European Journal of Operational Research 142 (2002) 108–127

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Stochastics and Statistics

Exploiting structure in adaptive dynamic programming algorithms for a stochastic batch service problem

Katerina P. Papadaki, Warren B. Powell [*]

Department of Operations Research and Financial Engineering, School of Engineering/Applied Science, Princeton University, Princeton, NJ 08544, USA

Received 26 February 2001; accepted 9 August 2001

Abstract

The purpose of this paper is to illustrate the importance of using structural results in dynamic programming algorithms. We consider the problem of approximating optimal strategies for the batch service of customers at a service station. Customers stochastically arrive at the station and wait to be served, incurring a waiting cost and a service cost. Service of customers is performed in groups of a fixed service capacity. We investigate the structure of cost functions and establish some theoretical results including monotonicity of the value functions. Then, we use our adaptive dynamic programming monotone algorithm that uses structure to preserve monotonicity of the estimates at each iterations to approximate the value functions. Since the problem with homogeneous customers can be solved optimally, we have a means of comparison to evaluate our heuristic. Finally, we compare our algorithm to classical forward dynamic programming methods.

Keywords: Dynamic programming; Inventory theory

1. Introduction

In this paper we identify the importance of using structural results in forward dynamic programming algorithms. Classical forward dynamic programming methods estimate the value at time t of being in each state independently of neighboring states. Our structural forward dynamic programming algorithm uses the properties of the value function to make inferences of the value of states that we have not visited from the values of neighboring states. We consider the problem of optimizing the serving of batches of customers over time. The simplicity of this problem enables us to gain an understanding of the structure of the problem by establishing results on the shape of the value functions. The case of homogeneous customers

[*] Corresponding author. Tel.: +1-609-258-5373; fax: +1-609-258-3796.
E-mail address: powell@princeton.edu (W.B. Powell).

can be solved optimally using backward dynamic programming which gives us the ability to evaluate the performance of our methods. The eventual goal of the research is to develop approximations that can be used to produce computationally tractable algorithms for vector-valued problems such as the problem with heterogeneous customers (but we do not consider this generalization in this paper).

The batch service problem consists of customers stochastically arriving at a service station at discrete time intervals over a finite horizon, incurring a waiting cost and a service cost. The basic decision is whether or not a batch of customers should be served if it is less than the size of the service capacity. Thus, there is a tradeoff between customer waiting costs and the cost of serving a batch. The randomness in the model comes from the exogenous stochastic nonstationary arrival process. We assume stationary deterministic service cost per batch and waiting cost per customer.

There are several variations of the problem depending on whether or not we assume that certain variables and model parameters are nonstationary or stochastic. In the case where arrivals of customers, server availability, service cost of a batch and possibly other model parameters are all stochastic, our outcome space becomes multidimensional. We could also consider problems where customers belong to different classes. These variations produce state, outcome and decision spaces that are multidimensional, making the problem intractably complex. For such problems, we would have to resort to approximation techniques based on the principles of forward dynamic programming (see Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998), which use approximations of the value function to move forward in time, updating estimates of the value function as the algorithm proceeds.

In this paper, we study a simple scalar version of a stochastic, batch dispatching process, where we can easily obtain optimal solutions using classical techniques. We use the optimal solution to evaluate the quality of different approximation methods which could be applied to more complex problems. We then compare the optimal solution to two classes of forward dynamic programming techniques (performed in the context of finite horizon problems) using discrete value function approximations. The first class uses observations of the value of being in a state to update the estimate of the value function. The second does the same, but also uses the structure of the function to impose additional updating steps to maintain this function. The version of the algorithm that maintains the structure of the function produces errors that appear to be an order of magnitude smaller than the technique that does not impose structure. We feel that this experiment serves to highlight the importance of identifying and exploiting problem structure in the estimation of value functions.

Our basic problem has a fairly rich history in the literature. It was first studied by Kosten (1967) (see also Medhi, 1984; Kosten, 1973), who described a custodian who dispatches trucks whenever the number of waiting passengers exceeds a certain threshold. Deb and Serfozo (1973) were the first to prove that in steady state the optimal decision rule of this system was monotone and therefore had a control limit structure, thereby proving the optimality of Kosten's custodian. They assume that the waiting cost per customer is an increasing function of the number of customers waiting in the queue. Weiss and Pliska (1976) assume that the waiting cost per customer is a function $h(w)$ if a customer has a waiting time w. They show that the optimal service policy is to send the vehicle if the server is available and the marginal waiting cost is at least as large as the optimal long run average cost. This is termed a *derivative policy*, in contrast to the control limit policy proved by Deb and Serfozo. The basic model is generalized in Deb (1978a) to include switching costs, and has been applied to the study of a two-terminal shuttle system where one or two vehicles cycle between a pair of terminals (Ignall and Kolesar, 1972, 1974; Barnett, 1973; Deb, 1978b; Weiss, 1981; Deb and Schmidt, 1987).

Once the structure of an optimal policy is known, the primary problem is one of determining the expected costs for a given control strategy, and then using this function to find the optimal control strategy. There has developed an extensive literature on strategies for steady state bulk queues, beginning with the original paper by Bailey (1954) (which did not consider a control strategy). Several authors have suggested different types of control strategies, motivated by different cost functions than those considered above.

Neuts (1967) introduced a lower limit to the batch size and termed the result a general bulk service rule. Powell (1985) was the first to introduce general holding and cancellation strategies, where a vehicle departure might be cancelled for a fixed period of time if a dispatch rule has not been satisfied within a period of time (reflecting, for example, the inability to keep a driver sitting and waiting). Powell and Humblet (1986) present a general, unified framework for the analysis of a broad class of (Markovian) dispatch policies, assuming stationary, stochastic demands. Excellent reviews of this literature are contained in Chaudhry and Templeton (1983) and Medhi (1984).

Speranza and Ukovich (1994, 1996) study the multiproduct single link problem by determining frequencies at which several products have to be shipped to minimize transportation and inventory costs. They develop integer and mixed integer linear programming models. Bertazzi and Speranza (1999a,b) consider the shipments of products from an origin to a destination through one or several intermediate nodes, and present several classes of heuristic algorithms including decomposition of the sequence of links, an EOQ-type solution and a dynamic programming-based heuristic. However, both models assume supply and demand rates are deterministic and constant over time. In Bertazzi et al. (2000) techniques of neuro-dynamic programming are implemented to approximate a stochastic, multiproduct version of the problem.

This paper makes the following contributions:

1. We show that the value function for the problem is monotone. This property has been shown (see, for example, Puterman, 1994) for the problem with infinite capacity, but we have not seen it for the finite capacity case.
2. We also show that the value function is K-convex, where K is the capacity.
3. Using K-convexity, we are able to also establish monotonicity of the dispatch function, a result that is well-known for the infinite capacity case but has not been established for the finite capacity case.
4. Finally, we show that a forward dynamic programming algorithm that exploits the structure of the value function (in particular monotonicity) dramatically improves solution quality when compared against the optimal solution.

We begin by defining the basic problem in Section 2. Section 3 provides theoretical results on the homogeneous batch service problem including structural properties of cost functions and a proof of the control limit structure of the service process. In Section 4 we introduce our monotone adaptive dynamic programming algorithm and compare it to the classical forward dynamic programming algorithm. The experiments and results are described in Section 5.

2. Problem definition

In this section we formally introduce the problem for homogeneous customers and we develop the basic model which is used for the rest of the paper. We state the assumptions, define the notation and the functions associated with the model.

We consider the problem of customers arriving at a station waiting to be served. Customers are served in groups by a server with finite service capacity. The service of customers can occur at discrete points in time called decision epochs and we evaluate the model for finitely many decision epochs (finite time horizon). Arrivals of customers occur at the service station at each time period; we assume only that the number of arrivals in each time period is independent of other time periods. Apart from arrivals and variables that depend on arrivals, all other variables, functions and parameters are assumed to be deterministic. The problem consists of determining optimal service policies to minimize total discounted costs by finding a trade-off between service and waiting costs.

Parameters

c	cost to perform a batch service
h	penalty per time period per customer for waiting at the station
K	service capacity, maximum number of customers that can be served in a batch
T	planning horizon for the model
α	discount factor

We assume constant cost parameters of batch service and waiting per customer.

2.1. The stochastic process

We assume an exogeneous stochastic process of arrivals. Let

$$\omega = (a_1, a_2, a_3, \dots, a_T),$$

where $a_t \in \mathscr{A}$ is a realization of the number of customers arriving at the station at decision epoch t, and $\mathscr{A}$ is the set of all possible arrivals. ω is a particular instance of a complete set of arrivals.

We can now define a standard probability space $(\Omega, \mathscr{F}, \mathscr{P})$, where Ω is the sample space of all ω's, $\mathscr{F}$ is the σ-algebra defined over Ω, and P is a probability measure defined over $\mathscr{F}$. We refer to $\mathscr{F}$ as the information field. We go further and let $\mathscr{F}_t$ be the information sub-field at time t, representing the set of all possible events up to time t. Since we have $\mathscr{F}_t \subseteq \mathscr{F}_{t+1}$, the process $\{\mathscr{F}_t\}_{t=1}^{T}$ is a filtration.

We define $A_t : \Omega \to \mathscr{A}$ such that $A_t(\omega) = a_t$ to be the random variable that determines the arrivals at time t. The process $\{A_t\}_{t=1}^{T}$ is our stochastic arrival process.

Now, we are ready to define our state variable. The physical quantity that the state describes is the number of customers waiting at the station to be served at a specified time epoch. We let $\mathscr{S}$ be the set of allowable states:

$S_t : \Omega \to \mathscr{S}$.
S_t: the random variable that determines the number of customers at the station at time t.
s_t: a realization of the number of customers at the station at time t.

The state of the system at time t, s_t, is measured at decision epoch t. We assume that arrivals occur at the station just before the decision epoch and service occurs just after the decision epoch.

2.2. Policies

We define the decision variables as follows:

$$z_t = \begin{cases} 1 & \text{if a service is performed at time } t, \\ 0 & \text{otherwise,} \end{cases}$$

$$\mathbf{z} = \{z_0, \dots, z_{T-1}\}.$$

We define the decision rules by $Z_t : \mathscr{S} \to \{0, 1\}$, where $\{0, 1\}$ is our action space with action 1 for service and 0 for no service. We define our decision rules to depend only on the current state and not on the history of states. A set of decision rules over the time horizon is a policy π;

$$\pi = (Z_0^\pi, Z_1^\pi, \dots, Z_{T-1}^\pi).$$

The set of all policies π is denoted Π.

2.3. System dynamics

Given that at time t, we were in state $S_t(\omega)$ and we chose action z_t, and given that the arrivals at time $t+1$ are $A_{t+1}(\omega)$, then we can derive the state variable at time $t+1$.

$$S_{t+1}(\omega) = [S_t(\omega) - Kz_t]^+ + A_{t+1}(\omega) \tag{1}$$

for all $\omega \in \Omega$. For a given $\omega \in \Omega$, we can generate the state process $\{S_t^\pi\}_{t=0}^T$ using (1) according to some policy π.

We are now ready to introduce the transition probabilities

$$q_t(i) = \text{Prob}(A_t = i),$$

$$p_{t+1}(s_{t+1}|s_t, z_t) = \text{Prob}(S_{t+1} = s_{t+1}|s_t, z_t),$$

$$p_{t+1}(j|s,0) = \begin{cases} q_{t+1}(j-s), & j \geqslant s, \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

$$p_{t+1}(j|s,1) = \begin{cases} q_{t+1}(j), & s \leqslant K,\ j \geqslant 0, \\ q_{t+1}(j-(s-K)), & s > K,\ j \geqslant s-K, \\ 0, & \text{otherwise}, \end{cases} \tag{3}$$

$$P_{t+1}(s_{t+1}|s_t, z_t) = \sum_{j=s_{t+1}}^{\infty} p_{t+1}(j|s_t, z_t). \tag{4}$$

2.4. Cost functions

We use the following cost functions:

- $g_t(s_t, z_t)$ = cost incurred in period t, given state s_t and service decision $z_t[= cz_t + h(s_t - Kz_t)^+]$,
- $g_T(s_T)$ = terminal cost function,
- $F(S_0)$ $= \min_{\pi\in\Pi} E\{\sum_{t=0}^{T-1} \alpha^t g_t(S_t, Z_t^\pi(S_t)) + \alpha^T g_T(S_T)\}$.

We calculate g_t in period t. The objective function $F(S_0)$ is the expected total discounted cost minimized over all policies π.

To simplify notation we let

$$w_t(s_t, z_t) \equiv g_t(s_t, z_t) + \alpha \sum_{s'\in\mathscr{S}} p_{t+1}(s'|s_t, z_t) V_{t+1}(s').$$

Finally, we define the value functions to be the functions V_t, for $t = 0, 1, \ldots, T$, such that the following recursive set of equations holds:

$$V_t(s_t) = \min_{z_t} \{g_t(s_t, z_t) + \alpha E[V_{t+1}(s_{t+1})|s_t, z_t]\} \tag{5}$$

for $t = 0, 1, \ldots, T-1$. These are the optimality equations which can also be written in the form

$$V_t(s_t) = \min_{z_t} \left\{ g_t(s_t, z_t) + \alpha \sum_{s'\in\mathscr{S}} p_{t+1}(s'|s_t, z_t) V_{t+1}(s') \right\}, \tag{6}$$

$$= \min_{z_t} \{w_t(s_t, z_t)\} \tag{7}$$

for $t = 0, \ldots, T-1$. Our goal is to solve the optimality equations which is equivalent to solving the objective function $F(S_0)$. For a scalar state variable this can be solved optimally using dynamic programming techniques.

3. Theoretical results

In this section we establish some theoretical results for the problem with homogeneous customers, in the form of structural properties. The major properties are stated and explained in Section 3.1 along with some technical properties. Section 3.2 provides proofs for all the properties. These results are used in Section 3.3 to derive corollaries on the behavior of the optimal decision rules.

3.1. Structural properties

The model we developed for the homogeneous customer problem has certain structural properties that we state in this section. These results are used to develop approximations of the value function and approximate the optimal decision rule.

We begin with a few definitions.

Definition 1. Let $f(x,y)$ be a real valued function on $X \times Y$. We say that f is submodular on $X \times Y$, if for $x^+ \geqslant x^-$ in X and $y^+ \geqslant y^-$ in Y,

$$f(x^+,y^+) - f(x^+,y^-) \leqslant f(x^-,y^+) - f(x^-,y^-). \tag{8}$$

There is an easier way to check for submodularity of a function which is presented in Puterman (1994, Lemma 4.7.6, p. 110).

Lemma 3.1. *Let $f(s,z)$ be a real valued function defined on $S \times Z$, with $Z = \{0,1\}$ and $S = \{0,1,\ldots\}$. If $f(s,z)$ satisfies*

$$f(s+1,1) - f(s+1,0) \leqslant f(s,1) - f(s,0) \tag{9}$$

for all s, it is submodular on $S \times Z$.

During the following discussion we use both of the above methods to prove submodularity. Another important property is what we call K-convexity:

Definition 2. A function u defined on $S = \{0,1,2,\ldots\}$ is *K-convex* if for $s^+ > s^-$:

$$u(s^+ + K) - u(s^- + K) \geqslant u(s^+) - u(s^-) \tag{10}$$

for some positive integer K.

In our work, K represents the capacity of the batch.

We start with the structural properties.

3.1.1. Major properties

Property 1. *The value function $V_t(s)$ is nondecreasing in s for $t = 0, \ldots, T$.*

Property 2. *The value function $V_t(s)$ is K-convex for $t = 0, \ldots, T$.*

We defer the proofs of these properties until later.

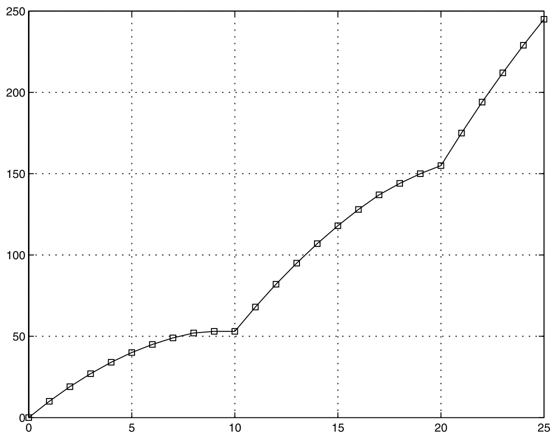

Fig. 1. A K-convex function with $K = 10$.

Properties 1 and 2 characterize a nondecreasing value function with the extra property of K-convexity (see Fig. 1); these properties are important because they give us insights into the shape of the value function. Knowledge of the shape of this function gives us ideas on what kind of approximations to use. Since these properties are structural results, they are particularly useful in deriving functional approximations. Suppose we have approximated the value function $\hat{V}$, at a specific state $\bar{s}$, then monotonicity gives us information about the value at other states. For example, if $s > \bar{s}$, then $V(s) > \hat{V}(\bar{s})$ and for $s < \bar{s}$, we have $V(s) < \hat{V}(\bar{s})$. K-convexity is important when comparing states that are K units apart.

We also introduce the following technical properties that are either intermediate results that arise in the process of deriving the major properties (Properties 3–6) or corollaries derived from the major results (Property 7).

3.1.2. Technical properties

Property 3. *The cost function $g_t(s,z)$ is nondecreasing in s.*

Property 4. *The cost function $g_t(s,z)$ is submodular on $S \times \{0,1\}$.*

Property 5. *$P_{t+1}(s'|s,z)$ is nondecreasing in s for all $s' \in \mathscr{S}$, $z \in \{0,1\}$.*

Property 6. $w_t(s^+ + K, 1) - w_t(s^- + K, 1) = w_t(s^+, 0) - w_t(s^-, 0)$.

Property 7. *The function $\sum_{j=0}^{\infty} p_{t+1}(j|s,z)V_t(j)$ is submodular for all $t = 1, \ldots, T$.*

3.2. Proofs of properties

The aim of this section is to prove the major structural results and the technical properties stated in Section 3.1.

We start with the proofs of Properties 3–5. These along with a lemma from Puterman (1994) give us the result of monotonicity of the value function (Property 1).

Property 3 is obvious from the definition of g_t.

Proof of Property 4. We want to show submodularity of g_t by establishing:

$$g_t(s^+,1) - g_t(s^+,0) \leqslant g_t(s^-,1) - g_t(s^-,0). \tag{11}$$

Substituting the function g_t into (11), we must show that

$$c + h(s^+ - K)^+ - hs^+ \leqslant c + h(s^- - K)^+ - hs^-,$$

which simplifies to

$$(s^+ - K)^+ - s^+ \leqslant (s^- - K)^+ - s^-. \tag{12}$$

Since $s^+ \geqslant s^-$, $(s^+ - K)^+ = 0 \Rightarrow (s^- - K)^+ = 0$. This implies there are three possible cases for Eq. (12):

Case 1. $(s^+ - K)^+ > 0$ and $(s^- - K)^+ > 0$. In this case, (12) reduces to $K = K$.

Case 2. $(s^+ - K)^+ > 0$ and $(s^- - K)^+ = 0$. Here (12) reduces to $s^- \leqslant K$, which follows since $(s^- - K)^+ = 0$ implies that $s^- \leqslant K$.

Case 3. $(s^+ - K)^+ = 0$ and $(s^- - K)^+ = 0$. Now (12) reduces to $s^- \leqslant s^+$. □

Proof of Property 5. Property 5 is equivalent to showing that

$$P(s'|s+1,z) \geqslant P(s'|s,z),$$

which is the same as showing

$$\text{Prob}[(s + 1 - Kz)^+ + A \geqslant s'] \geqslant \text{Prob}[(s - Kz)^+ + A \geqslant s'].$$

Let $X(\omega|s,z) = (s + 1 - Kz)^+ + A(\omega)$ and $Y(\omega|s,z) = (s - Kz)^+ + A(\omega)$. Clearly $X(\omega|s,z) \geqslant Y(\omega|s,z)$. Define the events $\mathscr{E}_Y = \{\omega|Y(\omega|s,z) \geqslant s'\}$ and $\mathscr{E}_X = \{\omega|X(\omega|s,z) \geqslant s'\}$. $X \geqslant Y$ implies that $\mathscr{E}_Y \subseteq \mathscr{E}_X$ which implies that $\text{Prob}[X \geqslant s'] = \text{Prob}[\mathscr{E}_X] \geqslant \text{Prob}[\mathscr{E}_Y] = \text{Prob}[Y \geqslant s']$, which proves our result. □

We use the following result from Puterman (1994, Lemma 4.7.3, p. 106) to prove Property 1.

Lemma 3.2. *Suppose the following statements hold*:

1. $g_t(s,z)$ *is nondecreasing in s for all* $z \in \{0,1\}$ *and* $t = 0,\ldots,T-1$.
2. $P_{t+1}(s'|s,z)$ *is nondecreasing in s for all* $s' \in \mathscr{S}$, $z \in \{0,1\}$.
3. $g_T(s)$ *is nondecreasing in s.*

Then $V_t(s)$ *is nondecreasing in s for* $t = 0,\ldots,T$.

We are now ready to prove monotonicity of the value function:

Proof of Property 1. The first assumption of Lemma 3.2 comes from Property 3. The second assumption is Property 5. The third assumption we have for free if we assume a zero terminal cost function: $g_T(s) = 0$. Thus the value function $V_t(s)$ is nondecreasing for all $t = 0,\ldots,T$. □

The next step is to prove K-convexity of the value function (Property 2). In order to do that we first prove Property 6 and two intermediate lemmas. Lemma 3.4 is proved by induction on t and it uses Lemma 3.3 and Property 6 in the induction step.

Proof of Property 6. We substitute w_t in (6) and split it into two equalities. The first contains only the immediate cost functions g_t and the second contains the expected value functions for the next time period. Adding the two equalities gives the desired result.

Thus we first prove:

$$g_t(s^+ + K, 1) - g_t(s^- + K, 1) = g_t(s^+, 0) - g_t(s^-, 0).$$

Substituting g_t, we get

$$c + h(s^+ + K - K)^+ - c - h(s^- + K - K)^+ = hs^+ - hs^-,$$

which is obviously true after performing the cancellations.

We also need to prove that

$$\sum_{j=0}^{\infty} p_{t+1}(j|s^+ + K, 1)V_{t+1}(j) - \sum_{j=0}^{\infty} p_{t+1}(j|s^- + K, 1)V_{t+1}(j) \\ = \sum_{j=0}^{\infty} p_{t+1}(j|s^+, 0)V_{t+1}(j) - \sum_{j=0}^{\infty} p_{t+1}(j|s^-, 0)V_{t+1}(j). \tag{13}$$

Substituting the transfer probabilities from (2) and (3) we have the following equality:

$$\sum_{j=s^+}^{\infty} q_{t+1}(j - s^+)V_{t+1}(j) - \sum_{j=s^-}^{\infty} q_{t+1}(j - s^-)V_{t+1}(j) \\ = \sum_{j=s^+}^{\infty} q_{t+1}(j - s^+)V_{t+1}(j) - \sum_{j=s^-}^{\infty} q_{t+1}(j - s^-)V_{t+1}(j) \tag{14}$$

that is obviously true. □

Lemma 3.3. *If u is a nondecreasing function on $S = \{0, 1, \ldots\}$ and it is K-convex, then $\sum_{j=0}^{\infty} p_{t+1}(j|s, z)u(j)$ is a submodular function on $S \times \{0, 1\}$.*

Proof of Lemma 3.3. We use Lemma 3.1 to prove submodularity. So we need to prove that

$$\sum_{j=0}^{\infty} p_{t+1}(j|s+1, 1)u(j) - \sum_{j=0}^{\infty} p_{t+1}(j|s+1, 0)u(j) \leqslant \sum_{j=0}^{\infty} p_{t+1}(j|s, 1)u(j) - \sum_{j=0}^{\infty} p_{t+1}(j|s, 0)u(j). \tag{15}$$

Case 1. $s, s+1 \leqslant K$. Then (15) becomes

$$\sum_{j=0}^{\infty} q_{t+1}(j)u(j) - \sum_{j=s+1}^{\infty} q_{t+1}(j - s - 1)u(j) \leqslant \sum_{j=0}^{\infty} q_{t+1}(j)u(j) - \sum_{j=s}^{\infty} q_{t+1}(j - s)u(j),$$

which simplifies to

$$\sum_{j=0}^{\infty} q_{t+1}(j)u(j) - \sum_{j=0}^{\infty} q_{t+1}(j)u(j + s + 1) \leqslant \sum_{j=0}^{\infty} q_{t+1}(j)u(j) - \sum_{j=0}^{\infty} q_{t+1}(j)u(j + s). \tag{16}$$

Since u is nondecreasing: $u(j + s + 1) \geqslant u(j + s)$. Thus (15) is satisfied.

Case 2. s, $s + 1 \geqslant K$. Then (15) becomes

$$\sum_{j=s+1-K}^{\infty} q_{t+1}(j - s - 1 + K)u(j) - \sum_{j=s+1}^{\infty} q_{t+1}(j - s - 1)u(j) \\ \leqslant \sum_{j=s-K}^{\infty} q_{t+1}(j - s + K)u(j) - \sum_{j=s}^{\infty} q_{t+1}(j - s)u(j), \tag{17}$$

which simplifies to

$$\sum_{j=0}^{\infty} q_{t+1}(j)[u(j+s+1-K) - u(j+s+1)] \leqslant \sum_{j=0}^{\infty} q_{t+1}(j)[u(j+s-K) - u(j+s)].$$

By K-convexity, we get the desired result. □

The following lemma uses Lemma 3.3 recursively to establish sufficient conditions for K-convexity.

Lemma 3.4. *Suppose the following statements hold*:

1. $g_t(s,z)$ *is nondecreasing in* s *for all* $z \in \{0,1\}$, *and submodular in* $\mathscr{S} \times \{0,1\}$ *for* $t = 0,\dots,T-1$.
2. $P_{t+1}(s'|s,z)$ *is nondecreasing in* s *for all* $s' \in \mathscr{S}$, $z \in \{0,1\}$, *and* $t = 0,\dots,T-1$.
3. $g_T(s)$ *is nondecreasing in* s *and* K-*convex*.

Then $V_t(s)$ *is* K-*convex for* $t = 0,\dots,T$.

Proof of Lemma 3.4. First note that by Lemma 3.2 and assumptions 1–3 of Lemma 3.4, we can conclude that V_t is nondecreasing for all t. We use an inductive argument to prove the proposition. Suppose V_{t+1} is K-convex. Then by Lemma 3.3 we can conclude that $\sum_{j=0}^{\infty} p_{t+1}(j|s,z)V_{t+1}(j)$ is submodular on $\mathscr{S} \times \{0,1\}$. The sum of two submodular functions is submodular; thus by Property 4, $w_t(s,z)$ is submodular

$$w_t(s^+,1) - w_t(s^-,1) \leqslant w_t(s^+,0) - w_t(s^-,0). \tag{18}$$

Define $\Delta w_t(s) = w_t(s,1) - w_t(s,0)$ for $s \in \mathscr{S}$. By submodularity of w_t, we conclude that $\Delta w_t(s)$ is a nonincreasing sequence in s:

$$\Delta w_t(0) \geqslant \Delta w_t(1) \geqslant \cdots \geqslant \Delta w_t(i) \geqslant \cdots$$

Let n be such that $\Delta w_t(n)$ is the first negative term in the sequence Δw_t.

For $s \geqslant n$, $\Delta w_t(s) < 0 \Rightarrow w_t(s,1) < w_t(s,0) \Rightarrow V_t(s) = w_t(s,1)$.
For $s < n$, $\Delta w_t(s) \geqslant 0 \Rightarrow w_t(s,0) \leqslant w_t(s,1) \Rightarrow V_t(s) = w_t(s,0)$.

Using (18) and Property 6 and the above we prove the result for the following five cases:

Case (*a*): $s^+ + K$, $s^- + K$, s^+, $s^- \geqslant n$

$$\begin{aligned} V_t(s^+ + K) - V_t(s^- + K) &= w_t(s^+ + K, 1) - w_t(s^- + K, 1) = w_t(s^+,0) - w_t(s^-,0) \\ &\geqslant w_t(s^+,1) - w_t(s^-,1) = V_t(s^+) - V_t(s^-). \end{aligned}$$

Case (*b*): $s^- < n \leqslant s^+$, $s^- + K$, $s^+ + K$

$$\begin{aligned} V_t(s^+ + K) - V_t(s^- + K) &= w_t(s^+ + K, 1) - w_t(s^- + K, 1) = w_t(s^+,0) - w_t(s^-,0) \\ &> w_t(s^+,1) - w_t(s^-,0) = V_t(s^+) - V_t(s^-), \end{aligned}$$

since $\Delta w_t(s^+) < 0$.

Case (*c*): s^-, $s^+ < n \leqslant s^- + K$, $s^+ + K$

$$V_t(s^+ + K) - V_t(s^- + K) = w_t(s^+ + K, 1) - w_t(s^- + K, 1) = w_t(s^+,0) - w_t(s^-,0) = V_t(s^+) - V_t(s^-).$$

Case (*d*): $s^-, s^+, s^- + K < n \leqslant s^+ + K$

$$V_t(s^+ + K) - V_t(s^- + K) = w_t(s^+ + K, 1) - w_t(s^- + K, 0) \geqslant w_t(s^+ + K, 1) - w_t(s^- + K, 1)$$
$$= w_t(s^+, 0) - w_t(s^-, 0) = V_t(s^+) - V_t(s^-),$$

since $\Delta w_t(s^- + K) \geqslant 0$.

Case (*e*): $s^-, s^+, s^- + K, s^+ + K < n$

$$V_t(s^+ + K) - V_t(s^- + K) = w_t(s^+ + K, 0) - w_t(s^- + K, 0) \geqslant w_t(s^+ + K, 1) - w_t(s^- + K, 1)$$
$$= w_t(s^+, 0) - w_t(s^-, 0) = V_t(s^+) - V_t(s^-).$$

We have proved that if V_{t+1} is K-convex then V_t is K-convex. Since $V_T = g_T$ is assumed to be K-convex, we have by induction that $V_t(s)$ is K-convex for $t = 0, \dots, T$. □

Now we are ready to prove K-convexity of the value function:

Proof of Property 2. We have already proved that in our model the one period cost function g_t is nondecreasing and submodular (Properties 3 and 4) and that P_{t+1} is nondecreasing in s (Property 5). Thus assumptions 1 and 2 of Lemma 3.4 are satisfied. Assume a zero terminal cost function which is both nondecreasing and K-convex and it satisfies assumption 3 of Lemma 3.4. Thus Lemma 3.4 gives us K-convexity of the value function for all time periods. □

Property 7 is a technical corollary that is of interest in Section 3.3.

Proof of Property 7. By Properties 1 and 2 we have that V_t is both K-convex and nondecreasing for all $t = 0, \dots, T$. Thus by Lemma 3.3 we conclude that $\sum_{j=0}^{\infty} p_{t+1}(j|s,z)V_t(j)$ is submodular for all $t = 0, \dots, T$. □

3.3. Monotone optimal policies

Deb and Serfozo (1973) showed that in steady state the optimal service strategy follows a control limit structure, which is to say that the service should be performed only when the length of the queue exceeds a particular limit. We prove optimality of policies with a nonstationary control limit. Puterman (1994) proves a similar general result on monotone policies:

Theorem 1 (Puterman, 1994, theorem 4.7.5, p. 108). *Suppose the following statements hold*:

1. *$g_t(s,z)$ is submodular on $\mathscr{S} \times \{0,1\}$ and nondecreasing in s.*
2. *$P_{t+1}(s'|s,z)$ is nondecreasing in s for all $s' \in \mathscr{S}$ and $z \in \{0,1\}$.*
3. *$\sum_{j=0}^{\infty} p_{t+1}(j|s,z)u(j)$ is a submodular function on $S \times \{0,1\}$ whenever u is a nondecreasing function on $S = \{0,1,\dots\}$.*
4. *$g_T(s) = V_T(s)$ is nondecreasing in s.*

Then there exists an optimal decision rule $z_t^*(s) \in \{0,1\}$ which is nondecreasing in s for all t.

The homogeneous customer batch service problem does not satisfy condition 3 of the above theorem. Monotonicity of the value function is not a sufficient condition to prove submodularity of $\sum_{j=0}^{\infty} p_{t+1}(j|s,z)V(j)$. According to Lemma 3.3 we also need the extra assumption of K-convexity of the value function. Using the structural properties of Section 3.1 we prove our version of the result which involves K-convexity.

Proposition 1. *For the homogeneous customer batch service problem there exists an optimal decision rule $z_t^*(s) \in \{0, 1\}$ which is nondecreasing in s for all t.*

We use the following lemma from Puterman (1994, Lemma 4.7.1, p. 104 and Exercise 4.1.7, p. 115).

Lemma 3.5. *Suppose w is a submodular function on $X \times Y$ and for each $x \in X$, $\min_{y \in Y} w(x, y)$ exists. Then*

$$f(x) = \max\left\{y' \in \arg\min_{y \in Y} w(x, y)\right\} \tag{19}$$

is monotone nondecreasing in x.

Proof of Proposition 1. By Property 4 we have that $g_t(s, z)$ is submodular for all t. By Property 7 we have that $\sum_{j=0}^{\infty} p_{t+1}(j|s, z)V_{t+1}(j)$ is submodular for all t. Thus w_t is submodular for all t, as a sum of submodular functions. Using Lemma 3.5 we conclude that there exists an optimal decision rule, say $z_t^*(s)$, that is nondecreasing in s for all t. □

Proposition 1 gives us the result that services can be performed based on a time varying control limit policy. Although intuitively obvious the result is important in classifying our problem in the class of problems with monotone optimal strategies.

4. Solution strategies

The optimality equations of the model we define in Section 2 are given by

$$V_t(S_t) = \min_{z_t} \{g_t(S_t, z_t) + \alpha E[V_{t+1}(S_{t+1})|S_t]\} \tag{20}$$

for $t = 0, \ldots, T - 1$. For the problem that we consider, which exhibits a scalar state variable, the optimality equations can be solved directly using a backward dynamic programming algorithm (see, for example, Puterman, 1994). That is, given $V_{t+1}(S_{t+1})$, we can find $V_t(S_t)$ by solving Eq. (20) for each possible state S_t. The problem with this approach is that it does not generalize to more complex problems where S_t is a vector. For this reason, this section proposes an adaptive dynamic programming algorithm. Our goal is to introduce an algorithm which is scalable to more complex problems, and study the properties of this algorithm in the context of a scalar problem which can be solved to optimality.

Our presentation begins by introducing the concept of an *incomplete state variable* which leads to a different dynamic programming recursion. Then we provide a formal description of the basic forward dynamic programming algorithm and finally we introduce our monotone adaptive dynamic programming algorithm as a variation of the basic algorithm.

4.1. The incomplete state variable

Although considerable attention is given to the complexity introduced when the state variable S_t is a vector (the so-called "curse of dimensionality") we have found that, in real applications, the expectation in Eq. (20) can be equally problematic. The standard approach is to replace the expectation with an approximation based on a subset of the full sample space, but this can still be computationally difficult for larger problems.

We want to consider an approximation using Monte Carlo sampling. For a sample realization $\omega = (a_1, \ldots, a_T)$ we propose the following approximation:

$$\tilde{V}_t(s_t) = \min_{z_t} \left\{ g_t(s_t, z_t) + \alpha \hat{V}_{t+1}(s_{t+1}) \right\}.$$

(Note that we refer to our approximation as $\hat{V}$; $\tilde{V}$ is a function that serves as a placeholder, not to be confused with $\hat{V}$.) However, $s_{t+1} = (s_t - z_t K)^+ + a_{t+1}$ is a function of a_{t+1} and thus when calculating the suboptimal z_t in the above minimization we are using future information from time $t+1$. To correct this problem we revisit the problem definition and introduce what we call the incomplete state variable.

The information process of the model as defined in Section 2 is as follows:

$$\{S_0, z_0, A_1, S_1, z_1, A_2, S_2, z_2, \ldots, A_T, S_T\}.$$

We want to measure the state variable at time t before the arrivals A_t. We call this new state variable the incomplete state variable and we denote it by S_t^-. Thus the information process becomes

$$\{S_0^-, A_0, z_0, S_1^-, A_1, z_1, S_2^-, A_2, z_2, \ldots, S_{T-1}^-, A_{T-1}, z_{T-1}, S_T^-\},$$

where $S_0^- = S_0 - A_0$ and $S_T^- = S_T - A_T$. Note that S_t contains the same amount of information as S_t^- and A_t together; in fact we have $S_t = S_t^- + A_t$. We substitute this into (20)

$$V_t(S_t^- + A_t) = \min_{z_t} \{ g_t(S_t^- + A_t, z_t) + \alpha E[V_{t+1}(S_{t+1}^- + A_{t+1}) | S_t^- + A_t] \}. \tag{21}$$

The system dynamics also change to

$$S_{t+1}^- = [S_t^- + A_t - K z_t]^+. \tag{22}$$

Now we take expectations of both sides of (21) conditioned on S_t^-:

$$E[V_t(S_t^- + A_t)|S_t^-] = E\left[\min_{z_t} \{ g_t(S_t^- + A_t, z_t) + \alpha E[V_{t+1}(S_{t+1}^- + A_{t+1}) | S_t^- + A_t] \} | S_t^- \right].$$

Since the expectation of $V_{t+1}(S_{t+1}^- + A_{t+1})$ conditioned on $S_t^- + A_t$ is also inside the minimization over z_t, we calculate the expectation of $V_{t+1}(S_{t+1}^- + A_{t+1})$ assuming that we know $S_t^- + A_t$ and z_t. The information contained in $S_t^- + A_t$ and z_t is the same as the information in S_{t+1}^-. Thus we can replace the conditional with S_{t+1}^-:

$$E[V_t(S_t^- + A_t)|S_t^-] = E\left[\min_{z_t} \{ g_t(S_t^- + A_t, z_t) + \alpha E[V_{t+1}(S_{t+1}^- + A_{t+1}) | S_{t+1}^-] \} | S_t^- \right]. \tag{23}$$

Now let us define the following functions:

$$V_t^-(s) \equiv E[V_t(S_t^- + A_t) | S_t^- = s], \tag{24}$$

$$g_t^-(s_t^-, a_t, z_t) \equiv g_t(s_t^- + a_t, z_t) = c z_t + h(s_t^- + a_t - K z_t)^+. \tag{25}$$

Then (23) becomes

$$V_t^-(S_t^-) = E\left[\min_{z_t} \{ g_t^-(S_t^-, A_t, z_t) + \alpha V_{t+1}^-(S_{t+1}^-) \} | S_t^- \right] \tag{26}$$

for all $t = 0, 1, \ldots, T-1$. These are the optimality equations corresponding to the incomplete state variable. Note that our new value function V^- is also nondecreasing since it is the conditional expectation of a nondecreasing function. The above expectation is taken over the random variable A_t. Thus if we considered an approximation using Monte Carlo sampling the decision variable z_t could be determined using only the sample realization of arrivals at time t, a_t.

4.2. The basic algorithm

In this section we provide a formal statement of the basic forward dynamic programming algorithm. In Section 4.3 we proceed to introduce our monotone adaptive dynamic programming algorithm as a variation of the algorithm described here.

We start with the new optimality equation (26). For a sample realization $A_t(\omega) = a_t$, we propose the following approximation:

$$\hat{V}_t(s_t^-) = \min_{z_t} \left\{ g_t^-(s_t^-, a_t, z_t) + \alpha \hat{V}_{t+1}(s_{t+1}^-) \right\}. \tag{27}$$

The forward dynamic programming algorithm simulates forward through time at each iteration using a sample realization of the arrival process and the current estimates of the value functions, thus creating a state and decision trajectory: At time t, state $s_t^{-,k}$, using a sample realization a_t^k and the current estimate of the value function $\hat{V}_{t+1}^{k-1}$, the suboptimal decision z_t^k is calculated by solving

$$z_t^k = \arg\min_z \left\{ g_t^-\left(s_t^{-,k}, a_t^k, z_t^k\right) + \alpha \hat{V}_{t+1}^{k-1}\left(\left[s_t^{-,k} + a_t^k - zK \right]^+ \right) \right\}. \tag{28}$$

Then the state at the next time period is calculated using the transfer function (22). This continues forward through time until the end of the horizon.

The backward pass is performed to update the value function estimates using the decision and state trajectories: $\{z_t^k\}_{t=0}^{T-1}$, $\{s_t^{-,k}\}_{t=0}^{T}$. We update the value function estimate $\hat{V}_t^k$ at the state $s_t^{-,k}$ using

$$\hat{V}_t^{k+1}\left(s_t^{-,k}\right) = (1-\alpha^k)\hat{V}_t^k\left(s_t^{-,k}\right) + \alpha^k\left(g_t^-\left(s_t^{-,k}, a_t^k, z_t^k\right) + \alpha \hat{V}_{t+1}^{k+1}\left(s_{t+1}^{-,k}\right) \right).$$

The above step is performed backward in time for $t = T-1, \dots, 0$.

The following description of the algorithm executes the algorithm for N iterations and assumes that the state variable does not exceed $S^{\max}$ which is chosen to be a small multiple of K.

Basic forward dynamic programming algorithm

Step 1. Given s_0: Set $\hat{V}_t^k(s) = 0$ for all t, k and $s \in \{0, \dots, S^{\max}\}$. Set $s_0^{-,k} = s_0$ for all k. Set $k = 1$.
Step 2. Choose a random outcome $\omega^k = (a_0^k, a_1^k, \dots, a_{T-1}^k)$.
Step 3. For $t = 0$ to $t = T-1$ perform:

$$z_t^k := \arg\min_{z\in\{0,1\}} \left\{ g_t^-(s_t^{-,k}, a_t^k, z) + \alpha \hat{V}_{t+1}^k([s_t^{-,k} + a_t^k - Kz]^+) \right\} \tag{29}$$

and

$$s_{t+1}^{-,k} := [s_t^{-,k} + a_t^k - Kz_t^k]^+. \tag{30}$$

Step 4. For $t = T-1$ to $t = 0$ perform:

Using a stepsize α^k, update the estimate at state $s_t^{-,k}$:

$$\hat{V}_t^{k+1}(s) := \begin{cases} 1 - \alpha^k \hat{V}_t^k(s) + \alpha^k \left\{ g_t^-(s_t^{-,k}, a_t^k, z_t^k) + \alpha \hat{V}_{t+1}^{k+1}(s_{t+1}^{-,k}) \right\} & \text{if } s = s_t^{-,k}, \\ \hat{V}_t^k(s) & \text{otherwise.} \end{cases}$$

Step 5. If $k < N$, then set $k = k+1$ and go to Step 2; else stop here.

4.3. Adaptive dynamic programming using structure

The monotone adaptive dynamic programming algorithm is simply a variation of the basic forward dynamic programming algorithm. The difference occurs in the way we update the value function estimates. Our algorithm consists of breaking step 4 into two steps. Our revised version of step 4 is given by:

Step 4. For $t = T - 1$ to $t = 0$ perform:

(a) Update the estimate at state $s_t^{-,k}$:

$$\hat{V}_t^{k+1}(s) := \begin{cases} 1 - \alpha^k \hat{V}_t^k(s) + \alpha^k \left\{ g_t^-(s_t^{-,k}, a_t^k, z_t^k) + \alpha \hat{V}_{t+1}^{k+1}(s_{t+1}^{-,k}) \right\} & \text{if } s = s_t^{-,k}, \\ \hat{V}_t^k(s) & \text{otherwise.} \end{cases}$$

(b) Enforce monotonicity:
1. For all $i = s_t^{-,k} - 1, \ldots, 0$ and while $\hat{V}_t^{k+1}(i) > \hat{V}_t^{k+1}(s_t^{-,k})$: set $\hat{V}_t^{k+1}(i) := \hat{V}_t^{k+1}(s_t^{-,k})$.
2. For all $i = s_t^{-,k} + 1, \ldots, S^{\max}$ and while $\hat{V}_t^{k+1}(i) < \hat{V}_t^{k+1}(s_t^{-,k})$: set $\hat{V}_t^{k+1}(i) := \hat{V}_t^{k+1}(s_t^{-,k})$.

Step 4(b) is to preserve monotonicity of the value function estimates. At time t iteration k the state whose value has been updated in Step 4(a) of the basic algorithm is $s_t^{-,k}$ and the new value is $\hat{V}_t^{k+1}(s_t^{-,k})$. We then check the value function estimates at time t of neighboring states of $s_t^{-,k}$ and if they violate monotonicity we set them to $\hat{V}_t^k(s_t^{-,k})$.

Step 4(b) is what makes this algorithm a functional approximation algorithm. This is because it uses information about the structure of the function (in this case monotonicity) to improve the estimates of neighboring cells. There is no need to cycle through the whole state space to check for violations of monotonicity. Once the value function estimate at a specific state is updated we only need to find and update states directly above and directly below that state until we find one whose value function estimate does not violate monotonicity.

Fig. 2 shows the process of updating neighboring states when their value violates monotonicity. The first plot shows the monotone estimate at iteration k of the value function $\hat{V}_t^k$, the new observation in iteration $k+1$ of the value function in state $s_t^{-,k} = 3$ as well as the new smoothed estimate of the value function in state 3, $\hat{V}_t^{k+1}(3)$. The second plot presents Step 4(a) of the basic algorithm where the new estimate of the

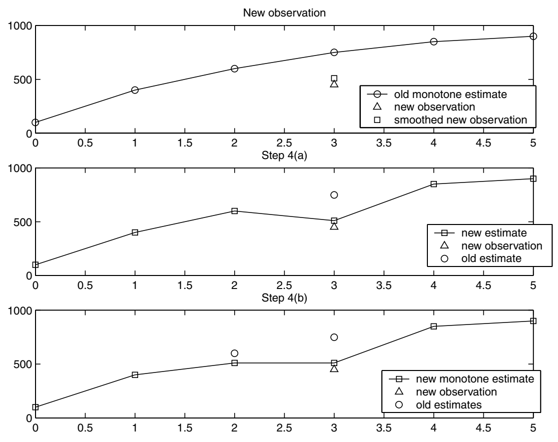

Fig. 2. Step 4 of the monotone forward dynamic programming algorithm. The second plot illustrates Step 4(a) and the third plot illustrates Step 4(b).

value function is shown at iteration $k+1$ by incorporating the new value at state 3. At this point we look for neighboring states of state 3 to see if monotonicity has been violated. We note that monotonicity has been violated by state 2 and so we set the estimate of state 2 equal to $\hat{V}_t^{k+1}(3)$, as shown in the third plot.

The algorithm with no updates of neighboring cells has been used and studied by Bertsekas and Tsitsiklis. This algorithm has proved to be convergent under the assumption that each state is visited infinitely many times (Bertsekas and Tsitsiklis, 1996). However, in many problems including the batch service problem there is no assurance that all states will be visited infinitely often. As the process converges to the optimal or sub-optimal policies certain states determined by these policies are visited far more frequently than others. There is no guarantee of convergence of this algorithm in the problem we are studying.

5. Experiments

In this section we test the performance of our algorithm. We take advantage of the fact that the homogeneous customer batch service problem can be solved optimally to evaluate the performance of the algorithms. We design the experiments in Section 5.1, construct the format of the algorithms in Section 5.2, and finally in Section 5.3 we discuss and compare the performance of different approximation methods.

5.1. Experimental design

This section describes the experiments that test the performance of the adaptive dynamic programming algorithm. We design the parameter data sets and describe the competing strategies.

5.1.1. Data sets

In order to test our algorithm, we create data sets by varying the arrival sets and some other model parameters.

We assume that the arrivals at time t, A_t, follow a Poisson distribution with known mean. We generate ten arrival scenarios for the means: $\{\mu_t^i\}_{t=0}^{T-1}$ for $i = 1, \ldots, 10$, where μ_t^i is the mean of the arrivals at time t in scenario i. Four of these arrival scenarios are periodic and the remaining six are aperiodic generated uniformly from $\{0, 1, 2, 3\}$ or $\{0, 1, \ldots, 6\}$. The periodic arrival sets go through periods of zero arrivals, which are what we call 'dead periods'. This is a natural phenomenon in actual arrival scenarios for some applications. Thus, we also introduce 'dead periods' in the aperiodic uniformly generated arrival sets. Fig. 3 shows the three types of arrival scenarios: periodic, uniformly generated aperiodic without dead periods and with dead periods. In general, we try to design our arrival sets so that they vary with respect to periodicity, length of dead periods, mean number of arrivals and maximum number of arrivals.

We fix the following parameters throughout the data sets: $T = 200$, $\Delta s = 1$, $c = 200$, $\alpha = 0.99$. However, we vary the waiting cost per customer per time period ($h = 2, 3, 5, 8, 10$) and the service capacity ($K = 8, 10, 15$). By varying these two parameters we create 10 situations where the average waiting cost per customer is greater than, approximately equal to, or less than the service cost per customer (c/K). Note that we approximate the average number of time periods that a customer waits at the station by $(K/2)/\lambda$, assuming that on average the size of service batch is half the service capacity. If we let λ be the average number of arrivals per time period then the average waiting cost per customer is $h(K/2)/\lambda$.

Thus we have a total of 100 data sets (10×10). We run the experiments for all 100 data sets but for convenience in reporting results we group them into six groups according to their different features. In Table 1 we see that the data sets are grouped according to periodicity and according to the relationship between average waiting cost per customer and service cost per customer. We report the performance results of these six groups.

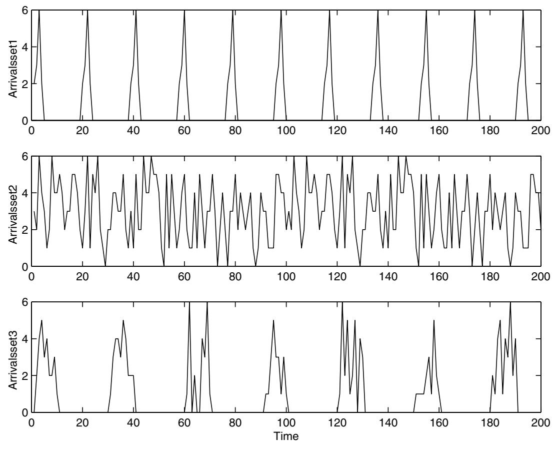

Fig. 3. Arrival set 1 is a periodic arrival set with dead periods. Arrival sets 2 and 3 are aperiodic uniformly generated without dead periods and with dead periods.

Table 1
Fractional cost produced by each algorithm relative to optimal

	Method	Monotone structure			No structure		
	Iterations	500	1000	2000	500	1000	2000
Data sets							
$h > c/K$	Periodic	0.0301	0.0190	0.0122	0.1966	0.1110	0.0621
	Aperiodic	0.0275	0.0188	0.0137	0.1468	0.0955	0.0655
$h \simeq c/K$	Periodic	0.0295	0.0145	0.0105	0.2892	0.1829	0.1181
	Aperiodic	0.0283	0.0205	0.0134	0.3318	0.1857	0.1377
$h < c/K$	Periodic	0.0453	0.0261	0.0203	0.4385	0.3050	0.2378
	Aperiodic	0.0443	0.0267	0.0184	0.4540	0.3164	0.2301
	Average	0.0342	0.0209	0.0147	0.3095	0.1994	0.1419

5.1.2. Competing strategies

Since our discussion in this paper is to emphasize the importance of using structural information we consider the adaptive dynamic programming algorithm with no structure to be our competition and we compare our results to this method. We compare both methods to the optimal solution.

5.2. Calibrating the algorithm

In this section we find the number of iterations to run the algorithms and we decide on a stepsize rule.

5.2.1. Number of iterations

Our stopping rules are based on determining the required number of iterations to run the algorithm, in order to achieve the appropriate performance levels for each approximation method. In general, experiments have two types of iterations: training and testing.

Training iterations are performed in order to estimate the value approximations. At each training iteration we randomly choose an initial state from the set $\{0, 1, \ldots, K-1\}$. This allows the algorithm to visit more states. To determine the appropriate number of training iterations, we ran a series of tests. We concluded that the performance of our monotone algorithm stabilizes after 1000 iterations and for comparison we choose to run it for 500, 1000 and 2000 iterations.

Testing iterations are performed to test the performance of the approximation method over different samples of arrivals. We perform the testing iterations for different values of the initial state ($s_0 = 0, \dots, K-1$) and we average costs accordingly. All experiments are performed for 100 testing iterations and the values are averaged over iterations.

5.2.2. Stepsize rule

We use a stepsize rule of the form $\alpha/(\beta + k)$, where k is the number of iterations, for smoothing the updated value function with the estimate of the last iteration. We tested different values of α and β on several data sets and found that a stepsize of $4/(5 + f_i(k))$ worked best, where $f_i(k)$ is the number of times that state i, whose value is being smoothed, has been visited up to iteration k. $f_i(k)$ does not count the updates to state i from neighboring states. Thus, the states that have not been visited many times will have high stepsizes and the ones visited many times will have low stepsizes. This stepsize rule weighs new information on the value of a state according to the amount of information received in past iterations regarding that state.

5.3. Experimental results

In this section we report on the experimental results and discuss the performance of each approximation method. The results of our experiments for each method and each data group are listed in Table 1. The entries in the table are the total costs expressed relative to the optimal solution, averaged over the data sets of each group.

5.3.1. Value approximation performance

The monotone functional adaptive dynamic programming method performs significantly better than the algorithm with no structure. On average our algorithm has percentage errors from optimal that are ten times smaller than the percentage errors of the basic forward dynamic programming algorithm. The satisfying performance of our approximation method is consistent throughout the data sets. This is not true for the algorithm with no structure, whose percentage errors fluctuate highly between data sets.

Our algorithm reaches satisfactory levels of performance at a smaller number of iterations than the algorithm with no structure. Even at 500 iterations our algorithm performs within 5% of optimal and at 2000 iterations it performs consistently within 2% of optimal. It seems that the algorithm with no structure has a slow rate of convergence. Even at 2000 iterations the percentage errors for some data sets are extremely high.

Another general observation is that the algorithm with no structure has the highest errors when the average waiting cost per customer is lower than the service cost per customer. The percentage errors vary from 6% to 23% when moving from high waiting cost data sets to low waiting cost. This suggests that this approximation method incurs a lower total waiting cost than the optimal, which means that its service decision rules have the server busy longer than the optimal service rules. These phenomena are not apparent in the case of our monotone algorithm whose performance is not affected by the nature of the data sets.

5.3.2. Conclusions

The monotone adaptive dynamic programming algorithm estimates the optimal decision rules fairly well with small percentage errors. The algorithm is robust in its consistency to perform well throughout different data sets and it converges faster than its competition. At the same number of iterations our monotone algorithm is ten times closer to optimal than the basic forward dynamic programming algorithm. It is very encouraging that our algorithm performs well with uncertainty because in practice arrivals are usually stochastic.

Finally, the introduction of structure in the algorithm is definitely an improvement since it allows the approximation technique to make use of important information on the shape of the value function when approximating the optimal decision policies. The use of structure in forward dynamic programming algorithms is very promising and could be extended in harder instances of the problem were the state variable is multidimensional and optimal solutions are intractable.

Acknowledgements

This research was supported in part by grant AFOSR-F49620-93-1-0098 from the Air Force Office of Scientific Research, and grant DMI00-85368 from the National Science Foundation. We would also like to acknowledge the comments and careful proofreading of a conscientious reviewer.

References

Bailey, N., 1954. On queueing processes with bulk service. Journal of the Royal Statistical Society B 16, 80–87.

Barnett, A., 1973. On operating a shuttle service. Networks 3, 305–314.

Bertazzi, L., Speranza, M.G., 1999a. Inventory control on sequences of links with given transportation frequencies. International Journal of Production Economics 59, 261–270.

Bertazzi, L., Speranza, M.G., 1999b. Minimizing logistic costs in multistage supply chains. Naval Research Logistics 46, 399–417.

Bertazzi, L., Bertsekas, D., Speranza, M.G., 2000. Optimal and neuro-dynamic programming solutions for a stochastic inventory transportation problem, Unpublished Technical Report, Universita Degli Studi Di Brescia.

Bertsekas, D., Tsitsiklis, J., 1996. Neuro-dynamic Programming. Athena Scientific, Belmont, MA.

Chaudhry, M., Templeton, J., 1983. A First Course in Bulk Queues. Wiley, New York.

Deb, R., 1978a. Optimal control of batch service queues with switching costs. Advances in Applied Probability 8, 177–194.

Deb, R., 1978b. Optimal dispatching of a finite capacity shuttle. Management Science 24, 1362–1372.

Deb, R., Schmidt, C., 1987. Optimal average cost policies for the two-terminal shuttle. Management Science 33, 662–669.

Deb, R., Serfozo, R., 1973. Optimal control of batch service queues. Advances in Applied Probability 5, 340–361.

Ignall, E., Kolesar, P., 1972. Operating characteristics of a simple shuttle under local dispatching rules. Operations Research 20, 1077–1088.

Ignall, E., Kolesar, P., 1974. Operating characteristics of an infinite capacity shuttle: Control at a single terminal. Operations Research 22, 1003–1024.

Kosten, L., 1967. The custodian problem. In: Cruon, R. (Ed.), Queueing Theory, Recent Developments and Applications. English University Press, London, pp. 65–70.

Kosten, L., 1973. Stochastic Theory of Service Systems. Pergamon Press, New York.

Medhi, J., 1984. Recent Developments in Bulk Queueing Models. Wiley Eastern, New Delhi.

Neuts, M., 1967. A general class of bulk queues with Poisson input. Annals of Mathematical Statistics 38, 759–770.

Powell, W.B., 1985. Analysis of vehicle holding and cancellation strategies in bulk arrival, bulk service queues. Transportation Science 19 (4), 352–377.

Powell, W.B., Humblet, P., 1986. The bulk service queue with a general control strategy: Theoretical analysis and a new computational procedure. Operations Research 34 (2), 267–275.

Puterman, M.L., 1994. Markov Decision Processes. Wiley, New York.

Speranza, M., Ukovich, W., 1994. Minimizing transportation and inventory costs for several products on a single link. Operations Research 42, 879–894.

Speranza, M., Ukovich, W., 1996. An algorithm for optimal shipments with given frequencies. Naval Research Logistics 43, 655–671.

Sutton, R., Barto, A., 1998. Reinforcement Learning. MIT Press, Cambridge, MA.

Weiss, H., 1981. Further results on an infinite capacity shuttle with control at a single terminal. Operations Research 29, 1212–1217.

Weiss, H., Pliska, S., 1976. Optimal control of some Markov processes with applications to batch queueing and continuous review inventory systems. Discussion paper no. 214, The Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, IL.