Probability in the Engineering and Informational Sciences, **24**, 2010, 27–45.
doi:10.1017/S026996480999012X

APPROXIMATE DYNAMIC PROGRAMMING TECHNIQUES FOR THE CONTROL OF TIME-VARYING QUEUING SYSTEMS APPLIED TO CALL CENTERS WITH ABANDONMENTS AND RETRIALS

DENNIS ROUBOS AND SANDJAI BHULAI
VU University Amsterdam
Faculty of Sciences
1081 HV Amsterdam, The Netherlands
E-mail: {droubos, sbhulai}@few.vu.nl

In this article we develop techniques for applying Approximate Dynamic Programming (ADP) to the control of time-varying queuing systems. First, we show that the classical state space representation in queuing systems leads to approximations that can be significantly improved by increasing the dimensionality of the state space by state disaggregation. Second, we deal with time-varying parameters by adding them to the state space with an ADP parameterization. We demonstrate these techniques for the optimal admission control in a retrial queue with abandonments and time-varying parameters. The numerical experiments show that our techniques have near to optimal performance.

1. INTRODUCTION

Many operational systems in the service and production industry are described by queuing models that are large and complex. The applicability of such models to derive (near) optimal control strategies is therefore very limited, since (1) the model is of high dimension so that standard algorithms (e.g., dynamic programming) are not computationally tractable and (2) the parameters of the model change over time so that steady-state solution techniques are inappropriate. Both difficulties prohibit the use of queuing theory in practice and need to be resolved.

 0269-9648/10 $25.00

The first difficulty is very much related to the model description. In practice, the control problem is usually described by a Markov decision model with the fewest number of variables so that the model remains compact and is of smallest dimension. Unfortunately, for many problems, the final description is still of very high dimension and does not elevate the intractability (both analytically and numerically). Therefore, one needs to take the refuge to approximate techniques that have proven to be successfully applied in specific application areas, such as state space aggregation [15,17], state decomposition/factoring [12], approximation via Linear Programming [3–5], value function approximation [2,16], neural networks [20], reinforcement learning [18,19], and Approximate Dynamic Programming (ADP) [1,13].

When a suitable approximation technique has been chosen, one still needs to deal with the second difficulty of the model (i.e., the time-varying nature of the parameters). One method of accommodating time-varying parameters is to solve the Chapman–Kolmogorov forward equations (see [11,21]). This is done by continuously approximating the varying parameters with small, discrete intervals and use the so-called randomization method (see [6]) to explicitly calculate the change in system occupancy from one small interval to the next. Another natural means of accommodating changes in the arrival rate is to reduce the interval over which a stationary measure is applied. This is the essence of the pointwise stationary approximation (PSA) of [7]. However, the PSA does not explicitly consider nonstationary behavior that might be induced by abrupt changes in the arrival rate, and it appears to perform less well in these cases. In [10] the accuracy and computational requirements of a number of approaches, including the exact calculation of the Chapman–Kolmogorov forward equations and the method of randomization have been evaluated. The results show that the method of randomization generally produces results that are close to exact, however the computation is quite burdensome. The PSA method is quicker but more approximate. The computational complexity increases and accuracy diminishes when the approximation methods to control the system are combined with methods to deal with time-varying parameters, making the problem numerically intractable again.

In this article we develop techniques for applying ADP to the control of time-varying queuing systems. First, we show that the classical state space representation in queuing systems leads to an inaccurate approximation. Instead of reducing the state space by aggregation, increasing the dimensionality of the state space by state disaggregation is preferable. Second, we deal with time-varying parameters by adding them to the state space with an ADP parameterization. By using a method to track the time-varying parameters (e.g., stochastic approximation), one can generate control strategies on-line. We demonstrate these techniques for the optimal admission control in a retrial queue with abandonments and time-varying parameters. The numerical experiments show that our techniques have near to optimal performance with very little computational effort.

The sequel of the article is organized as follows. In Section 2 we give a formal description of the value iteration algorithm in combination with an approximation structure of the relative value function, and we describe our approach to handle time-varying systems. By means of an example we show in Section 3 that using a larger

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

representation of the system leads to a better approximation of the relative value function. Section 4 is devoted to an example in which we illustrate our method to handle time-varying systems and we show that the gain in performance is significant compared to other ways to obtain policies. Finally, we conclude with Section 5.

2. FORMULATION

In this section we formulate the Markov decision problem (MDP) that will be used for the control of queuing systems. To this end, let the tuple $(\mathcal{X}, \mathcal{A}_x, p(x,a,y), c(x,a))$ define the MDP, where $\mathcal{X}$ represents the state space, $\mathcal{A}_x$ represents the action space if the system is in state $x \in \mathcal{X}$, $p(x,a,y)$ is the transition probability of going from state x to state y by choosing action $a \in \mathcal{A}_x$, and $c(x,a)$ is the direct cost function associated with state x and action a.

A deterministic Markov policy $\pi : \mathcal{X} \to \mathcal{A}_x$ is a function that maps states to actions (i.e., $\pi(x) = a$ for an action $a \in \mathcal{A}_x$). The policy π describes which actions to take in each state of the state space. The objective is to find a policy π in the class of deterministic Markovian policies such that the long-term average cost g is minimized. For a fixed policy π, the average cost g can be determined by solving the Poisson equations given by

$$g + V(x) = c(x, \pi(x)) + \sum_{y \in \mathcal{X}} p(x, \pi(x), y) V(y),$$

for all $x \in \mathcal{X}$, where $V(x)$ is the relative value function, which has the interpretation of the asymptotic difference in total cost that results from starting the process in state x instead of some reference state.

Note that the MDP that we have introduced in this section is for discrete-time processes. However, continuous-time queuing models can be cast as a discrete-time MDP through uniformizing the system (see [14, Sect. 11.5]). Moreover, results on existence and uniqueness of the average cost g and the relative value function V heavily depend on the state and action spaces $\mathcal{X}$ and $\mathcal{A}_x$. For notational clarity we assume that the state space is denumerable and that the action space is finite with the usual ergodicity assumptions (see [8,9]).

2.1. Approximate Value Iteration

The Poisson equations are hard to solve analytically in practice. Alternatively, the equations can also be solved by a recursive procedure known as value iteration. The value iteration algorithm starts with an initial value for all states (e.g., $V_0(x) = 0$ for all $x \in \mathcal{X}$). The method then iterates by calculating V_{n+1} for $n = 0, 1, \ldots$ through

$$V_{n+1}(x) = c(x, \pi(x)) + \sum_{y \in \mathcal{X}} p(x, \pi(x), y) V_n(y), \tag{1}$$

for all $x \in \mathcal{X}$. It is known that this method converges and an ϵ-optimal approximation can be obtained when span $(V_{n+1}(x) - V_n(x)) \leq \epsilon$.

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

The value iteration algorithm has a memory complexity that is of the order $|\mathcal{X}|$. For many practical problems, the size of the state space can be too large, making the value iteration algorithm intractable. Therefore, the relative value function $V(x)$ can be approximated by taking an approximation architecture $\tilde{V}(x,r) = \sum_{i=1}^{R} r_i \phi_i(x)$, with r a parameter vector and $\phi_i(x)$ a known set of basis functions that is used for the approximation (see [1,13]). In doing so, we only have to determine values for r rather than V for each state.

The approximation structure can also be used in the value iteration algorithm. In this case, each iteration step updates the parameter vector r. To this end, let $\tilde{V}_n(x,r)$ be the nth approximation of the relative value function. Based on the approximation structure $\tilde{V}_n(x,r)$, the next value for each state, denoted by $\hat{V}_{n+1}(x)$, is similarly computed as in (1). After the calculation of the next values, a (weighted) least squares problem is solved to tune r such that it minimizes the distances between $\tilde{V}_{n+1}(x,r)$ and $\hat{V}_{n+1}(x)$. Since we use a functional architecture (e.g., a polynomial), we can take a smaller set of states $\mathcal{Y} \subset \mathcal{X}$ that represents the set of representative states (see [16]). The steps of this approach are thus given as follows:

1. Calculate $\hat{V}_{n+1}(x)$ by

$$\hat{V}_{n+1}(x) = c(x,\pi(x)) + \sum_{y \in \mathcal{X}} p(x,\pi(x),y)\tilde{V}_n(y,r), \tag{2}$$

   for all $x \in \mathcal{Y}$.

2. Solve for r in the problem

$$\min_r \sum_{x \in \mathcal{Y}} w_x (\tilde{V}_{n+1}(x,r) - \hat{V}_{n+1}(x))^2.$$

   Let $n = |\mathcal{Y}|$; then the solution for this weighted least squares problem is given by

$$r = (A^T W A)^{-1} A^T W \hat{V},$$

   with the matrix A and the weight matrix W given by

$$A = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_R(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_R(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_n) & \phi_2(x_n) & \dots & \phi_R(x_n) \end{pmatrix},$$

   and

$$W = \begin{pmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & w_n \end{pmatrix},$$

   respectively.

3. Repeat step 1 with the new vector r until convergence is achieved.

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

The value iteration algorithm with the approximation architecture is called approximate value iteration. Normally, the average cost g is obtained by calculating $V_{n+1}(x) - V_n(x)$ after convergence. In our case, we have $g \approx \tilde{V}_{n+1}(x, r) - \tilde{V}_n(x, r)$. Therefore, there is no need to estimate g beforehand, in contrast to solving the Poisson equations directly.

Recall that we started out by fixing a policy π. However, for MDPs we are interested in finding the optimal policy. We therefore modify the approximate value iteration method such that it generates a sequence of policies, based on the relative value function, that converges to an optimal policy. This gives rise to the sequence $(\tilde{V}^{\pi^0}, \pi^1, \tilde{V}^{\pi^1}, \pi^2, \ldots)$ in which π^i is calculated using $\tilde{V}^{\pi^{i-1}}$:

$$\pi^i(x) = \arg\min_{a \in \mathcal{A}_x} \left[c(x, a) + \sum_{y \in \mathcal{X}} p(x, a, y) \tilde{V}^{\pi^{i-1}}(y, r) \right],$$

for all $x \in \mathcal{X}$. For the calculation of $\tilde{V}^{\pi^i}$ we use approximate value iteration as follows:

$$\tilde{V}^{\pi^i}_{n+1}(x, r) = c(x, \pi^i(x)) + \sum_{y \in \mathcal{X}} p(x, \pi^i(x), y) \tilde{V}^{\pi^{i-1}}_n(y, r).$$

The previous two steps can be combined into a single step that replaces (2) in the approximate value iteration algorithm. By doing so, the algorithm also increases significantly in speed. The resulting step is then given by

$$\hat{V}_{n+1}(x) = \min_{a \in \mathcal{A}_x} \left[c(x, a) + \sum_{y \in \mathcal{X}} p(x, a, y) \tilde{V}_n(y, r) \right].$$

2.2. State Disaggregation

The quality of the policy that one obtains by using the approximate value iteration algorithm is largely dependent on the approximation of the relative value function. This function plays a crucial role in the decision-making process. In its turn, the quality of the approximation to V depends on the parameter vector r and the choice of the basis functions $\phi_i(x)$. Hence, given a set of basis functions, the fit can only be improved by adjusting r such that every region of the state space is well approximated.

In many problem definitions, the model is chosen as small as possible; that is, it is described with the fewest number of state variables. For example, in many queuing systems, the state is chosen to be the number of customers in the queue plus in service instead of having two variables that denote the number of customers in the queue and in service, respectively. Given the above interpretation, using the same information with more state variables can result in a better approximation of the value function, since there is more freedom to fit a functional form to specific parts of the state space. This helps to improve the quality of the resulting policy. This is illustrated in Section 3.

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

2.3. Time-Varying Parameters

The approximate value iteration algorithm results in an approximation for the relative value function for fixed exogenous input parameters. However, in practice many of these input parameters change over time (e.g., think of arrival patterns of customers that vary over the day). To deal with such situations, the approximation architecture can be extended to include the time-varying parameters as well. The approximate value iteration algorithm can then be used to approximate the relative value function, such that for each different configuration of the time-varying input parameters, one gets a different value of r. Consequently, when the values of the input parameters can be tracked (e.g., by a stochastic approximation method), then the decision making for time-varying systems can be done on-line with little computational effort.

To formalize the idea, let γ be a configuration for the set of input parameters. Based on the γ, one can obtain an approximation $\tilde{V}_\gamma(x,r)$. Now, the vector r can be extended by parameterizing the input parameters γ as well so that we get $r_{(\gamma)}$. By choosing an appropriate model for this and calculating $\tilde{V}_\gamma(x,r)$ for different values of γ, we can fit the approximated relative value function for the relevant range of values of the input parameters.

The model that one has to take for $r(\gamma)$ is problem-specific. However, in Section 4 we will show that a rather simple regression model is quite sufficient, since the parameter vector r is continuous for small changes in γ. Therefore, we can take a polynomial function of low degree to describe the relationship between the input parameters γ and the vector r.

3. EXAMPLE: THE $M/M/C$ QUEUE

In this section we illustrate the idea of state disaggregation to obtain better approximations to the relative value function and thus better control policies. For this purpose, we will study the $M/M/c$ queue. In Section 4 we will extend this problem with time-varying parameters, retrials, and abandonments so that it can serve as a realistic model for call centers.

The $M/M/c$ queuing system is described as follows. Consider a service facility with c servers at which customers arrive according to a Poisson process with rate λ. Upon arrival of a customer, the customer is taken into service when a server is available, otherwise the customer waits in a buffer of infinite size. When a server becomes idle, the longest-waiting customer is being served if present. Serving a customer takes an exponentially distributed amount of time with parameter μ. The system is subject to holding costs for each customer per time unit.

The state space of the system can be chosen in many ways. The most common way to describe this system is to take as state space $\mathcal{X}_1 = \mathbb{N}_0$, where a state $x \in \mathcal{X}_1$ represents the total number of customers in the system (i.e., in queue and in service). However, another way to model the system is by choosing as state space $\mathcal{X}_2 = \{0,\ldots,c\} \times \mathbb{N}_0$, where a state $(s,q) \in \mathcal{X}_2$ represents the system with s customers being served and q customers waiting in the buffer.

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

Since there are holding costs for each customer in the system per time unit, we have $c_1(x) = x$ and $c_2(s,q) = s + q$. For simplicity and without loss of generality, we assume that $\lambda + c\mu < 1$; we can always get this by scaling. After uniformization, we obtain the following update rule for the value iteration algorithm:

$$\hat{V}^1_{n+1}(x) = x + \lambda \tilde{V}^1_n(x+1) + \min\{x,c\}\mu \tilde{V}^1_n([x-1]^+) + (1 - \lambda - \min\{x,c\}\mu)\tilde{V}^1_n(x),$$

$$\hat{V}^2_{n+1}(s,q) = s + q + \lambda H_n(s+q+1) + s\mu H_n(s+q-1) + (1 - \lambda - s\mu)\tilde{V}^2_n(s,q),$$

with $H_n(y)$ defined by

$$H_n(y) = \hat{V}^2_n(\min\{[y]^+, c\}, [y-c]^+).$$

We take a second-order polynomial function as the approximation structure. For the first representation, we take it equal to $\tilde{V}^1(x,r) = r_0 + r_1 x + r_2 x^2$, and for the second representation, we have $\tilde{V}^2(s,q,r) = r_0 + r_1 s + r_2 s^2 + r_3 q + r_4 q^2 + r_5 sq$. Note that the last term sq is redundant, since $sq = cq$. To see this, note that $q = 0$ if $s \leq c$, and $s = c$ when $q > 0$. To compare our approximation to the real-value function of the $M/M/c$ queue, we define two distance measures. These are given by

$$d_1(V) = \max |V - \tilde{V}|,$$

$$d_2(V) = ||V - \tilde{V}||.$$

Table 1 shows the the real average cost g, the average costs $g(V^1)$ and $g(V^2)$ obtained by using the first and second representation, respectively, and the distance measures for different values of λ, μ, and c. We used ρ^x and $\rho^{(s+q)}$ as weight factors in the weighted least squares minimization for the first and second approach, respectively, where $\rho = \lambda/(c\mu)$. The results show that by using the disaggregated description of the state space, the quality of the approximation improves by a large factor, without significantly much additional computational effort.

TABLE 1. Results for the M/M/c Queue with $\mathcal{Y} = \{0, \ldots, 20\}$

λ	μ	c	g	$g(V^1)$	$d_1(V^1)$	$d_2(V^1)$	$g(V^2)$	$d_1(V^2)$	$d_2(V^2)$
4	2	8	2.00	2.00	5.810	1.297	2.00	0.043	0.024
10	8	5	1.25	1.28	3.379	1.253	1.28	0.011	0.006
8	2	16	4.00	4.01	0.329	0.038	4.01	0.038	0.013
5	1	10	5.04	5.08	9.734	1.893	5.05	0.249	0.124
3	2	3	1.74	1.70	12.426	3.318	1.74	0.023	0.009
10	4	5	2.63	2.68	6.442	1.617	2.66	0.037	0.017
15	5	4	4.53	4.37	2.240	0.553	4.56	0.095	0.033
3	2	2	3.43	3.33	1.021	0.518	3.43	0.053	0.016
9	3	4	4.53	4.36	3.702	0.917	4.55	0.124	0.045

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

4. EXAMPLE: THE $M/M/C$ RETRIAL QUEUE WITH ABANDONMENTS

In this section we illustrate the approximate value iteration method for time-varying parameters. Therefore, we extend the model from the previous section to a multiserver queue with abandonments and retrials. Furthermore, the arrival rate of customers to the system is time dependent and serves as our time-varying parameter. The model can be seen as a call center to which customers call for service.

The formal description of the model is as follows. Consider a single multiserver queue at which calls arrive. The calls arrive according to a nonhomogeneous Poisson process with rate λ_t, as depicted in Figure 1. There is a controller that, upon arrival of a call, can decide to block the call. In that case, a cost of B is incurred and the call is dropped. When the call is allowed into the system, it is assigned to an agent if one is idle. The call is then served and has a service duration that is exponentially distributed with parameter μ. If all agents are busy, the call is placed in the queue and starts its service when an agent becomes available.

We assume that the operating hours of the call center are divided into K intervals; in our examples, we will take $K = 4$ intervals of 6 h. In each interval k, the number of agents c_k is constant but may differ from the number of agents in other intervals. This is current practice due to schedules and rosters of call center agents. In our examples, we choose the number of agents c_k such that at least 80% of the calls has a waiting time in the queue of less than 20 s (i.e., $\mathbb{P}(W_Q > 20\text{s}) < 0.2$). This number can be

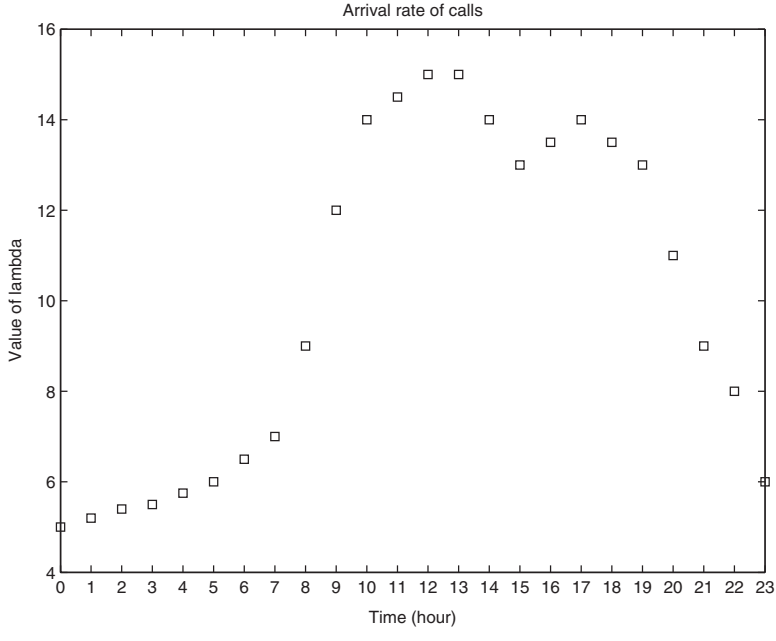

FIGURE 1. Call rate per hour.

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

found by using the Erlang-C formula (3), in which λ is equal to the average arrival rate in the corresponding interval, a equals the offered load (i.e., $a = \lambda/\mu$), and W_Q represents the stochastic variable for the waiting time in the queue.

$$\mathbb{P}(W_Q > t) = C(s,a)e^{-(s\mu-\lambda)t},$$

with

$$C(s,a) = \frac{a^s}{(s-1)!(s-a)}\left[\sum_{j=0}^{s-1}\frac{a^j}{j!} + \frac{a^s}{(s-1)!(s-a)}\right]^{-1}. \tag{3}$$

In case the number of agents decreases from one interval to the next, the agents that have to leave and are busy finish their service first. However, when the number of agents increases, the additional agents are available immediately at the beginning of the interval.

The customers have a finite patience that is exponentially distributed with parameter β. Hence, after their patience time has expired and the customer is still in the buffer, he abandons the queue. Once he has abandoned, there is a probability p that he will call back after an exponentially distributed time with parameter γ. We then say that the customer is in the retrial orbit. Otherwise, he will leave the system without having received service. This happens with probability $1-p$ and comes at cost R. The system is depicted in Figure 2.

We cast the admission control problem into the framework of MDPs. To this end, we denote by $(q,s,y) \in \mathcal{X} = \mathbb{N} \times \{0,\ldots,c\} \times \mathbb{N}$ that q calls are in the waiting buffer, s agents are occupied, and y calls are present in the retrial orbit. The system is subject to holding costs for the calls waiting, the calls in service, and the calls in the retrial orbit. Now, we can formulate the optimality equations for this system. For simplicity of the expression, we assume a fixed value for λ and a fixed number of agents c for the moment, but we study the more general case in the sequel.

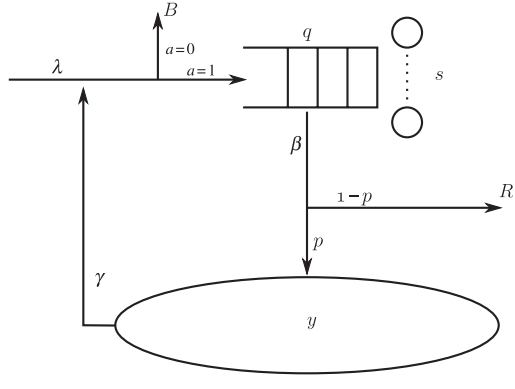

FIGURE 2. Representation of the $M/M/c$ queue with abandonments and retrials.

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

We uniformize the system similar to the $M/M/c$ queue in Section 3. However, in this case the total rate of change in the system is not bounded due to parallel retrials in the orbit and parallel abandonments in the queue. In our experiments we bound the size of the buffer and the size of the orbit so that the uniformization constant is finite. The bound is taken such that the steady-state probabilities of reaching the bound are very low. Then, just as in Section 3, we can (without loss of generality) assume that the uniformization constant is smaller than one. Therefore, the optimality equations are given by

$$\begin{aligned} g + V(q,s,y) = {} & (q+s+y) + \lambda \min\big\{V(q+\mathbb{1}_{\{s=c\}}, s+\mathbb{1}_{\{s<c\}}, y); \\ & \qquad V(q,s,y) + B\big\} \\ & + \gamma y \min\big\{V(q+\mathbb{1}_{\{s=c\}}, s+\mathbb{1}_{\{s<c\}}, [y-1]^+); \\ & \qquad V(q,s,[y-1]^+) + B\big\} \\ & + \min\{s,c\}\mu V(q-\mathbb{1}_{\{q>0\}}, [s-\mathbb{1}_{\{q=0\}}]^+, y) \\ & + p\beta q V([q-1]^+, s, y+\mathbb{1}_{\{q>0\}}) + (1-p)\beta q[V([q-1]^+, s, y) + R] \\ & + (1-\lambda-\gamma y - \min\{s,c\}\mu - \beta q)V(q,s,y). \end{aligned} \tag{4}$$

The first step is to choose an approximation structure for the relative value function. We take a second-order polynomial approximation function with cross-terms among the queue length, the number of agents that are busy, and the number of calls in the retrial orbit. Let the vector $k = (1, q, s, y)$ be the vector corresponding to the state (q, s, y). We use this vector to describe our approximation architecture defined by

$$\tilde{V}(q,s,y,r) = \sum_{i=1}^{4}\sum_{j=i}^{4} r_{i,j}k_i k_j.$$

Note that also in this case (as in Section 3), the term xy is redundant.

In the second step, we use this approximation structure in combination with the approximate value iteration algorithm. To this end, we need to define a weight function, whose entries are stored in the matrix W used in the algorithm. As in Section 3, we will use a geometric function with the load as decay factor. Without retrials and abandonments, the system would be a standard $M/M/c$ queuing system for which the load is equal to $\lambda/(c\mu)$ per server. However, in our system there are retrials, but that is only from calls that abandon from the queue and thus did not visit the server. Hence, if p is close to one, then the load per server is approximately equal to $\lambda/(c\mu)$. The load of the retrial orbit is harder to determine. Therefore, we use the following weight function:

$$W(q,s,y) = \rho^y \left(\min\left\{\frac{\lambda}{c\mu}, \alpha\right\}\right)^{(q+s)},$$

where ρ and α are fixed numbers; we take them equal to 0.8 and 0.6, respectively. In our numerical experiments, we have chosen the arrival rates such that the system is

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

TABLE 2. Results of the System with $\mathcal{Y} = \{0, \ldots, 15\} \times \{0, \ldots, c\} \times \{0, \ldots, 15\}$

λ	μ	c	β	γ	P	R	B	g^0	$g^*(\tilde{V})$	g^*	Δ
5	2	2	4	1	0.5	10	7	20.406	17.909	17.513	2.3%
4	2	3	3	2	0.5	6	3	4.365	4.185	4.001	4.6%
4	2	3	3	2	0.5	10	4.5	5.737	5.448	5.155	5.7%
6	4	3	1	1	0.7	5	1	2.302	2.143	2.077	3.2%
10	4	3	1	1	0.7	5	2.5	7.716	6.995	6.990	0.1%

in an overload situation during peak hours but is stable on average. This resembles a realistic setting for most call centers.

Table 2 shows the numerical results for the system with fixed values for λ and c; that is, there is no time-varying parameter involved and the number of agents is fixed over time as well. In Table 2, g^0 is the average cost for the policy where all calls are accepted to the system, $g^*(V)$ is the optimal average cost obtained by using the approximation structure, and g^* is the cost obtained by using the optimal policy. The relative difference (Δ) between the optimal cost and the optimal cost obtained by the approximation is given by $(g^*(V) - g^*)/g^*(V)$.

The results show that there is a significant decrease in costs when using the approximation structure and the approximate value iteration algorithm. The difference between the optimal cost and the cost obtained by the approximation is below 10% for the extensive set of parameters we studied. Based on these good results, we conclude that the second-order polynomial architecture, which we described earlier, works very well.

We continue with the third step, and that is to describe the relation between the values of the time-varying parameter and the coefficients r of the approximation structure. To illustrate this step, we consider one particular example and give the results based on different parameter settings in a table. Let λ_t be as in Figure 1. The maximum call rate equals 15, the minimum call rate equals 5, and the average call rate is approximately equal to 10. There are three servers, and each of them handles a call at rate 4. The values for β and γ are chosen to be 1, and the probability of retrial is equal to 0.7. Rejection and blocking costs are 5 and 2.5, respectively. This resembles a situation in which sometimes there is more work than can be handled in peak hours, whereas in off-peak hours, the length of the queue can be decreased significantly.

For different values of λ between $\lambda_{\min} = 5$ and $\lambda_{\max} = 15$, we determine the coefficients (i.e., the value of the parameter vector r) in the approximation structure. This is done by using the approximate value iteration algorithm and the optimality equations (4). The relation between each coefficient and the values of λ is described by a polynomial function of degree three. In Figure 3 the optimal coefficients are plotted and the least squares fit of the third-order polynomial function.

Let $\tilde{V}_c^{(\cdot)}$ denote the approximate relative value function when the system has c agents and the system is controlled in a specific way (i.e., based on λ_t, $\bar{\lambda}$, $\lambda_{\min}$, or $\lambda_{\max}$). We compare the control of the system based on our approach with other approaches

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

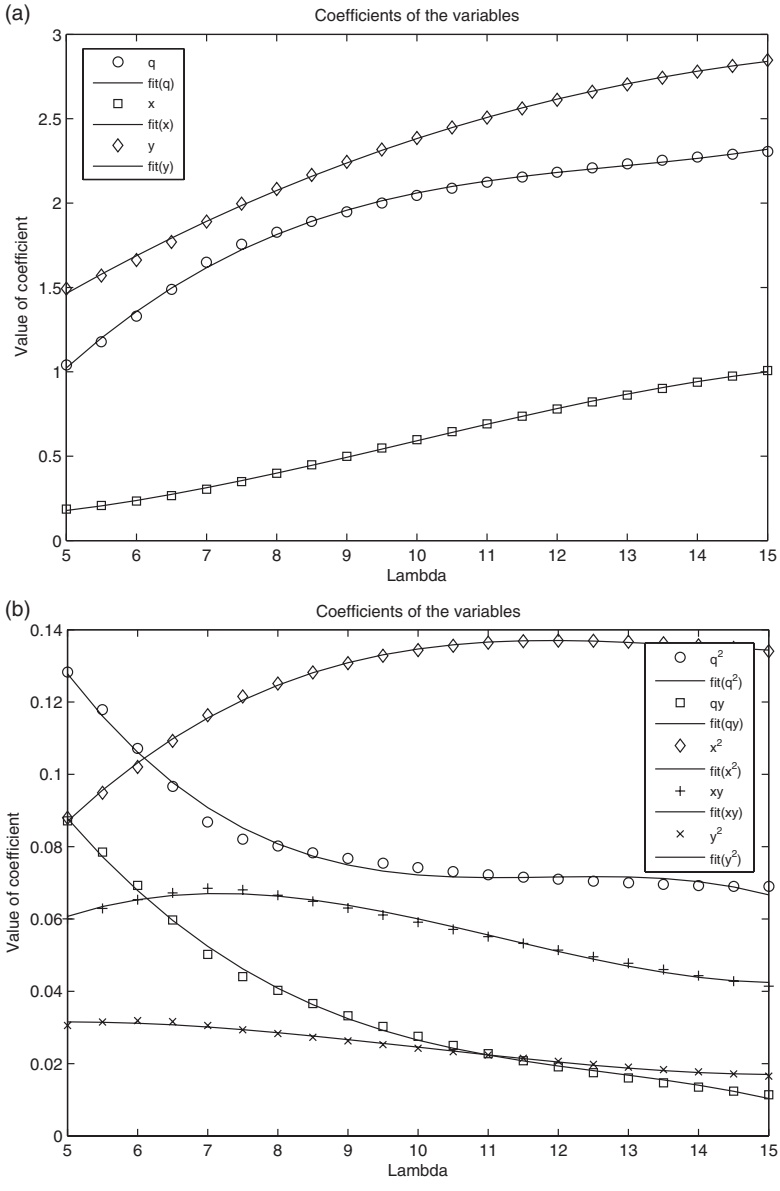

FIGURE 3. Relation between the value of λ and the coefficients in the approximation structure. (a) Relation for q, x and y; (b) Relation for q^2, qx, x^2, xy and y^2.

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

by means of simulation, since we have time-varying arrival rates of the calls. At each arrival of a call, the system admits or blocks the call based on

$$\min\left\{\tilde{V}_c^{(\cdot)}(q+\mathbb{1}_{\{s=c\}}, s+\mathbb{1}_{\{s<c\}}, y);\right.$$
$$\left.\tilde{V}_c^{(\cdot)}(q,s,y)+B\right\}$$

or

$$\min\left\{\tilde{V}_c^{(\cdot)}(q+\mathbb{1}_{\{s=c\}}, s+\mathbb{1}_{\{s<c\}}, [y-1]^+);\right.$$
$$\left.\tilde{V}_c^{(\cdot)}(q,s,[y-1]^+)+B\right\}$$

if an arriving call originates from outside the system or originates from the retrial orbit, respectively. In the decision, the approximation of $\tilde{V}_c^{(\cdot)}$ plays an important role. Moreover, the relative value function and, therefore, the approximation, is different for different values of λ. Therefore, to make good decisions, we should have knowledge about the relation between values of λ and the approximation of the relative value function. To quantify the impact of our approach, we compare it to the performance by using the average value of the arrival rate ($\tilde{V}_c^{(\bar{\lambda})}$, with $\bar{\lambda} = 10.0354$), the maximum value of the arrival rate ($\tilde{V}_c^{(\lambda_{\max})}$, with $\lambda_{\max} = 15$), and the minimum value of the arrival rate ($\tilde{V}_c^{(\lambda_{\min})}$, with $\lambda_{\min} = 5$). Table 3 shows the numerical results of these experiments, together with other problem instances. For these problem instances, the same shape of the arrival rate is taken, but the actual values are changed so that they fit between $\lambda_{\min}$ and $\lambda_{\max}$. Recall that we have four different periods in which the number of agents can vary.

The performance is determined by means of simulation. We simulate the system for a sufficiently large amount of time and divide the simulation into subruns, so that confidence intervals can be derived for the average costs of the system. In Table 3 the average costs are shown. For readability of the results, we have not included the confidence intervals but reside by mentioning that the standard deviations of the average costs are mostly of order 10^{-2}. To generate arrivals according to the inhomogeneous Poisson process with rate λ_t, we consider a Poisson process with rate $\lambda_{\max}$, where we select each point with probability $\lambda_t/\lambda_{\max}$. The selected points are then the call arrivals.

The initial policy that allows every arriving call to the system has the highest cost g^0. As expected, the admission control policy based on the approximation of the relative value function performs better in all cases. The control of the system based on the average arrival rate performs almost equally well compared to the control based on the minimum value of the arrival rate. Both perform worse than the control based on the maximum value of the arrival rate in most of the cases. Our approach ($\tilde{V}^{\lambda_t}$) outperforms all others, since this method uses the actual arrival rate into account. In combination with a parameter-tracking method, such as stochastic approximation, the controls can be taken on-line without the knowledge of the arrival rate function. However, this exercise of call arrival rate forecasting falls outside the scope of this article, since our aim is to show how to deal with time-varying parameters in a computationally feasible manner.

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

Table 3. Results of the System with $\mathcal{Y} = \{0, \ldots, 15\} \times \{0, \ldots, c\} \times \{0, \ldots, 15\}$

No.	$\lambda_{\min}$	$\lambda_{\max}$	μ	c	β	γ	P	R	B	g^0	$g(\tilde{V}^{\lambda_t})$	$g(\tilde{V}^{\bar{\lambda}})$	$g(\tilde{V}^{\lambda_{\min}})$	$g(\tilde{V}^{\lambda_{\max}})$
1	2	9	2	{3,4,6,4}	3	2	0.5	8	4	5.959	5.417	5.782	5.765	5.564
2	6	16	4	{3,4,5,4}	1	2	0.3	7	2	6.988	5.721	6.141	6.147	5.894
3	10	20	4	{4,5,6,5}	2	2	0.1	6	4	11.747	11.478	11.704	11.693	11.529
4	8	22	4	{3,5,7,5}	2	2	0.1	6	4	11.095	10.753	11.004	10.994	11.419
5	6	16	4	{3,4,5,4}	1	1	0.7	7	2	6.192	5.232	5.556	5.559	5.298
6	6	16	4	{3,4,5,4}	1	2	0.7	7	2	6.013	5.134	5.521	5.493	5.229
7	5	15	4	{3,4,5,4}	1	1	0.7	5	2.5	4.302	4.174	4.264	4.239	4.185
8	8	22	4	{3,5,7,5}	2	2	0.9	6	4	8.887	7.820	8.420	8.424	7.994
9	10	20	3	{5,7,8,7}	2	0.2	0.7	6	3	13.715	10.917	11.966	11.908	11.131
10	6	16	4	{3,4,5,4}	1	0.2	0.7	7	2	8.354	5.879	6.306	6.298	6.004
11	10	20	3	{5,7,8,7}	0.3	0.2	0.7	6	3	9.484	8.545	8.993	8.969	8.704
12	6	18	1	{9,15,20,15}	0.5	0.1	0.9	6	4	33.177	15.882	17.548	17.494	16.079
13	10	20	3	{5,7,8,7}	2	0.2	0.7	6	5	13.699	12.965	13.591	13.508	13.197
14	6	16	4	{3,4,5,4}	1	0.2	0.7	7	5	8.314	7.793	8.190	8.157	7.908
15	10	20	3	{5,7,8,7}	0.3	0.2	0.7	6	5	9.405	9.153	9.393	9.438	9.258
16	6	18	1	{9,15,20,15}	0.5	0.1	0.9	6	5	33.713	16.971	19.202	19.067	17.352
17	8	22	4	{3,5,7,5}	2	2	0.9	6	5	8.811	8.107	8.621	8.650	8.195

4.1. Unobservable Retrial Orbit

The optimal policy relies on the knowledge of the number of calls in the queue, the number of calls in service, as well as the number of calls in the retrial orbit. However, in practice, the latter is quite difficult, if not impossible, to obtain; when a customer abandons, it is not known beforehand if the customer will retry and thus it is not observed if the customer resides in the orbit.

The fact that the orbit cannot be observed in practice prohibits the direct application of our method. One straightforward way to deal with this is to ignore the information about the orbit. Thus, one can use the approximation architecture $\tilde{V}_c$ consisting only of the variables q and s, thus $\tilde{V}_c(q, s)$. A second approach would be to use $\tilde{V}_c(q, s, y)$ by providing a value for y. Numerical experiments show that the former method (i.e., ignoring the orbit) does not yield very good results. It is better to provide a reasonable value for y. Table 4 shows the results of these two approximation approaches, labeled App1 and App2, respectively. App2(a) uses $y = 0$ and App2(b) uses $y = 2$. Note that also here the standard deviations are not shown, but they are of order 10^{-2} as well.

It might be difficult to come up with a well-chosen value for y beforehand, since the best value can depend on all of the system parameters. It would be better to use $\tilde{V}_c(q, s, y)$ and try to estimate the number of calls in the retrial orbit at any point in time dynamically. To this purpose, we use a Bayesian estimation procedure. Let $u = (u_0, \ldots, u_N)$ be the vector of components u_i denoting the probability that there are i calls in the retrial orbit. We restrict the sample space for the orbit to $\{0, \ldots, N\}$.

When additional information becomes available through an event, we update the probability vector u. More specifically, an abandonment results with probability

TABLE 4. Results of the Unobservable Retrial Orbit System with $\mathcal{Y} = \{0, \ldots, 15\} \times \{0, \ldots, c\} \times \{0, \ldots, 15\}$

No.	$g(\tilde{V}^{\lambda_t})$	App1	App2(a)	App2(b)	App3(a)	App3(b)
1	5.417	5.683	5.431	5.452	5.436	5.421
2	5.721	5.819	5.723	5.738	5.724	5.717
3	11.478	11.419	11.508	11.427	11.464	11.456
4	10.753	10.743	10.746	10.732	10.747	10.743
5	5.232	5.508	5.258	5.223	5.225	5.220
6	5.134	5.216	5.132	5.120	5.138	5.112
7	4.174	4.277	4.171	4.166	4.154	4.149
8	7.820	8.826	8.049	7.927	7.800	7.830
9	10.917	13.764	10.910	10.920	10.935	10.904
10	5.879	7.426	5.874	5.870	5.899	5.884
11	8.545	8.819	8.572	8.545	8.565	8.536
12	15.882	31.225	15.889	15.910	15.881	15.896
13	12.965	13.706	13.110	13.025	12.904	12.916
14	7.793	8.316	7.824	7.812	7.778	7.749
15	9.153	9.180	9.137	9.180	9.153	9.140
16	16.971	33.377	16.947	17.019	16.973	16.947
17	8.107	8.809	8.190	8.205	8.080	8.091

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

p in a retrial, increasing the number of calls in the retrial orbit by one, whereas with probability $1-p$, nothing happens with the retrial orbit. As a consequence, after an abandonment occurs, the new probability vector u' has components $u'_i = pu_{i-1} + (1-p)u_i$, for $0 < i < N$, while setting $u'_0 = (1-p)u_0$ and $u'_N = p(u_{N-1} + u_N) + (1-p)u_N$. The term u_N in $u_{N-1} + u_N$ in the last expression is due to the fact that we have a finite sample space. Apart from updating the probability vector u at moments of abandonments, we also update the probability vector at moments a retrial occurs. This results in $u'_i = (1/a)u_{i+1}$, for $i < N$, and we let $u'_N = 0$, with $a = \sum_{i=1}^{N} u_i$. Furthermore, based on the distribution u, we expect to have $\mathbb{E}(u) = \sum_{i=0}^{N} iu_i$ calls in the retrial orbit, each having an exponentially distributed amount of time that they remain in the retrial orbit. Therefore, one expects to observe a retrial with probability, say .95, within τ time units, where $\tau = \arg\min_x F_X(x) \geq .95$, with $X \sim \exp(\gamma \mathbb{E}(u))$. If that amount of time elapses without the occurrence of a retrial, then we also update the probability vector u by $u'_i = (0.95/a)u_{i+1} + .05u_i$, for $i < N$, and we let $u'_N = 0.05u_N$. The value of τ is calculated each time an update of u occurs.

We use the probability vector u in two different ways. The first is to calculate the expectation $\mathbb{E}(u)$ and to use that as an estimation for the amount of calls in the retrial orbit each time a decision is made to control the system; that is, we let $y = \mathbb{E}(u)$. The second way is to calculate a weighted average of the value function with the probabilities as weights; that is, we let

$$V^a = \sum_{i=0}^{N} u_i \tilde{V}_c^{(\cdot)}(q + \mathbb{1}_{\{s=c\}}, s + \mathbb{1}_{\{s<c\}}, i),$$

$$V^r = \sum_{i=0}^{N} u_i \tilde{V}_c^{(\cdot)}(q, s, i) + B$$

for an arriving call originating from outside the system, where V^a and V^r denote the weighted value function for admission and rejection of the call, respectively. The optimal action is taken by comparing both weighted value functions. The optimal action to be taken at the arrival of a retrial is determined in a similar way. This first and second approach in using the probability vector u is denoted by App3(a) and App3(b), respectively, in Table 4.

Table 4 shows us that ignoring the number of calls in the retrial orbit completely (App1) does not yield very good results. A simple approach such as to fix the value for y (App2) gives good results. However, a good value for one problem instance could be bad for other problem instances. Estimating the number of calls in the retrial orbit (App3) outperforms the other methods. This is not only because these methods try to estimate the number of retrial calls, but they also circumvent the problem on how to choose a good value for y as required in App2. Note that the retrial orbit play an important role in cases in which blocking a call is relative expensive. However, this yields little improvement in the policies compared to the initial policy. On the other hand, to obtain large improvements, the blocking costs should be relatively small, but

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

that means that the number of calls in the retrial orbit is bounded by a small number in the optimal policy.

4.2. Structure of the Optimal Policy

In this subsection we study the structure of the optimal policy. To this end, we combine the number of waiting customers and the number of customers being served into one variable, and we use the number of calls in the retrial orbit as the second variable. This allows us to make a plot of the optimal actions. We consider the same problem instance as earlier: a system with three servers, each working at rate 4, and the other system parameters given by $\beta = \gamma = 1, p = .7, R = 5$, and $B = 2.5$. For three different values of a constant arrival rate ($\lambda = 5, 10$, and 15), the optimal actions are shown in Figure 4. The colored squares indicate the decision to allow an arriving call into the system that originates from outside the system. The dark, average, and light colored squares indicate that we allow the call in the particular states for all three values of λ, the values 5 and 10, and the value 5 for λ, respectively. The structure of the policy is such that for higher values of λ we block in more states. Note that the decision for an arriving call originating from the retrial orbit can be derived directly from that plot as well as by looking up the decision that corresponds to the state $(s + q, y - 1)$ instead of $(s + q, y)$.

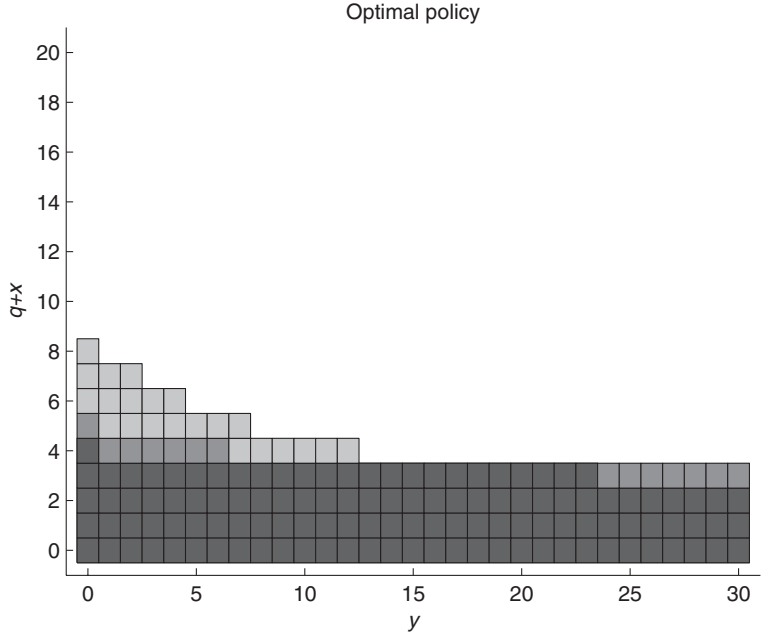

FIGURE 4. Optimal policy for $\lambda = 5, 10$, and 15.

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

5. CONCLUSIONS AND DISCUSSION

In this article we have studied the quality of approximations used in ADP by comparing the classical state space representation with a state space with increased dimensionality through state disaggregation. We observed that for queuing models, it is preferable to distinguish between entities in the waiting buffer and entities that are in service. This approach increases the quality of the approximation against little additional complexity.

Furthermore, we developed a technique to control time-varying queuing systems. The optimal admission control for queuing systems greatly depends on the arrival rate to the system. We have shown this by means of an example (cf. Fig. 4). To make optimal decisions over time with time-varying arrival rates, we expressed the parameters of the approximation structure as a function of the time-varying arrival rate. This allows us to take optimal decisions at any point in time using (an estimate of) the current arrival rate. Moreover, we have developed a Bayesian procedure to estimate the number of calls in the orbit, since this value cannot be observed in practice. We showed in our numerical example of a queuing system with abandonments and retrials that these techniques works well, although we used a simple approximation architecture and a simple function to describe the relation between the parameters of the approximation architecture and the time-varying arrival rate.

References

1. Bertsekas, D.P. & Tsitsiklis, J.N. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.
2. Bhulai, S. & Koole, G.M. (2003). On the structure of value functions for threshold policies in queueing models. *Journal of Applied Probability* 40: 613–622.
3. de Farias, D.P. & Van Roy, B. (2003). Approximate linear programming for average-cost dynamic programming. In Thrun, S., Becker, S., & Obermayer, K. (eds.), *Advances in neural information processing systems 15*, Cambridge, MA: MIT Press, pp. 1587–1594.
4. de Farias, D.P. & Van Roy, B. (2003). The linear programming approach to approximate dynamic programming. *Operations Research* 51(6): 850–865.
5. de Farias, D.P. & Van Roy, B. (2004). On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research* 29(3): 462–478.
6. Grassmann, W.K. (1977). Transient solutions in Markovian queueing systems. *Computers and Operations Research* 4: 47–53.
7. Green, L. & Kolesar, P. (1991). The pointwise stationary approximation for queues with nonstationary arrivals. *Management Science* 37: 84–97.
8. Hernández-Lerma, O. & Lasserre, J.B. (1996). *Discrete-time markov control processes: Basic optimality criteria*. New York: Springer-Verlag.
9. Hernández-Lerma, O. & Lasserre, J.B. (1999). *Further topics on discrete-time markov control processes*. New York: Springer-Verlag.
10. Ingolfsson, A., Akhmetshina, E., Budge, S., Li, Y., & Wu, X. (2007). A survey and experimental comparison of service level approximation methods for non-stationary M/M/*s* queueing systems. *INFORMS Journal of Computing* 19: 201–214.
11. Ingolfsson, A., Haque, M.A., & Umnikov, A. (2002). Accounting for time-varying queueing effects in workforce scheduling. *European Journal of Operational Research* 139: 585–597.

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press

12. Parr, R. (1990). Hierarchical control and learning for markov decision processes. Ph.D. dessertation, Berkeley, CA: University of California.
13. Powell, W.B. (2007). *Approximate dynamic programming: Solving the curses of dimensionality*. New York: Wiley.
14. Puterman, M.L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York: Wiley.
15. Ren, Z. & Krogh, B.H. (2002). State aggregation in Markov decision processes. In *Proceedings of the 41st IEEE Conference on Decision and Control*, vol 4, pp. 3819–3824.
16. Roubos, D. & Bhulai, S. (2007). Average-cost approximate dynamic programming for the control of birth-death processes. Technical report, VU University Amsterdam.
17. Singh, S., Jaakkola, T., & Jordan, M. (1995). Reinforcement learning with soft state aggregation. *Advances in Neural Information Processing Systems* 7: 361–368.
18. Singh, S.P. & Bertsekas, D.P. (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. *Advances in Neural Information Processing Systems* 9: 974–980.
19. Sutton, R.S. & Barto, A.G. (2000). *Reinforcement Learning: An introduction*. Cambridge, MA: MIT Press.
20. Tsitsiklis, J.N. & Van Roy, B. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning* 22: 59–94.
21. Yoo, J. (1996). Queueing models for staffing service operations. Ph.D. dessertation, College Park: University of Maryland.

https://doi.org/10.1017/S026996480999012X Published online by Cambridge University Press