2016

# Developing effective service policies for multiclass queues with abandonment: asymptotic optimality and approximate policy improveme

James, Terry; Glazebrook, Kevin; Lin, Kyle

Informs

T. James, K. Glazebrook, K. Lin, "Developing effective service policies for multiclass queues with abandonment: asymptotic optimality and approximate policy improvement," INFORMS Journals on Computing, v.28, no.2 (Spring 2016), pp. 251-264.
https://hdl.handle.net/10945/55152

*INFORMS Journal on Computing*
Vol. 28, No. 2, Spring 2016, pp. 251–264
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.2015.0675


# Developing Effective Service Policies for Multiclass Queues with Abandonment: Asymptotic Optimality and Approximate Policy Improvement

Terry James
STOR-i Doctoral Training Centre, Fylde College, Lancaster University, Lancaster, LA1 4YF, United Kingdom, t.james@lancaster.ac.uk

Kevin Glazebrook
Department of Management Science, Lancaster University, Lancaster, LA1 4YF, United Kingdom, k.glazebrook@lancaster.ac.uk

Kyle Lin
Operations Research Department, Naval Postgraduate School, Monterey, California 93943, kylin@nps.edu

We study a single server queuing model with multiple classes and impatient customers. The goal is to determine a service policy to maximize the long-run reward rate earned from serving customers net of holding costs and penalties respectively due to customers waiting for and leaving before receiving service. We first show that it is without loss of generality to study a pure-reward model. Since standard methods can usually only compute the optimal policy for problems with up to three customer classes, our focus is to develop a suite of heuristic approaches, with a preference for operationally simple policies with good reward characteristics. One such heuristic is the $R\mu\theta$ rule—a priority policy that ranks all customer classes based on the product of reward $R$, service rate $\mu$, and abandonment rate $\theta$. We show that the $R\mu\theta$ rule is asymptotically optimal as customer abandonment rates approach zero and often performs well in cases where the simpler $R\mu$ rule performs poorly. The paper also develops an approximate policy improvement method that uses simulation and interpolation to estimate the bias function for use in a dynamic programming recursion. For systems with two or three customer classes, our numerical study indicates that the best of our simple priority policies is near optimal in most cases; when it is not, the approximate policy improvement method invariably tightens up the gap substantially. For systems with five customer classes, our heuristics typically achieve within 4% of an upper bound for the optimal value, which is computed via a linear program that relies on a relaxation of the original system. The computational requirement of the approximate policy improvement method grows rapidly when the number of customer classes or the traffic intensity increases.

*Keywords*: multiclass queue; customer abandonment; Markov decision process; index policy; approximate policy improvement
*History*: Accepted by Winfried Grassmann, Area Editor for Computational Probability and Analysis; received April 2014; revised December 2014, May 2015, June 2015, July 2015; accepted August 2015. Published online March 8, 2016.

## 1. Introduction

This paper considers a setting in which a single server must preemptively serve impatient customers across $k$ customer classes. Different classes of customers arrive according to independent Poisson processes, with the arrival rate being $\lambda_i$ for class $i$ customers, $1 \le i \le k$. The service time for a class $i$ customer follows an exponential distribution with rate $\mu_i$. However, each class $i$ customer will only remain available for service for a random time that follows an exponential distribution with rate $\theta_i$, after which the customer will abandon the system, whether the customer is still waiting in the queue or is already in service. If a class $i$ customer is served to completion, then a reward $R_i$ is earned, but if he abandons the system before service completion, then a penalty $D_i$ is incurred. In addition, each class $i$ customer in the system incurs a linear holding cost at rate $c_i$ per time unit. We seek to determine a service policy that maximizes the long-run reward rate earned net of penalties and holding costs incurred.

Our model has real-world applications. For security surveillance, a customer in a class corresponds to a suspect in a particular area, with service being the screening of suspects by a security resource. The penalties $D_i$ represent the expected damage incurred when suspects leave the area and evade screening, and it is most natural to set $R_i = c_i = 0$. For call centers, the class of a customer indicates a particular service need, and the server corresponds to an agent. The rewards $R_i$ represent the revenue received by serving a customer, and $c_i$ and $D_i$ respectively model the loss of

goodwill incurred because of customers waiting and hang up. Customer abandonments are a key feature in both applications.

The first step of our analysis is to show that the three parameters in the reward structure—namely $R_i$, $D_i$, and $c_i$—can be consolidated into a single parameter through proper transformation. As it turns out, it is without loss of generality to consider a reward structure with only one of these three parameters while setting the other two equal to zero. In this paper, we choose the pure reward model (with $D_i = c_i = 0$ for all $i$). Although it is possible to formulate our model as a Markov Decision Process (MDP) and use standard methods of dynamic programming (DP) to compute the optimal policy by truncating the state space, the computation is usually only practical for problems with up to three customer classes. Hence, the paper focuses on developing strongly performing heuristic policies, with a preference for operationally simple policies with strong reward characteristics.

Our first approach is to develop a suite of simple *priority policies*, which are effective across much of the problem's parameter space. Such policies serve customers according to a strict priority ordering among the customer classes. In the case where the system is overloaded, it has been shown in the literature that the $R\mu$ rule—a priority policy that ranks all customer classes based on the product of reward $R$ and service rate $\mu$—performs well since it maximizes the instantaneous reward rate (Atar et al. 2010, Ayesta et al. 2011, Verloop 2014, Larrañaga et al. 2014). To complement the $R\mu$ rule in the light-traffic case, we study the $R\mu\theta$ rule, which ranks all customer classes based on the product of $R$, $\mu$, and the abandonment rate $\theta$. This ranking was proposed in Glazebrook et al. (2004, §2) for batch problems, and our paper extends its application to systems with customer arrivals. We prove that the $R\mu\theta$ rule is asymptotically optimal as customer abandonment rates approach zero in light traffic systems. Finally, we apply a pairwise-swapping (PaS) mechanism to both the $R\mu$ rule and the $R\mu\theta$ rule to search for an improved priority policy.

Our second approach—after having produced a set of simple priority policies—is to develop an effective approximate policy improvement (API) method. For a given policy, the API method uses simulation to estimate bias values for a set of carefully chosen states and then uses these values to interpolate the bias function for all states. This approximate bias function allows us to run policy improvement to obtain a new policy. Our numerical results indicate that, in most cases, the best priority policy is nearly optimal in systems with two or three customer classes; in the cases where it is not, the API method invariably tightens up the gap substantially. In one instance, the API method improves our best priority policy—which is 4.26% suboptimal—in yielding a policy that is only 0.04% suboptimal. In some applications such as security surveillance, even a small difference in reward rate performance can be of practical importance.

To evaluate our heuristic approaches for systems with more than three customer classes, where obtaining the optimal policy is computationally too intensive, we also develop a novel upper bound for the maximal long-run reward rate using linear programming methods. The linear program relies on a relaxation of the original system, and its tightness degrades as the number of customer classes increases. In our numerical study with five customer classes, our best heuristic is typically within 4% of the upper bound for the maximal long-run reward rate.

Our model has been studied in Glazebrook et al. (2004), which develops a heuristic policy via a two-stage process. The first stage analyzes a policy that allocates a fixed-service effort to each class at all time and computes the optimal allocation, and the second stage performs an exact policy improvement (PI). The parametric optimization involved at the first stage can pose computational challenges, especially when the number of customer classes increases, and was only implemented for cases with two customer classes in Glazebrook et al. (2004). Numerical tests show that our API method performs considerably better for a wide range of parameters. Down et al. (2011) studied a special case of our model with two customer classes and $\mu_1 = \mu_2$. They proved that the priority policy that serves a class 1 customer whenever possible is optimal if $R_1 \geq R_2$ and $\theta_1 \geq \theta_2$.

There are some recent works on control of queueing systems with impatient customers. An approach based on approximating Brownian control problems in heavy traffic has been explored by Harrison and Zeevi (2004), Ata and Tongarlak (2013), and Kim and Ward (2013). The latter article considers general arrival, service, and abandonment processes. Recent studies of controlled stochastic systems with customer abandonments have featured a range of application domains. Garnett et al. (2002), Jouini et al. (2010), and Bassamboo et al. (2005) consider call center applications. Abandonments have been used in military applications to model targets that move out of range of defensive forces. See, for example, Gaver et al. (2006), Lin et al. (2009), and Glazebrook and Punton (2008). In patient triage applications, abandonments have been used to model medical emergency patients in danger of dying while awaiting treatment. See, for example, Argon et al. (2008) and Li and Glazebrook (2010).

There are also some recent works on approximate approaches to DP seeking to overcome computational intractability. See, for example, Powell (2011, §1.2). Contributions that deploy value function approximations within a PI approach include those of Krishnan (1987),

Glazebrook et al. (2004), and Li and Glazebrook (2010), whereas API methods which utilize simulation are discussed in Powell (2011, §10.5) and Bertsekas (2012, Chap. 6 and 7). Our API approach can be viewed as a refined approximate dynamic programming (ADP) implementation with two distinctive features: (1) a suite of strongly performing priority policies to initialize the API algorithm and (2) a simulation/interpolation methodology to fit the bias surface by estimating biases both at states that are frequently visited and also at a carefully chosen set of widely spread states.

The rest of the paper proceeds as follows. Section 2 first shows that it is without loss of generality to consider a pure reward model; then it formulates the problem as an MDP and describes how, in principle, to compute the optimal policy. Section 3 presents the service policies based on class prioritization, including the $R\mu\theta$ rule, the $R\mu$ rule, and a potentially improved priority policy achieved by pairwise swapping. Section 4 describes the API method. Section 5 presents a linear program to compute an upper bound for the maximal long-run reward rate to evaluate our heuristics where the optimal policy is not available. Section 6 offers a numerical study, and §7 concludes the paper.

## 2. Model and MDP Formulation

Recall that our model has three reward parameters. For a class $i$ customer, there is a reward $R_i$ for service completion, a penalty $D_i$ for customer abandonment, and a linear holding cost rate $c_i$ per time unit, for $1 \le i \le k$. If we write $N^{\pi}_{i,\theta}$ for the number of class $i$ customers in the system and $\alpha^{\pi}_i$, $\beta^{\pi}_i$ for, respectively, the rate of class $i$ service completions and abandonments under policy $\pi$ in steady state, then the optimal long-run system reward rate net of holding costs and abandonment penalties can be written as

$$\max_{\pi} \sum_{i=1}^{k} (R_i \alpha^{\pi}_i - D_i \beta^{\pi}_i - c_i E[N^{\pi}_{i,\theta}]). \tag{1}$$

However, the guaranteed stability of the system implies that for all choices of class $i$ and policy $\pi$, we have that

$$\lambda_i = \alpha^{\pi}_i + \beta^{\pi}_i, \tag{2}$$

and

$$\beta^{\pi}_i = \theta_i E[N^{\pi}_{i,\theta}]. \tag{3}$$

Using (2) and (3), we can rewrite (1) in three different ways:

$$\max_{\pi} \sum_{i=1}^{k} \left( \left( R_i + D_i + \frac{c_i}{\theta_i} \right) \alpha^{\pi}_i - \left( D_i + \frac{c_i}{\theta_i} \right) \lambda_i \right) \tag{4}$$

$$= \max_{\pi} \sum_{i=1}^{k} \left( R_i \lambda_i - \left( R_i + D_i + \frac{c_i}{\theta_i} \right) \beta^{\pi}_i \right) \tag{5}$$

$$= \max_{\pi} \sum_{i=1}^{k} (R_i \lambda_i - ((R_i + D_i)\theta_i + c_i) E[N^{\pi}_{i,\theta}]). \tag{6}$$

Equation (4) transforms the original model into an equivalent pure-reward model with $R_i + D_i + c_i/\theta_i$ earned upon every class $i$ service completion. Similarly, Equation (5) shows an equivalent model with only penalties upon customer abandonment, whereas Equation (6) shows an equivalent model with only linear holding costs. Without loss of generality, we shall focus on the pure-reward model ($D_i = c_i = 0$ for all $i$) for the remainder of the paper.

Denote the system state by $\mathbf{n} = (n_1, \ldots, n_k)$, with $n_i$ the number of class $i$ customers present in the system. We further write $\mathbf{n}(t)$ for the system state at time $t$. Further details of the model are as follows:

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

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

$$A(\mathbf{n}) = \{a\colon n_a \ge 1, 1 \le a \le k\}.$$

We use $\mathbf{e}_i$ for the system state in which only a single customer of class $i$ is present in the system.

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

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

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

A standard approach to determine the $\epsilon$-optimal policies is through the application of DP to a version of the above system with finite state space $\times_{i=1}^{k} \{0, 1, \ldots, N_i\}$. In this truncated version, new class $i$ customers are blocked from entering the system when $N_i$ are already present. The $N_i$ must be chosen large enough to ensure that this system approximates the original model well enough for the purpose at hand. With a finite state space, it becomes possible to convert the problem to one in discrete time through the process of *uniformization*. We write $\Delta = \sum_{i=1}^{k} (\lambda_i + \mu_i + N_i\theta_i)$, a uniform upper bound on the rate of state transitions in the finite state

system. By the addition of fictitious transitions from a state to itself, we develop a uniformized system that makes transitions at a uniform rate $\Delta$. We write $V^\pi(\mathbf{n}, t)$ and $V(\mathbf{n}, t)$ for the expected reward earned under the application of policy $\pi$ and an optimal policy, respectively, over $t$ transitions of the uniformized process, beginning at time zero in system state $\mathbf{n}$. Standard theory enables us to write $V^\pi(\mathbf{n}, t) = (g^\pi/\Delta)t + \omega^\pi(\mathbf{n}) + o(1)$ and $V(\mathbf{n}, t) = (g/\Delta)t + \omega(\mathbf{n}) + o(1)$ as $t \to \infty$, where $g^\pi$ and $g$ are the long-run reward rates or gains earned, and $\omega^\pi$ and $\omega$ are the bias functions under application of $\pi$ and an optimal policy, respectively. Bias functions yield an estimate of the transient effect on rewards of the starting state $\mathbf{n}$ and will be further discussed in §4. Bellman's equation for the finite state system can now be written as

$$\frac{g}{\Delta} + \omega(\mathbf{n}) = \max_a \left\{ \frac{R_a \mu_a}{\Delta} + \sum_{\mathbf{n}' \in \mathcal{S}} p(\mathbf{n}' \mid \mathbf{n}, a)\omega(\mathbf{n}') \right\}, \quad (7)$$

where the $p(\mathbf{n}' \mid \mathbf{n}, a)$ are transition probabilities under the uniformization. It is now possible to compute the optimal gain and associated optimal policy for the finite state approximation by a recursive scheme such as DP value iteration or by linear programming; further details may be found in Puterman (1994, Chap. 8). However, since the state space grows exponentially in $k$, in practice, the computations quickly become intractable for $k \geq 4$. Hence, the focus of our paper is to develop near-optimal heuristic policies that require much less computation.

## 3. Service Policies Based on Class Prioritization

A policy that is easy to implement is for the server to prioritize all customer classes in an ordered list and always serve a customer highest on the list among all customers present in the system. In the case of an overloaded system, there are almost always many customers present in the system. It is therefore intuitive that the server should pay little attention to the possibility of idling and focus on continuously maximizing the instantaneous reward rate. To do so, the server simply needs to always serve a customer having the maximal $R\mu$ value among all customers present in the system—hence the $R\mu$ rule. As seen in Equations (4) and (6), the $R\mu$ rule in our pure-reward model is equivalent to the $c\mu/\theta$ rule in the linear-holding-cost only model. The strong performance of this rule in heavy traffic is affirmed in the work of Atar et al. (2010), Ayesta et al. (2011), Verloop (2014), and Larrañaga et al. (2014).

Away from heavy traffic, lost reward opportunities due to an empty system become a much more important concern. Motivated by this observation, §3.1 introduces the $R\mu\theta$ rule and establishes its asymptotic optimality. Section 3.2 compares the $R\mu\theta$ and the $R\mu$ rules. Section 3.3 presents a mechanism to explore local improvements on a given priority policy.

### 3.1. The $R\mu\theta$ Rule

If a system is not overloaded with customers, then it becomes important to take into account the lost reward opportunities when the system becomes empty because of customer abandonment. For example, consider a two-class system, with $R_1\mu_1 = R_2\mu_2$ and $\theta_1 < \theta_2$. If there is one customer present from each class, then intuition suggests that the server should first serve the class 2 customer since there is a better chance that the class 1 customer will still be available later on. Consequently, a class's priority should go up as its abandonment rate $\theta$ increases. We call the rule in which the server always serves a customer having the maximal $R\mu\theta$ value among all customers present in the system the $R\mu\theta$ rule. As seen in Equations (4) and (6), the $R\mu\theta$ rule in our pure-reward model is equivalent to the $c\mu$ rule in the linear-holding-cost only model. Whereas the $c\mu$ rule is optimal in queueing systems with no customer abandonment (see, for example, Gittins et al. 2011, §5.2), it is not optimal in systems with customer abandonment (Down et al. 2011).

The main result of this section is to show that the $R\mu\theta$ rule is asymptotically optimal as $\theta \to 0$. First, write $R^\pi(\theta)$ for the reward rate achieved by policy $\pi$, and $R^{R\mu\theta}(\theta)$ for the reward rate achieved by the $R\mu\theta$ rule. To describe the limiting regime, we suppose that the abandonment rate of each customer class is the multiple of some underlying rate $\theta$, such that $\theta_i = \theta\nu_i$, where $\nu_i > 0$, $1 \leq i \leq k$.

THEOREM 1. *If* $\sum_{i=1}^k (\lambda_i/\mu_i) < 1$, *then*

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

The proof of Theorem 1 is given in full in Appendix A in the online supplement (available as supplemental material at http://dx.doi.org/10.1287/ijoc.2015.0675), but we summarize the key elements here to facilitate the subsequent discussion. The main idea of the proof is to bound $R^\pi(\theta)$ below for priority policies and above for general nonidling policies. If we write $W_i^\pi$ for the waiting time (time to achieve completed service) of a class $i$ job in steady state under $\pi$ for the no abandonment case ($\theta = 0$), then for a priority policy $\varpi$, we show that under the conditions of the result,

$$R^\varpi(\theta) \geq \sum_{i=1}^k \lambda_i R_i - \theta \sum_{i=1}^k \lambda_i R_i \nu_i E[W_i^\varpi] + O(\theta^2), \quad (8)$$

whereas for all nonidling policies $\pi$, we have that

$$R^\pi(\theta) \leq \sum_{i=1}^k \lambda_i R_i - \theta \sum_{i=1}^k \lambda_i R_i \nu_i E[W_i^\pi] + O(\theta^2). \quad (9)$$

It must follow that

$$\max_{\pi} R^{\pi}(\theta) - R^{R\mu\theta}(\theta) \leq \theta\left\{\sum_{i=1}^{k}\lambda_i R_i \nu_i E[W_i^{R\mu\theta}] - \min_{\pi}\sum_{i=1}^{k}\lambda_i R_i \nu_i E[W_i^{\pi}]\right\} + O(\theta^2), \tag{10}$$

where the maximum and minimum in (10) are over all policies $\pi$. By Little's law, the minimization in (10) is of a holding cost rate objective for the system without abandonments. A classical queueing control result (the $c\mu$ rule) implies that this minimum is achieved by the $R\mu\theta$ rule. Theorem 1 easily follows.

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

$$h^{\pi} = \sum_{i=1}^{k}\lambda_i R_i \nu_i E[W_i^{\pi}].$$

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

Close inspection of the proof of Theorem 1 in the online supplement will reveal that we make little use of the stochastic structure of the system's service mechanism. The $R\mu\theta$ rule emerges as a priority policy which minimizes a holding cost-type objective for the no abandonment system ($\theta = 0$). It is therefore unsurprising that the result can be generalized to more complex service situations, provided that a priority policy continues to optimize an appropriate holding cost. An important model class to which a natural extension of Theorem 1 applies are *Klimov Networks* (see Klimov 1974 and 1978), in which each customer service has a sequence of phases, with movement between phases and toward service completion determined by customer class-specific Markovian routing matrices. This class *inter alia* provides an extension of Theorem 1 to a model in which service requirements are independent, have finite second moment, and are identically distributed within each class. Further details can be found in Appendix B in the online supplement.

All of the extensions to Theorem 1 mentioned above concern single-server systems. If we move to a multiserver version of our system with abandonments, with $m$ servers working in parallel, then the required stability condition becomes $\rho = \sum_{i=1}^{k}(\lambda_i/\mu_i) < m$ and the $R\mu\theta$ rule now allocates preemptive service to the $m$ customers present in the system whose associated $R\mu\theta$ are maximal. The proof of a suitable version of Theorem 1 for this system goes through up to (10). However, it is no longer true that the $R\mu\theta$ rule achieves the minimum in (10), though it does come close to doing so. To give a theoretical result for this system we need the quantity

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

where $I$ is an indicator and the maxima in the expression are taken over the customer classes. The following result makes use of Theorem 3 in Glazebrook and Niño-Mora (2001), which shows that $B(m)$ bounds above the quantity multiplying $\theta$ on the right-hand side of (10) when there are $m$ servers. It generalizes Theorem 1 to multiserver systems.

Proposition 1. *When there are $m$ servers and $\sum_{i=1}^{k}(\lambda_i/\mu_i) < m$, we have that*

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

### 3.2. Comparing the $R\mu\theta$ and $R\mu$ Rules

It follows from calculations in the proof of Theorem 1 that when $\sum_{i=1}^{k}(\lambda_i/\mu_i) < 1$, we have $R^{\pi}(\theta) \to \sum_{i=1}^{k}\lambda_i R_i$, as $\theta \to 0$, for all priority policies (and hence both the $R\mu\theta$ and $R\mu$ rules). Unsurprisingly, all priority policies achieve the maximal reward rate $\sum_{i=1}^{k}\lambda_i R_i$ in the no abandonment limit since in the limit all jobs are served. Think of a surveillance problem in which abandonments of the system are rare, but very damaging, and attention focuses on making the $O(\theta)$ loss rate from abandonments as small as possible. Now consider a situation in which the class orderings determined by the $R\mu\theta$ and $R\mu$ rules are distinct. It follows from (8) and (9) that

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

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

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

It follows that the percentage loss of reward rate due to abandonment from the use of $R\mu\theta$ relative to that experienced from the use of $R\mu$ becomes negligible in the limit $\theta \to 0$. Numerical support for this conclusion can be found in Appendix C in the online supplement.

A similar conclusion can be drawn for a multiserver system, if the condition in Proposition 1 is met. It follows from the analysis in Glazebrook and Niño-Mora (2001) that for any case in which $R\mu$ and $R\mu\theta$

rules differ in their choice of lowest priority customer class, the difference $h^{R\mu} - \min_\pi h^\pi$ diverges in the limit. It then follows from (8), (9), and Proposition 1 that when $\theta$ is small for problems close to the $\rho \to m$ limit, the suboptimality gap $\max_\pi R^\pi(\theta) - R^{R\mu\theta}(\theta)$ will be negligible compared to $\max_\pi R^\pi(\theta) - R^{R\mu}(\theta)$.

In the special case with $k = 2$, $\mu_1 = \mu_2$, $R_1 \geq R_2$, and $\theta_1 \geq \theta_2$, both the $R\mu\theta$ and $R\mu$ rules prescribe the priority policy $1 \to 2$, which is optimal according to Down et al. (2011).

### 3.3. The PaS Class of Priority Policies

We conclude this section by describing a simple *pairwise-swapping* mechanism to explore local improvements in any given priority policy. Given any class ordering $(\pi_1, \pi_2, \ldots, \pi_k)$, we take the classes in order from $\pi_2$ to $\pi_k$ and explore, in turn, whether each class should be promoted up the order. This is achieved for each class by a sequence of pairwise comparisons with the next highest class in the list to determine how high up the list the class can be promoted. In comparing classes $i$ and $j$, we consider the two-class subsystem comprising them alone (with their class parameters inherited from the full problem) and use value iteration to compute the respective performance of the two priority policies $i \to j$ and $j \to i$. If the better policy contradicts the current class ordering, then a pairwise swap is performed, and the procedure is repeated until a comparison does not result in a swapping. We then examine the potential promotion for the next class on the original list $\pi_2$ to $\pi_k$. We label this priority policy PaS.

## 4. Approximate Policy Improvement Algorithm

This section introduces an approximate policy improvement (API) algorithm to improve our suite of simple priority policies. Section 4.1 overviews the methodology, and §4.2 discusses the algorithm in detail.

### 4.1. Heuristic Based on Policy Improvement

Policy improvement (PI) develops optimal policies for MDPs by using the DP recursion to produce a sequence of successively improving policies (Howard 1960). In our problem, we truncate the state space and uniformize, as in §2, to develop an ergodic system with optimality equation in (7). To develop a PI step from policy $\pi$, let $\omega^\pi(\mathbf{n})$ be the bias associated with system state $\mathbf{n}$ under policy $\pi$. A new policy $\mathrm{PI}_\pi$, say, is obtained as follows:

$$\mathrm{PI}_\pi(\mathbf{n}) = \arg\max_a \left\{ \frac{R_a \mu_a}{\Delta} + \sum_{\mathbf{n}' \in \mathcal{S}} p(\mathbf{n}' \mid \mathbf{n}, a) \omega^\pi(\mathbf{n}') \right\}. \quad (11)$$

Accordingly, policy $\mathrm{PI}_\pi$ always takes the current decision optimally, given that all future decisions are made according to $\pi$. Tijms (1994) noted that the first few PI iterations usually yield the greatest improvement.

The challenge to implementation of PI in large systems lies in the intractability of the computation of the bias $\omega^\pi$. Hence, approximations are required, and the PI step in (11) can be replaced by

$$\mathrm{API}_\pi(\mathbf{n}) = \arg\max_a \left\{ \frac{R_a \mu_a}{\Delta} + \sum_{\mathbf{n}' \in \mathcal{S}} p(\mathbf{n}' \mid \mathbf{n}, a) \tilde{\omega}^\pi(\mathbf{n}') \right\}, \quad (12)$$

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

Computation of the bias $\omega^\pi$ involves specification of a reference state $\mathbf{m}$, which we take to be one frequently visited under $\pi$. We introduce the following quantities:

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

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

$$\omega^\pi(\mathbf{n}) = r^\pi(\mathbf{n}) - g^\pi t^\pi(\mathbf{n}), \quad (13)$$

where $g^\pi$ is the gain of policy $\pi$.

From (13), the approximations $\tilde{\omega}^\pi(\mathbf{n})$ can then be obtained by approximating the quantities $r^\pi(\mathbf{n})$, $t^\pi(\mathbf{n})$, and $g^\pi$. The heuristic policy can then be defined from (12).

### 4.2. The Algorithm

The implementation of an API step depends crucially on the approximation scheme used for the bias function. Because the bias function does not have an analytical form, we use simulation to estimate it. However, since simulation carries a computational cost, our constrained computational resource needs to be effectively managed through a carefully designed algorithm. The algorithm consists of five sequential, complementary stages, taking an initial policy $\pi$ as an input to produce a new policy $\mathrm{API}_\pi$. The five steps are summarized next, with more details to follow.

1. Pilot: Simulate the steady state of initial policy $\pi$ to estimate its gain and the frequency each state is visited.

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

3. Sampling: For each state $\mathbf{n}$ selected, simulate the system under $\pi$ from that state until some chosen reference state $\mathbf{m}$ is entered and estimate $\omega^\pi(\mathbf{n})$ using (13).

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

5. Improvement: Use (12) to produce a new policy $\text{API}_\pi$.

In step 1, we run a pilot steady state simulation to estimate the gain $g^\pi$ required to estimate $\omega^\pi$ from (13). To facilitate steps 2 and 3, we also collect data on how often each state is visited in steady state under $\pi$.

Step 2 consists of the selection of a small number of states, denoted $S_{\text{sel}}$, at which the bias $\omega^\pi$ will be estimated by simulation in step 3. Interpolation of $\omega^\pi$ at other states will follow in step 4. The set $S_{\text{sel}}$ consists of the *anchor set* together with a *support set*. The anchor set consists of the states most frequently visited in the pilot and hence influential to policy performance. However, anchor states are likely to be tightly grouped together, so alone they will not create an adequate basis for the construction of an effective interpolation scheme. The support set will complement the anchor set to ensure adequate coverage and wider exploration of the state space.

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

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

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

In step 3, we choose reference state $\mathbf{m}$ to be the one most visited in the pilot and use Monte Carlo simulation to estimate $r^\pi(\mathbf{n})$ and $t^\pi(\mathbf{n})$ for each $\mathbf{n} \in S_{\text{sel}}$. In what follows, we use $n$ for the size of $S_{\text{sel}}$ and $m$ for the number of simulated realizations of the system from each $\mathbf{n} \in S_{\text{sel}}$ until entry into reference state $\mathbf{m}$. If we write $R^\pi(\mathbf{n})$ and $T^\pi(\mathbf{n})$ for the simulation-based estimators of $r^\pi(\mathbf{n})$ and $t^\pi(\mathbf{n})$, respectively, and $G^\pi$ for the estimator of $g^\pi$ available from the pilot, then from (13) our estimator of $\omega^\pi(\mathbf{n})$ for $\mathbf{n} \in S_{\text{sel}}$ is $\Omega^\pi(\mathbf{n}) = R^\pi(\mathbf{n}) - G^\pi T^\pi(\mathbf{n})$. Since all estimators are unbiased, and $G^\pi$ is independent of $R^\pi(\mathbf{n})$ and $T^\pi(\mathbf{n})$, we conclude by conditioning on $G^\pi$ that

$$\text{Var}\{\Omega^\pi(\mathbf{n})\} = \text{Var}\{R^\pi(\mathbf{n}) - g^\pi T^\pi(\mathbf{n})\} + \text{Var}(G^\pi) E[(T^\pi(\mathbf{n}))^2]. \quad (14)$$

Equation (14) decomposes the variance of the bias estimators into two terms. The first term is controlled by the number of replicates $m$ used in the simulation relating to state $\mathbf{n}$ in step 3, and the second term is controlled by the size of the pilot study in step 1. The computational challenge is dominated by the need to control the first term in (14) because designing a pilot study large enough to control the second term has not proved to be an issue. One feature that helps reduce the first term is that $R^\pi(\mathbf{n})$ and $T^\pi(\mathbf{n})$ are positively associated. In addition, our choice of reference state $\mathbf{m}$ means that the biases at anchor states (with smaller $R^\pi(\mathbf{n})$ and $T^\pi(\mathbf{n})$) tend to be estimated with greater precision than those at support states, which is a feature shared with other approaches to ADP (see Powell 2011, §10.10). The central trade-off for the quality of the method for given computational effort is that between large $n$ supporting the quality of the interpolation and large $m$ supporting precision at the selected states.

In step 4, we use the bias estimates in $S_{\text{sel}}$ to interpolate a bias function approximation for the entire state space $S$. Although there are many interpolation algorithms, we use the *radial basis function* method (see Powell 1987) for its simplicity. Assume that some function $f\colon S \to \mathbb{R}$ has known values at each $\mathbf{x}_i \in S_{\text{sel}}$. An augmented radial basis function $h\colon S \to \mathbb{R}$ that takes the form

$$h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \phi(\|\mathbf{x} - \mathbf{x}_i\|) + \sum_{j=1}^{d} \beta_j p_j(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^k, \quad (15)$$

will be designed as a smooth interpolator of $f$, taking the values $f(\mathbf{x}_i)$ for $\mathbf{x}_i \in S_{\text{sel}}$. From (15), $h(\mathbf{x})$ is a weighted sum of $n = |S_{\text{sel}}|$ radial basis functions $\phi(\cdot)$, one centered on each $\mathbf{x}_i \in S_{\text{sel}}$, together with $d$ low order polynomials $p_j(\cdot)$. Note that $\|\cdot\|$ denotes the Euclidean norm. For $\phi(\cdot)$, we take the *thin plate spline* $\phi(r) = r^2 \log(r)$; for low order polynomials, we set $d = k + 1$ and use $p_1(\mathbf{x}) = 1$, $p_j(\mathbf{x}) = x_{j-1}$, $2 \le j \le k+1$. These choices produce a surface which minimizes a measure of smoothness (Powell 1999).

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

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

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

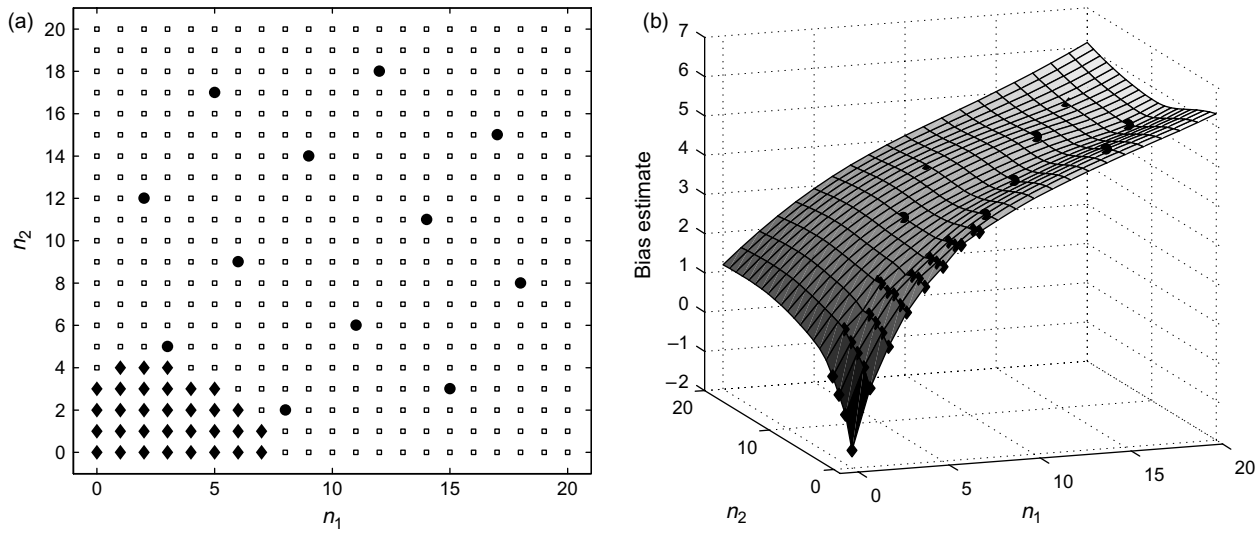


**Figure 1** **Illustration of the Selection and Interpolation Stages of the Algorithm in the Example System**

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

In step 5, we design a new policy $\text{API}_\pi$ by using the function $h$ from step 4 in place of $\tilde{\omega}^\pi$ in (12) to obtain

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

In principle, the preceding procedure can be repeated multiple times. Although obtaining progressively better policies—a feature of exact PI—can no longer be guaranteed, we have found that, in practice, improvement in policy performance is indeed achieved. To highlight key design choices, we denote the preceding procedure by $\text{API}(\pi, n, m, r, t)$. The parametrizing arguments offer great flexibility and are as follows: the initial policy is $\pi$, $n$ is the number of states selected for bias estimation via simulation, $m$ is the number of replicated simulations at each selected state, $r$ is the fraction of selected states in the anchor set (so $1-r$ the fraction in the support set), and $t$ is the number of iterations of the algorithm. In what follows, we write $\text{API}_\pi$ for the best performing policy from $t$ iterations of the algorithm, including the initial policy, which ensures that we only consider policies which improve as $t$ increases. The trade-off between different choices of the parameters will be explored in §6.2, where we will give a recommendation for their selection.

We now present an example to illustrate the algorithm. Consider a $k = 2$ example with the parameters $\lambda_1 = 2.5$, $\lambda_2 = 3$, $\mu_1 = 3.5$, $\mu_2 = 4$, $\theta_1 = 0.75$, $\theta_2 = 2.5$, $R_1 = 2.5$, $R_2 = 1.7$. We use truncation levels $N_1 = N_2 = 20$ throughout. Please note that for this example the $R\mu$ rule gives priority to class 1, and the $R\mu\theta$ rule gives priority to class 2. We use algorithms of the form $\text{API}(R\mu\theta, 45, m, 32/45, 1)$ to construct policies. Figure 1 illustrates the selection and interpolation stages of the algorithm for the case $m = 10^5$. Figure 1(a) shows $S_{\text{sel}}$, with anchor states shown as diamonds and support states as circles. Figure 1(b) shows the interpolated bias estimates over the entire state space. Although not shown here, the surfaces of exact biases $\omega^\pi$ and simulated bias estimates throughout $S$ closely resemble the interpolated surface, capturing its shape and curvature well, especially so around the anchor set. Figure 2 shows the actions taken in each state by the optimal policy for this example, along with the actions resulting from use of the above algorithm with $m$ set at $10^3$, $10^4$, and $10^5$. We observe that as $m$ increases, the corresponding policies approach more closely the switching curve structure of the optimal policy.

## 5. An Upper Bound on Achievable Rewards

To evaluate heuristic policies when the optimal solution in (7) is not available, we derive an upper bound for the long-run reward rate. For a given feasible policy, if $x_i$ represents the implied fraction of time the server spends on class $i$ customers, then

$$\sum_{i=1}^{k} R_i \mu_i x_i \tag{16}$$

is the long-run reward rate for the feasible policy. To compute an upper bound for the optimal long-run reward rate, we formulate a linear program with the variables $x_i \geq 0$, $1 \leq i \leq k$, and the objective function to maximize (16), subject to the constraint $\sum_{i=1}^{k} x_i \leq 1$. The key to get a tight upper bound is to impose additional constraints on the $x_i$ so that the resulting optimal

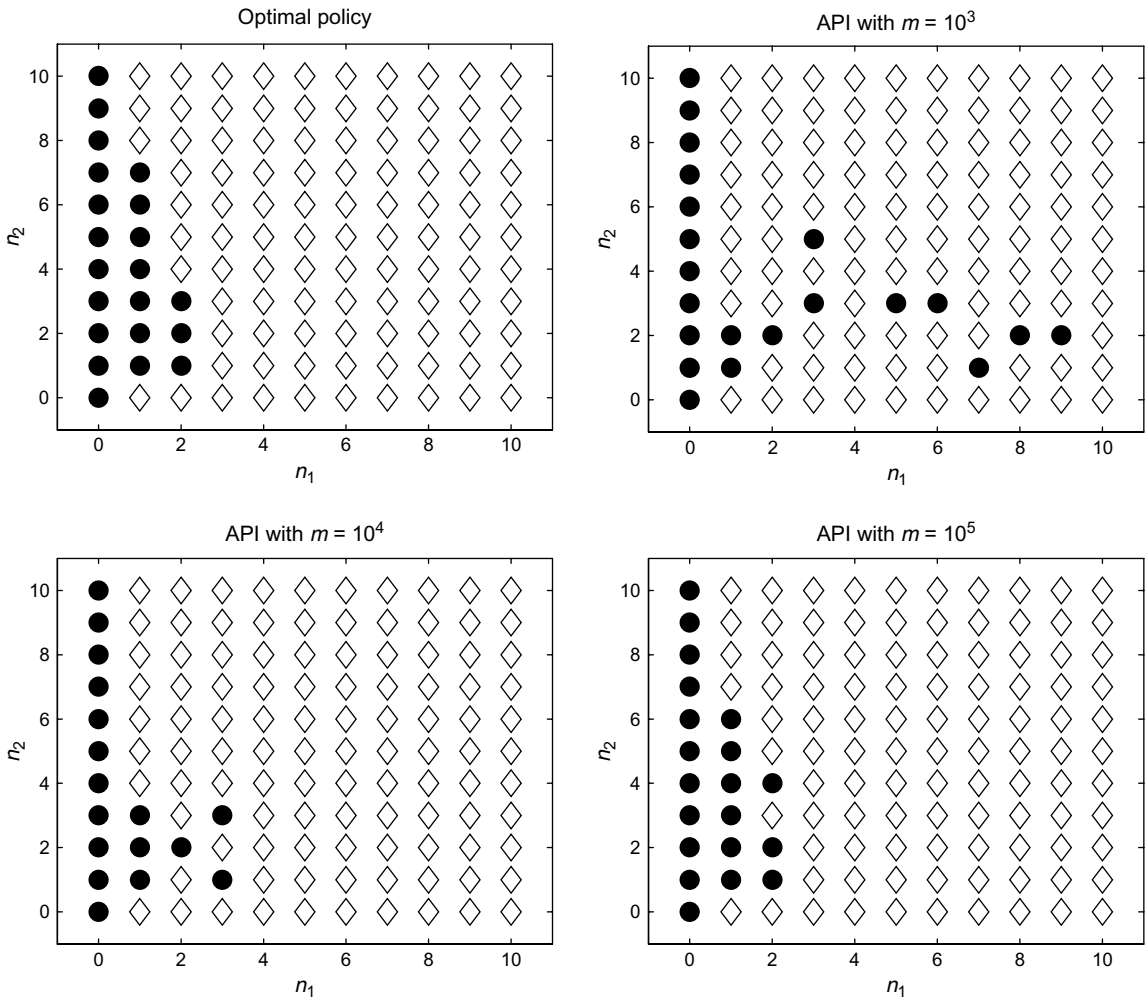


**Figure 2** **Class 1 (Diamonds) and Class 2 (Circles) Actions in Each State Under Various Policies in the Example System**

policies come as close as possible to those implied by a feasible policy.

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

$$A_{\{i\}} = 1 - \left[ \sum_{n=0}^{\infty} (\lambda_i)^n \left\{ \prod_{m=1}^{n} (\mu_i + m\theta_i) \right\}^{-1} \right]^{-1}, \quad 1 \le i \le k.$$

We can add $x_i \le A_{\{i\}}$ as a constraint in the aforementioned linear program, $1 \le i \le k$, or a total of $k$ constraints.

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

Computing $A_T$ when $|T| \ge 4$ is computationally infeasible, but we can still impose constraints derived from relaxed systems. To do so, we create a single fictitious class by aggregation and relaxation of the customer classes in $T$. Denote the arrival, service, and abandonment rates of this fictitious class by $\lambda = \sum_{i \in T} \lambda_i$, $\mu = \min_{i \in T}\{\mu_i\}$, and $\theta = \min_{i \in T}\{\theta_i\}$, respectively. Since the server can only be busier with this fictitious class, the long-run fraction of time that the server is busy in this relaxed system is a legitimate upper bound for $\sum_{i \in T} x_i$.

Taking this idea further, we could improve this upper bound by formulating a number of two-class MDPs. Divide customer classes in $T$ into two groups and aggregate the classes in each group into a fictitious class, as before. We then use DP value iteration to compute the maximal fraction of time that the server is busy dealing with these two fictitious classes. When $|T| \ge 4$, we write $B_T$ for the tightest upper bound for

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

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

$$\begin{aligned}
\max\ & \sum_{i=1}^{k} R_i \mu_i x_i \\
\text{subject to}\ & \sum_{i=1}^{k} x_i \le 1, \quad x_i \ge 0,\ 1 \le i \le k; \\
& \sum_{i\in T} x_i \le A_T \quad \text{for all } T \in \mathcal{S}_{k'},\ k' = 1,2,3; \\
& \sum_{i\in T} x_i \le B_T \quad \text{for all } T \in \mathcal{S}_{k'},\ 4 \le k' \le k.
\end{aligned}$$

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

## 6. Numerical Study

In this section, we conduct extensive numerical experiments to assess the impact and design of our API method as well as the performance of a range of heuristics that includes our suite of priority policies. Section 6.1 uses a numerical study based on cases with two customer classes to explore design choices for our API heuristics. We assess, *inter alia*, the relative performance of the candidate initializing priority rules $R\mu$ and $R\mu\theta$ as well as testing different choices of parameters for our API method. This test yields a recommended API policy that we denote rAPI. Using numerical studies based on cases with three and five customer classes, §6.2 compares the performance of rAPI with that of other heuristics. Section 6.3 contains a brief discussion of the computational burden of developing rAPI and the upper bound discussed in §5.

### 6.1. Selecting Parameters for the API Algorithm

To explore the trade-off between different choices of parameters for our API algorithm, we test the algorithm on systems with $k = 2$ customer classes. Problems were randomly generated to reflect a wide range of conditions with regard to (1) the length of job lifetimes in relation to service times (reflected in the categorization $A, B, C$ in (17c)–(17e)) and (2) the traffic intensity or workload in the corresponding system without abandonments. There are three categories of traffic—light, moderate, and heavy—as determined by the value of $\rho = \sum_{i=1}^{k}(\lambda_i/\mu_i)$; see (17f)–(17h). For all nine combinations of $A, B, C$ with the traffic categorization light, moderate, and heavy, 500 problems were generated at random. Parameters were sampled as follows:

$$\mu_i \sim U[0.2, 5] \quad \text{(all cases)}; \tag{17a}$$

$$\lambda_i \sim U[0.2, 5] \quad \text{(all cases)}; \tag{17b}$$

$$\theta_i^{-1}\mu_i \mid \mu_i \sim U[0.5, 2] \quad \text{(short lifetimes, A)}; \tag{17c}$$

$$\theta_i^{-1}\mu_i \mid \mu_i \sim U[5, 10] \quad \text{(moderate lifetimes, B)}; \tag{17d}$$

$$\theta_i^{-1}\mu_i \mid \mu_i \sim U[20, 200] \quad \text{(long lifetimes, C)}; \tag{17e}$$

$$\rho \in [0.6, 0.8] \quad \text{(light traffic)}; \tag{17f}$$

$$\rho \in [0.9, 1.1] \quad \text{(moderate traffic)}; \tag{17g}$$

$$\rho \in [1.2, 1.4] \quad \text{(heavy traffic)}; \tag{17h}$$

In the parameter generation, $\mu_i$ and $\lambda_i$ were sampled according to (17a) and (17b) by means of a rejection algorithm until a desired $\rho$ condition (17f)–(17h) was met. An additional rejection step ensured that the $R\mu\theta$ and $R\mu$ rules of each parameter set were distinct; otherwise, all parameters were resampled. In all cases, rewards were sampled as follows: $R_2 \sim U[1, 3]$ and $R_1 R_2^{-1} \mid R_2 \sim U[1.25, 2]$. To compute the optimal policy, we use DP value iteration by truncating the state space at $N_i = 40$ for each class $i$ with case A, and $N_i = 60$ with cases B and C, as discussed in §2.

Table 1 reports the numerical results with $k = 2$ customer classes. In comparing the $R\mu\theta$ and $R\mu$ rules, please recall that we use the descriptors light, moderate, and heavy as shorthand for ranges of the traffic intensity $\rho$. The actual volume of traffic in the system will also be strongly influenced by the abandonment rate $\theta$, with case C (small $\theta$) yielding higher volumes than case A (large $\theta$). Hence, although the $R\mu\theta$ rule performs very well in the case {C, light}, as is consistent with Theorem 1, it performs poorly in the heaviest traffic case of all, namely {C, heavy}. Its performance under A is less variable than under C because larger abandonment rates act as a moderator on traffic levels, though it still performs best under A when $\rho$ is small. The $R\mu\theta$ rule clearly outperforms the $R\mu$ rule when $\rho$ is small and $\theta$ not too large, whereas $R\mu$ is the better policy when $\rho$ is large, increasingly so as $\theta$ declines in value and the traffic levels increase. It is worth noting that at least one of these two priority rules delivers a median performance less than 1% suboptimal across all cases, so they complement each other well.

Table 1 also reports the performance of the policy PI-$R\mu\theta$, which is derived from exact application of a single PI step to the $R\mu\theta$ rule. The fact that the PI-$R\mu\theta$ is nearly optimal shows the promise of the

**Table 1 Percentage Suboptimalities in $k = 2$ Class Systems of Various Traffic and Abandonment Level Combinations**

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

*Notes.* Variations of API using $R\mu\theta$ as the initial policy with $n = 45$ are denoted by $(t, m)$, where $t \in \{1, 2, 3\}$ and $m \in \{m_1, m_2, m_3\}$ with $m_1 = 10^3$, $m_2 = 10^4$, $m_3 = 10^5$. In the last column, we report the percentage of the upper bound above the optimal policy.

API method, if the bias function can be approximated satisfactorily. The policy GADL—due to Glazebrook et al. (2004), with GADL referring to the paper's four coauthors—requires much computational effort but is typically not as good as the better between the $R\mu\theta$ and $R\mu$ rules.

The next seven columns in Table 1 explore the trade-off between different choices of parameters of the API$(R\mu\theta, n, m, r, t)$ proposed in §4.2. We use $n = 45$ selected states throughout, with $r = 32/45$. As one would expect, increasing $m$ and $t$ improves performance whilst increasing computational effort. For a given level of computational effort, the strong performance of PI-$R\mu\theta$ suggests that the policy API$(\pi, n, m, r, t)$ may perform better with a single, more detailed iteration. Based on our wider experimentation, there appears to be a degree of indifference in performance between multiple, less detailed iterations and a single, more detailed iteration. Further, and unsurprisingly, a strongly performing initial policy $\pi$ usually improves performance. Based on these observations, to choose parameters in API$(\pi, n, m, r, t)$ for general $k$ class systems, we recommend $t = 1$ and a large value of $m$ ($10^5$, say) and allow $n$, the number of selected states, to scale roughly linearly with $k$ so that $20k \le n \le 25k$. To choose the initial policy $\pi$ in general, we first run the pairwise-swapping mechanism in §3.3 on the $R\mu\theta$ rule and on the $R\mu$ rule, separately. It turns out that in all numerical tests in this section, the final orderings are the same, which we label PaS. The initial policy $\pi$ in the API

method is thus set to be the best performing among $R\mu\theta$, $R\mu$, and PaS. We shall denote our recommended API policy by rAPI. As seen in Table 1, the rAPI is nearly optimal in all our $k = 2$ class cases. The final column in Table 1 reports the quality of the upper bound presented in §5. The upper bound typically sits about 1%–2% above the optimal value.

### 6.2. Comparing the rAPI and Other Heuristics

This section compares the rAPI and other heuristics in systems with $k = 3$, 5 customer classes. Problem parameters were again generated according to (17a)–(17h), along with suitable rejection algorithms. We now use $R_i \sim U[1, 4]$ for sampled rewards. For each lifetime/traffic combination, 100 problems were generated at random.

Table 2 reports the performance of various service policies against the optimal solution for systems with $k = 3$ customer classes. The rAPI was constructed with $t = 1$, $m = 10^5$, $n = 75$, and $r = 52/75$. As seen in the table, the rAPI delivers near-optimal performance in all cases, which reaffirms the strength of policies based on a single, well-estimated (but nonetheless approximate) PI step applied to a well-chosen priority policy. Table 2 also shows that a naive heuristic that always serves the longest queue (labeled SLQ) can perform poorly. In most cases, PaS improves on both $R\mu\theta$ and $R\mu$, though this is not universal. The quality of the upper bound for $k = 3$ customer classes is similar to that for $k = 2$ customer classes.

Table 3 reports the performance of various service policies against an upper bound on the optimal solution, as discussed in §5, for systems with $k = 5$ customer classes. Since value iteration is not computationally feasible, the gain of each heuristic is estimated as the mean of 1,000 Monte Carlo realizations, which is then compared with the upper bound presented in §5. The policy rAPI was constructed with $t = 1$, $m = 10^5$, $n = 100$, and $r = 69/100$. As seen in Table 3, the relative quality among $R\mu\theta$, $R\mu$, PaS, and rAPI is consistent with that in Table 2. The PaS typically improves $R\mu\theta$ and $R\mu$, and then the rAPI further improves the PaS, although the improvement, on average, is rather marginal. The rAPI is the best-performing policy in all cases, and its median performance is within 4% of the upper bound derived in §5. Although it is difficult to judge how the rAPI compares with the optimal policy, the fact that the rAPI is much closer to the optimal value than it is to the upper bound in Tables 1 and 2 suggests that the figures in Table 3 are a conservative statement of where the policies stand in relation to the optimal value.

Whereas our numerical experiments in Tables 1–3 show that the suite of priority policies ($R\mu$, $R\mu\theta$, and PaS) generally perform very well, and in several cases the API method offers only marginal improvement on average, it is not always the case. To conclude our numerical study, we offer one example where the improvement of the rAPI method is substantial. Consider a $k = 3$ example in which the class parameters $(\lambda_j, \mu_j, \theta_j, R_j)$ are given for classes 1–3 by $(\lambda, 3, 0.1, 5)$, $(5\lambda/3, 5, 1, 2)$, and $(4\lambda/3, 4, 5, 1)$, respectively. With these parameters we have $\lambda = \rho$, and the $R\mu\theta$ rule gives class ordering 321, and the $R\mu$ rule gives 123. As seen in Figure 3, the $R\mu\theta$ rule performs well for small $\rho$, whereas the $R\mu$ rule performs well for large $\rho$, which coincides with intuition. For intermediate $\rho$ values, however, there is a substantial gap between the suite of priority policies ($R\mu$, $R\mu\theta$, and PaS) and the rAPI method (with $t = 1$, $m = 10^5$, $n = 75$). In particular, when $\rho = 1.7$, the $R\mu$ rule is 4.26% suboptimal, but the rAPI is 0.04% suboptimal.

**Table 2 Percentage Suboptimalities in $k = 3$ Class Systems of Various Traffic and Abandonment Level Combinations**

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

*Notes.* In the last column, we report the percentage above the optimal policy of the upper bound.

**Table 3 Percentage Below the Upper Bound in $k = 5$ Class Systems of Various Traffic and Abandonment Level Combinations**

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

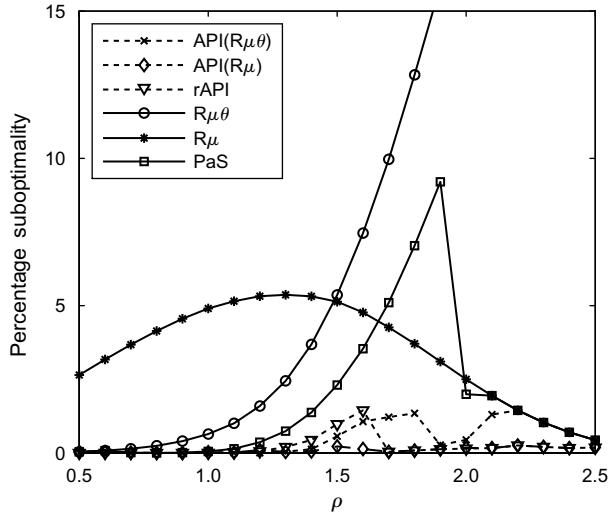


**Figure 3 Percentage Suboptimality for Six Heuristics for the Example at the End of §6.2**

*Note.* The variations of API shown denote the initializing policy.

### 6.3. Computational Time for rAPI and the Upper Bound

Table 4 summarizes the time needed to compute the rAPI heuristic. Please note that the algorithm was coded in the C programming language and carried out on a High Performance Computing cluster, with a typical node specification of 2.26 GHz Intel Xeon E5520 processor. Unsurprisingly, the computational burden grows with the number of customer classes $k$. Recall that the number of selected states $n$ used in the approximate PI step grows roughly linearly in $k$. Further, as $k$ increases, the balance of computational effort moves toward the sampling stage of the API algorithm and, within the sampling stage, toward the estimation of bias at the support states. These trends particularly reflect the nature of the growth in the mean times $t^{\pi}(\mathbf{n})$ for a single simulation run during the estimation of the bias $\omega^{\pi}(\mathbf{n})$. The mean computation times for the upper bound in each problem are in the order of 10, 400, and 3,000 seconds for systems with two, three, and five customer classes, respectively. This growth in the computational burden reflects the growth in the number of MDP subproblems that must be solved through DP methods to generate the constraints

**Table 4 Mean Computation Time (Secs) Needed to Generate the rAPI Policy in Each Problem of Various $k$ Class Systems**

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

*Note.* Also shown are the mean proportions of overall computation time spent on the pilot study, sampling of the anchor set, and sampling of the support set.

for the linear program in §5, when the number of customer classes increases.

## 7. Conclusions

This paper studies the problem of developing effective service policies for multiclass queues with abandonment in a computationally efficient manner. Whereas it is known in the literature that the server can do well simply by maximizing the instantaneous reward rate using the $R\mu$ rule in heavy-traffic systems, we show that in light-traffic systems it becomes important to take into account the abandonment rate using the $R\mu\theta$ rule. We also consider an approximate policy improvement algorithm to improve a given service policy. Our numerical study shows that the $R\mu\theta$ rule complements the $R\mu$ rule, and applying a pairwise-swapping mechanism to each often yields an even stronger priority policy. The best priority policy that we compute is often nearly optimal; in the cases where it is not, the approximate policy improvement algorithm invariably substantially tightens up either the optimality gap or the gap relative to the upper bound that we compute.

There are several interesting future research directions. If a customer's lifetime in the system does not follow an exponential distribution, then the server needs to take into account the arrival time of each customer when selecting which customer to serve. The problem will be further complicated if the time needed for the server to switch to another queue cannot be ignored. Still another research direction is to allow the customers to be active decision makers. For example, in military applications, an adversarial customer may not select a queue to join at random as assumed in our model but instead chooses the one that maximizes the expected gain from his standpoint.

### Supplemental Material

Supplemental material to this paper is available at http://dx.doi.org/10.1287/ijoc.2015.0675.

### Acknowledgments

The authors thank the associate editor and two anonymous referees for the many helpful comments that substantially improved the presentation of this paper.

### References

Argon NT, Ziya S, Righter R (2008) Scheduling impatient jobs in a clearing system with insights on patient triage in mass casualty incidents. *Probability Engrg. Informational Sci.* 22(3):301–322.

Ata B, Tongarlak MH (2013) On scheduling a multiclass queue with abandonments under general delay costs. *Queueing Systems* 74(1):65–104.

Atar R, Giat C, Shimkin N (2010) The $c\mu/\theta$-rule for many-server queues with abandonment. *Oper. Res.* 58(5):1427–1439.

Ayesta U, Jacko P, Novak V (2011) A nearly-optimal index rule for scheduling of users with abandonment. *INFOCOM, 2011 Proc. IEEE* (IEEE, New York), 2849–2857.

Bassamboo A, Harrison JM, Zeevi A (2005) Dynamic routing and admission control in high-volume service systems: Asymptotic analysis via multi-scale fluid limits. *Queueing Systems* 51(3): 249–285.

Bertsekas DP (2012) *Dynamic Programming and Optimal Control: Approximate Dynamic Programming*, Vol. 2 (Athena Scientific, Belmont, MA).

Down DG, Koole G, Lewis ME (2011) Dynamic control of a single-server system with abandonments. *Queueing Systems* 67(1):63–90.

Garnett O, Mandelbaum A, Reiman M (2002) Designing a call center with impatient customers. *Manufacturing Service Oper. Management* 4(3):208–227.

Gaver DP, Jacobs PA, Samorodnitsky G, Glazebrook KD (2006) Modeling and analysis of uncertain time-critical tasking problems. *Naval Res. Logist.* 53(6):588–599.

Gittins J, Glazebrook KD, Weber RR (2011) *Multi-Armed Bandit Allocation Indices* (John Wiley & Sons, Chichester, UK).

Glazebrook KD, Niño-Mora J (2001) Parallel scheduling of multiclass $M/M/m$ queues: Approximate and heavy-traffic optimization of achievable performance. *Oper. Res.* 49(4):609–623.

Glazebrook KD, Punton EL (2008) Dynamic policies for uncertain time-critical tasking problems. *Naval Res. Logist.* 55(2):142–155.

Glazebrook KD, Ansell PS, Dunn RT, Lumley RR (2004) On the optimal allocation of service to impatient tasks. *J. Appl. Probab.* 41(1):51–72.

Harrison MJ, Zeevi A (2004) Dynamic scheduling of a multiclass queue in the Halfin-Whitt heavy traffic regime. *Oper. Res.* 52(2):243–257.

Howard R (1960) *Dynamic Programming and Markov Processes* (MIT Press, Cambridge, MA).

Jouini O, Pot A, Koole G, Dallery Y (2010) Online scheduling policies for multiclass call centers with impatient customers. *Eur. J. Oper. Res.* 207(1):258–268.

Kim J, Ward AR (2013) Dynamic scheduling of a $GI/GI/1+GI$ queue with multiple customer classes. *Queueing Systems* 75(2):339–384.

Klimov GP (1974) Time-sharing service systems I. *Theory Probab. Its Appl.* 19(3):532–551.

Klimov GP (1978) Time-sharing service systems II. *Theory Probab. Its Appl.* 23(2):314–321.

Krishnan KR (1987) Joining the right queue: A Markov decision-rule. *Proc. 26th IEEE Conf. Decision Control*, Vol. 26 (IEEE, New York), 1863–1868.

Larrañaga M, Ayesta U, Verloop M (2014) Index policies for a multi-class queue with convex holding cost and abandonments. *2014 ACM Internat. Conf. Measurement Model. Comput. Systems* (ACM, New York), 125–137.

Li D, Glazebrook KD (2010) An approximate dynamic programing approach to the development of heuristics for the scheduling of impatient jobs in a clearing system. *Naval Res. Logist.* 57(3): 225–236.

Lin KY, Kress M, Szechtman R (2009) Scheduling policies for an antiterrorist surveillance system. *Naval Res. Logist.* 56(2):113–126.

Niederreiter H (1978) Existence of good lattice points in the sense of Hlawka. *Monatshefte für Mathematik* 86(3):203–219.

Powell MJD (1987) Radial basis functions for multivariable interpolation: A review. Mason JC, Cox MG, eds. *Algorithms for Approximation* (Clarendon Press, Oxford, UK), 143–167.

Powell MJD (1999) Recent research at Cambridge on radial basis functions. Müller MW, Buhmann MD, Mache DH, Felten M, eds. *New Developments in Approximation Theory*, Vol. 132 (Birkhäuser Verlag, Basel, Switzerland), 215–232.

Powell WB (2011) *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, 2nd ed. (John Wiley & Sons, Hoboken, NJ).

Puterman ML (1994) *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (John Wiley & Sons, Hoboken, NJ).

Tijms HC (1994) *Stochastic Models: An Algorithmic Approach*, Wiley Series in Probability and Mathematical Statistics (John Wiley & Sons, Hoboken, NJ).

Verloop M (2014) Asymptotic optimal control of multi-class restless bandits. Technical report hal-00743781, Université de Toulouse, France.