Control of Multiclass Queueing Systems with Abandonments and Adversarial Customers

Terry James, B.Sc.(Hons.), M.Res

Submitted for the degree of Doctor of Philosophy at Lancaster University.

April 2016

Abstract

This thesis considers the defensive surveillance of multiple public areas which are the open, exposed targets of adversarial attacks. We address the operational problem of identifying a real time decision-making rule for a security team in order to minimise the damage an adversary can inflict within the public areas. We model the surveillance scenario as a multiclass queueing system with customer abandonments, wherein the operational problem translates into developing service policies for a server in order to minimise the expected damage an adversarial customer can inflict on the system.

We consider *three different surveillance scenarios* which may occur in real-world security operations. In each scenario it is only possible to calculate optimal policies in small systems or in special cases, hence we focus on developing heuristic policies which can be computed and demonstrate their effectiveness in numerical experiments. In the *random adversary scenario*, the adversary attacks the system according to a probability distribution known to the server. This problem is a special case of a more general stochastic scheduling problem. We develop new results which complement the existing literature based on priority policies and an effective approximate policy improvement algorithm. We also consider the scenario of a *strategic adversary who chooses where to attack*. We model the interaction of the server and adversary as a two-person zero-sum game. We develop an effective heuristic based on an iterative algorithm which populates a small set of service policies to be randomised over. Finally, we consider the scenario of a *strategic adversary who chooses both where and when to attack* and formulate it as a robust

optimisation problem. In this case, we demonstrate the optimality of the last-come first-served policy in single queue systems. In systems with multiple queues, we develop effective heuristic policies based on the last-come first-served policy which incorporates randomisation both within service policies and across service policies.

Acknowledgements

I would like to take this opportunity to thank those that have helped me along the way and have been part of the journey this thesis has entailed. Firstly, I would like to thank my supervisors. Professor Kevin Glazebrook has been a source of great knowledge, experience, and support and his guidance has always been integral to my progress. Associate Professor Kyle Lin, of the Naval Postgraduate School in Monterey, California has been an absolute pleasure to work with. To add to his great support and guidance, he has always displayed the understanding, patience, and encouragement I have needed at times and for this I cannot express my gratitude enough. He was also a warm host upon my two visits to California, amazing experiences I have had the fortune to have during my research. I am grateful to Professor Adam Letchford for his advice and support, which has always had a positive impact on me. Further to this, I would like to thank Professor Jonathan Tawn, Director and co-founder of the STOR-i Doctoral Training Centre. Together with Professor Glazebrook, it is his vision and undiminished commitment to STOR-i which made everything possible. I hold Professor Tawn in high regards and would like to wish him well in the future. Another person I would like to thank is Dr Chris Kirkbride for the many useful discussions we had, especially when I was starting my research.

The person I would like to thank most of all is Hailey. You have gone through the journey by my side, every step of the way. You have always supported me, listened to me, talked with me, laughed with me, and agonised with me. You have taught me much about myself, challenging and stretching my beliefs and

through this equipped me to tackle the challenges I have faced. I firmly believe that everything would not have been possible without you. Thank you Hailey. After this, my parents. They have always been there when I have needed them, always had faith in me and my ability to achieve throughout my life, and part of what drives me is to make them proud. Finally, I reserve a word for the one person I would dearly have loved to be able to thank, Nicola. She made an indelible impression on me and will never be forgotten.

Declaration

I declare that the work in this thesis has been done by myself and has not been submitted elsewhere for the award of any other degree.

Terry James

Contents

List of Figures

List of Tables

List of Abbreviations

ADP	Approximate Dynamic Programming
API	Approximate Policy Iteration
CORS	Constrained Optimisation using Response Surfaces
DP	Dynamic Programming
DR	Departure Reselection
FCFS	First Come First Served
FP	Fictitious Play
HCP	Heuristic Cutting Plane
HCP$^+$	Enhanced Heuristic Cutting Plane
IFR	Increasing Failure Rate
KCP	Kelley's Cutting Plane
LCFS	Last Come First Served
LCFS-PS	Last Come First Served with Probabilistic Skipping
LP	Linear Programming
MDP	Markov Decision Process
PaS	Pairwise Swapping
rAPI	Recommended approximate policy iteration service policy
RR	Round Robin
SLQ	Serve the Longest Queue
SR	Service Reselection
TPZS	Two-Person Zero-Sum

List of Notation

λ_j	Poisson arrival rate into queue j.
μ_j	Service rate in queue j.
θ_j	Abandonment rate in queue j.
R_j	Reward received for each service completion in queue j.
D_j	Penalty incurred for each abandonment in queue j.
c_j	Holding cost per unit time incurred by each customer in queue j
d_j	Damage inflicted by an adversary if he is able to abandon queue j before service completion.
π	Service policy used by the server.
$R\mu$	The $R\mu$ rule, a priority service policy which ranks all customer classes according to the product of reward R and service rate μ and serves according to this order.
$R\mu\theta$	The $R\mu\theta$ rule, a priority service policy which ranks all customer classes according to the product of reward R, service rate μ, and abandonment rate θ and serves according to this order.
$\mathbf{n}$	State of the system, where $\mathbf{n} = (n_1, ..., n_k)$ and n_j reprsents the number of customers present in queue j.
g^π	Gain or long-run average reward rate under policy π.
$\omega^\pi(\mathbf{n})$	Bias function of state $\mathbf{n}$ under policy π.
α_j^π	Rate of service completions in queue j under policy π in steady state.

β_j^π	Rate of abandonments in queue j under policy π in steady state.
ρ	Workload or traffic intensity, defined as $\sum_{j=1}^{k} \lambda_j/\mu_j$.
$E(\cdot)$	Expectation operator.
$R^\pi(\theta)$	Reward rate under policy π in a system with abandonments.
N_j	Truncation level in queue j such that when the number of customers in the queue reaches this level, new customers are blocked from entering.
h^π	Reward rate loss from the system under policy π.
$p(\mathbf{n}'\|\mathbf{n}, a)$	Transition probability (uniformised) from state $\mathbf{n}$ to state $\mathbf{n}'$ under the action of serving queue a.
Δ	Uniformisation parameter such that the system makes transitions at this uniform rate.
$\text{API}(\pi, n, m, r, t)$	Approximate policy iteration algorithm with its parameters: the initial policy π, n the number of selected states, m the number of simulation replications, r the fraction of selected states in the anchor set, and t the number of iterations of the algorithm.
$\mathbf{p}$	Probability vector $\mathbf{p} = (p_1, ...p_k)$ with which the adversary joins the system, such that queue j is joined with probability p_j and $\sum_j p_j = 1$.
$D^\pi(\mathbf{p})$	Expected damage inflicted by the adversary in steady state when the server uses policy π and the adversary uses probability vector $\mathbf{p}$.
$\mathcal{P}$	Set of service policies the server adopts a mixed strategy over.
$\mathbf{q}$	Mixed strategy of the server of the policies in the set $\mathcal{P}$.
$C_j^\pi(\mathbf{n})$	Expected damage inflicted by the adversary when the adversary attacks queue j in state $\mathbf{n}$ and the server uses nonidling policy π.

C_j^π	Expected damage inflicted by the adversary when the adversary attacks queue j in the over-crowded state and the server uses nonidling policy π.
$\mathbf{r}$	Reselection vector $\mathbf{r} = (r_1, ..., r_k)$ of the DR and SR policies such that $0 \leq r_j \leq 1$ and $\sum_j = 1$.
$\mathbf{s}$	Skipping vector $\mathbf{s} = (s_1, ..., s_k)$ of the LCFS-PS policy such that $0 \leq s_j \leq 1$ and one of the elements is equal to one.
$V_j(\mathbf{r})$	Expected damage inflicted by the adversary when the adversary attacks queue j in the over-crowded state and the server uses the DR policy with reselection vector $\mathbf{r}$.
$W_j(\mathbf{s})$	Expected damage inflicted by the adversary when the adversary attacks queue j in the over-crowded state and the server uses the LCFS-PS policy with skipping vector $\mathbf{s}$.

Chapter 1

Introduction

1.1 Motivation

The threats faced on a daily basis from adversarial agents are a prominent feature of the modern world. Criminals wish to evade the authorities, thieves seek to steal valuable commodities, immigrants look to illegally cross borders, and terrorists hope to cause mass damage and disruption to daily life. When these threats become reality, the impact can be devastating. This is especially evidenced in the notorious terrorist attacks of 9/11 in 2001, as well as the terrorist bombings in Bali in 2002 and 2005, the 7/7 2005 London bombings, and recently the 2013 Boston marathon bombing. Given the potentially significant impact of adversarial threats, together with the fact that threats grow and evolve through time as adversaries become more capable, there is a real need for defensive efforts to mitigate these threats both now and in the future.

The abilities of authorities to engage in defensive efforts have been greatly enhanced by technological innovation. This trend is expected to continue into the future. Surveillance cameras, either static or as part of an unmanned aerial vehicle, enable the screening of public areas consisting of many people. Pictures or video-feeds can be relayed to a control centre in real time whereby subjects can be matched against a database by means of their biometric signatures to determine their identity or possible intentions. Use of such surveillance resources

for defensive purposes can be referred to as defensive surveillance. Central to the idea of Homeland Security, a term coined in the United States in response to terrorist threats, is that preventing threats from maturing or minimising their impact is certainly preferable to responding to events which do occur. For example, defensive efforts which prevent a terrorist from detonating a bomb in a public area are better than responding to the mass damage caused and subsequent retaliation. Defensive surveillance is part of a wider strategy to realise this goal.

The strong capabilities of surveillance resources are paired with the drawback that they are *finite in nature*. In a public area, such as a train station, consisting of a crowd of people, it is only possible to screen a finite number of individuals, often one, at any given time. Screening is not instantaneous as it requires some processing time. The only way every member of the crowd could be screened is if they all stayed in the public area for the necessary length of time needed to screen every person. However, the nature of public areas reveals that this is not realistic. In reality, people arrive into a public area and stay there for some finite, *random period of time before leaving*. Public areas evolve somewhat randomly, wherein the people within them have a finite lifetime. It is clear then that the choice to screen one individual ahead of many others presents an opportunity cost of not being able to screen the individuals who leave the area during the screening of the chosen individual. However, it is not known who will leave or how many. Typically, screening every individual is an impossible task. Fundamentally then, at a given point in time, which individual should be screened? This reflects one of the major challenges of defensive surveillance, *management of a scarce resource* in the context of screening subjects who have a finite lifetime.

A pertinent example of an adversarial threat is that of a terrorist attack. These attacks often take place in crowded, everyday locations or public areas. For example, tourist spots, transportation hubs, and organised public events. This indicates the carefully planned nature of the adversary. The first trait of this planning is that terrorists wish to strike in places which allow for enhanced impact. These

places are open, exposed, and not too difficult for the adversary to penetrate. In this sense, public areas are somewhat soft targets in comparison to a well defended and fortified military base, for example. The second trait is that terrorists wish to remain covert and avoid detection by defensive forces. Crowds within public areas facilitate this function, as the surveillance resource is drawn to other people the adversary is able to carry out and complete an attack before the surveillance resource is able to screen the terrorist. Alternatively, an immigrant could use the crowd in a public area to provide coverage whilst he passes across a border illegally. Given the limitation of the surveillance resource, such outcomes are possibilities. These possibilities are enhanced when the adversary is capable of behaving as a decision-making agent, *choosing which area to attack*. Consequently, this adversarial capability must be recognised by the defensive forces. Hence, another major challenge in the process of defensive surveillance is controlling surveillance technology in the presence of an *adversary capable of making strategic decisions*.

The main motivation of this thesis is to consider the operational challenges posed by defensive surveillance: How should a surveillance resource be utilised in real time to minimise the impact of adversarial threats? In particular, we wish to consider relevant scenarios within the context of the threats faced in public areas as described above. We will develop *multiclass queueing system models* for the defensive surveillance scenarios. The queueing systems will contain the main features of the scenarios; the notion of customer abandonment or impatience, and the notion of an adversarial customer seeking to enter the system among other customers, conduct an illicit activity, and leave without detection. The operational challenges of defensive surveillance then become the challenges of developing control policies within the queueing systems with respect to some stated objectives. Variation of assumptions and objectives in each model will give rise to different defensive surveillance scenarios in practice. Furthermore, even in the absence of an adversarial threat, the challenge of developing control policies in multiclass queueing systems with customer abandonments is an interesting and relevant problem

in other application areas such as the operation of call centres. Consequently, through consideration of the main defensive surveillance motivation we also hope to shed light on these more general problems.

1.2 Challenges and Objectives

In this section, we develop the defensive surveillance scenarios motivated previously and which will be the focus of the thesis. We consider a large public area which can be divided into multiple sub-areas. As such, we refer to k public areas to actually mean the sub-areas which aggregate to the original larger area. When we refer to public areas, we mean areas in which people are free to come and go at their own will. For example, a train station which could be divided into sub-areas of different platforms, ticketing lobbies, and walkways. Our description of the surveillance process in these public areas follows the one given by Lin et al. (2009) in their work which uses a queue to model an antiterrorist surveillance system.

Arrays of video cameras monitor the areas continuously, relaying video-feeds in real time to a control centre operated by a security team. The security team uses the video-feeds to screen people within the areas in two phases: an initial classification phase, and a screening phase. In the first initial classification phase, as soon as people arrive into an area they are assessed visually and classified as members of one of two groups: *nonsuspects and suspects*. In this classification, some civilians and all adversaries will be classified as suspects, while all nonsuspects are civilians. Nonsuspects are not subject to any further screening and are subsequently ignored by the security team. Suspects become eligible for second-phase screening. In second-phase screening, biometric signatures are extracted from a suspect, for example face structure and hair characteristics, and matched against a database. If no match is found the suspect is reclassified as a nonsuspect and is subsequently ignored. However, a positive match yields the identity of the suspect and appropriate action can be taken. If this identity is not of relevance to the security team then the suspect is reclassified as a nonsuspect and is subse-

quently ignored. However, if this identity is of relevance, for example the suspect is a known criminal or terrorist, then security forces can be notified to intervene.

People arrive at random into each of the k public areas which immediately initiates the first phase of the screening process. We assume that this phase is fast and so the time taken is negligible. The first phase screening identifies a set of suspects in each area. Whilst in an area, suspects conduct their own daily business and leave the area when this concludes. Since some suspects are potentially adversaries, this daily business could be something harmful. We say that each suspect has a lifetime in the area and the length of this lifetime is random and independent for each suspect. Expiration of the lifetime for a suspect corresponds to the suspect leaving the area. During this lifetime, each suspect is available for second-phase screening. The security team is only able to engage in second-phase screening for one suspect from one of the k areas at any given time. The processing time taken to complete second-phase screening is random and independent for each suspect. Lifetimes can expire during second-phase screening, in which event the screening is incomplete and the security team moves onto another suspect.

Each public area represents a target for an adversary such as a terrorist or illegal immigrant. The public area itself and people within it may be the target, or infrastructure located within an area may be the target, or indeed the area may provide passage to the target. The objective of the adversary is to enter one of the public areas, conduct an illicit activity such as planting or detonating a bomb, and leave (if applicable) before second-phase screening of the adversary can be completed. If an adversary enters an area, they are inconspicuous, appearing to the security team to be just like every other person in the area. The random time taken to conduct the illicit activity is the lifetime of the adversary within the area, and hence the time for which they are available for second-phase screening. The only way the security team is able to uncover the identity of each suspect is by *full second-phase screening* and so there is no way to know a priori when the adversary is in an area or which suspect it is. The purpose of surveillance

is then to screen suspects across all public areas in order to successfully identify an adversary before he is able to achieve his goal. The operational problem is to identify a *decision-making rule which declares which suspect should be engaged in second-phase screening* at each point in time to ultimately minimise the probability of evading detection or damage inflicted by the adversary.

We now develop a mathematical model for the surveillance problem faced by the security team. The scenario described can be modelled as a multiclass queueing system with customer abandonments and a single server. The k public areas are modelled as k parallel queues and the people classified as suspects in the areas as customers in the queues. We can also refer to the parallel queues as customer classes. Customers arrive into queue i according to a Poisson process with rate λ_i. The security team controlling the surveillance resource is modelled as the server, where *service of a customer is equivalent to second-phase screening.* Each queue i customer has two random quantities: a service requirement and a lifetime. Service requirements in queue i represent the time taken for the server to successfully serve a customer or equivalently the time taken to successfully complete second-phase screening of a suspect. These are given by independent and identically distributed random variables. Lifetimes in queue i represent the lengths of time customers stay in the queue and are given by independent and identically distributed random variables. Note that we allow the stochastic distributions of the arrival rates, service requirements, and lifetimes to differ between queues to represent areas with different characteristics. For example, longer lifetimes may reflect suspects waiting on a train platform as opposed to walking down a hall and longer service requirements may reflect the longer time needed to extract biometric signatures from suspects walking rather than standing. Expiration of a customer lifetime before completed service is referred to as a customer abandonment, where abandonment can occur *before or during service.*

After customers have arrived into a queue they queue for service until ultimately either service completion or abandonment occurs, after which they are

considered as having left the system regardless. Although suspects would not physically form a queue in practice, when we model them as customers we adopt the convention of a queue to refer to the order in which they arrived. Service of a customer can either be preemptive or nonpreemptive. Preemptive service means that the server can stop a service at any time and switch to another customer, which is equivalent to the second-phase screening process being stopped and another beginning on another suspect. Nonpreemptive service means that once the server begins a service, this continues until it is either completed or the customer abandons. In other words, the second-phase screening can only be stopped by the suspect leaving the area.

We do not model the adversary as a customer who actually enters the system, but rather as a potential customer possessing the ability to join or attack any of the k queues. If the adversary attacks queue i then his service requirement is identically distributed to that of every other queue i customer, and similarly for the lifetime of the adversary. In other words, when attacking queue i the adversary behaves like every other customer in that queue. Once in the system there are two possible outcomes, *service completion* and *abandonment* representing *unsuccessful* and *successful* attacks respectively. The goal of the adversary is to abandon his queue before being served to completion. If the adversary is able to abandon a queue, he is able to incur a *fixed amount of damage* in the process.

At any given point in time, the number of customers currently in each queue gives the state of the system. A service policy is a rule which uses the system state to decide which customer to serve from which queue at each point in time. Clearly, we have implicitly assumed here that the server always knows the state of the system in real time. The service policy adopted by the server reflects the way in which the security team controls the surveillance resource. Use of a specific service policy directly determines the probability of each outcome available to the adversary should he attack a specific queue. The choice of service policy is the mechanism through which the security team can affect the probability of the

adversary being able to abandon each queue and so complete a successful attack.

Importantly, the information available to and capability of the adversary dictates his decision-making ability. Moreover, the information the server has regarding the capability of the adversary affects the choice of service policy. In this thesis, we consider different surveillance scenarios based on three variations of information available to both the server and adversary and the capability of the adversary. The scenarios are as follows:

- *Random adversary*: The adversary attacks the different queues randomly according to a probability distribution which is known to the server.

- *Strategic adversary who chooses where to attack*: The adversary attacks the different queues according to a probability distribution under his control, which the server does not know.

- *Strategic adversary who chooses where and when to attack*: The adversary attacks the system based on knowledge of the state of the system. The server does not know where or when the adversary will attack.

The objective throughout this thesis is to develop service policies and surveillance strategies which *minimise the probability of abandonment* experienced by the adversary in each scenario. When the adversary can inflict different amounts of damage in each queue, the objective generalises to *minimising the expected damage* inflicted by the adversary. We will often use the two objectives interchangeably throughout the thesis.

In the random adversary scenario, the decision of the adversary is known to the server. The defensive surveillance problem can be seen as a special case of a more general stochastic scheduling problem in which a server attempts to maximise the average reward rate in a system affected by abandonments. This stochastic scheduling problem and other variants have been studied in recent years by numerous authors including Glazebrook et al. (2004), Atar et al. (2010), and Down et al. (2011) to name just a few. We develop new results and approaches for this

problem which complement the existing literature. In the strategic adversary scenarios, the unknown decision of the adversary poses a different problem which can be addressed using ideas from game theory or by formulation as a robust optimisation problem. These scenarios share a similar motivation to a number of other defender-attacker problems, see for example Lin et al. (2013). It is our belief that the strategic adversary scenarios within this thesis, in particular the use of a multiclass queueing system with abandonments to model the defensive surveillance of public areas, are novel and have not been previously studied. The closest known work is that of Lin et al. (2009) who model the defensive surveillance of a single public area as a single queue with impatient customers. However, the authors do not consider the case of multiple areas and an adversary capable of making strategic decisions which is the major feature of the strategic adversary scenarios within this thesis.

1.3 Thesis Structure

In this section, we provide an outline of the remaining chapters of the thesis, highlighting the main features, approaches, and contributions of each chapter. A paper based on the combined work of Chapters 3 and 4 has been published in INFORMS Journal on Computing; see James et al. (2016). We begin in Chapter 2 with a comprehensive review of the literature relevant to the subsequent chapters of the thesis. The literature review first considers broad theoretical areas which underpin the methods used within the thesis. We then consider literature concerning stochastic scheduling with customer abandonments as this is highly relevant to the problem formulated to address the random adversary defensive surveillance problem. Finally, we consider a class of related literature we refer to as defender-attacker problems as this shares similar motivations to the strategic defensive surveillance problems.

The remainder of the thesis studies the three defensive surveillance scenarios described in Section 1.2, each sharing the common foundations outlined. We dis-

tinguish between the scenarios based on the capability of the adversary and the knowledge of the server regarding the adversary. Together these form a set of defensive surveillance models which cover a number of potential scenarios which may occur in real-world security operations. We will see that the operational strategies suggested in each scenario are different, indicating the importance of identifying which scenario prevails before identifying the appropriate defensive actions.

Chapter 3 - Defence against Random Adversaries: Priority Policies

In Chapter 3 we study the problem of a single server faced with impatient customers spread across k customer classes. Each customer class can be seen as a separate queue, each containing customers of the same type. Each customer has a random lifetime during which he is available for preemptive service. Should a customer be served to completion before this lifetime expires, a reward is received; otherwise, the customer abandons the system and a penalty is inflicted. In addition, each customer in the system incurs a holding cost at a fixed rate. The random lifetimes, random service times, rewards, penalties, and holding costs are dependent upon the customer class. The goal of our analysis is to determine a service policy to maximise the long-run reward rate net of penalties and holding costs incurred. We show that the three parameters in the reward structure can be *consolidated into a single parameter* through a proper transformation and without loss of generality we are able to consider a pure-reward problem by ignoring the abandonment penalty and holding cost. The problem is studied with respect to arbitrary rewards and so in this sense is rather general.

As previously discussed, the principle motivating application of our model is to defensive surveillance. We consider the scenario of a random adversary who attacks the system in its long-run steady state conditions at random according to some probability vector $\mathbf{p}$ which assigns probability p_i to attack of queue i. The adversary could have chosen $\mathbf{p}$ entirely at random, or through some knowledge of the system in an attempt to maximise the damage he can inflict. In either

case, $\mathbf{p}$ is known to the server. By expressing the class dependent rewards for service completions in terms of the probability vector $\mathbf{p}$, the goal of maximising the long-run reward rate is equivalent to maximising the probability of serving an adversary. This objective is consistent with our general objective throughout the thesis of developing service policies which minimise the probability of abandonment experienced by the adversary, or minimising the expected damage inflicted. The random adversary scenario is hence studied in Chapter 3 as a special case of a more general stochastic scheduling problem featuring customer abandonments.

We model the problem as a Markov Decision Process (MDP) and use standard methods of Dynamic Programming (DP) to compute the optimal service policies. However, the computational burden of such methods make them only practical in most problems with *up to three customer classes*. Hence, our focus is on the development of strongly performing heuristic policies with a preference for operationally simple policies with strong reward rate characteristics.

We consider a set of priority policies which are effective across much of the problem's parameter space. Such policies serve customers according to a strict priority ordering among the customer classes. In the case where the system is overloaded, it has been shown in the literature that the $R\mu$ *rule*, a priority policy that ranks all customer classes based on the product of reward R and service rate μ, performs well, since it maximises the instantaneous reward rate (Atar et al., 2010; Ayesta et al., 2011; Verloop, 2014; Larrañaga et al., 2014). To complement the $R\mu$ rule in the light-traffic case, the main contribution of Chapter 3 is to present the $R\mu\theta$ *rule*, which ranks all customer classes based on the product of R, μ, and the abandonment rate θ. This ranking was proposed in Section 2 of Glazebrook et al. (2004) for batch problems, and we extend its application to systems with customer arrivals. We prove that the $R\mu\theta$ rule is *asymptotically optimal* as customer abandonment rates approach zero in light traffic systems. This result sheds light on cases where the time spent by individuals in the public areas tends to be large relative to the time taken to surveil them. Extensions of this

result are discussed for other model classes of interest, namely a multiserver version of our system and in Klimov Networks (see Klimov (1974) and Klimov (1978)). Further to this we develop a priority policy known as the *PaS policy* by applying a pairwise swapping mechanism to both the $R\mu$ and $R\mu\theta$ rules to search for an improved policy. Finally, we compare the $R\mu$ and $R\mu\theta$ rules both analytically and numerically in light traffic systems in a set of numerical experiments.

Chapter 4 - Defence against Random Adversaries: Approximate Policy Improvement

In Chapter 4, we further study the problem of developing effective service policies for multiclass queueing systems with customer abandonments, wherein the random adversary defensive surveillance scenario is a special case. Whereas Chapter 3 considers priority policies, Chapter 4 considers an approach which aims to overcome the computational intractability of DP methods in larger systems. This is consistent with the objective in the research field of approximate dynamic programming (ADP). In particular, we consider the policy iteration method (see Howard (1960)) in which the difficulty of utilising this method lies in the computation of bias functions at each state of the system under a given service policy. The main contribution of Chapter 4 is to develop an *approximate policy iteration* (API) method for the problem. For a given policy, the API method uses simulation to estimate bias values for a set of carefully chosen states, and then uses these values to interpolate the bias function for all states. This approximate bias function allows us to run policy improvement to obtain a new policy. The logic which underlies this method is that if the approximate bias functions are accurate representations of the true bias functions then the new policy will be at least as good as the initial policy.

Our API approach can be viewed as a refined ADP implementation with two distinctive features: (1) a suite of strongly-performing priority policies to initialise the API algorithm, and (2) a simulation / interpolation methodology to fit the bias surface by *estimating biases both at states that are frequently visited, and also*

at a carefully chosen set of widely spread states. Our numerical results indicate that, in most cases, the best priority policy is nearly optimal in systems with 2 or 3 customer classes and we have an effective service policy of simple structure. In the cases where it is not, the API method invariably tightens up the gap substantially and provides an improved policy, albeit of more complex structure. The structure and performance of API brings to light a trade-off between policy simplicity and quality, which potentially has useful managerial implications. In our motivating random adversary defensive surveillance application, even small improvements in reward rate performance can be of high practical importance. In Chapter 4 we also conduct an extensive numerical study to compare the performance of a range of heuristic policies, notably the API approach and the priority policies of Chapter 3. To assess the closeness to optimality of a given policy, we use a linear programming relaxation to develop *upper bounds* for the reward rate achievable in the system. The upper bounds are used to evaluate our heuristics in systems with 5 customer classes.

Chapter 5 - Defence against Strategic Adversaries Who Choose Where To Attack

In Chapter 5 we consider the defensive surveillance scenario in which a strategic adversary chooses which queue to attack. We adopt the same model as in the random adversary scenario of Chapters 3 and 4. Once more, the adversary attacks the system in its long-run steady state conditions according to some probability vector $\mathbf{p}$ which assigns probability p_i to attack of queue i. The difference in this scenario is that $\mathbf{p}$ is not known by the server. Given the capability of the adversary to choose any $\mathbf{p}$, the server wishes to find a robust service policy which achieves a low expected damage, regardless of the choice of $\mathbf{p}$. Whilst the server does not know the decision of the adversary, namely $\mathbf{p}$, the adversary does not know the decision of the server of which service policy to use. The objective of the adversary is to maximise the expected damage he can inflict on the system by deciding which

queue to attack. The objective of the server, on the other hand, is to determine a randomised service policy which minimises the expected damage inflicted. Given the lack of knowledge of each others decision, we model the interaction of the server and the adversary as a simultaneous move two-person zero-sum (TPZS) game.

We develop an optimal solution to the TPZS game from both the perspective of the server and the adversary, where in each case the optimal decision for each player is independent of the optimal decision of the other. We show that we can determine the optimal randomised policy of the server by formulating and solving an appropriate linear program. However, this is only possible for most systems of up to three queues. Subsequently, there is a need to develop heuristic approaches to the problem which can be used in larger systems. Our heuristic approach is motivated by our consideration of the problem from the perspective of the adversary. We show that the optimal probability vector $\mathbf{p}$ for the adversary can be found by formulating a *convex optimisation problem* which can be solved through *Kelley's cutting plane* (KCP) method (see Kelley (1960)). The KCP method provides a strong link to the random adversary scenario of Chapters 3 and 4. We restrict the server to a finite set of service policies which consists of the optimal service policies against given choices of $\mathbf{p}$ by the adversary. We see that the KCP method is equivalent to an iteratively expanding finite matrix game in which the server is restricted to using only policies in this set and more policies are added to the set. The value of the matrix game converges to the optimal expected damage as the policy set increases and the server can achieve this by adopting a mixed strategy over the policy set. However, the KCP method is computationally intractable for systems of more than three queues. Our heuristic approach is a *heuristic application of the KCP method*. We formulate a finite matrix game between the server and adversary and iteratively populate a finite set of service policies using the strongly performing heuristic policies developed in Chapters 3 and 4 for the random adversary scenario. This yields the heuristic cutting plane (HCP) method and an enhanced version of this method. We assess

the performance of our heuristic methods in a set of numerical experiments which indicate strong performance with favourable amounts of computation.

Chapter 6 - Defence against Strategic Adversaries Who Choose Where and When To Attack

In Chapter 6 we extend the strategic adversary scenario of Chapter 5. In addition to choosing which queue to attack, the adversary can also choose when to attack the system. In this case, the server wishes to find a robust service policy which provides a performance guarantee against any choice of queue and time the adversary could make. We first study a single queue system in which the adversary does not have a set of queues to choose from and *only chooses the time* at which he attacks the queue based on the state of the system. The state consists of two elements: the volume state which represents the number of customers in the system, and the server state which represents which customer is in service and how long that customer has been in service. We consider two scenarios in which the adversary chooses when to attack based on both the volume and server states or only the volume state. We refer to these as the full and partial information scenarios respectively. The first contribution of this chapter is to show that the *last come first served* (LCFS) policy, namely the policy which serves the most recently arrived customer, is *optimal* in a number of versions of the single queue problem. The strength of the LCFS policy is based on the fact that the abandonment probability of the adversary depends on the arrival process after the adversary has joined the queue, and not on the state of the system. In practice, the server may or may not know the capability of the adversary and the adversary may or may not know the state of the system. If the adversary is not as capable as assumed, our analysis provides an *upper bound* on the abandonment probability of a less capable adversary.

We generalise the problem to systems with multiple queues and consider two ways in which the security team is able to control the system. With decentralised

control, the decision of which queue to serve and which customer to serve are separate. The choice of which customer to serve is made locally, once the server is allocated to a queue. With centralised control, the server can decide which customer to serve from all customers in the system. In both cases, the objective of the server is to determine a robust service policy which provides the best performance guarantee against the best decision the adversary could make. However, in both cases it is very difficult to solve the optimisation problem. Hence, our goal is to develop *heuristic approaches* to the problem which provide strong upper bound performance guarantees for the server.

The first step is to simplify the optimisation problem by arguing that the worst case for the server occurs when the adversary attacks when the number of customers in each queue approaches infinity. In designing a heuristic decentralised policy, we develop heuristics based on the strong performance of the LCFS policy in a single queue system. We develop the *Departure Reselection* (DR) and *Service Reselection* (SR) policies which both allocate the server to a queue according to a probability vector (called the reselection vector) and then serve according to the LCFS policy within that queue. We seek to find the best performance guarantees for the server from within the classes of DR and SR policies by optimising the reselection vector. We define a method based on the work by Regis & Shoemaker (2005) which intelligently utilises *simulation within a response surface method* for global optimisation problems.

In the case of centralised control, the server knows and can compare the arrival times of all customers in the system. The server can build this information into the service policy, providing greater capability than in the case of decentralised control, hence the server can achieve better performance guarantees. In a multiple queue system, we extend the definition of the LCFS policy to be the policy which serves the most recently arrived customer into the system. In a *symmetric* multiple queue system, we show that the LCFS policy provides the best performance guarantee as a consequence of our analysis of single queue systems. We would

not expect the LCFS policy to provide the best performance guarantee in general *asymmetric* systems, so we develop a heuristic approach based on the LCFS policy. We develop the *Last Come First Served with Probabilistic Skipping* (LCFS-PS) policy to improve upon the LCFS policy. The LCFS-PS policy is parametrised by a vector of probabilities (called the skipping vector). The server orders the most recent arrivals in each queue and attempts to serve the first customer, but has the potential to skip over this customer. The server repeats this down the order until a customer is selected for service. The rationale is to reduce the abandonment probability in the most damaging queue under the LCFS policy by providing more service to that queue, at the expense of the other queues. We seek to find the best performance guarantees for the server from within the class of LCFS-PS policies by optimising the skipping vector. We adopt a similar approach to that taken for the DR and SR policies. We demonstrate the superior performance of the heuristic approach based on LCFS-PS over the decentralised approaches based on DR and SR in sets of numerical examples. We show that our carefully designed heuristics achieve *large performance improvements* over simpler policies such as the LCFS policy and a round robin (RR) policy which aims to treat each queue fairly.

Chapter 2

Literature Review

In this chapter we provide a review of various fields of literature which are relevant to the defensive surveillance problems and models studied within the thesis. We begin by reviewing the three subject areas of Markov decision processes, approximate dynamic programming, and game theory under the heading of theoretical underpinnings. Much of the modelling framework and many of the methodological tools within the rest of the thesis are provided within these areas. Hence, the review mainly focuses on aspects closest to the needs to the thesis from these otherwise vast subject areas. We then consider literature concerning stochastic scheduling, with particular emphasis on customer abandonment in the context of stochastic scheduling. This area is relevant to Chapters 3 and 4 of the thesis. Finally, we consider the literature related to defender-attacker problems. This area is relevant to Chapters 5 and 6 of the thesis.

2.1 Theoretical Underpinnings

2.1.1 Markov Decision Processes

There are many situations in which a decision-making agent wishes to make a sequence of decisions in order to optimise some previously stated objective. In such a sequential decision process, decisions are made based upon the information

available at a point in time, under the uncertainty of the future effect of those decisions. That is, each decision incurs an immediate contribution to some ongoing objective, but also affects the ability to make future decisions and so indirectly affects future contributions. Optimal decision-making must balance the trade-off between immediate effects and possible future effects.

Elements of Markov Decision Processes

Consider a system in which decisions are to be made at evenly spaced discrete points in time, known as *decision epochs*, over an infinite time horizon. At a given decision epoch t the system occupies one of a number of possible *states*. The set of possible states available is known as the *state space* $\mathcal{S}$. The state s allows the decision-maker to choose an action a from a finite set of feasible or allowable *decisions* $\mathcal{A}_s$ in that state. As a result of choosing action $a \in \mathcal{A}_s$ in state s at decision epoch t, then independent of the past, two things occur. Firstly, a non-negative *expected reward* $R(s, a)$ is received. Secondly, the system moves to a new state s' according to some *transition probability* $p(s'|s, a)$. From this new state, this process repeats. A *policy* π is a sequence of rules or procedures for action selection in each state at all decision epochs. A *stationary, Markovian* policy selects actions based only on the current state and not time, always selecting the same action in the same state. The policy is *deterministic* when actions are selected with certainty. Following such a policy through time yields a sequence of rewards for the decision-maker. The objective of the decision-maker is to determine a policy which maximises some function of the reward sequence generated. The dependence of expected rewards and state transitions only on the current state and action and not the prior history of the process means that such sequential decision processes are known as Markov Decision Processes (MDP). Whilst it is possible to adopt non-stationary policies where actions in states can change through time, Puterman (1994) established that it is often sufficient to only consider stationary, deterministic policies.

Discounted Reward Criterion

Over an infinite horizon the objective can take three main forms: maximising the expected total reward (assumed finite), maximising the expected total discounted reward, and maximising the average reward rate over all policies. In the latter two cases rewards $R(s,a)$ are assumed to be bounded and finite. Consider the case of an expected total discounted reward criterion in which rewards $R(s,a)$ are discounted at rate α, $0 < \alpha < 1$. Denoting the state and action at decision epoch t by s_t and a_t respectively, the objective of the decision maker is to maximise among all policies π

$$V^{\pi}(s) = \lim_{n\to\infty} \mathbb{E}^{\pi}\left\{ \sum_{t=1}^{n} \alpha^{t-1} R(s_t, a_t) \right\}.$$

This limit is guaranteed to exist when the rewards are bounded. The value of policy π for initial state s is given by $V^{\pi}(s)$ and the corresponding value of the MDP is given by

$$V^{*}(s) = \sup_{\pi} V^{\pi}(s).$$

A policy π^* is said to be discount optimal if for every initial state s we have that $V^{\pi^*}(s) = V^{*}(s)$. An effective method for solving MDPs first developed by Bellman (1957) is known as *dynamic programming* (DP). The idea of DP is reflected in the *Principle of Optimality* which states that:

> *"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."*

This can be formulated as an equation which links the decision at the current epoch with the decision at the next epoch. The resulting equation is known as the *Bellman equation* or *optimality equation*:

$$V^{*}(s) = \max_{a\in\mathcal{A}_s} \left(R(s,a) + \alpha \sum_{s'\in\mathcal{S}} p(s'|s,a) V^{*}(s') \right). \tag{2.1}$$

The theory of MDPs is given excellent coverage in the book by Puterman (1994). This theory shows the existence of a deterministic, stationary optimal policy which corresponds to selecting actions in states which maximise the right hand side of (2.1). The value function $V^*(s)$ captures the value of being in state s and following an optimal policy forward.

Average Reward Criterion

In the case of an average reward criterion, in an analogous way, the objective is to maximise among all policies π

$$g^\pi(s) = \lim_{n\to\infty} \frac{1}{n}\mathbb{E}^\pi\left\{ \sum_{t=1}^{n} R(s_t, a_t) \right\}.$$

This limit exists when the rewards are bounded, the state space is finite, and the policy is stationary. When the Markov chains induced by deterministic, stationary policies are unichain or ergodic, the average reward or gain $g^\pi(s)$ of policy π is constant and equal to g^π for all states. The average reward version of the optimality equation is given by

$$g^* + h^*(s) = \max_{a\in\mathcal{A}_s} \left(R(s,a) + \sum_{s'\in\mathcal{S}} p(s'|s,a)h^*(s') \right). \tag{2.2}$$

The quantity g^* is the optimal *gain* or maximum average reward rate and the function $h^*(s)$ is known as the *bias function* of state s under the optimal policy. The bias function measures the asymptotic relative difference in total reward which results from starting in state s as opposed to some other reference state. Another interpretation is that the bias is the expected total difference between reward received starting from state s and the stationary reward received starting from a steady state system. This difference occurs in the early phase of the decision-making process. A deterministic, stationary policy which maximises the right hand side of (2.2) is said to be bias-improving and is an average optimal policy.

From the previous discussion, in the expected total discounted reward and av-

erage reward cases respectively, if the optimal value function $V^*(s)$ or optimal bias function $h^*(s)$ was known for each state then an optimal policy can be determined from the actions which maximise the right hand side of (2.1) and (2.2) respectively. However, these quantities are unknown and difficult to calculate and so there is a need for other ways to identify optimal policies. The two main approaches are known as *value iteration* and *policy iteration* and were introduced by Bellman (1957) and Howard (1960) respectively. Both iterative methods have proven convergence properties. Other approaches such as *modified policy iteration* and *relative value iteration* are variants of these two main approaches, with the former being a cross between value and policy iteration. Another method based on a *linear programming reformulation* of the optimality equation was proposed by d'Epenoux (1963).

Value Iteration

In the expected total discounted reward case value iteration begins with a set of initial arbitrary bounded functions $V_0(s)$ for each state and finds $V_1(s)$ as the solution of (2.1), using $V_0(s)$ in the right hand side. In general, for $n > 1$, $V_n(s)$ is the solution of (2.1), using $V_{n-1}(s)$ in the right hand side. In theory $V_n(s)$ converges uniformly to $V^*(s)$ as $n \to \infty$. In practice, this procedure is repeated for a finite number of iterations until a termination criterion is met. The termination criterion holds that largest difference in the value functions between successive iterations across all states, as measured by the supremum norm, is smaller than some threshold controlled by a small parameter ϵ. This condition guarantees an error of less than $\epsilon/2$. For further details, see Puterman (1994, p161). The deterministic, stationary policy which solves the optimality equation in the final iteration is said to be ϵ-optimal. In the average reward case the algorithm is similar with termination occurring when the increase in the value functions for all states is close to constant. This constant increase approximates the gain g^*, the relative difference in value functions approximate the bias functions $h^*(s)$, and again the

deterministic, stationary policy which solves the optimality equation in the final iteration is said to be ϵ-optimal. Note, the gain is guaranteed to be positive due to the assumption of a non-negative expected reward.

Policy Iteration

Policy iteration consists of two main steps: *policy evaluation* and *policy improvement*. Suppose there is an arbitrary initial stationary policy π. Policy evaluation calculates the value functions or bias functions associated with π using a version of the optimality equation which drops the max operator, prescribes actions according to π, and replaces $(*)$ with π. Policy improvement uses these value functions or bias functions to define a new policy π' which is at least as good as π. This is done by determining actions which maximise $R(s,a) + \sum p(s'|s,a)h^{\pi}(s')$ in the case of average rewards. This procedure iterates through the class of deterministic, stationary policies until no further improvements can be made, at which point the current policy is optimal.

Continuous-time MDPs and Uniformisation

Standard MDP theory assumes that decision epochs occur at a discrete set of time points which are evenly spaced. However, there are many situations in which the times between successive decision epochs are random. These random times are dependent upon the states and actions. This naturally occurs in the control of queueing systems whereby state transitions occur at the random arrival instants or random departure instants of customers. Sequential decision processes of this form are considered in continuous time and are more generally known as *semi-Markov Decision Processes*. The special case in which the times between decision epochs each follow an exponential probability distribution is known as a *continuous-time Markov Decision Process*. It is possible to analyse these decision processes within the discrete-time framework outlined through the use of *uniformisation*, notably first applied by Lippman (1975) and Serfozo (1979). Uniformisation converts the

original continuous-time process into an *equivalent discrete-time process*. Suppose decision epochs occur at state transitions and the time taken to transition from state s into a new state under action a is exponentially distributed with rate $\Delta_s(a)$. The uniformisation defines a new transition rate Δ such that $\Delta_s(a) \leq \Delta$ for all s and a and allows fictitious transitions from a state to itself. A transition out of state s at rate $\Delta_s(a)$ in the original process is statistically identical to leaving at faster uniform rate Δ, but returning back to it with probability $1 - \Delta_s(a)/\Delta$ in the uniformised process. Transition probabilities in the uniformised process are denoted $\tilde{p}(s'|s,a)$, rewards are denoted $\tilde{R}(s,a)$, and in the case of average rewards the gain is denoted $\tilde{g}$. These elements allow the formulation of a uniformised discrete-time MDP with optimality equation given by:

$$\tilde{g}^* + h^*(s) = \max_{a \in \mathcal{A}_s} \left(\tilde{R}(s,a) + \sum_{s' \in \mathcal{S}} \tilde{p}(s'|s,a) h^*(s') \right).$$

In this uniformised decision process the unit of time is the expected transition time Δ^{-1} and the so the optimal gain is interpreted as the maximum *average reward per transition*. Multiplication by Δ converts this back into unit time. For a more comprehensive coverage of MDP theory, the reader is once more referred to the book by Puterman (1994).

2.1.2 Approximate Dynamic Programming

The formulation of a sequential decision problem as an MDP provides a powerful framework from which to identify optimal decision-making strategies. The standard approach of DP to solving such problems comes in the form of value iteration or policy iteration. These algorithms require the evaluation of value functions at every state in the state space, an assessment of all feasible actions which can be taken from each state, and consideration of all possible outcomes following each action. Identification of optimal policies through these methods in most sequential decision problems which occur in practice is typically not feasible, even with the most powerful computers. This issue was famously referred to by Bellman (1957)

as the *curse of dimensionality*. As the dimension of the number of variables within the problem grows, the actual size of the problem grows exponentially. For example, if the state space has n dimensions with each dimension taking m possible values, then there can be up to m^n states. Alternatively, if the action space has k dimensions with each dimension taking l possible values, then there can be up to l^k possible actions. When these values are large, tasks such as considering the value of all states or considering all actions are not achievable. Aside from the curse of dimensionality, DP requires that almost all aspects of the system are known, specifically a reward function and a set of transition probabilities under each state and action. In many settings these are not known, adding another issue to the solution of an MDP. However, despite these restrictions, there is still a need for near yet suboptimal policies or policies which simply improve existing strategies of decision-makers. Building upon the framework set out by DP, this has motivated the research field of *Approximate Dynamic Programming* (ADP).

ADP is a term which collectively describes methods designed to solve large, complex sequential decision problems to near optimality. It is not readily defined given the vast array of different approaches which can be taken, each with different components. Methods which originated in the artificial intelligence community are termed Reinforcement Learning in that systems "learn how to make good decisions by observing their own behaviour, and use built-in mechanisms for improving their actions through a reinforcement mechanism" (Sutton & Barto, 1998). This statement captures the essence of ADP since most methods are founded on the integration of four main themes: the *framework and methods of DP* for solving MDP problems; the use of *simulation* to replicate the behaviour of a stochastic system; the idea of *learning* from observations made within simulated environments; and the use of *function approximations* as means of representing value functions. Synthesis of these ideas within ADP presents the opportunity to identify strongly performing suboptimal policies. However, the vast range of ways in which algorithms can be designed is often the central challenge in ADP, and focus lies in

finding methods which work harmoniously together. The books by Bertsekas & Tsitsiklis (1996), Sutton & Barto (1998), Bertsekas (2012), and Powell (2011) give the best extensive coverage of ADP methods.

In an MDP the *value functions* $V^*(s)$ or $V^\pi(s)$ of an optimal policy or policy π in state s are of fundamental importance. Knowledge of these value functions allows for improvement of policies and the identification of optimal policies. The curse of dimensionality renders methods for evaluating these functions infeasible and so central to ADP are *value function approximations* $\tilde{V}^*(s)$ or $\tilde{V}^\pi(s)$. *Approximate policy iteration* (API) is an approach to ADP based on the policy iteration algorithm. It takes suitable approximations $\tilde{V}^\pi(s)$ of policy π and using an expected total discounted reward criterion as an example, identifies an improved policy π' with actions given by

$$\pi'(s) = \underset{a \in \mathcal{A}_s}{\operatorname{argmax}} \left(R(s,a) + \alpha \sum_{s' \in \mathcal{S}} p(s'|s,a) \tilde{V}^\pi(s') \right). \tag{2.3}$$

This performs an *approximate policy improvement step* based on an implicit set of approximate policy evaluations $\tilde{V}^\pi$. If the approximations are accurate, similar actions will be taken to those taken if the true value functions were known. Much of the difficulty in ADP lies in performing accurate approximate policy evaluations. The case in which there is a value function approximation stored in memory for every state is known as a *lookup table representation*. Lookup table representations are not practical in the case of massive state spaces and so *compact representations* are used. A compact value function approximation for policy π is written $\tilde{V}^\pi(s|\boldsymbol{\theta})$ and represents the true value function through some function with relatively few parameters $\boldsymbol{\theta}$. With this scheme, only the parameters and functional form need to be stored in memory and estimates can be readily computed for every state when required. This is a key idea in circumventing the challenge of a massive state space.

The use of a compact value function approximation requires the selection of an *approximation architecture* which defines the class of parametric functions which

can be used for $\tilde{V}^{\pi}(s|\boldsymbol{\theta})$. Architectures can be broadly separated into two main categories: *linear and non-linear*. A linear architecture is given by

$$\tilde{V}^{\pi}(s|\boldsymbol{\theta}) = \sum_{k=0}^{K} \theta_k \phi_k(s).$$

This expresses the value function approximation as a linear function of $K+1$ parameters and *basis functions*. Examples of architectures include polynomial regression, kernel regression, neural networks, radial basis functions, splines, and interpolation schemes to name a few. A challenge in ADP is finding an architecture which is rich enough for accurate approximation at all states. Once an approximation architecture is fixed, the idea is to find the best approximation within the defined class through the choice of parameters so that the true value function is accurately represented. This is referred to as *training the approximation*. However, it is not possible to train with respect to the true value functions. There is a need for estimates with which to train the approximation. Simulation can be used to directly obtain samples of value functions at a set of *representative states* $\tilde{S}$. Suppose for each $s \in \tilde{S}$ there are $M(s)$ samples of $V^{\pi}(s)$ and the mth such sample is denoted $v(s,m)$. The approximation is trained by solving the following *least squares* optimisation problem

$$\min_{\boldsymbol{\theta}} \sum_{s \in \tilde{S}} \sum_{m=1}^{M(s)} \left(\tilde{V}^{\pi}(s|\boldsymbol{\theta}) - v(s,m) \right)^2.$$

This naturally poses the following questions: How can the least squares problem be solved? How are the representative states identified? How are samples obtained? One possibility is to simulate the system using actions derived from π and observe those states visited most often by the simulation and let these comprise the representative set $\tilde{S}$. Then simulate trajectories or episodes starting from initial states within $\tilde{S}$, observe the rewards accumulated, and let the average over many trajectories form sample estimates. This results in a set of training pairs, state and sample estimates. The least squares problem can be solved using stan-

dard optimisation techniques. Alternatively, these issues are often addressed in a simultaneous manner. A simulated trajectory will reveal a sequence of states visited by policy π. All states within the trajectory comprise $\tilde{S}$ and the rewards accumulated from intermediate states along the trajectory can be used as sample estimates starting from those intermediate states. The least squares problem can be solved by iterative gradient methods, either in batch mode at the end of a trajectory or incrementally along the trajectory. These methods have roots in the *stochastic approximation* method of Robbins & Monro (1951). A popular way to implement iterative gradient methods is through temporal difference (TD) learning (see Sutton (1988)). Fitting an approximation through simulated observations can be seen as a learning procedure. Often the choice of architecture is coupled with the learning procedure. Other practical considerations involve the choice of step-sizes and the number and length of trajectories to ensure convergence to the solution of the least squares problem. API requires that the least squares problem is solved to deem the approximate policy evaluation step complete and allow for an update to the policy.

The fitted approximation $\tilde{V}^{\pi}(s|\boldsymbol{\theta})$ depends both on the set of representative states and the quality of the sample estimates at each state in this set. Iterative gradient methods will inherently give more weight to those states which are visited most frequently within the simulated trajectories, hence the least squares solution will provide better approximations for these states as opposed to those states less frequently visited. Furthermore, simulating trajectories under a specific policy π will drive the system towards certain parts of the state space upon which the approximation will be fitted. Consequently value function approximations at less visited states or in fact those states not visited at all may not be accurate which may cause errors in the approximate improvement step. This is a generic potential issue known as *inadequate exploration* which must be carefully considered when designing an ADP algorithm based on approximate policy iteration.

A variant of approximate policy iteration is known as *optimistic policy itera-*

tion. This is an ADP implementation of the modified policy iteration algorithm. Optimistic policy iteration follows the same approach as API, using an iterative gradient method within simulated trajectories. However, it does not wait until the least squares problem has been solved before implementing an approximate policy improvement step. Instead, the system makes a finite number of transitions under policy π, over which the parameter vector of the approximation is updated. The approximate policy improvement step is performed with respect to the current approximation and this process is repeated. API is an extreme case. Optimistic policy iteration is sometimes viewed as an *Actor-Critic* system. The actor uses policy π to make decisions within the system and the critic observes the outcomes of these decisions and maintains an evaluation of the policy. The critic provides feedback to the actor every so often and the actor subsequently attempts to improve his actions in light of this feedback. The choice of method for solving the least squares problem associated with a fixed policy controls how the critic views the system and the frequency of feedback provides a trade-off between being able to evaluate more policies at the expense of their true value functions not being approximated accurately.

Approximate value iteration is another approach to ADP based on the value iteration algorithm. This is also another extreme case of optimistic policy iteration. This method aims to directly approximate the optimal value functions $V^*(s)$ as opposed to the value functions of a fixed policy. Instead of simulating trajectories according to a fixed policy, they are simulated according to a greedy policy. In a given iteration n of the algorithm, suppose $\tilde{V}_n^*(s|\boldsymbol{\theta})$ is the current approximation of the optimal value function. Select a set S_n of representative states, perhaps by simulation under the current greedy policy. Using the expected total discounted reward criterion as an example, compute estimates of the optimal value functions at states within S_n by

$$v_{n+1}(s) = \max_{a \in \mathcal{A}_s} \left(R(s,a) + \alpha \sum_{s' \in \mathcal{S}} p(s'|s,a) \tilde{V}_n^*(s'|\boldsymbol{\theta}) \right), \quad \forall s \in S_n. \tag{2.4}$$

These estimates $v_{n+1}(s)$ are used to update the value function approximation to $\tilde{V}_{n+1}^*(s|\boldsymbol{\theta})$ by solving a least squares problem with respect to them. This process is then repeated for many iterations and finally actions can be determined at any state through greedy decision-making with respect to the final approximation. Approximate value iteration lends itself to iterative methods such as TD learning whereby the parameter vector of the value function approximation can be iteratively updated. Although this method can work well, it also suffers from the same potential issue of inadequate exploration as approximate policy iteration.

The methods discussed all rely upon an explicit model of the system which allow the expectation within (2.3) or (2.4) to be calculated. Alternatively this expectation may be intractable if the state space or action space or both are too large. A method which circumvents the need to compute an expectation is known as *Q-learning* (see Watkins (1989)). This method can be seen as a combination of approximate value iteration and simulation. This method uses an approximation architecture to represent the optimal Q-factor function $Q^*(s,a)$ which denotes the value of being in a particular state and taking a certain action. Value functions are defined as $V^*(s) = \max_a Q^*(s,a)$. In each iteration of Q-learning in state s an action is taken greedily with respect to the current approximation $\tilde{Q}^*(s,a|\boldsymbol{\theta})$ and a transition of the system to a new state s' is simulated under this greedy action. An estimate of the Q-factor associated with this state-action pair is calculated, taking the observed new state into account, and is then used to update the function approximation. This process is repeated. As a contrast, under approximate value iteration the new value function estimates are obtained before the transition with respect to an expectation over the current value function approximations. This subtle difference is what makes Q-learning suitable to the type of problems identified.

2.1.3 Game Theory

The interaction of multiple decision-making agents in a given situation has long been a problem of interest. Individual agents must not only consider their pursuit of some objective, but crucially the potential decisions of other agents and their impact on this pursuit. In a competitive environment, decision-making strategies of one agent towards attractive options may have undesirable outcomes if another agent engages in opposing decision-making strategies. With all agents analysing the situation in the same strategic fashion, agents can be thought of as playing a game against each other when making decisions. Game theory provides a language to formulate, structure, analyse, and understand such strategic scenarios.

The simplest strand of game theory considers strategic scenarios in which there are two decision-making agents engaged in competition or conflict with one another. Each agent is referred to as a player in a game, whereby each player has an objective. When the objectives of each player are completely opposed the game is said to be zero-sum and the situation is described as a *two-person zero-sum* game (TPZS). In contrast a scenario concerning n agents whose objectives are not completely opposed is described as an n-person general-sum game. A TPZS game in strategic form or normal form consists of players A and B, each with a nonempty set of strategies $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_m\}$ respectively. These strategy sets represent the set of choices available to each decision-maker. Elements of these sets are known as *pure strategies*. The game is played simultaneously such that A chooses $x \in X$ and B chooses $y \in Y$ which results in player A receiving a payoff $M(x, y)$. This payoff is a function of the strategies played by A and B and can be represented by a rectangular matrix. This is referred to as the payoff matrix M whose entries are given by $m_{ij} = M(x_i, y_j)$. This is sometimes called a *matrix game* wherein the rows of the matrix correspond to the pure strategies available to player A, columns correspond to the pure strategies of player B, and entries represent the payoff of a simultaneous row and column selection. Good coverage of the general theory on TPZS games is given by Washburn (2014).

In the game, the objective of player A is to maximise his payoff through his choice of strategy, whereas player B similarly wishes to minimise this payoff. Another interpretation is that player B views the payoff as his own loss and so wishes to minimise his loss. It is assumed that both players are *rational*, always choosing the best strategy to achieve their objective. It is also assumed that both players know the set of pure strategies available to the other player and the corresponding payoffs when chosen joint with their own pure strategies. However, due to simultaneous play each player does not actually know which strategy will be chosen by the opposite player. This is also equivalent to a situation in which one player announces his strategy first, but this is not revealed to the opponent. This kind of sequential play is equivalent to simultaneous play. Generally, if the announced strategy was known to the opponent this gives rise to an extensive form of the game. Two examples of simultaneous play matrix games with 3 pure strategies of each player are as follows:

$$M1 = \begin{pmatrix} 4 & 1 & 2 \\ 3 & 2 & 5 \\ 2 & 1 & 6 \end{pmatrix} \qquad M2 = \begin{pmatrix} 4 & 1 & 2 \\ 3 & 4 & 5 \\ 2 & 1 & 6 \end{pmatrix}$$

In matrix games, it is important to consider which strategy should be played if the strategy of the other player was known. In matrix game M1, if player A knew that player B would play y_1 then A should play x_1 to maximise the payoff. However, if B actually played y_2 then playing x_2 would be the best strategy to maximise payoff. Similarly, play x_3 for y_3. Conversely, B could conduct the same thought experiment to reason the best strategy to play if A chose certain strategies. In this game, by choosing x_2 player A maximises the payoff he can be guaranteed to receive. Similarly, choosing y_2 player B minimises the loss he can be guaranteed to incur. This is an example of the important concept of worst case analysis, each player does as well as they can under the assumption that the other player will know their strategy and maximise or minimise against this strategy. If either

player announced this strategy in advance, the other player cannot improve his position. This represents an equilibrium solution of the game known as a saddle point and the resulting payoff of 2 is known as the *value of the game* V. When saddle points exist in TPZS games, each player has an optimal pure strategy. A saddle point does not exist in matrix game M2 and there is no pure strategy which either maximises or minimises the guaranteed payoff for either player.

In general, we do not restrict the choice of players to elements within their sets of pure strategies. A mixed strategy for a player is a probability distribution over their set of pure strategies. A *mixed strategy* for player A is given by $\mathbf{p} = \{p_1, ..., p_n\}$ such that pure strategy x_i is selected with probability p_i and $\sum p_i = 1$. Similarly, a mixed strategy for player B is given by $\mathbf{q} = \{q_1, ..., q_m\}$ such that pure strategy y_j is selected with probability q_j and $\sum q_j = 1$. Any pure strategy can be written as a mixed strategy by specifying one non-zero element equal to one. Considering mixed strategies, the objective of each player is to maximise or minimise their expected payoff through their choice of mixed strategy. Suppose that player A uses mixed strategy $\mathbf{p}$ and player B uses mixed strategy $\mathbf{q}$, then the expected payoff to player A would be given by $K(\mathbf{p}, \mathbf{q}) = \sum_i \sum_j p_i m_{ij} q_j$. If the mixed strategy of player B was known to be a fixed $\mathbf{q}$ then player A would seek a mixed strategy $\mathbf{p}$ to maximise $K(\mathbf{p}, \mathbf{q})$. Alternatively, if the mixed strategy of player A was known to be a fixed $\mathbf{p}$ then player B would seek a mixed strategy $\mathbf{Q}$ to minimise $K(\mathbf{p}, \mathbf{q})$. These are known as best response strategies to particular mixed strategies of the opposing player. A *maximin* strategy for player A is one which maximises the expected payoff under the assumption of a best response strategy of player B. Likewise, a *minimax* strategy for player B is one which minimises the expected payoff under the assumption of a best response strategy of player A. The maximin and minimax strategies give the lower and upper values of the game respectively representing the maximum and minimum players A and B can guarantee the expected payoff to be no matter what the other player does. Conventionally, although maximin seems an appropriate term for the strategy of

player A, it is often also referred to as a minimax strategy since we could reverse the roles of players A and B and then player A would seek a minimax strategy. The terminology minimax is used to convey the idea that each player optimises their own expected payoff under the assumption of a best response strategy of the other player.

When the lower and upper values of the game are equal, the value of the game is said to exist and is equal to the common value V. The *Minimax Theorem* of von Neumann (1928) is one of the most fundamental results in game theory and states that every finite TPZS game has a value and both players have minimax strategies which are optimal. This theorem is powerful in that it guarantees the existence of a solution to every finite TPZS game in terms of mixed strategies. Moreover, it states that the optimal mixed strategies $\mathbf{p}^*$ and $\mathbf{q}^*$ are minimax strategies, which gives an indication of how to find these strategies. The solution $(\mathbf{p}^*, \mathbf{q}^*)$ is an equilibrium solution known as a *Nash equilibrium*. The concept of a Nash equilibrium is that assuming each player knows the strategies of the other players, no player can benefit by unilaterally changing strategy. This concept arose through the work of John Nash in studies of n-person general-sum games. Nash (1950) proved the existence of at least one Nash equilibrium for every finite game and the Minimax theorem is simply a special case of Nash's theorem.

The theory underlying *Linear Programming* (LP) is intertwined with the theory of TPZS games. During the inception of LP, in a meeting between George Dantzig and von Neumann (see Dantzig (2002)), the latter pointed out that the *Minimax Theorem is equivalent to the concept of duality*. A TPZS game can be transformed into a pair of primal-dual LP problems (see Dantzig (1951)). As such, there is an equivalence between the Minimax Theorem and the LP Duality theorem. This transformation of a TPZS game into an LP problem enables the use of LP methods to solve such games. In the general TPZS game described, the objective of player A is to find a mixed strategy $\mathbf{p}$ which maximises the minimum expected payoff when player B uses mixed strategy $\mathbf{q}$. It can be shown that for a given $\mathbf{p}$ the best

response strategy of B is some pure strategy j. Regardless of the pure strategy of B, player A is guaranteed an expected payoff of $\min_j \sum_i m_{ij}p_i$ and let this lower bound be v. The objective of A is then to maximise v and so an LP for player A can be formulated as follows (LP1):

$$\begin{aligned}
&\text{maximise} \quad v \\
&\text{subject to:} \\
&\sum_{i=1}^{n} m_{ij}p_i \geq v, \quad j = 1, ..., m. \\
&\sum_{i=1}^{n} p_i = 1 \\
&p_i \geq 0, \quad i = 1, ..., n
\end{aligned}$$

Here, the first set of constraints correspond to one constraint for each pure strategy of player B. Based on an analogous argument, from the perspective of player B, the dual LP reflects their objective. The dual LP can be formulated as follows (LP2):

$$\begin{aligned}
&\text{minimise} \quad u \\
&\text{subject to:} \\
&\sum_{j=1}^{m} m_{ij}q_j \leq u, \quad i = 1, ..., n. \\
&\sum_{j=1}^{m} q_j = 1 \\
&q_j \geq 0, \quad j = 1, ..., m
\end{aligned}$$

Suppose that LP1 has an optimal solution $\mathbf{p}^*$ with objective function v^*. Also suppose that LP2 has an optimal solution $\mathbf{q}^*$ with objective function u^*. Then by the LP Strong Duality theorem $v^* = u^*$, the optimal expected payoff to player A is equal to the optimal expected payoff to player B. This is precisely the result of the Minimax Theorem and so this shows its equivalence to the LP Duality theorem.

Thus by solving LP1 or LP2, whichever is easiest, the value of the game can be found.

It is widely recognised that transformation of a TPZS game into an LP problem is an efficient way to obtain optimal mixed strategies for each player, provided the game is not too large. In larger games with vast numbers of pure strategies for one or both players the dimensions of the resulting linear programs are too large to be solved. An alternative procedure for finding optimal mixed strategies in TPZS games called *Fictitious Play* (FP) was originally proposed by Brown (1949). Convergence of FP in finite TPZS games to the optimal mixed strategies and value was shown by Robinson (1951) and hence it is sometimes referred to as the Brown-Robinson method. However, this convergence is slow. Although FP is applicable in any finite TPZS game, it can be particularly appealing in larger games where LP methods no longer work.

The FP procedure is an iterative method in which both players are assumed to be *fictitiously playing* the game over a sequence of rounds in which they are able to learn and adapt their choices accordingly. The description of the procedure which follows is based on the article by Zafra (2010). Denote row i of M by M^i and column j by M^j and let k be the index count of the number of iterations. Define two vectors $\mathbf{v}(k) = (v_1(k), ..., v_n(k))$ and $\mathbf{u}(k) = (u_1(k), ..., u_m(k))$ as payoff vectors for players A and B respectively. For $k = 0$, set $\mathbf{v}(0) = (0, ..., 0)$ and $\mathbf{u}(0) = (0, ..., 0)$. For $k = 1$, player A plays pure strategy i arbitrarily and the payoff vector is updated as $\mathbf{u}(1) = \mathbf{u}(0) + M^i$. In response player B plays pure strategy j according to $j = \text{argmin}(u_1(1), ..., u_m(1))$. The other payoff vector is then updated as $\mathbf{v}(1) = \mathbf{v}(0) + M^j$. In subsequent iterations, player A plays pure strategy i according to $i(k) = \text{argmax}(v_1(k-1), ..., v_n(k-1))$ and player B responds according to $j(k) = \text{argmin}(u_1(k), ..., u_m(k))$. The payoff vectors are updated according to $\mathbf{v}(k) = \mathbf{v}(k-1) + M^{i(k)}$ and $\mathbf{u}(k) = \mathbf{u}(k-1) + M^{j(k)}$.

In each iteration of the FP algorithm, player A selects row i which *maximises the payoff against the empirical mixed strategy* so far of player B. Conversely, player

B does the opposite. Let $a_i(k)$ and $b_j(k)$ be the number of times pure strategies i and j are played in k iterations by player A and B respectively. Then after k iterations estimates of the mixed strategies of both players are given by the empirical distributions $\mathbf{p}(k) = (a_1(k)/k, ..., a_n(k)/k)$ and $\mathbf{q}(k) = (b_1(k)/k, ..., b_m(k)/k)$. The quantities $\max_i \mathbf{v}(k)/k$ and $\min_j \mathbf{u}(k)/k$ give upper and lower bounds respectively for the value of the game. As the number of iterations increase these bounds converge to the value of the game and the empirical mixed strategies converge to the optimal mixed strategies. In practice, the algorithm continues until an iteration limit has been reached or the difference in the bounds is sufficiently small. Even though this convergence can be slow, the FP algorithm can be useful in identifying good mixed strategies which guarantee an expected payoff close to the optimal value.

The discussion in this section has concerned classical results and concepts from game theory, most notably from the theory of TPZS games. Recent applications of game theory are to security and defence related problems wherein another problem such as a scheduling problem is contained within the context of a game. For example, Lin et al. (2014) and Alpern et al. (2011) use game theory to model and analyse a patroller defending a network against an attacker. Brown et al. (2006) study the defence of critical infrastructure from adversarial attacks using game theory. Further examples include the Bayesian Stackelberg games for security studied by Paruchuri et al. (2008) and the stream of related applications Pita et al. (2008), Tsai et al. (2009), Pita et al. (2011), and Shieh et al. (2012). These and further applications will be discussed more comprehensively in Section 2.3 under the label of defender-attacker problems.

2.2 Stochastic Scheduling with Abandonment

Scheduling problems concern the allocation of a resource within some system to a set of competing tasks over time to optimise some stated objective. The resource could refer to a machine on which jobs are processed, a server dealing with cus-

tomers, or a worker faced with a series of tasks. In each case there is a processing or service requirement of each job, customer, or task. When there are sources of randomness attached to elements of the system, the scheduling problem is stochastic. For example, the processing time of a job on a machine may not be known exactly, but instead only probabilistically over a range of possible values. In a queueing system, new customers may arrive adding to the workload of a server who does not know how long service of a given customer will take or indeed when these new arrivals will occur.

In the control of a stochastic system actions are applied at various time instances to adjust the behaviour of the system and directly impact upon some performance measure of interest. In a *clearing system*, which concerns the processing of a finite set of n jobs, actions relate to which job is selected for processing, for instance, each time a machine becomes available. If each job j incurs a holding cost c_j for each unit of time spent in the system and has expected processing time μ_j^{-1}, then a classic scheduling result asserts that with a single machine, processing jobs in decreasing order of the *index* $c_j\mu_j$ minimises the total expected holding cost. Such a scheduling policy is known as an index policy, which typically independently calculates a set of indices for competing jobs or job classes and uses an ordering of these to determine how processing should be prioritised. Index policies are common in many scheduling problems. In a single server multiclass queueing system in which customers arrive according to independent Poisson processes into one of k queues, incur holding costs c_j in queue j, and have expected service times μ_j^{-1}, the $c\mu$-rule is again optimal under nonpreemptive service for minimising the average holding cost per unit time, see Cox & Smith (1961). An extension of the $c\mu$-rule to a single server multiclass queueing system with Bernoulli feedback is attributed to Klimov in his seminal work (see Klimov (1974) and Klimov (1978)) which establishes an index policy for minimising the long-run average holding cost in the system.

Related to stochastic scheduling problems, which are the main focus in this

thesis, are those of *routing control* and *admission control* in queueing systems. Routing control problems invariably concern which queue an arriving customer should be sent to upon arrival into a queueing system whereas admission control problems concern whether arriving customers should be permitted entry into the system. In some classic control problems in these areas we often see that optimal policies are quite simple. For example, in an admission control problem of deciding whether an arriving customer should be admitted into a G/M/1 queue, Stidham (1985) showed that a threshold policy maximises the total expected net benefit over an infinite horizon. That is, reject customers who arrive when the queue length exceeds some threshold N. In a routing control problem of routing customers to one of many identical queues, the join-the-shortest-queue policy maximises the discounted number of jobs to complete service in any specified time interval (see Winston (1977)). Ephremides et al. (1980) also showed that it minimised the expected total time to complete the service of all jobs that arrived before some fixed time. The literature on queueing control problems, in fact in a wider sense the design, analysis, and control of queueing systems is vast. We refer to the excellent summaries given by Walrand (1988) and Stidham (2002) for further reading.

One important stream of work in the field of stochastic scheduling is *bandit models*. The multi-armed bandit problem is concerned with the optimal sequential allocation of resources between projects which require them and only yield rewards and change state when active. It is a classic problem of stochastic scheduling which has at its heart the trade-off between exploitation (obtaining the highest immediate reward) and exploration (learning about the system). The problem of maximising the expected total discounted reward was first solved and published by Gittins & Jones (1974). The optimal policy for this problem takes the form of an index policy which chooses at each time the project with maximal Gittins index, each computed as a function of its current state. The major success of this result is that it reduces an n-dimensional problem into a collection of n one-dimensional problems which can be solved. One extension of the multi-armed bandit problem is known as the

restless bandit problem in which projects evolve even when a control is not applied to it and further evolve differently when the control is applied. Control of queueing systems often fall into this class of problems. The restless bandit problem has been shown to be computationally intractable (see Papadimitriou & Tsitsiklis (1999)). See Whittle (1988) for a discussion of this problem in which he proposes a class of index heuristics. Unified coverage of bandit problems, as well as their relation to the control of queueing systems is given by Gittins et al. (2011).

There are many real-world scenarios in which tasks, jobs, customers, or items do not have limitless availability for processing or service. For example, in a military setting, targets may move out of the range of defensive forces and become unavailable to attack. In a call centre, customers placed on hold may hang up if made to wait too long. In a medical emergency, the condition of patients requiring treatment may deteriorate the longer they wait and indeed they may die if treatment is not provided in time. In a healthcare setting, blood passing through blood screening procedures with the aim of stocking a blood bank may become unusable if it not passed within a certain timeframe. In telecommunications, transmitted messages or data may be lost if it is not sent or received in time. In a surveillance setting, suspects subject to screening by a security team may leave an area before screening is completed. It is the latter application which is the focus of this thesis. In all applications this limited availability is referred to as impatience or abandonment. It is a key feature of many stochastic scheduling problems, having a major impact on the control strategies applied. Indeed there is now an extensive literature concerning optimal scheduling in the presence of abandonment.

Abandonment can be modelled as a deadline applicable to each job or customer. In the literature these deadlines are either applicable to the beginning of service or the end of service. The first case means that abandonments will not occur once service has commenced, whereas in the second case they can also occur during service. It is the second variation which we consider within the models in this thesis, namely abandonments may occur during service.

There is a large collection of work which must be distinguished from the modelling approach adopted within this thesis. In this work the notion of limited availability of a collection of jobs is modelled as a set of *hard deadlines* known to the controller of the system. These deadlines are either announced upon arrival into the system in the case of queueing systems or announced at the beginning of the scheduling horizon in the case of a clearing system. The system controller takes these deadlines into account when considering scheduling policies, often as the objective reflects some measure of missed deadlines or tardiness. For important examples, see Liu & Layland (1973), Glazebrook (1983), Panwar et al. (1988), Bhattacharya & Ephremides (1989), Bhattacharya & Ephremides (1991), De et al. (1991), Jiang et al. (1996), Doytchinov et al. (2001), Jang & Klein (2002), and Van Mieghem (2003).

From these works, Panwar et al. (1988) study the transmission of voice packets over a packet-switched network. They model the problem using a single server queue in which service times of customers are i.i.d random variables and service is nonpreemptive. Each customer upon arrival announces a deadline which is randomly drawn from a general probability distribution, independent of other customer deadlines. This gives rise to an extinction time for each customer whereby if their service has not begun by this extinction time this is equivalent to a voice-packet being deemed useless. They showed that the shortest time to extinction (STE) policy maximises the long-run fraction of successful customers if the server is prohibited from idling. Each time the server is available for service this policy schedules the customer closest to its extinction time. When the server is allowed to idle, they also show that when it exists the optimal policy belongs to the class of policies which are shortest time to extinction with unforced idling (STEI). This problem is also studied by Bhattacharya & Ephremides (1989), wherein they show that STE minimises the expected number of lost customers over any time interval within the class of nonpreemptive and non-idling policies under exponential service. They also show that the optimal policy with permitted idling belongs to the class

of STEI policies when available. The STE policy is also shown to be optimal in the case of preemptive service. In this case idling is never preferable. In a similar problem, Bhattacharya & Ephremides (1991) also show that a form of the shortest time to extinction policy minimises the average tardiness per customer in a two-queue problem where customers remain in the queue and are scheduled beyond their deadline, but incur a penalty for doing so.

In contrast to the case of known deadlines of jobs or customers, there are many situations in which availability is unknown prior to service. Work exhibiting this feature of unknown and *random deadlines* until abandonment is of particular interest in this thesis given the *lifetimes of suspects in surveillance applications are generally unknown*. In the literature, typically deadlines of jobs are assumed to be independent and identically distributed random variables, with this distribution known by the system controller, and exact realisations of which are not known until the deadline passes. There are three examples of this feature in the papers by Pinedo (1983), Boxma & Forst (1986), and Emmons & Pinedo (1990) which study clearing systems. Pinedo (1983) considered a problem in which a collection of jobs are to be processed on a single machine. Processing times are i.i.d exponential random variables with rate μ_i, deadlines are i.i.d random variables, and processing is nonpreemptive. Each job has a weight w_i which can be interpreted as a penalty paid for completion after its deadline. Static list scheduling policies are considered, which means that all jobs are ordered at time zero and scheduled according to this order. The order is never changed which means all jobs, including those whose deadline has passed, must be processed. Processing jobs in decreasing order of $w_i\mu_i$ was shown to minimise the expected weighted number of tardy jobs. Boxma & Forst (1986) studied further variations of this problem. Emmons & Pinedo (1990) extend this problem to multiple servers and also consider dynamic policies which can modify scheduling decisions in light of new information. Both papers identify optimal dynamic or list policies subject to certain conditions. An example from a queueing system is found in Bhattacharya & Ephremides (1989). An extension to

the known deadline problem in the same paper discussed previously, for i.i.d service times and i.i.d random unknown deadlines with increasing failure rate, the earliest arrival (EA) policy minimises the expected number of lost customers over any time interval when deadlines apply to the beginning of service. The same result holds when deadlines apply to the end of service provided the common distribution of service times is exponential.

In the clearing models studied by Pinedo (1983) and Boxma & Forst (1986) jobs did not depart the system upon expiration of their deadlines, but instead were still processed with a penalty for tardiness. This clearing model is further studied by Glazebrook et al. (2004) with the exception that jobs are lost from the system as soon as their lifetime expires. A collection of n jobs, present at time zero, are to be processed nonpreemptively on a single machine. The processing time of each job is an arbitrary random variable and jobs remain in the system for an exponentially distributed period of time unless processing begins first, meaning jobs do not abandon during service. A reward is received upon completion of a job and the objective is to find a scheduling policy which maximises the total expected reward earned. The authors develop a permutation policy which orders the jobs according to decreasing values of an index which favours large reward-rate losses and small mean processing times. This index is given by $R_j\theta_j/E(X_j)$, where R_j, θ_j, and X_j represent the reward, abandonment rate, and processing time of job j respectively. This policy is shown to be asymptotically optimal as abandonment rates all go to zero. Motivated by the use of limited medical resources in a mass casualty emergency, Argon et al. (2008) also study this problem, assuming jobs are placed into a number of priority classes after a triage assessment of patients is conducted. With the objective of maximising the number of jobs taken into service, if lifetimes admit the same hazard rate ordering as a likelihood ratio ordering on service times, the optimal policy gives priority to the job with shortest lifetime and service time. Such jobs are referred to as time-critical. For the case of exponential lifetimes and service times in which jobs with shorter lifetimes have longer service times, the

authors note that the optimal policy has a more complex structure. They develop a state-dependent heuristic known as the triangular heuristic which at each point prioritises the job class with the smallest mean number of abandonments during the next service.

One further contribution to the clearing model discussed is by Li & Glazebrook (2010). In this paper the authors note that the static priority policy proposed in Glazebrook et al. (2004) and the heuristic policy proposed in Argon et al. (2008) both perform well in certain parameter regimes, but are not robust, performing poorly in other parameter regimes. The authors take a somewhat different approach in their design of a heuristic policy. Allowing for arbitrary i.i.d service times and lifetimes within job classes, they use the concept of policy improvement from the field of stochastic dynamic programming in order to define a policy which improves the static priority policy in Glazebrook et al. (2004). Such a method quickly becomes computationally intractable with many job classes and certain distributional assumptions. Instead they adopt an approximate policy improvement step to define a heuristic policy, using a fluid model to approximate necessary quantities. The resulting heuristic policy is shown to be both robust and outperform the competitor policies in all parameter regimes over an extensive set of numerical experiments. One final application of scheduling in a clearing system with abandonment worth considering is by Glazebrook & Mitchell (2002). In this paper, jobs in the system improve while being processed and deteriorate while not being processed, hence affecting the reward yielding characteristics of job completion. With an objective of maximising the total discounted reward from job completions, this problem can be modelled as a restless bandit problem. A variation of the model covers the situation where jobs abandon the system once they reach a certain state. The authors show that the problem is indexable in the sense of Whittle and develop an index policy with strong performance. It must be highlighted here that the models studied in Glazebrook et al. (2004), Argon et al. (2008), and Li & Glazebrook (2010) are all similar to the model which we study in

Chapters 3 and 4 of this thesis. However, one key difference is that these models consider a finite set of jobs to be processed as opposed to allowing for the arrival of jobs over time. Hence we will now restrict attention solely to queueing systems with abandonment.

There have been a number of contributions in the areas of admission and routing control problems which feature customer abandonments. Ward & Kumar (2008) study admission control in a G/G/1 queue in which customers have i.i.d exponential patience times until the end of service. With the objective of minimising the long-run discounted cost from penalties for refusing to admit arriving customers plus costs from customer abandonments, a barrier policy is shown to be asymptotically optimal in an appropriate heavy traffic regime. Movaghar (1998) and Movaghar (2005) study two variations of a routing problem which concerns the routing of a stream of Poisson arrivals with exponential service requirements and random deadlines to a collection of identical queues. With exponentially distributed lifetimes, the policy of joining the shortest non-full queue minimises the number of customers lost by any time. With generally distributed lifetimes a condition is provided under which the policy of joining the shortest queue minimises the number of lost customers during any finite period in the long-run. Glazebrook et al. (2009) study a problem of both admission and routing control in which a Poisson arrival stream is either refused entry to the system or routed to one of a number of heterogeneous service stations. The model incorporates rewards for service completions, penalties for denied admissions, and costs for customer abandonments. An approach founded on the restless bandit problem is used to derive an index policy for maximising the net of rewards minus penalties and costs. This index policy is shown to be optimal in a number of asymptotic regimes.

In the context of the vast literature related to call centre applications, Bassamboo et al. (2005) study a problem of admission control of impatient customers to a collection of customer classes whilst simultaneously controlling the allocation of a group of identical server pools to the set of customer classes. Arrival

rates are permitted to vary and the skills of servers in each pool determine which customer classes they are able to serve in. There are penalties incurred for each customer refused admission to the system, penalties for abandonment, and holding costs whilst waiting for service. The objective is to find dynamic controls which minimise the expected total cost incurred over a finite horizon. In an asymptotic parameter regime, an approximating stochastic fluid model is solved by means of a linear program and shown to give an asymptotically optimal control in the original problem. Similar methods and results are found in the subsequent paper by Bassamboo et al. (2006) which replaces the admission control problem with a staffing problem relating to how many servers should be employed within each server pool at the expense of associated personnel costs. Jouini et al. (2010) study an interesting problem of a call centre with two queues and many parallel servers populated by two types of impatient customer, premium and regular. The objective is to develop scheduling policies which satisfy a target ratio constraint on the abandonment probabilities of premium customers to regular ones. Two parametric families of scheduling policies are developed with respect to this aim, the first being based on routing of customers under a fixed server allocation policy, and the second based on allocation of servers to waiting customers under a fixed routing policy.

An example application related to Homeland Security is presented by Lin et al. (2009). In this paper, a single server queue with abandonment is used to model an antiterrorist surveillance system in which suspects arriving into a public area are subject to screening by a security team. There are two types of customer, each with their own random variable representing their lifetime in the system during which they are available for service. One type represents terrorist suspects which are the sole interest of the security team, whereas the other type are civilians and are not of interest. Each arriving customer is independently a terrorist with some probability and the server receives a reward for serving this customer type before their lifetime expires. The objective of the server is to find a scheduling

policy which maximises the expected reward received. The authors show that the optimal policy in the case of exponentially distributed lifetimes is to serve first come first served (FCFS) or last come first served (LCFS) dependent on the ratio of the mean lifetime of the terrorists compared to the civilians. Gaver et al. (2006) consider a multiclass queueing system in a military setting. Customers arrive at random into one of a number of customer classes with i.i.d random service requirements and i.i.d exponential lifetimes. These customers represent enemy targets who may move out of range of a defensive force. Service completions yield rewards and the objective is to maximise the long-run average reward earned by the server. Versions of the model are presented which allow for the cases where both abandonments and/or service completions are not observed by the server during service. The main challenge in this paper is to decide both which customer to serve next and how much processing time to allocate to that customer. For the case of a single customer class the authors propose a heuristic which allocates a constant processing time to each customer. For the case of multiple customer classes the authors propose three heuristics, two of which extend the constant processing time idea, and one which myopically maximises the ratio between the expected reward received and expected reward lost during the next service. Glazebrook & Punton (2008) further study the problem of allocating processing times in a single customer class with the objective of maximising the throughput of the system. The authors develop two heuristic dynamic scheduling policies which allocate processing times based on the state of the system using DP policy improvement. These heuristics are shown to be near-optimal and improve upon the static policy in Gaver et al. (2006) in a number of numerical experiments.

There are a number of papers which are of direct relevance to this thesis as they study either the same model or minor variations of the scheduling model featured in Chapters 3 and 4. Glazebrook et al. (2004) study the same model in which a single server provides preemptive service to customers across k classes. Customers arrive according to independent Poisson processes with rate λ_j with class dependent

exponentially distributed service requirements and lifetimes, with corresponding rates μ_j and θ_j respectively. If a customer is served before his lifetime expires, a reward R_j is received, otherwise the customer departs the system both before or during service. The objective is to find a service policy which *maximises the long-run average reward earned per unit time*. In principle it is possible to find the optimal policy from the methods of DP, but in practice this is unrealistic for problems with many customer classes. The authors develop a *heuristic policy* via a two stage process. The first stage analyses a policy which allocates a fixed service effort to each class at all times, and computes the optimal allocation, and the second stage second stage performs an exact policy improvement step. The parametric optimisation involved at the first stage can pose computational challenges, especially for large k, and was only implemented for cases with $k = 2$. However, the approach does have the advantage that the structure of the resulting heuristic policy is simple to recover.

A variation of this model with many servers is studied by Atar et al. (2010) and further in Atar et al. (2011). In addition, as opposed to rewards for service completion, each customer incurs a class-dependent holding cost c_j for each unit of time spent waiting in the queue and the objective is to minimise the long-run average holding cost. In their model, customers cannot abandon during service and holding costs do not apply during service. This stands in contrast to the model in this thesis and are features of their model. The authors use a fluid scaling in an overload setting and show that the $c\mu/\theta$ *rule* is asymptotically fluid optimal under both preemptive and nonpreemptive service. Here overload refers to the workload defined as $\rho = \sum_j \lambda_j/\mu_j$ being greater than one. This rule calculates the index $c_j\mu_j/\theta_j$ for each queue and prioritises customers according to decreasing values of this index. This index rule is remarkably simple and independent of the arrival rates, resembling and modifying the $c\mu$ *rule* which is optimal in the case of no abandonments. The index rule easily extends to the case where additional penalties are incurred for abandonments. Ayesta et al. (2011) study a single server

version of this model in discrete time with no arrivals, but allow the customer in service to contribute a holding cost. The authors find the optimal solution to a relaxed version of the problem via Lagrangian methods which yields an index policy referred to as the *AJN rule*. The AJN rule has a similar form to the $c\mu/\theta$ rule and is used as a heuristic for the continuous-time problem with arrivals, exhibiting strong performance numerically in overload scenarios.

A version of the model with two customer classes was studied by Down et al. (2011). Abandonments may occur during service which is preemptive and provided at rate 1. Two problems were studied, firstly maximising average rewards received from service completions, and secondly minimising average net holding costs plus penalties from abandonment. In both cases, a priority policy is shown to be optimal if *parameters of the system are agreeably ordered* in an underloaded regime. However, more generally numerical examples illustrate that optimal policies can exhibit a threshold structure, leading to the poor performance of priority policies. Verloop (2014) studies a k class multiserver version of the model as a special case of a more general restless bandit model. In addition to the model in Atar et al. (2010), a more general framework is presented in which customers can abandon during service at different class dependent rates and also incur different class dependent holding costs and abandonment penalties whilst in service. An index policy is derived which is shown via fluid-scaling techniques and LP results to be asymptotically optimal in overload for minimising the long-run average cost. In special cases this *index policy coincides with both the $c\mu/\theta$ rule and the AJN rule*. The latter coincidence provides a proof of asymptotic optimality of the AJN-rule in overload for a continuous-time system with arrivals. A single server version of the full holding cost and abandonment penalty model in Verloop (2014) with preemptive service is studied by Larrañaga et al. (2014). In terms of the restless bandit literature, the problem is shown to be indexable and the corresponding *Whittle index* (see Whittle (1988)) is derived, allowing for convex holding costs. In the case of linear holding costs the Whittle index policy is equivalent to the in-

dex policy derived by Verloop (2014) which subsequently shows that the $c\mu/\theta$ rule and the AJN rule are themselves Whittle index policies which are asymptotically optimal in overload.

In an earlier paper, Larrañaga et al. (2013) study the same problem through a fluid control problem. The optimal solution in an overload setting coincides with the same index policy derived in Larrañaga et al. (2014). In underload where $\rho < 1$, the fluid control problem is solved optimally for the case of two customer classes. The *optimal solution exhibits a switching curve structure* whereby service is decided through the $c\mu$ rule when the system is closer to being empty, whereas service is decided through the $c\mu/\theta$ rule when the system has many customers. Numerical evidence suggests that the optimal policy in the original stochastic problem also exhibits this structure and the switching curve is well approximated by the fluid model. This motivates a heuristic policy based on this observation for systems with more customer classes. There are further contributions in the papers by Harrison & Zeevi (2004), Ata & Tongarlak (2013), and Kim & Ward (2012) which study *approximating Brownian control models* in different heavy traffic limits. The latter article considers general arrival, service, and abandonment processes. These papers show that optimal policies in the Brownian control problems are state dependent and assign service based on the changing workload in the system. This is opposed to static priority rules such as the $c\mu$ rule and index rule derived by Verloop (2014) which use fixed indices at all time points to assign service and the only state information used is knowledge of whether a queue is empty.

In the model which will be the focus of Chapters 3 and 4 of this thesis, the literature suggests that although the *$c\mu/\theta$ rule will perform well in highly loaded systems* which will contain many customers in steady state, it is not guaranteed to perform well across all loads. Alternatively, *comparatively little is known about lighter loaded systems*. The observed structure of the optimal policy suggests good performance of the $c\mu$ rule, although there are no results which guarantee this apart from in the case of no abandonments. Furthermore, there is potential value

in developing service policies which are not simply priority policies, induce the potential for state dependent decisions, and work well in all parameter regimes. The model and analysis in Chapters 3 and 4 of this thesis will shed further light on these observations.

2.3 Defender-Attacker Problems

There are many scenarios in which defensive agents are faced with adversarial attacking agents. Patrollers, security forces, or the government may all be seen as defensive agents whereas terrorists, smugglers, and criminals are examples of attacking agents, each within a given context. The term we adopt here to refer to such scenarios is defender-attacker problems. This is an umbrella term designed to refer to scenarios in which there are two decision-making agents with opposed objectives, one looking to protect or defend and the other looking to attack, damage, evade, or destroy. The context in which different scenarios arise describes the domain in which these agents operate and over which there is mutual interest. The domain can often lead to alternative and sometimes more informative descriptions such as security games, interdiction games, patrol games, and search games and can be found across a wide range of fields such as computer science, artificial intelligence, machine learning, operations research, and game theory.

In many systems, the decisions required from a defensive perspective occur under the *uncertainty of future events* and the *effect of making certain decisions*. This is of particular importance when uncertainty characterises some form of risk, threat, or vulnerability in the system which the defender wishes to protect against. In order to best account for the uncertainty of events outside his control, a defensive agent must also consider how such uncertainty occurs and hence the *type of threat being faced*. For example, a defensive agent seeking to protect infrastructure from the effects of an accident or natural disaster is faced with a random event. Alternatively, when protecting infrastructure from an attack, the defensive agent is faced with an intentional event. In these two situations Bier (2006) argues that

"protecting against intentional attacks is fundamentally different from protecting against accidents or acts of nature". Furthermore, this paper discusses the fact that intentional attacks are the product of intelligent and adaptable adversaries who take defensive efforts into account when planning their attacks. The role of game theory is highlighted as an essential tool for modelling and analysing such strategic threats. Unsurprisingly then, *game theoretic ideas form a critical part of most defender-attacker problems* in the literature. The paper by Golany et al. (2009) shares this view, stating that decision-making is somewhat different when faced with probabilistic and strategic risk, where the latter type can be addressed using game theory. The authors illustrate this through a case study of homeland security grant allocations in the United States. They consider the problem of a decision-maker allocating a fixed budget over many sites to reduce the expected damage from uncertain events when this uncertainty is either probabilistic or strategic. No allocation to a site results in maximal probability of an undesired event, whereas increasing an allocation reduces this. The probabilistic and strategic risk problems are shown to have similar formulations, but a stark difference in the optimal allocations made. Under probabilistic risk, the decision-maker must prioritise his budget and allocate in the highest impact sites. Under strategic risk it is best to spread the budget among sites, decreasing the damage levels of the most vulnerable sites first.

The vulnerability of critical infrastructure in the United States is considered in the paper by Brown et al. (2006). Given the scale of investment in critical infrastructure systems and their importance, any disruption to a system can have a substantial impact. The authors state that "random component failures offer a poor paradigm in a world with intelligent adversaries", which is consistent with the view expressed by both Bier (2006) and Golany et al. (2009). Moreover, the authors discuss the ability of terrorist organisations to collect relevant information about a range of features and use this to plan devastating attacks on an infrastructure system. The observation is made that most civilian infrastructure

is "soft" and so vulnerable. Golany et al. (2009) share this observation, referring to exposed homeland security assets whereby *terrorists can attack in amongst the public*, move around like ordinary citizens, and conduct valuable surveillance of defensive operations which allows them to assess with high accuracy their own chances of causing damage. Brown et al. (2006) state the importance of *worst case analysis* in defender-attack problems. Given the uncertainty over the exact knowledge and capabilities of attackers it is prudent to assume the attacker will act to maximise damage and has all the information and capabilities needed to do so.

This paper considers three variations of the general defender-attacker problem. Defender-attacker problems or conversely attacker-defender problems are formulated as bilevel programs which themselves are a type of *Stackelberg or sequential game*. In a Stackelberg game, one agent makes the first decision and hence is referred to as the leader and the other agent makes the second decision and is referred to as the follower. Having observed the first decision, the follower optimally decides upon a *best response*. Importantly, the leader possesses a perfect model which anticipates how the follower will best respond to his decision and consequently make his decision to his own best advantage. This setup is consistent with the observation that an attacker is able to conduct surveillance of defensive decisions before attacking. A defender-attacker-defender problem is a three stage sequential game in which, for example, the defender makes an initial decision which precedes the final two stages. The paper gives details of how to model these problems and illustrates their use through three examples: protection of the US Strategic Petroleum Reserve, border control on the US/Mexico border, and protection of an electric power grid. On reflection the authors highlight the potential significance of deception and secrecy as a tool for successful defence.

A three stage sequential game between a defender and an attacker is studied by Carlyle et al. (2011). A defensive planner wishes to design parallel service channels knowing that an intelligent attacker will attempt to maximise disruption

in the system. For example, should a community have one large hospital with economies of scale in operating costs or a few smaller hospitals which may be more resilient to disruption. The problem models the service channels as parallel M/M/1 queues and the ultimate decision for the defender is how to allocate a fixed service capacity to these queues. After this allocation is made an attacker disrupts the system by reducing these service capacities, then finally the defender routes jobs through the system under the reduced capacities to maximise throughput of jobs. When attacks are incremental in nature, meaning that service channels are not fully destroyed, it is optimal to concentrate service capacity in one large server to exploit associated economies of scale. This would be equivalent to planning a single large hospital. When attacks are more destructive, such that an attack can destroy an entire service channel, planning is more difficult. Under certain conditions the attacker wishes to concentrate attacks whereby the defender responds by completely distributing capacity, for example by planning many smaller hospitals. Under other conditions the attacker wishes to distribute attacks and so the defender responds by concentrating capacity in one large server. Here a detailed understanding of attacker capabilities is imperative for optimising defence plans.

An interesting example of a defender-attacker problem is due to Wein & Baveja (2005) who studied the potential security threat of visitors entering the United States. Under the US-VISIT Program, visitors are fingerprinted when they enter the United States and matched and scored against a watchlist containing millions of fingerprints of known criminals and suspected terrorists. When a fingerprint closely matches with one on the watchlist, the visitor is further investigated. The performance of the biometric matching system is known to be dependent upon the image quality of the fingerprint. The authors identify an opportunity for terrorists to exploit this fact by sending people with either worn fingers or deliberately altered fingers in order to have low image quality fingerprints and more chance of evading a match. The problem is formulated as a Stackelberg game in which the biometric identification strategy is first set to maximise detection probability sub-

ject to a constraint on processing times of visitors and the terrorist then chooses his image quality to minimise his detection probability. The results suggest alternative strategies for the defender which lead to large increases in detection probability when compared to the existing strategy in use under the assumption of a strategic attacker. Another interesting defender-attacker problem is studied in Brown et al. (2009). In this paper a proliferator seeks to develop nuclear weapons as quickly as possible and an interdictor wishes to maximally delay this project. In some sense the roles of attacker and defender are reversed in this problem since the interdictor corresponds to a nation attempting to delay the project through seemingly attack like actions. The problem is formulated as a Stackelberg game since the proliferator is able to observe the interdiction plan and adjust his own plan accordingly, hence the interdictor must take this into account. An algorithm based on Benders decomposition (see Benders (1962)) is presented to solve a detailed model which incorporates case study data.

It is evident from the literature previously discussed that Stackelberg games are commonly adopted to model defender-attacker problems since they can account for the ability of an attacker to gather information about defensive strategies before initiating their attack. Stackelberg games have been used extensively to model defender-attacker problems in security domains and consequently the term *security game* is often used here. In security domains, defensive measures often take the form of a schedule which is repeated over time. For example, a police patrol around an urban area looking for a criminal may take a specified route around a set of locations every few hours and repeat this schedule. To guard against an intelligent attacker observing and hence exploiting this deterministic behaviour, the patroller may decide to *randomise his route*. However, Ordónez et al. (2013) discuss the difficulty in effectively randomising, citing the potential for *humans to fall into predictable patterns* and so the need for game theoretic approaches to finding good randomisation strategies.

The objective in Stackelberg games is to find the optimal mixed strategy for

the leader to commit to given an optimal response of the follower. Conitzer & Sandholm (2006) state that in the special case of zero-sum games, the *optimal mixed strategy to commit to is a minimax strategy.* This means that zero-sum Stackelberg games can be considered as standard simultaneous move games and hence solved through LP methods. This is no longer true for general-sum games. In the extension to general-sum games, the authors show that LP can also be used to compute the optimal mixed strategy to commit to. This amounts to solving multiple linear programs in which the leader maximises his payoff under the constraint of a pure strategy response of the follower. A further extension is the concept of a *Bayesian Stackelberg game* in which there are multiple leader and follower types, where the leader has incomplete information regarding the follower type and subsequent payoffs in the game. Paruchuri et al. (2008) study Bayesian Stackelberg games for security in which there is a single defender type and multiple attacker types. Pita et al. (2008) illustrate that defender-attacker problems in security domains are well modelled as Bayesian Stackelberg games, owing to their flexibility in capturing multiple attacker types and the ability to weigh different targets according to their significance. Whereas the method of Conitzer & Sandholm (2006) is computationally intractable for these security games, Paruchuri et al. (2008) develop an efficient exact algorithm known as DOBSS (Decomposed Optimal Bayesian Stackelberg Solver) for finding the optimal mixed strategy to commit to. This method involves formulating and solving a single mixed integer linear program.

There have been some notable examples of the Bayesian Stackelberg game approach to defender-attacker problems in real-world security domains. Pita et al. (2008) developed a software assistant known as ARMOR which has been deployed at LAX airport in Los Angeles since 2007 to randomise checkpoints on inbound roads to the airport and canine patrol routes within the airport terminals. These problems are modelled alongside security experts as Bayesian Stackelberg games and subsequently solved using the DOBSS algorithm. The system enumerates

every possible pure strategy in the game for both defender and attacker, which for the defender corresponds to how resources can be allocated among targets. *Subjective assessments* of the rewards to each player of each pure strategy combination and attacker type embeds the ability to *weigh the significance* of each event. These rewards are not equal in magnitude and so the game is general-sum. Tsai et al. (2009) developed a software scheduling assistant known as IRIS for the Federal Air Marshal Service to randomise air marshal schedules aboard United States commercial flights. In this security domain, the set of attacker targets corresponds to tens of thousands of commercial flights each day and the set of defender strategies is how to schedule a fixed number of air marshals over all flights. Alongside scheduling constraints specific to the transportation network domain, this scale of problem cannot be addressed using the approach in ARMOR. In a closely related paper, Kiekintveld et al. (2009) study large Bayesian Stackelberg games motivated by the same security domain. The key idea is to consider a *compact form* of the game which subsequently *scales well to large problems*. The authors develop efficient algorithms for these compact form games, exploiting payoff structure and incorporating realistic scheduling constraints. The IRIS system uses this modelling approach together with a preference elicitation system for experts to determine the general-sum payoffs in the game. The algorithms from Kiekintveld et al. (2009) solve the resulting game efficiently. Pita et al. (2011) developed an application called GUARDS for the United States Transportation Security Administration to assist in randomising security resource allocation to tasks in the protection of over 400 airports. This problem considered heterogeneous security activities, for example perimeter patrols or baggage screening, as well as heterogeneous attacker modes, for example an outside attack or a suitcase bomb. The approach involved detailed modelling of strategies and payoffs using expert knowledge as well as a compact form representation which enabled solution through the DOBSS algorithm. Shieh et al. (2012) developed a system called PROTECT for scheduling randomised patrols for the United States Coast Guard in the port of Boston. This

problem divides the port into a number of patrol areas over which a patroller must decide both an order to visit and a set of activities to perform over this route, ensuring the route starts and ends with the same area. The authors determine a method for a compact form representation for increased computational efficiency. A key feature of this problem is the assumption of an attacker who is not perfectly rational in his choices, for which an optimal randomised patrol schedule is computed using the work of Yang et al. (2012).

A common feature of these real-world security systems (ARMOR, IRIS, GUARDS, PROTECT) is the considerable amount of *calibration* required to formulate a relevant Bayesian Stackelberg game which accurately represents the respective security domain. Critical to the success of those systems was the integration of *expert knowledge* of each domain to reduce the size of each problem and specify the required general-sum payoffs in the game. In contrast, the defensive surveillance scenarios we consider within Chapters 5 and 6 of this thesis involve an *infinite number of unique pure strategies* available to the defender, each with its own unique impact upon targets in the system. Consequently there is no obvious possibility of representing the problems there in a compact form to ease the computational challenge. Furthermore, the complex nature of both the domain and the defensive pure strategies is such that subjective assessment of payoff values is unlikely to be possible. Subsequently, payoff values under the joint pure strategies of defender and attacker are given as *probabilities which must be either computed exactly or accurately estimated* within a queueing system model of the domain of interest. Even for a finite set of defender pure strategies this can be time consuming. Although the models in this thesis allow for the ability to weigh targets differently dependent on their significance, the valuation of these targets is common to both the defender and attacker and hence we consider zero-sum games. Furthermore, in the surveillance scenarios considered in this thesis, the attacker is unable to observe the defensive strategies deployed, hence we analyse them as simultaneous move games.

An example of a zero-sum defender-attacker problem with simultaneous moves is studied by McMahan et al. (2003). This paper considers planning a route for a robot to navigate a known environment to reach a goal location whilst avoiding detection by sensors placed within the environment by an adversary. A MDP model describes the movement of the robot through the environment, whilst a specific placement of sensors imposes a specific cost vector upon the MDP. For a known set of placements, and so known cost vector, the planner (defender) can determine a route which minimises the observability through MDP methods. However, the adversary chooses a sensor placement to maximise the observability. The problem is formulated as a zero-sum game in which the pure strategies of the adversary correspond to a finite set of cost vectors and the pure strategies of the adversary correspond to a finite set of deterministic policies for the MDP. The authors develop an *iterative algorithm based on Benders decomposition* which utilises what they term an *Oracle*. Given any cost vector, the Oracle provides an optimal policy with respect to that cost vector. The iterative algorithm finds an optimal mixed strategy for the adversary given a subset of policies available to the planner and then uses the Oracle to provide a best response policy for the planner to this adversarial mixed strategy and adds this new policy to the policy set. The algorithm proceeds in this way, trading best responses back and forth, until the Oracle provides a policy the planner already has. The algorithm converges to the optimal mixed strategies for each player due to the corresponding convergence of Benders algorithm. This is referred to as a single Oracle algorithm since only the planner possesses an Oracle. In the case where the set of cost vectors available to the adversary is large, the authors develop and prove convergence of an analogous double Oracle algorithm in which both players possess an Oracle. Another example of use of a double Oracle algorithm is found in Tsai et al. (2014) in a problem in which a defensive force attempts to prevent the spread of a malign influence in a social network. Here *approximate Oracles* are used since computing best responses is not practical.

A paper which provides strong motivation for methods within this thesis is due to Lin et al. (2013). They consider a patrol problem in which a patroller traverses the edges of a graph whose nodes are potential targets for an attacker. The patrol problem reflects real scenarios such as a security guard patrolling a museum or soldiers patrolling borders and the structure of the graph model reflects the underlying topology of the domain in question. Attacks at nodes take random amounts of time to complete and costs are incurred if they are not detected by the time they complete. The objective is to design a patrol policy which minimises the expected cost incurred by the attacker. The approach of the authors is first to model the case of *random attackers* who choose which node to attack according to a probability distribution known to the patroller. They then consider the case of *strategic attackers* who intelligently choose which node to attack in a zero-sum game with the patroller. The *insights and methods derived from the random attacker case lead to effective methods for the strategic attacker case.* In the random attacker case, the problem is formulated as a MDP which can be solved optimally for small size problems, but larger problems become computationally intractable due to the exponential growth of the state space with an increasing number of nodes. Consequently, the authors develop effective *index-based heuristic policies.* In the strategic attacker case, similar computational problems exist for finding an optimal patrol policy and so an effective heuristic policy is proposed which randomises over a feasible set of patrol policies. The feasible set is constructed in an iterative manner based on Fictitious Play (FP). In a given round, the attacker identifies a mixed strategy based on a best response to the history of patrol policies used by the patroller in previous rounds. The patroller interprets this mixed strategy as the attacker choosing which node to attack according to the corresponding probability distribution. This relates to the random attacker case in which the best response of the patroller is given by the index-based heuristic. The set of patrol policies revealed by this iterative procedure is used in a matrix game with the attacker, the solution of which gives a heuristic randomised patrol

strategy which is shown to exhibit near-optimal performance in numerical experiments. An extension to this work by Lin et al. (2014) allows for the possibility of *overlooking*, whereby there is a chance the attacker is not detected, even if the patroller inspects the node under attack. The authors adopt the same approach of random and strategic attacker problems in this more difficult problem. In the strategic attacker problem, the iterative method of constructing a feasible patrol set is more efficient than the method based on FP. The method used is analogous to the single Oracle algorithm of McMahan et al. (2003), since the mixed strategy of the attacker in a given round is now computed as the best response to the patrol policies currently available to the patroller. The difference with the algorithm of McMahan et al. (2003) concerns the *use of a heuristic as opposed to a best response Oracle*.

Although the inherent features of the defensive surveillance scenarios we consider within this thesis are somewhat different to the patrol scenarios considered by Lin et al. (2013) and Lin et al. (2014), there are elements which are similar. For example, a specific patrol policy will have some performance impact upon each target in the system in terms of what the adversary will achieve if he attacks that target. The patroller is concerned with protecting against any possible decision the adversary could make. Embedded within the strategic attacker problem, the random attacker problem was a mechanism with which to best protect against a variety of adversarial decisions. In essence, this is similar to the defensive surveillance scenario in Chapter 5 of this thesis and so we adopt the same paradigm of *utilising a random adversary problem within a strategic adversary problem*. Chapters 3 and 4 are analogous to the random attacker problem and are interesting within their own right. Chapter 5 is analogous to the strategic attacker problem and draws upon the insights of the random adversary case in earlier chapters.

The work by Lin et al. and Shieh et al. are examples of patrol games, a further example of which is given by Alpern et al. (2011). There are a number of other notable related problems worthy of consideration which use a game-theoretic

framework to study the interaction between forms of defenders and attackers. In *search games*, a searcher wishes to minimise the time taken to find a hider who does not want to be found on a network or in a region, for example see Alpern & Gal (2003). In *inspection games* (see Avenhaus & Canty (2002)), an inspector wishes to verify that an inspectee adheres to certain legal rules, for example in an arms control treaty which the inspectee may benefit from violating. In *interdiction or infiltration games*, an intruder wishes to maximise his probability of penetrating a sensitive area protected by a guard who wishes to minimise this probability, see Washburn & Wood (1995), Auger (1991), and Alpern (1992) for examples. In *accumulation games*, a hider distributes material over a set of locations and a seeker searches over these locations to confiscate the material, for example see Kikuta & Ruckle (2002). These are selected examples of what is a large and distributed body of literature and we refer the reader to the reviews given in these for further examples.

Chapter 3

Defence against Random Adversaries: Priority Policies

In this chapter we consider a stochastic scheduling problem with customer abandonments based upon the mathematical model outlined in Chapter 1 for defensive surveillance. We describe the model in this chapter for clarity. We consider a setting in which a single server must preemptively serve impatient customers spread across k customer classes. We can imagine every customer class residing in one queue or equivalently the case where each class corresponds to a separate queue. Different classes of customers arrive according to independent Poisson processes, with the arrival rate being λ_j for class j customers, $1 \leq j \leq k$. The service time for a class j customer is given by a random variable which follows an exponential distribution with rate μ_j. A class j customer, however, will only remain available for service for a random time that follows an exponential distribution with rate θ_j, after which the customer will abandon the system, whether the customer is still waiting in the queue or already in service. If a class j customer is served to completion, then a reward R_j is earned, but if he abandons the system before service completion, then a penalty D_j is incurred. In addition, each class i customer in the system incurs a linear holding cost at rate c_j per time unit. All of the parameters listed are assumed to be *strictly positive*, unless stated otherwise. We

seek to determine a service policy that *maximises the long-run reward rate earned net of penalties and holding costs incurred.*

The random adversary defensive surveillance scenario presented in Chapter 1 will be shown to be a special case of this model in which the penalties for abandonment are viewed as damages the adversary can inflict. Minimising the probability of abandonment or expected damage inflicted by the adversary is shown to be equivalent to minimising the long-run pure penalty rate, which itself is a special case of the general reward, penalty, and holding cost model described. This is achieved through a suitable definition of the penalty D_j, related to the random attack configuration of the adversary. We will discuss this connection and further the equivalence of finding a policy which maximises the long-run pure reward rate with similarly suitably defined rewards, which will be the main focus of this chapter.

3.1 Model and MDP Formulation

We develop further the problem of the optimal preemptive scheduling of the multiclass queueing system described above and formulate it as a Markov Decision Process (MDP). Recall that our model has three reward parameters. For a class j customer, there is a reward R_j for service completion, a penalty D_j for customer abandonment, and a linear holding cost rate c_j per unit time, for $1 \leq j \leq k$. If we write $N^{\pi}_{j,\theta}$ for the number of class j customers in the system and α^{π}_j, β^{π}_j for, respectively, the rate of class j service completions and abandonments under policy π in *steady state*, then the optimal long-run system reward rate net of holding costs and abandonment penalties can be written as

$$\max_{\pi} \sum_{j=1}^{k} \left(R_j \alpha^{\pi}_j - D_j \beta^{\pi}_j - c_j E[N^{\pi}_{j,\theta}] \right). \tag{3.1}$$

However, the guaranteed stability of the system implies that, for all choices of class j and policy π, we have that

$$\lambda_j = \alpha_j^\pi + \beta_j^\pi, \tag{3.2}$$

and

$$\beta_j^\pi = \theta_j E[N_{j,\theta}^\pi]. \tag{3.3}$$

Using (3.2) and (3.3), we can rewrite (3.1) in *three different ways*:

$$\max_\pi \sum_{j=1}^k \left(\left(R_j + D_j + \frac{c_j}{\theta_j} \right) \alpha_j^\pi - \left(D_j + \frac{c_j}{\theta_j} \right) \lambda_j \right) \tag{3.4}$$

$$= \max_\pi \sum_{j=1}^k \left(R_j \lambda_j - \left(R_j + D_j + \frac{c_j}{\theta_j} \right) \beta_j^\pi \right) \tag{3.5}$$

$$= \max_\pi \sum_{j=1}^k \left((R_j \lambda_j - ((R_j + D_j)\theta_j + c_j) E[N_{j,\theta}^\pi] \right). \tag{3.6}$$

Equation (3.4) transforms the original model into an equivalent pure reward model with $R_j + D_j + c_j/\theta_j$ earned upon every class j service completion. Similarly, equation (3.5) shows an equivalent pure penalty problem with penalty $R_j + D_j + c_j/\theta_j$ incurred upon every class j customer abandonment. Finally, equation (3.6) shows an equivalent pure holding cost problem with holding cost $(R_j+D_j)\theta_j+c_j$ per unit of time and per class j customer present in the system. Although expressed as maximisation problems, we could have expressed our original model as one of minimising the long-run holding cost and penalty rate net of rewards and the transformed pure forms would then be minimisation problems. Objective values of policies in the equivalent maximisation and minimisation problems would be equal up to a sign change. From this discussion we see that any model which features any combination of rewards, penalties, and holding costs can simply be transformed into one of the pure forms of the problem and the *parameters consolidated into a single parameter*. Without loss of generality, we shall focus on the pure reward maximisation problem ($D_j = c_j = 0$ for all j) for (almost all of) the remainder of the chapter.

We now formulate the pure reward problem as a MDP. Denote the system state $\mathbf{n} = (n_1, \ldots, n_k)$, with n_j the number of class j customers present in the system. We further write $\mathbf{n}(t)$ for the system state at time t. Further details of the model are as follows:

1. Decision epochs occur at time zero and at all transitions of the system state.

2. At each decision epoch, the server must decide which waiting customer to serve next across all customer classes. The set of admissible actions for state $\mathbf{n} \neq \mathbf{0}$ is given by

$$A(\mathbf{n}) = \{a : n_a \geq 1, 1 \leq a \leq k\}.$$

   Note, $A(\mathbf{n})$ is not defined for $\mathbf{n} = \mathbf{0}$. An action $a \in A(\mathbf{n})$ denotes the service of a class a customer. We use $\mathbf{e}_j$ for the system state in which only a single customer of class j is present in the system. We assume the server never idles when there are customers waiting in the system.

3. In state $\mathbf{n} \neq \mathbf{0}$ under admissible action $a \in A(\mathbf{n})$, the effective transition rate is $\Lambda(\mathbf{n}, a) = \mu_a + \sum_{j=1}^{k}(\lambda_j + n_j\theta_j)$. Transitions to states $\mathbf{n} + \mathbf{e}_j, \mathbf{n} - \mathbf{e}_a$, and $\mathbf{n} - \mathbf{e}_i, i \neq a$, respectively, occur with probabilities $\lambda_j\{\Lambda(\mathbf{n}, a)\}^{-1}$, $(\mu_a + n_a\theta_a)\{\Lambda(\mathbf{n}, a)\}^{-1}$, and $n_i\theta_i\{\Lambda(\mathbf{n}, a)\}^{-1}$. The effective transition rate in the empty state $\mathbf{0}$ is $\Lambda(\mathbf{0}) = \sum_{j=1}^{k}\lambda_i$ with a transition from $\mathbf{0}$ to state $\mathbf{e}_j$ occurring with probability $\lambda_j\{\Lambda(\mathbf{0})\}^{-1}$. When a transition from $\mathbf{n}$ to $\mathbf{n} - \mathbf{e}_a$ occurs at a class a service completion, a reward R_a is earned.

4. A service policy is a rule for choosing admissible actions using the history of the process (past states and actions) only. An *admissible, deterministic, stationary*, and *Markov* policy is determined by a function $\pi : \mathbb{N}^k \to \{1, \ldots, k\}$ satisfying $\pi(\mathbf{n}) \in A(\mathbf{n}), \forall \mathbf{n}$. The theory of MDPs (see, for example, Chapter 8 of Puterman (1994)) implies that, to determine the optimal policy, it is *sufficient to consider only policies in this class.*

5. The goal of analysis is to choose a policy that will maximise the long-run reward rate earned or that will come close to doing so.

A standard approach to the determination of ϵ-optimal policies is through the application of DP to a version of the above system with *finite state space* $\times_{j=1}^{k}\{0,1,\ldots,\ N_j\}$. In this truncated version of the above model, new class i customers are blocked from entering the system when N_i are already present. The N_j must be chosen large enough to ensure that this system approximates that in $1-5$ well enough for the purpose at hand. This reduction of the state space facilitates the conversion of the problem to one in discrete time through the process of *uniformisation* (see, for example, Lippman (1975) and Serfozo (1979)). We write $\Delta = \sum_{j=1}^{k}(\lambda_j + \mu_j + N_j\theta_j)$, a uniform upper bound on the rate of state transitions in the finite state system. By the addition of fictitious transitions from a state to itself, we develop a uniformised system that makes transitions at a uniform rate Δ. We write $V^{\pi}(\mathbf{n},t)$ and $V(\mathbf{n},t)$ for the expected reward earned under the application of policy π and an optimal policy, respectively, over t transitions of the uniformised process, beginning at time zero in system state $\mathbf{n}$. Standard theory enables us to write $V^{\pi}(\mathbf{n},t) = \frac{g^{\pi}}{\Delta}t + \omega^{\pi}(\mathbf{n}) + o(1)$ and $V(\mathbf{n},t) = \frac{g}{\Delta}t + \omega(\mathbf{n}) + o(1)$ as $t \to \infty$, where g^{π} and g are the long-run reward rates or *gains* earned, and ω^{π} and ω the *bias functions* under application of π and an optimal policy, respectively. Bias functions yield an estimate of the transient effect on rewards of the starting state $\mathbf{n}$ and will be further discussed in Chapter 4. Bellman's equation for the finite state system can now be written

$$\frac{g}{\Delta} + \omega(\mathbf{n}) = \max_{a}\left\{\frac{R_a\mu_a}{\Delta} + \sum_{\mathbf{n}'\in\mathcal{S}} p(\mathbf{n}'|\mathbf{n},a)\omega(\mathbf{n}')\right\}, \tag{3.7}$$

where the $p(\mathbf{n}' \mid \mathbf{n},a)$ are transition probabilities under the uniformisation. It is now possible to compute the optimal gain and associated optimal policy for the finite state approximation by a recursive scheme such as DP value iteration or by LP. Further details may be found in Chapter 8 of Puterman (1994).

Although it is possible to formulate our model as a MDP and use standard methods of DP to compute the optimal policy, since the state space grows exponentially in k, in practice, the computations quickly become intractable for $k \geq 4$. Hence, our focus is to develop strongly performing heuristic policies that require much less computation, with a preference for operationally simple policies with strong reward characteristics. The first element of our approach, featured in this chapter, is the development of a suite of simple *priority policies* which are effective across much of the problem's parameter space. Such policies serve customers according to a strict priority ordering among the customer classes. An additional attraction of priority policies is their simple structure, making them operationally desirable. The second element of our approach which follows from this will be the feature of Chapter 4. This involves the development of an effective approximate policy improvement method.

In general, good policies will achieve an appropriate trade-off between securing high returns from the customers currently available, while avoiding inefficiencies in processing, most especially when the server is idle. In the case of an overloaded system, there are almost always many customers present in the system. It is therefore intuitive that the server should pay little attention to the possibility of idling, and focus on continuously maximising the instantaneous reward rate. This can be done by serving according to the $R\mu$ *rule*, a priority policy that ranks all customer classes based on the product of reward R and service rate μ, namely the instantaneous reward rate. The strong performance of this rule in heavy traffic is shown in the literature in the work of Atar et al. (2010), Ayesta et al. (2011), Verloop (2014), and Larrañaga et al. (2014). Although this work considers variations of the $c\mu/\theta$ *rule* in linear holding-cost only models, it is a consequence of our analysis in equations (3.4) and (3.6) that this is *equivalent to the* $R\mu$ *rule* in our pure-reward model. We must recall here that when we refer to a rule, we simply refer to a service policy, as discussed on page 66.

Away from heavy traffic, lost reward opportunities due to an empty system

become a much more important concern. To complement the $R\mu$ rule in the light traffic case, we present the $R\mu\theta$ *rule* which ranks all customer classes based on the product of R, μ, and the abandonment rate θ. This policy incorporates the effect abandonments have on each class. This ranking was proposed in Section 2 of Glazebrook et al. (2004) for batch problems and shown to be effective, and we extend its application to systems with customer arrivals.

Section 3.3 considers the $R\mu\theta$ rule in greater detail and establishes its *asymptotic optimality* as $\theta \to 0$. Section 3.4 describes the extension of this result to other complex service situations. Section 3.5 compares the $R\mu\theta$ and the $R\mu$ rules. Finally, Section 3.6 presents a mechanism to explore local improvements on a given priority policy. We first discuss in Section 3.2 how the random adversary defensive surveillance scenario in Chapter 1 is a special case of our model.

3.2 Special Case: Random Adversaries

Recall the mathematical model underlying the three defensive surveillance models in Chapter 1. An adversary is a potential customer possessing the ability to join any of the k queues. If the adversary joins queue j then he behaves like every other customer in the queue, carrying an exponential service requirement with rate μ_j and having an exponential lifetime with rate θ_j. The goal of the adversary is to abandon the queue he joins before being served to completion; if this occurs a fixed amount of damage d_j is inflicted.

In the three different surveillance scenarios identified in Chapter 1, we denoted the random adversary scenario by one in which the adversary decides which queue to join at random according to a probability vector which is known to the server. We assume the adversary decides which queue to join according to a fixed probability vector $\mathbf{p} = (p_1, ..., p_k)$, such that queue j is joined with probability p_j and $\sum_j p_j = 1$. We assume that the *server knows the probability vector* $\mathbf{p}$ used by the adversary, perhaps gathered through intelligence operations. We can imagine two situations in which this situation may occur. Firstly, the adversary may have

generated $\mathbf{p}$ entirely at random, without any knowledge of the system. Alternatively, the adversary may have chosen $\mathbf{p}$ based on knowledge of the system and the potential service policies which could be in use in order to maximise the expected damage he can inflict. In either case, the objective of the server is the same.

The server does not know when the adversary will enter the system or indeed exactly in which queue, and even then cannot uncover the identity of the adversary until he has completed service or abandoned. However, the server does know that the adversary will enter at some point and the probability of entering each queue when this occurs. We assume that the adversary will join the system at some random point in time *while the system is in its long-run steady state position* under a given service policy π. The objective of the server is to determine a service policy which minimises the expected damage conditional on $\mathbf{p}$. If the adversary joins queue j at some point while the system is in its long-run steady state position under a given service policy π, the damage which will be inflicted is a discrete random variable with two outcomes: *damage d_j if he abandons* before completing service or *zero if he completes service*. The probability of each outcome depends on the service policy π. Recall α_j^π and β_j^π for, respectively, the rate of class j service completions and abandonments under policy π in steady state. The probability of abandonment of the adversary is the same as every other arbitrary class j customer and is equal to β_j^π/λ_j. Similarly, the probability of service completion is equal to α_j^π/λ_j. It is a consequence of (3.2) that these probabilities sum to one. Therefore, the expected damage inflicted by the adversary under policy π conditional on joining queue j is given by $d_j\beta_j^\pi/\lambda_j$. Subsequently, using the law of total expectation we have that the expected damage inflicted by the adversary joining the system according to $\mathbf{p}$ is given by $\sum_j (d_j\beta_j^\pi/\lambda_j)p_j$. It is easy to see that when $d_j = 1$ for $1 \leq j \leq k$, the expected damage simply equals the abandonment probability of the adversary. We can now express the optimal expected damage for the server as follows

$$\min_{\pi} \left[\sum_{j=1}^{k} \frac{d_j \beta_j^{\pi}}{\lambda_j} p_j \right].$$

It is possible to express this differently as follows

$$\min_{\pi} \left[\sum_{j=1}^{k} \left(\frac{d_j p_j}{\lambda_j} \right) \beta_j^{\pi} \right] = \min_{\pi} \left[\sum_{j=1}^{k} D_j \beta_j^{\pi} \right],$$

which expresses the problem as one of minimising the long-run penalty rate, where penalties $D_j = d_j p_j / \lambda_j$ are incurred for class j abandonments. From our earlier discussion, from a policy perspective, this pure-penalty problem is equivalent to a pure-reward problem in which the server maximises the long-run reward rate where rewards $R_j = d_j p_j / \lambda_j$ are received for service completions.

Remark. *Suppose that adversaries arrive into the entire system as a Poisson process with rate λ_A, then the total arrival rate into queue j would be $\lambda_j + p_j \lambda_A$. Each customer in queue j would independently be an adversary with probability $(p_j \lambda_A)/(\lambda_j + p_j \lambda_A)$. Given that adversarial arrivals would be very rare, we can safely assume that λ_A would be very small. Each time a customer is served in queue j, the probability this customer is an adversary would then be approximately equal to $(p_j/\lambda_j)\lambda_A$. We can trivially set the quantity λ_A equal to one given the common scaling effect on these probabilities in each queue. We can then interpret the class j reward $R_j = d_j p_j / \lambda_j$ as the expected damage avoided by the server in each class j service completion. Maximising the long-run reward rate is equivalent to maximising the long-run rate at which expected damages are avoided, which is intuitively equivalent to minimising the long-run rate at which expected damages are inflicted.*

We have shown in this section that the random adversary defensive surveillance scenario in which the single server wishes to minimise the expected damage inflicted by an adversary who attacks randomly is a special case of a more general stochastic scheduling problem with customer abandonments in which a server wishes to maximise the long-run reward rate from service completions. It is the

latter problem which is the main focus of this chapter and so in what follows we will consider this problem with arbitrary rewards R_j and note that it is possible to realise the former problem through the definition $R_j = d_j p_j/\lambda_j$.

3.3 The $R\mu\theta$ Rule

If a system is not overloaded with customers, then it becomes important to take into account the lost reward opportunities when the system becomes empty due to customer abandonment. For example, consider a two-class system, with $R_1\mu_1 = R_2\mu_2$, and $\theta_1 < \theta_2$. If there is one customer present from each class, then intuition suggests that the server should first serve the class 2 customer, since there is a better chance that the class 1 customer will still be available upon the other's service completion. Consequently, a class's priority should go up as its abandonment rate θ increases. We call the rule in which the server always serves a customer having the maximal $R\mu\theta$ value among all customers present in the system, the $R\mu\theta$ rule. As seen in equations (3.4) and (3.6), the $R\mu\theta$ rule in our pure-reward model is *equivalent to the $c\mu$ rule* in the linear holding-cost only model. Whereas the $c\mu$ rule is optimal in queueing systems with no customer abandonment (see, for example, Section 5.2 in Gittins et al. (2011)), it is not optimal in systems with customer abandonment (see Down et al. (2011)). To the best of our knowledge, the asymptotic optimality of this rule in systems with customer abandonment has never been established in the literature.

The main result of this section is to show that the $R\mu\theta$ rule is asymptotically optimal as $\theta \to 0$. In the random adversary surveillance scenario, this result sheds light on cases where the time spent by individuals in the area of interest tends to be large relative to the time taken to surveil them. First, write $R^{\pi}(\theta)$ for the reward rate achieved by policy π, and $R^{R\mu\theta}(\theta)$ for the reward rate achieved by the $R\mu\theta$ rule. To describe the limiting regime simply, we suppose that the abandonment rate of each customer class is the multiple of some underlying rate θ such that $\theta_j = \theta\nu_j$, where $\nu_j > 0$, $1 \leq j \leq k$.

Theorem 3.1. *If* $\sum_j \lambda_j/\mu_j < 1$, *then*

$$\max_{\pi} R^{\pi}(\theta) - R^{R\mu\theta}(\theta) \leq O(\theta^2).$$

Proof. Consider the k-class queueing system under stationary nonidling service control policy π, under an assumption that $\sum_j \frac{\lambda_j}{\mu_j} < 1$ in which the system is stable under nonidling policies in the absence of abandonments. We shall consider the system in the limit as $\theta \to 0$.

The reward rate under policy π

Under policy π and with abandonment parameter θ write $W^{\pi}_{j,\theta}$ for the waiting time of a class j customer in steady state, to be understood as follows: $W^{\pi}_{j,\theta}$ is the time needed for a class j customer arriving at the system in steady state and with zero personal abandonment rate to complete its service. An arriving class j customer in steady state will actually complete its service if this waiting time is no greater than $Y_j \sim \exp(\theta_j)$, the time available to the customer in the system prior to her abandonment. The quantity $E\left(\exp\left(-\theta_j W^{\pi}_{j,\theta}\right)\right) = E\left(P\left(W^{\pi}_{j,\theta} < Y_j\right)\right)$ is the long-run proportion of class j customers who achieve service completion under policy π. We can now write the reward rate achieved under π as

$$R^{\pi}(\theta) = \sum_{j=1}^{k} \lambda_j R_j E\left(\exp\left(-\theta\nu_j W^{\pi}_{j,\theta}\right)\right). \tag{3.8}$$

Lower bound for the reward rate

Now consider any *priority policy* ϖ, namely any policy which operates a fixed priority ordering among the customer classes. The $R\mu\theta$ and $R\mu$ rules are such policies. We shall assume without loss of generality that ϖ chooses individual customers from the chosen class for service in a first-come-first-served fashion. Now write W^{ϖ}_{j} for the waiting time (time to achieve completed service) of a class

j customer in steady state under priority policy ϖ for the no abandonment case with $\theta = 0$. It is clear from a simple argument based on realisations of the system that the total workload (uncompleted service) in the system from customers in the l classes which have top priority under ϖ (for any $1 \leq l \leq k$) when $\theta = 0$ stochastically bounds above the corresponding quantity when $\theta > 0$. Since epochs at which any class j customer enters service are those at which the workload from higher priority classes is zero, it follows straightforwardly that for priority policies ϖ we have $W_j^{\varpi} \geq_{ST} W_{j,\theta}^{\varpi}$, and hence from (3.8) that

$$\begin{aligned} R^{\varpi}(\theta) &\geq \sum_{j=1}^{k} \lambda_j R_j E\left(\exp\left(-\theta \nu_j W_j^{\varpi}\right)\right) \\ &= \sum_{j=1}^{k} \lambda_j R_j - \theta \sum_{j=1}^{k} \lambda_j R_j \nu_j E\left(W_j^{\varpi}\right) + O\left(\theta^2\right). \end{aligned} \tag{3.9}$$

Upper bound for the reward rate

In (3.9), we have established a lower bound for the reward rate $R^{\varpi}(\theta)$ for any priority policy ϖ. We now develop an upper bound for $R^{\pi}(\theta)$ for any π. To achieve this, we consider first a realisation of the system under nonidling policy π and with no abandonments ($\theta = 0$). This realisation will be determined by π and a given set of arrival times $\mathbf{A}$ and service durations $\mathbf{S}$. We write the lengths of successive busy periods for this realisation as B_n, and the number of customers served in successive busy periods as $M_n, n \in \mathbb{N}$. Write l_{nj} for the number of class j customers in period n, and $W_{jl}^{\pi(n)}$ for their waiting times, for $1 \leq l \leq l_{nj}, 1 \leq j \leq k$, where $\sum_j l_{nj} = M_n$.

We now apply abandonment to this realisation. Hence we consider the stochastic process generated when the realisation determined by $\pi, \mathbf{A}$, and $\mathbf{S}$ is modified by random abandonments with class-specific rates $\theta\nu_j$, $1 \leq j \leq k$. It is trivial that at all epochs at which the system is empty for the no abandonment realisation, it will also be empty when abandonments are applied. Expressed differently, the busy periods for the process without abandonments contain (one or more) busy

periods for any generated process with abandonments. When abandonments are applied to the realisation with waiting times $W_{jl}^{\pi(n)}, 1 \leq l \leq l_{nj}, 1 \leq j \leq k, n \in \mathbb{N}$, then it is easy to show that the probability that none of the customers served in busy period n is abandoned is bounded below by $\exp\left(-\theta\nu^* B_n M_n\right) \geq 1-\theta\nu^* B_n M_n$ where $\nu^* = \max_j \nu_j$.

Use W_{j-}^{π} for the collection of waiting times for class j customers of the non-abandonment realisation, namely $W_{j-}^{\pi} := \{W_{jl}^{\pi(n)}, 1 \leq l \leq l_{nj}, n \in \mathbb{N}\}$. We now seek a lower bound for the conditional expectation $E(W_{j,\theta}^{\pi} \mid W_{j-}^{\pi})$, which is the mean class j waiting time (after abandonments) conditional on this no-abandonment realisation. To compute this conditional expectation we use $\mathbf{X}^{(n)}$ for the collection of exponential random variables which determine the available lifetimes (deadlines) for the customers concerned with busy period n. We write $\{W_{jl}^{\pi(n)}\left(\mathbf{X}^{(n)}\right), 1 \leq l \leq l_{nj}, n \in \mathbb{N}\}$ for the new (random) class j waiting times which result from the abandonment process when applied to successive busy periods of the non-abandonment process. By the above argument

$$P(W_{jl}^{\pi(n)}(\mathbf{X}^{(n)}) = W_{jl}^{\pi(n)}, 1 \leq l \leq l_{nj}) \geq \exp(-\theta\nu^* B_n M_n)$$

from which it follows that

$$\begin{aligned}
E(W_{j,\theta}^{\pi} \mid W_{j-}^{\pi}) &= \lim_{N\to\infty} \frac{\sum_{n=1}^{N}\sum_{l=1}^{l_{nj}} E(W_{jl}^{\pi(n)}(\mathbf{X}^{(n)}))}{\sum_{n=1}^{N}\sum_{l=1}^{l_{nj}} l_{nj}} \\
&\geq \lim_{N\to\infty} \frac{\sum_{n=1}^{N}\sum_{l=1}^{l_{nj}} W_{jl}^{\pi(n)} \exp\left(-\theta\nu^* B_n M_n\right)}{\sum_{n=1}^{N}\sum_{l=1}^{l_{nj}} l_{nj}} \\
&\geq \lim_{N\to\infty} \frac{\sum_{n=1}^{N}\sum_{l=1}^{l_{nj}} W_{jl}^{\pi(n)}}{\sum_{n=1}^{N}\sum_{l=1}^{l_{nj}} l_{nj}} \left(1 - O\left(\theta\right)\right) \\
&= E(W_j^{\pi}) - O(\theta),
\end{aligned}$$

where the equality follows from the ergodicity of the system, and the lower bound $E(W_j^{\pi}) - O(\theta)$ is a uniform one that does not depend on W_{j-}^{π}. Unconditioning,

we infer that

$$E(W^{\pi}_{j,\theta}) \geq E(W^{\pi}_{j}) - O(\theta).$$

Combining this with (3.8), we have that

$$R^{\pi}(\theta) \leq \sum_{j=1}^{k} \lambda_j R_j - \theta \sum_{j=1}^{k} \lambda_j R_j \nu_j E(W^{\pi}_{j}) + O(\theta^2). \tag{3.10}$$

Please note that in (3.9) and (3.10), all the $O\left(\theta^2\right)$ terms, upon division by θ^2 involve expectations which are uniformly bounded as ϖ, π range over their respective policy classes.

Inference from the lower and upper bounds

Applying (3.10), we infer that

$$\max_{\pi} \left(\sum_{j=1}^{k} \lambda_j R_j - \theta \sum_{j=1}^{k} \lambda_j R_j \nu_j E\left(W^{\pi}_{j}\right) + O\left(\theta^2\right) \right) \geq \max_{\pi} R^{\pi}\left(\theta\right), \tag{3.11}$$

where the maxima in (3.11) are over all policies π. Making the $R\mu\theta$ rule the choice of priority policy ϖ in (3.9), using the fact that $\max_{\pi} R^{\pi}(\theta) \geq R^{R\mu\theta}(\theta)$, and using (3.11) it follows that

$$\max_{\pi} R^{\pi}\left(\theta\right) - R^{R\mu\theta}\left(\theta\right) \leq \theta \left\{ \sum_{j=1}^{k} \lambda_j R_j \nu_j E\left(W^{R\mu\theta}_{j}\right) - \min_{\pi} \sum_{j=1}^{k} \lambda_j R_j \nu_j E\left(W^{\pi}_{j}\right) \right\} + O\left(\theta^2\right). \tag{3.12}$$

Little's Law and conclusion

We finally observe that the minimisation in (3.12) can alternatively be written, using Little's Law, as

$$\min_{\pi} \sum_{j=1}^{k} R_j \nu_j E(N^{\pi}_{j}), \tag{3.13}$$

with N^{π}_{j} the number of class j customers in the system (without abandonments) in steady state under policy π. The minimisation in (3.13) is of a holding cost rate for the system, with cost $R_j \nu_j$ incurred per class j customer and per unit

of time. A classical queueing control result (the $c\mu$ rule) asserts that for the no abandonment system, this holding cost rate is minimised by the $R\mu\nu$ rule which provides service according to a priority policy with class ordering determined by (decreasing) values of $R_j\mu_j\nu_j$. Upon multiplication of these values by θ it is clear that this is our $R\mu\theta$ rule. We infer from this fact and from (3.12) that

$$\max_{\pi} R^{\pi}(\theta) - R^{R\mu\theta}(\theta) \leq O(\theta^2)$$

as required, which concludes the proof. □

In heavy traffic, the $R\mu$ rule appropriately greedily chooses processing actions to maximise the instantaneous reward rate achieved. In the regime of Theorem 1, the focus is on choices that minimise reward rate loss from the system through abandonments. From the preceding proof, this loss rate is given by $\theta h^{\pi} + O(\theta^2)$, where

$$h^{\pi} = \sum_{j=1}^{k} \lambda_j R_j \nu_j E\left(W_j^{\pi}\right).$$

The *strong performance of the $R\mu\theta$ rule resides in its minimisation of the dominant $O(\theta)$ component* of this loss rate—a consequence of the optimality of the $c\mu$ rule for linear holding costs in the absence of abandonments.

3.4 Extensions

Close inspection of the proof of Theorem 3.1 reveals that we make little use of the stochastic structure of the system's service mechanism. The $R\mu\theta$ rule emerges as a priority policy, which minimises a holding cost-type objective for the no abandonment system ($\theta = 0$). It is therefore unsurprising that the result can be generalised to *more complex service situations*, provided that a priority policy continues to optimise an appropriate holding cost type objective in the absence of abandonments. We now give some examples.

3.4.1 Klimov Networks

Consider a multiclass M/G/1 queueing network with Bernoulli feedback, known as a *Klimov Network*; see Klimov (1974) and Klimov (1978). Exogenous arrivals to the system form independent Poisson streams, with λ_i the rate for class $i, 1 \leq i \leq k$. With each class i is associated a collection J_i of service stations with $S_{ij} \sim G_{ij}$ a generic class i service time at station $ij, 1 \leq j \leq J_i$. All service times are mutually independent and are assumed to have finite second moment. Each class i customer begins service at station $i1$ and is thereafter routed for further service according to the Markovian routing matrix P^i or exits the system. Hence the sequence of stations visited by each class i customer forms a Markov Chain with departure from the system represented by entry into an absorbing state. A single server is available to provide service at all service stations, namely those in the collection $\cup_{1\leq i\leq k} \cup_{j\in J_i} \{ij\}$. This service is provided nonpreemptively in the case of general service times, which is the case we now consider.

We write $S_i \sim G_i$ for the *total* service requirement of a class i customer, namely the aggregate of all individual service times until the system is exited. We suppose that the $\sum_{i=1}^{k} \lambda_i E(S_i) < 1$ and hence that the system is stable under nonidling service. If we write N_i for the total number of class i customers present in the system in steady state then a holding cost objective $E\left(\sum_{i=1}^{k} C_i N_i\right)$ is minimised by a service policy which imposes a priority ordering $KR(\mathbf{C})$ among the stations, where $\mathbf{C} = (C_1, C_2, \ldots, C_k)$. See Klimov (1974) and Klimov (1978) for details.

We modify the above Klimov Network by *imposing customer abandonment*. Hence all class i customers have their sojourn in the system terminated at a time after entry which has an exponential distribution with rate $\theta_i = \theta\nu_i$, unless the customer has already exited the system upon completion of all service. We suppose that each class i customer who completes all service prior to abandonment earns a reward R_i. As before we write $R^{\pi}(\theta)$ for the reward rate achieved under service policy π. We write $\mathbf{R}\theta = (R_1\theta_1, R_2\theta_2, \ldots, R_k\theta_k)$ and $KR(\mathbf{R}\theta)$ for the Klimov ordering determined by $\mathbf{R}\theta$. The proof of the following result is in all essentials

unchanged from that of Theorem 1.

Corollary 3.1. *For the above Klimov Network, if* $\sum_{i=1}^{k} \lambda_i E(S_i) < 1$, *then*

$$\max_{\pi} R^{\pi}(\theta) - R^{KR(\mathbf{R}\theta)}(\theta) \leq O\left(\theta^2\right).$$

Remark. *Our assumption that all service times* S_{ij} *have finite second moment and are mutually independent is important in completing the proof of Corollary 3.1 in the manner of the proof of Theorem 1. This enables the use of the approximation*

$$E(e^{-\theta\nu_j W_j^{\pi}}) = 1 - \theta\nu_j E(W_j^{\pi}) + O(\theta^2). \tag{3.14}$$

To see why this is true, let B *denote the busy period of an M/G/1 queue without abandonments, where the arrival rate is* $\sum_{i=1}^{k} \lambda_i$, *and the service time* S *is given by* $S \sim \sum_{i=1}^{k} \frac{\lambda_i}{\Lambda} \otimes S_i$, *where* $S_i = \sum_{j=1}^{J_i} S_{ij}$. *The busy period* B *is invariant to service discipline. Hence, for any policy* π, *the waiting time of class* j *customers* W_j^{π} *is stochastically smaller than* B *and we can write* $E[(W_j^{\pi})^n] < E[B^n]$, *for all* $n \in \mathbb{N}$.

We require conditions which allow us to write $f(\theta) := E(e^{-\theta B}) - (1 - \theta E(B)) = O(\theta^2)$. *This is equivalent to* $\lim_{\theta \to 0} \frac{f(\theta)}{\theta^2} < \infty$. *To compute this limit, we use Takacs functional equation (see Takács (1962)) for the Laplace Transform (LT) of the busy period given by*

$$E(e^{-\theta B}) = B^*(\theta) = S^*(\theta + \lambda - \lambda B^*(\theta))$$

where B^* *and* S^* *denote the LT of the busy period and service time distributions respectively. To find the limit, use l'hôpital's rule twice by taking the derivative of Takacs equation to give*

$$\lim_{\theta \to 0} \frac{f(\theta)}{\theta^2} = \frac{B^{*(2)}(0)}{2} = \frac{E(S^2)}{2(1-\rho)^3}.$$

Hence the limit exists if the second moment of the service time distribution is finite. Since $S \sim \sum_{i=1}^{k} \frac{\lambda_i}{\Lambda} \otimes S_i$, *we know that* $E[S^2]$ *is finite because of our assumption*

that the second moment of each class i, station j service time distribution is finite. Consequently we are able to write

$$E(e^{-\theta B}) = 1 - \theta E(B) + O(\theta^2),$$

which enables us to establish Equation (3.14). This completes the proof of Corollary 3.1 in the case of generic service times.

A version of the above result *also holds for Markovian Klimov Networks* in which all individual service times are exponentially distributed and priorities between customers are imposed preemptively. Please also note that trivially the above also provides an analysis for a version of the model in this chapter with *general service times and nonpreemptive service*, i.e., a conventional multiclass *M/G/1 queueing system with no feedback*.

One application of the above network structure has all customers of class i needing an initial period of service of $\exp(\mu_{i1})$ duration. This service is conclusive with probability α_i. Should the first phase of service not prove conclusive, a second phase of service of $\exp(\mu_{i2})$ duration is provided prior to exiting the system. Priorities are imposed preemptively. In the context of defensive surveillance, this may represent a situation in which a security team splits its second-phase screening into *two phases*, the first which quickly screens suspects but may not be as accurate and the second which takes longer but is more accurate.

Following the approach taken in Glazebrook (1996), in this case the Klimov Rule $KR(\mathbf{R}\theta)$ operates as follows: station $i1$ has an associated *Klimov Index* η_{i1} given by

$$\eta_{i1} = \max\left\{R_i\mu_{i1}\theta_i\alpha_i; \frac{R_i\theta_i}{\mu_{i1}^{-1} + (1-\alpha_i)\,\mu_{i2}^{-1}}\right\},$$

while the station $i2$ has an associated index

$$\eta_{i2} = R_i\theta_i\mu_{i2}.$$

In this case the Klimov Rule orders the stations according to the values of the indices $\{(\eta_{i1}, \eta_{i2}), 1 \leq i \leq k\}$, with highest priority accorded to stations of highest index.

3.4.2 Multiserver Systems

Consider a multiserver version of our system with abandonments, with m servers working in parallel, then the required stability condition becomes $\rho := \sum_{j=1}^{k} \frac{\lambda_j}{\mu_j} < m$ and the $R\mu\theta$ rule now allocates preemptive service to the m customers present in the system whose associated $R_j\mu_j\theta_j$ are maximal. The proof of a suitable version of Theorem 1 for this system goes through up to (3.12). However, it is no longer true that the $R\mu\theta$ rule achieves the minimum in (3.12), though it does come close to doing so. To give a theoretical result for this system we need the quantity

$$B(m) = \rho (R\mu\nu)_{\max} \left(\frac{1}{\mu}\right)_{\max} I(m > 1),$$

where I is an indicator and the maxima in the expression are taken over the customer classes. The following result makes use of Theorem 3 in Glazebrook & Niño-Mora (2001), which shows that $B(m)$ bounds above the quantity multiplying θ on the right-hand side of (3.12) when there are m servers. It generalises Theorem 3.1 to multiserver systems.

Proposition 1. *When there are m servers and $\sum_{j=1}^{k} \frac{\lambda_j}{\mu_j} < m$*

$$\max_{\pi} R^{\pi}(\theta) - R^{R\mu\theta}(\theta) \leq \theta B(m) + O\left(\theta^2\right).$$

3.5 Numerical Study: Comparing $R\mu\theta$ and $R\mu$

In this section we present a set of numerical experiments which both illustrate the performance of the $R\mu\theta$ rule as described in Theorem 1, and also enable us to draw

comparisons between its performance and that of the $R\mu$ rule in the corresponding asymptotic regime. It is worth noting that we are now back in the single server case ($m = 1$).

It follows from calculations in the proof of Theorem 1 that when $\sum_{j=1}^{k} \lambda_j/\mu_j < 1$, we have $R^{\pi}(\theta) \to \sum_{j=1}^{k} \lambda_j R_j$, as $\theta \to 0$, for all priority policies (and hence both the $R\mu\theta$ and $R\mu$ rules). Unsurprisingly, all priority policies achieve the *maximal reward rate* $\sum_{j=1}^{k} \lambda_j R_j$ in the no abandonment limit, since in the limit all jobs are served. Think of the random adversary surveillance problem in which abandonments of the system are rare, but very damaging, and attention focuses on making the $O(\theta)$ loss rate from abandonments as small as possible. Now consider a situation in which the class orderings determined by the $R\mu\theta$ and $R\mu$ rules are distinct. Recall from page 75 the definition of h^{π} as the reward rate loss due to abandonments when using policy π. It follows from (3.9) and (3.10) that

$$R^{R\mu\theta}(\theta) - R^{R\mu}(\theta) = \theta(h^{R\mu} - h^{R\mu\theta}) + O\left(\theta^2\right).$$

When $R\mu\theta$ and $R\mu$ are distinct, the quantity that multiplies θ in the above expression is strictly positive. Consequently, there exists θ^*, such that for $\theta < \theta^*$, we have that $R^{R\mu\theta}(\theta) > R^{R\mu}(\theta)$. Therefore,

$$\frac{\max_{\pi} R^{\pi}(\theta) - R^{R\mu\theta}(\theta)}{\max_{\pi} R^{\pi}(\theta) - R^{R\mu}(\theta)} \to 0, \qquad \text{as } \theta \to 0.$$

It follows that the percentage loss of reward rate due to abandonment from the use of $R\mu\theta$ relative to that experienced from the use of $R\mu$ becomes negligible in the limit $\theta \to 0$.

We illustrate the convergence of the $R\mu\theta$ and $R\mu$ rules in Table 3.1. Reward rates for the $R\mu\theta$ and $R\mu$ rules to a given accuracy are obtained by truncating the state space and using the uniformisation technique to facilitate the deployment of value iteration. Truncation levels are set so that the resulting finite-state model provides a sufficiently good approximation to the original model. All results con-

cern k class systems, where $k = 2, 3$. We use $N_j = 100$ for each class j. Problems were randomly generated with respect to assumptions on system parameter values. In light of the discussion in the preceding paragraph, care was taken to ensure that the problems generated were such that $R\mu\theta$ and $R\mu$ rules were distinct. As in the preamble to Theorem 1, abandonment rates are expressed as a multiple of some underlying abandonment rate, $\theta_j = \theta\nu_j$ and all problems studied are such that $\rho = \sum_j \lambda_j/\mu_j < 1$. Problems were randomly generated as follows:

$$\mu_j \sim U[0.2, 5] \qquad \text{(all cases)}; \tag{3.15a}$$

$$\lambda_j \sim U[0.2, 5] \qquad \text{(all cases)}; \tag{3.15b}$$

$$\rho \in [0.5, 0.9] \qquad \text{(light traffic)}; \tag{3.15c}$$

$$R_j \sim U[1, 3] \qquad \text{(all } k = 2 \text{ system cases)}; \tag{3.15d}$$

$$R_j \sim U[1, 5] \qquad \text{(all } k = 3 \text{ system cases)}; \tag{3.15e}$$

$$\nu_j \sim U[1, 3] \qquad \text{(all cases)}; \tag{3.15f}$$

In the parameter generation, the μ_j and λ_j were generated according to (3.15a) and (3.15b) by means of a rejection algorithm until the desired ρ condition (3.15c) was met. For each system, 100 problems were generated at random according to (3.15a) to (3.15f) for a given set of abandonment rates. For each problem, value iteration was used to compute the gains of the $R\mu\theta$ and $R\mu$ rules and an optimal policy.

As seen in Table 3.1, the results reflect and illustrate the above observations concerning the relative reward rate performances of the $R\mu\theta$ rule and $R\mu$ rule. It is evident from the table that the percentage suboptimality of both policies go to zero in the limit $\theta \to 0$. As stated above, this would indeed be the case for any priority policy. However, it is evident that this convergence is much more rapid (of order θ^2) in the case of $R\mu\theta$, where it is already the case at $\theta = 0.1$ that the median percentage suboptimality of $R\mu\theta$ is zero (to 2 *d.p.*) for both $k = 2$ and $k = 3$. The much slower $O(\theta)$ convergence of the percentage suboptimality of the

$R\mu$ rule is particularly clear from the 'median' columns of the $R\mu$ part of Table 3.1.

	$R\mu\theta$				$R\mu$			
	$k=2$		$k=3$		$k=2$		$k=3$	
θ	Median	90th	Median	90th	Median	90th	Median	90th
5	0.28	1.26	0.10	0.78	0.00	0.06	0.00	0.14
2.5	0.23	1.37	0.06	0.87	0.00	0.39	0.02	0.36
1	0.16	1.36	0.02	0.67	0.00	0.79	0.06	0.68
0.5	0.06	0.97	0.01	0.76	0.07	1.06	0.15	0.94
0.1	0.00	0.32	0.00	0.27	0.22	1.09	0.18	0.84
0.05	0.00	0.17	0.00	0.14	0.20	1.08	0.15	0.72
0.025	0.00	0.07	0.00	0.04	0.15	0.80	0.12	0.56
0.01	0.00	0.01	0.00	0.00	0.09	0.47	0.05	0.39
0.005	0.00	0.00	0.00	0.00	0.05	0.28	0.04	0.20
0.0025	0.00	0.00	0.00	0.00	0.03	0.16	0.02	0.12
0.001	0.00	0.00	0.00	0.00	0.01	0.07	0.01	0.06
0.0005	0.00	0.00	0.00	0.00	0.01	0.04	0.01	0.03
0.00025	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.01
0.0001	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01

Table 3.1: Percentage suboptimalities of the $R\mu\theta$ rule and $R\mu$ rule in sets of 100 randomly generated problems in $k=2$ and $k=3$ class systems. Median and 90th percentiles are shown.

3.6 The PaS Class of Priority Policies

In the parameter space we know that the $R\mu$ rule is effective in heavy traffic and the $R\mu\theta$ rule is effective in the regime of Theorem 1. Hence for much of the parameter space at least one of $R\mu$ and $R\mu\theta$ will be effective. We conclude this chapter by developing a simple *pairwise-swapping mechanism* to explore local improvements in any given priority policy.

Given any class ordering $(\pi_1, \pi_2, \ldots, \pi_k)$, we take the classes in order from π_2 to π_k and explore, in turn, whether each class should be promoted up the order. This is achieved for each class by a sequence of pairwise comparisons with the next highest class in the list to determine how high up the list the class can be promoted. In comparing classes i and j, we consider the *two-class subsystem* comprising them alone (with their class parameters inherited from the full problem) and use value iteration to compute the respective performance of the two priority policies $i \to j$

and $j \rightarrow i$. If the better policy contradicts the current class ordering, then a pairwise swap is performed, and the procedure is repeated until a comparison does not result in a swap. We then examine the potential promotion for the next class on the original list π_2 to π_k. This process ends once all classes in the original list have been considered. The resultant class ordering prescribes a priority policy we refer to as *PaS*. The acronymn PaS refers to the ***Pa****irwise* ***S****wapping* procedure used to derive the policy. The derivation of PaS via this procedure is shown in Algorithm 1.

Algorithm 1 Algorithm to derive PaS from a given priority policy.

Given a class ordering $(\pi_1, \pi_2, \ldots, \pi_k)$, take a copy given by the vector A.
Let A_j denote the customer class of element j such that A_1 is the top customer class.
Set $i = 2$;
Set $j = i$;
while $i \leq k$ **do**
    Formulate a two-class problem with class $a = A_j$ and $b = A_{j-1}$;
    if priority policy $a \rightarrow b$ is better than priority policy $b \rightarrow a$ **then**
        $i \leftarrow i + 1$
        $j \leftarrow i$
    else
        Swap customer classes A_j and A_{j-1}
        if $j > 1$ **then**
            $j \leftarrow j - 1$
        else
            $i \leftarrow i + 1$
            $j \leftarrow i$
        end if
    end if
end while
The resultant ordering $(A_1, A_2, \ldots, A_k)$ defines PaS.

This approach is founded on the observation that the evaluation of priority policies in a $k = 2$ class subsystem through DP methods is a relatively *cheap* computational operation. The pairwise-swapping mechanism is analogous to an insertion sort which has best and worst case performance of $\mathcal{O}(k)$ and $\mathcal{O}(k^2)$ respectively in terms of the number of operations. These reflect the number of $k = 2$ subsystems which must be evaluated to obtain the final PaS ordering. The computational cost of these evaluations is low. Although there is no guarantee that the

PaS policy is indeed an improvement on the initial priority policy, our basic guiding principle is that prioritising one class above another in the two-class subsystem comprising them alone may suggest that this *prioritisation should be retained* in the full system. Pairing this insight with the low computational expense suggests PaS as an effective heuristic priority policy.

We now illustrate the pairwise-swapping mechanism through a numerical example.

EXAMPLE 3.1: Consider a system with $k = 5$ queues. The class j parameters of the example system are as follows

$$(\lambda_j, \mu_j, \theta_j, R_j) = \begin{cases} (3\lambda/5, 3, 0.1, 7.5) & j = 1 \\ (\lambda/5, 5, 1, 2.5) & j = 2 \\ (4\lambda/5, 4, 5, 1) & j = 3 \\ (3.5\lambda/5, 3.5, 0.2, 5) & j = 4 \\ (4.5\lambda/5, 4.5, 1.5, 2) & j = 5 \end{cases}$$

The traffic intensity is $\rho := \sum_j \rho_j = \sum_j \lambda_j/\mu_j$ and in this example we have $\lambda = \rho = 1.1$, with equal traffic intensity in each class $\rho_1 = \rho_2 = \rho_3 = \rho_4 = \rho_5$. The $R\mu\theta$ rule prescribes the priority policy *35241* whereas the $R\mu$ rule prescribes the priority policy *14253*. Intuitively it may be better to give higher priority to class 2 and class 5. The low abandonment rates in classes 1 and 4 mean that customers are likely to stay in the queue for a long time, meaning that a class 2 or 5 customer could be served and the high reward class 1 or 4 customers will still be be available to serve. The high abandonment rate and low reward in class 3 suggests a lower chance of obtaining a lower reward if a class 3 customer were served ahead of a class 2 or 5 customer for example.

We take an initial class ordering from the $R\mu\theta$ rule, hence 35421. In Table 3.2 we show the reward rates under priority policies $i \to j$ in the two-class subsystems comprising classes i and j alone. Typically, these reward rates would be computed as required, but for illustrative purposes we have computed these in advance for

		j				
		1	2	3	4	5
i	1		7.310	5.506	8.905	6.511
	2	7.505		2.767	6.221	3.822
	3	5.550	2.749		4.279	1.898
	4	8.958	6.073	4.248		5.265
	5	6.655	3.813	1.911	5.374	

Table 3.2: Reward rates of priority policies $i \to j$ in two-class subsystems comprising classes i and j alone in Example 3.1.

every i, j combination. The pairwise-swapping mechanism proceeds as follows:

- We begin by comparing class 5 with class 3. The reward rate of 1.898 for $3 \to 5$ is less than 1.911 for $5 \to 3$, so we swap the order of these two classes. No further comparisons of class 5 can be made as it occupies the top position. The order is 53241.

- We next consider class 2 and first compare it with class 3. Priority policy $2 \to 3$ is better than $3 \to 2$, so we swap the order. We now compare class 2 with class 5. Priority policy $2 \to 5$ is better than $5 \to 2$, so we swap the order. Class 2 now occupies the top position and no further comparisons can be made. The order is 25341.

- We next consider class 4 and first compare it with class 3. Priority policy $4 \to 3$ is worse than $3 \to 4$, so we do not swap the order. Since the comparison did not yield a swap, we end consideration of class 4. The order is 25341.

- We finally consider class 1 and first compare it with class 4. Priority policy $1 \to 4$ is worse than $4 \to 1$, so we do not swap the order. Since the comparison did not yield a swap, we end consideration of class 1. The order is 25341.

- All classes in the original order have been considered so the resultant ordering defines *PaS to be 25341*. This final order would also have been obtained had the $R\mu$ rule been used to define the initial ordering.

The reward rates of the $R\mu\theta$ rule, the $R\mu$ rule, and PaS cannot be computed through DP methods due to computational intractability. However, we compute Monte Carlo estimates of the reward rates through discrete event simulation of each policy. These reward rate estimates are *11.14, 10.91, and 11.26 for the $R\mu\theta$ rule, the $R\mu$ rule, and PaS respectively*, where each estimate is the mean of 1000 independent replications. In this example we estimate the *percentage improvement of PaS over the $R\mu\theta$ and $R\mu$ rules to be* 1.1% *and* 3.2% *respectively.* This demonstrates, at least in this case, the potential for PaS to deliver improvements over a given priority policy. In this example we were only able to compare performance between policies and on this basis, together with its computational feasibility, it appears that PaS is a potentially effective heuristic policy. However, we were unable to ascertain the performance of any policy relative to the optimal policy, which would be the basis on which to assess the actual effectiveness of any service policy. In Chapter 4 we conduct an extensive numerical study to assess the performance of a range of heuristic policies, including $R\mu\theta$, $R\mu$, and PaS, with respect to optimality or an appropriate upper bound for a variety of $k = 2, 3$ and 5 class systems for a wide range of system parameters. Hence, we refer the reader to Chapter 4 for further numerical examples and a more in-depth numerical assessment of the priority policies in this chapter.

Conclusion

In this chapter we have studied a stochastic scheduling problem with customer abandonments, of which the random adversary surveillance scenario is a special case. In this scenario, the security team knows the decision of the adversary in a probabilistic sense and we found that it is very effective for the security team to use a single, deterministic service policy in response to this. The service policies proposed within this chapter are priority policies. In a defensive surveillance setting, a priority policy would translate into the security team ranking suspects from the different target areas according to some rule. Screening would then be

provided according to this ranking. In a heavy traffic system, the $R\mu$ rule is shown in the literature to perform well and would be the suggested policy to minmise the expected damage inflicted by the adversary. In a light traffic system, with small abandonment rates, we showed the strong performance of the $R\mu\theta$ rule and this would be the suggested policy. Furthermore, we suggest that the security team could perhaps improve performance by using another priority policy known as the pairwise swapping (PaS) policy. In this chapter we also discussed how results associated with the $R\mu\theta$ rule could be extended into some more general problems. For example, multiserver systems and Klimov Network models. The literature would suggest that a generalisation of the $R\mu$ rule would also perform well in heavy traffic in multiserver systems. In any more general versions of the random adversary surveillance problem, extensions of the priority policies discussed in this chapter would be good initial starting points for the security team.

Chapter 4

Defence against Random Adversaries: Approximate Policy Improvement

In Chapter 3 we considered a stochastic scheduling problem with customer abandonments with a focus on developing strongly performing heuristic policies. We developed the first element of our approach, a suite of simple priority policies which are both operationally simple and could be effective over much of the problem's parameter space. In this chapter we study this stochastic scheduling problem further and develop the second element of our approach, an effective approximate policy improvement (API) method. This approach attempts to improve an existing candidate policy, in this case one of our simple priority policies, using the concept of policy improvement from DP. The policy improvement is approximate due to the computational intractability of DP cited in Chapter 3. Our numerical results indicate that, in most cases, the *best of* our priority policies is nearly optimal in systems with *2 or 3 customer classes* and we have an effective service policy of *simple structure*. In the cases where it is not, the API method invariably *tightens up the gap* substantially and provides an improved policy, albeit of more *complex structure*. In one instance, the API method improves our best priority

policy that is 4.26% suboptimal, to an improved policy that is only 0.04% suboptimal. It is for the decision maker to perform the trade-off between simplicity of policy structure and strength of reward performance. In our motivating random adversary defensive surveillance application, even small differences in reward rate performance can be of practical importance, hence any approach capable of delivering improvements is valuable to the security team.

There are some recent works on approximate approaches to DP seeking to overcome computational intractability. See, for example, Powell (2011). Contributions that deploy value function approximations within a policy approach include those of Krishnan (1987), Glazebrook et al. (2004), and Li & Glazebrook (2010), while API methods which utilise simulation are discussed by Powell (2011) and Bertsekas (2012). Our approach can be viewed as an ADP implementation, where the novelty lies in its special choice of initial policies and its use of *simulation* to estimate the bias functions at a set of carefully chosen states, followed by *interpolation* to secure estimates elsewhere.

4.1 Heuristic Based on Policy Improvement

Policy improvement develops optimal policies for MDPs by using the DP recursion to produce a sequence of successively improving policies (Howard, 1960). In our problem, we truncate the state space and uniformise, as in Section 3.1, to develop an ergodic system with optimality equation in (3.7). To develop a PI step from policy π, let $\omega^{\pi}(\mathbf{n})$ *be the bias* associated with system state $\mathbf{n}$ under policy π. A new policy PI_{π}, say, is obtained as follows:

$$\mathrm{PI}_{\pi}(\mathbf{n}) = \underset{a}{\operatorname{argmax}} \left\{ \frac{R_a \mu_a}{\Delta} + \sum_{\mathbf{n}' \in \mathcal{S}} p(\mathbf{n}'|\mathbf{n}, a) \omega^{\pi}(\mathbf{n}') \right\}. \tag{4.1}$$

Accordingly, policy PI_{π} always takes the current decision optimally, given that all future decisions are made according to π. Tijms (1994) noted that the first few PI iterations usually yield the greatest improvement. Whereas policy PI_{π} is

a heuristic policy based on a single, exact, PI step, in principle it is possible to perform multiple PI steps to determine a heuristic policy with strong reward rate performance.

The challenge to implementation of PI in large systems lies in the intractability of the computation of the bias ω^π. Hence, approximations are required, and the PI step in (4.1) can be replaced by

$$\text{API}_\pi(\mathbf{n}) = \underset{a}{\text{argmax}} \left\{ \frac{R_a \mu_a}{\Delta} + \sum_{\mathbf{n}' \in \mathcal{S}} p(\mathbf{n}'|\mathbf{n}, a) \tilde{\omega}^\pi(\mathbf{n}') \right\}, \tag{4.2}$$

where $\tilde{\omega}^\pi$ approximates ω^π.

Recall $V^\pi(\mathbf{n}, t)$ for the expected reward earned under the application of policy π over t transitions of the uniformised process, beginning at time zero in system state $\mathbf{n}$. The bias functions, with respect to a reference state $\mathbf{m}$, are defined as

$$\omega^\pi(\mathbf{n}) = \lim_{t \to \infty} \{V^\pi(\mathbf{n}, t) - V^\pi(\mathbf{m}, t)\}.$$

The bias function measures the asymptotic relative difference in total reward which results from starting in state $\mathbf{n}$ as opposed to the reference state $\mathbf{m}$. Computation of the bias ω^π involves specification of the reference state $\mathbf{m}$, which we take to be one *frequently visited* under π. We introduce the following quantities:

- $r^\pi(\mathbf{n})$ is the expected reward received starting from state $\mathbf{n}$ until the system enters the reference state $\mathbf{m}$ for the first time, if policy π is used.
- $t^\pi(\mathbf{n})$ is the expected time starting from state $\mathbf{n}$ until the system enters the reference state $\mathbf{m}$ for the first time, if policy π is used.

The system evolving under policy π is ergodic and so $r^\pi(\mathbf{n})$ and $t^\pi(\mathbf{n})$ are guaranteed to be finite for all states. Using the fact that *the system regenerates upon entry to the reference state*, the theory of regenerative processes (see Tijms (1994)) indicates that

$$\omega^\pi(\mathbf{n}) = r^\pi(\mathbf{n}) - g^\pi t^\pi(\mathbf{n}), \tag{4.3}$$

where g^π is the gain of policy π.

From (4.3), the approximations $\tilde{\omega}^\pi(\mathbf{n})$ can then be obtained by approximating the quantities $r^\pi(\mathbf{n})$, $t^\pi(\mathbf{n})$, and g^π. A heuristic policy based on an API step can then be defined from (4.2).

4.2 The Algorithm

The implementation of an API step depends crucially on the approximation scheme used for the bias functions. As the bias function does not have an analytical form, we use *discrete-event simulation* to estimate it. However, since simulation carries a computational cost, our constrained computational resource needs to be effectively managed through a carefully designed algorithm. The algorithm consists of five sequential, complementary stages, taking an initial policy π as an input to produce a new policy API_π. The five steps are summarised below, with more details to follow.

1. *Pilot*: Simulate the steady state of initial policy π to estimate its gain, and the frequency each state is visited.

2. *Selection*: Based on the pilot run, select a set of states at which we estimate the bias function via simulation.

3. *Sampling*: For each state $\mathbf{n}$ selected, simulate the system under π from that state until some chosen reference state $\mathbf{m}$ is entered and estimate $\omega^\pi(\mathbf{n})$ using (4.3).

4. *Interpolation*: Use the simulation results for selected states to interpolate the bias functions for all other unselected states.

5. *Improvement*: Use (4.2) to produce a new policy API_π.

In step 1, we run a pilot steady state Monte Carlo simulation to estimate the gain g^π required to estimate ω^π from (4.3). We take the estimate of the gain g^π to be

the sample average of a fixed number of replications. To facilitate steps 2 and 3, we also collect data on how often each state is visited in steady state under π.

Step 2 consists of the selection of a small number of states, denoted S_{sel}, at which the bias ω^{π} will be estimated by simulation in step 3. Interpolation of ω^{π} at other states will follow in step 4. Although desirable, estimation of ω^{π} by simulation at all states is not feasible given a fixed computational resource, hence our design manages this resource. The set S_{sel} consists of the *anchor set* together with a *support set*. The anchor set consists of the states most frequently visited in the pilot and hence influential to policy performance. However, anchor states are likely to be tightly grouped together, so alone they will not create an adequate basis for the construction of an effective interpolation scheme. The support set will complement the anchor set to ensure adequate coverage and wider exploration of the state space. Insufficient exploration of the state space is a commonly cited drawback in API methods (see Bertsekas (2012)) and our support set aims to address this issue.

To select M support states, we adopt lattice points of the following form:

$$P_M = \{((\mathbf{z}j \bmod M)/M) = ((z_1 j \bmod M)/M, ..., (z_k j \bmod M)/M) \mid 0 \leq j \leq M-1\},$$

where $\mathbf{z}$ is an integer vector modulo M. The components of $\mathbf{z}$ are chosen to be *relatively prime to each other and to* M. In what follows, policies will be constructed for numerous problems with $k = 2, 3$, and 5 making use of the choices $\mathbf{z} = (2, 3)\,, (2, 3, 5)$, and $(2, 3, 5, 7, 11)$, respectively. These lattice points are then appropriately scaled and rounded from the unit hypercube to the state space to obtain the support states. Such well-spread points were proposed in the field of Quasi-Monte Carlo methods for numerical integration and shown to enable good approximations of integrals (Niederreiter, 1978).

In step 3, we choose reference state $\mathbf{m}$ to be *the one most visited* in the pilot and use Monte Carlo simulation to estimate $r^{\pi}(\mathbf{n})$ and $t^{\pi}(\mathbf{n})$ for each $\mathbf{n} \in S_{\text{sel}}$. In what follows, we use n for the size of S_{sel} and m for the number of simulated

realisations of the system from each $\mathbf{n} \in S_{\text{sel}}$ until entry into reference state $\mathbf{m}$. In each realisation we record the reward received and time taken for the system to go from each initial state $\mathbf{n} \in S_{\text{sel}}$ to the reference state $\mathbf{m}$ and use the sample averages of the m realisations to estimate $r^{\pi}(\mathbf{n})$ and $t^{\pi}(\mathbf{n})$.

If we write $R^{\pi}(\mathbf{n})$ and $T^{\pi}(\mathbf{n})$ for the simulation-based estimators of $r^{\pi}(\mathbf{n})$ and $t^{\pi}(\mathbf{n})$, respectively, and G^{π} for the estimator of g^{π} available from the pilot, then from (4.3) our estimator of $\omega^{\pi}(\mathbf{n})$ for $\mathbf{n} \in S_{\text{sel}}$ is $\Omega^{\pi}(\mathbf{n}) := R^{\pi}(\mathbf{n}) - G^{\pi}T^{\pi}(\mathbf{n})$. Since all estimators are unbiased, and G^{π} is independent of $R^{\pi}(\mathbf{n})$ and $T^{\pi}(\mathbf{n})$, we conclude by conditioning on G^{π}, that

$$\begin{aligned}
\text{Var}\{\Omega^{\pi}(\mathbf{n})\} &= \text{Var}\{E(R^{\pi}(\mathbf{n}) - G^{\pi}T^{\pi}(\mathbf{n})|G^{\pi})\} + E\{\text{Var}(R^{\pi}(\mathbf{n}) - G^{\pi}T^{\pi}(\mathbf{n})|G^{\pi})\} \\
&= (t^{\pi}(\mathbf{n}))^2\text{Var}\{G^{\pi}\} + \text{Var}\{R^{\pi}(\mathbf{n}) - g^{\pi}T^{\pi}(\mathbf{n})\} + \text{Var}\{G^{\pi}\}\text{Var}\{T^{\pi}(\mathbf{n})\} \\
&= \text{Var}\{R^{\pi}(\mathbf{n}) - g^{\pi}T^{\pi}(\mathbf{n})\} + \text{Var}\{G^{\pi}\}E\{(T^{\pi}(\mathbf{n}))^2\}. \qquad (4.4)
\end{aligned}$$

Equation (4.4) decomposes the variance of the bias estimators into two terms. The first term is controlled by the *number of replicates* m used in the simulation relating to state $\mathbf{n}$ in step 3, and the second term is controlled by the *size of the pilot study* in step 1. The computational challenge is dominated by the need to control the first term in (4.4), as designing a pilot study large enough to control the second term has not proved to be an issue. One feature that helps reduce the first term is that $R^{\pi}(\mathbf{n})$ and $T^{\pi}(\mathbf{n})$ are positively associated. In addition, our choice of reference state $\mathbf{m}$ means that the biases at anchor states (with smaller $R^{\pi}(\mathbf{n})$ and $T^{\pi}(\mathbf{n})$) tend to be estimated with *greater precision* than those at support states, which is a feature shared with other approaches to ADP (for example, see Powell (2011)). The central trade-off for the quality of the method for given computational effort is that between large n supporting the quality of the interpolation, and large m supporting precision at the selected states.

In step 4, we use the bias estimates in S_{sel} to construct a bias function approximation for the entire state space. This poses a multivariate function approximation

problem for which many methods are applicable, for example variations of least-squares regression (see Powell (2011)). Such parametric approaches to the problem require the design of an effective parametric model, which can be difficult in practice. *Nonparametric methods*, such as *interpolation*, are an effective alternative. While there are many interpolation algorithms, we use the *radial basis function* method (see Powell (1987)) for its simplicity.

Assume that some function $f : S \to \mathbb{R}$ has known values at each $\mathbf{x}_i \in S_{\text{sel}}$. An augmented radial basis function $h : S \to \mathbb{R}$ which takes the form

$$h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \phi(\|\mathbf{x} - \mathbf{x}_i\|) + \sum_{j=1}^{d} \beta_j p_j(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^k, \tag{4.5}$$

will be designed as a smooth interpolator of f, taking the values $f(\mathbf{x}_i)$ for $\mathbf{x}_i \in S_{\text{sel}}$. From (4.5), $h(\mathbf{x})$ is a weighted sum of $n = |S_{\text{sel}}|$ *radial basis functions* $\phi(\cdot)$, one centred on each $\mathbf{x}_i \in S_{\text{sel}}$, together with d *low order polynomials* $p_j(\cdot)$. Note that $\|\cdot\|$ denotes the Euclidean norm. For $\phi(\cdot)$, we take the *thin plate spline* $\phi(r) = r^2 \log(r)$; for low order polynomials, we set $d = k + 1$ and use $p_1(\mathbf{x}) = 1$, $p_j(\mathbf{x}) = x_{j-1}$, $2 \leq j \leq k + 1$. These choices produce a surface which *minimises a measure of smoothness* (Powell, 1999).

We write A for the $n \times n$ matrix with elements $A_{ij} = \phi(\|\mathbf{x}_i - \mathbf{x}_j\|)$, $1 \leq i, j \leq n$ and P for the $n \times (k + 1)$ matrix with elements $P_{ij} = p_j(\mathbf{x}_i)$, $1 \leq i \leq n$, $1 \leq j \leq k + 1$. We write $\mathbf{f}$ for the n-vector with $f_i = f(\mathbf{x}_i)$, $1 \leq i \leq n$. Let $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ be corresponding vectors of coefficients. The matrix form of the interpolation problem is

$$\begin{pmatrix} A & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{0} \end{pmatrix}.$$

The equations $A\boldsymbol{\alpha} + P\boldsymbol{\beta} = \mathbf{f}$ ensure that $h(\mathbf{x}_i) = f(\mathbf{x}_i)$, $\mathbf{x}_i \in S_{\text{sel}}$, while the $k + 1$ equations $P^T\boldsymbol{\alpha} = \mathbf{0}$ take up the extra degrees of freedom in the problem, which ensures the radial basis function $h(\cdot)$ is conditionally positive definite and the interpolation problem solvable. Consequently, the interpolation matrix delivers a

unique solution in the coefficients and hence in h. If we take $f(\mathbf{x}_i)$, $\mathbf{x}_i \in S_{\text{sel}}$, in the above to be the estimates of bias from step 3, we can then use the resulting $h(\mathbf{x})$, $\mathbf{x} \in S$, as bias estimates for all states.

In step 5, we design a new policy API_π by using the function h from step 4 in place of $\tilde{\omega}^\pi$ in (4.2) to obtain

$$\text{API}_\pi(\mathbf{n}) = \underset{a}{\text{argmax}} \left\{ \frac{R_a \mu_a}{\Delta} + \sum_{\mathbf{n}' \in \mathcal{S}} p(\mathbf{n}'|\mathbf{n}, a) h(\mathbf{n}') \right\}.$$

In principle, the above procedure can be repeated multiple times. Although obtaining progressively better policies—a feature of exact PI—can no longer be guaranteed, we have generally found that, in practice, improvement in policy performance is indeed achieved. To highlight key design choices, we denote the above procedure by $\text{API}(\pi, n, m, r, t)$. The parametrising arguments offer great flexibility and are as follows: the initial policy π, n the number of selected states to run simulation to estimate the bias function, m the number of replicated simulations at each selected state, r the fraction of selected states that are included in the anchor set (so $1 - r$ is the fraction in the support set), and t the number of iterations of the algorithm. In what follows, we write API_π for the best-performing policy from t iterations of the algorithm, including the initial policy, which ensures that we only consider policies which improve as t increases. The trade-off between different choices of the parameters will be explored in Section 4.4, where we will give a recommendation for their selection.

We now present an example to illustrate the algorithm.

EXAMPLE 4.1: Consider a $k = 2$ class system with the parameters: $\lambda_1 = 2.5, \lambda_2 = 3, \mu_1 = 3.5, \mu_2 = 4, \theta_1 = 0.75, \theta_2 = 2.5, R_1 = 2.5, R_2 = 1.7$. We use truncation levels $N_1 = N_2 = 20$ throughout. Please note that for this example the $R\mu$ rule gives priority to class 1, while the $R\mu\theta$ rule gives priority to class 2. We use algorithms of the form $\text{API}\big(R\mu\theta, 45, m, \frac{32}{45}, 1\big)$ to construct policies. Figures 4.1 to 4.3 illustrate the selection and interpolation stages of the algorithm for the case $m = 10^5$. Figure 4.1 shows S_{sel} within the selection stage, with anchor states shown

as diamonds and support states as circles. Figure 4.2 shows the interpolated bias estimates over the entire state space. Figure 4.3 adds to this the surface of the exact biases ω^π and plots the absolute difference between the interpolated and exact biases at each state. Analysing Figure 4.3, although this cannot be observed clearly, the *interpolated surface closely resembles the surface of exact biases* ω^π, capturing its shape and curvature well, especially so around the anchor set. This is further evidenced by the fact that the absolute differences are small. Although not shown here, the interpolated surface is also an accurate and smooth reconstruction of the surface which would have been obtained had simulation been possible at every state. Figure 4.4 shows the actions taken in each state by the optimal policy for this example, along with the actions resulting from use of the above algorithm with m set at 10^3, 10^4, and 10^5. We observe that as m increases, the corresponding policies *approach more closely the switching curve structure* of the optimal policy. This observation is also reflected in the percentage suboptimalities of each policy, which are 0.33%, 0.24%, and 0.01% for $m = 10^3$, 10^4, and 10^5 respectively. This performance can be compared to percentage suboptimalities of 1.56% and 0.34% for the $R\mu$ and $R\mu\theta$ rules respectively. This example indicates policy improvement delivered through the API algorithm, with greater improvement achieved as m increases.

4.3 An Upper Bound on Achievable Rewards

In order to evaluate heuristic policies when the optimal solution in (3.7) is not available, we derive an *upper bound* for the long-run reward rate. For a given feasible policy, if x_j represents the implied fraction of time the server spends serving class j customers, then

$$\sum_{i=1}^{k} R_i \mu_i x_i \tag{4.6}$$

is the long-run reward rate for the feasible policy. To compute an upper bound for the optimal long-run reward rate, we formulate a linear program with the

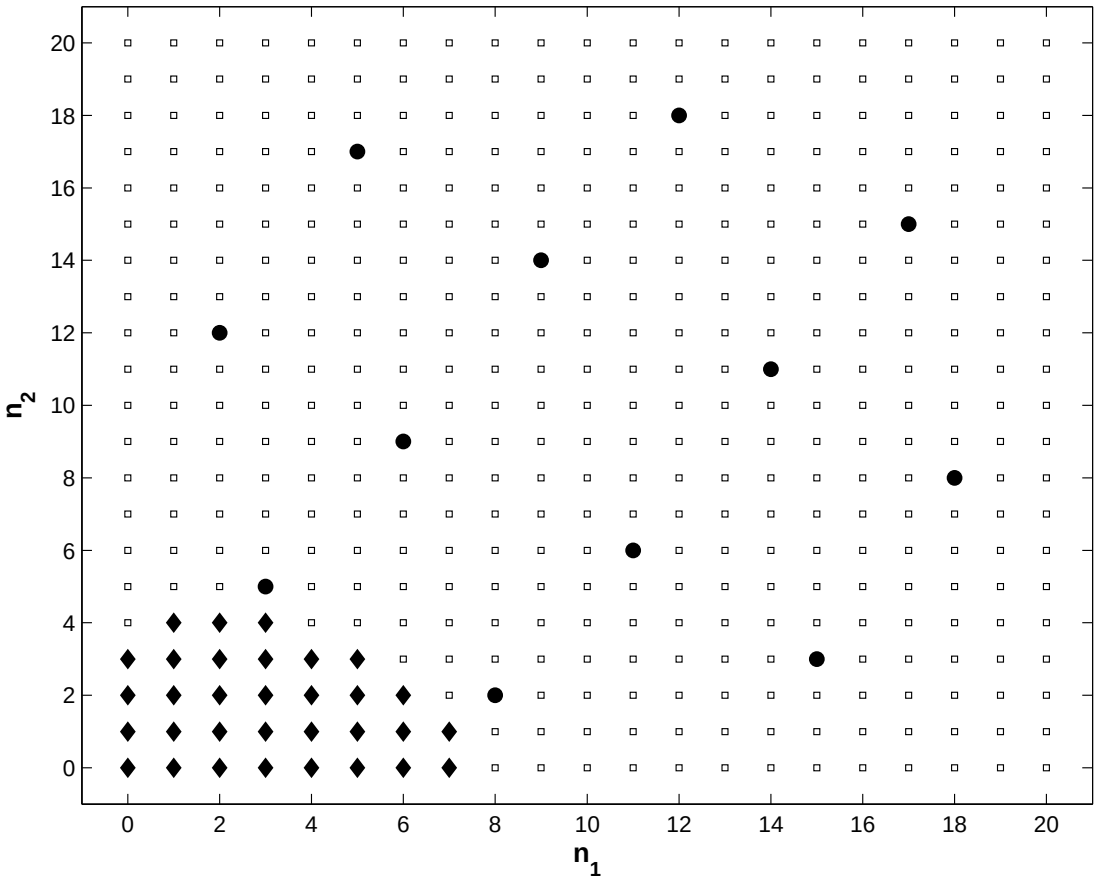

Figure 4.1: Illustration of the selection stage of the API algorithm in Example 4.1.

variables $x_j \geq 0$, $1 \leq j \leq k$, and the objective function to maximise (4.6), subject to the constraint $\sum_{j=1}^{k} x_j \leq 1$. The key to get a tight upper bound is to impose additional constraints on the x_j so that the resulting optimal policies come as close as possible to those implied by a feasible policy.

First, denote by $A_{\{j\}}$ the long-run fraction of time the server is busy if he serves only class j customers and ignores all other classes, $1 \leq j \leq k$. Taking the number of class j customers as the state, we have a birth-and-death process, so it is straightforward to compute

$$A_{\{j\}} = 1 - \left[\sum_{n=0}^{\infty} (\lambda_j)^n \left\{ \prod_{m=1}^{n} (\mu_j + m\theta_j) \right\}^{-1} \right]^{-1}, 1 \leq j \leq k.$$

We can *add* $x_j \leq A_{\{j\}}$ *as a constraint* in the aforementioned linear program, $1 \leq j \leq k$, or a total of k constraints.

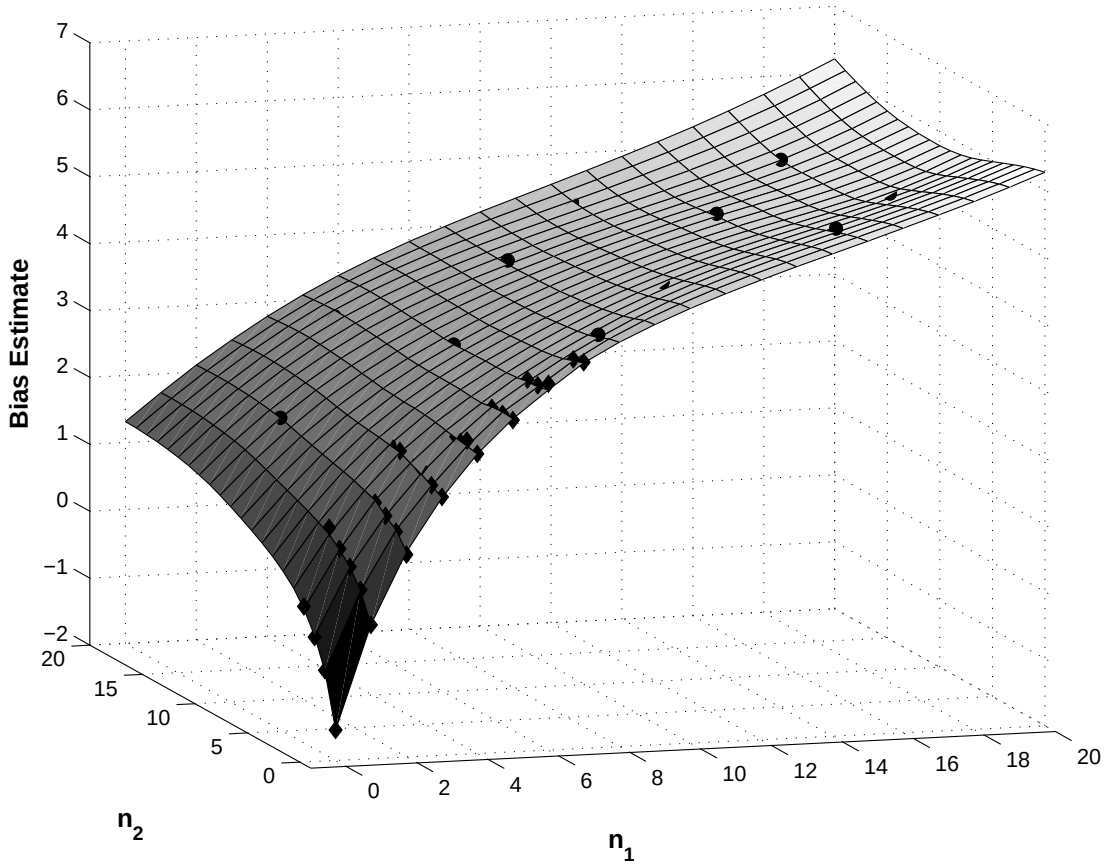

Figure 4.2: Illustration of the interpolation stage of the API algorithm in Example 4.1.

To extend this idea, for $T \subseteq \{1, \ldots, k\}$, we can *add a constraint* $\sum_{j \in T} x_i \leq A_T$, where A_T denotes the maximal long-run fraction of time that the server serves customer classes in T by *ignoring all other classes*. To compute A_T, consider the same pure-reward MDP model in Section 3.1 with customer class set T, and substitute $R_j = \mu_j^{-1}$, $j \in T$, so that the long-run reward rate becomes equivalent to the long-run fraction of time that the server is busy. Using DP value iteration to compute the optimal solution when $|T| = 2$ or $|T| = 3$ is computationally viable, resulting in $\binom{k}{2} + \binom{k}{3}$ *additional constraints*.

Computing A_T when $|T| \geq 4$ is computationally infeasible, but we can still impose constraints derived from relaxed systems. To do so, we create a single *fictitious class* by aggregation and relaxation of the customer classes in T. Denote the arrival, service, and abandonment rates of this fictitious class by $\lambda = \sum_{j \in T} \lambda_j$, $\mu = \min_{j \in T}\{\mu_j\}$, and $\theta = \min_{j \in T}\{\theta_j\}$, respectively. Since the server can only be

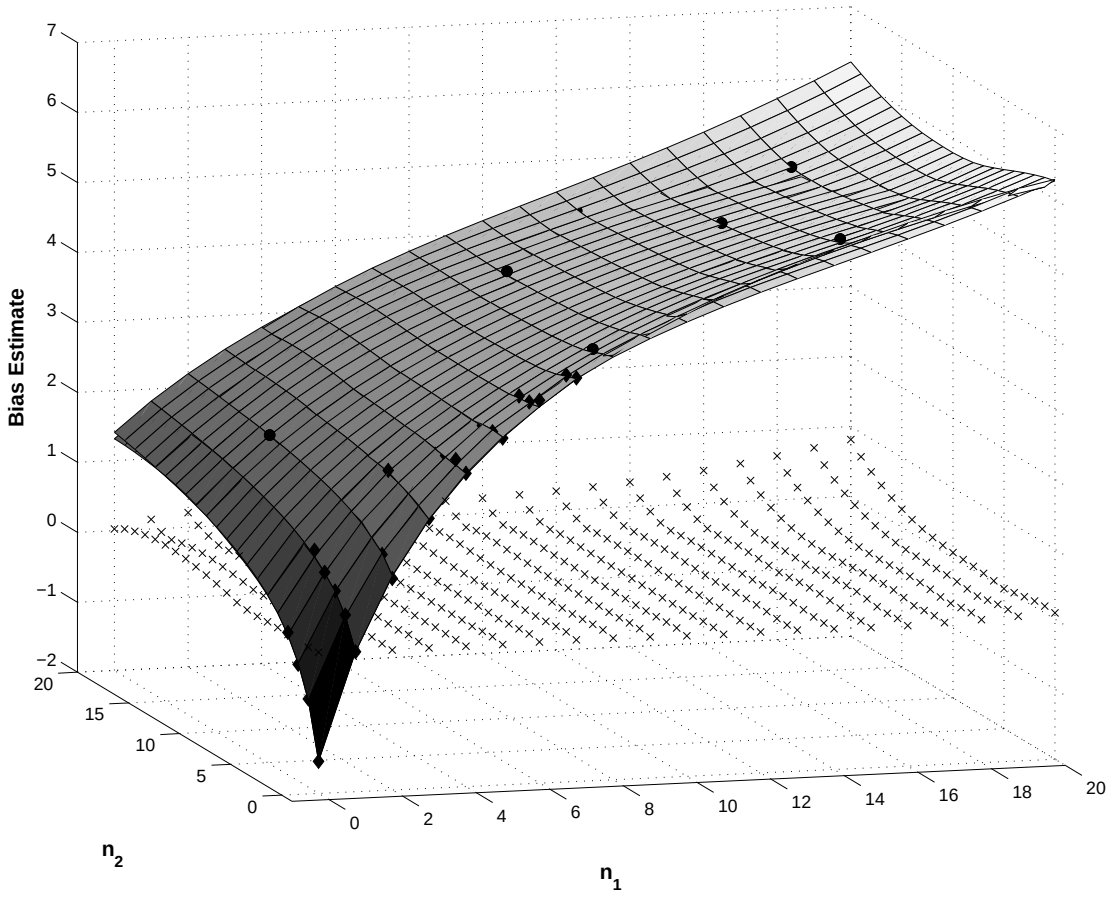

Figure 4.3: Interpolated and exact biases and their absolute difference (crosses) in Example 4.1.

busier with this fictitious class, the long-run fraction of time that the server is busy in this relaxed system is a legitimate upper bound for $\sum_{j \in T} x_i$.

Taking this idea further, we could improve this upper bound by formulating a *number of two-class MDPs*. Divide customer classes in T into two groups and aggregate the classes in each group into a fictitious class, as before. We then use DP value iteration to compute the maximal fraction of time that the server is busy dealing with these two fictitious classes. We do this for every way in which the customer classes in T can be divided into two groups. For example, when $|T| = 4$ there are $\binom{|T|}{1}$ combinations which take one class in the first group with the remaining classes in the other group. There are $\binom{|T|}{2}$ combinations which take two of the classes in the first group with the remaining classes in the other group. Hence we solve $\binom{|T|}{1} + \binom{|T|}{2}$ two-class MDPs. When $|T| \geq 4$, we write B_T for

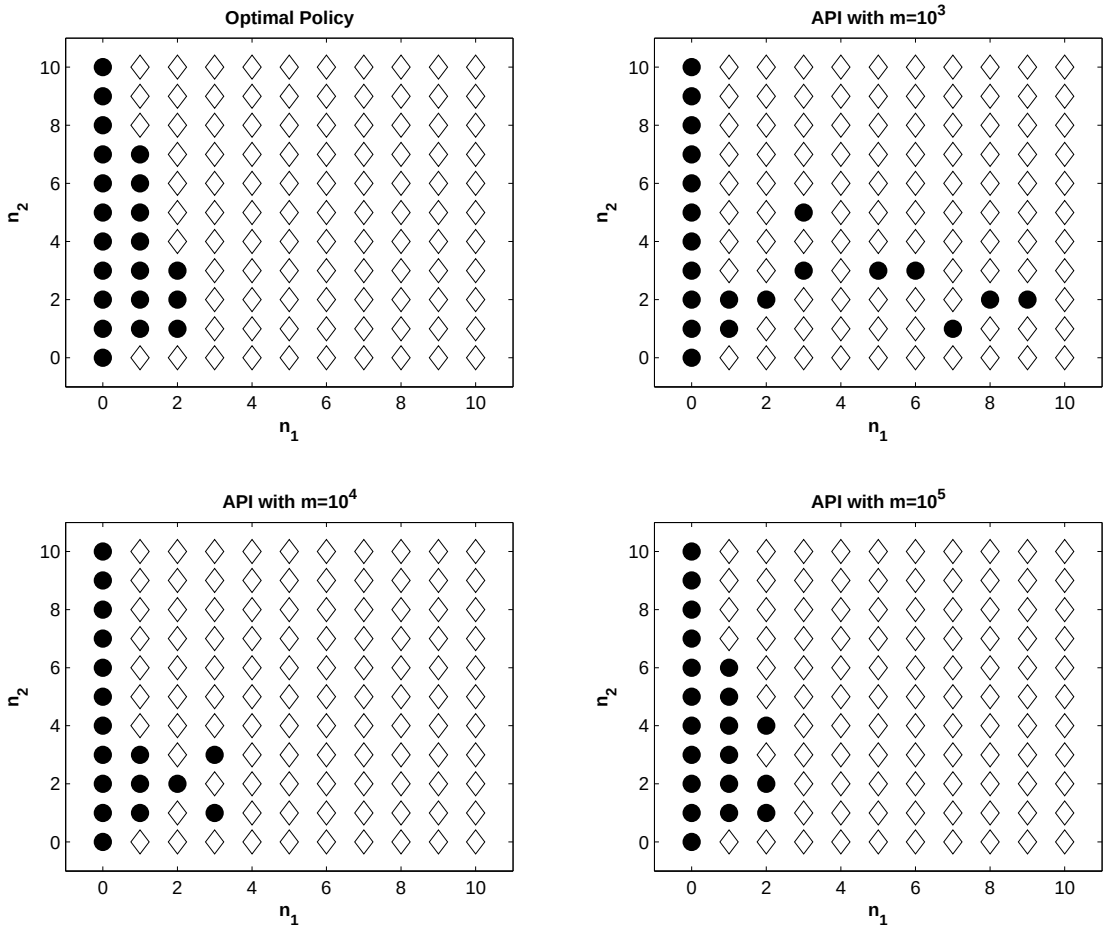

Figure 4.4: Class 1 (diamonds) and class 2 (circles) actions in each state under various policies in Example 4.1.

the tightest upper bound for $\sum_{j\in T} x_j$ derived with this method, and *add it as one constraint*. Although it is possible to divide T into three or more groups, the marginal benefit is outweighed by the increased computational burden.

To formulate a linear program to compute an upper bound for the optimal long-run reward rate, define $\mathcal{S}_{k'} \equiv \{T \subseteq \{1,\ldots,k\} : |T| = k'\}$, which is the set of all subsets of $\{1,...,k\}$ whose cardinality is k'. This linear program is thus given by

$$\begin{aligned}
\max \quad & \sum_{j=1}^{k} R_j \mu_j x_j \\
\text{subject to} \quad & \sum_{j=1}^{k} x_j \leq 1, \quad x_j \geq 0, \quad 1 \leq j \leq k; \\
& \sum_{j\in T} x_i \leq A_T \quad \text{for all} \quad T \in \mathcal{S}_{k'}, \quad k' = 1,2,3; \\
& \sum_{j\in T} x_i \leq B_T \quad \text{for all} \quad T \in \mathcal{S}_{k'}, \quad 4 \leq k' \leq k.
\end{aligned}$$

We would expect the upper bound to come close to the optimal long-run reward rate in smaller systems $k \leq 3$, mainly because of the optimised upper bounds A_T. The upper bounds B_T in subsystems of size $k' > 3$ will worsen as k' increases, due to a greater relaxation when creating more fictitious customer classes. Consequently, the quality of the upper bound tends to degrade as the size of the system k increases.

4.4 Numerical Study

In this section, we conduct extensive numerical experiments to assess the impact and design of our API method, as well as the performance of a range of heuristics which includes our suite of priority policies from Chapter 3. Section 4.4.1 uses a numerical study based on cases with two customer classes to explore design choices for our API heuristics. We assess, *inter alia*, the relative performance of the candidate initialising priority rules $R\mu$ and $R\mu\theta$ as well as testing different choices

of parameters for our API method. This test yields a *recommended API policy* which we denote *rAPI*. Using numerical studies based on cases with three and five customer classes, Section 4.4.2 compares the performance of rAPI with that of other heuristics. Section 4.4.3 contains a brief discussion of the computational burden of developing rAPI and the upper bound discussed in Section 4.3.

4.4.1 Selecting Parameters for the API Algorithm

To explore the trade-off between different choices of parameters for our API algorithm, we test the algorithm on systems with $k = 2$ customer classes. Problems were randomly generated to reflect a wide range of conditions with regard to (1) the length of customer lifetimes in relation to service times (reflected in the categorisation A, B, C in (4.7c)–(4.7e) below); and (2) the traffic intensity or workload in the corresponding system without abandonments. There are three categories of traffic—namely light, moderate, and heavy—as determined by the value of $\rho = \sum_{j=1}^{k} \lambda_j/\mu_j$; see (4.7f)–(4.7h) below. For all nine combinations of A, B, C with the traffic categorisation light, moderate, heavy, 500 problems were generated at random. Parameters were sampled as follows:

$$\mu_j \sim U[0.2, 5] \qquad \text{(all cases);} \tag{4.7a}$$

$$\lambda_j \sim U[0.2, 5] \qquad \text{(all cases);} \tag{4.7b}$$

$$\theta_j^{-1}\mu_j|\mu_j \sim U[0.5, 2] \qquad \text{(short lifetimes, A);} \tag{4.7c}$$

$$\theta_j^{-1}\mu_j|\mu_j \sim U[5, 10] \qquad \text{(moderate lifetimes, B);} \tag{4.7d}$$

$$\theta_j^{-1}\mu_j|\mu_j \sim U[20, 200] \qquad \text{(long lifetimes, C);} \tag{4.7e}$$

$$\rho \in [0.6, 0.8] \qquad \text{(light traffic);} \tag{4.7f}$$

$$\rho \in [0.9, 1.1] \qquad \text{(moderate traffic);} \tag{4.7g}$$

$$\rho \in [1.2, 1.4] \qquad \text{(heavy traffic);} \tag{4.7h}$$

In the parameter generation, μ_j and λ_j were sampled according to (4.7a) and (4.7b) by means of a rejection algorithm until a desired ρ condition (4.7f)–(4.7h)

was met. An additional rejection step ensured that the $R\mu\theta$ and $R\mu$ rules of each parameter set were distinct; otherwise, all parameters were resampled. In all cases, rewards were sampled as follows: $R_2 \sim U[1,3]$ and $R_1 R_2^{-1} | R_2 \sim U[1.25, 2]$. To compute the optimal policy, we use DP value iteration by *truncating* the state space at $N_j = 40$ for each class j with case A, and $N_j = 60$ with cases B and C, as discussed in Section 3.1.

Tables 4.1 to 4.3 report the numerical results with $k = 2$ customer classes. Firstly, Table 4.1 explores the trade-off between parameters n and m within the API algorithm, representing the number of selected states for simulation and the number of simulation replications at each selected state respectively. We considered API variations with the $R\mu\theta$ rule as the initial policy, $t = 1$ iteration, and $r = \frac{32}{45}$. We considered all combinations of $n \in \{15, 45, 75\}$ with $m \in \{m_0, m_1, m_2, m_3\}$ where $m_0 = 10^2, m_1 = 10^3, m_2 = 10^4$, and $m_3 = 10^5$. It is important to point out that the results here state the percentage suboptimality of the policy derived from the resulting API step and *do not* include the initial policy. This allows us to observe the underlying impact of different design choices within the API step, free from any additional effects of the initial policy. The results indicate that in designing the algorithm to effectively manage a fixed computational resource, increasing m brings larger performance improvements than increasing n. However, n must be sufficiently large (for example $n = 45$), as suggested by case {B, Heavy}, whilst increasing this parameter beyond this level does not yield much improvement.

Table 4.2 explores the trade-off between parameters t and m within the API algorithm, where t represents the number of iterations of the algorithm. We considered API variations with the $R\mu\theta$ rule as the initial policy, $n = 45$ selected states, and $r = \frac{32}{45}$. These variations are designed to provide fair comparisons between choices of t and m which reflect multiple, less detailed iterations and a single, more detailed iteration, both having the same level of computational effort. We denote the variations by (t, m) where $t \in \{1, 2, 3\}$ and $m \in \{m_1, m_2, m_3\}$ with

		Workload												
		Light				Moderate				Heavy				
Case	n	m_0	m_1	m_2	m_3	m_0	m_1	m_2	m_3	m_0	m_1	m_2	m_3	
A		0.29	0.13	0.00	0.00	0.32	0.15	0.01	0.00	0.64	0.18	0.04	0.01	90th
	15	0.11	0.04	0.00	0.00	0.14	0.04	0.00	0.00	0.39	0.06	0.01	0.00	75th
		0.02	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.13	0.00	0.00	0.00	Median
		0.28	0.13	0.00	0.00	0.32	0.15	0.01	0.00	0.62	0.18	0.03	0.00	90th
	45	0.11	0.04	0.00	0.00	0.13	0.04	0.00	0.00	0.39	0.06	0.01	0.00	75th
		0.03	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.14	0.01	0.00	0.00	Median
		0.28	0.13	0.00	0.00	0.32	0.15	0.01	0.00	0.62	0.18	0.03	0.00	90th
	75	0.11	0.04	0.00	0.00	0.13	0.04	0.00	0.00	0.39	0.06	0.01	0.00	75th
		0.03	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.14	0.01	0.00	0.00	Median
B		0.72	0.43	0.13	0.09	0.86	0.60	0.36	0.25	3.48	2.39	2.60	2.52	90th
	15	0.36	0.24	0.07	0.04	0.59	0.39	0.22	0.14	1.64	1.44	1.40	1.36	75th
		0.20	0.11	0.03	0.02	0.36	0.21	0.10	0.07	0.68	0.58	0.51	0.49	Median
		0.75	0.40	0.11	0.02	0.99	0.55	0.24	0.05	2.54	1.42	0.67	0.32	90th
	45	0.37	0.23	0.06	0.00	0.65	0.36	0.15	0.03	1.63	0.85	0.38	0.18	75th
		0.20	0.11	0.03	0.00	0.40	0.23	0.07	0.00	0.81	0.47	0.20	0.06	Median
		0.75	0.40	0.11	0.02	1.01	0.57	0.24	0.04	2.41	1.35	0.64	0.20	90th
	75	0.37	0.23	0.06	0.00	0.65	0.36	0.15	0.02	1.41	0.87	0.40	0.10	75th
		0.20	0.11	0.03	0.00	0.40	0.23	0.07	0.00	0.80	0.53	0.20	0.02	Median
C		0.69	0.43	0.24	0.13	0.93	0.77	0.68	0.62	2.37	1.24	0.41	0.41	90th
	15	0.42	0.24	0.15	0.09	0.54	0.44	0.42	0.36	0.90	0.17	0.01	0.01	75th
		0.15	0.09	0.06	0.05	0.23	0.20	0.17	0.16	0.02	0.00	0.00	0.00	Median
		0.88	0.58	0.24	0.09	1.42	0.97	0.56	0.36	3.09	1.53	0.71	0.30	90th
	45	0.47	0.37	0.15	0.06	0.74	0.63	0.38	0.20	2.02	0.79	0.14	0.06	75th
		0.18	0.17	0.08	0.03	0.35	0.29	0.21	0.12	0.90	0.08	0.00	0.00	Median
		0.88	0.59	0.25	0.10	1.37	1.03	0.62	0.29	3.38	2.02	0.60	0.30	90th
	75	0.48	0.37	0.16	0.07	0.79	0.66	0.42	0.20	1.89	0.68	0.13	0.06	75th
		0.19	0.16	0.09	0.04	0.38	0.33	0.23	0.13	0.85	0.10	0.02	0.00	Median

Table 4.1: Percentage suboptimalities in $k = 2$ class systems of various traffic and abandonment level combinations, exploring the trade-off between parameters n and m in the API algorithm. Note, the initial policy is not included in the reported API results shown here. Median, 75th, and 90th percentiles are shown.

$m_1 = 10^3, m_2 = 10^4$, and $m_3 = 10^5$. The results here report the performance of the best performing policy from t iterations of the algorithm and so is consistent with our definition of the API method. From the results there appears to be a degree of indifference in performance between multiple, less detailed iterations and a single, more detailed iteration. In designing the algorithm there is a direct choice between larger t with smaller m and larger m with $t = 1$, without any performance lost in this choice. Given that a *single iteration is simpler to implement*, this could be the basis upon which to make this choice.

Table 4.3 reports the performance of a range of heuristic policies. In comparing the $R\mu\theta$ and $R\mu$ rules, please recall that we use the descriptors 'light', 'moderate', and 'heavy' as a shorthand for ranges of the traffic intensity ρ. The actual volume of traffic in the system will also be strongly influenced by the abandonment rate θ, with case C (small θ) yielding higher volumes than case A (large θ). Hence, while the $R\mu\theta$ rule performs very well in the case {C, light}, as is consistent with Theorem 1 in Chapter 3, it performs poorly in the heaviest traffic case of all, namely {C, heavy}. Its performance under A is less variable than under C, as larger abandonment rates act as a moderator on traffic levels, though it still performs best under A when ρ is small. The $R\mu\theta$ rule clearly outperforms the $R\mu$ rule when ρ is small and θ not too large, while $R\mu$ is the better policy when ρ is large, increasingly so as θ declines in value and the traffic levels increase. The $R\mu$ rule performs very well in the case {C, heavy} which is consistent with the work of Atar et al. (2010), Ayesta et al. (2011), Verloop (2014), and Larrañaga et al. (2014). It is worth noting that at least one of these two priority rules delivers a median performance less than 1% suboptimal across all cases, so they complement each other well.

Table 4.3 also reports the performance of the policy *PI-*$R\mu\theta$, which is derived from exact application of a single PI step to the $R\mu\theta$ rule. The fact that the PI-$R\mu\theta$ is nearly optimal shows promise of the API method, if the bias function can be approximated satisfactorily. The policy *GADL*—due to Glazebrook et al. (2004),

			API						
Case	Workload		$(1, 3m_1)$	$(3, m_1)$	$(1, 3m_2)$	$(3, m_2)$	$(2, m_3/2)$	$(3, m_3/3)$	$(1, m_3)$
A	Light	90th	0.10	0.08	0.00	0.00	0.00	0.00	0.00
		75th	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		Median	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Moderate	90th	0.02	0.05	0.00	0.00	0.00	0.00	0.00
		75th	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		Median	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Heavy	90th	0.03	0.06	0.00	0.01	0.00	0.00	0.00
		75th	0.00	0.01	0.00	0.00	0.00	0.00	0.00
		Median	0.00	0.00	0.00	0.00	0.00	0.00	0.00
B	Light	90th	0.10	0.10	0.04	0.03	0.01	0.02	0.02
		75th	0.04	0.05	0.01	0.01	0.00	0.00	0.00
		Median	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Moderate	90th	0.27	0.30	0.10	0.08	0.04	0.04	0.04
		75th	0.16	0.18	0.04	0.04	0.01	0.02	0.02
		Median	0.06	0.08	0.00	0.01	0.00	0.00	0.00
	Heavy	90th	1.07	0.77	0.43	0.32	0.18	0.18	0.31
		75th	0.61	0.54	0.23	0.17	0.09	0.09	0.17
		Median	0.29	0.32	0.06	0.07	0.01	0.01	0.02
C	Light	90th	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		75th	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		Median	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Moderate	90th	0.40	0.24	0.29	0.21	0.22	0.18	0.27
		75th	0.15	0.11	0.11	0.10	0.10	0.09	0.11
		Median	0.03	0.03	0.03	0.02	0.03	0.02	0.03
	Heavy	90th	1.00	0.80	0.33	0.19	0.15	0.11	0.29
		75th	0.21	0.33	0.09	0.06	0.03	0.02	0.06
		Median	0.01	0.04	0.00	0.00	0.00	0.00	0.00

Table 4.2: Percentage suboptimalities in $k = 2$ class systems of various traffic and abandonment level combinations, exploring the trade-off between parameters t and m in the API algorithm. Median, 75th, and 90th percentiles are shown.

							API								
Case	Workload		$R\mu\theta$	$R\mu$	PI-$R\mu\theta$	GADL	$(1, m_1)$	$(2, m_1)$	$(3, m_1)$	$(1, m_2)$	$(2, m_2)$	$(3, m_2)$	$(1, m_3)$	rAPI	UB
A	Light	90th	1.07	1.17	0.00	0.80	0.12	0.10	0.08	0.00	0.00	0.00	0.00	0.00	1.30
		75th	0.59	0.45	0.00	0.38	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.95
		Median	0.17	0.00	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.65
	Moderate	90th	1.69	1.29	0.00	1.08	0.13	0.07	0.05	0.00	0.00	0.00	0.00	0.00	1.89
		75th	1.03	0.38	0.00	0.52	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.34
		Median	0.36	0.00	0.00	0.12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.90
	Heavy	90th	3.64	1.39	0.00	1.61	0.12	0.07	0.06	0.02	0.01	0.01	0.00	0.00	2.41
		75th	2.01	0.35	0.00	0.90	0.02	0.01	0.01	0.00	0.00	0.00	0.00	0.00	1.77
		Median	0.76	0.00	0.00	0.27	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.13
B	Light	90th	0.52	1.51	0.00	0.79	0.15	0.11	0.10	0.07	0.04	0.03	0.02	0.01	1.37
		75th	0.21	0.82	0.00	0.39	0.06	0.05	0.05	0.03	0.02	0.01	0.00	0.00	1.00
		Median	0.00	0.25	0.00	0.19	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.65
	Moderate	90th	1.70	1.81	0.00	0.99	0.40	0.32	0.30	0.18	0.10	0.08	0.04	0.03	2.17
		75th	0.93	0.67	0.00	0.67	0.27	0.20	0.18	0.10	0.06	0.04	0.02	0.00	1.53
		Median	0.28	0.01	0.00	0.35	0.10	0.09	0.08	0.03	0.01	0.01	0.00	0.00	0.97
	Heavy	90th	6.16	1.10	0.01	1.97	1.41	0.88	0.77	0.65	0.36	0.32	0.31	0.05	2.23
		75th	3.75	0.11	0.00	1.41	0.81	0.59	0.54	0.36	0.21	0.17	0.17	0.00	1.67
		Median	1.71	0.00	0.00	0.77	0.40	0.35	0.32	0.14	0.09	0.07	0.02	0.00	1.11
C	Light	90th	0.00	1.79	0.00	0.54	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.45
		75th	0.00	0.96	0.00	0.34	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.31
		Median	0.00	0.41	0.00	0.16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.18
	Moderate	90th	0.97	2.92	0.03	1.09	0.34	0.28	0.24	0.33	0.24	0.21	0.27	0.09	1.75
		75th	0.33	1.71	0.01	0.62	0.14	0.12	0.11	0.13	0.11	0.10	0.11	0.04	1.25
		Median	0.04	0.62	0.00	0.33	0.03	0.03	0.03	0.03	0.03	0.02	0.03	0.00	0.79
	Heavy	90th	16.48	0.01	0.00	2.16	1.53	0.96	0.80	0.69	0.25	0.19	0.29	0.01	0.26
		75th	11.57	0.00	0.00	1.65	0.79	0.40	0.33	0.14	0.07	0.06	0.06	0.00	0.04
		Median	7.38	0.00	0.00	1.08	0.08	0.06	0.04	0.00	0.00	0.00	0.00	0.00	0.00

Table 4.3: Percentage suboptimalities in $k = 2$ class systems of various traffic and abandonment level combinations. In the last column, we report the percentage of the upper bound above the optimal policy. Median, 75th, and 90th percentiles are shown.

with GADL referring to the paper's four coauthors—requires much computational effort but is typically not as good as the better between the $R\mu\theta$ and $R\mu$ rules.

The next seven columns in Table 4.3 further explore the trade-off between different choices of parameters of the API$(R\mu\theta, n, m, r, t)$ proposed in Section 4.2. We use $n = 45$ selected states throughout, with $r = \frac{32}{45}$. As before, we denote the variations of API by (t, m) where $t \in \{1, 2, 3\}$ and $m \in \{m_1, m_2, m_3\}$ with $m_1 = 10^3, m_2 = 10^4$, and $m_3 = 10^5$. As one would expect, increasing m and t improves performance whilst increasing computational effort. For a given level of computational effort, the strong performance of PI-$R\mu\theta$ suggests that the policy API(π, n, m, r, t) may perform better with a single, more detailed iteration. As discussed, our results in Table 4.2 reveal a degree of indifference in performance between multiple, less detailed iterations and a single, more detailed iteration. Further, and unsurprisingly, a strongly performing initial policy π improves performance. This is shown in the improvement made by the inclusion of the initial policy when compared to the policy derived from the API step in the relevant parts of Table 4.1. Based on these observations, to choose parameters in API(π, n, m, r, t) for general k class systems, we recommend $t = 1$, a large value of m (10^5, say) and allow n, the number of selected states, to scale roughly linearly with k, so that $20k \leq n \leq 25k$. Although *we have not explored the impact of the parameter* r, we use a proportion roughly equal to 0.7. To choose the initial policy π, we first run the pairwise-swapping mechanism in Section 3.6 on the $R\mu\theta$ rule, and on the $R\mu$ rule, separately, and it turns out that in all numerical tests in this section, the final orderings are the same, which we label *PaS*. The initial policy π in the API method is thus set to be the *best performing among* $R\mu\theta$*,* $R\mu$*, and PaS*. Note that in our systems with $k = 2$ customer classes, PaS is simply the better performing policy between $R\mu\theta$ and $R\mu$. We shall denote our recommended API policy by rAPI. As seen in Table 4.3, the rAPI is nearly optimal in all cases.

4.4.2 Comparing the rAPI and Other Heuristics

This section compares the rAPI and other heuristics in systems with $k = 3$ and 5 customer classes. Problem parameters were again generated according to (4.7a)–(4.7h), along with suitable rejection algorithms. We now use $R_j \sim U[1, 4]$ for sampled rewards. For each lifetime/traffic combination, 100 problems were generated at random.

Table 4.4 reports the performance of various service policies against the optimal solution for systems with $k = 3$ customer classes. The rAPI was constructed with $t = 1$, $m = 10^5$, $n = 75$, and $r = \frac{52}{75}$. As seen in the table, the rAPI delivers near-optimal performance in all cases, which reaffirms the strength of policies based on a single, well-estimated (but nonetheless approximate) PI step applied to a well-chosen priority policy. Table 4.4 also shows that a naive heuristic that always *serves the longest queue (labeled SLQ) can perform poorly.* This indicates the merit in searching for strongly performing heuristic policies. In considering our suite of priority policies from Chapter 3, the performance of the $R\mu\theta$ and $R\mu$ rules is much the same as observed in the $k = 2$ customer class study results in Table 4.3. In every case other than {C, Heavy}, PaS improves on both $R\mu\theta$ and $R\mu$ and is near-optimal. In the {C, Heavy} case where it is not, the $R\mu$ rule is strong and rAPI is near-optimal. In all cases rAPI delivers a *marginal improvement* on PaS. The quality of the upper bound for $k = 3$ customer classes is similar to that for $k = 2$ customer classes.

Table 4.5 reports the performance of various service policies against an upper bound on the optimal solution, as discussed in Section 4.3, for systems with $k = 5$ customer classes. Since value iteration is not computationally feasible, the gain of each heuristic is estimated as the mean of 1000 Monte Carlo realisations, which is then compared with the upper bound presented in Section 4.3. The policy rAPI was constructed with $t = 1$, $m = 10^5$, $n = 100$, and $r = \frac{69}{100}$. As seen in Table 4.5, the relative quality among $R\mu\theta$, $R\mu$, PaS, and rAPI, is consistent with that in Table 4.4. The PaS typically improves $R\mu\theta$ and $R\mu$, and then the rAPI further

Case	Workload		$R\mu\theta$	$R\mu$	PaS	SLQ	rAPI	UB
A	Light	90th	0.76	0.46	0.01	5.54	0.00	1.03
		75th	0.23	0.23	0.00	3.29	0.00	0.70
		Median	0.03	0.07	0.00	1.20	0.00	0.34
	Moderate	90th	1.30	0.75	0.01	6.64	0.00	1.25
		75th	0.62	0.39	0.00	3.25	0.00	0.94
		Median	0.11	0.03	0.00	1.50	0.00	0.51
	Heavy	90th	1.32	0.86	0.02	8.52	0.00	1.54
		75th	0.62	0.32	0.00	5.28	0.00	0.99
		Median	0.11	0.02	0.00	2.41	0.00	0.61
B	Light	90th	0.26	0.69	0.04	3.39	0.01	1.15
		75th	0.07	0.24	0.01	2.32	0.00	0.76
		Median	0.01	0.09	0.00	1.43	0.00	0.45
	Moderate	90th	0.85	0.89	0.08	6.03	0.02	1.58
		75th	0.38	0.30	0.01	4.27	0.00	0.92
		Median	0.10	0.05	0.00	2.58	0.00	0.55
	Heavy	90th	1.52	0.86	0.16	10.10	0.03	1.65
		75th	0.84	0.32	0.04	7.15	0.01	1.03
		Median	0.13	0.02	0.00	3.70	0.00	0.58
C	Light	90th	0.01	0.97	0.00	1.44	0.00	0.63
		75th	0.00	0.50	0.00	0.90	0.00	0.29
		Median	0.00	0.21	0.00	0.50	0.00	0.14
	Moderate	90th	0.67	1.52	0.29	4.76	0.10	1.51
		75th	0.24	0.70	0.07	3.28	0.02	0.93
		Median	0.02	0.22	0.00	1.88	0.00	0.56
	Heavy	90th	6.62	0.45	3.02	13.48	0.09	0.80
		75th	2.78	0.13	1.03	8.83	0.01	0.38
		Median	0.76	0.00	0.09	5.23	0.00	0.17

Table 4.4: Percentage suboptimalities in $k = 3$ class systems of various traffic and abandonment level combinations. In the last column, we report the percentage above the optimal policy of the upper bound. Median, 75th, and 90th percentiles are shown.

Case	Workload		$R\mu\theta$	$R\mu$	PaS	SLQ	rAPI
A	Light	90th	3.86	4.05	3.76	7.95	3.72
		75th	3.40	3.43	3.24	6.28	3.23
		Median	2.88	2.89	2.73	5.46	2.73
	Moderate	90th	5.47	4.91	4.91	11.63	4.89
		75th	4.08	3.92	3.86	9.86	3.86
		Median	3.32	3.19	3.13	7.38	3.13
	Heavy	90th	6.00	5.86	5.52	13.39	5.52
		75th	5.16	4.97	4.80	10.60	4.80
		Median	4.38	4.05	3.94	8.79	3.94
B	Light	90th	3.76	3.97	3.74	7.85	3.74
		75th	3.25	3.51	3.25	6.52	3.25
		Median	2.74	2.88	2.70	5.36	2.70
	Moderate	90th	5.76	5.96	5.73	13.30	5.73
		75th	4.77	5.15	4.71	11.34	4.67
		Median	3.40	3.41	3.37	9.30	3.37
	Heavy	90th	6.45	6.29	6.08	17.56	6.07
		75th	4.86	4.94	4.75	14.64	4.74
		Median	3.87	3.82	3.78	11.66	3.64
C	Light	90th	1.00	1.55	1.00	2.85	1.00
		75th	0.80	1.23	0.80	2.32	0.80
		Median	0.59	0.77	0.59	1.81	0.59
	Moderate	90th	3.85	4.09	3.85	9.21	3.65
		75th	2.53	3.29	2.53	7.23	2.51
		Median	2.03	2.25	2.03	5.82	2.02
	Heavy	90th	4.73	1.77	4.40	19.11	1.51
		75th	2.92	1.01	2.24	15.09	0.88
		Median	1.20	0.48	1.04	11.03	0.41

Table 4.5: Percentage below the upper bound in $k = 5$ class systems of various traffic and abandonment level combinations. Median, 75th, and 90th percentiles are shown.

improves the PaS, although the improvement, on average, is rather marginal. The rAPI is the best-performing policy in all cases, and its median performance is *within 4%* of the upper bound derived in Section 4.3. Although it is difficult to judge how the rAPI compares with the optimal policy, the fact that the rAPI is much closer to the optimal value than it is to the upper bound in Tables 4.3 and 4.4 suggests that the figures in Table 4.5 are a *conservative statement* of where the policies stand in relation to the optimal value.

Whereas our numerical experiments in Tables 4.1 to 4.5 show that the suite of priority policies ($R\mu$, $R\mu\theta$, and PaS) generally perform very well and, in several

cases the API method offers only marginal improvement *on average*, it is not always the case. To conclude our numerical study, we offer one example where the improvement of the rAPI method is *substantial*.

EXAMPLE 4.2: Consider a $k = 3$ class example system in which the class parameters (λ_j, μ_j, θ_j, R_j) are given for classes 1, 2, and 3 by (λ, 3, 0.1, 5), ($5\lambda/3$, 5, 1, 2), and ($4\lambda/3$, 4, 5, 1), respectively. With these parameters we have $\lambda = \rho$, and the $R\mu\theta$ rule gives class ordering 321, while the $R\mu$ rule gives 123. As seen in Figure 4.5, the $R\mu\theta$ rule performs well for small ρ, while the $R\mu$ rule performs well for large ρ, which coincides with intuition. For intermediate ρ values, intuition suggests that there should be more service of class 2 since class 1 customers are likely to still be available for service at the completion of a class 2 service and also the instantaneous reward rate of class 2 is larger than that of class 3. Figure 4.5 shows a substantial gap in the range $\rho = 1.4$ to 2.4 between the suite of priority policies ($R\mu$, $R\mu\theta$, and PaS) and the rAPI method (with $t = 1$, $m = 10^5$, $n = 75$). In particular, when $\rho = 1.7$, the $R\mu$ rule is 4.26% suboptimal, the $R\mu\theta$ rule is 9.96% suboptimal, PaS is 5.10% suboptimal, while the rAPI is 0.04% suboptimal. This may be suggestive of greater service of class 2 customers by rAPI. In the random adversary defensive surveillance scenario, even small performance improvements can have a profound impact on safety, hence a large improvement in this example represents an even greater impact.

4.4.3 Computational Time for rAPI and the Upper Bound

Table 4.6 summarises the time needed to compute the rAPI heuristic. Please note that the algorithm was coded in the *C programming language* and carried out on a *High Performance Computing cluster*, with a typical node specification of 2.26Ghz Intel Xeon E5520 processor. Unsurprisingly, the computational burden grows with the number of customer classes k. Recall that the number of selected states n used in the approximate PI step grows roughly linearly in k. Further, as k increases, the balance of computational effort moves toward the sampling stage of the API

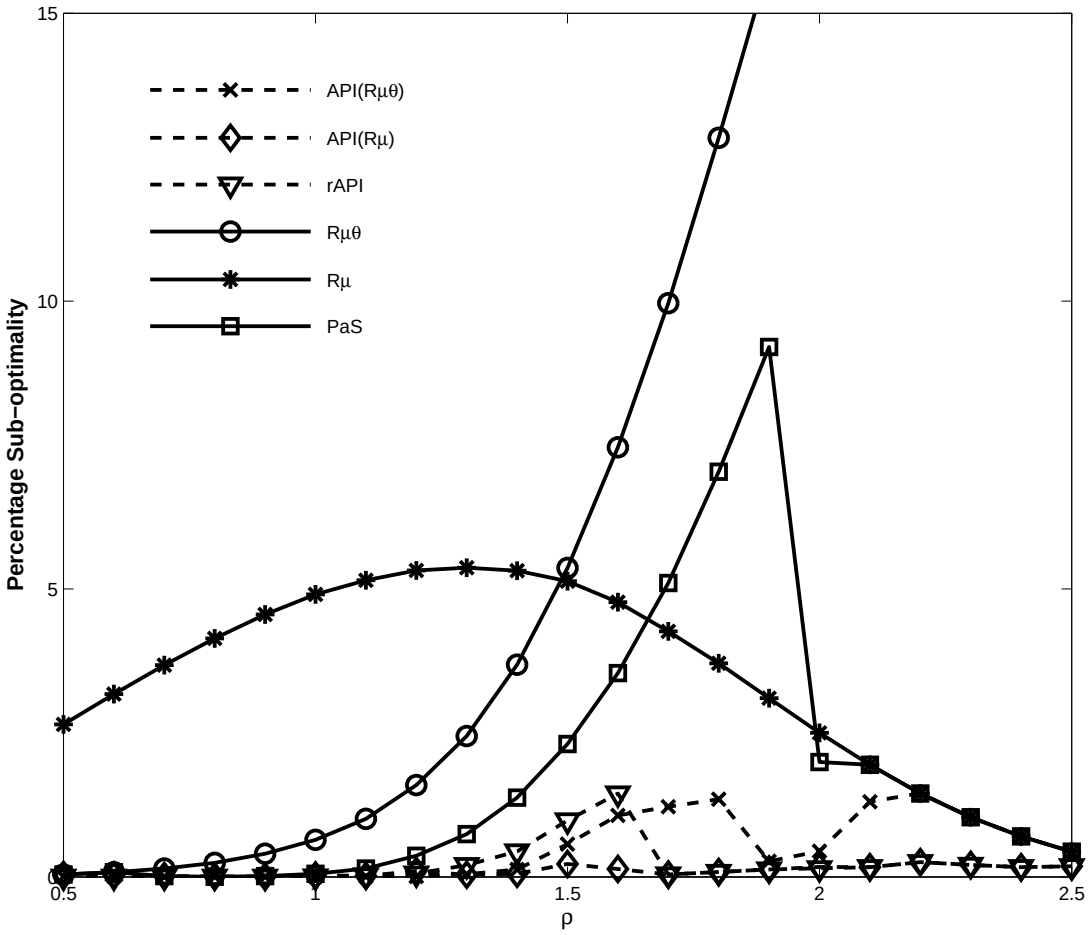

Figure 4.5: Percentage suboptimality for six heuristics in Example 4.2.

algorithm and, within the sampling stage, toward the estimation of bias at the support states. These trends particularly reflect the nature of the growth in the mean times $t^{\pi}(\mathbf{n})$ for a single simulation run during the estimation of the bias $\omega^{\pi}(\mathbf{n})$.

The mean computation times for the upper bound in each problem are in the order of 10 seconds, 400 seconds, and 3000 seconds for systems with 2, 3, and 5 customer classes, respectively. This growth in the computational burden reflects the growth in the number of MDP subproblems that must be solved through DP methods to generate the constraints for the linear program in Section 4.3, when the number of customer classes increases.

Conclusion

In this chapter have further studied the stochastic scheduling problem with customer abandonments in Chapter 3, of which the random adversary surveillance scenario is a special case. The main contribution of the chapter is to develop an approximate policy iteration (API) method for the problem which aims to improve the suite of priority policies in Chapter 3. Our numerical results indicate that, in most cases, the best priority policy from Chapter 3 is nearly optimal in systems with 2 or 3 customer classes and the security team can adopt an effective service policy with a simple structure. In the cases where it is not, the API method invariably tightens up the gap substantially and provides an improved policy, albeit of more complex structure and requiring lots of computation time. Consequently, the suggestion for the security team derived from this chapter is to develop a single, deterministic service policy by following the API method, using the priority policies in Chapter 3 as a starting point. The resulting policy will have a complex structure which can be heavily state dependent, meaning the security team must decide which suspect to screen based on exactly how many suspects are present in each target area. Although this requires lots of computational effort by the security team to develop and analyse such a policy, the potential offered by the API

				Proportion		
k	Case	Workload	Time	Pilot	Anchor	Support
2	A	Light	25	0.07	0.54	0.39
		Moderate	26	0.07	0.54	0.39
		Heavy	29	0.06	0.54	0.39
	B	Light	39	0.05	0.43	0.52
		Moderate	49	0.04	0.44	0.52
		Heavy	73	0.03	0.47	0.50
	C	Light	69	0.03	0.36	0.61
		Moderate	233	0.01	0.45	0.54
		Heavy	291	0.01	0.50	0.48
3	A	Light	76	0.04	0.30	0.67
		Moderate	81	0.03	0.31	0.66
		Heavy	87	0.03	0.31	0.65
	B	Light	127	0.02	0.24	0.74
		Moderate	157	0.02	0.27	0.71
		Heavy	209	0.02	0.31	0.67
	C	Light	249	0.01	0.19	0.80
		Moderate	806	0.01	0.32	0.68
		Heavy	1822	0.00	0.45	0.54
5	A	Light	198	0.02	0.17	0.81
		Moderate	210	0.02	0.18	0.80
		Heavy	224	0.02	0.19	0.79
	B	Light	338	0.01	0.14	0.84
		Moderate	422	0.01	0.18	0.81
		Heavy	559	0.01	0.24	0.75
	C	Light	810	0.01	0.11	0.88
		Moderate	2371	0.00	0.29	0.71
		Heavy	8786	0.00	0.49	0.51

Table 4.6: Mean computation time (secs) needed to generate the rAPI policy in each problem of various k class systems. Also shown are the mean proportions of overall computation time spent on the pilot study, sampling of the anchor set, and sampling of the support set.

method to deliver even small improvements in minimising the expected damage inflicted by the adversary can be of high practical importance in the surveillance setting. Should no improvements be delivered, it is proposed that the security team uses the priority policies in Chapter 3.

In principle, it would be possible to extend the main elements of the API method to more general surveillance problems. The main challenges would arise in the underlying MDP structure of the problem for performing the policy improvement steps and any associated increase in computation time. However, the framework of the API method, namely the simulation and interpolation methodology, presented in this chapter is general and could be adpated to more general problems to provide effective service policies for the security team.

Chapter 5

Defence against Strategic Adversaries Who Choose Where To Attack

In this chapter we consider the defensive surveillance scenario identified in Chapter 1 in which the adversary acts strategically and can choose which queue to attack. This surveillance scenario is an extension of the random adversary scenario discussed in Chapters 3 and 4. Consequently, in this chapter we consider the same mathematical model with an identical stochastic structure to that of the model in the random adversary scenario. Whereas in Chapters 3 and 4, the adversary joined the system according to a fixed probability vector $\mathbf{p}$ which was known to the server, we now consider the case in which $\mathbf{p}$ is unknown to the server. In this case, the server wishes to find a *robust service policy* which achieves a low expected damage regardless of the adversary's choice of $\mathbf{p}$.

The structure of this chapter is as follows. In Section 5.1 we formulate a game-theoretic model for the surveillance problem. In Section 5.2 we discuss how to obtain the optimal solution to the model. We consider the optimal solution from both the *perspective of the server* and the *perspective of the adversary*. In Section 5.3 we develop a heuristic approach for systems where the optimal solution cannot

be computed. Our heuristic approach is based on a matrix game formulation of the model, which utilises insights gained from the method used to compute the adversary's optimal policy and the random adversary scenario studied in Chapters 3 and 4. We conclude the chapter with a numerical study in Section 5.4.

5.1 A Game-Theoretic Model

We consider a setting in which a single server must preemptively serve impatient customers spread across k queues. As in Chapter 3, we may also refer to each queue as a customer class. Different classes of customers arrive according to independent Poisson processes, with the arrival rate being λ_j for class j customers, $1 \leq j \leq k$. The service time for a class j customer is given by the random variable which follows an exponential distribution with rate μ_j. A class j customer, however, will only remain available for service for a random time that follows an exponential distribution with rate θ_j, after which the customer will abandon the system, whether the customer is still waiting in the queue or is already in service. An adversary is a potential customer possessing the ability to join any of the k queues. If the adversary joins queue j then he behaves like every other customer in the queue, carrying an exponential service requirement with rate μ_j and having an exponential lifetime with rate θ_j. The goal of the adversary is to abandon the queue he joins before being served to completion; if this occurs a fixed amount of damage d_j is inflicted.

The adversary decides which of the k queues to join according to a fixed probability vector $\mathbf{p} = (p_1, ..., p_k)$ such that queue j is joined with probability p_j and $\sum_j p_j = 1$. Recall in Chapters 3 and 4, $\mathbf{p}$ was known to the server and the goal was to find a service policy to minimise the expected damage for a given $\mathbf{p}$. In practice, it is likely that there are many situations in which $\mathbf{p}$ *is not known to the server*. If $\mathbf{p}$ is not known, then it is natural to consider a robust formulation in which the server seeks a service policy which achieves a low expected damage regardless of the adversary's choice of $\mathbf{p}$.

The server decides upon a service policy π to use in the system. In Chapters 3 and 4, it was sufficient to only consider deterministic service policies, which in every state defined which queue to serve. We now extend the definition of a service policy π to allow for randomisation in the actions of the server. A *randomised policy* π maps from the state space to a probability vector $\mathbf{q} = (q_1, ..., q_k)$, such that whenever the system enters a state, the server chooses queue j to serve with probability q_j. There are an *infinite number of service policies* the server could use.

Neither the server, nor the adversary know each other's choices: the server does not know which queue the adversary will join and the adversary does not know which service policy is used. However, they do know each other's decision spaces. *We assume that the adversary will join the system while it is in its long-run steady state* under the service policy π. Given that the adversary can be considered as an arbitrary steady state arrival in his chosen queue, the abandonment probability he will experience is the steady state abandonment probability of that queue under the service policy in use π. *Both the server and the adversary possess the ability to hypothetically determine the abandonment probability of the adversary* if he joins queue j and the server uses policy π. They both use this information as the basis upon which to make their decision of which service policy to use and which queue to join respectively.

The objective of the adversary is to maximise the expected damage he can inflict on the system by deciding which queue to attack. The objective of the server, on the other hand, is to determine a randomised service policy which minimises the expected damage inflicted. In the case where each $d_j = 1$, the expected damage inflicted is equivalent to the abandonment probability of the adversary. Given the lack of knowledge of each other's decision, we model the interaction of the server and the adversary as a *simultaneous move two-person zero-sum (TPZS) game.*

In the TPZS game, suppose the server uses policy π and the adversary joins queue j. Recall α_j^π and β_j^π for, respectively, the rate of class j service completions

and abandonments under policy π in steady state. The abandonment probability of the adversary is the same as every other arbitrary class j customer and is equal to β_j^π/λ_j. The expected damage inflicted by the adversary under policy π, conditional on joining queue j, is given by $D_j^\pi = d_j\beta_j^\pi/\lambda_j$. Therefore, for a policy pair (π, j), the payoff to the adversary is given by D_j^π. Now, if the adversary uses probability vector $\mathbf{p}$, the payoff is equal to the expected damage computed using the law of total expectation as $D^\pi(\mathbf{p}) = \sum_j p_j D_j^\pi$. The server wishes to find a randomised service policy which minimises this quantity over all of the adversary's possible choices of $\mathbf{p}$. The adversary wishes to find $\mathbf{p}$ which maximises this quantity over all possible service policy choices of the server. The value of the game V^* represents the optimal expected damage each player can guarantee himself and is given by

$$V^* = \min_\pi \max_{\mathbf{p}} D^\pi(\mathbf{p}) = \max_{\mathbf{p}} \min_\pi D^\pi(\mathbf{p}). \tag{5.1}$$

The optimisation problem in (5.1) is first written from the perspective of the server seeking to find an optimal policy and is second written from the perspective of the adversary seeking to find an optimal probability vector. Our primary interest is solving (5.1) from the *perspective of the server* to determine a performance guarantee for the server regardless of the adversary's choice of $\mathbf{p}$.

5.2 The Optimal Policy

In this section we will develop an optimal solution to (5.1) from the perspective of both the server and the adversary. In both cases, the optimal decision of each player is independent of the optimal decision of the other. For the server, we will show that we can find the optimal randomised policy by formulating the problem as a *linear program* and solving it through standard methods. Although it is not our primary interest, for the adversary, we will show that we can find the optimal probability vector $\mathbf{p}$ by formulating the problem as a *convex optimisation problem* which can be solved through Kelley's cutting plane (KCP) method (see Kelley

(1960)). Considering the problem from the adversary's perspective motivates a heuristic approach to the problem for the server, which will be discussed in Section 5.3.

5.2.1 The Server's Optimal Policy

Firstly, we will consider (5.1) from the perspective of the server. It is possible to formulate (5.1) as a linear program and hence find the optimal randomised service policy and associated value of the game through standard LP methods. The optimal randomised policy of the server is independent of the decision made by the adversary. Before we do this, firstly recall from Chapter 3 the random adversary scenario in which $\mathbf{p}$ is fixed and known to the server. The optimisation problem for the server is given by

$$\min_{\pi} D^{\pi}(\mathbf{p}) \tag{5.2}$$

A deterministic service policy π which is optimal in (5.2) is said to be a *best response* policy to the probability vector $\mathbf{p}$. In Chapter 3 we discussed how this problem was a special case of a more general stochastic scheduling problem which, in principle, could be solved through standard DP methods such as value iteration. It is also possible, in principle, to formulate this problem as a linear program. To formulate a linear program to compute the server's optimal policy π, first recall the MDP model described in Section 3.1, in which we truncate the state space and consider a uniformisation. Denote the truncated set of states by $\mathbf{S}$. Let $x(\mathbf{n}, j)$ denote the steady state proportion of time spent in state $\mathbf{n}$ whilst serving in queue j, according to policy π. The set of $x(\mathbf{n}, j)$ over all states and actions summarises the performance of the system under policy π. There is a one-to-one correspondence between the set of $x(\mathbf{n}, j)$ and the policy π, hence they can be used to define the policy π. The truncation must be large enough such that $x(\mathbf{n}, j)$ is negligible for states outside the truncation. Let $p(\mathbf{n}'|\mathbf{n}, j)$ be the probability of a

transition from state $\mathbf{n}$ to $\mathbf{n}'$ whilst serving in queue j under the uniformisation. Further details regarding the LP formulation of MDPs is given in Puterman (1994). A linear program to compute the optimal policy in (5.2) is then given by:

$$\min_{x} \quad \sum_{j=1}^{k} p_j d_j \left(1 - \frac{\mu_j}{\lambda_j} \sum_{\mathbf{n}\in\mathcal{S}} x(\mathbf{n}, j)\right) \tag{5.3}$$

$$\text{subject to} \quad \sum_{j\in A(\mathbf{n}')} x(\mathbf{n}', j) - \sum_{\mathbf{n}\in\mathcal{S}} \sum_{j\in A(\mathbf{n})} p(\mathbf{n}'|\mathbf{n}, j) x(\mathbf{n}, j) = 0, \quad \mathbf{n}' \in \mathbf{S} \tag{5.4}$$

$$\sum_{\mathbf{n}\in\mathcal{S}} \sum_{j\in A(\mathbf{n})} x(\mathbf{n}, j) = 1 \tag{5.5}$$

$$x(\mathbf{n}, j) \geq 0 \quad \text{for} \quad j \in A(\mathbf{n}) \quad \text{and} \quad \mathbf{n} \in \mathbf{S}. \tag{5.6}$$

In the objective function (5.3), the quantity $\mu_j \sum_{\mathbf{n}} x(\mathbf{n}, j)$ equals the rate of class j *service completions* under policy π, namely α_j^π. Hence, the objective function is equal to $\sum_j p_j d_j (1 - (\alpha_j^\pi / \lambda_j))$, the expected damage $D^\pi(\mathbf{p})$. Constraint (5.4) states that, under any policy, the long-run transition rates into and leaving a state must be the same. This is a necessary condition for the system to be stable under policy π. Constraints (5.5) and (5.6) ensure that $x(\mathbf{n}, j)$ defines a probability distribution and indeed all of the time of the server is fully accounted for. Together, the constraints ensure a *feasible service policy*.

We now extend this to the strategic adversary problem (5.1) from the perspective of the server as a LP problem. Suppose the server uses a fixed service policy π, then the expected damage $D^\pi(\mathbf{p}) = \sum_j p_j D_j^\pi$ under probability vector $\mathbf{p}$ defines a *hyperplane in* $k-1$ *dimensions*. This is equivalent to a straight line for $k = 2$ queues and a plane for $k = 3$ queues. The best response of the adversary, and hence worst case for the server, against this fixed service policy is achieved at one of the *corners of the hyperplane*, that is set $p_j = 1$ for some queue j and set $p_i = 0$ for queue $i \neq j$. Hence, the best response of the adversary and the largest

expected damage the server can achieve by using policy π is given by

$$\max_{\mathbf{p}} D^{\pi}(\mathbf{p}) = \max_{j} D_j^{\pi}.$$

Let the value of the maximum expected damage under π given by this equation be z. This is the largest expected damage the server can guarantee himself by using π regardless of the adversary's choice of $\mathbf{p}$. This provides an upper bound for the optimal expected damage V^*. The objective of the server is then to find a policy which minimises the upper bound z. In other words, the server wishes to find a policy which minimises the expected damage the server can guarantee himself regardless of the adversary's choice of $\mathbf{p}$. We can *modify* the linear program given for the random adversary problem to define a linear program for (5.1) to find the optimal policy of the server as follows:

$$\begin{aligned} \min_{x} \quad & z \\ \text{subject to} \quad & d_j\left(1 - \frac{\mu_j}{\lambda_j}\sum_{\mathbf{n}\in\mathcal{S}} x(\mathbf{n}, j)\right) \leq z, \quad j = 1, ..., k \qquad (5.7) \\ & \text{and contraints (5.4) to (5.6)} . \end{aligned}$$

The k constraints (5.7) ensure that the expected damages in each queue are no greater than z for any choice of $\mathbf{p}$ by the adversary. The constraints (5.4) to (5.6) ensure the server adopts a feasible service policy. Solving this linear program will give the optimal expected damage V^*. There is a *one-to-one correspondence* between the optimal set of variables $x(\mathbf{n}, j)$ and the optimal service policy π. The variables $x(\mathbf{n}, j)$ map to the optimal randomised service policy by defining actions for the server in each state as follows: in state $\mathbf{n}$, the server chooses action j with probability equal to $x(\mathbf{n}, j)/\sum_{i=1}^{k} x(\mathbf{n}, i)$. Further details regarding this correspondence are given in Puterman (1994).

The computations required to solve this linear program quickly become *intractable* even for systems of moderate size. This is since the state space grows

exponentially in the number of queues k. This requires a vast number of variables and constraints within the linear program, which makes solving the linear program computationally infeasible. Consequently there is a need to consider heuristic approaches to (5.1) from the perspective of the server, which are readily computable and have strong performance.

5.2.2 The Adversary's Optimal Policy

We will now consider (5.1) from the perspective of the adversary. The optimal probability vector of the adversary is independent of the service policy used by the server. If the adversary uses probability vector $\mathbf{p}$, the smallest expected damage the adversary can guarantee himself is given by the deterministic best response policy of the server to $\mathbf{p}$. The best response policy is determined by solving the random adversary problem (5.2). Define the following function

$$V(\mathbf{p}) = \min_{\pi} D^{\pi}(\mathbf{p})$$

to be the optimal expected damage as a function of $\mathbf{p}$ in the random adversary problem (5.2). The function $V(\mathbf{p})$ represents the smallest expected damage the adversary can guarantee himself under each choice of $\mathbf{p}$.

In general, for an arbitrary service policy π, the expected damage $D^{\pi}(\mathbf{p})$ as a function of $\mathbf{p}$ defines a hyperplane in $k-1$ dimensions. Subsequently, $V(\mathbf{p})$ can be seen as the *lower envelope of the infinite set of hyperplanes* corresponding to the infinite set of service policies. This implies that $V(\mathbf{p})$ is a concave, continuous function. Moreover, the hyperplanes of best response policies from the random adversary problem (5.2) are tangents to $V(\mathbf{p})$ at their respective points $\mathbf{p}$. This implies that $V(\mathbf{p})$ is a differentiable function.

The adversary wishes to maximise, through his choice of $\mathbf{p}$, the expected damage he can guarantee himself. This is equivalent to maximising the function $V(\mathbf{p})$ over $\mathbf{p}$. Hence, we can formulate the strategic adversary problem (5.1), from the

perspective of the adversary, as a convex optimisation problem as follows:

$$\begin{aligned} \max_{\mathbf{p}} \quad & V(\mathbf{p}) \\ \text{subject to} \quad & \sum_{j=1}^{k} p_j \leq 1 \\ & p_j \geq 0 \quad i = 1, ..., k. \end{aligned} \tag{5.8}$$

The constraint $\sum_j p_j \leq 1$ allows for the case that the adversary may not attack the system at all, but in the optimal solution this constraint will always hold with equality. Solving (5.8) will give an optimal probability vector for the adversary in (5.1) and its objective value will be equal to the optimal expected damage V^*. However, (5.8) cannot be solved directly, as opposed to solving the LP formulation for the server's optimal randomised policy. Let the value of the minimum level of expected damage the adversary can guarantee himself through his choice of $\mathbf{p}$ be given by w. This provides a lower bound for the optimal expected damage V^*. We can equivalently write (5.8) as one of the adversary seeking to maximise this lower bound as follows:

$$\begin{aligned} \max_{\mathbf{p},w} \quad & w \\ \text{subject to} \quad & V(\mathbf{p}) - w \geq 0 \\ & \sum_{j=1}^{k} p_j \leq 1 \\ & p_j \geq 0 \quad i = 1, ..., k. \end{aligned} \tag{5.9}$$

The purpose of reformulating (5.8) as (5.9) is that this formulation can be solved using *Kelley's cutting plane* (KCP) method (see Kelley (1960)), which is an iterative method developed to solve convex optimisation problems. Application of this method to (5.9) will converge to the optimal probability vector $\mathbf{p}^*$ and optimal expected damage V^*. A proof of this result is given in Luenberger & Ye (2008) under these conditions: (1) The objective function is a continuously

differentiable, concave function; (2) The constraints are continuously differentiable, concave functions. Since the optimisation in (5.9) meets these conditions, the KCP method can be used to compute $\mathbf{p}^*$ and V^*.

The application of the method is based on the fact that *every concave function can be approximated by a set of piecewise linear functions* that are tangents to the function at a finite subset of points. The concavity property means that this approximation will lie above the function to be maximised. We can approximate $V(\mathbf{p})$ as the lower envelope of a set of hyperplanes corresponding to the best response policies at a finite subset of probability vector points. Suppose we have a set of m probability vectors $\mathcal{M} = \{\mathbf{p}_1, ..., \mathbf{p}_m\}$. Suppose also we have a set of m service policies $\mathcal{P} = \{\pi_1, ..., \pi_m\}$ such that each element π_t corresponds to the best response policy to the element $\mathbf{p}_t$ in $\mathcal{M}$ for $t = 1, ...m$. From these, suppose we can also obtain their corresponding expected damages $D_j^{\pi_t}$ in each queue, hence also the hyperplanes $D^{\pi_t}(\mathbf{p}) = \sum_{j=1}^{k} p_j D_j^{\pi_t}$ as functions of $\mathbf{p}$. We can *approximate* $V(\mathbf{p})$ as follows:

$$V(\mathbf{p}) \approx \min_t \{D^{\pi_t}(\mathbf{p})\}$$

Using this approximation we can replace (5.9) with the following relaxation:

$$\begin{aligned} \max_{\mathbf{p}, w} \quad & w \\ \text{subject to} \quad & D^{\pi_t}(\mathbf{p}) - w \geq 0 \quad \text{for } 1 \leq t \leq |\mathcal{P}| \\ & \sum_{j=1}^{k} p_j \leq 1 \\ & p_j \geq 0 \quad i = 1, ..., k. \end{aligned} \tag{5.10}$$

Importantly, this relaxation is a linear program, whose optimal solution gives an upper bound for the optimal expected damage V^*. In addition, the expected damages $D^{\pi_t}(\mathbf{p}_t)$ are feasible for the adversary in the original problem, ensuring a minimum level of expected damage and so provide lower bounds to V^*. Since each hyperplane in the approximation is a tangent to $V(\mathbf{p})$ at each respective

point, the approximation will get better as more hyperplanes are added to the approximation. Each time a new hyperplane is added, a constraint is added to the relaxed linear program. Continuation of this process eventually recovers the true function and both the lower and upper bounds approach V^*.

We have described the principle underlying the KCP method without actually describing the application of the method itself. We will now do this. The KCP method iteratively constructs the approximation of $V(\mathbf{p})$, using solutions of the relaxed linear program as the probability vector points at which to improve the approximation. We summarise the algorithm as follows:

1. Set $U = \infty$ and $L = 0$ as upper and lower bounds respectively for V^* and let the sets $\mathcal{M}$ and $\mathcal{P}$ initially be empty.

2. Set iteration counter $t = 1$ and initialise the probability vector of the adversary $\mathbf{p}_t$. Solve the random adversary problem (5.2) for $\mathbf{p}_t$ to obtain the best response policy π_t for the server. Add $\mathbf{p}_t$ to $\mathcal{M}$ and π_t to $\mathcal{P}$.

3. Formulate and solve the linear program (5.10) in which the server is restricted to using the service policies in $\mathcal{P}$. Denote the solution by $(\mathbf{p}_{t+1}, V)$, where $\mathbf{p}_{t+1}$ is the optimal probability vector of the adversary and V is the corresponding objective function value.

4. Update the upper bound $U = V$.

5. If U and L are close enough, then stop.

6. For probability vector $\mathbf{p}_{t+1}$, find the corresponding best response policy π_{t+1} by solving the random adversary problem (5.2). Update the lower bound $L = \max(L, D^{\pi_{t+1}}(\mathbf{p}_{t+1}))$.

7. Add $\mathbf{p}_{t+1}$ to $\mathcal{M}$ and π_{t+1} to $\mathcal{P}$. Increase the iteration counter $t = t + 1$ and go to step 3.

In steps 3 and 4 of the algorithm, (5.10) is a relaxation of (5.9) from the perspective of the adversary and a restriction from the perspective of the server.

Hence, the optimal objective function value provides an upper bound to V^*. This upper bound will be at least as good as the previous upper bound since an extra constraint has been added to (5.10), tightening the relaxation. In step 6 of the algorithm, $\mathbf{p}_{t+1}$ and the corresponding expected damage $D^{\pi_{t+1}}(\mathbf{p}_{t+1})$ is feasible in the original problem (5.9). This means that the adversary can use $\mathbf{p}_{t+1}$ and guarantee an expected damage at least equal to $D^{\pi_{t+1}}(\mathbf{p}_{t+1})$, a lower bound for V^*. It is not the case that the lower bound will necessarily improve, hence we set the lower bound to be the highest known feasible expected damage. The upper and lower bounds of the algorithm eventually converge to V^*. Step 5 is a stopping criterion which terminates the algorithm when the upper and lower bounds become sufficiently close, namely when $|U - L| < \epsilon$ for some small $\epsilon > 0$.

The KCP algorithm is illustrated for $k = 2$ queues in Figure 5.1. The red curve represents $V(\mathbf{p})$, which the adversary wishes to maximise through his choice of $\mathbf{p} = (p, 1-p)$. In the first two iterations of the algorithm, the adversary uses $p = 0$ and $p = 1$ respectively. Hence, the server responds with π_1 and π_2, which are priority policies. These two policies form an initial approximation of $V(\mathbf{p})$, which is iteratively improved in the next two iterations through the addition of π_3 and π_4. The improving upper bounds are shown as blue points whereas feasible lower bounds are shown as green points. After four iterations of the algorithm, the approximation of $V(\mathbf{p})$ is very good and the upper and lower bounds are close to V^*.

The most important part of the KCP method is the ability to repeatedly solve the random adversary problem (5.2) to generate best response policies for the server. Whilst this is possible in principle, in practice it was shown in Chapters 3 and 4 that the random adversary problem is computationally challenging. Best response policies can be found through DP methods for systems of up to $k = 3$ queues, meaning that the KCP method is feasible for systems of this size. However, for larger systems the random adversary problem is computationally intractable, meaning that the KCP method is also computationally intractable for

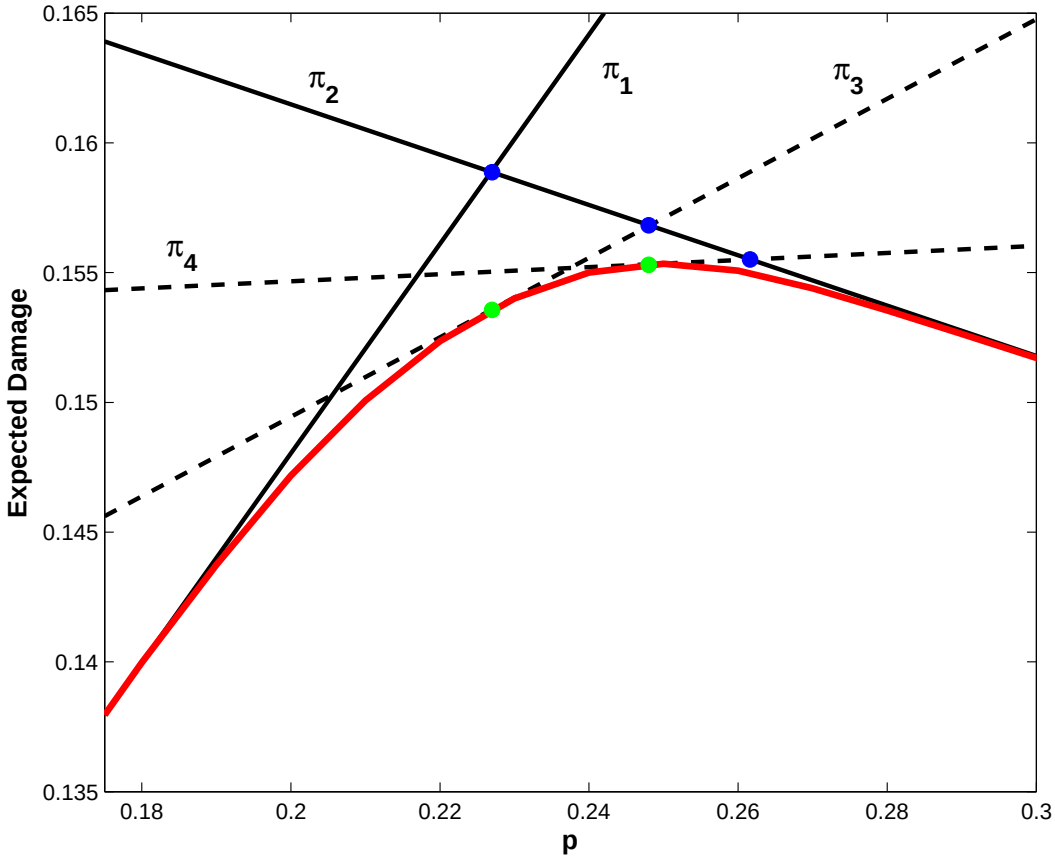

Figure 5.1: Illustration of the KCP algorithm.

larger systems of more than three queues.

In general, *we are not interested in finding the optimal probability vector* of the adversary. Our interest lies in finding the *optimal randomised policy* of the server. In the next section, we will see that the KCP method provides the motivation for a heuristic approach to (5.1) from the server's perspective.

5.3 A Matrix Game Formulation

Many TPZS games can be formulated as matrix games. We can conceptually consider the TPZS game in Section 5.1 as a matrix game. Each row in the matrix game represents a *pure strategy* of the adversary, which corresponds to joining each of the k queues. A *mixed strategy* for the adversary is given by a probability vector over his set of pure strategies. In Section 5.1, we stated that the adversary joined the system according to a fixed probability vector $\mathbf{p} = (p_1, ..., p_k)$, which is equivalent to a mixed strategy of the adversary in the matrix game. Each column in the matrix game represents a pure strategy of the server, which corresponds to the choice of service policy π. Each entry in the matrix game corresponds to the payoff of a pure strategy pair (π, j), namely D_j^π. A mixed strategy for the server is given by a probability vector over his set of pure strategies.

It is impossible to formulate the matrix game equivalent to (5.1) since there are an *infinite number of pure strategies* of the server, hence an infinite number of columns in the matrix. *Our heuristic approach is to formulate a finite version of the matrix game.* In the finite matrix game, the adversary will still have k pure strategies, however, the server will be restricted to a finite number of m pure strategies corresponding to m service policies in a set $\mathcal{P}$. A mixed strategy for the server in the finite matrix game is represented by a probability vector $\mathbf{q} = (q_1, ..., q_m)$ over the policies in $\mathcal{P}$. In practice, the mixed strategy of the server would be implemented by first selecting a policy from $\mathcal{P}$ using $\mathbf{q}$, then using the selected policy thereafter. The heuristic is a restriction of the conceptual infinite version of the matrix game equivalent to (5.1), hence its solution provides an upper

bound to the optimal expected damage V^*.

Our heuristic approach is motivated by the KCP method. In the KCP method we restricted the server to a set of service policies and iteratively populated this set with various deterministic best response policies. The relaxed linear program (5.10) in the KCP method in which the server was restricted to this set of service policies is in fact *equivalent to the LP formulation of a finite matrix game* from the perspective of the adversary. From this observation, we can view the KCP method as an *iteratively expanding* finite matrix game. Hence, in our heuristic approach, we propose to *populate the set of policies $\mathcal{P}$ using the KCP method*. The LP formulation of the finite matrix game from the perspective of the server is given by the dual of (5.10). Solving the dual gives a mixed strategy for the server over the policies in $\mathcal{P}$. In theory, we know from the KCP method that the value of the finite matrix game will converge to the optimal expected damage V^* as more policies are added to $\mathcal{P}$.

The KCP method was used in a search problem by Lin & Singham (2015), in which a searcher wishes to find an object hidden by an adversary. In this work, the KCP method iteratively solves the problem since the sub-problem equivalent to the random adversary problem (5.2) has a simple analytical solution. The KCP method has also been used by McMahan et al. (2003) in a problem of planning a route for a robot through sensors controlled by an adversary. The algorithm in this work is referred to as an application of Benders decomposition. The sub-problem corresponding to (5.2) is an MDP whose optimal solution can be readily computed. In both of these examples, since the sub-problems do not pose any challenges, the algorithms are able to deliver optimal solutions. Lin et al. (2014) consider a graph patrol problem in which a patroller traverses the edges of a graph whose nodes are potential targets for an attacker. The sub-problem equivalent to (5.2) is an MDP problem which is computationally intractable in larger systems. The authors develop an effective algorithm which is a heuristic application of the KCP method in that instead of using best response policies in the MDP sub-problem, strongly

performing heuristics are used.

5.3.1 Heuristic Cutting Plane Method

As we have already noted, solving the random adversary sub-problem (5.2) is computationally intractable for larger systems of more than three queues. Whilst this prohibits the application of the KCP method exactly in our heuristic approach, the method provides a structural framework within which to adjust our heuristic approach. We can adjust our heuristic approach by populating the set of policies $\mathcal{P}$ in the finite matrix game using a heuristic application of the KCP method. We will refer to this as the *heuristic cutting plane* (HCP) method. The HCP method relaxes the necessity to provide best response policies to (5.2) in steps 2 and 5 of the KCP algorithm in return for strongly performing heuristic policies which can be computed. Lin et al. (2014) develop an analogous heuristic method with success.

The effectiveness of this heuristic approach depends critically upon the ability to handle the random adversary problem (5.2). Intuitively, generating near best response policies in (5.2) will enable the HCP method to closely match the characteristics of the KCP method. In Chapters 3 and 4 we studied this problem in detail and identified a set of strongly performing priority policies: the $R\mu\theta$ rule, the $R\mu$ rule, and the PaS policy. Recall, to determine these priority policies, we set the rewards R_j based on the mixed strategy of the adversary, namely $R_j = d_j p_j / \lambda_j$. The best of these priority policies was near-optimal in the majority of problems. We also developed a heuristic policy based on an approximate policy improvement method, which we labelled the API policy. This policy provides the capability to tighten up the suboptimality gap in the relatively few problems where the best priority policy is not near-optimal.

We summarise the HCP algorithm as follows:

1. Let the set $\mathcal{P}$ initially be empty.

2. Set iteration counter $t = 1$ and add an initial priority policy π_t to $\mathcal{P}$.

3. Formulate and solve the TPZS matrix game (5.10) in which the server is restricted to using the service policies in $\mathcal{P}$. Denote the solution to this matrix game by $(\mathbf{p}_{t+1}, V)$.

4. For mixed strategy $\mathbf{p}_{t+1}$, determine the priority policy π_{t+1} as a heuristic policy for the random adversary problem (5.2).

5. If $\pi_{t+1} \in \mathcal{P}$, then skip step 6 and go to step 7.

6. Add π_{t+1} to $\mathcal{P}$. Increase the iteration counter $t = t + 1$ and go to step 3.

7. Formulate and solve the dual of (5.10) to find a mixed strategy $\mathbf{q}$ for the server over the set of policies $\mathcal{P}$. The value of the game V gives the expected damage of this mixed strategy of the server. Stop.

In step 4 of the algorithm, the priority policy response of the server to mixed strategy $\mathbf{p}_{t+1}$ is determined by choosing one of the *$R\mu\theta$ rule, the $R\mu$ rule, and the PaS rule*. A selection rule determines which of these priority policies is used in every iteration of the algorithm, for example use the $R\mu$ rule each time. In steps 2 to 6 of the algorithm, the server populates his policy set $\mathcal{P}$ with priority policies until no new priority policy can be added under the selection rule. The reason for using priority policies here as heuristic responses is due to the observation that the chosen priority rule will be either optimal or near-optimal for many of the mixed strategy choices the server could make, as evidenced in Chapters 3 and 4. This is especially true when a few elements of $\mathbf{p}$ are high, for example p_j close to 1, corresponding to much higher rewards/penalties in some queues compared to others. Priority policies also typically occur naturally as best response policies in the early iterations of the KCP method, when the KCP method is initialised with a priority policy best response. Another advantage is the simplicity of priority policies and the ease with which they can be constructed. The only computational cost associated with this is the *time it takes* to evaluate the payoffs $D_j^{\pi_{t+1}}$ for the matrix game.

In the initialisation of the algorithm in step 2, the initial priority policy could be determined by first selecting a mixed strategy $\mathbf{p}$ with $p_j = 1$ for some queue j and $p_i = 0$ for queue $i \neq j$. Then choose a priority policy response through the selection rule. Whichever policy is chosen will prioritise queue j. Alternatively, we can suppose the server only serves in queue j and ignores all others for $1 \leq j \leq k$. The expected damages D_j can be calculated by considering a birth-death process from which we can compute

$$D_j = d_j \left[1 - \frac{\mu_j}{\lambda_j}\right] \left[1 - \left[\sum_{n=0}^{\infty} (\lambda_j)^n \left\{\prod_{m=1}^{n} (\mu_j + m\theta_j)\right\}^{-1}\right]^{-1}\right], 1 \leq j \leq k.$$

The expected damages D_j are equivalent to the expected damages in queue j under a policy which prioritises that queue. We can determine the initial priority policy by sorting the D_j in descending order and prioritising according to this order. In some sense this gives top priority to the queue in which the adversary can inflict the most damage.

5.3.2 Enhancement to Heuristic Cutting Plane Method

It is possible to *enhance* the HCP method by replacing step 7 with the following three steps, giving the HCP$^+$ method:

7a Formulate and solve the TPZS matrix game (5.10) in which the server is restricted to using the service policies in $\mathcal{P}$. Denote the solution to this matrix game by $(\mathbf{p}_*, V)$.

7b For mixed strategy $\mathbf{p}_*$, construct the API policy π_* (as discussed in Chapter 4) as a heuristic policy for the random adversary problem (5.2). Add π_* to $\mathcal{P}$.

7c Formulate and solve the dual of (5.10) to find a mixed strategy $\mathbf{q}$ for the server over the set of policies $\mathcal{P}$. The value of the game V gives the expected damage of this mixed strategy of the server. Stop.

In the replacement steps 7a and 7b, it is no longer possible to add priority policies to $\mathcal{P}$ under the chosen selection rule. Furthermore, for $\mathbf{p}_*$, there is a suboptimality gap between the expected damages of the best priority policies currently in $\mathcal{P}$ and the optimal expected damage. Consequently, we attempt to improve upon the expected damages of the policies in $\mathcal{P}$ for $\mathbf{p}_*$ by using the API policy as a strongly performing heuristic. Any improvement will improve the value of the matrix game. The construction of the API policy is more computationally intensive, which is why it is only used once in this enhanced algorithm in the place it is likely to have the largest impact on performance. However, due to its approximate nature there is no guarantee of improvement. The *HCP method will be significantly faster than the HCP$^+$ method*. There is a trade-off between any performance improvement offered by the HCP$^+$ method and the increased computation time.

An alternative to the HCP and HCP$^+$ methods is again to populate a policy set $\mathcal{P}$, but not through an iterative procedure. We could populate $\mathcal{P}$ directly with all $k!$ possible priority policies and solve the finite matrix game consisting of these policies. We will refer to this as the *PA method*. The rationale behind the PA method is that $\mathcal{P}$ will give the server policies which provide the best possible defence of each queue, as well as good coverage across the queues. It can be seen as a limiting form of the HCP method, since the priority policies developed during the HCP method will form a subset of the priority policies in the PA method. Consequently, the expected damage under the PA method will be less than or equal to the HCP method. However, the HCP method is likely to be significantly faster than the PA method, especially for systems with more than 3 queues. Comparing the relative performance of these two methods will also give an indication of any impact in the choice of the priority policy selection rule (the $R\mu\theta$, the $R\mu$ rule, or the PaS policy).

5.4 Numerical Study

In this section we conduct a set of numerical experiments to assess the performance of the heuristic approaches to (5.1) developed in Sections 5.3.1 and 5.3.2. We begin by considering an example in a $k = 2$ queue system in order to illustrate the KCP method and the heuristic approaches in more detail. We then consider performance in large sets of randomly generated problems.

EXAMPLE 5.1: Consider the following example system in which there are $k = 2$ queues. The parameters of the system are as follows:

$$(d_1, d_2) = (1, 1)$$

$$(\lambda_j, \mu_j, \theta_j) = \begin{cases} (2, 3, 1) & j = 1 \\ (3, 4, 0.5) & j = 2 \end{cases}$$

The optimal expected damage $V^* = 0.3903$ can be computed by solving the linear program developed for the server's optimal randomised policy in Section 5.2.1. We can also compute V^* by applying the KCP method to the problem. The KCP method iteratively develops a finite matrix game in which the server is restricted to a set of service policies $\mathcal{P}$ containing deterministic best response policies.

Since there are $k = 2$ queues, a mixed strategy of the adversary is given by $\mathbf{p} = (p, 1 - p)$ and the expected damage hyperplane of policy π, namely $D^\pi(\mathbf{p})$, is a straight line function of p. Figure 5.2 shows the first 3 iterations of the KCP method along with both the HCP and HCP$^+$ methods. The red curve shows $V(\mathbf{p})$, which is maximised at $p = 0.441$, at which point $V(\mathbf{p}) = V^* = 0.3903$. First, we consider the KCP method. Initially, the adversary sets $p = 1$ and the server's best response policy π_1 is the priority policy $1 \to 2$. The adversary then sets $p = 0$ and the server's best response policy π_2 is the priority policy $2 \to 1$. The expected damages, as functions of p, are shown by the solid black lines in Figure 5.2.

The solution of the finite matrix game (5.10) in which $\mathcal{P} = \{\pi_1, \pi_2\}$ is given by $(\mathbf{p}_3, V)$, where $\mathbf{p}_3 = (0.4233, 0.5767)$ and $V = 0.3925$ gives an upper bound for

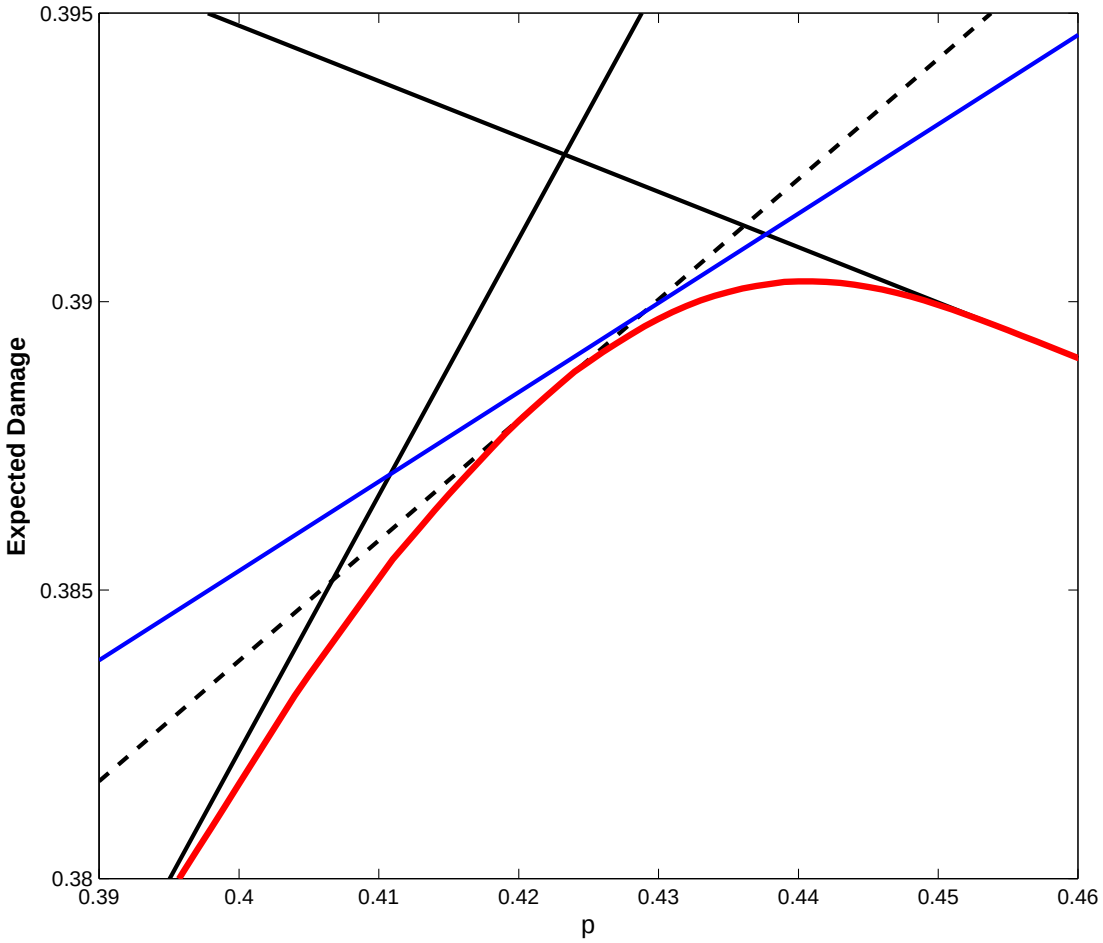

Figure 5.2: The first 3 iterations of the KCP method along with both the HCP and HCP$^+$ methods in Example 5.1. The red curve shows $V(\mathbf{p})$. The bold black lines show the expected damages of the priority policies $1 \to 2$ and $2 \to 1$ as functions of p. These are best response policies in the first 2 iterations of the KCP method and all priority policies in the HCP method. The dashed line shows the expected damage of the best response policy in the 3rd iteration of the KCP method. The bold blue line shows the expected damage of the API policy in the HCP$^+$ method.

$\lvert\mathcal{P}\rvert$	V	Percentage above V^*
1	0.4332	10.98
2	0.3925	0.56
3	0.3913	0.25
4	0.3907	0.08
5	0.3904	0.01
6	0.3903	0.01
7	0.3903	0.00
8	0.3903	0.00

Table 5.1: Value of the finite matrix game V and the percentage above the optimal expected damage V^* as the the size of the policy set $\mathcal{P}$ increases through the KCP method in Example 5.1.

V^*. The solution is achieved by maximising, with respect to p, the lower envelope of the straight lines given by the policies in $\mathcal{P}$. The lower envelope serves as an approximation of the red curve $V(\mathbf{p})$. The best response policy π_3 to $\mathbf{p}_3$ is shown by the dashed line. The expected damage $D^{\pi_3}(\mathbf{p}_3)$ gives a lower bound of 0.3886 for V^*. Solving the finite matrix game (5.10) in which $\mathcal{P} = \{\pi_1, \pi_2, \pi_3\}$ gives $(\mathbf{p}_4, V)$, where $\mathbf{p}_4 = (0.4361, 0.5639)$ and the value of the game $V = 0.3913$ gives an improved upper bound. The first 8 iterations of the KCP method are shown in Figure 5.3. When $|\mathcal{P}| = 8$, the value of the finite matrix game is 0.3903 and the resulting mixed strategy over the policies in $\mathcal{P}$ is *essentially optimal*. Table 5.1 shows the percentage above the optimal expected damage V^* of the solution to the finite matrix game as the size of the policy set $\mathcal{P}$ increases through the KCP method. Figure 5.3 shows that the approximation of $V(\mathbf{p})$ is extremely good after 8 iterations, especially around the maximising p. Table 5.1 further shows that the rate of convergence of the KCP method is quick, where the value of the finite matrix game is already 0.56% above V^* after two iterations. Also, much of the improvement in the finite matrix game is achieved in the first few iterations.

We now consider the HCP and HCP$^+$ methods. In the first two iterations of the HCP method, we used the $R\mu$ rule to add the priority policies $1 \to 2$ and $2 \to 1$ to $\mathcal{P}$. The expected damages, as functions of p, are shown by the solid black lines in Figure 5.2. There are only two priority policies in a $k = 2$ queue system, hence

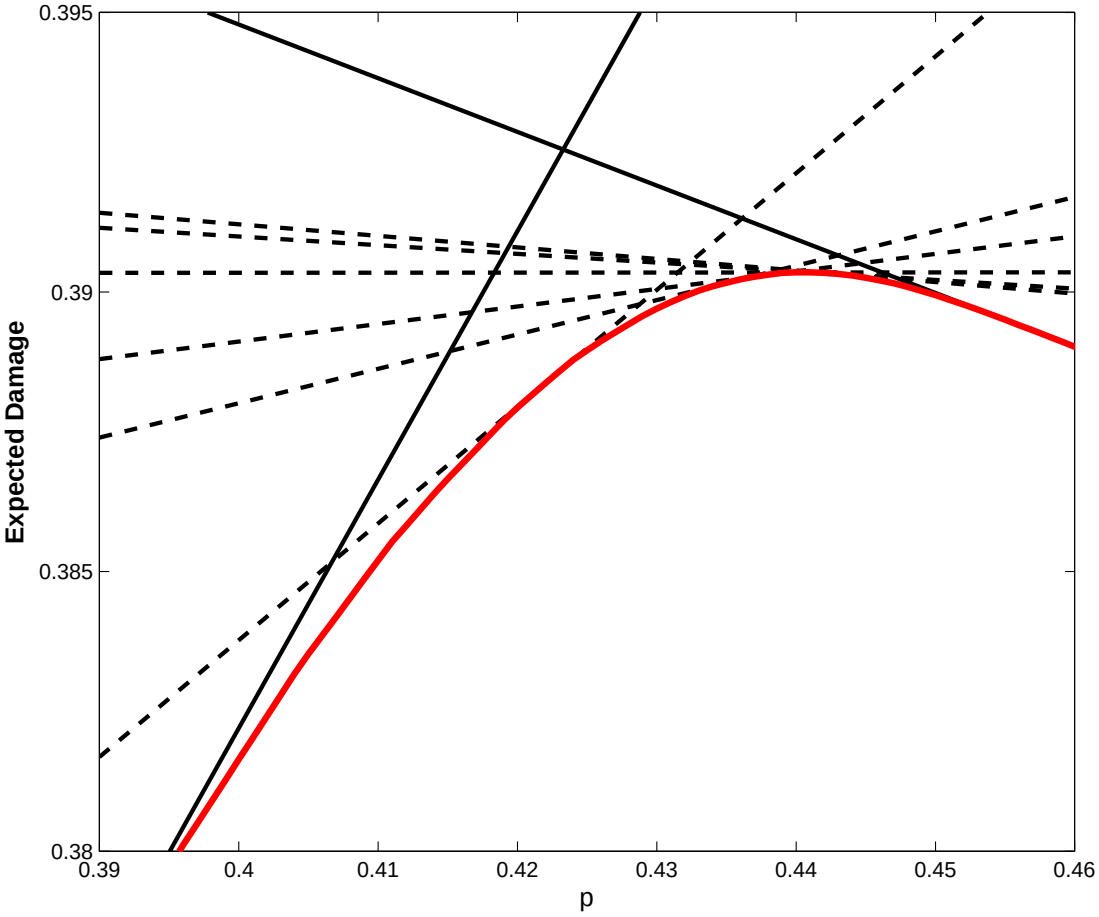

Figure 5.3: The first 8 iterations of the KCP method in Example 5.1. The red curve shows $V(\mathbf{p})$. The bold black lines show the expected damages of the priority policies $1 \to 2$ and $2 \to 1$ as functions of p. These are best response policies in the first 2 iterations of the KCP method. The dashed lines are each the expected damages of the best response policies in iterations 3 to 8 of the KCP method. Each bold black line and each dashed line is a tangent to $V(\mathbf{p})$. After 5 iterations, the percentage of the finite matrix game above the optimal expected damage V^* is 0.01%.

the HCP method terminates after two iterations. In this example, the first two iterations of the HCP method and the KCP method are identical, hence the HCP method achieves an expected damage 0.56% above optimal. The HCP method is also identical to the PA method. For the HCP$^+$ method, the solution of the finite matrix game (5.10) yields mixed strategy $\mathbf{p}_* = (0.4233, 0.5767)$. To construct the API policy as a heuristic response to $\mathbf{p}_*$, we used the $R\mu$ rule as the initial policy, along with the parameters $t = 1$, $m = 10^5$, $n = 45$, and $r = \frac{32}{45}$. The resulting API policy is shown by the solid blue line in Figure 5.2. As it is not a tangent to the red curve, it is not a best response policy, however its distance from the red curve indicates a small optimality gap in the random adversary problem (5.2). Solving the finite matrix game consisting of the two priority policies and the API policy, the HCP$^+$ method achieves an expected damage of 0.3911, which is 0.21% above V^*. The HCP$^+$ method improves upon the HCP method, albeit at increased computational expense. In this example, both heuristic approaches are not too far from the optimal expected damage.

To assess the performance of the heuristics more generally, we now conduct large sets of numerical experiments in $k = 2, 3$, and 5 queue systems. Problems were randomly generated to reflect a wide range of conditions with regard to (1) the length of customer lifetimes (reflected in the categorisation A, B, C in (5.11c)–(5.11e) below); and (2) the traffic intensity or workload in the corresponding system without abandonments. There are three categories of traffic—namely light, moderate, and heavy—as determined by the value of $\rho = \sum_{j=1}^{k} \lambda_j/\mu_j$; see (5.11f)–(5.11h) below. For all nine combinations of A, B, C with the traffic categorisation light, moderate, heavy, 100 problems were generated at random for

$k = 2, 3$, and 5 queue systems. Parameters were sampled as follows:

$$\mu_j \sim U[0.2, 5] \quad \text{(all cases)}; \tag{5.11a}$$

$$\lambda_j \sim U[0.2, 5] \quad \text{(all cases)}; \tag{5.11b}$$

$$\theta_j \sim U[1, 5] \quad \text{(short lifetimes, A)}; \tag{5.11c}$$

$$\theta_j \sim U[0.1, 0.5] \quad \text{(moderate lifetimes, B)}; \tag{5.11d}$$

$$\theta_j \sim U[0.01, 0.05] \quad \text{(long lifetimes, C)}; \tag{5.11e}$$

$$\rho \in [0.5, 0.7] \quad \text{(light traffic)}; \tag{5.11f}$$

$$\rho \in [0.9, 1.1] \quad \text{(moderate traffic)}; \tag{5.11g}$$

$$\rho \in [1.3, 1.5] \quad \text{(heavy traffic)}; \tag{5.11h}$$

In the parameter generation, μ_j and λ_j were sampled according to (5.11a) and (5.11b) by means of a rejection algorithm until a desired ρ condition (5.11f)–(5.11h) was met. In addition, in each generated problem we set $d_j = 1$ for each queue and consequently the objective concerned the abandonment probability of the adversary.

In each $k = 2$ problem, we used LP to compute the abandonment probability of the server's optimal randomised policy by truncating the state space at $N_j = 40$ for each queue j in case A, and $N_j = 60$ for each queue j in cases B and C, as discussed in Section 5.2.1. In the HCP and HCP$^+$ methods, we used DP value iteration with the same truncated state spaces to compute the payoffs for the finite TPZS matrix game (5.10), which was subsequently solved using LP to obtain the abandonment probability of each method. Within the HCP$^+$ method, we constructed the API policy using the $R\mu$ rule as the initial policy, along with the parameters $t = 1$, $m = 10^5$, $n = 45$, and $r = \frac{32}{45}$. Table 5.2 reports the numerical results for $k = 2$ queues.

In each $k = 3$ problem, we used the KCP method to compute the optimal expected damage V^*, since finding the server's optimal randomised policy through LP was computationally infeasible. In each iteration of the KCP method, we

Case	Workload		HCP	HCP$^+$
A	Light	90th	0.00	0.00
		75th	0.00	0.00
		Median	0.00	0.00
	Moderate	90th	0.03	0.01
		75th	0.01	0.00
		Median	0.00	0.00
	Heavy	90th	0.10	0.03
		75th	0.03	0.01
		Median	0.00	0.00
B	Light	90th	0.22	0.10
		75th	0.10	0.06
		Median	0.04	0.02
	Moderate	90th	0.98	0.54
		75th	0.59	0.27
		Median	0.23	0.11
	Heavy	90th	1.33	0.72
		75th	0.73	0.47
		Median	0.33	0.18
C	Light	90th	0.23	0.19
		75th	0.13	0.12
		Median	0.04	0.04
	Moderate	90th	3.51	3.03
		75th	1.85	1.66
		Median	0.94	0.90
	Heavy	90th	0.15	0.11
		75th	0.06	0.03
		Median	0.01	0.01

Table 5.2: Percentage above the optimal expected damage in $k = 2$ queue systems of various traffic and abandonment level combinations.

used DP value iteration with the same truncation levels used in the $k = 2$ queue problems to compute the payoffs for the finite TPZS matrix game (5.10), which was subsequently solved using LP. We also used this method to compute the expected damages of the heuristics HCP, HCP$^+$, and PA. In addition, we recorded the number of priority policies generated in the HCP method, n(HCP). Within the HCP$^+$ method, we constructed the API policy using the $R\mu$ rule as the initial policy, along with the parameters $t = 1$, $m = 10^5$, $n = 75$, and $r = \frac{52}{75}$. Table 5.3 reports the numerical results for $k = 3$ queues.

As seen in Tables 5.2 and 5.3, the performances of HCP, HCP$^+$, and PA are

Case	Workload		HCP	HCP$^+$	PA	n(HCP)
A	Light	90th	0.01	0.00	0.01	3
		75th	0.00	0.00	0.00	1
		Median	0.00	0.00	0.00	1
	Moderate	90th	0.02	0.01	0.02	3
		75th	0.00	0.00	0.00	2
		Median	0.00	0.00	0.00	1
	Heavy	90th	0.07	0.05	0.07	4
		75th	0.02	0.01	0.02	3
		Median	0.00	0.00	0.00	2
B	Light	90th	0.34	0.28	0.29	4
		75th	0.20	0.14	0.19	4
		Median	0.03	0.02	0.03	3
	Moderate	90th	1.27	1.04	1.18	4
		75th	0.82	0.61	0.75	4
		Median	0.50	0.35	0.43	3
	Heavy	90th	1.33	1.03	1.14	4
		75th	0.97	0.76	0.80	4
		Median	0.60	0.41	0.53	4
C	Light	90th	0.35	0.31	0.30	4
		75th	0.23	0.21	0.20	4
		Median	0.10	0.10	0.09	3
	Moderate	90th	4.01	3.43	3.65	4
		75th	2.81	2.40	2.65	4
		Median	1.73	1.51	1.61	3
	Heavy	90th	0.20	0.19	0.20	4
		75th	0.07	0.04	0.06	4
		Median	0.02	0.01	0.01	4

Table 5.3: Percentage above the optimal expected damage in $k = 3$ queue systems of various traffic and abandonment level combinations.

excellent, with the median percentage above the optimal expected damage being at most 0.6%. There is an exception in the category {C, Moderate}, in which the performance of each heuristic is less than 1% and 2% above the optimal expected damage in the $k = 2$ and $k = 3$ queue problems respectively. In Table 5.2, the performance of PA is not reported since it is equivalent to the performance of HCP. Table 5.3 shows that while PA requires the evaluation of 6 priority policies, it only performs marginally better than HCP which requires the evaluation of at most 4 priority policies in terms of the 90th percentile shown. The comparative performance of HCP and PA also indicates that using the $R\mu$ rule to select priority

policies within the HCP method does not adversely affect its performance. Both tables illustrate that HCP$^+$ outperforms HCP, with the median improvement offering at most a 0.16% reduction in the expected damage relative to HCP. This marginal improvement comes at the expense of increased computational cost. It is also the case that HCP$^+$ marginally outperforms PA in most cases.

In each $k = 5$ queue problem, we are unable to compute the optimal expected damage through either LP or the KCP method. Furthermore, we are unable to compute the expected damages of the heuristics HCP, HCP$^+$, and PA using DP value iteration, as we did in the case of $k = 2$ and $k = 3$ queues. Instead, for each heuristic, we use simulation to estimate the payoffs for each policy which comprises the set $\mathcal{P}$ in the finite TPZS matrix game (5.10). We estimate the payoffs using the mean of 2000 Monte Carlo realisations. Based on these estimates, we solve the matrix game using LP to obtain an estimate of the expected damage of each heuristic. Within the HCP$^+$ method, we constructed the API policy using the $R\mu$ rule as the initial policy, along with the parameters $t = 1$, $m = 10^5$, $n = 100$, and $r = \frac{69}{100}$.

Table 5.4 reports the numerical results for $k = 5$ queues. It shows the percentage improvement of both PA and HCP$^+$ over HCP, shown as Δ(HCP, PA) and Δ(HCP, HCP$^+$) respectively. The column denoted n(HCP) reports the number of priority policies evaluated in the HCP method. From the table, we see that while PA requires the evaluation of 120 priority policies, it only performs at most 0.31% better in terms of the median percentage improvement. The HCP method requires the evaluation of at most 8 priority policies in terms of the 90th percentiles and often only requires 6 in terms of the median and 75th percentiles. Hence, HCP requires approximately *5% of the computation time* of PA and achieves a *similar level of performance*. Once again, the comparative performance of HCP and PA indicates that using the $R\mu$ rule to select priority policies within the HCP method does not adversely affect its performance. HCP$^+$ offers marginal improvements over HCP, with the median improvement offering at most a 0.11% reduction in

Case	Workload		Δ(HCP, PA)	Δ(HCP, HCP$^+$)	n(HCP)
A	Light	90th	0.00	0.02	2
		75th	0.00	0.00	2
		Median	0.00	0.00	1
	Moderate	90th	0.00	0.02	5
		75th	0.00	0.00	3
		Median	0.00	0.00	1
	Heavy	90th	0.02	0.02	7
		75th	0.00	0.00	5
		Median	0.00	0.00	2
B	Light	90th	0.09	0.07	7
		75th	0.00	0.00	6
		Median	0.00	0.00	5
	Moderate	90th	0.30	0.27	7
		75th	0.18	0.17	7
		Median	0.09	0.06	6
	Heavy	90th	0.58	0.23	7
		75th	0.31	0.15	7
		Median	0.16	0.06	6
C	Light	90th	0.35	0.00	7
		75th	0.00	0.00	6
		Median	0.00	0.00	6
	Moderate	90th	0.83	0.27	6
		75th	0.54	0.17	6
		Median	0.31	0.11	6
	Heavy	90th	0.12	0.00	8
		75th	0.04	0.00	7
		Median	0.03	0.00	6

Table 5.4: Percentage improvement over HCP in $k = 5$ queue systems of various traffic and abandonment level combinations.

the expected damage.

Remark. *In systems with more than 3 customer classes, we cannot evaluate the performance of each heuristic with respect to the optimal damage. Subsequently, we seek a tight lower bound for the optimal expected damage. We can develop a lower bound based on a modification of the method used to derive an upper bound for the optimal reward rate in Section 4.3. For a given feasible policy, if x_j represents the implied fraction of time spent serving in queue j, then*

$$d_j\Big(1 - \frac{\mu_j x_j}{\lambda_j}\Big)$$

is the expected damage the adversary will inflict if he joins queue j. Let z represent the largest expected damage across all queues, then z is an upper bound for the optimal expected damage. To compute a lower bound for the optimal expected damage, we formulate a linear program similar in principle to the linear program given in Section 5.2.1 for the server's optimal randomised policy, but allow the x_j to be implied by an infeasible policy. We formulate the following linear program with the variables $x_j \geq 0$, $1 \leq j \leq k$, and the objective function to minimise z:

$$\begin{aligned} \min \quad & z \\ \textit{subject to} \quad & d_j\Big(1 - \frac{\mu_j x_j}{\lambda_j}\Big) \leq z, \quad 1 \leq j \leq k; \\ & \sum_{j=1}^{k} x_j \leq 1, \quad x_j \geq 0, \quad 1 \leq j \leq k; \\ & \textit{plus additional constraints.} \end{aligned}$$

We impose additional constraints on the x_j so that their values under the resulting optimal policy come as close as we can make them to those implied by a feasible policy. The additional constraints used are those based on sub-systems, derived in Section 4.3, to which the reader is referred for further details.

However, in practice we have found that the lower bound does not produce consistent results upon which we can assess the performance of each heuristic in $k = 5$ queue systems. Whilst it can be tight in some cases, it is not in others. Consequently, further investigation is required to improve the lower bound.

We draw upon a number of observations in order to infer the quality of the heuristics in the $k = 5$ queue systems. The excellent performance of all heuristics in $k = 2$ and $k = 3$ queue systems suggests that their performance is likely to be also be excellent in $k = 5$ queue systems. As observed in our earlier example, since HCP will often match the early iterations of the KCP method, which improves rapidly in the early iterations, this suggests the quality of the HCP method. Furthermore, in the random adversary problem studied in Chapters 3 and 4, we observed that

the suboptimality gap of the best priority policy in the majority of problems was not too large. This observation suggests that the PA method, which considers all priority policies, will always have at least one priority policy which is not too far from the optimal expected damage for any choice of $\mathbf{p}$ by the adversary. This also infers the strength of the HCP and HCP$^+$ methods.

Conclusion

In this chapter we have studied the defensive surveillance scenario of a strategic adversary who chooses where to attack. Since the server and the adversary do not know each others decision, we modelled their interaction as a simultaneous move two-person zero-sum (TPZS) game. An important feature of this approach is that it is important for the server to randomise his actions, which can be done within service policies or between service policies. By considering the TPZS game from the perspective of the adversary, we developed the heuristic cutting plane (HCP) and enhanced heuristic cutting plane (HCP$^+$) methods. These methods are heuristic applications of Kelley's cutting plane (KCP) method which is theoretically capable of delivering an optimal surveillance strategy. The methods exploit a strong connection with the random adversary surveillance scenario and iteratively develop a set of service policies to randomise over. Numerical experiments indicate the strong performance of this approach. Consequently, our suggestion for the security team is randomise over the set of deterministic service policies generated through a computed mixed strategy. In effect, this will require the security team to select one of the deterministic service policies according to a discrete probability distribution and then adopt this service policy thereafter, perhaps re-selecting in the same way every so often.

The methodology used in this chapter is widely applicable to other defender-attacker problems, namely the use of the random adversary problem as a sub-problem within the master strategic adversary problem. This method has been used successfully in other problems in the literature (see McMahan et al. (2003),

Lin et al. (2014), and Lin & Singham (2015)). This illustrates the general and flexible nature of the method. The main challenge in extending this approach to more general problems lies in the associated sub-problem and whether effective heuristic solutions to this problem can be found.

Chapter 6

Defence against Strategic Adversaries Who Choose Where and When To Attack

In this chapter we consider the defensive surveillance scenario identified in Chapter 1 in which the adversary acts strategically and can choose both which queue to attack and when to attack that queue. This scenario is an extension of the strategic scenario studied in Chapter 5. Subsequently, we consider a mathematical model with a near-identical stochastic structure. The only differences in this chapter are that we allow for both the case of *nonpreemptive service* and also allow the service times in each queue to follow some *general probability distribution*. We will make these differences clear when they arise. Whereas in Chapter 5, the adversary was able to choose which queue to join, we now consider the case in which the adversary can also choose the time at which he arrives into his chosen queue. In this case, the server wishes to find a robust service policy which provides a performance guarantee against any choice of queue and time the adversary could make.

The structure of this chapter is as follows. Section 6.1 studies a single queue system in which the adversary can choose when to attack the queue. Section 6.2 studies a system with multiple queues and control of the system by the security

team is decentralised. Section 6.3 studies a system with multiple queues and control of the system by the security team is centralised. We conclude the chapter by studying numerical examples in Section 6.4.

6.1 Single Queue

We will begin by formulating an optimisation problem in the context of a single queue system. The problem will then be generalised later in the chapter to systems with multiple queues. Given that we consider a single queue system, the adversary does not have a set of queues to choose from. *The adversary chooses the time* at which he joins the queue based on the state of the system. The state consists of two elements: the *volume state* and the *server state*. The volume state represents the *number of customers in the queue* and the server state represents *which customer is in service and how long that customer has been in service.*

We consider two scenarios which represent the capability of the adversary:

1. The adversary chooses both the volume state and the server state to attack. We refer to this as the *full information* scenario.

2. The adversary chooses the volume state to attack, but not the server state. We refer to this as the *partial information* scenario

To motivate these scenarios in practice, consider a defensive surveillance scenario in which an adversary targets a train station. Suppose the adversary has hacked into the security system, or has a spy in the security team. The adversary makes his way to the train station repeatedly and prepares to initiate an attack. Before entering the train station, the adversary checks with the spy to learn some information about the system. In the full information scenario, the spy tells the adversary the volume state and the server state. In the partial information scenario, the spy can only tell the adversary the volume state. Based on this information provided by the spy, the adversary decides whether or not to initiate an attack. If he chooses not to attack, *he leaves and will try again another time.* If he chooses

to attack he will enter the train system and join the queue. In practice, the adversary does not monitor the volume state or the server state in real time, instead he learns about them at the point of initiating an attack and bases his decision upon this information.

The server decides upon a service policy π to use in the system, of which there are an infinite number he could use. A policy is a rule followed by the server to decide which customer to serve next, each time the server becomes available. *The server can track the arrival times* of each customer and is able to define a policy based on these. The service policy can be randomised and the server can choose to idle when there are customers present, which we will refer to as *strategic idling*.

We study both preemptive service and nonpreemptive service. With nonpreemptive service, once the service of a customer begins, it only concludes once the service has been completed or the customer has abandoned. The server chooses a new customer for service at these points. On the other hand, preemptive service allows the service of a customer to be stopped at any moment so that the server can switch to another customer.

For a given scenario, full information scenario or partial information scenario, the objective of the server is to determine a robust service policy which provides the best performance guarantee against an adversary.

6.1.1 The Case of Preemptive Service

This section concerns a single queue case with preemptive service. We first assume that the service times follow an exponential distribution with rate μ. In this chapter, the abandonment probability of the adversary is heavily policy and state dependent. To see this, let us compare two service policies: the *first-come first-served* (FCFS) policy and the *last-come first-served* (LCFS) policy. Each time the server becomes available to start a new service, the FCFS policy selects the customer with the earliest arrival time into the queue whereas the LCFS policy selects the most recent arrival.

Under the FCFS policy, the abandonment probability of the adversary is heavily state dependent. Consider two scenarios: (a) the adversary joins when there is one customer in the system, (b) he joins when there are 100 customers in the system. In (a), the abandonment probability will be low since the server is likely to take the adversary into service before he abandons. In (b), the abandonment probability will he high since there are 100 customers ahead of the adversary and the server is unlikely to take him into service before he abandons.

Under the LCFS policy, however, the probability of abandonment is *state independent*. The abandonment probability depends entirely on the arrival process after the adversary has joined the queue, hence the probability in (a) and (b) will be the same. Since with the LCFS policy, a customer's abandonment probability does not depend on the number of customers in the queue when he arrives, it is intuitive that the LCFS policy is a strong policy against an adversary who chooses when to attack. We next present a theorem to formalise this idea.

Theorem 6.1. *Consider a M/M/1 queue with preemptive service and customer abandonment. Customers only remain available for service for a random time that follows an exponential distribution, after which the customer will abandon the system, whether the customer is still waiting in the queue or is already in service. The LCFS policy minimises the abandonment probability of an adversary who chooses which volume state to attack (partial information).*

Proof. Let x_i^π denote the probability of abandonment if a customer arrives when there are i customers in the system under policy π. Let p_i^π denote the steady state probability that there are i customers already in the system upon arrival under policy π. We define A^* as the long-run fraction of customers who abandon with any nonidling policy, which is invariant to any nonidling service policy. For any policy π, including policies that incorporate strategic idling, we have that

$$A^* \leq \sum_{i=0}^{\infty} x_i^\pi p_i^\pi, \tag{6.1}$$

where the equality holds if π is nonidling. If the adversary chooses which volume state to attack, then the worst case is for the adversary to achieve abandonment probability $\max_i x_i^\pi$. The optimisation problem faced by the server is given by

$$\min_\pi \max_i x_i^\pi \tag{6.2}$$

Since the right-hand side of (6.1) is a weighted average of x_i^π, it follows that a lower bound for $\max_i x_i^\pi$ is given by A^* for all policies π. Hence, a lower bound for (6.2) for all policies is given by

$$\min_\pi \max_i x_i^\pi \geq A^*. \tag{6.3}$$

Under the LCFS policy, x_i^{LCFS} is equal for all $i \geq 0$. Denote this probability by x^{LCFS}. Using (6.1), we have that

$$A^* = \sum_{i=0}^{\infty} x_i^{\text{LCFS}} p_i = x^{\text{LCFS}},$$

where equality holds since the LCFS policy is nonidling. Since the abandonment probability of the LCFS policy achieves the lower bound A^* given in (6.3), we can conclude that the LCFS policy minimises the abandonment probability of an adversary who chooses which volume state to attack. □

Corollary 6.1. *Consider a M/M/1 queue with preemptive service and customer abandonment. Customers only remain available for service for a random time that follows an exponential distribution, after which the customer will abandon the system, whether the customer is still waiting in the queue or is already in service. The LCFS policy minimises the abandonment probability of an adversary who chooses which volume state and server state to attack (full information).*

Proof. Since the adversary is more capable in the full information scenario, the optimal value A^* in Theorem 6.1 is a lower bound on the optimal value. Since the LCFS policy achieves the lower bound A^*, it is optimal. □

In the case that the service time distribution is *not exponential*, then the problem becomes much more complicated. Preemptive service has two variations. In the first variation, the service needs to start anew once interrupted; in the second variation, the previous service effort is retained after service interruption, and the service resumes when the same customer enters service again. In either of these two cases, interrupted service often translates to wasted effort, because the customer that receives partial service may abandon before entering the service again. The optimal policy remains an *open problem*, and is outside the scope of this thesis.

6.1.2 The Case of Nonpreemptive Service

This section concerns a single queue case with nonpreemptive service. We first assume that the service times follow an exponential distribution with rate μ. With nonpreemptive service, once the service of a customer begins, it only concludes once the service has been completed or the customer has abandoned. In practice, it is simpler to implement service nonpreemptively to avoid the complication of service interruption and resumption. We next show that the LCFS policy provides the best performance guarantee among all nonidling policies in a single queue system with exponential service times and nonpreemptive service.

Theorem 6.2. *Consider a M/M/1 queue with nonpreemptive service and customer abandonment. Customers only remain available for service for a random time that follows an exponential distribution, after which the customer will abandon the system, whether the customer is still waiting in the queue or is already in service. Among all nonidling policies, the LCFS policy minimises the abandonment probability of an adversary who chooses which volume state to attack (partial information).*

Proof. Let x_i^π denote the probability of abandonment if a customer arrives when there are i customers in the system under nonidling policy π. This probability does not depend on the server state, because of the memoryless property of the exponential service distribution. Let p_i denote the steady state probability that

there are i customers already in the system upon arrival for any nonidling service policy. The long-run fraction of arriving customers who abandon by any nonidling policy is given by

$$A^* = \sum_{i=0}^{\infty} x_i^\pi p_i = x_0^\pi p_0 + \sum_{i=1}^{\infty} x_i^\pi p_i.$$

Denote the arrival rate by λ, the exponential service rate by μ, and the exponential abandonment rate by θ. For all nonidling policies π, the probability of abandonment of those customers arriving into an empty queue, x_0^π, is given by

$$x_0^\pi = \frac{\theta}{\theta + \mu}.$$

The probability x_0^π is equal to the probability that a customer in service abandons before service is complete. It is a function of the parameters μ and θ which are independent of the service policy and hence is constant. The probability of arriving into an empty queue is given by

$$p_0 = \left[\sum_{n=0}^{\infty} (\lambda)^n \left\{ \prod_{m=1}^{n} (\mu + m\theta) \right\}^{-1} \right]^{-1},$$

which is a function of the arrival rate λ and the parameters μ and θ, and hence is constant. Therefore,

$$\sum_{i=1}^{\infty} x_i^\pi p_i = A^* - x_0^\pi p_0, \tag{6.4}$$

remains constant under all nonidling policies.

Since an arrival into an empty queue is taken into service immediately whereas an arrival into a non-empty queue faces some initial wait before potential service, $x_i^\pi > x_0^\pi$ for all $i \geq 1$. From this, we see that the adversary will have a greater abandonment probability if he joins the queue when it is non-empty. In the worst case, the adversary maximises the abandonment probability and achieves $\max_{i \geq 1} x_i^\pi$. The optimisation problem faced by the server is given by

$$\min_{\pi} \max_{i \geq 1} x_i^\pi \tag{6.5}$$

Since $x_i^\pi \leq \max_{i\geq 1} x_i^\pi$, we have the following

$$\sum_{i=1}^{\infty} x_i^\pi p_i \leq \sum_{i=1}^{\infty} \big(\max_{i\geq 1} x_i^\pi\big) p_i = \big(\max_{i\geq 1} x_i^\pi\big) \sum_{i=1}^{\infty} p_i = \big(\max_{i\geq 1} x_i^\pi\big)(1-p_0).$$

Hence, using (6.4), for all nonidling policies π we have that

$$\max_{i\geq 1} x_i^\pi \geq \frac{A^* - x_0^\pi p_0}{1-p_0}. \tag{6.6}$$

The constant given in the right-hand side of (6.6) provides a lower bound for (6.5). Under the LCFS policy, x_i^{LCFS} is equal for all $i \geq 1$. Denote this probability by x^{LCFS}. From Equation (6.4), we have that

$$x^{\mathrm{LCFS}} = \frac{A^* - x_0^\pi p_0}{1-p_0}.$$

Since the abandonment probability of the LCFS policy achieves the lower bound for (6.5), we can conclude that the LCFS policy minimises the abandonment probability of an adversary who chooses which volume state to attack in the partial information scenario. □

Please note that Theorem 6.2 states that the LCFS policy is optimal among nonidling policies. *If strategic idling is allowed*, then the server may not always want to serve a customer who arrives to an empty system. Instead, the server can stay idle from time to time, such that a customer who arrives at a nonempty system also has a probability to enter service right away. An optimal strategy would use strategic idling so that the abandonment probability of a customer will be the same regardless of the number of customers in system when he arrives. The derivation of the optimal policy is outside the scope of this thesis, and remains an *open problem*.

Corollary 6.2. *Consider a M/M/1 queue with nonpreemptive service and customer abandonment. Service times follow an exponential distribution with rate μ. Customers only remain available for service for a random time that follows*

an exponential distribution with rate θ, after which the customer will abandon the system, whether the customer is still waiting in the queue or is already in service. Among all nonidling policies, the LCFS policy minimises the abandonment probability of an adversary who chooses which volume state and server state to attack (full information).

Proof. Since the adversary is more capable in the full information scenario, the optimal value is no smaller than that in Theorem 6.2. The additional service time of the customer currently in service when the adversary arrives at a chosen time point follows the same exponential distribution as the additional service time when the adversary arrives at a random time point. Consequently, using the LCFS policy, the server can obtain the same abandonment probability as in Theorem 6.2. Hence the LCFS policy is optimal. □

We next *relax the assumption* on the exponential service distribution, and allow the service time to follow an arbitrary probability distribution. We next show that the LCFS policy provides the best performance guarantee in a single queue system with general service time distributions and nonpreemptive service.

Theorem 6.3. *Consider a M/G/1 queue with nonpreemptive service and customer abandonment. Customers only remain available for service for a random time that follows an exponential distribution with rate θ, after which the customer will abandon the system, whether the customer is still waiting in the queue or is already in service. Among all nonidling policies, the LCFS policy minimises the abandonment probability of an adversary who chooses which volume state to attack (partial information).*

Proof. The proof is exactly the same as the proof of Theorem 6.2, apart from the following small detail. Let Z denote a random variable which represents the service time of a customer, then $x_0^\pi = 1 - E[e^{-\theta Z}]$, which is constant under all nonidling policies. The probability of arriving into an empty queue is also different to that quoted in the proof of Theorem 6.2, but also remains constant under all nonidling

policies. □

In the case of generally distributed service times, we do not have a corollary for the full information scenario. This is because the full and partial information scenarios are no longer equivalent. If the adversary can choose which server state to attack, then he prefers a server state which will occupy the server for the longest in some stochastic sense. To facilitate the discussion, suppose that the service time distribution possesses the *increasing failure rate* (IFR) property. This property asserts that if x time units have already been spent on the service of a customer, the residual service time is stochastically decreasing in x. This assumption is suitable if the service of a customer draws increasingly nearer completion as it proceeds. With IFR service times, the abandonment probability for the adversary arriving at some chosen time point in the service cycle will not be equal to that when arriving at some random point. In the full information scenario, the adversary knows the server state. The adversary knows that he can maximise his abandonment probability from each initial volume state by joining the queue at the exact instant a new service is initiated. This choice ensures the longest possible wait for the adversary before he can potentially be taken into service. However, if the server knows the adversary will do this, he can use this information to recognise that the new customer joining the system at this time instant is the adversary. Hence, at the end of the service cycle, the server will pick out the adversary for service. A service policy which does this is clearly not LCFS. Taking this further, in response the adversary may wish to disguise his arrival somehow, perhaps by waiting some small random time after a new service cycle is initiated before joining the system. Identifying a robust service policy in the full information scenario with general IFR service times remains an open problem, and is outside the scope of this thesis.

In practice, *the server may or may not know the capability of the adversary.* The adversary may or may not know the volume state or the server state, or in some cases the adversary may be able to take a guess with some success. If the adversary is not as capable as assumed in the full information scenario or

the partial information scenario, then the theorems in this section can provide an *upper bound on the abandonment probability of a less capable adversary.*

6.2 Multiple Queues with Decentralised Control

We next generalise the problem to systems with multiple queues. We will consider the case in which service is nonpreemptive and service times are generally distributed with the *IFR property.* In a multiple queue system, the adversary chooses *which queue* to attack and *the time* at which he joins that queue based on the state of the system. As before, the state consists of two elements: the volume state and the server state. The volume state now refers to the number of customers in each queue. We will denote the volume state by $\mathbf{n} = (n_1, ..., n_k)$, where n_j is the number of customers in queue j. We consider the partial information scenario, such that the adversary chooses the volume state to attack, but not the server state. This scenario may impose too strong an assumption on the capability of the adversary, however, we study this scenario to develop performance guarantees for the server in practice. We will study decentralised control in this section, and centralised control in the next section.

Control of the system by the security team is *decentralised.* Consider the case in which the server represents a centralised computer centre which is capable of comparing a suspect's biometric data to a large database. Also, each queue represents a major airport. In decentralised control, the information available to the server consists of which queues require service. The server does not have information regarding the arrival times of customers within each queue. Each time the centralised server becomes available it is allocated to a queue (an airport), which is under its own local control. At this point, the server gains access to the arrival time information of customers within the queue and consequently a customer is selected for service based on this information. In effect, a feasible service policy for the server then comprises a two-staged decision: which queue, then which customer.

The server does not know the queue and state in which the adversary will attack the system. If the server uses policy π and the adversary attacks queue j in state $\mathbf{n}$, denote the expected damage inflicted by $C_j^\pi(\mathbf{n})$. The objective of the server is to determine a robust service policy which provides the best performance guarantee against the best decision the adversary could make. We consider the case of *nonidling* service policies. Namely, the server wants to solve the following optimisation problem over the class of nonidling service policies:

$$\min_{\pi} \max_{j,\mathbf{n}} C_j^\pi(\mathbf{n}). \tag{6.7}$$

In general, it is *very difficult* to solve (6.7). Firstly, there is no known method to compute $C_j^\pi(\mathbf{n})$ for any combination of policy π, queue j, and volume state $\mathbf{n}$. The best we can do is estimate these expected damages through simulation for given choices of π, j, and $\mathbf{n}$. Secondly, the decision spaces of both the server and adversary are infinite. Hence, estimating the expected damages under all combinations of π, j, and $\mathbf{n}$ is not possible. Given the inability to solve (6.7), our objective is to develop a *heuristic approach* to this problem to obtain a strong upper bound performance guarantee for (6.7).

The first step in our heuristic approach is to simplify the optimisation problem in (6.7). We argue that the largest expected damages inflicted by the adversary occur when the adversary attacks when there are a *vast number of customers* in each queue. Our rationale is that the adversary joins the system in amongst a large number of other customers, since the server will always be kept busy with demands from other customers in all queues. No other queue will empty during the lifetime of the adversary and there will always be other customers which the server could potentially serve ahead of the adversary. We will assume that by vast we mean that the worst case for the server occurs when the adversary attacks when the number of customers in each queue approaches infinity, which will be referred to as the *over-crowded state*. Although such an over-crowded state would never be realised in practice, the analysis emerging from this *worst case scenario* would

provide a *performance guarantee in practice*. We will drop the dependence on the volume state $\mathbf{n}$ and write C_j^π for the expected damage inflicted when the adversary joins queue j in the over-crowded state and the server uses policy π. Let the value of the best performance guarantee be given by C^*, then the optimisation problem given in (6.7) is restricted to the following optimisation problem

$$C^* = \min_\pi \max_j C_j^\pi. \tag{6.8}$$

The simplification from (6.7) to (6.8) does not mean it can be solved. However, it does make it easier to develop a heuristic approach since this formulation incorporates the effect of the adversary's choice of volume state and the problem which remains is one of the adversary's choice of queue and the server's choice of policy.

6.2.1 Heuristic Policies

In designing a heuristic decentralised policy, recall the strong performance of the LCFS policy in the case of a single queue. In the two-staged decision required in a decentralised policy, we propose that once the server has been allocated to a queue, the queue will exercise the *LCFS policy locally*; that is, allocate service to the customer who most recently joined the queue. Consequently, we will focus on the first part of the server's decision of which queue to serve each time the server becomes available.

We study three heuristic policies. A simple heuristic policy is the *Round Robin* (RR) policy, which we define as follows:

Definition 6.1. *Under the Round Robin (RR) policy, the server continues to serve the same customer until either service completion or customer abandonment. Each time the server needs to select a new customer to serve, allocate the server to a queue according to a simple schedule. Under this schedule, the server is allocated to each queue in turn and after which all queues have been visited, the cycle is*

repeated. If the server is allocated to an empty queue, it is skipped and the server is allocated to the next queue in the schedule. Once allocated to a queue, a customer is selected for service according to the LCFS policy.

Consider the simple case of a symmetric system in which all queues have the same arrival rates, customer lifetimes, service times, and damages. Intuitively, since all queues are identical, a good service policy will treat all queues fairly and offer the same amount of service in each queue. The RR policy is fair to each queue, offering each the same amount of service and returning to each queue after random amounts of time which are equal in distribution. From this observation, it is intuitive that the RR policy is a strong policy in the symmetric system described. However, in the case of an asymmetric system where the queues differ in their characteristics, there is no way to naturally extend the RR policy to account for the asymmetry.

We introduce two heuristic policies which can be extended to *asymmetric* systems. These heuristics reflect an alternative approach to allocating the server. Instead of allocating according to a schedule, we can randomly allocate the server to a queue according to a probability vector. The first of these heuristic policies is the *Departure Reselection* (DR) policy. We define the DR policy as follows:

Definition 6.2. *The Departure Reselection (DR) policy is parametrised by a* probability vector*, or reselection vector* $\mathbf{r} = (r_1, ..., r_k)$ *in which* $0 \leq r_j \leq 1$ *and* $\sum_j r_j = 1$*. Under the DR policy with reselection vector* $\mathbf{r}$*, the server continues to serve the same customer until either service completion or customer abandonment. Each time the server needs to select a new customer to serve, allocate the server to a queue according to the reselection vector* $\mathbf{r}$*. If there are one or more empty queues, the server is allocated to nonempty queue* j *with probability proportional to* r_j*. Once allocated to a queue, a customer is selected for service according to the LCFS policy.*

In the simple symmetric system described previously, setting each r_j equal is another way to achieve a fair policy which we believe would be strong. We also

define the *Service Reselection* (SR) policy in an analogous way as follows:

Definition 6.3. *The Service Reselection (SR) policy is parametrised by a probability vector, or reselection vector* $\mathbf{r} = (r_1, ..., r_k)$ *in which* $0 \leq r_j \leq 1$ *and* $\sum_j r_j = 1$. *Under the SR policy with reselection vector* $\mathbf{r}$, *the server continues to serve in the same queue until a service completion occurs. If the current customer in service abandons, another customer from within the same queue is selected according to the LCFS policy. If the queue is now empty and there is no customer from within the same queue to serve, the server needs to be allocated to a new queue. Each time the server needs to be allocated to a new queue, allocate according to the reselection vector* $\mathbf{r}$. *If there are one or more empty queues, the server is allocated to nonempty queue* j *with probability proportional to* r_j. *Once allocated to a queue, a customer is selected for service according to the LCFS policy.*

The DR and SR policies are similar in the way the server is allocated to a queue according to a reselection vector. However, they differ in when the server becomes available for re-allocation. The server will be re-allocated *far more frequently* under the DR policy than under the SR policy.

The DR and SR policies define two classes of service policies, each parametrised by the reselection vector $\mathbf{r}$. We will focus on the DR policy from this point, but point out that all of the subsequent discussion can also be applied to the SR policy in the same way. We write $V_j(\mathbf{r})$ for the expected damage from the over-crowded state when the adversary joins queue j under the DR policy with parameter $\mathbf{r}$. If we restrict the server to only using service policies from the class of DR policies, we can seek the best performance guarantee for the system from *within the class of DR policies* by solving the following optimisation problem

$$\min_{\mathbf{r}} \max_{j} V_j(\mathbf{r}). \tag{6.9}$$

The best performance guarantee from within the class of DR policies is an upper bound for the best performance guarantee C^* over all decentralised policies.

The optimisation problem given in (6.9) of finding the best performance guarantee within the class of DR policies corresponds to *optimising the reselection vector* $\mathbf{r}$. Let $\mathcal{D}$ be the set or domain over which we define the functions $V_j : \mathcal{D} \to \mathbb{R}$ for $j = 1, ..., k$. The set $\mathcal{D} = \{\mathbf{r} : 0 \leq r_j \leq 1, \sum_j r_j = 1\}$, namely the unit simplex in $k - 1$ dimensions. The optimisation problem given in (6.9) is equivalent to a global optimisation problem of finding $\mathbf{r}^* \in \mathcal{D}$ such that $\max_j V_j(\mathbf{r}^*) \leq \max_j V_j(\mathbf{r})$ for all $\mathbf{r} \in \mathcal{D}$.

We must estimate the expected damages $V_j(\mathbf{r})$ for given vectors $\mathbf{r} \in \mathcal{D}$ through simulation. If we were able to estimate these for every $\mathbf{r} \in \mathcal{D}$, then we would be able to solve the global optimisation problem. However, this is not practical. Simulation is a computationally *costly* process, hence we can view the estimation of $V_j(\mathbf{r})$ through simulation as an *expensive black box*. In practice we wish to find $\tilde{\mathbf{r}} \in \mathcal{D}$ such that $\max_j V_j(\tilde{\mathbf{r}})$ is close to $\max_j V_j(\mathbf{r}^*)$ without requiring too much estimation using the black box. The next section presents an algorithm to compute $\mathbf{r}^*$.

6.2.2 Computing the Best DR and SR Policies

We will present a method for solving (6.9) through an intelligent use of the expensive simulation black box which is based on response surfaces. Response surfaces are approximations of parametrised functions based on response values of the function. The main advantage of response surfaces in our application are their ability to provide *inexpensive approximations* to the simulation black box functions. These approximations can be used to identify candidate vectors $\mathbf{r}$ for estimation using the expensive black box. We will follow a method in Regis & Shoemaker (2005) known as the *Constrained Optimization using Response Surfaces* (CORS) method. This method was developed in the context of a classic problem of minimising an expensive black box function, as opposed to the minimax problem posed in (6.9); however, the adjustment is straightforward. The CORS method was shown to *converge to the global minimum* under quite general conditions.

In the original application of the CORS method in Regis & Shoemaker (2005), an initial set of points in $\mathcal{D}$ are chosen and evaluated and a response surface is fitted to the data to approximate the expensive black box function to be minimised. The initial set of points are chosen to be well-spread, for which the authors use a symmetric latin hypercube design. The response surface used is a radial basis function approximation. The next point for costly function evaluation is chosen to be a point which minimises the response surface subject to distance constraints. These distance constraints ensure that the proposed point is chosen to be at least some distance from all previously evaluated points. The method is iterative since it updates the response surface model after each new point is evaluated and then reselects the next point for function evaluation. One aim is to find points with good objective function values, which can be done by exploitation of the current information. Another aim is to improve the response surface and hence future iterations by exploration of regions for which little information exists. The method manages the *trade-off between exploration and exploitation* by allowing the distance constraint to cycle from high to low values.

We now summarise our application of the CORS method to (6.9), with more details to follow. To facilitate discussion, we present the method in the context of the DR policy. The best SR policy can be computed in a similar manner.

1. Select n well-spread initial points $\mathbf{r}_i$ for $i = 1, ..., n$. Let $R = \{\mathbf{r}_1, ..., \mathbf{r}_n\}$.

2. For $\mathbf{r} \in R$, simulate the DR policy with reselection vector $\mathbf{r}$, and record the expected damages in each queue $V_j(\mathbf{r})$ for $j = 1, ..., k$.

   Repeat steps 3 to 6 until some stopping condition is met:

3. Fit k response surface models $\tilde{V}_j$ for each queue j using the data, $j = 1, \ldots, k$. We use a radial basis function interpolation for each response surface.

4. Compute the maximin point

$$\delta = \max_{\mathbf{r} \in \mathcal{D}} \min_{r_i \in R} ||\mathbf{r} - \mathbf{r}_i||$$

5. Select a candidate point $\mathbf{r}_{\text{new}}$ for simulation by solving the following constrained approximate optimisation problem

$$\min_{\mathbf{r}\in\mathcal{D}} \max_{j} \tilde{V}_j(\mathbf{r}) \tag{6.10}$$

$$\text{Subject to} \quad ||\mathbf{r} - \mathbf{r}_i|| \geq \beta\delta \quad \text{for all } \mathbf{r}_i \in R$$

where $\beta \in [0, 1]$ is a predetermined constant.

6. Add the candidate point $\mathbf{r}_{\text{new}}$ to R, simulate its performance $V_j(\mathbf{r}_{\text{new}})$ for each queue j, update the data, and go to Step 3.

In step 1, in contrast to selecting initial points in a hypercube through a deterministic design, as was done in Regis & Shoemaker (2005), the well-spread initial points are selected by generating a set of uniform random samples from the unit simplex in $k-1$ dimensions. This is done by sampling from the symmetric Dirichlet distribution with parameter 1. In Regis & Shoemaker (2005), the convergence of the method did not depend on the initial points used. Hence, we adopt an approach which sensibly achieves a set of well-spread points in a unit simplex. In step 3, each response surface is fitted using a radial basis function interpolation method. This is the same radial basis function interpolation method described in Chapter 4 in the context of the API algorithm. We encourage the reader to refer to Chapter 4 for details.

The maximin point δ is the point in $\mathcal{D}$ which is the furthest away from all previously evaluated points. The distance constraint is implemented using the maximin point along with the distance parameter β, where $0 \leq \beta \leq 1$. The distance parameter β is set to cycle over a sequence of decreasing values, starting with a high value close to 1 and ending with $\beta = 0$. We represent the cycle sequence of β values by $(\beta_1, \beta_2, ..., \beta_N = 0)$, where N is the cycle length. The cycling of β divides the CORS method into a sequence of rounds, where each round consists of N iterations. The number of rounds can be used as a stopping condition for the method. The complete CORS method then consists of an *initial phase* followed

by a *fixed number of further rounds*.

Solving the constrained approximate minimax problem (6.10) with $\beta = 1$ explores the set $\mathcal{D}$, whereas solving with $\beta = 0$ exploits the current approximations $\tilde{V}_j(\mathbf{r})$. A cycle consisting of β values close to one focuses on exploration. This is not desirable since the method uses all of its costly function evaluations constructing good radial basis function approximations without ever trying to identify a point with a good objective function value. A cycle consisting of zeros represents the extreme case of pure exploitation. This may not be desirable if the underlying radial basis function approximation is not adequate and the method may become trapped at an undesirable point. We want a cycle sequence which *balances* between these two extremes, which *explores earlier* and *exploits later* in each round. Hence, this underpins the rationale of cycling over a sequence of decreasing values. Ending each round with $\beta = 0$ ensures that we periodically solve an approximate version of the optimisation problem (6.9).

We note that in step 4 when the maximin point is computed we must solve an optimisation problem. We propose an *approximate solution* of this optimisation problem by considering a set of points defined on a *fine grid* over $\mathcal{D}$. Through evaluation of the objective at each point in the fine grid we simply take the largest of these to identify approximately the maximin point. We adopt a similar approach in step 5 when we solve (6.10) since we are able to evaluate both the objective function and the distance constraints at each fine grid point. The point which satisfies the distance constraints with the lowest objective function is the approximate solution to (6.10). In both cases, a finer grid results in a more accurate solution at the cost of more computational effort.

At the end of the CORS method we obtain a strong reselection vector $\mathbf{r}$ which should be close to to the optimal reselection vector $\mathbf{r}^*$. However, the CORS method is restricted to reselection vectors on a grid in $\mathcal{D}$. Below we attempt to improve the output from the CORS method by finding a reselection vector $\mathbf{r}$ *off the grid*. Consider a TPZS game between the server and the adversary. The robust opti-

misation problem of the server is equivalent to the server's problem in a TPZS game between the server and adversary. In the TPZS game, pure strategies of the adversary correspond to joining each of the k queues in the over-crowded state and pure strategies of the server correspond to decentralised service policies. We can formulate a finite version of the TPZS game in matrix form in which the server is restricted to a finite set of service policies. We can use the set of DR policies developed throughout the CORS method as the server's set of service policies in the matrix game. The payoffs in the matrix game are the estimated expected damages $V_j(\mathbf{r})$ for each queue j and vector $\mathbf{r} \in R$. We can formulate the matrix game from the perspective of the server to obtain a mixed strategy over his set of DR policies. The value of the matrix game provides a performance guarantee for the server. This performance guarantee will be at least as good as the one given by the DR policy with the final reselection vector in the CORS method, which corresponds to a *pure strategy in the matrix game.*

It is possible to *extend this approach* by first solving the matrix game to obtain a mixed strategy $\mathbf{q}$ for the server over his set of pure strategies. Suppose the server has m pure strategies where $m = |R|$. We define a new reselection vector $\mathbf{r}_{\text{new}}$ according to the weighted average $\mathbf{r}_{\text{new}} = \sum_{i=1}^{m} q_i \mathbf{r}_i$. We simulate the DR policy with reselection vector equal to $\mathbf{r}_{new}$, estimate the expected damages in each queue, and *add this new service policy to the matrix game.* We repeat this procedure until no further new points can be identified. In adding new service policies to the matrix game, the server can *potentially improve* the value of the game. This procedure allows us to identify reselection vectors which do not lie on the grid and which may be better than any feasible vector which could be found on the grid. Consequently, our heuristic approach is to adopt the optimal mixed strategy from the finite TPZS matrix game resulting from the CORS method followed by the extension described.

6.3 Multiple Queues with Centralised Control

In this section, we study centralised control for systems with multiple queues. In centralised control, the server decides which customer to serve from all customers in the system. The control of each queue is centralised and not local. The server has access to more information upon which to base the decision of which customer to select for service. In the example of the server representing a centralised computer centre and each queue representing a major airport, the server keeps track of all of the suspects in each airport. It is possible to compare the arrival times of customers across queues, whereas in the case of decentralised control it was only possible to compare the arrival times of customers within queues once the server was allocated to that queue. Decentralised service policies are still feasible in the case of centralised control, but given the greater capability of the server in the centralised control, we would expect the server to achieve better performance guarantees on the system.

All other aspects of the problem are the same as in the case of decentralised control. The objective of the server is to determine a robust service policy which provides the best performance guarantee against the best decision the adversary could make in the partial information scenario. The optimisation problem for the server is given by (6.7). The difference we now have is that the server seeks the *robust service policy over the set of centralised service policies.*

The key observation when moving from decentralised control to centralised control is that the server now knows and can *compare the arrival times of all customers in the system*, across queues. Hence, the server can build this information into the service policy. In a single queue system, we considered the LCFS policy as the policy which served the most recently arrived customer in the queue. We can generalise this idea to define the LCFS policy in a multiple queue system. In a multiple queue system, the server can sort the arrival times of every customer in the system and the LCFS policy serves the most recently arrived customer into the system.

We will first consider the case of a system with multiple symmetric queues. It is a consequence of Theorem 6.3 that the LCFS policy provides the best performance guarantee in (6.7).

Corollary 6.3. *Consider a system with multiple queues in which service is non-preemptive and customers can abandon the system. Customers arrive into each queue according to independent Poisson processes. Service times in each queue follow an arbitrary probability distribution. Customers only remain available for service in each queue for random times that follow exponential distributions, after which customers will abandon the system, whether the customer is still waiting in the queue or is already in service. The system is symmetric, in that the service time distributions, the customer lifetime distributions, and damages inflicted in each queue are common. The Poisson arrival rates in each queue may differ. Among all nonidling policies, the LCFS policy minimises the expected damage of an adversary who chooses which queue and volume state to attack (partial information).*

Proof. The system with multiple symmetric queues is equivalent to a single queue with multiple classes of customers, if we label a customer in queue i as a class i customer. The arrival rate is $\sum_i \lambda_i$ for the single queue.

From Theorem 6.3, the LCFS policy is optimal for a single queue model. The optimal value in a single queue model is the abandonment probability of a customer that arrives into a nonempty queue under the LCFS policy. Denote this value by V^*.

The difference between the M/G/1 queue with multiple classes of customers we consider here and the M/G/1 queue in Theorem 6.3 is that, in the latter the adversary can decide to join the queue based on $\sum_i n_i$, while in the former the adversary can choose based on specific volume state $\mathbf{n} = (n_1, n_2, ..., n_k)$. The adversary chooses the volume state and a class for himself. Since the adversary has more choices in the M/G/1 queue with multiple classes of customers, the optimal abandonment probability is at least V^*. Hence, V^* is a lower bound for

the optimal value.

However, being able to choose the volume state and a class for himself does not help the adversary, since the service time and lifetime of each customer class is identical. If the server uses the LCFS policy, the abandonment probability will be V^* for $\sum_i n_i \geq 1$. If the adversary chooses to arrive in the volume state $(0, 0, ..., 0)$, then the abandonment probability is less than V^*.

For the M/G/1 queue with multiple customer classes, since the LCFS policy achieves V^*, which is a lower bound for the optimal value, the LCFS policy is optimal. Hence, the LCFS is optimal for the system with multiple symmetric queues.

□

Whilst the LCFS policy provides the best performance guarantee in symmetric systems, we would not expect this to be the case in general when the characteristics of each queue are different. Because (6.7) is very difficult to solve, our objective is to develop a heuristic approach to this problem to obtain a strong upper bound performance guarantee for the system. We will evaluate our heuristic based on its performance when an adversary arrives in an over-crowded state, since the adversary will achieve the maximal abandonment probability for any service policy.

6.3.1 Last Come First Served with Probabilistic Skipping

Motivated by the strong performance of the LCFS policy, we propose a heuristic based on the spirit of the LCFS policy. Under the LCFS policy, from the over-crowded state, denote the expected damage if the adversary joins queue j by C_j^{LCFS}. These expected damages will be *unequal in each queue* and the worst-case expected damage will be achieved if the adversary joins queue ω which satisfies $\omega = \text{argmax}_j C_j^{\text{LCFS}}$. Intuitively, we can *reduce the expected damage in queue* ω by instead using a policy which provides *more service in this queue* than the LCFS policy. This would be provided at the expense of *less service in the other queues*, increasing the expected damage if the adversary attacks the other queues.

However, the net effect would be to reduce the largest of these expected damages, hence a better performance guarantee for the server.

We can use this observation as the motivation for a heuristic policy designed to achieve this reduction in expected damage relative to the LCFS policy. We will refer to this heuristic policy as *Last Come First Served with Probabilistic Skipping* (LCFS-PS). We define the LCFS-PS policy as follows:

Definition 6.4. *The Last Come First Served with Probabilistic Skipping (LCFS-PS) policy is parametrised by a vector of probabilities, or skipping vector* $\mathbf{s} = (s_1, ..., s_k)$ *in which* $0 \leq s_j \leq 1$ *and* $s_\omega = 1$. *Under the LCFS-PS policy with skipping vector* $\mathbf{s}$*, the server continues to serve the same customer until either service completion or customer abandonment. Each time the server needs to select a new customer to serve, identify the most recent arrival in each queue and order them by their arrival times into a list. Starting with the most recent arrival in this list, allocate the service to the customer with probability* s_j *or skip it and move to the next customer with probability* $1 - s_j$*, if the customer comes from queue* j*. The process is repeated if all customers in the list are skipped, until the service is allocated.*

Note that each nonempty queue has only *one customer who is eligible to receive service*, the most recently arrived customer. None of the other customers in the same queue will be served, whether the first customer is skipped or not. If the most recent arrival in the sorted list is skipped, the second customer considered may not be the second most recent arrival into the entire system, rather it is the most recent arrival into one of the other queues. In the skipping vector, setting $s_\omega = 1$ ensures that customers from the queue with the highest expected damage under the LCFS policy will not be skipped, as they would not be under the LCFS policy. For the other queues, setting $s_j \leq 1$ for $j \neq \omega$, ensures that sometimes service is not provided in these queues, even if they contain the most recent arrival, and instead is provided in queue ω. The result of this procedure is to reduce the expected damage achieved by the adversary if he joins queue ω under the LCFS

policy at the expense of higher expected damages in the other queues, with the net effect being to reduce the maximal expected damage the adversary could inflict over all queues.

The LCFS-PS policy defines a class of service policies, parametrised by the *skipping vector* $\mathbf{s}$. The extreme case of $\mathbf{s}$ with $s_j = 1$ for $j = 1$ to k represents the LCFS policy, hence the LCFS policy belongs to the class of LCFS-PS policies. We write $W_j(\mathbf{s})$ for the expected damage from the over-crowded state when the adversary joins queue j under the LCFS-PS policy with parameter $\mathbf{s}$. If we restrict the server to only using service policies from the class of LCFS-PS policies, we can seek the best performance guarantee for the system from *within the class of LCFS-PS policies* by solving the following optimisation problem

$$\min_{\mathbf{s}} \max_{j} W_j(\mathbf{s}). \tag{6.11}$$

The best performance guarantee from within the class of LCFS-PS policies is an upper bound for the best performance guarantee over all centralised policies. The optimisation problem given in (6.11) of finding the best performance guarantee within the class of LCFS-PS policies corresponds to *optimising the skipping vector* $\mathbf{s}$.

Problem (6.11) is analogous to the problem of optimising the reselection vector for the DR and SR policies in (6.9) in the case of decentralised control. Our approach to solving (6.11) is analogous to the approach taken to solve (6.9). One difference with the LCFS-PS policy is that the the domain $\mathcal{D}$ over which we define the functions $W_j : \mathcal{D} \to \mathbb{R}$ for $j = 1, ..., k$ are given by $\mathcal{D} = \{\mathbf{s} : \mathbf{s} \in [0, 1]^k$ with $s_w = 1\}$, namely the unit *hypercube in* $k - 1$ *dimensions*. Another key difference is that estimating the expected damages $W_j(\mathbf{s})$ for given vectors $\mathbf{s} \in \mathcal{D}$ through simulation is a far more challenging task for the LCFS-PS policy. Other than this, we will use the same approach of the CORS method for optimising the skipping vector of the LCFS-PS policy.

6.3.2 Simulation Model for LCFS-PS Policy

Estimating the expected damages in each queue under the LCFS-PS policy, namely $W_j(\mathbf{s})$, through simulation is *not straightforward*. To replicate the system experienced by the adversary, firstly we must create an over-crowded state for the adversary to arrive into. Secondly, upon arrival the server must have been serving according to LCFS-PS policy in an over-crowded state for some time to the extent that the service process is in equilibrium. This is required to ensure that the adversary arrives correctly at a random point in the service cycle. The main difficulty in simulating the LCFS-PS policy is the need to allocate the server based on the arrival time of customers. This difficulty is enhanced with an over-crowded state.

We will first give a broad description of the simulation model before focusing on specific details. In the simulation model, we denote the adversary by customer A who arrives into the system at time 0. Customers who arrive before time 0 are referred to as *old customers*, while customers who arrive after time 0 are referred to as *new customers*. At time 0 there are m old customers in the system who arrived before time 0 and did not yet abandon. We want to evaluate the LCFS-PS policy in the over-crowded state, so we will take $m \to \infty$. In the time interval $(-\infty, 0]$, the server uses the LCFS-PS policy. At time 0, the service process is in equilibrium and customer A arrives into one of the queues at some random point during the service cycle. In particular, the server will be engaged in service in a particular queue, with some additional service time remaining. After the arrival of customer A, other new customers also arrive into the system according to independent Poisson processes. After time 0, each time the server becomes available he applies the LCFS-PS policy to select a customer. The simulation continues until customer A is either taken into service or abandons the queue.

Within the simulation, we maintain a state vector for the new customers (including customer A). The state vector includes the arrival time, abandonment time, service requirement, and queue of each customer. We use the state vector

when allocating the server according to the LCFS-PS policy after time 0. Before time 0 and at times when the server skips over all new customers under LCFS-PS, the server is allocated to old customers. It turns out that *we do not need to maintain a state vector for old customers*, and below we explain why.

Consider the k queues without the service process. Each queue can be viewed as an *M/M/∞ queue*, if we interpret the lifetimes until abandonment as the service times. Let N_j denote the number of customers in queue j in steady state for $j = 1$ to k. The random variables $N_1, ..., N_k$ are independent, with N_j having a Poisson distribution with *mean* λ_j/θ_j. One way to see that N_j follows a Poisson distribution is to reset the clock time to 0 at steady state. A queue j customer that arrived at time $t \leq 0$ would still be in the queue at time 0 if his lifetime is greater than $-t$, the probability of which is $e^{\theta_j t}$ for $t \leq 0$. Therefore, we can think of customers in queue j at time 0 as the number of events arriving according to a nonhomogeneous Poisson process in $(-\infty, 0]$, with *intensity function* $\lambda_j e^{\theta_j t}$ for $t \leq 0$. Hence, N_j follows a Poisson distribution with mean given by

$$\int_{-\infty}^{0} \lambda_j e^{\theta_j t} dt = \frac{\lambda_j}{\theta_j}.$$

Let there be m old customers in the system at time 0. The joint probability distribution of $N_1, ..., N_k$, conditional on $\sum_j N_j = m$, follows a multinomial distribution. Write $N_{j,m}(t)$ for the random variable representing the number of queue j old customers at time $t \geq 0$, conditional on there being m customers at time 0. Taking the marginals of the multinomial joint distribution at $t = 0$, we infer that $N_{j,m}(0)$ follows a binomial distribution with parameters m and

$$\frac{\lambda_j/\theta_j}{\sum_{i=1}^{k} \lambda_i/\theta_i}.$$

In addition, conditional on $N_{j,m}(0) = n_{j,m}(0)$, the random variable $N_{j,m}(t)$ follows a binomial distribution with parameters $n_{j,m}(0)$ and $e^{-\theta_j t}$. The random variable

$N_{j,m}(t)$ follows a binomial distribution with parameters m and

$$q_j \equiv \frac{\lambda_j/\theta_j}{\sum_{i=1}^{k}(\lambda_i/\theta_i)} e^{-\theta_j t}.$$

Conditional on $N_{j,m}(t) = n_{j,m}(t)$, the joint distribution of the arrival times of the old customers in queue j at time t is the *order statistics* of $n_{j,m}(t)$ independent random variables with density function $\theta_j e^{\theta_j} t$ for $t \leq 0$. This is the exponential distribution with rate θ_j, when we stand at time 0 and look backwards in time.

Suppose there are m old customers at time 0. Write $A_{j,m}(t)$ for the event that the most recent old customer is in queue j at time t. Let $\mathbf{N}_m(t)$ be the joint distribution of the $N_{j,m}(t)$, with $\mathbf{n}_m(t)$ a realisation of this joint distribution. The probability of $A_{j,m}(t)$, conditional on $\mathbf{n}_m(t)$, is derived from the fact that the minimum of a set of exponential random variables is also exponential. We are interested in the probability that the minimum of these is from queue j. Hence, we have

$$P\{A_{j,m}(t)|\mathbf{N}_m(t) = \mathbf{n}_m(t)\} = \frac{\theta_j n_{j,m}(t)}{\sum_{i=1}^{k} \theta_i n_{i,m}(t)}.$$

We wish to determine

$$\lim_{m\to\infty} P\{A_{j,m}(t)\} = \lim_{m\to\infty} E\left[\frac{\theta_j N_{j,m}(t)}{\sum_{i=1}^{k} \theta_i N_{i,m}(t)}\right].$$

Since $N_{j,m}(t)$ follows a binomial distribution with parameters m and q_j, using the law of large numbers, as $m \to \infty$,

$$\frac{\theta_j N_{j,m}(t)}{m} \to \theta_j q_j \tag{6.12}$$

almost surely. Similarly, as $m \to \infty$,

$$\frac{\sum_{i=1}^{k} \theta_i N_{i,m}(t)}{m} \to \sum_{i=1}^{k} \theta_i q_i$$

almost surely, and so

$$\frac{m}{\sum_{i=1}^{k} \theta_i N_{i,m}(t)} \to \frac{1}{\sum_{i=1}^{k} \theta_i q_i} \tag{6.13}$$

almost surely.

Putting equations (6.12) and (6.13) together we get

$$\frac{\theta_j N_{j,m}(t)}{\sum_{i=1}^{k} \theta_i N_{i,m}(t)} = \left(\frac{\theta_j N_{j,m}(t)}{m}\right)\left(\frac{m}{\sum_{i=1}^{k} \theta_i N_{i,m}(t)}\right) \to \frac{\theta_j q_j}{\sum_{i=1}^{k} \theta_i q_i}$$

almost surely, as $m \to \infty$. Since the sequence of random variables

$$\frac{\theta_j N_{j,m}(t)}{\sum_{i=1}^{k} \theta_i N_{i,m}(t)}, \qquad m = 1, 2, \ldots$$

converges almost surely to

$$\frac{\theta_j q_j}{\sum_{i=1}^{k} \theta_i q_i},$$

it follows that the sequence of real numbers

$$E\left[\frac{\theta_j N_{j,m}(t)}{\sum_{i=1}^{k} \theta_i N_{i,m}(t)}\right], \qquad m = 1, 2, \ldots$$

converges to

$$\frac{\theta_j q_j}{\sum_{i=1}^{k} \theta_i q_i},$$

or equivalently,

$$\lim_{m\to\infty} E\left[\frac{\theta_j N_{j,m}(t)}{\sum_{i=1}^{k} \theta_i N_{i,m}(t)}\right] = \frac{\theta_j q_j}{\sum_{i=1}^{k} \theta_i q_i}.$$

Finally, we can conclude that

$$\begin{aligned}
\lim_{m\to\infty} P\{A_{j,m}(t)\} &= \lim_{m\to\infty} E\left[\frac{\theta_j N_{j,m}(t)}{\sum_{i=1}^{k} \theta_i N_{i,m}(t)}\right] \\
&= \frac{\theta_j q_j}{\sum_{i=1}^{k} \theta_i q_i} \\
&= \frac{\lambda_j e^{-\theta_j t}}{\sum_{i=1}^{k} \lambda_i e^{-\theta_i t}}.
\end{aligned} \tag{6.14}$$

Further to the probability that the most recent customer being in queue j at time

t being given by (6.14), we are also interested in the *identity* of the second most recent customer and so forth. Suppose, at time t, we sort all old customers by their arrival times and try to identify each customer, then as $m \to \infty$, each customer is independently from queue j with probability given by (6.14).

In the simulation model we let the server start to work at a time before 0 long enough so that the server will go through many service cycles before customer A arrives at time 0. The actual time used depends on the customer lifetime and service time distributions. Each time the server needs to select a customer according to the LCFS-PS policy, we use the probabilities in (6.14) with $t = 0$ to create a temporary ordered list to make this allocation. The ordered list corresponds to the order in which the most recent old customers in each queue arrived, with the most recent in the system being first. We can use the probabilities in (6.14) with $t = 0$ for all time points $t < 0$, since there is no difference between the overcrowded state at $t = 0$ and $t < 0$. Further, although these probabilities were derived by considering the system without the service process, since $m \to \infty$, including the service process would not change the ordered list based on (6.14).

After the server has been allocated to a customer, we generate the time at which the server will next become available as the minimum of the lifetime and service time of the customer in service and move the clock time to this point. We repeat this process of allocation under the LCFS-PS policy and advance the simulation clock until the clock reaches time 0. At this point we record the time at which the server will next become available for service. At time 0, customer A arrives into one of the queues. We add customer A to a state vector of new customers and generate new customers who arrive before the abandonment time of customer A and add these to the state vector of new customers. From the time at which the server is next available, the server uses the LCFS-PS policy, until either customer A abandons or enters service.

We point out here that a simulation model for the DR and SR policies adopts the same structure as described here. However, the DR and SR simulation models

are significantly simpler to implement. At all times, we can allocate the server to a queue simply using a probability vector, without the need to use ordered lists. After time 0, we only need to maintain a state vector of customers in the same queue as customer A in order to appropriately select the correct customer if the server is allocated to this queue. At no point does the server skip a customer or do we need to create an ordered list of old customers. These aspects of the simulation models make them *comparatively simpler*.

Recall that our goal was to estimate the expected damages $W_j(\mathbf{s})$ for $j = 1, ..., k$. For a fixed queue j, let $W_j^i(\mathbf{s})$ be an estimate of the expected damage from replication i of the simulation model if the adversary joins queue j. We estimate $W_j(\mathbf{s})$ as the sample mean of a fixed number of replications of the simulation model and repeat this process for $j = 1, ..., k$. To obtain the estimates from a single replication for a fixed queue j, we consider 10,000 realisations of the system and record the proportion of the realisations in which customer A is taken into service. These proportions estimate the probability of the adversary entering into service. Denote these proportions by e_j. Let Z_j be a random variable representing the service time in queue j. Once in service, the probability of completing service is given by $E[e^{-\theta_j Z_j}]$. The estimate of the expected damage in a single replication is then given by

$$d_j\big(1 - e_j E[e^{-\theta_j Z_j}]\big).$$

6.3.3 Computing the Best LCFS-PS Policy

Recall the CORS method described in Section 6.2.2 for optimising the reselection vector for the DR policy. We can apply the CORS method in a similar way to find the best skipping vector for the LCFS-PS policy. We summarise our application of the CORS method for the LCFS-PS policy as follows:

1. Select n well-spread initial points $\mathbf{s}_i$ for $i = 1, ..., n$ using a lattice point set method. Let $S = \{\mathbf{s}_1, ..., \mathbf{s}_n\}$.

2. For $\mathbf{s} \in S$, simulate the LCFS-PS policy with skipping vector s, and record the expected damages in each queue $W_j(\mathbf{s})$ for $j = 1, ..., k$.

   Repeat steps 3 to 6 until some stopping condition is met:

3. Fit k response surface models $\tilde{W}_j$ for each queue j using the data, $j = 1, \ldots, k$. We use a radial basis function interpolation for each response surface.

4. Compute the maximin point

$$\delta = \max_{\mathbf{s} \in \mathcal{D}} \min_{s_i \in S} ||\mathbf{s} - \mathbf{s}_i||$$

5. Select a candidate point $\mathbf{s}_{\text{new}}$ for simulation by solving the following constrained approximate optimisation problem

$$\min_{\mathbf{s} \in \mathcal{D}} \max_{j} \tilde{W}_j(\mathbf{s}) \tag{6.15}$$

$$\text{Subject to} \quad ||\mathbf{s} - \mathbf{s}_i|| \geq \beta\delta \quad \text{for all } \mathbf{s}_i \in S$$

   where $\beta \in [0, 1]$ is a predetermined constant.

6. Add the candidate point $\mathbf{s}_{\text{new}}$ to S, simulate its performance $W_j(\mathbf{s}_{\text{new}})$ for each queue j, update the data, and go to Step 3.

In step 1, the well-spread initial points are selected using a lattice point set method. This is the same lattice point set method described in Chapter 4 in the context of the API algorithm. We encourage the reader to refer to Chapter 4 for details. All other aspects of the method are the same as described in Section 6.2.2.

Once again, as we did for the DR and SR policies, we can potentially improve upon the performance guarantee offered by the LCFS-PS with the final skipping vector in the CORS method. We can formulate a finite TPZS matrix game in which pure strategies of the server correspond to the set of LCFS-PS policies developed in the CORS method. The procedure for adding more pure strategies to the

matrix game which may not lie on the grid used in the CORS method is described in Section 6.2.2. Overall, our heuristic approach is to adopt the optimal mixed strategy from the finite TPZS matrix game resulting from the CORS method followed by the iterative extension in Section 6.2.2. The value of the resulting matrix game provides a performance guarantee for the server.

6.4 Numerical Examples

In this section we consider a set of numerical examples to assess the relative performance of the heuristic approaches to (6.7) developed in Sections 6.2.1 to 6.2.2 and Sections 6.3.1 to 6.3.3. We begin by considering an example in a $k = 3$ queue system in order to illustrate the CORS method in more detail.

EXAMPLE 6.1: Consider the following example system in which there are $k = 3$ queues. Service is nonpreemptive and the service time distributions in each queue follow independent Gamma distributions. The shape and rate parameters, a_j and b_j respectively, characterise the service time distribution in queue j, hence expected service times are given by a_j/b_j. The parameters of the system are as follows:

$$(d_1, d_2, d_3) = (1, 1, 1)$$

$$(\lambda_j, a_j, b_j, \theta_j) = \begin{cases} (4, 3, 20, 4) & j = 1 \\ (3, 1, 10, 0.5) & j = 2 \\ (2, 2, 14, 1.5) & j = 3 \end{cases}$$

We will demonstrate the application of the CORS method for finding the best skipping vector for the LCFS-PS heuristic, discussed in Section 6.3.3. Firstly, we use the simulation method discussed in Section 6.3.2 to estimate the expected damages in each queue from an over-crowded state under the LCFS policy. This corresponds to the LCFS-PS policy with skipping vector $\mathbf{s} = (s_1 = 1, s_2 = 1, s_3 = 1)$. Since the damages in each queue are equal to one, the expected damages are equivalent to the abandonment probabilities of the adversary. The vector of

estimated abandonment probabilities in each queue under the LCFS policy is given by $(W_1(\mathbf{s}), W_2(\mathbf{s}), W_3(\mathbf{s})) = (0.6635, 0.1556, 0.3863)$. The estimated abandonment probability the server can guarantee by using the LCFS policy is 0.6635. Since the solution of $\omega = \text{argmax}_j W_j(\mathbf{s})$ is given by $\omega = 1$, we set $s_1 = 1$ in all skipping vectors of the LCFS-PS heuristic. Hence, we seek to find the best skipping vector over the domain $\mathcal{D} = \{\mathbf{s} : \mathbf{s} \in [0,1]^2 \text{ with } s_1 = 1\}$, which is equivalent to optimising the elements s_2 and s_3.

In step 1 of the CORS method, we select $n = 5$ initial skipping vectors using the lattice point set method described in Chapter 4. In the lattice point set method, we generate skipping vectors $\mathbf{s} = (1, s_2, s_3)$, where (s_2, s_3) are given by $((\mathbf{z}j \bmod 5)/5)$ for $0 \leq j \leq 4$ and $\mathbf{z} = (2, 3)$. We set the distance parameter β to cycle over the sequence $(\beta_1, \beta_2, \beta_3, \beta_4) = (0.95, 0.25, 0.03, 0)$. We also set the number of rounds to be 5, where each round consists of 4 iterations. In defining a grid over which we solve the optimisation problems, we take each of s_2 and s_3 in the range from 0 to 1 inclusive, at intervals of 0.01. Hence, the grid consists of 10,201 points.

Figure 6.1 shows the points (s_2, s_3) from the set of skipping vectors $\mathbf{s}$ in S developed through the CORS method at the end of the initial phase and at the end of each round. Figure 6.2 shows the radial basis function approximations $\tilde{W}_j$ for each queue $1 \leq j \leq 3$ at the end of the CORS method, after all 5 rounds. In Figure 6.1, the skipping vectors identified throughout round 1 indicate good approximations from the initial phase which were further refined in round 1. The method made larger refinements in unexplored regions in further rounds and small refinements in the previously explored regions. We see that at the end of each round, in which the method exploits the current approximations, the skipping vectors identified all lie in a small region where the main density of points are situated, with s_2 slightly smaller than 0.2 and s_3 slightly larger than 0.3. From Figure 6.2 we see that in this region, the approximations $\tilde{W}_j$ are nearly equal. The interpolating nature of the approximations, together with the number of points simulated in the main density, ensure accurate approximations and lead to a firm

belief that the best skipping vector lies in this small region.

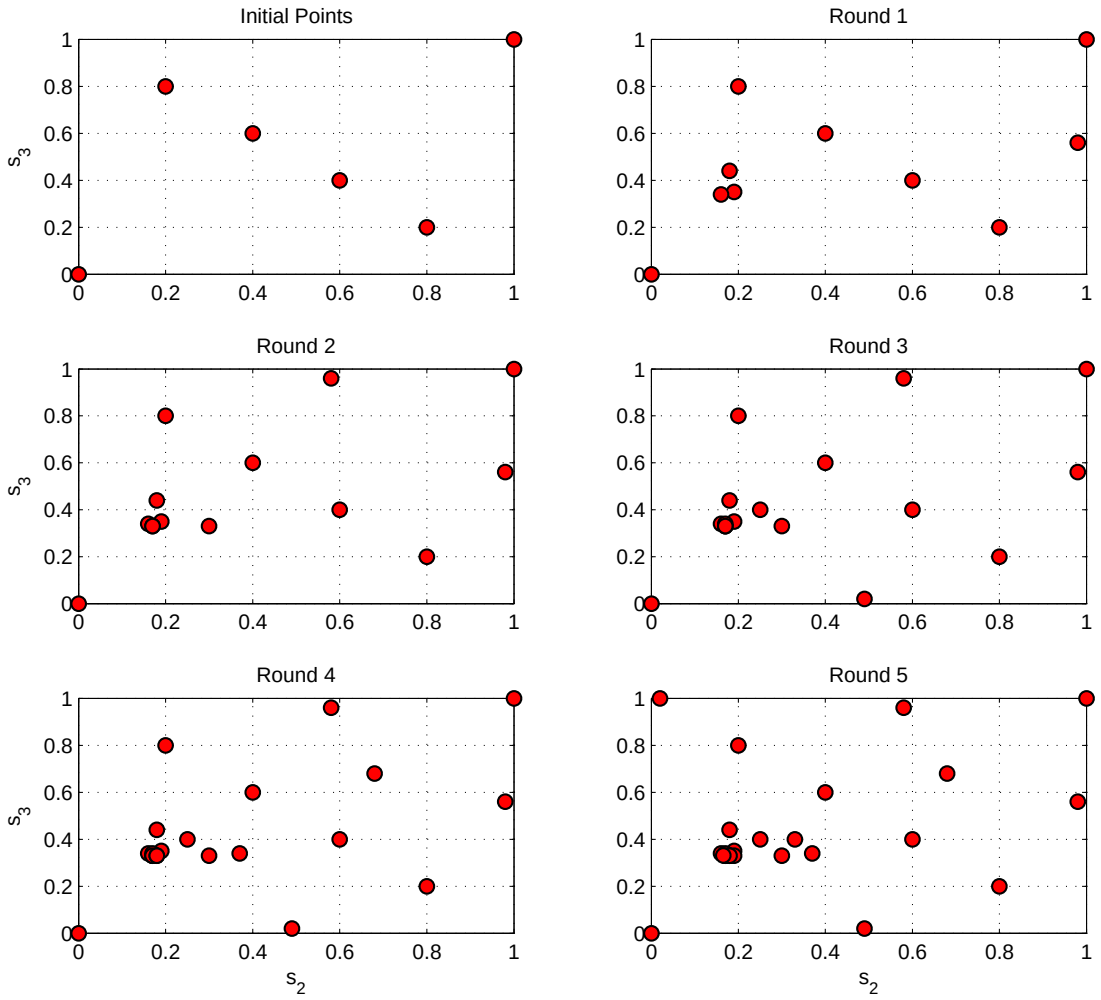

Figure 6.1: Points (s_2, s_3) from the set of skipping vectors $\mathbf{s}$ in S developed in the initial phase and each round of the CORS method for the LCFS-PS policy in Example 6.1.

The final skipping vector identified by the method was $(s_2, s_3) = (0.18, 0.33)$ and the corresponding estimated abandonment probabilities were $(W_1(\mathbf{s}), W_2(\mathbf{s}), W_3(\mathbf{s})) = (0.6134, 0.5838, 0.6124)$. The estimated abandonment probability the server can guarantee by using the LCFS-PS policy with this skipping vector is 0.6134. We can potentially find a better skipping vector which does not lie on the grid by formulating a finite TPZS matrix game, as discussed in Section 6.3.3. The skipping vector derived from the solution of the finite matrix game was $(s_2, s_3) = (0.1655, 0.3311)$ and the corresponding estimated abandonment probabilities were $(W_1(\mathbf{s}), W_2(\mathbf{s}), W_3(\mathbf{s})) = (0.6133, 0.6105, 0.6109)$. The server can guarantee an esti-

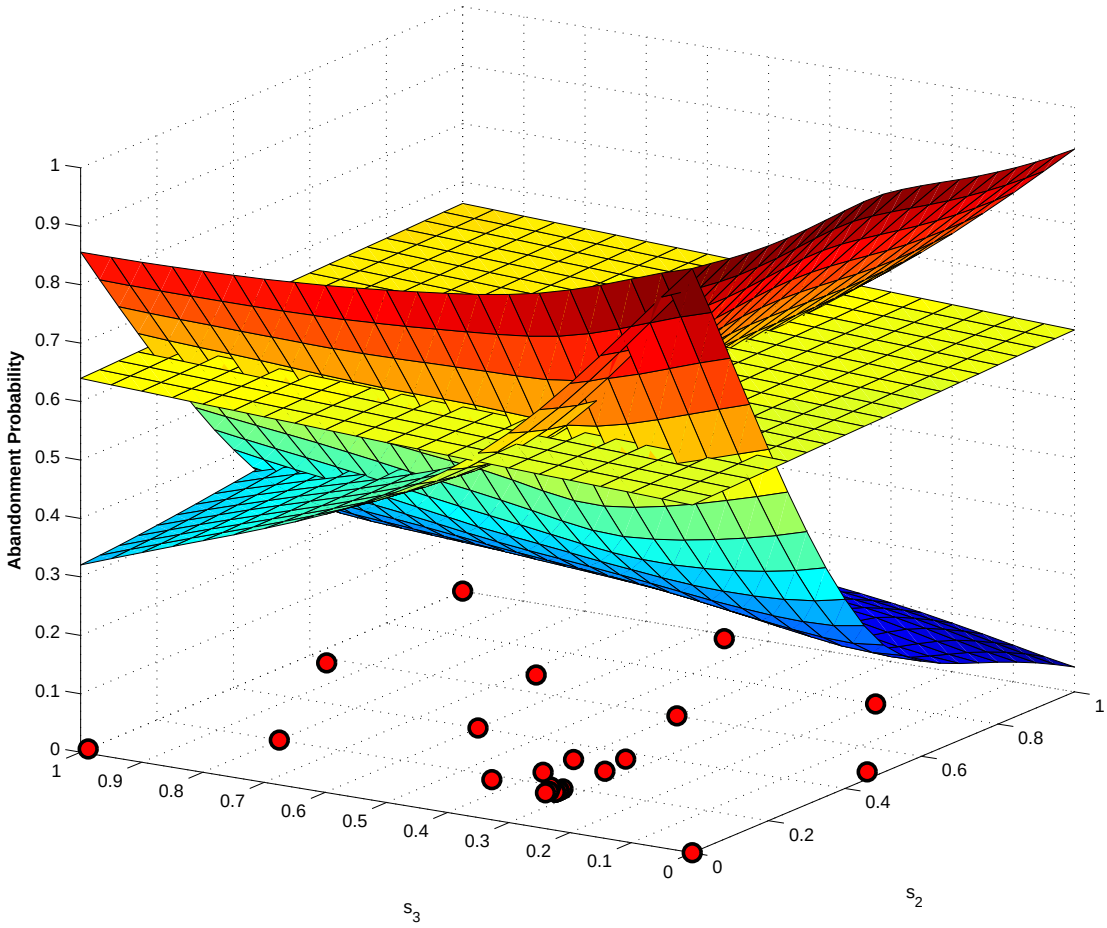

Figure 6.2: Approximations $\tilde{W}_j$ for each queue $1 \leq j \leq 3$ at the end of 5 rounds of the CORS method in Example 6.1. Also shown are the points (s_2, s_3) from the set of skipping vectors $\mathbf{s}$ in S developed in the CORS method.

mated abandonment probability of 0.6133 with this skipping vector. In fact, using the optimal mixed strategy for the server in the finite matrix game consisting of all LCFS-PS policies developed, the server can achieve an estimated abandonment probability of 0.6132. This is an improvement of 7.6% over the LCFS policy. If we formulate the matrix game following the initial phase of the method, we can guarantee an estimated abandonment probability of 0.6211, which shows that the subsequent rounds of the CORS method deliver an improvement of 1.3% over this alternative.

We will now consider wider sets of numerical examples for $k = 3$ queues, within which we will estimate the performance of a number of heuristic policies. Specifically, we will consider the following heuristics which were discussed throughout

the chapter: LCFS, LCFS-PS, DR, SR, and RR. In addition, we will estimate the performance of a heuristic we will label as PA. The PA heuristic formulates a finite TPZS matrix game in which the server's pure strategies correspond to the k policies which prioritise each queue. Prioritisation can be achieved by setting a reselection vector $\mathbf{r}$ in the DR policy with a single element equal to one and all other elements equal to zero. The PA heuristic is analogous to the heuristic policy developed and shown to perform well in the surveillance problem in Chapter 5, which we also labelled PA.

We are *unable to identify optimal service policies*, or indeed the corresponding best performance guarantees, in both cases of decentralised and centralised control. There is also *no known lower bound* for the best performance guarantees of optimal policies. Consequently, we will focus on a *comparison* between the performance of each of our heuristic policies. When referring to the performance of a given heuristic, this corresponds to the estimated expected damage which can be guaranteed by the heuristic. Whilst the LCFS-PS, DR, and SR heuristics represent classes of service policies and individual policies within these classes provide performance guarantees, we are interested in the best performance guarantee we are able to achieve within each class. Subsequently, as discussed in Sections 6.2.2 and 6.3.3, for the LCFS-PS, DR, and SR heuristics, we develop policies using the CORS method and an iterative extension and the performance guarantee is the value of the resulting finite TPZS matrix game. For the policies LCFS and RR, the performance guarantees are simply the largest expected damages among the queues under each policy. As discussed above, the performance guarantee of the PA policy corresponds to the value of the associated finite TPZS matrix game.

In what follows, we make the following choices in our application of the CORS method for the LCFS-PS, DR, and SR heuristics. Firstly, for the LCFS-PS heuristic, we will apply the CORS method in the same manner as in the preceding example system. That is, we fix ω by simulating the LCFS policy. For simplicity, suppose $\omega = 1$. We then generate $n = 5$ initial skipping vectors $\mathbf{s} = (1, s_2, s_3)$

using the lattice point set method described in Chapter 4, where (s_2, s_3) are given by $((\mathbf{z}j \bmod 5)/5)$ for $0 \leq j \leq 4$ and $\mathbf{z} = (2, 3)$. We set the distance parameter β to cycle over the sequence $(\beta_1, \beta_2, \beta_3, \beta_4) = (0.95, 0.25, 0.03, 0)$ and set the number of rounds to be 5, where each round consists of 4 iterations. In defining a grid over which we solve the optimisation problems, we take each of s_2 and s_3 in the range from 0 to 1 inclusive, at intervals of 0.01. For both the DR and SR heuristics, we use the same number of rounds and β cycle. However, in contrast, we generate $n = 5$ initial reselection vectors $\mathbf{r} = (1 - r_2 - r_3, r_2, r_3)$ by sampling the parameters r_2 and r_3 uniformly from the unit simplex in $k - 1$ dimensions. In defining a grid, we take each of r_2 and r_3 in the range from 0 to 1 inclusive, at intervals of 0.01, and apply the constraint $r_2 + r_3 \leq 1$.

We study three sets of examples with a similar design, with each set containing nine scenarios reflecting different service time and lifetime conditions. In all of our examples, service is nonpreemptive and the service time distributions in each queue follow independent *Gamma distributions*. In describing the parameters of the example systems, we denote the arrival rates in each queue by the vector $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \lambda_3)$, the abandonment rates in each queue by the vector $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$, and the damages in each queue by the vector $\mathbf{d} = (d_1, d_2, d_3)$. The service time distribution in queue j is characterised by the shape and rate parameter pair (a_j, b_j), hence we denote the service time distribution in each queue by the vector $\mathbf{g} = ((a_1, b_1), (a_2, b_2), (a_3, b_3))$.

In the first set of examples, there are three cases for the service time distribution vector $\mathbf{g}$ (labelled case 1 to 3) and three cases for the abandonment rate vector $\boldsymbol{\theta}$ (labelled case A to C). Every combination of service time case and abandonment rate case leads to nine distinct scenarios, labelled A1 to C3. The parameters in

this set of examples are as follows:

$$\boldsymbol{\lambda} = (4, 2, 3) \quad \text{(all scenarios)}; \tag{6.16}$$

$$\mathbf{d} = (1, 1, 1) \quad \text{(all scenarios)}; \tag{6.17}$$

$$\mathbf{g} = ((3, 20), (1, 10), (2, 14)) \quad \text{(service case 1)}; \tag{6.18}$$

$$\mathbf{g} = ((3, 10), (1, 5), (2, 7)) \quad \text{(service case 2)}; \tag{6.19}$$

$$\mathbf{g} = ((3, 5), (2, 5), (4, 7)) \quad \text{(service case 3)}; \tag{6.20}$$

$$\boldsymbol{\theta} = (4, 2, 3) \quad \text{(abandonment case A)};$$

$$\boldsymbol{\theta} = (2, 1, 3/2) \quad \text{(abandonment case B)};$$

$$\boldsymbol{\theta} = (2/3, 1/3, 1/2) \quad \text{(abandonment case C)};$$

We refer to this set of examples as the (2:1) set, since this reflects the ratio of the largest to smallest abandonment rates in the vector $\boldsymbol{\theta}$ for all scenarios. This ratio is used as an intuitive measure of asymmetry between the queues in the system. In each scenario in the (2:1) set, the queue with the shortest expected service times also has the smallest arrival rate and longest lifetimes. Conversely, the queue with the longest expected service times also has the largest arrival rate and shortest lifetimes. In selecting these parameters, we attempt to reflect relative conditions which may occur in a practical surveillance setting. For example, one area such as a corridor through which suspects pass quickly, making them more difficult to screen, compared to another area such as a lobby through which suspects pass slowly, making them easier to screen. The estimated performance guarantees of each heuristic in the (2:1) set of examples are reported in Table 6.1.

In our second set of examples, we use the same arrival rate and damage vectors and the same cases for the service time distribution vector. We alter the abandonment rate vectors $\boldsymbol{\theta}$ in our definition of the abandonment cases A to C. Again, we create nine distinct scenarios, A1 to C3. We refer to this set of examples as the (4:1) set, once again to reflect the ratio of the largest to smallest abandonment rates in the vector $\boldsymbol{\theta}$ for all scenarios. The parameters in the (4:1) set of examples

Scenario	LCFS	LCFS-PS	DR	SR	RR	PA
A1	0.6444	0.6208	0.7047	0.7272	0.7510	0.8292
		3.67	*-9.35*	*-12.85*	*-16.55*	*-28.69*
A2	0.8325	0.8099	0.8515	0.8692	0.8908	0.8988
		2.71	*-2.28*	*-4.40*	*-7.00*	*-7.97*
A3	0.9407	0.9276	0.9435	0.9528	0.9636	0.9567
		1.39	*-0.30*	*-1.28*	*-2.43*	*-1.70*
B1	0.4918	0.4617	0.5653	0.5800	0.6185	0.7718
		6.12	*-14.94*	*-17.92*	*-25.76*	*-56.91*
B2	0.7316	0.6971	0.7555	0.7698	0.8087	0.8405
		4.71	*-3.27*	*-5.22*	*-10.54*	*-14.89*
B3	0.8892	0.8649	0.8880	0.8978	0.9230	0.9127
		2.74	*0.14*	*-0.96*	*-3.79*	*-2.64*
C1	0.3170	0.2885	0.3800	0.3847	0.4558	0.7117
		8.98	*-19.89*	*-21.37*	*-43.79*	*-124.56*
C2	0.6364	0.6126	0.6371	0.6406	0.7162	0.7641
		3.75	*-0.11*	*-0.65*	*-12.53*	*-20.06*
C3	0.8334	0.8173	0.8207	0.8223	0.8703	0.8463
		1.92	*1.52*	*1.33*	*-4.43*	*-1.55*

Table 6.1: Estimated abandonment probability under each heuristic in the (2:1) set of examples. We use the LCFS policy as a reference and show the percentage improvement of each heuristic over the LCFS policy in italics.

are given by (6.16) to (6.20), together with the following abandonment cases:

$$\boldsymbol{\theta} = (4, 1, 2) \qquad \text{(abandonment case A)};$$

$$\boldsymbol{\theta} = (2, 1/2, 1) \qquad \text{(abandonment case B)};$$

$$\boldsymbol{\theta} = (2/3, 1/6, 1/3) \qquad \text{(abandonment case C)};$$

The (4:1) set represents example systems which have greater asymmetry between the queues than the (2:1) set examples. The estimated performance guarantees of each heuristic in the (4:1) set of examples are reported in Table 6.2.

In our final set of examples, once again we create nine distinct scenarios, A1 to C3, using the same parameters as used in the (2:1) and (4:1) example sets. Once again, we alter the abandonment rate vectors $\boldsymbol{\theta}$ in our definition of the abandonment cases A to C. We refer to this set of examples as the (8:1) set, once

Scenario	LCFS	LCFS-PS	DR	SR	RR	PA
A1	0.6563	0.6156	0.6760	0.6925	0.7607	0.8119
		6.20	*-3.01*	*-5.51*	*-15.92*	*-23.70*
A2	0.8456	0.8071	0.8367	0.8495	0.8995	0.8849
		4.56	*1.05*	*-0.46*	*-6.37*	*-4.65*
A3	0.9486	0.9255	0.9356	0.9423	0.9684	0.9476
		2.44	*1.38*	*0.67*	*-2.09*	*0.11*
B1	0.5036	0.4521	0.5335	0.5443	0.6260	0.7572
		10.22	*-5.96*	*-8.09*	*-24.31*	*-50.37*
B2	0.7476	0.6904	0.7367	0.7470	0.8175	0.8247
		7.65	*1.46*	*0.08*	*-9.35*	*-10.31*
B3	0.9001	0.8601	0.8772	0.8840	0.9292	0.9002
		4.44	*2.55*	*1.79*	*-3.23*	*-0.01*
C1	0.3265	0.2756	0.3547	0.3582	0.4596	0.7042
		15.59	*-8.64*	*-9.70*	*-40.77*	*-115.68*
C2	0.6475	0.6102	0.6274	0.6298	0.7216	0.7526
		5.77	*3.11*	*2.74*	*-11.43*	*-16.23*
C3	0.8402	0.8146	0.8167	0.8174	0.8746	0.8368
		3.04	*2.79*	*2.71*	*-4.10*	*0.40*

Table 6.2: Estimated abandonment probability under each heuristic in the (4:1) set of examples. We use the LCFS policy as a reference and show the percentage improvement of each heuristic over the LCFS policy in italics.

again to reflect the ratio of the largest to smallest abandonment rates in the vector $\boldsymbol{\theta}$ for all scenarios. The parameters in the (8:1) set of examples are given by (6.16) to (6.20), together with the following abandonment cases:

$$\boldsymbol{\theta} = (4, 1/2, 3/2) \qquad \text{(abandonment case A)};$$

$$\boldsymbol{\theta} = (2, 1/4, 3/4) \qquad \text{(abandonment case B)};$$

$$\boldsymbol{\theta} = (2/3, 1/12, 3/12) \qquad \text{(abandonment case C)};$$

The (8:1) set represents example systems which have greater asymmetry between the queues than the (4:1) and (2:1) set examples. The estimated performance guarantees of each heuristic in the (8:1) set of examples are reported in Table 6.3.

Observing the results in Tables 6.1 to 6.3, we see that the *LCFS-PS heuristic*

Scenario	LCFS	LCFS-PS	DR	SR	RR	PA
A1	0.6635	0.6130	0.6625	0.6751	0.7657	0.8020
		7.61	*0.16*	*-1.74*	*-15.40*	*-20.86*
A2	0.8532	0.8059	0.8306	0.8399	0.9045	0.8766
		5.54	*2.65*	*1.55*	*-6.02*	*-2.75*
A3	0.9531	0.9245	0.9324	0.9373	0.9712	0.9428
		2.99	*2.17*	*1.66*	*-1.90*	*1.08*
B1	0.5101	0.4473	0.5182	0.5270	0.6302	0.7492
		12.33	*-1.58*	*-3.31*	*-23.53*	*-46.85*
B2	0.7562	0.6877	0.7289	0.7373	0.8228	0.8155
		9.05	*3.60*	*2.49*	*-8.81*	*-7.85*
B3	0.9056	0.8582	0.8730	0.8785	0.9326	0.8933
		5.24	*3.60*	*3.00*	*-2.97*	*1.36*
C1	0.3317	0.2693	0.3432	0.3462	0.4618	0.7002
		18.82	*-3.48*	*-4.38*	*-39.21*	*-111.10*
C2	0.6533	0.6093	0.6247	0.6265	0.7244	0.7462
		6.74	*4.38*	*4.11*	*-10.89*	*-14.22*
C3	0.8432	0.8133	0.8152	0.8159	0.8769	0.8318
		3.55	*3.32*	*3.23*	*-4.00*	*1.35*

Table 6.3: Estimated abandonment probability under each heuristic in the (8:1) set of examples. We use the LCFS policy as a reference and show the percentage improvement of each heuristic over the LCFS policy in italics.

achieves the best performance guarantee in every example scenario. This is not surprising given that we developed the heuristic to improve upon the LCFS policy, which it significantly does. The observed improvement over the LCFS policy is expected in the asymmetric scenarios studied, since the LCFS policy is only optimal in symmetric systems. Moreover, we would *expect centralised service policies to outperform decentralised service policies*, which underpins the superiority of LCFS-PS over DR and SR. The superiority of the centralised LCFS policy over the decentralised DR and SR heuristics in many cases strengthens this observation. However, the DR and SR heuristics outperform the LCFS policy in some scenarios, in part due to the weakness of the LCFS policy and in part due to the strength of the DR and SR heuristics in those scenarios.

If the only feasible service policies in the system were decentralised policies, we see that the *DR heuristic achieves the smallest performance guarantee in every*

example scenario, outperforming SR, RR, and PA. The superior performance of DR compared to SR indicates that it is better for the server to potentially switch queues at the conclusion of each attempted service rather than remain in the same queue and attempt to complete a service there. The observed improvement of DR and SR over the RR policy is expected in the asymmetric scenarios studied, since the RR policy is only suitable for entirely symmetric systems. The poor performance of the PA heuristic illustrates a *fundamental difference* between the surveillance problems in this chapter and in Chapter 5. In this chapter, the PA heuristic performs poorly when the adversary can choose when and where to attack, whereas in Chapter 5 an analogous heuristic to PA performs well when the adversary can only choose where to attack. In Chapter 5, the adversary arrives into a steady state system, so the priority policies within PA still serve all queues during the lifetime of the adversary. Here, we look at the overly-crowded state, so the priority policies within PA only serve one queue. Hence, if the adversary selects any other queue, his abandonment probability is 1. This underlines the poor performance of PA here.

The performance guarantee of the LCFS policy increases in each scenario from the (2:1) set of examples to the (8:1) set of examples. In each scenario, the performance guarantee is derived from the adversary attacking the first queue. Increasing the amount of asymmetry from (2:1) to (8:1), θ_1 is the same across the sets, whereas the abandonment rates in the other queues are smaller. Consequently, under the LCFS policy, the server spends more time serving in the other queues in the examples with greater asymmetry, which leads to a higher abandonment probability in the first queue. In contrast, the performance guarantee of the LCFS-PS heuristic decreases from (2:1) to (8:1). This is due to the greater amount of asymmetry increasing the amount to which the LCFS-PS heuristic can reduce the abandonment probability in the first queue at the expense of increasing the abandonment probabilities in the other queues. The RR policy exhibits similar behaviour to the LCFS policy and similarly, the DR and SR heuristics exhibit similar behaviour to the LCFS-PS heuristic. *More asymmetry between the queues*

increases the effectiveness of the DR and SR heuristics.

In conclusion, while it is simple to adopt the LCFS policy in a centralised system and the RR policy in a decentralised system, these policies can perform relatively poorly compared to carefully designed heuristics. Comparison of the LCFS-PS heuristic with the LCFS policy in a centralised system and the DR heuristic with the RR policy in a decentralised system, illustrate the large improvements which can be made in performance. We recommend the LCFS-PS heuristic for centralised control systems and the DR heuristic for decentralised control systems. Furthermore, in making these recommendations, through our application of the CORS method and formulation as a finite TPZS matrix game, we are confident that we obtain a guaranteed performance *close to the the best* performance guarantee possible within the respective classes of the LCFS-PS and DR policies. If it was possible to upgrade the security system from decentralised control to centralised control, the comparative performance of the LCFS-PS and DR heuristics indicates the performance improvement which could be obtained. This improvement could then be used to decide whether it was worth upgrading, given such an upgrade may require a costly investment.

Conclusion

In this chapter we have studied the defensive surveillance scenario of a strategic adversary who chooses both where and when to attack the system. The server wishes to find a robust service policy which provides a performance guarantee against any choice of queue and time the adversary could make. In practice, the adversary may not be as capable as we assume in the chapter, hence the analysis provides an upper bound on the abandonment probability of a less capable adversary. In considering variations of the problem consisting of a single queue, we proved that the last-come first-served policy (LCFS) is optimal for the security team. In adopting the LCFS policy, the security team would track the arrival times of suspects into the public area and screen the suspect who arrived most

recently. The strength of the LCFS policy resides in the fact that in ensures the abandonment probability of the adversary depends entirely on the arrival process after he joins the queue. This highlights the importance of considering which customer to serve when the adversary has knowledge of the state of the system, whilst the LCFS policy seeks to remove the value of this information for the adversary. A consequence of this analysis is that the LCFS policy is also optimal in a system with multiple symmetric queues. In such a system, the security team should screen the most recent arrival across all of the public areas.

In systems with multiple asymmetric queues, we developed heuristic policies based on the strength of the LCFS policy which provide upper bound performance guarantees for the server. Within the approaches developed, an important feature is that the server randomises his actions. We developed the Departure Reselection (DR) and last-come first-served with probabilistic skipping (LCFS-PS) policies, each parametrised by vectors. In adopting the DR policy, the security team first randomly selects an area to screen according to a probability distribution, then screens a suspect according to LCFS. In adopting the LCFS-PS policy, the security team orders the most recent arrivals across all areas, then selects a suspect to screen based on a possible sequence of random decisions governed by a probability rule. In both cases, we found the best performance guarantees for the server from within the policy classes by optimising their parameter vectors. This was acheived through a method which intelligently uses simulation within a response surface method for global optimisation problems. Sets of numerical experiments demonstrate the superior performance of the heuristic approach based on LCFS-PS over approaches based on DR and other policies. Consequently, our suggestion for the security team is to adopt the approach based on LCFS-PS in some cases, otherwise adopt the approach based on DR in other cases.

It is unclear whether the methodology in this chapter could be extended to other defender-attacker problems. Our methodology centred around the consideration of simpler single queue problems to gain insights into the more complex

multiple queue problems. It would seem this would be a valuable approach in other problems which share similar characteristics, notably the insights gained from the LCFS policy in queueing system models where an adversary acts on state information. There are a number of open problems arising from this chapter, for example the consideration of strategic idling in the service policy. It would also be valuable to derive a lower bound on the optimal policy in the asymmetric systems to gain a better understanding of the quality of the heuristic methods developed.

Chapter 7

Conclusion

Our motivation in this thesis was that of how authorities can engage in defensive efforts against the many threats faced in the modern world from adversarial agents. In particular, we considered the use of technology in the defensive surveillance of public areas which are the open, exposed targets of adversarial attacks. Our broad research question asked the following: How should a surveillance resource be utilised in real time to minimise the impact of adversarial threats?

We developed an underpinning surveillance scenario to reflect our motivation. A security team continuously monitors multiple public areas and applies a screening process to people within the areas. However, suspects have lifetimes within the areas, after which they leave the area if they have not yet been screened. The capability of the security team is such that only one suspect can be screened at any given time, hence the loss of suspects is inevitable. Each public area is a target for an adversary who wishes to enter an area, conduct an illicit activity, and leave before being screened. The security team does not know when the adversary is in an area, which area he is in, or which suspect he is and can only detect the adversary through complete screening. The purpose of surveillance is ultimately to detect the adversary before he is able to achieve his goal. This translates our broad research question into the operational problem of identifying a real time decision-making rule for the screening process of the security team to minimise the the probability of the adversary evading detection or the damage he can inflict.

Our approach to the operational problem was to model the surveillance scenario as a multiclass queueing system with customer abandonments. A single server corresponds to the security team, whereby service by the server corresponds to screening by the security team. Customers in the queueing system correspond to suspects, whereby the event of suspects leaving at the conclusion of their lifetimes corresponds to the abandonment of customers from the queueing system. We allow the stochastic distributions of elements of each queue in the queueing system to differ to represent public areas with different characteristics. The adversary is modelled as a potential customer, able to arrive into any queue. The operational problem in our model was to develop control policies within the queueing system which minimise the abandonment probability of the adversary. Control policies from our model are directly equivalent to real time decision-making rules for the screening process of the security team. Hence, the insights gained from our model lead to insights in the security team's operational problem.

Based on the underpinning surveillance scenario and associated multiclass queueing system model, we considered three different surveillance scenarios which apply to a number of potential scenarios which may occur in real-world security operations. We distinguish between the scenarios based on the capability of the adversary and the knowledge of the server regarding the adversary. In Chapters 3 and 4 we considered the scenario of a *random adversary*. In Chapter 5 we considered the scenario of a *strategic adversary who chooses where to attack*. Finally, in Chapter 6 we considered the scenario of a *strategic adversary who chooses where and when to attack*. The adversary is increasingly more capable as we progress through the thesis, representing an increasing threat for the security team. Although sharing many common features, each scenario is inherently different and so the operational strategies suggested for each are different. This illustrates the importance in security operations to first identify the prevailing scenario encountered before deploying an operational defensive surveillance strategy.

The research shares a similar motivation to a number of other defender-attacker

problems, which we discuss in the literature review of Chapter 2; see for example Lin et al. (2013). However, we believe that the defensive surveillance scenarios and the modelling approach used are novel and have not been previously studied. Consequently, the research makes a novel contribution to the wide ranging body of literature of defender-attacker problems. The closest known work is that of Lin et al. (2009), which is the work upon which we base our underpinning surveillance scenario. However, in this work, the focus is on a single public area, whereas the major features of our research are those of multiple public areas and the capability of the adversary to act as a decision-making agent. The random adversary scenario in Chapters 3 and 4 is a special case of a more general stochastic scheduling problem which, together with other variants, has been studied extensively in recent years see Glazebrook et al. (2004), Atar et al. (2010), Ayesta et al. (2011), Down et al. (2011), Verloop (2014) and Larrañaga et al. (2014), as well as Harrison & Zeevi (2004), Kim & Ward (2012), and Ata & Tongarlak (2013)). We develop new results and approaches for this problem which complement the existing literature. We are also the first to make clear the connection with random adversary surveillance scenario, which is a further contribution to the literature.

It is a common feature of each surveillance scenario in each chapter of the thesis that often we can only compute the optimal service policy in systems with a small number of queues or in special cases. Consequently, our focus is on the development of strongly performing heuristic service policies which can be computed by the security team. In the stochastic scheduling problem with customer abandonments in Chapter 3 (of which the random adversary scenario is a special case), we focus on priority service policies. A priority policy known as the $R\mu$ rule is shown in the literature to perform well in overloaded systems. To complement the $R\mu$ rule in the light traffic case, the main contribution of Chapter 3 is to present another priority service policy known as the $R\mu\theta$ rule and prove that it is asymptotically optimal as customer abandonment rates approach zero in light traffic systems. Extensions of this result are discussed for other model classes of interest, namely a multiserver

version of our system and in Klimov Networks. Further to this we develop a priority policy known as the pairwise swapping (PaS) policy. We consider the same problem in Chapter 4, whereby the main contribution is to develop an approximate policy iteration (API) method for the problem which aims to improve the suite of priority policies in Chapter 3. Our numerical results indicate that, in most cases, the best priority policy from Chapter 3 is nearly optimal in systems with 2 or 3 customer classes and we have an effective service policy of simple structure. In the cases where it is not, the API method invariably tightens up the gap substantially and provides an improved policy, albeit of more complex structure and requiring lots of computation time. In our motivating random adversary defensive surveillance application, even small improvements in performance can be of high practical importance. A paper based on the combined work of Chapters 3 and 4 has been published in INFORMS Journal on Computing; see James et al. (2016).

In Chapter 5, since the server and the adversary do not know each others decision, we model their interaction as a simultaneous move two-person zero-sum (TPZS) game. By considering the TPZS game from the perspective of the adversary, for which Kelley's cutting plane (KCP) method applied to a suitably defined convex optimisation problem delivers an optimal solution, we develop the heuristic cutting plane (HCP) and enhanced heuristic cutting plane (HCP^+) methods. These methods are a heuristic application of the KCP method in which we define a set of service policies for the server and iteratively populate this set using the heuristic service policies developed for the random adversary problem. This exploits the strong connection between the random adversary scenario and strategic adversary scenario in Chapter 5. Our suggestion for the security team is to randomise over this set of service policies according to a mixed strategy. Numerical experiments indicate the strong performance of this approach.

Whilst there is a strong connection between the random adversary scenario and the strategic adversary scenario in Chapter 5, the scenario studied in Chapter 6 is very different. This is due to the fact that the adversary no longer attacks

the system in steady state and can time his attack. The first contribution of this chapter is to prove that the last-come first-served (LCFS) policy is optimal in a number of variations of the problem consisting of a single queue. The strength of the LCFS policy is based on the fact that the abandonment probability of the adversary depends entirely on the arrival process after he joins the queue. In practice, the adversary may not be as capable as we assume, hence our analysis provides an upper bound on the abandonment probability of a less capable adversary. The scenario studied in Chapter 6 is something of a worst case scenario for the server. In multiple queue systems, we prove that the LCFS policy applied to multiple queues is optimal in the special case of a symmetric system. This follows from our analysis of single queue systems. In asymmetric systems, we use these insights to develop heuristic policies based on the strength of the LCFS policy. We develop the Departure Reselection (DR), Service Reselection (SR), and last-come first-served with probabilistic skipping (LCFS-PS) policies, which are each parametrised by vectors. In each case, we find the best performance guarantees for the server from within the classes of DR, SR, and LCFS-PS policies by optimising their parameter vectors through a method based on the work by Regis & Shoemaker (2005) which intelligently utilises simulation within a response surface method for global optimisation problems. Sets of numerical examples demonstrate the superior performance of the heuristic approach based on LCFS-PS over the approaches based on DR and SR, and other simpler policies. Our suggestion for the security team is to adopt the heuristic approach based on LCFS-PS in some cases, otherwise adopt the heuristic approach based on DR in other cases.

When we compare the findings of each chapter, we see that our answers to the general research question are quite different in each surveillance scenario. When the server knows the decision of the adversary, at least in a probabilistic sense, and the adversary can only choose which queue to attack, it is often very effective for the server to use a single, deterministic, priority service policy, as seen in the $R\mu$ and $R\mu\theta$ rules. When the server does not know the decision of the adversary,

it becomes important for the server to randomise. This can be via a randomised policy or by randomising over a set of deterministic service policies, which each provide good defence against various decisions the adversary may make, as seen in the HCP method. When the adversary can also time his attack, it is important for the server to consider which customer to serve in addition to which queue to serve, as seen in the LCFS policy. The customer served is not a concern in the scenarios in which the adversary attacks the system in steady state. Serving the most recently arrived customers into the system seeks to remove the advantage of the adversary knowing the state of the system. When the server does not know where or when the adversary will attack, it is also good for the server to randomise his actions, both within service policies and across a set of service policies, as seen in the heuristics based on the DR and LCFS-PS policies.

We recognise the limitations of the research. The surveillance scenarios we have considered are limited by the model assumptions that have been imposed on them. In Chapters 3, 4, and 5 we assume that service times follow an exponential distribution and that service is provided preemptively. In Chapter 6 we study the case in which service times follow arbitrary probability distributions and service is provided nonpreemptively. In real security operations, the model of the screening process may require any combination of these assumptions. Application of the general service time, nonpreemptive setup to the scenarios in Chapters 3, 4, and 5 would require further research. Such further research would be valuable as it would allow the security team to directly compare their performance in each of the three surveillance scenarios, indicating the increased value to the adversary of greater capability.

Throughout the thesis we make the assumption that the lifetimes of customers are exponentially distributed. It would be more realistic to assume an arbitrary probability distribution for lifetimes and this would be a challenging direction of future research. Moreover, we assume that when the adversary joins a queue, his lifetime is distributed in the same way as the other customers in the queue.

In the absence of any information regarding the lifetime of an adversary, this assumption seems a sensible one to make. However, an interesting direction for future research would be to suppose the adversary's lifetime is independent from the system and is the same regardless of which queue he attacks. This problem would be significantly more difficult since, for example if the adversary arrives in steady state, his abandonment probability would no longer be equal to the steady state abandonment probability experienced by any arbitrary customer in that queue.

There are a few aspects of the research which were identified as being outside the scope of the thesis in Chapter 6, for example consideration of strategic idling in the case of nonpreemptive service. These open problems would be an immediate and interesting direction for future research. Other directions for future research involve extending the surveillance scenarios and associated models to make them more realistic or changing them into related scenarios. For example, the server may need to take some time to switch from one queue or customer to another. Also, the screening may not be perfect and there may be some form of overlooking, wherein suspects are not identified with a certain probability. The Klimov Network model in Section 3.4.1 is one example of this type of scenario. Another direction could be to consider surveillance scenarios with multiple servers, perhaps where each server is only responsible for a subset of areas. One obvious limitation of the research is the fact that we study time-homogeneous systems, whereas real security operations may concern public areas with time-varying characteristics. We suggest that a time-varying problem may be initially approached by approximating it with a number of variations of our time-homogeneous problems in which the security team changes their operational strategies to suit in line with our suggestions.

While our broad research motivation was concerned with the defensive efforts of authorities against adversarial agents, our focus has been on the operational aspects of the defensive surveillance part of these efforts. In this thesis, we believe we have studied some interesting and novel defensive surveillance scenarios and

have developed some valuable models, insights, and methods which could inform real security operations. Furthermore, as we have discussed, we believe that our research can be used as the motivation and a basis for a wide range of further avenues of future research. Hopefully this will lead to a greater understanding and provide a greater range of operational strategies which could be used to defend against adversarial agents.

Bibliography

Alpern, S. (1992). Infiltration games on arbitrary graphs. *Journal of Mathematical Analysis and Applications*, 163(1), 286–288.

Alpern, S. & Gal, S. (2003). *The theory of search games and rendezvous*. Norwell, MA: Kluwer Academic Publishers.

Alpern, S., Morton, A., & Papadaki, K. (2011). Patrolling games. *Operations Research*, 59(5), 1246–1257.

Argon, N. T., Ziya, S., & Righter, R. (2008). Scheduling impatient jobs in a clearing system with insights on patient triage in mass casualty incidents. *Probability in the Engineering and Informational Sciences*, 22(3), 301–322.

Ata, B. & Tongarlak, M. H. (2013). On scheduling a multiclass queue with abandonments under general delay costs. *Queueing Systems*, 74(1), 65–104.

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

Atar, R., Giat, C., & Shimkin, N. (2011). On the asymptotic optimality of the $c\mu/\theta$ rule under ergodic cost. *Queueing Systems*, 67(2), 127–144.

Auger, J. M. (1991). An infiltration game on k arcs. *Naval Research Logistics*, 38(4), 511–529.

Avenhaus, R. & Canty, M. J. (2002). Inspection games. In R. A. Meyers (Ed.), *Computational Complexity: Theory, Techniques, and Applications* (pp. 1605–1618). New York, NY: Springer.

Ayesta, U., Jacko, P., & Novak, V. (2011). A nearly-optimal index rule for scheduling of users with abandonment. In *INFOCOM, 2011 Proceedings IEEE* (pp. 2849–2857).

Bassamboo, A., Harrison, J. M., & Zeevi, A. (2005). Dynamic routing and admission control in high-volume service systems: Asymptotic analysis via multi-scale fluid limits. *Queueing Systems*, 51(3-4), 249–285.

Bassamboo, A., Harrison, J. M., & Zeevi, A. (2006). Design and control of a large call center: Asymptotic analysis of an LP-based method. *Operations Research*, 54(3), 419–435.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1), 238–252.

Bertsekas, D. P. (2012). *Dynamic Programming and Optimal Control: Approximate Dynamic Programming*. Belmont, MA: Athena Scientific.

Bertsekas, D. P. & Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

Bhattacharya, P. P. & Ephremides, A. (1989). Optimal scheduling with strict deadlines. *IEEE Transactions on Automatic Control*, 34(7), 721–728.

Bhattacharya, P. P. & Ephremides, A. (1991). Optimal allocation of a server between two queues with due times. *IEEE Transactions on Automatic Control*, 36(12), 1417–1423.

Bier, V. (2006). Game-theoretic and reliability methods in counterterrorism and security. In A. Wilson, G. Wilson, & D. H. Olwell (Eds.), *Statistical Methods in Counterterrorism* (pp. 23–40). New York, NY: Springer.

Boxma, O. J. & Forst, F. G. (1986). Minimizing the expected weighted number of tardy jobs in stochastic flow shops. *Operations Research Letters*, 5(3), 119–126.

Brown, G., Carlyle, M., Harney, R. C., Skroch, E. M., & Wood, K. (2009). Interdicting a nuclear-weapons project. *Operations Research*, 57(4), 866–877.

Brown, G., Carlyle, M., Salmerón, J., & Wood, K. (2006). Defending critical infrastructure. *Interfaces*, 36(6), 530–544.

Brown, G. W. (1949). Some notes on computation of games solutions. RAND Report P-78.

Carlyle, M., Henderson, S. G., & Szechtman, R. (2011). Allocating capacity in parallel queues to improve their resilience to deliberate attack. *Naval Research Logistics*, 58(8), 731–742.

Conitzer, V. & Sandholm, T. (2006). Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM Conference on Electronic Commerce* (pp. 82–90). New York, NY: ACM.

Cox, D. R. & Smith, W. L. (1961). *Queues*. London: Metheun & Co. Ltd.

Dantzig, G. B. (1951). A proof of the equivalence of the programming problem and the game problem. In T. C. Koopmans (Ed.), *Activity Analysis of Production and Allocation*, Cowles Commission Monograph No. 13 (pp. 330–335). New York, NY: John Wiley & Sons Inc.

Dantzig, G. B. (2002). Linear programming. *Operations Research*, 50(1), 42–47.

De, P., Ghosh, J. B., & Wells, C. E. (1991). On the minimization of the weighted number of tardy jobs with random processing times and deadline. *Computers & Operations Research*, 18(5), 457–463.

d'Epenoux, F. (1963). A probabilistic production and inventory problem. *Management Science*, 10(1), 98–108.

Down, D. G., Koole, G., & Lewis, M. E. (2011). Dynamic control of a single-server system with abandonments. *Queueing Systems*, 67(1), 63–90.

Doytchinov, B., Lehoczky, J., & Shreve, S. (2001). Real-time queues in heavy traffic with earliest-deadline-first queue discipline. *Annals of Applied Probability*, 11(2), 332–378.

Emmons, H. & Pinedo, M. (1990). Scheduling stochastic jobs with due dates on parallel machines. *European Journal of Operational Research*, 47(1), 49–55.

Ephremides, A., Varaiya, P., & Walrand, J. (1980). A simple dynamic routing problem. *IEEE Transactions on Automatic Control*, 25(4), 690–693.

Gaver, D. P., Jacobs, P. A., Samorodnitsky, G., & Glazebrook, K. D. (2006). Modeling and analysis of uncertain time-critical tasking problems. *Naval Research Logistics*, 53(6), 588–599.

Gittins, J., Weber, R. R., & Glazebrook, K. D. (2011). *Multi-armed Bandit Allocation Indices*. Chichester: John Wiley & Sons, Ltd.

Gittins, J. C. & Jones, D. M. (1974). A dynamic allocation index for the sequential design of experiments. In J. Gani, K. Sarkadi, & I. Vincze (Eds.), *Progress in Statistics: European Meeting of Statisticians, Budapest, 1972* (pp. 241–266). Amsterdam: North-Holland Publishing Company.

Glazebrook, K. D. (1983). On stochastic scheduling problems with due dates. *International Journal of Systems Science*, 14(11), 1259–1271.

Glazebrook, K. D. (1996). On the undiscounted tax problem with precedence constraints. *Advances in Applied Probability*, 28(4), 1123–1144.

Glazebrook, K. D., Ansell, P. S., Dunn, R. T., & Lumley, R. R. (2004). On the optimal allocation of service to impatient tasks. *Journal of Applied Probability*, 41(1), 51–72.

Glazebrook, K. D., Kirkbride, C., & Ouenniche, J. (2009). Index policies for the admission control and routing of impatient customers to heterogeneous service stations. *Operations Research*, 57(4), 975–989.

Glazebrook, K. D. & Mitchell, H. M. (2002). An index policy for a stochastic scheduling model with improving/deteriorating jobs. *Naval Research Logistics*, 49(7), 706–721.

Glazebrook, K. D. & Niño-Mora, J. (2001). Parallel scheduling of multiclass M/M/m queues: Approximate and heavy-traffic optimization of achievable performance. *Operations Research*, 49(4), 609–623.

Glazebrook, K. D. & Punton, E. L. (2008). Dynamic policies for uncertain time-critical tasking problems. *Naval Research Logistics*, 55(2), 142–155.

Golany, B., Kaplan, E. H., Marmur, A., & Rothblum, U. G. (2009). Nature plays with dice–terrorists do not: Allocating resources to counter strategic versus probabilistic risks. *European Journal of Operational Research*, 192(1), 198–208.

Harrison, M. J. & Zeevi, A. (2004). Dynamic scheduling of a multiclass queue in the Halfin-Whitt heavy traffic regime. *Operations Research*, 52(2), 243–257.

Howard, R. (1960). *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press.

James, T., Glazebrook, K. D., & Lin, K. Y. (2016). Developing effective service policies for multiclass queues with abandonment: Asymptotic optimality and approximate policy improvement. *INFORMS Journal on Computing*, 28(2), 251–264.

Jang, W. & Klein, C. M. (2002). Minimizing the expected number of tardy jobs when processing times are normally distributed. *Operations Research Letters*, 30(2), 100–106.

Jiang, Z., Lewis, T. G., & Colin, J. Y. (1996). Scheduling hard real-time constrained periodic tasks on multiple processors. *Journal of Systems and Software*, 19(11), 102–118.

Jouini, O., Pot, A., Koole, G., & Dallery, Y. (2010). Online scheduling policies for multiclass call centers with impatient customers. *European Journal of Operational Research*, 207(1), 258–268.

Kelley, J. E. J. (1960). The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4), 703–712.

Kiekintveld, C., Jain, M., Tsai, J., Pita, J., Ordóñez, F., & Tambe, M. (2009). Computing optimal randomized resource allocations for massive security games. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2* (pp. 689–696). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Kikuta, K. & Ruckle, W. H. (2002). Continuous accumulation games on discrete locations. *Naval Research Logistics*, 49(1), 60–77.

Kim, J. & Ward, A. R. (2012). Dynamic scheduling of a GI/GI/1+ GI queue with multiple customer classes. *Queueing Systems*, 75(2-4), 339–384.

Klimov, G. P. (1974). Time-sharing service systems. I. *Theory of Probability and Its Applications*, 19(3), 532–551.

Klimov, G. P. (1978). Time-sharing service systems. II. *Theory of Probability and Its Applications*, 23(2), 314–321.

Krishnan, K. R. (1987). Joining the right queue: A Markov decision-rule. In *Proceedings of the 26th IEEE Conference on Decision and Control, 1987* (pp. 1863–1868).

Larrañaga, M., Ayesta, U., & Verloop, I. M. (2014). Index policies for a multiclass queue with convex holding cost and abandonments. In *The 2014 ACM*

International Conference on Measurement and Modeling of Computer Systems (pp. 125–137). New York, NY: ACM.

Larrañaga, M., Ayesta, U., & Verloop, I. M. (2013). Dynamic fluid-based scheduling in a multi-class abandonment queue. *Performance Evaluation*, 70(10), 841–858.

Li, D. & Glazebrook, K. D. (2010). An approximate dynamic programing approach to the development of heuristics for the scheduling of impatient jobs in a clearing system. *Naval Research Logistics*, 57(3), 225–236.

Lin, K. Y., Atkinson, M. P., Chung, T. H., & Glazebrook, K. D. (2013). A graph patrol problem with random attack times. *Operations Research*, 61(3), 694–710.

Lin, K. Y., Atkinson, M. P., & Glazebrook, K. D. (2014). Optimal patrol to uncover threats in time when detection is imperfect. *Naval Research Logistics*, 61(8), 557–576.

Lin, K. Y., Kress, M., & Szechtman, R. (2009). Scheduling policies for an antiterrorist surveillance system. *Naval Research Logistics*, 56(2), 113–126.

Lin, K. Y. & Singham, D. I. (2015). A classical search model revisited: Robust search policies against an intelligent evader. Unpublished manuscript, Naval Postgraduate School, Monterey, California.

Lippman, S. A. (1975). Applying a new device in the optimization of exponential queueing systems. *Operations Research*, 23(4), 687–710.

Liu, C. L. & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 46–61.

Luenberger, D. G. & Ye, Y. (2008). *Linear and Nonlinear Programming*. New York, NY: Springer Science & Business Media.

McMahan, H. B., Gordon, G. J., & Blum, A. (2003). Planning in the presence of cost functions controlled by an adversary. In T. Fawcett & N. Mishra (Eds.),

Twentieth International Conference on Machine Learning (pp. 536–543). Palo Alto, CA: AAAI.

Movaghar, A. (1998). On queueing with customer impatience until the beginning of service. *Queueing Systems*, 29(2-4), 337–350.

Movaghar, A. (2005). Optimal control of parallel queues with impatient customers. *Performance Evaluation*, 60(1), 327–343.

Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(1), 48–49.

Niederreiter, H. (1978). Existence of good lattice points in the sense of Hlawka. *Monatshefte für Mathematik*, 86(3), 203–219.

Ordónez, F., Tambe, M., Jara, J. F., Jain, M., Kiekintveld, C., & Tsai, J. (2013). Deployed security games for patrol planning. In J. Herrmann (Ed.), *Handbook of Operations Research for Homeland Security* (pp. 45–72). New York, NY: Springer.

Panwar, S. S., Towsley, D., & Wolf, J. K. (1988). Optimal scheduling policies for a class of queues with customer deadlines to the beginning of service. *Journal of the ACM*, 35(4), 832–844.

Papadimitriou, C. H. & Tsitsiklis, J. N. (1999). The complexity of optimal queuing network control. *Mathematics of Operations Research*, 24(2), 293–305.

Paruchuri, P., Pearce, J. P., Marecki, J., Tambe, M., Ordonez, F., & Kraus, S. (2008). Playing games for security: an efficient exact algorithm for solving bayesian stackelberg games. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2* (pp. 895–902). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Pinedo, M. (1983). Stochastic scheduling with release dates and due dates. *Operations Research*, 31(3), 559–572.

Pita, J., Jain, M., Marecki, J., Ordóñez, F., Portway, C., Tambe, M., Western, C., Paruchuri, P., & Kraus, S. (2008). Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track* (pp. 125–132). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Pita, J., Tambe, M., Kiekintveld, C., Cullen, S., & Steigerwald, E. (2011). Guards: game theoretic security allocation on a national scale. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1* (pp. 37–44). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Powell, M. J. D. (1987). Radial basis functions for multivariable interpolation: a review. In J. C. Mason & M. G. Cox (Eds.), *Algorithms for approximation* (pp. 143–167). New York, NY: Clarendon Press.

Powell, M. J. D. (1999). Recent research at cambridge on radial basis functions. In M. W. Müller, M. D. Buhmann, D. H. Mache, & M. Felten (Eds.), *New Developments in Approximation Theory* (pp. 215–232). Basel: Birkhäuser.

Powell, W. B. (2011). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Hoboken, NJ: John Wiley & Sons, Inc.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ: John Wiley & Sons, Inc.

Regis, R. G. & Shoemaker, C. A. (2005). Constrained global optimization of expensive black box functions using radial basis functions. *Journal of Global Optimization*, 31(1), 153–171.

Robbins, H. & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(2), 400–407.

Robinson, J. (1951). An iterative method of solving a game. *The Annals of Mathematics*, 54(2), 296–301.

Serfozo, R. F. (1979). An equivalence between continuous and discrete time markov decision processes. *Operations Research*, 27(3), 616–620.

Shieh, E., An, B., Yang, R., Tambe, M., Baldwin, C., DiRenzo, J., Maule, B., & Meyer, G. (2012). Protect: A deployed game theoretic system to protect the ports of the united states. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1* (pp. 13–20). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Stidham, S. (1985). Optimal control of admission to a queueing system. *IEEE Transactions on Automatic Control*, 30(8), 705–713.

Stidham, S. (2002). Analysis, design, and control of queueing systems. *Operations Research*, 50(1), 197–216.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1), 9–44.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Takács, L. (1962). *Introduction to the Theory of Queues*. New York, NY: Oxford University Press.

Tijms, H. C. (1994). *Stochastic Models: An Algorithmic Approach*. Hoboken, NJ: John Wiley & Sons, Inc.

Tsai, J., Kiekintveld, C., Ordonez, F., Tambe, M., & Rathi, S. (2009). IRIS - A tool for strategic security allocation in transportation networks. In *Proceedings*

of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track (pp. 37–44). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Tsai, J., Nguyen, T. H., Weller, N., & Tambe, M. (2014). Game-theoretic target selection in contagion-based domains. *The Computer Journal*, 57(6), 893–905.

Van Mieghem, J. A. (2003). Due-date scheduling: Asymptotic optimality of generalized longest queue and generalized largest delay rules. *Operations Research*, 51(1), 113–122.

Verloop, I. M. (2014). *Asymptotic optimal control of multi-class restless bandits*. Technical report. CNRS Technical Report hal-00743781.

von Neumann, J. (1928). Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1), 295–320.

Walrand, J. (1988). *An Introduction to Queueing Networks*. Englewood Cliffs, NJ: Prentice Hall.

Ward, A. R. & Kumar, S. (2008). Asymptotically optimal admission control of a queue with impatient customers. *Mathematics of Operations Research*, 33(1), 167–202.

Washburn, A. & Wood, K. (1995). Two-person zero-sum games for network interdiction. *Operations Research*, 43(2), 243–251.

Washburn, A. R. (2014). *Two-Person Zero-Sum Games*. New York, NY: Springer US.

Watkins, C. (1989). *Learning from delayed rewards*. PhD thesis, Cambridge University.

Wein, L. M. & Baveja, M. (2005). Using fingerprint image quality to improve the identification performance of the us visitor and immigrant status indicator

technology program. *Proceedings of the National Academy of Sciences of the United States of America*, 102(21), 7772–7775.

Whittle, P. (1988). Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability*, 25, 287–298.

Winston, W. (1977). Optimality of the shortest line discipline. *Journal of Applied Probability*, 14(1), 181–189.

Yang, R., Ordonez, F., & Tambe, M. (2012). Computing optimal strategy against quantal response in security games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2* (pp. 847–854). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Zafra, P. (2010). Fictitious Play Algorithm. In J. J. Cochran (Ed.), *Wiley Encyclopedia of Operations Research and Management Science*. Hoboken, NJ: Wiley-Blackwell.