Index heuristics for routing and service control problems within queueing systems

Richard R Lumley

Submitted for the degree of Ph.D.,

The University of Edinburgh,

2005.

Acknowledgements

I would like to thank my supervisor Professor Kevin Glazebrook for all of the help and encouragement he has given me during my studies. I would also like to thank the staff at Newcastle & Edinburgh Universities and my family for their support. Thanks too to the EPSRC for the funding that made this work possible.

Declaration

I hereby declare that I, Richard R. Lumley, have composed this thesis. The thesis contains my own work, prepared and completed with Professor K.D. Glazebrook as first supervisor.

This work has not been submitted for any other degree or professional qualification except as specified.

Richard R. Lumley

Abstract

This thesis is naturally broken down into two main problems, one concerning optimal routing control and the other optimal service control. In the routing control problem the arriving customers must be allocated to one of the 'K' possible service stations. We assume that the customers arrive in a single Poisson stream. We take the service at each of the stations to be exponentially distributed, but perhaps with different parameters. The system cost rate is additive across the queues formed at each station. We also have that at each station the holding cost function is increasing convex. Following Whittle's approach to a class of restless bandit problems, we develop a Lagrangian relaxation of the routing control problem which serves to motivate the development of index heuristics. The index by a particular station is characterised as a fair charge for rejecting the arriving customer at that station. We also consider a policy improvement index for comparison to the heuristic. We develop these indices and report an extensive numerical investigation which exhibits strong performance of the index heuristic for both discounted and average costs.

The second problem concerns the optimal service control of a multi-class M/G/1 queueing system in which customers are served non preemptively. The system cost rate is additive across classes and increasing convex in the numbers present within each class. We again follow the method prescribed by Whittle when considering a class of restless bandits. Hence we develop a Lagrangian relaxation of the service control problem which motivates the development of a class of index heuristics. For a particular customer class the index is characterised as a fair charge for service of that class. These indices are developed and we again report representative results from an extensive numerical study which again implies a strong performance of the index heuristic for both discounted and average costs.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Throughout their lives people have many decisions to make. For example, we may have to decide how to best allocate time amongst a number of competing demands. The outcome of such a decision is often uncertain and can affect the options which are available to us in the future. The more rational amongst us will make these decisions with the aim of achieving certain goals or maximizing some measure of 'utility'.

Similar resource allocation problems are found in many areas in industrial, financial, computing and telecommunication settings. Within these problems an optimal strategy for allocating a resource is often deemed to be the one that optimises some measure of performance. Consider the two following queueing examples:

(i) Which of N possible routes should a telecommunications company use to send a message when the total delivery time, via each route, and the arrival times of future messages are unknown?

(ii) In what order should a computer allocate processing, amongst a number of competing classes of job awaiting service, when exact processing requirements and the times of future arrivals are unknown?

Problem (i) may be characterised as a *routing control* problem whereas problem (ii) could be looked upon as a *service control* problem. In the next Section 1.1 we will explain both routing and service control problems further.

1.1 Service and Routing Control for Queueing Systems

A queue forms in a system when the demands of the arriving customers cannot be met instantaneously. The term *queueing system* will be referred to many times throughout this thesis. There are many types of queueing system with many subtle differences. In this thesis we look only at certain routing and service control problems. However, we are fully aware that there is a large amount of literature concerning the control of queueing systems, not only in the areas we consider but also in many other areas. Roughly speaking the queueing systems we shall discuss are characterised by an *input process*, a *service policy* and a *cost structure*.

The input process describes the manner in which the customers enter the system. For example all the customers requiring service could be present initially, or they could arrive in batches of 8 every 20 minutes, or they may enter the system according to some continuous time random process. It is the latter example that we use throughout this thesis. It can be that all arriving customers are identical or they can have distinct attributes which yield a grouping into *classes*. Classes of customers can differ in their arrival rates, service requirements and costs. Systems with different classes of arriving customer are called *multi-class queueing systems*.

The service policy relates to the way in which the customers waiting in the queue are processed. For example there could be a first come first served (FCFS) policy, in which customers are processed in order of their arrival, or there could be a priority policy in which all customers of type 1 are processed before any customers of type 2. We could use only one server processing all of the customers or we could use multiple servers. The latter case often poses greater challenges when searching for an optimal service policy. See for example Glazebrook and Wilkinson (2000) who discover that Gittins index policies, for multi-armed bandits with discounted rewards earned over an infinite horizon, are no longer optimal when the single server is replaced by a collection of single servers working in parallel.

The cost structure relates to the manner in which costs are incurred. The cost of the system is measured by some form of customer utility, often a function of the time spent awaiting service or a measure of the system running costs. System running costs are often assumed to be linearly related to the number of customers present in the queue or to the time spent by customers in the system. In fact much previous work has assumed that costs are linearly related to the number of customers present in the queue but within this work we take the relationship to be increasingly convex.

We now give a brief explanation of a general queueing system, for both routing and service control problems, before going into further detail. We first consider the routing control problem.

Routing Control

Our **routing control** problem concerns the allocation of arriving customers to alternative service stations. As an aid to understanding the setup of this system let us consider Figure 1.1. We have customers arriving into the system at **A**. These customers need to be allocated to one of the possible N service stations (**B**). The decision here is about which service station to send each customer to. Hence this

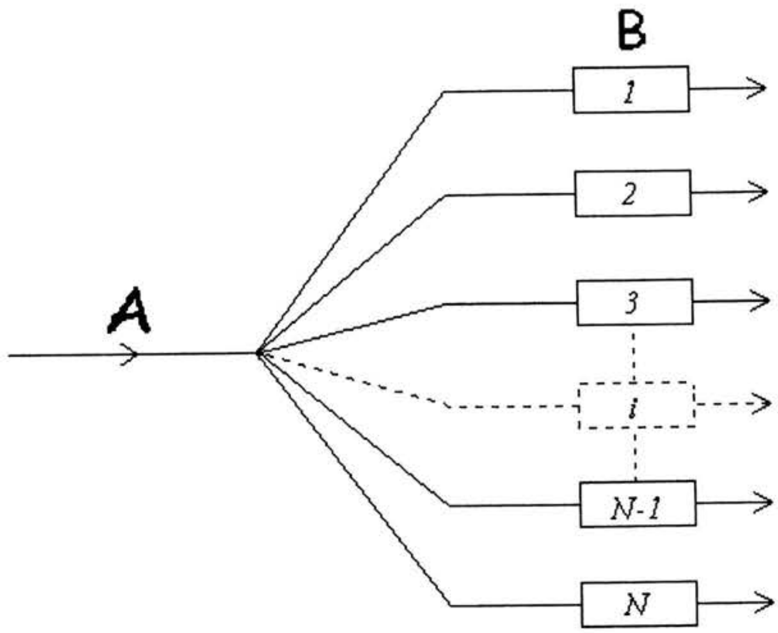

Figure 1.1: Our routing control problem queueing system.

problem is essentially about how to organize the arriving customers into queues. The routing control problem considered in this thesis assumes that all arriving customers consist of a single class and arrive as a Poisson stream. However, the nature of the service offered at distinct stations may differ. We aim to find a routing control policy which minimizes some measure of total costs incurred over an infinite horizon.

In the main, previous routing control research has focussed on special cases of the issues and models considered in Chapter 2. For example, much work has been preoccupied with the routing of a single class of arriving jobs to a collection of homogenous stations. For such problems, simple round robin policies and Bernoulli routing with equal probabilities have been shown to provide optimal load balancing regimes when little information is available to the system controller. For example consider Chang (1992). Also in a paper by Ephremides *et al.* (1980) it was shown that for the two-server models considered, round robin policies are optimal if the queue lengths are not known but the destination station of the previous arrival is known. Further Koole (1996) showed that for the case of i.i.d. exponential service times, splitting the arriving customers equally among the queues, provides an optimal return. A paper by Lui and Townsley (1994) also proves the optimality of the round robin policy when servers are identical and there is no state information. When full information on the queue lengths at each station is available the 'join the shortest queue' strategy has been shown to be optimal for a range of models. See, for example, Hordijk and Koole (1990), Johri (1989). Weber (1978) also showed that for systems with several identical servers the join the shortest queue (JSQ) discipline maximised the expected number of customers served by a given time. Winston (1997) also shows this to be the optimal strategy for the discounted version of this problem. See Gelenbe and Pekergin (1993) for an overview of some of the practical issues involved in developing load balancing regimes. The index policies developed in Chapter 2 do indeed become "join the shortest queue" in the special case of homogeneous stations. Work has also been done in this area on problems with linear holding costs, but with the added complication that classes of jobs entering the system may be more effectively served by particular servers. See for example Ansell et al (2001) where a policy is found for routing customers based on a measure of congestion at each station.

One area of application for such systems is known as *the grid*. See for example the work of Foster and Kesselman (1998). In a grid environment a provider offers a number of different services to the public, using a collection of networked machines, which may or may not have other tasks to perform. The routing problem is how to distribute requests for service, among the service stations, so as to make the best possible use of available resources and provide the best possible quality of service. Braun et al (2001) gave a detailed discussion of high performance computing environments which are well suited to meet the computational demands of large diverse groups of applications. Another similar example is discussed in the work of

Becker et al (2000) who considered a routing problem motivated by call centers of companies producing a range of products. Customers telephone such centres with requests for service or technical support. These calls are then routed to agents. Calls concerning a particular product should be preferably assigned to an agent with the requisite expertise but that may not always be possible in a timely fashion.

Service Control

Our **service control** problem concerns decisions about how to allocate service among several classes of customer awaiting service. Again to get a better understanding of this type of system let us consider Figure 1.2. Here we have

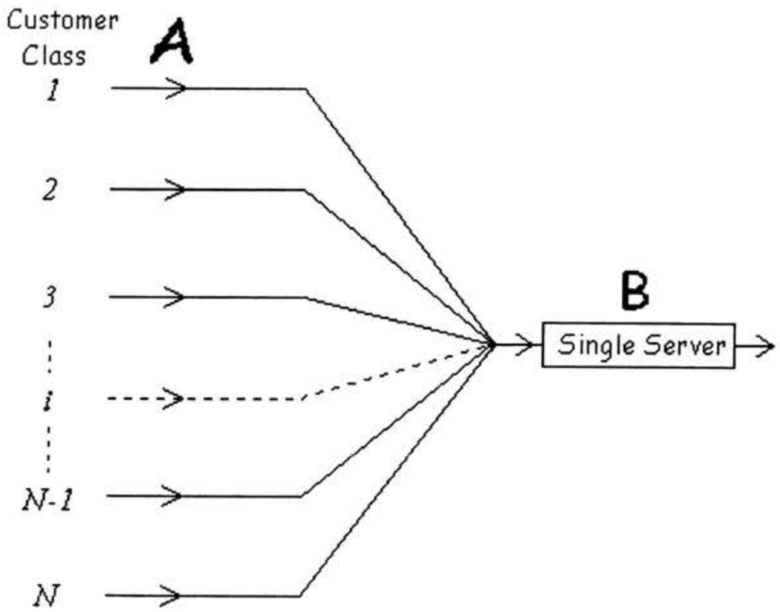

Figure 1.2: Our service control problem queueing system.

different classes of customer arriving into the system at **A**. These customers require service from a single server (**B**). The choice here concerns which of the waiting customers should be served next. The multi-class service control problem considered in this thesis assumes that we have N classes of customers each arriving as independent Poisson streams. We aim to find a service policy which minimizes the total costs incurred.

The service control section, Chapter 3, considers a cost only approach to the problem. Therefore we do not receive any reward for service but we do incur costs when customers are waiting in the system. In much of the existing literature it has been assumed that such holding costs are linear in the number of customers from each class present in the system. This assumption has at least in part been motivated by the relative tractability of the resulting models. In particular, simple priority policies in which the server(s) chooses from among the the customers waiting for service, according to a fixed ordering of the classes, have been shown to be optimal for linear costs in a variety of contexts. See, for example, Cox and Smith (1961), Klimov (1974). Also Harrison (1975) considers a non-preemptive, multi-class single server model and shows the optimality of a priority ranking where certain classes are never served. Meilijson and Weiss (1977) show the optimality of a fixed priority policy, in a set up in which the service rendered a customer is a branching process of operations, where each operation cannot be interrupted. Gittins (1979) considers bandit processes and dynamic allocation indices to show how previously intractable problems can be reduced to the problem of calculating such indices.However van Meighem (1995) has argued that assumptions of linear costs are often inappropriate. His study uses cost-delay functions to move away from this linear assumption. In a related contribution, Ansell *et al.* (1999) point to unsatisfactory features of the priority policies resulting from linear models including a propensity to produce excessive queue lengths and waiting times of large variance for low priority customer classes. As a result of such concerns in this thesis we have taken holding cost rates to be additive across classes and increasing convex in the numbers present within each class.

Both the routing and service control problem we consider in the body of this thesis are strongly related to an intractable class of problems called *restless bandits*, which is explained further in Section 1.3.3. It was Whittle (1988) who introduced this class of decision problems and used a Langrangian relaxation from which an index heuristic emerged naturally. Whittle (1996) considered the application of his ideas to undiscounted service control models of the kind mentioned above but suggested these ideas were not helpful in this context. This was because following his method directly for the undiscounted case does not lead to sensible indices. However, Whittle's approach can indeed be used, as can be seen in Section 3.4. The idea behind our successful analysis is outlined in the following paragraph. The key is to begin with the apparently more difficult discounted costs problem and recover the average costs version as a limiting form. By this indirect route we can indeed develop a *Whittle index policy* for this undiscounted costs problem.

1.2 Traditional Approaches

Stochastic dynamic optimisation problems, such as the routing and service control problems considered above, have been traditionally tackled within a *Dynamic Programming* (DP) framework. The central idea of DP is based on a principle of optimality discussed by Bellman (1957). The principle states that,

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

In stochastic dynamic optimisation this principle is often expressed mathematically by an equation of the form:

$$V_n(i) = \min_{a \in A(i)} \left\{ c_i(a) + \sum_{j \in I} p_{ij}(a) V_{n-1}(j) \right\}, \quad i \in I. \tag{1.1}$$

In the above optimality equation i denotes the current state of the system, and is a member of state space I. Further, $V_n(i)$ denotes the minimum expected cost for an n stage problem that starts in state i, a is an action chosen from the set $A(i)$ of possible actions in state i, $c_i(a)$ is the cost incurred when the state is i and action a is selected and $p_{ij}(a)$ is the probability that, given the current state is i and action a is taken, the next state will be j. The application of Bellman's principle to dynamic optimization problems yield recursive equations (Dynamic Programming equations - DPEs) for the optimal cost (or reward) function. Very occasionally it is possible to find an analytical solution to these DPEs, and thereby derive the optimal policy. When an exact analytical solution cannot be found, properties of the optimal objective function can be deduced which translate into results regarding the structure of an optimal policy. When such approaches fail the problem can be solved numerically. However for larger problems this becomes computationally infeasible. In multi-class systems computational infeasibility may arise because of the high dimensionality of the state space. Index results, like those of Gittins, may be understood as effecting a reduction in the dimensionality of the problem.

Interchange arguments are standard in stochastic scheduling, optimality of a policy is proven by demonstrating that any other policy can be improved by interchanging action times. See for example Cox and Smith (1961) who use this method to show the optimality of the non-preemptive $c\mu$-rule. They consider a service problem where jobs of different classes arrive as independent Poisson processes and must be served non-preemptively by the single server. The non-preemptive $c\mu$-rule is one that at any service completion time, starts serving the customer class with the largest value of $c_i\mu_i$ among the present customers, where c_i is the cost rate and μ_i is the service rate for class i. Forward induction has also been used to prove results in this area. An explanation for this method is: let the event times be labelled as follows $t_1 < t_2 < t_3 < \ldots$ then certain properties (which imply that the policy of interest has an associated cost which is not larger than any other policy) are proven to be true for the initial case (usually at $t = 0$) then the assumption that they hold at time t is used to prove these properties hold at time t_n, where $t_n - 1 \leq t < t_n$. Examples include Ephremides, et al (1980) where the optimality of round robin policies are shown. This paper considers a routing problem, where one must decide which of the identical $M/M/1$ queues the arriving customer should join and the queue lengths and customer arrival times are not observable. Round robin policies allocate arriving customers to the queues in order then repeat allocation in the same order. A major research success of particular relevance to us is the classical index result of Gittins and Jones (1974). They solved the multi-armed bandit problem which had previously proved frustratingly difficult. The problem they considered was one in which a gambler makes a sequence of plays on N gambling machines ('bandits'), and wishes to choose at each stage of the game the machine to play so as to maximize the total expected payoff. The success probability of the ith machine is a parameter whose value is unknown. However the gambler builds up an estimate of this parameter which becomes more precise as he gains more experience of the machine. The decision conflict is between playing a machine which is known to have a good pay-off parameter value and experimenting with a machine about which little is known, but which could prove even better. To resolve this conflict one formulates an optimisation problem. The resulting index policy found by Gittins and Jones uses an index $v_i(x_i)$ attached to the ith machine which is a function of the machine label i and its current state x_i. The optimal policy is then simply to choose a machine of current greatest index at each stage. Furthermore, the Gittins index v_i is determined by the statistical properties of machine i alone. See Gittins (1979) for a wide ranging discussion of this result and Whittle (1980) for a proof of Gittins' Index Theorem using dynamic programming arguments. Whittle (1981) has also produced a DP proof of the optimality of Gittins index policies for "open" systems in which new machines arrive over time. Simpler proofs of the optimality of Gittins index policies have been given by Tsitsiklis (1986), Weber (1992) and Garbe and Glazebrook (1996). Whittle's 1981 paper lead onto the work of Weiss (1988) on branching bandits. This is of interest since branching bandit models are reasonably general models for service control control problems with a single server.

By the mid-1980's it was generally felt that successes from DP in the field of optimal dynamic control of complex stochastic systems were sparse and gained at great expense in terms of time and effort. This was because such techniques seemed too general to exploit any special structure and the techniques used to complement them (for example, interchange arguments) seemed rather limited in scope. However, because of the automisation of manufacturing processes and the increased importance of computer and communication systems the need for research into stochastic scheduling in complex systems was growing.

1.3 Recent Developments

1.3.1 Achievable Region Approach

This approach seeks solutions to stochastic optimisation problems by firstly characterizing the space of all possible performances (the achievable region) of the stochastic system and then by optimizing the overall system-wide performance objective over this space. This method does have its merits, such as the vast reduction in state space. The performance space mentioned is often a polyhedron of special structure which means that the optimization can be solved via a mathematical program (usually a linear program (LP)) for which efficient algorithms exist. Rather than use standard LP formulations in the variable space of state-action frequencies (which is typically huge or infinite) work has been done to develop analyses in some projected space (of reduced dimensionality) of natural performance variables. The earliest work on this approach was due to Gelenbe and

Mitrani (1980) followed by Federgruen and Groenvelt (1988). Contributions by Shanthikumar and Tao (1992) and Bertsimas and Niño-Mora (1996) took the approach decisively further forward, the latter giving an account of Gittins indices from this perspective. Dacre et al (1999) also considered this alternative approach to the optimal control of stochastic systems. In their paper they consider both service and routing control problems.

1.3.2 DP Policy Improvement

Fairly recently Ansell et al (2001) have studied a routing control problem for a class of multi-class service systems. The work in the aforementioned paper develops an idea proposed in the context of a simple single class system by Krishnan (1987) and discussed by Tijms (1994) and applies it to a complex multi-class system. The method applies a single policy improvement approach to an optimal static (state independent) policy for the problem. The system considered in Chapter 2 is in some respects simpler. We consider only a single class of customer and do not allow feedback into the system (customers returning to the system after they have been served). However, we do suppose that holding costs for the system are increasing convex. Ansell et al (2001) first of all determine an optimal static policy and then improve on it by considering the difference in total expected costs over an infinite horizon for each station individually between starting in state $\underline{n} + \underline{1}^j$ and starting in state $\underline{n}$, when the optimal static policy is followed. In this case the *state* refers to the number of customers of each class present in each queue, and hence is a vector whose dimension is the same as the product of the number of job classes and number of service stations in the system. Note that $\underline{1}^j$ is a vector with a one in position j and zeros elsewhere which represents a single customer of class j. This difference forms the basis of an index for each station, dependent both upon its current state, $\underline{n}_k$ and the class of job to be allocated, j. The system controller will send the arriving type j customer to the station with the smallest index. It was shown numerically that the result of this analysis is the development of simply structured dynamic routing policies which are close to optimal. In Section 2.4.2 we apply similar ideas to our queueing system of interest to develop a policy improvement index policy.

1.3.3 Relaxations

One paper which is of further relevance to us is that of Whittle (1988). In this paper he considers the multi-armed bandit problem, as previously mentioned, where the unused bandit states also change over time. Such problems, as we have already mentioned, are referred to as *restless bandits.* In formulating this problem Whittle was concerned with the maximisation of rewards where the level of reward for any action depended upon the current state and whether the bandit was active or not. The problem is also generalized to the case where m bandits are active at all times. Whittle's solution method involves relaxing this constraint so that *on average* m bandits are active. Whittle incorporates the relaxed constraint into the maximization problem by using a Lagrangian multiplier. This Lagrangian multiplier can be viewed as a 'subsidy for passivity' which needs to be set at just the level to ensure that m bandits are active on average. This subsidy will be independent of the project as the constraint is one on total activity, not individual project activity. Whittle then goes on to define an index $v_i(x_i)$ for bandit i when in state x_i as the value of the subsidy which makes the choice of playing the bandit or not equally attractive. However for the index to be meaningful the bandit must satisfy a condition of *indexability.* A bandit is indexable when, if it is optimal not to operate it under subsidy v then it will also not be operated under a subsidy $v' > v$. He then shows that if all bandits are indexable, then the projects which are in operation under a v-subsidy policy are those for which $v_i(x_i) > v$. Since such a policy must solve the relaxed problem above, Whittle proposes that the policy which always operates the m bandits of largest index will give a reasonable solution to the original restless bandit problem.

Building from Whittle's work, Ansell et al (2003a) consider the service control of a multi-class, single server queueing system with convex costs. The authors of this paper follow Whittle's prescription for the development of an index appropriate for their multi-class queueing system. Namely, they relax the original problem and incorporate the relaxed constraint via a Lagrangian multiplier. They establish indexability and then use the multiplier to form the basis for the definition of a selection index. It is this approach developed by Whittle which is used throughout this thesis to lead us to policies of interest.

Niño-Mora (2001a) maps out an alternative route to the demonstration of indexability for restless bandits and to index calculation which utilises the stronger notion of PCL (partial conservation laws) - indexability. This in turn is a development of the achievable region analysis of multi-armed bandits given by Bertsimas and Niño-Mora (1996). In brief let us suppose that we wish to allocate service in a system with a countably infinite collection of job classes indexed by the natural numbers $\mathbb{N}$. Denote by $\mathcal{U}$ the collection of admissible scheduling policies. The stochastic optimisation problem of interest is assumed to consist of the minimisation of some linear objective. Niño-Mora (2001b) uses the above formulation to develop sufficient conditions for the indexability of countable state restless bandits in terms of model parameters. We write

$$\min_{u} \sum_{i \in \mathbb{N}} c_i x_i^u \tag{1.2}$$

where $c_i > 0$ is a cost rate for job class i and x_i^u is a performance measure for class i under some scheduling policy u. When the system satisfies a collection of so-called partial work conservation laws (PCL) then the stochastic optimisation problem in (1.2) is solved by an index policy for *some* choices of the cost rate vector $\mathbf{c}$.

Whether a particular choice is in this admissible class or not may be determined by running an adaptive greedy algorithm. A system which satisfies PCL and whose cost rate vector $\mathbf{c}$ is in the admissible class is called PCL-indexable.

1.4 Thesis Structure

As the thesis title suggests this work concerns two different problems of stochastic dynamic control. The Chapter 2 discusses a routing control problem in the context of a multi-service station queueing system. Chapter 3 addresses the problem of service control of a multi-class queueing system.

In this introductory chapter we have already alluded to the problems and general system setups which we shall address throughout our work. We have also mentioned work by various authors on routing control problems of related systems. We begin Chapter 2 by describing in detail the routing control problem of interest and the criteria by which we intend to assess policies. In Section 2.2 we explain the specifications of the system used and introduce notation. We then consider the performance criteria required to assess the policies considered. Once we have formulated our optimisation problem explicitly we then consider the resource constraint which defines this problem. Following Whittle's approach we then relax the constraint and incorporate it into the optimisation problem by using a Lagrangian multiplier W. We observe that W plays the economic role of a constant charge for not accepting a customer into the system. The next step is very important in the solution of the problem, since it is here we notice that our relaxed optimisation problem can be naturally decoupled into single-station subproblems. Hence by this means we can solve the relaxed problem and verify indexability by determining the optimal policy for appropriately defined single station problems.

It is in Section 2.3 that we study the discounted version of this problem. The Lagrangian relaxation approach yields a reduction of the discounted problem to a set of single station problems. In Section 2.3.1 we introduce this discounted single station problem in more detail. The choice that we must make at each decision epoch in this problem is simply whether to admit an arriving customer into the queue at this service station and incur additional holding costs, or not to admit the customer and pay a charge W. Standard DP techniques are used to develop optimality equations. We then define the index for state m, $W(m)$, as the rejection charge required so that both options of accepting the customer or not are optimal for state m. Next we calculate the total expected costs of stationary policies which respectively accept and reject an arriving customer in state m, equate them and re-arrange to give a formula for the index. We then proceed to prove from our assumptions that this proposed index is increasing in the queue length and hence that the station is indexable with the proposed index equal to the true one.

In Section 2.4 we proceed to look at an undiscounted version of the problem. In Section 2.4.1 we use the formula for the discounted Whittle index to yield the undiscounted index by taking a limit. Section 2.4.2 then considers an alternative policy improvement index for comparison with the Whittle index. This policy improvement index is derived by implementing a single policy improvement step on an optimal static (state independent) policy for the problem. We firstly discuss and then calculate this policy improvement index.

We end this chapter by looking at a numerical investigation of our proposed heuristics for the routing control problem in Section 2.5. Section 2.5.1 considers the discounted routing control problem for a two station example, under a range of different convex cost structures and parameters, for stochastic evolution. We compare the discounted costs for the Whittle index policy we have derived with the optimal policy derived from DP and also with an alternative index policy. This alternative index policy has been calculated by making a simplifying assumption that it would be possible to have a negative number of customers present in the queue (incurring a zero cost). Then in Section 2.5.2 we proceed to study an average (undiscounted) cost routing control problem for a system with two service stations. For this example we look at a range of different convex cost structures and stochastic evolution parameters. Here we compare the performance of our Whittle index policy with that of the policy improvement index policy and the optimal policy derived from DP. Finally in Section 2.5.3 we consider the average cost routing control problem for a system with five service stations. The size of the state space of such a problem means that it is not computationally feasible to obtain a direct numerical comparison between costs incurred by our index policy and an optimal policy. The application of DP is computationally infeasible. So in this section we use simulation to compare our index policy to some other standard, widely accepted heuristics. Yet again we consider a range of different convex cost structures and stochastic evolution parameters. In all cases the results of the numerical investigations testify to strong performance of the index policies derived by our analyses.

Chapter 3 considers the service control problem which we have previously mentioned in this introduction. Section 3.1 recaps the system in question, remarks on the performance criteria and on the work of others in this area. We follow a similar structure to that in Chapter 2. In this section we firstly introduce notation and define the system parameters we shall employ for both the discounted and undiscounted problems. We then formulate the optimality equation used to assess our policies and make a note of the constraints to which the problem adheres. We use the approach espoused by Whittle (1988), of relaxing the problem and using Lagrangian multipliers to incorporate the relaxed constraint into the objective. In doing this we introduce a new quantity, W, which plays the economic role of a constant charge for service. We next discover that this relaxed problem can again be naturally decoupled into single-class subproblems. We then proceed to consider the optimal policy for these single-class problems.

In Section 3.3 we study a discounted service control problem exclusively. Whittle (1996) argued that you could not use his approach to solve average cost versions of the service control problem. However we show in this section that you can, but you have to work from discounted problems and then take limits. In Section 3.3.1 we consider the single class system with a charge for service under this discounted criterion, introducing the problem in more detail. The choice that we must make at each decision epoch in this single-class problem is whether to serve or not. If we serve then we incur the charge for service but we do stand to reduce holding costs. We then use standard DP techniques to develop optimality equations. Next we define the index for state m, $W(m)$ to be equal to the service charge required so that both options of serving a customer or not are optimal. Then we calculate the total expected costs for two stationary policies which differ only in the action they take when the queue length is m. We equate these and re-arrange to give a formula for a proposed index. Following this we go on to prove that the proposed index is increasing, that the station is indexable and that the proposed index is indeed the true one. Following a similar development to Chapter 2 we go on to consider the undiscounted problem in Section 3.4. In this section we show how the formula we found for the discounted index can be used to find the undiscounted index by taking a suitable limit.

This chapter is concluded by a report in Section 3.5 of a numerical investigation into the policies developed. Section 3.5.1 reports on a discounted problem for a system with two customer classes. The Whittle index policy is compared with the optimal policy for a range of different convex cost structures and stochastic evolution parameters. We then proceed to look at undiscounted problems in Section 3.5.2. For the undiscounted problems we again consider a system with two customer classes for a range of different convex cost structures and stochastic evolution parameters.

In Section 3.5.3 a service control problem for a range of systems with five customer classes is considered. Again due to the size of this problem it is not computationally feasible to obtain a direct numerical comparison between costs incurred by our index policy and an optimal policy. So in this section we use techniques of simulation to compare the cost performance of the index policy to some other standard, widely accepted heuristics. Yet again we consider a range of different convex cost structures and stochastic evolution parameters. In all cases the results of the numerical investigation testify to strong performance of the index policies derived by our analysis.

Note that some of the work presented in Chapter 3 of this thesis was published in the Queueing Systems journal, see Ansell et al (2003b).

Chapter 2

Routing Control Problems

2.1 Introduction

We consider queueing systems where customers entering the system must be allocated to one of K possible stations for service. In a bid to help us make such decisions we ask the question "by routing the arriving customer to which service station do we gain the most?". In other words sending this customer to which service station will reduce our costs or increase our rewards by the largest amount. The aim of this chapter is to construct a dynamic policy which will select the service station for each arriving customer, to achieve results near some defined optimal performance.

In Section 2.2 of Chapter 2 onwards we develop and apply the method employed by Whittle (1988) which is based on Lagrangian relaxations of the original problem to construct index heuristics for our routing problems. We make the assumption that arrivals occur due to a Poisson process and that service times at each service station are independent and exponentially distributed. We seek to minimise a holding cost criterion which is additive across the queues formed at each station. In our model we take the holding cost function for each class to be increasing convex, in much previous work it has been assumed to be linear which we mentioned on page 7.

The routing control problems considered here concern multiple service stations and a single customer class. It could possibly be called a "multi-class" system since once a customer is sent to a server there stochastic evolution will be particular to the server. However I believe such terminology could be confusing to the reader, so although we may refer to the customer class it should be noted that this is just the group customers waiting at a particular server. Recall that what we actually have in this chapter, is single class with multiple service stations. We may however use the terms queue, station, server and service station to describe the possible locations to which we can send an arriving customer.

We initially consider a problem where the costs incurred in the future have less weight than costs incurred now. This is the *discounted* cost service control problem. We do this by allowing future costs to be discounted at some rate, α. A cost of A incurred at time t is accounted for at time 0 as a cost of $Ae^{-\alpha t}$. We progress to the undiscounted problem, deriving our routing policies as limits, by allowing the discount rate α to tend to zero. In the undiscounted version of the model we seek to optimize the average cost of the system per unit time.

Section 2.2 considers the general set up of the problem of interest and considers both discounted and undiscounted formulations. The work encompasses a range of modelling possibilities. This section then moves on to define and study a relaxation of the problem. It uses a Lagrangian approach to determine the structure of the optimal solution to the relaxed problem. We argue that the optimal solution to the relaxed problem gives insights into the form of a "good" policy for our original problem. Section 2.3 considers the discounted version of our problem in more detail, looking at the required solution for the derived single service station problems, where a charge for admission is incurred. In Section 2.4 we derive an appropriate index for the undiscounted problem. This index is found by allowing the discount rate α to tend to zero in the equivalent discounted index. Within this section, is SubSection 2.4.2 which contains the calculation of an alternative index for the average cost admission control problem obtained from a dynamic programming policy improvement approach. We then conclude this chapter by reporting some results of a numerical investigation into the performance of the Whittle index policy. These can be found in Section 2.5. Within this investigation we consider the two service station discounted case but the main focus is on the average costs scenario. In the average costs case we consider two service station examples deriving the optimal policy using methods of dynamic programming. We also use simulation techniques to study systems with a larger number of service stations. Simulation is required since direct numerical comparison is not a reasonable computational goal for problems of this size.

2.2 The multi-class admission control system with convex costs

Recall that we are considering queueing systems where customers enter the system and then must be allocated to one of the possible K service stations to await service. Arrivals into the system follow a single Poisson stream with rate λ. Service times are independent and follow an exponential distribution, with μ_k the rate for server k. We will suppose that

$$\rho = \frac{\lambda}{\sum_{k=1}^{K} \mu_k} < 1 \tag{2.1}$$

for stability. The goal is to allocate the arriving customers to the service stations to minimise some measure of expected holding cost over an infinite horizon. As previously mentioned we shall consider both discounted and average cost (undiscounted) criteria. In order to set this problem up formally we need to introduce and explain some of the notation we shall use.

We may call the customers waiting at server k, class k customers and when we refer to the state of a particular class we actually mean the number of customers waiting at that server, including any customer currently in service. We write the state of class k, at time t, as $N_k(t)$ and the state of the system at time t is given by $\mathbf{N}(t) = \{N_1(t), N_2(t), \ldots, N_K(t)\}$, the vector of queue lengths, $t \in \mathbb{R}^+$. The decision epochs are all the customer arrival times. Let action a_k denote the allocation of an arriving customer to server k, $1 \leq k \leq K$. At each decision epoch t, the controller must choose an action a_k, $1 \leq k \leq K$. We seek the choice of which action to take at each decision epoch, in order to minimise some measure of expected costs.

Now to help us get more of a feel for the system consider the following. Suppose the system is in state $\mathbf{m}$ at time t, where $m_l > 0$, $1 \leq l \leq K$. The next change of state will occur at time $t + \tilde{Q}$ where $\tilde{Q} \sim \exp(\lambda + \sum_{j=1}^{K} \mu_j)$. If at time $t + \tilde{Q}$ an arrival into the system occurs and we assume action a_k is taken (i.e. the arrival is routed to station k). The system state at time $t + \tilde{Q}$ will be given by

$$\mathbf{N}(t+\tilde{Q})^+ = \begin{cases} \mathbf{m} - \mathbf{1}^l, & \text{with probability } \mu_l(\lambda + \sum_{j=1}^{K} \mu_j)^{-1}, \ 1 \leq l \leq K, \\ \mathbf{m} + \mathbf{1}^k, & \text{with probability } \lambda(\lambda + \sum_{j=1}^{K} \mu_j)^{-1}. \end{cases}$$

Note that in the above $\mathbf{1}^k$ denotes a K-vector whose k^{th} component is 1, with zeros elsewhere.

In the *discounted costs* version of the queueing control problems of interest, discounted costs are incurred, with rate

$$\sum_{j=1}^{K} C_j(N_j(t)) \tag{2.2}$$

at time t. The cost functions $C_k : \mathbb{N} \to \mathbb{R}^+$ are assumed increasing, convex and bounded above by some polynomial of finite order and with $C_k(0) = 0$, $1 \leq k \leq K$. A policy u is a rule for choosing actions in light of the history of the process to date and $\mathcal{U}$ is the collection of all such policies. Our goal is to seek a policy which minimises total costs incurred over an infinite horizon. We write

$$\mathbf{V}(\mathbf{m}, \alpha) = \inf_{u \in \mathcal{U}} E_u \Big[\int_0^{\infty} \sum_{k=1}^{K} C_k(N_k(t)) e^{-\alpha t} | \mathbf{N}(0) = \mathbf{m} \Big] \tag{2.3}$$

for the associated value function. The function $V(., \alpha)$ satisfies a collection of optimality equations. For example, if $m_l > 0$, $1 \leq l \leq K$, then it holds that

$$\begin{aligned}(\alpha + \lambda + \sum_{j=1}^{K} \mu_j) V(\mathbf{m}, \alpha) &= \sum_{j=1}^{K} C_j(m_j) + \sum_{j=1}^{K} \mu_j V(\mathbf{m} - \mathbf{1}^j, \alpha) \\ &\quad + \lambda \min_{1 \leq k \leq K} \{ V(\mathbf{m} + \mathbf{1}^k, \alpha) \}. \end{aligned} \tag{2.4}$$

If the minimum in (2.4) is achieved at k^* then action a_{k^*} is optimal in state $\mathbf{m}$.

The general theory of stochastic dynamic programming (DP) indicates the existence of an optimal policy which is stationary (i.e. makes decisions in light of the current state only) and whose value function satisfies the DP optimality equations, see Puterman (1994). However for our multi-class admission control problem a pure DP approach is unlikely to be insightful. Also this approach is computationally intractable for problems of a reasonable size. Hence we look for heuristic policies which are simple in form and close to optimal.

The routing policy we develop will be of *index form.* This means that there exist K *index functions* $W_{k,\alpha} : \mathbb{N} \to \mathbb{R}^+$, $1 \leq k \leq K$, such that at all decision epochs the index policy u_W, chooses to route a customer to the minimal index class, i.e.

$$u_W\{\mathbf{N}(t)\} = a_k \implies W_{k,\alpha}\{\mathbf{N}(t)\} = \min_{1 \leq j \leq K} W_{j,\alpha}\{\mathbf{N}(t)\}. \tag{2.5}$$

The average cost version of the multi-class admission control model of interest may be expressed via the equation

$$\mathbf{V}^{OPT} = \inf_{u \in \mathcal{U}} \tilde{E}_u \Big\{ \sum_{k=1}^{K} C_k(N_k) \Big\} \tag{2.6}$$

where in (2.6) $\tilde{E}_u$ is the expectation taken with respect to the steady-state distribution of the system under policy u. From standard results in DP we have that

$$\lim_{\alpha \to 0} \mathbf{V}(\mathbf{m}, \alpha) = \mathbf{V}^{OPT} \tag{2.7}$$

In light of (2.7) we can develop index heuristics for the average cost problems as limits ($\alpha \to 0$) of the index policies for discounted costs. However for this admission control problem we can also develop index policies directly. This is in contrast to the service control problem discussed in the next chapter.

To facilitate our discussion, we write $a_k(t)$ for the action (either $a =$ admit (active) or $b =$ do not admit (passive)) applied to queue k at time t. We develop the following *performance measure* for policy u, where $n \in \mathbb{N}$, $1 \leq k \leq K$:

- ◇ $y_{k,n}^u(\mathbf{m})$ - which is the expected discounted time spent by queue k in state n, where the initial state is $\mathbf{m}$.

So we can see that we have

$$y_{k,n}^u(\mathbf{m}) \;\; = \;\; E_u\Big[\int_0^\infty I\{N_k(t) = n\} e^{-\alpha t} | \mathbf{N}(0) = \mathbf{m}\Big] \tag{2.8}$$

where $I\{.\}$ is the indicator function. We now re-express our discounted costs problem in (2.3) using these performance variables, to give

$$\mathbf{V}(\mathbf{m}, \alpha) = \inf_{u \in \mathcal{U}} \sum_{k=1}^{K} \sum_{n \in \mathbb{N}} C_k(n) y_{k,n}^u(\mathbf{m}) \tag{2.9}$$

As previously mentioned, Whittle's (1988) approach to the development of index heuristics is via Langrangian relaxations. To use Whittle's method we must also develop the following *performance measure* for policy u, where $n \in \mathbb{N}$, $1 \leq k \leq K$:

- ◇ $x_{k,n}^u(\mathbf{m})$ - which is the expected discounted time queue k spends in state n and *does not* accept an arriving customer to this queue, where the initial state is $\mathbf{m}$.

To write this mathematically we use $\{t_i, i \in \mathbb{N}\}$ for the sequence of arrival times into the system (event times of a Poisson process of rate λ) and use the indicator functions

$$I_{k,i,n} = \begin{cases} 1 & \text{if, at the time of the } i^{\text{th}} \text{ arrival station } k \text{ is in state } n \text{ and does } \textbf{not} \text{ accept} \\ & \text{the new arrival;} \\ 0 & \text{otherwise.} \end{cases}$$

Using this notation we then have, for any $u \in \mathcal{U}$, $n \in \mathbb{N}$, $1 \leq k \leq K$:

$$x^u_{k,n}(\mathbf{m}) = E_u\Big[\sum_{i=1}^{\infty} e^{-\alpha t_i} I_{k,t_i,n} | \mathbf{N}(0) = \mathbf{m}\Big] \tag{2.10}$$

We now wish to develop a relaxation of (2.9), but to do this we must first consider the quantity $\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x^u_{k,n}(\mathbf{m})$. The first thing to note about this quantity is that it is policy invariant within $\mathcal{U}$, since we know that we must send each arriving customer to exactly one queue, no matter which routing policy we follow. This means that we will not accept each arriving customer into $K-1$ of the queues. Hence

$$\begin{aligned} \sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x^u_{k,n}(\mathbf{m}) &= E_u\Big[\sum_{i=1}^{\infty} (K-1) e^{-\alpha t_i}\Big] \\ &= E_u\Big[(K-1)(e^{-\alpha t_1} + e^{-\alpha t_2} + e^{-\alpha t_3} + \ldots)\Big] \end{aligned} \tag{2.11}$$

where recall that t_i is the arrival time of the i^{th} customer. Since the arrivals follow a Poisson process with rate λ we can see that,

$$t_{i+1} - t_i \equiv R \sim \exp(\lambda), \ \forall \ i \geq 0,$$

and these interarrival times are independent. Using this information within (2.11) we can see that

$$\sum_{k=1}^{K}\sum_{n\in\mathbb{N}} x_{k,n}^{u}(\mathbf{m}) = E_u\big[(K-1)(e^{-\alpha R}+e^{-\alpha 2R}+e^{-\alpha 3R}+\ldots)\big] \tag{2.12}$$

$$= E_u\Big[\frac{(K-1)e^{-\alpha R}}{1-e^{-\alpha R}}\Big]$$

$$= \frac{(K-1)E(e^{-\alpha R})}{1-E(e^{-\alpha R})}$$

$$= \frac{(K-1)\lambda}{\alpha}. \tag{2.13}$$

Note that to get to (2.13) in the above we used the formula for the sum of a geometric progression to infinity and the fact that,

$$E(e^{-\alpha R}) = \int_0^\infty e^{-\alpha t}\lambda e^{-\lambda t}dt$$

$$= \frac{\lambda}{\alpha+\lambda} \tag{2.14}$$

We now relax the stochastic optimization problem in (2.9) by expanding the policy class to $\bar{\mathcal{U}}$, namely the set of policies in which the arriving customer (or at least identical copies of that customer) can be sent to any number of service stations, and then by imposing the relation in (2.13) as a constraint. This constraint will mean that *on average* we will still admit the arriving customers to just one station. We call this relaxed stochastic optimization problem *Whittle's relaxation* and write it as follows

$$\underline{\mathbf{V}}(\mathbf{m},\alpha) = \inf_{u\in\bar{\mathcal{U}}}\sum_{k=1}^{K}\sum_{n\in\mathbb{N}} C_k(n) y_{k,n}^{u}(\mathbf{m})$$

subject to

$$\sum_{k=1}^{K}\sum_{n\in\mathbb{N}} x_{k,n}^{u}(\mathbf{m}) = E_u\Big[\sum_{i=1}^{\infty} J(t_i)e^{-\alpha t_i}|\mathbf{N}(0)=\mathbf{m}\Big]$$

$$= \frac{\lambda(K-1)}{\alpha}. \tag{2.15}$$

Note that $J(t_i)$ denotes the number of queues the i^{th} arriving customer is *not* accepted into and constraint (2.15) delimits the set of allowable policies within $\bar{\mathcal{U}}$. For any policy within $\mathcal{U}$ we will have $J(t_i) = K-1$ for all i. We now use a

Lagrangian approach to help us find the structure of the optimal solution to *Whittle's relaxation.* We accommodate constraint (2.15) by incorporating a Langrange multiplier W, to obtain the minimisation problem

$$\mathbf{V}(\mathbf{m}, \alpha, W) = \inf_{u \in \bar{\mathcal{U}}} E_u \Big\{ \sum_{k=1}^{K} \sum_{n \in \mathbb{N}} C_k(n) y_{k,n}^n(\mathbf{m}) + W \Big[\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^n(\mathbf{m}) - \frac{\lambda(K-1)}{\alpha} \Big] \Big\} \tag{2.16}$$

We can see from (2.16) that W plays the economic role of a constant charge for rejecting an incoming customer. The optimization problem we have here involves the control u that tells us to which stations each customer should be routed. Problem (2.16) is naturally decoupled into K single-class subproblems

$$\mathbf{V}(\mathbf{m}, \alpha, W) = \sum_{k=1}^{K} V_k(m_k, \alpha, W) - \frac{W\lambda(K-1)}{\alpha}. \tag{2.17}$$

In (2.17), $V_k(m_k, \alpha, W)$ is the minimised total of holding costs and rejection charge costs incurred by service station k, the minimisation being taken over all policies for choosing between action a (admit) and b (reject) for that station only. So for the single class problem we are merely concerned with the total cost incurred at service station k only. This will consist of both holding costs and rejection charges. The policy that we implement at service station k tells us if we should accept the arriving customer (and pay the increase in holding costs) or reject the customer (and pay the rejection charge) at this service station only. $V_k(m_k, \alpha, W)$ is the minimised value of this total cost over all of these possible policies.

It will be shown later in this chapter (see page 71) that there exists a multiplier $W(\mathbf{m}, \alpha)$ such that

$$\mathbf{V}\{\mathbf{m}, \alpha, W(\mathbf{m}, \alpha)\} = \underline{\mathbf{V}}(\mathbf{m}, \alpha).$$

This will lead us to infer that there exists an optimal policy for the Langrangian relaxation in (2.16) with $W = W(\mathbf{m}, \alpha)$ which satisfies the constraint in (2.15) and hence solves Whittle's relaxation.

Therefore to analyze Whittle's relaxation we will progress as follows:

- Find the optimal policies for the K single station subproblems in (2.17), which will be dependent on the value of W.

- Combine these single-station optimal policies into the required optimal policy for the corresponding multi-station problem in (2.16).

- Find the value of W which ensures the constraint in (2.15) is met and hence obtain the optimal policy for Whittle's relaxation.

So as we can see for this agenda, the first thing we must do is find the optimal policies for the single station problems, which we shall denote (k, α, W), $1 \leq k \leq K$, $W \in \mathbb{R}$. By standard DP theory we can assume that optimal policies for (k, α, W) are *stationary*. The solutions to these single class problems become simple under a condition of *indexability*.

To describe this condition, we use $\Pi_{k,\alpha}(W)$ to denote the set of queue lengths m for which the active action a is optimal in the single class problem (k, α, W). We would expect this set to grow with the rejection charge W.

Definition 1

Service station k is *α-indexable* if $\Pi_{k,\alpha}(W) : \mathbb{R} \to 2^{\mathbb{N}}$ is increasing, namely

$$W_1 > W_2 \implies \Pi_{k,\alpha}(W_1) \supseteq \Pi_{k,\alpha}(W_2)$$

Should we have α-indexability for station k, the idea of an α-index for state (i.e. queue length) m as the minimum rejection charge which makes the active action optimal there is a natural one.

Definition 2

When service station k is α-indexable, the *Whittle α-index* for class k in state m is given by

$$W_{k,\alpha}(m) = \inf\{W : m \in \Pi_{k,\alpha}(W)\}, m \in \mathbb{N}.$$

It will now follow that if each customer class k is α-indexable, Whittle's relaxation is solved by a policy in which a decision is taken to route an incoming customer to station k at each decision epoch t whenever $W_{k,\alpha}\{N_k(t)\} < W(\mathbf{m}, \alpha)$ and not to route to station k whenever $W_{k,\alpha}\{N_k(t)\} > W(\mathbf{m}, \alpha)$, for all choices of k, t. Should $W_{k,\alpha}\{N_k(t)\} = W(\mathbf{m}, \alpha)$ then some randomisation between the two actions will be appropriate. Note that the constraint (2.15) will ensure that on average we only route each incoming customer to a single station.

We now follow Whittle (1988) in arguing that the index-like nature of solutions to the relaxation in (2.15) makes it reasonable to propose an *index heuristic* for our original discounted costs problem in (2.3) and (2.9) when all customer classes are α-indexable. This heuristic will be structured as in (2.5) with index functions recovered from Definition 2. Note that under this definition it is natural to interpret $W_{k,\alpha}(m)$ as a *fair charge* for rejecting the arriving customer from queue k when it is in state m. The derived heuristic then always sends each incoming customer to the station for which the fair charge for rejection is smallest. Following the discussion about the average costs version, earlier in this section, we develop can an index heuristic for average cost problems as the limit policy ($\alpha \to 0$) of the index heuristics for discounted costs. Alternatively we shall see that we can develop index heuristics for average cost problems directly.

Definition 3

If customer class k is α-indexable for all $\alpha > 0$ then the *average cost Whittle index* for state m is given by

$$W_k(m) = \lim_{\alpha \to 0} W_{k,\alpha}(m), \;\; m \in \mathbb{Z}^+ \tag{2.18}$$

when the above limit exists.

In light of the above discussion, we now proceed to study the single class problems (k, α, W) in the next section. We shall establish α-indexability, derive α-indices and the average cost indices which are appropriate for our admission control problems.

2.3 The Discounted Problem

We first look at the discounted routing control problem. So in this section all of our expected future costs are discounted with time according to the discount rate α. From the above discussion we can see that to obtain Whittle's indices for the original discounted cost problem in (2.3) and (2.9) we first consider the single class problem (k, α, W).

2.3.1 The single class system with a charge for rejection

Throughout this section we concentrate on the single class routing control problems (k, α, W), and so it will be notationally convenient to drop the class identifier k. The problem we look at is one of arriving customers who can be sent to the given server or rejected. However if we do reject a customer then a rejection charge must be paid. There are also holding cost charges incurred by the customers for the time they are in the system, assumed increasing convex in the number of customers in the system. If we accept a customer we must pay the resulting increased holding costs and if we do not accept we must pay a rejection charge. It is the balance between these two costs which is central to our study. For this single station we have $M/M/1$ dynamics. Hence arrivals form a Poisson(λ) stream, note that λ is the system arrival rate previously considered, i.e. the single server faces the entire arrival stream for the whole system - but we now consider the option of rejecting the arrivals. The service times follow exponential, exp(μ) distributions and *all* interarrival times and service times are independent. We can view this system pictorially in Figure 2.1. The goal here is to choose when we should accept

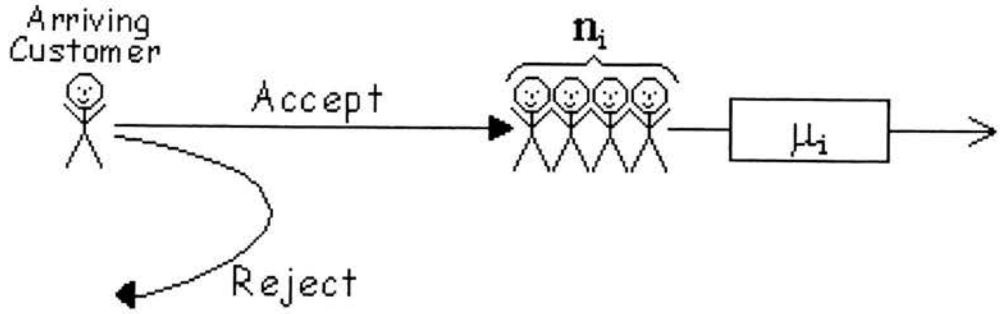

Figure 2.1: The options when considering a single class.

customers at the station in order to minimise the sum of the costs incurred through the rejection charge and through holding costs. We formulate this as a Semi Markov Decision Process (SMDP) as follows:

(a) We use $N(t)$ to denote the state of the station at time $t \in \mathbb{R}$, i.e. the number of customers at the station. Decision epochs will occur at all customer arrival times, which will be the event times of a Poisson process with rate λ. So in the problem (k, α, W) the single station will be facing the entire incoming arrival stream which has rate λ. Hence if t is a decision epoch then, regardless of the action we take, the next epoch will occur at time $t + \tilde{A}$, where $\tilde{A} \sim \exp(\lambda)$, since the inter-arrival times will be exponentially distributed. At each decision epoch the following two actions are available:

1. a (active), which is the choice to admit the arriving customer at this station, or

2. b (passive), which is the choice to not admit the arriving customer at this station.

Suppose at time t, that the station is in state $m > 0$. The next random event epoch will occur at time $t+Q$ where $Q \sim \exp(\lambda+\mu)$. If action a is taken in state m then we have that

$$N(t+Q)^+ = \begin{cases} m+1, & \text{with probability } \lambda(\lambda+\mu)^{-1}, \text{ and,} \\ m-1, & \text{with probability } \mu(\lambda+\mu)^{-1}. \end{cases}$$

If action b is taken in state m, then

$$N(t+Q)^+ = \begin{cases} m, & \text{with probability } \lambda(\lambda+\mu)^{-1}, \text{ and,} \\ m-1, & \text{with probability } \mu(\lambda+\mu)^{-1}. \end{cases}$$

(b) Let $C : \mathbb{N} \to \mathbb{R}^+$ be the increasing convex holding cost function for the station concerned and let α, W be positive constants. Hence when we have n customers present at the station the discounted holding costs will be incurred at rate $C(n)$, where recall that $C(0) = 0$. We also incur a fixed cost of W whenever we reject an arriving customer. So the total discounted expected costs incurred will be equivalent to a system where we have discounted holding costs only, incurred at rate

$C(n)$ while we are in a state where we will accept an arriving customer, and
$C(n) + \lambda W$ while we are in a state where we will not accept an arriving customer.

See also (2.20) below. Note that W is the amount charged whenever we reject from the queue in question and λ is that rate at which the charge is incurred, if we are in a state where the policy dictates that we reject.

(c) A policy is a rule for choosing between the actions a and b in the light of the system history to date. Recall now standard theory from the area of stochastic DP (see section (1.2)). This indicates the existence of and optimal policy which is stationary (makes decisions in light of the current state only) and whose value function satisfies the DP optimality equations. See Puterman (1994). If we use I_i for the indicator function

$$I_i = \begin{cases} 1, & \text{if the } i^{th} \text{ arriving customer (at time } t_i) \\ & \text{is rejected;} \\ 0, & \text{otherwise, } t \in \mathbb{R}^+ \end{cases}$$

then we can write the total expected cost incurred under policy u from initial state m as

$$V_u(m,\alpha,W) = E_u\Big[\int_0^\infty C\big(N(t)\big)e^{-\alpha t}dt + \sum_{i=1}^{\infty} W I_i e^{-\alpha t_i}|N(0)=m\Big]. \quad (2.19)$$

Now (2.19) is equivalent to

$$V_u(m,\alpha,W) = E_u\Big[\int_0^\infty \{C(N(t)) + \lambda W I(t)\}e^{-\alpha t}dt|N(0)=m\Big] \quad (2.20)$$

where we use $I(t)$ for the indicator function

$$I(t) = \begin{cases} 1, & \text{if we are in a state at time t, where policy } u \\ & \text{rejects an arriving customer} \\ 0, & \text{otherwise, } t \in \mathbb{R}^+ \end{cases}$$

The goal here is to find a policy which will minimise the cost in (2.20), which is the problem we have labelled (k,α,W). We denote this minimised total cost to be

$$V(m,\alpha,W) = \inf_u\{V_u(m,\alpha,W)\}. \quad (2.21)$$

We now develop the form of the optimality equations for this single class problem. The first thing to note is that decision epochs are the arrival times but we also have service completions occurring and both these random events change the costs incurred by the system. Hence we must consider all such events. We now consider the total expected cost under a policy from state $m > 0$ if this policy tells us to take the *active* action from this state and act optimally beyond the first event epoch. This cost will comprise the discounted cost until the next event + the discounted cost from state $m+1$ if that event is an arrival + the discounted cost from $m-1$ if that event is a service completion. Both these last two terms also need to be discounted. Hence we have that the total cost incurred is

$$
\begin{aligned}
& C(m)E\Big[\int_0^Q e^{-\alpha t}dt\Big] + \frac{\lambda}{\mu+\lambda}V(m+1,\alpha,W)E[e^{-\alpha Q}] \\
& \qquad\qquad + \frac{\mu}{\mu+\lambda}V(m-1,\alpha,W)E[e^{-\alpha Q}] \\
= \; & C(m)E\Big[\frac{1-e^{-\alpha Q}}{\alpha}\Big] + \frac{\lambda}{\mu+\lambda}V(m+1,\alpha,W)\int_0^\infty e^{-\alpha q}(\lambda+\mu)e^{-(\lambda+\mu)q}dq \\
& + \frac{\mu}{\mu+\lambda}V(m-1,\alpha,W)\int_0^\infty e^{-\alpha q}(\lambda+\mu)e^{-(\lambda+\mu)q}dq \\
= \; & \frac{C(m)}{\alpha}\Big\{1-\int_0^\infty e^{-\alpha q}(\lambda+\mu)e^{-(\lambda+\mu)q}dq\Big\} + \frac{\lambda}{\mu+\lambda}V(m+1,\alpha,W)\frac{\lambda+\mu}{\alpha+\lambda+\mu} \\
& + \frac{\mu}{\mu+\lambda}V(m-1,\alpha,W)\frac{\lambda+\mu}{\alpha+\lambda+\mu} \\
= \; & \frac{C(m)}{\alpha}\Big\{1-\frac{\lambda+\mu}{\alpha+\lambda+\mu}\Big\} + \frac{\lambda}{\alpha+\lambda+\mu}V(m+1,\alpha,W) \\
& + \frac{\mu}{\alpha+\lambda+\mu}V(m-1,\alpha,W) \\
= \; & \frac{C(m)}{\alpha+\lambda+\mu} + \frac{\lambda}{\alpha+\lambda+\mu}V(m+1,\alpha,W) + \frac{\mu}{\alpha+\lambda+\mu}V(m-1,\alpha,W) \qquad (2.22)
\end{aligned}
$$

Note that $Q \sim \exp(\lambda+\mu)$ is the time until the next event (either an arrival or service completion). We also consider total expected cost under a policy from state $m > 0$ if this policy tells us to take the *passive* action from this state and act optimally beyond the first event epoch. This cost can be constructed in a similar way i.e., the discounted cost until the next event + the discounted cost from state m if that event is an arrival + the discounted cost from $m-1$ if that event is a service completion. So the resulting discounted cost is

$$
\begin{aligned}
& \big(C(m)+\lambda W\big)E\Big[\int_0^Q e^{-\alpha t}dt\Big] \\
& \qquad + \frac{\lambda}{\mu+\lambda}V(m,\alpha,W)E[e^{-\alpha Q}] + \frac{\mu}{\mu+\lambda}V(m-1,\alpha,W)E[e^{-\alpha Q}] \\
= \; & \big(C(m)+\lambda W\big)E\Big[\frac{1-e^{-\alpha Q}}{\alpha}\Big] + \frac{\lambda}{\mu+\lambda}V(m,\alpha,W)\int_0^\infty e^{-\alpha q}(\lambda+\mu)e^{-(\lambda+\mu)q}dq \\
& + \frac{\mu}{\mu+\lambda}V(m-1,\alpha,W)\int_0^\infty e^{-\alpha q}(\lambda+\mu)e^{-(\lambda+\mu)q}dq \\
= \; & \frac{C(m)+\lambda W}{\alpha+\lambda+\mu} + \frac{\lambda}{\alpha+\lambda+\mu}V(m,\alpha,W) + \frac{\mu}{\alpha+\lambda+\mu}V(m-1,\alpha,W) \qquad (2.23)
\end{aligned}
$$

Since the choice in any state m is between taking action a or b, until the next event, the value function $V(.,\alpha,W)$ satisfies

$$
\begin{aligned}
V(m,\alpha,W) &= \min\Big\{\frac{C(m)}{\alpha+\lambda+\mu}+\frac{\lambda}{\alpha+\lambda+\mu}V(m+1,\alpha,W) \\
&\quad +\frac{\mu}{\alpha+\lambda+\mu}V(m-1,\alpha,W);\frac{C(m)+\lambda W}{\alpha+\lambda+\mu}+\frac{\lambda}{\alpha+\lambda+\mu}V(m,\alpha,W) \\
&\quad +\frac{\mu}{\alpha+\lambda+\mu}V(m-1,\alpha,W)\Big\}, m\in\mathbb{Z}^+. \qquad (2.24)
\end{aligned}
$$

In state 0, no service completions are possible. the resulting optimality equation is

$$
V(0,\alpha,W) = \min\Big[\frac{\lambda}{\alpha+\lambda}V(1,\alpha,W),\frac{\lambda W}{\alpha+\lambda}+\frac{\lambda}{\alpha+\lambda}V(0,\alpha,W)\Big].
$$

Following the discussion around Definitions 1 and 2 of Section 2.2, we write $\Pi_\alpha(W)$ for the set of states for which active action a is optimal in the above problem. We write this as

$$
\begin{aligned}
\Pi_\alpha(W) &= \{m\in\mathbb{N} \text{ such that the active action is optimal in } m \text{ when the charge for} \\
&\qquad \text{rejection is } W\}, W\in\mathbb{R} \qquad (2.25)
\end{aligned}
$$

If we have α-indexability, namely that $\Pi_\alpha(W)$ is increasing in W, we then write $W_\alpha(m)$ for the Whittle α-index for the customer class concerned in state m, as in Definition 2. We proceed to give a heuristic argument which yields a formula for $W_\alpha(m)$ in terms of model parameters when $W_\alpha(.)$ is assumed to be an *increasing function* as would seem plausible.

Consider the service control problem (a)-(c) with $N(0)=m>0$, discount rate α and with rejection charge $W=\bar{W}_\alpha(m)$ equal to the assumed value of the α index in state m. We make the following two assumptions:

1. The α-index, $W_\alpha(m)$, is increasing in the state, m, and

2. When the rejection charge, W, is equal to the α-index, $W_\alpha(m)$, in some state m, both the actions a and b are optimal in that state.

Both these assumptions will be established properly later in the analysis. We can now infer that the optimal policy for the single class problem (k, α, W) with $W = \bar{W}_\alpha(m)$ will have the form:

i) take the active action a in states $\{0, 1, 2, \ldots, m-1\}$,

ii) take the passive action b in states $\{m+1, m+2, m+3, \ldots\}$,

iii) take either the active or passive action in state m.

Note that (*i*) and (*ii*) follow from Assumption 1 and Definition 2 while (*iii*) follows from Assumption 2. So we can see that under these assumptions there are two stationary policies which are optimal when $W = \bar{W}_\alpha(m)$. We use the label u_1 for the optimal policy which chooses action a in state m, and the label u_2 for the optimal policy which chooses action b in state m. Note that both optimal policies make choices according to (*i*) and (*ii*) above. Before proceeding any further we also introduce the following random time variable:

$$T_n = \text{the time it takes for the system to translate from state } n \text{ to state } n+1 \text{ for the first time, under continuous application of the active action.} \quad (2.26)$$

Note that since the state space is bounded below by $n = 0$, we can see that T_n will have an obvious dependence on n. Since both policies u_1 and u_2 are optimal for the problem with $W = \bar{W}_\alpha(m)$, their discounted expected costs to infinity should be the same. Our approach will be to find this cost for both policies and equate them in order to obtain an expression for the index value $\bar{W}_\alpha(m)$.

Calculating the discounted cost to infinity of following policy u_1

Recall we have $N(0) = m$ so policy u_1 will take the active action a until time T_m where

$$T_m = \inf\{t; N(t) = m+1\}$$

We denote the cost incurred during this initial active phase as $\bar{C}(m,\alpha)$ where

$$\bar{C}(m,\alpha) = E\Big[\int_0^{T_m} C\{N(t)\}e^{-\alpha t}dt|N(0)=m,\ \mathrm{a}\Big] \tag{2.27}$$

Let us fist consider the situation where we have $m = 0$. Policy u_1 dictates that we should take the active action and hence accept arriving customers in state $m = 0$. Since we are in state 0, we have no customers and so an arrival is the only option possible. Hence the cost until the first event will be $\bar{C}(0,\alpha)$, then when an arrival occurs we will incur costs at the rate $C(1) + \lambda\bar{W}_\alpha(0)$. When this arrival occurs we stop admitting customers and so remain in this state until the customer is served, which will happen at time $T_0 + X$ where $X \sim \exp(\mu)$. Then the cost incurred (discounted back to the time when the customer arrived) is:

$$\bar{C}(0,\alpha)+E(e^{-\alpha T_0})\Big(C(1)+\lambda W(0)\Big)E_X\Big[\int_0^X e^{-\alpha t}dt\Big] = \bar{C}(0,\alpha)+E(e^{-\alpha T_0})\frac{C(1)+\lambda\bar{W}_\alpha(0)}{\alpha+\mu}.$$

Where we require the $E(e^{-\alpha T_0})$ coefficient since all costs must be discounted back to time 0. After the service completion we return to state 0 and the cycle continues *ad infinitum*. Hence we can find the total discounted expected cost to infinity from following this policy from state $m = 0$, by finding the sum of the discounted expected cost of these cycles to infinity. We must adjust each cycle cost to take account of the relevant discounting. Hence the total expected discounted cost to infinity can be found using the formula for the sum of a geometric progression to infinity. So we can see that this cost can be calculated as

$$V_{u_1}\{0,\alpha,\bar{W}_\alpha(0)\} = \frac{\bar{C}(0,\alpha)+E(e^{-\alpha T_0})\{C(1)+\lambda\bar{W}_\alpha(0)\}(\alpha+\mu)^{-1}}{1-\mu E(e^{-\alpha T_0})(\alpha+\mu)^{-1}} \tag{2.28}$$

We now move on to consider the cost of following policy u_1 from state $m > 0$. When the system arrives in state $m+1$, policy u_1 indicates that the passive action b be taken. Hence in state $m+1$ the only events which can occur are service completions. We will continue to take passive action b until we have a service completion and move back to state m. This service completion will occur at time $T_m + X$ where $X \sim \exp(\mu)$. Hence we can see that the discounted expected cost (from time T_m) until we return to state m (i.e. we have a service completion) will be

$$\begin{aligned} & (C(m+1)+\lambda\bar{W}_\alpha(m))E_X\Big[\int_0^X e^{-\alpha t}dt\Big], \quad \text{where } X \sim \exp(\mu) \\ = \; & \frac{C(m+1)+\lambda\bar{W}_\alpha(m)}{\alpha+\mu}. \end{aligned} \tag{2.29}$$

However this cost must also be discounted back to time 0, and so must be multiplied by $E(e^{-\alpha T_m})$. Hence, when following policy u_1, we can see that the expected discounted cost to move through a cycle from state m to $m+1$ then back to m is

$$\bar{C}(m,\alpha)+E(e^{-\alpha T_m})\frac{C(m+1)+\lambda\bar{W}_\alpha(m)}{\alpha+\mu}. \tag{2.30}$$

Policy u_1 now repeats this above cycle *ad infinitum* from time T_m+X. Hence we can find the total discounted expected cost to infinity of following this policy by finding the sum of the discounted expected cost of these cycles to infinity. We must remember to adjust each cycle cost by the relevant discounting term. When we take this discounting into account it can be seen that the total expected discounted cost to infinity can be found using the formula for the sum of a geometric progression to infinity. So the total expected discounted cost associated with this policy may be calculated as

$$V_{u_1}\{m,\alpha,\bar{W}_\alpha(m)\} = \frac{\bar{C}(m,\alpha)+E(e^{-\alpha T_m})\{C(m+1)+\lambda\bar{W}_\alpha(m)\}(\alpha+\mu)^{-1}}{1-\mu E(e^{-\alpha T_m})(\alpha+\mu)^{-1}}. \tag{2.31}$$

We now find an expression for the cost of following policy u_2.

Calculating the discounted cost to infinity of following policy u_2

Again lets us first of all consider the situation where we have $m=0$. Following policy u_2 we take the passive action in this state $m=0$, and so do not admit any customers. However since we are in the empty state we also cannot serve. Hence we will merely incur costs at the rate $C(0)+\lambda=\lambda W$. at all times. So the total discounted cost to infinity of following policy u_2 from state $m = 0$ is

$$V_{u_2}\{0, \alpha, \bar{W}_\alpha(0)\} = \lambda \bar{W}_\alpha(0) \int_0^\infty e^{-\alpha t} dt = \frac{\lambda \bar{W}_\alpha(0)}{\alpha}. \tag{2.32}$$

We now move on to look at the cost of following policy u_2 from a situation where we have $N(0) = m > 0$. Under policy u_2 the passive action b is taken in state m. From the arguments above one can see that the first event to occur after time zero in this instance must be a service completion which will occur at time X where $X \sim \exp(\mu)$. So we can see the discounted expected cost incurred until this event is

$$\big(C(m) + \lambda \bar{W}_\alpha(m)\big) E_X\Big[\int_0^X e^{-\alpha t} dt\Big] = \frac{C(m) + \lambda \bar{W}_\alpha(m)}{\alpha + \mu}. \tag{2.33}$$

When this event occurs the system state will move to state $m - 1$ and policy u_2 dictates that in this state we should take the active action a until the state returns to m. So now we will have events which could either be service completions or customer arrivals. Using the notation above one can see that the discounted expected cost until we return to state m from $m - 1$ is $\bar{C}(m - 1, \alpha)$. However this cost must be discounted back from the time when the service completion occurred, say Y to time 0, i.e. we need to multiply it by the term

$$E(e^{-\alpha Y}) = \int_0^\infty \mu e^{-(\alpha+\mu)y} dy = \frac{\mu}{\alpha + \mu}.$$

So we can see that under policy u_2 the system will also follow a cycle, from state m to $m - 1$ then back to state m. The expected discounted cost of this first cycle will be

$$\frac{C(m) + \lambda \bar{W}_\alpha(m)}{\alpha + \mu} + \frac{\mu \bar{C}(m - 1, \alpha)}{\alpha + \mu}. \tag{2.34}$$

The subsequent cycles must also be discounted back to time 0 accordingly, which allows us to find the discounted expected cost to infinity from following policy u_2 as follows:

$$V_{u_2}\{m,\alpha,\bar{W}_\alpha(m)\} = \frac{(C(m)+\lambda\bar{W}_\alpha(m))(\alpha+\mu)^{-1} + (\mu\bar{C}(m-1,\alpha))(\alpha+\mu)^{-1}}{1-\mu E(e^{-\alpha T_{m-1}})(\alpha+\mu)^{-1}}. \tag{2.35}$$

We now have the discounted expected cost to infinity from following both policies u_1 and u_2. Since these policies are both optimal then these costs will be identical. We now use this fact to find an expression for the index $\bar{W}_\alpha(m)$. Firstly we consider that $m = 0$ case, equating (2.28) and (2.32) leads us to

$$\frac{(\alpha+\mu)\bar{C}(0,\alpha) + E(e^{-\alpha T_0})\big(C(1)+\lambda\bar{W}_\alpha(0)\big)}{\alpha+\mu-\mu E(e^{-\alpha T_0})} = \frac{\lambda\bar{W}_\alpha(0)}{\alpha}. \tag{2.36}$$

Also equating (2.31) with (2.35) leads us to

$$\frac{(\alpha+\mu)\bar{C}(m,\alpha) + E(e^{-\alpha T_m})\{C(m+1)+\lambda\bar{W}_\alpha(m)\}}{\alpha+\mu-\mu E(e^{-\alpha T_m})} = \frac{C(m)+\lambda\bar{W}_\alpha(m)+\mu\bar{C}(m-1,\alpha)}{\alpha+\mu-\mu E(e^{-\alpha T_{m-1}})}. \tag{2.37}$$

We would like to solve this equation to obtain $\bar{W}_\alpha(m)$. However before one could practically find $\bar{W}_\alpha(m)$, it would assist matters greatly if expressions could be found for $\bar{C}(m,\alpha)$ and $E(e^{-\alpha T_m})$. We firstly consider $\bar{C}(0,\alpha)$. Plainly in state 0 the only possible events are customer arrivals, so we have

$$\begin{aligned} \bar{C}(0,\alpha) &= C(0)E_{\tilde{A}}\Big[\int_0^{\tilde{A}} e^{-\alpha t}dt\Big], \quad \text{where } \tilde{A}\sim\exp(\lambda) \\ &= 0 \end{aligned} \tag{2.38}$$

since we know that $C(0) = 0$. We now study $\bar{C}(m,\alpha)$ for $m > 0$. We build an expression for this cost using standard conditioning arguments. We can see that this cost will be made up of the following elements: the cost until the first random event; if the first event is a service completion then we also need the discounted cost from state $m-1$ to state m followed by the discounted cost from state m to state $m+1$; if the first event is a customer arrival then the system is in state $m+1$ and no further costs are incurred. For the sake of brevity the following notation has been used:

$$\begin{aligned} X_m &= E(e^{-\alpha T_m}), \\ Q &= \text{time until the next (first) event, when in the active state,} \end{aligned}$$

where $Q \sim \exp(\lambda + \mu)$. The expected discounted cost until, Q, can then be written as

$$\begin{aligned} & E_Q\Big(\int_0^Q C(m)e^{-\alpha t}dt\Big) \\ =\ & E_Q\Big(\frac{C(m)}{\alpha}(1 - e^{-\alpha Q})\Big) \\ =\ & \frac{C(m)}{\alpha}\Big(1 - \int_0^\infty (\mu+\lambda)e^{-(\alpha+\mu+\lambda)q}dq\Big) \\ =\ & \frac{C(m)}{\alpha+\lambda+\mu}. \end{aligned} \tag{2.39}$$

Using this notation the above conditioning arguments yield the following:

$$\begin{aligned} \bar{C}(m,\alpha) &= E\Big(\int_0^Q C(m)e^{-\alpha u}du\Big) + \frac{\mu}{\lambda+\mu}\Big[\bar{C}(m-1,\alpha)E(e^{-\alpha Q}) \\ &\qquad + \bar{C}(m,\alpha)E(e^{-\alpha Q})X_{m-1}\Big] + \frac{\lambda}{\lambda+\mu}\Big[0\Big] \\ &= \frac{C(m)}{\lambda+\mu+\alpha} + \frac{\mu}{\lambda+\mu}\Big[\bar{C}(m-1,\alpha)\int_0^\infty (\lambda+\mu)e^{-(\alpha+\lambda+\mu)q}dq \\ &\qquad + \bar{C}(m,\alpha)X_{m-1}\int_0^\infty (\lambda+\mu)e^{-(\alpha+\lambda+\mu)q}dq\Big] \\ &= \frac{C(m) + \mu\bar{C}(m-1,\alpha) + \mu\bar{C}(m,\alpha)X_{m-1}}{\alpha+\lambda+\mu} \\ \Longrightarrow \bar{C}(m,\alpha) &= \frac{\mu\bar{C}(m-1,\alpha) + C(m)}{\alpha+\lambda+\mu-\mu X_{m-1}}. \end{aligned} \tag{2.40}$$

Using similar conditioning arguments we can also find an expression for the term $E(e^{-\alpha T_m})$. Again we will initially consider $E(e^{-\alpha T_0})$, since in this state the only events that can occur are arrivals, hence

$$\begin{aligned} X_0 = E(e^{-\alpha T_0}) &= \int_0^\infty \lambda e^{-(\alpha+\lambda)t}dt \\ &= \frac{\lambda}{\alpha+\lambda}. \end{aligned} \tag{2.41}$$

Now looking at $E(e^{-\alpha T_m})$ $(= X_m)$ for $m > 0$ using these standard conditioning arguments leads us to

$$\begin{aligned} X_m &= E(e^{-\alpha Q})\Big[\frac{\mu}{\lambda+\mu}X_{m-1}X_m + \frac{\lambda}{\lambda+\mu}\times 1\Big] \\ &= \frac{\lambda+\mu}{\alpha+\lambda+\mu}\Big[\frac{\mu X_{m-1}X_m + \lambda}{\lambda+\mu}\Big] \\ \Longrightarrow X_m = E(e^{-\alpha T_m}) &= \frac{\lambda}{\alpha+\lambda+\mu-\mu X_{m-1}}. \qquad (2.42) \end{aligned}$$

Again considering the situation where $m = 0$ first, we can see that using relations (2.38) and (2.41) and simplifying we can see that (2.36) leads us to

$$\begin{aligned} \frac{C(1)+\lambda\bar{W}_\alpha(0)}{(\alpha+\mu)(\alpha+\lambda)-\mu\lambda} &= \frac{\bar{W}_\alpha(0)}{\alpha} \\ \Leftrightarrow \alpha C(1) + \alpha\lambda\bar{W}_\alpha(0) &= \bar{W}_\alpha(0)\big[(\alpha+\mu)(\alpha+\lambda)-\mu\lambda\big] \\ \Leftrightarrow \bar{W}_\alpha(0) &= \frac{C(1)}{\alpha+\mu}. \qquad (2.43) \end{aligned}$$

Also for the case $m > 0$, using forms of the relations (2.40), (2.42) and simplifying we can see that (2.37) implies that

$$\begin{aligned} \lambda\bar{W}_\alpha(m)\Big\{\frac{1}{X_{m+1}} - 2 + X_m\Big\} &= C(m+1)[1-X_m] - \alpha\bar{C}(m,\alpha) \\ \Longrightarrow \lambda\bar{W}_\alpha(m)\Big\{\frac{1-2X_{m+1}+X_mX_{m+1}}{1-X_m}\Big\} &= X_{m+1}\Big\{C(m+1) - \frac{\alpha\bar{C}(m,\alpha)}{1-X_m}\Big\} \\ \Longrightarrow \lambda\bar{W}_\alpha(m)\Big\{\frac{1-X_{m+1}}{1-X_m} - X_{m+1}\Big\} &= X_{m+1}\Big\{C(m+1) - \frac{\alpha\bar{C}(m,\alpha)}{1-X_m}\Big\}. \end{aligned}$$

Hence the expression we have inferred from the above argument for the α-index is

$$\bar{W}_\alpha(m) = \frac{X_{m+1}}{\lambda}\Big\{C(m+1) - \frac{\alpha\bar{C}(m,\alpha)}{1-X_m}\Big\}\Big/\Big\{\frac{1-X_{m+1}}{1-X_m} - X_{m+1}\Big\}. \qquad (2.44)$$

Using the the relations (2.38), (2.41) and (2.42) for $\bar{C}(0,\alpha)$, X_0 and X_1 we can see that expression (2.43) is equivalent to expression (2.44) when $m = 0$.

The following Lemma asserts that our conjectured index $\bar{W}_\alpha(m)$ is increasing in m, as was assumed to be the case for the true index in the argument used to infer this index expression.

Lemma 1

$\bar{W}_\alpha(m)$ is increasing in m.

Proof

Firstly note from (2.44), that the formula for the index, $\bar{W}_\alpha(m)$, is quite complex so proving $\bar{W}_\alpha(m)$ is increasing with m could be difficult. For this reason we split the proof into two parts:

A. Prove that

$$C(m+1) - \frac{\alpha \bar{C}(m,\alpha)}{1-X_m} \tag{2.45}$$

is positive and increasing with m.

B. Prove that

$$X_{m+1}/\Big(\frac{1-X_{m+1}}{1-X_m} - X_{m+1}\Big) = \frac{X_{m+1}(1-X_m)}{1-X_{m+1}-X_{m+1}(1-X_m)} \tag{2.46}$$

is positive and increasing with m.

Obviously if we can prove A and B, Lemma 1 will follow as an immediate consequence.

However before this we shall show that X_m is decreasing with m. This relation will be useful throughout the proof. We use a proof by induction to show this relation holds. The first thing we must do is prove the initial case, i.e. show that

$$X_0 \geq X_1. \tag{2.47}$$

Now by use of (2.41) and (2.42), we can see that (2.47) is equivalent to

$$\begin{aligned} \alpha+\lambda+\mu-\mu X_0 &\geq \frac{\lambda}{X_0} \\ \Leftrightarrow \alpha+\lambda+\mu-\mu X_0 &\geq \alpha+\lambda \\ \Leftrightarrow \mu &\geq \mu X_0 \\ \Leftrightarrow 1 &\geq \frac{\lambda}{\alpha+\lambda} \end{aligned}$$

Since we know that the discount rate α must be positive we have shown (2.47) is true. Now that we have proved the initial case we use the induction hypothesis $X_{j-1} \geq X_j$ to infer that $X_j \geq X_{j+1}$, $j > 0$. By use of relation (2.42) we can see that what we must infer is

$$\begin{aligned} X_j &\geq X_{j+1} \\ \Leftrightarrow \frac{\lambda}{\alpha+\lambda+\mu-\mu X_{j-1}} &\geq \frac{\lambda}{\alpha+\lambda+\mu-\mu X_j} \\ \Leftrightarrow -\mu X_j &\geq -\mu X_{j-1} \\ \Leftrightarrow X_j &\leq X_{j-1}. \end{aligned}$$

Note that in the second line of the above working we multiply through by $(\alpha+\lambda+\mu-\mu X_{j-1})(\alpha+\lambda+\mu-\mu X_j)$ to get to the third line. We know this quantity is positive because,

$$\mu - \mu X_i \geq 0 \quad \text{since } 0 \leq X_i \equiv E(e^{-\alpha T_i}) \leq 1 \quad \forall\, i.$$

One can see that the last line is just our induction hypothesis and hence we have shown that X_m is decreasing with m as required.

We firstly look at showing that the quantity in (2.46) is positive. We now have that $X_m \geq X_{m+1}$ and know that $X_m, X_{m+1} \in (0,1)$ hence we can see that

$$\begin{aligned} 1 - X_{m+1} - X_{m+1}(1-X_m) &> 0 \qquad \text{and,} \\ X_{m+1}(1-X_m) &> 0. \end{aligned} \tag{2.48}$$

It therefore follows that the quantity is (2.46) is also positive. We now prove that the quantity in (2.46) is increasing with m. Notice now that the expression in (2.46) is equal to

$$1 \Big/ \Big(\frac{1-X_{m+1}}{(1-X_m)X_{m+1}} - 1 \Big).$$

Hence it is enough to show that $(1-X_{m+1}) \big/ (1-X_m)X_{m+1}$ is greater than 1 and decreasing with m. Now we have that $X_m \geq X_{m+1}$ which implies that

$$\begin{aligned} 1 - X_{m+1} &\geq 1 - X_m \\ \Leftrightarrow \frac{1 - X_{m+1}}{1 - X_m} &\geq 1 \\ \Leftrightarrow \frac{1 - X_{m+1}}{X_{m+1}(1 - X_m)} &\geq 1, \end{aligned}$$

as required. We also wish to show that

$$\begin{aligned} \frac{1 - X_{m+1}}{(1 - X_m)X_{m+1}} &\leq \frac{1 - X_m}{(1 - X_{m-1})X_m} \\ \Leftrightarrow (1 - X_{m+1})\big[\lambda - (\alpha + \lambda)X_m\big]\mu^{-1} &\leq (1 - X_m)\big[\lambda - (\alpha + \lambda)X_{m+1}\big]\mu^{-1} \\ \Leftrightarrow (\alpha + \lambda)X_{m+1} - \lambda X_{m+1} &\leq (\alpha + \lambda)X_m - \lambda X_m \\ \Leftrightarrow X_{m+1} &\leq X_m, \qquad \text{since } \alpha > 0. \end{aligned}$$

By the previous proof on page 46 we have shown this to be true, and hence we have shown that (2.46) is indeed increasing, as required. Note that we get to the second line in the above working by using the relation (2.42) to infer that

$$\begin{aligned} \lambda &= X_n(\alpha + \lambda + \mu - \mu X_{n-1}) \\ \Leftrightarrow \lambda &= \mu X_n(1 - X_{n-1}) + (\alpha + \lambda)X_n \\ \Leftrightarrow \lambda - (\alpha + \lambda)X_n &= \mu X_n(1 - X_{n-1}), \ \forall\, n > 0. \end{aligned}$$

We have now proved part B, that the quantity in (2.46) is positive and increasing with m and so now must move on to prove part A, that the expression in (2.45) is positive and increasing with m. We can see from the increasing nature of the cost function C that the expression in (2.45) will be positive. This is due to the fact that

$$\begin{aligned} \frac{\alpha \bar{C}(m, \alpha)}{1 - X_m} &= \frac{\bar{C}(m, \alpha)}{E\big[\int_0^{T_m} e^{-\alpha t} dt\big]} \\ &= \sum_{n=0}^{m} C(n) g_n \text{ where } \sum_{n=0}^{m} g_n = 1. \end{aligned}$$

In other words we can see that $\alpha \bar{C}(m, \alpha)(1 - X_m)^{-1}$ is a weighted average of the cost rates incurred until the system gets to state $m + 1$. Hence we must have that

$C(m+1) > \alpha\bar{C}(m,\alpha)(1-X_m)^{-1}$. Proving the increasing nature of part A does turn out to be somewhat more difficult. To do this we introduce the following performance variable

y_i – the discounted expected time spent in state i, from time 0 to infinity, when starting in state m, under the policy which admits customers in states $\{0,1,2,\ldots,m\}$ only.

Because of the definition of y_i we can see that $y_i = 0$ for all $i \geq m+2$. We can write this definition mathematically as follows,

$$y_i = E_u\Big[\int_0^\infty I\{N(t)=i\}e^{-\alpha t}dt\Big|N(0)=m\Big], \tag{2.49}$$

where u is the policy which admits customers in states $\{0,1,2,\ldots,m\}$ only. We firstly try to formulate an expression for y_{m+1}. To do this consider the following state transition diagram for a single cycle shown in Figure 2.2. In Figure 2.2 X is the a single service time and hence $X \sim \exp(\mu)$, and so we can use the memoryless property of the exponential distribution. From this diagram we can see that the discounted expected time spent in state $m+1$ in the first time loop is

$$\begin{aligned} & E(e^{-\alpha T_m})E\{\int_0^X e^{-\alpha t}dt\} \\ = & \frac{E(e^{-\alpha T_m})}{\alpha+\mu}. \end{aligned} \tag{2.50}$$

The time in subsequent loops will take the same form initially but will obviously also require further discounting. So for example the contribution to y_{m+1} in the second loop will also need to be discounted by

$$\begin{aligned} & E(e^{-\alpha T_m})E(e^{-\alpha X}) \\ = & \frac{\mu E(e^{-\alpha T_m})}{\alpha+\mu}. \end{aligned} \tag{2.51}$$

So we can see that y_{m+1} is the sum of a geometric progression to infinity and hence

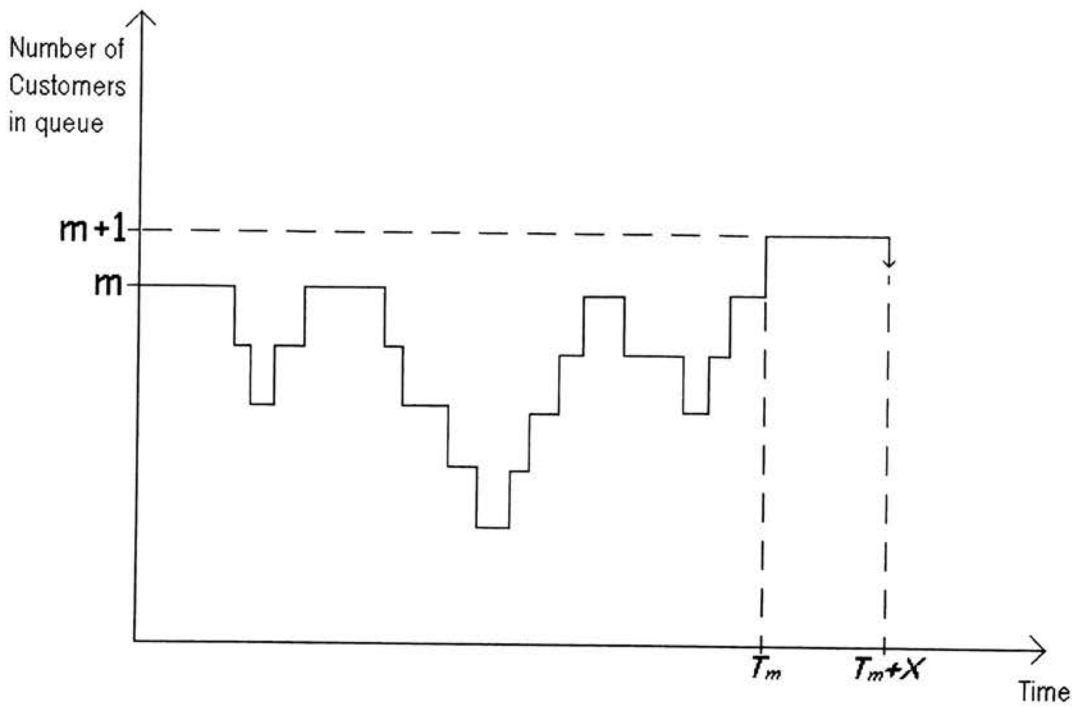

Figure 2.2: Possible state transition diagram from state m.

we have that

$$
\begin{aligned}
y_{m+1} &= \frac{E(e^{-\alpha T_m})(\alpha+\mu)^{-1}}{1-\mu E(e^{-\alpha T_m})(\alpha+\mu)^{-1}} \\
&= \frac{E(e^{-\alpha T_m})}{\alpha+\mu-\mu E(e^{-\alpha T_m})} \\
&= \frac{X_m}{\alpha+\mu-\mu X_m}. \qquad (2.52)
\end{aligned}
$$

We now consider the relationship between the y's. To help simplify matters we use a tool from standard theory called uniformisation, in which events are deemed to occur at a uniform rate in all states. This means that we will allow virtual arrivals to occur, in state $m+1$ but these will have no effect on the state. We will also allow virtual service completions in state 0. We use Q to denote a generic between event time. We know that $Q \sim \exp(\lambda+\mu)$. Now let us first find an alternative expression for y_{m+1}. The system can enter state $m+1$ in one of two ways, either

1. The system is in state m and a customer arrival occurs, or

2. The system is in state $m+1$ and a virtual customer arrival occurs.

We can deduce that

$$\begin{aligned} y_{m+1} &= y_m E(e^{-\alpha Q})\frac{\lambda}{\lambda+\mu} + y_{m+1}E(e^{-\alpha Q})\frac{\lambda}{\lambda+\mu} \\ &= y_m \frac{\lambda+\mu}{\alpha+\lambda+\mu}\frac{\lambda}{\lambda+\mu} + y_{m+1}\frac{\lambda+\mu}{\alpha+\lambda+\mu}\frac{\lambda}{\lambda+\mu} \\ \Leftrightarrow (\alpha+\mu)y_{m+1} &= \lambda y_m. \end{aligned} \tag{2.53}$$

We now follow a similar argument to obtain an expression for y_m. However we must remember that m is assumed to be the initial state of the system. The discounted expected time in state m until the first event is therefore given by

$$\begin{aligned} & E\Big[\int_0^Q e^{-\alpha t}dt\Big] \\ = & \frac{1}{\alpha+\lambda+\mu}. \end{aligned}$$

The system can subsequently enter state m (> 0) in one of two ways, either

1. the system is in state $m-1$ and a customer arrival occurs, or
2. the system in state $m+1$ and a service completion occurs.

Hence we deduce that

$$\begin{aligned} y_m &= E\Big[\int_0^Q e^{-\alpha t}dt\Big] + y_{m-1}E(e^{-\alpha Q})\frac{\lambda}{\lambda+\mu} + y_{m+1}E(e^{-\alpha Q})\frac{\mu}{\lambda+\mu} \\ &= \frac{1}{\alpha+\lambda+\mu} + y_{m-1}\frac{\lambda+\mu}{\alpha+\lambda+\mu}\frac{\lambda}{\lambda+\mu} + y_{m+1}\frac{\lambda+\mu}{\alpha+\lambda+\mu}\frac{\mu}{\lambda+\mu} \\ \Leftrightarrow (\alpha+\lambda+\mu)y_m &= 1 + \mu y_{m+1} + \lambda y_{m-1}. \end{aligned} \tag{2.54}$$

We can again follow a similar argument for y_j, $1 \leq j \leq m-1$, for $m \geq 2$, since the system can enter state j by two possible routes, either

1. the system is in state $j-1$ and a customer arrival occurs, or

2. the system is in state $j+1$ and a service completion occurs.

Hence for $1 \leq j \leq m-1$, for $m \geq 2$ we have that

$$\begin{aligned} y_j &= y_{j-1}E(e^{-\alpha Q})\frac{\lambda}{\lambda+\mu} + y_{j+1}E(e^{-\alpha Q})\frac{\mu}{\lambda+\mu} \\ \Leftrightarrow (\alpha+\lambda+\mu)y_j &= \mu y_{j+1} + \lambda y_{j-1}. \end{aligned} \tag{2.55}$$

Finally we consider y_0, for $m > 0$. The system can enter state 0 via two possible routes, either

1. the system is in state 0 and a virtual service completion occurs, or
2. the system is in state 1 and a service completion occurs.

Hence we deduce that

$$\begin{aligned} y_0 &= y_0E(e^{-\alpha Q})\frac{\mu}{\lambda+\mu} + y_1E(e^{-\alpha Q})\frac{\mu}{\lambda+\mu} \\ \Leftrightarrow (\alpha+\lambda)y_0 &= \mu y_1. \end{aligned} \tag{2.56}$$

We now introduce another similar, but different performance variable,

z_i – the discounted expected time spent in state i, from time 0 to infinity, when starting in state $m-1$, under the policy which admits customers in states $\{0, 1, 2, \ldots, m-1\}$ only.

Note that here we must have $m > 0$. Hence in this case, the transition diagram for a single cycle will take the form as shown in Figure 2.3. In Figure 2.3 X is again a single service time, and hence $X \sim \exp(\mu)$. Similar arguments to those previously seen yield,

$$\begin{aligned} z_m &= \frac{E(e^{-\alpha T_{m-1}})}{\alpha+\mu-\mu E(e^{-\alpha T_{m-1}})} \\ &= \frac{X_{m-1}}{\alpha+\mu-\mu X_{m-1}}. \end{aligned} \tag{2.57}$$

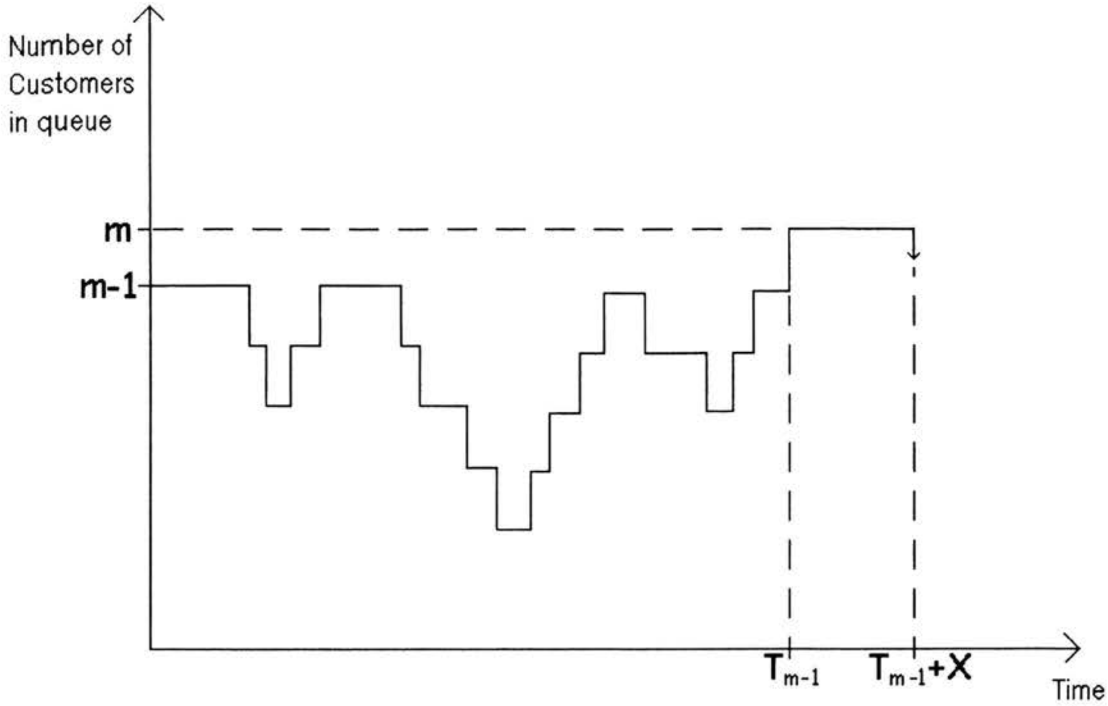

Figure 2.3: Possible state transition diagram from state $m-1$.

Also by applying a similar analysis to the above we deduce that

$$(\alpha+\mu)z_m = \lambda z_{m-1}, \tag{2.58}$$

$$(\alpha+\lambda+\mu)z_{m-1} = 1+\mu z_m+\lambda z_{m-2}, \tag{2.59}$$

$$(\alpha+\lambda+\mu)z_j = \mu z_{j+1}+\lambda z_{j-1}, \text{ for } 1 \leq j \leq m-2, \text{ where } m \geq 3 \tag{2.60}$$

$$(\alpha+\lambda)z_0 = \mu z_1 \text{ where } m > 1. \tag{2.61}$$

Now recall that we have previously shown that $X_{n-1} > X_n$, $n \geq 1$. Using this fact and the formulae (2.52) and (2.57) we can see that

$$y_{m+1} < z_m. \tag{2.62}$$

Now using (2.62) we can see that

$$y_m = \frac{\alpha+\mu}{\lambda} y_{m+1} < \frac{\alpha+\mu}{\lambda} z_m = z_{m-1}. \tag{2.63}$$

Suppose that we can write the solution to (2.53) - (2.56) in the algebraic form

$$y_r = K_r y_{m+1} + A_r, \text{ for } 0 \leq r \leq m, \tag{2.64}$$

for some $K_r \in \mathbb{R}$, $A_r \in \mathbb{R}$. We now verify that (2.64) is indeed true. From (2.53) we have that

$$\begin{aligned} y_m &= \frac{(\alpha+\mu)y_{m+1}}{\lambda} \\ &= K_m y_{m+1} + A_m, \quad \text{where } K_m = (\alpha+\mu)\lambda^{-1},\ A_m = 0. \end{aligned} \tag{2.65}$$

Then from (2.54) and (2.65) we have that

$$\begin{aligned} y_{m-1} &= \frac{\alpha+\lambda+\mu}{\lambda}\big[K_m y_{m+1} + A_m\big] - \frac{1}{\lambda} - \frac{\mu y_{m+1}}{\lambda} \\ &\equiv K_{m-1}y_{m+1} - A_{m-1}, \\ &\quad \text{where } K_{m-1} = (\alpha+\lambda+\mu)\lambda^{-1}K_m - \mu\lambda^{-1}, \\ &\quad \text{and} \quad A_{m-1} = -\lambda^{-1}. \end{aligned} \tag{2.66}$$

Similarly from (2.55) we have that

$$y_r = (\alpha+\lambda+\mu)\lambda^{-1}y_{r+1} - \mu\lambda^{-1}y_{r+2}, \text{ for } 0 \le r \le m-2. \tag{2.67}$$

Finally to verify that the relation in (2.64) is true, we use a proof by induction. We have show that the relation does hold for y_m and y_{m-1} so we can now take the induction hypothesis

$$\begin{aligned} y_{r+2} &= K_{r+2}y_{m+1} + A_{r+2}, \text{ and} \\ y_{r+1} &= K_{r+1}y_{m+1} + A_{r+1}. \end{aligned}$$

Then infer that the relation holds for y_r. Now from (2.67) and the induction hypothesis we can see that for $0 \le r \le m-2$ we have

$$\begin{aligned} y_r &= (\alpha+\lambda+\mu)\lambda^{-1}\big[K_{r+1}y_{m+1} + A_{r+1}\big] - \mu\lambda^{-1}\big[K_{r+2}y_{m+1} + A_{r+2}\big] \\ &\equiv K_r y_{m+1} + A_r, \\ &\quad \text{where } K_r = (\alpha+\lambda+\mu)\lambda^{-1}K_{r+1} - \mu\lambda^{-1}K_{r+2}, \\ &\quad \text{and} \quad A_r = (\alpha+\lambda+\mu)\lambda^{-1}A_{r+1} - \mu\lambda^{-1}A_{r+2}, \text{ for } 0 \le r \le m-2. \end{aligned} \tag{2.68}$$

So we have shown that relation (2.64) does hold. We can rewrite (2.68) as,

$$\mu A_{r+1} - (\alpha+\lambda+\mu)A_r + \lambda A_{r-1} = 0, \text{ for } 1 \le r \le m-1. \tag{2.69}$$

Standard theory tell us that the solution to (2.69) is of the form

$$A_r = Aw_1^r + Bw_2^r, \text{ for } 1 \leq r \leq m-1. \tag{2.70}$$

where w_1, w_2 are the distinct roots of the quadratic

$$\mu x^2 - (\alpha + \lambda + \mu)x + \lambda = 0. \tag{2.71}$$

On studying the quadratic in (2.71) it is easy to see that both roots are positive, one of them less than one and the other greater than one. Now without loss of generality we take w_1 to be the smaller root, i.e. we have

$$0 < w_1 < 1 < w_2.$$

We now utilize boundary conditions to obtain the constants A and B. We can easily show that

$$\begin{aligned} A_{m-1} = -\lambda^{-1} &= Aw_1^{m-1} + Bw_2^{m-1}. \\ A_{m-2} = -(\alpha + \lambda + \mu)\lambda^{-2} &= Aw_1^{m-2} + Bw_2^{m-2}. \end{aligned}$$

Solving for B yields

$$B = \frac{(\alpha + \lambda + \mu)w_1 - \lambda}{\lambda^2 (w_2 - w_1) w_2^{m-2}}. \tag{2.72}$$

From (2.72) we can see that $B > 0$ if and only if

$$\begin{aligned} (\alpha + \lambda + \mu)w_1 - \lambda &> 0 \\ \Leftrightarrow w_1 &> \lambda(\alpha + \lambda + \mu)^{-1}. \end{aligned} \tag{2.73}$$

Recall that w_1 is the smaller root of the quadratic in (2.71). If we evaluate quadratic (2.71) at $x = \lambda(\alpha + \lambda + \mu)^{-1} < 1$ we find that the result is positive which implies that x must lie between 0 and w_1 and hence the inequality in (2.73) is indeed true. Therefore we can now conclude that $B > 0$. Now recall that

$$\begin{aligned} Aw_1^{m-1} + Bw_2^{m-1} &= -\lambda^{-1} \\ \Leftrightarrow A &= \frac{-1}{\lambda w_1^{m-1}} - B\Big(\frac{w_2}{w_1}\Big)^{m-1}. \end{aligned}$$

Using this expression for A in (2.70) we can see that for $1 \leq r \leq m-1$ we have

$$\begin{aligned} A_r &= \Big[\frac{-1}{\lambda w_1^{m-1}} - B\Big(\frac{w_2}{w_1}\Big)^{m-1}\Big] w_1^r + B w_2^r \\ &= w_1^r \Big[\frac{-1}{\lambda w_1^{m-1}} - B\Big(\frac{w_2}{w_1}\Big)^{m-1} + B\Big(\frac{w_2}{w_1}\Big)^r\Big] \\ &\leq 0. \end{aligned}$$

From the definition of y_i we have that $y_i > 0$ for $0 \leq r \leq m+1$ hence from (2.64) we can see that we must have

$$K_r > 0, \text{ for } 1 \leq r \leq m. \tag{2.74}$$

One can now repeat the above process for the z_i to discover that we have

$$z_{r-1} = K_r z_m + A_r, \text{ for } 1 \leq r \leq m,$$

where the K_r and A_r are as in (2.64). Using this and the inequalities in (2.62), (2.63) and (2.74) it can be seen that

$$y_r < z_{r-1}, \text{ for all } 1 \leq r \leq m+1. \tag{2.75}$$

However if you recall, the quantity that we are actually interested in is (2.45), i.e.

$$C(m+1) - \frac{\alpha \bar{C}(m,\alpha)}{1 - X_m}.$$

We consider, as before, the station with $N(0) = m$ under a policy which takes the active action (admits customers) on states $\{0, 1, \ldots, m\}$ only. The expected holding cost for such a system can now be written as

$$\sum_{n=0}^{m+1} C(n) y_n.$$

Let us now define

$\hat{y}_n$ = discounted time spent in state n, when starting from state m, until the time when we enter state $m+1$ for the first time, under policy u_1.

Using this definition we can see that

$$\bar{C}(m,\alpha) = \sum_{n=0}^{m} C(n)\hat{y}_n. \tag{2.76}$$

Note that the above summation is only up to state m. We can see that this system will have the recursive nature where it starts in state m, has a period of activity until it reaches state $m+1$ then remains in this state until it returns to state m where this process is repeated ad infinitum. From this recursive nature we can see that

$$\begin{aligned} y_n &= \hat{y}_n\Big[1 + \Big(\frac{\mu X_m}{\alpha+\mu}\Big) + \Big(\frac{\mu X_m}{\alpha+\mu}\Big)^2 + \dots\Big] \\ &= \frac{(\alpha+\mu)\hat{y}_n}{\alpha+\mu-\mu X_m}. \end{aligned} \tag{2.77}$$

We can now use (2.77) and (2.76) to see that

$$\begin{aligned} C(m+1) - \frac{\alpha\bar{C}(m,\alpha)}{1-X_m} &= C(m+1) - \alpha\sum_{n=0}^{m} C(n)y_n\frac{\alpha+\mu-\mu X_m}{(\alpha+\mu)(1-X_m)} \\ &= C(m+1) - \alpha\sum_{n=0}^{m} C(n)\bar{y}_n \\ &\quad \text{where } \bar{y}_n = y_n\frac{\alpha+\mu-\mu X_m}{(\alpha+\mu)(1-X_m)}. \end{aligned} \tag{2.78}$$

Following a similar method (but using the z_i) we can show that,

$$\begin{aligned} C(m) - \frac{\alpha\bar{C}(m-1,\alpha)}{1-X_{m-1}} &= C(m) - \alpha\sum_{n=0}^{m-1} C(n)\bar{z}_n \\ &= \text{where } \bar{z}_n = z_n\frac{\alpha+\mu-\mu X_{m-1}}{(\alpha+\mu)(1-X_{m-1})}. \end{aligned} \tag{2.79}$$

Now we have previously shown that $X_{m-1} \geq X_m$ hence we have that

$$\begin{aligned} X_{m-1}\big(1-\mu(\alpha+\mu)^{-1}\big) &\geq X_m\big(1-\mu(\alpha+\mu)^{-1}\big) \\ \Leftrightarrow -X_m - X_{m-1}\big(\mu(\alpha+\mu)^{-1}\big) &\geq -X_{m-1} - X_m\mu(\alpha+\mu)^{-1} \\ \Leftrightarrow 1 - X_m - X_{m-1}\mu(\alpha+\mu)^{-1} + X_{m-1}X_m\mu(\alpha+\mu)^{-1} &\geq 1 - X_{m-1} - X_m\mu(\alpha+\mu)^{-1} \\ &\quad + X_{m-1}X_m\mu(\alpha+\mu)^{-1} \\ \Leftrightarrow \Big(1 - X_{m-1}\mu(\alpha+\mu)^{-1}\Big)(1-X_m) &\geq \Big(1 - X_m\mu(\alpha+\mu)^{-1}\Big)(1-X_{m-1}) \\ \Leftrightarrow \frac{\alpha+\mu-\mu X_{m-1}}{(\alpha+\mu)(1-X_{m-1})} &\geq \frac{\alpha+\mu-\mu X_m}{(\alpha+\mu)(1-X_m)}. \end{aligned}$$

Therefore using this above inequality we can see from (2.75) that

$$\bar{y}_m < \bar{z}_{m-1};\ \bar{y}_{m-1} < \bar{z}_{m-1};\ \ldots ; \bar{y}_1 < \bar{z}_0. \tag{2.80}$$

Now with this information we again consider the quantity of interest

$$\begin{aligned}
& C(m+1) - \frac{\alpha \bar{C}(m,\alpha)}{1-X_m} \\
= \ & C(m+1) - \alpha \sum_{n=0}^{m} C(n)\bar{y}_n \\
\geq \ & C(m+1) - \alpha \sum_{n=1}^{m} C(n)\bar{z}_{n-1} \\
= \ & \sum_{n=1}^{m} \alpha\{C(m+1) - C(n)\}\bar{z}_{n-1} \\
\geq \ & \sum_{n=1}^{m} \alpha\{C(m) - C(n-1)\}\bar{z}_{n-1} \\
= \ & C(m) - \alpha \sum_{n=1}^{m} C(n-1)\bar{z}_{n-1} \\
= \ & C(m) - \frac{\alpha \bar{C}(m-1,\alpha)}{1-X_{m-1}}.
\end{aligned}$$

In the above working we get to line 2 by using (2.78); we get to line 3 by using (2.80) and the fact that $C(0) = 0$; we get to line 4 since $\sum_{n=1}^{m} \alpha \bar{z}_{n-1} = 1$; we get to line 5 by using the convexity property of the cost function $C(.)$ and line 7 follows from (2.79). Therefore we have finally shown that (2.46) is increasing with m. So this together with the fact that (2.45) is also increasing with m, which we have previously shown proves that Lemma 1 is true and $\bar{W}_\alpha(m)$ is indeed increasing with m.

We now go on to prove Theorem 1, which we assumed to hold when making the argument used to find our conjectured index. Initially when trying to prove this we encountered a few difficulties, so in order to gain more of an insight into the problem we looked at what the solution would be if we were allowed to have a negative number of customers in the queue (for which we would pay zero costs). This helped us to find the required solution, and I have reported some numerics from this solution in Section 2.5.1, but I will not confuse the matter by including the solution to this virtual problem.

Theorem 1 is the key result needed to establish that the state m α-index is given by (2.44). This proof is long and utilises the methods of stochastic dynamic programming.

Theorem 1

(a) If $\bar{W}_\alpha(m) \leq W < \bar{W}_\alpha(m+1)$ then the policy which chooses the active action a in states $\{0, 1, 2, \ldots, m\}$ and the passive action b otherwise is optimal for our routing control problem with rejection charge W, $m \in \mathbb{Z}^+$.

(b) If $0 \leq W < \bar{W}_\alpha(0)$ then the policy which chooses the passive action in all states is optimal.

Proof - Theorem 1 part (a)

Given a value for the rejection charge W in the range $\left[\bar{W}_\alpha(m), \bar{W}_\alpha(m+1)\right)$, we must show that it is optimal to accept the arriving customers in states $\{0, 1, 2, \ldots, m\}$ and optimal to reject in all other states. By standard DP theory it is enough to show that $\bar{V}(., \alpha, W)$ satisfies the optimality equations (2.24), where $\bar{V}$ is the value function for the policy described in the statement of the theorem. In other words we must show that when V is replaced by $\bar{V}$ the first expression on the r.h.s. of (2.24) is the smaller of the two if we are in one of the states j, where $0 \leq j \leq m$, and that the second expression on the r.h.s. of (2.24) is the smaller if we are in one of the states j, where $j \geq m+1$. For $\boldsymbol{1 \leq j \leq m}$ we must show that

$$\begin{aligned} & \frac{C(j)}{\alpha+\lambda+\mu} + \frac{\lambda}{\alpha+\lambda+\mu}\bar{V}(j+1, \alpha, W) + \frac{\mu}{\alpha+\lambda+\mu}\bar{V}(j-1, \alpha, W) \\ \leq \quad & \frac{C(j)+\lambda W}{\alpha+\lambda+\mu} + \frac{\lambda}{\alpha+\lambda+\mu}\bar{V}(j, \alpha, W) + \frac{\mu}{\alpha+\lambda+\mu}\bar{V}(j-1, \alpha, W) \\ \implies \quad & \{\bar{V}(j+1, \alpha, W) - \bar{V}(j, \alpha, W)\} \leq W. \qquad (2.81) \end{aligned}$$

For the case where $j = 0$ we use the technique of uniformisation, as discussed on page 49 in the proof of Lemma 1. Hence for $j = 0 \leq m$ we have

$$
\begin{aligned}
& \frac{C(0)}{\alpha+\lambda+\mu} + \frac{\lambda}{\alpha+\lambda+\mu}\bar{V}(1,\alpha,W) + \frac{\mu}{\alpha+\lambda+\mu}\bar{V}(0,\alpha,W) \\
\leq \quad & \frac{C(0)+\lambda W}{\alpha+\lambda+\mu} + \frac{\lambda}{\alpha+\lambda+\mu}\bar{V}(0,\alpha,W) + \frac{\mu}{\alpha+\lambda+\mu}\bar{V}(0,\alpha,W) \\
\Longrightarrow \quad & \{\bar{V}(1,\alpha,W) - \bar{V}(0,\alpha,W)\} \leq W. \qquad (2.82)
\end{aligned}
$$

So we can see that (2.82) in fact holds for $0 \leq j \leq m$. We must also show that for $\boldsymbol{j \geq m+1}$ we have

$$
\begin{aligned}
& \frac{C(j)}{\alpha+\lambda+\mu} + \frac{\lambda}{\alpha+\lambda+\mu}\bar{V}(j+1,\alpha,W) + \frac{\mu}{\alpha+\lambda+\mu}\bar{V}(j-1,\alpha,W) \\
\geq \quad & \frac{C(j)+\lambda W}{\alpha+\lambda+\mu} + \frac{\lambda}{\alpha+\lambda+\mu}\bar{V}(j,\alpha,W) + \frac{\mu}{\alpha+\lambda+\mu}\bar{V}(j-1,\alpha,W) \\
\Longrightarrow \quad & \{\bar{V}(j+1,\alpha,W) - \bar{V}(j,\alpha,W)\} \geq W. \qquad (2.83)
\end{aligned}
$$

In order to demonstrate that (2.81), (2.82) and (2.83) hold we consider the following four cases in turn.

1. $j = m$
2. $j < m$
3. $j = m+1$
4. $j > m+1$

(1) $\boldsymbol{j = m}$

For this case we must show that (2.81) holds. Using the same method that we employed to find (2.35) we can see that

$$\bar{V}(m+1,\alpha,W) = \frac{C(m+1) + \lambda W + \mu\bar{C}(m,\alpha)}{\alpha+\mu-\mu X_m}. \qquad (2.84)$$

We also have using (2.31) that

$$\bar{V}(m,\alpha,W) = \frac{\bar{C}(m,\alpha) + X_m\{C(m+1)+\lambda W\}(\alpha+\mu)^{-1}}{1-\mu X_m(\alpha+\mu)^{-1}}.$$

Using (2.31), (2.84) and simplifying we can see that

$$
\begin{aligned}
& \{\bar{V}(m+1,\alpha,W) - \bar{V}(m,\alpha,W)\} \leq W \\
\Leftrightarrow \quad & \{C(m+1) + \lambda W\}(1 - X_m) \leq W\{\alpha + \mu - \mu X_m\} + \alpha \bar{C}(m,\alpha).
\end{aligned}
$$

Now using the identity (2.42) and rearranging leads us to

$$
\begin{aligned}
& C(m+1) - \frac{\alpha \bar{C}(m,\alpha)}{1 - X_m} \leq \lambda W \Big\{ \frac{1 - X_{m+1} - X_{m+1}(1 - X_m)}{(1 - X_m)X_{m+1}} \Big\} \\
\Leftrightarrow \quad & \frac{X_{m+1}}{\lambda} \Big\{ C(m+1) - \frac{\alpha \bar{C}(m,\alpha)}{1 - X_m} \Big\} \Big/ \Big\{ \frac{1 - X_{m+1}}{1 - X_m} - X_{m+1} \Big\} \leq W \\
\Leftrightarrow \quad & \bar{W}_\alpha(m) \leq W,
\end{aligned}
$$

where, we use the expression we found for the index in (2.44) to get to the last line. However we can see from the statement of Theorem 1 itself that we have $\bar{W}_\alpha(m) \leq W < \bar{W}_\alpha(m+1)$. So we have shown that the required condition holds when the system is in state m. We now move onto case 2.

(2) $j < m$

Firstly note that in this case we have $j < m$ and so under the policy described in the statement of the theorem we will initially start off in the active mode. Therefore using the definitions of $\bar{C}(.,\alpha)$ and $\bar{V}(j,\alpha,W)$ we have that

$$
\bar{V}(j,\alpha,W) = \bar{C}(j,\alpha) + X_j \bar{V}(j+1,\alpha,W), \qquad j \leq m-1. \tag{2.85}
$$

We prove this case by induction. To use a proof by induction we firstly need to prove the initial case, i.e. prove that (2.81) holds when $j = m-1$, we will then assume that this inequality holds for $j = r$ and prove it for $j = r-1$. So I shall first prove that (2.81) holds for the initial case of $j = m-1$. Using the expression (2.85) within (2.81) then formula (2.31) for $\bar{V}(m,\alpha,W)$ and rearranging, we can see that we need to show that

$$
\begin{aligned}
(1-X_{m-1})V(m,\alpha,W)-\bar{C}(m-1,\alpha) &\leq W \\
\Leftrightarrow (1-X_{m-1})\Big\{\frac{\bar{C}(m,\alpha)(\alpha+\mu)+X_mC(m+1)+\lambda X_m W}{\alpha+\mu-\mu X_m}\Big\}-\bar{C}(m-1,\alpha) &\leq W \\
\Leftrightarrow \frac{X_m(1-X_{m-1})}{\alpha+\mu-\mu X_m}\Big\{C(m+1)-\frac{\alpha\bar{C}(m,\alpha)}{1-X_m}\Big\} & \\
+\frac{1-X_{m-1}}{\alpha+\mu-\mu X_m}\Big\{\frac{\alpha X_m\bar{C}(m,\alpha)}{1-X_m}+\bar{C}(m,\alpha)(\alpha+\mu)\Big\}-\bar{C}(m-1,\alpha) & \\
\leq W\Big[1-\frac{X_m(1-X_m)\lambda}{\alpha+\mu+\mu X_m}\Big]. & \quad (2.86)
\end{aligned}
$$

However rearranging, simplifying and using the expression (2.40) that we found for $\bar{C}(n,\alpha)$ and then using a form of (2.42) we can see that

$$
\begin{aligned}
& \frac{1-X_{m-1}}{\alpha+\mu-\mu X_m}\Big\{\frac{\alpha X_m\bar{C}(m,\alpha)}{1-X_m}+\bar{C}(m,\alpha)(\alpha+\mu)\Big\}-\bar{C}(m-1,\alpha) \\
= \;& \frac{1-X_{m-1}}{1-X_{m+1}}\bar{C}(m,\alpha)-\bar{C}(m-1,\alpha) \\
\leq \;& \frac{1-X_{m-1}}{1-X_m}\bar{C}(m,\alpha)-\bar{C}(m-1,\alpha) \\
= \;& \frac{1-X_{m-1}}{\alpha+\mu-\mu X_{m-1}}\Big[C(m)+\mu\bar{C}(m-1,\alpha)\Big]-\bar{C}(m-1,\alpha) \\
= \;& \frac{1-X_{m-1}}{\alpha+\mu-\mu X_{m-1}}\Big[C(m)-\frac{\bar{C}(m-1,\alpha)}{1-X_{m-1}}\big\{-\mu(1-X_{m-1})+\alpha+\mu-\mu X_{m-1}\big\}\Big] \\
= \;& \frac{1-X_{m-1}}{\alpha+\mu-\mu X_{m-1}}\Big[C(m)-\frac{\alpha\bar{C}(m-1,\alpha)}{1-X_{m-1}}\Big]. \qquad (2.87)
\end{aligned}
$$

We can use the above expression, then use formula (2.44) (which we found for our index) to see that the left hand side of (2.86) is less than or equal to

$$
\begin{aligned}
& \frac{X_m(1-X_{m-1})}{\alpha+\mu-\mu X_m}\Big\{C(m+1)-\frac{\alpha\bar{C}(m,\alpha)}{1-X_m}\Big\} \\
& \qquad +\frac{1-X_{m-1}}{\alpha+\mu-\mu X_{m-1}}\Big[C(m)-\frac{\alpha\bar{C}(m-1,\alpha)}{1-X_{m-1}}\Big] \\
= \;& \frac{\lambda\bar{W}_\alpha(m)}{X_{m+1}}\frac{X_m(1-X_{m-1})}{\alpha+\mu-\mu X_m}\Big[\frac{1-X_{m+1}}{1-X_m}-X_{m+1}\Big] \\
& \qquad +\frac{\lambda\bar{W}_\alpha(m-1)}{X_m}\frac{1-X_{m-1}}{\alpha+\mu-\mu X_{m-1}}\Big[\frac{1-X_m}{1-X_{m-1}}-X_m\Big]. \qquad (2.88)
\end{aligned}
$$

Then from (2.42) we have that $(\alpha+\mu+\mu X_m)X_{m+1}=\lambda(1-X_{m+1})$. Using this then recalling that $\bar{W}_\alpha(m-1) \le \bar{W}_\alpha(m) \le W$ we can see that (2.88) is equal to

$$\begin{aligned}
& \bar{W}_\alpha(m)\frac{\lambda X_m(1-X_{m-1})}{\lambda(1-X_{m+1})}\Big[\frac{1-X_{m+1}}{1-X_m} - X_{m+1}\Big] \\
& \qquad\qquad + \bar{W}_\alpha(m-1)\frac{\lambda(1-X_{m-1})}{\lambda(1-X_m)}\Big[\frac{1-X_m}{1-X_{m-1}} - X_m\Big] \\
\le \; & W\Big[\frac{X_m(1-X_{m-1})}{1-X_m} - \frac{\lambda X_m X_{m+1}(1-X_{m-1})}{\lambda(1-X_{m+1})} + 1 - \frac{X_m(1-X_{m-1})}{1-X_m}\Big] \\
= \; & W\Big[1 - \frac{\lambda X_m(1-X_{m-1})}{\alpha+\mu+\mu X_m}\Big],
\end{aligned}$$

where we get this last line by using relation (2.42) to see that $X_{m+1}/\big(\lambda(1-X_{m+1})\big) = 1/(\alpha+\mu-\mu X_m)$. So we can therefore see that the inequality in (2.86) does indeed hold so we have therefore established our initial case. In other words we have shown that (2.81) holds when $j = m-1$. So we now assume that (2.81) holds for $j = r \le m-1$ and try to prove it for $j = r-1$, in which case we would have proven that (2.81) holds for all $j < m$ as required. So we assume that

$$\begin{aligned}
V(r+1,\alpha,W) - V(r,\alpha,W) &\le W, \quad \text{where } r \le m-1 \\
\Leftrightarrow V(r+1,\alpha,W) &\le \frac{W}{1-X_r} + \frac{\bar{C}(r,\alpha)}{1-X_r} \qquad (2.89)
\end{aligned}$$

which follows from relation (2.85). Using (2.85) we can see that we must show

$$\begin{aligned}
V(r,\alpha,W) - V(r-1,\alpha,W) &\le W \\
\Leftrightarrow (1-X_{r-1})\bar{C}(r,\alpha) + (1-X_{r-1})X_r V(r+1,\alpha,W) - \bar{C}(r-1,\alpha) &\le W. \qquad (2.90)
\end{aligned}$$

Now using the inductive hypothesis (2.89) we can see that (2.90) will be proved if we can show that

$$\begin{aligned}
& (1-X_{r-1})\bar{C}(r,\alpha) + (1-X_{r-1})X_r\Big[\frac{W}{1-X_r} + \tfrac{\bar{C}(r,\alpha)}{1-X_r}\Big] - \bar{C}(r-1,\alpha) \le W \\
\Leftrightarrow \; & \bar{C}(r,\alpha)(1-X_{r-1})\Big[1 + \frac{X_r}{1-X_r}\Big] - \bar{C}(r-1,\alpha) \le W\Big[1 - \frac{X_r(1-X_{r-1})}{1-X_r}\Big] \qquad (2.91)
\end{aligned}$$

Using (2.40) and (2.42) we can see that $\lambda\bar{C}(r,\alpha) = X_r(C(r) + \mu\bar{C}(r-1,\alpha))$ and hence the left-hand side of inequality (2.91) becomes

$$
\begin{aligned}
& \{C(r)+\mu\bar{C}(r-1,\alpha)\}\frac{X_r}{\lambda}\Big[\frac{1-X_{r-1}}{1-X_r}\Big]-\bar{C}(r-1,\alpha) \\
= & \frac{X_rC(r)(1-X_{r-1})}{\lambda(1-X_r)}+\bar{C}(r-1,\alpha)\Big[\frac{\mu X_r(1-X_r)-\lambda(1-X_r)}{\lambda(1-X_r)}\Big] \\
= & \frac{X_rC(r)(1-X_{r-1})}{\lambda(1-X_r)}+\bar{C}(r-1,\alpha)\Big[\frac{\lambda-(\alpha+\lambda)X_r-\lambda+\lambda X_r}{\lambda(1-X_r)}\Big] \\
= & \frac{X_r(1-X_{r-1})}{\lambda(1-X_r)}\Big[C(r)-\frac{\alpha\bar{C}(r-1,\alpha)}{1-X_{r-1}}\Big],
\end{aligned}
\tag{2.92}
$$

where we move from line 2 to line 3 above by using relation (2.42) to infer that $\mu X_r(1-X_{r-1})=\lambda-(\alpha+\lambda)X_r$. So using (2.92) we can now see that the inequality in (2.91) becomes

$$
\begin{aligned}
\frac{X_r}{\lambda}\Big[C(r)-\frac{\alpha\bar{C}(r-1,\alpha)}{1-X_{r-1}}\Big] &\leq W\Big[\frac{1-X_r}{1-X_{r-1}}-X_r\Big] \\
\Leftrightarrow \frac{X_r}{\lambda}\Big[C(r)-\frac{\alpha\bar{C}(r-1,\alpha)}{1-X_{r-1}}\Big]\Big/\Big[\frac{1-X_r}{1-X_{r-1}}-X_r\Big] &\leq W \\
\Leftrightarrow \bar{W}_\alpha(r-1) &\leq W.
\end{aligned}
$$

We know that this last line is true since we have that $r<m-1$ and that $\bar{W}_\alpha(m)<W$ (by hypothesis) and required condition (2.81) holds by Lemma 1, (i.e. that $\bar{W}_\alpha(.)$ is increasing). Therefore combining this with case 1 one can see that we have proved that (2.81) does hold for $0\leq j\leq m$ as required. We can now move onto case 3.

(3) $j=m+1$

Here we use $j=m+1$ and so from (2.83) we can see that we must show that

$$
\bar{V}(m+2,\alpha,W)-\bar{V}(m+1,\alpha,W)\geq W. \tag{2.93}
$$

If the system is in state $m+2$ then one can see that following the policy described in the statement of Theorem 1 will dictate that the passive action is taken initially. Hence $V(m+2,\alpha,W)$ will be made up of the total discounted cost until we enter state $m+1$ (via a single service completion) plus the total discounted cost from state $m+1$ onwards, discounted accordingly. So we have

$$\begin{aligned}\bar{V}(m+2,\alpha,W) &= \big(C(m+2)+\lambda W\big)E_X\Big[\int_0^X e^{-\alpha t}dt\Big]+\bar{V}(m+1,\alpha,W)E(e^{-\alpha X}),\\ &\qquad\qquad\text{where } X\sim\exp(\mu)\\ &= \frac{C(m+2)+\lambda W}{\alpha+\mu}+\bar{V}(m+1,\alpha,W)\Big(\frac{\mu}{\alpha+\mu}\Big). \qquad (2.94)\end{aligned}$$

Using relation (2.94) in the required inequality (2.93) and rearranging, we can see that we must show that

$$\begin{aligned}C(m+2)-\alpha\bar{V}(m+1,\alpha,W) &\geq W(\alpha+\mu-\lambda)\\ \Leftrightarrow C(m+2)-\alpha\Big\{\frac{C(m+1)+\lambda W+\mu\bar{C}(m,\alpha)}{\alpha+\mu-\mu X_m}\Big\} &\geq W(\alpha+\mu-\lambda)\\ \Leftrightarrow C(m+2)-\alpha\Big\{\frac{C(m+1)+\lambda W+\bar{C}(m+1,\alpha)(\alpha+\lambda+\mu-\mu X_m)-C(m+1)}{\alpha+\mu-\mu X_m}\Big\}&\\ &\geq W(\alpha+\mu-\lambda)\\ \Leftrightarrow C(m+2)-\alpha\Big\{\frac{\lambda W+\bar{C}(m+1,\alpha)(\lambda/X_{m+1})}{\lambda(1-X_{m+1})/X_{m+1})}\Big\} &\geq W(\alpha+\mu-\lambda)\qquad (2.95)\end{aligned}$$

In the above calculations we used relation (2.84) to get to the second line, relation (2.40) to get to the third line, cancellation and the relation (2.42) to infer that $\alpha+\lambda+\mu-\mu X_m=\lambda/X_{m+1}$ and that $\alpha+\mu-\mu X_m=\lambda(1-X_{m+1})/X_{m+1}$ to get to the fourth line. Rearranging (2.95) leads us to

$$C(m+2)-\frac{\alpha\bar{C}(m+1,\alpha)}{1-X_{m+1}}\geq W\Big[\frac{(\alpha+\mu-\lambda)(1-X_{m+1})+\alpha X_{m+1}}{1-X_{m+1}}\Big]. \qquad (2.96)$$

Now if we just concentrate on the right-hand side of (2.96) for a moment, we can see that we can simplify and use the relation (2.42), to infer that $\alpha+\mu-\mu X_{m+1}=\lambda(1-X_{m+2})/X_{m+2}$, as follows:

$$\begin{aligned}W\Big[\frac{(\alpha+\mu-\lambda)(1-X_{m+1})+\alpha X_{m+1}}{(1-X_{m+1})}\Big] &= W\Big[\frac{\alpha+\mu-\mu X_{m+1}}{1-X_{m+1}}-\frac{\lambda(1-X_{m+1})}{1-X_{m+1}}\Big]\\ &= W\Big[\frac{\lambda(1-X_{m+2})}{X_{m+2}}\frac{1}{1-X_{m+1}}-\lambda\Big]\\ &= \frac{\lambda W}{X_{m+2}}\Big[\frac{1-X_{m+2}}{1-X_{m+1}}-X_{m+2}\Big]. \qquad (2.97)\end{aligned}$$

Using relation (2.97) within (2.96) we can see that in order to prove that the inequality (2.83) holds for $j = m+1$ we need to show

$$\begin{aligned} C(m+2) - \frac{\alpha\bar{C}(m+1,\alpha)}{1-X_{m+1}} &\geq \frac{\lambda W}{X_{m+2}}\Big[\frac{1-X_{m+2}}{1-X_{m+1}} - X_{m+2}\Big] \\ \Leftrightarrow \frac{X_{m+2}}{\lambda}\Big\{C(m+2) - \frac{\alpha\bar{C}(m+1,\alpha)}{1-X_{m+1}}\Big\}\Big/\Big\{\frac{1-X_{m+2}}{1-X_{m+1}} - X_{m+2}\Big\} &\geq W \\ \Leftrightarrow \bar{W}_\alpha(m+1) &\geq W \end{aligned}$$

We have from the hypothesis in the theorem that we do have $\bar{W}_\alpha(m+1) \geq W$, hence can see that the above does indeed prove that inequality (2.83) holds for $j = m+1$. So we now move on to the final case.

(4) $j \geq m+2$

Here we have $j \geq m+2$ and we must show that inequality (2.83) holds. Note that since in this case we have $j \geq m+2$ then according to the policy in Theorem 1 we will initially take the passive action. Therefore using the definition of $\bar{V}(j,\alpha,W)$ and the fact that in the passive mode we have service only (which follows the $\exp(\mu)$ distribution), we have that

$$\bar{V}(j,\alpha,W) = \frac{C(j)+\lambda W}{\alpha+\mu} + \frac{\mu\bar{V}(j-1,\alpha,W)}{\alpha+\mu}. \tag{2.98}$$

We prove this case by induction also. Here we use $j = m+1$ as our initial situation. However we have already established (2.83) for this in case 3. So we now assume that (2.83) holds for $j = k$ and infer it for $j = k+1$, i.e. we have our inductive hypothesis

$$\bar{V}(k+1,\alpha,W) - \bar{V}(k,\alpha,W) \geq W, \tag{2.99}$$

and we wish to infer that

$$\bar{V}(k+2,\alpha,W) - \bar{V}(k+1,\alpha,W) \geq W. \tag{2.100}$$

Using the relation (2.98) for $\bar{V}(k+2,\alpha,W)$ and $\bar{V}(k+1,\alpha,W)$ then simplifying, we can see that (2.100) is equivalent to

$$C(k+2) - C(k+1) + \mu(\bar{V}(k+1,\alpha,W) - \bar{V}(k,\alpha,W)) \;\geq\; W(\alpha+\mu).$$

From (2.99) it will be enough to show that

$$C(k+2) - C(k+1) \geq \alpha W. \tag{2.101}$$

So, in order to prove that (2.83) holds for $j \geq m+1$ it is enough to show that the inequality in (2.101) holds. To do this we consider the relation that we have already proved in case 3. From (2.93) we have that

$$\begin{aligned} \bar{V}(m+2,\alpha,W) - \bar{V}(m+1,\alpha,W) &\geq W \\ \Leftrightarrow C(m+2) - C(m+1) + \mu(\bar{V}(m+1,\alpha,W) - \bar{V}(m,\alpha,W)) &\geq W(\alpha+\mu) \end{aligned} \tag{2.102}$$

We have also shown that $\bar{V}(m+1,\alpha,W) - \bar{V}(m,\alpha,W) \leq W$ (in case 1), so using this we can see that (2.102) implies that

$$C(m+2) - C(m+1) \geq \alpha W. \tag{2.103}$$

Now since $k > m$ the convex nature of the cost curve $C(.)$, (2.103) implies that

$$C(k+2) - C(k+1) \geq \alpha W.$$

Hence we have shown the inequality in (2.101) does indeed hold and so we have shown that (2.83) does hold for $j \geq m+1$ as required.

Proof - Theorem 1 part (b)

Given a value for the rejection charge $W < \bar{W}_\alpha(0)$, we must show that it is optimal to not accept the arriving customers in any state. Again by standard DP theory it is enough to show that $\bar{V}(.,\alpha,W)$ satisfies the optimality equations (2.24), where $\bar{V}$ is the value function for the policy described in the statement of the theorem. In other words we must show that when V is replaced by $\bar{V}$ the second expression in the r.h.s. of (2.24) is the smaller of the two if we are in any of the possible states $j \geq 0$. Following a similar progression to that in the proof of part (a) we find that for $j \geq 0$ we must show

$$\{\bar{V}(j+1,\alpha,W) - \bar{V}(j,\alpha,W)\} \geq W. \tag{2.104}$$

To prove this we consider the following two cases in turn,

1. $j = 0$

2. $j \geq 1$

(1) j = 0

Following a similar derivation as for (2.94) we find that

$$\bar{V}(1, \alpha, W) = \frac{C(1) + \lambda W}{\alpha + \mu} + \bar{V}(0, \alpha, W)\Big(\frac{\mu}{\alpha + \mu}\Big). \tag{2.105}$$

Following the policy in part (b) of Theorem 1 we always reject the arriving customers. So when we are in state 0, we will always remain in this state and therefore incur costs at a rate $C(0) + \lambda W$. Hence

$$\begin{aligned} \bar{V}(0, \alpha, W) &= \int_0^\infty (C(0) + \lambda W)e^{-\alpha t} dt \\ &= \frac{C(0) + \lambda W}{\alpha}. \end{aligned} \tag{2.106}$$

So using (2.105) and then (2.106) in the required inequality (2.104), we can see that we must show that

$$\begin{aligned} C(1) - \alpha\bar{V}(0, \alpha, W) &\geq W(\alpha + \mu - \lambda) \\ \implies C(1) - C(0) - \lambda W &\geq W(\alpha + \mu - \lambda) \\ \implies C(1) &\geq W(\alpha + \mu), \end{aligned} \tag{2.107}$$

since we have that $C(0) = 0$. Using expressions (2.38), (2.41) and (2.44) we can see that

$$\begin{aligned} \bar{W}_\alpha(0) &= \frac{X_1}{\lambda}\Big(C(1) - \frac{\alpha\bar{C}(0, \alpha)}{1 - X_0}\Big)\Big/\Big(\frac{1 - X_1}{1 - X_0} - X_1\Big) \\ &= \frac{X_1 C(1)}{\lambda}\Big/\Big(\frac{(\alpha + \lambda)(1 - X_1)}{\alpha} - X_1\Big) \\ &= \frac{\alpha X_1 C(1)/\lambda}{(\alpha + \lambda)(1 - X_1) - \alpha X_1}. \end{aligned} \tag{2.108}$$

Now using (2.41) and (2.42) we can easily show that

$$X_1 = \frac{\lambda(\alpha+\lambda)}{(\alpha+\lambda+\mu)(\alpha+\lambda)-\mu\lambda}. \tag{2.109}$$

Using this expression for X_1 in (2.108) we see that,

$$\begin{aligned} \bar{W}_\alpha(0) &= \frac{\alpha(\alpha+\lambda)C(1)}{(\alpha+\lambda)^2(\alpha+\mu)-\mu\lambda-\alpha\lambda(\alpha+\lambda)} \\ &= \frac{C(1)}{(\alpha+\mu)}. \end{aligned} \tag{2.110}$$

So using (2.110) we can see that the required inequality (2.107) is equivalent to

$$\bar{W}_\alpha(0) \geq W, \tag{2.111}$$

which is given in Theorem 1 part (b), hence we can see that we have now proved part (b) of Theorem 1 when $j = 0$.

(2) $\mathbf{j \geq 1}$

Here we have $j \geq 1$ and we must show that inequality (2.104) holds. In this situation the policy in part (b) of the Theorem 1 dictates that we take the passive action. So using the definition of $\bar{V}(j, \alpha, W)$ and the fact what we will have only service completions (and no arrivals), we can see that

$$\bar{V}(j, \alpha, W) = \frac{C(j)+\lambda W}{\alpha+\mu} + \frac{\mu\bar{V}(j-1, \alpha, W)}{\alpha+\mu}. \tag{2.112}$$

We prove this case by induction. Here we use $j = 0$ as our initial situation. However we have already established (2.104) for this in the previous case. So we now assume that (2.104) holds for $j = k$ and infer it for $j = k + 1$, i.e. we have our inductive hypothesis

$$\bar{V}(k+1, \alpha, W) - \bar{V}(k, \alpha, W) \geq W, \tag{2.113}$$

and we wish to infer that

$$\bar{V}(k+2, \alpha, W) - \bar{V}(k+1, \alpha, W) \geq W. \tag{2.114}$$

Using the relation (2.112) for $\bar{V}(k+2,\alpha,W)$ and $\bar{V}(k+1,\alpha,W)$ then simplifying, we can see that (2.114) is equivalent to

$$C(k+2) - C(k+1) + \mu(\bar{V}(k+1,\alpha,W) - \bar{V}(k,\alpha,W)) \geq W(\alpha+\mu).$$

From (2.113) it will be enough to show that

$$C(k+2) - C(k+1) \geq \alpha W, \tag{2.115}$$

in order to prove that (2.104) holds for $j \geq 1$. To do this we notice that from part (b) of Theorem 1 we have

$$\begin{aligned} \bar{W}_\alpha(0) &> W \\ \Longrightarrow C(1) - C(0) &> W(\alpha+\mu). \end{aligned} \tag{2.116}$$

Since $\mu \geq 0$, $k > 0$ and we know that the cost curve $C(.)$ is convex, inequality (2.116) implies that

$$C(k+2) - C(k+1) > \alpha W.$$

Hence we have shown the inequality in (2.115) does indeed hold and so we have shown that (2.104) does hold for $j \geq 1$ as required.

Now since we have proved all possible cases we have completed the proof of Theorem 1.

By studying the calculations in the proof of Theorem 1 carefully we can see that when $\bar{W}_\alpha(m) < W < \bar{W}_\alpha(m+1)$ the policy described in the theorem is strictly optimal. Suppose now that $W = \bar{W}_\alpha(m)$. We can see from Theorem 1 that for this W-value, the policy which chooses the active action in states $\{0,1,\ldots,m\}$ and the passive action otherwise is optimal, we call this policy u_1. Recall that u_2 chooses the active action in states $\{0,1,\ldots,m-1\}$ and the passive action otherwise. From (2.37) and following we have that

$$V_{u_1}\{m,\alpha,\bar{W}_\alpha(m)\} = V_{u_2}\{m,\alpha,\bar{W}_\alpha(m)\}. \tag{2.117}$$

From this and the fact that u_1 and u_2 take the same actions in all states other than m it follows easily from (2.117) that

$$V_{u_1}\{n,\alpha,\bar{W}_\alpha(m)\} = V_{u_2}\{n,\alpha,\bar{W}_\alpha(m)\},\ n \in \mathbb{N},$$

and hence, policy u_2 must also be optimal when $W = \bar{W}_\alpha(m)$. It follows that when $W = \bar{W}_\alpha(m)$ both actions are optimal in state m. The following result is now immediate.

Theorem 2

The customer class is α-indexable with the Whittle α-index $W_\alpha(m) = \bar{W}_\alpha(m)$, $m \in \mathbb{N}$.

Proof

By Theorem 1 and the definitions of the quantities involved we have that

$$\Pi_\alpha(W) = \{0,1,\dots,m\},\ \bar{W}_\alpha(m) \leq W < \bar{W}_\alpha(m+1),\ m \in \mathbb{N}, \qquad (2.118)$$

and the requirements of Definition 1 are met, with α-indexability an immediate consequence. That $\bar{W}_\alpha(m)$ is the Whittle α-index for state m follows from (2.118) and Definition 2.

Comments

1. We can now see that the Whittle α-index is indeed given by expression (2.44). Also note that (2.42) and (2.40) are strongly suggestive of the following computational schemes for the computation of X_m and $\bar{C}(m,\alpha)$.

- Use $X^R_.$ to denote the R^{th} iterate of the target function $X_.$, take $X^1_m = 0$, $m \in \mathbb{Z}^+$, then
$$\bar{X}^{R+1}_m = \frac{\lambda}{\alpha + \lambda + \mu - \mu X^R_{m-1}}.$$
- Use $\bar{C}^R(.,\alpha)$ to denote the R^{th} iterate of the target function $\bar{C}(.,\alpha)$. Take

$\bar{C}^1(m,\alpha) = 0$, $m \in \mathbb{Z}^+$, then

$$\bar{C}^{R+1}(m,\alpha) = \frac{\mu \bar{C}'^R(m-1,\alpha) + C(m)}{\alpha + \lambda + \mu - \mu X_{m-1}}.$$

2. We will now substantiate the claims made for the Langrangian relaxation in Section 2.2 in the discussion preceding Definition 1. We consider class k (server k) and its associated routing control problem (k, α, W). Use $\{W^r_{k,\alpha};\, r = 0, 1, \ldots, R_k\}$ for the set of *distinct* index values for class k, numbered in ascending order. So note that we have $R_k + 1$ distinct index values, which may be infinite. So we have

$$W^0_{k,\alpha} < W^1_{k,\alpha} < W^2_{k,\alpha} < \ldots < W^{R_k}_{k,\alpha}$$

and,

$$\{W^r_{k,\alpha}; r = 0, 1, 2, \ldots, R_k\} = \{W_{k,\alpha}(n); n \in \mathbb{N}\}.$$

If $W \notin \{W^r_{k,\alpha}; r = 0, 1, 2, \ldots, R_k\}$, we use $u_k(W)$ to denote the unique optimal policy for the problem (k, α, W) as given by Theorem 1. If $W = W^r_{k,\alpha}$ for some r then we use $u_k(W)$ to denote the optimal policy which chooses the passive action in all states for which both actions are optimal. Using the notation of Section 2.2 we write

$$x^{u_k(W)}_{k,n}(m_k) = E_{u_k(W)}\Big[\sum_{i=1}^{\infty} e^{-\alpha t_i} I_{k,t_i,n} | N_k(0) = m_k\Big]$$

for the first of the associated performance measures. Recall that we have

$$I_{k,t_i,n} = \begin{cases} 1 & \text{if, when the } i^{\text{th}} \text{ customer arrives, we have } n \text{ class } k \text{ customers present} \\ & \text{and we choose not to admit her to station } k, \\ 0 & \text{otherwise.} \end{cases}$$

So we have that

$$\sum_{n \in \mathbb{N}} x^{u_k(W)}_{k,n}(m_k) = E_{u_k(W)}\Big[\sum_{i=1}^{\infty} e^{-\alpha t_i} I_{k,t_i} | N_k(0) = m_k\Big],$$

where we now have that

$$I_{k,t_i} = \begin{cases} 1 & \text{if, we do not admit the } i^{\text{th}} \text{ arriving customer to station } k, \\ 0 & \text{otherwise.} \end{cases}$$

From the characterization of $u_k(W)$ in Theorem 1, it easily follows that for any choice of m_k and r, $0 \leq r \leq R_k - 1$,

$$\sum_{n \in \mathbb{N}} x_{k,n}^{u_k(W)}(m_k) \tag{2.119}$$

is constant for $W \in (W_{k,\alpha}^r, W_{k,\alpha}^{r+1})$ since in this range $u_k(W)$ does not change. Finally, it is straightforward to show that

$$\sum_{n \in \mathbb{N}} x_{k,n}^{u_k(W)}(m_k) \to 0, \ W \to \infty.$$

and hence

$$\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{u_k(W)} m_k \to 0, \quad W \to \infty.$$

To summarise, the quantity in (2.119) when regarded as a function of W is piecewise constant, decreasing with jump discontinuities at distinct index values and tends to 0 as W approaches infinity. These characteristics are inherited in the obvious way by the aggregated quantity

$$\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{u_k(W)}(m_k) \equiv \sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{u_k(W)}(\mathbf{m}),$$

which is the appropriate performance measure for an optimal policy $\mathbf{u}(W)$ for the K-class stochastic optimisation problem in (2.16) obtained by independent operation of $u_k(W)$ for each class k. Further we can see that if $W = 0 \leq \bar{W}_{k,\alpha}(0)$, $1 \leq k \leq K$, (i.e. the charge for rejection is 0), $u_k(W)$ takes the passive action at all decision epochs, hence

$$\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{u_k(0)}(m_k) = \frac{\lambda K}{\alpha} > \frac{\lambda(K-1)}{\alpha}.$$

So we can see that for each decision epoch t we should take

$$W(\mathbf{m}, \alpha) = \inf\{W; \sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{u_k(W)}(\mathbf{m}) > \frac{\lambda(K-1)}{\alpha}\}. \tag{2.120}$$

Hence the policy $\mathbf{u}\{W(\mathbf{m}, \alpha)\}$ is optimal for the Lagrangian relaxation in (2.16) with $W = W(\mathbf{m}, \alpha)$, satisfies the constraint in (2.15)and hence solves Whittle's relaxation.

3. Following Theorem 2 and the discussion on page 29, an index policy for the K-class problem with discounted costs of Section 2.2 is constructed by computing the index function $W_{k,\alpha}(.)$ for each server k from an appropriate form of (2.44). At each epoch t, the policy will send the arriving customer to the server with the minimal index $W_{k,\alpha}\{N_k(t)\}$.

2.4 The Undiscounted Problem

We now look at the undiscounted problem. We could find an undiscounted Whittle index by one of two possible methods. We could reformulate the problem from scratch in an undiscounted manner and follow a similar method as for the discounted problem above. Or we could use the method documented here, where we start with the discounted index and allow α to tend to 0 to give us the undiscounted form of the index. We have actually used both these methods to find the index and the result (as we would expect) is the same. For this undiscounted problem we also compute another index, called the policy improvement index, for comparison to the Whittle index.

2.4.1 The Undiscounted Whittle index

So we now look at the undiscounted problem given by equation (2.6). By use of the information we have gained about the discounted problem we find an index policy for the undiscounted problem. We again drop the class identifier k and observe that we can now develop a suitable Whittle index $W : \mathbb{N} \rightarrow \mathbb{R}^+$ for the average cost problem from the limit of the corresponding α-index

$$W(m) = \lim_{\alpha \to 0} W_\alpha(m) = \lim_{\alpha \to 0} \bar{W}_\alpha(m), \; m \in \mathbb{N}, \tag{2.121}$$

as in Definition 3. Utilising (2.121) within (2.44) we obtain the following result.

Theorem 3 (The Whittle index for average costs)

The Whittle index for the average cost admission control problem is given by

$$
\begin{aligned}
W(m) &= \frac{1}{\mu}\{C(m+1)-C(0)\}+\frac{\lambda}{\mu^2}\{C(m+1)-C(1)\}+\dots \\
&+ \frac{\lambda^{m-1}}{\mu^m}\{C(m+1)-C(m-1)\}+\frac{\lambda^m}{\mu^{m+1}}\{C(m+1)-C(m)\} \qquad (2.122)
\end{aligned}
$$

Proof

Firstly note that here we use the fact that all moments of T_n are finite - which is easy to show. Notice that

$$X_n = E(e^{-\alpha T_n}) = E\big(1 - T_n\alpha\big) + 0(\alpha^2). \qquad (2.123)$$

So using (2.123) in (2.44) we have that

$$\bar{W}_\alpha(m) = \frac{X_{m+1}}{\lambda}\Big\{C(m+1) - \frac{\bar{C}(m,\alpha)}{E(T_m)}\Big\}\Big/\Big\{\frac{E(T_{m+1})}{E(T_m)} - X_{m+1}\Big\} + O(\alpha). \qquad (2.124)$$

Note that T_m is the time when the system firsts enters state $m+1$, we can see from (2.27) that the discounted cost from state m to state $m+1$ is

$$
\begin{aligned}
\bar{C}(m,\alpha) &= E\Big[\int_0^{T_m} C(N(t))e^{-\alpha t}dt|N(0)=m\Big], \\
&= E\Big[\int_0^{T_m} C(N(t))dt|N(0)=m\Big] + O(\alpha).
\end{aligned}
$$

So when we allow α to tends towards 0, we can see that

$$\bar{C}(m,\alpha) \to \bar{C}(m) \text{ as } \alpha \to 0.$$

where,

$$\bar{C}(m) = \int_0^{T_m} C(N(t))dt|N(0) = m.$$

Therefore we can see that when $\alpha \to 0$ (2.124) becomes

$$W(m) = \frac{1}{\lambda}\Big\{C(m+1) - \frac{\bar{C}(m)}{E(T_m)}\Big\}\Big/\Big\{\frac{E(T_{m+1})}{E(T_m)} - 1\Big\}. \qquad (2.125)$$

We will now consider the quantity $E(T_m)$. We can find a relation between these quantities if we condition on the first event after 0 given that m is the initial state. Hence we have, for $m \geq 1$, that

$$\begin{aligned} E(T_m) &= \frac{1}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu} \times 0 + \frac{\mu}{\lambda+\mu}\{E(T_{m-1} + E(T_m))\} \\ \Rightarrow \lambda E(T_m) &= 1 + \mu E(T_{m-1}), \end{aligned}$$

since the first event must either be a service completion or an arrival hence the time until the first event $\sim \exp(\lambda+\mu)$. Also note that in state 0 we can only have customer arrivals, hence

$$E(T_0) = \frac{1}{\lambda}.$$

Now using these equations recursively we can see that

$$E(T_m) = \frac{1}{\lambda} + \frac{\mu}{\lambda^2} + \ldots + \frac{\mu^m}{\lambda^{m+1}}, \tag{2.126}$$

and also therefore that,

$$E(T_m) - E(T_{m-1}) = \frac{\mu^m}{\lambda^{m+1}}. \tag{2.127}$$

We also consider the variable $\bar{C}(m)$, which is the expected cost incurred up to T_m. We again condition on the first event to find that for $m \geq 1$, we have

$$\begin{aligned} \bar{C}(m) &= \frac{C(m)}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu} \times 0 + \frac{\mu}{\lambda+\mu}\{\bar{C}(m-1) + \bar{C}(m)\} \\ \Rightarrow \lambda\bar{C}(m) &= C(m) + \mu\bar{C}(m-1). \end{aligned} \tag{2.128}$$

Again note the slightly different form in state 0,

$$\bar{C}(0) = \frac{C(0)}{\lambda} = 0,$$

since we know that $C(0) = 0$. Now using these equations recursively we find that,

$$\bar{C}(m) = \frac{C(m)}{\lambda} + \frac{\mu C(m-1)}{\lambda^2} + \ldots + \frac{\mu^{m-1}C(1)}{\lambda^m}. \tag{2.129}$$

From (2.127) we can see that $E(T_{m+1})/E(T_m) = \left(\mu^{m+1}/E(T_m)\lambda^{m+2}\right) + 1$, using this we can see from expression (2.125) that,

$$\begin{aligned}\bar{W}(m) &= \frac{\lambda^{m+1}}{\mu^{m+1}}\{C(m+1)E(T_m) - \bar{C}(m)\} \\ &= C(m+1)\Big\{\frac{1}{\mu} + \frac{\lambda}{\mu^2} + \ldots + \frac{\lambda^m}{\mu^{m+1}}\Big\} \\ &\quad -C(m)\frac{\lambda^m}{\mu^{m+1}} - C(m-1)\frac{\lambda^{m-1}}{\mu^m} - \ldots - C(1)\frac{\lambda}{\mu^2}. \end{aligned} \quad (2.130)$$

Note that we get the second line of the above by use of equations (2.126) and (2.129). Now since $C(0) = 0$ is is easy to see that (2.130) is equivalent to the expression for $\bar{W}(m)$ in Theorem 3, as required. We now move on to calculating another index policy for this system, for comparison.

2.4.2 The Undiscounted policy improvement index

Note that in this section we maintain the system setup and notation previously established, but may add some extra structure and notation where required. The approach to index development described here builds from the best static policy for the system. A static policy is one whose application does not change over time (or with the system state). To find an optimal static policy we initially consider the whole system. For illustrative purposes, we shall consider a system with 2 service stations present. The static policy specifies a proportion of the arriving customers to be sent to each station. More specifically, each arriving customer is sent to server 1 with some specified probability p_1 and to server 2 with probability $p_2 = 1 - p_1$.

The Optimal Static Policy

A two-server system can be represented by the the diagram shown in Figure 2.4. In Figure 2.4 λ is the system arrival rate, and μ_i is the service rate of queue i, $i = 1, 2$.

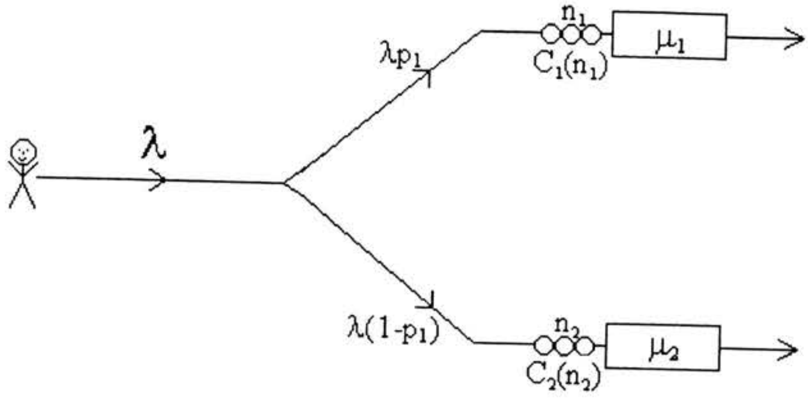

Figure 2.4: Two-server, static policy, example.

Also on this diagram p_1 is the proportion of customers to be sent to queue 1. Note that we will now require that $\mu_1 > \lambda p_1$ and $\mu_2 > \lambda(1-p_1)$ for stability. The optimal static policy is the one whose parameter p_1 minimises the average holding cost rate of the system. It can be seen that the average cost rate for for the system will take the form

$$\sum_{n\geq 0} C_1(n)\hat{p}_{1,n} + \sum_{n\geq 0} C_2(n)\hat{p}_{2,n}, \tag{2.131}$$

where $C_i(n)$ is the cost rate for queue i in state n, and $\hat{p}_{i,n}$ is the probability that queue i is in state n under the static policy. Assuming that our stability requirements are met, we know from standard M/M/1 theory that

$$\hat{p}_{i,n} = \Big(\frac{\lambda p_i}{\mu_i}\Big)^n \Big(1 - \frac{\lambda p_i}{\mu_i}\Big), \quad n \in \mathbb{N}, \quad i = 1, 2. \tag{2.132}$$

Therefore using (2.132) within (2.131) we can see that the expected cost rate for the system is

$$\sum_{n\geq 0} C_1(n)\Big(\frac{\lambda p_1}{\mu_1}\Big)^n\Big(1 - \frac{\lambda p_1}{\mu_1}\Big) + \sum_{n\geq 0} C_2(n)\Big(\frac{\lambda p_2}{\mu_2}\Big)^n\Big(1 - \frac{\lambda p_2}{\mu_2}\Big). \tag{2.133}$$

So the optimal static policy is found by selecting p_1 to minimise (2.133) and to meet our stability requirements. We will label the optimal p_1 by p_1^*.

Finding the policy improvement index

We now develop a dynamic routing policy by imposing a single DP policy improvement step on the optimal static policy. To help us make this decisions under this policy, assume that we have an arriving customer. Now consider for each i the difference between

- the total cost to infinity of sending this customer to queue i and then following the optimal static policy, and

- the total cost to infinity of rejecting this customer from queue i and then following the optimal static policy, $i = 1, 2$.

Note that while each of the above quantities is infinite, their difference (suitably defined) is finite. This fact relates to the theory of relative costs for undiscounted Markov Decision Processes. See Tijms (1994). We calculate this difference for each of our queues. It follows from a simple DP-type argument that, among policies which make all subsequent decisions according to the optimal static policy, the best current decision is to send the arriving customer to the queue where this difference is the smallest. Hence, for each station we require, for each n the cost difference between following the optimal static policy from initial states $n+1$ and from n. We define our policy improvement index for state n to be this difference. We now recover this difference in closed form. To help us in this task we introduce the following notation:

$K_i(n) =$ the expected holding cost incurred under implementation of the optimal static policy until we empty queue i for the first time, when starting with n customers at time zero;

and

$T_i(n) =$ the expected time under implementation of the optimal static policy until we empty queue i for the first time, when starting with n customers at time zero.

Also to help us gain further understanding we consider the state transition diagrams shown in figures 2.5 and 2.6. So if we now consider some large time T we

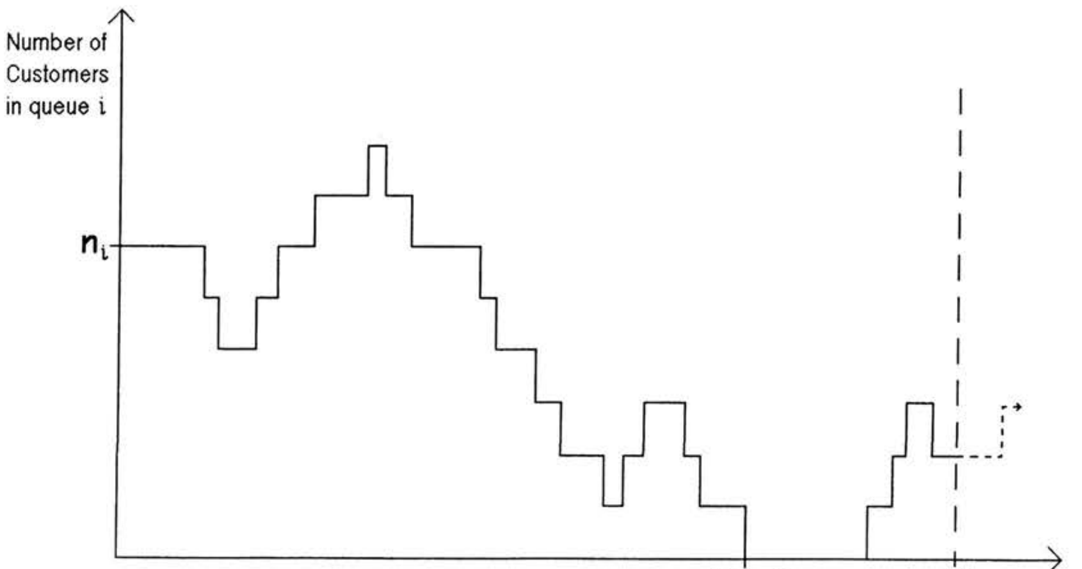

Figure 2.5: Possible state transition diagram from state n_i down to state 0.

can see that our policy improvement index (the difference defined on page 78) will

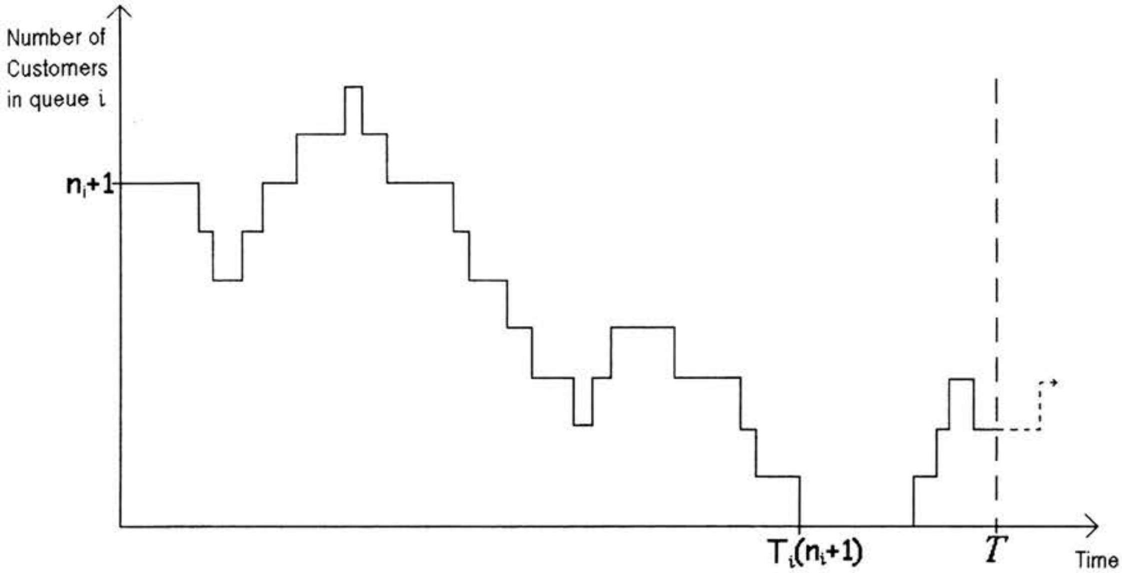

Figure 2.6: Possible state transition diagram from state $n_i + 1$ down to state 0.

take the approximate form

$$\begin{aligned} PI_i(n) &\cong K_i(n+1) + [T - T_i(n+1)]C_i^* - \big(K_i(n) + [T - T_i(n)]C_i^*\big) \\ &= K_i(n+1) - K_i(n) - [T_i(n+1) - T_i(n)]C_i^*. \end{aligned} \quad (2.134)$$

Note that in (2.134) we have used C_i^* to denote the average queue i cost rate when following the optimal static policy. In fact, the theory of Markov Decision Processes indicates that the expression in (2.134) is exactly the index we require. See Tijms (1994). To use (2.134) we need to be able to calculate the terms $K_i(.)$ and $T_i(.)$. To find an expression for $T_i(.)$, we condition upon the first event after zero for queue i to obtain for $n \geq 0$ that

$$\begin{aligned} T_i(n+1) &= \frac{1}{\lambda p_i^* + \mu_i} + \frac{\lambda p_i^*}{\lambda p_i^* + \mu_i} T_i(n+2) + \frac{\mu_i}{\lambda p_i^* + \mu_i} T_i(n) \\ \Rightarrow \mu_i\{T_i(n+1) - T_i(n)\} &= 1 + \lambda p_i^*\{T_i(n+2) - T_i n + 1\}. \end{aligned} \quad (2.135)$$

We now introduce

$$\delta_i(n) = T_i(n+1) - T_i(n).$$

Hence we can see from (2.135) that

$$\begin{aligned}
\delta_i(n) &= \frac{1}{\mu_i} + \frac{\lambda p_i^*}{\mu_i}\delta_i(n+1) \\
&= \frac{1}{\mu_i} + \frac{\lambda p_i^*}{\mu_i}\Big\{\frac{1}{\mu_i} + \frac{\lambda p_i^*}{\mu_i}\delta_i(n+2)\Big\} \\
&= \frac{1}{\mu_i} + \frac{\lambda p_i^*}{\mu_i}\Big\{\frac{1}{\mu_i} + \Big(\frac{\lambda p_i^*}{\mu_i}\Big)\frac{1}{\mu_i} + \Big(\frac{\lambda p_i^*}{\mu_i}\Big)^2\frac{1}{\mu_i} + \Big(\frac{\lambda p_i^*}{\mu_i}\Big)^3\frac{1}{\mu_i} + \dots\Big\} \\
&= \frac{1}{\mu_i}\frac{\lambda p_i^*}{\mu_i}\Big\{\frac{\mu_i^{-1}}{1-\lambda p_i^*\mu_i^{-1}}\Big\} \\
&= \frac{1}{\mu_i - \lambda p_i^*}. \qquad (2.136)
\end{aligned}$$

This calculation may be confirmed by standard queueing theory. The expression in (2.136) is the expected busy period for an M/M/1 queue with arrival rate λp_i^* and service rate μ_i. To find an expression for $K_i(.)$, we similarly condition upon the first event after 0. Hence for $n \geq 0$, we have

$$\begin{aligned}
K_i(n+1) &= \frac{C_i(n+1)}{\lambda p_i^* + \mu_i} + \frac{\lambda p_i^*}{\lambda p_i^* + \mu_i}K_i(n+2) + \frac{\mu}{\lambda p_i^* + \mu_i}K_i(n) \\
\Rightarrow \mu_i\{K_i(n+1) - K_i(n)\} &= C_i(n+1) + \lambda p_i^*\{K_i(n+2) - K_in+1\}. \qquad (2.137)
\end{aligned}$$

We now define

$$\hat{\delta}_i(n) = K_i(n+1) - K_i(n).$$

Hence we can see from (2.137) that

$$\begin{aligned}
\hat{\delta}_i(n) &= \frac{C_i(n+1)}{\mu_i} + \frac{\lambda p_i^*}{\mu_i}\hat{\delta}_i(n+1) \\
&= \frac{C_i(n+1)}{\mu_i} + \frac{\lambda p_i^*}{\mu_i}\frac{C_i(n+2)}{\mu_i} + \Big(\frac{\lambda p_i^*}{\mu_i}\Big)^2\hat{\delta}_1(n+2) \\
&= \frac{C_i(n+1)}{\mu_i} + \frac{\lambda p_i^*}{\mu_i}\frac{C_i(n+2)}{\mu_i} + \Big(\frac{\lambda p_i^*}{\mu_i}\Big)^2\frac{C_i(n+3)}{\mu_i} + \Big(\frac{\lambda p_i^*}{\mu_i}\Big)^3\frac{C_i(n+4)}{\mu_i} \\
&\quad + \Big(\frac{\lambda p_i^*}{\mu_i}\Big)^4\frac{C_i(n+5)}{\mu_i} + \dots \\
&= \frac{1}{\mu_i}\sum_{x=0}^{\infty} C_i(n+1+x)\Big(\frac{\lambda p_i}{\mu_i}\Big)^x. \qquad (2.138)
\end{aligned}$$

Also note that using equation (2.132) we can see that the average cost rate for queue i is

$$C_i^* = \sum_{x=0}^{\infty} C_i(x)\Big(\frac{\lambda p_i^*}{\mu_i}\Big)^x\Big(\frac{\mu_i - \lambda p_i^*}{\mu_i}\Big). \qquad (2.139)$$

So now using expressions (2.136), (2.138) and (2.139) we can see that the expression in (2.134) becomes

$$
\begin{aligned}
PI_i(n) &= \hat{\delta}_i(n) - \delta_i(n)C_i^* \qquad (2.140)\\
&= \frac{1}{\mu_i}\sum_{x=0}^{\infty} C_i(n+1+x)\Big(\frac{\lambda p_i}{\mu_i}\Big)^x - \frac{1}{\mu_i}\sum_{x=0}^{\infty} C_i(x)\Big(\frac{\lambda p_i^*}{\mu_i}\Big)^x \\
&= \frac{1}{\mu_i}\sum_{x=0}^{\infty}\Big(\frac{\lambda p_i^*}{\mu_i}\Big)^x\Big(C_i(n+1+x) - C_i(x)\Big). \qquad (2.141)
\end{aligned}
$$

So we have now found our policy improvement index for this two server example.

Comments

1. Following Theorem 3 and the discussion in Section 2.2, the Whittle index policy for the K-class service control problem with average costs described in (2.6), is constructed by computing the index function $W_k(.)$ for each customer class k from an appropriate form of (2.122). At each epoch t, the index policy will admit the arriving customer to the queue with minimal index $W_k\{N_k(t)\}$.

2. Following the above formulation of the policy improvement index we can see that, the policy improvement index policy for the 2-class service control problem with average costs described in (2.6), is constructed by computing the index function $PI_k(.)$ for each customer class k from an appropriate form of (2.141). At each epoch t, the index policy will admit the arriving customer to the queue with minimal index $PI_k\{N_k(t)\}$.

3. Note that the form of the index in (2.140) will hold for the K class service control problem. However, a general formulation will be required for the optimal static policy and the queue k average cost rate, C_k^*.

2.5 Numerical investigation of routing index policies for multi-class systems

We have used a Lagrangian relaxation for our routing problem and studied the consequential service station problem with a charge for admission in Section 2.3.1. This has led us to a set of index heuristics for the problem with multiple service stations as in Section 2.2. An index for the discounted costs problem in (2.3) is obtained as a fair charge for rejection with an appropriate index for the average costs problem (2.6) obtained as a limit.

We will now investigate the performance of the index heuristics numerically. In the discounted case the investigation compares the expected cost of following the Whittle index policy with the optimal expected cost for problems with two service stations. However, our prime focus will be on average cost problems. For the average cost scenario we compare the average cost rate for the Whittle index policy to the optimal cost rate and the average cost rate for the policy improvement index policy. Further, for the average costs problem we use simulation techniques to compare cost rates for the Whittle index policy with those of competitor policies for problems with five service stations. For the five station problems, direct calculation of the cost rates would prove computationally intractable so we adopt a simulation approach. We begin with the study of some two service station problems with discounted costs.

2.5.1 Discounted cost problems with two service stations

In this section we study routing problems of the type described in Section 2.2 with two service stations. We consider the following four cost rate structures:

(a) $C_1(n) = n + 2n^2$; $\quad C_2(n) = 2n + 2n^2$; (quadratic)

(b) $C_1(n) = n^2 + 2n^3$; $\quad C_2(n) = 2n^2 + 2n^3$; (cubic)

(c) $C_1(n) = n^3 + 2n^4$; $\quad C_2(n) = 2n^3 + 2n^4$; (quartic)

(d) $C_1(n) = (n-2)^+ + 2\{(n-2)^+\}^2$; $\quad C_2(n) = 2(n-2)^+ + 2\{(n-2)^+\}^2$; (shifted quadratic)

Tables 2.1 - 2.16 contain the results of part of a study comparing the discounted costs incurred by the index heuristic described in Comment 3, on page 73, with those incurred by a similar heuristic found following a similar approach but which has followed a simpler analysis which allowed the number of customers present in the queue to take negative values, where zero holding costs are incurred. These index heuristics are also compared to the optimal policy for a range of service control problems with two customer classes. Tables 2.1 - 2.4 correspond to the cost structure (a), tables 2.5 - 2.8 correspond to the cost structure (b), tables 2.9 - 2.12 correspond to the cost structure (c) and tables 2.13 - 2.16 correspond to the cost structure (d) above. In these tables, the first row gives the starting state for the first customer class, and the first column gives the starting state for the second customer class. The choice of the arrival rate and the service rates for both queues are detailed in the caption on the bottom of each table. For case 1, λ is chosen such that the value of the $\Gamma = \frac{\lambda}{\mu_1+\mu_2}$ is 0.60, while for case 2, Γ is set to be 0.85. In cases 3 and 4 we can see that the mean service times are further apart than in 1 and 2. Again in case 3, λ is chosen to yield $\Gamma = 0.60$ while for 4 we have $\Gamma = 0.85$. Each block of data in each table consists of 3 data entries. The top entry is the discounted cost for the index policy as in comment 3, on page 73, the middle entry is the discounted cost for the index policy which allows negative customers and the bottom entry is the optimal cost.

In each case the the fully optimal policy is found using dynamic programming techniques and all costs are found by use of DP value iteration; see Chapter 3 of Tijms (1994). It is possible to use such methods for problems of this size, but computationally expensive.

state	0	1	2	3	4
	210.6236	214.6101	224.2219	241.0394	266.5968
0	211.0498	215.0440	224.6764	241.5365	267.1736
	210.5974	214.5836	224.1947	241.0106	266.5652
	214.1985	220.0420	231.5569	250.3041	277.8074
1	214.6584	220.5163	232.0634	250.8733	278.4908
	214.1985	220.0420	231.5569	250.3041	277.8074
	222.8250	230.8983	246.0696	268.5123	299.7393
2	223.2779	231.3760	246.6017	269.1446	300.5497
	222.7978	230.8702	246.0395	268.4786	299.6991
	237.8850	248.2554	267.6224	295.4074	332.0016
3	238.3777	248.7917	268.2537	296.2115	333.1077
	237.8564	248.2252	267.5887	295.3674	331.9506
	260.8293	273.5564	297.2191	330.9804	374.5085
4	261.3965	274.1992	298.0262	332.0853	376.1315
	260.7980	273.5223	297.1790	330.9294	374.4387

Table 2.1: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with quadratic costs and two customer classes. Case 1:* $b_1 = 1.0$, $b_2 = 2.0$, $\lambda = 3.0$, $\mu_1 = 2.65$, $\mu_2 = 2.35$.

state	0	1	2	3	4
	1050.8856	1064.2904	1092.9608	1138.6483	1203.0212
0	1050.8856	1064.2904	1092.9608	1138.6483	1203.0212
	1050.8840	1064.2887	1092.9591	1138.6466	1203.0195
	1063.5680	1083.6053	1119.1804	1172.0052	1243.7222
1	1063.5680	1083.6053	1119.1804	1172.0052	1243.7222
	1063.5663	1083.6037	1119.1788	1172.0035	1243.7205
	1090.5355	1118.1584	1165.1772	1229.8043	1313.6476
2	1090.5355	1118.1584	1165.1772	1229.8043	1313.6476
	1090.5338	1118.1567	1165.1755	1229.8027	1313.6460
	1133.5411	1169.0950	1228.7413	1308.4203	1407.6973
3	1133.5411	1169.0950	1228.7413	1308.4203	1407.6973
	1133.5394	1169.0934	1228.7397	1308.4187	1407.6957
	1194.2525	1238.0498	1310.7918	1406.6997	1523.8699
4	1194.2525	1238.0498	1310.7918	1406.6997	1523.8699
	1194.2509	1238.0482	1310.7902	1406.6980	1523.8682

Table 2.2: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with quadratic costs and two customer classes. Case 2:* $\lambda = 4.25$, $\mu_1 = 2.65$, $\mu_2 = 2.35$.

state	0	1	2	3	4
	208.6207	212.9358	223.3266	241.4969	269.0792
0	249.0692	252.7041	262.2288	279.5219	306.2667
	207.9200	212.2215	222.5758	240.6712	268.1491
	212.1878	218.2640	230.4185	250.3453	279.6542
1	253.3278	258.2361	269.3797	288.4064	316.8913
	211.4753	217.5281	229.6296	249.4531	278.6353
	220.5476	229.0665	244.7221	268.1279	300.8412
2	260.7760	269.1014	283.6986	306.2448	338.1859
	219.8044	228.2810	243.8458	267.0842	299.6304
	235.0967	246.1805	266.3587	294.9530	332.6982
3	274.4531	285.3669	305.2758	333.1736	370.2452
	234.2906	245.3001	265.3196	293.6280	331.1646
	257.2081	270.9803	295.8938	330.9430	375.2701
4	295.7851	309.4634	334.2895	369.4299	413.1829
	256.2847	269.9279	294.5669	329.1233	373.2524

Table 2.3: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with quadratic costs and two customer classes. Case 3:* $\lambda = 3.0$, $\mu_1 = 2.9$, $\mu_2 = 2.1$.

state	0	1	2	3	4
	1050.6659	1065.0865	1095.7043	1144.2248	1212.2337
0	1050.6659	1065.0865	1095.7043	1144.2248	1212.2337
	1046.7996	1061.1677	1091.6720	1140.0068	1207.8153
	1063.3456	1084.1245	1121.2328	1176.3613	1251.0558
1	1063.3456	1084.1245	1121.2328	1176.3613	1251.0558
	1059.4327	1080.1325	1117.0958	1172.0008	1246.4852
	1089.8994	1118.8187	1166.9900	1233.3734	1319.4345
2	1089.8994	1118.8187	1166.9900	1233.3735	1319.4345
	1085.8868	1114.6896	1162.6600	1228.7525	1314.6009
	1132.0602	1169.5495	1230.9220	1311.9617	1412.7757
3	1132.0602	1169.5495	1230.9220	1311.9617	1412.7757
	1127.8839	1165.2113	1226.3087	1306.9635	1407.5994
	1191.4890	1237.9490	1313.1064	1410.8746	1528.9781
4	1191.4890	1237.9490	1313.1064	1410.8746	1528.9781
	1187.0719	1233.3142	1308.1058	1405.3686	1523.4175

Table 2.4: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with quadratic costs and two customer classes. Case 4:* $\lambda = 4.25$, $\mu_1 = 2.9$, $\mu_2 = 2.1$.

state	0	1	2	3	4
	579.5455	590.1897	620.2025	682.9277	796.2813
0	579.5456	590.1898	620.2027	682.9278	796.2815
	579.4717	590.1148	620.1246	682.8435	796.1854
	589.4542	607.2851	646.3164	719.7386	845.3558
1	589.4544	607.2852	646.3166	719.7388	845.3560
	589.3793	607.2082	646.2350	719.6483	845.2493
	616.5730	644.4031	702.0086	797.0745	947.2250
2	616.5732	644.4033	702.0088	797.0747	947.2252
	616.4953	644.3218	701.9193	796.9704	947.0948
	672.5226	712.4430	793.6634	922.8173	1111.0222
3	672.5227	712.4432	793.6636	922.8174	1111.0224
	672.4390	712.3531	793.5595	922.6879	1110.8485
	773.0782	827.1199	935.5585	1105.8381	1347.7988
4	773.0784	827.1200	935.5587	1105.8383	1347.7990
	772.9837	827.0142	935.4287	1105.6645	1347.5491

Table 2.5: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with cubic costs and two customer classes. Case 1:* $\lambda = 3.0$, $\mu_1 = 2.65$, $\mu_2 = 2.35$.

state	0	1	2	3	4
	7106.0088	7195.7841	7393.0781	7720.6491	8206.1591
0	7106.0088	7195.7841	7393.0781	7720.6491	8206.1591
	7106.0082	7195.7836	7393.0776	7720.6486	8206.1585
	7191.7658	7331.3238	7585.7445	7978.0101	8536.0043
1	7191.7658	7331.3238	7585.7445	7978.0101	8536.0043
	7191.7652	7331.3233	7585.7439	7978.0095	8536.0038
	7378.8421	7579.8333	7931.7508	8434.0154	9114.8721
2	7378.8421	7579.8333	7931.7508	8434.0154	9114.8721
	7378.8415	7579.8328	7931.7502	8434.0149	9114.8715
	7688.8258	7960.0768	8427.4549	9076.2076	9919.9344
3	7688.8258	7960.0768	8427.4549	9076.2076	9919.9344
	7688.8252	7960.0762	8427.4544	9076.2070	9919.9338
	8147.8351	8498.4886	9095.8332	9913.1245	10955.6911
4	8147.8351	8498.4886	9095.8332	9913.1245	10955.6911
	8147.8345	8498.4880	9095.8327	9913.1239	10955.6906

Table 2.6: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with cubic costs and two customer classes. Case 2:* $\lambda = 4.25$, $\mu_1 = 2.65$, $\mu_2 = 2.35$.

state	0	1	2	3	4
	574.9301	586.4806	619.0386	687.1567	810.4836
0	598.2604	610.2718	644.0628	713.5463	837.9525
	570.1384	581.5945	613.8998	681.4983	803.8525
	584.7599	603.2600	644.4131	722.5880	857.3360
1	608.4892	627.7805	670.7283	750.3845	886.0301
	579.8865	598.2244	639.0098	716.4692	849.9110
	611.1339	640.4110	699.7668	798.9148	957.2751
2	635.9035	666.6096	729.0360	829.5837	988.2723
	606.0473	635.0315	693.7581	791.7458	948.0414
	665.0901	707.6189	792.1219	925.0937	1121.1018
3	691.9848	737.0278	826.8965	960.0432	1155.0850
	659.5680	701.5816	784.9851	915.9748	1108.5168
	761.6057	819.9256	933.9976	1110.9633	1361.5073
4	790.5301	851.3064	969.5408	1150.7708	1398.2818
	755.2704	812.6960	924.8661	1098.4163	1343.0415

Table 2.7: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with cubic costs and two customer classes. Case 3:* $\lambda = 3.0$, $\mu_1 = 2.9$, $\mu_2 = 2.1$.

state	0	1	2	3	4
	7149.3137	7246.5323	7458.5645	7808.6500	8325.5481
0	7080.7352	7177.0227	7387.0405	7733.8313	8245.9022
	7075.3750	7171.5898	7381.4502	7727.9835	8239.6771
	7235.5933	7381.0814	7647.6980	8058.9289	8643.7842
1	7166.1871	7310.2729	7574.3155	7981.5793	8560.7918
	7160.7623	7304.7385	7568.5799	7975.5336	8554.3052
	7420.9079	7632.2092	7994.4350	8512.5139	9215.8830
2	7349.7321	7558.9669	7917.6270	8430.5436	9126.8176
	7344.1689	7553.2423	7911.6237	8424.1368	9119.8563
	7726.4481	8013.3554	8496.1399	9159.0948	10022.6700
3	7652.3696	7936.4023	8414.3057	9070.4315	9924.8798
	7646.5796	7930.3876	8407.9095	9063.5016	9917.2366
	8177.8308	8550.6427	9169.8429	10006.6384	11068.3765
4	8099.4777	8468.4273	9081.1360	9908.9637	10958.9430
	8093.3536	8462.0013	9074.2027	9901.3295	10950.3899

Table 2.8: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with cubic costs and two customer classes. Case 4:* $\lambda = 4.25$, $\mu_1 = 2.9$, $\mu_2 = 2.1$.

state	0	1	2	3	4
	2109.2934	2147.5442	2256.6994	2505.7378	3018.8653
0	2109.2817	2147.5323	2256.6869	2505.7241	3018.8494
	2109.2619	2147.5125	2256.6669	2505.7039	3018.8288
	2145.3561	2212.8907	2364.7870	2671.6584	3259.0952
1	2145.3442	2212.8784	2364.7738	2671.6435	3259.0772
	2145.3243	2212.8585	2364.7538	2671.6231	3259.0563
	2245.4987	2359.0160	2602.6807	3031.2814	3772.8443
2	2245.4862	2359.0029	2602.6660	3031.2639	3772.8217
	2245.4663	2358.9828	2602.6456	3031.2431	3772.8000
	2469.1778	2645.9765	3018.8554	3651.1780	4645.3840
3	2469.1642	2645.9617	3018.8379	3651.1556	4645.3531
	2469.1441	2645.9413	3018.8170	3651.1339	4645.3299
	2923.7507	3183.2774	3722.2799	4621.4964	5998.0101
4	2923.7351	3183.2596	3722.2575	4621.4656	5997.9647
	2923.7145	3183.2387	3722.2358	4621.4423	5997.9389

Table 2.9: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with quartic costs and two customer classes. Case 1:* $\lambda = 3.0$, $\mu_1 = 2.65$, $\mu_2 = 2.35$.

state	0	1	2	3	4
	64887.4241	65705.9711	67496.6050	70468.7657	74906.8094
0	64887.4241	65705.9711	67496.6050	70468.7657	74906.8094
	64887.4238	65705.9708	67496.6047	70468.7653	74906.8090
	65670.5008	66950.5934	69289.9911	72911.8057	78114.1912
1	65670.5008	66950.5934	69289.9911	72911.8057	78114.1912
	65670.5005	66950.5931	69289.9907	72911.8054	78114.1908
	67374.5753	69240.7964	72525.9772	77266.1509	83782.1075
2	67374.5753	69240.7964	72525.9772	77266.1509	83782.1075
	67374.5750	69240.7961	72525.9768	77266.1506	83782.1072
	70200.8249	72765.8042	77214.7761	83474.4813	91770.2630
3	70200.8249	72765.8042	77214.7761	83474.4813	91770.2630
	70200.8246	72765.8039	77214.7758	83474.4810	91770.2626
	74418.0943	77810.2515	83633.8381	91718.9023	102250.6385
4	74418.0943	77810.2515	83633.8381	91718.9023	102250.6385
	74418.0939	77810.2512	83633.8378	91718.9019	102250.6381

Table 2.10: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with quartic costs and two customer classes. Case 2:* $\lambda = 4.25$, $\mu_1 = 2.65$, $\mu_2 = 2.35$.

state	0	1	2	3	4
0	2127.0951 2180.9986 2071.1835	2169.3041 2224.2724 2112.2884	2289.5092 2347.3262 2229.5395	2563.7419 2627.4181 2497.6964	3129.6435 3199.1128 3052.2272
1	2163.4622 2218.2873 2106.5949	2234.6058 2291.2591 2175.8428	2396.7966 2457.5962 2333.7340	2727.5370 2796.4033 2656.1096	3365.9486 3440.6608 3279.2494
2	2262.4082 2319.6368 2203.0485	2383.4031 2443.9336 2320.6196	2638.0194 2705.6441 2567.8794	3091.0231 3171.7306 3007.3170	3884.4790 3969.6287 3776.6340
3	2480.9823 2543.1208 2416.5312	2671.6879 2739.6351 2601.2135	3065.2015 3145.5459 2981.8719	3727.0094 3829.7011 3620.5065	4780.8898 4880.6945 4633.8632
4	2921.5586 2992.8661 2847.5998	3205.3988 3286.7887 3120.9850	3781.4766 3884.3102 3674.8266	4733.7817 4875.1238 4587.2002	6190.5413 6304.1979 5974.7592

Table 2.11: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with quartic costs and two customer classes. Case 3:* $\lambda = 3.0$, $\mu_1 = 2.9$, $\mu_2 = 2.1$.

state	0	1	2	3	4
0	65041.7051 64595.5474 64595.1725	65924.8975 65472.6825 65472.3026	67841.9398 67376.6201 67376.2291	71005.2463 70518.4916 70518.0827	75709.5283 75191.3686 75190.9333
1	65826.6437 65375.1017 65374.7222	67155.9544 66695.2890 66694.9020	69596.6226 69119.2119 69118.8108	73374.3762 72871.1559 72870.7331	78797.0890 78257.1584 78256.7047
2	67508.6157 67045.5608 67045.1718	69461.6124 68985.1134 68984.7130	72827.8211 72328.1245 72327.7047	77691.5835 77158.3018 77157.8537	84383.0007 83803.5594 83803.0726
3	70284.7936 69802.8548 69802.4498	72982.8695 72482.2291 72481.8084	77554.1393 77021.7431 77021.2958	83918.0782 83341.2533 83340.7687	92359.2651 91723.0623 91722.5279
4	74417.8206 73908.0719 73907.6436	78000.5661 77465.6895 77465.2401	83999.5624 83422.4535 83421.9687	92228.7285 91593.2774 91592.7436	102892.3862 102180.4346 102179.8366

Table 2.12: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with quartic costs and two customer classes. Case 4:* $\lambda = 4.25$, $\mu_1 = 2.9$, $\mu_2 = 2.1$.

state	0	1	2	3	4
0	31.2491	31.6183	32.7888	36.6648	45.0288
	31.2844	31.6540	32.8258	36.7047	45.0743
	26.9477	27.4341	28.7390	32.7448	41.2298
1	31.7834	32.4481	34.2664	38.9945	48.3504
	31.8193	32.4847	34.3051	39.0374	48.4011
	27.4085	28.2842	30.2525	35.1092	44.5814
2	32.7988	34.2566	37.5818	44.0896	55.4401
	32.8359	34.2953	37.6242	44.1391	55.5022
	28.6332	30.2047	33.6487	40.2692	51.7241
3	36.0104	38.4631	43.7577	53.1087	67.5417
	36.0501	38.5058	43.8071	53.1704	67.6248
	32.0053	34.5311	39.9218	49.3684	63.8910
4	43.0500	46.6710	54.2223	66.9752	85.6920
	43.0949	46.7212	54.2842	67.0582	85.8120
	39.1796	42.8536	50.4824	63.3166	82.1069

Table 2.13: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with shifted quadratic costs and two customer classes. Case 1:* $\lambda = 3.0$, $\mu_1 = 2.65$, $\mu_2 = 2.35$.

state	0	1	2	3	4
0	475.8122	480.0174	490.8712	511.0357	542.7625
	475.8122	480.0174	490.8712	511.0358	542.7625
	451.1069	456.7966	469.2034	490.7747	523.7593
1	481.5544	488.1356	502.7966	527.4117	564.0322
	481.5544	488.1356	502.7966	527.4117	564.0322
	456.5510	465.4553	481.7261	507.6839	545.4952
2	490.9464	502.7274	524.4071	556.6570	601.5267
	490.9464	502.7274	524.4071	556.6570	601.5267
	468.3720	481.3952	504.3447	537.7662	583.7152
3	509.4345	526.4045	556.1981	598.5014	654.1519
	509.4345	526.4045	556.1981	598.5014	654.1519
	488.5927	506.3915	537.2245	580.5523	637.1744
4	538.8534	561.3471	599.9941	653.5595	722.1017
	538.8534	561.3471	599.9941	653.5595	722.1017
	519.4092	542.5396	582.0633	636.5476	705.9767

Table 2.14: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with shifted quadratic costs and two customer classes. Case 2:* $\lambda = 4.25$, $\mu_1 = 2.65$, $\mu_2 = 2.35$.

state	0	1	2	3	4
	30.2241	30.6454	31.9404	36.1923	45.3463
0	32.5826	33.0367	34.4328	38.7933	48.0329
	26.4916	27.0149	28.4152	32.7522	41.9585
	30.7409	31.4643	33.3931	38.4541	48.5294
1	33.1397	33.9195	35.9988	41.1757	51.3219
	26.9446	27.8432	29.8813	35.0147	45.1204
	31.7661	33.3003	36.7184	43.5075	55.4888
2	34.2449	35.8988	39.5836	46.4732	58.4810
	28.1260	29.7491	33.2362	40.0606	52.0224
	34.8702	37.4670	42.9533	52.5552	67.5519
3	37.5290	40.3454	46.3050	55.8782	70.8068
	31.3362	33.9751	39.4885	49.0670	63.9572
	41.5983	45.4800	53.3873	66.5881	85.8655
4	44.4224	48.5122	56.7659	70.2929	89.3824
	38.1297	42.0154	49.8926	62.9838	82.0107

Table 2.15: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with shifted quadratic costs and two customer classes. Case 3:* $\lambda = 3.0$, $\mu_1 = 2.9$, $\mu_2 = 2.1$.

state	0	1	2	3	4
	472.5335	477.3083	489.2157	510.9831	544.9370
0	471.4551	476.2191	488.0993	509.8199	543.7036
	449.3343	455.4347	468.6328	491.4826	526.3142
	478.2361	485.4279	501.0034	526.9642	565.4672
1	477.1448	484.3202	499.8601	525.7638	564.1842
	454.7570	463.9453	480.8099	507.7632	547.0533
	487.8988	500.3802	522.7890	556.1150	602.5016
2	486.7854	499.2384	521.5960	554.8466	601.1286
	466.3519	479.8917	503.2545	537.3771	584.3971
	506.2955	524.2528	555.0362	598.2433	655.1402
3	505.1417	523.0575	553.7696	596.8759	653.6375
	486.0705	504.6858	536.1870	579.9889	637.3412
	535.2722	559.1513	599.1586	653.9117	723.5491
4	534.0567	557.8791	597.7905	652.4105	721.8727
	516.0351	540.3562	580.8817	636.0608	705.9687

Table 2.16: *Comparative performance of the index heuristics and an optimal policy with various starting states for the discounted problem with shifted quadratic costs and two customer classes. Case 4:* $\lambda = 4.25$, $\mu_1 = 2.9$, $\mu_2 = 2.1$.

2.5.2 Average cost problems with two customer classes

Tables 2.17 - 2.24 below contain the results of part of a study comparing the average costs incurred by the index heuristic described in the comment following Theorem 3 with rates incurred by an optimal policy. Again the optimal policies were found using dynamic programming techniques, and the cost rates by DP value iteration. All the admission control problems studied here have two service stations. Each cell in the body of the table gives results for four different cost structures in the form

a	[a]	(a)	b	[b]	(b)
c	[c]	(c)	d	[d]	(d)

The corresponding cost rates are as follows:

(a) $C_1(n) = b_1 n + 2n^2$; $C_2(n) = b_2 n + 2n^2$; (quadratic)

(b) $C_1(n) = b_1 n^2 + 2n^3$; $C_2(n) = b_2 n^2 + 2n^3$; (cubic)

(c) $C_1(n) = b_1 n^3 + 2n^4$; $C_2(n) = b_2 n^3 + 2n^4$; (quartic)

(d) $C_1(n) = b_1(n-1)^+ + 2\{(n-1)^+\}^2$; $C_2(n) = b_2(n-1)^+ + 2\{(n-1)^+\}^2$; (shifted quadratic)

In all cases the unbracketed figure (a, b, c or d) is the average cost rate for the index policy deduced in Section 2.4.1, the figure in square brackets is the corresponding average cost rate for the policy improvement index policy of Section 2.4.2, with the relevant optimal cost in round brackets, $(\cdot)$. The first two columns of tables 2.17 - 2.24, give the service rates for the queues which apply to the values in the corresponding row. The vales of the cost coefficients, b_1, b_2 are also clearly labelled in the tables. The arrival rate λ is chosen to give a Γ-value of 0.60 in tables 2.17 - 2.20. The arrival rate λ is modified in tables 2.21 - 2.24 to give a *Gamma*-value of 0.85, as indicated. Recall that $\Gamma = \frac{\lambda}{\mu_1+\mu_2}$.

$\Gamma = 0.6$

μ_1	μ_2	$b_1 = 0.4$, $b_2 = 0.4$					
3.0	3.0	4.6834	[4.6834]	(4.6833)	12.0902	[12.0902]	(12.0902)
		42.3444	[42.3444]	(42.3444)	1.8371	[1.8371]	(1.8370)
2.9	3.1	4.6548	[4.6551]	(4.6544)	11.9880	[11.9872]	(11.9872)
		41.9291	[41.9151]	(41.9148)	1.8334	[1.8210]	(1.8199)
2.8	3.2	4.6314	[4.6340]	(4.6311)	11.9229	[11.9092]	(11.9092)
		41.7396	[41.5874]	(41.5874)	1.8320	[1.8090]	(1.8056)
2.7	3.3	4.6129	[4.6200]	(4.6116)	11.8948	[11.8560]	(11.8556)
		42.1028	[41.3606]	(41.3604)	1.8329	[1.8007]	(1.7933)

μ_1	μ_2	$b_1 = 0.6$, $b_2 = 0.4$					
3.0	3.0	5.0408	[5.0408]	(5.0161)	13.1741	[13.1648]	(13.1260)
		46.6334	[46.6104]	(46.5482)	1.9678	[1.9645]	(1.9509)
2.9	3.1	4.9863	[5.0012]	(4.9795)	13.0368	[13.0703]	(13.0145)
		46.1237	[46.2167]	(46.1193)	1.9553	[1.9489]	(1.9306)
2.8	3.2	4.9454	[4.9816]	(4.9432)	12.9196	[12.9640]	(12.9194)
		45.8709	[45.8658]	(45.7717)	1.9420	[1.9171]	(1.9106)
2.7	3.3	4.9151	[4.9474]	(4.9148)	12.8515	[12.9178]	(12.8398)
		45.7536	[45.6833]	(45.5361)	1.9309	[1.9058]	(1.8907)

μ_1	μ_2	$b_1 = 1.0$, $b_2 = 0.4$					
3.0	3.0	5.6167	[5.6594]	(5.5732)	15.0649	[14.9225]	(14.8707)
		54.1689	[54.1310]	(53.6550)	2.1697	[2.1623]	(2.1267)
2.9	3.1	5.5571	[5.5571]	(5.5266)	14.7828	[14.8010]	(14.7241)
		53.2493	[53.7398]	(53.0168)	2.1323	[2.1174]	(2.0947)
2.8	3.2	5.5124	[5.5240]	(5.4879)	14.5915	[14.6923]	(14.5626)
		52.5176	[52.9124]	(52.5149)	2.1139	[2.0863]	(2.0680)
2.7	3.3	5.4467	[5.5015]	(5.4286)	14.4342	[14.6336]	(14.4328)
		52.1898	[52.7031]	(52.1431)	2.0951	[2.0668]	(2.0460)

μ_1	μ_2	$b_1 = 1.4$, $b_2 = 0.4$					
3.0	3.0	6.1719	[6.1615]	(6.0729)	16.6469	[16.5798]	(19.3000)
		60.6306	[59.6079]	(59.3869)	2.3272	[2.3079]	(2.2672)
2.9	3.1	6.0632	[6.0867]	(5.9874)	16.2531	[16.4682]	(16.0830)
		59.0027	[59.2081]	(58.7548)	2.2966	[2.2754]	(2.2331)
2.8	3.2	5.9792	[5.9958]	(5.9053)	15.9983	[16.1123]	(15.8998)
		58.2726	[58.5643]	(58.2243)	2.2598	[2.2288]	(2.2031)
2.7	3.3	5.8552	[5.9275]	(5.8344)	15.7721	[16.0090]	(15.7526)
		57.6491	[58.2733]	(57.6315)	2.2444	[2.2098]	(2.1734)

Table 2.17: *Comparative performance of the index heuristic, policy improvement and optimal policies for a range of average costs problems with two customer classes, where* $\Gamma = 0.6$.

$\Gamma = 0.6$

μ_1	μ_2	$b_1 = 0.4,\ b_2 = 0.6$					
3.0	3.0	5.2306	[5.2305]	(5.2301)	13.7915	[13.8052]	(13.7913)
		49.0665	[49.1625]	(49.0660)	2.0526	[2.0520]	(2.0521)
2.9	3.1	5.2020	[5.2005]	(5.1952)	13.6974	[13.7203]	(13.6691)
		48.7855	[50.7610]	(48.5914)	2.0488	[2.0319]	(2.0318)
2.8	3.2	5.1708	[5.1668]	(5.1658)	13.5886	[13.5694]	(13.5631)
		48.4274	[48.1787]	(48.1787)	2.0470	[2.0158]	(2.0157)
2.7	3.3	5.1531	[5.1429]	(5.1428)	13.5871	[13.4816]	(13.4798)
		48.5035	[47.9160]	(47.8483)	2.0606	[2.0036]	(2.0035)

μ_1	μ_2	$b_1 = 0.6,\ b_2 = 0.6$					
3.0	3.0	5.6039	[5.6039]	(5.6039)	14.9319	[14.9319]	(14.9319)
		53.6952	[53.8206]	(53.6950)	2.1900	[2.1900]	(2.1900)
2.9	3.1	5.5660	[5.5659]	(5.5659)	14.7964	[14.7938]	(14.7938)
		53.1266	[53.1146]	(53.1141)	2.1846	[2.1693]	(2.1693)
2.8	3.2	5.5382	[5.5369]	(5.5368)	14.7219	[14.6873]	(14.6873)
		52.9498	[52.6639]	(52.6643)	2.1851	[2.1536]	(2.1535)
2.7	3.3	5.5190	[5.5168]	(5.5152)	14.6888	[14.6120]	(14.6119)
		52.9600	[52.3428]	(52.3430)	2.1905	[2.1428]	(2.1415)

μ_1	μ_2	$b_1 = 1.0,\ b_2 = 0.6$					
3.0	3.0	6.3120	[6.3119]	(6.2737)	17.0824	[17.0616]	(16.9974)
		62.2495	[62.2249]	(62.0352)	2.4466	[2.4371]	(2.4245)
2.9	3.1	6.2385	[6.2633]	(6.2229)	16.8554	[16.9385]	(16.8525)
		61.5005	[61.5964]	(61.4332)	2.4189	[2.4160]	(2.3984)
2.8	3.2	6.1802	[6.2082]	(6.1750)	16.7127	[16.7492]	(16.7126)
		61.0723	[61.1007]	(60.9819)	2.4127	[2.4009]	(2.3757)
2.7	3.3	6.1386	[6.1864]	(6.1363)	16.6181	[16.6770]	(16.6076)
		61.1909	[60.8508]	(60.6114)	2.4001	[2.3675]	(2.3500)

μ_1	μ_2	$b_1 = 1.4,\ b_2 = 0.6$					
3.0	3.0	6.9305	[6.9188]	(6.8333)	18.9399	[18.9295]	(18.7490)
		70.0573	[69.6030]	(69.1490)	2.6461	[2.6414]	(2.6121)
2.9	3.1	6.8150	[6.8160]	(6.7724)	18.6083	[18.8028]	(18.5751)
		68.5342	[68.5706]	(68.4657)	2.6308	[2.6124]	(2.5708)
2.8	3.2	6.7557	[6.7738]	(6.7230)	18.4493	[18.5058]	(18.4032)
		67.7822	[68.6854]	(67.7812)	2.5856	[2.5949]	(2.5362)
2.7	3.3	6.6938	[6.7253]	(6.6832)	18.2300	[18.4022]	(18.2290)
		67.4087	[67.7663]	(67.2660)	2.5654	[2.5280]	(2.5079)

Table 2.18: *Comparative performance of the index heuristic, policy improvement and optimal policies for a range of average costs problems with two customer classes, where* $\Gamma = 0.6$.

$\Gamma = 0.6$

μ_1	μ_2	$b_1 = 0.4,\ b_2 = 1.0$					
3.0	3.0	6.0729	[6.1094]	(6.0402)	16.2659	[16.2925]	(16.1942)
		58.3800	[58.3485]	(58.2064)	2.3891	[2.4192]	(2.3735)
2.9	3.1	6.1026	[6.0766]	(6.0654)	16.3191	[16.4827]	(16.2430)
		58.4861	[59.0959]	(58.2364)	2.4100	[2.4366]	(2.3683)
2.8	3.2	6.0976	[6.1131]	(6.0612)	16.4339	[16.7084]	(16.2221)
		60.2213	[60.0497]	(58.2440)	2.4338	[2.4601]	(2.3616)
2.7	3.3	6.1963	[6.1488]	(6.0618)	16.7823	[16.5975]	(16.2371)
		61.1739	[61.0907]	(58.1657)	2.4647	[2.3811]	(2.3536)

μ_1	μ_2	$b_1 = 0.6,\ b_2 = 1.0$					
3.0	3.0	6.6158	[6.6195]	(6.6100)	17.9803	[18.0207]	(17.9731)
		65.4245	[65.4218]	(65.4105)	2.5904	[2.5946]	(2.5876)
2.9	3.1	6.6047	[6.5917]	(6.5878)	18.0426	[18.3330]	(17.9235)
		66.8533	[66.5525]	(65.1962)	2.6035	[2.5897]	(2.5678)
2.8	3.2	6.6082	[6.5792]	(6.5643)	18.0356	[17.8293]	(17.8261)
		66.1251	[64.9661]	(64.9063)	2.6069	[2.5905]	(2.5521)
2.7	3.3	6.5747	[6.5415]	(6.5396)	17.9279	[17.9635]	(17.7492)
		66.0280	[68.9507]	(64.6110)	2.6113	[2.5378]	(2.5368)

μ_1	μ_2	$b_1 = 1.0,\ b_2 = 1.0$					
3.0	3.0	7.4448	[7.4448]	(7.4447)	20.6151	[20.6151]	(20.6151)
		76.3966	[76.3966]	(76.3966)	2.8959	[2.8959]	(2.8959)
2.9	3.1	7.3878	[7.3878]	(7.3877)	20.4096	[20.4072]	(20.4072)
		75.5371	[75.5133]	(75.5129)	2.8863	[2.8660]	(2.8660)
2.8	3.2	7.3458	[7.3429]	(7.3429)	20.2782	[20.2437]	(20.2435)
		75.0952	[74.8172]	(74.8176)	2.8860	[2.8428]	(2.8428)
2.7	3.3	7.3175	[7.3104]	(7.3104)	20.3021	[20.1240]	(20.1239)
		75.4889	[74.3073]	(74.3073)	2.8940	[2.8262]	(2.8262)

μ_1	μ_2	$b_1 = 1.4,\ b_2 = 1.0$					
3.0	3.0	8.1664	[8.1643]	(8.1490)	22.7889	[22.7852]	(22.7680)
		84.9780	[84.9767]	(84.9516)	3.1630	[3.1613]	(3.1522)
2.9	3.1	8.0941	[8.1100]	(8.0863)	22.5724	[22.5960]	(22.5630)
		84.1288	[84.1562]	(84.1286)	3.1490	[3.1338]	(3.1209)
2.8	3.2	8.0359	[8.0649]	(8.0353)	22.4123	[22.4261]	(22.4064)
		83.7622	[83.5558]	(83.4947)	3.1358	[3.1056]	(3.0920)
2.7	3.3	7.9985	[8.0397]	(7.9981)	22.3542	[22.3289]	(22.2796)
		83.8105	[83.1097]	(83.0475)	3.1348	[3.0811]	(3.0692)

Table 2.19: *Comparative performance of the index heuristic, policy improvement and optimal policies for a range of average costs problems with two customer classes, where* $\Gamma = 0.6$.

$\Gamma = 0.6$

μ_1	μ_2	$b_1 = 0.4,\ b_2 = 1.4$					
3.0	3.0	6.7472	[6.7214]	(6.6788)	18.1817	[18.1435]	(18.0414)
		66.2591	[65.4984]	(65.4007)	2.6540	[2.6266]	(2.6122)
2.9	3.1	6.8067	[6.7982]	(6.7180)	18.5032	[18.3637]	(18.1553)
		67.3365	[66.5663]	(65.8428)	2.6803	[2.6657]	(2.6086)
2.8	3.2	6.8797	[6.8703]	(6.7523)	18.6453	[18.6008]	(18.3039)
		67.7784	[67.5016]	(66.0204)	2.7121	[2.6877]	(2.6105)
2.7	3.3	6.9114	[6.9619]	(6.7954)	18.8816	[18.9638]	(18.3505)
		70.2469	[68.8888]	(66.1573)	2.7626	[2.7718]	(2.6190)

μ_1	μ_2	$b_1 = 0.6,\ b_2 = 1.4$					
3.0	3.0	7.3734	[7.3785]	(7.3225)	20.1766	[20.1999]	(20.0887)
		73.9962	[73.9030]	(73.6908)	2.8845	[2.8918]	(2.8683)
2.9	3.1	7.4088	[7.4478]	(7.3511)	20.3271	[20.4256]	(20.2040)
		74.5335	[74.3698]	(73.9675)	2.9077	[2.9328]	(2.8606)
2.8	3.2	7.4389	[7.5303]	(7.3924)	20.4119	[20.4168]	(20.2310)
		75.0856	[75.9914]	(73.9404)	2.9382	[2.9465]	(2.8597)
2.7	3.3	7.5068	[7.4657]	(7.3943)	21.1617	[21.0369]	(20.2398)
		78.7698	[75.7845]	(74.1005)	3.0222	[2.9841]	(2.8607)

μ_1	μ_2	$b_1 = 1.0,\ b_2 = 1.4$					
3.0	3.0	8.4569	[8.4635]	(8.4550)	23.6638	[23.7822]	(23.6628)
		88.1260	[88.3171]	(88.1259)	3.2964	[3.3561]	(3.2955)
2.9	3.1	8.4816	[8.4249]	(8.4268)	23.8277	[23.5746]	(23.5738)
		89.2531	[87.9864]	(87.7884)	3.3161	[3.2704]	(3.2698)
2.8	3.2	8.4151	[8.4133]	(8.3842)	23.6061	[24.0673]	(23.4464)
		88.3830	[91.1501]	(87.3327)	3.3067	[3.2520]	(3.2458)
2.7	3.3	8.3805	[8.3526]	(8.3446)	23.5081	[23.3762]	(23.3101)
		88.2253	[87.2574]	(86.8990)	3.3197	[3.2264]	(3.2262)

μ_1	μ_2	$b_1 = 1.4,\ b_2 = 1.4$					
3.0	3.0	9.2857	[9.2857]	(9.2856)	26.2985	[26.2985]	(26.2985)
		99.0985	[99.0987]	(99.0984)	3.6017	[3.6017]	(3.6017)
2.9	3.1	9.2097	[9.2094]	(9.2093)	26.0244	[26.0207]	(26.0207)
		97.9471	[97.9118]	(97.9117)	3.5881	[3.5627]	(3.5627)
2.8	3.2	9.1545	[9.1488]	(9.1488)	25.8545	[25.7998]	(25.7998)
		97.3841	[96.9712]	(96.9714)	3.5878	[3.5321]	(3.5321)
2.7	3.3	9.1186	[9.1040]	(9.1040)	25.7745	[25.6358]	(25.6358)
		98.0236	[96.2721]	(96.2711)	3.5997	[3.5097]	(3.5097)

Table 2.20: *Comparative performance of the index heuristic, policy improvement and optimal policies for a range of average costs problems with two customer classes, where* $\Gamma = 0.6$.

$\Gamma = 0.85$

μ_1	μ_2	$b_1 = 0.4$, $b_2 = 0.4$					
3.0	3.0	25.8164	[25.8236]	(25.8164)	211.4338	[211.4373]	(211.4341)
		2421.9676	[2421.9690]	(2421.9674)	18.8234	[18.8234]	(18.8155)
2.9	3.1	25.7395	[25.7468]	(25.7312)	210.8907	[210.6859]	(210.6858)
		2422.9175	[2412.8310]	(2412.8310)	18.7867	[18.7626]	(18.7662)
2.8	3.2	25.6884	[25.6870]	(25.6671)	211.7843	[210.1986]	(210.1974)
		2460.2556	[2408.1363]	(2406.7730)	18.7774	[18.7136]	(18.6932)
2.7	3.3	25.7176	[25.6507]	(25.6203)	214.2302	[209.9905]	(209.9715)
		2541.8947	[2404.4482]	(2403.8347)	18.8414	[18.6830]	(18.6532)

μ_1	μ_2	$b_1 = 0.6$, $b_2 = 0.4$					
3.0	3.0	29.5924	[29.2174]	(28.9026)	246.3017	[243.3362]	(241.5407)
		2851.2381	[2810.8441]	(2804.7759)	21.5524	[21.2689]	(20.9804)
2.9	3.1	29.1506	[29.1179]	(28.7739)	241.6720	[242.6235]	(240.5096)
		2797.9065	[2807.1102]	(2793.1346)	21.2030	[21.1385]	(20.8756)
2.8	3.2	28.8517	[28.9872]	(28.6615)	239.8262	[240.5711]	(239.7197)
		2792.7519	[2786.5804]	(2784.8574)	20.9950	[20.9770]	(20.7903)
2.7	3.3	28.7011	[28.7900]	(28.5668)	239.9712	[240.5790]	(239.0994)
		2835.2423	[2787.9343]	(2779.1684)	20.8573	[20.9078]	(20.7122)

μ_1	μ_2	$b_1 = 1.0$, $b_2 = 0.4$					
3.0	3.0	36.0308	[34.7843]	(33.3411)	305.2166	[293.3307]	(286.0954)
		3585.8893	[3450.4940]	(3382.4618)	26.0803	[25.0129]	(23.9295)
2.9	3.1	35.3547	[34.5127]	(33.1038)	296.3065	[292.3175]	(284.5124)
		3454.2268	[3401.4067]	(3366.6708)	25.5474	[24.8077]	(23.7462)
2.8	3.2	34.5763	[34.2876]	(32.9158)	289.0565	[288.3444]	(283.2866)
		3369.5437	[3399.6713]	(3353.6506)	24.9680	[24.5358]	(23.5916)
2.7	3.3	33.8529	[33.8236]	(32.7039)	285.1398	[287.3589]	(282.1356)
		3349.5504	[3368.7248]	(3343.6295)	24.4291	[24.3506]	(23.4608)

μ_1	μ_2	$b_1 = 1.4$, $b_2 = 0.4$					
3.0	3.0	42.5630	[39.3851]	(36.7545)	364.2297	[336.0389]	(319.7543)
		4294.4414	[3912.3691]	(3825.8441)	30.6657	[28.1282]	(26.0534)
2.9	3.1	40.8805	[38.8540]	(36.4226)	346.3158	[331.5464]	(317.4964)
		4049.2986	[3909.7359]	(3804.1129)	29.6173	[27.5931]	(25.8105)
2.8	3.2	40.2702	[38.6694]	(36.0798)	334.6207	[327.3566]	(315.5333)
		3886.2964	[3873.2084]	(3786.3108)	28.4826	[27.2911]	(25.5778)
2.7	3.3	38.8779	[38.2531]	(35.7677)	324.1891	[326.3348]	(314.0311)
		3791.1130	[3849.4338]	(3773.4217)	27.5069	[27.1213]	(25.3640)

Table 2.21: *Comparative performance of the index heuristic, policy improvement and optimal policies for a range of average costs problems with two customer classes, where* $\Gamma = 0.85$.

$\Gamma = 0.85$

μ_1	μ_2	$b_1 = 0.4,\ b_2 = 0.6$					
3.0	3.0	29.9513	[29.9427]	(29.7541)	249.2428	[248.8191]	(248.1423)
		2880.9098	[2876.0652]	(2869.9107)	21.8268	[21.8376]	(21.6779)
2.9	3.1	30.1397	[29.9416]	(29.7009)	252.2198	[250.1110]	(247.4768)
		2937.7561	[2890.0079]	(2861.3257)	21.9992	[21.7841]	(21.6220)
2.8	3.2	30.2853	[30.0059]	(29.6528)	256.4838	[248.1398]	(246.9557)
		3034.2240	[2863.8060]	(2855.0773)	22.1844	[21.7227]	(21.5827)
2.7	3.3	30.6165	[29.8128]	(29.6310)	262.7594	[254.0821]	(246.6993)
		3190.1729	[2934.5583]	(2851.6886)	22.5270	[21.6823]	(21.5623)

μ_1	μ_2	$b_1 = 0.6,\ b_2 = 0.6$					
3.0	3.0	34.0029	[34.0043]	(34.0029)	287.2810	[287.2809]	(287.2807)
		3356.1648	[3356.1670]	(3356.1640)	24.7589	[24.7599]	(24.7589)
2.9	3.1	33.8964	[33.8920]	(33.8919)	286.6787	[286.1990]	(286.1989)
		3359.8954	[3342.9694]	(3342.8704)	24.7179	[24.6722]	(24.6721)
2.8	3.2	33.8695	[33.8182]	(33.8177)	288.3598	[285.4726]	(285.4719)
		3417.8618	[3334.8099]	(3333.8414)	24.7484	[24.6152]	(24.6148)
2.7	3.3	33.9229	[33.7806]	(33.7673)	292.5324	[285.2302]	(285.1057)
		3542.1691	[3329.7658]	(3329.1431)	24.8544	[24.5867]	(24.5749)

μ_1	μ_2	$b_1 = 1.0,\ b_2 = 0.6$					
3.0	3.0	41.5404	[40.7636]	(40.0954)	357.0158	[348.6593]	(346.5593)
		4214.7000	[4140.7580]	(4106.6073)	30.2169	[29.5749]	(29.0470)
2.9	3.1	40.6644	[40.6644]	(39.9110)	348.1653	[347.6797]	(345.0356)
		4102.9244	[4098.8879]	(4089.3129)	29.5719	[29.2867]	(28.8917)
2.8	3.2	40.2112	[40.2232]	(39.7419)	344.2166	[346.9783]	(343.8081)
		4081.4767	[4098.4158]	(4076.5796)	29.2034	[29.1903]	(28.7676)
2.7	3.3	39.9079	[40.1017]	(39.5828)	343.4990	[344.2199]	(342.8837)
		4129.1877	[4071.1735]	(4068.0153)	28.9314	[29.0386]	(28.6556)

μ_1	μ_2	$b_1 = 1.4,\ b_2 = 0.6$					
3.0	3.0	47.6867	[46.3303]	(44.5806)	414.3644	[401.9812]	(391.3966)
		4929.7099	[4731.6909]	(4684.8575)	34.6166	[33.2897]	(32.0610)
2.9	3.1	46.8509	[45.9270]	(44.2853)	403.0834	[396.7625]	(389.2539)
		4764.6524	[4720.4049]	(4663.2269)	33.8629	[32.9451]	(31.8185)
2.8	3.2	45.9587	[45.5870]	(44.0287)	394.2141	[395.0320]	(387.5789)
		4660.0618	[4683.4774]	(4647.0653)	33.2001	[32.7705]	(31.6168)
2.7	3.3	45.0704	[45.2369]	(43.8133)	387.7473	[390.7746]	(386.1563)
		4646.8262	[4674.1971]	(4633.0925)	32.6215	[32.7317]	(31.4558)

Table 2.22: *Comparative performance of the index heuristic, policy improvement and optimal policies for a range of average costs problems with two customer classes, where* $\Gamma = 0.85$.

$\Gamma = 0.85$

μ_1	μ_2	$b_1 = 0.4$, $b_2 = 1.0$					
3.0	3.0	37.0207	[36.0418]	(35.0446)	314.8620	[304.6801]	(300.3290)
		3691.7799	[3554.9916]	(3527.9525)	26.9651	[26.1866]	(25.3901)
2.9	3.1	37.7565	[36.2418]	(35.1745)	323.3715	[307.9613]	(300.6360)
		3853.9793	[3596.9406]	(3525.3484)	27.5212	[26.3768]	(25.4462)
2.8	3.2	38.6687	[36.5728]	(35.2783)	338.5075	[309.6795]	(300.7945)
		4093.3148	[3609.4973]	(3522.5615)	28.5005	[26.5758]	(25.4919)
2.7	3.3	39.4980	[37.0987]	(35.3785)	352.2128	[311.8569]	(300.9824)
		4351.1821	[3672.4643]	(3521.8065)	29.0928	[26.8251]	(25.5426)

μ_1	μ_2	$b_1 = 0.6$, $b_2 = 1.0$					
3.0	3.0	41.9629	[41.6114]	(41.1929)	360.1748	[356.9988]	(355.0841)
		4245.1603	[4195.7362]	(4191.2210)	30.5499	[30.3127]	(29.9601)
2.9	3.1	42.4739	[41.8532]	(41.2145)	368.2657	[358.4932]	(354.7344)
		4375.7007	[4233.0625]	(4183.0645)	31.0981	[30.5035]	(29.9343)
2.8	3.2	42.8950	[42.1364]	(41.2050)	375.9271	[362.5121]	(354.3359)
		4541.1381	[4254.7849]	(4175.8784)	31.4061	[30.7275]	(29.9120)
2.7	3.3	43.4683	[42.0219]	(41.2126)	386.3732	[365.0079]	(354.1029)
		4805.1652	[4307.3862]	(4172.1725)	32.0008	[30.9779]	(29.9109)

μ_1	μ_2	$b_1 = 1.0$, $b_2 = 1.0$					
3.0	3.0	50.3614	[50.3614]	(50.3614)	438.9745	[438.9749]	(438.9745)
		5224.5594	[5224.5629]	(5224.5594)	36.6299	[36.6299]	(36.6299)
2.9	3.1	50.2058	[50.1822]	(50.1822)	438.2655	[437.2252]	(437.2251)
		5234.1695	[5267.5390]	(5202.9502)	36.5759	[36.4913]	(36.4913)
2.8	3.2	50.2006	[50.0599]	(50.0593)	441.3994	[436.1664]	(436.0206)
		5342.1216	[5188.4425]	(5187.9847)	36.6611	[36.3976]	(36.3971)
2.7	3.3	50.3834	[49.9967]	(49.9938)	448.4333	[435.5324]	(435.3695)
		5531.2484	[5180.2694]	(5179.7542)	36.9234	[36.3503]	(36.3481)

μ_1	μ_2	$b_1 = 1.4$, $b_2 = 1.0$					
3.0	3.0	57.8989	[57.4880]	(57.0727)	508.7098	[504.4738]	(503.1854)
		6083.1006	[6055.3174]	(6030.0021)	42.0879	[41.7272]	(41.4195)
2.9	3.1	57.2406	[57.2238]	(56.8267)	502.3566	[503.1621]	(501.1417)
		6007.7254	[6009.9127]	(6006.1066)	41.5996	[41.5851]	(41.2419)
2.8	3.2	56.7790	[57.0922]	(56.6345)	499.7969	[502.1507]	(499.5896)
		6030.7278	[6003.3302]	(5988.9137)	41.3069	[41.4078]	(41.0813)
2.7	3.3	56.5057	[56.7048]	(56.4847)	501.9215	[499.3982]	(498.5791)
		6164.8989	[5979.0731]	(5978.6279)	41.1986	[41.1422]	(40.9728)

Table 2.23: *Comparative performance of the index heuristic, policy improvement and optimal policies for a range of average costs problems with two customer classes, where* $\Gamma = 0.85$.

$\Gamma = 0.85$

μ_1	μ_2	$b_1 = 0.4,\ b_2 = 1.4$					
3.0	3.0	43.5501	[41.2968]	(38.9238)	373.3550	[348.6520]	(338.6877)
		4397.2655	[4103.1832]	(4024.2545)	31.6619	[29.5239]	(27.9571)
2.9	3.1	45.1902	[41.6309]	(39.2199)	391.1769	[352.8731]	(340.0192)
		4675.6449	[4139.6842]	(4027.2878)	32.8870	[30.0002]	(28.0929)
2.8	3.2	46.1024	[41.6558]	(39.4577)	411.7731	[355.3721]	(340.7961)
		5080.8386	[4185.2733]	(4027.7345)	33.8360	[30.2411]	(28.2344)
2.7	3.3	47.7973	[42.5114]	(39.7166)	438.7666	[363.1619]	(341.6391)
		5572.8923	[4218.0627]	(4031.3198)	35.2065	[30.5822]	(28.3761)

μ_1	μ_2	$b_1 = 0.6,\ b_2 = 1.4$					
3.0	3.0	48.7847	[47.5052]	(46.2272)	424.9810	[410.4314]	(404.9537)
		5038.5555	[4863.4187]	(4824.0068)	35.5013	[34.5218]	(33.4506)
2.9	3.1	49.8317	[47.8888]	(46.4201)	437.8875	[412.4625]	(405.5212)
		5271.6520	[4931.8329]	(4821.0854)	36.2918	[34.7435]	(33.5468)
2.8	3.2	51.2335	[48.2209]	(46.5799)	457.9353	[418.5392]	(405.8545)
		5621.5704	[4940.3996]	(4818.1585)	37.5273	[35.0530]	(33.6108)
2.7	3.3	52.2945	[48.6743]	(46.7124)	475.8578	[421.0071]	(406.1084)
		5964.4071	[5035.1542]	(4817.1444)	38.5085	[35.3904]	(33.6932)

μ_1	μ_2	$b_1 = 1.0,\ b_2 = 1.4$					
3.0	3.0	58.3215	[58.2156]	(57.8846)	511.8683	[510.4789]	(509.3213)
		6113.5609	[6105.7769]	(6089.0284)	42.4209	[42.2752]	(42.0904)
2.9	3.1	58.9747	[58.5241]	(57.8745)	520.1239	[513.1484]	(508.5532)
		6259.2696	[6133.2153]	(6074.7558)	42.9695	[42.6057]	(42.0279)
2.8	3.2	59.3179	[58.8879]	(57.8156)	529.2031	[512.9002]	(507.6829)
		6482.1124	[6093.0257]	(6062.5008)	43.3961	[42.8915]	(41.9723)
2.7	3.3	59.8743	[59.3028]	(57.7724)	545.3752	[520.9987]	(507.2200)
		6808.9917	[6222.8459]	(6056.5213)	44.0100	[43.2170]	(41.9450)

μ_1	μ_2	$b_1 = 1.4,\ b_2 = 1.4$					
3.0	3.0	66.7200	[66.7200]	(66.7200)	590.6686	[590.6684]	(590.6684)
		7092.9521	[7092.9577]	(7092.9521)	48.5009	[48.5009]	(48.5009)
2.9	3.1	66.5151	[66.4725]	(66.4725)	589.8816	[588.2520]	(588.2519)
		7109.5959	[7155.7674]	(7063.0228)	48.4339	[48.3105]	(48.3105)
2.8	3.2	66.5412	[66.3008]	(66.3003)	594.6141	[586.7057]	(586.5694)
		7266.3912	[7042.5441]	(7042.1188)	48.5822	[48.1794]	(48.1791)
2.7	3.3	66.8441	[66.2084]	(66.2050)	604.8786	[585.7901]	(585.6327)
		7523.7200	[7121.4531]	(7030.3647)	48.9815	[48.1328]	(48.1077)

Table 2.24: *Comparative performance of the index heuristic, policy improvement and optimal policies for a range of average costs problems with two customer classes, where* $\Gamma = 0.85$.

2.5.3 Simulation study of average costs problems with five customer classes

We now look at some examples of the undiscounted admission control problems encountered in this chapter, where we have five service stations. In the two service station problems of Sections 2.5.1 and 2.5.2 it was possible to obtain a direct numerical comparison between costs incurred by our index heuristics and those incurred by an optimal policy. However this is not a reasonable computational goal for larger problems. The simulation study reported in Table 2.25 concern a collection of admission control problems involving five customer classes under the average cost criterion.

Table 2.25 contains the results of studies of ten problems with quadratic costs $(1-5,\, 1^{'}-5^{'})$ and five problems with quartic costs (1-5). All problems in this table have the exponential arrival and service time distributions associated with the two service station problem. Each of the problems with quadratic costs is characterised by three five-vectors and the system arrival rate namely, $\mathbf{b}$, $\mathbf{c}$, $\boldsymbol{\mu}$ and λ. Both $\mathbf{b}$ and $\mathbf{c}$ are vectors of cost coefficients such that the cost rate for service station k is given by

$$C_k(n) = b_k n + c_k n^2,\ \ 1 \leq k \leq 5, \tag{2.142}$$

while $\boldsymbol{\mu}$ is a vector of service rates with λ the arrival rate for the system. For example, for quadratic problem 1 we take $\mathbf{b} = (1.5, 1.2, 0.9, 0.6, 0.3)$, $\mathbf{c} = (0.2, 0.4, 0.6, 0.8, 1.0)$, $\boldsymbol{\mu} = (0.60, 1.50, 2.70, 3.90, 5.00)$ and $\lambda = 8.22$ with a resulting *Gamma*-value of 0.60. To obtain quadratic problems 2-5 we keep $\mathbf{b}$, $\mathbf{c}$ and λ fixed, but reassign $\boldsymbol{\mu}$ by means of a series of permutations. For example for problem 2 we take $\boldsymbol{\mu} = (1.50, 2.70, 3.90, 5.00, 0.60)$ and so on. We obtain quadratic problems $1^{'}$-$5^{'}$ respectively from 1-5 by rescaling λ to give a Γ-value of 0.85, while keeping other aspects fixed. We obtain quartic problems 1-5 from the corresponding quadratic problems upon replacing (2.142) by

$$C_k(n) = b_k n^3 + c_k n^4, \;\; 1 \leq k \leq 5.$$

In the body of Table 2.25 we have included estimates of the average costs incurred by the above problems under five service control heuristics, as follows: INDEX denotes our index heuristic for average costs while SQ routes the arriving customer at each decision epoch to whichever customer class has the shortest queue (and chooses among the candidate classes at random in the event of a tie). MYOPIC always routes the arriving customer to whichever station is currently incurring the smallest instantaneous cost rate. At each decision epoch, RANDOM chooses one of the service stations at random and routes a single customer to that station. When doing this we could not always allow the probability of a customer being sent to each queue to be equal as this could yield unstable queues. So to overcome this problem we calculated the upper bounds of each probability such that we had stable queues and then re-scaled to convert them into true probabilities (i.e. so that the sum of the five probabilities equalled one). In other words we took

$$p_k = \frac{\mu_k}{\lambda},$$

and then re-scaled by θ such that

$$\theta \sum_{k=1}^{5} p_k = 1. \tag{2.143}$$

The estimate of average cost is obtained in each case by Monte Carlo simulation. Typically, we allowed a "burn-in" period of around 10,000 time units in each case, followed by a period of around 15,000 time units during which costs were tracked. This was repeated around 50 times and the average costs (per unit time) were estimated. The corresponding standard errors are given in brackets in the table. The details of the mechanics of the simulations varied a little across the different cases in order to obtain standard errors which would enable meaningful comparisons between service policies to be made. For example, when we increased the Γ-value to 0.85 we had to increase the number of runs. Note that we did not have access to sufficient computer resources for satisfactory standard errors to be achieved for problems with quartic costs and a Γ-value of 0.85. This is why no such cases are reported in the table.

2.5.4 Comments

One can see that all the numerical evidence suggests that our index heuristic policy performs very well. We can see this because the index policy cost rate is usually close to the optimal cost rate or indeed, in the the five service stations examples, significantly better than the cost rates for alternative policies.

When looking at the discounted data in tables 2.1 - 2.16 one can see that the costs increase when the initial state indicates that more customers are present initially, when the cost functions are of a higher power and when we increase the arrival rate as we would expect. The actual performance of the index policy considered in the chapter seems to be very promising, coming close to optimality in many examples. The alternative index policy (which allows a negative number of customers) also performs well. The ideas on which this is based could possibly be an option for other models where the main index put forward in this chapter could not be obtained for some reason. These data seems to suggest that when moving to the higher arrival rates the index policy can still return values close to optimal. From these data there does seem to be some evidence to suggest that when we make the servers increasingly distinct (by altering their service rates) then the index policy performs sightly less well. However as one can see the percentage sub-optimality of such cases remains at a low level.

The numerical data for the two server average cost problem seen in tables 2.17 -

Quadratic Costs	INDEX	LQ	MYOPIC	RANDOM
1	5.6563	9.7632	8.0407	24.8424
	(0.0091)	(0.0123)	(0.0097)	(0.0968)
2	5.6109	7.9092	8.8725	24.8504
	(0.0085)	(0.0091)	(0.0108)	(0.1446)
3	5.4177	6.9344	8.5727	24.7059
	(0.0078)	(0.0102)	(0.0100)	(0.1288)
4	5.3544	7.0292	8.0357	24.8651
	(0.0078)	(0.0099)	(0.0104)	(0.1045)
5	5.5034	7.9973	7.8512	24.7684
	(0.0071)	(0.0092)	(0.0118)	(0.0995)
1'	26.3657	31.9143	29.5310	233.2947
	(0.1466)	(0.1241)	(0.1299)	(1.9239)
2'	24.2011	29.9548	28.9355	231.7829
	(0.1492)	(0.1312)	(0.1344)	(2.9537)
3'	22.0233	27.8126	28.9351	233.0175
	(0.1265)	(0.1390)	(0.1295)	(3.0151)
4'	21.4462	27.5006	28.7909	236.3840
	(0.1224)	(0.1527)	(0.1292)	(2.4817)
5'	22.3605	28.8711	28.7192	236.4076
	(0.1418)	(0.1147)	(0.1355)	(2.3820)
Quartic Costs				
1	16.2846	39.6315	25.8457	1006.2201
	(0.0986)	(0.1333)	(0.1065)	(16.1015)
2	15.4626	25.6113	29.6577	6242.1436
	(0.0931)	(0.0986)	(0.1028)	(15.5941)
3	15.1293	20.0892	28.7645	987.4523
	(0.0753)	(0.0980)	(0.1054)	(21.5152)
4	15.0922	20.7926	26.2705	1012.2977
	(0.0931)	(0.0938)	(0.1012)	(25.8314)
5	15.5607	26.5092	25.0162	994.6270
	(0.0850)	(0.0964)	(0.1126)	(16.7460)

Table 2.25: *Comparative performance of the index heuristic and other control rules for a range of average costs problems with five service stations.*

2.24 shows that the cost rates will increase if the cost coefficients increase or if the order of the cost function is increased. However, the index heuristic put forward in this chapter seems to do consistently well. In many cases this index heuristic seems to perform better than the policy improvement index, even though the policy improvement index is allowed to consider initially the system as a whole. This is not required by the index heuristic. This means that the policy improvement approach will be much more problematic for larger numbers of stations and also if the number of stations altered (due to the addition of a new station etc). There is possibly some evidence from the data to suggest that as the servers become more distinct that the proposed index policy does not perform quite as well (especially when we have higher arrival rates) but do note that the percentage sub-optimality remains small in the vast majority of cases.

Table 2.25 show the simulation data for our proposed index heuristic and some other control rules for a range of problems with five service stations. These data show that as the arrival rate is increased or the order of the cost functions is increased the cost rates also increase. The data in this table suggest that our index heuristic performs very well, significantly better than all the other control rules considered.

Hence all the numerical data suggests that the index policy presented in this chapter perform very well under a variety of models. Hence my conclusion is that this would be a good policy to use to minimise cost rates with a small amount of computational effort as the indexable nature means that it is not difficult to implement.

Chapter 3

Service Control Problems

3.1 Introduction

We consider a multi-class queueing system in which customers from classes $\{1, 2, \ldots, K\}$ receive service. An important decision within a multi-class queueing system is which customer class should be served at any given time. If there are customers from different classes present we must ask the question, "by serving which class do we gain the most?" - i.e. serving which class, at this time, reduces our costs or increases our rewards by the highest amount. The aim within this chapter is to find a dynamic policy which chooses between the customer classes awaiting service to achieve results near some defined optimal performance.

In this chapter we build from the work of Ansell *et al.* (2003a), in which the assumption that customer service times were independent and exponentially distributed was made. However here we consider the much more challenging case of general service time distributions. Such general service distributions considerably complicate the analysis, however the results that we achieve will be more widely applicable. The first thing to note is that without the exponential service distribution assumption we no longer have the benefit of its memoryless property. As a result we shall consider non-preemptive service policies only - i.e. once a customer has started service they must complete that service before another can be served. Note that most practical problems have this non-preemptive character. Without this restriction to non-preemptive policies we could possibly have a number of partially served customers still waiting for service at any given time. To take account of this via a suitably extended state space would cause this problem to be yet more challenging.

Section 3.2 considers the general set up of the service control problem of interest and describes both discounted and undiscounted formulations. The work encompasses a range of modelling possibilities. This section then moves on to define a relaxation of the problem and takes Lagrangian approach to find the structure of its optimal solution. We propose that a heuristic derived from the optimal solution to the relaxed problem will provide a "good" policy for our original problem. Section 3.3 investigates the discounted version of our problem in more detail, looking at the required solution for a single class problem derived from the Lagrangian relaxation in which a charge for service is incurred. In Section 3.4 we then derive an appropriate index for the discounted problem, with a corresponding index for the undiscounted problem derived as a limit. We then conclude this chapter by reporting some results of a numerical investigation into the performance of the Whittle index policy. This can be found in Section 3.5. Within this investigation we consider the two server undiscounted case but the main focus is on the average costs scenario. In the average costs case we consider not only the two server example using methods of dynamic programming but also use simulation techniques to consider systems with a larger number of servers. Simulation is required since direct numerical comparisons is not a reasonable computational goal for larger problems.

3.2 The multi-class service control system

Recall that we are considering a multi-class queueing system in which customers from classes $\{1, 2, \ldots, K\}$ receive service. Our goal within this chapter is to allocate service to the waiting customers to minimize some measure of expected holding cost over an infinite horizon. We make the assumption that the arrivals into the system follow K independent Poisson processes where each class can have a different arrival rate, denoted λ_k for class k. As we have already said in the introduction, we assume general service distributions, so in practice we can select distributions which best fits our application. Each class k customer has a service time which we denote as S_k and a corresponding distribution function, G_k. The service times are independent for different customers and identically distributed for customers within a single class. We suppose that the system is stable in that work coming into the system can be handled by the single non-idling server, so that we never observe infinite queue lengths, i.e. we require that

$$\rho \equiv \sum_{k=1}^{K} \lambda_k E(S_k) < 1. \tag{3.1}$$

As alluded to earlier we consider both discounted and average cost (undiscounted) criteria. In order to set this problem up formally we need to introduce and explain the notation we shall use.

When we refer to the state of customer class k at time t we are talking about the length of the class k queue at time t, which includes any customers in service. We write this state as $N_k(t)$, $1 \leq k \leq K$, $t \in \mathbb{R}^+$. The state of the system at time t is given by $\mathbf{N}(t) = \{N_1(t), N_2(t), N_3(t), \ldots, N_K(t)\}$ the vector of queue lengths, $t \in \mathbb{R}^+$.

The decision epochs occur at all service completion times which do not result in an empty system together with all the times of arrivals at an empty system. These are the only times when a decision can be made concerning who to serve next in our class of non-preemptive policies.

We use a_k to denote the action of allocating service to a class k customer, $1 \leq k \leq K$. At each decision epoch t, the controller chooses an action a_k from the set of k for which $N_k(t) \geq 1$. It is the choice of which action to take at each decision epoch we are seeking, in this chapter, in order to minimise some measure of expected costs.

Suppose that t is a decision epoch, that system state $\mathbf{N}(t) = \mathbf{n}$ with $n_k > 0$, and that action a_k is taken at t. The next decision epoch will occur at the end of this class k customer service, $t + S_k$, where $S_k \sim G_k$, provided the system is nonempty at this time. The system state then has a probability distribution given by,

$$\begin{aligned} P[\mathbf{N}(t+S_k)^+ = \mathbf{n} - \mathbf{1}^k + \mathbf{m}] &= E_{S_k}\Big\{P(m_1 \text{ class 1 arrivals in time } S_k) \\ &\quad \times P(m_2 \text{ class 2 arrivals in time } S_k) \\ &\quad \times \ldots \times P(m_K \text{ class } K \text{ arrivals in time } S_k)\Big\} \\ &= \int_0^\infty \Big\{\prod_{j=1}^{K} \frac{(\lambda_j t)^{m_j}}{m_j!} e^{-\lambda_j t}\Big\} dG_k, \quad \mathbf{m} \in \mathbb{N}^K, \quad (3.2) \end{aligned}$$

since arrivals occur in independent Poisson streams with rates λ_j, $1 \leq j \leq K$. Note that in (3.2), $\mathbf{1}^k$ denotes a K vector whose k^{th} component is 1, with zeros elsewhere and also that the processing of the class k customer which begins at time t is non-preemptive.

In the *discounted costs* version of this queueing control problem we say discounted costs are incurred by class k with rate

$$\alpha C_k(N_k(t)),$$

at time t. The cost functions $C_k : \mathbb{N} \to \mathbb{R}^+$ are assumed to be increasing, convex and bounded above by some polynomial of finite order (in order to ensure that all expectations taken in this chapter will be finite) and with $C_k(0) = 0$, $1 \leq k \leq K$. We have already stated that the costs are additive across the classes and so the system incurs costs at rate

$$\sum_{j=1}^{K} \alpha C_j(N_j(t)), \tag{3.3}$$

at time t.

A policy u is a rule for choosing actions in light of the history of the process to date. We use $\mathcal{U}$ to denote the set of all such policies which are non-idling for the single server. Our goal is to find a policy in order to achieve the best performance (i.e. minimum cost) of the system. In this case we take our performance measure to be the total discounted costs incurred over an infinite horizon, and we wish to find a policy to minimize this measure. We write

$$\mathbf{V}(\mathbf{m}, \alpha) = \inf_{u \in \mathcal{U}} E_u \left[\int_0^\infty \sum_{k=1}^{K} \alpha C_k(N_k(t)) e^{-\alpha t} | \mathbf{N}(0) = \mathbf{m} \right], \tag{3.4}$$

for the value function associated with this policy. Note that the α multiplier has been introduced into the holding cost rate in (3.3) and (3.4) to guarantee that $\mathbf{V}(\mathbf{m}, \alpha)$ remains finite and approaches the minimum average cost per unit time for the system in the limit as α approaches 0 see (3.6) below. As has been previously mentioned, this limit is central to the consideration of the average cost (undiscounted) problem of interest to us. Further justification for the inclusion of this α multiplier can be seen in Section 3.4. Plainly, inclusion of the multiplier will have no impact on the optimal policy in (3.4).

The *average cost* version of the multi-class queueing problem of interest is expressed via the equation

$$\mathbf{V}^{OPT} = \inf_{u \in \mathcal{U}} \tilde{E}_u \Big\{ \sum_{k=1}^{K} C_k(N_k) \Big\}. \tag{3.5}$$

In (3.5) $\tilde{E}_u$ is the expectation taken with respect to the steady-state distribution of the system under policy u. From standard results in dynamic programming, we have that

$$\lim_{\alpha \to 0} \mathbf{V}(\mathbf{m}, \alpha) = \mathbf{V}^{OPT}. \tag{3.6}$$

Using the relation in (3.6) we can develop heuristics for the average cost problems as limits ($\alpha \to 0$) of the corresponding heuristics for discounted costs.

Over the next few pages we investigate the discounted costs version of the multi-class problem. We know from stochastic dynamic programming (DP) theory that for the discounted costs problem, a stationary optimal policy exists (i.e. a policy that makes decisions based on the current state only). The value function of this policy will satisfy the DP optimality equations, see Puterman (1994). In this multi-class queueing control problem a pure DP approach will be computationally intractable for problems of any reasonable size and is unlikely to be insightful. So we adopt the method used by Whittle (1988)

To develop the ideas needed for the application of Whittle's approach we introduce the following performance measures for policy u:

$$\begin{aligned} x_{k,n}^{a,u}(\mathbf{m}) &= \text{the expected amount of discounted time spent in state } n \text{, taking action} \\ &\quad a_k, \text{"serve class } k\text{", from initial state } \mathbf{m}\text{, when under control policy } u \\ &= E_u\Big[\int_0^\infty I\{a_k(t) = a, N_k(t) = n\} e^{-\alpha t} dt | \mathbf{N}(0) = \mathbf{m}\Big]. \end{aligned} \tag{3.7}$$

In (3.7), $\mathbf{m} \in \mathbb{N}^K$, $n \in \mathbb{N}$, $1 \leq k \leq K$ and we have written $a_k(t)$ for the action (either a = serve (active) or b = do not serve (passive)) applied to queue k at time t. Also $I\{.\}$ is the indicator function, so

$$I\{a_k(t) = a, N_k(t) = n\} = \begin{cases} 1 & \text{if, at time } t\text{, we have } n \text{ class } k \text{ customers present} \\ & \text{and we choose to serve class } k, \\ 0 & \text{otherwise.} \end{cases}$$

Note that the passive action b is applied to class k whenever the active action a is not applied. We now define a similar performance measure for the passive action, b, i.e.

$$x_{k,n}^{b,u}(\mathbf{m}) = E_u\Big[\int_0^\infty I\{a_k(t) \neq a, N_k(t) = n\}e^{-\alpha t}dt|\mathbf{N}(0) = \mathbf{m}\Big].$$

Using these performance measures we can re-write our discounted cost function (3.4), as

$$\mathbf{V}(\mathbf{m}, \alpha) = \inf_{u \in \mathcal{U}} \sum_{k=1}^{K} \sum_{n \in \mathbb{N}} \alpha C_k(n)\Big\{x_{k,n}^{a,u}(\mathbf{m}) + x_{k,n}^{b,u}(\mathbf{m})\Big\}. \tag{3.8}$$

We have said that for all policies in $\mathcal{U}$ whenever there are customers present in the system the server must be active, i.e. service must be offered whenever the system in non-empty. Hence we have that

$$\begin{aligned}\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{a,u}(\mathbf{m}) &= \text{the expected amount of discounted time spent in the system} \\ &\quad\ \text{taking the active action} \\ &= E_u\Big[\int_0^\infty I\{\mathbf{N}(t) \neq \mathbf{0}\}e^{-\alpha t}dt|\mathbf{N}(0) = \mathbf{m}\Big],\end{aligned}$$

where $\mathbf{0}$ is the zero K vector. We now develop a relaxation of the problem in (3.8) by first noticing that for all policies in $\mathcal{U}$,

$$\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{a,u}(\mathbf{m}) \quad \text{is policy invariant.} \tag{3.9}$$

This is because the quantity in (3.9) involves only the discounted time and does not involve the holding costs. Also within every policy in $\mathcal{U}$ only the order that the customers are served is affected and the server will serve **all** of the customers in the system. Obviously, regardless of the order the customers are served, the expected discounted time to serve them all will remain constant. So we can see that the duration of the first busy period and of all subsequent busy periods have probability distributions which do not depend upon the control policy u. Hence (3.9) is indeed policy invariant. It is also true however that the total discounted cost to the system (i.e. the value function) will vary as we change the control policy.

From queueing theory we know that in the long run the proportion of time a system is non-empty is

$$\rho = \sum_{k=1}^{K} \lambda_k E(S_k).$$

In fact it holds that

$$\begin{aligned}\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{a,u}(\mathbf{m}) &= \int_0^\infty \rho e^{-\alpha t} dt + O(1) \\ &= \frac{\rho}{\alpha} + \Theta(\mathbf{m}, \alpha) \text{ where } \Theta(\mathbf{m}, \alpha) = O(1). \end{aligned} \tag{3.10}$$

We now consider a relaxed version of the stochastic optimization problem in (3.8) obtained by expanding the admissible class of policies to the set in which *any* number of non-empty customer classes may be served at any time. Note that we still must maintain the non-preemptive nature of service, so any service once started must be completed. We will call this new policy class $\bar{\mathcal{U}}$. We also extend $\bar{\mathcal{U}}$ to include randomisations over such policies. However we shall only allow those policies in $\bar{\mathcal{U}}$ which satisfy (3.10). This constraint will ensure that *on average* (in the discounted sense of (3.10)) one class is served at each decision epoch. We call this *Whittle's relaxation* and write

$$\begin{aligned}\underline{\mathbf{V}}(\mathbf{m}, \alpha) &= \inf_{u \in \bar{\mathcal{U}}} \sum_{k=1}^{K} \sum_{n \in \mathbb{N}} \alpha C_k(n) \Big\{ x_{k,n}^{a,u}(\mathbf{m}) + x_{k,n}^{b,u}(\mathbf{m}) \Big\} \\ \text{subject to} \quad & \sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{a,u}(\mathbf{m}) = E_u \Big[\int_0^\infty J(t) e^{-\alpha t} dt | \mathbf{N}(0) = \mathbf{m} \Big] \\ & \qquad = \alpha^{-1} \rho + \Theta(\mathbf{m}, \alpha). \end{aligned} \tag{3.11}$$

In the above expression, $J(t)$ denotes the number of customer classes served at time t and the constraint (3.11) delimits the set of allowable policies within $\bar{\mathcal{U}}$. Obviously $\mathcal{U}$ is contained within this new admissible class of policies, so as a consequence we have that $\underline{\mathbf{V}}(\mathbf{m}, \alpha) \leq \mathbf{V}(\mathbf{m}, \alpha)$. Also for any policy within $\mathcal{U}$ we have $J(t) = I\{\mathbf{N}(t) \neq 0\}$, $t \in (0, \infty)$. But now we proceed to the above minimization problem with constraint (3.11). This will not be easy to work with directly so we use a Langrangian approach to find the structure of the optimal solution to

Whittle's relaxation. Hence we accommodate constraint (3.11) by incorporating a Langrange multiplier W to obtain the minimization problem

$$\begin{aligned}\underline{\mathbf{V}}(\mathbf{m},\alpha,W) &= \inf_{u\in\bar{\mathcal{U}}}\Big[\sum_{k=1}^{K}\sum_{n\in\mathbb{N}}\alpha C_k(n)\Big\{x_{k,n}^{a,u}(\mathbf{m})+x_{k,n}^{b,u}(\mathbf{m})\Big\}\\ &\qquad -W\Big\{\alpha^{-1}\rho+\Theta(\mathbf{m},\alpha)-\sum_{k=1}^{K}\sum_{n\in\mathbb{N}}x_{k,n}^{a,u}\Big\}\Big]\\ &= \inf_{u\in\bar{\mathcal{U}}}\left[\sum_{k=1}^{K}\sum_{n\in\mathbb{N}}\{\alpha C_k(n)+W\}x_{k,n}^{a,u}(\mathbf{m})+\sum_{k=1}^{K}\sum_{n\in\mathbb{N}}\alpha C_k(n)x_{k,n}^{b,u}(\mathbf{m})\right]\\ &\qquad -W\Big\{\alpha^{-1}\rho+\Theta(\mathbf{m},\alpha)\Big\}. \end{aligned}\tag{3.12}$$

Note we can see here that the last term in (3.12) will play no part in the choice of the optimal control policy u. We can also see from (3.12) that the W plays the economic role of a *constant charge for service*. Recall that the optimization problem that we have in (3.12) involves a control which can activate any number of non-empty customer classes. This problem is naturally decoupled into K single-class subproblems, expressed by

$$\underline{\mathbf{V}}(\mathbf{m},\alpha,W)=\sum_{k=1}^{K}V_k(m_k,\alpha,W)-W\{\alpha^{-1}\rho+\Theta(\mathbf{m},\alpha)\}.\tag{3.13}$$

In (3.13), $V_k(m_k,\alpha,W)$ is the minimized total holding and service charge costs incurred by customer class k only, the minimization being taken over all (non-preemptive) policies for choosing between actions a and b for that class only. In other words we have

$$V_k(m_k,\alpha,W)=\inf_{u\in\bar{\mathcal{U}}_{\|}}\Big[\sum_{n\in\mathbb{N}}\{\alpha C_k(n)+W\}x_{k,n}^{a,u}(m_k)+\sum_{n\in\mathbb{N}}\alpha C_k(n)x_{k,n}^{b,u}(m_k)\Big],$$

where $\bar{\mathcal{U}}_{\|}$ is the set of all non-preemptive policies for choosing between actions a and b for this class only. We will denote this single class problem $(k,\alpha,W), W\in\mathbb{R}, 1<k\leq K$.

We later show (see Comment 2 on page 147) that we can choose the value of the multiplier $W=W(\mathbf{m},\alpha)$ in order to ensure that the optimal policy for the

Lagrangian relaxation in (3.12) meets constraint (3.11). So we have that

$$\mathbf{V}\big(\mathbf{m}, \alpha, W(\mathbf{m}, \alpha)\big) \quad = \quad \underline{\mathbf{V}}(\mathbf{m}, \alpha). \tag{3.14}$$

Hence the optimal policy for the Lagrangian relaxation in (3.12) with $W = W(\mathbf{m}, \alpha)$ satisfies the constraint in (3.11) and solves Whittle's relaxation.

So our progression through this problem will be as follows:

- Find the optimal policies for the K single-class subproblems in (3.13), which will be dependent on the value of W.
- Combine these single-class optimal policies into the required optimal policy for the corresponding multi-class problem in (3.12).
- Find the value $W = W(\mathbf{m}, \alpha)$ which ensures the constraint (3.11) is met and hence obtain the optimal policy for Whittle's relaxation in (3.11).

Hence the first issue that needs to be addressed concerns the optimal policies for the single class problems $(k, \alpha, W), 1 \leq k \leq K, W \in \mathbb{R}$. As in the previous chapter, the solutions are simple because the single class problems have the condition of *indexability.* To describe this condition, we again use $\Pi_{k,\alpha}(W)$ to denote the set of queue lengths m for which the passive action b is optimal in the single class problem (k, α, W). We recall Definitions 1 - 3 from Section (2.2).

Definition 1

Customer class k *α-indexable* if $\Pi_{k,\alpha}(W) : \mathbb{R} \to 2^{\mathbb{N}}$ is increasing, namely

$$W_1 > W_2 \implies \Pi_{k,\alpha}(W_1) \supseteq \Pi_{k,\alpha}(W_2), \tag{3.15}$$

Should we have α-indexability for class k, the idea of an α-index for state (i.e. queue length) m as the minimum service charge which makes the passive action optimal there is a natural one.

Definition 2

When customer class k is α-indexable, the *Whittle α-index* for class k in state m is given by

$$W_{k,\alpha}(m) = \inf\{W : m \in \Pi_{k,\alpha}(W)\}, m \in \mathbb{Z}^+. \tag{3.16}$$

It will now follow that if each customer class k is α-indexable, Whittle's relaxation in (3.11) is solved by a policy in which a decision is taken to serve customer class k at each decision epoch t for each (k, α, W) whenever $W_{k,\alpha}\{N_k(t)\} > W(\mathbf{m}, \alpha)$ and not to serve k whenever $W_{k,\alpha}\{N_k(t)\} < W(\mathbf{m}, \alpha)$, for all choices of k, t. Should $W_{k,\alpha}\{N_k(t)\} = W(\mathbf{m}, \alpha)$ then some randomisation between the two actions will be appropriate. Note that the constraint (3.11) will ensure that on average we only serve one customer at any given time.

We now follow Whittle (1988) in arguing that the index-like nature of solutions to the relaxation in (3.11) makes it reasonable to propose an *index heuristic* for our original discounted costs problem in (3.4) and (3.8) when all customer classes are α-indexable. This heuristic will be structured as in (3.19) with index functions recovered from Definition 2. Note that under this definition it is natural to interpret $W_{k,\alpha}(m)$ as a *fair charge* for serving customer class k in state m. The derived heuristic then always serves that class for which the fair charge for service is highest. Following the discussion about the average costs version in Section 3.2, we develop an index heuristic for average cost problems as the limit policy ($\alpha \to 0$) of the index heuristics for discounted costs.

Definition 3

If customer class k is α-indexable for all $\alpha > 0$ then the *average cost Whittle index* for state m is given by

$$W_k(m) = \lim_{\alpha \to 0} W_{k,\alpha}(m), \ m \in \mathbb{Z}^+, \tag{3.17}$$

when the above limit exists.

Note that the inclusion of the α multiplier in the holding cost rates in the discounted problem guarantees that the limits in (3.17) exist and yield sensible indices. To see why, revisit the Langrangian in (3.12). As policy u varies within the stable policies in $\bar{\mathcal{U}}$ it is known from standard MDP theory that the holding cost component of (3.12) will vary by amounts which are $O(1)$. However, it must also be true for such policies that

$$\sum_{k=1}^{K}\sum_{n\in\mathbb{N}} x_{k,n}^{\alpha,u}(\mathbf{m}) = \alpha^{-1}\rho + O(1), \tag{3.18}$$

and hence, for any finite W, varying u can only change the service charge component of (3.12) by $O(1)$. It is this balancing of the contributions to the total cost in (3.12) which guarantees the good behavior of the limits in (3.17).

Taking our cue from the above discussion, in the next section we study the single class problems (k, α, W). We shall establish α-indexability and derive α-indices and the average cost indices which are appropriate for out service control problems.

3.3 The Discounted Problem

As previously mentioned we firstly consider the discounted service control system in which future costs are discounted with time according to the rate α. We know of two special cases of this queueing control problem which have previously been studied and which can be solved to optimality by simple index policies.

i) The **batch case** with discounted costs can be solved using a multi-armed bandit model as in Gittins (1989). In this system all arrival rates are zero and the goal is to serve to completion all the customers present at time 0 (i.e. to empty the system) to minimize total expected discounted costs. In the Gittins (1989) paper the batch case was indeed formulated as a *multi-armed bandit* *problem* and a Gittins index policy was shown to be optimal.

ii) The case in which **holding costs** are **linear** in the queue lengths and discounted over time was first solved by Harrison (1975). This linear cost assumption allows an analysis at the level of the individual customer (each carrying their own holding cost rate) rather than at the level of the customer class. The linear cost problem was later formulated as a branching bandit problem for which Gittins index policies are also know to be optimal; see Bertsimas and Niño Mora (1996).

In both of these special cases the optimal policy is known to be of index form. This means that there exists K index functions,

$$W_{k,\alpha} : \mathbb{N} \to \mathbb{R}^+, \quad 1 \leq k \leq K,$$

such that at all decision epochs an optimal policy chooses to serve a customer from the maximal index class, i.e.

$$u^*\{\mathbf{N}(t)\} = a_k \implies W_{k,\alpha}\{N_k(t)\} = \max_{1\leq j\leq K} W_{j,\alpha}\{N_j(t)\}, \quad \text{where } u^* \text{ is optimal.} \tag{3.19}$$

We see that our discounted Whittle index policy leads us to the same optimal index policy as in the special discounted problem considered in (i) and (ii).

As I have noted above to obtain Whittle's indices for the original discounted cost problem in (3.4) and (3.8) we must initially look at the single class problem (k, α, W).

3.3.1 The single class system with a charge for service

In this section we study the single class problems (k, α, W), so it will be notationally convenient to drop the class identifier k. The problem we look at is one of a single server who is able to serve a single customer from the given class at any time. However there is a charge for the server's work and we have the option to not serve any customers if we believe it more cost effective to do so. We maintain the non-preemptive structure, so once a service has started on a customer it will continue until that service is complete. There are also holding cost charges incurred at a rate which is assumed increasing convex in the number of customers in the system. For this single class of customers we have $M/G/1$ dynamics. Hence arrivals form a Poisson(λ) stream. We use S to denote a generic service time with associated distribution function G. We do as always require that $\lambda E(S) < 1$ for stability. We can view this system pictorially in Figure 3.1. The goal is to choose

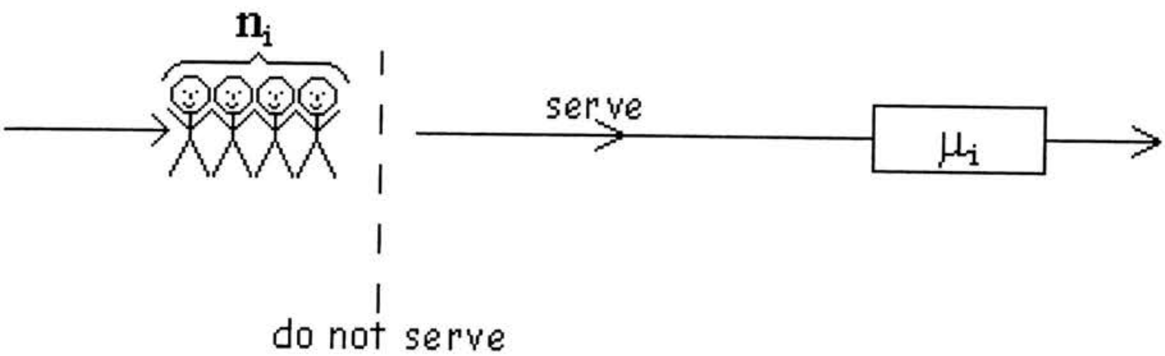

Figure 3.1: The options when considering a single service station.

how and when to deploy the server to minimize the the sum of the costs incurred in holding customers in the system and those incurred in paying for service. We formulate this problem as a Semi Markov Decision Process (SMDP) as follows:

(a) We use $N(t)$ to denote the state of the system at time $t \in \mathbb{R}^+$, i.e. the number of customers in the system. Decision epochs will occur at all service completion times which do not result in an empty system and at all times when we are in the passive mode (i.e. not serving) and we observe a customer arrival. At each decision epoch we must decide whether to take action a (active) or b (passive), where the active action a is the choice to serve a waiting customer through to completion and the passive action b is the choice not to serve. If t is a decision epoch we can see that the next epoch will occur at time $t+S$ if we choose action a, and time $t+X$ if we choose action b, where X is the time until the next customer arrival. By standard theory $X \sim \exp(\lambda)$ since the arrivals follow a Poisson(λ) process. According to standard $M/G/1$ dynamics we have that

$$\begin{aligned} P[N\big((t+S)^{+}\big) = m+n-1 | N(t) = m, a] &= E_S\left[\frac{(\lambda S)^n}{n!}e^{-\lambda S}\right], m \in \mathbb{Z}^{+}, n \in \mathbb{N} \\ &= \int_0^{\infty} \frac{(\lambda t)^n}{n!}e^{-\lambda t} dG, m \in \mathbb{Z}^{+}, n \in \mathbb{N} \end{aligned}$$

since the above is just the probability that we have n arrivals between t and $t+S$. We also know that

$$P[N\big((t+X)^{+}\big) = m+1 | N(t) = m, b] \quad = \quad 1, m \in \mathbb{N}. \tag{3.20}$$

This is evident since if we take this passive action b at the time t decision epoch, that means we are not serving. So the time of the next customer arrival, $t+X$, will be when our next decision epoch occurs, at which point we will obviously have $m+1$ customers. Note that the passive action is the only admissible action when $N(t) = 0$.

(b) We denote by $C : \mathbb{N} \to \mathbb{R}^{+}$ our increasing convex holding cost rate function for the class concerned. Then we can see that when we have n customers present in the system, our discounted costs will be incurred at rates

$$\alpha C(n) + W \quad \text{while the server is serving, and}$$

$$\alpha C(n) \qquad \text{while the server is not serving.}$$

In the above, α and W are positive constants. These rates are as in (3.12) above. Hence W is the rate charged for service, while $\alpha C(n)$ is the holding cost rate when there are n customers in the system, where recall that $C(0) = 0$.

(c) A policy is a rule for choosing between the two actions a or b in light of the history of the system to date. We can write the total expected cost incurred under policy u from initial state m as

$$V_u(m, \alpha, W) \;=\; E_u\Big[\int_0^\infty \{\alpha C(N(t)) + W I(t)\} e^{-\alpha t} dt | N(0) = m\Big]. \quad (3.21)$$

In (3.21) $I(t)$ is the indicator function

$$I(t) = \begin{cases} 1, & \text{if the server is active at time t} \\ 0, & \text{otherwise, } t \in \mathbb{R}^+ \end{cases}$$

It is clear that the immediate goal of analysis is to find the policy which will minimize the cost in (3.21). We denote the value of this minimized total cost to be

$$V(m, \alpha, W) = \inf_u \{V_u(m, \alpha, W)\}. \quad (3.22)$$

This is the problem we denoted by (k, α, W) in Section 3.2, where k is the class identifier (now dropped).

Recall the central idea of stochastic DP on page (8) of Section 1.2. This indicates the existence of an optimal policy which is stationary (i.e. makes decisions in light of the current state only). Also from general theory we know that the value function of this optimal policy will satisfy the DP optimality equations; see (3.24). In this simple single class, single server system we know that the decision in any state m is between taking action a (until the next service completion - as we have non-preemptive controls) or action b (until the next arrival). Now we can see that, if we are in state m and the policy u takes the *passive* action now and acts optimally from the next decision epoch onwards, then the total expected cost under policy u can be disaggregated into the discounted cost until the next arrival plus the discounted cost from state $m+1$. This total cost will be

$$\begin{aligned} &= \alpha C(m) E\Big(\int_0^X e^{-\alpha t} dt\Big) + V(m+1, \alpha, W) E(e^{-\alpha X}), \\ &= C(m) E\big(1 - e^{-\alpha X}\big) + V(m+1, \alpha, W) E(e^{-\alpha X}) \\ &= \frac{\alpha C(m)}{\alpha + \lambda} + \frac{\lambda V(m+1, \alpha, W)}{\alpha + \lambda}, \text{ since } E(e^{-\alpha X}) = \int_0^\infty e^{-\alpha x} \lambda e^{-\lambda x} dx = \frac{\lambda}{\alpha + \lambda}. \end{aligned}$$

However if we are in state m and the policy u says take the *active* action, then acts optimally from the next decision epoch, the total expected cost under policy u can be disaggregated into the discounted cost until the next service completion plus the discounted cost from the state at that conclusion of service. This cost will be

$$\begin{aligned} &= \tilde{C}(m,\alpha) + WE\Big(\int_0^S e^{-\alpha t}dt\Big) + \sum_{n=0}^{\infty}\int_0^{\infty}\frac{(\lambda t)^n}{n!}e^{-\lambda t}e^{-\alpha t}V(m+n-1,\alpha,W)dG \\ &= \tilde{C}(m,\alpha) + \frac{WE(1-e^{-\alpha S})}{\alpha} + \sum_{n=0}^{\infty}\int_0^{\infty}\frac{(\lambda t)^n}{n!}e^{-(\alpha+\lambda)t}V(m+n-1,\alpha,W)dG \end{aligned}$$

Note that $\tilde{C}(m,\alpha)$ is the holding costs incurred during a single service completion beginning at time 0 in state m, which we write as

$$\tilde{C}(m,\alpha) = E\Big[\int_0^S \alpha C(N(t))e^{-\alpha t}dt|N(0)=m,a\Big], \quad m \in \mathbb{Z}^+. \tag{3.23}$$

Hence we can see that the value function $V(.,\alpha,W)$ will choose the option in order to minimize these expected costs. Hence we obtain the optimality equation

$$\begin{aligned} V(m,\alpha,W) &= \min\Big\{\frac{\alpha C(m)}{\alpha+\lambda} + \frac{\lambda V(m+1,\alpha,W)}{\alpha+\lambda}; \tilde{C}(m,\alpha) + \frac{WE(1-e^{-\alpha S})}{\alpha} \\ &\quad + \sum_{n=0}^{\infty}\int_0^{\infty}\frac{(\lambda t)^n}{n!}e^{-(\alpha+\lambda)t}V(n+m-1,\alpha,W)dG\Big\}, m \in \mathbb{Z}^+. \end{aligned} \tag{3.24}$$

The analysis becomes a little cleaner if we substitute

$$\begin{aligned} \mathcal{V}(m,\alpha,W) &= V(m,\alpha,W) - W\int_0^{\infty}e^{-\alpha t}dt, \quad m \in \mathbb{N} \\ &= V(m,\alpha,W) - \frac{W}{\alpha}, \quad m \in \mathbb{N} \end{aligned} \tag{3.25}$$

in (3.24). We can see that $\mathcal{V}(m,\alpha,W)$ is the value function for an equivalent decision process but where the cost rate for the active action a is $\alpha C(n)$, and for the passive action b is $\alpha C(n) - W$. So now the W has an interpretation as a *subsidy for passivity.* Using the identity (3.25) in (3.24), we obtain

$$\begin{aligned} \mathcal{V}(m,\alpha,W) &= \min\Big\{\frac{\alpha C(m)-W}{\alpha+\lambda} + \frac{\lambda \mathcal{V}(m+1,\alpha,W)}{\alpha+\lambda}; \tilde{C}(m,\alpha) \\ &\quad + \sum_{n=0}^{\infty}\int_0^{\infty}\frac{(\lambda t)^n}{n!}e^{-(\alpha+\lambda)t}\mathcal{V}(n+m-1,\alpha,W)dG)\Big\}, m \in \mathbb{N} \end{aligned} \tag{3.26}$$

Also note that if we are in state 0 then passive is the only admissible action so we also have that

$$\begin{aligned}
\mathcal{V}(0,\alpha,W) &= \frac{\alpha C(0) - W}{\alpha+\lambda} + \frac{\lambda \mathcal{V}(1,\alpha,W)}{\alpha+\lambda} \\
\implies \quad (\alpha+\lambda)\mathcal{V}(0,\alpha,W) &= -W + \lambda \mathcal{V}(1,\alpha,W),
\end{aligned}$$

since $C(0) = 0$.

It is this problem in (3.26) which we consider. So we have W with the economic interpretation as a subsidy for passivity, i.e. a payment made to the system whenever we take the action "do not serve". We use $\Pi_\alpha(W)$ to denote the set of states for which the passive action b is optimal in this problem. So we have

$$\begin{aligned}
\Pi_\alpha(W) \;=\; & \{0\} \cup \{m \in \mathbb{Z}^+ \text{ such that the passive action is optimal in } m \text{ when} \\
& \text{the subsidy for passivity is } W\}, W \in \mathbb{R}
\end{aligned} \tag{3.27}$$

If we have α-indexability, namely that $\Pi_\alpha(W)$ is increasing with W, we then use $W_\alpha(m)$ for the Whittle α-index for the customer class concerned in state m. We now give a heuristic argument to lead us to a formula for this index $W_\alpha(m)$, in terms of model parameters, when $W_\alpha(.)$ is assumed to be an increasing function, as would seem plausible. When we have found this formula for the index we will then verify its increasing nature.

We consider the service control problem (a) - (c), except now we have changed from the charge for service to the subsidy for passivity as noted above. We start with the number of customers initially in the queue at m, i.e. $N(0) = m$. We also have a discount rate of α and passive subsidy $W = \bar{W}_\alpha(m)$ set equal to the *assumed* value of the α-index in state m. We make the following two assumptions:

1. The α-index, $W_\alpha(n)$, is increasing in the state, n.

2. When the passivity subsidy, W, is equal to the α-index $W_\alpha(m)$ in state m, both of the actions a and b are optimal in that state.

Both of these assumptions will be verified later in the analysis. We can now infer the following for our problem with passive subsidy W set equal to the assumed index value:

(i) the active action a must be optimal in states $\{m+1, m+2, \ldots\}$;

(ii) the passive action b must be optimal in states $\{0, 1, \ldots\ m-1\}$;

(iii) actions a and b are both optimal in state m.

Note that (i) and (ii) follow from Assumption 1 and the definition in (3.27), while (iii) follows from Assumption 2.

Hence under these assumptions we can see that there are two stationary policies which are optimal when $W = \bar{W}_\alpha(m)$. Both optimal policies make choices according to (i) and (ii) above. Let the stationary optimal policy which chooses the active action a in state m be denoted by u_1, and the optimal policy which chooses the passive action b in state m be denoted by u_2. Our approach which leads us to a formula for $\bar{W}_\alpha(m)$, involves calculating the total expected discounted cost of following u_1 and also of following u_2, then equating these and solving for the passive subsidy.

Since we have the initial state $N(0) = m$, policy u_1 will take the active action a from time 0, until the time when the state first enters $m-1$. If we denote this time by T, then we write

$$T = \inf\{t; N(t) = m-1\}. \tag{3.28}$$

Note that since the state space is not bounded above, we can see that T will be independent of the current state m. The cost incurred during this initial active phase i.e. the discounted holding cost until T is,

$$E\Big[\int_0^T \alpha C(N(t))e^{-\alpha t}dt|N(0) = m, a\Big] \;=\; \bar{C}(m, \alpha). \tag{3.29}$$

Note that this random variable T is stochastically identical to the busy period of an $M/G/1$ queueing system, starting with a single customer and having arrival rate λ and generic customer service time S. Having arrived in state $m-1$ at time T, according to (ii) above, policy u_1 will now take the passive action b, until a customer arrives - taking the state back up to m. The inter-arrival time of a Poisson process follows an exponential distribution and hence we know that the arrival will occur at time $T+X$ where $X \sim \exp(\lambda)$. The expected cost incurred during this passive phase will be the passive cost rate when we have $m-1$ customers multiplied by the discounted time until arrival all discounted back from time T to 0, which can be written as

$$\begin{aligned} &= E(e^{-\alpha T}) \times \big(\alpha C(m-1) - \bar{W}_\alpha(m)\big) \times E_X\Big(\int_0^X e^{-\alpha t}dt\Big) \\ &= E(e^{-\alpha T})\frac{\alpha C(m-1) - \bar{W}_\alpha(m)}{\alpha+\lambda}. \end{aligned} \tag{3.30}$$

Since $N((T+X)^+) = m$, the policy u_1 now repeats the above cycle *ad infinitum* from time $T+X$. The total expected cost associated with this policy may be found as the sum of an infinite geometric progression. The expected cost of a single cycle will remain fixed but the expected discounting applied will decrease for each successive term by the factor

$$\begin{aligned} & E(e^{-\alpha(T+X)}) \\ &= E(e^{-\alpha T})E(e^{-\alpha X}) \\ &= \frac{\lambda E(e^{-\alpha T})}{\alpha+\lambda}. \end{aligned} \tag{3.31}$$

So using (3.29), (3.30) and (3.31) we find that

$$\mathcal{V}_{u_1}\{m, \alpha, \bar{W}_\alpha(m)\} \;=\; \frac{\bar{C}(m,\alpha) + E(e^{-\alpha T})\{\alpha C(m-1) - \bar{W}_\alpha(m)\}(\alpha+\lambda)^{-1}}{1 - \lambda E(e^{-\alpha T})(\alpha+\lambda)^{-1}}. \tag{3.32}$$

We do still need to find expressions for the terms, $E(e^{-\alpha T})$ and $\bar{C}(m, \alpha)$. However I will for now continue by finding the corresponding total expected discounted cost of following policy u_2.

We again start from the initial state $N(0) = m$, so now under policy u_2 the passive action b will be taken at time 0 and remain in force for a period of time we denote by X, i.e. until the first arrival after 0 occurs. As above we have that $X \sim \exp(\lambda)$. At the conclusion of this time period a transition to state $m+1$ will occur. The expected cost incurred during this initial passive phase will therefore be the passive cost rate multiplied by discounted expected time until the arrival, i.e.

$$
\begin{aligned}
& (\alpha C(m) - \bar{W}_\alpha(m)) \times E\Big(\int_0^X e^{-\alpha t} dt \Big) \\
= \; & \frac{\alpha C(m) - \bar{W}_\alpha(m)}{\alpha + \lambda}.
\end{aligned} \tag{3.33}
$$

After this initial passive phase the active action will be taken until the queue length returns to m for the first time. This will take a further amount of time which is stochastically identical to T above. So the expected cost incurred during this active phase is the discounted holding cost from time X to time $X + T$,

$$
\begin{aligned}
& \bar{C}(m+1, \alpha) E(e^{-\alpha X}) \\
= \; & \frac{\lambda \bar{C}(m+1, \alpha)}{\alpha + \lambda}.
\end{aligned} \tag{3.34}
$$

As with policy u_1, policy u_2 now repeats this cycle *ad infinitum*. So the total expected cost can again be found as the sum of an infinite geometric progression, with common ratio given by the quantity in (3.31). So using (3.31), (3.33) and (3.34) we have

$$
\mathcal{V}_{u_2}\{m, \alpha, \bar{W}_\alpha(m)\} = \frac{\{\alpha C(m) - \bar{W}_\alpha(m) + \lambda \bar{C}(m+1, \alpha)\}(\alpha + \lambda)^{-1}}{1 - \lambda E(e^{-\alpha T})(\alpha + \lambda)^{-1}}. \tag{3.35}
$$

But as we have already said, both policies u_1 and u_2 are optimal when the service charge is $W = \bar{W}_\alpha(m)$ and hence it must follow from (3.32) and (3.35) that

$$\begin{aligned} \mathcal{V}_{u_1}\{m, \alpha, \bar{W}_\alpha(m)\} &= \mathcal{V}_{u_2}\{m, \alpha, \bar{W}_\alpha(m)\} \\ \Longrightarrow (\alpha+\lambda)\bar{C}(m,\alpha) + E(e^{-\alpha T})\{\alpha C(m-1) - \bar{W}_\alpha(m)\} &= \alpha C(m) - \bar{W}_\alpha(m) \\ &\quad +\lambda \bar{C}(m+1,\alpha) \\ \Longrightarrow \bar{W}_\alpha(m)\{1 - E(e^{-\alpha T})\} = \alpha C(m) - E(e^{-\alpha T})\alpha C(m-1) &+ \lambda \bar{C}(m+1,\alpha) \\ -(\alpha+\lambda)\bar{C}(m,\alpha) & \qquad m \in \mathbb{Z}^+ \qquad (3.36) \end{aligned}$$

So using the above argument we infer that the α-index takes the form,

$$\bar{W}_\alpha(m) = \frac{\lambda \bar{C}(m+1,\alpha) - (\alpha+\lambda)\bar{C}(m,\alpha) + \alpha C(m) - \alpha E(e^{-\alpha T})C(m-1)}{1 - E(e^{-\alpha T})}, \quad m \in \mathbb{Z}^+. \tag{3.37}$$

We now use standard conditioning arguments to find formulae for the quantities $E(e^{-\alpha T})$ and $\bar{C}(m,\alpha)$ so we are able to calculate the index in (3.37). Firstly consider $E(e^{-\alpha T})$, where T is the time it takes the system to get from its current state (in the active mode), to a state where it has one less customer. The difficulty with this is that new customers constantly arrive into the system. When we are in the current state m, by the time we have served the customer currently in service we may have had, say, r customer arrivals and so will be in state $m-1+r$. We consider the probability distribution of the number of arrivals in the general service time, S.

$$\begin{aligned} P(r) &= \text{the probability of } r \text{ arrivals in service time } S \\ &= \int_0^\infty P(r|S=t)dG(t) \\ &= \int_0^\infty \frac{(\lambda t)^r}{r!} e^{-\lambda t} dG, \end{aligned} \tag{3.38}$$

where we have said the general service distribution has distribution function G, and we know that the arrivals occur according to a Poisson process at rate λ. We can see that if we have r arrivals during the first service, we then need r subsequent busy periods to return to the original state, so we therefore have

$$\begin{aligned}
E(e^{-\alpha T}) &= \sum_{r=0}^{\infty} \int_0^{\infty} \frac{(\lambda t)^r}{r!} e^{-(\alpha+\lambda)t} \{E(e^{-\alpha T})\}^r dG \\
&= \int_0^{\infty} \exp\{\lambda E(e^{-\alpha T})t - (\alpha+\lambda)t\} dG \\
&= \tilde{G}\Big(\alpha + \lambda\big[1 - E(e^{-\alpha T})\big]\Big) \qquad (3.39)
\end{aligned}$$

where

$$\tilde{G}(\xi) = \int_0^{\infty} e^{-\xi t} dG.$$

We now have an equation for $E(e^{-\alpha T})$ and so we continue by using standard conditioning arguments to find a formula for $\bar{C}(m, \alpha)$.

In this chapter we consider a general service distribution and so must find expressions for all the required quantities on this basis. However one could perhaps find these formulae more easily if the actual service distribution was known. In fact we initially considered service to be Gamma distributed, as this choice has some simplifying features. However now we are able to now give an account appropriate for a general distributional form.

Recall from (3.29), that $\bar{C}(m, \alpha) =$ the expected discounted holding cost associated with the initial active phase (of duration T from state m down to $m - 1$) i.e.

$$\bar{C}(m, \alpha) = E\Big[\int_0^T \alpha C(N(t)) e^{-\alpha t} dt | N(0) = m, a\Big].$$

Also recall the notation,

$$\begin{aligned}
\tilde{C}(m, \alpha) &= \text{total discounted cost during the initial service offered in state } m. \\
&= E\Big[\int_0^S \alpha C\big(N(t)\big) e^{-\alpha t} dt \big| N(0) = m, a\Big] \qquad (3.40)
\end{aligned}$$

We now aim to simplify the algebra by using,

$$A = E(e^{-\alpha T}). \qquad (3.41)$$

Should n customers arrive during the initial service, then $m+n-1$ customers will be present after the first service and successive busy periods will reduce the queue length such that

$$m+n-1 \to m+n-2 \to \ldots \to m \to m-1.$$

Hence we can see that $\bar{C}(m,\alpha)$ can be disaggregated into the discounted cost until the first service completion and the cost to get from state $m+n-1$ down to $m-1$ multiplied by the probability of n arrivals during this initial service, for all $n \in \mathbb{Z}^+$, all discounted accordingly. This can be written as

$$\begin{aligned}
\bar{C}(m,\alpha) &= \tilde{C}(m,\alpha) + \int_0^\infty \frac{\lambda t}{1!} e^{-\lambda t} \bar{C}(m,\alpha) e^{-\alpha t} dG \\
&\quad + \int_0^\infty \frac{(\lambda t)^2}{2!} e^{-\lambda t} \big[\bar{C}(m+1,\alpha)e^{-\alpha t} + \bar{C}(m,\alpha) A e^{-\alpha t}\big] dG \\
&\quad + \int_0^\infty \frac{(\lambda t)^3}{3!} e^{-\lambda t} \big[\bar{C}(m+2,\alpha)e^{-\alpha t} + \bar{C}(m+1,\alpha) A e^{-\alpha t} \\
&\qquad + C(m,\alpha) A^2 e^{-\alpha t}\big] dG \\
&\quad + \ldots \qquad\qquad (3.42) \\
&= \tilde{C}(m,\alpha) + \int_0^\infty \frac{\lambda t}{1!} e^{-(\alpha+\lambda)t} \bar{C}(m,\alpha) dG \\
&\quad + \int_0^\infty \frac{(\lambda t)^2}{2!} e^{-(\alpha+\lambda)t} \sum_{r=0}^{1} \bar{C}(m+r,\alpha) A^{1-r} dG \\
&\quad + \int_0^\infty \frac{(\lambda t)^3}{3!} e^{-(\alpha+\lambda)t} \sum_{r=0}^{2} \bar{C}(m+r,\alpha) A^{2-r} dG \\
&\quad + \ldots \qquad\qquad (3.43) \\
&= \tilde{C}(m,\alpha) + \sum_{n=1}^{\infty} \int_0^\infty \frac{(\lambda t)^n}{n!} e^{-(\alpha+\lambda)t} \left[\sum_{r=0}^{n-1} \bar{C}(m+r,\alpha) \big\{E(e^{-\alpha T})\big\}^{n-1-r}\right] dG. \qquad (3.44)
\end{aligned}$$

Where we use (3.38) to get (3.42) and then (3.41) to get to (3.44). So we can now see how expression (3.44) disaggregates the total expected cost incurred during $[0,T)$ in (3.29) into that incurred during the processing of the first customer and the residual cost (if any) incurred by customers arriving during this initial service. The second term on the r.h.s. of (3.44) gives the expected cost associated with this residual processing.

We now look at a couple of expressions which will prove useful when we try to prove Lemma 2 below. The first of these expressions records the special form of the distribution function for a gamma $\Gamma(n, \lambda)$, where $n \in \mathbb{Z}^+$. Let Z be the time of the n^{th} arrival in a Poisson process. Then $Z \sim \Gamma(n, \lambda)$. Hence the distribution function is given by,

$$\begin{aligned} P(Z \leq t) &= P(Q(t) \geq n) \\ &= \sum_{r=n}^{\infty} \frac{(\lambda t)^m}{m!} e^{-\lambda t}, \end{aligned} \tag{3.45}$$

where $Q(t)$ represents the number of events in a Poisson process up to t, i.e. $Q(t) \sim P(\lambda t)$. The second useful expression is a formula for $\tilde{C}(m, \alpha)$, namely the expected discounted holding costs incurred during the service of a single customer when the queue is in state m at time 0. Figure 3.2 may be useful when formulating this expression.

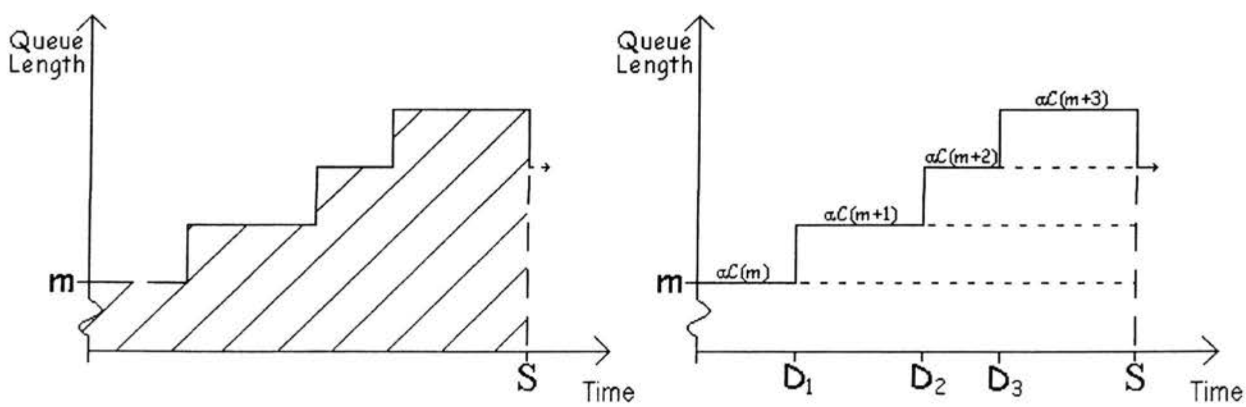

Figure 3.2: Possible state transition diagram until a customer is served.

In Figure 3.2, S is the service time of the m^{th} customer and we can see that $\tilde{C}(m, \alpha)$ will be the discounted area under this graph. Hence $\tilde{C}(m, \alpha)$ can be disaggregated into the state m active cost rate multiplied by the discounted time between 0 and S plus the difference in the cost rates for states $m+i$ and $m+i-1$ multiplied by the discounted time between D_i and S and by the probability that the i^{th} arrival occurs at $D_i < S$, for all $i \in \mathbb{Z}^+$. This can be written as

$$\begin{aligned}
\tilde{C}(m,\alpha) &= E_S\Big\{\alpha C(m)\int_0^S e^{-\alpha u}du \\
&\quad +\big(\alpha C(m+1)-\alpha C(m)\big)E_{D_1}\Big[\int_{D_1}^S e^{-\alpha u}I(D_1<S)du\Big] \\
&\quad +\big(\alpha C(m+2)-\alpha C(m+1)\big)E_{D_2}\Big[\int_{D_2}^S e^{-\alpha u}I(D_2<S)du\Big]+\dots\Big\} \\
&= E_S\Big\{C(m)(1-e^{\alpha S}) \\
&\quad +\big(C(m+1)-C(m)\big)E_{D_1}\Big[(e^{-\alpha D_1}-e^{-\alpha S})I(D_1<S)\Big] \\
&\quad +\big(C(m+2)-C(m+1)\big)E_{D_2}\Big[(e^{-\alpha D_2}-e^{-\alpha S})I(D_2<S)\Big]+\dots\Big\} \\
&= E_S\Big\{C(m)(1-e^{-\alpha S}) \\
&\quad +\big(C(m+1)-C(m)\big)\int_0^S (e^{-\alpha d_1}-e^{-\alpha S})\lambda e^{-\lambda d_1}dd_1 \\
&\quad +\big(C(m+2)-C(m+1)\big)\int_0^S (e^{-\alpha d_2}-e^{-\alpha S})\lambda^2 d_2 e^{-\lambda d_2}dd_2+\dots\Big\}+\dots \\
&= C(m)E(1-e^{-\alpha S}) \\
&\quad +\big(C(m+1)-C(m)\big)\int_0^\infty\Big[\int_0^s (e^{-\alpha d_1}-e^{-\alpha s})\lambda e^{-\lambda t_1}dd_1\Big]dG(s) \\
&\quad +\big(C(m+2)-C(m+1)\big)\int_0^\infty\Big[\int_0^s (e^{-\alpha d_2}-e^{-\alpha s})\lambda^2 d_2 e^{-\lambda d_2}dt_2\Big]dG(s)+\dots \\
&= C(m)E(1-e^{-\alpha S})+\sum_{n=1}^{\infty}\big\{C(n+m)-C(n+m-1)\big\}\times \\
&\qquad \int_0^\infty\Big[\int_0^s \frac{\lambda^n d^{n-1}e^{-\lambda d}}{(n-1)!}\{e^{-\alpha d}-e^{-\alpha s}\}dd\Big]dG, \qquad (3.46)
\end{aligned}$$

where we use the fact that the time between the start of service and the arrival of the n^{th} customer will follow a gamma $\Gamma(n, \lambda)$ distribution.

Using (3.45), the form of the $\Gamma(n, \lambda)$ distribution function, in the last term of (3.46) we can see that

$$\begin{aligned}\int_0^s \frac{\lambda^n t^{n-1} e^{-\lambda t}}{(n-1)!}\{e^{-\alpha t} - e^{-\alpha s}\}dt &= \int_0^s \frac{(\lambda+\alpha)^n t^{n-1} e^{-(\alpha+\lambda)t}}{(n-1)!}\frac{\lambda^n}{(\lambda+\alpha)^n}dt \\ &\quad - \int_0^s \frac{\lambda^n t^{n-1} e^{-\lambda t}}{(n-1)!} e^{-\alpha s} dt \\ &= \frac{\lambda^n}{(\lambda+\alpha)^n}\sum_{r=n}^{\infty}\frac{(\lambda+\alpha)^r s^r e^{-(\lambda+\alpha)s}}{r!} \\ &\quad - \sum_{r=n}^{\infty}\frac{\lambda^r s^r e^{-(\lambda+\alpha)s}}{r!}. \end{aligned} \tag{3.47}$$

Therefore we can see that (3.46) becomes,

$$\begin{aligned}\tilde{C}(m,\alpha) &= C(m)E(1-e^{-\alpha S}) + \sum_{n=1}^{\infty}\{C(n+m) - C(n+m-1)\} \times \\ &\quad \left[\frac{\lambda^n}{(\lambda+\alpha)^n}E\Big\{\sum_{r=n}^{\infty}\frac{\big((\lambda+\alpha)S\big)^r e^{-(\lambda+\alpha)S}}{r!}\Big\} - E\Big\{\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\lambda+\alpha)S}}{r!}\Big\}\right]. \end{aligned} \tag{3.48}$$

Lemma 2 asserts that our conjectured index $\bar{W}_\alpha(m)$ in (3.37) is increasing in m, as was assumed for the true index in the preceding argument on page 3.3.1. In Lemma 2, we take $\bar{W}_\alpha(0)$ to be zero. Also recall that for the economy of notation we have introduced A for the quantity $E(e^{-\alpha T})$.

Lemma 2

$\bar{W}_\alpha(m)$ is increasing in m.

Proof

Using identity (3.44) in (3.37)we can infer that, for $m \in \mathbb{Z}^+$

$$\begin{aligned}
(1 \quad - \quad A)\bar{W}_\alpha(m) \quad &= \quad \lambda\bar{C}(m+1,\alpha) - (\alpha+\lambda)\bar{C}(m,\alpha) + \alpha C(m) - \alpha A C(m-1) \\
&= \lambda\tilde{C}(m+1,\alpha) - (\alpha+\lambda)\tilde{C}(m,\alpha) + \alpha C(m) - \alpha A C(m-1) \\
&\quad +\lambda\sum_{n=1}^{\infty}\int_0^{\infty}\frac{(\lambda t)^n}{n!}e^{-(\alpha+\lambda)t}\Big\{\sum_{r=0}^{n-1}\bar{C}(m+1+r,\alpha)A^{n-1-r}\Big\}dG \\
&\quad -(\alpha+\lambda)\sum_{n=1}^{\infty}\int_0^{\infty}\frac{(\lambda t)^n}{n!}e^{-(\alpha+\lambda)t}\Big\{\sum_{r=0}^{n-1}\bar{C}(m+r,\alpha)A^{n-1-r}\Big\}dG \\
&= \lambda\tilde{C}(m+1,\alpha) - (\alpha+\lambda)\tilde{C}(m,\alpha) + \alpha C(m) - \alpha A C(m-1) \\
&\quad +\sum_{j=1}^{\infty}E\Big[\sum_{k=j}^{\infty}\frac{(\lambda S)^k}{k!}e^{-(\alpha+\lambda)S}\Big]\big\{\lambda\bar{C}(m+j,\alpha) - (\alpha+\lambda)\bar{C}(m+j-1,\alpha)\big\}A^{k-j} \\
&= \lambda\tilde{C}(m+1,\alpha) - (\alpha+\lambda)\tilde{C}(m,\alpha) + \alpha C(m) - \alpha A C(m-1) \\
&\quad -\alpha\sum_{n=1}^{\infty}E\Big[\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\alpha+\lambda)S}}{r!}A^{r-n}\Big]\big\{C(m+n-1) - AC(m+n-2)\big\} \\
&\quad +\sum_{n=1}^{\infty}E\Big[\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\alpha+\lambda)S}}{r!}A^{r-n}\Big]\big\{\lambda\bar{C}(m+n,\alpha) - (\lambda+\alpha)\bar{C}(m+n-1,\alpha) \\
&\qquad\qquad +\alpha C(m+n-1) - \alpha A C(m+n-2)\big\}.
\end{aligned} \tag{3.49}$$

We can now use (3.37) in (3.49) to show that,

$$\begin{aligned}
\frac{(1-A)\bar{W}_\alpha(m)}{\alpha} \quad &= \quad \frac{\lambda}{\alpha}\tilde{C}(m+1,\alpha) - \left(\frac{\alpha+\lambda}{\alpha}\right)\tilde{C}(m,\alpha) + C(m) - AC(m-1) \\
&\quad -\sum_{n=1}^{\infty}E\Big[\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\alpha+\lambda)S}}{r!}A^{r-n}\Big]\big\{C(m+n-1) - AC(m+n-2)\big\} \\
&\quad +\sum_{n=1}^{\infty}E\Big[\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\alpha+\lambda)S}A^{r-n}}{r!}\Big]\Big\{\frac{(1-A)\bar{W}_\alpha(m+n-1)}{\alpha}\Big\} \\
&= \quad \frac{\lambda}{\alpha}\tilde{C}(m+1,\alpha) - \left(\frac{\alpha+\lambda}{\alpha}\right)\tilde{C}(m,\alpha) + C(m) - AC(m-1) \\
&\quad +\sum_{n=1}^{\infty}\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}A^n C(m-1) - \sum_{n=1}^{\infty}\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}C(m+n-1) \\
&\quad +\sum_{n=1}^{\infty}E\Big[\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\alpha+\lambda)S}A^{r-n}}{r!}\Big]\Big\{\frac{(1-A)\bar{W}_\alpha(m+n-1)}{\alpha}\Big\}.
\end{aligned} \tag{3.50}$$

Now using expression (3.48) together with some algebraic manipulation the above expression becomes

$$\begin{aligned}
\frac{(1-A)\bar{W}_\alpha(m)}{\alpha} &= \sum_{n=2}^{\infty}\{C(m+n)-C(m+n-1)\}\times E\Big[\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\alpha+\lambda)S}}{r!}\Big] \\
&\qquad +\big\{C(m+1)-C(m)\big\}E(e^{-\alpha S}-e^{-(\alpha+\lambda)S}) \\
&\qquad\qquad -C(m)E(1-e^{-\alpha S})+C(m)-AC(m-1) \\
&\quad +\sum_{n=1}^{\infty}\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}A^n C(m-1)-\sum_{n=1}^{\infty}\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}C(m+n-1) \\
&\quad +\sum_{n=1}^{\infty}E\Big[\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\alpha+\lambda)S}A^{r-n}}{r!}\Big]\Big\{\frac{(1-A)\bar{W}_\alpha(m+n-1)}{\alpha}\Big\} \\
&= C(m)E\big[\lambda S e^{-(\alpha+\lambda)S}\big]-C(m+1)\sum_{n=2}^{\infty}E\Big[\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}\Big] \\
&\quad +\sum_{n=2}^{\infty}E\Big[\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}\Big]\{C(m+n)-C(m+n-1)\} \\
&\qquad +\big\{C(m+1)-C(m)\big\}E(e^{-\alpha S}-e^{-(\alpha+\lambda)S}) \\
&\qquad\qquad -C(m)E(1-e^{-\alpha S})+C(m)-AC(m-1) \\
&\quad +\sum_{n=1}^{\infty}\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}A^n C(m-1) \\
&\quad +\sum_{n=1}^{\infty}E\Big[\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\alpha+\lambda)S}A^{r-n}}{r!}\Big]\Big\{\frac{(1-A)\bar{W}_\alpha(m+n-1)}{\alpha}\Big\}. \quad (3.51)
\end{aligned}$$

Now notice that

$$\sum_{n=2}^{\infty}E\Big[\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}\Big] = E\big[e^{-\alpha S}-e^{-(\alpha+\lambda)S}-\lambda S e^{-(\alpha+\lambda)S}\big].$$

Further, using (3.39) we have that

$$\sum_{n=1}^{\infty}E\Big[\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}\Big]A^n = A-E\big[e^{-(\alpha+\lambda)S}\big]. \quad (3.52)$$

Using these relations and further algebra we see that (3.51) becomes

$$\begin{aligned}
\frac{(1-A)\bar{W}_\alpha(m)}{\alpha} &= \sum_{n=0}^{\infty}E\Big[\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}\Big]\{C(m+n)-C(m+n-1)\} \\
&\quad +\sum_{n=1}^{\infty}E\Big[\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\alpha+\lambda)S}A^{r-n}}{r!}\Big]\Big\{\frac{(1-A)\bar{W}_\alpha(m+n-1)}{\alpha}\Big\}. \quad (3.53)
\end{aligned}$$

However, identity (3.53) is strongly suggestive of the following computational scheme for $\alpha^{-1}(1-A)\bar{W}_\alpha(m)$, $m\in\mathbb{Z}^+$: Use $\bar{W}_\alpha^R(.)$ to denote the R^{th} iterate of the target function $\bar{W}_\alpha(.)$. Take $\bar{W}_\alpha^1(m) = 0$, $m \in \mathbb{Z}^+$, and

$$\frac{(1-A)\bar{W}_\alpha^{R+1}(m)}{\alpha} = \sum_{n=0}^{\infty} E\Big\{\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}\Big\}\{C(m+n) - C(m+n-1)\} + \sum_{n=1}^{\infty} E\Big\{\sum_{r=n}^{\infty} \frac{\lambda^r S^r e^{-(\alpha+\lambda)S} A^{r-n}}{r!}\Big\}\Big\{\frac{(1-A)\bar{W}_\alpha^R(m+n-1)}{\alpha}\Big\}. \tag{3.54}$$

The aim is now to use this computational scheme as a vehicle to prove Lemma 2. To do this we follow the steps laid out below.

1. Prove that $\bar{W}_\alpha^R(m)$ is increasing in R $\forall$ m, by induction on R.

2. Prove that $\bar{W}_\alpha(m) \geq \bar{W}_\alpha^R(m)$ $\forall$ m, by induction on R, (so this relation will hold for all m and all R).

3. This leads us to the fact that

$$\lim_{R\to\infty} \bar{W}_\alpha^R(m) = \Upsilon(m) \leq \bar{W}_\alpha(m) \ \ \forall \ \ m.$$

4. Prove $\bar{W}_\alpha(m) = \Upsilon(m)$ $\forall$ m, by an argument based on $\sup_m\{\bar{W}_\alpha(m) - \Upsilon(m)\}$.

5. This leads us to the fact that

$$\lim_{R\to\infty} \bar{W}_\alpha^R(m) = \bar{W}_\alpha(m) \ \ \forall \ \ m.$$

6. Prove $\bar{W}_\alpha^R(m)$ is increasing in m for all R, by induction.

7. We can then deduce that $\bar{W}_\alpha(m)$ is increasing in m, as required.

Let us now consider step 1: Prove that $\bar{W}_\alpha^R(m)$ is increasing in R $\forall$ m, by induction on R. Consider the iterative function in (3.54). We can obviously see that,

$$\bar{W}_\alpha^1(m) \leq \bar{W}_\alpha^2(m) \ \ \forall \ \ m \in \mathbb{Z}^+.$$

since we have that $\bar{W}^1_\alpha(m) = 0$ and the holding cost function, $C(.)$ is increasing. We now suppose that

$$\bar{W}^R_\alpha(m) \geq \bar{W}^{R-1}_\alpha(m) \geq \ldots \geq \bar{W}^1_\alpha(m) = 0 \;\forall\; m \in \mathbb{Z}^+$$

Now notice that,

$$\begin{aligned}
\alpha^{-1}(1-A)\bar{W}^{R+1}_\alpha(m) &= \sum_{n=0}^{\infty} E\Big\{\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}\Big\}\{C(m+n) - C(m+n-1)\} \\
&\quad + \sum_{n=1}^{\infty} E\Big\{\sum_{r=n}^{\infty} \frac{\lambda^r S^r e^{-(\alpha+\lambda)S} A^{r-n}}{r!}\Big\}\Big\{\frac{(1-A)\bar{W}^R_\alpha(m+n-1)}{\alpha}\Big\} \\
&\geq \sum_{n=0}^{\infty} E\Big\{\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}\Big\}\{C(m+n) - C(m+n-1)\} \\
&\quad + \sum_{n=1}^{\infty} E\Big\{\sum_{r=n}^{\infty} \frac{\lambda^r S^r e^{-(\alpha+\lambda)S} A^{r-n}}{r!}\Big\}\Big\{\frac{(1-A)\bar{W}^{R-1}_\alpha(m+n-1)}{\alpha}\Big\} \\
&= \alpha^{-1}(1-A)\bar{W}^R_\alpha(m) \\
\Longrightarrow \bar{W}^{R+1}_\alpha(m) &\geq \bar{W}^R_\alpha(m), \text{ since } \alpha > 0,\ (1-A) > 0.
\end{aligned}$$

We have therefore proved step 1 as required. So we now look at step 2: Prove that $\bar{W}_\alpha(m) \geq \bar{W}^R_\alpha(m)\ \forall\ m$, by induction on R. To do this first recall the formula (3.51), and since we have $\bar{W}^1_\alpha(m) = 0$ then obviously $\bar{W}_\alpha(m) \geq \bar{W}^1_\alpha(m)\ \forall\ m$. Hence the initial case holds, so we now suppose that,

$$\bar{W}_\alpha(m) \geq \bar{W}^i_\alpha(m)\ \forall\ m, \text{ and } 1 \leq i \leq R, \tag{3.55}$$

and must infer that $\bar{W}_\alpha(m) \geq \bar{W}^{R+1}_\alpha(m)\ \forall\ m$. We have, using (3.53) and (3.54) that

$$\begin{aligned}
\alpha^{-1}(1-A)\bar{W}^{R+1}_\alpha(m) &= \sum_{n=0}^{\infty} E\Big\{\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}\Big\}\{C(m+n) - C(m+n-1)\} \\
&\quad + \sum_{n=1}^{\infty} E\Big\{\sum_{r=n}^{\infty} \frac{\lambda^r S^r e^{-(\alpha+\lambda)S} A^{r-n}}{r!}\Big\}\Big\{\frac{(1-A)\bar{W}^R_\alpha(m+n-1)}{\alpha}\Big\} \\
&\leq \sum_{n=0}^{\infty} E\Big\{\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}\Big\}\{C(m+n) - C(m+n-1)\} \\
&\quad + \sum_{n=1}^{\infty} E\Big\{\sum_{r=n}^{\infty} \frac{\lambda^r S^r e^{-(\alpha+\lambda)S} A^{r-n}}{r!}\Big\}\Big\{\frac{(1-A)\bar{W}_\alpha(m+n-1)}{\alpha}\Big\} \\
&= \alpha^{-1}(1-A)\bar{W}_\alpha(m) \\
\Longrightarrow \bar{W}^{R+1}_\alpha(m) &\leq \bar{W}_\alpha(m)\ \forall\ m, \text{ since } \alpha > 0,\ (1-A) > 0,
\end{aligned}$$

where the second line follows from the induction hypothesis. Hence we have proved step 2. So we now move on to step 3, since $\bar{W}_\alpha^R(m)$ is increasing in R and is bounded above by $\bar{W}_\alpha(m)$, we can see that $\bar{W}_\alpha^R(m)$ must tend to some limit as R tends to ∞. Let us call this limit $\Upsilon(m)$, i.e.

$$\lim_{R\to\infty} \bar{W}_\alpha^R(m) = \Upsilon(m) \ \forall \ m \in \mathbb{Z}^+.$$

We also know that

$$\Upsilon(m) \leq \bar{W}_\alpha(m) \ \forall \ m \in \mathbb{Z}^+,$$

since we have demonstrated that $\bar{W}_\alpha(m) \geq \bar{W}_\alpha^R(m)$ for all m and R. Now we will consider an expression which will prove useful in step 4. By straightforward algebraic manipulation we have that

$$\begin{aligned}\sum_{n=1}^{\infty} E\left\{\sum_{r=n}^{\infty} \frac{\lambda^r S^r e^{-(\lambda+\alpha)S} A^{r-n}}{r!}\right\} &= \frac{1}{1-A} E\left\{\sum_{r=1}^{\infty} \frac{\lambda^r S^r e^{-(\lambda+\alpha)S}}{r!} - \sum_{r=1}^{\infty} \frac{\lambda^r S^r e^{-(\lambda+\alpha)S} A^r}{r!}\right\} \\ &= \frac{1}{1-A} E\left\{e^{-(\lambda+\alpha)S}(e^{\lambda S}-1) - e^{-(\lambda+\alpha)S}(e^{A\lambda S}-1)\right\} \\ &= \frac{E\{e^{-\alpha S} - A\}}{1-A} < 1. \end{aligned} \tag{3.56}$$

It can also be seen that this expression will be less than 1 since $0 < A < 1$ and $0 < E(e^{-\alpha S}) < 1$. It is also true that $E(e^{-\alpha S}) > A$ and hence

$$0 < \frac{E\{e^{-\alpha S}\} - A}{1-A} < 1.$$

We now move onto step 4 and show that $\bar{W}_\alpha(m) = \Upsilon(m) \ \forall \ m$. To do this we consider

$$\begin{aligned}
&\sup_m \left[\alpha^{-1}(1-A)\{\bar{W}_\alpha(m)-\Upsilon(m)\}\right] \\
= &\sup_m \Bigg[\sum_{n=1}^{\infty} E\Big\{\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\lambda+\alpha)S}A^{r-n}}{r!}\Big\}\Big\{\alpha^{-1}(1-A)\bar{W}_\alpha(n+m-1)\Big\} \\
&\qquad -\sum_{n=1}^{\infty} E\Big\{\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\lambda+\alpha)S}A^{r-n}}{r!}\Big\}\Big\{\alpha^{-1}(1-A)\Upsilon(n+m-1)\Big\}\Bigg] \\
= &\sup_m \Bigg[\sum_{n=1}^{\infty} E\Big\{\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\lambda+\alpha)S}A^{r-n}}{r!}\Big\}\Big\{\alpha^{-1}(1-A)\Big\}\Big\{\bar{W}_\alpha(n+m-1) \\
&\qquad -\Upsilon(n+m-1)\Big\}\Bigg] \\
\leq &\sum_{n=1}^{\infty} E\Big\{\sum_{r=n}^{\infty}\frac{\lambda^r S^r e^{-(\lambda+\alpha)S}A^{r-n}}{r!}\Big\}\Big\{\alpha^{-1}(1-A)\Big\}\sup_m\Big\{\bar{W}_\alpha(m)-\Upsilon(m)\Big\},
\end{aligned}$$

where the second line follows from equations (3.53) and (3.54). Then using relation (3.56) and rearranging we can see that

$$\Longrightarrow \sup_m\{\bar{W}_\alpha(m)-\Upsilon(m)\} \;\leq\; \frac{E(e^{-\alpha S}-A)}{1-A}\sup_m\big\{\bar{W}_\alpha(m)-\Upsilon(m)\big\}.$$

Since we know inequality (3.56) is true, this could only occur when

$$\begin{aligned}
\sup_m\{\bar{W}_\alpha(m)-\Upsilon(m)\} &= 0 \\
\Longrightarrow \qquad \bar{W}_\alpha(m) &= \Upsilon(m)\ \forall\ m. \qquad (3.57)
\end{aligned}$$

So this proves step 4 and leads us to conclude that we do have the relation in step 5, i.e.

$$\Upsilon(m)=\lim_{R\to\infty}\bar{W}_\alpha^R(m)=\bar{W}_\alpha(m)\ \ \forall\ \ m.$$

So we now proceed to step 6 and prove that $\bar{W}_\alpha^R(m)$ is increasing in m for all R, by induction. To do this we recall $\bar{W}_\alpha^1(m)=0\ \forall\ m\in\mathbb{Z}^+$. So from (3.54) we can see that,

$$\alpha^{-1}(1-A)\bar{W}_\alpha^2(m)=\sum_{n=0}^{\infty}E\Big\{\frac{\lambda S^n e^{-(\lambda+\alpha)S}}{n!}\Big\}\big(C(n+m)-C(n+m-1)\big).$$

Therefore we can see that $\alpha^{-1}(1-A)\bar{W}_\alpha^2(m)$ is increasing in m since we know that $C(.)$ is a convex function and satisfies
$C(n+m) - C(n+m-1) \leq C(n+m+1) - C(n+m)\ \forall\ m$ and n. So we conclude that $\bar{W}_\alpha^2(m)$ is increasing in m and we have our initial case for the proof by induction. We now hypothesize that $\bar{W}_\alpha^R(m)$ is increasing in m and infer it for $\bar{W}_\alpha^{R+1}(m)$. Recall the computational scheme (3.54)

$$\begin{aligned}\frac{(1-A)\bar{W}_\alpha^{R+1}(m)}{\alpha} &= \sum_{n=0}^{\infty} E\Big\{\frac{\lambda^n S^n e^{-(\alpha+\lambda)S}}{n!}\Big\}\{C(m+n) - C(m+n-1)\} \\ &\quad + \sum_{n=1}^{\infty} E\Big\{\sum_{r=n}^{\infty} \frac{\lambda^r S^r e^{-(\alpha+\lambda)S} A^{r-n}}{r!}\Big\}\Big\{\frac{(1-A)\bar{W}_\alpha^R(m+n-1)}{\alpha}\Big\}.\end{aligned}$$

We have that $\bar{W}_\alpha^R(m)$ is increasing in m (by the induction hypothesis) and that $C(.)$ is a convex function, hence we can see that

$$\begin{aligned} & \alpha^{-1}(1-A)\bar{W}_\alpha^{R+1}(m) \text{ is increasing in } m \\ \Longrightarrow\quad & \bar{W}_\alpha^{R+1}(m) \text{ is increasing in } m,\end{aligned}$$

since $\alpha^{-1}(1-A)$ is a positive constant. Hence we have shown that $\bar{W}_\alpha^{R+1}(m)$ is increasing in $m\ \forall\ R$. So we now must just complete the final step and infer that $\bar{W}_\alpha(m)$ is increasing in m. To do this we suppose that there exists an m and $m+1$ for which

$$\bar{W}_\alpha(m+1) < \bar{W}_\alpha(m). \tag{3.58}$$

Then by step 5 we can see that this implies that

$$\bar{W}_\alpha^R(m+1) < \bar{W}_\alpha^R(m) \quad \text{for some large enough } R.$$

However we know by step 6 that this is false and therefore we must conclude that (3.58) must also be false. So we have

$$\bar{W}_\alpha(m+1) \geq \bar{W}_\alpha(m)\ \forall\ m, \tag{3.59}$$

which is our final step. So we have completed the proof that $W_\alpha(.)$ is indeed increasing, as required.

We now proceed to Theorem 1, which is the key result needed to establish both that the class is α-indexable and that the state m α-index is given by (3.37). The proof, which is due to Glazebrook, is long and utilises the methods of stochastic dynamic programming, here we just give an outline, for the full proof see Ansell et al (2003b).

Theorem 1

If $\bar{W}_\alpha(m-1) \leq W < \bar{W}_\alpha(m)$ then the policy which chooses the passive action b in states $\{0, 1, \ldots, m-1\}$ and the active action a otherwise is optimal for our service control problem with service charge W, $m \in \mathbb{Z}^+$.

Outline of Proof

Use $\bar{\mathcal{V}}(., \alpha, W)$ to denote the value function for the policy $\bar{u}$ described in the statement of the Theorem. Recall that we have introduced W as a passivity subsidy.

By standard DP theory to prove Theorem 1 we must show that $\bar{\mathcal{V}}(., \alpha, W)$ satisfies the optimality equations (3.26). From (3.26) and straightforward algebra, it suffices to show that when $\bar{W}_\alpha(m-1) \leq W < \bar{W}_\alpha(m)$ we have that.

$$\begin{aligned} W \leq\ & \alpha C(n) + \lambda \bar{\mathcal{V}}(n+1, \alpha, W) - (\alpha+\lambda)\tilde{C}(n, \alpha) \\ & -(\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^{\infty}\frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)t}\bar{\mathcal{V}}(n+r-1, \alpha, W)dG, \ \ n \geq m, \quad (3.60) \end{aligned}$$

and

$$\begin{aligned} W \geq\ & \alpha C(n) + \lambda \bar{\mathcal{V}}(n+1, \alpha, W) - (\alpha+\lambda)\tilde{C}(n, \alpha) \\ & -(\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^{\infty}\frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)t}\bar{\mathcal{V}}(n+r-1, \alpha, W)dG, \ \ 1 \leq n \leq m-1 \quad (3.61) \end{aligned}$$

One can then demonstrate (3.60) and (3.61) by considering the the following four cases separately.

1. $n = m$

2. $n \geq m+1$

3. $n = m - 1 \geq 1$

4. $m - 2 \geq n \geq 1$.

For $n = m$, (3.60) can be shown by firstly finding an expression for $\bar{\mathcal{V}}(m, \alpha, W)$ by consideration of the costs incurred within the first service and those beyond it. Then this new form of $\bar{\mathcal{V}}(m, \alpha, W)$ can be used to find that (3.60) is equivalent to

$$W \leq \alpha C(m) + \lambda\bar{\mathcal{V}}(m+1, \alpha, W) - (\alpha + \lambda)\bar{\mathcal{V}}(m, \alpha, W). \tag{3.62}$$

Similarly an expression for $\bar{\mathcal{V}}(m+1, \alpha, W)$ can be found involving $\bar{\mathcal{V}}(m, \alpha, W)$. Also $\bar{\mathcal{V}}(m, \alpha, W)$ can also be expressed using methods similar to those used to find (3.32). Using these new relations we can then see that

$$\alpha C(m) + \lambda\bar{\mathcal{V}}(m+1, \alpha, W) - (\alpha + \lambda)\bar{\mathcal{V}}(m, \alpha, W) = (1 - A)\bar{W}_\alpha(m) + AW, \tag{3.63}$$

where recall that $A = E(e^{-\alpha T})$. It is then clear that the hypothesis of Theorem 1 and the above expression exceeds W and (3.62) is established. To prove (3.60) holds for $n \geq m + 1$ first introduce the following new notation, use $u(n)$ to denote the policy which chooses the active action at states n and above with the passive action chosen otherwise and $\mathcal{V}^{(n)}$ for the corresponding costs. Note that $u(m) \equiv \bar{u}$ and $\mathcal{V}^{(m)} \equiv \bar{\mathcal{V}}$. Then by calculations similar to (3.32) and (3.35) respectively we can find formulae for $\mathcal{V}^{(n)}(n, \alpha, W)$ and $\mathcal{V}^{(n+1)}(n, \alpha, W)$. We use these formulae to deduce that

$$\mathcal{V}^{(n)}(n, \alpha, W) - \mathcal{V}^{(n+1)}(n, \alpha, W) = \{W - \bar{W}_\alpha(n)\}(1 - A)(\alpha + \lambda - \lambda A)^{-1}, n \in \mathbb{N}. \tag{3.64}$$

Now we take $r \in \mathbb{Z}^+$, and allow the policies to operate from initial state $n + r$. Because each begins with a busy period during which the active action is taken, we have that

$$\mathcal{V}^{(n)}(n + r, \alpha, W) = \bar{C}(n + r, \alpha) + A\mathcal{V}^{(n)}(n + r - 1, \alpha, W),$$

and

$$\mathcal{V}^{(n+1)}(n + r, \alpha, W) = \bar{C}(n + r, \alpha) + A\mathcal{V}^{(n+1)}(n + r - 1, \alpha, W).$$

Then from (3.64) we can see that

$$\begin{aligned}
& \mathcal{V}^{(n)}(n+r,\alpha,W) - \mathcal{V}^{(n+1)}(n+r,\alpha,W) \\
= \;& A\big\{\mathcal{V}^{(n)}(n+r-1,\alpha,W) - \mathcal{V}^{(n+1)}(n+r-1,\alpha,W)\big\} \\
= \;& A^r\{W - \bar{W}_\alpha(n)\}(1-A)(\alpha+\lambda-\lambda A)^{-1}, \quad n,r\in\mathbb{N}. \qquad (3.65)
\end{aligned}$$

We now fix state $M \geq m+1$. Using a lot of algebraic manipulation and the relations (3.65), (3.63) and (3.39) we can find that

$$\begin{aligned}
& \alpha C(M) + \lambda\bar{\mathcal{V}}(M+1,\alpha,W) - (\alpha+\lambda)\tilde{C}(M,\alpha) \\
& -(\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^\infty \frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)t}\bar{\mathcal{V}}(M+r-1,\alpha,W)dG \\
= \;& (1-A)\Big[\bar{W}_\alpha(M) + \sum_{n-m}^{M-1} A^{M-n}\bar{W}_\alpha(n)\Big] + AW - (1-A)\sum_{n=m}^{M-1}A^{M-n}W \\
\geq \;& W, \qquad (3.66)
\end{aligned}$$

as required. Note that inequality (3.66) is a consequence of the fact that

$$\bar{W}_\alpha(n) \geq W, \quad n \geq m.$$

So that establishes (3.60). So we now move on to the outline of showing (3.61) for $n = m-1 \geq 1$. It can be shown that (3.61) is equivalent to

$$\begin{aligned}
W \;\geq\; & \alpha C(m-1) + \lambda\bar{\mathcal{V}}(m,\alpha,W) - (\alpha+\lambda)\tilde{C}(m-1,\alpha) \\
& -(\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^\infty \frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)t}\bar{\mathcal{V}}(m+r-2,\alpha,W)dG \\
= \;& \alpha C(m-1) + \lambda\mathcal{V}^{(m-1)}(m,\alpha,W) - (\alpha+\lambda)\tilde{C}(m-1,\alpha) \\
& (\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^\infty \frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)t}\mathcal{V}^{(m-1)}(m+r-2,\alpha,W)dG \\
& \lambda\{\mathcal{V}^{(m)}(m,\alpha,W) - \mathcal{V}^{(m-1)}(m,\alpha,W)\} - (\alpha+\lambda) \\
& \times\sum_{r=0}^{\infty}\int_0^\infty \Big[\frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)t}\{\mathcal{V}^{(m)}(m+r-2,\alpha,W) \\
& -\mathcal{V}^{(m-1)}(m+r-2,\alpha,W)\}\Big]dG. \qquad (3.67)
\end{aligned}$$

Now note that both policies $u(n)$ and $u(n+1)$ will take the passive action in state

$n+r$ when $r<0$. From this it easily follows that

$$\begin{aligned}
&\mathcal{V}^{(n)}(n+r,\alpha,W)-\mathcal{V}^{(n+1)}(n+r,\alpha,W)\\
=&\ \Big(\frac{\lambda}{\alpha+\lambda}\Big)^{-r}\Big\{\mathcal{V}^{(n)}(n,\alpha,W)-\mathcal{V}^{(n+1)}(n,\alpha,W)\Big\}\\
=&\ \Big(\frac{\lambda}{\alpha+\lambda}\Big)^{-r}W-\bar{W}_{\alpha}(n)\}(1-A)(\alpha+\lambda+\lambda A)^{-1}, n\in\mathbb{N}, r\in\mathbb{Z}^{-}, \quad (3.68)
\end{aligned}$$

by (3.65). Now if we use an appropriate version of the calculation to (3.63) along with (3.65) and (3.68) within (3.67), then use identity (3.39) we obtain that

$$\begin{aligned}
&\alpha C(m-1)+\lambda\bar{\mathcal{V}}(m,\alpha,W)-(\alpha+\lambda)\tilde{C}(m-1,\alpha)\\
&-(\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^{\infty}\frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)t}\bar{\mathcal{V}}(m+r-2,\alpha,W)dG\\
=&\ W+(1-A)A^{-1}\{\bar{W}_{\alpha}(m-1)-W\}\int_0^{\infty}e^{-(\alpha+\lambda)t}dG\\
\le&\ W,
\end{aligned}$$

since $\bar{W}_{\alpha}(m-1)\le W$. So that establishes inequality (3.61) for the case $n=m-1$. So just the final case of the outline of the proof that inequality (3.61) holds for $1\le n\le m-2$. For this case fix state $1\le M\le m-2$, we can then see that (3.61) is equivalent to

$$\begin{aligned}
W\ \ge&\ \alpha C(M)+\lambda\bar{\mathcal{V}}(M+1,\alpha,W)-(\alpha+\lambda)\tilde{C}(M,\alpha)\\
&-(\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^{\infty}\frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)t}\bar{\mathcal{V}}(M+r-1,\alpha,W)dG\\
=&\ \alpha C(M)+\lambda\mathcal{V}^{(M)}(M+1,\alpha,W)-(\alpha+\lambda)\tilde{C}(M,\alpha)\\
&(\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^{\infty}\frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)t}\mathcal{V}^{(M)}(M+r-1,\alpha,W)dG\\
&\lambda\{\sum_{n=M}^{m-1}\mathcal{V}^{(n+1)}(M+1,\alpha,W)-\mathcal{V}^{(n)}(M+1,\alpha,W)\}\\
&-(\alpha+\lambda)\sum_{n=M}^{n-1}\sum_{r=0}^{\infty}\Big[\frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)t}\big\{\mathcal{V}^{(n+1)}(M+r-1,\alpha,W)\\
&-\mathcal{V}^{(n)}(M+r-1,\alpha,W)\big\}\Big]dG. \qquad (3.69)
\end{aligned}$$

We now define the sequences

$$S_r\equiv\Big\{\Big(\frac{\lambda}{\alpha+\lambda}\Big)^{r},\Big(\frac{\lambda}{\alpha+\lambda}\Big)^{r-1},\dots,\Big(\frac{\lambda}{\alpha+\lambda}\Big),1,A,A^2,\dots\Big\}, r\in\mathbb{N}$$

and

$$S_{-r} \equiv \Big\{A^r, A^{r+1}, \dots \Big\} r \in \mathbb{Z}^+.$$

We use $S_{n,r}$ to denote the n^{th} term in the sequence S_r, $n \in \mathbb{Z}^+$, $r \in \mathbb{Z}$. We also observe that, for all choices of $s \geq 0$,

$$\begin{aligned}
&(\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^{\infty}\Big(\sum_{n=1}^{s+2} S_{n,s+2-r}\Big)\frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)t}dG \\
\leq\ & \lambda\sum_{r=0}^{\infty}\Big(\sum_{n=1}^{s+2} S_{n,s+1}\Big)\frac{((\alpha+\lambda)t)^r}{r!}e^{-(\alpha+\lambda)t} \\
\leq\ & \lambda\Big\{\frac{\alpha+\lambda}{\lambda}+\sum_{n=1}^{s+1} S_{n,s}\Big\}. \qquad (3.70)
\end{aligned}$$

Recall also that the first four terms on the r.h.s. of (3.69) when aggregated, are equal to

$$(1-A)\bar{W}_\alpha(M) + AW. \qquad (3.71)$$

Using these sequences and (3.71) we can express the r.h.s. of (3.69) as

$$W + \sum_{n=M}^{m-1}\{\bar{W}_\alpha(n) - W\}a_n, \qquad (3.72)$$

where

$$\begin{aligned}
a_M &= 1 - A + \lambda S_{m-M,m-M-2}(1-A)(\alpha+\lambda+\lambda A)^{-1} \\
&\quad -(\alpha+\lambda)\Big\{\sum_{r=0}^{\infty}\int_0^{\infty} S_{r+1,1}\frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)t}dG\Big\}(1-A)(\alpha+\lambda+\lambda A)^{-1} \\
&= (\alpha+\lambda)\Big\{1-\sum_{r=0}^{\infty}\int_0^{\infty} S_{r+1,1}\frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)t}dG\Big\}(1-A)(\alpha+\lambda+\lambda A)^{-1} \qquad (3.73)
\end{aligned}$$

and

$$\begin{aligned}
a_n &= \Big\{\lambda S_{m-n,m-M-2} - (\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^{\infty} S_{r+1,n-M+1}\frac{(\lambda t)^r}{r!}e^{-(\alpha+\lambda)tdG}\Big\} \\
&\quad \times(1-A)(\alpha+\lambda+\lambda A)^{-1}, M+1 \leq n \leq m-1. \qquad (3.74)
\end{aligned}$$

But from (3.70), (3.73) and (3.74) we can deduce that for all choices of s, $m-1 \geq s \geq M$,

$$\sum_{n=M}^{s} a_n \geq 0. \qquad (3.75)$$

Combining (3.72) and (3.75) we see that the r.h.s. of (3.69) is given by

$$W+\{\bar{W}_\alpha(m-1)-W\}\Big(\sum_{n=M}^{m-1} a_n\Big)+\sum m-1_{n=M}\{\bar{W}_\alpha(n)-\bar{W}_\alpha(n+1)\}\Big(\sum_{r=M}^{n} a_r\Big)\leq W, \tag{3.76}$$

as required. The inequality in (3.76) follows from (3.75) and the assumptions concerning W and the values of $\bar{W}_\alpha$. This concludes the outline of the proof for Theorem 1.

Careful study of the proof of Theorem 1 yields the conclusion that when $\bar{W}_\alpha(m-1) < W < \bar{W}_\alpha(m)$ the policy described in the statement of the theorem is strictly optimal. Suppose now that $W = \bar{W}_\alpha(m)$. It follows from Theorem 1 that for this W-value, the policy which chooses the passive action in states $\{0, 1, \ldots, m\}$ and the active action otherwise is certainly optimal. In the heuristic argument in section 3.3.1 and following where we develop the form of the index, this is policy u_2. Recall that u_1 chooses the passive action in states $\{0, 1, \ldots, m-1\}$ and the active action otherwise. From (3.36) we have that

$$\mathcal{V}_{u_1}\{m, \alpha, \bar{W}_\alpha(m)\} \;=\; \mathcal{V}_{u_2}\{m, \alpha, \bar{W}_\alpha(m)\}.$$

From this and the fact that u_1 and u_2 take the same actions in all states other than m it follows easily that

$$\mathcal{V}_{u_1}\{n, \alpha, \bar{W}_\alpha(m)\} \;=\; \mathcal{V}_{u_2}\{n, \alpha, \bar{W}_\alpha(m)\}, n \in \mathbb{N},$$

and hence that policy u_1 must also be optimal. It follows that when $W = \bar{W}_\alpha(m)$ both actions are optimal in state m. The following result is now immediate.

Theorem 2 (Indexability for the customer class)

The customer class is α-indexable with Whittle α-index $W_\alpha(m) = \bar{W}_\alpha(m)$, $m \in \mathbb{N}$.

Proof

By Theorem 1 and the preceding discussion we have that

$$\Pi_\alpha(W) = \{0, 1, \ldots, m\},\;\; \bar{W}_\alpha(m) \leq W < \bar{W}_\alpha(m+1), m \in \mathbb{N}, \tag{3.77}$$

and the requirements of Definition 1 are met, with α-indexability an immediate consequence. That $\bar{W}_\alpha(m)$ is the Whittle α-index for state m follows from (3.77) and Definition 2.

Comments

1. Hence the α-index is indeed given by expression (3.37). Also the proof of Lemma 2 contains within it a method of computation for the index, expressed by (3.54).

2. We now substantiate the claims made for the Lagrangian relaxation on page 116. Consider class k and its associated allocation problem (k, α, W). We use $\{W^r_{k,\alpha}; r = 0, 1, \ldots, R_k\}$ for the set of *distinct* index values for class k, numbered in ascending order. Note that there will be $R_k + 1$ distinct index values, which may be infinite. Hence we have that $W^0_{k,\alpha} = W_{k,\alpha}(0) = 0$,

$$0 < W^1_{k,\alpha} < W^2_{k,\alpha} < \ldots$$

and

$$\{W^r_{k,\alpha}; r = 0, 1, \ldots, R_k\} = \{W_{k,\alpha}(n); n \in \mathbb{N}\}.$$

For $W \notin \{W^r_{k,\alpha}; r = 0, 1, \ldots, R_k\}$ use $u_k(W)$ for the unique optimal policy for the problem (k, α, W) as given by Theorem 1. If $W = W^r_{k,\alpha}$ for some r then we use $u_k(W)$ to denote the optimal policy which chooses the active action in all states for which both actions are optimal. Developing the notation used in (3.7), we write

$$x^a_{k,n}(m_k, W) = E_{u_k(W)}\Big[\int_0^\infty I\{a_k(t) = a, N_k(t) = n\} e^{-\alpha t} dt | N_k(0) = m_k\Big]$$

for the associated active performance measures, with

$$\sum_{n \in \mathbb{N}} x^a_{k,n}(m_k, W) = E_{u_k(W)}\Big[\int_0^\infty I\{a_k(t) = a\} e^{-\alpha t} dt | N_k(0) = m_k\Big].$$

From the characterization of $u_k(W)$ in Theorem 1, it follows easily that for any choice of m_k and r, $0 \leq r \leq R_{k-1}$,

$$\sum_{n \in \mathbb{N}} x^a_{k,n}(m_k, W) \tag{3.78}$$

is constant for $W \in \left(W_{k,\alpha}^{r}, W_{k,\alpha}^{r+1}\right)$ since in this range $u_k(W)$ does not change. Further, it is left continuous such that for any r, $0 \le r \le R_k$,

$$\lim_{W \uparrow (W_{k,\alpha}^{r})^{-}} \sum_{n \in \mathbb{N}} x_{k,n}^{a}(m_k, W) > \sum_{n \in \mathbb{N}} x_{k,n}^{a}(m_k, \hat{W}), \;\; \hat{W} > W_{k,\alpha}^{r}.$$

Finally it is straightforward to show that

$$\sum_{n \in \mathbb{N}} x_{k,n}^{a}(m_k, W) \to 0, \;\; W \to \infty.$$

To summarise, the quantity in (3.78) when regarded as a function of W is piecewise constant, decreasing with jump discontinuities at distinct index values and tends to 0 as W approaches infinity. These characteristics are inherited in the obvious way by the aggregated quantity

$$\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{a}(m_k, W) = \sum_{k=1}^{K} \sum_{n = \mathbb{N}} x_{k,n}^{a}(\mathbf{m}, W), \tag{3.79}$$

which is the appropriate active performance measure for an optimal policy $\mathbf{u}(W)$ for the K-class stochastic optimisation problem in (3.12) obtained by a super-position of the $u_k(W)$, $1 \le k \le K$ (i.e. independent operation of $u_k(W)$ for each class k). Further, it is a straight forward consequence of the fact that when $W = 0$, $u_k(W)$ takes the active action whenever class k is non-empty, that

$$\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{a}(\mathbf{m}, 0) > \alpha^{-1}\rho + \Theta(\mathbf{m}, \alpha), \tag{3.80}$$

where the constant $\Theta(\mathbf{m}, \alpha)$ is as given in (3.10). Now introduce $W(\mathbf{m}, \alpha)$ as

$$W(\mathbf{m}, \alpha) = \sup\Big\{W; \sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{a}(\mathbf{m}, W) \ge \alpha^{-1}\rho + \Theta(\mathbf{m}, \alpha)\Big\}.$$

By the analysis above, $W(\mathbf{m}, \alpha)$ must be an index value. Suppose that $W(\mathbf{m}, \alpha) = W_{k,\alpha}^{r}$. There are two possibilities. Either

$$\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{a}\{\mathbf{m}, W(\mathbf{m}, \alpha)\} = \alpha^{-1}\rho + \Theta(\mathbf{m}, \alpha), \tag{3.81}$$

in which case policy $\mathbf{u}\{W(m,\alpha)\}$ is optimal for the Langrangian relaxation in (3.12) with $W = W(\mathbf{m},\alpha)$, satisfies the constraint in (3.11) and hence solves Whittle's relaxation. Alternatively

$$\sum_{k=1}^{K}\sum_{n\in\mathbb{N}} x_{k,n}^{a}\{\mathbf{m}, W(\mathbf{m},\alpha)\} > \alpha^{-1}\rho + \Theta(\mathbf{m},\alpha) \tag{3.82}$$

and so the same claims can be made for some randomisation between $\mathbf{u}\{W(\mathbf{m},\alpha)\}$ and an alternative policy which is identical except it replaces the active action by passive in class k states whose index is $W_{k,\alpha}^{r}$.

3. Following Theorem 2 and the discussion in Section 3.2, an index policy for the K-class problem with discounted costs of Section 3.2 is constructed by computing the index function $W_{k,\alpha}(.)$ for each customer class k from an appropriate form of (3.37). At each epoch t, the policy serves a customer from a non-empty class with maximal index $W_{k,\alpha}\{N_k(t)\}$.

3.4 The Undiscounted Problem

We now look at the undiscounted problem. We use the information we have gained from the discounted problem to suggest an index policy for the undiscounted problem. We again drop the class identifier k and observe that we can now develop a suitable Whittle index $W : \mathbb{N} \to \mathbb{R}^{+}$ for the average cost problem from the limit of the corresponding α-index

$$W(m) = \lim_{\alpha\to 0} W_{\alpha}(m) = \lim_{\alpha\to 0} \bar{W}_{\alpha}(m), m \in \mathbb{N}, \tag{3.83}$$

as in Definition 3. Utilising (3.83) within (3.37) we obtain the following result.

Theorem 3 (The Whittle index for average costs)

The Whittle index for the average cost problem is given by $W(0) = 0$ and

$$W(m) = \frac{\lambda\{\bar{C}(m+1) - \bar{C}(m)\} + C(m) - C(m-1)}{E(T)}, m \in \mathbb{Z}^+ \tag{3.84}$$

$$= \frac{E\{C(N+m)\} - E\{C(N+m-1)\}}{E(S)}, m \in \mathbb{Z}^+, \tag{3.85}$$

where in (3.84) we have

$$\bar{C}(m) = \lim_{\alpha \to 0} \alpha^{-1}\bar{C}(m, \alpha) = E\Big[\int_0^T C(N(t))dt|N(0) = m, a\Big], m \in \mathbb{Z}^+,$$

and in (3.85), the random variable N has the steady state distribution of the number of customers present in the single class $M/G/1$ system with non-idling service.

Proof

First notice that

$$E(e^{-\alpha T}) = E\big(1 - T\alpha\big) + 0(\alpha^2). \tag{3.86}$$

Hence when $\alpha \to 0$

$$\begin{aligned} E(e^{-\alpha T}) &\to 1 \\ 1 - E(e^{-\alpha T}) &\to E(T). \end{aligned}$$

Using the above relations the form of the index in (3.84) follows readily from the discounted index (3.37). It also follows from the definitions of the quantities concerned and standard results that

$$E\{C(N+m)\} = \frac{\bar{C}(m+1) + \lambda^{-1}C(m)}{E(T) + \lambda^{-1}}. \tag{3.87}$$

We now write $\{\Pi_k,\, k \in \mathbb{N}\}$ for the steady state distribution of N, the number of customers present in the single class $M/G/1$ system with non-idling service. We know from queueing theory that the probability we are in state 0 is given by

$$\Pi_0 = 1 - \rho = 1 - \lambda E(S),$$

where $E(S)$ is the expected service time. By standard theory, another way to express Π_0 is as the proportion of time spent in the empty state, i.e.

$$\Pi_0 = \frac{\lambda^{-1}}{E(T) + \lambda^{-1}}.$$

Then equating the two expressions for Π_0 allows us to conclude that

$$E(T) = E(S)\big(1 + \lambda E(T)\big). \tag{3.88}$$

Expression (3.85) now follows easily from (3.84), (3.87) and (3.88).

Comment

Following Theorem 3 and the discussion in Section 3.2, an index policy for the K-class service control problem with average costs described in (3.5) of Section 3.2 is constructed by computing the index function $W_k(.)$ for each customer class k from an appropriate form of (3.85). The required (steady state) distribution of a single class $M/G/1$ system is available by standard methods. At each epoch t, the index policy serves a customer from a non-empty class with maximal index $W_k\{N_k(t)\}$.

3.5 Numerical investigation of service index policies for multi-class $M/G/1$ systems

By use of the Langrangian relaxation we have found a class of index heuristics for the multi-class service control problems of Section 3.2 by studying the single class problems with a service charge of Section 3.3.1. An index for the discounted costs problem of (3.4) is obtained as a fair charge for service with an appropriate index for the average costs problem (3.5) obtained as a limit. We now investigate the performance of the index heuristics numerically. We will do this by comparing the expected index costs with the expected optimal cost for problems with two customer classes. We shall also use simulation to compare costs for our index policy with those of competitor policies for problems with five customer classes. For such five class problems, the direct calculation of the expected costs would prove computationally intractable. While our prime focus will be on average cost problems we begin with a study of some two class problems with discounted costs.

3.5.1 Discounted costs problems with two customer classes

In this section we look at a problem of the type described in Section 3.2, where we have two customer classes. We consider the following four cost rate structures:

(a) $C_1(n) = b_1 n + 2n^2$; $\quad C_2(n) = b_2 n + 2n^2$; (quadratic)

(b) $C_1(n) = b_1 n^2 + 2n^3$; $\quad C_2(n) = b_2 n^2 + 2n^3$; (cubic)

(c) $C_1(n) = b_1 n^3 + 2n^4$; $\quad C_2(n) = b_2 n^3 + 2n^4$; (quartic)

(d) $C_1(n) = b_1(n-2)^+ + 2\{(n-2)^+\}^2$; $\quad C_2(n) = b_2(n-2)^+ + 2\{(n-2)^+\}^2$; (shifted quadratic).

Contained in tables 3.1 - 3.32 are the results of part of a study comparing the discounted costs incurred by the index heuristic described in Comment 3 following Theorem 2 with those incurred by an optimal policy for a range of service control problems with two customer classes. Each table 3.1 - 3.32 corresponds to the above four cost structures (a) - (d) as indicated on the table labels. In these tables, the first row gives the starting state for the first customer class, and the first column gives the starting state for the second customer class. The caption in each table contains in it a bracketed triple denoting the parameters of that problem. The first two entries of this triple indicate respectively the choice of cost coefficients b_1, b_2 with the final labels 1, $1'$, 2 and $2'$ specifying features of the stochastic structure. Labels 1, $1'$ denote problems for which $S_1 \sim \Gamma(2, 1.25)$, $S_2 \sim \Gamma(3, 2.25)$ and

$\lambda_1 = 0.20$. For case 1, λ_2 is chosen such that the value of the traffic intensity ρ is 0.60, while for case $1'$, ρ is set to be 0.85. The labels 2, $2'$ denote problems with $S_1 \sim \Gamma(2,1)$, $S_2 \sim \Gamma(3,3)$ and $\lambda_1 = 0.20$, hence the mean service times are further apart than in 1, $1'$. Again in case 2, λ_2 is chosen to yield $\rho = 0.60$ while for $2'$ we have $\rho = 0.85$. The top value in each cell of the table is the discounted cost for the index policy, with the corresponding optimal cost shown below it.

In each case the the fully optimal policy is found using dynamic programming techniques and the costs are found by use of DP value iteration; see Chapter 3 of Tijms (1994). It is possible to use such methods for problems of this size, but computationally expensive.

state	0	1	2	3	4
0	97.9099	107.4536	134.6200	184.7400	262.7000
	97.8780	107.4190	134.5756	184.6722	262.5843
1	112.9996	137.4599	178.6827	243.3309	336.0253
	112.9623	137.4102	178.6027	243.1842	335.7478
2	152.0503	190.7346	252.4624	336.3064	448.0414
	151.9936	190.6392	252.2868	335.9471	447.3186
3	221.0508	274.4949	356.6669	465.1315	600.2272
	220.9429	274.2843	356.2428	464.2181	598.2495
4	325.3263	394.0378	496.1470	633.5953	794.1842
	325.1052	393.5658	495.1201	628.0944	789.0664

Table 3.1: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems with two customer classes, with parameters denoted by* $(2,1,1)$.

state	0	1	2	3	4
0	96.3177	107.0332	136.6032	190.3174	272.9647
	96.2575	106.9677	136.5186	190.1900	272.7702
1	109.7686	133.5806	176.3111	243.6770	340.1883
	109.6985	133.4863	176.1546	243.3934	339.7544
2	145.1065	185.6069	247.0235	332.7542	447.1806
	145.0004	185.4280	246.6708	332.0213	446.2219
3	209.3751	266.0313	351.6516	459.1784	594.9018
	209.1794	265.6390	349.5852	457.3180	593.1413
4	307.9122	381.2022	487.0522	622.1404	782.8971
	307.5471	380.3873	485.2095	620.2585	780.9608

Table 3.2: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems with two customer classes with parameters denoted by* $(1,2,1)$.

state	0	1	2	3	4
0	268.1926	287.8366	340.0128	429.4488	560.2975
	263.7965	283.1198	334.5410	422.9741	552.7386
1	297.9327	345.1218	423.3403	537.0395	690.3825
	293.0465	338.9149	415.2020	527.1207	678.9212
2	366.3270	447.0645	555.3085	696.2495	874.6288
	359.7742	436.0634	541.4663	680.4583	857.3682
3	480.1059	596.9210	735.6682	905.6771	1111.4933
	469.9933	573.6368	713.0714	882.2169	1087.1390
4	642.1767	779.9106	957.7787	1161.2508	1398.7699
	626.7104	757.3364	926.3101	1130.6713	1366.7081

Table 3.3: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems with two customer classes with parameters denoted by (2,1,1').*

state	0	1	2	3	4
0	270.5770	291.3575	345.9148	439.1252	575.1806
	267.7749	288.3548	342.4138	434.8776	570.0112
1	298.6695	343.8673	422.2284	538.1533	695.6364
	295.5476	339.9539	417.0959	531.5887	687.5602
2	361.9004	438.7417	546.0461	688.6325	870.7782
	357.7339	432.7368	537.4957	677.8644	857.9374
3	468.4298	580.8376	719.0607	891.0665	1100.6201
	462.0042	566.7095	703.9019	873.5501	1080.7324
4	622.7501	760.4596	937.5622	1143.8976	1384.2716
	612.2763	744.8813	914.8480	1117.6770	1354.9300

Table 3.4: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems with two customer classes with parameters denoted by (1,2,1').*

state	0	1	2	3	4
0	102.1608	108.4329	126.8771	161.7071	216.8526
	101.9858	108.2507	126.6692	161.4407	216.4882
1	120.9491	138.9311	169.2265	216.0841	283.3169
	120.7377	138.6749	168.8639	215.5061	282.4620
2	168.0558	201.8883	247.7642	310.4601	393.0207
	167.7143	201.3676	246.9154	308.8274	390.8570
3	251.1120	299.8022	368.9451	449.1828	549.1324
	250.4517	298.6307	363.7200	444.3485	544.5226
4	375.5996	439.8294	528.5174	630.4170	751.6317
	374.3650	437.4312	523.8478	625.8179	746.6019

Table 3.5: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems with two customer classes with parameters denoted by (2,1,2).*

state	0	1	2	3	4
0	98.4711	105.2637	124.4684	160.1065	215.9192
	97.6724	104.4267	123.4851	158.9081	214.4833
1	116.9331	133.9756	164.1790	210.7638	277.5065
	115.9678	132.7717	162.2630	208.3579	274.7413
2	163.8483	199.4060	242.9869	303.1766	383.6063
	162.3281	194.0062	237.9625	298.3383	378.7785
3	248.1218	299.1940	360.6891	438.8753	536.7236
	245.5057	294.3288	355.8425	433.3997	530.5768
4	375.3203	441.6154	528.1996	625.7281	743.4560
	371.9915	436.5399	519.0374	616.5602	732.9569

Table 3.6: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems with two customer classes with parameters denoted by (1,2,2).*

state	0	1	2	3	4
0	261.9274	273.2460	304.6781	360.8762	446.0644
	261.7704	273.0833	304.5005	360.6729	445.8237
1	303.6363	334.8294	385.6722	460.6340	563.8232
	303.4515	334.6161	385.4151	460.3150	563.4240
2	391.5918	447.3044	521.3062	618.0908	741.8122
	391.3240	446.9694	520.8752	617.5295	741.0947
3	533.7979	616.9206	716.1511	836.6658	982.6127
	533.3527	616.3383	715.3673	835.6061	981.2731
4	735.2171	844.5321	971.1808	1117.4324	1287.1052
	734.4319	843.5013	969.6900	1115.2893	1284.6054

Table 3.7: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems with two customer classes with parameters denoted by (2,1,2').*

state	0	1	2	3	4
0	261.8411	274.0385	307.2367	366.1411	454.9985
	261.6491	273.8393	307.0185	365.8923	454.7094
1	301.5502	331.0230	381.5181	457.4739	562.9375
	301.3253	330.7602	381.1978	457.0808	562.4648
2	383.0290	435.3447	507.7364	604.5094	729.6550
	382.7027	434.9221	507.1793	603.8131	728.8363
3	515.9370	594.9014	692.2450	812.1852	958.9126
	515.3917	594.1302	691.1523	810.9071	957.5305
4	706.7309	815.6819	938.5455	1082.6345	1252.0726
	705.7699	813.1550	936.2643	1080.5026	1250.0379

Table 3.8: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems with two customer classes with parameters denoted by (1,2,2').*

state	0	1	2	3	4
0	211.9724	231.9491	305.4685	479.5791	817.9163
	211.9462	231.9207	305.4323	479.5243	817.8213
1	245.3607	305.4330	420.6748	645.3533	1042.2161
	245.3301	305.3921	420.6101	645.2376	1041.9941
2	357.3472	461.5940	662.2616	971.9395	1463.8086
	357.3001	461.5160	662.1209	971.6656	1463.2590
3	610.3339	768.9518	1060.5649	1517.1287	2152.1354
	610.2422	768.7780	1060.2219	1516.4533	2150.7460
4	1084.9786	1308.4270	1701.5124	2313.9543	3169.1281
	1084.7774	1308.0187	1700.6713	2312.2229	3165.6114

Table 3.9: *Comparative performance of the index heuristic and the optimal policy with various starting states for the cubic discounted costs problems with two customer classes and parameters denoted by (2,1,1).*

state	0	1	2	3	4
0	208.0710	229.5188	309.4655	497.9836	861.4387
	207.7231	229.1413	308.9819	497.2445	860.1765
1	238.9141	296.5405	416.1317	652.8269	1072.5299
	238.5081	295.9991	415.2591	651.2304	1069.5056
2	340.6820	447.5798	645.3872	964.1093	1476.4896
	340.0625	446.5401	643.4749	960.2079	1468.6339
3	574.3526	738.4834	1038.2687	1495.4702	2152.0010
	573.1738	736.1858	1033.6464	1485.5708	2130.5979
4	1017.8894	1251.4416	1661.0171	2328.6244	3175.6882
	1015.4641	1246.2787	1649.8163	2271.5294	3120.3114

Table 3.10: *Comparative performance of the index heuristic and the optimal policy with various starting states for the cubic discounted costs problems with two customer classes and parameters denoted by (1,2,1).*

state	0	1	2	3	4
0	932.9235	1000.2929	1206.0746	1617.8101	2322.7824
	914.5342	980.6092	1182.9740	1589.0409	2286.1943
1	1038.2904	1209.2900	1528.8974	2063.7101	2899.7034
	1017.7576	1183.9754	1495.5502	2019.0300	2841.0296
2	1299.2972	1629.5610	2108.6471	2815.9532	3831.7929
	1272.1775	1591.1161	2055.1363	2741.2275	3734.0206
3	1799.8973	2288.3698	3013.2301	3961.9206	5217.5595
	1759.1490	2230.6426	2921.9657	3830.2233	5053.2583
4	2626.9641	3306.1039	4350.4741	5586.3710	7139.9202
	2562.0487	3210.4043	4122.3189	5349.7532	6874.0626

Table 3.11: *Comparative performance of the index heuristic and the optimal policy with various starting states for the cubic discounted costs problems with two customer classes and parameters denoted by (2,1,1').*

state	0	1	2	3	4
0	949.8748	1019.9609	1235.9398	1669.0089	2409.4596
	924.7464	993.0326	1204.5408	1630.9075	2363.0505
1	1054.1897	1221.9745	1547.2835	2098.1876	2963.3254
	1026.1941	1186.8817	1501.2551	2039.3474	2890.9427
2	1304.1536	1626.9180	2111.1602	2825.7780	3860.2126
	1266.8067	1573.0899	2034.4534	2729.3889	3745.5557
3	1785.0196	2324.3807	3010.4232	3939.1884	5196.6432
	1727.5132	2196.3500	2874.5315	3782.9542	5020.6610
4	2587.3720	3281.5215	4279.7655	5494.9237	7049.6210
	2494.2130	3142.4260	4056.1373	5266.3561	6795.5851

Table 3.12: *Comparative performance of the index heuristic and the optimal policy with various starting states for the cubic discounted costs problems with two customer classes and parameters denoted by (1,2,1').*

state	0	1	2	3	4
0	229.8926	242.2890	290.4940	408.5210	643.1619
	229.8295	242.2234	290.4201	408.4288	643.0338
1	274.2022	318.9822	402.4289	561.1136	841.5162
	274.1258	318.8901	402.3037	560.9257	841.2109
2	416.2863	510.4366	660.8111	890.7007	1248.0301
	416.1611	510.2481	660.5298	890.2212	1247.1633
3	733.1247	883.8315	1137.4807	1479.7660	1955.4637
	732.8717	883.3962	1136.7266	1478.4330	1952.7427
4	1319.1944	1540.5222	1898.9274	2419.7744	3058.1167
	1318.6684	1539.5376	1897.0643	2410.5191	3049.7922

Table 3.13: *Comparative performance of the index heuristic and the optimal policy with various starting states for the cubic discounted costs problems with two customer classes and parameters denoted by (2,1,2).*

state	0	1	2	3	4
0	227.1538	239.5258	288.4229	408.1899	645.3813
	226.5507	238.9021	287.7263	407.3265	644.2101
1	273.0391	315.6666	398.2874	556.6299	837.0589
	272.3019	314.7928	397.0983	554.8148	834.2576
2	419.6999	515.3319	659.4176	885.6963	1239.0029
	418.4912	513.4906	656.6687	880.6972	1231.2603
3	749.0497	902.2493	1159.8487	1488.5031	1954.0881
	746.6884	898.0763	1146.9151	1473.7356	1932.8069
4	1361.3104	1585.9638	1951.8892	2476.3916	3087.2919
	1356.7794	1577.2125	1934.7206	2420.6893	3036.8903

Table 3.14: *Comparative performance of the index heuristic and the optimal policy with various starting states for the cubic discounted costs problems with two customer classes and parameters denoted by (1,2,2).*

state	0	1	2	3	4
0	917.1907	952.5356	1066.9187	1308.2232	1739.6531
	916.9421	952.2785	1066.6387	1307.9030	1739.2717
1	1072.8934	1186.3585	1386.6322	1720.3025	2249.5761
	1072.6003	1186.0220	1386.2297	1719.8053	2248.9475
2	1432.7432	1671.8097	2001.5714	2469.5683	3137.2543
	1432.3186	1671.2844	2000.9059	2468.7109	3136.1373
3	2110.0005	2520.0584	3028.0820	3678.5397	4531.7370
	2109.2961	2519.1518	3026.9010	3676.9868	4529.6819
4	3229.0516	3812.0941	4568.5437	5457.9736	6553.1389
	3227.8030	3810.4544	4566.3846	5455.1015	6549.2897

Table 3.15: *Comparative performance of the index heuristic and the optimal policy with various starting states for the cubic discounted costs problems with two customer classes and parameters denoted by (2,1,2').*

state	0	1	2	3	4
0	917.7822	954.3979	1074.4090	1328.7461	1783.3376
	915.7203	952.2504	1072.0450	1326.0710	1780.3187
1	1070.7771	1178.4315	1377.9658	1718.6996	2265.3577
	1068.3791	1175.5729	1374.4130	1714.4558	2260.5740
2	1410.8360	1636.5259	1960.1292	2428.7216	3107.0207
	1407.3815	1631.8098	1953.5724	2421.3372	3099.4143
3	2050.2732	2457.0631	2945.3810	3585.3750	4439.8562
	2044.6374	2441.2427	2931.8652	3573.7206	4429.6664
4	3115.1898	3693.7571	4420.2222	5293.2644	6382.9373
	3106.3111	3681.8486	4409.7951	5283.9417	6374.3554

Table 3.16: *Comparative performance of the index heuristic and the optimal policy with various starting states for the cubic discounted costs problems with two customer classes and parameters denoted by (1,2,2').*

state	0	1	2	3	4
0	598.9984	653.6202	876.4294	1514.6884	3031.7818
	598.9022	653.5159	876.2968	1514.4882	3031.4356
1	695.2685	881.8920	1252.6973	2089.1081	3857.8318
	695.1561	881.7419	1252.4603	2088.6864	3857.0274
2	1056.1241	1403.5274	2124.3925	3333.2768	5564.1253
	1055.9511	1403.2417	2123.8768	3332.2811	5562.1462
3	2035.6164	2604.4373	3730.3524	5720.4911	8751.3045
	2035.2770	2603.7992	3729.1024	5718.0374	8746.3367
4	4260.7526	5123.4998	6757.9191	9623.4757	14135.7959
	4259.9972	5121.9860	6754.8426	9617.2095	14123.2900

Table 3.17: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quartic discounted costs problems with two customer classes and parameters denoted by (2,1,1).*

state	0	1	2	3	4
0	585.8202	642.3468	881.5399	1571.8829	3206.1336
	585.5572	642.0616	881.1770	1571.3331	3205.1798
1	676.7107	854.2440	1234.5347	2113.4351	3987.7515
	676.4034	853.8341	1233.8854	2112.2742	3985.5245
2	1007.3345	1357.5213	2059.6062	3294.9491	5611.5085
	1006.8624	1356.7386	2058.1933	3292.2015	5605.9968
3	1909.9866	2485.8847	3627.3283	5587.9664	8675.4881
	1909.0653	2484.1405	3623.8869	5581.1912	8661.5591
4	3976.5066	4855.1859	6521.5054	9431.0998	13909.7046
	3974.4850	4851.0853	6513.0642	9413.7328	13874.4594

Table 3.18: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quartic discounted costs problems with two customer classes and parameters denoted by (1,2,1).*

state	0	1	2	3	4
0	4254.7857	4555.8533	5516.2100	7609.8739	11629.6276
	4167.4051	4462.4044	5406.1117	7470.1363	11444.7356
1	4747.6239	5537.7253	7085.4948	9872.8491	14700.2353
	4649.8939	5418.5748	6928.8388	9657.8100	14401.8234
2	5949.3590	7546.1818	9988.9720	13831.3237	19874.7190
	5821.2323	7372.9450	9744.1560	13478.5270	19371.8435
3	8393.0375	10974.8319	14893.8793	20324.2086	28190.1371
	8203.0809	10705.6782	14491.5049	19723.3966	27315.6554
4	12855.4714	16678.6065	22270.5616	30310.9502	40894.0045
	12552.6169	16233.5593	21614.0987	29261.9036	39318.4880

Table 3.19: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quartic discounted costs problems with two customer classes and parameters denoted by (2,1,1').*

state	0	1	2	3	4
0	4376.4714	4689.0530	5696.5612	7903.3522	12140.6006
	4205.4300	4505.8335	5482.4753	7641.2232	11816.0592
1	4877.6338	5671.8573	7264.1830	10155.4299	15178.1218
	4686.9302	5434.1086	6952.0785	9750.3653	14667.6115
2	6067.5064	7669.4031	10176.3415	14100.6594	20286.1876
	5813.9393	7308.5058	9662.3812	13434.4095	19464.4590
3	8476.2339	11357.9315	15166.7209	20592.8523	28466.4399
	8088.5914	10570.6126	14267.3824	19490.0833	27164.4230
4	12880.5618	16847.4148	22894.9213	30515.5056	40841.6258
	12254.0940	15914.4737	21280.4403	28799.0255	38862.8653

Table 3.20: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quartic discounted costs problems with two customer classes and parameters denoted by (1,2,1').*

state	0	1	2	3	4
0	681.1326	713.8403	854.4677	1274.4443	2303.2207
	681.1326	713.8403	854.4677	1274.4443	2303.2207
1	817.1657	960.3496	1227.5133	1805.1928	3024.9891
	817.1657	960.3496	1227.5133	1805.1928	3024.9890
2	1302.6125	1627.6920	2176.5351	3068.0702	4656.8152
	1302.6125	1627.6919	2176.5350	3068.0701	4656.8150
3	2585.9084	3149.2140	4157.9534	5663.5269	7921.5990
	2585.9084	3149.2139	4157.9532	5663.5266	7921.5985
4	5429.3678	6328.1033	7874.7147	10379.6802	13779.7543
	5429.3676	6328.1031	7874.7142	10379.6795	13779.7532

Table 3.21: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quartic discounted costs problems with two customer classes and parameters denoted by (2,1,2).*

state	0	1	2	3	4
0	693.9780	724.9232	862.4337	1279.5328	2305.6620
	693.6048	724.5375	862.0078	1279.0225	2304.9940
1	841.0151	980.6620	1241.6407	1810.7443	3019.4434
	840.5586	980.1202	1240.9299	1809.7441	3017.9378
2	1363.7157	1686.9184	2216.9937	3089.5186	4651.9089
	1362.9533	1685.7964	2215.4058	3087.0523	4647.8966
3	2738.9551	3297.2904	4307.5916	5758.5810	7965.9287
	2737.3828	3294.6709	4303.1959	5751.9480	7954.4479
4	5783.2449	6671.2175	8221.5445	10734.0473	14006.5223
	5779.8487	6665.0994	8210.4524	10714.3068	13973.0809

Table 3.22: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quartic discounted costs problems with two customer classes and parameters denoted by (1,2,2).*

state	0	1	2	3	4
0	4236.5454	4385.0623	4879.5851	6017.2164	8316.0471
	4233.1641	4381.5642	4875.7727	6012.8532	8310.8485
1	4988.9145	5522.7622	6468.9490	8124.3258	11007.4696
	4984.9289	5518.1844	6463.4646	8117.5331	10998.8715
2	6748.5389	7954.9984	9649.1707	12136.4145	15936.7162
	6742.7699	7947.8582	9640.0912	12124.6409	15921.3168
3	10299.7277	12574.5381	15432.8043	19194.3843	24382.5933
	10290.1829	12562.2271	15416.6599	19172.8356	24353.7351
4	16802.1511	20293.7965	24983.1200	30663.2276	37911.3547
	16785.3894	20271.6531	24953.9586	30622.6164	37854.8882

Table 3.23: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quartic discounted costs problems with two customer classes and parameters denoted by (2,1,2').*

state	0	1	2	3	4
0	4239.4764	4389.8497	4901.5114	6094.7140	8518.2074
	4221.4696	4371.1425	4880.9771	6071.3099	8491.1796
1	4988.4203	5499.0647	6438.1980	8124.5724	11106.1739
	4967.3711	5474.3508	6407.8402	8087.4531	11062.1616
2	6679.6067	7830.1129	9492.0527	11978.7591	15840.0992
	6649.2369	7790.3540	9438.3040	11912.8853	15764.9312
3	10069.6481	12323.7438	15082.2602	18790.2574	23974.7770
	10019.9415	12223.4191	14976.8390	18671.4974	23852.8086
4	16289.9515	19747.4036	24404.2044	29898.5637	37027.6531
	16208.6761	19638.2502	24185.2168	29710.8942	36865.0698

Table 3.24: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quartic discounted costs problems with two customer classes and parameters denoted by (1,2,2').*

state	0	1	2	3	4
0	7.2059	7.8343	10.4585	20.6069	44.7342
	7.2047	7.8330	10.4568	20.6043	44.7299
1	8.3941	10.8467	15.5756	28.6818	56.6597
	8.3927	10.8448	15.5726	28.6765	56.6495
2	13.1472	17.8287	28.0376	46.9557	82.2388
	13.1450	17.8251	28.0311	46.9431	82.2135
3	29.8404	37.7832	54.0322	86.4038	134.4766
	29.8362	37.7752	54.0164	86.3727	134.4126
4	66.0017	78.3146	102.2859	147.6468	215.1555
	65.9925	78.2959	102.2473	147.5671	214.9934

Table 3.25: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems where costs are not incurred below state 2 with two customer classes and parameters denoted by (2,1,1).*

state	0	1	2	3	4
0	6.8896	7.5211	10.4232	22.4467	49.8845
	6.8874	7.5187	10.4201	22.4420	49.8764
1	7.9935	10.2098	15.0538	29.7884	60.7622
	7.9909	10.2063	15.0482	29.7784	60.7430
2	12.1162	16.7950	26.3586	46.4602	84.1570
	12.1123	16.7884	26.3466	46.4364	84.1084
3	26.2135	34.2117	50.7279	82.4612	132.0501
	26.2058	34.1969	50.6985	82.4021	131.9241
4	58.0657	70.6304	95.2803	142.4656	209.6702
	58.0492	70.5963	95.2085	142.3164	209.3433

Table 3.26: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems where costs are not incurred below state 2 with two customer classes and parameters denoted by (1,2,1).*

state	0	1	2	3	4
0	53.2241	57.0021	69.2034	97.4051	149.6032
	52.1260	55.8266	67.8226	95.6832	147.4204
1	59.3655	69.4123	89.0901	125.9033	187.6939
	58.1395	67.9020	87.0948	123.2124	184.1865
2	74.5350	95.3767	126.4648	176.2732	252.0502
	72.9183	93.1521	123.2633	171.7075	246.2152
3	109.0767	141.5486	193.0676	261.9866	357.4321
	106.6656	138.1118	185.8264	253.9127	347.8137
4	171.2264	218.1757	286.3200	383.6363	505.0237
	167.4682	212.5535	277.9705	371.2108	490.0522

Table 3.27: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems where costs are not incurred below state 2 with two customer classes and parameters denoted by (2,1,1').*

state	0	1	2	3	4
0	54.4016	58.3026	71.3262	102.3833	159.2024
	53.3628	57.1909	70.0213	100.7569	157.1363
1	60.6004	70.2253	90.2197	129.3911	195.2625
	59.4404	68.7956	88.3361	126.8578	191.9390
2	74.9362	94.5352	125.4704	177.0741	256.1701
	73.4031	92.4156	122.4448	172.8048	250.6045
3	106.4513	138.6779	188.4639	258.6192	355.9323
	104.1404	135.4085	183.2936	250.9608	346.5437
4	164.3196	211.4271	286.5230	378.9838	498.6711
	160.5847	205.8960	272.0895	364.8394	483.6678

Table 3.28: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems where costs are not incurred below state 2 with two customer classes and parameters denoted by (1,2,1').*

state	0	1	2	3	4
0	8.6761	9.0541	10.6779	17.4108	34.0219
	8.4078	8.7653	10.3079	16.9946	33.5829
1	10.4546	12.8539	16.0903	25.0052	44.3879
	10.1415	12.0260	15.3422	24.3250	43.7623
2	17.1448	21.7838	29.3899	43.1388	68.2917
	16.7136	21.0855	28.7493	42.5389	67.7089
3	38.5356	46.4882	60.8474	85.0045	120.8406
	38.0675	45.8744	60.2515	84.4034	120.1685
4	83.3838	96.1263	118.4176	157.3502	207.8021
	82.8981	95.5293	117.7586	156.5446	206.6977

Table 3.29: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems where costs are not incurred below state 2 with two customer classes and parameters denoted by (2,1,2).*

state	0	1	2	3	4
0	8.6704	9.0065	10.5565	17.9662	35.7497
	8.3285	8.6405	10.0814	17.4223	35.1604
1	10.5580	12.9670	15.9724	25.3469	45.6218
	10.1524	11.8878	14.9798	24.4157	44.7147
2	17.4510	22.0615	28.9622	42.8104	68.3545
	16.8829	21.1152	28.0524	41.8736	67.2815
3	38.3465	46.1388	60.4130	83.1926	118.5509
	37.6825	45.2012	59.3574	81.9076	116.5619
4	83.1134	95.5966	117.8899	158.1127	205.8851
	82.3220	94.4631	116.2963	152.6469	200.8014

Table 3.30: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems where costs are not incurred below state 2 with two customer classes and parameters denoted by (1,2,2).*

state	0	1	2	3	4
0	52.8168	54.6619	60.9671	77.0778	108.9355
	52.7911	54.6353	60.9382	77.0447	108.8960
1	62.2110	68.9239	80.9280	103.4952	142.4702
	62.1807	68.8891	80.8864	103.4438	142.4051
2	84.3403	99.8519	121.2924	154.1405	203.9562
	84.2964	99.7976	121.2236	154.0518	203.8406
3	132.7915	160.7817	197.7619	245.8991	310.9742
	132.7187	160.6879	197.6398	245.7384	310.7614
4	218.5776	260.0223	315.4989	383.3894	467.6202
	218.4486	259.8528	315.2761	383.0923	467.2209

Table 3.31: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems where costs are not incurred below state 2 with two customer classes and parameters denoted by (2,1,2').*

state	0	1	2	3	4
0	52.9643	54.8338	61.5402	79.4725	114.4473
	52.8245	54.6890	61.3818	79.2909	114.2326
1	62.3415	68.5181	80.3712	104.2003	145.7126
	62.1772	68.3282	80.1412	103.9132	145.3551
2	82.9807	97.1362	117.8182	151.1991	202.7624
	82.7432	96.8385	117.4298	150.6845	202.1172
3	126.8854	154.4310	188.8808	236.3851	302.0260
	126.4946	153.9273	188.1660	235.3701	300.8376
4	206.0873	246.9705	302.0525	367.3275	450.6047
	205.4204	246.0781	299.7017	365.2045	448.6193

Table 3.32: *Comparative performance of the index heuristic and the optimal policy with various starting states for the quadratic discounted costs problems where costs are not incurred below state 2 with two customer classes and parameters denoted by (1,2,2').*

3.5.2 Average cost problems with two customer classes

Contained in table 3.33 below are the results of part of a study comparing the average cost rates incurred by the index heuristic described in the comment following Theorem 3 with those incurred by an optimal policy. Again the optimal policies were found using dynamic programming techniques, and the cost rates by DP value iteration. All the service control problems studied here have two customer classes. Each cell in the body of the table gives results for four different cost structures in the form

a	(a)	b	(b)
c	(c)	d	(d)

The corresponding cost rates are as follows:

a) $C_1(n) = b_1 n + 2n^2$; $C_2(n) = b_2 n + 2n^2$; (quadratic)

b) $C_1(n) = b_1 n^2 + 2n^3$; $C_2(n) = b_2 n^2 + 2n^3$; (cubic)

c) $C_1(n) = b_1 n^3 + 2n^4$; $C_2(n) = b_2 n^3 + 2n^4$; (quartic)

d) $C_1(n) = b_1(n-2)^+ + 2\{(n-2)^+\}^2$; $C_2(n) = b_2(n-2)^+ + 2\{(n-2)^+\}^2$; (shifted quadratic).

In all cases the time average cost for the index policy is given by the unbracketed figure (a, b, c or d) is, with the corresponding optimal cost in brackets. The first column of Table 3.33, lists the cost coefficients b_1, b_2 which apply to the values in the corresponding row. In the main body of the table the left hand side concerns a server control problem with $S_1 \sim \Gamma(2, 1.25)$, $S_2 \sim \Gamma(3, 2.25)$, $\lambda_1 = 0.20$ and λ_2 is chosen to give traffic intensity of 0.60. The value of λ_2 is modified for the figures on the right hand side to give traffic intensity of 0.85.

b_1	b_2	$\rho = 0.60$				$\rho = 0.85$			
0.10	0.10	2.0727	(2.0727)	4.2932	(4.2930)	7.7935	(7.7928)	39.9968	(39.9819)
		11.7500	(11.7500)	0.2160	(0.2160)	289.3491	(289.5327)	3.2089	(3.2086)
0.10	0.20	2.2337	(2.2334)	4.6542	(4.6531)	8.9225	(8.9122)	46.5323	(46.5011)
		12.9877	(12.9834)	0.2318	(0.2318)	343.8621	(343.2882)	3.6701	(3.6661)
0.10	0.50	2.5564	(2.5530)	5.4729	(5.4729)	10.4631	(10.4407)	58.7729	(58.5350)
		15.9330	(15.5675)	0.2688	(0.2661)	448.4251	(446.8978)	4.3366	(4.3228)
0.10	1.00	2.9461	(2.9458)	6.5898	(6.5808)	12.2439	(12.2382)	71.3354	(70.9773)
		19.2520	(19.1304)	0.3122	(0.3019)	555.7236	(554.9071)	5.0380	(4.9689)
0.10	2.00	3.7181	(3.7181)	8.4563	(8.4269)	15.7226	(15.7224)	89.1588	(88.6533)
		25.6285	(24.9345)	0.3729	(0.3715)	702.9505	(698.2963)	6.0162	(5.9198)
0.20	0.10	2.1649	(2.1649)	4.5764	(4.5764)	8.2195	(8.2195)	44.2131	(44.2127)
		12.7201	(12.7201)	0.2253	(0.2253)	328.6532	(328.3382)	3.3805	(3.3804)
0.20	0.20	2.3407	(2.3407)	4.9506	(4.9506)	9.8432	(9.8335)	52.6082	(52.6079)
		13.9763	(13.9763)	0.2420	(0.2420)	396.5049	(396.5009)	4.0348	(4.0311)
0.20	0.50	2.7172	(2.7107)	5.9251	(5.8729)	12.1076	(12.0930)	68.5622	(68.2829)
		17.1763	(16.9382)	0.2824	(0.2819)	530.7424	(529.6970)	4.9834	(4.9777)
0.20	1.00	3.1339	(3.1325)	7.0314	(7.0313)	14.2092	(14.1831)	84.6577	(84.0422)
		20.7434	(20.7342)	0.3288	(0.3200)	670.5373	(667.4028)	5.8720	(5.8022)
0.20	2.00	3.9077	(3.9075)	8.9965	(8.9851)	17.7387	(17.7320)	105.5860	(105.1594)
		27.3193	(26.9502)	0.3911	(0.3911)	850.6627	(846.7620)	6.9959	(6.9090)
0.50	0.10	2.4343	(2.4343)	5.3592	(5.3592)	8.8808	(8.8792)	52.7127	(52.7122)
		15.4227	(15.4227)	0.2525	(0.2525)	413.0458	(413.0351)	3.6408	(3.6402)
0.50	0.20	2.6317	(2.6317)	5.8035	(5.8035)	11.3447	(11.3446)	65.5261	(65.4681)
		16.8787	(16.8725)	0.2707	(0.2707)	515.2392	(515.1302)	4.6367	(4.6367)
0.50	0.50	3.0962	(3.0961)	6.9233	(6.9231)	15.6224	(15.5923)	90.0431	(90.0192)
		20.6543	(20.6528)	0.3169	(0.3169)	720.0842	(719.3984)	6.3695	(6.3582)
0.50	1.00	3.6383	(3.6377)	8.2931	(8.2880)	19.1765	(19.1429)	115.1593	(114.8845)
		25.0046	(25.0046)	0.3789	(0.3704)	932.7753	(931.1808)	7.8445	(7.8266)
0.50	2.00	4.4710	(4.4683)	10.5650	(10.5287)	23.5551	(23.5206)	146.3636	(146.3157)
		32.4936	(32.2729)	0.4453	(0.4451)	1209.4349	(1204.6509)	9.5578	(9.4703)

Table 3.33: *Comparative performance of the index heuristic and an optimal policy for a range of average costs problems with two customer classes.*

b_1	b_2	$\rho = 0.60$				$\rho = 0.85$			
1.00	0.10	2.8802	(2.8802)	6.5241	(6.5241)	9.5440	(9.5418)	61.4237	(61.3950)
		19.3929	(19.3929)	0.2977	(0.2963)	505.1724	(505.1617)	3.9002	(3.8991)
1.00	0.20	3.0861	(3.0859)	7.0706	(7.0706)	12.5758	(12.5756)	78.8265	(78.8256)
		21.2598	(21.2598)	0.3165	(0.3165)	647.5939	(647.3051)	5.1201	(5.1200)
1.00	0.50	3.6352	(3.6352)	8.3852	(8.3852)	18.9538	(18.9537)	114.5855	(114.5529)
		25.5310	(25.5310)	0.3682	(0.3682)	937.6228	(937.5127)	7.6957	(7.6956)
1.00	1.00	4.3379	(4.3365)	10.2027	(10.1928)	25.0574	(25.0438)	152.2819	(152.1247)
		31.7557	(31.6629)	0.4405	(0.4404)	1257.7162	(1251.5248)	10.1860	(10.1807)
1.00	2.00	5.3396	(5.3291)	12.7169	(12.7169)	31.8017	(31.7742)	199.3951	(198.8927)
		39.7882	(39.7414)	0.5461	(0.5304)	1660.0632	(1655.7462)	12.9627	(12.8847)
2.00	0.10	3.7713	(3.7712)	8.7142	(8.7120)	10.6295	(10.6279)	71.9051	(71.7997)
		26.6031	(26.6027)	0.3734	(0.3721)	622.5756	(622.5390)	4.2643	(4.2283)
2.00	0.20	3.9790	(3.9790)	9.3873	(9.3649)	14.0270	(14.0257)	95.1909	(95.1781)
		28.8583	(28.8583)	0.4010	(0.4001)	819.2224	(819.2021)	5.6788	(5.6604)
2.00	0.50	4.5853	(4.5834)	11.0135	(11.0135)	22.4480	(22.4478)	145.9129	(145.9097)
		34.5466	(34.5465)	0.4625	(0.4623)	1234.8649	(1234.8568)	9.0829	(9.0827)
2.00	1.00	5.4561	(5.4561)	13.1438	(13.1438)	32.1849	(32.1848)	202.4307	(202.3046)
		41.5439	(41.5438)	0.5457	(0.5457)	1699.1895	(1698.4475)	13.0174	(13.0173)
2.00	2.00	6.8041	(6.7923)	16.7250	(16.5944)	43.9525	(43.9089)	276.4430	(275.6991)
		53.8595	(53.1546)	0.6866	(0.6844)	2326.4592	(2312.4604)	17.8278	(17.7953)

Table 3.34: *Comparative performance of the index heuristic and an optimal policy for a range of average costs problems with two customer classes.*

3.5.3 Simulation study of average costs problems with five customer classes

We now look at some examples of the undiscounted service control problems encountered in this chapter where we have five customer classes. In the two class problems of Sections 3.5.1 and 3.5.2 it was possible to obtain a direct numerical comparison between costs incurred by our index heuristics and those incurred by an optimal policy. However this is not a reasonable computational goal for larger problems. The simulation study reported in Tables 3.35 and 3.36 concern a collection of service control problems involving five customer classes under the average cost criterion.

Table 3.35 contains the results of studies of ten problems with quadratic costs ($1-5$, $1^{'}-5^{'}$) and five problems with quartic costs (1-5). All problems in this table feature deterministic service times. Each of the problems with quadratic costs is characterised by four five-vectors $\mathbf{b}$, $\mathbf{c}$, $\boldsymbol{\lambda}$ and $\mathbf{S}$. Both $\mathbf{b}$ and $\mathbf{c}$ are vectors of cost coefficients such that the class k cost rate is given by

$$C_k(n) = b_k n + c_k n^2, \;\; 1 \leq k \leq 5, \tag{3.89}$$

while $\boldsymbol{\lambda}$ is a vector of arrival rates with λ_k the rate for class k. Finally, $\mathbf{S}$ is a vector of deterministic service times. For example, for quadratic problem 1 we take $\mathbf{b} = (5, 4, 3, 2, 1)$, $\mathbf{c} = (1, 2, 3, 4, 5)$, $\boldsymbol{\lambda} = (0.40, 0.30, 0.25.0.10, 0.05)$ and $\mathbf{S} = (0.6, 0.5, 0.4, 0.7, 0.8)$ with a resulting traffic intensity of 0.60. To obtain quadratic problems 2-5 we keep $\boldsymbol{\lambda}$ and $\mathbf{S}$ fixed, but reassign the cost coefficients by means of a series of permutations. For example for problem 2 we take $\mathbf{b} = (1, 5, 4, 3, 2)$, $\mathbf{c} = (5, 1, 2, 3, 4)$ and so on. We obtain quadratic problems $1^{'}$-$5^{'}$ respectively from 1-5 by rescaling λ to give a traffic intensity of 0.85, while keeping other aspects fixed. We obtain quartic problems 1-5 from the corresponding quadratic problems upon replacing (3.89) by

$$C_k(n) = b_k n^3 + c_k n^4, \;\; 1 \leq k \leq 5.$$

In the body of Table 3.35 we have included estimates of the average cost rates incurred for the above problems under five service control heuristics, as follows: INDEX denotes our index heuristic for average costs while LQ allocates service at each decision epoch to whichever customer class has the longest queue (and chooses among the candidate classes at random in the event of a tie). MYOPIC always chooses for processing whichever customer class is incurring the largest instantaneous cost rate and MYOPIC* modifies this criterion by dividing the instantaneous cost rate by the corresponding mean service time. At each decision epoch, RANDOM chooses one of the non-empty customer classes at random (with equal probabilities) and serves a single customer from the class chosen. The estimate of average cost is obtained in each case by Monte Carlo simulation. Typically, we allowed a "burn-in" period of around 10,000 time units in each case, followed by a period of around 15,000 time units during which costs were tracked. This was repeated around 50 times and the average costs (per unit time) were estimated. The corresponding standard errors are given in brackets in the table. The details of the mechanics of the simulations varied a little across the different cases in order to obtain standard errors which would enable meaningful comparisons between service policies to be made. For example, when we increased the traffic intensity to 0.85 we had to increase the number of runs. Note that we did not have access to sufficient computer resources for satisfactory standard errors to be achieved for problems with quartic costs and a traffic intensity of 0.85. This is why no such cases are reported in the table. The study in Table 3.36 mirrors that in Table 3.35 and differs only in that the service times are now Gamma distributed. Hence, for quadratic problem 1 the single five-vector $\mathbf{S}$ of deterministic times is replaced by two five-vectors $\mathbf{m} = (1, 2, 3, 4, 5)$ and $\boldsymbol{\mu} = (5/3, 6, 5, 40/7, 25/4)$. We now suppose that $S_k \sim \Gamma(m_k, \mu_k)$, $1 \leq k \leq 5$. All other details are as in the study in Table 3.35.

Quadratic Costs	INDEX	LQ	MYOPIC	MYOPIC*	RANDOM
1	6.7103	6.9759	6.8919	7.2142	7.0933
	(0.0358)	(0.0394)	(0.0449)	(0.0496)	(0.0507)
2	6.9778	7.4549	7.3400	7.6648	7.7823
	(0.0430)	(0.0568)	(0.0550)	(0.0645)	(0.0840)
3	7.1444	7.8734	7.8815	7.9003	8.8498
	(0.0489)	(0.0601)	(0.0475)	(0.0531)	(0.0778)
4	7.3377	7.9216	7.7673	7.9249	8.7709
	(0.0423)	(0.0585)	(0.0541)	(0.0632)	(0.1152)
5	7.2164	7.6448	7.6566	7.7806	8.2742
	(0.0493)	(0.0489)	(0.0451)	(0.0497)	(0.1077)
1'	23.2539	25.5787	24.0424	28.3180	28.9242
	(0.4346)	(0.4844)	(0.5170)	(0.5113)	(0.5900)
2'	25.2815	30.7615	27.9366	30.3640	57.1030
	(0.5172)	(0.8053)	(0.4614)	(0.4835)	(1.0815)
3'	24.7591	33.8409	29.4795	32.1201	83.3331
	(0.4060)	(0.6157)	(0.4755)	(0.4777)	(3.4087)
4'	25.6866	31.1344	30.1719	30.2028	72.1357
	(0.3649)	(0.6197)	(0.4898)	(0.4667)	(2.6194)
5'	26.3250	29.7588	29.3930	29.5962	55.3345
	(0.5261)	(0.4981)	(0.5977)	(0.4620)	(2.0550)
Quartic Costs					
1	15.5772	15.7914	16.0158	17.8664	22.3649
	(0.1703)	(0.1851)	(0.2050)	(0.2282)	(0.5133)
2	17.2057	18.6310	18.2118	20.2739	25.5776
	(0.1961)	(0.2237)	(0.2003)	(0.2743)	(0.6412)
3	18.2476	22.2612	21.6834	22.1398	42.3787
	(0.2390)	(0.2658)	(0.2661)	(0.3997)	(1.9690)
4	19.4305	22.8196	23.1101	22.2155	49.2510
	(0.2524)	(0.3014)	(0.3425)	(0.3057)	(6.2762)
5	18.5401	21.9044	22.1773	21.4857	40.9507
	(0.2185)	(0.3103)	(0.2912)	(0.3282)	(2.2664)

Table 3.35: *Comparative performance of the index heuristic and four other control rules for a range of average costs problems with five customer classes and deterministic service times.*

Quadratic Costs	INDEX	LQ	MYOPIC	MYOPIC*	RANDOM
1	8.9812	9.3200	9.3366	9.3885	9.5438
	(0.0941)	(0.0733)	(0.0894)	(0.0917)	(0.0878)
2	9.5892	10.2201	10.2700	10.0731	11.1100
	(0.1010)	(0.0860)	(0.1506)	(0.0935)	(0.1380)
3	9.9218	11.2622	10.9091	11.1442	13.9702
	(0.0904)	(0.0970)	(0.1127)	(0.1143)	(0.2522)
4	10.2312	10.9974	10.7825	11.0971	13.3023
	(0.1098)	(0.1136)	(0.0866)	(0.0997)	(0.4585)
5	10.0943	10.7580	10.6351	11.2773	12.4465
	(0.1153)	(0.0962)	(0.1296)	(0.1306)	(0.1832)
1'	39.4936	45.6291	42.0556	41.1953	58.1367
	(1.3472)	(1.2900)	(1.0080)	(0.9626)	(3.0910)
2'	44.1563	52.1205	49.7436	52.9404	86.0343
	(1.1356)	(1.1165)	(1.0747)	(1.4466)	(2.9641)
3'	42.5420	60.9430	53.6382	54.9029	187.7974
	(0.9720)	(1.6908)	(1.4915)	(1.2248)	(10.9604)
4'	47.2808	56.1806	52.0994	58.2293	157.9946
	(1.1669)	(1.1536)	(1.4938)	(1.3649)	(6.5433)
5'	45.9588	52.8616	49.0092	57.8623	113.7342
	(1.4101)	(1.5572)	(1.1121)	(1.4052)	(3.9717)
Quartic Costs					
1	34.4928	33.7941	33.3589	38.0270	60.5706
	(0.8522)	(0.7745)	(0.7173)	(0.8749)	(2.8492)
2	39.1317	41.1258	40.5730	44.3442	72.3138
	(0.7614)	(0.8847)	(0.7935)	(1.0462)	(3.2612)
3	42.9542	49.1543	48.4376	50.3789	150.1279
	(0.9132)	(0.9074)	(1.2642)	(1.4623)	(11.2225)
4	45.4567	53.0129	51.2151	52.2439	144.0640
	(1.2018)	(1.0876)	(1.0783)	(1.0614)	(7.8021)
5	43.9029	54.1418	48.5072	54.1950	113.3488
	(0.8862)	(1.4611)	(0.9447)	(1.1625)	(4.9810)

Table 3.36: *Comparative performance of the index heuristic and four other control rules for a range of average costs problems with five customer classes and gamma distributed service times.*

3.5.4 Comments

As one can see all the numerical evidence seems to suggest that our index heuristic policy performs very well. We can see this because the index policy is usually close to the optimal policy costs or indeed, in the simulation of the five customer classes example, better than the alternative policies.

From the discounted numerical data of Tables 3.1 - 3.32 we can see that obviously the total costs increase if we start with an increasingly congested initial state and also when the cost functions are of a higher order. The index policy seems to perform slightly less well when we look at the more congested initial states. However the sub-optimality of this policy always remains small in percentage terms. Also notice that as we alter the service distributions of the class types so that they are less similar, relatively speaking our index does not perform quite as well. However the sub-optimality is still reassuringly small. Another thing to notice is how well the index heuristic performs even when we increase the traffic intensity.

Looking at the numerics for the average cost performance in Tables 3.33 and 3.34 one can see that the costs increase if we use higher cost coefficients or consider cost functions of a larger order or when we have a larger traffic intensity. However in all cases the index policy continues to perform extremely well with small percentage cost sub-optimality throughout.

We proceed to consider the simulation results of tables 3.35 and 3.36 for the five customer class example. The cost rate for the index policy is smaller than for all other policies considered in every example bar one. In the example where the index does not return the lowest cost rate it comes a very close second and is certainly within sampling variation of this lowest cost. In all the other examples where the index policy does return the lowest cost rate it is significantly below its closest competitor in the majority of cases.

The numerical data strongly suggest that the index policy presented in this chapter performs very well for a variety of models. Hence the evidence is that it is an effective policy in cost terms and is easy to compute and implement.

Chapter 4

Concluding Remarks

4.1 Summary

We have considered the problems of routing and service control as outlined in the previous chapters. As we have seen, we have applied a similar approach to these different problems and found index heuristics for them both which perform well. The formulae for these indices was calculated in each case. The key was to decompose the original multi-dimensional problem into a collection of one-dimensional problems, which are much easier to deal with. This was achieved by considering a relaxation of the original problem with a constraint, then using a Lagrangian multiplier to incorporate the constraint. The index formula for the routing control problem is given in equation (2.44) and the index formula for the service control problem is given in (3.37). Once the indices have been calculated for all the current class states the policy merely requires that in the routing problem the system controller sends the arriving customer to the server with the smallest index and in the service problem that the class with the highest index is served first. Using these formulae we were able to consider possible queueing systems and produce some numerical evidence to assess the effectiveness of our proposed index policies. This evidence seems to indicate that the policies proposed do perform well in a range of different scenarios. Not only that but the index nature of our policies mean that implementation of the policies is fairly straightforward. These are the reasons that I am confident the policies proposed would return positive results in a suitable real world scenario such as the ones mentioned in the introductory Chapter 1, namely;

(i) Which of N possible routes should a telecommunications firm use to send a message when the total delivery time via each route and the arrival times of future messages are unknown?

(ii) In what order should a computer allocate processing amongst a number of competing classes of job awaiting service, when exact processing requirements and the times of future arrivals are unknown?

4.2 Possible Further Work

From the past chapters one can notice that the routing control problem assumes that the service times of the customers follow an exponential distribution. However, in the service control problem we allow the service times to follow a general distribution. So the first suggestion for possible future work would be to allow the routing control problem also to have a general service distribution. However, without the memoryless property of the exponential distribution this problem would prove considerably more difficult to analyse.

A further suggestion for possible future study would be to consider both of these two problems in a single queueing system. So that we have a truly multi-class system, with each class having its own arrival rate. We would first need to make the decision about which station to send each arrival to. Then at each server there could be a queue consisting of a number of different customer classes, each class with its own attributes. The second decision to make then would be which of the classes to serve at each station. The second part of the above problem would be very similar to the service control problem considered in Chapter 3 but note now that now the arrival streams will more complex. It would perhaps be possible to try and model the second part of this problem roughly just using the system setup from Chapter 3, leaving the prime issue in the analysis being the routing control part of the problem. Again similarities could be taken from the routing problem of Chapter 2. However now we would have K customer classes each possibly arriving at different rates and each class possibly possessing different cost rates even if they are served at the same station. A development of the DP policy improvement approach of Ansell et al (2001) may be the best hope for progress here for undiscounted versions of the problem.

Chapter 5

References

Ansell P.S., Dacre M.J., Glazebrook K.D. and Kirkbride C. (2001) Optimal load balancing and scheduling in distributed multi-class service systems. Technical Report, *Newcastle Upon Tyne.*

Ansell, P.S., Glazebrook, K.D., Niño Mora, J. and O'Keeffe, M. (2003a), Whittle's index policy for a multi-class queueing system with convex holding costs. *Mathematical Methods of Operations Research*, **57**, 21-39.

Ansell, P.S., Glazebrook, K.D., Mitrani, I. and Niño Mora, J. (1999). A semi definite programming approach to the optimal control of a single server queueing system with imposed second moment constriaints. *J. Oper. Res. Soc.*, **50**, 765-773.

Ansell P.S., Glazebrook, K.D., and Lumley, R.R. (2003b). Index heuristics for multi-class M/G/1 systems with non-preemptive service and convex holding costs, *Queueing Systems, Theory and Applications*, *Queueing Systems*, **45**, 84-111.

Ansell, P.S., Glazebrook K.D., and Kirkbride, C. (2003c). Generalised 'join the shortest queue' policies for the dynamic routing of jobs to multi-class queues. *J.*

Oper. Res. Soc., **54**, 379-389.

Becker, K.J., Gaver, D.P., Glazebrook, K.D., Jacobs, P.A. and Lawphongpanich, S. (2000) Allocation of tasks to specialised processors: a planning approach. *Eur. J. Oper. Res.*, **126**, 80-88.

Bellman, (1957). Dynamic programming. *Journal of the Franklin Institute*, **v265**, 157-158.

Bertsimas, D. and Niño-Mora, J. (1996) Conservation laws, extended polymatroids and multi-armed bandit problems: a polyhedral approach to indexable systems, *Math. Oper. Res.*, **21**, 257-306.

Braun, T.D., Siegel, H.J. and Maciejewski, A.A. (2001) Hetrogeneous computing: goals, methods and open problems. "PDPTA 2001: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, (ed. Arabnia HR), pp 1-12. CSREA: Athens".

Chang, C.S. (1992) A new ordering for stochastic majorization: theory and applications. *Adv. Appl. Prob.*, **24**, 604-634.

Cox, Smith (1961) Queues. *Meuthuen*, London.

Dacre, M.J., Glazebrook, K.D. and Niño-Mora, J. The achieveable region approach to the optimal control of stochastic system (with discussion). *J. R Statis. Soc. B*, **61**, 747-791.

Ephremides, A., Varaiya, P. and Walrand, J. (1980) A simple dynamic routing problem. *IEEE Trans. Aut. Control.*, **AC-25**, 690-693.

Federgruen, A. and Groenvelt, H. (1988) Characterisation and optimasation of achieveable performance in queueing systems. *Oper. Res.*, **36**, 336-346.

Foster I and Kesselman C (1998) The Grid: Blueprint for a new computing infrastructure. *Morgan Kaufman: San Francisco.*

Garbe and Glazebrook (1996) Reflections on a new approach to Gittins indexation, *J. Oper. Res. Soc.* **47**, 1301-1309.

Gelenbe, E. and Pekergin, F. (1993) Load balancing pragmatics. Technical report, *EHEI Université René Descartes.*

Gittins, J. C. (1989), *Multi-armed Bandit Allocation Indices*, Wiley, New York.

Gittins, J.C. and Jones, D.M. (1974) A dynamic allocation index for the sequential design of experiments. In *Progress in statistics (European Meeting of Statistics, Budapest, 1972)*, Colloq. Math. soc. Janos Bolyai, **Vol 9**, 241-266, North-Holland, Amsterdam.

Gittins, J.C. (1979) Bandit processes and dynamic allocation indices (with discussion). *J. Roy. Statist. So.*, **41**, 148-177.

Glazebrook, K.D. and Wilkinson D.J. (2000) Index based policies for discounted multi-armed bandits on parallel machines. *The Annals of Applied Probability*, **10**, 877-896.

Glenebe, E. and Mitrani, I. (1980) *Analysis and Synthesis of Computer Systems.* London: Academic Press.

Harrison, J.M. (1975) Dynamic scheduling of a multiclass queue: discount optimality. *Oper. Res.*, **23**, 270-282.

Hordijk, A. and Koole G. (1990) On the optimality of the generalized shortest queue policy. *Prob. Eng. Inf. Sci.*, **4**, 477-487.

Johri (1989) Optimality of the shortest line discpline with state dependent service times. *Eur. J. Op. Res.*, **41**, 157-161.

Klimov, G.P. (1974) Time sharing service systems I. *Theory Prob. Appl.*, **19**, 532-551.

Koole, G. (1996) On the pathwise optimal Bernoulli routing policy for homogeneous parallel servers. *Math. Oper. Res.*, **21**, 469-476.

Krishnan, K.R. (1987) Joining the right queue: a Markov decision rule. In *Proceedings of the 28th IEEE Conference on Decision Control*, pages 1863-1868.

Lui, Z. and Townsley, D. (1994) Optimality of the round robin policy. *J. Appl. Prob.*, **31**, 466-478.

Meilijson, I. and Weiss, G. (1977) Multiple feedback at a single service station. *Stoch. Proc. Appl.*, **5**, 195-205.

van Meighem, J. A. (1995). Dynamic scheduling with convex delay costs: the generalized $c\mu$-rule. *Ann. Appl. Prob.*, **5**, 809-833.

Niño-Mora, J. (2001a) Restless bandits, partial conservation laws, and indexability. *Adv. Appl. Prob.*, **33**, 76-98.

Niño-Mora, J. (2001b) Countable partial conservation laws, Whittle's restless bandit index and a dynamic $c\mu$ rule for scheduling a multiclass $M/M/1$ queue with convex holding costs. *Adv. Appl. Prob.*, **to appear**.

Putterman, M. L. (1994), *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley, New York.

Shanthikumar, J.G. and Tao, D.D. (1992) Multi-class queueing systems:

polymatroidal structure and optimal scheduling control. *Oper. Res.*, **40**, 293-299.

Tijms, H.C (1994) Stochastic Models: An Algorithmic Approach. *John Wiley & Sons.*

Tsitsiklis, J. and Papadimitriou, C.H. (1986) The performance of a preceedence based queueing discipline. *J. Assoc. Comput. Mach*, **v33**, 593-602.

Weber, R.R. (1978) On the optimal assignment of customers to parallel queues. *J. Appl. Prob.*, **15**, 406-413.

Weber, R.R. (1992) On the Gittins index for multi-armed bandits. *Ann. Appl. Prob.*, **2**, 1024-1033.

Weber, R.R. and Weiss, G. (1990) On an index policy for restless bandits. *J. Appl. Probab.*, **27**, 637-648.

Weiss, G. (1988) Branching bandit processes. *Prob. Eng. Inf. Sci.*, **2**, 269-278.

Whittle, P. (1980) Multi-armed bandits and the Gittins index. *J. Roy. Statist. Soc.*, **B42**, 143-149.

Whittle, P. (1996), Optimal control: Basics and Beyond, Wiley, Chichester.

Whittle, P. (1988) Restless bandits: activity allocation in a changing world. In: ed. J. Gandi ed., *A Celebration of Applied Probability, J. Appl. Prob., special vol.* **25A**, 287-298.

Whittle (1981) Arm-acquiring bandits. *Annals of Probability*, **9**, 284-292.

Winston, W. (1997) Optimality of the shortest line discipline. *J. Appl. Prob.*, **14**, 181-189.

Chapter 6

APPENDICES

Appendix A

This appendix contains the Fortran 95 code for the programme we used to calculate the discounted routing control costs as in Section 2.5.1. Here we consider the optimal and index policies for a 2 class system.

```
program ser_ad_both
implicit none

integer :: h,buffer
double precision :: pow1,pow2

integer :: FAIL, n, nmax, s, i, n1, n2, p
integer, allocatable, dimension (:,:) :: MVFAIL
integer, allocatable, dimension (:) :: VFAIL,CHFAIL,CHFAILN,MVFAIL1
double precision :: l,alf,a,b,d,e,TOL,inds,opts
double precision, allocatable, dimension(:) :: m
double precision, allocatable, dimension(:,:) :: C, CNeg, Copt, Cind, CindNeg,
What, Chat, X
double precision, allocatable, dimension(:,:) :: Compare, CompareNeg,
CompareInds


!input the restricted state space size & allocate Expectation maxtrix size &
the number of servers
nmax = 159
s = 2
print*,"a"
allocate( C(s,0:nmax) )
allocate( CNeg(s,-nmax:nmax) )
allocate( m(s) )
allocate( Copt(0:nmax,0:nmax) )
allocate( Cind(0:nmax,0:nmax) )
allocate( CindNeg(0:nmax,0:nmax) )
allocate( Compare(0:nmax,0:nmax) )
allocate( CompareNeg(0:nmax,-0:nmax) )
allocate( CompareInds(0:nmax,0:nmax) )
allocate( CHFAIL(s) )
allocate( CHFAILN(s) )

allocate( MVFAIL(s,2:nmax-1) )
allocate( MVFAIL1(s) )
allocate( VFAIL(s) )
allocate( What(s,0:nmax) )
allocate( Chat(s,0:nmax) )
allocate( X(s,0:nmax) )

MVFAIL = 0

!get the inital starting values of the arrays before updating them with our
value iteration.

do h = 1,8

call starting_vals(inds,opts,h)

!get the inital values of the queue, i.e. arrival & service rates & cost
function values.
call queue_values(l,m,s,alf,a,b,d,e,TOL,buffer,pow1,pow2,h)

!check that all the constraints hold
call check(h,l,m,s,FAIL)

C = 0.0

do n = buffer,nmax
    C(1,n) = a*(real(n-buffer)**pow1) + b*(real(n-buffer)**pow2)
    C(2,n) = d*(real(n-buffer)**pow1) + e*(real(n-buffer)**pow2)
end do

do n = -nmax,nmax
    if ( n-buffer >= 0) then
        CNeg(1,n) = a*(real(n-buffer)**pow1) + b*(real(n-buffer)**pow2)
        CNeg(2,n) = d*(real(n-buffer)**pow1) + e*(real(n-buffer)**pow2)
    else
        CNeg(1,n) = 0.0
        CNeg(2,n) = 0.0
    end if
end do

call costs_index(h,C,nmax,s,Cind,What,X,Chat,CHFAIL)
```

```
call costs_opt(h,C,nmax,s,Copt)
call costs_index_neg(h,CNeg,nmax,s,CindNeg,CHFAILN)

do n1 = 0,nmax
  do n2 = 0,nmax
    Compare(n1,n2) = (Cind(n1,n2) - Copt(n1,n2))*(100.0)/Copt(n1,n2)
  end do
end do

do n1 = 0,nmax
  do n2 = 0,nmax
    CompareNeg(n1,n2) = (CindNeg(n1,n2) - Copt(n1,n2))*(100.0)/Copt(n1,n2)
  end do
end do

do n1 = 0,nmax
  do n2 = 0,nmax
    CompareInds(n1,n2) = (CindNeg(n1,n2) - Cind(n1,n2))*(100.0)/Cind(n1,n2)
  end do
end do

print*,"the discounted cost to infinity for the optimal policy when starting
from states (0,0) to (5,5) &
         & on this restless bandit admission control system is: "
do i = 0,5
  write(unit=6,fmt="(6f12.4)") Copt(i,0:5)
end do

print*,"and the discounted cost to infinity when starting in state (0,0) to
(5,5) for our index policy is: "
do i = 0,5
  write(unit=6,fmt="(6f12.4)") Cind(i,0:5)
end do

print*,"so comparing these two policies we find, the degree of suboptimallity
of the index compared to the optimal is : "
do i = 0,9
  write(unit=6,fmt="(10f12.8)") Compare(i,0:9)

end do
print*," "

print*,"and the discounted cost to infinity when starting in state (0,0) to
(5,5) for our index policy (with -ve customers) is: "
do i = 0,5
  write(unit=6,fmt="(6f12.4)") CindNeg(i,0:5)
end do

print*," "

print*,"so comparing the index policy which assumes -ve customers possible,
with the optimal policy"
print*," (which does NOT assume customers can take a -ve number), gives the
degree of suboptimallity"
print*," of the index (with -ve) compared to the optimal (without -ve) (states
(0,0) to (5,5)) : "

do i = 0,5
  write(unit=6,fmt="(6f12.4)") CompareNeg(i,0:5)
end do
print*," "

print*,"so comparing the two index policies we find, the degree of
suboptimallity of the regular index"
print*,"compared to the index where -ve customers are allowed is (states (0,0)
to (5,5)) : "

do i = 0,5
  write(unit=6,fmt="(6f12.4)") CompareInds(i,0:5)
end do
print*," "

open(unit=7, file="spoldat2b.dat")

if(h == 1) write(unit=7,fmt="(a)") "Quadratic costs"
```

```
if(h == 3) write(unit=7,fmt="(a)") "Cubic costs"
if(h == 5) write(unit=7,fmt="(a)") "Quartic costs"
if(h == 7) write(unit=7,fmt="(a)") "Quadratic costs with buffer = 2"

write(unit=7,fmt="(a)") "        a     :     b     :     d     :     e     :
 l     :   m(1)     :     m(2)    :    alpha "
write(unit=7,fmt="(8f12.6)") a,b,d,e,l,m(1),m(2),alf
print*,"nmax = ",nmax
write(unit=7,fmt="('nmax : ',i6)") nmax

write(unit=7,fmt="(a)") "If FAIL /= 0 some of the constraints do not hold:"
write(unit=7,fmt="(a,i3)") "FAIL = ",FAIL

write(unit=7,fmt="('Index policy starting   : ',f12.6)") inds
write(unit=7,fmt="('Optimal policy starting   : ',f12.6)") opts

write(unit=7,fmt="(a)") "  "

write(unit=7,fmt="(a)") "discounted cost to infinity when starting in state
(0,0) - (5,5) for the optimal policy is "
do i = 0,4
  write(unit=7,fmt="(f12.4,a,f12.4,a,f12.4,a,f12.4,a,f12.4,a)") &
  & Copt(i,0)," & ",Copt(i,1)," & ",Copt(i,2)," & ",Copt(i,3)," &
",Copt(i,4)," \\ "
end do

write(unit=7,fmt="(a)") "  "

write(unit=7,fmt="(a)") "discounted cost to infinity when starting in state
(0,0) - (4,4) for our index policy is "
do i = 0,4
  write(unit=7,fmt="(f12.4,a,f12.4,a,f12.4,a,f12.4,a,f12.4,a)") &
  & Cind(i,0)," & ",Cind(i,1)," & ",Cind(i,2)," & ",Cind(i,3)," &
",Cind(i,4)," \\ ",
end do

write(unit=7,fmt="(a)") " "

write(unit=7,fmt="(a)") "the degree of suboptimallity of the index compared to
the optimal policies (states (0,0) - (5,5)) are : "
do i = 0,4
  write(unit=7,fmt="(f12.4,a,f12.4,a,f12.4,a,f12.4,a,f12.4,a)") &
  & Compare(i,0)," & ",Compare(i,1)," & ",Compare(i,2)," & ",Compare(i,3)," &
",Compare(i,4)," \\ "
end do

write(unit=7,fmt="(a)") "  "

write(unit=7,fmt="(a)") "discounted cost to infinity when starting in state
(0,0) - (5,5)"
write(unit=7,fmt="(a)") "for our index policy (with -ve customers) is "
do i = 0,4
  write(unit=7,fmt="(f12.4,a,f12.4,a,f12.4,a,f12.4,a,f12.4,a)") &
  & CindNeg(i,0)," & ",CindNeg(i,1)," & ",CindNeg(i,2)," & ",CindNeg(i,3)," &
",CindNeg(i,4)," \\ "
  write(unit=7,fmt="(a)") "  "
end do

write(unit=7,fmt="(a)") " "

write(unit=7,fmt="(a)") "the degree of suboptimallity of the index (which
takes -ve # of customers)"
write(unit=7,fmt="(a)") "compared to the optimal (which takes only +ve # of
customers) policy"
write(unit=7,fmt="(a)") "(state (0,0) to (5,5) are : "
do i = 0,4
  write(unit=7,fmt="(f12.4,a,f12.4,a,f12.4,a,f12.4,a,f12.4,a)") &
  & CompareNeg(i,0)," & ",CompareNeg(i,1)," & ",CompareNeg(i,2)," &
",CompareNeg(i,3)," & ",CompareNeg(i,4)," \\ "
end do

write(unit=7,fmt="(a)") " "

write(unit=7,fmt="(a)") "the degree of suboptimallity of the index (which
takes -ve # of customers)"
```

```
write(unit=7,fmt="(a)") "compared to the index (which takes only +ve # of
customers) policy"
write(unit=7,fmt="(a)") "(state (0,0) to (5,5) are : "
do i = 0,5
  write(unit=7,fmt="(f12.4,a,f12.4,a,f12.4,a,f12.4,a,f12.4,a)") &
  & CompareInds(i,0)," & ",CompareInds(i,1)," & ",CompareInds(i,2)," &
",CompareInds(i,3)," & ",CompareInds(i,4)," \\ ",
end do

write(unit=7,fmt="(a)") "  "

!write(unit=7,fmt="(a)") "if CHFAIL = 0 Chat is increasing convexly"
!write(unit=7,fmt="(a)") "if CHFAIL = 1 Chat is not increasing"
!write(unit=7,fmt="(a)") "if CHFAIL = 2 Chat is increasing but not convexly"
!write(unit=7,fmt="(a)") "  "
!write(unit=7,fmt="(a,2i4)") "CHFAIL = ",CHFAIL(:)

write(unit=7,fmt="(a)") "  "


!write(unit=7,fmt="(a)") "if CHFAILN = 0 Chat_neg is increasing convexly"
!write(unit=7,fmt="(a)") "if CHFAILN = 1 Chat_neg is not increasing"
!write(unit=7,fmt="(a)") "if CHFAILN = 2 Chat_neg is increasing but not
convexly"
!write(unit=7,fmt="(a)") "  "
!write(unit=7,fmt="(a,2i4)") "CHFAILN = ",CHFAILN(:)

do p = nmax-1,2,-1
  call checkV(h,C,What,Chat,X,nmax,s,p,VFAIL)
  MVFAIL(:,p) = VFAIL(:)
end do

!print*," "
!print*,"MVFAIL is :"
!do i = 2,nmax-1
!  write(unit=6,fmt="(a,i4,a,2i4)") "MVFAIL(:,",i,") = ",MVFAIL(:,i)
!end do
!print*," "

MVFAIL = 0
do p = 2,nmax-1
  do i = 1,s
    if (MVFAIL(i,p) /= 0) MVFAIL1(i) = 1
  end do
end do

!write(unit=7,fmt="(a)") "  "
!write(unit=7,fmt="(a,2i4)") "MVFAIL1 = ",MVFAIL1(:)
!write(unit=7,fmt="(a)") "  "
!write(unit=7,fmt="(a)") "if MVFAIL1 /= 0 we have problems with V, to
investigate further look at MVFAIL(0:s,2:nmax-1)"
!write(unit=7,fmt="(a)") "if MVFAIL = 1 l*(V(i,n+1) - V(i,n)) > What(i,n)"
!write(unit=7,fmt="(a)") "if MVFAIL = 2 l*(V(i,n) - V(i,n-1)) > What(i,n)"
!write(unit=7,fmt="(a)") "if MVFAIL = 3 V(i,n+1) - V(i,n) < V(i,n) - V(i,n-1)"
!write(unit=7,fmt="(a)") "if MVFAIL = 4 V(i,k+1) - V(i,k) < V(i,k) - V(i,k-1)
where 0<k<n"

!write(unit=7,fmt="(a)") "MVFAIL is:"
!do i = 2,nmax-1
!  write(unit=7,fmt="(a,i4,a,2i4)") "MVFAIL(:,",i,") = ",MVFAIL(:,i)
!  write(unit=7,fmt="(a)") " "
!end do

!write(unit=7,fmt="(a)") "  "


end do
close(unit=7)

end program

!-----------------------------------------------------------------------------
-----------

subroutine check(h,l,m,s,FAIL)
```

```
implicit none

integer :: FAIL,s,h
double precision :: l,maxm
double precision, dimension(s) :: m

FAIL = 0

print*," m1 = ",m(1)
print*," m2 = ",m(2)
print*," l = ",l

if (l >= m(1)+m(2)) FAIL = 1

if (m(1) > m(2)) then
  maxm = m(1)
else
  maxm = m(2)
end if

if (maxm >= l) FAIL = 1

return
end subroutine

!-----------------------------------------------------------------------------
------------
!this subroutine calculates the index value for our policy, so we know which
queue to send
!the arriving customer to then calculates the costs to infinity using this
policy.

subroutine costs_index(h,C,nmax,s,Cind,W,TE,Chat,CHFAIL)
implicit none

integer :: h,buffer
double precision :: pow1,pow2

integer :: s,nmax,n,i,ENDFX1,ENDFX2,FAIL,n1,n2,count,ExpFAIL,printto,num
integer, dimension (s) :: CHFAIL
double precision :: l,alf,a,b,d,e,smallest,largest,TOL,sroot,inds,opts
double precision, dimension(s) :: m,temp
double precision, dimension(s,0:nmax) :: C,W,Chat,TE
double precision, dimension(0:nmax,0:nmax) :: Cind,Cindo,bound

call queue_values(l,m,s,alf,a,b,d,e,TOL,buffer,pow1,pow2,h)
call starting_vals(inds,opts,h)
FAIL = 0
ExpFAIL = 0
call check(h,l,m,s,FAIL)

ENDFX1 = 1
ENDFX2 = 1

TE(:,0) = l/(alf+l)

do i = 1,s
  do n = 1,nmax
!    if(i==1) print*,TE(i,n-1)
    TE(i,n) = l/(alf + l + m(i) - (m(i)*TE(i,n-1)))
  end do
end do

!a check that this is indeed working*************************************
do i = 1,s
  call quad_roots(h,i,sroot)
!  print*,sroot,"    ",i
  if (TE(i,nmax) > sroot-TOL .and. TE(i,nmax) < sroot+TOL) then
    print*,"Expectation OKAY"
!    ExpFail = 0
  else
    print*,"Expectation error"
!    ExpFail = 1
  end if
end do
```

```
do i = 1,s
  do n = 0,nmax-1
    if (TE(i,n) < TE(i,n+1)) ExpFAIL = 1
  end do
end do

if (ExpFAIL == 1) print*,"Expectation error: non-decreasing with n"

Chat(:,0) = 0.0

do n = 1,nmax
  do i = 1,s

    Chat(i,n) = (alf*C(i,n) + m(i)*Chat(i,n-1))/(alf + l + m(i) -
(m(i)*TE(i,n-1)))

  end do
end do

W = 0.0

do n = 0,nmax-1
  do i = 1,s

    W(i,n) = alf*(TE(i,n+1)*(C(i,n+1) -
(Chat(i,n)/(1.0-TE(i,n)))))/(((1.0-TE(i,n+1))/(1.0-TE(i,n))) - TE(i,n+1))

  end do
end do

Cind = inds
Cindo = inds

count = 0

10 Cindo = Cind

count = count + 1

do n1 = 0,nmax-1
  do n2 = 0,nmax-1

    if (n1 > 0) then
      temp(1) = Cindo(n1-1,n2)
    else
      temp(1) = Cindo(n1,n2)
    end if

    if (n2 > 0) then
      temp(2) = Cindo(n1,n2-1)
    else
      temp(2) = Cindo(n1,n2)
    end if


    if (W(1,n1) <= W(2,n2)) then

      Cind(n1,n2) =  (C(1,n1) + C(2,n2))/(alf+l+m(1)+m(2)) +
(l*Cindo(n1+1,n2))/(alf+l+m(1)+m(2)) &
                    &  + (m(1)*temp(1))/(alf+l+m(1)+m(2)) +
(m(2)*temp(2))/(alf+l+m(1)+m(2))

    else

      Cind(n1,n2) = (C(1,n1) + C(2,n2))/(alf+l+m(1)+m(2)) +
(l*Cindo(n1,n2+1))/(alf+l+m(1)+m(2)) &
                    &  + (m(1)*temp(1))/(alf+l+m(1)+m(2)) +
(m(2)*temp(2))/(alf+l+m(1)+m(2))

    end if


  end do
end do
```

```
Cind(:,nmax) = Cind(:,nmax-1)
Cind(nmax,:) = Cind(nmax-1,:)
Cind(nmax,nmax) = Cind(nmax-1,nmax-1)

do n1 = 0,nmax - ENDFX1
  do n2 = 0,nmax - ENDFX2

    bound(n1,n2) =  -Cindo(n1,n2) + Cind(n1,n2)

  end do
end do

smallest = 100000000.0
largest = -100000000.0

do n1 = 0,nmax - ENDFX1
  do n2 = 0,nmax - ENDFX2

    if (smallest > bound(n1,n2)) then
      smallest = bound(n1,n2)
!       svec = (/n1,n2/)
    end if

    if (largest < bound(n1,n2)) then
      largest = bound(n1,n2)
!       lvec = (/n1,n2/)
    end if

  end do
end do

!open(unit=7,file="temp.dat")
!write(unit=7,fmt="(2f16.7)") smallest,largest

if ((largest - smallest) <=  TOL) then !*sqrt((smallest)*(smallest))) then !
then   !TOL*smallest
  goto 100
else
  goto 10
end if

!close(unit=7)

100 if (FAIL == 1) print*,"Error: Some constraints do not hold"

!indC = (largest + smallest)/2.0

!if CHFAIL = 0 Chat is increasing convexly
!if CHFAIL = 1 Chat is not increasing
!if CHFAIL = 2 Chat is increasing but not convexly

CHFAIL = 0
do i = 1,s
  do num = 1,nmax-1
    if (Chat(i,num+1)/(1.0-TE(i,num+1)) - Chat(i,num)/(1.0-TE(i,num)) < &
& Chat(i,num)/(1.0-TE(i,num)) - Chat(i,num-1)/(1.0-TE(i,num-1))) CHFAIL(i) = 2
  end do
end do

do i = 1,s
  do num = 1,nmax-1
    if (Chat(i,num+1) - Chat(i,num) < Chat(i,num) - Chat(i,num-1)) CHFAIL(i) =
1
  end do
end do

!!num = 0
!open(unit=7, file="Chat_pos_data.dat")
!write(unit=7,fmt="(a)") "non-negative customers"
!write(unit=7,fmt="(a)") "     n        Chat(n)-1    Chat(n)-2     Ch1/1-X1
  Ch2/1-X2"
!printto = nmax
!!if (nmax > 32) printto = 32
!do num=0,printto
```

```
!  write(unit=7,fmt="(i6,4f14.4)")
num,Chat(:,num),Chat(1,num)/(1.0-TE(1,num)),Chat(2,num)/(1.0-TE(2,num))
!!  num = num + 1
!end do
!close(unit=7)

!!num = 0
!open(unit=7, file="serve_exp_W_data.dat")
!write(unit=7,fmt="(a)") "non-negative customers"
!write(unit=7,fmt="(a)") "       n        E(T) - 1    E(T) - 2    W(n) - 1
W(n) - 2"
!printto = nmax
!if (nmax > 32) printto = 32
!do num=0,printto
!  write(unit=7,fmt="(i6,4f12.6)") num,TE(:,num),W(:,num)
!!  num = num + 1
!end do
!close(unit=7)

!print*," "
!print*,"largest index : ",largest
!print*,"smallest index : ",smallest
print*,"index count = ",count

return
end subroutine
!----------------------------------------------------------------------------
------------
!this subroutine calculates the optimal (smallest possible) cost to infinity -

!but we have no actual policy to follow to get such optimal costs

subroutine costs_opt(h,C,nmax,s,Copt)
implicit none

integer :: h,buffer
double precision :: pow1,pow2

integer :: s,nmax,ENDFX1,ENDFX2,FAIL,n1,n2,count
double precision :: l,alf,a,b,d,e,smallest,largest,TOL,inds,opts
double precision, dimension(s) :: m,val
double precision, dimension(s,0:nmax) :: C
double precision, dimension(0:nmax,0:nmax) :: Copt,Copto,bound
double precision, dimension(4) :: temp

call queue_values(l,m,s,alf,a,b,d,e,TOL,buffer,pow1,pow2,h)
call starting_vals(inds,opts,h)
FAIL = 0
call check(h,l,m,s,FAIL)

ENDFX1 = 1
ENDFX2 = 1

Copt = opts
Copto = opts

count = 0

20 Copto = Copt

count = count + 1

do n1 = 0,nmax
  do n2 = 0,nmax

    if (n1 > 0) then
      temp(1) = Copto(n1-1,n2)
    else
      temp(1) = Copto(n1,n2)
    end if

    if (n2 > 0) then
      temp(2) = Copto(n1,n2-1)
    else
      temp(2) = Copto(n1,n2)
```

```
    end if

    if (n1 < Nmax) then
      temp(3) = Copto(n1+1,n2)
    else
      temp(3) = Copto(n1,n2)
    end if

    if (n2 < Nmax) then
      temp(4) = Copto(n1,n2+1)
    else
      temp(4) = Copto(n1,n2)
    end if


    val(1) = (C(1,n1) + C(2,n2))/(alf+l+m(1)+m(2)) +
(l*temp(3))/(alf+l+m(1)+m(2)) &
             &  + (m(1)*temp(1))/(alf+l+m(1)+m(2)) +
(m(2)*temp(2))/(alf+l+m(1)+m(2))

    val(2) = (C(1,n1) + C(2,n2))/(alf+l+m(1)+m(2)) +
(l*temp(4))/(alf+l+m(1)+m(2)) &
             &  + (m(1)*temp(1))/(alf+l+m(1)+m(2)) +
(m(2)*temp(2))/(alf+l+m(1)+m(2))

    if(val(1) <= val(2)) then
      Copt(n1,n2) = val(1)
    else
      Copt(n1,n2) = val(2)
    end if

!     if (val(1) /= val(2))
print*,"******************HURRAH**********************"

  end do
end do

!Copt(:,nmax) = Copt(:,nmax-1)
!Copt(nmax,:) = Copt(nmax-1,:)
!Copt(nmax,nmax) = Copt(nmax-1,nmax-1)

do n1 = 0,nmax - ENDFX1
  do n2 = 0,nmax - ENDFX2

    bound(n1,n2) = -Copto(n1,n2) + Copt(n1,n2)

  end do
end do

smallest = 100000000.0
largest = -100000000.0

do n1 = 0,nmax - ENDFX1
  do n2 = 0,nmax - ENDFX2

    if (smallest > bound(n1,n2)) then
      smallest = bound(n1,n2)
!       svec = (/n1,n2/)
    end if

    if (largest < bound(n1,n2)) then
      largest = bound(n1,n2)
!       lvec = (/n1,n2/)
    end if

  end do
end do

if ((largest - smallest) <=  TOL) then !*sqrt((smallest)*(smallest))) then !
then   !TOL*smallest
  goto 200
else
  goto 20
end if
```

```
200 if (FAIL == 1) print*,"Error: Some constraints do not hold"

!optC = (largest + smallest)/2.0

!print*," "
!print*,"largest opt : ",largest
!print*,"smallest opt : ",smallest
print*,"optimal count = ",count

return
end subroutine

!---------------------------------------------------------------------------
------------
!this subroutine calculates the index assuming that we can have negative
numbers of customers,
!but when working out the cost to infinity, we do NOT assume we can have
negative customers!

subroutine costs_index_neg(h,C,nmax,s,Cind3,CHFAIL)
implicit none

integer :: h,buffer
double precision :: pow1,pow2

integer :: s,nmax,n,i,ENDFX1,ENDFX2,FAIL,n1,n2,count,printto,num
integer, dimension(s) :: CHFAIL
double precision :: l,alf,a,b,d,e,smallest,largest,TOL,inds,opts
double precision, dimension(s) :: m,TE
double precision, dimension(s,-nmax:nmax) :: C,W,Chat
double precision, dimension(0:nmax,0:nmax) :: Cind3,Cindo,bound
double precision, dimension(4) :: temp

call queue_values(l,m,s,alf,a,b,d,e,TOL,buffer,pow1,pow2,h)
call starting_vals(inds,opts,h)

FAIL = 0

call check(h,l,m,s,FAIL)
if (FAIL==1) print*,"ERROR: FAIL = 1 - Check constraints"

ENDFX1 = 1
ENDFX2 = 1

do i = 1,s
    TE(i) = (l + m(i) + alf - sqrt(((l + m(i) + alf)**2) -
(4*m(i)*l)))/(2*m(i))
end do

Chat(:,-nmax) = 0.0

do n = 1-nmax,nmax
  do i = 1,s

    Chat(i,n) = (alf*C(i,n) + m(i)*Chat(i,n-1))/(alf + l + m(i) -
(m(i)*TE(i)))

  end do
end do

W = 0.0

do n = 1-nmax,nmax-1
  do i = 1,s

    W(i,n) = alf*(TE(i)*(C(i,n+1) - (Chat(i,n)/(1.0-TE(i)))))/(1.0-TE(i))

  end do
end do

Cind3 = inds
Cindo = inds

count = 0
```

```
50 Cindo = Cind3

count = count + 1

do n1 = 0,nmax
  do n2 = 0,nmax

    if (n1 > 0) then
      temp(1) = Cindo(n1-1,n2)
    else
      temp(1) = Cindo(n1,n2)
    end if

    if (n2 > 0) then
      temp(2) = Cindo(n1,n2-1)
    else
      temp(2) = Cindo(n1,n2)
    end if

    if (n1 < Nmax) then
      temp(3) = Cindo(n1+1,n2)
    else
      temp(3) = Cindo(n1,n2)
    end if

    if (n2 < Nmax) then
      temp(4) = Cindo(n1,n2+1)
    else
      temp(4) = Cindo(n1,n2)
    end if


    if (W(1,n1) <= W(2,n2)) then

      Cind3(n1,n2) =  (C(1,n1) + C(2,n2))/(alf+l+m(1)+m(2)) +
(l*temp(3))/(alf+l+m(1)+m(2)) &
                  &  + (m(1)*temp(1))/(alf+l+m(1)+m(2)) +
(m(2)*temp(2))/(alf+l+m(1)+m(2))

    else

      Cind3(n1,n2) = (C(1,n1) + C(2,n2))/(alf+l+m(1)+m(2)) +
(l*Temp(4))/(alf+l+m(1)+m(2)) &
                  &  + (m(1)*temp(1))/(alf+l+m(1)+m(2)) +
(m(2)*temp(2))/(alf+l+m(1)+m(2))

    end if

  end do
end do

!Cind3(:,nmax) = Cind3(:,nmax-1)
!Cind3(nmax,:) = Cind3(nmax-1,:)
!Cind3(nmax,nmax) = Cind3(nmax-1,nmax-1)

do n1 = 0,nmax - ENDFX1
  do n2 = 0,nmax - ENDFX2

    bound(n1,n2) =  -Cindo(n1,n2) + Cind3(n1,n2)

  end do
end do

smallest = 100000000.0
largest = -100000000.0

do n1 = 0,nmax - ENDFX1
  do n2 = 0,nmax - ENDFX2

    if (smallest > bound(n1,n2)) then
      smallest = bound(n1,n2)
!      svec = (/n1,n2/)
    end if

    if (largest < bound(n1,n2)) then
```

```
      largest = bound(n1,n2)
!      lvec = (/n1,n2/)
    end if

  end do
end do

if ((largest - smallest) <=  TOL) then !*sqrt((smallest)*(smallest))) then !
then   !TOL*smallest
  goto 500
else
  goto 50
end if

500 if (FAIL == 1) print*,"Error: Some constraints do not hold"

!indC = (largest + smallest)/2.0

!if CHFAIL = 0 Chat is increasing convexly
!if CHFAIL = 1 Chat is not increasing
!if CHFAIL = 2 Chat is increasing but not convexly

CHFAIL = 0
do i = 1,s
  do num = 1,nmax-1
    if (Chat(i,num+1)/(1.0-TE(i)) - Chat(i,num)/(1.0-TE(i)) < &
& Chat(i,num)/(1.0-TE(i)) - Chat(i,num-1)/(1.0-TE(i))) CHFAIL(i) = 2
  end do
end do

!!num = -8.0
!open(unit=7, file="Chat_neg_data.dat")
!write(unit=7,fmt="(a)") "negative customers"
!write(unit=7,fmt="(a)") "       n        Chat(n)-1  Chat(n)-2    Ch1/1-X1
Ch2/1-X2"
!printto = nmax
!!if (nmax > 32) printto = 32
!do num=-8,printto
!  write(unit=7,fmt="(i6,4f14.4)")
num,Chat(:,num),Chat(1,num)/(1.0-TE(1)),Chat(2,num)/(1.0-TE(2))
!!  num = num + 1
!end do
!close(unit=7)

!!num = -2.0
!open(unit=7, file="ser_neg_exp_W_data.dat")
!write(unit=7,fmt="(a)") "negative customers (index3)"
!write(unit=7,fmt="(a)") "       n        E(T) - 1    E(T) - 2    W(n) - 1
W(n) - 2"
!printto = nmax
!if (nmax > 32) printto = 32
!do num=-2,printto
!  write(unit=7,fmt="(i6,4f12.6)") num,TE(:),W(:,num)
!!  num = num + 1.0
!end do
!close(unit=7)

!write(unit=7,fmt="(a)") " "

!write(unit=7,fmt="(a)") "discounted cost to infinity when starting in state
(0,0) - (5,5)"
!write(unit=7,fmt="(a)") "(with -ve customers) for our index3 policy is "
!do i = 0,5
!  write(unit=7,fmt="(6f12.4)") Cind3(i,0:5)
!  write(unit=7,fmt="(a)") "  "
!end do

!write(unit=7,fmt="(a)") " "

!close(unit=7, file="serve_exp_data.dat")

!print*," "
!print*,"largest index : ",largest
!print*,"smallest index : ",smallest
print*,"index count neg = ",count
```

```
return
end subroutine

!-----------------------------------------------------------------------------
------------

subroutine quad_roots(h,i,sroot)
implicit none

integer :: h,buffer
double precision :: pow1,pow2

integer :: s,i
double precision :: a,b,d,e,l,alf,a1,a2,a3,TOL,root1,root2,sroot
double precision, dimension (2) :: m

call queue_values(l,m,s,alf,a,b,d,e,TOL,buffer,pow1,pow2,h)

a1 = m(i)
a2 = alf + l + m(i)
a3 = l

root1 = (a2 + sqrt((a2**2) - (4*a1*a3)))/(2*a1)
root2 = (a2 - sqrt((a2**2) - (4*a1*a3)))/(2*a1)

if (root1 > root2) then
  sroot = root2
else
  sroot = root1
end if

return
end subroutine

!-----------------------------------------------------------------------------
----------
subroutine checkV(h,C,What,Chat,X,nmax,s,n,FAIL)
implicit none

integer :: h,buffer
double precision :: pow1,pow2

integer :: nmax,s,i,n,k
integer, dimension (s) :: FAIL
double precision, dimension (s,0:n+1) :: C,Chat,X,V,What

double precision :: l,alf,a,b,d,e,TOL
double precision, dimension(s) :: m

call queue_values(l,m,s,alf,a,b,d,e,TOL,buffer,pow1,pow2,h)

FAIL = 0

do i = 1,s

  V(i,n) = ((alf+m(i))*Chat(i,n) + X(i,n)*(alf*C(i,n+1) + What(i,n)))/(alf +
m(i) - m(i)*X(i,n))

  V(i,n+1) = (alf*C(i,n+1) + What(i,n) + m(i)*Chat(i,n))/(alf + m(i) -
m(i)*X(i,n))

end do

do i = 1,s
  do k = n-1,0,-1

    V(i,k) = Chat(i,k) + X(i,k)*V(i,k+1)

  end do
end do

!if (n == 12) then
!  print*,"V(i,k) when n = 12 is:"
!  print*," "
```

```
!  do k = 0,n+1
!    print*,V(:,k)
!  end do

!print*," "
!print*,"X(:,0) = ",X(:,0)
!print*," "
!print*,"Chat(:,0) = ",Chat(:,0)
!end if

do i=1,s
  if (l*(V(i,n+1) - V(i,n)) > 1.002*What(i,n)) then
    FAIL(i) = 1
  else if (l*(V(i,n) - V(i,n-1)) > 1.002*What(i,n)) then
    FAIL(i) = 2
  else if ((V(i,n+1) - V(i,n)) < (V(i,n) - V(i,n-1))) then
    FAIL(i) = 3
  end if
  do k = n-1,1,-1
    if ((V(i,k+1) - V(i,k)) < (V(i,k) - V(i,k-1))) FAIL(i) = 4
  end do
end do

return
end subroutine

!-----------------------------------------------------------------------------
----------

subroutine starting_vals(inds,opts,h)
implicit none

integer :: h
double precision :: inds,opts

inds = 0.0
opts = 0.0

return
end subroutine

!-----------------------------------------------------------------------------
----------

subroutine queue_values(l,m,s,alf,a,b,d,e,TOL,buffer,pow1,pow2,h)
implicit none

integer :: s,h,buffer
integer, dimension(8) :: buffera
double precision :: l,alf,a,b,d,e,TOL,pow1,pow2
double precision, dimension(s) :: m

double precision, dimension(8) :: power1a,power2a,la


alf = 0.05129

m(1) = 2.9
m(2) = 2.1

s = 2

a = 1.0 ! 1.0 ! 0.0 ! 1.0 ! 1.5
b = 2.0 ! 0.0       ! 1.0 ! 1.1
d = 2.0 ! 1.5 ! 0.0 ! 1.0 ! 0.7
e = 2.0 ! 0.0       ! 1.0 ! 1.9

TOL = 0.0005

la = (/0.6,0.85,0.6,0.85,0.6,0.85,0.6,0.85/)
power1a = (/1.0,1.0,2.0,2.0,3.0,3.0,1.0,1.0/)
power2a = (/2.0,2.0,3.0,3.0,4.0,4.0,2.0,2.0/)
buffera = (/0,0,0,0,0,0,2,2/)

pow1 = power1a(h)
```

```
pow2 = power2a(h)
buffer = buffera(h)

l = (m(1)+m(2))*la(h)

return
end subroutine

!---------------------------------------------------------------------------
------------
```

Appendix B

This appendix contains the Fortran 95 code for the programme we used to calculate the undiscounted routing control costs as in Section 2.5.2. Here we consider the optimal, policy improvement and Whittle index policies for a 2 class system.

```
program routing
implicit none

integer :: Nmax,BError,h,buffer
double precision :: a,b,d,e,l,TOL,Wcost,Optcost,PIcost
double precision, dimension(2) :: m
open(unit=7,file="Routing.dat")

do h = 1,64

call qvals(buffer,a,b,d,e,l,m,Nmax,BError,TOL,h)

Wcost = 0.0
Optcost = 0.0
PIcost = 0.0

call optimal(Optcost,h)
call whittle(Wcost,h)
call policyimp(PIcost,h)

write(unit=7,fmt="(a)") "   l    m(1)    m(2)    a      b      d      e
Nmax   "
write(unit=7,fmt="(7f7.3,i5)") l,m(1),m(2),a,b,d,e,Nmax
write(unit=7,fmt="(a)") " "
write(unit=7,fmt="(a,f12.6)") "The optimal cost for any policy with this
queue setup & parameters is : ",OptCost
write(unit=7,fmt="(a)") " "
write(unit=7,fmt="(a,f12.6)") "the cost when following the whittle index
policy is :                   ",WCost
write(unit=7,fmt="(a,f12.6)") "the suboptimality is:
",(Wcost-OptCost)*100/Optcost
write(unit=7,fmt="(a)") " "
write(unit=7,fmt="(a,f12.6)") "the cost when following the policy
improvement index policy is :       ",PICost
write(unit=7,fmt="(a,f12.6)") "the suboptimality is:
",(PIcost-OptCost)*100/Optcost
write(unit=7,fmt="(a,f12.6)") "the Whittle is sub Policy Improvment by:
",(Wcost-PICost)*100/PIcost

end do

close(unit=7)

end program

!------------------------------------------------------------------------

subroutine whittle(Wcost,h)
implicit none

integer :: n,n1,n2,BError,Nmax,count,r,h,buffer
double precision :: a,b,d,e,l,smallest,largest,diff,TOL,WCost,U
integer, dimension(2) :: ismall,ilarge
double precision, dimension(2) :: m
double precision, dimension(4) :: temp
double precision, allocatable, dimension(:,:) :: C,W,Vnew,Vold,W2,TEMP2

!call subroutine to get queue parameter values
call qvals(buffer,a,b,d,e,l,m,Nmax,BError,TOL,h)

allocate( C(2,0:Nmax+1) )
allocate( W(2,0:Nmax) )
allocate( W2(2,0:Nmax) )
allocate( TEMP2(2,Nmax+1) )
allocate( Vnew(0:Nmax,0:Nmax) )
allocate( Vold(0:Nmax,0:Nmax) )

!initialize costs C, and index W, vectors
C = 0.0
W = 0.0
count = 0


!set cost function using queue parameters subroutine - convex costs
do n = buffer,Nmax+1
```

```
  C(1,n) = a*real(n-buffer) + b*(real(n-buffer)**2.0)
  C(2,n) = d*real(n-buffer) + e*(real(n-buffer)**2.0)

end do

!calculate whittle index : method 1
do n = 0,Nmax

  W(1,n) = ( C(1,n+1) - C(1,n) )*(1.0/l)*( (l/m(1)) - (
(l/m(1))**(real(n+2)) ) )/( 1.0 - (l/m(1)) )

  W(2,n) = ( C(2,n+1) - C(2,n) )*(1.0/l)*( (l/m(2)) - (
(l/m(2))**(real(n+2)) ) )/( 1.0 - (l/m(2)) )

end do

do n = 1,Nmax

  W(1,n) = W(1,n) + W(1,n-1)

  W(2,n) = W(2,n) + W(2,n-1)

end do

!calculate whittle index a different way : method 2
TEMP2 = 0.0
W2 = 0.0

do n = 1,Nmax
  do r = 1,n
    TEMP2(1,n) = TEMP2(1,n) + (l/m(1))**real(r)
    TEMP2(2,n) = TEMP2(2,n) + (l/m(2))**real(r)
  end do
end do

do n = 0,Nmax
  W2(1,n) = C(1,n+1)*TEMP2(1,n+1)
  W2(2,n) = C(2,n+1)*TEMP2(2,n+1)
end do

do n = 0,Nmax
  do r = 0,n
    W2(1,n) = W2(1,n) - C(1,r)*(l/m(1))**real(r+1)
    W2(2,n) = W2(2,n) - C(2,r)*(l/m(2))**real(r+1)
  end do
end do

W2 = W2/l

open(unit=7,file="storeVW.dat")

do n = 0,Nmax
  write(unit=7,fmt="(i6,4f25.5)") n,W(:,n),W2(:,n)
end do

!uniformise queue parameters so that on average have one event per unit time
U = l + m(1) + m(2)

l = l/U
m(1) = m(1)/U
m(2) = m(2)/U

!initialize value function vectors
Vnew = 0.0
Vold = 0.0

!compute value function - using value iteration algorithm
30 Vold =  Vnew

count = count + 1

do n1 = 0,Nmax
  do n2 = 0,Nmax
```

```
!use temp vector to deal with boundary cases

    if (n1>0) then
      temp(1) = Vold(n1-1,n2)
    else
      temp(1) = Vold(n1,n2)
    end if

    if (n2>0) then
      temp(2) = Vold(n1,n2-1)
    else
      temp(2) = Vold(n1,n2)
    end if

    if (n1<Nmax) then
      temp(3) = Vold(n1+1,n2)
    else
      temp(3) = Vold(n1,n2)
    end if

    if (n2<Nmax) then
      temp(4) = Vold(n1,n2+1)
    else
      temp(4) = Vold(n1,n2)
    end if

!if W1 smaller send to queue one & similar for queue 2

    if (W2(1,n1) <= W2(2,n2)) then

      Vnew(n1,n2) = C(1,n1) + C(2,n2) + m(1)*temp(1) + m(2)*temp(2) +
l*temp(3)

    else

      Vnew(n1,n2) = C(1,n1) + C(2,n2) + m(1)*temp(1) + m(2)*temp(2) +
l*temp(4)

    end if

  end do
end do


!compute the bounds
smallest = 1000000000000.0
largest = -1000000000000.0

do n1 = 0,Nmax-BError
  do n2 = 0,Nmax-BError

    if (smallest > Vnew(n1,n2) - Vold(n1,n2)) then
      smallest = Vnew(n1,n2) - Vold(n1,n2)
      ismall = (/n1,n2/)
    end if
    if (largest  < Vnew(n1,n2) - Vold(n1,n2)) then
      largest = Vnew(n1,n2) - Vold(n1,n2)
      ilarge = (/n1,n2/)
    end if

  end do
end do

!if bounds within set tolerance stop, otherwise repeat.
diff = largest - smallest

!write(unit=7,fmt="(3f20.6,4i4)") smallest, largest, diff, ismall, ilarge

if (count > 2000000) goto 300
if (diff < 0.0 .or. diff > TOL*smallest) goto 30

!calculate average cost of following this policy
300 WCost = (largest + smallest)/2.0

!close(unit=7)
```

```
print*,"count = ",count
print*,"the cost when following the whittle index policy is : ",WCost

return
end subroutine

!-----------------------------------------------------------------------
-----

subroutine optimal(Optcost,h)
implicit none

integer :: Nmax,BError,n,n1,n2,count,h,buffer
double precision :: a,b,d,e,l,TOL,U,smallest,largest,diff,OptCost
double precision, dimension(2) :: m,cost
double precision, dimension(4) :: temp
double precision, allocatable, dimension(:,:) :: C,Vnew,Vold

!get queue parameters
call qvals(buffer,a,b,d,e,l,m,Nmax,BError,TOL,h)

allocate( C(2,0:Nmax) )
allocate( Vnew(0:Nmax,0:Nmax) )
allocate( Vold(0:Nmax,0:Nmax) )

!uniformise queue parameters so that on average have one event per unit time
U = l + m(1) + m(2)

l = l/U
m(1) = m(1)/U
m(2) = m(2)/U

!initialize cost C vectors
C = 0.0

!set cost function using queue parameters subroutine - convex costs
do n = buffer,Nmax+1

  C(1,n) = a*real(n-buffer) + b*(real(n-buffer)**2.0)
  C(2,n) = d*real(n-buffer) + e*(real(n-buffer)**2.0)

end do

!do value iteration to find the average optimal cost per unit time

Vnew = 0.0
Vold = 0.0
count = 0

!open(unit=7,file="storeVO.dat")

20 Vold = Vnew

count = count + 1
!print*,"count = ",count

do n1 = 0,Nmax
  do n2 = 0,Nmax

    if (n1>0) then
      temp(1) = Vold(n1-1,n2)
    else
      temp(1) = Vold(n1,n2)
    end if

    if (n2>0) then
      temp(2) = Vold(n1,n2-1)
    else
      temp(2) = Vold(n1,n2)
    end if

    if (n1<Nmax) then
      temp(3) = Vold(n1+1,n2)
    else
```

```
      temp(3) = Vold(n1,n2)
    end if

    if (n2<Nmax) then
      temp(4) = Vold(n1,n2+1)
    else
      temp(4) = Vold(n1,n2)
    end if

!calculate costs if send to queue 1 and if send to queue 2
    cost(1) = C(1,n1) + C(2,n2) + m(1)*temp(1) + m(2)*temp(2) + l*temp(3)
    cost(2) = C(1,n1) + C(2,n2) + m(1)*temp(1) + m(2)*temp(2) + l*temp(4)

!set value function to be the one with the smaller costs
    if (cost(1) < cost(2)) then
      Vnew(n1,n2) = cost(1)
    else
      Vnew(n1,n2) = cost(2)
    end if

  end do
end do

smallest = 1000000.0
largest = -1000000.0

do n1 = 0,Nmax-BError
  do n2 = 0,Nmax-BError

    if (smallest > Vnew(n1,n2) - Vold(n1,n2)) smallest = Vnew(n1,n2) -
Vold(n1,n2)
    if (largest  < Vnew(n1,n2) - Vold(n1,n2)) largest  = Vnew(n1,n2) -
Vold(n1,n2)

  end do
end do

diff = largest - smallest

!write(unit=7,fmt="(3f12.6)") smallest, largest, diff

if (count > 1999999) goto 200
if (diff > smallest*TOL .or. diff < 0.0) goto 20

200 OptCost = (smallest + largest)/2.0

!close(unit=7)

print*,"count = ",count
print*,"The optimal cost for any policy with this queue setup & parameters
is ",OptCost
print*," "

return
end subroutine

!----------------------------------------------------------------------------

subroutine policyimp(PIcost,h)
implicit none

integer :: n,n1,n2,Nmax,BError,IFAIL,count,h,buffer
double precision ::
a,b,d,e,l,TOL,Th,temp1,temp2,smallest,largest,PICost,U,diff,y,z,FC1,FC2,X,F,
BoundE
integer, dimension(2) :: ismall, ilarge
double precision, dimension(2) :: m,p,T,Pim
double precision, dimension(4) :: temp
double precision, allocatable, dimension(:,:) :: Kh
double precision, allocatable, dimension(:,:) :: C,K,Vold,Vnew

!get queue parameters
call qvals(buffer,a,b,d,e,l,m,Nmax,BError,TOL,h)

allocate( C(2,0:Nmax+80) )
```

```
allocate( Kh(2,0:Nmax) )
allocate( K(2,0:Nmax) )
allocate( Vold(0:Nmax,0:Nmax) )
allocate( Vnew(0:Nmax,0:Nmax) )

!uniformise queue parameters so that on average have one event per unit time
U = l + m(1) + m(2)

l = l/U
m(1) = m(1)/U
m(2) = m(2)/U

! check certain system constraints hold
IFAIL = 0
call check(IFAIL,h)
if (IFAIL == 0) then
  print*,"system okay"
else if (IFAIL == 1) then
  print*,"ERROR: unstable queues"
else if (IFAIL == 2) then
  print*,"ERROR: uninteresting problem"
else if (IFAIL == 3) then
  print*,"ERROR: Holding costs must be +ve convex function"
else
  print*,"ERROR: ?"
end if

!initialize cost & index vectors
C = 0.0
Pim = 0.0
T = 0.0
Th = 0.0
K = 0.0
Kh = 0.0
count = 0

!calculate holding costs function - convex
do n = buffer,Nmax

  C(1,n) = a*real(n-buffer) + b*(real(n-buffer)**2.0)
  C(2,n) = d*real(n-buffer) + e*(real(n-buffer)**2.0)

end do

!call subroutine to find the best possible static policy after finding
allowed range
temp1 = m(1)/l
temp2 = 1.0 - m(2)/l

if (temp2 > 0.0 .and. temp2 < 1.0) then
  y = temp2
else
  y = 0.0
end if

if (temp1 < 1.0 .and. temp1 > 0.0) then
  z = temp1
else
  z = 1.0
end if

BoundE = (z-y)*0.05
z = z - BoundE
y = y + BoundE

call Statp(y,z,p,h)

!p(1) = 0.532000
!p(2) = 1.0 - 0.532000

!can only use this if have access to NagRoutines
!call  NAGMIN(X,F)
!p(1) = X
!p(2) = 1.0 - X
```

```
!calculate holding costs function - convex
do n = buffer,Nmax

  C(1,n) = a*real(n-buffer) + b*(real(n-buffer)**2.0)
  C(2,n) = d*real(n-buffer) + e*(real(n-buffer)**2.0)

end do

!call subroutines to calculate values required to find the required index
call Kdiff(1,p,Nmax,Kh,h)
K(1,:) = Kh(1,:)
print*,"Kh(1,0) = ",Kh(1,0)
print*,"K(1,0) = ",K(1,0)

!calculate holding costs function - convex
do n = buffer,Nmax

  C(1,n) = a*real(n-buffer) + b*(real(n-buffer)**2.0)
  C(2,n) = d*real(n-buffer) + e*(real(n-buffer)**2.0)

end do

call Tdiff(1,p,Th,h)
T(1) = Th
print*,"T(1) = ",T(1)


call Kdiff(2,p,Nmax,Kh,h)
K(2,:) = Kh(2,:)
print*,"Kh(2,0) = ",Kh(2,0)
print*,"K(2,0) = ",K(2,0)

!calculate holding costs function - convex
do n = buffer,Nmax

  C(1,n) = a*real(n-buffer) + b*(real(n-buffer)**2.0)
  C(2,n) = d*real(n-buffer) + e*(real(n-buffer)**2.0)

end do

call Tdiff(2,p,Th,h)
T(2) = Th
print*,"T(2) = ",T(2)

call FUNCT2(p(1),FC1,FC2,h)
!use value iteration algirithm to find expected average cost per unit time

print*,"p = ",p
print*,"FC1 = ",FC1
print*,"FC2 = ",FC2

!open(unit=7,file="storeVPI.dat")
!do n = 0,Nmax
!  write(unit=7,fmt="(i5,2f18.6)") n,K(1,n),K(2,n)
!end do

!  write(unit=7,fmt="(i5,4f18.6)") n,FC1,T(1),FC2,T(2)

!do n = 0,Nmax
!  write(unit=7,fmt="(i5,2f18.6)") n,K(1,n) - FC1*T(1),K(2,n) - FC2*T(2)
!end do


Vold = 0.0
Vnew = 0.0

!calculate holding costs function - convex
do n = buffer,Nmax

  C(1,n) = a*real(n-buffer) + b*(real(n-buffer)**2.0)
  C(2,n) = d*real(n-buffer) + e*(real(n-buffer)**2.0)

end do

10 Vold = Vnew
```

```
count = count + 1

do n1 = 0,Nmax
  do n2 = 0,Nmax

    if (n1 > 0) then
      temp(1) = Vold(n1-1,n2)
    else
      temp(1) = Vold(n1,n2)
    end if

    if (n2 > 0) then
      temp(2) = Vold(n1,n2-1)
    else
      temp(2) = Vold(n1,n2)
    end if

    if (n1 < Nmax) then
      temp(3) = Vold(n1+1,n2)
    else
      temp(3) = Vold(n1,n2)
    end if

    if (n2 < Nmax) then
      temp(4) = Vold(n1,n2+1)
    else
      temp(4) = Vold(n1,n2)
    end if

    Pim(1) = K(1,n1) - FC1*T(1)
    Pim(2) = K(2,n2) - FC2*T(2)

!     print*,"Pim(1) = ",Pim(1)
 !    print*,"Pim(2) = ",Pim(2)

    if (Pim(1) < Pim(2)) then

      Vnew(n1,n2) = C(1,n1) + C(2,n2) + m(1)*temp(1) + m(2)*temp(2) +
l*temp(3)

    else

      Vnew(n1,n2) = C(1,n1) + C(2,n2) + m(1)*temp(1) + m(2)*temp(2) +
l*temp(4)

    end if

  end do
end do

smallest = 1000000000000.0
largest = -1000000000000.0

do n1 = 0,Nmax-BError
  do n2 = 0,Nmax-BError

  if (smallest > Vnew(n1,n2) - Vold(n1,n2)) then
    smallest = Vnew(n1,n2) - Vold(n1,n2)
    ismall = (/n1,n2/)
  end if
  if (largest  < Vnew(n1,n2) - Vold(n1,n2)) then
    largest  = Vnew(n1,n2) - Vold(n1,n2)
    ilarge = (/n1,n2/)
  end if

  end do
end do

diff = largest - smallest

!write(unit=7,fmt="(3f18.6,4i4)") smallest, largest, diff, ismall, ilarge

if (count > 2000000) goto 100
if (diff > TOL*smallest .or. diff < 0.0) goto 10
```

```
100 print*," "

!close(unit=7)

PICost = (largest + smallest)/2.0

print*,"count = ",count
print*,"the policy improvement index gives us a policy with costs of
",PICost

return
end subroutine

!---------------------------------------------------------------------------
------------------------------

subroutine check(IFAIL,h)
implicit none

integer :: Nmax,IFAIL,BError,h,buffer
double precision :: a,b,d,e,l,TOL,service,maxm
double precision, dimension(2) :: m

call qvals(buffer,a,b,d,e,l,m,Nmax,BError,TOL,h)

service = m(1) + m(2)

if (m(1) > m(2)) then
  maxm = m(1)
else
  maxm = m(2)
end if


if (l > service) IFAIL = 1
if (maxm > l)    IFAIL = 2
if (a<0.0 .or. b<0.0 .or. d<0.0 .or. e<0.0) IFAIL = 3

return
end subroutine

!---------------------------------------------------------------------------
------------------------------

subroutine NAGMIN(X,F,h)
implicit none

integer :: NOUT, IFAIL, MAXCAL, Nmax, BError, h, buffer
double precision :: y,z,EPS,F,T,X,l,a,b,d,e,TOL,BoundE
double precision, dimension(2) :: m
EXTERNAL E04ABF, FUNCT

call qvals(buffer,a,b,d,e,l,m,Nmax,BError,TOL,h)

EPS = 0.0e0
T = 0.0e0

if ((l - m(2))/l > 0.0) then
  y = (l - m(2))/l
else
  y = 0.0e0
end if

if (m(1)/l < 1.0) then
  z = m(1)/l
else
  z = 1.0e0
end if

BoundE = (z-y)*0.05
z = z - BoundE
y = y + BoundE
```

```
MAXCAL = 30
IFAIL = 1

CALL E04ABF(FUNCT,EPS,T,Y,Z,MAXCAL,X,F,IFAIL)

IF (IFAIL == 1) THEN
  PRINT*,"Parameter outside expected range"
ELSE
  IF (IFAIL == 2) THEN
    PRINT*,"Results after MAXCAL function evaluations are"
    PRINT*," "
  END IF
  PRINT*,"The minimum lies in the interval ",Y," to ",Z
  PRINT*,"Its estimated position is ",X
  PRINT*,"Where the value function is ",F
  PRINT*, MAXCAL," Function evaluations were required"
END IF

RETURN
END SUBROUTINE

!-----------------------------------------------------------------------
--------------

subroutine Statp(y,z,p,h)
implicit none

integer :: i,h
double precision :: y,z,minimum,xc,FC
double precision, dimension(2) :: p

minimum = 1000000000.0
!open(unit=7,file="funct.dat")
do i = 1,99

  xc = y + ( (z-y)/100.0 )*real(i)
  call FUNCT(xc,FC,h)

  !write(unit=7,fmt="(2f16.6)") xc,FC
  if (minimum > FC) then
    p(1) = xc
    minimum = FC
  end if

end do
!close(unit=7)

p(2) = 1.0 - p(1)

print*,"I calculate the minimum using my vulgar method as ",minimum
print*,"which can be found at p = ",p
print*," "

return
end subroutine

!-----------------------------------------------------------------------
------------------------------

subroutine FUNCT(xc,FC,h)
implicit none

integer :: n,Nmax,BError,h,buffer
double precision :: a,b,d,e,l,TOL,xc,FC1,FC2,FC,temp1,temp2,temp3,temp4
double precision, dimension(2) :: m,p
double precision, allocatable, dimension(:,:) :: C

!get queue/cost parameters
call qvals(buffer,a,b,d,e,l,m,Nmax,BError,TOL,h)

allocate( C(2,0:Nmax+(1*Nmax)) )

FC1 = 0.0
FC2 = 0.0
FC  = 0.0
```

```
C = 0.0

do n = buffer,Nmax+(1.0*Nmax)

  C(1,n) = a*real(n-buffer) + b*(real(n-buffer)**2.0)
  C(2,n) = d*real(n-buffer) + e*(real(n-buffer)**2.0)

end do

p(1) = xc
p(2) = 1.0 - xc

temp1 = l*p(1)/m(1)
temp2 = 1.0 - temp1
temp3 = l*p(2)/m(2)
temp4 = 1.0 - temp3

do n = 0,Nmax+(1.0*Nmax)

  FC1 = FC1 + C(1,n)*(temp1**(real(n)))*temp2
  FC2 = FC2 + C(2,n)*(temp3**(real(n)))*temp4

end do

FC = FC1 + FC2

return
end subroutine
!-----------------------------------------------------------------------------
------------------------------

subroutine FUNCT2(xc,FC1,FC2,h)
implicit none

integer :: n,Nmax,BError,h,buffer
double precision :: a,b,d,e,l,TOL,xc,FC1,FC2,temp1,temp2,temp3,temp4
double precision, dimension(2) :: m,p
double precision, allocatable, dimension(:,:) :: C

!get queue/cost parameters
call qvals(buffer,a,b,d,e,l,m,Nmax,BError,TOL,h)

allocate( C(2,0:Nmax+(2*Nmax)) )

FC1 = 0.0
FC2 = 0.0
C = 0.0

do n = buffer,Nmax+(2.0*Nmax)

  C(1,n) = a*real(n-buffer) + b*(real(n-buffer)**2.0)
  C(2,n) = d*real(n-buffer) + e*(real(n-buffer)**2.0)

end do

p(1) = xc
p(2) = 1.0 - xc

temp1 = l*p(1)/m(1)
temp2 = 1.0 - temp1
temp3 = l*p(2)/m(2)
temp4 = 1.0 - temp3

do n = 0,Nmax+(2.0*Nmax)

  FC1 = FC1 + C(1,n)*(temp1**real(n))*temp2
  FC2 = FC2 + C(2,n)*(temp3**real(n))*temp4

end do

return
end subroutine

!-----------------------------------------------------------------------------
------------------------------
```

```
subroutine Kdiff(i,p,Nmax,Kh,h)
implicit none

integer ::n,i,Nmax,BError,j,h,buffer
double precision :: a,b,d,e,l,TOL,temp1,counter
double precision, dimension(2) :: m,p
double precision, dimension(2,0:Nmax) :: Kh
double precision, allocatable, dimension(:,:) :: C,temp2

!get queue parameters
call qvals(buffer,a,b,d,e,l,m,Nmax,BError,TOL,h)
Kh(i,:) = 0.0

allocate( C(2,0:Nmax+Nmax+(2*Nmax)+1) )
allocate( temp2(2,0:Nmax+Nmax) )

do n = buffer,Nmax+Nmax+(2.0*Nmax)+1

  C(1,n) = a*real(n-buffer) + b*(real(n-buffer)**2.0)
  C(2,n) = d*real(n-buffer) + e*(real(n-buffer)**2.0)

end do

temp1 = l*p(i)/m(i)

do n = 0,Nmax+Nmax

  temp2(i,n) = C(i,n)/m(i)

end do

!calc K(n) - K(n-1)
!do n = 0,Nmax

!  counter = 0.0
!  do j = n,n+Nmax  !could try "n+80" instead of "Nmax+80" so that have the
same accuracy on all values?

!    Kh(i,n) = Kh(i,n)+( temp1**(counter) )*temp2(i,j)

!    counter = counter + 1.0
!  end do

!end do

!!!!!!!!!!!! second go

do n = 0,Nmax

  do j = 0,n+(2.0*Nmax)

    Kh(i,n) = Kh(i,n) + ((l*p(i)/m(i))**(real(j)))*C(i,n+j+1)/m(i)

  end do

end do

print*,"Kh(",i,",0) = ",Kh(i,0)


return
end subroutine

!---------------------------------------------------------------------------
------------------------------

subroutine Tdiff(i,p,T,h)
implicit none

integer :: Nmax,BError,i,h,buffer
double precision :: a,b,d,e,l,TOL,temp1,temp2,T
double precision, dimension(2) :: m,p

!get queue parameters
```

```
call qvals(buffer,a,b,d,e,l,m,Nmax,BError,TOL,h)

!temp1 = 1.0/m(i)

!temp2 = temp1/(1.0 - l*p(i)*temp1)

!calc T(n) - T(n-1)
!T = temp1 + l*p(i)*temp1*temp2

!!!!!!!!1!second go
T = 1.0/(m(i) - l*p(i))

return
end subroutine

!-----------------------------------------------------------------------
-

subroutine qvals(buffer,a,b,d,e,l,m,Nmax,BError,TOL,h)
implicit none

integer :: Nmax,BError,h,buffer
double precision :: a,b,d,e,l,TOL
double precision, dimension(2) :: m
double precision, dimension(64) :: la,m1a,m2a,aa,ba,da,ea

!allocate( la(h) )
!allocate( m1a(h) )
!allocate( m2a(h) )
!allocate( aa(h) )
!allocate( ba(h) )
!allocate( da(h) )
!allocate( ea(h) )

Nmax = 199
BError = 10
buffer = 0

la = 0.6
m1a =  (/3.0,2.9,2.8,2.7,3.0,2.9,2.8,2.7,3.0,2.9,2.8,2.7,3.0,2.9,2.8,2.7, &
       & 3.0,2.9,2.8,2.7,3.0,2.9,2.8,2.7,3.0,2.9,2.8,2.7,3.0,2.9,2.8,2.7, &
       & 3.0,2.9,2.8,2.7,3.0,2.9,2.8,2.7,3.0,2.9,2.8,2.7,3.0,2.9,2.8,2.7, &
       & 3.0,2.9,2.8,2.7,3.0,2.9,2.8,2.7,3.0,2.9,2.8,2.7,3.0,2.9,2.8,2.7/)
m2a =  (/3.0,3.1,3.2,3.3,3.0,3.1,3.2,3.3,3.0,3.1,3.2,3.3,3.0,3.1,3.2,3.3, &
       & 3.0,3.1,3.2,3.3,3.0,3.1,3.2,3.3,3.0,3.1,3.2,3.3,3.0,3.1,3.2,3.3, &
       & 3.0,3.1,3.2,3.3,3.0,3.1,3.2,3.3,3.0,3.1,3.2,3.3,3.0,3.1,3.2,3.3, &
       & 3.0,3.1,3.2,3.3,3.0,3.1,3.2,3.3,3.0,3.1,3.2,3.3,3.0,3.1,3.2,3.3/)

aa =  2.0
ba = (/0.1,0.1,0.1,0.1,0.6,0.6,0.6,0.6,1.0,1.0,1.0,1.0,2.0,2.0,2.0,2.0, &
     & 0.1,0.1,0.1,0.1,0.6,0.6,0.6,0.6,1.0,1.0,1.0,1.0,2.0,2.0,2.0,2.0, &
     & 0.1,0.1,0.1,0.1,0.6,0.6,0.6,0.6,1.0,1.0,1.0,1.0,2.0,2.0,2.0,2.0, &
     & 0.1,0.1,0.1,0.1,0.6,0.6,0.6,0.6,1.0,1.0,1.0,1.0,2.0,2.0,2.0,2.0/)
da =  1.0
ea = (/0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1, &
     & 0.6,0.6,0.6,0.6,0.6,0.6,0.6,0.6,0.6,0.6,0.6,0.6,0.6,0.6,0.6,0.6, &
     & 1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0, &
     & 2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0/)

m(1) = m1a(h)
m(2) = m2a(h)
l = (m(1)+m(2))*la(h)

a = aa(h)
b = ba(h)
d = da(h)
e = ea(h)

TOL = 0.001

return
end subroutine
```

Appendix C

This appendix contains the Fortran 95 code for the programme we used to simulate the undiscounted routing control costs as in Section 2.5.3. Here we consider Whittle index policy for a 5 class system compared to some other standard policies as explained in the numerical section.

```
program simulation
implicit none

integer :: size,k,count,Nmax,num,BError,actsize,numsim,simnumb,s,r
integer, dimension(5) :: buffer
integer, dimension(5) :: m
double precision ::
Tsize,TOL,SUMINDEXC,INDEXC,SUMINDEXSQ,INDEXVAR,WIcost,LONGQC,LQcost,MYOPICC,
MYcost,STATICC,STcost
double precision :: SUMLONGQC,SUMLONGQSQ,LONGQVAR, &
                    &
SUMSTATICC,SUMSTATICSQ,STATICVAR,SUMMYOPICC,SUMMYOPICSQ,MYOPICVAR,in2stat,l
double precision, dimension(5) :: mu,stationary2
double precision, dimension(5) :: a,b
double precision, dimension(500000) :: IA,AA
double precision, allocatable, dimension(:,:) :: C,W,pi

call inputdata(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

allocate( C(5,0:Nmax+1) )
allocate( W(5,0:Nmax) )
allocate( pi(5,0:Nmax) )

s = 5

in2stat = Tsize*0.667

numsim = 70
!open(unit=7,file="Simulationadcontdata.dat")
  IA = 0.0
  AA = 0.0
  C = 0.0

open(unit=7,file="aSim_quad_high_rho_check.dat")!,status="old")

do r=5,9
  if (r == 0) then
    write(unit=7,fmt="(a)") "rho = 0.6 - linear costs buffer=0"
  else if (r == 5) then
    write(unit=7,fmt="(a)") "rho = 0.85 - linear costs buffer=0"
  else if (r == 10) then
    write(unit=7,fmt="(a)") "rho = 0.6 - linear costs buffer=2"
  else if (r == 15) then
    write(unit=7,fmt="(a)") "rho = 0.85 - linear costs buffer=2"
  else if (r == 20) then
    write(unit=7,fmt="(a)") "rho = 0.6 - linear costs buffer=4"
  else if (r == 24) then
    write(unit=7,fmt="(a)") "rho = 0.85 - linear costs buffer=4"

    end if

  call inputdata(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

  do k = 1,5
    do num=buffer(k),Nmax+1
      C(k,num) = a(k)*(real(num-buffer(k))**1.0) +
b(k)*(real(num-buffer(k))**2.0)
    end do
  end do

  write(unit=7,fmt="(a,5f10.6)") "a cost vector       = ",a
  write(unit=7,fmt="(a,5f10.6)") "b cost vector       = ",b
  write(unit=7,fmt="(a,f10.6)") "arrival rate        = ",l
  write(unit=7,fmt="(a,5f10.6)") "service time vector = ",mu
  write(unit=7,fmt="(a,f10.4,a,f10.4) ") "Tsize = ",Tsize,"     in2stat =
",in2stat
  write(unit=7,fmt="(a,i8,a,i5)") "Nmax = ",Nmax,"     numsim = ",numsim

  W = 0.0
  call Windex(r,s,Nmax,C,W)

  SUMINDEXC = 0.0
  SUMINDEXSQ = 0.0
  INDEXVAR = 0.0
```

```
  INDEXC = 0.0

  SUMLONGQC = 0.0
  SUMLONGQSQ = 0.0
  LONGQVAR = 0.0
  LONGQC = 0.0

  SUMSTATICC = 0.0
  SUMSTATICSQ = 0.0
  STATICVAR = 0.0
  STATICC = 0.0

  SUMMYOPICC = 0.0
  SUMMYOPICSQ = 0.0
  MYOPICVAR = 0.0
  MYOPICC = 0.0


  print*,"Index Policy"
  !write(unit=7,fmt="(a)") "INDEX POLICY"

  do simnumb = 1,numsim

    call getarrivals(Nmax,actsize,IA,AA)
!print*,"1"

    call indexcost(r,Nmax,actsize,IA,AA,W,WIcost)

    SUMINDEXSQ = SUMINDEXSQ + WICOST**2.0
    SUMINDEXC  = SUMINDEXC + WIcost

!print*,"2"
  end do
  INDEXVAR = (SUMINDEXSQ -
(real(numsim)*((SUMINDEXC/real(numsim))**2.0)))/(real(numsim-1))
!  (SUMINDEXSQ/real(numsim)) - ((SUMINDEXC/real(numsim))**2.0)
  INDEXC = SUMINDEXC/real(numsim)

  print*,"simulation ",simnumb," INDEX cost = ",WIcost

print*,"Finished INDEXC = ",INDEXC
write(unit=7,fmt="(a)") "******** INDEX ********"
write(unit=7,fmt="(a,f18.5)") "INDEX Cost = ",INDEXC
write(unit=7,fmt="(a,f18.5)") "Sub index = ",(Indexc-INDEXC)*100.0/INDEXC
write(unit=7,fmt="(a,f18.5)") "Sample Mean S.D. = ",sqrt(INDEXVAR/numsim)
write(unit=7,fmt="(a)") " "

print*,"Longest Queue"
do simnumb = 1,numsim
!    print*,"number = ",simnumb
  call getarrivals(Nmax,actsize,IA,AA)
  call longestq(r,Nmax,actsize,IA,AA,LQcost)
  SUMLONGQSQ = SUMLONGQSQ + (LQCOST**2.0)
  SUMLONGQC  = SUMLONGQC + LQcost
end do
LONGQVAR = (SUMLONGQSQ -
(real(numsim)*((SUMLONGQC/real(numsim))**2.0)))/(real(numsim-1))
!(SUMLONGQSQ/real(numsim)) - ((SUMLONGQC/real(numsim))**2.0)
LONGQC = SUMLONGQC/real(numsim)
print*,"Finished LONGQ = ",LONGQC
write(unit=7,fmt="(a)") "******** LONGEST QUEUE ********"
write(unit=7,fmt="(a,f18.5)") "COST = ",LONGQC
write(unit=7,fmt="(a,f18.5)")   "SUB INDEX  = ",100.0*(LONGQC-INDEXC)/INDEXC
write(unit=7,fmt="(a,f18.5)")   "Sample Mean S.D. = ",sqrt(LONGQVAR/numsim)
write(unit=7,fmt="(a)") " "

print*,"Myopic Policy"
do simnumb = 1,numsim
!    print*,"number = ",simnumb
  call getarrivals(Nmax,actsize,IA,AA)
  call myopic(r,Nmax,actsize,IA,AA,MYcost)
  SUMMYOPICSQ = SUMMYOPICSQ + (MYCOST**2.0)
  SUMMYOPICC  = SUMMYOPICC + MYcost
end do
!print*,"next"
```

```
MYOPICVAR = (SUMMYOPICSQ -
(real(numsim)*((SUMMYOPICC/real(numsim))**2.0)))/(real(numsim-1))
!(SUMMYOPICSQ/real(numsim)) - ((SUMMYOPICC/real(numsim))**2.0)
MYOPICC = SUMMYOPICC/real(numsim)
print*,"Finished MYOPICC = ",MYOPICC
write(unit=7,fmt="(a)") "******** MYOPIC ********"
write(unit=7,fmt="(a,f18.15)") "COST = ",MYOPICC
write(unit=7,fmt="(a,f18.15)")   "SUB INDEX  =
",100.0*(MYOPICC-INDEXC)/INDEXC
write(unit=7,fmt="(a,f18.15)")   "Sample Mean S.D. =
",sqrt(MYOPICVAR/numsim)
write(unit=7,fmt="(a)") " "

print*,"Static Policy"
do simnumb = 1,numsim
!   print*,"number = ",simnumb
  call getarrivals(Nmax,actsize,IA,AA)
  call static(r,Nmax,actsize,IA,AA,STcost,stationary2)
!  print*,"static cost : ",STcost
  SUMSTATICSQ = SUMSTATICSQ + (STCOST**2.0)
  SUMSTATICC  = SUMSTATICC + STcost
end do
STATICVAR = (SUMSTATICSQ -
(real(numsim)*((SUMSTATICC/real(numsim))**2.0)))/(real(numsim-1))
!(SUMSTATICSQ/real(numsim)) - ((SUMSTATICC/real(numsim))**2.0)
STATICC = SUMSTATICC/real(numsim)
print*,"Finished STATICC = ",STATICC
write(unit=7,fmt="(a)") "******** STATIC ********"
write(unit=7,fmt="(a,5f9.6)") "static policy = ",stationary2
write(unit=7,fmt="(a,f18.5)") "COST = ",STATICC
write(unit=7,fmt="(a,f18.5)") "SUB INDEX  = ",100.0*(STATICC-INDEXC)/INDEXC
write(unit=7,fmt="(a,f18.5)") "Samp varience = ",STATICVAR
write(unit=7,fmt="(a,f18.5)") "Sample Mean S.D. =
",sqrt(STATICVAR/real(numsim))
!write(unit=7,fmt="(a)") " "

end do

close(unit=7)

!print*,"3"

end program

!-----------------------------------------------------------------------
-

subroutine check(l,mu,s,FAIL)
implicit none

integer :: FAIL,s,i
double precision :: l,maxmu,summu
double precision, dimension(s) :: mu

FAIL = 0
summu = 0.0

do i = 1,s
  summu = summu + mu(i)
end do


if (l >= summu) FAIL = 1

!if (mu(1) > mu(2)) then
!  maxmu = mu(1)
!else
!  maxmu = mu(2)
!end if

maxmu = max(mu(1),mu(2),mu(3),mu(4),mu(5))

if (maxmu >= l) FAIL = 2

return
```

```
end subroutine

!---------------------------------------------------------------------
-

subroutine Windex(r,s,Nmax,C,W)
implicit none

integer :: s,Nmax,r,n,i,BError,size,num,k
double precision :: TOL,Tsize,l
integer, dimension(5) :: buffer
integer, dimension(5) :: m
double precision, dimension(5) :: mu
double precision, dimension(5) :: a,b
double precision, dimension(s,0:Nmax+1) :: C,W

call inputdata(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

W = 1000000.009
C = 0.0

!print*,"C(2,1) = ",C(2,1)
!print*,"C(2,0) = ",C(2,0)
!print*,"l = ",l
!print*,"mu(2) = ",mu(2)

!print*,"C(2,0+1) - C(2,0) = ",C(2,0+1) - C(2,0)
!print*,"l/mu(2) - (l/mu(2))**(0+2) = ",l/mu(2) - ((l/mu(2))**2)
!print*,"(1.0 - (l/mu(2))) = ",1.0 - (l/mu(2))

!print*,"l/mu(2) = ",l/mu(2)
!print*,"(l/mu(2))**0+2 = ",(l/mu(2))**(0+2)

do k = 1,5
  do num=buffer(k),Nmax+1
    C(k,num) = a(k)*(real(num-buffer(k))**1.0) +
b(k)*(real(num-buffer(k))**2.0)
  end do
end do

!do i = 1,s
!  do n = 0,Nmax
!
!    if (mu(i) < 999999.9) W(i,n) = ( C(i,n+1) - C(i,n) )*( (l/m(i)) - (
(l/m(i))**(n+2) ) )/(l*( 1.0 - (l/m(i)) ))
!
!  end do
!
!  do n = 1,Nmax
!
!    W(i,n) = W(i,n) + W(i,n-1)
!
!  end do
!end do

do i = 1,s

  do n = 0,Nmax
      if (mu(i) < 999999.9) W(i,n) = (C(i,n+1) - C(i,n))*((l/mu(i)) -
((l/mu(i))**(real(n+2))))/(l*(1.0 - (l/mu(i))))
  end do

  do n = 1,Nmax
    W(i,n) = W(i,n) + W(i,n-1)
  end do

end do


!open(unit=7,file="Windex.dat")
!do n = 0,Nmax
!write(unit=7,fmt="(5f25.5)") W(:,n)
!end do
!write(unit=7,fmt="(a)") " 2 "
!do n = 0,Nmax
```

```
!write(unit=7,fmt="(5f25.5)") C(:,n)
!end do
!close(unit=7)

return
end subroutine

!------------------------------------------------------------------------
-
!must be changed to undiscounted index
subroutine Windex2(s,Nmax,W)
implicit none

integer :: s,Nmax,r,n,i,FAIL,count,ExpFAIL,num,BError,size,k
double precision :: alf,TOL,sroot,Tsize,l
integer, dimension(5) :: buffer
integer, dimension(0:5) :: m
double precision, dimension(5) :: mu
double precision, dimension(5) :: a,b
double precision, dimension(s,0:Nmax+1) :: C,W,Chat,TE

call inputdata(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)
alf = 0.7
FAIL = 0
ExpFAIL = 0
call check(l,mu,s,FAIL)
C = 0.0

if (FAIL == 1) print*,"Error: Some constraints do not hold"

do k = 1,5
  do num=buffer(k),Nmax+1
    C(k,num) = a(k)*(real(num-buffer(k))**1.0) +
b(k)*(real(num-buffer(k))**2.0)
  end do
end do

TE(:,0) = l/(alf+l)

do i = 1,s
  do n = 1,Nmax
    TE(i,n) = l/(alf + l + mu(i) - (mu(i)*TE(i,n-1)))
  end do
end do

!do a check here that this thing is indeed
working****************************************
do i = 1,s
  call quad_roots(r,i,sroot)
  print*,"sroot = ",sroot
  print*,"TE(",i,",",Nmax,") = ",TE(i,Nmax)
  if (TE(i,Nmax) > sroot-TOL .and. TE(i,Nmax) < sroot+TOL) then
    print*,"Expectation OKAY"
  else
    print*,"Expectation error"
  end if
end do

do i = 1,s
  do n = 0,Nmax-1
    if (TE(i,n) < TE(i,n+1)) ExpFAIL = 1
  end do
end do

if (ExpFAIL == 1) print*,"Expectation error: non-decreasing with n"

Chat(:,0) = 0.0

do n = 1,Nmax
  do i = 1,s

    Chat(i,n) = (alf*C(i,n) + mu(i)*Chat(i,n-1))/(alf + l + mu(i) -
(mu(i)*TE(i,n-1)))

  end do
```

```
end do

do n = 0,Nmax-1
  do i = 1,s

    W(i,n) = (TE(i,n+1)*(C(i,n+1) -
(Chat(i,n)/(1.0-TE(i,n)))))/(((1.0-TE(i,n+1))/(1.0-TE(i,n))) - TE(i,n+1))

  end do
end do

return
end subroutine

!---------------------------------------------------------------------------
----------------------

subroutine getarrivals(Nmax,actsize,IA,AA)
implicit none

integer :: size,k,count,Nmax,r,BError,actsize
integer, dimension(5) :: buffer,m
double precision :: x,Tsize,TOL,l
double precision, dimension(5) :: mu
double precision, dimension(5) :: a,b
double precision, dimension(500000) :: IA,AA

call inputdata(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

IA = 0.0
AA = 0.0

count = 0

do k = 1,10
  call random_number(x)
end do

10 count = count + 1
call random_number(x)
IA(count) = -1.0*log(x)/l
if (count == 1) then
  AA(count) = IA(count)
else
  AA(count) = AA(count-1) + IA(count)
end if

if (AA(count) < Tsize .and. count < size) goto 10

if (count >= size) print*,"ERROR: Need bigger matrices & to simulate more
values"

actsize = count

!open(unit=7,file="simdata2.dat")
!write(unit=7,fmt="(a)") "IA = "
!do k=1,5
!  write(unit=7,fmt="(50f12.6)") IA(k,1:500)
!end do
!print*," "

!write(unit=7,fmt="(a)") "AA = "
!do k=1,5
!  write(unit=7,fmt="(50f12.6)") AA(k,1:500)
!end do
!print*," "

!write(unit=7,fmt="(a)") "TL1 = "
!write(unit=7,fmt="(100i4)") TL1(:)
!
!write(unit=7,fmt="(a)") "TL2 = "
!write(unit=7,fmt="(100f12.6)") TL2(1:100)
!
!print*," "
!!print*,"csize  = ",csize
```

```
!print*,"actsize = ",actsize
!print*,"TLactsize = ",TLactsize
!close(unit=7)
!
return
end subroutine

!--------------------------------------------------------------------------
----------------------

subroutine indexcost(r,Nmax,actsize,IA,AA,W,WIcost)
implicit none

integer :: size,k,count,queueserve,smallevent,state,Nmax,r,num,BError, &
           & actsize,i,event,j
integer, dimension(5) :: buffer,m,n
double precision :: Tsize,Tservice,stable,in2stat,summu,l,number
double precision :: smallind,Tcost,WIcost,TOL
double precision, dimension(5) :: mu
double precision, dimension(5) :: a,b,lastevent,endserve,NEtime
double precision, dimension(500000) :: IA,AA
double precision, dimension(5,0:Nmax+1) :: C,W


call inputdata(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

!Tservice = 0.0
in2stat = Tsize*0.667

WIcost = 0.0
summu = 0.0
endserve = 99999999.99

!test to ensure that we have stable queues
stable = 0.0
do k = 1,5
  if (mu(k) < 99999.99) summu = summu + mu(k)
end do

stable = l/summu
if (stable >= 1.0) print*,"ERROR: UNSTABLE SYSTEM!!!"

C = 0.0

do k = 1,5
  do num=buffer(k),Nmax+1
    C(k,num) = a(k)*(real(num-buffer(k))**1.0) +
b(k)*(real(num-buffer(k))**2.0)
  end do
end do

Tcost = 0.0
lastevent = 0.0
NEtime = 0.0
n = 0

event = 1

smallind = 9999999999999.99

do k = 1,5
  if (smallind > W(k,n(k))  .and. W(k,n(k)) < 99999999999.99) then
    smallind = W(k,n(k))
    queueserve = k
  end if
end do

n(queueserve) = n(queueserve) + 1
NEtime(queueserve) = AA(1)
lastevent(queueserve) = AA(1)

call expservice(r,queueserve,Tservice)
endserve(queueserve) = AA(1) + Tservice
```

```
event = 2

20  state = 0
  do k = 1,5
    state = state + n(k)
  end do


!  open(unit=7,file="tempstore.dat")
!  write(unit=7,fmt="(i3)") MINLOC(endserve)
!  close(unit=7)

  !smallevent = minloc(endserve)

!  open(unit=7,file="tempstore.dat")
!  read(unit=7,fmt="(i3)") smallevent
!  close(unit=7)

  !smallevent = MINLOC(endserve)
  smallevent = 1
  number = endserve(1)
  do j = 1,5
    if(number>endserve(j)) then
      smallevent = j
      number = endserve(j)
    end if
  end do


  if (AA(event) < endserve(smallevent)) smallevent = 0

  smallind = 999999999999999999.99

  if (smallevent == 0) then
    do k = 1,5
      if (smallind > W(k,n(k)) .and. W(k,n(k)) < 99999999999.99) then
        smallind = W(k,n(k))
        queueserve = k
      end if
    end do

    if (n(queueserve) == 0) then
      call expservice(r,queueserve,Tservice)
      endserve(queueserve) = AA(event) + Tservice
    end if

    if (n(queueserve) < Nmax) n(queueserve) = n(queueserve) + 1
    !if (lastevent > 0.0) then
      NEtime(queueserve) = AA(event) - lastevent(queueserve)
    !else
    !  NEtime = 0.0
    !end if

    if (AA(event) > in2stat .and. lastevent(queueserve) < in2stat) NEtime =
AA(event) - in2stat
    if (AA(event) > in2stat) Tcost = Tcost +
NEtime(queueserve)*C(queueserve,n(queueserve)-1)

    lastevent(queueserve) = AA(event)

    event = event + 1
  end if

  do i=1,5
    if (smallevent == i) then
      if (n(i) > 0) n(i) = n(i) - 1
      NEtime(i) = endserve(i) - lastevent(i)

      if (endserve(i) > in2stat .and. lastevent(i) < in2stat) NEtime =
endserve(i) - in2stat
      if (endserve(i) > in2stat) Tcost = Tcost + NEtime(i)*C(i,n(i)+1)

      lastevent(i) = endserve(i)

      if (n(i) > 0) then
```

```
        call expservice(r,i,Tservice)
        endserve(i) = endserve(i) + Tservice
      else if (n(i) == 0) then
        endserve(i) = 9999999999.999
      end if
    end if

  end do


!  write(unit=7,fmt="(a,i5)") "event # = ",event
!  write(unit=7,fmt="(a,i5)") "queueserve class = ",queueserve
!  write(unit=7,fmt="(a,f12.6)") "endserve = ",endserve
!  write(unit=7,fmt="(a,f12.6)") "Tservice = ",Tservice
!  write(unit=7,fmt="(a,f12.6)") "NEtime = ",NEtime
!  write(unit=7,fmt="(a,6i5)") "state = ",n
!  write(unit=7,fmt="(a,f12.6)") "TL2 = ",TL2(event)
!  write(unit=7,fmt="(a,f20.3)") "Tcost = ",Tcost
!  write(unit=7,fmt="(a,f12.6)") "Tcost/size = ",Tcost/(Lastevent-in2stat)
!  write(unit=7,fmt="(a)") " "

if (event < actsize) goto 20
!write(unit=7,fmt="(a,f12.6)") "TTservice = ",TTservice
!write(unit=7,fmt="(a,6i6)") "numserved = ",numserved
!print*,"Tcost = ",Tcost
!print*,"Tsize = ",Tsize
!print*,"in2stat = ",in2stat
WIcost = Tcost/(Tsize-in2stat)
!print*,"INDEX: average costs = ",Acost
!print*,"stable = ",stable
!close(unit=7)

return
end subroutine

!---------------------------------------------------------------------------
----------------------

subroutine longestq(r,Nmax,actsize,IA,AA,LQcost)
implicit none

integer :: size,k,count,queueserve,smallevent,state,Nmax,r,num,BError, &
            & actsize,i,event,j
integer, dimension(5) :: buffer,m,n
double precision :: Tsize,Tservice,stable,in2stat,summu,l,number
double precision :: smallind,Tcost,LQcost,TOL
double precision, dimension(5) :: mu
double precision, dimension(5) :: a,b,lastevent,endserve,NEtime
double precision, dimension(500000) :: IA,AA
double precision, dimension(5,0:Nmax+1) :: C

call inputdata(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

Tservice = 0.0
in2stat = Tsize*0.667

LQcost = 0.0
summu = 0.0
endserve = 99999999.99

!test to ensure that we have stable queues
stable = 0.0
do k = 1,5
  if (mu(k) < 99999.99) summu = summu + mu(k)
end do

stable = l/summu
if (stable >= 1.0) print*,"ERROR: UNSTABLE SYSTEM!!!"

C = 0.0

do k = 1,5
  do num=buffer(k),Nmax+1
    C(k,num) = a(k)*(real(num-buffer(k))**1.0) +
b(k)*(real(num-buffer(k))**2.0)
```

```
  end do
end do

Tcost = 0.0
lastevent = 0.0
NEtime = 0.0
n = 0

event = 1

smallind = 9999999999999.99

do k = 1,5
  if (smallind > n(k) .and. mu(k) < 99999.99) then
    smallind = n(k)
    queueserve = k
  end if
end do

n(queueserve) = n(queueserve) + 1
NEtime(queueserve) = AA(1)
lastevent(queueserve) = AA(1)

call expservice(r,queueserve,Tservice)
endserve(queueserve) = AA(1) + Tservice

event = 2

30  state = 0
  do k = 1,5
    state = state + n(k)
  end do


!  open(unit=7,file="tempstore.dat")
!  write(unit=7,fmt="(i3)") MINLOC(endserve)
!  close(unit=7)

  !smallevent = minloc(endserve)

!  open(unit=7,file="tempstore.dat")
!  read(unit=7,fmt="(i3)") smallevent
!  close(unit=7)

  !smallevent = MINLOC(endserve)

  smallevent = 1
  number = endserve(1)
  do j = 1,5
    if(number>endserve(j)) then
      smallevent = j
      number = endserve(j)
    end if
  end do



  if (AA(event) < endserve(smallevent)) smallevent = 0

  smallind = 999999999999999999.99

  if (smallevent == 0) then
    do k = 1,5
      if (smallind > n(k) .and. mu(k) < 99999.99) then
        smallind = n(k)
        queueserve = k
      end if
    end do

    if (n(queueserve) == 0) then
      call expservice(r,queueserve,Tservice)
      endserve(queueserve) = AA(event) + Tservice
    end if

    if (n(queueserve) < Nmax) n(queueserve) = n(queueserve) + 1
```

```
    !if (lastevent > 0.0) then
      NEtime(queueserve) = AA(event) - lastevent(queueserve)
    !else
    !  NEtime = 0.0
    !end if

    if (AA(event) > in2stat .and. lastevent(queueserve) < in2stat) NEtime =
AA(event) - in2stat
    if (AA(event) > in2stat) Tcost = Tcost +
NEtime(queueserve)*C(queueserve,n(queueserve)-1)

    lastevent(queueserve) = AA(event)

    event = event + 1
  end if

  do i=1,5
    if (smallevent == i) then
      if (n(i) > 0) n(i) = n(i) - 1
      NEtime(i) = endserve(i) - lastevent(i)

      if (endserve(i) > in2stat .and. lastevent(i) < in2stat) NEtime =
endserve(i) - in2stat
      if (endserve(i) > in2stat) Tcost = Tcost + NEtime(i)*C(i,n(i)+1)

      lastevent(i) = endserve(i)

      if (n(i) > 0) then
        call expservice(r,i,Tservice)
        endserve(i) = endserve(i) + Tservice
      else if (n(i) == 0) then
        endserve(i) = 9999999999.999
      end if
    end if
  end do


!  write(unit=7,fmt="(a,i5)") "event # = ",event
!  write(unit=7,fmt="(a,i5)") "queueserve class = ",queueserve
!  write(unit=7,fmt="(a,f12.6)") "endserve = ",endserve
!  write(unit=7,fmt="(a,f12.6)") "Tservice = ",Tservice
!  write(unit=7,fmt="(a,f12.6)") "NEtime = ",NEtime
!  write(unit=7,fmt="(a,6i5)") "state = ",n
!  write(unit=7,fmt="(a,f12.6)") "TL2 = ",TL2(event)
!  write(unit=7,fmt="(a,f20.3)") "Tcost = ",Tcost
!  write(unit=7,fmt="(a,f12.6)") "Tcost/size = ",Tcost/(Lastevent-in2stat)
!  write(unit=7,fmt="(a)") " "

if (event < actsize) goto 30
!write(unit=7,fmt="(a,f12.6)") "TTservice = ",TTservice
!write(unit=7,fmt="(a,6i6)") "numserved = ",numserved
!print*,"Tcost = ",Tcost
!print*,"Tsize = ",Tsize
!print*,"in2stat = ",in2stat
LQcost = Tcost/(Tsize-in2stat)
!print*,"INDEX: average costs = ",Acost
!print*,"stable = ",stable
!close(unit=7)

return
end subroutine

!---------------------------------------------------------------------------
-----------------------

subroutine myopic(r,Nmax,actsize,IA,AA,MYcost)
implicit none

integer :: size,k,count,queueserve,smallevent,state,Nmax,r,num,BError, &
            & actsize,i,event,j
integer, dimension(5) :: buffer,m,n
double precision :: Tsize,Tservice,stable,in2stat,summu,l,number
double precision :: smallind,Tcost,MYcost,TOL
double precision, dimension(5) :: mu
double precision, dimension(5) :: a,b,lastevent,endserve,NEtime
```

```
double precision, dimension(500000) :: IA,AA
double precision, dimension(5,0:Nmax+1) :: C

call inputdata(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

Tservice = 0.0
in2stat = Tsize*0.667

MYcost = 0.0
summu = 0.0
endserve = 99999999.99

!test to ensure that we have stable queues
stable = 0.0
do k = 1,5
  if (mu(k) < 99999.99) summu = summu + mu(k)
end do

stable = l/summu
if (stable >= 1.0) print*,"ERROR: UNSTABLE SYSTEM!!!"

C = 0.0

do k = 1,5
  do num=buffer(k),Nmax+1
    C(k,num) = a(k)*(real(num-buffer(k))**1.0) +
b(k)*(real(num-buffer(k))**2.0)
  end do
end do

Tcost = 0.0
lastevent = 0.0
NEtime = 0.0
n = 0

event = 1

smallind = 9999999999999.99

do k = 1,5
  if (smallind > C(k,n(k)) .and. mu(k) < 99999.99) then
    smallind = C(k,n(k))
    queueserve = k
  end if
end do

n(queueserve) = n(queueserve) + 1
NEtime(queueserve) = AA(1)
lastevent(queueserve) = AA(1)

call expservice(r,queueserve,Tservice)
endserve(queueserve) = AA(1) + Tservice

event = 2

40  state = 0
  do k = 1,5
    state = state + n(k)
  end do


!  open(unit=7,file="tempstore.dat")
!  write(unit=7,fmt="(i3)") MINLOC(endserve)
!  close(unit=7)

  !smallevent = minloc(endserve)

!  open(unit=7,file="tempstore.dat")
!  read(unit=7,fmt="(i3)") smallevent
!  close(unit=7)

  !smallevent = MINLOC(endserve)

  smallevent = 1
  number = endserve(1)
```

```
  do j = 1,5
    if(number>endserve(j)) then
      smallevent = j
      number = endserve(j)
    end if
  end do



  if (AA(event) < endserve(smallevent)) smallevent = 0

  smallind = 9999999999999999.99

  if (smallevent == 0) then
    do k = 1,5
      if (smallind > C(k,n(k)) .and. mu(k) < 99999.99) then
        smallind = C(k,n(k))
        queueserve = k
      end if
    end do

    if (n(queueserve) == 0) then
      call expservice(r,queueserve,Tservice)
      endserve(queueserve) = AA(event) + Tservice
    end if

    if (n(queueserve) < Nmax) n(queueserve) = n(queueserve) + 1
    !if (lastevent > 0.0) then
      NEtime(queueserve) = AA(event) - lastevent(queueserve)
    !else
    !  NEtime = 0.0
    !end if

    if (AA(event) > in2stat .and. lastevent(queueserve) < in2stat) NEtime =
AA(event) - in2stat
    if (AA(event) > in2stat) Tcost = Tcost +
NEtime(queueserve)*C(queueserve,n(queueserve)-1)

    lastevent(queueserve) = AA(event)

    event = event + 1
  end if

  do i=1,5
    if (smallevent == i) then
      if (n(i) > 0) n(i) = n(i) - 1
      NEtime(i) = endserve(i) - lastevent(i)

      if (endserve(i) > in2stat .and. lastevent(i) < in2stat) NEtime =
endserve(i) - in2stat
      if (endserve(i) > in2stat) Tcost = Tcost + NEtime(i)*C(i,n(i)+1)

      lastevent(i) = endserve(i)

      if (n(i) > 0) then
        call expservice(r,i,Tservice)
        endserve(i) = endserve(i) + Tservice
      else if (n(i) == 0) then
        endserve(i) = 9999999999.999
      end if
    end if
  end do


!  write(unit=7,fmt="(a,i5)") "event # = ",event
!  write(unit=7,fmt="(a,i5)") "queueserve class = ",queueserve
!  write(unit=7,fmt="(a,f12.6)") "endserve = ",endserve
!  write(unit=7,fmt="(a,f12.6)") "Tservice = ",Tservice
!  write(unit=7,fmt="(a,f12.6)") "NEtime = ",NEtime
!  write(unit=7,fmt="(a,6i5)") "state = ",n
!  write(unit=7,fmt="(a,f12.6)") "TL2 = ",TL2(event)
!  write(unit=7,fmt="(a,f20.3)") "Tcost = ",Tcost
!  write(unit=7,fmt="(a,f12.6)") "Tcost/size = ",Tcost/(Lastevent-in2stat)
!  write(unit=7,fmt="(a)") " "
```

```
if (event < actsize) goto 40
!write(unit=7,fmt="(a,f12.6)") "TTservice = ",TTservice
!write(unit=7,fmt="(a,6i6)") "numserved = ",numserved
!print*,"Tcost = ",Tcost
!print*,"Tsize = ",Tsize
!print*,"in2stat = ",in2stat
MYcost = Tcost/(Tsize-in2stat)
!print*,"INDEX: average costs = ",Acost
!print*,"stable = ",stable
!close(unit=7)

return
end subroutine

!---------------------------------------------------------------------------
----------------------

subroutine static(r,Nmax,actsize,IA,AA,STcost,stationary2)
implicit none

integer :: size,k,count,queueserve,smallevent,state,Nmax,r,num,BError, &
           & actsize,i,event,j
integer, dimension(5) :: buffer,m,n
double precision :: Tsize,Tservice,stable,in2stat,summu,l,number
double precision :: Tcost,STcost,TOL,x,statsum2
double precision, dimension(5) :: mu,stationary2
double precision, dimension(0:5) :: stationary
double precision, dimension(5) :: a,b,lastevent,endserve,NEtime
double precision, dimension(500000) :: IA,AA
double precision, dimension(5,0:Nmax+1) :: C

call inputdata(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

Tservice = 0.0
in2stat = Tsize*0.667

STcost = 0.0
summu = 0.0
endserve = 99999999.99

stationary = 0.0

!find a stationary distribution that will not be unstable
do i = 1,5
  stationary2(i) = mu(i)/l
end do

statsum2 = 0.0
do i = 1,5
  statsum2 = statsum2 + stationary2(i)
end do

do i = 1,5
  stationary2(i) = stationary2(i)/statsum2
end do

do i = 1,5
  stationary(i) = stationary(i-1) + stationary2(i)
end do

!test to ensure that we have stable queues
stable = 0.0
do k = 1,5
  if (mu(k) < 99999.99) summu = summu + mu(k)
end do

stable = l/summu
if (stable >= 1.0) print*,"ERROR: UNSTABLE SYSTEM!!!"

do k = 1,5
  if (l*stationary2(k) > mu(k)) print*,"ERROR: UNSTABLE STATIC SYSTEM!!!"
end do

C = 0.0
```

```
do k = 1,5
  do num=buffer(k),Nmax+1
    C(k,num) = a(k)*(real(num-buffer(k))**1.0) +
b(k)*(real(num-buffer(k))**2.0)
  end do
end do

Tcost = 0.0
lastevent = 0.0
NEtime = 0.0
n = 0

event = 1

call random_number(x)

if (x < stationary(1)) then
  queueserve = 1
else if (x < stationary(2)) then
  queueserve = 2
else if (x < stationary(3)) then
  queueserve = 3
else if (x < stationary(4)) then
  queueserve = 4
else if (x < stationary(5)) then
  queueserve = 5
end if

n(queueserve) = n(queueserve) + 1
NEtime(queueserve) = AA(1)
lastevent(queueserve) = AA(1)

call expservice(r,queueserve,Tservice)
endserve(queueserve) = AA(1) + Tservice

event = 2

50  state = 0
  do k = 1,5
    state = state + n(k)
  end do

!  open(unit=7,file="tempstore.dat")
!  write(unit=7,fmt="(i3)") MINLOC(endserve)
!  close(unit=7)

  !smallevent = minloc(endserve)

!  open(unit=7,file="tempstore.dat")
!  read(unit=7,fmt="(i3)") smallevent
!  close(unit=7)

  !smallevent = MINLOC(endserve)

  smallevent = 1
  number = endserve(1)
  do j = 1,5
    if(number>endserve(j)) then
      smallevent = j
      number = endserve(j)
    end if
  end do



  if (AA(event) < endserve(smallevent)) smallevent = 0

  if (smallevent == 0) then
    call random_number(x)

    if (x < stationary(1)) then
      queueserve = 1
    else if (x < stationary(2)) then
      queueserve = 2
    else if (x < stationary(3)) then
```

```
      queueserve = 3
    else if (x < stationary(4)) then
      queueserve = 4
    else if (x < stationary(5)) then
      queueserve = 5
    end if

    if (n(queueserve) == 0) then
      call expservice(r,queueserve,Tservice)
      endserve(queueserve) = AA(event) + Tservice
    end if

    if (n(queueserve) < Nmax) n(queueserve) = n(queueserve) + 1
    !if (lastevent > 0.0) then
      NEtime(queueserve) = AA(event) - lastevent(queueserve)
    !else
    !  NEtime = 0.0
    !end if

    if (AA(event) > in2stat .and. lastevent(queueserve) < in2stat) NEtime =
AA(event) - in2stat
    if (AA(event) > in2stat) Tcost = Tcost +
NEtime(queueserve)*C(queueserve,n(queueserve)-1)

    lastevent(queueserve) = AA(event)

    event = event + 1
  end if

  do i=1,5
    if (smallevent == i) then
      if (n(i) > 0) n(i) = n(i) - 1
      NEtime(i) = endserve(i) - lastevent(i)

      if (endserve(i) > in2stat .and. lastevent(i) < in2stat) NEtime =
endserve(i) - in2stat
      if (endserve(i) > in2stat) Tcost = Tcost + NEtime(i)*C(i,n(i)+1)

      lastevent(i) = endserve(i)

      if (n(i) > 0) then
        call expservice(r,i,Tservice)
        endserve(i) = endserve(i) + Tservice
      else if (n(i) == 0) then
        endserve(i) = 9999999999.999
      end if
    end if
  end do


!  write(unit=7,fmt="(a,i5)") "event # = ",event
!  write(unit=7,fmt="(a,i5)") "queueserve class = ",queueserve
!  write(unit=7,fmt="(a,f12.6)") "endserve = ",endserve
!  write(unit=7,fmt="(a,f12.6)") "Tservice = ",Tservice
!  write(unit=7,fmt="(a,f12.6)") "NEtime = ",NEtime
!  write(unit=7,fmt="(a,6i5)") "state = ",n
!  write(unit=7,fmt="(a,f12.6)") "TL2 = ",TL2(event)
!  write(unit=7,fmt="(a,f20.3)") "Tcost = ",Tcost
!  write(unit=7,fmt="(a,f12.6)") "Tcost/size = ",Tcost/(Lastevent-in2stat)
!  write(unit=7,fmt="(a)") " "

if (event < actsize) goto 50
!write(unit=7,fmt="(a,f12.6)") "TTservice = ",TTservice
!write(unit=7,fmt="(a,6i6)") "numserved = ",numserved
!print*,"Tcost = ",Tcost
!print*,"Tsize = ",Tsize
!print*,"in2stat = ",in2stat
STcost = Tcost/(Tsize-in2stat)
!print*,"INDEX: average costs = ",Acost
!print*,"stable = ",stable
!close(unit=7)

return
end subroutine
```

```
!---------------------------------------------------------------------------
----------------------

subroutine expservice(r,queueserve,Tservice)
implicit none

integer :: Nmax,r,BError,size,queueserve
integer, dimension(5) :: buffer
integer, dimension(5) :: m
double precision, dimension(5) :: a,b
double precision, dimension(5) :: mu
double precision :: TOL,Tsize,Tservice,x,l

a = 0.0
b = 0.0
l = 0.0
mu = 0.0
m = 1
Nmax = 0
buffer = 0
BError = 0
TOL = 0.0
size = 0
Tsize = 0.0

call inputdata(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

!Tservice = 0.0
call random_number(x)
Tservice = (-1.0*log(x)/mu(queueserve))

return
end subroutine

!---------------------------------------------------------------------------
-

subroutine quad_roots(r,i,sroot)
implicit none

integer :: Nmax,r,BError,size,i
integer, dimension(5) :: buffer
integer, dimension(5) :: m
double precision, dimension(5) :: a,b
double precision, dimension(5) :: mu
double precision :: TOL,Tsize,alf,l,root1,root2,sroot,a1,a2,a3

call inputdata(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

alf = 0.7

a1 = mu(i)
a2 = alf + l + mu(i)
a3 = l

root1 = (a2 + sqrt((a2**2) - (4*a1*a3)))/(2*a1)
root2 = (a2 - sqrt((a2**2) - (4*a1*a3)))/(2*a1)

if (root1 > root2) then
  sroot = root2
else
  sroot = root1
end if

return
end subroutine

!---------------------------------------------------------------------------
-

subroutine inputdata(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)
implicit none

integer :: Nmax,r,BError,size,i,j
integer, dimension(5) :: buffer
```

```
integer, dimension(5) :: m,mold
double precision, dimension(5) :: a,b
double precision, dimension(5) :: mu,muold
double precision :: TOL,Tsize,l,summu,rho,lold


rho  = 0.0
summu = 0.0

Nmax = 139
BError = 5

size = 500000
Tsize = 40000.0

l =    8.525 !first
muold =  (/0.6,1.5,2.7,3.9,5.0/)!(/0.2,0.9,1.7,2.5,3.2/) ! first
m = (/1,1,1,1,1/) ! first

do i = 1,5
  if(muold(i) < 999999.9) summu = summu + muold(i)
end do

rho = l/summu
l = (l/rho)*0.85


if (r == 0) then
  buffer = 0
  mu = muold
else if (r == 1) then
  buffer = 0
  mu(1) = muold(2)
  mu(2) = muold(3)
  mu(3) = muold(4)
  mu(4) = muold(5)
  mu(5) = muold(1)
else if (r == 2) then
  buffer = 0
  mu(1) = muold(3)
  mu(2) = muold(4)
  mu(3) = muold(5)
  mu(4) = muold(1)
  mu(5) = muold(2)
else if (r == 3) then
  buffer = 0
  mu(1) = muold(4)
  mu(2) = muold(5)
  mu(3) = muold(1)
  mu(4) = muold(2)
  mu(5) = muold(3)
else if (r == 4) then
  buffer = 0
  mu(1) = muold(5)
  mu(2) = muold(1)
  mu(3) = muold(2)
  mu(4) = muold(3)
  mu(5) = muold(4)
else if (r == 5) then
  buffer = 0
  rho = l/summu
  l = (l/rho)*0.85
  mu = muold
else if (r == 6) then
  buffer = 0
  rho = l/summu
  l = (l/rho)*0.85
  mu(1) = muold(2)
  mu(2) = muold(3)
  mu(3) = muold(4)
  mu(4) = muold(5)
  mu(5) = muold(1)
else if (r == 7) then
  buffer = 0
  rho = l/summu
```

```
   l = (l/rho)*0.85
   mu(1) = muold(3)
   mu(2) = muold(4)
   mu(3) = muold(5)
   mu(4) = muold(1)
   mu(5) = muold(2)
else if (r == 8) then
   buffer = 0
   rho = l/summu
   l = (l/rho)*0.85
   mu(1) = muold(4)
   mu(2) = muold(5)
   mu(3) = muold(1)
   mu(4) = muold(2)
   mu(5) = muold(3)
else if (r == 9) then
   buffer = 0
   rho = l/summu
   l = (l/rho)*0.85
   mu(1) = muold(5)
   mu(2) = muold(1)
   mu(3) = muold(2)
   mu(4) = muold(3)
   mu(5) = muold(4)
else if (r == 10) then
   buffer = 2
   mu = muold
else if (r == 11) then
   buffer = 2
   mu(1) = muold(2)
   mu(2) = muold(3)
   mu(3) = muold(4)
   mu(4) = muold(5)
   mu(5) = muold(1)
else if (r == 12) then
   buffer = 2
   mu(1) = muold(3)
   mu(2) = muold(4)
   mu(3) = muold(5)
   mu(4) = muold(1)
   mu(5) = muold(2)
else if (r == 13) then
   buffer = 2
   mu(1) = muold(4)
   mu(2) = muold(5)
   mu(3) = muold(1)
   mu(4) = muold(2)
   mu(5) = muold(3)
else if (r == 14) then
   buffer = 2
   mu(1) = muold(5)
   mu(2) = muold(1)
   mu(3) = muold(2)
   mu(4) = muold(3)
   mu(5) = muold(4)
else if (r == 15) then
   buffer = 2
   rho = l/summu
   l = (l/rho)*0.85
   mu = muold
else if (r == 16) then
   buffer = 2
   rho = l/summu
   l = (l/rho)*0.85
   mu(1) = muold(2)
   mu(2) = muold(3)
   mu(3) = muold(4)
   mu(4) = muold(5)
   mu(5) = muold(1)
else if (r == 17) then
   buffer = 2
   rho = l/summu
   l = (l/rho)*0.85
   mu(1) = muold(3)
   mu(2) = muold(4)
```

```
  mu(3) = muold(5)
  mu(4) = muold(1)
  mu(5) = muold(2)
else if (r == 18) then
  buffer = 2
  rho = l/summu
  l = (l/rho)*0.85
  mu(1) = muold(4)
  mu(2) = muold(5)
  mu(3) = muold(1)
  mu(4) = muold(2)
  mu(5) = muold(3)
else if (r == 19) then
  buffer = 2
  rho = l/summu
  l = (l/rho)*0.85
  mu(1) = muold(5)
  mu(2) = muold(1)
  mu(3) = muold(2)
  mu(4) = muold(3)
  mu(5) = muold(4)
else if (r == 20) then
  buffer = 4
  mu = muold
else if (r == 21) then
  buffer = 4
  mu(1) = muold(2)
  mu(2) = muold(3)
  mu(3) = muold(4)
  mu(4) = muold(5)
  mu(5) = muold(1)
else if (r == 22) then
  buffer = 4
  mu(1) = muold(3)
  mu(2) = muold(4)
  mu(3) = muold(5)
  mu(4) = muold(1)
  mu(5) = muold(2)
else if (r == 23) then
  buffer = 4
  mu(1) = muold(4)
  mu(2) = muold(5)
  mu(3) = muold(1)
  mu(4) = muold(2)
  mu(5) = muold(3)
else if (r == 24) then
  buffer = 4
  mu(1) = muold(5)
  mu(2) = muold(1)
  mu(3) = muold(2)
  mu(4) = muold(3)
  mu(5) = muold(4)
else if (r == 25) then
  buffer = 4
  rho = l/summu
  l = (l/rho)*0.85
  mu = muold
else if (r == 26) then
  buffer = 4
  rho = l/summu
  l = (l/rho)*0.85
  mu(1) = muold(2)
  mu(2) = muold(3)
  mu(3) = muold(4)
  mu(4) = muold(5)
  mu(5) = muold(1)
else if (r == 27) then
  buffer = 4
  rho = l/summu
  l = (l/rho)*0.85
  mu(1) = muold(3)
  mu(2) = muold(4)
  mu(3) = muold(5)
  mu(4) = muold(1)
  mu(5) = muold(2)
```

```
else if (r == 28) then
  buffer = 4
  rho = l/summu
  l = (l/rho)*0.85
  mu(1) = muold(4)
  mu(2) = muold(5)
  mu(3) = muold(1)
  mu(4) = muold(2)
  mu(5) = muold(3)
else if (r == 29) then
  buffer = 4
  rho = l/summu
  l = (l/rho)*0.85
  mu(1) = muold(5)
  mu(2) = muold(1)
  mu(3) = muold(2)
  mu(4) = muold(3)
  mu(5) = muold(4)
end if


!do i = 1,5
!  j = mod(i + r,5)
!  if (j == 0) j = 5
!  muold(j) = mu(i)
!  mold(j) = m(i)
!end do
!lold = l
!l = lold
!mu = muold
!m = mold


!a = (/1.1,0.6,0.0,0.0,0.0/)
!b = (/0.5,1.0,0.0,0.0,0.0/)

a = (/1.5,1.2,0.9,0.6,0.3/)
b = (/0.20,0.40,0.60,0.80,1.0/)

TOL = 0.0001

return
end subroutine

!-----------------------------------------------------------------------
----------------------

!subroutine whichqueue(W,n,queueserve)
!implicit none
!
!!integer, dimension() :: n
!double precision :: queueserve
!double precision, dimension :: W
!
!smallind = 999999.99
!do k = 1,5
!  if (smallind > W(k,n(k))) then
!     smallind = W(k,n(k))
!     queueserve = k
!  end if
!end do
!
!n(queueserve) = n(queueserve) + 1
!lastevent = TL2(1)
!
!
!
!return
!end subroutine
!
!-----------------------------------------------------------------------
----------------------
```

Appendix D

This appendix contains the Fortran 95 code for the programme we used to calculate the discounted service control costs as in Section 3.5.1. Here we consider the optimal and index policies for a 2 class system.

```
program generalS_wittle
implicit none

integer :: nmax,i,j,k,g,BError,r
integer, dimension(2) :: m,buffer
double precision :: alf
double precision, dimension(2) :: l,mu,a,b
double precision, dimension (200,8) :: indata
double precision, allocatable, dimension(:,:,:,:) ::
WIcost,OPTcost,STATcost,LONGQcost,subopt

indata = 0.0
open(unit=7,file="GSinputdata.dat")
do i  = 1,8
  read(unit=7,fmt="(8f10.4)") indata(i,:)
end do
close(unit=7)
r=1

!get the system parameters
call inputdata(indata,r,l,mu,m,alf,nmax,buffer,a,b,BError)

allocate( WIcost(0:Nmax,0:m(1),0:Nmax,0:m(2)) )
allocate( OPTcost(0:Nmax,0:m(1),0:Nmax,0:m(2)) )
allocate( STATcost(0:Nmax,0:m(1),0:Nmax,0:m(2)) )
allocate( LONGQcost(0:Nmax,0:m(1),0:Nmax,0:m(2)) )
allocate( subopt(0:Nmax,0:m(1),0:Nmax,0:m(2)) )

do r = 1,50

!get the system parameters
call inputdata(indata,r,l,mu,m,alf,nmax,buffer,a,b,BError)

WIcost = 0.0
OPTcost = 0.0
STATcost = 0.0
LONGQcost = 0.0
subopt = 0.0

call Whittle_Costs3(indata,r,m,Nmax,WIcost)
call Optimal_Costs3(indata,r,m,Nmax,OPTcost)
!call Static_Costs3(indata,r,m,Nmax,STATcost)
!call LongestQ_Costs3(indata,r,m,Nmax,LONGQcost)
do i = 0,Nmax
  do j = 0,Nmax
    do k=0,m(1)
      do g = 0,m(2)
        if (OPTcost(i,j,k,g) > 0.00001) subopt(i,j,k,g) =
100*(WIcost(i,j,k,g)-OPTcost(i,j,k,g))/OPTcost(i,j,k,g)
      end do
    end do
  end do
end do

print*,"this is number ",r
open(unit=7,file="GSsubquart139buff0.dat")
write(unit=7,fmt="(a)") "    b1    :    c1    :    b2    :    c2    : Nmax"
write(unit=7,fmt="(4f10.4,i5)") a(1),b(1),a(2),b(2),Nmax
write(unit=7,fmt="(a)") "    l1    :    l2    :    m1 : mu1    :    m2 : mu2    :
  alpha"
write(unit=7,fmt="(2f10.4,i5,f10.4,i5,2f10.4)")
l(1),l(2),m(1),mu(1),m(2),mu(2),alf
write(unit=7,fmt="(a)") " "

write(unit=7,fmt="(a)") "-------------Whittle-------------"

!do k=0,m(1)
 ! do g = 0,m(2)
    if(WIcost(4,0,4,0) > 0.000001) then
      write(unit=7,fmt="(a)") " "
      write(unit=7,fmt="(a,i4,a,i4,a)") "WIcost(0:4,",0,",0:4,",0,") = "
      do i = 0,4
        write(unit=7,fmt="(a,i4,a,f12.4,a,f12.4,a,f12.4,a,f12.4,a,f12.4,a)")
&
        & "&",i," &",WIcost(i,0,0,0)," &",WIcost(i,0,1,0),"
```

```
&",WIcost(i,0,2,0)," &",WIcost(i,0,3,0)," &",WIcost(i,0,4,0)," \\"
        end do
      end if
   ! end do
!end do

write(unit=7,fmt="(a)") "-------------Optimal-------------"

do k=0,0 !m(1)
  do g = 0,0 !m(2)
    if(OPTcost(4,k,4,g) > 0.000001) then
      write(unit=7,fmt="(a)") " "
      write(unit=7,fmt="(a,i4,a,i4,a)") "OPTcost(0:4,",k,",0:4,",g,") = "
      do i = 0,4
        write(unit=7,fmt="(a,i4,a,f12.4,a,f12.4,a,f12.4,a,f12.4,a,f12.4,a)")
&
        & "&",i," &",OPTcost(i,0,0,0)," &",OPTcost(i,0,1,0),"
&",OPTcost(i,0,2,0)," &",OPTcost(i,0,3,0)," &",OPTcost(i,0,4,0)," \\"
                !write(unit=7,fmt="(5f12.6)") OPTcost(i,k,0:4,g)
      end do
    end if
  end do
end do

write(unit=7,fmt="(a)") "-------------Suboptimallity-------------"

do k=0,0!m(1)
  do g = 0,0!m(2)
    if(WIcost(4,k,4,g) > 0.000001) then
      write(unit=7,fmt="(a)") " "
      write(unit=7,fmt="(a,i4,a,i4,a)") "subopt(0:4,",k,",0:4,",g,") = "
      do i = 0,4
        write(unit=7,fmt="(a,i4,a,f12.4,a,f12.4,a,f12.4,a,f12.4,a,f12.4,a)")
&
        & "&",i," &",subopt(i,0,0,0)," &",subopt(i,0,1,0),"
&",subopt(i,0,2,0)," &",subopt(i,0,3,0)," &",subopt(i,0,4,0)," \\"
        !write(unit=7,fmt="(5f12.6)") subopt(i,k,0:4,g)
      end do
    end if
  end do
end do


!do i = 0,4
!  do j = 0,4
!    do k=0,m(1)
!      do g = 0,m(2)
!        if(WIcost(i,k,j,g)>0.00001) write(unit=7,fmt="(a,4i4,a,f12.6)")
"WIcost(",i,k,j,g,") = ",WIcost(i,k,j,g)
!      end do
!    end do
!  end do
!end do

!print*,"this is number 2!"

!write(unit=7,fmt="(a)") "-------------Optimal-------------"

!do i = 0,4
!  do j = 0,4
!    do k=0,m(1)
!      do g = 0,m(2)
!        if(OPTcost(i,k,j,g)>0.00001) write(unit=7,fmt="(a,4i4,a,f12.6)")
"OPTcost(",i,k,j,g,") = ",OPTcost(i,k,j,g)
!      end do
!    end do
!  end do
!end do

!write(unit=7,fmt="(a)") "-------------Suboptimallity-------------"
!do i = 0,4
!  do j = 0,4
!    do k=0,m(1)
!      do g = 0,m(2)
!        if(WIcost(i,k,j,g)>0.00001 .or. OPTcost(i,k,j,g)>0.00001) &
```

```
!                       & write(unit=7,fmt="(a,4i4,a,f12.6)")
"subopt(",i,k,j,g,") = ",subopt(i,k,j,g)
!       end do
!    end do
!  end do
!end do

!print*,"this is number 3!"

!write(unit=7,fmt="(a)") "-------------Static-------------"

!do i = 0,4
!  do j = 0,4
!    do k=0,m(1)
!     do g = 0,m(2)
!!        if(STATcost(i,k,j,g)>0.00001) write(unit=7,fmt="(a,4i4,a,f12.6)")
"STATcost(",i,k,j,g,") = ",STATcost(i,k,j,g)
!       end do
!    end do
!  end do
!end do
!
!write(unit=7,fmt="(a)") "-------------Longest Q-------------"
!
!do i = 0,4
!  do j = 0,4
!    do k=0,m(1)
!       do g = 0,m(2)
!!          if(LONGQcost(i,k,j,g)>0.00001) write(unit=7,fmt="(a,4i4,a,f12.6)")
"LONGQcost(",i,k,j,g,") = ",LONGQcost(i,k,j,g)
!       end do
!    end do
!  end do
!end do
!
!print*,"WIcost(0,0,0,0) = ",WIcost(0,0,0,0)

end do

close(unit=7)

end program

!---------------------------------------------------------------------------
----------

!NB here we have a state space of (n1,m1,n2,m2) where m1 is the
!number of phase completions we have done for class 1, i.e, m1
!starts off at 0 and goes upto m(1)-1, as when the m(1)th phase
!completion is over n1 goes to n1-1 and m1 goes back to 0 as this
!is the start of the service of the next queuing customer. (similarly for
m2)

!this subroutine only looks at real possible events occuring, i.e. there
!are no virtual events (except when n=boundery case).

!Now we have an extra state, m=1 is where we have started a service
!but have not finished the first phase of that service

subroutine Whittle_Costs3(indata,r,m,Nmax,Vnew)
implicit none

integer ::
num,nmax,count,n1,n2,m1,m2,BError,k,mumb1,mumb2,num1,num2,numb1,numb2,r
integer, dimension (2) :: m,buffer
integer, dimension (4) :: Sele,Lele
double precision :: alf,U,largest,smallest,diff,TOL
double precision, dimension(2) :: l,mu,a,b
double precision, dimension(2,0:nmax) :: W,C
double precision, dimension (200,8) :: indata
double precision, dimension(0:Nmax,0:m(1),0:Nmax,0:m(2)) :: Vold,Vnew

call inputdata(indata,r,l,mu,m,alf,nmax,buffer,a,b,BError)

TOL = 0.00001 ! 0.00001 !0.005
```

```
C = 0.0

do k = 1,2
  do num = buffer(k),Nmax
    C(k,num) = a(k)*(real(num-buffer(k))**3.0) +
b(k)*(real(num-buffer(k))**4.0)
  end do
end do

call WIndex(indata,r,m,Nmax,W)

!BError = 1

Vold = 0.0
Vnew = 0.0
count = 0

30 Vold = Vnew

count = count + 1

do n1 = 0,Nmax
  do n2 = 0,Nmax
    do m1 = 0,m(1)
      do m2 = 0,m(2)

        call arrnext(1,Nmax,n1,n2,num1,num2)
        call arrnext(2,Nmax,n1,n2,num1,num2)
        call sernext(1,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)
        call sernext(2,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)

        if(n1>0 .and. n2>0) then

          if(m1==0 .and. m2==0) then

            !this calculates which queue we are serving if we have the
            !choice & there are customers present
            if (W(1,n1) >= W(2,n2)) then
              U = l(1) + l(2) + mu(1) + alf
              Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,1,n2,0) &
                                          & +  (l(2)/U)*Vold(n1,1,num2,0) +
(mu(1)/U)*Vold(numb1,mumb1,n2,0)
            else
              U = l(1) + l(2) + mu(2) + alf
              Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,0,n2,1) &
                                          & +  (l(2)/U)*Vold(n1,0,num2,1) +
(mu(2)/U)*Vold(n1,0,numb2,mumb2)
            end if

          else if(m1>0 .and. m2==0) then
            U = l(1) + l(2) + mu(1) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                          & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(1)/U)*Vold(numb1,mumb1,n2,0)

          else if(m1==0 .and. m2>0) then
              U = l(1) + l(2) + mu(2) + alf
              Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                          & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(2)/U)*Vold(n1,0,numb2,mumb2)

          end if

        else if(n1>0 .and. n2==0) then
          if(m1==0 .and. m2==0) then
            U = l(1) + l(2) + mu(1) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,1,n2,0) &
                                          & +  (l(2)/U)*Vold(n1,1,num2,0) +
(mu(1)/U)*Vold(numb1,mumb1,n2,0)
          else if(m1>0 .and. m2==0) then
```

```
            U = l(1) + l(2) + mu(1) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                        & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(1)/U)*Vold(numb1,mumb1,n2,m2)
          end if

        else if(n1==0 .and. n2>0) then
          if(m1==0 .and. m2==0) then
            U = l(1) + l(2) + mu(2) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,0,n2,1) &
                                        & +  (l(2)/U)*Vold(n1,0,num2,1) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)
          else if(m1==0 .and. m2>0) then
            U = l(1) + l(2) + mu(2) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                        & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)
          end if
        else if(n1==0 .and. n2==0) then
          if(m1==0 .and. m2==0) then
            U = l(1) + l(2) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                        & +  (l(2)/U)*Vold(n1,m1,num2,m2)
          end if
        end if

      end do
    end do
  end do
end do

!open (unit=7,file="VvaluesD1.dat")
!write(unit=7,fmt="(3f16.4)") Vnew(39,0,39,0)

smallest = 1000000.0
largest = -1000000.0
Sele = 999
Lele = 999

!do n1 = 0,Nmax-BError
!  do n2 = 0,Nmax-BError
!    do m1 = 0,m(1)
!      do m2 = 0,m(2)
!
!        if(n1>0 .and. n2>0) then
!          if(m1==0 .and. m2==0) then
!
!            if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!              smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!              Sele = (/n1,m1,n2,m2/)
!            end if
!            if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!              largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!              Lele = (/n1,m1,n2,m2/)
!            end if
!
!          else if(m1>0 .and. m2==0) then
!            if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!            smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!            Sele = (/n1,m1,n2,m2/)
!            end if
!            if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!              largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!              Lele = (/n1,m1,n2,m2/)
!            end if
!          else if(m1==0 .and. m2>0) then
!            if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!              smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!              Sele = (/n1,m1,n2,m2/)
!            end if
!            if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
```

```
!               largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!               Lele = (/n1,m1,n2,m2/)
!             end if
!
!           end if
!
!         else if(n1>0 .and. n2==0) then
!           if(m1==0 .and. m2==0) then
!             if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!               smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!               Sele = (/n1,m1,n2,m2/)
!             end if
!             if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!               largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!               Lele = (/n1,m1,n2,m2/)
!             end if
!
!           else if(m1>0 .and. m2==0) then
!             if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!               smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!               Sele = (/n1,m1,n2,m2/)
!             end if
!             if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!               largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!               Lele = (/n1,m1,n2,m2/)
!             end if
!           end if
!
!         else if(n1==0 .and. n2>0) then
!           if(m1==0 .and. m2==0) then
!             if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!               smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!               Sele = (/n1,m1,n2,m2/)
!             end if
!             if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!               largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!               Lele = (/n1,m1,n2,m2/)
!             end if
!           else if(m1==0 .and. m2>0) then
!             if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!               smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!               Sele = (/n1,m1,n2,m2/)
!             end if
!             if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!               largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!               Lele = (/n1,m1,n2,m2/)
!             end if
!           end if
!
!         else if(n1==0 .and. n2==0) then
!           if(m1==0 .and. m2==0) then
!             if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!               smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!               Sele = (/n1,m1,n2,m2/)
!             end if
!             if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
!               largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
!               Lele = (/n1,m1,n2,m2/)
!             end if
!           end if
!
!         end if
!
!       end do
!     end do
!   end do
!end do
!
!

do n1 = 1,Nmax-BError
  do n2 = 0,Nmax-BError
    do m1 = 1,m(1)

     m2 = 0
```

```
        if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
       end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do

do n1 = 0,Nmax-BError
  do n2 = 1,Nmax-BError
    do m2 = 1,m(2)

        m1 = 0

        if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do

n1 = 0 !do n1 = 0,0
n2 = 0 !  do n2 = 0,0

    m2 = 0
    m1 = 0

    if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
      smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
      Sele = (/n1,m1,n2,m2/)
    end if
    if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
      largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
      Lele = (/n1,m1,n2,m2/)
    end if

!  end do
!end do


diff = largest - smallest

!open (unit=7,file="GSWdiff.dat")
!write(unit=7,fmt="(3f16.4,8i4)") smallest,largest,diff,Sele,Lele
!close(unit=7)
if (count > 9000) goto 300
if (largest > TOL .or. largest < 0.0) goto 30 !not have smallest*TOL as have
discounted costs
                                        !hence must converge (to 0?)

!print*,"4"

!100 WIcost = (smallest + largest)/2.0

!close(unit=7)
!close(unit=7)


!print*,"U = ",U
300 print*,"Count = ",count
print*,"The W index policy cost this queue setup & parameters is
",Vnew(0,0,0,0)
!close(unit=7)
```

```
return
end subroutine

!-----------------------------------------------------------------------------
----------

!NB here we have a state space of (n1,m1,n2,m2) where m1 is the
!number of phase completions we have done for class 1, i.e, m1
!starts off at 0 and goes upto m(1)-1, as when the m(1)th phase
!completion is over n1 goes to n1-1 and m1 goes back to 0 as this
!is the start of the service of the next queuing customer. (similarly for
m2)

!this subroutine only looks at real possible events occuring, i.e. there
!are no virtual events (except when n=boundery case).

!Now we have an extra state, m=1 is where we have started a service
!but have not finished the first phase of that service

subroutine Optimal_Costs3(indata,r,m,Nmax,Vnew)
implicit none

integer ::
num,nmax,count,n1,n2,m1,m2,BError,k,mumb1,mumb2,num1,num2,numb1,numb2,r
integer, dimension (2) :: m,buffer
integer, dimension (4) :: Sele,Lele
double precision :: alf,U,largest,smallest,diff,TOL,opt1,opt2
double precision, dimension(2) :: l,mu,a,b
double precision, dimension(2,0:nmax) :: C
double precision, dimension (200,8) :: indata
double precision, dimension(0:Nmax,0:m(1),0:Nmax,0:m(2)) :: Vold,Vnew

call inputdata(indata,r,l,mu,m,alf,nmax,buffer,a,b,BError)

TOL = 0.00001 ! 0.0005
C = 0.0

do k = 1,2
  do num = buffer(k),Nmax
    C(k,num) = a(k)*(real(num-buffer(k))**3.0) +
b(k)*(real(num-buffer(k))**4.0)
  end do
end do

!BError = 1

Vold = 0.0
Vnew = 0.0
count = 0

40 Vold = Vnew

count = count + 1

do n1 = 0,Nmax
  do n2 = 0,Nmax
    do m1 = 0,m(1)
      do m2 = 0,m(2)

        call arrnext(1,Nmax,n1,n2,num1,num2)
        call arrnext(2,Nmax,n1,n2,num1,num2)
        call sernext(1,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)
        call sernext(2,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)

        if(n1>0 .and. n2>0) then

          if(m1==0 .and. m2==0) then

            !this calculates which queue we are serving if we have the
            !choice & there are customers present
              U = l(1) + l(2) + mu(1) + alf
              opt1 = ((C(1,n1) + C(2,n2))/U) + (l(1)/U)*Vold(num1,1,n2,0) &
                    & +  (l(2)/U)*Vold(n1,1,num2,0) +
 (mu(1)/U)*Vold(numb1,mumb1,n2,0)
```

```
            U = l(1) + l(2) + mu(2) + alf
            opt2 = ((C(1,n1) + C(2,n2))/U) + (l(1)/U)*Vold(num1,0,n2,1) &
                 & +  (l(2)/U)*Vold(n1,0,num2,1) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)


          if (opt1 <= opt2) then
            Vnew(n1,m1,n2,m2) = opt1
          else
            Vnew(n1,m1,n2,m2) = opt2
          end if

        else if(m1>0 .and. m2==0) then
          U = l(1) + l(2) + mu(1) + alf
          Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                   & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(1)/U)*Vold(numb1,mumb1,n2,0)

        else if(m1==0 .and. m2>0) then
            U = l(1) + l(2) + mu(2) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                   & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)

        end if

      else if(n1>0 .and. n2==0) then
        if(m1==0 .and. m2==0) then
          U = l(1) + l(2) + mu(1) + alf
          Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,1,n2,0) &
                                   & +  (l(2)/U)*Vold(n1,1,num2,0) +
(mu(1)/U)*Vold(numb1,mumb1,n2,0)
        else if(m1>0 .and. m2==0) then
          U = l(1) + l(2) + mu(1) + alf
          Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                   & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(1)/U)*Vold(numb1,mumb1,n2,m2)
        end if

      else if(n1==0 .and. n2>0) then
        if(m1==0 .and. m2==0) then
          U = l(1) + l(2) + mu(2) + alf
          Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,0,n2,1) &
                                   & +  (l(2)/U)*Vold(n1,0,num2,1) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)
        else if(m1==0 .and. m2>0) then
          U = l(1) + l(2) + mu(2) + alf
          Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                   & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)
        end if
      else if(n1==0 .and. n2==0) then
        if(m1==0 .and. m2==0) then
          U = l(1) + l(2) + alf
          Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                   & +  (l(2)/U)*Vold(n1,m1,num2,m2)
        end if
      end if

    end do
  end do
 end do
end do

!open (unit=7,file="VvaluesD1.dat")
!write(unit=7,fmt="(3f16.4)") Vnew(39,0,39,0)
```

```
smallest = 1000000.0
largest = -1000000.0
Sele = 999
Lele = 999

do n1 = 1,Nmax-BError
  do n2 = 0,Nmax-BError
    do m1 = 1,m(1)

      m2 = 0

        if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do

do n1 = 0,Nmax-BError
  do n2 = 1,Nmax-BError
    do m2 = 1,m(2)

        m1 = 0

        if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do

n1 = 0 !do n1 = 0,0
n2 = 0 !  do n2 = 0,0

    m2 = 0
    m1 = 0

    if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
      smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
      Sele = (/n1,m1,n2,m2/)
    end if
    if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
      largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
      Lele = (/n1,m1,n2,m2/)
    end if

!  end do
!end do


diff = largest - smallest

!open (unit=7,file="GSOdiff.dat")
!write(unit=7,fmt="(3f16.4,8i4)") smallest,largest,diff,Sele,Lele
!close(unit=7)
if (count > 9000) goto 400
if (largest > TOL .or. largest < 0.0) goto 40

!print*,"4"

!100 WIcost = (smallest + largest)/2.0
```

```
!close(unit=7)
!close(unit=7)


!print*,"U = ",U
400 print*,"Count = ",count
print*,"The Optimal policy cost this queue setup & parameters is
",Vnew(0,0,0,0)
!close(unit=7)

return
end subroutine

!---------------------------------------------------------------------------
----------
!this one does not work properly?!? I not sure why put not going to
!use it anyway.

!NB here we have a state space of (n1,m1,n2,m2) where m1 is the
!number of phase completions we have done for class 1, i.e, m1
!starts off at 0 and goes upto m(1)-1, as when the m(1)th phase
!completion is over n1 goes to n1-1 and m1 goes back to 0 as this
!is the start of the service of the next queuing customer. (similarly for
m2)

!this subroutine only looks at real possible events occuring, i.e. there
!are no virtual events (except when n=boundery case).

!Now we have an extra state, m=1 is where we have started a service
!but have not finished the first phase of that service

subroutine Static_Costs3(indata,r,m,Nmax,Vnew)
implicit none

integer ::
num,nmax,count,n1,n2,m1,m2,BError,k,mumb1,mumb2,num1,num2,numb1,numb2,r
integer, dimension (2) :: m,buffer
integer, dimension (4) :: Sele,Lele
double precision :: alf,U,largest,smallest,diff,TOL,x
double precision, dimension(2) :: l,mu,a,b
double precision, dimension(2,0:nmax) :: C
double precision, dimension (200,8) :: indata
double precision, dimension(0:Nmax,0:m(1),0:Nmax,0:m(2)) :: Vold,Vnew

call inputdata(indata,r,l,mu,m,alf,nmax,buffer,a,b,BError)

TOL = 0.00001 ! 0.0005
C = 0.0
x = 0.0

do k = 1,2
  do num = buffer(k),Nmax
    C(k,num) = a(k)*(real(num-buffer(k))**3.0) +
b(k)*(real(num-buffer(k))**4.0)
  end do
end do

!BError = 1

Vold = 0.0
Vnew = 0.0
count = 0

50 Vold = Vnew

count = count + 1

do n1 = 0,Nmax
  do n2 = 0,Nmax
    do m1 = 0,m(1)
      do m2 = 0,m(2)

        call arrnext(1,Nmax,n1,n2,num1,num2)
        call arrnext(2,Nmax,n1,n2,num1,num2)
```

```
        call sernext(1,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)
        call sernext(2,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)

        if(n1>0 .and. n2>0) then

          if(m1==0 .and. m2==0) then

            !this calculates which queue we are serving if we have the
            !choice & there are customers present
            call Random_Number(x)
            if (x <= 0.5) then
              U = l(1) + l(2) + mu(1) + alf
              Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,1,n2,0) &
                                    & +  (l(2)/U)*Vold(n1,1,num2,0) +
(mu(1)/U)*Vold(numb1,mumb1,n2,0)
            else
              U = l(1) + l(2) + mu(2) + alf
              Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,0,n2,1) &
                                    & +  (l(2)/U)*Vold(n1,0,num2,1) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)
            end if

          else if(m1>0 .and. m2==0) then
            U = l(1) + l(2) + mu(1) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                    & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(1)/U)*Vold(numb1,mumb1,n2,0)

          else if(m1==0 .and. m2>0) then
              U = l(1) + l(2) + mu(2) + alf
              Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                    & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)

          end if

        else if(n1>0 .and. n2==0) then
          if(m1==0 .and. m2==0) then
            U = l(1) + l(2) + mu(1) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,1,n2,0) &
                                    & +  (l(2)/U)*Vold(n1,1,num2,0) +
(mu(1)/U)*Vold(numb1,mumb1,n2,0)
          else if(m1>0 .and. m2==0) then
            U = l(1) + l(2) + mu(1) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                    & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(1)/U)*Vold(numb1,mumb1,n2,m2)
          end if

        else if(n1==0 .and. n2>0) then
          if(m1==0 .and. m2==0) then
            U = l(1) + l(2) + mu(2) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,0,n2,1) &
                                    & +  (l(2)/U)*Vold(n1,0,num2,1) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)
          else if(m1==0 .and. m2>0) then
            U = l(1) + l(2) + mu(2) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                    & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)
          end if
        else if(n1==0 .and. n2==0) then
          if(m1==0 .and. m2==0) then
            U = l(1) + l(2) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                    & +  (l(2)/U)*Vold(n1,m1,num2,m2)
```

```
          end if
        end if

      end do
    end do
  end do
end do

!open (unit=7,file="VvaluesD1.dat")
!write(unit=7,fmt="(3f16.4)") Vnew(39,0,39,0)

smallest = 1000000.0
largest = -1000000.0
Sele = 999
Lele = 999

do n1 = 1,Nmax-BError
  do n2 = 0,Nmax-BError
    do m1 = 1,m(1)

      m2 = 0

        if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do

do n1 = 0,Nmax-BError
  do n2 = 1,Nmax-BError
    do m2 = 1,m(2)

        m1 = 0

        if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do

n1 = 0 !do n1 = 0,0
n2 = 0 !  do n2 = 0,0

    m2 = 0
    m1 = 0

    if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
      smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
      Sele = (/n1,m1,n2,m2/)
    end if
    if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
      largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
      Lele = (/n1,m1,n2,m2/)
    end if

!  end do
!end do


diff = largest - smallest
```

```
!open (unit=7,file="GSSdiff.dat")
!write(unit=7,fmt="(3f16.4,8i4)") smallest,largest,diff,Sele,Lele
!close(unit=7)
if (count > 9000) goto 500
if (largest > TOL .or. largest < 0.0) goto 50

!print*,"4"

!100 WIcost = (smallest + largest)/2.0

!close(unit=7)
!close(unit=7)


!print*,"U = ",U
500 print*,"Count = ",count
print*,"The static (equal splitting) policy cost this queue setup &
parameters is ",Vnew(0,0,0,0)
!close(unit=7)

return
end subroutine

!-----------------------------------------------------------------------
----------

!NB here we have a state space of (n1,m1,n2,m2) where m1 is the
!number of phase completions we have done for class 1, i.e, m1
!starts off at 0 and goes upto m(1)-1, as when the m(1)th phase
!completion is over n1 goes to n1-1 and m1 goes back to 0 as this
!is the start of the service of the next queuing customer. (similarly for
m2)

!this subroutine only looks at real possible events occuring, i.e. there
!are no virtual events (except when n=boundery case).

!Now we have an extra state, m=1 is where we have started a service
!but have not finished the first phase of that service

subroutine LongestQ_Costs3(indata,r,m,Nmax,Vnew)
implicit none

integer ::
num,nmax,count,n1,n2,m1,m2,BError,k,mumb1,mumb2,num1,num2,numb1,numb2,r
integer, dimension (2) :: m,buffer
integer, dimension (4) :: Sele,Lele
double precision :: alf,U,largest,smallest,diff,TOL
double precision, dimension(2) :: l,mu,a,b
double precision, dimension(2,0:nmax) :: C
double precision, dimension (200,8) :: indata
double precision, dimension(0:Nmax,0:m(1),0:Nmax,0:m(2)) :: Vold,Vnew

call inputdata(indata,r,l,mu,m,alf,nmax,buffer,a,b,BError)

TOL = 0.00001 ! 0.0005
C = 0.0

do k = 1,2
  do num = buffer(k),Nmax
    C(k,num) = a(k)*(real(num-buffer(k))**3.0) +
b(k)*(real(num-buffer(k))**4.0)
  end do
end do

!BError = 1

Vold = 0.0
Vnew = 0.0
count = 0

60 Vold = Vnew

count = count + 1
```

```
do n1 = 0,Nmax
  do n2 = 0,Nmax
    do m1 = 0,m(1)
      do m2 = 0,m(2)

        call arrnext(1,Nmax,n1,n2,num1,num2)
        call arrnext(2,Nmax,n1,n2,num1,num2)
        call sernext(1,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)
        call sernext(2,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)

        if(n1>0 .and. n2>0) then

          if(m1==0 .and. m2==0) then

            !this calculates which queue we are serving if we have the
            !choice & there are customers present
            if (n1 >= n2) then
              U = l(1) + l(2) + mu(1) + alf
              Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,1,n2,0) &
                                            & +  (l(2)/U)*Vold(n1,1,num2,0) |
(mu(1)/U)*Vold(numb1,mumb1,n2,0)
            else
              U = l(1) + l(2) + mu(2) + alf
              Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,0,n2,1) &
                                            & +  (l(2)/U)*Vold(n1,0,num2,1) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)
            end if

          else if(m1>0 .and. m2==0) then
            U = l(1) + l(2) + mu(1) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                            & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(1)/U)*Vold(numb1,mumb1,n2,0)

          else if(m1==0 .and. m2>0) then
              U = l(1) + l(2) + mu(2) + alf
              Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                            & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)

          end if

        else if(n1>0 .and. n2==0) then
          if(m1==0 .and. m2==0) then
            U = l(1) + l(2) + mu(1) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,1,n2,0) &
                                            & +  (l(2)/U)*Vold(n1,1,num2,0) +
(mu(1)/U)*Vold(numb1,mumb1,n2,0)
          else if(m1>0 .and. m2==0) then
            U = l(1) + l(2) + mu(1) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                            & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(1)/U)*Vold(numb1,mumb1,n2,m2)
          end if

        else if(n1==0 .and. n2>0) then
          if(m1==0 .and. m2==0) then
            U = l(1) + l(2) + mu(2) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,0,n2,1) &
                                            & +  (l(2)/U)*Vold(n1,0,num2,1) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)
          else if(m1==0 .and. m2>0) then
            U = l(1) + l(2) + mu(2) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                            & +  (l(2)/U)*Vold(n1,m1,num2,m2) +
(mu(2)/U)*Vold(n1,m1,numb2,mumb2)
          end if
```

```
        else if(n1==0 .and. n2==0) then
          if(m1==0 .and. m2==0) then
            U = l(1) + l(2) + alf
            Vnew(n1,m1,n2,m2) = ((C(1,n1) + C(2,n2))/U) +
(l(1)/U)*Vold(num1,m1,n2,m2) &
                                & +  (l(2)/U)*Vold(n1,m1,num2,m2)
          end if
        end if

      end do
    end do
  end do
end do

!open (unit=7,file="VvaluesD1.dat")
!write(unit=7,fmt="(3f16.4)") Vnew(39,0,39,0)

smallest = 1000000.0
largest = -1000000.0
Sele = 999
Lele = 999

do n1 = 1,Nmax-BError
  do n2 = 0,Nmax-BError
    do m1 = 1,m(1)

      m2 = 0

        if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do

do n1 = 0,Nmax-BError
  do n2 = 1,Nmax-BError
    do m2 = 1,m(2)

        m1 = 0

        if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do

n1 = 0 !do n1 = 0,0
n2 = 0 !  do n2 = 0,0

    m2 = 0
    m1 = 0

    if ( smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
      smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
      Sele = (/n1,m1,n2,m2/)
    end if
    if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
      largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
      Lele = (/n1,m1,n2,m2/)
    end if
```

```
!  end do
!end do

diff = largest - smallest

!open(unit=7,file="GSLQdiff.dat")
!write(unit=7,fmt="(3f16.4,8i4)") smallest,largest,diff,Sele,Lele
if (count > 9000) goto 600
if (largest > TOL .or. largest < 0.0) goto 60

!print*,"4"

!100 WIcost = (smallest + largest)/2.0

!close(unit=7)
!close(unit=7)


!print*,"U = ",U
600 print*,"Count = ",count
print*,"The longest queue policy cost this queue setup & parameters is
",Vnew(0,0,0,0)
!close(unit=7)

return
end subroutine

!-----------------------------------------------------------------------------
----------

subroutine WIndex(indata,r,m,Nmax,W)
implicit none

integer :: n,num,nmax,i,k,BError,r
integer, dimension(2) :: buffer,m
double precision :: alf
double precision, dimension(2) :: l,mu,a,b,EalfT
double precision, dimension(2,0:nmax) :: Chat2,W,C
double precision, dimension(0:nmax,m(1)) :: Chata
double precision, dimension(0:nmax,m(2)) :: Chatb
double precision, dimension (200,8) :: indata

call inputdata(indata,r,l,mu,m,alf,nmax,buffer,a,b,BError)
C = 0.0

do k = 1,2
  do num = buffer(k),Nmax
    C(k,num) = a(k)*(real(num-buffer(k))**3.0) +
b(k)*(real(num-buffer(k))**4.0)
  end do
end do

call iteration(indata,r,Nmax,m,EalfT,Chata,Chatb)

!We now have to convert EalfT and Chat to be the correct values
!for a service completion as they are only currently for a phase
!completion.

Chat2 = 0.0
W = 0.0

do k = 1,2
  do n = 0,Nmax
    if(k==1) Chat2(k,n) = Chata(n,m(k))
    if(k==2) Chat2(k,n) = Chatb(n,m(k))
    do i = 1,m(k)-1
      if(k==1) Chat2(k,n) = Chat2(k,n) +
(EalfT(k)**(real(i)))*Chata(n,m(k)-i)
      if(k==2) Chat2(k,n) = Chat2(k,n) +
(EalfT(k)**(real(i)))*Chatb(n,m(k)-i)
    end do
  end do
end do

do k = 1,2
```

```
  EalfT(k) = EalfT(k)**(real(m(k)))
end do

do k = 1,2
  do n = 1,Nmax
    W(k,n) = (alf*(C(k,n)-(EalfT(k)*C(k,n-1))) + l(k)*Chat2(k,n) +
(alf+l(k))*Chat2(k,n))/(1.0 - EalfT(k))
  end do
end do

return
end subroutine

!------------------------------------------------------------------------------
----------
!NB here we have a state space of (n,s) where s is the
!number of phase completions we have left to do 1 before
!the service completion is over for that customer, i.e, s starts
!off at m and goes down to 1, as when the s=1 phase completion
!is over (we don't let s=0 since) n goes to n-1 and s goes back
!to m as this is the start of the service of the next queuing
!customer. (NB m = number of phase completions in a service completion).

!this calculates Discounted cost of moving from state (n,s) to (n,s-1)
!and calculates E(T) where T = a phase completion time.
subroutine iteration(indata,r,Nmax,m,EalfT,Chata,Chatb)
implicit none

integer :: i,nmax,count,s,n,k,BError,r
integer, dimension(2) :: buffer,m
double precision :: old,new,TOL,initial,alf
double precision, dimension(2) :: l,mu,a,b,EalfT
double precision, dimension(0:nmax,m(1)) :: new2a,old2a,Chata
double precision, dimension(0:nmax,m(2)) :: new2b,old2b,Chatb
double precision, dimension (200,8) :: indata

call inputdata(indata,r,l,mu,m,alf,nmax,buffer,a,b,BError)

TOL = 0.00001 ! 0.0000005
initial = 0.0
new = initial
old = initial
old2a = initial
new2a = initial
old2b = initial
new2b = initial

do k=1,2
  new = initial
  old = initial
  if (k==1) old2a = initial
  if (k==1) new2a = initial
  if (k==2) old2b = initial
  if (k==2) new2b = initial

!  print*,"k l m mu",k,l(k),m(k),mu(k)
  do i = 1,1000000
    call function1(indata,r,k,old,new)
    if (new-old < TOL .and. new-old > -TOL) goto 10
    old = new
!     print*,new
  end do
  10 print*,"i is :",i
  EalfT(k) = new

!  10 print*,"the value we get for E(exp(-alf*T)) = ",new
!  print*,"i = ",i

  count = 0
  20 count = count + 1
!     print*,"internal count = ",count
    if (k==1) old2a = new2a
    if (k==2) old2b = new2b
    call function2(indata,r,k,old2a,old2b,new2a,new2b,nmax,m,new)
    do n = 0,nmax
```

```
      do s = 1,m(k)
        if(k==1) then
          if (new2a(n,s)-old2a(n,s) > TOL .or. new2a(n,s)-old2a(n,s) < -TOL
.and. count < 120000) goto 20
        else if(k==2) then
          if (new2b(n,s)-old2b(n,s) > TOL .or. new2b(n,s)-old2b(n,s) < -TOL
.and. count < 120000) goto 20
        end if
      end do
    end do

    if(k==1) Chata = new2a
    if(k==2) Chatb = new2b
!    if(k==2) then
!    open(unit=7,file="iterationChat.dat")

!!    write(unit=6,fmt="(a)") "   b1   :    c1   :    b2   :    c2   :
Nmax"
!!    write(unit=6,fmt="(4f10.4,i5)") a(1),b(1),a(2),b(2),Nmax

!  write(unit=7,fmt="(a)") "the value we get for (k=1) Chat(n,s,alf)) is: "
!  do n = 0,nmax
!    do s = 1,m
!      write(unit=7,fmt="(2i5,f10.4)") n,s,Chata(n,s)
!    end do
!  end do
!  write(unit=7,fmt="(a)") "the value we get for (k=2) Chat(n,s,alf)) is: "
!  do n = 0,nmax
!    do s = 1,m
!      write(unit=7,fmt="(2i5,f10.4)") n,s,Chatb(n,s)
!    end do
!  end do

!  close(unit=7)
!  end if

  print*,"count inter = ",count
  print*," EalfT(",k,") = ", EalfT(k)

  !if(k==1) Chata = new2a
  !if(k==2) Chatb = new2b
end do

return
end subroutine

!---------------------------------------------------------------------------
!this calculates E(T) where T = a phase completion time
subroutine function1(indata,r,k,old,new)
implicit none

integer :: nmax,k,BError,r
integer, dimension(2) :: m,buffer
double precision :: old,new,alf
double precision, dimension(2) :: l,mu,a,b
double precision, dimension (200,8) :: indata

call inputdata(indata,r,l,mu,m,alf,nmax,buffer,a,b,BError)

new = (mu(k) + l(k)*(old**(real(m(k)+1))))/(alf + mu(k) + l(k))

return
end subroutine

!---------------------------------------------------------------------------
!NB here we have a state space of (n,s) where s is the
!number of phase completions we have left to do 1 before
!the service completion is over for that customer, i.e, s starts
!off at m and goes down to 1, as when the s=1 phase completion
!is over (we don't let s=0 since) n goes to n-1 and s goes back
!to m as this is the start of the service of the next queuing
!customer. (NB m = number of phase completions in a service completion).

!this calculates Discounted cost of moving from state (n,s) to (n,s-1)
subroutine function2(indata,r,k,old2a,old2b,new2a,new2b,nmax,m,new)
```

```
implicit none

integer :: nmax,num,n,s,k,BError,r
integer, dimension(2) :: m,buffer
double precision :: alf,temp1,temp2,new
double precision, dimension(2) :: l,mu,a,b
double precision, allocatable, dimension(:,:) :: C
double precision, dimension(0:nmax,m(1)) :: new2a,old2a
double precision, dimension(0:nmax,m(2)) :: new2b,old2b
double precision, dimension (200,8) :: indata

call inputdata(indata,r,l,mu,m,alf,nmax,buffer,a,b,BError)

allocate( C(2,0:nmax) )
!allocate( new2(nmax,m) )
!allocate( old2(nmax,m) )

C(k,:) = 0.0


do num = buffer(k),Nmax
  C(k,num) = a(k)*(real(num-buffer(k))**3.0) +
b(k)*(real(num-buffer(k))**4.0)
end do

if (k==1) then
  do n = 0,nmax
    do s = 1,m(k)

      if (n < nmax) then
        temp1 = old2a(n+1,s)
      else
        temp1 = old2a(n,s)
      end if

      if (s > 1) then
        temp2 = old2a(n,s-1)
      else
        temp2 = 0.0
      end if

      new2a(n,s) = (C(k,n)/(l(k)+mu(k)+alf)) +
((l(k)/(l(k)+mu(k)+alf))*(temp1 + (old2a(n,m)*(new**(real(s)))))) &
                 & + ((mu(k)/(l(k)+mu(k)+alf))*temp2)

    end do
  end do
else if(k==2) then
  do n = 0,nmax
    do s = 1,m(k)

      if (n < nmax) then
        temp1 = old2b(n+1,s)
      else
        temp1 = old2b(n,s)
      end if

      if (s > 1) then
        temp2 = old2b(n,s-1)
      else
        temp2 = 0.0
      end if

      new2b(n,s) = (C(k,n)/(l(k)+mu(k)+alf)) +
((l(k)/(l(k)+mu(k)+alf))*(temp1 + (old2b(n,m(k))*(new**(real(s)))))) &
                 & + ((mu(k)/(l(k)+mu(k)+alf))*temp2)

    end do
  end do
end if

return
end subroutine

!------------------------------------------------------------------------
```

```
subroutine arrnext(k,Nmax,n1,n2,num1,num2)
implicit none

integer :: Nmax,n1,n2,k,num1,num2

if(k==1) then
  num1 = n1
  if(n1<Nmax) then
    num1 = n1+1
  end if
end if

if(k==2) then
  num2 = n2
  if(n2<Nmax) then
    num2 = n2+1
  end if
end if

return
end subroutine

!---------------------------------------------------------------------------
-------------------

subroutine sernext(k,m,n1,n2,num1,num2,m1,m2,mum1,mum2)
implicit none

integer :: n1,n2,k,num1,num2,m1,m2,mum1,mum2
integer, dimension(2) :: m


if(k==1) then
  num1 = n1
  mum1 = m1
  if(m1<m(1) .and. m1>0) then
    mum1 = m1+1
  else if(m1<m(1) .and. m1==0) then
    mum1 = 2
  else if(m1==m(1)) then
    num1 = n1-1
    mum1 = 0
  end if
end if

if(k==2) then
  num2 = n2
  mum2 = m2
  if(m2<m(2) .and. m2>0) then
    mum2 = m2+1
  else if(m2<m(2) .and. m2==0) then
    mum2 = 2
  else if(m2==m(2)) then
    num2 = n2-1
    mum2 = 0
  end if
end if

return
end subroutine

!---------------------------------------------------------------------------
-------------------

subroutine inputdata(indata,r,l,mu,m,alf,nmax,buffer,a,b,BError)
implicit none

integer :: nmax,BError,r
integer, dimension(2) :: m,buffer
double precision :: alf
double precision, dimension(2) :: l,mu,a,b
double precision, dimension (200,8) :: indata

nmax = 139
```

```
buffer = (/0,0/)
alf = -log(0.95)
BError = 9

m = (/2,3/)
mu(1) = indata(r,1)
mu(2) = indata(r,2)
l(1) = indata(r,3)
l(2) = ( indata(r,4) - (m(1)*l(1)/mu(1)) )*mu(2)/m(2)
!print*,"l2 = ",l(2)
!traffic intensity = 2.0*l(1)/mu(1) + 3.0*l(2)/mu(2)

a(1) = indata(r,5)
b(1) = indata(r,7)
a(2) = indata(r,6)
b(2) = indata(r,8)

!l = (/0.25,0.4/)
!mu = (/6.0,5.0/)
!a = (/1.0,2.0/)
!b = (/0.5,0.2/)

return
end subroutine

!---------------------------------------------------------------------
```

Appendix E

This appendix contains the Fortran 95 code for the programme we used to calculate the undiscounted service control costs as in Section 3.5.2. Here we consider the optimal and index policies for a 2 class system.

```
program generalservicemk2
implicit none

integer :: Nmax,BError,n,r,i
integer, dimension(2) :: m,buffer
double precision :: a,b,d,e,TOL,WC2,OC2,LC2
double precision, dimension(2) :: l,mu
double precision, dimension (200,8) :: indata
double precision, allocatable, dimension(:,:,:) :: P
double precision, allocatable, dimension(:,:) :: C,W,Delta,pi

indata = 0.0
open(unit=7,file="GSinputdata.dat")
do i  = 1,200
  read(unit=7,fmt="(8f10.4)") indata(i,:)
end do
close(unit=7)
r=1
call qvals(indata,r,a,b,d,e,l,m,mu,Nmax,buffer,BError,TOL)

allocate( C(2,0:Nmax) )
allocate( W(2,0:Nmax) )
allocate( P(2,0:Nmax,0:Nmax) )
allocate( Delta(2,0:Nmax) )
allocate( pi(2,0:Nmax) )

WC2 = 0.0
OC2 = 0.0

!open(unit=7, file="GSsubstatquad69.dat")
do r = 23,200
  call qvals(indata,r,a,b,d,e,l,m,mu,Nmax,buffer,BError,TOL)

C = 0.0
do n=buffer(1),Nmax
  C(1,n) = a*(real(n-buffer(1))**1.0) + b*(real(n-buffer(1))**2.0)
end do

do n=buffer(2),Nmax
  C(2,n) = d*(real(n-buffer(2))**1.0) + e*(real(n-buffer(2))**2.0)
end do

print*," r = ",r

!calculate the correct stationary distributions
call Windex1(indata,r,Nmax,pi)

call Windex2(indata,r,Nmax,pi,W)

call WHITcosts3(indata,r,Nmax,W,WC2)

call OPTcosts3(indata,r,Nmax,OC2)

!call LQcosts3(indata,r,Nmax,LC2)

open(unit=7, file="GSsubstatquad69buff2_pt2.dat")!,status="old")

write(unit=7,fmt="(a)") "    b1    :    c1    :    b2    :    c2    : Nmax"
write(unit=7,fmt="(4f10.4,i5)") a,b,d,e,Nmax
write(unit=7,fmt="(a)") "    l1    :    l2    :    m1 : mu1    :    m2 : mu2    "

write(unit=7,fmt="(2f10.4,i5,f10.4,i5,f10.4)")
l(1),l(2),m(1),mu(1),m(2),mu(2)
write(unit=7,fmt="(a)") " "
write(unit=7,fmt="(a,f15.8)") "Optimal Policy Cost              : ",OC2
write(unit=7,fmt="(a)") " "
write(unit=7,fmt="(a,f15.8)") "Dynamic Index Policy Cost        : ",WC2
write(unit=7,fmt="(a,f15.8)") "Percentage (cost) Suboptimallity :
",100.0*(WC2-OC2)/OC2
write(unit=7,fmt="(a)") " "
!write(unit=7,fmt="(a,f15.8)") "Longest Queue Cost               : ",LC2
!write(unit=7,fmt="(a,f15.8)") "Percentage (cost) Suboptimallity :
",100.0*(LC2-OC2)/OC2
!write(unit=7,fmt="(a)") " "
```

```
end do

close(unit=7)

end program

!-----------------------------------------------------------------------------
------------------------------

subroutine factorial(z,fact)
implicit none

integer :: z,i
double precision :: fact,tot

tot = 1

if (z > 0) then

  do i = 1,z

    tot = tot*real(i)

  end do

  fact = tot

else if (z == 0) then

  fact = 1.0

else

  print*,"ERROR cannot find factorial of negative number"
  fact = 0.0

end if

return
end subroutine

!-----------------------------------------------------------------------------
------------------------------

subroutine matmult(Nmax,P,pi)
implicit none

integer :: n,iz,opt,ifail,count,Nmax,k !,j
double precision,dimension(1) :: Z
double precision, dimension(0:Nmax,0:Nmax) :: A,B
double precision, dimension(2,0:Nmax) :: pi
double precision, dimension(2,0:Nmax,0:Nmax) :: P
external F01CKF

n = Nmax+1

do k = 1,2
  B = P(k,:,:)
  !print*,"B = "
  !do j = 0,n-1
  !  if ( i == 1) print*,B(0,j)
  !end do

  opt = 1
  iz = 1
  ifail = 0

  count = 0


  10 call F01CKF(A,B,B,n,n,n,Z,iz,opt,ifail)
     count = count + 1
     B = A

     if (count < 30) goto 10
```

```
!open(unit=7, file="statdist.dat")

!write(unit=7,fmt="(1f12.6)") A(0,:)
!write(unit=7,fmt="(1f12.6)") 10000.00000

  pi(k,:) = A(0,:)

end do

return
end subroutine

!----------------------------------------------------------------------------
------------------------------

subroutine windex1(indata,r,Nmax,pi)
implicit none

integer :: Nmax,BError,j,i,n,k,r
integer, dimension(2) :: m,buffer
double precision :: a,b,d,e,TOL,temp1,temp2,temp3
double precision, dimension(2) :: l,mu,row
double precision, dimension (200,8) :: indata
double precision, dimension(2,0:Nmax) :: Delta,pi
double precision, dimension(0:Nmax,0:Nmax) :: AM,BM
double precision, dimension(2,0:Nmax,0:Nmax) :: P

call qvals(indata,r,a,b,d,e,l,m,mu,Nmax,buffer,BError,TOL)

!calculate the markov chain transition matrix

do k = 1,2

  call factorial(m(k)-1,temp3)

  do j = 0,Nmax

    call factorial(m(k)+j-1,temp1)
    call factorial(j,temp2)

    Delta(k,j) = ( temp1/(temp2*temp3) )*( (l(k)/(l(k)+mu(k)))**(real(j))
)*( (mu(k)/(l(k)+mu(k)))**(real(m(k))) )

  end do

  P(k,:,:) = 0.0

  do i = 0,Nmax

    P(k,0,i) = Delta(k,i)

  end do

  P(k,1,:) = P(k,0,:)

  do j = 1,Nmax-1
    do i = j,Nmax

      P(k,j+1,i) = Delta(k,i-j)

    end do
  end do

end do

!calculate the state probabilities - pi(k,j)
pi = 0.0
row = 0.0

AM = P(1,:,:)
BM = P(2,:,:)

do i = 1,25
  AM = matmul(AM,AM)
```

```
  BM = matmul(BM,BM)
end do

pi(1,:) = AM(0,:)
pi(2,:) = BM(0,:)

!open (unit=7,file="GSstatdistmat.dat")
!write(unit=7,fmt="(a)") "     b1    :     c1    :     b2    :     c2    : Nmax"
!write(unit=7,fmt="(4f10.4,i5)") a,b,d,e,Nmax
!write(unit=7,fmt="(a)") "     l1    :     l2    :     m1 : mu1    :    m2 : mu2
"
!write(unit=7,fmt="(2f10.4,i5,f10.4,i5,f10.4)")
l(1),l(2),m(1),mu(1),m(2),mu(2)
!write(unit=7,fmt="(a)") "The stationary distn is:"
!do i = 1,2
!   write(unit=7,fmt="(a,i4)") "class = ",i
!  do n = 0,Nmax
!    write(unit=7,fmt="(a,i4,a,i5,a,f16.12)") "pi(",i,",",n,")= ",pi(i,n)
!  end do
!end do
!close(unit=7)

return
end subroutine

!--------------------------------------------------------------------------
------------------------------

subroutine Windex2(indata,r,Nmax,pi,W)
implicit none

integer :: Nmax,BError,j,n,k,r
integer, dimension(2) :: m,buffer
double precision :: a,b,d,e,TOL
double precision, dimension(2) :: l,mu
double precision, dimension (200,8) :: indata
double precision, dimension(2,0:Nmax) :: pi
double precision, dimension(2,0:Nmax) :: W,EC
double precision, dimension(2,0:Nmax+Nmax) :: C

call qvals(indata,r,a,b,d,e,l,m,mu,Nmax,buffer,BError,TOL)

!calculate the markov chain transition matrix
W = 0.0
C = 0.0
EC = 0.0

do n=buffer(1),Nmax+Nmax
  C(1,n) = a*(real(n-buffer(1))**1.0) + b*(real(n-buffer(1))**2.0)
end do

do n=buffer(2),Nmax+Nmax
  C(2,n) = d*(real(n-buffer(2))**1.0) + e*(real(n-buffer(2))**2.0)
end do

do k = 1,2

  do n = 0,Nmax
    do j = 0,Nmax

      EC(k,n) = EC(k,n) + C(k,n+j)*pi(k,j)

!       if (n >= 32) then
!         open (unit=7,file="forming.dat")
!         write(unit=7,fmt="(3i4,3f16.4)") k,n,j,EC(k,n),C(k,n+j),pi(k,j)
!       end if

    end do
  end do

  !calculate the actual index

!  open (unit=7,file="forming1.dat")

  do n = 1,Nmax
```

```
!    write(unit=7,fmt="(2i4,3f16.4)") k,n,EC(k,n)
    W(k,n) = ( EC(k,n) - EC(k,n-1) )/(m(k)/mu(k))

  end do

  W(k,0) = 0.0

end do
!close(unit=7)

!open (unit=7,file="Indices.dat")
!write(unit=7,fmt="(a)") "The indices are:"
!do i = 0,Nmax
!write(unit=7,fmt="(2f16.4)") W(:,i)
!end do
!close(unit=7)

return
end subroutine

!-----------------------------------------------------------------------------
-------------------

subroutine arrnext(k,Nmax,n1,n2,num1,num2)
implicit none

integer :: Nmax,n1,n2,k,num1,num2

if(k==1) then
  num1 = n1
  if(n1<Nmax) then
    num1 = n1+1
  end if
end if

if(k==2) then
  num2 = n2
  if(n2<Nmax) then
    num2 = n2+1
  end if
end if

return
end subroutine

!-----------------------------------------------------------------------------
-------------------

subroutine sernext(k,m,n1,n2,num1,num2,m1,m2,mum1,mum2)
implicit none

integer :: n1,n2,k,num1,num2,m1,m2,mum1,mum2
integer, dimension(2) :: m


if(k==1) then
  num1 = n1
  mum1 = m1
  if(m1<m(1) .and. m1>0) then
    mum1 = m1+1
  else if(m1<m(1) .and. m1==0) then  !unsure
    mum1 = 2
  else if(m1==m(1)) then
    num1 = n1-1
    mum1 = 0
  end if
end if

if(k==2) then
  num2 = n2
  mum2 = m2
  if(m2<m(2) .and. m2>0) then
    mum2 = m2+1
  else if(m2<m(2) .and. m2==0) then         !unsure
```

```
    mum2 = 2
  else if(m2==m(2)) then
    num2 = n2-1
    mum2 = 0
  end if
end if

return
end subroutine

!--------------------------------------------------------------------------
-------------------

!NB here we have a state space of (n1,m1,n2,m2) where m1 is the
!number of phase completions we have done for class 1, i.e, m1
!starts off at 0 and goes upto m(1)-1, as when the m(1)th phase
!completion is over n1 goes to n1-1 and m1 goes back to 0 as this
!is the start of the service of the next queuing customer. (similarly for
m2)

!this subroutine only looks at virtual possible events occuring, i.e. there
!are events occuring which could not really happen but the effects of
!such events is nothing.

!Now we have an extra state, m=1 is where we have started a service
!but have not finished the first phase of that service.

subroutine Optcosts3(indata,r,Nmax,OC)
implicit none

integer ::
Nmax,BError,n,n1,n2,count,m1,m2,r,num1,num2,mumb1,mumb2,numb1,numb2
integer, dimension(2) :: m,buffer
integer, dimension(4) :: Sele,Lele
double precision :: a,b,d,e,TOL,U,smallest,largest,diff,OC,opt1,opt2
double precision, dimension(2) :: l,mu
double precision, dimension (200,8) :: indata
double precision, dimension(2,0:Nmax) :: C
double precision, allocatable, dimension(:,:,:,:) :: Vold,Vnew

call qvals(indata,r,a,b,d,e,l,m,mu,Nmax,buffer,BError,TOL)

allocate( Vold(0:Nmax,0:m(1),0:Nmax,0:m(2)) )
allocate( Vnew(0:Nmax,0:m(1),0:Nmax,0:m(2)) )

TOL = 0.0005
C = 0.0

do n=buffer(1),Nmax
  C(1,n) = a*(real(n-buffer(1))**1.0) + b*(real(n-buffer(1))**2.0)
end do

do n=buffer(2),Nmax
  C(2,n) = d*(real(n-buffer(2))**1.0) + e*(real(n-buffer(2))**2.0)
end do

U = l(1) + l(2) + mu(1) + mu(2)
l(1) = l(1)/U
l(2) = l(2)/U
mu(1) = mu(1)/U
mu(2) = mu(2)/U

Vold = 0.0
Vnew = 0.0
count = 0

14 Vold = Vnew

count = count + 1

do n1 = 0,Nmax
  do n2 = 0,Nmax
    do m1 = 0,m(1)
      do m2 = 0,m(2)
```

```
        call arrnext(1,Nmax,n1,n2,num1,num2)
        call arrnext(2,Nmax,n1,n2,num1,num2)
        call sernext(1,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)
        call sernext(2,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)

        if(n1>0 .and. n2>0) then

          if(m1==0 .and. m2==0) then

            opt1 = C(1,n1) + C(2,n2) + l(1)*Vold(num1,1,n2,0) +
l(2)*Vold(n1,1,num2,0) &
                & + mu(1)*Vold(numb1,mumb1,n2,0) + mu(2)*Vold(n1,1,n2,0)

            opt2 = C(1,n1) + C(2,n2) + l(1)*Vold(num1,0,n2,1) +
l(2)*Vold(n1,0,num2,1) &
                & + mu(1)*Vold(n1,0,n2,1) + mu(2)*Vold(n1,0,numb2,mumb2)

            if(opt1<=opt2) then
              Vnew(n1,m1,n2,m2) = opt1
            else
              Vnew(n1,m1,n2,m2) = opt2
            end if

          else if(m1>0 .and. m2==0) then

            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                          & + mu(1)*Vold(numb1,mumb1,n2,m2) +
mu(2)*Vold(n1,m1,n2,m2)

          else if(m1==0 .and. m2>0) then

            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                          & + mu(1)*Vold(n1,m1,n2,m2) +
mu(2)*Vold(n1,m1,numb2,mumb2)

          end if

        else if(n1>0 .and. n2==0) then

          if(m1==0 .and. m2==0) then
            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,1,n2,0) +
l(2)*Vold(n1,1,num2,0) &
                          & + mu(1)*Vold(numb1,mumb1,n2,0) +
mu(2)*Vold(n1,1,n2,0)
          else if(m1>0 .and. m2==0) then
            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                          & + mu(1)*Vold(numb1,mumb1,n2,m2) +
mu(2)*Vold(n1,m1,n2,m2)
          end if

        else if(n1==0 .and. n2>0) then

          if(m1==0 .and. m2==0) then
            Vnew(n1,m1,n2,m2)= C(1,n1) + C(2,n2) + l(1)*Vold(num1,0,n2,1) +
l(2)*Vold(n1,0,num2,1) &
                & + mu(1)*Vold(n1,0,n2,1) + mu(2)*Vold(n1,0,numb2,mumb2)
          else if(m1==0 .and. m2>0) then
            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                          & + mu(1)*Vold(n1,m1,n2,m2) +
mu(2)*Vold(n1,m1,numb2,mumb2)
          end if

        else if(n1==0 .and. n2==0) then

          if(m1==0 .and. m2==0) then
            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                          & + mu(1)*Vold(n1,m1,n2,m2) +
mu(2)*Vold(n1,m1,n2,m2)
          end if
```

```
        end if

      end do
    end do
  end do
end do

smallest = 1000000.0
largest = -1000000.0
Sele = 999
Lele = 999

do n1 = 1,Nmax-BError
  do n2 = 0,Nmax-BError
    do m1 = 1,m(1)

        m2 = 0

        if (smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do

do n1 = 0,Nmax-BError
  do n2 = 1,Nmax-BError
      do m2 = 1,m(2)

        m1 = 0

        if (smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do


n1 = 0
  n2 = 0
    m1 = 0
      m2 = 0

        if (smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

diff = largest - smallest

!open(unit=7,file="GSOdiff.txt")
!write(unit=7,fmt="(3f16.4,8i4)") smallest,largest,diff,Sele,Lele

if (count > 92500) goto 140
if (diff > smallest*TOL .or. diff < 0.0) goto 14

140 OC = (smallest + largest)/2.0
print*,100.0/OC
```

```
!close(unit=7)

print*,"Count = ",count
print*,"The optimal policy cost this queue setup & parameters is ",OC

return
end subroutine

!---------------------------------------------------------------------------
------------------------------

!NB here we have a state space of (n1,m1,n2,m2) where m1 is the
!number of phase completions we have done for class 1, i.e, m1
!starts off at 0 and goes upto m(1)-1, as when the m(1)th phase
!completion is over n1 goes to n1-1 and m1 goes back to 0 as this
!is the start of the service of the next queuing customer. (similarly for
m2)

!this subroutine only looks at virtual possible events occuring, i.e. there
!are events occuring which could not really happen but the effects of
!such events is nothing.

!Now we have an extra state, m=1 is where we have started a service
!but have not finished the first phase of that service.


subroutine WHITcosts3(indata,r,Nmax,W,WC)
implicit none

integer ::
Nmax,BError,n,n1,n2,count,m1,m2,r,num1,num2,mumb1,mumb2,numb1,numb2
integer, dimension(2) :: m,buffer
integer, dimension(4) :: Sele,Lele
double precision :: a,b,d,e,TOL,U,smallest,largest,diff,WC
double precision, dimension(2) :: l,mu
double precision, dimension (200,8) :: indata
double precision, dimension(2,0:Nmax) :: C,W
double precision, allocatable, dimension(:,:,:,:) :: Vold,Vnew

call qvals(indata,r,a,b,d,e,l,m,mu,Nmax,buffer,BError,TOL)

allocate( Vold(0:Nmax,0:m(1),0:Nmax,0:m(2)) )
allocate( Vnew(0:Nmax,0:m(1),0:Nmax,0:m(2)) )

TOL = 0.0005
C = 0.0

do n=buffer(1),Nmax
  C(1,n) = a*(real(n-buffer(1))**1.0) + b*(real(n-buffer(1))**2.0)
end do

do n=buffer(2),Nmax
  C(2,n) = d*(real(n-buffer(2))**1.0) + e*(real(n-buffer(2))**2.0)
end do

U = l(1) + l(2) + mu(1) + mu(2)
l(1) = l(1)/U
l(2) = l(2)/U
mu(1) = mu(1)/U
mu(2) = mu(2)/U

Vold = 0.0
Vnew = 0.0
count = 0

12 Vold = Vnew

count = count + 1

do n1 = 0,Nmax
  do n2 = 0,Nmax
    do m1 = 0,m(1)
      do m2 = 0,m(2)
```

```
        call arrnext(1,Nmax,n1,n2,num1,num2)
        call arrnext(2,Nmax,n1,n2,num1,num2)
        call sernext(1,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)
        call sernext(2,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)

        if(n1>0 .and. n2>0) then

          if(m1==0 .and. m2==0) then

            if(W(1,n1) >= W(2,n2)) then

              Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,1,n2,0)
+ l(2)*Vold(n1,1,num2,0) &
                        & + mu(1)*Vold(numb1,mumb1,n2,0) +
mu(2)*Vold(n1,1,n2,0)
            else

              Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,0,n2,1)
+ l(2)*Vold(n1,0,num2,1) &
                        & + mu(1)*Vold(n1,0,n2,1) +
mu(2)*Vold(n1,0,numb2,mumb2)
            end if

          else if(m1>0 .and. m2==0) then

            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                        & + mu(1)*Vold(numb1,mumb1,n2,m2) +
mu(2)*Vold(n1,m1,n2,m2)

          else if(m1==0 .and. m2>0) then

            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                        & + mu(1)*Vold(n1,m1,n2,m2) +
mu(2)*Vold(n1,m1,numb2,mumb2)

          end if

        else if(n1>0 .and. n2==0) then

          if(m1==0 .and. m2==0) then
            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,1,n2,0) +
l(2)*Vold(n1,1,num2,0) &
                        & + mu(1)*Vold(numb1,mumb1,n2,0) +
mu(2)*Vold(n1,1,n2,0)
          else if(m1>0 .and. m2==0) then
            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                        & + mu(1)*Vold(numb1,mumb1,n2,m2) +
mu(2)*Vold(n1,m1,n2,m2)
          end if

        else if(n1==0 .and. n2>0) then

          if(m1==0 .and. m2==0) then
            Vnew(n1,m1,n2,m2)= C(1,n1) + C(2,n2) + l(1)*Vold(num1,0,n2,1) +
l(2)*Vold(n1,0,num2,1) &
                  & + mu(1)*Vold(n1,0,n2,1) + mu(2)*Vold(n1,0,numb2,mumb2)
          else if(m1==0 .and. m2>0) then
            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                        & + mu(1)*Vold(n1,m1,n2,m2) +
mu(2)*Vold(n1,m1,numb2,mumb2)
          end if

        else if(n1==0 .and. n2==0) then

          if(m1==0 .and. m2==0) then
            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                        & + mu(1)*Vold(n1,m1,n2,m2) +
mu(2)*Vold(n1,m1,n2,m2)
          end if
```

```
        end if

      end do
    end do
  end do
end do

smallest = 1000000.0
largest = -1000000.0
Sele = 999
Lele = 999

do n1 = 1,Nmax-BError
  do n2 = 0,Nmax-BError
    do m1 = 1,m(1)

        m2 = 0

        if (smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do

do n1 = 0,Nmax-BError
  do n2 = 1,Nmax-BError
      do m2 = 1,m(2)

        m1 = 0

        if (smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do


n1 = 0
  n2 = 0
    m1 = 0
      m2 = 0

        if (smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

diff = largest - smallest

!open(unit=7,file="GSWdiff.txt")
!write(unit=7,fmt="(3f16.4,8i4)") smallest,largest,diff,Sele,Lele

if (count > 92500) goto 120
if (diff > smallest*TOL .or. diff < 0.0) goto 12

120 WC = (smallest + largest)/2.0
print*,100.0/WC
```

```
!close(unit=7)

print*,"Count = ",count
print*,"The index policy cost this queue setup & parameters is ",WC

return
end subroutine

!-------------------------------------------------------------------------------------------------------

!NB here we have a state space of (n1,m1,n2,m2) where m1 is the
!number of phase completions we have done for class 1, i.e, m1
!starts off at 0 and goes upto m(1)-1, as when the m(1)th phase
!completion is over n1 goes to n1-1 and m1 goes back to 0 as this
!is the start of the service of the next queuing customer. (similarly for
m2)

!this subroutine only looks at virtual possible events occuring, i.e. there
!are events occuring which could not really happen but the effects of
!such events is nothing.

!Now we have an extra state, m=1 is where we have started a service
!but have not finished the first phase of that service.

subroutine LQcosts3(indata,r,Nmax,LC)
implicit none

integer ::
Nmax,BError,n,n1,n2,count,m1,m2,r,num1,num2,mumb1,mumb2,numb1,numb2
integer, dimension(2) :: m,buffer
integer, dimension(4) :: Sele,Lele
double precision :: a,b,d,e,TOL,U,smallest,largest,diff,LC
double precision, dimension(2) :: l,mu
double precision, dimension (200,8) :: indata
double precision, dimension(2,0:Nmax) :: C
double precision, allocatable, dimension(:,:,:,:) :: Vold,Vnew

call qvals(indata,r,a,b,d,e,l,m,mu,Nmax,buffer,BError,TOL)

allocate( Vold(0:Nmax,0:m(1),0:Nmax,0:m(2)) )
allocate( Vnew(0:Nmax,0:m(1),0:Nmax,0:m(2)) )

TOL = 0.0005
C = 0.0

do n=buffer(1),Nmax
  C(1,n) = a*(real(n-buffer(1))**1.0) + b*(real(n-buffer(1))**2.0)
end do

do n=buffer(2),Nmax
  C(2,n) = d*(real(n-buffer(2))**1.0) + e*(real(n-buffer(2))**2.0)
end do

U = l(1) + l(2) + mu(1) + mu(2)
l(1) = l(1)/U
l(2) = l(2)/U
mu(1) = mu(1)/U
mu(2) = mu(2)/U

Vold = 0.0
Vnew = 0.0
count = 0

16 Vold = Vnew

count = count + 1

do n1 = 0,Nmax
  do n2 = 0,Nmax
    do m1 = 0,m(1)
      do m2 = 0,m(2)

        call arrnext(1,Nmax,n1,n2,num1,num2)
```

```
        call arrnext(2,Nmax,n1,n2,num1,num2)
        call sernext(1,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)
        call sernext(2,m,n1,n2,numb1,numb2,m1,m2,mumb1,mumb2)

        if(n1>0 .and. n2>0) then

          if(m1==0 .and. m2==0) then

            if(n1 >= n2) then

              Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,1,n2,0)
+ l(2)*Vold(n1,1,num2,0) &
                                          & + mu(1)*Vold(numb1,mumb1,n2,0) +
mu(2)*Vold(n1,1,n2,0)
            else

              Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,0,n2,1)
+ l(2)*Vold(n1,0,num2,1) &
                                          & + mu(1)*Vold(n1,0,n2,1) +
mu(2)*Vold(n1,0,numb2,mumb2)
            end if

          else if(m1>0 .and. m2==0) then

            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                                        & + mu(1)*Vold(numb1,mumb1,n2,m2) +
mu(2)*Vold(n1,m1,n2,m2)

          else if(m1==0 .and. m2>0) then

            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                                        & + mu(1)*Vold(n1,m1,n2,m2) +
mu(2)*Vold(n1,m1,numb2,mumb2)

          end if

        else if(n1>0 .and. n2==0) then

          if(m1==0 .and. m2==0) then
            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,1,n2,0) +
l(2)*Vold(n1,1,num2,0) &
                                        & + mu(1)*Vold(numb1,mumb1,n2,0) +
mu(2)*Vold(n1,1,n2,0)
          else if(m1>0 .and. m2==0) then
            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                                        & + mu(1)*Vold(numb1,mumb1,n2,m2) +
mu(2)*Vold(n1,m1,n2,m2)
          end if

        else if(n1==0 .and. n2>0) then

          if(m1==0 .and. m2==0) then
            Vnew(n1,m1,n2,m2)= C(1,n1) + C(2,n2) + l(1)*Vold(num1,0,n2,1) +
l(2)*Vold(n1,0,num2,1) &
                          & + mu(1)*Vold(n1,0,n2,1) + mu(2)*Vold(n1,0,numb2,mumb2)
          else if(m1==0 .and. m2>0) then
            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                                        & + mu(1)*Vold(n1,m1,n2,m2) +
mu(2)*Vold(n1,m1,numb2,mumb2)
          end if

        else if(n1==0 .and. n2==0) then

          if(m1==0 .and. m2==0) then
            Vnew(n1,m1,n2,m2) = C(1,n1) + C(2,n2) + l(1)*Vold(num1,m1,n2,m2)
+ l(2)*Vold(n1,m1,num2,m2) &
                                        & + mu(1)*Vold(n1,m1,n2,m2) +
mu(2)*Vold(n1,m1,n2,m2)
          end if

        end if
```

```
      end do
    end do
  end do
end do

smallest = 1000000.0
largest = -1000000.0
Sele = 999
Lele = 999

do n1 = 1,Nmax-BError
  do n2 = 0,Nmax-BError
    do m1 = 1,m(1)

        m2 = 0

        if (smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do

do n1 = 0,Nmax-BError
  do n2 = 1,Nmax-BError
      do m2 = 1,m(2)

        m1 = 0

        if (smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

    end do
  end do
end do


n1 = 0
  n2 = 0
    m1 = 0
      m2 = 0

        if (smallest > Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          smallest = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Sele = (/n1,m1,n2,m2/)
        end if
        if ( largest  < Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2) ) then
          largest  = Vnew(n1,m1,n2,m2) - Vold(n1,m1,n2,m2)
          Lele = (/n1,m1,n2,m2/)
        end if

diff = largest - smallest

!open(unit=7,file="GSLdiff.txt")
!write(unit=7,fmt="(3f16.4,8i4)") smallest,largest,diff,Sele,Lele

if (count > 92500) goto 160
if (diff > smallest*TOL .or. diff < 0.0) goto 16

160 LC = (smallest + largest)/2.0
print*,100.0/LC
```

```
!close(unit=7)

print*,"Count = ",count
print*,"The longest queue policy cost this queue setup & parameters is ",LC

return
end subroutine

!-----------------------------------------------------------------------------
-------------------------------

subroutine qvals(indata,r,a,b,d,e,l,m,mu,Nmax,buffer,BError,TOL)
implicit none

integer :: Nmax,BError,r
integer, dimension(2) :: m,buffer
double precision, dimension(2) :: mu,l
double precision :: a,b,d,e,TOL
double precision, dimension (200,8) :: indata

Nmax = 69
buffer(1) = 2
buffer(2) = 2
BError = 4

m(1) = 2
m(2) = 3

mu(1) = indata(r,1)
mu(2) = indata(r,2)
l(1) = indata(r,3)
l(2) = ( indata(r,4) - (m(1)*l(1)/mu(1)) )*mu(2)/m(2)
!print*,"l2 = ",l(2)
!traffic intensity = 2.0*l(1)/mu(1) + 3.0*l(2)/mu(2)

a = indata(r,5)
b = indata(r,7)
d = indata(r,6)
e = indata(r,8)

!b = 0.0
!e = 0.0

TOL = 0.0001

return
end subroutine
!-----------------------------------------------------------------------------
-----------------------
```

Appendix F

This appendix contains the Fortran 95 code for the programme we used to simulate the undiscounted service control costs as in Section 3.5.3. Here we consider the index policy for a 5 class system compared to some other standard policies as explained in the numerical section.

```
program simulation
implicit none

!a program to simulate a 5 customer class system in order to calc ave. cost

integer :: size,k,count,Nmax,num,r,BError,TLactsize,numsim,simnumb,i,temp
integer, dimension(3) :: seed
integer, dimension(500000) :: TL1
integer, dimension(5) :: buffer
integer, dimension(0:5) :: m
double precision ::
Tsize,TOL,SUMINDEXC,INDEXC,SUMINDEXSQ,INDEXVAR,AIcost,LONGQC,LQcost,MYOPICC,
MYcost,CMEWC,CMcost,STATICC,STcost,x
double precision ::
SUMLONGQC,SUMLONGQSQ,LONGQVAR,SUMCMEWC,SUMCMEWSQ,CMEWVAR, &
                    &
SUMSTATICC,SUMSTATICSQ,STATICVAR,SUMMYOPICC,SUMMYOPICSQ,MYOPICVAR,in2stat
double precision, dimension(0:5) :: l,mu
double precision, dimension(5) :: a,b
double precision, dimension(500000) :: TL2
double precision, dimension(5,100000) :: IA,AA
double precision, allocatable, dimension(:,:) :: C,W,pi

r = 1
call qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

allocate( C(5,0:Nmax) )
allocate( W(5,0:Nmax) )
allocate( pi(5,0:Nmax) )

in2stat = Tsize*0.667

numsim = 70
open(unit=7,file="Simulationgam2data.dat")

!seed = (/29708,29005,30503/)
  TLactsize = 0
  IA = 0.0
  AA = 0.0
  TL1 = 0
  TL2 = 0.0

call random_number(x)
temp = 10 + int(10.0*x)
temp = 15
do simnumb = 1,temp
  call getarrivals(r,seed,Nmax,TLactsize,IA,AA,TL1,TL2)
!    print*,"end part ",simnumb
end do

print*,"end first"

do r=4,4

  call qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

  do k = 1,5
    do num=buffer(k),Nmax
      C(k,num) = a(k)*(real(num-buffer(k))**3.0) +
b(k)*(real(num-buffer(k))**4.0)
    end do
  end do

  call qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)
  W = 0.0
  !call Windex(r,Nmax,W)
  call Windex1(r,Nmax,pi)
  call Windex2(r,Nmax,pi,W)

  SUMINDEXC = 0.0
  SUMINDEXSQ = 0.0
  INDEXVAR = 0.0
  INDEXC = 0.0

  SUMLONGQC = 0.0
```

```
  SUMLONGQSQ = 0.0
  LONGQVAR = 0.0
  LONGQC = 0.0

  SUMMYOPICC = 0.0
  SUMMYOPICSQ = 0.0
  MYOPICVAR = 0.0
  MYOPICC = 0.0

  SUMSTATICC = 0.0
  SUMSTATICSQ = 0.0
  STATICVAR = 0.0
  STATICC = 0.0

  SUMCMEWC = 0.0
  SUMCMEWSQ = 0.0
  CMEWVAR = 0.0
  CMEWC = 0.0


!  open(unit=7,file="temp.dat")
  print*,"Index Policy"
  write(unit=7,fmt="(a)") "INDEX POLICY"
  do simnumb = 1,numsim
!     print*,"number = ",simnumb
    call getarrivals(r,seed,Nmax,TLactsize,IA,AA,TL1,TL2)
!    write(unit=7,fmt="(a,1500i4)") "TL1 = ",TL1(1500:3000)
!    write(unit=7,fmt="(a)") " "
!    write(unit=7,fmt="(a,1500f20.4)") "TL2 = ",TL2(1500:3000)
    call indexcost(r,seed,Nmax,TLactsize,TL1,TL2,W,AIcost)
    SUMINDEXSQ = SUMINDEXSQ + AICOST**2.0
    SUMINDEXC  = SUMINDEXC + AIcost
!    write(unit=7,fmt="(a,f12.6)") "INDEXC = ",AICOST
  end do
  INDEXVAR = (SUMINDEXSQ -
(real(numsim)*((SUMINDEXC/real(numsim))**2.0)))/(real(numsim-1))
  !(SUMINDEXSQ/real(numsim)) - ((SUMINDEXC/real(numsim))**2.0)
  INDEXC = SUMINDEXC/real(numsim)
  print*,"Finished INDEXC = ",INDEXC
!  write(unit=7,fmt="(a,f12.6)") "Finished INDEXC = ",INDEXC
!  write(unit=7,fmt="(a,f12.6)") "Finished INDEXVAR = ",INDEXVAR
!  close(unit = 7)

  print*,"Longest Queue"
  do simnumb = 1,numsim
!     print*,"number = ",simnumb
    call getarrivals(r,seed,Nmax,TLactsize,IA,AA,TL1,TL2)
    call longestq(r,seed,Nmax,TLactsize,TL1,TL2,LQcost)
    SUMLONGQSQ = SUMLONGQSQ + (LQCOST**2.0)
    SUMLONGQC  = SUMLONGQC + LQcost
  end do
  LONGQVAR = (SUMLONGQSQ -
(real(numsim)*((SUMLONGQC/real(numsim))**2.0)))/(real(numsim-1))
  !(SUMLONGQSQ/real(numsim)) - ((SUMLONGQC/real(numsim))**2.0)
  LONGQC = SUMLONGQC/real(numsim)
  print*,"Finished LONGQ = ",LONGQC

  print*,"C Mew Rule"
  do simnumb = 1,numsim
!     print*,"number = ",simnumb
    call getarrivals(r,seed,Nmax,TLactsize,IA,AA,TL1,TL2)
    call cmew(r,seed,Nmax,TLactsize,TL1,TL2,CMcost)
    SUMCMEWSQ = SUMCMEWSQ + (CMCOST**2.0)
    SUMCMEWC = SUMCMEWC + CMcost
  end do
  CMEWVAR = (SUMCMEWSQ -
(real(numsim)*((SUMCMEWC/real(numsim))**2.0)))/(real(numsim-1))
  !(SUMCMEWSQ/real(numsim)) - ((SUMCMEWC/real(numsim))**2.0)
  CMEWC = SUMCMEWC/real(numsim)
  print*,"Finished CMEWC = ",CMEWC

  print*,"Static Policy"
  do simnumb = 1,numsim
!     print*,"number = ",simnumb
    call getarrivals(r,seed,Nmax,TLactsize,IA,AA,TL1,TL2)
```

```
    call static(r,seed,Nmax,TLactsize,TL1,TL2,STcost)
    SUMSTATICSQ = SUMSTATICSQ + (STCOST**2.0)
    SUMSTATICC  = SUMSTATICC + STcost
  end do
  STATICVAR = (SUMSTATICSQ -
(real(numsim)*((SUMSTATICC/real(numsim))**2.0)))/(real(numsim-1))
  !(SUMSTATICSQ/real(numsim)) - ((SUMSTATICC/real(numsim))**2.0)
  STATICC = SUMSTATICC/real(numsim)
  print*,"Finished STATICC = ",STATICC

  print*,"Myopic Policy"
  do simnumb = 1,numsim
!     print*,"number = ",simnumb
    call getarrivals(r,seed,Nmax,TLactsize,IA,AA,TL1,TL2)
    call myopic(r,seed,Nmax,TLactsize,TL1,TL2,MYcost)
    SUMMYOPICSQ = SUMMYOPICSQ + (MYCOST**2.0)
    SUMMYOPICC  = SUMMYOPICC + MYcost
  end do
  MYOPICVAR = (SUMMYOPICSQ -
(real(numsim)*((SUMMYOPICC/real(numsim))**2.0)))/(real(numsim-1))
  !(SUMMYOPICSQ/real(numsim)) - ((SUMMYOPICC/real(numsim))**2.0)
  MYOPICC = SUMMYOPICC/real(numsim)
  print*,"Finished MYOPICC = ",MYOPICC

  write(unit=7,fmt="(a,i6)")"# simulations       = ",numsim
  write(unit=7,fmt="(a,5f10.6)")"a cost vector       = ",a
  write(unit=7,fmt="(a,5f10.6)")"b cost vector       = ",b
  write(unit=7,fmt="(a,6f10.6)")"arrivals vector     = ",l
  write(unit=7,fmt="(a,6f10.6)")"service time vector = ",mu
  write(unit=7,fmt="(a,f10.4,a,f10.4)") "Tsize = ",Tsize,"     in2stat =
",in2stat
  write(unit=7,fmt="(a,i8,a,i5)") "Nmax = ",Nmax,"     numsim = ",numsim
  write(unit=7,fmt="(a)") "******** INDEX ********"
  write(unit=7,fmt="(a,f19.12)") "COST = ",INDEXC
  write(unit=7,fmt="(a,f19.12)")   "SUB INDEX  =
",100.0*(INDEXC-INDEXC)/INDEXC
  write(unit=7,fmt="(a,f19.12)")   "Sample Mean S.D. =
",sqrt(INDEXVAR/numsim)
  write(unit=7,fmt="(a)") " "
  write(unit=7,fmt="(a)") "******** LONGEST QUEUE ********"
  write(unit=7,fmt="(a,f19.12)") "COST = ",LONGQC
  write(unit=7,fmt="(a,f19.12)")   "SUB INDEX  =
",100.0*(LONGQC-INDEXC)/INDEXC
  write(unit=7,fmt="(a,f19.12)")   "Sample Mean S.D. =
",sqrt(LONGQVAR/numsim)
  write(unit=7,fmt="(a)") " "
  write(unit=7,fmt="(a)") "******** CMEW ********"
  write(unit=7,fmt="(a,f19.12)") "COST = ",CMEWC
  write(unit=7,fmt="(a,f19.12)")   "SUB INDEX  =
",100.0*(CMEWC-INDEXC)/INDEXC
  write(unit=7,fmt="(a,f19.12)")   "Sample Mean S.D. =
",sqrt(CMEWVAR/numsim)
  write(unit=7,fmt="(a)") " "
  write(unit=7,fmt="(a)") "******** MYOPIC ********"
  write(unit=7,fmt="(a,f19.12)") "COST = ",MYOPICC
  write(unit=7,fmt="(a,f19.12)")   "SUB INDEX  =
",100.0*(MYOPICC-INDEXC)/INDEXC
  write(unit=7,fmt="(a,f19.12)")   "Sample Mean S.D. =
",sqrt(MYOPICVAR/numsim)
  write(unit=7,fmt="(a)") " "
  write(unit=7,fmt="(a)") "******** STATIC ********"
  write(unit=7,fmt="(a,f19.12)") "COST = ",STATICC
  write(unit=7,fmt="(a,f19.12)")   "SUB INDEX  =
",100.0*(STATICC-INDEXC)/INDEXC
  write(unit=7,fmt="(a,f19.12)")   "Sample Mean S.D. =
",sqrt(STATICVAR/numsim)
  write(unit=7,fmt="(a)") " "

!  write(unit=7,fmt="(a,f16.12,a,f16.12,a,f16.12)") "LONGEST Q : COST =
",LONGQC," : SUB INDEX = ",100.0*(LONGQC-INDEXC)/INDEXC,"Sample Error = ",
!  write(unit=7,fmt="(a,f16.12,a,f16.12,a,f16.12)") "CMEW      : COST =
",CMEWC," : SUB INDEX = ",100.0*(CMEWC-INDEXC)/INDEXC,"Sample Error = ",
!  write(unit=7,fmt="(a,f16.12,a,f16.12,a,f16.12)") "MYOPIC    : COST =
",MYOPICC," : SUB INDEX = ",100.0*(MYOPICC-INDEXC)/INDEXC,"Sample Error = ",
!  write(unit=7,fmt="(a,f16.12,a,f16.12,a,f16.12)") "STATIC    : COST =
```

```
",STATICC," : SUB INDEX = ",100.0*(STATICC-INDEXC)/INDEXC,"Sample Error = ",
!  write(unit=7,fmt="(a)") " "
!
end do

close(unit=7)

end program

!-----------------------------------------------------------------------------
-----

subroutine uniform(seed,Low,Upp,x)
implicit none
integer, dimension(3) :: seed
double precision :: r,s,x,Low,Upp

seed(1) = mod(171*seed(1),30269)
seed(2) = mod(172*seed(2),30307)
seed(3) = mod(170*seed(3),30323)

s = seed(1)*1d0/30269 + seed(2)*1d0/30307 + seed(3)*1d0/30323
r = s - int(s)

x = Low + (Upp-Low)*r


return
end subroutine

!-----------------------------------------------------------------------------
----------------------

subroutine factorial(z,fact)
implicit none

integer :: z,i
double precision :: fact,tot

tot = 1

if (z > 0) then

  do i = 1,z

    tot = tot*real(i)

  end do

  fact = tot

else if (z == 0) then

  fact = 1.0

else

  print*,"ERROR cannot find factorial of negative number"
  fact = 0.0

end if

return
end subroutine

!-----------------------------------------------------------------------------
------------------------------

subroutine Windex1(r,Nmax,pi)
implicit none

integer :: Nmax,BError,j,i,n,k,r,size,STATFAIL,h,nummatmul
integer, dimension(5) :: buffer
integer, dimension(0:5) :: m
double precision :: TOL,temp1,temp2,temp3,Tsize,upp,low,Psum
```

```
double precision, dimension(5) :: a,b
double precision, dimension(0:5) :: l,mu

double precision, dimension(5,0:Nmax) :: Delta,pi
double precision, dimension(0:Nmax,0:Nmax,5) :: P

call qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

!calculate the markov chain transition matrix

Delta = 0.0
STATFAIL = 0
nummatmul = 0

do k = 1,5

  call factorial(m(k)-1,temp3)

  do j = 0,Nmax

    call factorial(m(k)+j-1,temp1)
    call factorial(j,temp2)

    Delta(k,j) = ( temp1/(temp2*temp3) )*( (l(k)/(l(k)+mu(k)))**(real(j))
)*( (mu(k)/(l(k)+mu(k)))**(real(m(k))) )
!     Delta(k,j) =  ((l(k)*mu(k))**real(j))*(exp(-1.0*l(k)*mu(k)))/temp2

  end do
print*,"delta(",k,",0) = ",delta(k,0)
  P(:,:,k) = 0.0

  do i = 0,Nmax

    P(0,i,k) = Delta(k,i)

  end do
print*,"P(",k,",0,0) = ",P(0,0,k)
  P(1,:,k) = P(0,:,k)

  do j = 1,Nmax-1
    do i = j,Nmax

      P(j+1,i,k) = Delta(k,i-j)

    end do
  end do

end do

!renormalize to ensure that sum of probabilities = 1
do k = 1,5
  do i = 0,Nmax
    Psum = 0.0
    do j = 0,Nmax
      Psum = Psum + P(i,j,k)
    end do
    do j = 0,Nmax
      P(i,j,k) = P(i,j,k)/Psum
    end do
  end do
end do

!calculate the state probabilities - pi(k,j)
pi = 0.0

!AM = P(1,:,:)
!BM = P(2,:,:)

do k = 1,5
  nummatmul = 0
  75 STATFAIL = 0
  nummatmul = nummatmul + 1
  P(:,:,k) = matmul(P(:,:,k),P(:,:,k))
  !renormalize to ensure that sum of probabilities = 1
  do i = 0,Nmax
```

```
    Psum = 0.0
    do j = 0,Nmax
      Psum = Psum + P(i,j,k)
    end do
    do j = 0,Nmax
      P(i,j,k) = P(i,j,k)/Psum
    end do
  end do

    !check that all rows of P are the same - i.e. stat distn
  do h=0,Nmax
    upp = P(0,h,k) + 0.00005
    low = P(0,h,k) - 0.00005
    do j = 0,Nmax
      if (P(j,h,k) > upp .or. P(j,h,k) < low) then
!         print*,"ERROR: problem with stat distn",k,j,h
        STATFAIL = 1
      end if
    end do
  end do
  if (nummatmul >= 40) goto 80
  if (STATFAIL == 1) goto 75
  80 print*,"nummatmul ",k," = ",nummatmul
  print*,"P(5,0,",k,") = ",P(5,0,k)
  print*,"P(,0,5",k,") = ",P(0,5,k)
end do

STATFAIL = 0

do k = 1,5
  !check that all rows of P are the same - i.e. stat distn
  do i=0,Nmax-3
    upp = P(0,i,k) + 0.00005
    low = P(0,i,k) - 0.00005
    do j = 0,Nmax-3
      if (P(j,i,k) > upp .or. P(j,i,k) < low) then
!         print*,"ERROR: problem with stat distn",j,i,k
        STATFAIL = 1
      end if
    end do
  end do

end do

do k = 1,5
  pi(k,:) = P(0,:,k)
!pi(2,:) = BM(0,:)
end do

!open (unit=7,file="GSstatdistmat.dat")
!write(unit=7,fmt="(a)") "    b1    :    c1    :    b2    :    c2    : Nmax"
!write(unit=7,fmt="(4f10.4,i5)") a(1),b(1),a(2),b(2),Nmax
!write(unit=7,fmt="(a)") "    l1    :    l2    :    m1 : mu1    :    m2 : mu2
"
!write(unit=7,fmt="(2f10.4,i5,f10.4,i5,f10.4)")
l(1),l(2),m(1),mu(1),m(2),mu(2)
!write(unit=7,fmt="(a)") "The stationary distn is:"
!do k = 1,2
!  write(unit=7,fmt="(a)") " "
!  write(unit=7,fmt="(a,i4)") "class = ",k
!  do i = 0,Nmax
!    write(unit=7,fmt="(70f12.6)") P(:,i,k)
!  end do
!end do
!
!do i = 1,5
!   write(unit=7,fmt="(a,i4)") "class = ",i
!  do n = 0,Nmax
!    write(unit=7,fmt="(a,i4,a,i5,a,f16.12)") "pi(",i,",",n,")= ",pi(i,n)
!  end do
!end do
!close(unit=7)
!
return
end subroutine
```

```
!-------------------------------------------------------------------------
-------------------------------

subroutine Windex1old(r,Nmax,pi)
implicit none

integer :: Nmax,BError,j,i,n,k,r,size
integer, dimension(5) :: buffer
integer, dimension(0:5) :: m
double precision :: TOL,temp1,temp2,temp3,Tsize
double precision, dimension(5) :: a,b
double precision, dimension(0:5) :: l,mu

double precision, dimension(5,0:Nmax) :: Delta,pi
double precision, dimension(5,0:Nmax,0:Nmax) :: P

call qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

!calculate the markov chain transition matrix

Delta = 0.0

do k = 1,5

  call factorial(m(k)-1,temp3)

  do j = 0,Nmax

    call factorial(m(k)+j-1,temp1)
    call factorial(j,temp2)

   Delta(k,j) = ( temp1/(temp2*temp3) )*( (l(k)/(l(k)+mu(k)))**(real(j)) )*(
(mu(k)/(l(k)+mu(k)))**(real(m(k))) )
!    Delta(k,j) =  ((mu(k)*l(k))**j)*(exp(-1.0*mu(k)*l(k)))/temp2

  end do

  P(k,:,:) = 0.0

  do i = 0,Nmax

    P(k,0,i) = Delta(k,i)

  end do

  P(k,1,:) = P(k,0,:)

  do j = 1,Nmax-1
    do i = j,Nmax

      P(k,j+1,i) = Delta(k,i-j)

    end do
  end do

end do

!calculate the state probabilities - pi(k,j)
pi = 0.0

!AM = P(1,:,:)
!BM = P(2,:,:)

do k = 1,5
  do i = 1,20
    P(k,:,:) = matmul(P(k,:,:),P(k,:,:))
!  BM = matmul(BM,BM)
  end do
end do

do k = 1,5
  pi(k,:) = P(k,0,:)
!pi(2,:) = BM(0,:)
end do
```

```
!open (unit=7,file="GSstatdistmat.dat")
!write(unit=7,fmt="(a)") "    b1    :    c1    :    b2    :    c2    : Nmax"
!write(unit=7,fmt="(4f10.4,i5)") a(1),b(1),a(2),b(2),Nmax
!write(unit=7,fmt="(a)") "    l1    :    l2    :    m1 : mu1    :    m2 : mu2
"
!write(unit=7,fmt="(2f10.4,i5,f10.4,i5,f10.4)")
l(1),l(2),m(1),mu(1),m(2),mu(2)
!write(unit=7,fmt="(a)") "The stationary distn is:"
!do k = 1,2
!  write(unit=7,fmt="(a)") " "
!  write(unit=7,fmt="(a,i4)") "class = ",k
!  do i = 0,Nmax
!    write(unit=7,fmt="(101f12.6)") P(k,i,:)
!  end do
!end do

!do i = 1,5
!   write(unit=7,fmt="(a,i4)") "class = ",i
!  do n = 0,Nmax
!    write(unit=7,fmt="(a,i4,a,i5,a,f16.12)") "pi(",i,",",n,")= ",pi(i,n)
!  end do
!end do
!close(unit=7)

return
end subroutine
!--------------------------------------------------------------------------
------------------------------

subroutine Windex2(r,Nmax,pi,W)
implicit none

integer :: Nmax,BError,j,n,k,r,size,num
integer, dimension(5) :: buffer
integer, dimension(0:5) :: m
double precision :: TOL,Tsize
double precision, dimension(5) :: a,b
double precision, dimension(0:5) :: l,mu

double precision, dimension(5,0:Nmax) :: pi
double precision, dimension(5,0:Nmax) :: W,EC
double precision, dimension(5,0:Nmax+Nmax) :: C

call qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

!calculate the markov chain transition matrix
W = 0.0
C = 0.0
EC = 0.0

do k = 1,5

  do num=buffer(1),Nmax+Nmax
    C(k,num) = a(k)*(real(num-buffer(k))**3.0) +
b(k)*(real(num-buffer(k))**4.0)
  end do

  do n = 0,Nmax
    do j = 0,Nmax

      EC(k,n) = EC(k,n) + C(k,n+j)*pi(k,j)

!       if (n >= 32) then
!         open (unit=7,file="forming.dat")
!         write(unit=7,fmt="(3i4,3f16.4)") k,n,j,EC(k,n),C(k,n+j),pi(k,j)
!       end if

    end do
  end do

  !calculate the actual index

!  open (unit=7,file="forming1.dat")
```

```
  do n = 1,Nmax

!     write(unit=7,fmt="(2i4,3f16.4)") k,n,EC(k,n)
    if (mu(k) < 999999.999999) W(k,n) = ( EC(k,n) - EC(k,n-1) )/(m(k)/mu(k))

  end do

  W(k,0) = 0.0

end do
!close(unit=7)

!open (unit=7,file="Indices2.dat")
!write(unit=7,fmt="(a)") "The indices are:"
!do k = 0,Nmax
!write(unit=7,fmt="(5f16.4)") W(:,k)
!end do
!close(unit=7)

return
end subroutine

!-----------------------------------------------------------------------------
----------------------

subroutine getarrivals(r,seed,Nmax,TLactsize,IA,AA,TL1,TL2)
implicit none

integer :: size,sclass,k,col,count,Nmax,r,BError,TLactsize
integer, dimension(3) :: seed
integer, dimension(500000) :: TL1
integer, dimension(5) :: buffer
integer, dimension(0:5) :: numarr,m
double precision :: x,smallest,smallestold,ssum,Tsize,actsize,TOL
double precision, dimension(0:5) :: l,mu
double precision, dimension(5) :: a,b
double precision, dimension(500000) :: TL2
double precision, dimension(5,100000) :: IA,AA

call qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

IA = 0.0
AA = 0.0
TL1 = 0
TL2 = 0.0
numarr = 0

count = 0


do k = 1,10
  call random_number(x)
end do

10 count = count + 1
  !print*,""
  ssum = 99999.99
do k = 1,5
  call random_number(x)
  if (l(k) > 0.00001) IA(k,count) = -1.0*log(x)/l(k)
  if (count == 1) then
    AA(k,count) = IA(k,count)
  else
    AA(k,count) = AA(k,count-1) + IA(k,count)
  end if
  if (ssum > AA(k,count) .and. l(k) > 0.0000001) ssum = AA(k,count)
end do
!print*,"ssum = ",ssum
if (ssum < Tsize .and. count < size) goto 10

if (count >= size) print*,"ERROR: Need bigger matrices & to simulate more
values"

actsize = count
```

```
do k = 1,5
  do col = 1,actsize
    if ( AA(k,col) < Tsize .and. l(k) > 0.0000001) numarr(k) = col
  end do
end do

!print*,"numarr = ",numarr

TLactsize = 0
do k = 1,5
  TLactsize = TLactsize + numarr(k)
end do

smallestold = -99999999.99
do count = 1,TLactsize+5
  if (count > 1) smallestold = smallest
  smallest = 999999999.99
  do col = 1,actsize
    do k = 1,5
      if (smallest > AA(k,col) .and. AA(k,col) > smallestold .and. l(k) >
0.000001) then
        smallest = AA(k,col)
        sclass = k
      end if
    end do
  end do
  TL1(count) = sclass
  TL2(count) = smallest
end do

!open(unit=7,file="simdata2.dat")
!write(unit=7,fmt="(a)") "IA = "
!do k=1,5
!  write(unit=7,fmt="(50f12.6)") IA(k,1:500)
!end do
!print*," "

!write(unit=7,fmt="(a)") "AA = "
!do k=1,5
!  write(unit=7,fmt="(50f12.6)") AA(k,1:500)
!end do
!print*," "

!write(unit=7,fmt="(a)") "TL1 = "
!write(unit=7,fmt="(100i4)") TL1(:)
!
!write(unit=7,fmt="(a)") "TL2 = "
!write(unit=7,fmt="(100f12.6)") TL2(1:100)
!
!print*," "
!!print*,"csize  = ",csize
!print*,"actsize = ",actsize
!print*,"TLactsize = ",TLactsize
!close(unit=7)
!
return
end subroutine

!-----------------------------------------------------------------------------
-----------------------

subroutine indexcost(r,seed,Nmax,TLactsize,TL1,TL2,W,AIcost)
implicit none

integer :: size,k,count,custserve,event,state,Nmax,num,r,BError, &
           & i,TLactsize,marker,custserveold,arrivalold
integer, dimension(3) :: seed
integer, dimension(500000) :: TL1
integer, dimension(5) :: buffer
integer, dimension(0:5) :: n,numserved,m
double precision :: Tsize,Tservice,stable,in2stat,TTservice
double precision :: lastevent,endserve,NEtime,bigind,Tcost,AIcost,TOL
double precision, dimension(0:5) :: l,mu
double precision, dimension(5) :: a,b
double precision, dimension(500000) :: TL2
```

```
!double precision, dimension(5,100000) :: IA,AA
double precision, dimension(5,0:Nmax) :: C,W

call qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)
TTservice = 0.0
Tservice = 0.0
marker = 0
numserved = 0
in2stat = Tsize*0.667

AIcost = 0.0

!test to ensure that we have stable queues
stable = 0.0
do k = 1,5
  if (mu(k) < 999999.999999) stable = stable + l(k)*m(k)/mu(k)
end do
if (stable >= 1.0) print*,"ERROR: UNSTABLE SYSTEM!!!"

do k = 1,5
  do num=buffer(k),Nmax
    C(k,num) = a(k)*(real(num-buffer(k))**3.0) +
b(k)*(real(num-buffer(k))**4.0)
  end do
end do

Tcost = 0.0
n = 0
TTservice = 0.0

event = 1
n(TL1(1)) = n(TL1(1)) + 1
lastevent = TL2(1)

custserve = TL1(1)
call gammaservice(r,custserve,Tservice)
endserve = lastevent + Tservice

if (lastevent > in2stat) then
  marker = 1
end if

!open(unit=7,file="SimIndexCostR.dat")
!write(unit=7,fmt="(a)") "       W(1,:)  :  W(2,:)  :  W(3,:)  :  W(4,:)  :
W(5,:)  "
!do i = 0,Nmax
!  write(unit=7,fmt="(i4,5f12.6)") i,W(:,i)
!end do
!close(unit=7)


!open(unit=7,file="simresultsIN.txt")!"serviceIn.txt")

!  write(unit=7,fmt="(a,i5)") "event # = ",event
!  write(unit=7,fmt="(a,i5)") "custserve class = ",custserve
!  write(unit=7,fmt="(a,f12.6)") "endserve = ",endserve
!  write(unit=7,fmt="(a,f12.6)") "Tservice = ",Tservice
!  write(unit=7,fmt="(a,f12.6)") "NEtime = ",NEtime
!  write(unit=7,fmt="(a,6i5)") "state = ",n
!  write(unit=7,fmt="(a,f12.6)") "TL2 = ",TL2(event)
!  write(unit=7,fmt="(a,f20.3)") "Tcost = ",Tcost
!  write(unit=7,fmt="(a,f12.6)") "Tcost/size = ",Tcost/(Lastevent-in2stat)
!  write(unit=7,fmt="(a)") " "

event = 2
20  custserveold = -1
    arrivalold  = -1

  state = 0
  do k = 1,5
    state = state + n(k)
  end do

  if(TL2(event) < endserve .or. state == 0) then
    if(TL2(event) < Tsize) then
```

```
      NEtime = TL2(event) - lastevent
      lastevent = TL2(event)
    else
      NEtime = Tsize - lastevent
      lastevent = Tsize
    end if

    if ( state == 0 ) then
      custserve = TL1(event)
      call gammaservice(r,custserve,Tservice)
      endserve = lastevent + Tservice

!       write(unit=7,fmt="(a,f12.6)") "Tservice = ",Tservice !!
      numserved(custserve) = numserved(custserve) + 1
    end if

    arrivalold = TL1(event)
    event = event + 1

  else
    if(endserve < Tsize) then
      NEtime = endserve - lastevent
      lastevent = endserve
    else
      NEtime = Tsize - lastevent
      lastevent = Tsize
    end if
    custserveold = custserve
    n(custserve) = n(custserve) - 1
    state = 0
    do k = 1,5
      state = state + n(k)
    end do
    bigind = -9999.99
    do k = 1,5
      if (n(k) > 0) then
        if (bigind < W(k,n(k))) then
          bigind = W(k,n(k))
          custserve = k
        end if
      end if
    end do
    if (state == 0) custserve = 0

    call gammaservice(r,custserve,Tservice)
    if (state > 0) endserve = lastevent + Tservice

!     write(unit=7,fmt="(a,f12.6)") "Tservice = ",Tservice  !!
    n(custserveold) = n(custserveold) + 1
    numserved(custserve) = numserved(custserve) + 1
  end if

  if (lastevent > in2stat) then
    if (marker == 0) then
      NEtime = lastevent - in2stat
    end if
    marker = 1
  end if

  do k = 1,5
    Tcost = Tcost + C(k,n(k))*NEtime*real(marker)
  end do

  !!!!!!!!!
  if (arrivalold > 0 .and. n(arrivalold) < Nmax) n(arrivalold) =
n(arrivalold) + 1
  if (custserveold >= 0 .and. n(custserveold) >= 1) n(custserveold) =
n(custserveold) - 1
  state = 0
  do k = 1,5
    state = state + n(k)
  end do

  TTservice = TTservice + Tservice
```

```
!  write(unit=7,fmt="(a,i5)") "event # = ",event
!  write(unit=7,fmt="(a,i5)") "custserve class = ",custserve
!  write(unit=7,fmt="(a,f12.6)") "endserve = ",endserve
!  write(unit=7,fmt="(a,f12.6)") "Tservice = ",Tservice
!  write(unit=7,fmt="(a,f12.6)") "NEtime = ",NEtime
!  write(unit=7,fmt="(a,6i5)") "state = ",n
!  write(unit=7,fmt="(a,f12.6)") "TL2 = ",TL2(event)
!  write(unit=7,fmt="(a,f20.3)") "Tcost = ",Tcost
!  write(unit=7,fmt="(a,f12.6)") "Tcost/size = ",Tcost/(Lastevent-in2stat)
!  write(unit=7,fmt="(a)") " "

if (event < TLactsize) goto 20
!write(unit=7,fmt="(a,f12.6)") "TTservice = ",TTservice
!write(unit=7,fmt="(a,6i6)") "numserved = ",numserved
!print*,"Tcost = ",Tcost
!print*,"Tsize = ",Tsize
!print*,"in2stat = ",in2stat
AIcost = Tcost/(Tsize-in2stat)
!print*,"INDEX: average costs = ",Acost
!print*,"stable = ",stable
!close(unit=7)

return
end subroutine

!--------------------------------------------------------------------------
----------------------

subroutine longestq(r,seed,Nmax,TLactsize,TL1,TL2,LQcost)
implicit none

integer :: size,k,count,custserve,event,state,Nmax,num,r,BError, &
            & i,TLactsize,bigind,marker,custserveold,arrivalold
integer, dimension(3) :: seed
integer, dimension(500000) :: TL1
integer, dimension(5) :: buffer
integer, dimension(0:5) :: n,numserved,m
double precision :: x,Tsize,Tservice,stable,in2stat,TTservice
double precision :: lastevent,endserve,NEtime,Tcost,LQcost,TOL
double precision, dimension(0:5) :: l,mu
double precision, dimension(5) :: a,b
double precision, dimension(500000) :: TL2
double precision, dimension(5,0:Nmax) :: C

call qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)
TTservice = 0.0
marker = 0
numserved = 0
in2stat = Tsize*0.667

!test to ensure that we have stable queues
stable = 0.0
do k = 1,5
  if (mu(k) < 999999.999999) stable = stable + l(k)*m(k)/mu(k)
end do
if (stable >= 1.0) print*,"ERROR: UNSTABLE SYSTEM!!!"

do k = 1,5
  do num=buffer(k),Nmax
    C(k,num) = a(k)*(real(num-buffer(k))**3.0) +
b(k)*(real(num-buffer(k))**4.0)
  end do
end do

do i = 1,10
  call random_number(x)
end do

Tcost = 0.0
n = 0

n(TL1(1)) = n(TL1(1)) + 1
lastevent = TL2(1)

custserve = TL1(1)
```

```
call gammaservice(r,custserve,Tservice)
endserve = lastevent + Tservice

if (lastevent > in2stat) then
  marker = 1
end if

!do k = 1,5
!  Tcost = Tcost + C(k,n(k))*NEtime
!end do

!open(unit=7,file="SimIndexCostR.dat")
!write(unit=7,fmt="(a)") "  W(1,:)  :  W(2,:)  :  W(3,:)  :  W(4,:)  :
W(5,:)  "
!do i = 0,Nmax
!  write(unit=7,fmt="(i4,5f12.6)") i,W(:,i)
!end do
!close(unit=7)

!open(unit=7,file="simresultsLQ.txt")

event = 2
20  custserveold = -1
    arrivalold  = -1

  state = 0
  do k = 1,5
    state = state + n(k)
  end do
  if(TL2(event) < endserve .or. state == 0) then
    if(TL2(event) < Tsize) then
      NEtime = TL2(event) - lastevent
      lastevent = TL2(event)
    else
      NEtime = Tsize - lastevent
      lastevent = Tsize
    end if

    if(state == 0) then
      custserve = TL1(event)
      call gammaservice(r,custserve,Tservice)
      endserve = lastevent + Tservice
      numserved(custserve) = numserved(custserve) + 1
    end if

    arrivalold = TL1(event)
    event = event + 1

  else
    if(endserve < Tsize) then
      NEtime = endserve - lastevent
      lastevent = endserve
    else
      NEtime = Tsize - lastevent
      lastevent = Tsize
    end if
    custserveold = custserve
    n(custserve) = n(custserve) - 1
    state = 0
    do k = 1,5
      state = state + n(k)
    end do
    bigind = -999
    do k = 1,5
      if (n(k) > 0) then
        if (bigind < n(k)) then
          bigind = n(k)
          custserve = k
        end if
      end if
    end do
    if (state == 0) custserve = 0
    call gammaservice(r,custserve,Tservice)
    endserve = lastevent + Tservice
```

```
    n(custserveold) = n(custserveold) + 1
    numserved(custserve) = numserved(custserve) + 1

  end if

  if (lastevent > in2stat) then
    if (marker == 0) then
      NEtime = lastevent - in2stat
    end if
    marker = 1
  end if

  do k = 1,5
    Tcost = Tcost + C(k,n(k))*NEtime*real(marker)
  end do

  if (arrivalold > 0 .and. n(arrivalold) < Nmax) n(arrivalold) =
n(arrivalold) + 1
  if (custserveold >= 0 .and. n(custserveold) >= 1) n(custserveold) =
n(custserveold) - 1
  state = 0
  do k = 1,5
    state = state + n(k)
  end do

  TTservice = TTservice + Tservice

!  write(unit=7,fmt="(a,i5)") "event # = ",event
!  write(unit=7,fmt="(a,f12.6)") "Tservice = ",Tservice
!  write(unit=7,fmt="(a,f12.6)") "endserve = ",endserve
!  write(unit=7,fmt="(a,f12.6)") "NEtime = ",NEtime
!  write(unit=7,fmt="(a,6i5)") "state = ",n
!  write(unit=7,fmt="(a,f12.6)") "TL2 = ",TL2(event)
!  write(unit=7,fmt="(a,f20.3)") "Tcost = ",Tcost
!  write(unit=7,fmt="(a,f12.6)") "Tcost/size =
",Tcost/(Lastevent+NEtime-in2stat)
!  write(unit=7,fmt="(a)") " "

if (event < TLactsize) goto 20
!write(unit=7,fmt="(a,f12.6)") "TTservice = ",TTservice
!write(unit=7,fmt="(a,6i6)") "numserved = ",numserved
LQcost = Tcost/(Tsize-in2stat)
!print*,"LONGEST Q: average costs = ",LQcost
!print*,"stable = ",stable
!close(unit=7)

return
end subroutine

!---------------------------------------------------------------------------
----------------------

subroutine myopic(r,seed,Nmax,TLactsize,TL1,TL2,MYcost)
implicit none

integer :: size,k,count,custserve,event,state,Nmax,num,r,BError, &
            & i,TLactsize,marker,custserveold,arrivalold
integer, dimension(3) :: seed
integer, dimension(500000) :: TL1
integer, dimension(5) :: buffer
integer, dimension(0:5) :: n,numserved,m
double precision :: Tsize,Tservice,stable,in2stat,TTservice
double precision :: lastevent,endserve,NEtime,bigind,Tcost,MYcost,TOL
double precision, dimension(0:5) :: l,mu
double precision, dimension(5) :: a,b
double precision, dimension(500000) :: TL2
double precision, dimension(5,0:Nmax) :: C

call qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)
TTservice = 0.0
marker = 0
numserved = 0
in2stat = Tsize*0.667

!test to ensure that we have stable queues
```

```
stable = 0.0
do k = 1,5
  if (mu(k) < 999999.999999) stable = stable + l(k)*m(k)/mu(k)
end do
if (stable >= 1.0) print*,"ERROR: UNSTABLE SYSTEM!!!"

do k = 1,5
  do num=buffer(k),Nmax
    C(k,num) = a(k)*(real(num-buffer(k))**3.0) +
b(k)*(real(num-buffer(k))**4.0)
  end do
end do

Tcost = 0.0
n = 0

n(TL1(1)) = n(TL1(1)) + 1
lastevent = TL2(1)
custserve = TL1(1)
call gammaservice(r,custserve,Tservice)
endserve = lastevent + Tservice

if (lastevent > in2stat) then
  marker = 1
end if

!do k = 1,5
!  Tcost = Tcost + C(k,n(k))*NEtime
!end do

!open(unit=7,file="SimIndexCostR.dat")
!write(unit=7,fmt="(a)") "  W(1,:)  :  W(2,:)  :  W(3,:)  :  W(4,:)  :
W(5,:)  "
!do i = 0,Nmax
!  write(unit=7,fmt="(i4,5f12.6)") i,W(:,i)
!end do
!close(unit=7)

!open(unit=7,file="simresultsMY.txt")

event = 2
20 custserveold = -1
   arrivalold  = -1

  state = 0
  do k = 1,5
    state = state + n(k)
  end do
  if(TL2(event) < endserve .or. state == 0) then
    if(TL2(event) < Tsize) then
      NEtime = TL2(event) - lastevent
      lastevent = TL2(event)
    else
      NEtime = Tsize - lastevent
      lastevent = Tsize
    end if

    if ( state == 0 ) then
      custserve = TL1(event)
      call gammaservice(r,custserve,Tservice)
      endserve = lastevent + Tservice
      numserved(custserve) = numserved(custserve) + 1
    end if

    arrivalold = TL1(event)
    event = event + 1
  else
    if(endserve < Tsize) then
      NEtime = endserve - lastevent
      lastevent = endserve
    else
      NEtime = Tsize - lastevent
      lastevent = Tsize
    end if
    custserveold = custserve
```

```
    n(custserve) = n(custserve) - 1
    state = 0
    do k = 1,5
      state = state + n(k)
    end do
    bigind = -9999.99
    do k = 1,5
      if (n(k) > 0) then
        if (bigind < C(k,n(k))) then
          bigind = C(k,n(k))
          custserve = k
        end if
      end if
    end do
    if (state == 0) custserve = 0
    call gammaservice(r,custserve,Tservice)
    endserve = lastevent + Tservice

    n(custserveold) = n(custserveold) + 1
    numserved(custserve) = numserved(custserve) + 1

  end if

  if (lastevent > in2stat) then
    if (marker == 0) then
      NEtime = lastevent - in2stat
    end if
    marker = 1
  end if

  do k = 1,5
    Tcost = Tcost + C(k,n(k))*NEtime*real(marker)
  end do

  if (arrivalold > 0 .and. n(arrivalold) < Nmax) n(arrivalold) =
n(arrivalold) + 1  !!!!!!!!!!!!!!!!!!!!!!!!!!!!
  if (custserveold >= 0 .and. n(custserveold) >= 1) n(custserveold) =
n(custserveold) - 1
  state = 0
  do k = 1,5
    state = state + n(k)
  end do

  TTservice = TTservice + Tservice
!  write(unit=7,fmt="(a,i5)") "event # = ",event
!  write(unit=7,fmt="(a,f12.6)") "Tservice = ",Tservice
!  write(unit=7,fmt="(a,f12.6)") "endserve = ",endserve
!  write(unit=7,fmt="(a,f12.6)") "NEtime = ",NEtime
!  write(unit=7,fmt="(a,6i5)") "state = ",n
!  write(unit=7,fmt="(a,f12.6)") "TL2 = ",TL2(event)
!  write(unit=7,fmt="(a,f20.3)") "Tcost = ",Tcost
!  write(unit=7,fmt="(a,f12.6)") "Tcost/size =
",Tcost/(Lastevent+NEtime-in2stat)
!  write(unit=7,fmt="(a)") " "

if (event < TLactsize) goto 20
!write(unit=7,fmt="(a,f12.6)") "TTservice = ",TTservice
!write(unit=7,fmt="(a,6i6)") "numserved = ",numserved
MYcost = Tcost/(Tsize-in2stat)
!print*,"MYOPIC: average costs = ",MYcost
!print*,"stable = ",stable
!close(unit=7)

return
end subroutine

!--------------------------------------------------------------------------
----------------------

subroutine cmew(r,seed,Nmax,TLactsize,TL1,TL2,CMcost)
implicit none

integer :: size,k,count,custserve,event,state,Nmax,num,r,BError, &
           & i,TLactsize,marker,custserveold,arrivalold
integer, dimension(3) :: seed
```

```
integer, dimension(500000) :: TL1
integer, dimension(5) :: buffer
integer, dimension(0:5) :: n,numserved,m
double precision :: Tsize,Tservice,stable,in2stat,TTservice
double precision :: lastevent,endserve,NEtime,bigind,Tcost,CMcost,TOL
double precision, dimension(0:5) :: l,mu
double precision, dimension(5) :: a,b
double precision, dimension(500000) :: TL2
double precision, dimension(5,0:Nmax) :: C

call qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)
TTservice = 0.0
marker = 0
numserved = 0
in2stat = Tsize*0.667

!test to ensure that we have stable queues
stable = 0.0
do k = 1,5
  if (mu(k) < 999999.999999) stable = stable + l(k)*m(k)/mu(k)
end do
if (stable >= 1.0) print*,"ERROR: UNSTABLE SYSTEM!!!"

do k = 1,5
  do num=buffer(k),Nmax
    C(k,num) = a(k)*(real(num-buffer(k))**3.0) +
b(k)*(real(num-buffer(k))**4.0)
  end do
end do

Tcost = 0.0
n = 0

n(TL1(1)) = n(TL1(1)) + 1
lastevent = TL2(1)
custserve = TL1(1)
call gammaservice(r,custserve,Tservice)
endserve = lastevent + Tservice

if (lastevent > in2stat) then
  marker = 1
end if

!do k = 1,5
!  Tcost = Tcost + C(k,n(k))*NEtime
!end do

!open(unit=7,file="SimIndexCostR.dat")
!write(unit=7,fmt="(a)") "  W(1,:)  :  W(2,:)  :  W(3,:)  :  W(4,:)  :
W(5,:)  "
!do i = 0,Nmax
!  write(unit=7,fmt="(i4,5f12.6)") i,W(:,i)
!end do
!close(unit=7)

!open(unit=7,file="simresultsCM.txt")

event = 2
20 custserveold = -1
   arrivalold  = -1

  state = 0
  do k = 1,5
    state = state + n(k)
  end do
  if(TL2(event) < endserve .or. state == 0) then
    if(TL2(event) < Tsize) then
      NEtime = TL2(event) - lastevent
      lastevent = TL2(event)
    else
      NEtime = Tsize - lastevent
      lastevent = Tsize
    end if

    if ( state == 0 ) then
```

```
        custserve = TL1(event)
        call gammaservice(r,custserve,Tservice)
        endserve = lastevent + Tservice
        numserved(custserve) = numserved(custserve) + 1
      end if

      arrivalold = TL1(event)
      event = event + 1

    else
      if(endserve < Tsize) then
        NEtime = endserve - lastevent
        lastevent = endserve
      else
        NEtime = Tsize - lastevent
        lastevent = Tsize
      end if

      custserveold = custserve
      n(custserve) = n(custserve) - 1
      state = 0
      do k = 1,5
        state = state + n(k)
      end do
      bigind = -9999.99
      do k = 1,5
        if (n(k) > 0) then
          if (bigind < C(k,n(k))*mu(k)) then
            bigind = C(k,n(k))*mu(k)
            custserve = k
          end if
        end if
      end do
      if (state == 0) custserve = 0
      call gammaservice(r,custserve,Tservice)
      endserve = lastevent + Tservice

      n(custserveold) = n(custserveold) + 1
      numserved(custserve) = numserved(custserve) + 1

    end if

    if (lastevent > in2stat) then
      if (marker == 0) then
        NEtime = lastevent - in2stat
      end if
      marker = 1
    end if

    do k = 1,5
      Tcost = Tcost + C(k,n(k))*NEtime*real(marker)
    end do

    if (arrivalold > 0 .and. n(arrivalold) < Nmax) n(arrivalold) =
  n(arrivalold) + 1  !!!!!!!!!!!!!!!!!!!!!!!!!!!!
    if (custserveold >= 0 .and. n(custserveold) >= 1) n(custserveold) =
  n(custserveold) - 1
    state = 0
    do k = 1,5
      state = state + n(k)
    end do

    TTservice = TTservice + Tservice
!   write(unit=7,fmt="(a,i5)") "event # = ",event
!   write(unit=7,fmt="(a,f12.6)") "Tservice = ",Tservice
!   write(unit=7,fmt="(a,f12.6)") "endserve = ",endserve
!   write(unit=7,fmt="(a,f12.6)") "NEtime = ",NEtime
!   write(unit=7,fmt="(a,6i5)") "state = ",n
!   write(unit=7,fmt="(a,f12.6)") "TL2 = ",TL2(event)
!   write(unit=7,fmt="(a,f20.3)") "Tcost = ",Tcost
!   write(unit=7,fmt="(a,f12.6)") "Tcost/size =
",Tcost/(Lastevent+NEtime-in2stat)
!   write(unit=7,fmt="(a)") " "

  if (event < TLactsize) goto 20
```

```
!write(unit=7,fmt="(a,f12.6)") "TTservice = ",TTservice
!write(unit=7,fmt="(a,6i6)") "numserved = ",numserved
CMcost = Tcost/(Tsize-in2stat)
!print*,"C*MEW: average costs = ",CMcost
!print*,"stable = ",stable
!close(unit=7)

return
end subroutine

!--------------------------------------------------------------------------
---------------------

subroutine static(r,seed,Nmax,TLactsize,TL1,TL2,STcost)
implicit none

integer :: size,k,count,custserve,event,state,Nmax,num,r,BError, &
          & i,TLactsize,marker,custserveold,arrivalold
integer, dimension(3) :: seed
integer, dimension(500000) :: TL1
integer, dimension(5) :: buffer
integer, dimension(0:5) :: n,numserved,m
double precision :: x,Tsize,Tservice,stable,in2stat,TTservice,renormstat
double precision :: lastevent,endserve,NEtime,Tcost,STcost,TOL
double precision, dimension(0:5) :: l,mu,stationary
double precision, dimension(5) :: a,b,stationary2
double precision, dimension(500000) :: TL2
double precision, dimension(5,0:Nmax) :: C

call qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)
TTservice = 0.0
marker = 0
numserved = 0
stationary = 0.0
stationary2 = (/0.2,0.2,0.2,0.2,0.2/)
do i = 1,5
  stationary(i) = stationary(i-1) + stationary2(i)
end do

in2stat = Tsize*0.667

!test to ensure that we have stable queues
stable = 0.0
do k = 1,5
  if (mu(k) < 999999.999999) stable = stable + l(k)*m(k)/mu(k)
end do
if (stable >= 1.0) print*,"ERROR: UNSTABLE SYSTEM!!!"

do k = 1,5
  do num=buffer(k),Nmax
    C(k,num) = a(k)*(real(num-buffer(k))**3.0) +
b(k)*(real(num-buffer(k))**4.0)
  end do
end do

do i = 1,10
  call random_number(x)
end do

Tcost = 0.0
n = 0

n(TL1(1)) = n(TL1(1)) + 1
lastevent = TL2(1)
custserve = TL1(1)
call gammaservice(r,custserve,Tservice)
endserve = lastevent + Tservice

if (lastevent > in2stat) then
  marker = 1
end if

!do k = 1,5
!  Tcost = Tcost + C(k,n(k))*NEtime
!end do
```

```
!open(unit=7,file="SimIndexCostR.dat")
!write(unit=7,fmt="(a)") "  W(1,:)  :  W(2,:)  :  W(3,:)  :  W(4,:)  :
W(5,:)  "
!do i = 0,Nmax
!  write(unit=7,fmt="(i4,5f12.6)") i,W(:,i)
!end do
!close(unit=7)


!open(unit=7,file="simresultsST.txt")

event = 2
20 do i = 1,5
     stationary(i) = stationary(i-1) + stationary2(i)
   end do

  custserveold = -1
  arrivalold  = -1

  state = 0
  do k = 1,5
    state = state + n(k)
  end do
  if(TL2(event) < endserve .or. state == 0) then
    if(TL2(event) < Tsize) then
      NEtime = TL2(event) - lastevent
      lastevent = TL2(event)
    else
      NEtime = Tsize - lastevent
      lastevent = Tsize
    end if

    if ( state == 0 ) then
      custserve = TL1(event)
      call gammaservice(r,custserve,Tservice)
      endserve = lastevent + Tservice
      numserved(custserve) = numserved(custserve) + 1
    end if

    arrivalold = TL1(event)
    event = event + 1

  else
    if(endserve < Tsize) then
      NEtime = endserve - lastevent
      lastevent = endserve
    else
      NEtime = Tsize - lastevent
      lastevent = Tsize
    end if

    custserveold = custserve
    n(custserve) = n(custserve) - 1

    state = 0
    do k = 1,5
      state = state + n(k)
    end do
    call random_number(x)
    renormstat = 0.0
    do i = 1,5
      if (n(i) > 0) renormstat = renormstat + stationary2(i)
    end do
    do i = 1,5
      if (n(i) > 0) then
        stationary(i) = stationary(i)/renormstat
      else
        stationary(i) = stationary(i-1)
      end if
    end do

    if (x < stationary(1)) then
      custserve = 1
    else if (x < stationary(2)) then
```

```
      custserve = 2
    else if (x < stationary(3)) then
      custserve = 3
    else if (x < stationary(4)) then
      custserve = 4
    else if (x < stationary(5)) then
      custserve = 5
    end if

    if (state == 0) custserve = 0
    call gammaservice(r,custserve,Tservice)
    endserve = lastevent + Tservice

    n(custserveold) = n(custserveold) + 1
    numserved(custserve) = numserved(custserve) + 1

  end if

  if (lastevent > in2stat) then
    if (marker == 0) then
      NEtime = lastevent - in2stat
    end if
    marker = 1
  end if

  do k = 1,5
    Tcost = Tcost + C(k,n(k))*NEtime*real(marker)
  end do

  if (arrivalold > 0 .and. n(arrivalold) < Nmax) n(arrivalold) =
n(arrivalold) + 1  !!!!!!!!!!!!!!!!!!!!!!!!!!!!
  if (custserveold >= 0 .and. n(custserveold) >= 1) n(custserveold) =
n(custserveold) - 1
  state = 0
  do k = 1,5
    state = state + n(k)
  end do

  TTservice = TTservice + Tservice
!  write(unit=7,fmt="(a,i5)") "event # = ",event
!  write(unit=7,fmt="(a,f12.6)") "Tservice = ",Tservice
!  write(unit=7,fmt="(a,f12.6)") "endserve = ",endserve
!  write(unit=7,fmt="(a,f12.6)") "NEtime = ",NEtime
!  write(unit=7,fmt="(a,6i5)") "state = ",n
!  write(unit=7,fmt="(a,f12.6)") "TL2 = ",TL2(event)
!  write(unit=7,fmt="(a,f20.3)") "Tcost = ",Tcost
!  write(unit=7,fmt="(a,f12.6)") "Tcost/size =
",Tcost/(Lastevent+NEtime-in2stat)
!  write(unit=7,fmt="(a)") " "

if (event < TLactsize) goto 20
!write(unit=7,fmt="(a,f12.6)") "TTservice = ",TTservice
!write(unit=7,fmt="(a,6i6)") "numserved = ",numserved
STcost = Tcost/(Tsize-in2stat)
!print*,"STATIC average costs = ",STcost
!print*,"stable = ",stable
!close(unit=7)

return
end subroutine

!--------------------------------------------------------------------------
----------------------------------------

subroutine gammaservice(r,custserve,Tservice)
implicit none

integer :: Nmax,BError,r,size,custserve,top,i
integer, dimension(5) :: buffer
integer, dimension(0:5) :: m
double precision, dimension(5) :: a,b
double precision, dimension(0:5) :: mu,l
double precision :: TOL,Tsize,Tservice,Tphase,x

a = 0.0
```

```
b = 0.0
l = 0.0
mu = 0.0
m = 0
Nmax = 0
buffer = 0
BError = 0
TOL = 0.0
size = 0
Tsize = 0.0
call qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)

top = m(custserve)
Tphase = 0.0
do i = 1,top
  call random_number(x)
  Tphase = Tphase + (-1.0*log(x)/mu(custserve))
end do
Tservice = Tphase

return
end subroutine

!------------------------------------------------------------------------------
-

subroutine qvals(r,a,b,l,mu,m,Nmax,buffer,BError,TOL,size,Tsize)
implicit none

integer :: Nmax,BError,r,size,i,j
integer, dimension(5) :: buffer
integer, dimension(0:5) :: m,mold
double precision, dimension(5) :: a,b
double precision, dimension(0:5) :: mu,l,lold,muold,row
double precision :: TOL,Tsize

row  = 0.0

Nmax = 69
buffer = 0
BError = 5

size = 100000
Tsize = 15000.0

l =    (/0.0,0.4,0.3,0.25,0.1,0.05/)
mu = (/100.0,1.6667,6.0,5.0,5.7143,6.25/)
m = (/1,1,3,2,4,5/)

l =    (/0.0,0.4,0.3,0.25,0.1,0.05/) ! first
mu = (/100.0,1.6667,6.0,5.0,5.7143,6.25/) ! first
m = (/1,1,3,2,4,5/) ! first

do i = 1,5
  row(i) = l(i)*(m(i)/mu(i))
  row(0) = row(0) + row(i)
end do

do i = 1,5
  l(i) = (l(i)/row(0))*0.85
end do

!lold =    (/0.0,l(4),l(5),l(1),l(2),l(3)/)
!muold = (/100.0,mu(4),mu(5),mu(1),mu(2),mu(3)/)
!mold = (/1,m(4),m(5),m(1),m(2),m(3)/)

do i = 1,5
  j = mod(i + r,5)
  if (j == 0) j = 5
  lold(j) = l(i)
  muold(j) = mu(i)
  mold(j) = m(i)
end do

l = lold
```

```
mu = muold
m = mold


a = (/5.0,4.0,3.0,2.0,1.0/)
b = (/1.0,2.0,3.0,4.0,5.0/)

TOL = 0.0001

return
end subroutine
!---------------------------------------------------------------------------
----------------------
```

Appendix G

Associated Published Work

Queueing Systems 45, 81–111, 2003

Index Heuristics for Multiclass $M/G/1$ Systems with Nonpreemptive Service and Convex Holding Costs

K.D. GLAZEBROOK and R.R. LUMLEY
School of Management, Edinburgh University, EH8 9JY, UK

P.S. ANSELL
Department of Statistics, University of Newcastle upon Tyne, NE1 7RU, UK

Received 20 September 2001; Revised 10 April 2003

Abstract. We consider the optimal service control of a multiclass $M/G/1$ queueing system in which customers are served nonpreemptively and the system cost rate is additive across classes and increasing convex in the numbers present in each class. Following Whittle's approach to a class of restless bandit problems, we develop a Langrangian relaxation of the service control problem which serves to motivate the development of a class of index heuristics. The index for a particular customer class is characterised as a fair charge for service of that class. The paper develops these indices and reports an extensive numerical investigation which exhibits strong performance of the index heuristics for both discounted and average costs.

Keywords: indexability, index policy, service control, stochastic dynamic programming, restless bandit

1. Introduction

A prime focus of much of the literature concerning the optimal dynamic control of service in a multiclass queueing environment has been the development of policies to minimise some measure of total holding cost in the system. An assumption that holding cost rates be linear in the number of customers (or, equivalently, that each class have a fixed holding cost rate per unit time and per customer in the system) has been central to the elucidation of simple priority policies as optimal in a variety of contexts. See, for example, [5,6,9,10]. Theoretical connections of this work with ideas concerning Gittins indices for multi-armed bandit problems are developed in [4,8,16]. However, criticisms of the appropriateness of the assumption of linear holding costs and of some aspects of the performance of the resulting priority policies have been voiced, *inter alia*, by van Meighem [14] and Ansell et al. [2]. Contributions to the literature of multiclass queueing systems which allow for nonlinear costs are few. They include those of [2,13–15].

In response to the need for further work in this area, this paper will be concerned with the optimal service control of a multiclass $M/G/1$ queueing system in which customers are served nonpreemptively and the system cost rate is assumed to be additive across classes and increasing convex in the numbers present in each class. In attempting this, we develop work of Ansell et al. [3] who consider the relatively simple special case of an $M/M/1$ system with preemptive service.

In section 2 both discounted and average cost versions of our multiclass service control problem are presented. These semi-Markov decision problems are strongly related to an intractable class of resource allocation models called *restless bandits*, introduced by Whittle [17]. On this basis we argue for the development of effective *index policies* which make decisions concerning the direction of service effort on the basis of calibrating functions (or indices) associated with the customer classes. Despite the belief of Whittle [18] that his approach to index development based on Langrangian methods could not be applied to (average cost versions of) such service control problems, we present such an approach in section 3. Indices emerge as values of Lagrange multipliers associated with a work conservation constraint. Alternatively, the index function for a particular class may be understood as a *fair charge* for serving that class. Our index heuristics always direct service effort to whichever customer class has the largest associated fair charge for service.

These index characterisations necessitate a digression in section 4 toward the study of a service control problem (one for each customer class) for a single class $M/G/1$ system with a charge for service. This study establishes that the desired class indices are well defined and yields formulae for them and methods for their computation. All of this is in terms of discounted costs. Appropriate indices for average costs are derived as limits (as the discount rate $\alpha \to 0$). The derived single class problems of section 4 have some affinities with the growing literature on queueing models in which the server periodically takes one or more vacations, usually when the system empties. The associated control problem is how to decide dynamically when the server should be reintroduced. See, for example, [1,7]. The results we describe in section 4 for our single class problems are established using the techniques of stochastic dynamic programming. Niño Mora [11] has espoused an alternative approach to indexability/index development based on polyhedral methods. This approach is summarised in [3, section 4]. See also other important work on restless bandit models due to Weber and Weiss [15,16].

The paper concludes in section 5 with an extensive numerical investigation into the quality of performance of the derived index heuristics. Study of a range of two class problems for both discounted and average costs shows that the index policy is sometimes indistinguishable from an optimal policy in cost terms. This very strong cost performance is further evidenced in a simulation study based on larger five class problems.

2. Service control of multiclass $M/G/1$ systems

We shall consider multiclass $M/G/1$ queueing systems in which customers from classes $\{1, 2, \dots, K\}$ receive service provided by a single server. Arrivals into the system are in K independent Poisson streams with λ_k the rate for class k. Each customer has a service time and these are independent for different customers and identically distributed for customers within a single class. We write S_k for a generic class k service time and G_k

for the corresponding distribution function. We shall suppose that all moments of S_k exists and that

$$\rho \equiv \sum_{k=1}^{K} \lambda_k E(S_k) < 1$$

for stability. The goal is to allocate service among the waiting customers to minimise some measure of expected holding cost over an infinite horizon. We shall consider both discounted and average cost criteria. We formalise the queueing control problems of interest as semi-Markov Decision Problems (SMDPs) as follows:

(a) The state of the system at time t is $\mathbf{N}(t) = \{N_1(t), N_2(t), \ldots, N_K(t)\}$, the vector of queue lengths, $t \in \mathbb{R}^+$. The decision epochs are all service completion times which do not result in an empty system together with all times of arrivals at an empty system. Let action a_k denote the allocation of service to a class k customer, $1 \leqslant k \leqslant K$. At each decision epoch t, the controller chooses an action a_k from the set of k for which $N_k(t) \geqslant 1$;

(b) Suppose that $\mathbf{N}(t) = \mathbf{n}$ with $n_k > 0$, that t is a decision epoch and that action a_k is taken then. The next decision epoch will occur at time $t + S_k$, where $S_k \sim G_k$ and the system state then has probability distribution given by

$$P\big[\mathbf{N}\big((t+S_k)^+\big) = \mathbf{n} - \mathbf{1}^k + \mathbf{m}\big] = \int_0^\infty \left\{ \prod_{j=1}^{K} \frac{(\lambda_j t)^{m_j}}{m_j!} \mathrm{e}^{-\lambda_j t} \right\} \mathrm{d}G_k, \quad \mathbf{m} \in \mathbb{N}^K. \tag{1}$$

Note that in (1), $\mathbf{1}^k$ denotes a K-vector whose kth component is 1, with zeroes elsewhere. Note also that the processing of the class k customer which begins at time t is nonpreemptive.

(c) In the *discounted costs* version of the queueing control problems of interest, discounted costs are incurred, with rate

$$\alpha \sum_{j=1}^{K} C_j\big(N_j(t)\big) \tag{2}$$

at time t. The cost functions $C_k : \mathbb{N} \to \mathbb{R}^+$ are assumed increasing, convex and bounded above by some polynomial of finite order and with $C_k(0) = 0,\ 1 \leqslant k \leqslant K$. A policy u is a rule for choosing actions in light of the history of the process to date and $\mathcal{U}$ is the collection of all such policies which are non-idling for the single server. Our goal is to seek a policy which minimises total costs incurred over an infinite horizon. We write

$$\mathbf{V}(\mathbf{m}, \alpha) = \inf_{u \in \mathcal{U}} E_u \left[\int_0^\infty \sum_{k=1}^{K} \alpha C_k\big(N_k(t)\big) \mathrm{e}^{-\alpha t} \mid \mathbf{N}(0) = \mathbf{m} \right] \tag{3}$$

for the associated value function. Please note that the multiplier α has been introduced into the holding cost rate in (2) to guarantee that $\mathbf{V}(\mathbf{m}, \alpha)$ remains finite in the limit as α approaches zero. This limit is central to the consideration of average cost problems which are of great importance to us. See (6) below. Further reasons for the inclusion of the α multiplier are given in section 3 following definition 3. Plainly, the multiplier has no impact upon the optimal policy in (3).

The general theory of stochastic dynamic programming (DP) indicates the existence of an optimal policy which is stationary (i.e., makes decisions in light of the current state only) and whose value function satisfies the DP optimality equations. See [12]. However, for our multiclass queueing control problem a pure DP approach is unlikely to be insightful and will be computationally intractable for problems of reasonable size.

In two special cases the optimal policy is known to be of *index form*. This means that there exist K *index functions* $W_{k,\alpha}: \mathbb{N} \to \mathbb{R}^+$, $1 \leqslant k \leqslant K$, such that the *index policy* u_W which at all decision epochs chooses to process a customer from the maximal index class, i.e.

$$u_W\{\mathbf{N}(t)\} = a_k \Rightarrow W_{k,\alpha}\{N_k(t)\} = \max_{1 \leqslant j \leqslant K} W_{j,\alpha}\{N_j(t)\} \tag{4}$$

is optimal. These special instances are (i) the *batch case* and (ii) when all holding cost rates C_k are *linear* in the queue lengths. The batch case occurs when all arrival rates are 0 and the goal is to empty the system (by serving to completion all customers present at time 0) at minimum cost. This may be formulated as a *multi-armed bandit problem* for which a *Gittins index policy* may be shown to be optimal. See [8]. The linear costs case was first solved by Harrison [9]. The theoretical force of an assumption of cost linearity is that an analysis at the level of *the individual customer* (each of whom carries her own holding cost rate) rather than at the level of the customer class is possible. Latterly, the linear cost problem has been formulated as a *branching bandit problem* for which Gittins index policies are known to be optimal. See [4]. These special cases apart, the service control problem in (a)–(c) is strongly related to an intractable class of problems called *restless bandits*. Whittle [17] introduced this class of decision problems and proposed an index heuristic which emerged naturally from a Langrangian relaxation of the problem. Whittle [18] himself thought that these ideas could not be applied to queueing control models of the kind discussed here. In fact they can be, as is explained in outline in the next section. Hence, in section 4 we shall develop a *Whittle index policy* for the discounted costs problem. This policy will coincide with the optimal index policies in the special cases (i) and (ii) above.

The *average cost* version of the multiclass queueing control model of interest is expressed via the equation

$$\mathbf{V}^{\mathrm{OPT}} = \inf_{u \in \mathcal{U}} \widetilde{E}_u \left\{ \sum_{k=1}^{K} C_k(N_k) \right\}, \tag{5}$$

where in (5) $\widetilde{E}_u$ is the expectation taken with respect to the steady-state distribution of the system under policy u. From standard results in DP we have that

$$\lim_{\alpha \to 0} \mathbf{V}(\mathbf{m}, \alpha) = \mathbf{V}^{\mathrm{OPT}}. \tag{6}$$

In light of (6), we shall develop natural heuristics for average cost problems as limits ($\alpha \to 0$) of the index policies for discounted costs.

3. Indexability and Whittle indices for service control

As is mentioned above, Whittle's [17,18] approach to the development of index heuristics for restless bandit problems was via Langrangian relaxations. An attempt in [18] to analyse average cost versions of our service control problems directly by these means failed and it was suggested that these ideas were not helpful in this context. As we shall see, the key to progress is to begin with the apparently more difficult discounted costs problem and to recover the average costs version as a limiting form, as in (6).

To facilitate our discussion, we write $a_k(t)$ for the action (either $a =$ serve (active) or $b =$ do not serve (passive)) applied to queue k at time t. We then develop the following *performance measures* for policy u:

$$x_{k,n}^{a,u}(\mathbf{m}) = E_u\left[\int_0^\infty I\left\{a_k(t) = a,\, N_k(t) = n\right\} \mathrm{e}^{-\alpha t}\,\mathrm{d}t \mid \mathbf{N}(0) = \mathbf{m}\right], \tag{7}$$

and similarly for $x_{k,n}^{b,u}(\mathbf{m})$, $\mathbf{m} \in \mathbb{N}^K$, $n \in \mathbb{N}$, $1 \leqslant k \leqslant K$. In (7), $I\{\cdot\}$ is the indicator function. We may now re-express our discounted cost problem in (3) as

$$\mathbf{V}(\mathbf{m}, \alpha) = \inf_{u \in \mathcal{U}} \sum_{k=1}^{K} \sum_{n \in \mathbb{N}} \alpha C_k(n)\left\{x_{k,n}^{a,u}(\mathbf{m}) + x_{k,n}^{b,u}(\mathbf{m})\right\}. \tag{8}$$

We develop a relaxation of (8) by first observing that for all policies in $\mathcal{U}$, the quantity

$$\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{a,u}(\mathbf{m}) = E_u\left[\int_0^\infty I\left\{\mathbf{N}(t) \neq \mathbf{0}\right\}\mathrm{e}^{-\alpha t}\,\mathrm{d}t \mid \mathbf{N}(0) = \mathbf{m}\right] = \alpha^{-1}\rho + \Theta(\mathbf{m}, \alpha) \tag{9}$$

is policy invariant. This arises from the fact that the duration of the first busy period (i.e., the time required to empty the system from initial state $\mathbf{m}$) and all subsequent busy periods have probability distributions which do not depend upon u. In (9), $\mathbf{0}$ is the zero K-vector and $\Theta(\mathbf{m}, \alpha)$ is an $\mathrm{O}(1)$ quantity (as $\alpha \to 0$) which does not depend upon u. Note that the form of the constant in (9) follows from standard queueing theory considerations. We now relax the stochastic optimisation problem (8) by both expanding the policy class to $\overline{\mathcal{U}}$, namely, the set of policies in which *any* number of non-empty customer classes may be served at any time (but where any service, once started, must be completed) and then by imposing the relation in (9) as a constraint. Roughly speaking, we are relaxing the sample path requirement that a single class be served at each time (at which the system is non-empty) to one in which one class is served *on average*, in the sense of (9). We also extend $\overline{\mathcal{U}}$ to include randomisations over such policies. We call this *Whittle's relaxation* and write

$$\underline{\mathbf{V}}(\mathbf{m}, \alpha) = \inf_{u \in \overline{\mathcal{U}}} \sum_{k=1}^{K} \sum_{n \in \mathbb{N}} \alpha C_k(n) \left\{ x_{k,n}^{a,u}(\mathbf{m}) + x_{k,n}^{b,u}(\mathbf{m}) \right\}$$

$$\text{subject to} \quad \sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{a,u}(\mathbf{m}) = E_u \left[\int_0^{\infty} K(t) \mathrm{e}^{-\alpha t} \, \mathrm{d}t \mid \mathbf{N}(0) = \mathbf{m} \right] = \alpha^{-1} \rho + \Theta(\mathbf{m}, \alpha). \tag{10}$$

Note that $K(t)$ denotes the number of customer classes served at t and constraint (10) delimits the set of allowable policies within $\overline{\mathcal{U}}$. For any policy within $\mathcal{U}$ we have $K(t) = I\{\mathbf{N}(t) \neq \mathbf{0}\},\ t \in (0, \infty)$. We now adopt a Langrangian approach to elucidating the structure of the optimal solution to Whittle's relaxation. Hence we accommodate constraint (10) by incorporating a Langrange multiplier W to obtain the minimisation problem

$$\mathbf{V}(\mathbf{m}, \alpha, W) = \inf_{u \in \overline{\mathcal{U}}} \left[\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} \left\{ \alpha C_k(n) + W \right\} x_{k,n}^{a,u}(\mathbf{m}) + \sum_{k=1}^{K} \sum_{n \in \mathbb{N}} \alpha C_k(n) x_{k,n}^{b,u}(\mathbf{m}) \right]. \tag{11}$$

We see from (11) that W plays the economic role of a *constant charge for service*. Problem (11) is naturally decoupled into K single-class subproblems

$$\mathbf{V}(\mathbf{m}, \alpha, W) = \sum_{k=1}^{K} V_k(m_k, \alpha, W). \tag{12}$$

In (12), $V_k(m_k, \alpha, W)$ is the minimised total of holding costs and service charge costs incurred by customer class k, the minimisation being taken over all (nonpreemptive) policies for choosing between actions a and b for that class *only*. Call this single class problem $(k, \alpha, W),\ W \in \mathbb{R},\ 1 \leqslant k \leqslant K$.

It will be shown in section 4 that there exists a multiplier $W(\mathbf{m}, \alpha)$ such that

$$\mathbf{V}\left\{\mathbf{m}, \alpha, W(\mathbf{m}, \alpha)\right\} - W(\mathbf{m}, \alpha)\left\{\alpha^{-1}\rho + \Theta(m, \alpha)\right\} = \underline{\mathbf{V}}(\mathbf{m}, \alpha),$$

and that there exists an optimal policy for the Langrangian relaxation in (11) with $W = W(\mathbf{m}, \alpha)$ which satisfies the constraint in (10) and hence solves Whittle's relaxation. However, by (12), this optimal policy is a superposition of optimal policies for the single class problems $\{k, \alpha, W(\mathbf{m}, \alpha)\},\ 1 \leqslant k \leqslant K$. But the solutions to these problems become especially simple under a condition of *indexability*. To describe this condition, we write $\Pi_{k,\alpha}(W)$ for the set of queue lengths m for which the passive action b is optimal for single class problem (k, α, W).

Definition 1. Customer class k is *α-indexable* if $\Pi_{k,\alpha}: \mathbb{R} \to 2^{\mathbb{N}}$ is increasing, namely,

$$W_1 > W_2 \quad \Rightarrow \quad \Pi_{k,\alpha}(W_1) \supseteq \Pi_{k,\alpha}(W_2).$$

Should we have α-indexability for class k, the idea of an α-index for state (i.e. queue length) m as the minimum service charge which makes the passive action optimal there is a natural one.

Definition 2. When customer class k is α-indexable, the *Whittle α-index* for class k in state m is given by

$$W_{k,\alpha}(m) = \inf\{W\colon m \in \Pi_{k,\alpha}(W)\}, \quad m \in \mathbb{Z}^+.$$

It will now follow that if each customer class k is α-indexable, Whittle's relaxation is solved by a policy in which a decision is taken to serve customer class k at each decision epoch t for each (k, α, W) whenever $W_{k,\alpha}\{N_k(t)\} > W(\mathbf{m}, \alpha)$ and not to serve k whenever $W_{k,\alpha}\{N_k(t)\} < W(\mathbf{m}, \alpha)$, for all choices of k, t. Should $W_{k,\alpha}\{N_k(t)\} = W(\mathbf{m}, \alpha)$ then some randomisation between the two actions will be appropriate.

We now follow [17] in arguing that the index-like nature of solutions to the relaxation in (10) makes it reasonable to propose an *index heuristic* for our original discounted costs problem in (3) and (8) when all customer classes are α-indexable. This heuristic will be structured as in (4) with index functions recovered from definition 2. Note that under this definition, it is natural to interpret $W_{k,\alpha}(m)$ as a *fair charge* for serving customer class k in state m. The derived heuristic then always serves that class for which the fair charge for service is highest. Following the discussion at the end of section 2, we develop an index heuristic for average cost problems as the limit policy (as $\alpha \to 0$) of the index heuristics for discounted costs.

Definition 3. If customer class k is α-indexable for all $\alpha > 0$ then the *average cost Whittle index* for state m is given by

$$W_k(m) = \lim_{\alpha \to 0} W_{k,\alpha}(m), \quad m \in \mathbb{Z}^+, \tag{13}$$

when the above limits exist.

Note that the inclusion of the α multiplier in the holding cost rates for the discounted problem guarantees that the limits in (13) exist and yield sensible indices. To see why, revisit the Langrangian in (11). As policy u varies within the stable policies in $\overline{\mathcal{U}}$ it is known from standard MDP theory that the holding cost component of (11) will vary by amounts which are O(1). However, it must also be true for such policies that

$$\sum_{k=1}^{K} \sum_{n \in \mathbb{N}} x_{k,n}^{a,u}(\mathbf{m}) = \alpha^{-1}\rho + \mathrm{O}(1)$$

and hence, for any finite W, varying u can only change the service charge component of (11) by O(1). It is this balancing of the contributions to the total cost in (11) which guarantees the good behaviour of the limits in (13).

Taking our cue from the above discussion, we now proceed to study the single class problems (k, α, W) in the next section. We shall establish α-indexability and derive α-indices and the average cost indices which are appropriate for our service control problems.

4. The $M/G/1$ queue with a charge for service

Following section 3, we study the single class problems (k, α, W), $1 \leqslant k \leqslant K$. In doing so, it will be notationally convenient to drop the class identifier k. Hence the problems of interest concern a single server who is available to process a single class of customers in a queueing system. However, there is a charge for the server's work and the server can be stood down when it is cost effective to do so. The goal is to choose how and when to deploy the server to minimise the sum of the costs incurred in holding customers in the system and those incurred in paying for service. This problem is formulated as a SMDP as follows:

(a) The state of the system at time $t \in \mathbb{R}^+$ is $N(t)$, the number of customers in the system. If t is a decision epoch and $N(t) > 0$ then two actions are available at t, labelled a (serve-active) and b (do not serve-passive). Choice of action a corresponds to the deployment of the server to process a waiting customer through to completion. In this case the next epoch is at time $t+S$ where we use S to denote a generic service time with associated distribution function G. We shall suppose that all moments of S exist. New customers arrive at the system according to a Poisson process with rate $\lambda > 0$, where $\lambda E(S) < 1$ for stability. According to standard $M/G/1$ dynamics we have that

$$P\big[N\big((t+S)^+\big) = n+m-1 \mid N(t) = m, a\big] = \int_0^\infty \frac{(\lambda t)^n}{n!}\, \mathrm{e}^{-\lambda t}\, \mathrm{d}G, \quad m \in \mathbb{Z}^+,\ n \in \mathbb{N}. \tag{14}$$

The choice of action b at t means that no service will be allocated to waiting customers for the period until the next customer arrives. In this case the next epoch is at time $t+X$ where $X \sim \exp(\lambda)$ and

$$P\big[N\big((t+X)^+\big) = m+1 \mid N(t) = m, b\big] = 1, \quad m \in \mathbb{N}. \tag{15}$$

Note that the passive action is the only admissible one when $N(t) = 0$.

(b) Let $C : \mathbb{N} \to \mathbb{R}^+$ be the increasing convex holding cost function for the class concerned and let α, W be positive constants. While the server is on, discounted costs are incurred at rate $\alpha C(n) + W$ when n customers are present in the system. This drops to $\alpha C(n)$ when the server is off. This is as in (11) above. Hence W is the rate charged for service, while $\alpha C(n)$ is the holding cost rate when there are n customers in the system.

(c) A policy is a rule for choosing between the actions a and b in light of the history of the system to date. If we use $I(t)$ to be the indicator function

$$I(t) = \begin{cases} 1, & \text{if the server is active at } t, \\ 0, & \text{otherwise, } t \in \mathbb{R}^+, \end{cases}$$

then we write the total expected cost incurred under policy u from initial state m as

$$V_u(m, \alpha, W) = E_u\left(\int_0^\infty \left[\alpha C\bigl(N(t)\bigr) + WI(t)\right]\mathrm{e}^{-\alpha t}\,\mathrm{d}t \mid N(0) = m\right). \tag{16}$$

The immediate goal of analysis is to determine a policy which will minimise the cost in (16). We write

$$V(m, \alpha, W) = \inf_u\bigl\{V_u(m, \alpha, W)\bigr\}. \tag{17}$$

The general theory of stochastic DP indicates the existence of an optimal policy which is stationary (i.e. makes decisions in light of the current state only) and whose value function satisfies the DP optimality equations. See [12]. Since the choice in any state m is between taking action a (until the next service completion) and taking action b (until the next arrival), the value function $V(\cdot, \alpha, W)$ satisfies

$$\begin{aligned} V(m, \alpha, W) = \min\Bigg\{ & \frac{\alpha C(m)}{\alpha + \lambda} + \frac{\lambda V(m+1, \alpha, W)}{\alpha + \lambda}; \widetilde{C}(m, \alpha) + \frac{WE(1 - \mathrm{e}^{-\alpha S})}{\alpha} \\ & + \sum_{n=0}^{\infty} \int_0^\infty \frac{(\lambda t)^n}{n!}\mathrm{e}^{-(\alpha+\lambda)t} V(n + m - 1, \alpha, W)\,\mathrm{d}G \Bigg\}, \quad m \in \mathbb{Z}^+. \end{aligned} \tag{18}$$

Note that in (18), $\widetilde{C}(m, \alpha)$ is the holding cost incurred during a single active period beginning at time 0 in state m, which we write as

$$\widetilde{C}(m, \alpha) = E\left[\int_0^S \alpha C\bigl(N(t)\bigr)\mathrm{e}^{-\alpha t}\,\mathrm{d}t \mid N(0) = m, a\right], \quad m \in \mathbb{Z}^+. \tag{19}$$

In fact, the analysis becomes a little cleaner if we substitute

$$\mathcal{V}(m, \alpha, W) = V(m, \alpha, W) - \frac{W}{\alpha}, \quad m \in \mathbb{N}, \tag{20}$$

in (18). Note that $\mathcal{V}(\cdot, \alpha, W)$ is the value function for an equivalent decision process, but where the cost rates for actions a and b in state n are $\alpha C(n)$ and $\alpha C(n) - W$, respectively. Note that W now has an interpretation as a *subsidy for passivity*. Rewriting (18) using (20), we obtain

$$\mathcal{V}(m,\alpha,W)=\min\left\{\frac{\alpha C(m)-W}{\alpha+\lambda}+\frac{\lambda\mathcal{V}(m+1,\alpha,W)}{\alpha+\lambda};\ \widetilde{C}(m,\alpha)\right.$$
$$\left.+\sum_{n=0}^{\infty}\int_0^{\infty}\frac{(\lambda t)^n}{n!}\mathrm{e}^{-(\alpha+\lambda)t}\mathcal{V}(n+m-1,\alpha,W)\,\mathrm{d}G\right\},\qquad m\in\mathbb{Z}^+. \tag{21}$$

Since passive is the only admissible action in state 0, we also have that

$$(\alpha+\lambda)\mathcal{V}(0,\alpha,W)=-W+\lambda\mathcal{V}(1,\alpha,W). \tag{22}$$

Following the discussion around definitions 1 and 2 of section 3, we write $\Pi_\alpha(W)$ for the set of states for which action b is optimal in the above problem. If we have α-indexability, namely that $\Pi_\alpha(W)$ is increasing in W, we then write $W_\alpha(m)$ for the Whittle α-index for the customer class concerned in state m, as in definition 2. We proceed to give a heuristic argument which yields a formula for $W_\alpha(m)$ in terms of model parameters when $W_\alpha(\cdot)$ is assumed to be an *increasing function* as would seem plausible.

Consider the service control problem (a)–(c) with $N(0)=m$, discount rate α and with service charge $W=\overline{W}_\alpha(m)$ equal to the *assumed* value of the α-index in state m. We make the assumptions (1) that the α-index is increasing in the state and (2) that when the service charge is equal to the α-index in some state, both a and b are optimal in that state. Both of these facts will be established properly later in the analysis. We now infer the following for this problem:

(i) the active action a must be optimal in states $\{m+1,m+2,\ldots\}$;

(ii) the passive action b must be optimal in states $\{0,1,\ldots,m-1\}$;

(iii) actions a and b are both optimal in state m.

Hence, under these assumptions there are two stationary policies which are optimal when $W=\overline{W}_\alpha(m)$. Label these policies u_1 and u_2. Policies u_1 and u_2 choose the actions a and b, respectively, in state m in addition to making choices according to (i) and (ii) above. Since $N(0)=m$, policy u_1 will take action a until time T where

$$T=\inf\bigl\{t;\ N(t)=m-1\bigr\}.$$

The cost incurred during this initial active phase may be written as

$$\overline{C}(m,\alpha)+\overline{W}_\alpha(m)\frac{E(1-\mathrm{e}^{-\alpha T})}{\alpha},$$

where

$$\overline{C}(m,\alpha)=E\left[\int_0^T\alpha C\bigl(N(t)\bigr)\mathrm{e}^{-\alpha t}\,\mathrm{d}t\mid N(0)=m,a\right]. \tag{23}$$

Note that random variable T is stochastically identical to the busy period of an $M/G/1$ queueing system, starting with a single customer and having arrival rate λ and generic customer service time S. Having arrived in state $m-1$ at time T, according to (ii) above policy u_1 now takes action b until a customer arrives, taking the system state back to m. This arrival will occur at time $T+X$ where $X \sim \exp(\lambda)$. The expected cost incurred during this passive phase is $E(e^{-\alpha T})\alpha C(m-1)(\alpha+\lambda)^{-1}$. Since $N((T+X)^+) = m$, policy u_1 now repeats the above cycle *ad infinitum* from time $T+X$. The total expected cost associated with this policy may now be calculated as

$$\mathcal{V}_{u_1}\{m, \alpha, \overline{W}_\alpha(m)\} = V_{u_1}\{m, \alpha, \overline{W}_\alpha(m)\} - \frac{\overline{W}_\alpha(m)}{\alpha}$$
$$= \frac{\overline{C}(m,\alpha) + E(e^{-\alpha T})\{\alpha C(m-1) - \overline{W}_\alpha(m)\}(\alpha+\lambda)^{-1}}{1 - \lambda E(e^{-\alpha T})(\alpha+\lambda)^{-1}}. \quad (24)$$

In addition, standard conditioning arguments yield

$$E(e^{-\alpha T}) = \sum_{n=0}^{\infty} \int_0^\infty \frac{(\lambda t)^n}{n!} e^{-(\alpha+\lambda)t} \{E(e^{-\alpha T})\}^n \, dG = \widetilde{G}[\alpha + \lambda\{1 - E(e^{-\alpha T})\}], \quad (25)$$

where

$$\widetilde{G}(\alpha) = \int_0^\infty e^{-\alpha t} \, dG,$$

and also

$$\overline{C}(m,\alpha) = \widetilde{C}(m,\alpha) + \sum_{n=1}^{\infty} \int_0^\infty \frac{(\lambda t)^n}{n!} e^{-(\alpha+\lambda)t} \left[\sum_{r=0}^{n-1} \overline{C}(m+r,\alpha) \{E(e^{-\alpha T})\}^{n-1-r} \right] dG. \quad (26)$$

Expression (26) disaggregates the total expected cost incurred during $[0, T)$ in (23) into that incurred during the processing of the first customer and the residual cost (if any) incurred by customers arriving during this initial service. Should n customers arrive, then $n+m-1$ customers will be present after the first service and successive busy periods will reduce the queue length such that

$$n+m-1 \to n+m-2 \to \cdots \to m \to m-1. \quad (27)$$

The second term on the right-hand side of (26) gives the expected cost associated with this residual processing.

Consider now policy u_2 which chooses passive action b in state m in addition to making choices according to (i) and (ii) above. Under u_2, the action b will be taken at time 0 and will remain in force for a period of time with an $exp(\lambda)$ distribution, at the conclusion of which a transition to state $m+1$ will occur. The expected cost incurred during this initial passive phase is easily shown to be $\alpha C(m)(\alpha+\lambda)^{-1}$. Thereafter, the active action will be taken until the queue length returns to m for the first time. This will take a further amount of time which is stochastically identical to T above. The expected cost incurred during this active phase is

$$\lambda\left\{\overline{C}(m+1,\alpha)+\overline{W}_\alpha(m)\frac{E(1-\mathrm{e}^{-\alpha T})}{\alpha}\right\}(\alpha+\lambda)^{-1}.$$

As with u_1, policy u_2 now repeats this cycle *ad infinitum*. We write the total expected cost associated with this policy as

$$\begin{aligned}\mathcal{V}_{u_2}\{m,\alpha,\overline{W}_\alpha(m)\} &= V_{u_2}\{m,\alpha,\overline{W}_\alpha(m)\}-\frac{\overline{W}_\alpha(m)}{\alpha}\\ &=\frac{\{\alpha C(m)-\overline{W}_\alpha(m)+\lambda\overline{C}(m+1,\alpha)\}(\alpha+\lambda)^{-1}}{1-\lambda E(\mathrm{e}^{-\alpha T})(\alpha+\lambda)^{-1}}.\end{aligned} \tag{28}$$

But both policies u_1 and u_2 are optimal when the service charge is $W=\overline{W}_\alpha(m)$ and hence it must follow from (24) and (28) that

$$\begin{aligned}&\mathcal{V}_{u_1}\{m,\alpha,\overline{W}_\alpha(m)\}=\mathcal{V}_{u_2}\{m,\alpha,\overline{W}_\alpha(m)\}\\ &\quad\Rightarrow\quad \overline{W}_\alpha(m)=\{\lambda\overline{C}(m+1,\alpha)-(\alpha+\lambda)\overline{C}(m,\alpha)+\alpha C(m)\\ &\qquad\qquad -\alpha E(\mathrm{e}^{-\alpha T})C(m-1)\}\{1-E(\mathrm{e}^{-\alpha T})\}^{-1},\quad m\in\mathbb{Z}^+.\end{aligned} \tag{29}$$

Hence it is the expression on the right-hand side of (29) which is the form of the α-index inferred from the above argument.

Lemma 2 asserts that our conjectured index $\overline{W}_\alpha(m)$ is increasing in m, as was supposed to be the case for the true index in the preceding argument. In lemma 2, we take $\overline{W}_\alpha(0)$ to be zero. Also, for economy of notation we shall write A for the quantity $E(\mathrm{e}^{-\alpha T})$ in what follows.

Lemma 2. $\overline{W}_\alpha(m)$ is increasing in m.

Proof. First, consider the quantity $\widetilde{C}(m,\alpha)$, defined in (19). By conditioning upon the times of successive arrivals after time 0, we obtain that

$$\begin{aligned}\widetilde{C}(m,\alpha)&=C(m)E(1-\mathrm{e}^{-\alpha S})+\sum_{n=1}^{\infty}\{C(n+m)-C(n+m-1)\}\\ &\quad\times\int_0^\infty\left[\int_0^s\frac{\lambda^n t^{n-1}\mathrm{e}^{-\lambda t}}{(n-1)!}\{\mathrm{e}^{-\alpha t}-\mathrm{e}^{-\alpha s}\}\,\mathrm{d}t\right]\mathrm{d}G \qquad (30)\\ &=C(m)E(1-\mathrm{e}^{-\alpha S})+\sum_{n=1}^{\infty}\{C(n+m)-C(n+m-1)\}\\ &\quad\times\left[\left(\frac{\lambda}{\alpha+\lambda}\right)^n E\left\{\sum_{r=n}^{\infty}\frac{(\alpha+\lambda)^r S^r\mathrm{e}^{-(\alpha+\lambda)S}}{r!}\right\}-E\left\{\sum_{r=n}^{\infty}\frac{\lambda^r S^r\mathrm{e}^{-(\alpha+\lambda)S}}{r!}\right\}\right], \qquad (31)\end{aligned}$$

where (31) follows from (30) by utilisation of the form of the distribution function of a $\Gamma(n, \lambda)$ random variable for $n \in \mathbb{Z}^+$.

We now use identity (26) in (29) to infer that, for $m \in \mathbb{Z}^+$,

$$\begin{aligned}(1-A)\overline{W}_\alpha(m) &= \lambda\overline{C}(m+1,\alpha) - (\alpha+\lambda)\overline{C}(m,\alpha) + \alpha C(m) - \alpha A C(m-1)\\ &= \lambda\widetilde{C}(m+1,\alpha) - (\alpha+\lambda)\widetilde{C}(m,\alpha) + \alpha C(m) - \alpha A C(m-1)\\ &\quad - \alpha\sum_{n=1}^{\infty} E\left\{\sum_{r=n}^{\infty}\frac{\lambda^r S^r \mathrm{e}^{-(\alpha+\lambda)S}}{r!}A^{r-n}\right\}\\ &\quad \times\left\{C(n+m-1) - AC(n+m-2)\right\}\\ &\quad + \sum_{n=1}^{\infty} E\left\{\sum_{r=n}^{\infty}\frac{\lambda^r S^r \mathrm{e}^{-(\alpha+\lambda)S}}{r!}A^{r-n}\right\}\\ &\quad \times\left\{\lambda\overline{C}(n+m,\alpha) - (\alpha+\lambda)\overline{C}(n+m-1,\alpha)\right.\\ &\quad \left. + \alpha C(n+m-1) - \alpha A C(n+m-2)\right\}. \end{aligned} \tag{32}$$

Using (29) and (31) within (32) it follows, after extensive but straightforward algebra that

$$\begin{aligned}\alpha^{-1}(1-A)\overline{W}_\alpha(m) &= \sum_{n=0}^{\infty} E\left\{\frac{\lambda^n S^n \mathrm{e}^{-(\alpha+\lambda)S}}{n!}\right\}\left\{C(n+m) - C(n+m-1)\right\}\\ &\quad + \sum_{n=1}^{\infty} E\left\{\sum_{r=n}^{\infty}\frac{\lambda^r S^r \mathrm{e}^{-(\alpha+\lambda)S}A^{r-n}}{r!}\right\}\left\{\alpha^{-1}(1-A)\overline{W}_\alpha(n+m-1)\right\}. \end{aligned} \tag{33}$$

However, identity (33) is strongly suggestive of the following computational scheme for $\alpha^{-1}(1-A)\overline{W}_\alpha(m),\ m \in \mathbb{Z}^+$: Use $\overline{W}_\alpha^R(\cdot)$ to denote the Rth iterate of the target function $\overline{W}_\alpha(\cdot)$. Take $\overline{W}_\alpha^1(m) = 0,\ m \in \mathbb{Z}^+$, and

$$\begin{aligned}&\alpha^{-1}(1-A)\overline{W}_\alpha^{R+1}(m)\\ &\quad = \sum_{n=0}^{\infty} E\left\{\frac{\lambda^n S^n \mathrm{e}^{-(\alpha+\lambda)S}}{n!}\right\}\left\{C(n+m) - C(n+m-1)\right\}\\ &\quad\quad + \sum_{n=1}^{\infty} E\left\{\sum_{r=n}^{\infty}\frac{\lambda^r S^r \mathrm{e}^{-(\alpha+\lambda)S}A^{r-n}}{r!}\right\}\left\{\alpha^{-1}(1-A)\overline{W}_\alpha^R(n+m-1)\right\}. \end{aligned} \tag{34}$$

From (34) it is a trival consequence of the increasing convex nature of $C(\cdot)$ that each iterate $\overline{W}_\alpha^R(\cdot)$ is an increasing function. Further, in this numerical scheme it is easy to demonstrate inductively that, for each fixed m, the sequence $\{\overline{W}_\alpha^R(m),\ R \in \mathbb{Z}^+\}$ is increasing in R and bounded above by $\overline{W}_\alpha(m)$. We use (33) and (34) and the choice of $\overline{W}_\alpha^1$ in the argument. It must then follow that $\overline{W}_\alpha^R(m) \to \phi_\alpha(m),\ R \to \infty$, where

$\phi_\alpha(m) \leqslant \overline{W}_\alpha(m)$, $m \in \mathbb{Z}^+$. That ϕ_α and $\overline{W}_\alpha$ must be identical is a consequence of the fact that

$$\sum_{n=1}^{\infty} E\left\{\sum_{r=n}^{\infty} \frac{\lambda^r S^r \mathrm{e}^{-(\alpha+\lambda)S} A^{r-n}}{r!}\right\} = \frac{E(\mathrm{e}^{-\alpha S} - A)}{1-A} < 1$$

together with the contraction mapping fixed point theorem. We now conclude that

$$\lim_{R\to\infty} \overline{W}_\alpha^R(m) = \overline{W}_\alpha(m), \quad m \in \mathbb{Z}^+. \tag{35}$$

Since each iterate $\overline{W}_\alpha^R(\cdot)$ is increasing, it follows that the limit function $\overline{W}_\alpha(\cdot)$ must also be. This concludes the proof of the lemma. □

We now proceed to theorem 1, which is the key result needed to establish both that the class is α-indexable and that the state m α-index is given by (29). The proof is long and utilises the methods of stochastic dynamic programming. It may be found in the appendix.

Theorem 1 (Optimal policy for the service control problem). If $\overline{W}_\alpha(m-1) \leqslant W < \overline{W}_\alpha(m)$ then the policy which chooses the passive action b in states $\{0, 1, \ldots, m-1\}$ and the active action a otherwise is optimal for our service control problem with service charge W, $m \in \mathbb{Z}^+$.

Careful study of the calculations in the proof of theorem 1 yield the conclusion that when $\overline{W}_\alpha(m-1) < W < \overline{W}_\alpha(m)$ the policy described in the statement of the theorem is strictly optimal. Suppose now that $W = \overline{W}_\alpha(m)$. It follows from theorem 1 that for this W-value, the policy which chooses the passive action in states $\{0, 1, \ldots, m\}$ and the active action otherwise is certainly optimal. In the heuristic argument preceding the statement of theorem 2, this is policy u_2. Recall that u_1 chooses the passive action in states $\{0, 1, \ldots, m-1\}$ and the active action otherwise. From (29) we have that

$$\mathcal{V}_{u_1}\{m, \alpha, \overline{W}_\alpha(m)\} = \mathcal{V}_{u_2}\{m, \alpha, \overline{W}_\alpha(m)\}.$$

From this and the fact that u_1 and u_2 take the same actions in all states other than m it follows easily that

$$\mathcal{V}_{u_1}\{n, \alpha, \overline{W}_\alpha(m)\} = \mathcal{V}_{u_2}\{n, \alpha, \overline{W}_\alpha(m)\}, \quad n \in \mathbb{N},$$

and hence that policy u_1 must also be optimal. It follows that when $W = \overline{W}_\alpha(m)$ both actions are optimal in state m. The following result is now immediate.

Theorem 2 (Indexability for the customer class). The customer class is α-indexable with Whittle α-index $W_\alpha(m) = \overline{W}_\alpha(m)$, $m \in \mathbb{N}$.

Proof. By theorem 1 and the preceding discussion we have that

$$\Pi_\alpha(\overline{W}) = \{0, 1, \ldots, m\}, \qquad \overline{W}_\alpha(m) \leqslant W < \overline{W}_\alpha(m+1), \quad m \in \mathbb{N}, \tag{36}$$

and the requirements of definition 1 are met, with α-indexability an immediate consequence. That $\overline{W}_\alpha(m)$ is the Whittle α-index for state m follows from (36) and definition 2. □

Comments

1. Hence the α-index is indeed given by expression (29). Observe that the proof of lemma 2 contains within it a method of computation for the index, expressed by (34). The subsequent discussion implies that the rate of convergence to the index will be geometric.

2. We now substantiate the claims made for the Langrangian relaxation in section 3 in the discussion preceding definition 1. Consider class k and its associated service allocation problem (k, α, W). Use $\{W^r_{k,\alpha};\ r = 0, 1, \ldots, R_k\}$ for the set of *distinct* index values for class k, numbered in ascending order. Note that $R_k + 1$ is the number of distinct index values, which may be infinite. Hence we have that $W^0_{k,\alpha} = W_{k,\alpha}(0) = 0$,

$$0 < W^1_{k,\alpha} < W^2_{k,\alpha} < \cdots$$

and

$$\left\{W^r_{k,\alpha};\ r = 0, 1, \ldots, R_k\right\} = \left\{W_{k,\alpha}(n);\ n \in \mathbb{N}\right\}.$$

For $W \notin \{W^r_{k,\alpha};\ r = 0, 1, \ldots, R_k\}$ use $u_k(W)$ for the unique optimal policy for the problem (k, α, W) as given by theorem 1. If $W = W^r_{k,\alpha}$ for some r then we use $u_k(W)$ to denote that optimal policy which chooses the active action in all states for which both actions are optimal. Developing the notation of section 3, we write

$$x^a_{k,n}(m_k, W) = E_{u_k(W)}\left[\int_0^\infty I\left\{a_k(t) = a,\ N_k(t) = n\right\} \mathrm{e}^{-\alpha t}\,\mathrm{d}t \mid N_k(0) = m_k\right]$$

for the associated active performance measures, with

$$\sum_{n \in \mathbb{N}} x^a_{k,n}(m_k, W) = E_{u_k(W)}\left[\int_0^\infty I\left\{a_k(t) = a\right\} \mathrm{e}^{-\alpha t}\,\mathrm{d}t \mid N_k(0) = m_k\right].$$

From the characterisation of $u_k(W)$ in theorem 1, it follows easily that for any choice of m_k and r, $0 \leqslant r \leqslant R_k - 1$,

$$\sum_{n \in \mathbb{N}} x^a_{k,n}(m_k, W) \tag{37}$$

is constant for $W \in (W^r_{k,\alpha}, W^{r+1}_{k,\alpha})$ since in this range $u_k(W)$ does not change. Further, it is left continuous such that for any r, $0 \leqslant r \leqslant R_k$,

$$\lim_{W \uparrow (W^r_{k,\alpha})} \sum_{n \in \mathbb{N}} x^a_{k,n}(m_k, W) > \sum_{n \in \mathbb{N}} x^a_{k,n}\big(m_k, \widehat{W}\big), \quad \widehat{W} > W^r_{k,\alpha}.$$

Finally, it is straightforward to show that

$$\sum_{n\in\mathbb{N}} x^{a}_{k,n}(m_k, W) \to 0, \quad W \to \infty.$$

To summarise, the quantity in (37) when regarded as a function of W is piecewise constant, decreasing with jump discontinuities at distinct index values and tends to 0 as W approaches infinity. These characteristics are inherited in the obvious way by the aggregated quantity

$$\sum_{k=1}^{K}\sum_{n\in\mathbb{N}} x^{a}_{k,n}(m_k, W) \equiv \sum_{k=1}^{K}\sum_{n=\mathbb{N}} x^{a}_{k,n}(\mathbf{m}, W)$$

which is the appropriate active performance measure for an optimal policy $\mathbf{u}(W)$ for the K-class stochastic optimisation problem in (11) obtained by superposition of the $u_k(W)$, $1 \leqslant k \leqslant K$ (i.e., independent operation of $u_k(W)$ for each class k). Further, it is a straightforward consequence of the fact that when $W = 0$, $u_k(W)$ takes the active action whenever class k is non-empty, that

$$\sum_{k=1}^{K}\sum_{n\in\mathbb{N}} x^{a}_{k,n}(\mathbf{m}, 0) > \alpha^{-1}\rho + \Theta(\mathbf{m}, \alpha), \tag{38}$$

where the constant $\Theta(\mathbf{m}, \alpha)$ is as given in (9). Now introduce $W(\mathbf{m}, \alpha)$ as

$$W(\mathbf{m}, \alpha) = \sup\left\{ W;\ \sum_{k=1}^{K}\sum_{n\in\mathbb{N}} x^{a}_{k,n}(\mathbf{m}, W) \geqslant \alpha^{-1}\rho + \Theta(\mathbf{m}, \alpha)\right\}.$$

By the above analysis, $W(\mathbf{m}, \alpha)$ must be an index value. Suppose that $W(\mathbf{m}, \alpha) = W^{r}_{k,\alpha}$. There are two possibilities. Either

$$\sum_{k=1}^{K}\sum_{n\in\mathbb{N}} x^{a}_{k,n}\{\mathbf{m}, W(\mathbf{m}, \alpha)\} = \alpha^{-1}\rho + \Theta(\mathbf{m}, \alpha)$$

in which case policy $\mathbf{u}\{W(\mathbf{m}, \alpha)\}$ is optimal for the Lagrangian relaxation in (11) with $W = W(\mathbf{m}, \alpha)$, satisfies the constraint in (10) and hence solves Whittle's relaxation. Alternatively

$$\sum_{k=1}^{K}\sum_{n\in\mathbb{N}} x^{a}_{k,n}\{\mathbf{m}, W(\mathbf{m}, \alpha)\} > \alpha^{-1}\rho + \Theta(\mathbf{m}, \alpha)$$

in which case the same claims can be made for some randomisation between $\mathbf{u}\{W(\mathbf{m}, \alpha)\}$ and a modification of it which replaces the active action by passive in class k states whose index is $W^{r}_{k,\alpha}$.

3. Following theorem 2 and the discussion in section 3, an index policy for the K-class problem with discounted costs of section 2 is constructed by computing the

index function $W_{k,\alpha}(\cdot)$ for each customer class k from an appropriate form of (29). At each epoch t, the policy serves a customer from a non-empty class with maximal index $W_{k,\alpha}\{N_k(t)\}$.

We again drop the class identifier k and observe that we can now develop a suitable Whittle index $W:\mathbb{N} \to \mathbb{R}^+$ for the average cost problem from the limit of the corresponding α-index

$$W(m) = \lim_{\alpha\to 0} W_\alpha(m) = \lim_{\alpha\to 0} \overline{W}_\alpha(m), \quad m \in \mathbb{N}, \tag{39}$$

as in definition 3. Utilising (39) within (29), we obtain the following result.

Theorem 3 (The Whittle index for average costs). The Whittle index for the average cost problem is given by $W(0) = 0$ and

$$W(m) = \frac{\lambda\{\overline{C}(m+1) - \overline{C}(m)\} + C(m) - C(m-1)}{E(T)}, \quad m \in \mathbb{Z}^+, \tag{40}$$

$$= \frac{E\{C(N+m)\} - E\{C(N+m-1)\}}{E(S)}, \quad m \in \mathbb{Z}^+, \tag{41}$$

where in (40) we have

$$\overline{C}(m) = \lim_{\alpha\to 0} \alpha^{-1}\overline{C}(m,\alpha) = E\left[\int_0^T C\big(N(t)\big)\,\mathrm{d}t \mid N(0) = m, a\right], \quad m \in \mathbb{Z}^+,$$

and in (41), the random variable N has the steady-state distribution of the number of customers present in the single class $M/G/1$ system with non-idling service.

Proof. The form of the index in (40) follows readily from earlier results. To obtain (41), observe that it follows readily from the definitions of the quantities concerned and standard results concerning regenerative processes that

$$E\big\{C(N+m)\big\} = \big\{\overline{C}(m+1) + C(m)\lambda^{-1}\big\}\big\{E(T) + \lambda^{-1}\big\}^{-1}, \tag{42}$$

where

$$E(T) = E(S)\big\{1 - \lambda E(S)\big\}^{-1}. \tag{43}$$

Expression (41) follows now from (40), (42) and (43). □

Comment

Following theorem 3 and the discussion in section 3, an index policy for the K-class service control problem with average costs described in (5) of section 2 is constructed by computing the index function $W_k(\cdot)$ for each customer class k from an appropriate form of (41). The required (steady-state) distribution of a single class $M/G/1$ system is available by standard methods. At each epoch t, the index policy serves a customer from a non-empty class with maximal index $W_k\{N_k(t)\}$.

5. Numerical investigation of index policies for multiclass $M/G/1$ systems

Utilisation of the Lagrangian relaxation of section 3 has yielded a class of index heuristics for the multiclass service control problems of section 2 via the study of single class problems with service charge. An index for the discounted costs problem of (3) is obtained as a *fair charge for service* with an appropriate index for the average costs problem of (5) obtained as a limit. We now investigate the performance of the index heuristics numerically. While our prime focus will be on average costs problems we begin with a study of some two class problems with discounted costs.

5.1. Discounted costs problems with two customer classes

Table 1 below contains the results of part of a study comparing the discounted costs incurred by the index heuristic described in comment 3 following theorem 2 with those incurred by an optimal policy for a range of service control problems with two customer classes. Each cell of the table gives results for four different cost structures in the form

$$\begin{array}{cccc} a & (a) & b & (b) \\ c & (c) & d & (d). \end{array}$$

The corresponding class cost rates are as follows:

(a) $C_1(n) = b_1 n + 2n^2$; $C_2(n) = b_2 n + 2n^2$ (quadratic);

(b) $C_1(n) = b_1 n^2 + 2n^3$; $C_2(n) = b_2 n^2 + 2n^3$ (cubic);

(c) $C_1(n) = b_1 n^3 + 2n^4$; $C_2(n) = b_2 n^3 + 2n^4$ (quartic);

(d) $C_1(n) = b_1(n-2)^+ + 2\{(n-2)^+\}^2$; $C_2(n) = b_2(n-2)^+ + 2\{(n-2)^+\}^2$ (shifted quadratic).

In all cells of the table the unbracketed figure (a,b,c or d) is the discounted cost for the index policy beginning at time zero from an empty system, with the corresponding optimal cost in brackets. Note that the optimal costs given in table 1 are $\alpha^{-1}\mathbf{V}(\mathbf{m}, \alpha)$ with $\alpha = 0.05129$ ($e^{-\alpha} = 0.95$), namely the value of the total discounted costs without incorporation of the multiplier α in (3). Corresponding values for the discounted costs associated with the index heuristic are also given. All figures were obtained by use of DP value iteration. This is possible to implement for problems of this size, but computationally expensive. In the left-hand column of table 1, the first two entries in the bracketed triple indicate respectively the choice of cost coefficients b_1, b_2 with the final labels 1, $1'$, 2 and $2'$ specifying features of the stochastic structure. The labels 1, $1'$ correspond to problems for which $S_1 \sim \Gamma(2, 1.25)$, $S_2 \sim \Gamma(3, 2.25)$ and $\lambda_1 = 0.20$. For case 1, λ_2 is chosen such that the value of the traffic intensity ρ is 0.60, while for case $1'$, ρ is set to be 0.85. The labels 2, $2'$ correspond to problems with $S_1 \sim \Gamma(2, 1)$ and $S_2 \sim \Gamma(3, 3)$ and $\lambda_1 = 0.20$. Hence the mean service times are further apart than in 1, $1'$. Again for case 2, λ_2 is chosen to yield $\rho = 0.60$ while for $2'$ we have $\rho = 0.85$.

Table 1
Comparative performance of the index heuristic and an optimal policy for a range of discounted costs problems with two customer classes.

(2,1,1)	97.9099 (97.8780)	211.9724 (211.9462)
	598.9984 (598.9022)	7.2059 (7.2047)
(1,2,1)	96.3177 (96.2575)	208.0710 (207.7231)
	585.8202 (585.5572)	6.8896 (6.8874)
(2,1,1′)	268.1926 (263.7965)	932.9235 (914.5342)
	4254.7857 (4167.4051)	53.2241 (52.1260)
(1,2,1′)	270.5770 (267.7749)	949.8748 (924.7464)
	4376.4714 (4205.4300)	54.4016 (53.3628)
(2,1,2)	102.1608 (101.9858)	229.8926 (229.8295)
	681.1326 (681.1326)	8.6761 (8.4078)
(1,2,2)	98.4711 (97.6724)	227.1538 (226.5507)
	693.9780 (693.6048)	8.6704 (8.3285)
(2,1,2′)	261.9274 (261.7704)	917.1907 (916.9421)
	4236.5454 (4233.1641)	52.8168 (52.7911)
(1,2,2′)	261.8411 (261.6491)	917.7822 (915.7203)
	4239.4764 (4221.4696)	52.9643 (52.8245)

5.2. *Average costs problems with two customer classes*

Table 2 below contains the results of part of a study comparing the average costs incurred by the index heuristic described in the comment following theorem 3 with those incurred by an optimal policy. All service control problems studied have two customer classes. Each cell in the body of the table gives results for four different cost structures in the form

$$\begin{matrix} a & (a) & b & (b) \\ c & (c) & d & (d). \end{matrix}$$

The corresponding class cost rates are as follows:

(a) $C_1(n) = 2n + c_1n^2$; $C_2(n) = n + c_2n^2$ (quadratic);

(b) $C_1(n) = 2n^2 + c_1n^3$; $C_2(n) = n^2 + c_2n^3$ (cubic);

(c) $C_1(n) = 2n^3 + c_1n^4$; $C_2(n) = n^3 + c_1n^4$ (quartic);

(d) $C_1(n) = 2(n-2)^+ + c_1\{(n-2)^+\}^2$; $C_2(n) = (n-2)^+ + c_2\{(n-2)^+\}^2$ (shifted quadratic).

In all cases the unbracketed figure (a, b, c or d) is the time average cost (in (5)) with the corresponding optimal cost in brackets. All costs were obtained by use of DP value iteration. In the left-hand column of table 2, the entries are the cost coefficients c_1, c_2 which apply to the values in the corresponding row. In the main body of the table each left-hand cell concerns a server control problem with $S_1 \sim \Gamma(2, 1.25)$, $S_2 \sim \Gamma(3, 2.25)$, $\lambda_1 = 0.20$ and λ_2 chosen to give a traffic intensity of 0.60. The value of λ_2 is modified for each right-hand cell to give a traffic intensity of 0.85.

Table 2
Comparative performance of the index heuristic and an optimal policy for a range of average costs problems with two customer classes.

0.10 0.10	2.0727 (2.0727)	4.2932 (4.2930)	7.7935 (7.7928)	39.9968 (39.9819)
	11.7501 (11.7501)	0.2160 (0.2160)	289.3492 (288.5327)	3.2089 (3.2086)
0.10 0.20	2.2337 (2.2334)	4.6542 (4.6531)	8.9225 (8.9122)	46.5325 (46.5011)
	12.9877 (12.9834)	0.2318 (0.2318)	343.8621 (343.2882)	3.6701 (3.6661)
0.10 0.50	2.5564 (2.5530)	5.4729 (5.4729)	10.4631 (10.4407)	58.7729 (58.5350)
	15.9330 (15.5675)	0.2688 (0.2661)	448.4251 (446.8978)	4.3366 (4.3228)
0.10 1.00	2.9461 (2.9458)	6.5898 (6.5808)	12.2439 (12.2382)	71.3354 (70.9773)
	19.2520 (19.1304)	0.3122 (0.3019)	555.7236 (554.9071)	5.0380 (4.9689)
0.10 2.00	3.7181 (3.7181)	8.4563 (8.4269)	15.7226 (15.7224)	89.1588 (88.6533)
	25.6285 (24.9345)	0.3729 (0.3715)	702.9505 (698.2963)	6.0162 (5.9198)
0.20 0.10	2.1649 (2.1648)	4.5764 (4.5764)	8.2195 (8.2195)	44.2131 (44.2127)
	12.7201 (12.7201)	0.2253 (0.2253)	328.6532 (328.3382)	3.3805 (3.3804)
0.20 0.20	2.3407 (2.3407)	4.9506 (4.9506)	9.8432 (9.8335)	52.6082 (52.6079)
	13.9763 (13.9763)	0.2420 (0.2420)	396.5049 (396.5009)	4.0348 (4.0311)
0.20 0.50	2.7172 (2.7107)	5.9251 (5.8729)	12.1076 (12.0930)	68.5622 (68.2829)
	17.1763 (16.9382)	0.2825 (0.2819)	530.7424 (529.6970)	4.9834 (4.9777)
0.20 1.00	3.1339 (3.1325)	7.0314 (7.0313)	14.2092 (14.1831)	84.6577 (84.0423)
	20.7434 (20.7342)	0.3288 (0.3200)	670.5373 (667.4028)	5.8720 (5.8022)
0.20 2.00	3.9077 (3.9075)	8.9965 (8.9851)	17.7387 (17.7320)	105.5860 (105.1594)
	27.3193 (26.9502)	0.3911 (0.3911)	850.6627 (846.7620)	6.9959 (6.9090)
0.50 0.10	2.4343 (2.4343)	5.3592 (5.3592)	8.8808 (8.8792)	52.7127 (52.7122)
	15.4227 (15.4227)	0.2525 (0.2525)	413.0458 (413.0351)	3.6408 (3.6402)
0.50 0.20	2.6317 (2.6317)	5.8035 (5.8035)	11.3447 (11.3446)	65.5261 (65.4681)
	16.8787 (16.8725)	0.2707 (0.2707)	515.2392 (515.1302)	4.6367 (4.6367)
0.50 0.50	3.0962 (3.0961)	6.9233 (6.9231)	15.6224 (15.5923)	90.0431 (90.0192)
	20.6543 (20.6528)	0.3169 (0.3169)	720.0842 (719.3984)	6.3695 (6.3582)
0.50 1.00	3.6383 (3.6377)	8.2931 (8.2880)	19.1765 (19.1429)	115.1593 (114.8845)
	25.0046 (25.0046)	0.3789 (0.3704)	932.7753 (931.1808)	7.8445 (7.8266)
0.50 2.00	4.4710 (4.4683)	10.5650 (10.5287)	23.5551 (23.5206)	146.3636 (146.3157)
	32.4936 (32.2729)	0.4453 (0.4451)	1209.4349 (1204.6509)	9.5578 (9.4703)
1.00 0.10	2.8802 (2.8802)	6.5241 (6.5241)	9.5440 (9.5418)	61.4237 (61.3950)
	19.3929 (19.3929)	0.2977 (0.2963)	505.1724 (505.1617)	3.9002 (3.8991)
1.00 0.20	3.0861 (3.0859)	7.0706 (7.0706)	12.5758 (12.5756)	78.8265 (78.8256)
	21.2598 (21.2598)	0.3165 (0.3165)	647.5939 (647.3051)	5.1201 (5.1200)
1.00 0.50	3.6352 (3.6352)	8.3852 (8.3852)	18.9538 (18.9537)	114.5855 (114.5529)
	25.5310 (25.5310)	0.3682 (0.3682)	937.6228 (937.5127)	7.6957 (7.6956)
1.00 1.00	4.3379 (4.3365)	10.2027 (10.1928)	25.0574 (25.0438)	152.2819 (152.1247)
	31.7557 (31.6629)	0.4405 (0.4404)	1257.7162 (1251.5248)	10.1860 (10.1807)
1.00 2.00	5.3396 (5.3291)	12.7169 (12.7169)	31.8017 (31.7742)	199.3951 (198.8927)
	39.7882 (39.7414)	0.5461 (0.5304)	1660.0632 (1655.7462)	12.9627 (12.8847)
2.00 0.10	3.7713 (3.7712)	8.7142 (8.7120)	10.6295 (10.6279)	71.9051 (71.7997)
	26.6031 (26.6027)	0.3734 (0.3721)	622.5756 (622.5390)	4.2643 (4.2283)
2.00 0.20	3.9790 (3.9790)	9.3873 (9.3649)	14.0270 (14.0257)	95.1909 (95.1781)
	28.8583 (28.8583)	0.4010 (0.4001)	819.2224 (819.2021)	5.6788 (5.6604)
2.00 0.50	4.5853 (4.5834)	11.0135 (11.0135)	22.4480 (22.4478)	145.9129 (145.9097)
	34.5466 (34.5465)	0.4625 (0.4623)	1234.8649 (1234.8568)	9.0829 (9.0827)
2.00 1.00	5.4561 (5.4561)	13.1438 (13.1438)	32.1849 (32.1848)	202.4307 (202.3046)
	41.5439 (41.5438)	0.5457 (0.5457)	1699.1895 (1698.4475)	13.0174 (13.0173)
2.00 2.00	6.8041 (6.7923)	16.7250 (16.5944)	43.9525 (43.9089)	276.4430 (275.6991)
	53.8595 (53.1546)	0.6866 (0.6844)	2326.4592 (2312.4604)	17.8278 (17.7953)

5.3. *Simulation study of average costs problems with five customer classes*

While it was possible to obtain a direct numerical comparison between costs incurred by our index heuristics and those incurred by an optimal policy for the two class problems in (i) and (ii), this is not a reasonable computational goal for larger problems. The simulation study reported in tables 3 and 4 concerns a collection of service control problems involving five customer classes under the average cost criterion.

Table 3 contains the results of studies of ten problems with quadratic costs (1–5, $1'$–$5'$) and five problems with quartic costs (1–5). All problems feature deterministic service times. Each of the problems with quadratic costs is characterised by four five-

Table 3
Comparative performance of the index heuristic and four other service control rules for a range of average costs problems with five customer classes and deterministic service times.

Quadratic costs	INDEX	LQ	MYOPIC	MYOPIC*	RANDOM
1	6.7103	6.9759	6.8919	7.2142	7.0933
	(0.0358)	(0.0394)	(0.0449)	(0.0496)	(0.0507)
2	6.9778	7.4549	7.3399	7.6648	7.7825
	(0.0430)	(0.0568)	(0.0550)	(0.0645)	(0.0840)
3	7.1444	7.8734	7.8815	7.9003	8.6498
	(0.0489)	(0.0601)	(0.0475)	(0.0531)	(0.0778)
4	7.3377	7.9216	7.7673	7.9249	8.7709
	(0.0423)	(0.0585)	(0.0541)	(0.0632)	(0.1152)
5	7.2164	7.6448	7.6566	7.7806	8.2742
	(0.0493)	(0.0489)	(0.0451)	(0.0497)	(0.1077)
$1'$	23.2539	25.5787	24.0424	28.3180	28.9243
	(0.4346)	(0.4844)	(0.5170)	(0.5113)	(0.5900)
$2'$	25.2815	30.7615	27.9366	30.3640	39.7180
	(0.5172)	(0.8053)	(0.4614)	(0.4835)	(1.0815)
$3'$	24.7591	33.8409	29.4795	32.1201	83.3331
	(0.4060)	(0.6157)	(0.4755)	(0.4777)	(3.4087)
$4'$	25.6866	31.1344	30.1719	30.2082	72.1357
	(0.3649)	(0.6197)	(0.4898)	(0.4667)	(2.6194)
$5'$	26.3250	29.7588	29.3930	29.5962	55.3344
	(0.5261)	(0.4981)	(0.5977)	(0.4620)	(2.0550)
Quartic costs					
1	15.5772	15.7914	16.0158	17.8664	22.3649
	(0.1703)	(0.1851)	(0.2282)	(0.2050)	(0.5133)
2	17.2057	18.6310	18.2118	20.2739	25.5776
	(0.1691)	(0.2237)	(0.2003)	(0.2744)	(0.6412)
3	18.2476	22.2612	21.6834	22.1398	42.3787
	(0.2390)	(0.2658)	(0.3997)	(0.2661)	(1.9690)
4	19.4305	22.8196	23.1101	22.2155	49.2510
	(0.2524)	(0.3014)	(0.3425)	(0.3057)	(6.2762)
5	18.5410	21.9044	22.1773	21.4857	40.9507
	(0.2185)	(0.3103)	(0.2912)	(0.3282)	(2.2664)

vectors $\mathbf{b}, \mathbf{c}, \boldsymbol{\lambda}$ and $\mathbf{S}$. Both $\mathbf{b}$ and $\mathbf{c}$ are vectors of cost coefficients such that the class k cost rate is given by

$$C_k(n) = b_k n + c_k n^2, \quad 1 \leqslant k \leqslant 5, \tag{44}$$

while $\boldsymbol{\lambda}$ is a vector of arrival rates with λ_k the rate for class k. Finally, $\mathbf{S}$ is a vector of deterministic service times. For example, for quadratic problem 1 we take $\mathbf{b} = (5, 4, 3, 2, 1)$, $\mathbf{c} = (1, 2, 3, 4, 5)$, $\boldsymbol{\lambda} = (0.40, 0.30, 0.25, 0.10, 0.05)$ and $\mathbf{S} = (0.6, 0.5, 0.4, 0.7, 0.8)$ with a resulting traffic intensity of 0.60. To obtain quadratic problems 2–5 we keep $\boldsymbol{\lambda}$ and $\mathbf{S}$ fixed, but reassign the cost coefficients by means of a series of permutations. For example, for problem 2 we take $\mathbf{b} = (1, 5, 4, 3, 2)$ and $\mathbf{c} = (5, 1, 2, 3, 4)$ and so on. We obtain quadratic problems $1'$–$5'$ respectively from 1–5 by rescaling $\boldsymbol{\lambda}$ to give a traffic intensity of 0.85, while keeping other aspects fixed. We obtain quartic problems 1–5 from the corresponding quadratic problems upon replacing (44) by

$$C_k(n) = b_k n^3 + c_k n^4, \quad 1 \leqslant k \leqslant 5.$$

In the body of table 3 find estimates of the average costs incurred by the above problems under five service control heuristics, as follows: INDEX denotes our index heuristic for average costs while LQ allocates service at each decision epoch to whichever customer class has the longest queue (and chooses among the candidate classes at random in the event of a tie). MYOPIC always chooses for processing whichever customer class is incurring the largest instantaneous cost rate and MYOPIC* modifies this criterion by dividing the instantaneous cost rate by the corresponding mean service time. At each decision epoch RANDOM chooses one of the non-empty customer classes at random (with equal probabilities) and serves a single customer from the class chosen. The estimate of average cost is obtained in each case by Monte Carlo simulation. Typically, we allowed a "burn-in" period of around 10,000 time units in each case, followed by a period of around 15,000 time units during which costs were tracked. This was repeated around 50 times and average costs (per unit time) estimated as given. The corresponding standard errors are given in brackets in the table. The details of the mechanics of the simulations varied a little across the different cases in order to obtain standard errors which would enable meaningful comparisons between service policies to be made. Note that we did not have access to sufficient computer resources for this to be achieved for problems with quartic costs and a traffic intensity of 0.85. This is why no such cases are reported in the table.

The study reported in table 4 mirrors that in table 3 and differs only in that service times are now gamma distributed. Hence, for quadratic problem 1 the single five-vector $\mathbf{S}$ of deterministic times is replaced by two five-vectors $\mathbf{m} = (1, 3, 2, 4, 5)$ and $\boldsymbol{\mu} = (5/3, 6, 5, 40/7, 25/4)$. We now suppose that $S_k \sim \Gamma(m_k, \mu_k)$, $1 \leqslant k \leqslant 5$. All other details are as in the study in table 3.

Table 4
Comparative performance of the index heuristic and four other service control rules for a range of average costs problems with four customer classes and gamma distributed service times.

Quadratic costs	INDEX	LQ	MYOPIC	MYOPIC*	RANDOM
1	8.9812	9.3200	9.3366	9.3885	9.5438
	(0.0941)	(0.0733)	(0.0894)	(0.0917)	(0.0878)
2	9.5892	10.2201	10.2700	10.0731	11.1100
	(0.1010)	(0.0860)	(0.1506)	(0.0935)	(0.1380)
3	9.9218	11.2622	10.9091	11.1442	13.9702
	(0.0904)	(0.0970)	(0.1127)	(0.1143)	(0.2522)
4	10.2312	10.9974	10.7825	11.0971	13.3023
	(0.1098)	(0.1136)	(0.0866)	(0.0997)	(0.4585)
5	10.0943	10.7580	10.6351	11.2773	12.4465
	(0.1153)	(0.0962)	(0.1296)	(0.1306)	(0.1832)
$1'$	39.4936	45.6291	42.0555	41.1953	58.1367
	(1.3472)	(1.2900)	(1.0080)	(0.9626)	(3.0910)
$2'$	44.1563	52.1205	49.7436	52.9404	86.0343
	(1.1356)	(1.1165)	(1.0747)	(1.4466)	(2.9641)
$3'$	42.5420	60.9430	53.6382	54.9029	187.7974
	(0.9720)	(1.6908)	(1.4915)	(1.2248)	(10.9604)
$4'$	47.2808	56.1806	52.0994	58.2293	157.9946
	(1.1669)	(1.1536)	(1.4938)	(1.3649)	(6.5433)
$5'$	45.9588	52.8616	49.0092	57.8623	113.7342
	(1.4101)	(1.5572)	(1.1121)	(1.4052)	(3.9718)
Quartic costs					
1	34.4928	33.7941	33.3589	38.0270	60.5706
	(0.8522)	(0.7745)	(0.7173)	(0.8749)	(2.8492)
2	39.1317	41.1258	40.5730	44.3442	72.3138
	(0.7614)	41.1258	(0.7935)	(1.0462)	(3.2612)
3	42.9542	49.1543	48.4376	50.3789	150.1279
	(0.9132)	(0.9074)	(1.2642)	(1.4622)	(11.2225)
4	45.4567	53.0129	51.2151	52.2439	144.0640
	(1.2018)	(1.0876)	(1.0783)	(1.0613)	(7.8021)
5	43.9029	54.1418	48.5072	54.1950	113.3488
	(0.8862)	(1.4610)	(0.9447)	(1.1625)	(4.9810)

5.4. *Comments*

Please note the very strong performance of our index heuristics throughout the above study. In the average cost problems reported in table 2 the index heuristic is indistinguishable from optimal for many cases. The highest degree of suboptimality observed throughout tables 1 and 2 is 4%. Tables 3 and 4 contain compelling evidence that this strong performance carries over to larger problems. In 29 of the 30 cases reported, the index heuristic outperforms the other service control rules studied, in many instances clearly so. In the exceptional case, the degree of inferiority of the heuristic is not statistically significant. The most consistent of the competitor policies is MYOPIC, but even this incurs costs which exceed that of INDEX by over 25% on occasion. In general, the degree of cost superiority of INDEX over the competitor heuristics appears to grow with ρ.

Acknowledgements

The authors express appreciation to the Engineering and Physical Research Council for its support and to Professor Niño-Mora for his insightful comments. They are also grateful to referees for a range of comments which stimulated crucial additional work.

Appendix

Theorem 1 (Optimal policy for the service control problem). If $\overline{W}_\alpha(m-1) \leqslant W < \overline{W}_\alpha(m)$ then the policy which chooses the passive action b in states $\{0, 1, \ldots, m-1\}$ and the active action a otherwise is optimal for our service control problem with service charge W, $m \in \mathbb{Z}^+$.

Proof. We use $\overline{V}(\cdot, \alpha, W)$ to denote the value function for the policy $\bar{u}$ described in the statement of the theorem. We write

$$\overline{\mathcal{V}}(n, \alpha, W) = \overline{V}(n, \alpha, W) - \frac{W}{\alpha}, \quad n \in \mathbb{N}.$$

By standard DP theory, it remains to show that $\overline{V}(\cdot, \alpha, W)$ satisfies the optimality equations (18). From (21) and straightforward algebra, it suffices to show that when $\overline{W}_\alpha(m-1) \leqslant W < \overline{W}_\alpha(m)$ we have that

$$\begin{aligned} W \leqslant{}& \alpha C(n) + \lambda \overline{\mathcal{V}}(n+1, \alpha, W) - (\alpha+\lambda)\widetilde{C}(n, \alpha) \\ &- (\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^\infty \frac{(\lambda t)^r}{r!}\,\mathrm{e}^{-(\alpha+\lambda)t}\overline{\mathcal{V}}(n+r-1, \alpha, W)\,\mathrm{d}G, \quad n \geqslant m, \end{aligned} \tag{A.1}$$

and

$$\begin{aligned} W \geqslant{}& \alpha C(n) + \lambda \overline{\mathcal{V}}(n+1, \alpha, W) - (\alpha+\lambda)\widetilde{C}(n, \alpha) \\ &- (\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^\infty \frac{(\lambda t)^r}{r!}\,\mathrm{e}^{-(\alpha+\lambda)t}\overline{\mathcal{V}}(n+r-1, \alpha, W)\,\mathrm{d}G, \quad 1 \leqslant n \leqslant m-1. \end{aligned} \tag{A.2}$$

We shall demonstrate that (A.1) and (A.2) hold by considering four cases in turn.

(1) $n = m$. Policy $\bar{u}$ chooses the active action at states m and above. Hence, by considering costs incurred within the first service and beyond it, the total cost $\overline{\mathcal{V}}(m, \alpha, W)$ may be written

$$\overline{\mathcal{V}}(m, \alpha, W) = \widetilde{C}(m, \alpha) + \sum_{r=0}^{\infty} \int_0^{\infty} \frac{(\lambda t)^r}{r!} \mathrm{e}^{-(\alpha+\lambda)t} \overline{\mathcal{V}}(m + r - 1, \alpha, W) \, \mathrm{d}G. \tag{A.3}$$

But, upon utilising (A.3), the form of (A.1) required for the case $n = m$ becomes

$$W \leqslant \alpha C(m) + \lambda \overline{\mathcal{V}}(m + 1, \alpha, W) - (\alpha + \lambda) \overline{\mathcal{V}}(m, \alpha, W). \tag{A.4}$$

We also have that

$$\overline{\mathcal{V}}(m + 1, \alpha, W) = \overline{C}(m + 1, \alpha) + A \overline{\mathcal{V}}(m, \alpha, W), \tag{A.5}$$

where recall that $A = E(\mathrm{e}^{-\alpha T})$. Moreover, a calculation akin to that which yielded (24) gives

$$\overline{\mathcal{V}}(m, \alpha, W) = \frac{\overline{C}(m, \alpha) + A\{\alpha C(m - 1) - W\}(\alpha + \lambda)^{-1}}{1 - \lambda A(\alpha + \lambda)^{-1}}. \tag{A.6}$$

From (A.5) and (A.6) we have that

$$\begin{aligned}
&\alpha C(m) + \lambda \overline{\mathcal{V}}(m + 1, \alpha, W) - (\alpha + \lambda) \overline{\mathcal{V}}(m, \alpha, W) \\
&\quad = \alpha C(m) + \lambda \overline{C}(m + 1, \alpha) + \big\{\lambda A - (\alpha + \lambda)\big\} \overline{\mathcal{V}}(m, \alpha, W) \\
&\quad = \alpha C(m) + \lambda \overline{C}(m + 1, \alpha) - (\alpha + \lambda) \overline{C}(m, \alpha) - A\big\{\alpha C(m - 1) - W\big\} \\
&\quad = (1 - A) \overline{W}_{\alpha}(m) + AW,
\end{aligned} \tag{A.7}$$

using (29). But it is plain from the hypotheses of the theorem that the expression in (A.7) exceeds W and (A.4) is established.

(2) $n \geqslant m + 1$. Fix state $M \geqslant m + 1$. From (A.1) we require that

$$\begin{aligned}
W \leqslant\ &\alpha C(M) + \lambda \overline{\mathcal{V}}(M + 1, \alpha, W) - (\alpha + \lambda) \widetilde{C}(M, \alpha) \\
&- (\alpha + \lambda) \sum_{r=0}^{\infty} \int_0^{\infty} \frac{(\lambda t)^r}{r!} \mathrm{e}^{-(\alpha+\lambda)t} \overline{\mathcal{V}}(M + r - 1, \alpha, W) \, \mathrm{d}G.
\end{aligned} \tag{A.8}$$

In what follows, we shall use $u(n)$ to denote the policy which chooses the active action at states n and above with the passive action chosen otherwise and $\mathcal{V}^{(n)}$ for the corresponding costs. Note that $u(m) \equiv \bar{u}$ and $\mathcal{V}^{(m)} \equiv \overline{\mathcal{V}}$. By calculations similar to those which yielded (28) we conclude that

$$\mathcal{V}^{(n+1)}(n, \alpha, W) = \big\{\alpha C(n) - W + \lambda \overline{C}(n + 1, \alpha)\big\}(\alpha + \lambda - \lambda A)^{-1}, \quad n \in \mathbb{N}. \tag{A.9}$$

Combining a version of (A.6) with n replacing m and (A.9) we deduce that

$$\begin{aligned}
\mathcal{V}^{(n)}(n, \alpha, W) - \mathcal{V}^{(n+1)}(n, \alpha, W) &= \big[-\lambda \overline{C}(n + 1, \alpha) + (\alpha + \lambda) \overline{C}(n, \alpha) \\
&\quad - \alpha C(n) + \alpha A C(n - 1) + W(1 - A)\big](\alpha + \lambda - \lambda A)^{-1} \\
&= \big\{W - \overline{W}_{\alpha}(n)\big\}(1 - A)(\alpha + \lambda - \lambda A)^{-1}, \quad n \in \mathbb{N}.
\end{aligned} \tag{A.10}$$

Now let $r \in \mathbb{Z}^+$ and consider policies $u(n)$ and $u(n+1)$ operating from initial state $n+r$. Since each begins with a busy period during which the active action is taken, we have that

$$\mathcal{V}^{(n)}(n+r,\alpha,W) = \overline{C}(n+r,\alpha) + A\mathcal{V}^{(n)}(n+r-1,\alpha,W) \tag{A.11}$$

and

$$\mathcal{V}^{(n+1)}(n+r,\alpha,W) = \overline{C}(n+r,\alpha) + A\mathcal{V}^{(n+1)}(n+r-1,\alpha,W). \tag{A.12}$$

It is a straightforward consequence of (A.10)–(A.12) that

$$\begin{aligned}
&\mathcal{V}^{(n)}(n+r,\alpha,W) - \mathcal{V}^{(n+1)}(n+r,\alpha,W) \\
&\quad = A\left\{\mathcal{V}^{(n)}(n+r-1,\alpha,W) - \mathcal{V}^{(n+1)}(n+r-1,\alpha,W)\right\} \\
&\quad = A^r\left\{W - \overline{W}_\alpha(n)\right\}(1-A)(\alpha+\lambda-\lambda A)^{-1}, \quad n,r \in \mathbb{N}.
\end{aligned} \tag{A.13}$$

We now write the right-hand side of (A.8) as

$$\begin{aligned}
&\alpha C(M) + \lambda\overline{\mathcal{V}}(M+1,\alpha,W) - (\alpha+\lambda)\widetilde{C}(M,\alpha) \\
&\quad -(\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^\infty \frac{(\lambda t)^r}{r!}\,\mathrm{e}^{-(\alpha+\lambda)t}\overline{\mathcal{V}}(M+r-1,\alpha,W)\,\mathrm{d}G \\
&= \alpha C(M) + \lambda\mathcal{V}^{(M)}(M+1,\alpha,W) - (\alpha+\lambda)\widetilde{C}(M,\alpha) \\
&\quad -(\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^\infty \frac{(\lambda t)^r}{r!}\,\mathrm{e}^{-(\alpha+\lambda)t}\mathcal{V}^{(M)}(M+r-1,\alpha,W)\,\mathrm{d}G \\
&\quad +\lambda\left\{\sum_{n=m}^{M-1}\mathcal{V}^{(n)}(M+1,\alpha,W) - \mathcal{V}^{(n+1)}(M+1,\alpha,W)\right\} \\
&\quad -(\alpha+\lambda)\left[\sum_{n=m}^{M-1}\sum_{r=0}^{\infty}\int_0^\infty \frac{(\lambda t)^r}{r!}\,\mathrm{e}^{-(\alpha+\lambda)t}\left\{\mathcal{V}^{(n)}(M+r-1,\alpha,W)\right.\right. \\
&\quad \left.\left. -\mathcal{V}^{(n+1)}(M+r-1,\alpha,W)\right\}\mathrm{d}G\right]
\end{aligned} \tag{A.14}$$

$$\begin{aligned}
&= (1-A)\overline{W}_\alpha(M) + AW + \lambda\sum_{n=m}^{M-1}A^{M+1-n}\left\{W - \overline{W}_\alpha(n)\right\}(1-A)(\alpha+\lambda-\lambda A)^{-1} \\
&\quad -(\alpha+\lambda)(1-A)(\alpha+\lambda-\lambda A)^{-1} \\
&\quad \times\left\{\sum_{n=m}^{M-1}A^{M-1-n}\left\{W - \overline{W}_\alpha(n)\right\}\sum_{r=0}^{\infty}\frac{(\lambda t)^r}{r!}\,\mathrm{e}^{-(\alpha+\lambda)t}A^r\,\mathrm{d}G\right\}
\end{aligned} \tag{A.15}$$

$$= (1-A)\overline{W}_\alpha(M) + AW + \lambda A\sum_{n=m}^{M-1}\left[A^{M-n}\left\{W - \overline{W}_\alpha(n)\right\}(1-A)(\alpha+\lambda-\lambda A)^{-1}\right]$$

$$
\begin{aligned}
&- (\alpha+\lambda)\sum_{n=m}^{M-1}\left[A^{M-n}\left\{W-\overline{W}_\alpha(n)\right\}(1-A)(\alpha+\lambda-\lambda A)^{-1}\right] && \text{(A.16)}\\
&= (1-A)\left[\overline{W}_\alpha(M)+\sum_{n=m}^{M-1}A^{M-n}\overline{W}_\alpha(n)\right]+AW-(1-A)\sum_{n=m}^{M-1}A^{M-n}W\\
&\geqslant W, && \text{(A.17)}
\end{aligned}
$$

as required. Note that (A.15) makes use of (A.13) and a version of (A.7) with M replacing m while (A.16) follows from (25). Inequality (A.17) is a consequence of the fact that

$$
\overline{W}_\alpha(n)\geqslant W,\quad n\geqslant m.
$$

We have now established (A.1). We now proceed to show that (A.2) holds in cases 3 and 4.

(3) $n=m-1\geqslant 1$. From (A.2) we are required to show that

$$
\begin{aligned}
W &\geqslant \alpha C(m-1)+\lambda\overline{\mathcal{V}}(m,\alpha,W)-(\alpha+\lambda)\widetilde{C}(m-1,\alpha)\\
&\quad-(\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^\infty\frac{(\lambda t)^r}{r!}\,\mathrm{e}^{-(\alpha+\lambda)t}\overline{\mathcal{V}}(m+r-2,\alpha,W)\,\mathrm{d}G && \text{(A.18)}\\
&=\alpha C(m-1)+\lambda\mathcal{V}^{(m-1)}(m,\alpha,W)-(\alpha+\lambda)\widetilde{C}(m-1,\alpha)\\
&\quad-(\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^\infty\frac{(\lambda t)^r}{r!}\,\mathrm{e}^{-(\alpha+\lambda)t}\mathcal{V}^{(m-1)}(m+r-2,\alpha,W)\,\mathrm{d}G\\
&\quad+\lambda\left\{\mathcal{V}^{(m)}(m,\alpha,W)-\mathcal{V}^{(m-1)}(m,\alpha,W)\right\}-(\alpha+\lambda)\\
&\quad\times\sum_{r=0}^{\infty}\int_0^\infty\left[\frac{(\lambda t)^r}{r!}\,\mathrm{e}^{-(\alpha+\lambda)t}\left\{\mathcal{V}^{(m)}(m+r-2,\alpha,W)\right.\right.\\
&\quad\left.\left.-\mathcal{V}^{(m-1)}(m+r-2,\alpha,W)\right\}\right]\mathrm{d}G. && \text{(A.19)}
\end{aligned}
$$

For the last term in (A.19) we need to consider expressions of the form

$$
\mathcal{V}^{(n)}(n+r,\alpha,W)-\mathcal{V}^{(n+1)}(n+r,\alpha,W),\quad n\in\mathbb{N},\ r\in\mathbb{Z}^-. \qquad \text{(A.20)}
$$

But both policies $u(n)$ and $u(n+1)$ will take the passive action in state $n+r$ when $r<0$. From this it easily follows that

$$
\begin{aligned}
&\mathcal{V}^{(n)}(n+r,\alpha,W)-\mathcal{V}^{(n+1)}(n+r,\alpha,W)\\
&=\left(\frac{\lambda}{\alpha+\lambda}\right)^{-r}\left\{\mathcal{V}^{(n)}(n,\alpha,W)-\mathcal{V}^{(n+1)}(n,\alpha,W)\right\}\\
&=\left(\frac{\lambda}{\alpha+\lambda}\right)^{-r}\left\{W-\overline{W}_\alpha(n)\right\}(1-A)(\alpha+\lambda-\lambda A)^{-1},\quad n\in\mathbb{N},\ r\in\mathbb{Z}^-, \qquad \text{(A.21)}
\end{aligned}
$$

by (A.13). If we now use an appropriate version of the calculation to (A.7) along with (A.13) and (A.21) within (A.19) we obtain that

$$\begin{aligned}
&\alpha C(m-1) + \lambda \overline{\mathcal{V}}(m, \alpha, W) - (\alpha+\lambda)\widetilde{C}(m-1, \alpha) \\
&\quad - (\alpha+\lambda) \sum_{r=0}^{\infty} \int_0^{\infty} \frac{(\lambda t)^r}{r!} \mathrm{e}^{-(\alpha+\lambda)t} \overline{\mathcal{V}}(m+r-2, \alpha, W)\, \mathrm{d}G \\
&= (1-A)\overline{W}_\alpha(m-1) + AW \\
&\quad + \lambda A\big\{\overline{W}_\alpha(m-1) - W\big\}(1-A)(\alpha+\lambda-\lambda A)^{-1} \\
&\quad - (\alpha+\lambda) \sum_{r=0}^{\infty} \int_0^{\infty} \left[\frac{(\lambda t)^r}{r!} \mathrm{e}^{-(\alpha+\lambda)t} A^{r-1}\big\{\overline{W}_\alpha(m-1) - W\big\}\right. \\
&\quad \left. \times (1-A)(\alpha+\lambda-\lambda A)^{-1}\right] \mathrm{d}G \\
&\quad + (\alpha+\lambda) \int_0^{\infty} \left(\mathrm{e}^{-(\alpha+\lambda)t}\left[A^{-1} - \frac{\lambda}{\lambda+\alpha}\right]\big\{\overline{W}_\alpha(m-1) - W\big\}\right. \\
&\quad \left. \times (1-A)(\alpha+\lambda-\lambda A)^{-1}\right) \mathrm{d}G \qquad \text{(A.22)} \\
&= W + (1-A)A^{-1}\big\{\overline{W}_\alpha(m-1) - W\big\} \int_0^{\infty} \mathrm{e}^{-(\alpha+\lambda)t}\, \mathrm{d}G \qquad \text{(A.23)} \\
&\leqslant W, \qquad \text{(A.24)}
\end{aligned}$$

since $\overline{W}_\alpha(m-1) \leqslant W$. Note that (A.23) follows from (A.22) by way of identity (25). We have now established inequality (A.2) for the case $n = m-1$.

(4) $1 \leqslant n \leqslant m-2$. Fix state $1 \leqslant M \leqslant m-2$. From (A.2) we require that

$$\begin{aligned}
W &\geqslant \alpha C(M) + \lambda \overline{\mathcal{V}}(M+1, \alpha, W) - (\alpha+\lambda)\widetilde{C}(M, \alpha) \\
&\quad - (\alpha+\lambda) \sum_{r=0}^{\infty} \int_0^{\infty} \frac{(\lambda t)^r}{r!} \mathrm{e}^{-(\alpha+\lambda)t} \overline{\mathcal{V}}(M+r-1, \alpha, W)\, \mathrm{d}G \\
&= \alpha C(M) + \lambda \mathcal{V}^{(M)}(M+1, \alpha, W) - (\alpha+\lambda)\widetilde{C}(M, \alpha) \\
&\quad - (\alpha+\lambda) \sum_{r=0}^{\infty} \int_0^{\infty} \frac{(\lambda t)^r}{r!} \mathrm{e}^{-(\alpha+\lambda)t} \mathcal{V}^{(M)}(M+r-1, \alpha, W)\, \mathrm{d}G \\
&\quad + \lambda \left\{\sum_{n=M}^{m-1} \mathcal{V}^{(n+1)}(M+1, \alpha, W) - \mathcal{V}^{(n)}(M+1, \alpha, W)\right\} \\
&\quad - (\alpha+\lambda) \sum_{n=M}^{n-1} \sum_{r=0}^{\infty} \left[\frac{(\lambda t)^r}{r!} \mathrm{e}^{-(\alpha+\lambda)t} \big\{\mathcal{V}^{(n+1)}(M+r-1, \alpha, W)\right. \\
&\quad \left. - \mathcal{V}^{(n)}(M+r-1, \alpha, W)\big\}\right] \mathrm{d}G. \qquad \text{(A.25)}
\end{aligned}$$

We now use (A.13) and (A.21) to analyse terms on the right-hand side of (A.25). In order to do so, we need to utilise sequences of the form

$$S_r \equiv \left\{ \left(\frac{\lambda}{\alpha+\lambda}\right)^r, \left(\frac{\lambda}{\alpha+\lambda}\right)^{r-1}, \dots, \left(\frac{\lambda}{\alpha+\lambda}\right), 1, A, A^2, \dots, \right\}, \quad r \in \mathbb{N},$$

and

$$S_{-r} \equiv \left\{ A^r, A^{r+1}, \dots, \right\}, \quad r \in \mathbb{Z}^+.$$

We shall use $S_{n,r}$ to denote the nth term in the sequence S_r, $n \in \mathbb{Z}^+$, $r \in \mathbb{Z}$. The fifth term on the right-hand side of (A.25) may be expressed as

$$\lambda \left\{ \sum_{n=M}^{m-1} \mathcal{V}^{(n+1)}(M+1, \alpha, W) - \mathcal{V}^{(n)}(M+1, \alpha, W) \right\}$$
$$= \lambda \left[\sum_{n=M}^{m-1} \left\{ \overline{W}_\alpha(n) - W \right\} S_{m-n, m-M-2} \right] (1-A)(\alpha+\lambda-\lambda A)^{-1} \qquad \text{(A.26)}$$

and the sixth term on the right-hand side of (A.25) as

$$(\alpha+\lambda) \sum_{n=M}^{n-1} \sum_{r=0}^{\infty} \int_0^\infty \left[\frac{(\lambda t)^r}{r!} \mathrm{e}^{-(\alpha+\lambda)t} \left\{ \mathcal{V}^{(n+1)}(M+r-1, \alpha, W) \right.\right.$$
$$\left.\left. - \mathcal{V}^{(n)}(M+r-1, \alpha, W) \right\} \right] \mathrm{d}G$$
$$= (\alpha+\lambda) \sum_{n=M}^{m-1} \left\{ \overline{W}_\alpha(n) - W \right\} \sum_{r=0}^{\infty} \int_0^\infty S_{r+1, n-M+1} \frac{(\lambda t)^r}{r!} \mathrm{e}^{-(\alpha+\lambda)t} (1-A)$$
$$\times (\alpha+\lambda-\lambda A)^{-1} \, \mathrm{d}G. \qquad \text{(A.27)}$$

In order to develop an analysis based on (A.26) and (A.27), we observe that, for all choices of $s \geqslant 0$,

$$(\alpha+\lambda) \sum_{r=0}^{\infty} \int_0^\infty \left(\sum_{n=1}^{s+2} S_{n, s+2-r} \right) \frac{(\lambda t)^r}{r!} \mathrm{e}^{-(\alpha+\lambda)t} \, \mathrm{d}G$$
$$\leqslant \lambda \sum_{r=0}^{\infty} \left(\sum_{n=1}^{s+2} S_{n, s+1} \right) \frac{((\alpha+\lambda)t)^r}{r!} \mathrm{e}^{-(\alpha+\lambda)t} \qquad \text{(A.28)}$$
$$\leqslant \lambda \left\{ \left(\frac{\alpha+\lambda}{\lambda} \right) + \sum_{n=1}^{s+1} S_{n,s} \right\}. \qquad \text{(A.29)}$$

Recall also that the first four terms on the right-hand side of (A.25) when aggregated, are equal to

$$(1-A)\overline{W}_\alpha(M) + AW. \qquad \text{(A.30)}$$

Combining (A.25)–(A.27) with (A.30) we can express the right-hand side of (A.25) as

$$W + \sum_{n=M}^{m-1} \{\overline{W}_\alpha(n) - W\} a_n, \tag{A.31}$$

where

$$\begin{aligned} a_M &= 1 - A + \lambda S_{m-M,m-M-2}(1-A)(\alpha+\lambda+\lambda A)^{-1} \\ &\quad - (\alpha+\lambda)\left\{\sum_{r=0}^{\infty}\int_0^{\infty} S_{r+1,1}\frac{(\lambda t)^r}{r!}\,\mathrm{e}^{-(\alpha+\lambda)t}\,\mathrm{d}G\right\}(1-A)(\alpha+\lambda+\lambda A)^{-1} \\ &= (\alpha+\lambda)\left\{1 - \sum_{r=0}^{\infty}\int_0^{\infty} S_{r+1,1}\frac{(\lambda t)^r}{r!}\,\mathrm{e}^{-(\alpha+\lambda)t}\,\mathrm{d}G\right\}(1-A)(\alpha+\lambda-\lambda A)^{-1}, \end{aligned} \tag{A.32}$$

and

$$\begin{aligned} a_n &= \left\{\lambda S_{m-n,m-M-2} - (\alpha+\lambda)\sum_{r=0}^{\infty}\int_0^{\infty} S_{r+1,n-M+1}\frac{(\lambda t)^r}{r!}\,\mathrm{e}^{-(\alpha+\lambda)t}\,\mathrm{d}G\right\} \\ &\quad \times(1-A)(\alpha+\lambda-\lambda A)^{-1}, \quad M+1 \leqslant n \leqslant m-1. \end{aligned} \tag{A.33}$$

But from (A.29), (A.32) and (A.33) we deduce that, for all choices of s, $m-1 \geqslant s \geqslant M$,

$$\sum_{n=M}^{s} a_n \geqslant 0. \tag{A.34}$$

Combining (A.31) and (A.34) we see that the right-hand side of (A.25) is given by

$$W + \{\overline{W}_\alpha(m-1) - W\}\left(\sum_{n=M}^{m-1} a_n\right) + \sum_{n=M}^{m-2}\{\overline{W}_\alpha(n) - \overline{W}_\alpha(n+1)\}\left(\sum_{r=M}^{n} a_r\right) \leqslant W, \tag{A.35}$$

as required. The inequality in (A.35) follows from (A.34) and the assumptions concerning W and the values of $\overline{W}_\alpha$. This concludes the proof. □

References

[1] E. Altman and P. Nain, Optimal control of the $M/G/1$ queue with repeated vacations of the server, IEEE Trans. Automat. Control 38 (1993) 1766–1775.

[2] P.S. Ansell, K.D. Glazebrook, I. Mitrani and J. Niño Mora, A semidefinite programming approach to the optimal control of a single server queueing system with imposed second moment constraints, J. Oper. Res. Soc. 50 (1999) 765–773.

[3] P.S. Ansell, K.D. Glazebrook, J. Niño Mora and M. O'Keeffe, Whittle's index policy for a multiclass queueing system with convex holding costs, Math. Methods Oper. Res. (to appear).

[4] D. Bertsimas and J. Niño Mora, Conservation laws, extended polymatroids and multi-armed bandit problems: A polyhedral approach to indexable systems, Math. Oper. Res. 21 (1996) 257–306.

[5] A. Cobham, Priority assignment in waiting line problems, Oper. Res. 2 (1954) 70–76.

[6] D.R. Cox and W.L. Smith, *Queues* (Methuen, London, 1961).

[7] A. Federgruen and K.C. So, Optimality of threshold policies in single-server queueing systems with server vacations, Adv. in Appl. Probab. 23 (1991) 288–405.

[8] J.C. Gittins, *Multi-Armed Bandit Allocation Indices* (Wiley, New York, 1989).

[9] J.M. Harrison, Dynamic scheduling of a multiclass queue: discount optimality, Oper. Res. 23 (1975) 270–282.

[10] G.P. Klimov, Time sharing systems I, Theory Probab. Appl. 19 (1974) 532–551.

[11] J. Niño Mora, Countable partial conservation laws, Whittle's restless bandit index and a dynamic $c\mu$ rule for scheduling a multiclass $M/M/1$ queue with convex holding costs, Technical Report, Universitat Pompeu Fabra (2001).

[12] M.L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (Wiley, New York, 1994).

[13] R. Righter and S.H. Xu, Scheduling jobs on nonindentical IFR processors to minimize general cost functions, Adv. in Appl. Probab. 23 (1991) 909–924.

[14] J.A. van Meighem, Dynamic scheduling with convex delay costs: The generalized $c\mu$-rule, Ann. Appl. Probab. 5 (1995) 809–833.

[15] R.R. Weber, Stochastic scheduling on parallel processors and minimization of concave functions of completion times, in: *Stochastic Differential Systems, Stochastic Control Theory and Applications*, Vol. 10 (Springer, Berlin, 1988) pp. 601–609.

[16] G. Weiss, Branching bandit processes, Probab. Engrg. Inform. Sci. 2 (1988) 269–278.

[17] P. Whittle, Restless bandits: activity allocation in a changing world, J. Appl. Probab. A 25 (1988) 287–298.

[18] P. Whittle, *Optimal Control: Basics and Beyond* (Wiley, Chichester, 1996).