periodic=periodicityTest(newD,allDs,perioDicityTolerance)
%
%Tests whether the policy newD has been visited before in the ...
   current
%iteration, to within the periodicityTolerance.


%Assume that the policy is non-periodic until the
%opposite has been proven.
periodic = 0;

%Compare the current policy to each of the previous policies.
for i=counter-1:-1:1
    if all(all(abs(allDs(:,:,i)-newD)<=periodicityTolerance))
        %it  is periodic
        periodic = 1;
        return
    end
end

end
```

```
function periodic=periodicityTestzHatRouting(NewD1, NewD2, allD1s,...
    allD2s, periodicityTolerance,counter)
%periodic=periodicityTestzHatRouting(NewD1, NewD2, allD1s, allD2s,...
    periodicityTolerance)
%
%Tests whether the policies newD1 and NewD2 have been jointly ...
   visited
%before in the current iteration, to within the ...
   periodicityTolerance.

%Assumes that the policy is non-periodic until the
%opposite has been proven.
periodic = 0;

%Compare the current policies to each of the previous policies.
for i=counter-1:-1:1
    %   if all(all(abs(D-Dmtrx(:,:,i))<=delta))
    if all(all(abs(allD1s(:,:,i)-NewD1)<=periodicityTolerance)) &&...
        all(all(abs(allD2s(:,:,i)-NewD2)<=periodicityTolerance))
        %'it  is periodic'
        periodic = 1;
        return
    end
end

end

function monotonic=monotonicityTest(D,boundaryCondition)
%monotonic=monotonicityTest(D)
%
%Checks decisionpolicy D for monotonicity. monotonic is 1 if the
%decisionpolicy is monotonic, 0 otherwise.

if strcmp(boundaryCondition,'zHatRouting')
    %The decision policies are assumed to be non-monotonic until
    %the opposite has been proven.
    monotonic=[0 0];

    %Separate out the two decision policies
    D1=D(:,:,1);
    D2=D(:,:,2);
```

```
    %Check for monotonicity by making sure that the value is ...
       always
    %stationary or increasing in increased x and stationary or ...
       decreasing
    %in negative y, for queue 1.
    if all(all(ge(diff(D1,1,2),0))) && all(all(le(diff(D1,1,1),0))...
       )
        monotonic(1) = 1;
    end

    %And then the other way around for the marginal policy for ...
       queue 2.
    if all(all(le(diff(D2,1,2),0))) && all(all(ge(diff(D2,1,1),0))...
       )
        monotonic(2) = 1;
    end
else
    %Initially assuming the policy is non-monotonic
    monotonic= 0;

    %Picking out the parts of the decision policy that represent a...
        choice in
    %case of rejection boundaries
    if strcmp(boundaryCondition,'rejectionRouting')
        Dtest=D(1:end-1,1:end-1);
    elseif strcmp(boundaryCondition,'autoRouting')
        Dtest=D;
    end

    %Check for monotonicity by making sure that the value is ...
       always stationary
    %or increasing in increased x and stationary or decreasing in ...
       negative
    if all(all(ge(diff(Dtest,1,2),0))) && all(all(le(diff(Dtest...
       ,1,1),0)))
        monotonic = 1;
    end
end
end
```

A.2 Calculating stationary joining probabilities

```
function absProb = assignProb(sigma1, sigma2, lambda, D, mu1, mu2,...
    N1, N2)
%absProb = assignProb(sigma1, sigma2, lambda, D, mu1, mu2, N1, N2)
%
%This function returns the stationary joining probability for a
%set of parameters (sigma1,sigma2,lambda,mu1,mu2,N1,N2) and a ...
   Decision
%Policy D with decision rules on [0,1] and rejection boundaries.

coefficientMtrx=zeros((N1+1)*(N2+1),5);

Q=zeros(N1+1,N2+1);

%added column and row so that it is always possible to  look at ...
   any state
%in the decision policy. The content of the other columns doesn't ...
   matter at
%all as the states in which they are accessed are always zeroed ...
   out by
%indicator functions.
kludgecol=zeros(N1,1);
kludgeD=[D kludgecol];
kludgerow=ones(1,N2+1);
kludgeD=[kludgeD; kludgerow];

%The values for the empty states
n1=0;
n2=0;
currRow=((N2+1)*n1+n2+1);
I = [n1>0 n2>0 1-(sum([n1 n2]>=[N1 N2])>=1) n2<N2 n1<N1]; %Setting...
    up the Indicator function
Q(currRow,currRow) = -(I(1)*mu1 + I(2)*mu2 + I(3)*lambda + I(4)*...
   sigma2 + I(5)*sigma1); %to (n1,n2)
Q(currRow, currRow+1) = I(4)*sigma2+I(3)*(1-kludgeD(n1+1,n2+1))*...
   lambda; %to (n1,n2+1)
Q(currRow, currRow+N2+1) = I(5)*sigma1+I(3)*(kludgeD(n1+1,n2+1)*...
   lambda); %to (n1+1, n2)


%Values for all states in which queue 1 is empty
for n2=1:N2
    currRow=((N2+1)*n1+n2+1);
```

```
    I = [n1>0 n2>0 1-(sum([n1 n2]>=[N1 N2])>=1) n2<N2 n1<N1]; %...
        Setting up the Indicator function
    Q(currRow,currRow-1) = I(2)*mu2; %to (n1,n2-1)
    Q(currRow,currRow) = -(I(1)*mu1 + I(2)*mu2 + I(3)*lambda + I...
        (4)*sigma2 + I(5)*sigma1); %to (n1,n2)
    Q(currRow, currRow+1) = I(4)*sigma2+I(3)*(1-kludgeD(n1+1,n2+1)...
        )*lambda; %to (n1,n2+1)
    Q(currRow, currRow+N2+1) = I(5)*sigma1+I(3)*(kludgeD(n1+1,n2...
        +1)*lambda); %to (n1+1, n2)
end

%values for all states where n1 is over 0 but below N1
for n1=1:N1-1
    for n2=0:N2
    currRow=((N2+1)*n1+n2+1);
    I = [n1>0 n2>0 1-(sum([n1 n2]>=[N1 N2])>=1) n2<N2 n1<N1]; %...
        Setting up the Indicator function
    Q(currRow,currRow-(N2+1)) = I(1)*mu1; %to (n1-1, n2)
    Q(currRow,currRow-1) = I(2)*mu2; %to (n1,n2-1)
    Q(currRow,currRow) = -(I(1)*mu1 + I(2)*mu2 + I(3)*lambda + I...
        (4)*sigma2 + I(5)*sigma1); %to (n1,n2)
    Q(currRow, currRow+1) = I(4)*sigma2+I(3)*(1-kludgeD(n1+1,n2+1)...
        )*lambda; %to (n1,n2+1)
    Q(currRow, currRow+N2+1) = I(5)*sigma1+I(3)*(kludgeD(n1+1,n2...
        +1)*lambda); %to (n1+1, n2)
    end
end

%Values for states where n1=N1
n1=N1;
for n2=0:N2-1
    currRow=((N2+1)*n1+n2+1);
    I = [n1>0 n2>0 1-(sum([n1 n2]>=[N1 N2])>=1) n2<N2 n1<N1]; %...
        Setting up the Indicator function
    Q(currRow,currRow-(N2+1)) = I(1)*mu1; %to (n1-1, n2)
    Q(currRow,currRow-1) = I(2)*mu2; %to (n1,n2-1)
    Q(currRow,currRow) = -(I(1)*mu1 + I(2)*mu2 + I(3)*lambda + I...
        (4)*sigma2 + I(5)*sigma1); %to (n1,n2)
    Q(currRow, currRow+1) = I(4)*sigma2+I(3)*(1-kludgeD(n1+1,n2+1)...
        )*lambda; %to (n1,n2+1)
end

%values for state n1=N1 and N2=n2
```

```
n2=N2;
currRow=((N2+1)*n1+n2+1);
I = [n1>0 n2>0 1-(sum([n1 n2]>=[N1 N2])>=1) n2<N2 n1<N1]; %Setting...
    up the Indicator function
Q(currRow,currRow-(N2+1)) = I(1)*mu1; %to (n1-1, n2)
Q(currRow,currRow-1) = I(2)*mu2; %to (n1,n2-1)
Q(currRow,currRow) = -(I(1)*mu1 + I(2)*mu2 + I(3)*lambda + I(4)*...
   sigma2 + I(5)*sigma1); %to (n1,n2)

%find the nullspace of the matrix transpose
space = abs(null(Q')');

%Make sure the sum of the pi-vector is 1
piRes=space/sum(space);

%Turn the Dmatrix into a vector that matches the pi
Dvec=zeros(N1*N2,1);
DvecD=D';
for i=1:(N1*N2)
    Dvec(i)=DvecD(i);
end

%Pick out the parts of the stationary distribution that matches ...
   the states
%in D:
pivec=[];
for n1=0:N1-1
    for n2=0:N2-1
        pivec(1,end+1)=piRes(n1*(N2+1)+n2+1);
    end
end

%condition the relevant parts of the pi-vector on being in the ...
   relevant
%states
pivec=pivec/sum(pivec);

absProb=pivec*Dvec;
```

A.3 Plotting mechanism

```
function [plotMtrx resultingQueue1Policies resultingQueue2Policies...
    monotonicityMtrx periodicityMtrx]=plotMonotonicityPeriodicity(...
   mu1,mu2,lambda,sigma1,sigma2,pTilde,delta,N1,N2,D_0,updateRule,...
   boundaryCondition,xVariable,xVariableMin,xVariableMax,yVariable...
   ,yVariableMin,yVariableMax,resolution)

%The same number of steps are used in the x and y direction and is...
    chosen
%here.
nSteps=resolution;

switch xVariable
    case 'mu1'
        xLabel='$\mu_1$';
        switch yVariable
            case 'mu1'
                %Stop the function if the two variables chosen are...
                    identical
                display('The x-variable and y-variable must be ...
                   different')
                return
            case 'mu2'
                yLabel='$\mu_2$';
                policyfunction = @(x,y) policyIteration(x,y,lambda...
                   ,sigma1,sigma2,pTilde,delta,N1,N2,D_0,...
                   updateRule,boundaryCondition);
            case 'sigma1'
                yLabel='$\sigma_1$';
                policyfunction = @(x,y) policyIteration(x,mu2,...
                   lambda,y,sigma2,pTilde,delta,N1,N2,D_0,...
                   updateRule,boundaryCondition);
            case 'sigma2'
                yLabel='$\sigma_2$';
                policyfunction = @(x,y) policyIteration(x,mu2,...
                   lambda,sigma1,y,pTilde,delta,N1,N2,D_0,...
                   updateRule,boundaryCondition);
            case 'lambda'
                yLabel='$\lambda$';
                policyfunction = @(x,y) policyIteration(x,mu2,y,...
                   sigma1,sigma2,pTilde,delta,N1,N2,D_0,updateRule...
                   ,boundaryCondition);
        end
    case 'mu2'
```

```
        xLabel='$\mu_2$';
        switch yVariable
            case 'mu1'
                yLabel='$\mu_1$';
                policyfunction = @(x,y) policyIteration(y,x,lambda...
                    ,sigma1,sigma2,pTilde,delta,N1,N2,D_0,...
                    updateRule,boundaryCondition);
            case 'mu2'
                %Stop the function if the two variables chosen are...
                     identical
                display('The x-variable and y-variable must be ...
                    different')
                return
            case 'sigma1'
                yLabel='$\sigma_1$';
                policyfunction = @(x,y) policyIteration(mu1,x,...
                    lambda,y,sigma2,pTilde,delta,N1,N2,D_0,...
                    updateRule,boundaryCondition);
            case 'sigma2'
                yLabel='$\sigma_2$';
                policyfunction = @(x,y) policyIteration(mu1,x,...
                    lambda,sigma1,y,pTilde,delta,N1,N2,D_0,...
                    updateRule,boundaryCondition);
            case 'lambda'
                yLabel='$\lambda$';
                policyfunction = @(x,y) policyIteration(mu1,x,y,...
                    sigma1,sigma2,pTilde,delta,N1,N2,D_0,updateRule...
                    ,boundaryCondition);
        end
    case 'sigma1'
        xLabel='$\sigma_1$';
        switch yVariable
            case 'mu1'
                yLabel='$\mu_1$';
                policyfunction = @(x,y) policyIteration(y,mu2,...
                    lambda,x,sigma2,pTilde,delta,N1,N2,D_0,...
                    updateRule,boundaryCondition);
            case 'mu2'
                yLabel='$\mu_2$';
                policyfunction = @(x,y) policyIteration(mu1,y,...
                    lambda,x,sigma2,pTilde,delta,N1,N2,D_0,...
                    updateRule,boundaryCondition);
            case 'sigma1'
```

```
            %Stop the function if the two variables chosen are...
                identical
            display('The x-variable and y-variable must be ...
                different')
            return
        case 'sigma2'
            yLabel='$\sigma_2$';
            policyfunction = @(x,y) policyIteration(mu1,mu2,...
                lambda,x,y,pTilde,delta,N1,N2,D_0,updateRule,...
                boundaryCondition);
        case 'lambda'
            yLabel='$\lambda$';
            display('under construction')
    end
case 'sigma2'
    xLabel='$\sigma_2$';
    switch yVariable
        case 'mu1'
            yLabel='$\mu_1$';
            policyfunction = @(x,y) policyIteration(y,mu2,...
                lambda,sigma1,x,pTilde,delta,N1,N2,D_0,...
                updateRule,boundaryCondition);
        case 'mu2'
            yLabel='$\mu_2$';
            policyfunction = @(x,y) policyIteration(mu1,y,...
                lambda,sigma1,x,pTilde,delta,N1,N2,D_0,...
                updateRule,boundaryCondition);
        case 'sigma1'
            yLabel='$\sigma_1$';
            policyfunction = @(x,y) policyIteration(mu1,mu2,...
                lambda,y,x,pTilde,delta,N1,N2,D_0,updateRule,...
                boundaryCondition);
        case 'sigma2'
            %Stop the function if the two variables chosen are...
                identical
            display('The x-variable and y-variable must be ...
                different')
            return
        case 'lambda'
            yLabel='$\lambda$';
            policyfunction = @(x,y) policyIteration(mu1,mu2,y,...
                sigma1,x,pTilde,delta,N1,N2,D_0,updateRule,...
                boundaryCondition);
```

```
        end
    case 'lambda'
        xLabel='$\lambda$';
        switch yVariable
            case 'mu1'
                yLabel='$\mu_1$';
                policyfunction = @(x,y) policyIteration(y,mu2,x,...
                    sigma1,sigma2,pTilde,delta,N1,N2,D_0,updateRule...
                    ,boundaryCondition);
            case 'mu2'
                yLabel='$\mu_2$';
                policyfunction = @(x,y) policyIteration(mu1,y,x,...
                    sigma1,sigma2,pTilde,delta,N1,N2,D_0,updateRule...
                    ,boundaryCondition);
            case 'sigma1'
                yLabel='$\sigma_1$';
                policyfunction = @(x,y) policyIteration(mu1,mu2,x,...
                    y,sigma2,pTilde,delta,N1,N2,D_0,updateRule,...
                    boundaryCondition);
            case 'sigma2'
                yLabel='$\sigma_2$';
                policyfunction = @(x,y) policyIteration(mu1,mu2,x,...
                    sigma1,y,pTilde,delta,N1,N2,D_0,updateRule,...
                    boundaryCondition);
            case 'lambda'
                %Stop the function if the two variables chosen are...
                     identical
                display('The x-variable and y-variable must be ...
                    different')
                return
        end
end

%Preallocate all the matrices that will be returned
resultingQueue1Policies=zeros(N1+1,N2+1,nSteps,nSteps);
resultingQueue2Policies=zeros(N1+1,N2+1,nSteps,nSteps);
switch boundaryCondition
    case 'zHatRouting'
        monotonicityMtrx=zeros(nSteps,nSteps,2);

        plotMtrx=zeros(nSteps,nSteps,2);
        jointPlotMtrx=zeros(nSteps);
    otherwise
```

```
        monotonicityMtrx=zeros(nSteps);
        plotMtrx=zeros(nSteps);
end
periodicityMtrx=zeros(nSteps);

%Plotting preliminaries
%Decide how many ticks should be explicitly displayed on each axis
nTicks=6;
%Defining the color map
%myColorMap=[64 64 64; 0 255 255; 255 0 255]./256;
%myColorMap=[90 86 86; 0 255 255; 255 0 255]./256;
myColorMap=[95 91 91; 0 255 255; 255 0 255]./256;

%Calculate the decision policies and fill up the matrices in two ...
   loops. As
%zhatRouting returns a vector as monotonicity rather than a ...
   boolean this
%gets different treatments depending on boundary conditions.
switch boundaryCondition
    case 'zHatRouting'
        for ii=1:nSteps
            for jj=1:nSteps
                [resultingQueue1Policies(:,:,jj,ii) ...
                   resultingQueue2Policies(:,:,jj,ii) allD1s ...
                   allD2s periodicityMtrx(jj,ii) monotonicityMtrx(...
                   jj,ii,:)]=policyfunction(xVariableMin+(...
                   xVariableMax-xVariableMin)*ii/nSteps,...
                   yVariableMin+(yVariableMax-yVariableMin)*jj/...
                   nSteps);
            end
        end

        %Make a matrix for plotting where all points with periodic...
            policies has
        %value 1 all points with non-monotonic policies has value ...
           2 and all
        %monotonic aperiodic policies has value 0.

        %Set the value 2 at all non-monotonic policies
        plotMtrx(monotonicityMtrx==0)=2;
        %Set the value 1 at all periodic policies. By doing this ...
           second we ensure
```

```
        %that periodicity trumps non-monotonicity. This is because...
            we view
        %periodicity as a form of non-convergence, so monotonicity...
            is largely
        %irrelevant.

        %As the periodicity is the same for both policies we  ...
           duplicate
        %it before transferring the information to the matrix to ...
           be
        %plotted.
        periodicityMtrx(:,:,2)=periodicityMtrx;

        %Then the periodicities are incorporated in the plot mtrx.
        plotMtrx(periodicityMtrx==1)=1;

        for ii=1:2;
            %Display the resulting figure
            figure,
            surface(plotMtrx(:,:,ii))
            shading flat
            colormap(myColorMap);
            caxis([0,2])

            %If a colorbar is wanted, uncomment this.
            %colorbar;

%             %Setting the title to differentiate between the ...
   different
%             %monotonicities.
%             currTitle=['Monotonicities from $D_' num2str(ii) '$...
   '];
%             title(currTitle,'Interpreter','LaTex', 'fontsize...
   ',15)

            %Set the x-axis to the actual units of the calculation...
                rather than number
            %of iterative steps.
            set(gca,'XTick',0:nSteps/nTicks:nSteps)
            set(gca,'XTickLabel',xVariableMin:(xVariableMax-...
               xVariableMin)/nTicks:xVariableMax)
```

```
            %Set the y-axis to the actual units of the calculation...
                rather than number
            %of iterative steps.
            set(gca,'YTick',0:nSteps/nTicks:nSteps)
            set(gca,'YTickLabel',yVariableMin:(yVariableMax-...
               yVariableMin)/nTicks:yVariableMax)

            %Set y-label, and adjust its position so that it doesn...
               't blend into the
            %axis
            ylabel(yLabel,'Interpreter','LaTex', 'fontsize',15)
%             ylabh = get(gca,'YLabel');
%             set(ylabh,'Position',get(ylabh,'Position') - [3 0 ...
    0])

            %set x-label
            xlabel(xLabel,'Interpreter','LaTex','fontsize',15)
        end
    otherwise
        for ii=1:nSteps
            for jj=1:nSteps
                [resultingQueue1Policies(:,:,jj,ii) ...
                   resultingQueue2Policies(:,:,jj,ii) allD1s ...
                   allD2s periodicityMtrx(jj,ii) monotonicityMtrx(...
                   jj,ii)]=policyfunction(xVariableMin+(...
                   xVariableMax-xVariableMin)*ii/nSteps,...
                   yVariableMin+(yVariableMax-yVariableMin)*jj/...
                   nSteps);
            end
        end

        %Make a matrix for plotting where all points with periodic...
            policies has
        %value 1 all points with non-monotonic policies has value ...
           2 and all
        %monotonic aperiodic policies has value 0.

        %Plotting preliminaries
        %Decide how many ticks should be explicitly displayed on ...
           each axis
        nTicks=6;

        %Initializing the plot matrix
```

```
        plotMtrx=zeros(nSteps);
        %Set the value 2 at all non-monotonic policies
        plotMtrx(monotonicityMtrx==0)=2;
        %Set the value 1 at all periodic policies. By doing this ...
           second we ensure
        %that periodicity trumps non-monotonicity. This is because...
            we view
        %periodicity as a form of non-convergence, so monotonicity...
            is largely
        %irrelevant.
        plotMtrx(periodicityMtrx==1)=1;

        %Display the resulting figure
        figure,
        surface(plotMtrx)
        shading flat
        colormap(myColorMap);
        caxis([0,2])

        %If a colorbar is wanted, uncomment this.
        %colorbar;

        %Set the x-axis to the actual units of the calculation ...
           rather than number
        %of iterative steps.
        set(gca,'XTick',0:nSteps/nTicks:nSteps)
        set(gca,'XTickLabel',xVariableMin:(xVariableMax-...
           xVariableMin)/nTicks:xVariableMax)

        %Set the y-axis to the actual units of the calculation ...
           rather than number
        %of iterative steps.
        set(gca,'YTick',0:nSteps/nTicks:nSteps)
        set(gca,'YTickLabel',yVariableMin:(yVariableMax-...
           yVariableMin)/nTicks:yVariableMax)

        %Set y-label, and adjust its position so that it doesn't ...
           blend into the
        %axis
        ylabel(yLabel,'Interpreter','LaTex', 'fontsize',15)
%         ylabh = get(gca,'YLabel');
%         set(ylabh,'Position',get(ylabh,'Position') - [3 0 0])
```

```
        %set x-label
        xlabel(xLabel,'Interpreter','LaTex','fontsize',15)

end
```

Bibliography

1. Kendall, D. G. Some Problems in the Theory of Queues. *Journal of the Royal Statistical Society. Series B (Methodological)* **13,** 151–185. ISSN: 00359246 (1951).
2. Voorsluys, W., Broberg, J. & Buyya, R. Introduction to cloud computing. *Cloud computing: Principles and paradigms,* 1–41 (2011).
3. Walrand, J. *An introduction to queueing networks* eng. ISBN: 013474487X (Prentice Hall, Englewood Cliffs, N.J., 1988).
4. Bolch, G. *Queueing networks and Markov chains : modeling and performance evaluation with computer science applications* ISBN: 0-471-56525-3 (Wiley-Interscience, Hoboken, N.J, 2006).
5. Walrand, J. Chapter 11 Queueing networks. *Handbooks in Operations Research and Management Science* **2.** Stochastic Models, 519 –603. ISSN: 0927-0507 (1990).
6. Evstigneev, V. P., Holyavka, M. G., Khrapatiy, S. V. & Evstigneev, M. P. Theoretical Description of Metabolism Using Queueing Theory. *Bulletin of Mathematical Biology* **76,** 2238–2248. ISSN: 1522-9602 (2014).
7. Kelly, F. & Yudovina, E. *A Markov model of a limit order book: thresholds, recurrence, and trading strategies* 2015. eprint: arXiv:1504.00579.
8. Kelly-Bootle, S. & Lutek, B. W. in *Probability, Statistics, and Queuing Theory with Computer Science Applications (Second Edition)* (ed Allen, A. O.) Second Edition (Academic Press, San Diego, 1990). ISBN: 978-0-08-057105-8. doi:https://doi.org/10.1016/B978-0-08-057105-8.50016-3. <http://www.sciencedirect.com/science/article/pii/B9780080571058500163>.
9. Robert, P. *Stochastic Networks and Queues* ISBN: 978-3-662-13052-0 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2003).

10. Bell, C. E. & Stidham Jr., S. Individual versus social optimization in the allocation of customers to alternative servers. *Management Science* **29,** 831–839 (1983).

11. Medhi, J. *Stochastic models in queueing theory* ISBN: 978-0-12-487462-6 (Academic Press, Amsterdam Boston, 2003).

12. Altman, E., Boulogne, T., El-Azouzi, R., Jiménez, T. & Wynter, L. A survey on networking games in telecommunications. *Computers & Operations Research* **33.** Game Theory: Numerical Methods and Applications, 286 –311. ISSN: 0305-0548 (2006).

13. Kelly, F. P. Network Routing. *Philosophical Transactions: Physical Sciences and Engineering* **337,** 343–367. ISSN: 09628428 (1991).

14. Braess, D. Über ein Paradoxon aus der Verkehrsplanung. German. *Unternehmensforschung* **12,** 258–268. ISSN: 0042-0573 (1968).

15. Braess, D., Nagurney, A. & Wakolbinger, T. On a Paradox of Traffic Planning. *Transportation Science* **39,** 446–450. ISSN: 00411655, 15265447 (2005).

16. Nash, J. F. Equilibrium Points in n-Person Games. English. *Proceedings of the National Academy of Sciences of the United States of America* **36,** pp. 48–49. ISSN: 00278424 (1950).

17. Hassin, R. & Haviv, M. Nash equilibrium and subgame perfection in observable queues. *Annals of Operations Research* **113,** 15–26 (2002).

18. Grimmett, G. *Probability and random processes* ISBN: 0198572239 (Oxford University Press, Oxford New York, 2001).

19. Trefethen, L. *Numerical linear algebra* ISBN: 978-0-898713-61-9 (Society for Industrial and Applied Mathematics, Philadelphia, 1997).

20. Thorisson, H. *Coupling, Stationarity, and Regeneration (Probability and Its Applications)* ISBN: 1461270553 (Springer, 2013).

21. Puterman, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* ISBN: 0471619779. <https://www.amazon.com/Markov-Decision-Processes-Stochastic-Programming/dp/0471619779%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0471619779> (Wiley-Interscience, 1994).

22. Tijms, H. C. *Stochastic Models: An Algorithmic Approach (Wiley Series in Probability and Statistics - Applied Probability and Statistics Section)* ISBN: 0471943800. <http://www.amazon.com/Stochastic-Models-Algorithmic-Probability-Statistics/dp/0471943800%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0471943800> (Wiley, 1995).

23. De Leve, G., Tijms, H. & Weeda, P. *Generalized Markovian Decision Processes: Applications* (Mathematisch Centrum Amsterdam, 1970).

24. Guo, X. *Continuous-time markov decision processes : theory and applications* ISBN: 9783642025471; 3642025471. <http://www.springer.com/mathematics/applications/book/978-3-642-02546-4> (Springer, 2009).

25. White, D. J. *Markov decision processes* ISBN: 0471936278 (Wiley & Sons, Chichester England, 1993).

26. Feinberg, E & Shwartz, A. *Handbook of Markov decision processes : methods and applications* ISBN: 0792374592 (Kluwer Academic Publishers, Boston, 2002).

27. Littman, M. in *International Encyclopedia of the Social & Behavioral Sciences* (eds in Chief: Neil J. Smelser, E. & Baltes, P. B.) 9240 –9242 (Pergamon, Oxford, 2001). ISBN: 978-0-08-043076-8. doi:http://dx.doi.org/10.1016/B0-08-043076-7/00614-8. <http://www.sciencedirect.com/science/article/pii/B0080430767006148>.

28. Chang, H. S., Lee, H.-G., Fu, M. & Marcus, S. Evolutionary policy iteration for solving Markov decision processes. *Automatic Control, IEEE Transactions on* **50,** 1804–1808. ISSN: 0018-9286 (2005).

29. Zhu, Q. Average optimality for continuous-time Markov decision processes with a policy iteration approach. *Journal of Mathematical Analysis and Applications* **339,** 691–704 (2008).

30. Nisan, N. *Algorithmic game theory* ISBN: 9780521872829 (Cambridge University Press, Cambridge New York, 2007).

31. Kleinrock, L. Time-shared Systems: A Theoretical Treatment. *J. ACM* **14,** 242–261. ISSN: 0004-5411 (1967).

32. Rosberg, Z. *et al.* Insensitive Job Assignment With Throughput and Energy Criteria for Processor-Sharing Server Farms. *IEEE/ACM Transactions on Networking* **22,** 1257–1270. ISSN: 1063-6692 (2014).

33. Mukhopadhyay, A. & Mazumdar, R. R. Analysis of Randomized Join-the-Shortest-Queue (JSQ) Schemes in Large Heterogeneous Processor-Sharing Systems. *IEEE Transactions on Control of Network Systems* **3,** 116–126. ISSN: 2325-5870 (2016).

34. Altman, E., Ayesta, U. & Prabhu, B. J. Load balancing in processor sharing systems. *Telecommunication Systems* **47,** 35–48. ISSN: 1572-9451 (2011).

35. Lee, J. Y., Kim, S., Kim, D. & Sung, D. K. Bandwidth optimization for Internet traffic in generalized processor sharing servers. *IEEE Transactions on Parallel and Distributed Systems* **16,** 324–334. ISSN: 1045-9219 (2005).

36. Bonomi, F. On job assignment for a parallel system of processor sharing queues. *IEEE Transactions on Computers* **39,** 858–869. ISSN: 0018-9340 (1990).

37. Gupta, V., Balter, M. H., Sigman, K. & Whitt, W. Analysis of join-the-shortest-queue routing for web server farms. *Performance Evaluation* **64.** Performance 2007, 1062 –1081. ISSN: 0166-5316 (2007).

38. Gupta, V., Sigman, K., Harchol-Balter, M. & Whitt, W. Insensitivity for PS Server Farms with JSQ Routing. *SIGMETRICS Perform. Eval. Rev.* **35,** 24–26. ISSN: 0163-5999 (2007).

39. Fu, J., Guo, J., Wong, E. W. M. & Zukerman, M. Energy-Efficient Heuristics for Insensitive Job Assignment in Processor-Sharing Server Farms. *IEEE Journal on Selected Areas in Communications* **33,** 2878–2891. ISSN: 0733-8716 (2015).

40. Fu, J., Moran, B., Guo, J., Wong, E. W. M. & Zukerman, M. Asymptotically Optimal Job Assignment for Energy-Efficient Processor-Sharing Server Farms. *IEEE Journal on Selected Areas in Communications* **34,** 4008–4023. ISSN: 0733-8716 (2016).

41. Lefebvre, S., Chiky, R. & Prabhu.K, S. *WACA: Workload and cache aware load balancing policy for web services* in *2012 1st International Conference on Systems and Computer Science (ICSCS)* (2012), 1–6. doi:10.1109/IConSCS.2012.6502459.

42. Yi, S. *et al. Combinational backfilling for parallel job scheduling* in *2010 2nd International Conference on Education Technology and Computer* **2** (2010), V2–112–V2–116. doi:10.1109/ICETC.2010.5529424.

43. Hyytiä, E., Virtamo, J., Aalto, S. & Penttinen, A. M/M/1-PS queue and size-aware task assignment. *Performance Evaluation* **68,** 1136–1148 (2011).

44. Hoekstra, G. J., van der Mei, R. D. & Bhulai, S. Optimal Job Splitting in Parallel Processor Sharing Queues. *Stochastic Models* **28,** 144 –166. ISSN: 15326349 (2012).

45. Chen, Y. *User Equilibria in Stochastic Networks* PhD thesis (University of Auckland, Department of Statistics, 2012).

46. Altman, E. & Shimkin, N. Individual equilibrium and learning in processor sharing systems. *Operations Research* **46,** 776–784 (1998).

47. Nahir, A., Orda, A. & Raz, D. *Workload factoring with the cloud: A game-theoretic perspective* in *2012 Proceedings IEEE INFOCOM* (2012), 2566–2570. doi:10.1109/INFCOM.2012.6195654.

48. Simhon, E. & Starobinski, D. *Game-theoretic analysis of advance reservation services* in *2014 48th Annual Conference on Information Sciences and Systems (CISS)* (2014), 1–6. doi:10.1109/CISS.2014.6814104.

49. Haviv, M. Stable strategies for processor sharing systems. *European Journal of Operational Research* **52,** 103 –106. ISSN: 0377-2217 (1991).

50. Hassin, R. Equilibrium customers' choice between FCFS and random servers. *Queueing Systems* **62,** 243–254 (2009).

51. Borst, S. C. Optimal probabilistic allocation of customer types to servers. *SIGMETRICS Perform. Eval. Rev.* **23,** 116–125. ISSN: 0163-5999 (1995).

52. Yashkov, S. & Yashkova, A. Processor sharing: A survey of the mathematical theory. *Automation and Remote Control* **68,** 1662–1731 (2007).

53. Altman, E., Avrachenkov, K. & Ayesta, U. A survey on discriminatory processor sharing. *Queueing Systems* **53,** 53–63. ISSN: 1572-9443 (2006).

54. Yu, M., Tang, Y. & Wu, W. Individually and socially optimal joining rules for an egalitarian processor-sharing queue under different information scenarios. *Computers & Industrial Engineering* **78,** 26 –32. ISSN: 0360-8352 (2014).

55. Abramov, V. M. *Characterization and asymptotic analysis of the stationary probabilities in Discriminatory Processor Sharing Systems* 2013. eprint: arXiv:1308.3756.

56. Selen, J., Adan, I., Kapodistria, S. & van Leeuwaarden, J. Steady-state analysis of shortest expected delay routing. *Queueing Systems* **84,** 309–354. ISSN: 1572-9443 (2016).

57. Ziedins, I. A paradox in a queueing network with state-dependent routing and loss. *Journal of Applied Mathematics and Decision Sciences* **2007.** <http://www.scopus.com/inward/record.url?eid=2-s2.0-38949191405&partnerID=40&md5=506e43efc93ac9c82bcc15579c19242e> (2007).

58. Naor, P. The Regulation of Queue Size by Levying Tolls. *Econometrica* **37,** 15–24. ISSN: 00129682, 14680262 (1969).

59. Roughgarden, T. & Tardos, E. How bad is selfish routing? *J. ACM* **49,** 236–259. ISSN: 0004-5411 (2002).

60. Parlaktürk, A. & Kumar, S. Self-interested routing in queueing networks. *Management Science* **50,** 949–966 (2004).

61. Haviv, M. & Roughgarden, T. The price of anarchy in an exponential multi-server. *Operations Research Letters* **35,** 421 –426. ISSN: 0167-6377 (2007).

62. Hassin, R. *To Queue or Not to Queue: Equilibrium Behavior in Queueing Systems* ISBN: 9781461503590 (Springer US, Boston, MA, 2003).

63. Berg, J. L. V. D. *Sojourn times in Feedback and Processor Sharing Queues* ISBN: 906196427X (Centrum Voor Wiskunde En Informatica. Amsterdam, the Netherlands, 1993).

64. Kleinrock, L. *Queueing Systems* ISBN: 0471491101 (Wiley, 1974).

65. Demmel, J. *Applied numerical linear algebra* ISBN: 0898713897 (Society for Industrial and Applied Mathematics, Philadelphia, 1997).

66. Granas, A. *Fixed point theory* ISBN: 9780387215938 (Springer, New York London, 2003).

67. Afimeimounga, H., Solomon, W. & Ziedins, I. User equilibria for a parallel queueing system with state dependent routing. *Queueing Systems* **66,** 169–193 (2010).