1997

Solution methods for controlled queueing networks

Sabri Tankut Atan
Iowa State University

Solution methods for controlled queueing networks

by

Sabri Tankut Atan

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Industrial Engineering

Major Professor: Douglas McBeth

Iowa State University

Ames, Iowa

1997

Graduate College
Iowa State University

This is to certify that the doctoral dissertation of

Sabri Tankut Atan

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

MDP:	Markov Decision Process
SMDP:	Semi-Markov Decision Process
DMDP:	Discrete-time Markov Decision Process
QN:	Queueing Network
QS:	Queueing System
RWDA:	Routing problem With Dedicated Arrivals
RNDA:	Routing problem with No Dedicated Arrivals
PI:	Policy Iteration
VI:	Value Iteration
LP:	Linear Programming
IOPT:	Individual-OPTimum rule
NQ:	Never-Queue rule
GT:	Greedy Throughput rule
SEP:	SEParable rule
m:	Number of parallel queues
$A(i)$:	Action set for state i
I:	State space
R:	A policy
ρ:	Traffic intensity
L:	Average number of customers in the system

W:	Average waiting time of a customer
N_i:	Capacity of queue i
n_i:	Number of people in queue i including the one in service
μ_i:	Service rate for server i
λ_i:	Dedicated arrival rate to queue i
λ_0:	Generic arrival rate
$v_i(R)$:	Relative value of state i under policy R
$g(R)$:	Average cost under policy R
c_i:	The expected costs until the next decision epoch in the present state i
$p_{ij}(a)$:	The probability of being in state j in the next decision epoch if action a is taken in the present state i
T_i:	The expected time until the next decision epoch for the present state i

ACKNOWLEDGMENTS

There are many people I would like to acknowledge. If you are not on the list then you haven't paid me enough: My mom and sister, my stepfather and the rest of my family, Susanne Pollmann, Doug McBeth, Ram Pandit, John Jackman. John Even, David Fernandez-Baca, Ananda Weerasinghe, Anand, Chizuko and Seiko Shastri, Volker and Jutta Luft, Müfit and Aşula Yorulmaz, Oğuzhan Nuri, İhsan Giray, Cengiz Taş, Erol and Yıldız Aksoy, Rıfat Sönmez, Mike Moon, Bakthavatchalam Muralidharan, Scott Singleton, Peter Brust, Cheng-Kang Chen, Tai-Hung Yang, Cem Şahin, İlhan Tan, Onur Yalçın, Zafer and Çiğdem Bulut, Tom Devany, Alejandro Leon-Escamilla, Marcelo Pinto, Christoph and Audra Hiemcke, Lori Hilpipre, Lynn Franco, Donna Cerka.

Not to mention: Turkish Education Foundation, Parks library, the Engineering Annex, the Rec Center, 223 E 9th Street, my Merkur and AAA, Iowa's huge skies, and the Amazon forest. This last one may need explanation: During an award ceremony in a Cheers episode, Dianne thanked the Amazon forest for supplying most of the oxygen we breathe. This got stuck in my mind. I think it helps you to keep things in perspective.

CHAPTER 1 INTRODUCTION AND LITERATURE REVIEW

1.1 Introduction

The building blocks of a queueing network (QN) consist of one or more input streams of customers (single units or batches), one or more buffers (finite or infinite), and one or more servers. In addition to these physical components a set of policies are defined to specify the rules for queueing and servicing the customers. By adding several of these building blocks -queueing systems (QSs)- together, a QN is obtained.

Since the birth of queueing theory at the beginning of the twentieth century, literature on the subject has grown enormously. In this dissertation, we focus on controlled QNs as opposed to descriptive problems where one is interested in a certain performance measure for the system such as average waiting time of each customer, or expected number of customers in the system per unit time. In control problems, the process happens in continuous time. However, a controller can, for example, change certain parameters of the system, or affect how the customers are routed through the system at discrete time points, called decision epochs, pursuing a goal such as the average waiting time of each customer. The controller tries to find a policy, i.e. a sequence of decisions -e.g. which queue to join, to accept or reject the incoming customers- to achieve its goal.

In this dissertation, we consider a problem which is the building block of many systems, and its solution is important from a practical point of view. We develop an exact methodology, which is better than the current methods, for solving this type of problem. This methodology is not specific to our problem; it can be used to solve many other problems in controlled QN settings. We also compare several heuristic methods, which can be used for very large problems. In this comparison study, we use all the heuristic ideas that we have encountered in the literature, and also suggest another heuristic method. The comparisons are made against the exact methodology we develop since it offers a very efficient way of evaluating a given policy. Moreover, we point out a misconception in the literature regarding our type of problems.

To be more specific, we concentrate on a problem where the rates (service and arrival) and the number of servers are fixed. The servers serve in parallel and they have their own queues with infinite capacity in front of them. There are several arrival streams to the system. Each queue has an arrival stream that automatically joins it. On top of this, there is a general arrival stream. The general customers can be served by any of the servers, and it is the job of the controller to decide which queue each general customer should join to receive service. When making decisions the controller is trying to minimize the average number of customers in the system per unit time. We will give a formal and detailed description of the problem in Chapter 2.

Due to their wide applicability in manufacturing systems and computer networks these type of QNs have recently received a lot of attention. Researchers have difficulty in analyzing such systems because of their analytical complexity. Most of the results indicate what the structure of the optimal policy is, which in many cases boils down to showing that something very intuitive is in fact the optimal policy. These structural results are only limited to very small networks (two servers). On the other hand, even for such small systems, it is not clear how the parameters of the optimal policy should be obtained in an efficient way. Without the parameters, the structural properties alone are not of much help for practical purposes. Moreover, computational problems and memory limitations severely affect the size of problems that can be solved computationally. For the problems that can be solved within the given computer memory limitations, the efficiency in solving for the optimal control parameters is very important which is what we are focusing on in this dissertation. The larger the problem size, the more computational efficiency becomes a handicap. With our methodology that will be described in Chapter 3 we efficiently solve larger problems than previous researchers.

1.2 Classification of Problems

The problems in the area of controlled QNs can be classified with respect to many attributes:

1. Routing vs. scheduling: An important portion of previous work has been dedicated to either routing problems -in which customers are allocated at the time of their arrival- or to scheduling problems -in which customers are maintained in a single queue and allocated to servers when they become idle. It should be noted though that this distinction of terms is not universal. Our problem is a routing problem.

2. The amount of system information available to the controller: Complete, partial or none. If the controller has complete information then it knows the length of all queues, and all of the arrival and service rates at the time of an action. On the other hand, it may have partial information or no information at all. The research focuses on the first and third cases. We assume that all the information is available.

3. The goal: While the individual customers are interested in optimizing their interests, the controller seeks the social optimum. Although for some systems these two goals coincide, this is not true in general. Analytically, it is easier to find the individual optimum, but the more interesting case is finding the social optimum.

   The objective often is to minimize the discounted cost or long-run average cost. We are interested in the long-run average cost. Often, structural properties of the optimal policies are the same for both of the cost problems. Also, the optimal policies for the long-run average cost can be derived using the results from the solution to the former objective, the discounted cost problem. But certain conditions should be satisfied for the long-run average cost to have an optimal stationary policy (a stationary policy is one where at any time point the decision taken in a given state remains the same).

4. Heterogeneous vs. homogeneous servers: Homogeneous servers are those that have equal service rates. This assumption simplifies the analysis, and most of the reported results are related to this case. When servers have different service rates they are called heterogeneous. We deal with heterogeneous servers.

5. Topology: The QN may have different layouts such as parallel or tandem. Our layout is parallel.

6. Dynamic vs. static assignment: In static assignment rules the actions do not depend on the state of the system. Dynamic assignment rules give better performance but their implementation is much more complicated. Due to their superior performance, we are looking for dynamic assignment rules.

The problems that occur within the framework of controlled QNs are very popular due to their applicability in many fields such as manufacturing, telecommunications, and computer networks. This leads to a large number of publications that are published in very different journals. In our review, we concentrate on problems where the common characteristic is parallel layout. We would like to emphasize that the existing research focuses on proving the structure of the optimal policy, and that this can only be done for small networks with two servers. The literature review that follows can be skipped during the first reading.

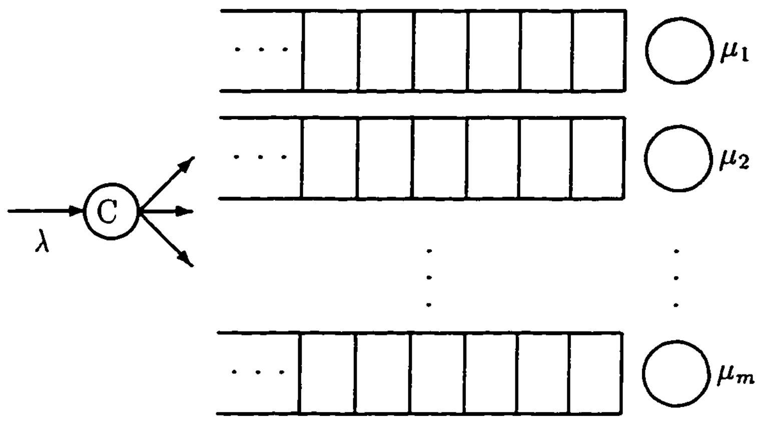

Figure 1.1 A routing problem (RNDA)

1.3 Literature Review

Stidham and Weber [46] give a comprehensive survey of Markov decision models for control of networks of queues. While we are reviewing the literature, we will focus on QNs with parallel queues. As mentioned earlier, this can be done in two distinctive categories, namely routing and scheduling.

1.3.1 Routing

Consider the system in Figure 1.1 which will be denoted as routing with no dedicated arrivals (RNDA). Customers arrive to the system according to a Poisson process with rate λ. At the time of their arrival a controller routes them to a queue, queue k, served by an exponential server with rate μ_k. Also we assume that the servers are numbered in decreasing order of server speed. Once routed, a customer stays in that queue until it finishes service. We also assume that the service discipline is FIFO (first-in-first-out). The exponentiality is widely assumed in the literature. If more general distributions are used this will be indicated.

Winston [58], Weber [54] and Ephremides *et al.* [10] showed that if the system consists of identical $M/M/1$ queues in which the queue lengths are known at any time, then the expected discounted cost is minimized by the shortest queue policy, i.e. by a policy that routes the customers to the shortest queue at their arrivals. Johri [23] extends the domain of optimality of the shortest queue policy to state-dependent service rate case. Hordijk and Koole [17], and Towsley *et al.* [52] consider finite buffer queues with more general arrival processes, and once again optimality of shortest queue policy is proved. Xu *et al.* [62] considered a routing problem where there are two classes of customers to be served by two stations, with parallel servers in each station. The servers are homogeneous. Class-1 customers can be served only by station 1 whereas class-2 customers are free to choose a station. It is shown that to minimize the long-run average cost a class-j customer, whenever possible, should be assigned to an idle server in station j, and a class-2 customer should be assigned to an idle server in station 1 only if (no class-1 customers are waiting, and) the length of queue 2 exceeds a critical number.

As opposed to the above cases where the controller can observe the queue lengths, there are applications where this information is not available. Ephremides *et al.* [10] proved for two queues that the round-robin policy is optimal when the controller has access to only the past routing decisions and the service times are exponentially distributed with the same rate. The round-robin policy sends customers one by one to different servers until all of the servers are used, and continues sending the customers to the servers in the same order. Since the servers are equally fast the order is not important, and by keeping the same order in each cycle the times between arrivals to any server are balanced. Stoyan [48] showed that the round-robin policy gives stochastically smaller waiting times than the Bernoulli policy with equal routing probabilities to each queue. This particular Bernoulli routing policy has been shown to be optimal among all the Bernoulli policies [6]. Recently, Liu and Towsley [30] extended the optimality of round-robin policy to service times with an increasing failure rate distribution.

In the case of heterogeneous servers, the structure of the optimal policy is only known for the two-queue case. Hajek [12] proved that if the system has two heterogeneous $M/M/1$ queues, then a switch-over policy minimizes the discounted (or average) cost with linear holding cost. A switch-over policy can be formally described as: If x_i denotes the queue length at queue i, then there exists an increasing function $F(x_1)$ such that an arriving customer is routed to server 2 if $x_2 \leq F(x_1)$. Xu and Chen [60] recently showed that with the discounted cost criterion the optimal switching curve has a finite asymptotic limit when $c_1 \neq c_2$, where c_i is the unit holding cost per unit time at station i. They also show that there is no finite asymptote in case of long-run average cost. Xu and Zhao [63] allow jockeying, i.e. the controller can change its routing decision and send some customers to the other queue after they have joined one queue, between the two queues and characterize the structure of the dynamic routing and jockeying policies that minimize the expected total cost, for both discounted and long-run average cost criteria. Whitt [57] gives examples in which the shortest expected delay policy is not optimal. There are several examples; with multiple exponential servers, under general service times, and also in the case where only the number of customers in the queues are known. Houck [19] uses a simulation study showing that the shortest expected delay policy performs nearly optimally when there are two stations with parallel, identical servers, where each station has a single queue. Banawan and Zahorjan [2] did a numerical study showing that in RNDA the individually optimal policy is actually the optimal decision in most of the states. The optimal strategies were obtained using policy iteration. Rosberg and Kermani [37] compared an overflow routing heuristic against a lower bound. For high traffic intensities their method moves away from the lower bound. The problem in Houck's paper with heterogeneous servers was solved by Krishnan [24] using a one-step policy improvement algorithm starting with a Bernoulli-splitting policy. This heuristic method performs better than the shortest queue policy. Later, Shenker and Weinrib [42] give cases where the shortest queue policy does not perform very well. They also use simulation to compare several heuristic policies. Hlynka *et al.* [15] talk about a case with two queues where a single smart customer can observe the queues before joining and only after some arrivals or service departures have occurred he enters a queue. They show that this single customer can lower its expected sojourn time by using the information gained before entering the queue rather than joining the shortest queue immediately.

In case of heterogeneous servers, when the queue length information is not available, there also exists considerable amount of research to find the best sequence for routing customers to the queues. Hajek [13] gives the optimal sequence for two queues given that fraction p of the customers should be sent to the first queue. For more than two queues the optimal sequence is not known, but many good sequences have been obtained. The sequences given in Yum [64], and Itai and Rosberg [22] are some of them. Arian and Levy [1] give a sequence based on Hajek's optimal two-queue sequence which outperforms the previous sequences. Combe and Boxma [7] obtained another sequence based on Hajek's sequence but they don't include a numerical study showing how good their sequence is. Hordijk *et al.* [18] give a close-to-optimal sequence. Their numerical studies show that both their sequence and Combe and Boxma's sequence indeed perform very well, and the period of their sequence is smaller. Combe and Boxma discuss general properties and optimization aspects for probabilistic assignment, and for a policy that follows a fixed pattern which may have been generated by any of the above-mentioned sources.

1.3.2 Scheduling

By allowing the servers in an $M/M/n$ system to be heterogeneous we arrive at the system in Figure 1.2. In this system, the controller chooses between utilizing a server or not when a server becomes idle, or when an arrival occurs. In certain situations, it may be advantageous to wait for a faster server to become idle instead of immediately joining an idle slow server.

Larsen and Agrawala [26] conjectured that with two servers an arbitrary customer's mean sojourn time is minimized by threshold policy. A threshold policy is one in which

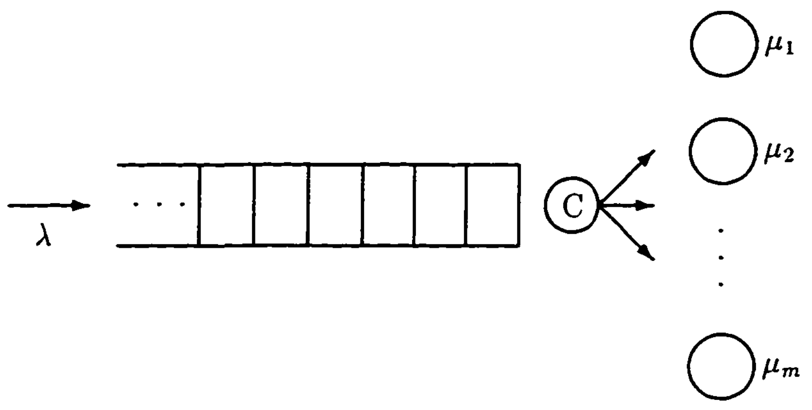

Figure 1.2 A scheduling problem

the slow server is put into service when the queue grows sufficiently large. Lin and Kumar [28] and Walrand [53] proved this conjecture. Lin and Kumar also give a closed-form solution to compute the cost for a given threshold. Kumar and Walrand [25] find the individually optimal policy with any number of servers. Shenker and Weinrib [42] evaluate some heuristic policies using simulation. Sobel [44] proves that routing the customers to the fastest available server is the optimal policy that minimizes throughput when there is no waiting room. This result cannot be extended to non-exponential service rates as shown by a counterexample with two servers in Seth [41]. Xu and Shanthikumar [61] impose an admission control on this system, rather than a scheduling control, and show that when the number in the system reaches a threshold, the incoming customers will be rejected to maximize the expected discounted or long-run average profit. In another related paper, Xu [59] applies both admission and scheduling controls. These two papers do not use dynamic programming techniques, but instead define a dual problem and explore the problem from an individual's point of view whose behaviour is the socially optimal policy for the primal. This approach allowed Xu to derive an approximation for the threshold in the two-server scheduling problem. Rosberg and Makowski [38] show that for the case with multiple servers, and with small arrival rates, the optimal policy is from the class of optimal policies that minimize the expected flow time for a system with fixed population and no new arrivals. With no new arrivals the optimal policy is known to be of threshold type.

1.4 Organization

In Chapter 2 we formally introduce the problem we have worked on. We also give several formulations for the problem. In Chapter 3 we discuss our methodology. We give related computational results in Chapter 4. Chapter 5 consists of several heuristic methods that are useful for very large problems. We summarize, and suggest some further research in Chapter 6. Appendix A is a formal discussion of MDP techniques. Appendix B discusses a resource allocation problem from our research.

CHAPTER 2 ROUTING WITH DEDICATED ARRIVALS

In this chapter, we will first introduce a problem that will be the basis of the later chapters. We will also discuss several formulations of the problem.

We denote our problem by RWDA (routing with dedicated arrivals), see Figure 2.1 for an example. In this routing problem, there are several servers in parallel with possibly different service rates. Each server has its own infinite queue. Service times at each server are distributed $\exp(\mu_i)$. Let m be the number of the servers. The state of the system at any time can be described with a vector $\vec{n} = \{n_1, n_2, \ldots, n_m\}$ where n_i denotes the number of people in front of server i plus the one in service.

There can be as many as $m+1$ different arrival processes to the system. For each server i there is an arrival stream of customers that must be served at server i. These arrivals follow a Poisson process with rate λ_i, $\lambda_i \geq 0$. Note that we allow $\lambda_i = 0$. There is also a general stream of arrivals following a Poisson(λ_0) process with $\lambda_0 > 0$. These arrivals may be served at any of the servers. When a general arrival occurs, it is assigned to a queue by the controller. Jockeying is not allowed, that is once a customer has been assigned to a queue, it cannot join another queue. The controller knows which state the system is in at any time. We also assume that the customers are served in a FIFO manner.

There are many goals the controller may have. In this work, we will focus on the goal of minimizing the average number of customers in the system, L. To that end, the controller finds a policy: A policy is a rule that for each time point and each state of the system dictates what decision to make. The decisions in our problem are to which

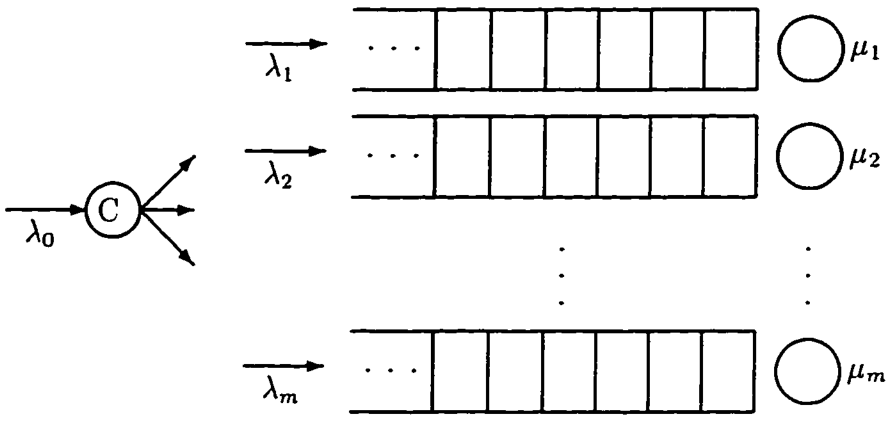

Figure 2.1 Routing with dedicated arrivals (RWDA)

queue a general arrival is sent. The controller uses a stationary policy, i.e. its decisions in a state do not change with time. It is sufficient to only consider stationary policies to find an optimal policy, see for example [50]. For this reason, we mean a stationary policy whenever we use the term policy from now on.

In the literature, typically, the average waiting time of a customer in the system, W, is minimized. L and W are related to each other by the famous Little's law which says $L = \lambda_{eff} W$, where λ_{eff} is the effective average arrival rate to the system. The effective average arrival rate to the system is the average rate at which the customers enter the system. There could be a difference between the arrival rate and the effective arrival rate if for some reason some of the customers do not enter the system. In the above system -if no customers are lost- this average effective arrival rate is equal to the sum of all arrival rates, that is $\lambda_{eff} = \sum_{i=0}^{m} \lambda_i$. In many cases, minimizing L is equivalent to minimizing W by Little's law. In the literature, however, it is incorrectly stated that minimizing W is equivalent to minimizing L. Because the formulation for minimizing W is much more difficult than minimizing L, to minimize W researchers usually minimize L. However, there are systems where this equivalence does not hold. One example of that is our system with finite buffers. We will explain this further later in this chapter.

The problem we have described is a basic component of many models that arise in routing or load balancing problems in different settings such as computer networks, or manufacturing systems. For example, our servers may represent CPU's or printers. While some jobs directly go to these servers, some jobs can be processed by any of them. A controller decides to which server a new-coming general job goes.

Our problem has not been solved in the literature. A few researchers have used dedicated arrivals, but they have looked at different problems: Ni and Hwang [33] consider a class of probabilistic load balancing problems, i.e. the controller randomly assigns the general customers to the servers with some given probabilities for each server. Ni and Hwang determine the optimal set of probabilities from among all possible probabilities to optimize the average waiting time. They also claim to have the closed form solutions for the optimal probabilities. Their results are incorrect, but were later corrected by Bonomi and Kumar [4] when solving a problem that arose in another context. They develop adaptive load balancing methods for a problem with the same layout as in RWDA but where the load parameters (arrival and service rates) are not known. In Appendix B we give a simpler derivation of the above-mentioned formulae. These formulae play a critical role in our methodology, which will be described in Chapter 3.

2.1 Semi-Markov Decision Process Formulation

A common approach to modeling a problem such as the RWDA is to use a semi-Markov decision process (SMDP).

In an SMDP model the system is observed at random points, decision epochs, and the state of the system is determined. After this determination, an action (decision) is taken and costs are incurred as a result of this action. These costs could be either lump sums that incur at discrete time points or costs incurred continuously in time with some rate. After the action has been decided about, the system stays in that state for a random amount of time, and then makes a transition to another state with some probability that only depends on the state now and the action that has been taken. Some other Markovian properties are also satisfied: The time until the next decision epoch does not depend on the past history of the system but only on the state the system is in now and the action that is being taken. The same is true for the costs.

If we only consider the times where the controller actually makes a decision (an arrival of a general customer) the problem by definition is not exactly an SMDP. In between the decision epochs the system may have many state changes via dedicated arrivals or departures. This, on the other hand, affects the cost structure.

However, it is not difficult to change the formulation so that this is a SMDP. We model the time of every state change, an arrival or a departure as a decision epoch. With this formulation, at any decision epoch that does not correspond to a general arrival, the controller has only one action available, that of doing nothing. The controller is only needed when a general arrival occurs in which case the action to be taken is to assign the customer to one of the queues, $a = 1, \dots, m$.

There are three main techniques for solving MDPs: Policy iteration (PI), value iteration (VI) and linear programming (LP). These methods can only be implemented when the state space is finite. Our state space is $\{(n_1, n_2, \dots, n_m) : 0 \leq n_1 < \infty, \dots, 0 \leq n_m < \infty\}$ which is obviously not finite. A reasonable approach to get around this is to solve the related finite buffer problem which has the state space $\{(n_1, n_2, \dots, n_m) : 0 \leq n_1 \leq N_1, \dots, 0 \leq n_m \leq N_m\}$. The finite buffer problem will be referred as RWDA$_{\bar{N}}$.

As mentioned above, one important difference between RWDA and RWDA$_{\bar{N}}$ is that minimizing W is no longer the same as minimizing L. Because of the finite capacity, some customers will be rejected when the system, or some of the queues are full. This affects λ_{eff}. The fullness of the system, on the other hand, is dictated by the policy the controller is using. For example, on one extreme, with a policy such as "send to the slowest server" many customers cannot enter the system. Hence, the effective arrival rate and a policy are directly related. This, of course, means that a policy that minimizes W does not necessarily minimize L. In our work we also examine how the value and the solution of RWDA$_{\vec{N}}$ change as $\vec{N}$ is changed. In the literature, it is often incorrectly stated that minimizing L is equivalent to minimizing W for problems of this type (see for example [47]). As stated earlier, our goal is to minimize L.

Let T_i be the average holding time in state i. The average holding time of a state is the average time until some event changes the state of the system. This can happen two ways, either an arrival of some kind occurs, or some server finishes serving a customer. Therefore, the holding time in a state is an exponential random variable with average holding time

$$T_i = \frac{1}{\lambda_0 + \sum\limits_{\substack{i=1 \\ n_i<N_i}}^{m} \lambda_i + \sum\limits_{\substack{i=1 \\ n_i>0}}^{m} \mu_i}.$$

When the system is full, nothing but departures can occur.

When the system is in state i, it incurs a cost at the rate of the number in the system at that state. On average, each visit to a state will have a cost that can be found by multiplying the total number of people in state i by the expected time you spend in state i, T_i. We denote this average cost in state i by c_i.

The probabilities for state changes are straightforward to find. Let $\vec{n}_{i+} = \{n_1, \ldots, n_i + 1, \ldots, n_m\}$, and $\vec{n}_{i-} = \{n_1, \ldots, n_i - 1, \ldots, n_m\}$. Then,

$$p_{\vec{n},\vec{n}_{i+}}(a) = \begin{cases} \dfrac{\lambda_i}{\lambda_0 + \sum\limits_{\substack{i=1 \\ n_i<N_i}}^{m} \lambda_i + \sum\limits_{\substack{i=1 \\ n_i>0}}^{m} \mu_i} & a \neq i, n_i < N_i \\ \dfrac{\lambda_0 + \lambda_i}{\lambda_0 + \sum\limits_{\substack{i=1 \\ n_i<N_i}}^{m} \lambda_i + \sum\limits_{\substack{i=1 \\ n_i>0}}^{m} \mu_i} & a = i, n_i < N_i \end{cases}$$

$$p_{\vec{n},\vec{n}_{i-}}(a) = \frac{\mu_i}{\lambda_0 + \sum\limits_{\substack{i=1 \\ n_i<N_i}}^{m} \lambda_i + \sum\limits_{\substack{i=1 \\ n_i>0}}^{m} \mu_i} \quad n_i > 0.$$

In the above equations the numerator is the rate at which a particular event happens. It is well known that the minimum of exponential random variables is another exponential random variable. This new exponential random variable's rate is the sum of all rates. Therefore, the denominator gives the rate at which an event (any) happens in a state.

A very important characteristic of this problem is that the possible number of state changes at any state is not that many. There are $\prod_{i=1}^{m}(N_i + 1)$ total states, but from any state there are at most $2 * m$ states which are possible to go to. Either m types of arrivals, or m departures can occur. For some states, the number of choices are even more limited. This feature will be very useful when designing a methodology which we will discuss in Chapter 3.

2.2 Other Formulations

Our methodology, which is a modification of PI, uses the above formulation. When using the other methods, different formulations must be used.

2.2.1 Value iteration formulation

The VI algorithm requires a discrete-time MDP (DMDP) in which the times between the events are fixed. However, in the previous formulation this does not hold true as the decision epochs are separated from each other by transition times which do not have identical rates. In some states, not all the events are possible, e.g. there can be no departure from an empty system. To make the transition rates the same, we allow transitions from a state to itself, and let the transition times have the same rate. Thus, the SMDP model can be converted to a DMDP model by the following data transformation ([40]).

The actions and the states stay the same. Let

$$T = \frac{1}{\lambda_0 + \sum_{i=1}^{m} \lambda_i + \sum_{i=1}^{m} \mu_i}.$$

Then, the new costs $c_i' = c_i/T_i$, and the new probabilities are found by

$$p_{ij}'(a) = \begin{cases} \frac{T}{T_i} p_{ij}(a) & j \neq i \\ \frac{T}{T_i} p_{ij}(a) + \left(1 - \frac{T}{T_i}\right) & j = i. \end{cases}$$

The new formulation has the same class of stationary policies as the original model (see [50]). Also, the average costs per unit time are the same for each stationary policy in both models.

In this formulation, the expected time until the next decision epoch is

$$\frac{1}{\lambda_0 + \sum_{i=1}^{m} \lambda_i + \sum_{i=1}^{m} \mu_i}.$$

For the implementation of the VI algorithm refer to Appendix A.

2.2.2 Linear programming formulation

Here we will give a general LP formulation. In Appendix A we discuss another LP formulation (in fact, both formulations are duals of each other). This one is useful because it gives the steady state probabilities for when the system is in state i and action a, $a = 1, \ldots, m$, is made. The actions are again sending the customers to a queue. Let $u_i(a) = x_i(a)/T_i(a)$ where $x_i(a)$ is the long-run fraction of decision epochs at which the system is in state i and action a is taken. S denotes the set of all states. Note that the state space is larger than the state space we have defined for use in our methodology. It can be described with a vector $\{(n_1, n_2, \ldots, n_m, a) : 0 \leq n_1 \leq N_1, \ldots, 0 \leq n_m \leq N_m, 1 \leq a \leq m\}$. $A(i)$ stands for the set of actions when one is in state i.

Minimize

$$\sum_{i \in S} \sum_{a \in A(i)} c_i(a) u_{ia}$$

subject to

$$\begin{aligned} \sum_{a \in A(j)} u_{ja} - \sum_{i \in S} \sum_{a \in A(i)} p_{ij}(a) u_{ia} &= 0, \qquad j \in I, \\ \sum_{i \in S} \sum_{a \in A(i)} u_{ia} &= 1, \\ u_{ia} &\geq 0 \qquad \forall i, \forall a. \end{aligned}$$

The objective function is to minimize the long-run average cost per unit time. The first set of constraints is similar to the balance equations while the last constraint requires the fractions u_{ia} to add up to 1.

CHAPTER 3 A NEW METHODOLOGY

We have introduced a routing problem with dedicated arrivals in Chapter 2. In this chapter, we give a methodology which is at least as good and in most cases better than the current techniques for solving this type of problem. For a detailed description and comparison of standard techniques please refer to Appendix A. Both standard policy iteration and LP involve simultaneously solving a system of linear equations at each iteration -note that iterations for PI are different than those for LP- which is computationally burdensome. In general, VI is considered the best method to use for large systems because it avoids solving a system of linear equations. We overcome the shortcomings of the standard PI with several changes to the algorithm. Our methodology makes three major changes to the standard PI-method:

1. It uses iterative methods in solving the system of linear equations.

2. It takes advantage of the sparsity in our problem. It also allocates the state space cleverly.

3. It finds an easy-to-compute initial solution.

We use our methodology to solve the RWDA problem, but this technique can be used to solve many other controlled QN problems and SMDPs in general.

3.1 The Standard PI-algorithm

In order to discuss our methodology we must first describe PI in more detail: PI begins with an initial policy, which can be any policy. This policy is evaluated, and then based on the evaluation an improved policy is developed. These evaluation and improvement steps are then iterated until an optimal policy is reached. It is well known [50] that this procedure will reach an optimum in a finite number of iterations. The evaluation step consists of calculating what are known as relative values, $v_i(R)$, for a given policy R. The relative values give a relative measure of the effect of starting at state i on the total expected costs using the policy R. The difference, $v_j(R) - v_i(R)$, has an economic interpretation: It represents the difference in total expected costs over an infinitely long period of time by starting in state i rather than in j when using the policy R.

Let $g(R)$ be the average cost (in our problem L) for a given policy R. I denotes the set of all possible states, and $A(i)$ stands for the action set available in state i. $R(i)$ denotes the action at state i with policy R. Now, the standard PI algorithm can be given using the notation from Chapter 2 as follows:

1. (initialization) Choose a stationary policy R.

2. (value-determination step) To compute the average cost $g(R)$, and the relative values $v_i(R)$, $i \in I$, for the current rule R, solve the following system of linear equations for its unique minimum:

$$v_i(R) = c_i - g(R)T_i + \sum_{j \in I} p_{ij}(R(i))v_j(R), \qquad i \in I, \tag{3.1}$$

$$v_r(R) = 0, \tag{3.2}$$

where r is arbitrarily chosen.

3. (policy-improvement step) For each state i determine an action a_i that gives the minimum in

$$\min_{a \in A(i)} \left\{ c_i - g(R)T_i + \sum_{j \in I} p_{ij}(a) v_j(R) \right\}.$$

The improved policy is the one which applies the actions that yield the minimum in the above equation for each state.

4. (convergence test) If the new policy is the same as the old one, stop. Otherwise replace the old policy with the new one, and go to the value-determination step.

The bottleneck of this algorithm is the value-determination step where for a given policy a system of linear equations has to be solved for the relative values and the average cost. Assuming that the queue capacities are all N, the number of equations and unknowns is equal to $(N+1)^m + 1$. There are two problems that make this step computationally burdensome: Traditional techniques like Gaussian elimination to solve for systems of linear equations are not time-efficient. Moreover, with our problem the storage requirements grow very quickly. Thus, the memory limitations severely affect the size of the problems that can be solved using traditional methods. Based on these facts, many researchers use VI since it does not involve solving a system of linear equations. Our methodology is a new approach to perform PI which drastically reduces the amount of work done in the value-determination step.

3.2 Calculation of Relative Values

In this section we will discuss the value-iteration step in more detail, and show how the computation of the relative values can be improved.

In the value-determination step, the linear equation for state i can be rearranged as

$$-v_i(R) - g(R)T_i + \sum_{j \in I} p_{ij}(R(i)) v_j(R) = -c_i, \tag{3.3}$$

$$v_r(R) = 0. \tag{3.4}$$

The total size of the system is

$$\left[\prod_{i=1}^{m}(N_i + 1) + 1\right]^2$$

where $\prod_{i=1}^{m}(N_i + 1)$ is the total number of relative values (one for each state). We also need to compute the average number in the system. As mentioned earlier, one of the relative values is arbitrarily set to zero.

Letting x be the unknowns of the system, that is the relative values -$\vec{v}(R)$- and the average cost $g(R)$ ($\equiv L$ of a given policy R), we can write the above system of equations as $Ax = b$ in matrix-vector form where the entries of A are the corresponding coefficients of $\vec{v}(R)$ and $g(R)$. Using this notation, PI results in a sequence of matrices $A_0, A_1, \ldots, A_f$ and a sequence vectors $x_0, x_1, \ldots, x_f$ satisfying

$$A_k x_k = b. \tag{3.5}$$

The main work in PI is calculating the sequence $x_0, x_1, \ldots, x_f$. The policy R_{k+1} is determined from the policy R_k and the vector x_k. A_{k+1} is then determined from R_{k+1}. This suggests that the vectors $x_0, x_1, \ldots, x_f$ are closely related to each other, which happens to be the case. Our approach is to take advantage of this information.

If one uses Gaussian elimination, the above-mentioned observation does not bring any gain because in Gaussian elimination every system has to be solved independently of each other. We use the so called iterative methods to take advantage of this observation.

Next, we will discuss briefly how to solve a system of linear equations, and how we take advantage of the way iterative methods work.

3.2.1 Solving a system of linear equations

Perhaps the best known method for solving linear systems is Gaussian elimination (with partial pivoting). This is a direct method which is very robust, but very time-inefficient in solving large systems. In fact, to solve n equations simultaneously, one needs computation time in the order of n^3. It also requires a lot of memory, and even for small systems in our problem it will perform poorly. This was also an obstacle in the work of other researchers.

Another class of methods is iterative methods. These methods repeatedly improve an approximate solution until it is accurate enough. Suppose you want to solve the system $Ax = b$ which has the unique solution x^*. Iterative methods involve finding a sequence $z_0, z_1, \ldots, z_f$ with $z_n \to x^*$. The user only needs to supply a routine that will compute a vector y given the approximate solution z_l by using $y = Az_l$. If y is close enough to b then the algorithm is terminated.

Thus, the way the iterative methods work allows us to use the observation we made about the dependence of relative values in consecutive iterations of PI. Once, an iteration k of PI has been completed we have a solution, x_k, for the system of linear equations, $A_k x_k = b$. In iteration $k+1$ of PI, we want to find x_{k+1} which is close to x_k. Therefore, we can apply the iterative method in $(k+1)$th value-determination step starting with x_k.

There are different types of iterative methods. Stationary methods are older but they are not as effective as non-stationary methods. An iterative method can be expressed as

$$z_{l+1} = B_l z_l + h_l,$$

where B_l and h_l are determined depending on what method one is using. In the stationary methods there are no changes to B_l and h_l from one step to another. Jacobi, and Gauss-Seidel methods are probably the best known stationary methods. The nice thing about the stationary methods is that they are easy to implement. Unfortunately, they did not converge for many instances of our problem. The non-stationary iterative methods, on the other hand, use the information such as residuals that is obtained at each iteration to compute constants for the next step. We are not going to overview all these methods in detail. For more information we refer the reader to Barrett *et al.* [3], and Hackbusch [11]. Only recently, Stewart published a book [45] that discusses the use of these iterative methods in the numerical solution of Markov chains. It is also important to incorporate the iterative methods into the PI algorithm.

3.3 Sparsity and State Space Allocation

The previous discussions motivate the use of iterative methods in the PI algorithm. The limitation of computer memory poses another obstacle in solving a system of linear equations. The number of states grow very quickly (hence the transition matrices for our linear systems) with the number of queues and queue capacities. Table 3.1 illustrates how the number of states grows for our problem.

Table 3.1 The number of states for the routing problem

# queues (30 each)	# states	Queue capacity	# states (3 queues)
2	961	30	29,791
5	28,629,151	60	226,981
10	$\approx 8 * 10^{14}$	100	1,030,301

Now consider a two-queue case. From any state there are at most four transitions possible. An arrival can be assigned to either of the queues, or a departure can occur from any of the queues. When the number of queues increases, it is clear that the percentage of the actual transitions to the whole state-transition matrix becomes smaller. In PI, the equations (Howard equations) that need to be solved are of the form

$$-v_i(R) - g(R)T_i + \sum_{j\in I} p_{ij}(R(i))v_j(R) = -c_i, \qquad i \in I.$$

One of the relative values is chosen arbitrarily and set to zero. For each row in the A matrix, there are at most $2*m+1$ nonzero entries out of $\prod_{i=1}^{m}(N_i+1)+1$ total entries. Therefore, out of $(\prod_{i=1}^{m}(N_i+1)+1)^2$ entries, at most $(2*m+1)(\prod_{i=1}^{m}(N_i+1)+1)$ are nonzero.

With two queues, and queue capacities of 30, in every row there are at most 5 nonzero entries out of 962 entries. That corresponds to a density of at most 0.5%. If queue capacities are increased to 100, the density drops to at most 0.05%.

In general, if a matrix has a density which is less than 2 or 3% sparse it is considered good. Sparse techniques tremendously reduce the amount of storage and the amount of computational work. This seems to be ignored in the literature. For a general SMDP. PI is not practical. However, for controlled queues, by using sparse techniques it is.

Some researchers set the queue capacities to the same number (see for example [24]. When the service rates are different, the queues are not expected to be equally full with the optimal policy. Hence, some states that are allowed using equal queue capacities are visited with a very small probability. A more reasonable approach is to give the faster servers more waiting room because they are expected to be utilized more often than the slower ones. There is no rigorous way of finding the buffer capacities. However. allocating them proportional to the service rates helped in improving the efficiency of our algorithm quite significantly.

3.4 Initial Solution

In general, PI picks an initial policy and calculates the relative values for it. Therefore, the initial values, x_0, are calculated from scratch as opposed to $x_1, \ldots, x_f$. We get around this by picking an initial policy in which we already know what the solution vector is. Our initial solution vector comes from a Bernoulli-splitting policy which we will discuss in this section. The relative values and L for Bernoulli-splitting can be obtained very easily.

The Bernoulli-splitting policy splits the incoming arrival stream among the queues with probabilities, $p_1, \ldots, p_m$, based on the arrival and service rates. After that, the queues behave as independent M/M/1 queues because addition (or splitting) a Poisson arrival creates yet another Poisson arrival. The optimal splitting probabilities are the subject of Appendix B. Figure 3.1 will make the idea of Bernoulli-splitting clear. The general arrival stream is split into m separate Poisson arrival streams with rates $p_1\lambda_0, \ldots, p_m\lambda_m$. The overall arrival to queue i is another Poisson process with rate $\lambda_i + p_i\lambda_0$.

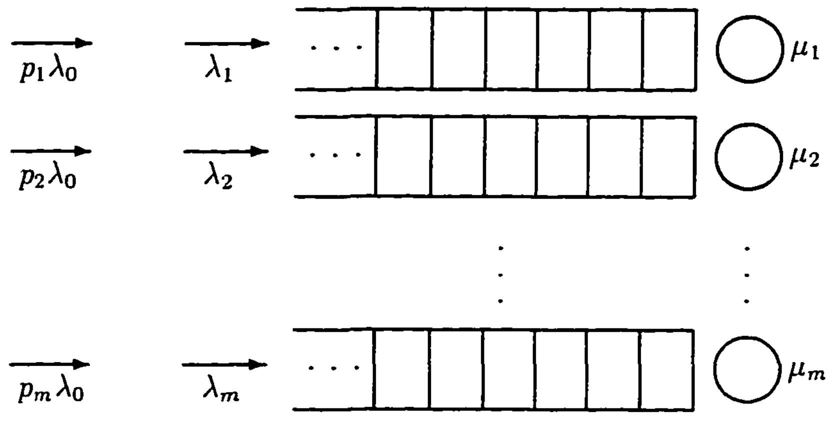

Figure 3.1 Bernoulli-splitting

An M/M/1 queueing process can be looked at as a very straightforward SMDP with two possible decisions at each decision epoch, namely join the queue (if arrival) or do nothing (if departure). Once we have the relative values for the single queues, the relative values for the Bernoulli-splitting process can be found by adding them up using the appropriate probabilities.

Lemma 3.1 *For an M/M/1, the relative values can be expressed as*

$$v_n = \frac{n(n+1)}{2}W,$$

with $v_0 = 0$ and $W = 1/(\mu - \lambda)$ where λ and μ are the corresponding arrival and service rates for the M/M/1 queue.

Proof: With a single queue there are two events possible, an arrival or a service completion. An arrival takes the state n to state $n+1$, while a departure changes n to

$n-1$. The Howard equations become

$$v_n = \frac{n}{\lambda+\mu} - \frac{L}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu} v_{n+1} + \frac{\mu}{\lambda+\mu} v_{n-1}.$$

L for an M/M/1 queue is found by $\lambda/(\mu-\lambda)$. The lemma follows by substitution. ∎

L for Bernoulli-splitting can be expressed as

$$\sum_{i=1}^{m} \frac{p_i\lambda_0 + \lambda_i}{\mu_i - p_i\lambda_0 - \lambda_i},$$

which is the sum of M/M/1 formulae for L.

Now, we have a policy (Bernoulli-splitting) and an easy way to evaluate that policy. By applying a policy-improvement step -which does not pose an important computational burden- we can obtain a better policy. In the policy-improvement step the basic idea is to minimize for each state $\vec{n}$ the difference in total expected costs over an infinite time horizon. This is accomplished by taking a different first action, a, than what the Bernoulli-splitting policy, R_B, dictates. After the first action Bernoulli-splitting is used. Denote this difference $\delta(\vec{n}, a, R_B)$, and the probability of sending a customer to queue j with the Bernoulli-splitting policy p_j. Then for each state $\vec{n}$ and action $a = k$

$$\begin{aligned}\delta(\vec{n}, a, R_B) &= \sum_{\substack{j=1\\ j\neq k}}^{m} p_j[D_k(n_k) - D_j(n_j)] + p_k * 0\\ &= -\sum_{j=1}^{m} p_j D_j(n_j) + D_k(n_k),\end{aligned}$$

where $D_j(n_j)$ is the difference in total expected costs over an infinite time period by starting with n_j+1 customers in queue j rather than with n_j customers. The summation term in the final equation does not depend on the action a. Hence, the policy-improvement step of PI is only

$$\min_{k=1,\ldots,m} D_k(n_k).$$

The difference $D_k(n_k)$ for each queue k is the difference between the relative values, v_{n_k+1} and v_{n_k}. As can be seen by the above expression for $\delta(\vec{n}, a, R_0)$, we don't need to find the relative values for the Bernoulli-splitting policy to perform the policy-improvement step. Looking at the impact of a new arrival on individual queues separately suffices.

3.5 The Algorithm

At this point we will give the algorithm we propose, and later we will discuss some other important issues related to the algorithm.

Let x_{kl} be the approximate solution vector to Equations 3.5 at the kth iteration of PI, and lth iteration of the iterative method. Furthermore, let $x_{kl} = \{\vec{v}_{kl}(R_k), g_{kl}(R_k)\}$, where $\vec{v}_{kl}(R_k)$ is the relative value vector in iteration k of PI, and iteration l of the iterative method for a given policy R_k. $g_{kl}(R_k)$ is the respective average cost (average number in the system). We will also let $v_{kl}(i, R_k)$ denote the entry of the relative value vector for state i. x_{kf} stands for the final solution vector in the kth iteration of PI.

1. (initialization) Choose a stationary policy R_0 using a one-step policy-improvement over Bernoulli-splitting policy. Set $\vec{v}_{00}(R_0) = \vec{v}(\text{Bernoulli})$, $g_{00}(R_0) = L(\text{Bernoulli})$, and k = 0.

2. (value-determination step) To compute the average cost $g_{kf}(R_k)$, and the relative values $\vec{v}_{kf}(R_k)$ for the current rule R_k, solve Equations 3.5 using a non-stationary iterative method.

3. (policy-improvement step) For each state i determine an action a_i that gives the minimum in

$$\min_{a \in A(i)} \left\{ c_i - g_{kf}(R_k)T_i + \sum_{j \in I} p_{ij}(a) v_{kf}(j, R_k) \right\}.$$

   The improved policy, R_{k+1}, is the one which applies the actions that yield the minimum in the above equation for each state.

4. (convergence test) If R_{k+1} is the same as R_k, stop. Otherwise, find A_{k+1} by only changing A_k for the rows where R_{k+1} is different than R_k. Let $R_k = R_{k+1}$, k = k+1, and go to the second step.

3.5.1 Other computational issues

Now we will discuss some aspects of the algorithm we also deem important:

- Since the policies in PI's consecutive iterations are not that much different, it is possible to increase the speed of the algorithm by not writing the coefficient matrix from scratch during each value-determination step. Instead, the matrix is updated during the convergence test step when the old policy is compared to the new one to check convergence.

- A closer look at the policy-improvement step will also bring some improvement. Since the costs c_i and the average holding times T_i are not dependent on the action taken the equation can be written differently,

$$\min_{a\in A(i)} \left\{ c_i - g_{kf}(R_k)T_i + \sum_{j\in I} \rho_{ij}(\vec{n}, R_k)T_i v_{kf}(j, R_k) \right\},$$

where $\rho_{ij}(\vec{n}, R_k)$ is the rate at which a transition from state i to j occurs under policy R_k. Then, the comparison reduces to

$$\min_{a\in A(i)} \left\{ \sum_{j\in I} \rho_{ij}(\vec{n}, R_k) v_{kf}(j, R_k) \right\}.$$

- We mentioned to use a non-stationary iterative method to solve the linear system. That is because stationary methods such as Jacobi, Gauss-Seidel may not converge for our problems. It is well known that if the system is strictly diagonally dominant then the Jacobi and Gauss-Seidel methods converge for any starting value. A system of equations in which the coefficients satisfy

$$\sum_{j=1, j\neq i} m|a_{ij}| < |a_{ii}|, \quad i = 1, 2, \ldots, m$$

is said to be strictly diagonally dominant. Our equations,

$$-v_{kl}(i, R_k) - g_{kl}(R_k)T_i + \sum_{j\in I} p_{ij}(R_k(i))v_{kl}(j, R_k) = -c_i,$$

never satisfy this condition. This can be seen easily by recognizing that the absolute value of the diagonal element is 1, and since the sum of probabilities is equal to 1 as well, this condition is never satisfied. This does not mean that the stationary methods will not converge, but our numerical experience has shown that for our type of problems, stationary methods do not always converge.

3.6 Evaluation of a Policy

Often, one is interested in how a particular policy will perform against the others. VI and LP cannot be used to evaluate a policy. Both of these methods yield an optimal policy only when they are finished. Our methodology and traditional PI, on the other hand, evaluate a policy at each value-determination step. Therefore, they are useful to evaluate a given policy which is a major advantage of this approach.

Of course, once a policy is given the underlying process can also be analyzed as a continuous-time Markov chain which can be solved for its stationary distribution. Once the stationary distribution is obtained, the average number in the system can easily be found. Resnick [36] gives a numerical way of solving for the stationary probability vector η':

$$\eta' = (1, \ldots, 1)(A + ONE)^{-1},$$

where A is the generator matrix of the continuous-time Markov chain, and ONE is a matrix with all of its entries equal to 1. Once we have the stationary probability vector, we can compute the average number in the system which we are interested in. With our PI-based methodology the stationary distribution is not found. However, we have already seen that it can be used to find the average number in the system very efficiently.

On the other hand, the sparsity of the systems is lost by using the above formula to find the stationary distribution vector. Hence, it is not an efficient way of finding L for the size of problems we are solving.

Simulation is also an option to evaluate a policy. In this work, however, we focus on exact methods.

CHAPTER 4 COMPUTATIONAL RESULTS

We have written our programs in C/C++. All the programs were run on DEC Alpha workstations (100 MHz, 32 MB RAM). The public software LASPack [43] has been incorporated into our programs to solve the systems of linear equations. Our iterative method of choice was the biconjugate gradient stabilized method. We also needed to precondition our linear systems so that they had nicer convergence properties. A preconditioner is a matrix that transforms a linear system into one that has the same solution and nicer properties. To our luck, the so called symmetric successive over-relaxation preconditioner which does not require much extra work was enough in our experiments.

In the tables the optimal results correspond to the average number of customers in the system, L. ρ is the traffic intensity which is the ratio of the total arrival rate to the total service rate.

Some researchers worked on RNDA where there are no dedicated arrivals. We summarize their results to show the size of problems that could be solved previously. Krishnan [24] has developed a heuristic method for the RNDA problem, and compared it to the almost-optimal solution which he has obtained using PI. He could only report results on two-queue cases, and those only for cases where the capacity on each queue is 30. That corresponds to 961 states. Banawan and Zahorjan [2] also used PI to evaluate a heuristic rule. Their results were limited to about 1000-1500 states which they claim is a "reasonable limit". As opposed to Krishnan, they solved problems with up to four queues but they could only allow at most 10 people in the system! With a high traffic intensity their system will be full very frequently, and is not a good approximation to the actual uncapacitated system.

Table 4.1 demonstrates how the size could affect the quality of the solution. As a rule of thumb the quality of the solution values is indicated by a larger number since it reflects the true value of the uncapacitated system better. This, however, may not be true in all the cases. For example, when one queue is full the arrivals are sent to the other queue. On the other hand, with an uncapacitated system the optimal policy could have called for sending those customers to the queue that became full in the finite capacity problem. Nevertheless, in most of the cases the rule, "the larger the value the better", applies, and we will use this in our discussions.

In all of the cases in Table 4.1, there is quite a significant difference between the solutions when the queue capacities are increased from 30 to 50 and 75. Compared to the values with 30, solutions for 50 increase more than 30%, whereas with 75 more than 70%. These increases are expected: When the system capacity is comparatively small, more arrivals are rejected. That causes the average number in the system to be smaller for smaller systems. As the capacity is increased, more customers can enter the system, and L increases. The differences will not be this dramatic with lower traffic intensities. However, the high traffic intensity cases are the more interesting ones to look at for practical purposes.

Table 4.1 The effect of queue capacities on L. $\rho = 0.98$, RNDA

	$\mu_1 = 2$	$\mu_1 = 2.5$	$\mu_1 = 5$
Capacities	$\mu_2 = 1$	$\mu_2 = 1$	$\mu_2 = 1$
10	9.7617	9.7798	10.0664
30	24.5074	24.4970	24.5036
50	34.6273	34.6151	34.5974
75	42.2980	42.2843	42.2641
100	46.3250	46.3105	46.2891
150	49.2003	49.1849	49.1632
200	49.7771	49.7617	49.7404

We mentioned that Krishnan set both queue capacities to 30. We compared how the solution values change with different queue capacity combinations. The comparisons were made against keeping the queue capacities the same. The other queue capacities give about 10, 20 or 30 % decreases in the state space compared to the equal-capacity cases. We also tried to keep the queue ratios proportional to the service ratios. As one can see from the Tables 4.2- 4.4 the quality of the solutions are better even with 10% decrease in the state space. As the first server becomes faster, even a larger reduction in the state space brings about better solutions. This is because the faster server will be utilized more the faster it is, and therefore, it needs more buffer. One exception to our approximate rule can be seen from Table 4.4 and (29,2) case. When the second queue is full (which must happen pretty often with a queue capacity of 2), the customers are forced to join the first queue. The first queue cannot handle all the incoming traffic which increases the average number in the system. Then, the solution value of (29,2)-system is a poorer indicator than the values of other buffer allocations, e.g. (26,3).

Table 4.2 The effects of allocating the state space on L. $\rho = 0.98$, $\mu_1 = 2$, $\mu_2 = 1$, RNDA. Numbers in brackets give the capacities of queues

	Approximate percent reduction in state space		
Base case	10%	20%	30%
(10,10) 9.7617	(13,7) 9.7462	(12,6) 8.8708	(11,6) 8.4277
(30,30) 24.5074	(40,20) 24.5074	(38,19) 23.5724	(36,17) 22.2844
(50,50) 34.6273	(66,34) 34.6273	(62,31) 33.1493	(59,21) 32.0220
(75,75) 42.2980	(100,50) 42.2981	(94,17) 41.2413	(89,44) 40.1949
(100,100) 46.3250	(134,67) 46.3803	(126,63) 45.6618	(118,59) 44.8103
(150,150) 49.2003	(202,101) 49.2354	(190,95) 48.9994	(177,89) 48.6637
(200,200) 49.7771	(268,134) 49.7816	(254,127) 49.7282	(236,118) 49.6235

The changes we made in the PI-method speeds it up very dramatically. It becomes even a better alternative than VI-method. Table 4.5 shows the gain in run-time in PI by using the iterative methods instead of the direct methods. Table 4.6 compares the CPU seconds for the algorithms. Observe that as the traffic intensity becomes higher

Table 4.3 The effects of allocating the state space on L. $\rho = 0.98$. $\mu_1 = 5$, $\mu_2 = 1$, RNDA. Numbers in brackets give the capacities of queues

	Approximate percent reduction in state space		
Base case	10%	20%	30%
(10,10) 10.0664	(20,4) 11.4306	(19,4) 11.0081	(16,4) 9.7204
(30,30) 24.5036	(64,12) 29.0314	(60,12) 27.9603	(55,11) 26.2711
(50,50) 34.5974	(105,21) 39.1572	(100,20) 38.2252	(95,18) 37.0499
(75,75) 42.2641	(160,31) 45.7546	(150,30) 45.0015	(140,28) 44.0373
(100,100) 46.2891	(210,42) 48.3063	(200,40) 47.9706	(185,37) 47.3358
(150,150) 49.1632	(320,64) 49.6995	(300,60) 49.6139	(280,56) 49.4847
(200,200) 49.7404	(425,85) 49.8464	(400,80) 49.8338	(375,75) 49.8129

Table 4.4 The effects of allocating the state space on L. $\rho = 0.98$. $\mu_1 = 10$, $\mu_2 = 1$, RNDA. Numbers in brackets give the capacities of queues

	Approximate percent reduction in state space		
Base case	10%	20%	30%
(10,10) 10.9305	(26,3) 13.4667	(23,3) 12.2396	(29,2) 14.2905
(30,30) 24.8745	(90,9) 34.3652	(80,8) 31.9658	(75,8) 30.7775
(50,50) 34.6464	(150,15) 43.7395	(140,14) 42.6649	(130,13) 41.4231
(75,75) 42.2406	(220,22) 47.9978	(210,21) 47.6418	(200,20) 47.2215
(100,100) 46.2574	(300,30) 49.4096	(280,28) 49.2182	(260,26) 48.9432
(150,150) 49.1295	(450,45) 49.8066	(420,42) 49.7882	(390,39) 49.7552
(200,200) 49.7056	(600,60) 49.8279	(560,56) 49.8263	(530,53) 49.8251

Table 4.5 PI with direct vs. iterative methods in CPU seconds. RNDA, $\mu_1 = 2$, $\mu_2 = 1$, $N = 25$

ρ	PI-direct	PI-iterative
0.2	687.8	1.8
0.4	455.4	2.8
0.6	910.6	3.7
0.8	910.1	4.2
0.9	910.1	4.6

Table 4.6 Modified PI vs. VI in CPU seconds. RNDA, $\mu_1 = 2$, $\mu_2 = 1$, $N = 100$ for $\rho < 0.9$, $N = 200$ otherwise

ρ	MPI	VI-relaxation	VI plain
0.10	74.2	105.6	121.1
0.15	32.9	107.5	120.6
0.20	27.4	107.7	125.4
0.25	39.2	95.5	130.2
0.30	62.7	98.1	135.3
0.35	88.2	95.1	140.5
0.40	118.1	106.7	148.5
0.45	112.3	116.5	172.0
0.50	134.0	206.2	202.7
0.55	194.1	242.0	241.6
0.60	203.9	265.6	292.1
0.65	352.5	513.7	360.7
0.70	302.4	467.4	468.3
0.75	356.0	506.1	606.3
0.80	257.8	457.6	849.2
0.85	473.1	553.0	1310.4
0.90	7786.1	7378.2	13951.4
0.92	4570.1	12176.0	19024.7
0.95	5331.8	25882.9	36623.7
0.98	38526.3	37378.8	96271.5

the performance of PI does not worsen as badly as VI's. This is an important point because at low intensities finding a solution to the problem does not pose a difficulty. One can use any heuristic policy for low intensities, and because the next arrival will not arrive very soon it is very likely that customers in service will leave the system before then. Although VI with relaxation may sometimes perform better than our modified PI, in most cases our methodology also beats VI with relaxation. The convergence of VI with relaxation is theoretically not guaranteed. We actually had some problems which did not converge. Moreover, the extra work to find the relaxation factor and update the states may make the VI with relaxation slower than even the plain VI if the number of states is sufficiently large.

The run-time of the modified PI is directly related to the total number of iterations to solve the systems of linear equations until PI converges. Due to the fact that some problems are harder than some others to solve numerically the run-time does not increase with the increasing traffic intensity. Table 4.7 illustrates these points.

A more clever allocation of queue capacities speeds our methodology significantly. For example, Table 4.8 shows the difference in run-times for the problems in Table 4.6 for $\rho \geq 0.9$ with queue capacities (268,134).

The relaxation factor in the VI-method also causes an irregular run-time behaviour. However, its use helps in reducing the number of the iterations needed in VI. For a comparison in the number of iterations needed by VI-plain and VI-relaxation to converge see Table 4.9.

Table 4.7 Modified PI: CPU seconds and # of iterations. RNDA, $\mu_1 = 2$, $\mu_2 = 1$, $N = 100$ for $\rho < 0.9$, $N = 200$ otherwise

ρ	CPU	Total # iterations
0.10	74.2	117
0.15	32.9	50
0.20	27.4	41
0.25	39.2	60
0.30	62.7	98
0.35	88.2	140
0.40	118.1	189
0.45	112.3	180
0.50	134.0	216
0.55	194.1	315
0.60	203.9	331
0.65	352.5	576
0.70	302.4	493
0.75	356.0	581
0.80	257.8	420
0.85	473.1	774
0.90	7786.1	3208
0.92	4570.6	1887
0.95	5331.8	2199
0.98	38526.3	15919

Table 4.8 Effects of a more clever state space allocation in CPU seconds. RNDA, $\mu_1 = 2$, $\mu_2 = 1$, $N_1 = 268$, $N_2 = 134$

ρ	MPI	VI-relaxation
0.90	2241.9	10145.6
0.92	1558.2	17319.1
0.95	1491.6	36804.7
0.98	1922.6	78780.8

Table 4.9 Number of iterations for VI with and without relaxation. $N = 30$

Service ratio	ρ	VI-plain	VI-relaxation
2:1	0.2	220	190
	0.4	333	189
	0.6	731	534
	0.8	2390	671
	0.9	5783	1171
2.5:1	0.2	260	274
	0.4	335	271
	0.6	731	360
	0.8	2390	906
	0.9	5786	1959
5:1	0.2	460	379
	0.4	531	417
	0.6	749	470
	0.8	2398	1053
	0.9	5805	3122

CHAPTER 5 HEURISTIC METHODS

The state spaces in our problems grow very quickly as it has been demonstrated in Chapter 3. Although, the methodology we have discussed in that chapter is useful in obtaining the almost-optimal policy efficiently, it will become inappropriate for solving these type of problems when the state space becomes sufficiently large. Therefore, it is important to develop approximation methods that can be used to find good policies for very large systems. Here, we will discuss several heuristic methods that can be applied to the routing problem with dedicated arrivals.

We experiment with several different heuristic methods.

1. Never queue rule (NQ)

2. Individual-optimum rule (IOPT)

3. Separable rule (SR)

4. Greedy throughput rule (GT)

5. Hybrid rule (HYB)

5.1 Never Queue Rule

This rule sends an arrival to the fastest available server. If no server is available a queue with minimal n_i/μ_i is chosen. This rule differs only slightly from the individual-optimum rule where the service time of the new-coming customer is taken into account.

5.2 Individual-optimum Rule

This rule is widely used as a heuristic for different problems in controlled QN area. It minimizes the expected waiting time of an individual after arrival, but ignores the effect of a customer's self-interested policy on the other customers. For our problem it corresponds to joining the shortest queue. Some researchers use the term the shortest expected delay rule which is equivalent to joining the shortest queue in our case. When the servers are homogeneous this is the optimal policy. A customer is assigned to a queue that satisfies

$$\min_{i=1,\ldots,m} \frac{n_i + 1}{\mu_i}.$$

If there is a tie, the ties can be broken in favor of a queue that minimizes the variance (assuming exponential service distributions)

$$\min_{i=1,\ldots,m} \frac{n_i + 1}{\mu_i^2}.$$

5.3 Separable Rule

In PI, for a given policy one first needs to compute the relative values for each state. Then, these values are used to improve the old policy. The bottleneck in this procedure is finding the relative values. A candidate for a good heuristic is using a reasonable policy -if it exists- that allows finding the relative values without solving a system of linear equations , and then doing a one step policy-improvement.

A policy that makes computation of L and the relative values easy is the Bernoulli-splitting policy. Therefore, another good candidate for an approximate method is to use Bernoulli-splitting as your initial policy, compute its relative values, perform a one-step policy-improvement, and stop with the new policy. We have discussed the Bernoulli-splitting policy and how to improve on that policy in Chapter 3. It was shown that it is sufficient to look at the impact of a new arrival on individual queues separately to find the improved policy, hence the name: Separable rule.

5.4 Greedy Throughput Rule

This rule maximizes the expected number of service completions before the next expected customer arrival. With this rule the server that satisfies

$$\max_{i=1,\dots,m} \left(\frac{\mu_i}{\lambda_0 + \lambda_i + \mu_i} \right)^{n_i+1}$$

is chosen.

5.5 Hybrid Rule

Our experiments indicated that no previous rule has an overall advantage. We suggest the following rule as an alternative heuristic method: In any state, the action that is picked is the one chosen by the majority of the other rules. We call this rule the hybrid rule.

5.6 Computational Results

Krishnan [24] worked on the RNDA problem, and compared his heuristic method only to the IOPT rule to measure the performance of his heuristic. However, other rules do perform better than his as can be seen from Table 5.1.

Table 5.2- 5.4 show how the heuristic methods behave for different problems. The heuristic rules, in general, give reasonable solutions. In most cases, they are not more than 10% above the optimum solution. IOPT and NQ rules do not consider the arrival streams at all. For that reason, when the faster server is expected to be heavily loaded with dedicated arrivals they perform poorly against GT. GT, since it maximizes the

Table 5.1 Performance in L for different heuristic methods. Problems reported in Krishnan's paper. RNDA, $N = 30$

Service ratio	ρ	NQ	IOPT	SEP	GT	HYB	Optimal
2:1	0.2	0.3773	0.3904	0.3787	0.3773	0.3773	0.3773
	0.4	0.9419	0.9981	0.9511	0.9419	0.9419	0.9419
	0.6	1.9451	2.0610	1.9630	1.9462	1.9451	1.9451
	0.8	4.6594	4.8506	4.6797	4.6584	4.6594	4.6564
	0.9	9.6884	9.9238	9.7054	9.6817	9.6884	9.6774
2.5:1	0.2	0.3733	0.3660	0.3734	0.3659	0.3659	0.3659
	0.4	0.9410	0.9512	0.9425	0.9412	0.9410	0.9397
	0.6	1.9419	2.0009	1.9517	1.9429	1.9419	1.9419
	0.8	4.6480	4.7993	4.6766	4.6494	4.6480	4.6458
	0.9	9.6724	9.8916	9.7127	9.6791	9.6724	9.6638
5:1	0.2	0.3967	0.3154	0.3152	0.3152	0.3152	0.3150
	0.4	0.9935	0.8929	0.8832	0.8659	0.8659	0.8656
	0.6	1.9877	2.0484	1.9233	1.8936	1.8871	1.8871
	0.8	4.6771	5.0950	4.6957	4.6264	4.6047	4.6036
	0.9	9.7059	10.3906	9.7724	9.6743	9.6494	9.6336

number of service completions until the next expected customer arrival, considers the arrival streams partially. The SR, to a certain extent, considers the arrival streams as well since the optimal Bernoulli-splitting probabilities are also based on the arrival rates. The performance of the SR is difficult to assess because it involves a one-step policy-improvement. The hybrid rule's performance is also difficult to guess. Although it does achieve the best results in many cases, it may also perform very poorly compared to the others in some cases; the behaviour cannot be predicted beforehand.

Table 5.2 Performance in L for different heuristic methods. $\mu_1 = 2$, $\mu_2 = 1$, $N = 50$

λ_1, λ_2	ρ	NQ	IOPT	SEP	GT	HYB	Optimal
0.5, 0.5	0.4	1.5451	1.5324	1.5322	1.5451	1.5324	1.5322
	0.6	2.5578	2.5328	2.6149	2.5704	2.5578	2.5316
	0.8	5.3120	5.3141	5.4588	5.3710	5.3120	5.2955
	0.9	10.4735	10.5005	10.7047	10.5848	10.4735	10.4041
0.1, 0.1	0.2	0.4266	0.4293	0.4266	0.4266	0.4266	0.4266
	0.4	0.9949	1.0282	1.0062	0.9949	0.9949	0.9949
	0.6	2.0043	2.0910	2.0297	2.0072	2.0043	2.0043
	0.8	4.7271	4.8855	4.7640	4.7328	4.7271	4.7271
	0.9	9.8631	10.0662	9.9061	9.8690	9.8631	9.8394
1.0, 0.0	0.4	1.2200	1.2675	1.2164	1.2405	1.2200	1.2164
	0.6	2.2666	2.4390	2.2248	2.2783	2.2666	2.2176
	0.8	5.0980	5.4005	4.9686	4.9586	5.0980	4.9528
	0.9	10.3150	10.6935	10.1177	10.1170	10.3150	9.9803
0.4, 0.2	0.4	1.0778	1.0963	1.0860	1.0860	1.0778	1.0778
	0.6	2.1045	2.1729	2.1270	2.1270	2.1045	2.1045
	0.8	4.8537	4.9958	4.8878	4.8862	4.8537	4.8534
	0.9	10.0024	10.1916	10.0475	10.0417	10.0024	9.9361

Table 5.3 Performance in L for different heuristic methods. $\mu_1 = 2.5$, $\mu_2 = 1$, $N = 50$

λ_1, λ_2	ρ	NQ	IOPT	SEP	GT	HYB	Optimal
0.3, 0.3	0.2	0.6234	0.6189	0.6189	0.6189	0.6189	0.6189
	0.4	1.1913	1.1582	1.1869	1.1589	1.1589	1.1582
	0.6	2.2108	2.1859	2.2342	2.2162	2.2108	2.1835
	0.8	4.9418	4.9828	5.0276	4.9644	4.9418	4.9313
	0.9	10.0764	10.1585	10.2249	10.1312	10.0764	10.0389
0.1, 0.1	0.2	0.4290	0.4178	0.4178	0.4178	0.4178	0.4178
	0.4	1.0006	0.9933	1.0025	1.0009	1.0006	0.9923
	0.6	2.0083	2.0393	2.0220	2.0105	2.0083	2.0059
	0.8	4.7223	4.8362	4.7657	4.7301	4.7223	4.7223
	0.9	9.8512	10.0303	9.9201	9.8640	9.8512	9.8327
1.5, 0.0	0.6	2.3811	2.5294	2.3639	2.3886	2.3811	2.3429
	0.8	5.2156	5.5720	5.0918	5.0674	5.2156	5.0513
	0.9	10.4557	10.9467	10.2365	10.2385	10.4557	10.0673
0.5, 0.2	0.4	1.0810	1.0715	1.0825	1.0825	1.0810	1.0708
	0.6	2.1073	2.1289	2.1210	2.1210	2.1073	2.1048
	0.8	4.8478	4.9523	4.8936	4.8649	4.8478	4.8477
	0.9	9.9842	10.1557	10.0659	10.0504	9.9842	9.9240

Table 5.4 Performance in L for different heuristic methods. $\mu_1 = 5$, $\mu_2 = 1$, $N = 50$

λ_1, λ_2	ρ	NQ	IOPT	SEP	GT	HYB	Optimal
0.5, 0.5	0.2	1.1822	1.1628	1.1628	1.1628	1.1628	1.1628
	0.4	1.7955	1.6129	1.6126	1.6282	1.6129	1.6125
	0.6	2.8048	2.5740	2.5972	2.6902	2.5972	2.5719
	0.8	5.4828	5.3888	5.7106	5.5063	5.3856	5.3135
	0.9	10.5892	10.6436	11.0921	10.8258	10.5158	10.4567
0.1, 0.1	0.2	0.4691	0.3930	0.3929	0.3937	0.3929	0.3929
	0.4	1.0723	0.9440	0.9308	0.9308	0.9307	0.9302
	0.6	2.0698	2.0651	2.0067	1.9710	1.9613	1.9538
	0.8	4.7584	5.0758	4.8038	4.7190	4.6801	4.6800
	0.9	9.8826	10.4581	10.0048	9.8822	9.8209	9.8129
4.0, 0.0	0.8	5.9861	6.7382	5.7438	5.7430	6.0215	5.6285
	0.9	11.3240	12.6557	10.6422	10.7607	11.3543	10.4068
1.0, 0.2	0.4	1.1321	1.0268	1.0230	1.0498	1.0230	1.0226
	0.6	2.1587	2.1241	2.1072	2.0979	2.0617	2.0477
	0.8	4.8739	5.1343	4.9374	4.8932	4.8032	4.8032
	0.9	9.9835	10.4966	10.1569	10.0883	9.9289	9.8905

CHAPTER 6 CONCLUSION

In this dissertation we have looked at a controlled QN where a controller routed the incoming arrivals to parallel queues using state-dependent rules. The queues also had dedicated arrivals which could only be serviced by them. We assumed Poisson arrival processes, and exponential service times.

The problem was modeled as an SMDP. We pointed out a misconception in the literature about problems similar to ours. We developed a PI-based exact methodology which performed better than the current methods including VI which is widely thought as the method to use for large-scale problems. VI with relaxation converges -if at all- in fewer number of iterations than VI-plain. However, there could be two problems with this procedure; it is not guaranteed to converge, and with sufficiently large state space, the extra work involved in VI-relaxation could actually make it even slower than VI-plain.

We made several changes to the traditional PI-method to be able to solve our problem efficiently. Observing the interdependence of the solutions in consecutive iterations of PI-method made it possible to use the iterative methods in solving our systems of linear equations very effectively. Sparsity, a good initial solution, and allocation of the state space were other factors that affected the performance of our methodology. Using this methodology we solved much larger problems than reported in the literature. Our methodology is a candidate to solve other problems efficiently as well. One other advantage of our methodology was to use it to evaluate a given policy efficiently which cannot be done with other MDP techniques.

We also looked at how several heuristic methods performed on our problem. The study was comprehensive, and it included all the heuristic ideas we have encountered in the literature that could be applied to our problem. We have used our exact methodology rather than simulation to evaluate these methods. No heuristic method has surfaced as the best heuristic to use for all instances. In general, however, these heuristic methods offer very quick and reasonable solutions to very big problems. This is important since with MDP techniques the size of the problems one can solve is limited by the current computer technology.

Part of the research in this dissertation is a first-step in combining PI with iterative methods. We have demonstrated that PI is better than VI or LP for solving our problem. and we feel that the methodology we have proposed has large room for improvement. In our opinion, an iterative method that should be used in solving the systems of linear equations that arise in a particular problem should be tailored for that problem. For further research, one could pursue developing better iterative methods that take the characteristics of our problem into consideration. It is important to make efforts in this direction since the other techniques do not promise any more improvements. We also believe that this should be the path to take in solving other problems one may encounter in controlled QN settings.

APPENDIX A GENERAL SOLUTION TECHNIQUES

Markov decision processes provide a general framework to solve controlled QNs. White [55] [56] gives a list of real applications of MDPs. Markov chains can be used to model problems when the probabilistic law of motion is fixed. In MDPs, the interest lies in the policy that will achieve a certain performance measure the best. Since in practical cases the number of policies are finite, one can evaluate each policy using Markov chains and then pick the best policy. Often though, this is a very inefficient procedure. Let us first give a more formal description of MDPs. We observe a dynamic system at discrete points in an infinite time-horizon. Then, the system is classified into one of the finite states, and an action is taken. The discrete time points when the actions are taken are called decision epochs. The action set is assumed to be finite. An action results in an immediate cost (which does not depend on the history of the system, but only on the state you are in, and the action you have taken), and with some probability the system goes to another state. The probabilities also do not depend on the history of the system. If we observe the system in equidistant points than the underlying process is called a discrete-time MDP (DMDP). However, in many problems, the times between actions are random. Now assume, that the time till the next decision epoch only depends on the state you are in and the action you have taken. Such a model is referred to as semi-Markov decision process. A semi-Markov decision model can also be converted to a discrete-time model.

In general, an optimal policy does not need to be stationary (a stationary policy is one where at any time point the decision taken in a given state remains the same). It could be randomized, too. However, because of the Markovian assumptions, we only need to consider the stationary policies. For all practical purposes, the stationary policies also satisfy the so called unichain assumption which says that there exists some state which can be reached from any other state under a policy. This assumption is needed to prove the existence of a stationary distribution for a given policy.

We will not attempt to review all literature about MDPs. The following books are some good access points to literature. Ross [39] discusses the theory for these processes in his excellent book, while Tijms' [51] [50] approach is a more computational one. Hillier and Lieberman [14] also have an introductory chapter on MDPs.

There are three general algorithms to solve MDPs:

1. Policy iteration

2. Value iteration

3. Linear programming

In what follows we will focus on SMDPs. However, most of the discussions also apply to DMDPs.

Policy Iteration

Once a policy is fixed, the underlying process is nothing but a continuous-time Markov chain. Therefore, in theory we could go through the finite number of policies, each time solving for the Markov chain, and finally pick the best policy as our optimum. However, this turns out to be a very inefficient method most of the time. Instead, policy iteration method constructs a sequence of improved policies until the optimum is reached. Typically, this method goes only through a few number of policies to find the optimum. This method was developed by Howard [20]. The algorithm uses the so called relative values, $v_i(R)$, which give a relative measure of the effect of starting at state i on the total expected costs using the policy R. The difference $v_j(R) - v_i(R)$ has an economic interpretation: It represents the difference in total expected costs over an infinitely long period of time by starting in state i rather than j when using the policy R. Other notation we use is

$c_i(a)$: the expected costs until the next decision epoch if action a is chosen in the present state i,

$p_{ij}(a)$: the probability of being in state j in the next decision epoch if action a is taken in the present state i,

$T_i(a)$: the expected time until the next decision epoch if action a is taken in the present state i.

Now, the algorithm can be given as follows([51]):

1. (initialization) Choose a stationary policy R.

2. (value-determination step) To compute the average cost $g(R)$, and the relative values $v_i(R)$, $i \in I$, for the current rule R, solve the following system of linear equations for its unique minimum:

$$v_i = c_i(R_i) - gT_i(R_i) + \sum_{j \in I} p_{ij}(R_i)v_j, \qquad i \in I,$$
$$v_r = 0,$$

   where r is arbitrarily chosen.

3. (policy-improvement step) For each state i determine an action a_i that gives the minimum in

$$\min_{a \in A(i)} \left\{ c_i(a) - g(R)T_i(a) + \sum_{j \in I} p_{ij}(a)v_j(R) \right\}.$$

   The improved policy is the one which applies the actions that yield the minimum in the above equation for each state.

4. (convergence test) If the new policy is the same as the old one, stop. Otherwise replace the old policy with the new one, and go to the value-determination step.

Value Iteration

In VI one avoids solving a system of linear equations. Instead, a recursive solution approach from dynamic programming is used, i.e. recursively a sequence of value functions is computed which approximate the minimum average cost per unit time. Here, we need to introduce some more notation. Let $V_n(i)$ be the minimal total expected costs when n periods are left when the current state is i, and a final cost $V_0(j)$ is incurred if the system ends up in state j. The difference $V_n(i) - V_{n-1}(i)$ will come very close to the minimum average cost per unit time for large n. In the algorithm the value function $V_n(i)$ is computed from

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

starting with an arbitrarily chosen $V_0(i)$.

An alert reader must have recognized that in the above recursion relationship the random times between two consecutive decision epochs have not been taken into account. For the VI method to work for SMDPs, the system is converted into a DMDP and then the method is applied. We now will give the algorithm with the appropriate data transformation. This transformation is due to Schweitzer [40] (see also the related work of Lippman [29]). In the following, $0 < \tau \leq \min_{i,a} T_i(a)$.

1. Choose $V_0(i)$ such that $0 \leq V_0(i) \leq \min_a \{c_i(a)/T_i(a)\}$ for all i. Set $n = 1$.

2. Compute the functional equations for each $i \in I$ from

$$V_n(i) = \min_{a \in A(i)} \left\{ \frac{c_i(a)}{T_i(a)} + \frac{\tau}{T_i(a)} \sum_{j \in I} p_{ij}(a) V_{n-1}(j) + \left[1 - \frac{\tau}{T_i(a)} \right] V_{n-1}(i) \right\}.$$

   $R(n)$ is the stationary policy that minimizes the right side of the above equation.

3. Compute the bounds

$$b_n = \min_{j \in I} \{V_n(j) - V_{n-1}(j)\},$$
$$B_n = \max_{j \in I} \{V_n(j) - V_{n-1}(j)\}.$$

Stop with the current policy when $0 \leq (B_n - b_n) \leq \epsilon b_n$, where ϵ is a prespecified accuracy number. Otherwise, $n = n + 1$ and go to the second step.

The number of iterations in this algorithm can be significantly decreased by using a dynamic relaxation factor. This modification to the algorithm is due to Popyack et al. [35]. Although the modified method is not guaranteed to converge, it is useful for all practical purposes. Only the last step of the previously stated algorithm changes in this modified version. In step 3,

Determine the states m and M such that

$$V_n(m) - V_{n-1}(m) = b_n, \; V_n(M) - V_{n-1}(M) = B_n.$$

Compute the relaxation factor

$$\omega = (B_n - b_n)/$$
$$\left\{ B_n - b_n + \sum_{j \in I} [p_{mj}(R_n(m)) - p_{Mj}(R_n(M))] [V_n(j) - V_{n-1}(j)] \right\}.$$

For each state i, recompute

$$V_n(i) = V_{n-1}(i) + \omega[V_n(i) - V_{n-1}(i)]$$

Set, $n = n + 1$ and go to step 2.

In case of a tie when determining the states m (M) one chooses the minimizing (maximizing) state from the previous state if it is a candidate, too. Otherwise, the first state that achieves the minimum (maximum) can be chosen. When the relaxation factor is used the difference between the value functions from the current iteration to the next for the current m and M will be zero. This usually helps in decreasing the difference between the new lower and upper bound more quickly.

Linear Programming

Next, we give a general LP formulation for SMDPs that finds the optimal average cost g^*.

Minimize g

subject to

$$v_i - \sum_{j\in I} p_{ij}(a)v_j + gT_i(a) \geq c_i(a), \qquad i \in I, a \in A(i),$$

g and v_i unrestricted in sign.

An LP formulation for MDPs was first given by Manne [31]. Several other researchers worked on LP formulations including Denardo and Fox[8], Osaki and Mine [34], Derman [9], and Hordijk and Kallenberg [16].

Comparison of Standard Methods

The LP and PI methods are related. In fact, PI can be looked at as an LP in which more than one basic variable is being changed in every iteration (block pivoting). Nevertheless, both of these methods involve simultaneously solving a system of linear equations. PI usually converges in a few iterations. However, each iteration is computationally more involved than a simplex iteration because at each iteration a system of linear equations is solved simultaneously. On the other hand, the number of simplex iterations depends on the problem and this number could be very large. One should also note that the number of variables needed in the LP formulation is considerably more because one needs a variable for each state and for each action in that state.

In general, VI is considered the method to use for large systems. It is argued that solving a big system of linear equations is computationally more cumbersome than solving the recursive relationships of value functions in VI. Value functions give an approximation to the average cost per unit time. VI needs quite a few number of iterations before finding a solution within the tolerance limits. At each of these iterations, with a one-pass computation, the value functions of the states for the next iteration are computed using the values from the previous iteration. However, to compute each value function one has to consider the states that can be reached from that particular state because they are used in computing the value function for that state. Therefore, the computational burden of VI is directly related to the number of states, and to the number states that can be reached from each state. One advantage of VI is that it is very easy to write your own code.

VI with the relaxation factor may converge faster (require fewer iterations) if it converges at all. However, it is not guaranteed to converge. Also, with many many states the overhead that is required to find the relaxation factor and updating the value functions can be a burden. Therefore, the plain VI can actually finish earlier although with the relaxation the number of iterations is less.

Often, one is interested in how a particular policy will perform against the others. VI and LP cannot be used to evaluate a policy. Both of these methods yield an optimal policy only when they finish. PI, on the other hand, evaluates a policy at each value-determination step. Therefore, it is useful to evaluate a policy which is a major advantage of this algorithm.

APPENDIX B A RESOURCE ALLOCATION PROBLEM

If the controller uses a probabilistic rule rather than a state-dependent rule in taking its decisions, then the problem falls into the category of resource allocation problems. We have been calling this probabilistic rule the Bernoulli-splitting rule. Due to its wide applications this has been the topic for several research papers including Tang and van Vliet [49], Mitrani and Wright [32], and Lee [27] among others. Buzen and Chen [5] is one of the earlier publications on the subject. Although, in more general cases the optimal probabilities have to be obtained by solving a nonlinear optimization problem, with M/M/1 queues the closed-form solution for them can be written explicitly. These formulae have been given by Bonomi and Kumar [4]. Here we offer another and easier way of solving for them.

The problem can be stated as

minimize

$$\sum_{i=1}^{m} \frac{\lambda_i + p_i\lambda_0}{\mu_i - \lambda_i - p_i\lambda_0}$$

such that

$$\sum_{i=1}^{m} p_i = 1$$
$$0 \leq \lambda_i + p_i\lambda_0 \leq \mu_i, \qquad \forall i.$$

Now, consider a relaxation of this problem

minimize

$$\sum_{i=1}^{m} \frac{\lambda_i + p_i\lambda_0}{\mu_i - \lambda_i - p_i\lambda_0}$$

such that

$$\sum_{i=1}^{m} p_i = 1 \tag{B.1}$$

$$\lambda_i + p_i\lambda_0 \leq \mu_i, \qquad \forall i, \tag{B.2}$$

where we have ignored the lower bounds.

Both of these problems are separable and convex problems. Ibaraki and Katoh [21] show that for this type of problem if any solution is negative it can be set at 0. Their proof involves relaxation of the upper bound, but a similar proof applies here. Based on this knowledge one can give an easy algorithm to find the optimal probabilities.

1. Set $I = 1, ..., m$ and $k = m$.

2. Solve the relaxed problem for $\vec{p} = p_1, ..., p_i, ..., p_m$.

3. If $p_i \geq 0$, $\forall i$, stop.

4. Otherwise set $p_i = 0$ for all i with $p_i < 0$, say n of them. Set $k = k - n$ and $I = I - \{j \in I | p_j < 0\}$ and go to the second step.

It remains to figure out how to solve for $\vec{p}$ in Step 2. Consider a problem of the kind

optimize $f(\vec{p})$

such that $\vec{g}(\vec{p}) = \vec{c}$,

where $\vec{g}(\vec{p}) = \vec{c}$ represents a set of constraints with k of them. The so called Lagrangian Multiplier Rule can then be stated for this problem as

Theorem B.1 *Assume f and each g_i are differentiable and continuous. Let $\bar{p}$ an interior point which gives a relative minimum (or maximum). Then, there are numbers $\psi_1, ..., \psi_k, \psi_0$, called Lagrange multipliers, that are not all zero, such that*

$$\psi_1 g_1'(\bar{p}) + ... + \psi_k g_k'(\bar{p}) + \psi_0 f'(\bar{p}) = 0$$

$$\vec{g}(\bar{p}) = \vec{c}.$$

Let, $J(\bar{p})$ be the $(k+1)$ by m Jacobian matrix

$$\begin{bmatrix} \vec{g}'(\bar{p}) \\ \vec{f}'(\bar{p}) \end{bmatrix}.$$

Then, by the Lagrangian Multiplier Rule the system $[\psi_1, \ldots, \psi_k, \psi_0]J(\bar{p}) = 0$ of m linear equations in the $k+1$ unknowns has a non-trivial solution. In matrix theory, this is equivalent to every $(k+1)$ by $(k+1)$ submatrix of $J(\bar{p})$ having determinant zero. Therefore, to find the candidates $\bar{p}$ for solutions to the problem we set each such determinant to zero to get $m-k$ new conditions on $\vec{p}$ which one can combine with the original constraints.

The Jacobian for our relaxed problem is

$$\begin{bmatrix} 1 & 1 & \ldots & 1 \\ \frac{\lambda_0\mu_1}{(\mu_1-\lambda_1-p_1\lambda_0)^2} & \frac{\lambda_0\mu_2}{(\mu_2-\lambda_2-p_2\lambda_0)^2} & \cdots & \frac{\lambda_0\mu_m}{(\mu_m-\lambda_m-p_m\lambda_0)^2} \end{bmatrix}.$$

Every two by two submatrix should have determinant zero. Taking the first two columns as an example we obtain

$$\frac{\mu_1}{(\mu_1-\lambda_1-p_1\lambda_0)^2} = \frac{\mu_2}{(\mu_2-\lambda_2-p_2\lambda_0)^2}.$$

Solving for p_2 in terms of p_1 we get a quadratic equation with the following solutions

$$p_2(p_1) = \frac{\mu_1(\mu_2-\lambda_2) \pm (\mu_1-\lambda_1-p_1\lambda_0)\sqrt{\mu_1\mu_2}}{\lambda_0\mu_1}.$$

Since $\lambda_2 + p_2\lambda_0 < \mu_2$ we only need to consider the solution with the negative sign. Using the other submatrices we get every p_i in terms of one of them, say p_1. Now using Constraint B.1 we can solve for p_1 which yields

$$\frac{\mu_1}{\lambda_0}\left[\frac{\lambda_0 - \sum_{i=2}^{m}(\mu_i-\lambda_i) + \sum_{i=2}^{m}\sqrt{\mu_1\mu_i}}{\sum_{i=1}^{m}\sqrt{\mu_1\mu_i}}\right] - \frac{\lambda_1}{\lambda_0}\left[\frac{\sum_{i=2}^{m}\sqrt{\mu_1\mu_i}}{\sum_{i=1}^{m}\sqrt{\mu_1\mu_i}}\right].$$

In general, $\forall\, i$

$$p_i = \frac{\mu_i}{\lambda_0}\left[\frac{\lambda_0 - \sum\limits_{\substack{j=1\\ j\neq i}}^{m}(\mu_j - \lambda_j) + \sum\limits_{\substack{j=1\\ j\neq i}}^{m}\sqrt{\mu_i\mu_j}}{\sum\limits_{j=1}^{m}\sqrt{\mu_i\mu_j}}\right] - \frac{\lambda_i}{\lambda_0}\left[\frac{\sum\limits_{\substack{j=1\\ j\neq i}}^{m}\sqrt{\mu_i\mu_j}}{\sum\limits_{j=1}^{m}\sqrt{\mu_i\mu_j}}\right]. \tag{B.3}$$

BIBLIOGRAPHY

[1] Y. Arian and Y. Levy. Algorithms for generalized round-robin routing. *Operations Research Letters*, 12(5):313-319, 1992.

[2] S.A. Banawan and J. Zahorjan. Load sharing in heterogeneous queueing systems. In *Proceedings of IEEE INFOCOM '89*, pages 731–739, 1989.

[3] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* SIAM, Philadelphia, 1994.

[4] F. Bonomi and A. Kumar. Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler. *IEEE Transactions on Computers*, 39(10):1232–1250, 1990.

[5] J.P. Buzen and P.P.S. Chen. Optimal load balancing in memory hierarchies. In *Proceedings of IFIP*, pages 271–275, 1974.

[6] C.S. Chang, X.L. Chao, and M. Pinedo. A note on queues with Bernoulli routing. In *Proceedings of the 29th IEEE Conference on Decision and Control*, pages 897-902, 1990.

[7] M.B. Combe and O.J. Boxma. Optimization of static traffic allocation policies. *Theoretical Computer Science*, 125:17–43, 1994.

[8] E.V. Denardo and B.L. Fox. Multichain Markov renewal programs. *SIAM Journal of Applied Mathematics*, 16:468–487, 1968.

[9] C. Derman. *Finite State Markovian Decision Processes.* Academic Press, New York, 1970.

[10] A. Ephremides, P. Varaiya, and J. Walrand. A simple dynamic routing problem. *IEEE Transactions on Automatic Control*, 25(4):690–693, 1980.

[11] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations.* Springer-Verlag, New York, 1994.

[12] B. Hajek. Optimal control of two interacting service stations. *IEEE Transactions on Automatic Control*, 29(6):491–499, 1984.

[13] B. Hajek. Extremal splittings of point processes. *Mathematics of Operations Research*, 10(4):543–556, 1985.

[14] F.S. Hillier and G.J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, New York, 1990.

[15] M. Hlynka, D.A. Stanford, W.H. Poon, and T. Wang. Observing queues before joining. *Operations Research*, 42(2):365–371, 1994.

[16] A. Hordijk and L.C.M. Kallenberg. Linear programming and Markov decision chains. *Management Science*, 25(4):352–362, 1979.

[17] A. Hordijk and G.M. Koole. On the optimality of the generalized shortest queue policy. *Probability in the Engineering and Informational Sciences*, 4:477–487, 1990.

[18] A. Hordijk, G.M. Koole, and J.A. Loeve. Analysis of a customer assignment model with no state information. Technical Report TW-93-15, Leiden University, 1993.

[19] D.J. Houck. Comparison of policies for routing customers to parallel queueing systems. *Operations Research*, 35(2):306–310, 1987.

[20] R.A. Howard. *Dynamic Programming and Markov Processes*. Wiley, New York, 1960.

[21] T. Ibaraki and N. Katoh. *Resource Allocation Problems: Algorithmic Approaches*. The MIT Press, Cambridge, Massachusetts, 1988.

[22] A. Itai and Z. Rosberg. A golden ratio control policy for multi-access channel. *IEEE Transactions on Automatic Control*, 29:712–718, 1984.

[23] P.K. Johri. Optimality of the shortest line discipline with state-dependent service rates. *European Journal of Operational Research*, 41:157–161, 1989.

[24] K.R. Krishnan. Joining the right queue: A Markov decision rule. In *Proceedings of the 26th Conference on Decision and Control*, pages 1863–1868, 1987.

[25] P.R. Kumar and J. Walrand. Individually optimal routing in parallel systems. *Journal of Applied Probability*, 22:989–995, 1985.

[26] R.L. Larsen and A.K. Agrawala. Control of a heterogeneous two-server exponential queueing system. *IEEE Transactions on Software Engineering*, 9(4):522–526, 1983.

[27] H. Lee. Simultaneous determination of capacities in parallel M/M/1 queues. *European Journal of Operational Research*, 73:95–102, 1994.

[28] W. Lin and P.R. Kumar. Optimal control of a queueing system with two heterogeneous servers. *IEEE Transactions on Automatic Control*, 29(8):696–703, 1984.

[29] S.A. Lippman. Applying a new device in the optimization of exponential queueing systems. *Operations Research*, 23:687–710, 1975.

[30] Z. Liu and D. Towsley. Optimality of the round-robin routing policy. *Journal of Applied Probability*, 31:466–475, 1994.

[31] A.S. Manne. Linear programming and sequential decisions. *Management Science*, 6:259–267, 1960.

[32] I. Mitrani and P.E. Wright. Routing in the presence of breakdowns. *Performance Evaluation*, 20:151–164, 1994.

[33] L.M. Ni and K. Hwang. Optimal load balancing in a multiple processor system with many job classes. *IEEE Transactions on Software Engineering*, 11(5):491–496, 1985.

[34] S. Osaki and H. Mine. Linear programming algorithms for semi-Markovian decision processes. *Journal of Mathematical Analysis and Applications*, 22:256–381, 1968.

[35] J.L. Popyack, R.L. Brown, and C.C. White III. Discrete versions of an algorithm due to Varaiya. *IEEE Transactions on Automatic Control*, 24:503–504, 1979.

[36] S.I. Resnick. *Adventures in Stochastic Processes.* Birkhäuser, Boston, 1992.

[37] Z. Rosberg and P. Kermani. Customer routing to different servers with complete information. *Advances in Applied Probability*, 21:861–882, 1989.

[38] Z. Rosberg and A.M. Makowski. Optimal routing to parallel heterogeneous servers-small arrival rates. *IEEE Transactions on Automatic Control*, 35(7):789–796, 1990.

[39] S.M. Ross. *Applied Probability Models with Optimization Applications.* Holden-Day, San Francisco, 1970.

[40] P.J. Schweitzer. Iterative solution of the functional equations of undiscounted Markov renewal programming. *Journal of Mathematical Analysis and Applications*, 34:495–501, 1971.

[41] K. Seth. Optimal service policies, just after idle periods, in two-server heterogeneous queueing systems. *Operations Research*, 25(2):356–360, 1977.

[42] S. Shenker and A. Weinrib. The optimal control of heterogeneous queueing systems: A paradigm for load-sharing and routing. *IEEE Transactions on Computers*, 38(12):1724–1735, 1989.

[43] T. Skalicky. *LASPack Reference Manual.* Dresden University of Technology, 1995.

[44] M.J. Sobel. Throughput maximization in a loss queueing system with heterogeneous servers. *Journal of Applied Probability*, 27:693–700, 1990.

[45] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, Princeton, New Jersey, 1994.

[46] S. Stidham and R. Weber. A survey of Markov decision models for control of networks of queues. *Queueing Systems*, 13:291–314, 1993.

[47] R.H. Stockbridge. A martingale approach to the slow server problem. *Journal of Applied Probability*, 28:480–486, 1991.

[48] D. Stoyan. *Comparison Methods for Queues and Other Stochastic Models.* John Wiley & Sons, Chichester, 1983.

[49] C.S. Tang and M. van Vliet. Traffic allocation for manufacturing systems. *European Journal of Operational Research*, 75:171–185, 1994.

[50] H.C. Tijms. *Stochastic Modelling and Analysis: A Computational Approach.* Wiley, New York, 1986.

[51] H.C. Tijms. *Stochastic Models: An Algorithmic Approach.* Wiley, New York, 1994.

[52] D. Towsley, P.D. Sparaggis, and C.G. Cassandras. Optimal routing and buffer allocation for a class of finite capacity queueing systems. *IEEE Transactions on Automatic Control*, 37(9):1446–1451, 1992.

[53] J. Walrand. A note on "Optimal control of a queueing system with two heterogeneous servers". *Systems and Control Letters*, 4:131–134, 1984.

[54] R.R. Weber. On the optimal assignment of customers to parallel servers. *Journal of Applied Probability*, 15:406–413, 1978.

[55] D.J. White. Real applications of Markov decision processes. *Interfaces*, 15(6):73–83, 1985.

[56] D.J. White. Further real applications of Markov decision processes. *Interfaces*, 18(5):55–61, 1988.

[57] W. Whitt. Deciding which queue to join: Some counterexamples. *Operations Research*, 34(1):55–62, 1986.

[58] W. Winston. Optimality of the shortest line discipline. *Journal of Applied Probability*, 14:181–189, 1977.

[59] S.H. Xu. A duality approach to admission and scheduling controls of queues. *Queueing Systems*, 18:273–300, 1994.

[60] S.H. Xu and H. Chen. On the asymptote of the optimal routing policy for two service stations. *IEEE Transactions on Automatic Control*, 38(1):187–189, 1993.

[61] S.H. Xu and J.G. Shanthikumar. Optimal expulsion control-a dual approach to admission control of an ordered-entry system. *Operations Research*, 41(6):1137–1152, 1993.

[62] S.H. Xu and R. Righter J.G. Shanthikumar. Optimal dynamic assignment of customers to heterogeneous servers in parallel. *Operations Research*, 40(6):1126–1138, 1992.

[63] S.H. Xu and Y.Q. Zhao. Dynamic routing and jockeying controls in a two-station queueing system. To be published.

[64] T.P. Yum. The design and analysis of a semidynamic deterministic routing rule. *IEEE Transactions on Communications*, 29(4):498–504, 1981.