Finite-capacity Multi-class Production Scheduling with Set-up Times

Eungab Kim* **and** **Mark P. Van Oyen****

***Joseph L. Rotman School of Management**
University of Toronto
Toronto, Ontario, Canada M5S 3E6

****Department of Industrial Engineering and Management Sciences**
Northwestern University
Evanston, IL 60208-3119

Forthcoming in *IIE Transactions, 1999*

Corresponding Author: Mark P. Van Oyen
Phone: 847-491-7008
FAX: 847-491-8005
E-mail: vanoyen@nwu.edu
URL http://primal.iems.nwu.edu/~vanoyen/

March 31, 1998
Revised September 14, 1998

Finite-capacity Multi-class Production Scheduling with Set-up Times

Eungab Kim and Mark P. Van Oyen

Abstract

We treat the scheduling of a single server in a finite-buffer capacity, multi-class, make-to-order production system subject to inventory holding costs, set-up times, and customer rejection costs. We employ theoretical and numerical analysis of a Markov decision process model to investigate the structure of optimal policies and the performance of heuristic policies. We establish the monotonicity of optimal performance with respect to the system parameters. Based on our insights, we provide a heuristic policy called the Capacitated Modified Index Rule (CMIR) for capacitated scheduling with customer loss penalties. The CMIR heuristic can easily be precomputed and stored for real-time control. Numerical benchmarking with respect to the optimal performance as well as an existing heuristic suggests that CMIR is very effective.

Keywords: *Polling system; Finite buffer; Capacitated production scheduling; Markov decision process model; Near-optimal heuristic scheduling*

1. Introduction

We focus on production systems that have multiple models (which we will refer to as job types), a single production line (i.e., a single server), significant set-up times, and small work in process (WIP) buffer sizes. In this context, we design a very effective heuristic scheduling policy, called the Capacitated Modified Index Rule (CMIR), for production systems that are limited not only by the rate of the production process and its inherent randomness, but also by limitations on WIP buffer space. Systems which include features and issues of this sort arise in many applications. Perhaps the most obvious application is to a flexible workstation or cell that processes several job types within a larger production system. In some cases, it may be possible to model an entire line or subsystem, provided only one job type can be served at a time. Examples such as this with significant set-up times occur in the operation of large metal rolling operations that produce sheet metal from ingots. Some communication systems and subsystems such as network switches, pagers, and multiple-access mechanisms contain similar issues and features. The treatment of these scheduling issues for systems with small buffers is our research contribution in this work. Consider the class of make-to-stock production operations that predict job completion times and quote them to the customer. There will be a limit as to how long a customer is willing to wait for the finished product. Beyond that limit, customers will not "enter" the system. One way to address this issue is through models with finite buffers. A more widespread motivation is the fact that many operations have limited floorspace, conveyers, or WIP storage buffers that place strict limits on the amount of raw material of semi-finished goods that can be queued up at a station. These are some of the typical problems that make this class of models important. Although analytical approaches to production and control decisions have traditionally assumed infinite buffer capacities for tractability, we provide insights into the effects that finite-buffers can have on the structure of optimal scheduling policies and devise an effective heuristic policy.

In a make-to-order environment, we consider a machine or production module that may be shared across N product types and must be dedicated to only one product type at any instant. A model for this may take the form of a single-server with N input queues, one for each job type (model). Orders of type n arrive at rate λ_n, and the service rate for type n is denoted μ_n. To provide an incentive to minimize average weighted WIP levels, we include linear holding costs at rate c_n for each unit of time that jobs of type n wait in the system. By Little's law (see Ross [20]) the holding costs provide an incentive to minimize weighted cycle times/response times. In addition, the question of how to sequence the production process is complicated by set-up times with mean $\bar{D}_n$ that are incurred when the server switches to job type n from serving a different type. The class of systems described thus far as sequential production at a shared resource is a common problem that has been treated under the rubric of polling systems. Polling systems have received considerable attention in the case of infinite buffer capacity (see Browne and Yechiali [2], Levy and Sidi [12], Takagi [22], Duenyas and Van Oyen [3], Koole [10], and Reiman and Wein [16]). Many intuitively reasonable policies exist and are being used in a wide variety of applications (see Levy and Sidi [12] and Takagi [22]). These works address infinite-buffer models to limit the complexity of the resulting policy and the analysis required. Even so, a thorough understanding of the structural properties of optimal policies is not presently available. Most of the literature focuses on the performance analysis of ad hoc policies of broad applicability.

In contrast, we focus on systems with limited buffers of size M_n for type n, which operate as loss systems. For systems with finite buffers and inventory holding costs, we have seen that there is an incentive to reject customers based on the state of the system and the particular parameters. In some cases with high utilizations or long set-up times, an optimal policy may even forever abandon a queue. In most manufacturing and communication applications, revenue is collected based on the number of jobs processed among other factors. For this reason, it is usually appropriate to include a job rejection penalty that represents the revenue lost by a rejected customer or a measure of the ill will generated. Under the assumption that high throughputs are desirable and unmet demands are costly to the operation, we charge a lump sum customer rejection cost $S_n \geq 0$ for each loss of type n. Our objective is to understand the structure of dynamic scheduling policies under the long-run average cost per unit time criterion and to capture our insights in the form of a practicable heuristic policy that can be tailored to specific applications. For the sake of mathematical tractability and simplicity, we assume that customers/demands for each model type are well approximated as independent Poisson arrival processes with rate λ_n, and job service times (set-up times) are exponential with rate μ_n ($\bar{D}_n^{-1}$, respectively) for type n. Thus, we have a problem of scheduling a single server in a multi-class $\mathrm{M/M/1/}\{M_n\}_1^N$ queueing system.

Analytically, we show that there are clear monotonic relationships between the problem parameters and the optimal performance. For example, we show that reductions in the set-up time cannot

degrade system performance, provided an optimal policy is employed. This stands in contrast to cases in which service and set-up distribution may take general distributions and suboptimal policies are employed. In such cases, Sarkar and Zangwill [21] presents paradoxical cases in which reductions in both the mean and variance of the set-up time may result in degraded performance.

We offer insights into the structure of optimal policies with finite buffers, which we capture in the design of CMIR. In manufacturing contexts, CMIR is of value for real-time machine control. It is particularly important that the heuristics we derive are based upon comparisons of closed-form functions of the basic problem parameters. The number of comparisons per state is equal to the number of job classes, so the computational complexity is roughly on the order of the complexity of one value iteration step in a dynamic programming algorithm. For systems with a state space that is not too large, CMIR can be precomputed, stored on a hard drive, and applied in real time without further computation. If the state space is too large to store in memory, the heuristic is simple enough that the computations can be made in real time for many manufacturing problems. In addition, CMIR is an analytical heuristic based on mean system parameters, and thus it can be implemented adaptively on-line to self-tune to changing demand rates and other parameters. It is our hope that the insights gained here will remain useful in addressing other systems for which exponential distributions are inappropriate. We point out that the approach used in constructing our heuristics is applicable to distributions other than exponential.

Variations on the model described above have been well investigated in the literature for the case of infinite-buffers. If there are no switching penalties, the $c\mu$ rule (also known as Smith's rule or weighted shortest processing time (WSPT)) that always serves the available job with the largest $c\mu$ index, is known to be optimal, provided the buffers are infinite (see Varaiya et al. [26] and Walrand [27]). Work has also been done to extend the results to allow switching penalties. Hofri and Ross [5] and Liu et al. [13] partially characterize the scheduling of homogeneous systems. For heterogeneous systems, Duenyas and Van Oyen [3], [4] and Koole [10] prove a partially characterization the optimal policy and provide heuristic approaches. Reiman and Wein [16] gives heuristic scheduling policies under a heavy traffic assumption. Olsen [14] develops heuristics that consider the mean waiting time (equivalently, unit holding costs) and also waiting time variance and outer percentiles of waiting time.

When the queue capacity constraint is introduced in scheduling multi-class systems, analysis becomes complex even without switching penalties and rejection penalties. Few results have been reported for problems of this type in the optimal control of queues. Rosberg and Kermani [18] treats a scheduling problem that maximizes the long-run average reward in a finite queueing model without switching penalties. Under a light traffic assumption, they proposed a threshold policy called the "overflow scheduling policy," which they showed to be asymptotically optimal. If all queues are below a threshold, it serves a queue with the highest $c\mu$ index among non-empty queues;

otherwise, it serves a queue with the highest $\lambda c\mu$ index among queues which have jobs beyond their threshold values. For some problem instances, their heuristic never serves queues with low $c\mu$ indices. Because their heuristic is derived under the assumption that all arrival rates are close to zero, the CMIR heuristic we develop will differ from the idea that drives the $\lambda c\mu$ portion of their rule.

Our presentation is outlined as follows. In Section 2 we provide a careful formulation of the problem and its variants. In Section 3 we provide a result that gives insight into the appropriate use of numerical approximations for infinite-buffer systems, and we prove the monotonicity of system performance with respect to all the problem parameters, except for buffer size. In Section 4 we survey known structural properties of optimal policies in systems with infinite capacity, and we develop the CMIR heuristic policy under the assumption that holding costs and set-up times are significant. We note our conclusions in Section 5.

2. Problem Formulation

A single server is to be allocated to jobs in a system of parallel queues indexed by $\mathcal{N} \stackrel{\Delta}{=} \{1, 2 \ldots, N\}$, with queue n fed by Poisson arrivals with rate $\lambda_n > 0$ (independent of all other processes). Successive services in node n are independent and identically distributed (i.i.d.) exponential random variables with mean μ_n^{-1} $(0 < \mu_n^{-1} < \infty)$ and are independent of all else. We focus attention on systems with a finite buffer M_n at queue $n \in \mathcal{N}$. With $\mathbb{R}^+$ denoting the nonnegative reals and $t \in \mathbb{R}^+$, let $x_n(t) \in \{0, 1, \ldots, M_n\}$ be the queue length of node n (including any customer of node n in service) at time t. If $x_n(t) = M_n$ and an arrival of type n occurs at time n, then the arrival is simply lost and the state remains unchanged. Let $\rho_n \stackrel{\Delta}{=} \lambda_n/\mu_n$ and $\rho \stackrel{\Delta}{=} \sum_{n=1}^{N} \rho_n$. The scope of the heuristic we develop is limited to $\rho < 1$, which is typical of production systems in an approximately steady state environment.

A switching or set-up time, D_n, is incurred at each instant (including time 0) the server switches to queue n from a different queue to process a job. We assume that successive set-ups for node n require strictly positive periods which are i.i.d., possess an exponential distribution with mean $\bar{D}_n (= 1/d_n)$, and are independent of all else. The system incurs an inventory holding cost at rate c_n cost units per job per unit time as well as a lump-sum rejection cost S_n $(0 \leq S_n < \infty)$ for each customer which arrives to a full queue.

A policy specifies, at each decision epoch, that the server either remains working in the present queue, idles in the present queue, or sets up another queue for service. The class of admissible strategies, $\mathcal{U}$, is taken to be the set of non-preemptive, stationary, nonrandomized, greedy (that is, never idling in a nonempty queue) policies that are functions of the current state, based on perfect observations of the queue length processes (see page 103 of Ross [19] and pages 38, 42, and 152 of Kumar and Varaiya [11] for terminology). By non-preemptive, we mean that service times and set ups cannot be interrupted once initiated. Because of the non-preemptive assumption, the set of

decision epochs is assumed to be the set of all arrival epochs to an idle server, service completion epochs, and set-up completion epochs.

We describe the state under policy π by the vector $x(t) = (x_1(t), \ldots, x_N(t), \delta(t), n(t))$, where $n(t)$ denotes that the server is located at node $n(t)$ at time t and $\delta(t)$ is as follows. We set $\delta(t) = 0$ if the set-up of node $n(t)$ is not complete at time t, $\delta(t) = 1$ if the set-up of node $n(t)$ is complete but the service of current job at $n(t)$ is not complete, and $\delta(t) = 2$ if both set-up and service of node $n(t)$ are complete. Let the action space be $\mathcal{A} = \{1, \ldots, N\}$. Suppose at a decision epoch, t, the state is $x(t) = (x_1, \ldots, x_N, \delta(t), n(t))$. Action $A(t) = n \in \mathcal{A}$, where $n \neq n(t)$, causes the server to set up node n. Action $A(t) = n(t)$ results in the service of a job in $n(t)$ if $x_{n(t)}(t) > 0$; otherwise, idle in the current queue until the next arrival to the system. No other actions are possible. Thus, the state space is $\mathcal{S} \triangleq \Pi_{n=1}^{N} \{0, 1, \ldots, M_n\} \times \{0, 1, 2\} \times \{1, \ldots, N\}$.

Let $\{n^\pi(t) : t \in R^+\}$ be the right-continuous process describing the location of the server at time t under policy π. Define $\Delta_n^\pi(T)$ to be the set of random instances on $[0, T]$ in which the buffer is full ($x_n(t) = M_n$) and an arrival is rejected under π. The average cost per unit time under policy π, $\bar{J}_\pi$, can be expressed as

$$\bar{J}_\pi = \limsup_{T \to \infty} \frac{1}{T} E \left\{ \int_0^T \sum_{n=1}^{N} c_n x_n(t) dt + \sum_{n=1}^{N} \sum_{t \in \Delta_n^\pi(T)} S_n \right\} . \tag{2.1}$$

3. On Optimal Policies for Infinite and Finite Buffer Systems

As a starting point, consider the problem in the special case without set-up times. Kim and Van Oyen [6] examines the structure of optimal policies with finite buffers and job rejection penalties. For two-class M/M/1 systems, it proves that there exists a monotonic threshold type of the optimal switching curve, provided that the delay of serving a job is always less costly than the cost of rejecting an arrival. The threshold curve divides the state space into a region in which queue 1 is served and another in which queue 2 is served. The case of a completely symmetric system, including buffers of equal size, and zero holding costs is treated in Kim and Van Oyen [7]. It proves that an optimal policy is simply to *serve the longest queue*.

Problems with set-up times differ from those without. Roughly speaking, as the set-up times increase from zero, the switching threshold separates into two thresholds (set up queue 1 and set up queue 2) and in the region between them it is always optimal to remain in the current queue.

Although we have argued that finite-buffer systems are important in their own right, they can also be viewed as approximations to large or infinite-buffer systems. Because the size of the state space must be limited for numerical computation, we must choose an appropriate manner in which to augment the transition probability matrix (t.p.m.) to account for truncation. That is to say, how does one construct the t.p.m. at the artificial boundaries of the state space caused by truncation so as to obtain a good approximation of the infinite-buffer problem. We have described in the formulation one natural augmentation from among infinite possibilities: we simply reject any

arrival to a full queue. To numerically compute the performance of optimal and heuristic policies, we solved the uniformized (discrete-time) version of the continuous-time Markov chain. We set the system transition rate to $\gamma = \max\{ \max_n(\lambda_n + \mu_n), \max_n(\lambda_n + d_n)\}$ (see Ross [20]). Thus for system transition rate γ, with probability λ_n/γ, the transition from (x, δ, n) to (x, δ, n) (a self-loop) is made due to an arrival to the n^{th} queue when $x_n = M_n$. This choice is sensible, and we will demonstrate that the truncated system provides a lower bound to the performance of the original system with an optimal (possibly nonstationary and randomized) policy. Thus, as we would hope, our computations can be used to guarantee (subject to a tolerance limit, ϵ) that the degree of suboptimality of a heuristic does not exceed a given amount for a particular problem instance. For brevity, all the proofs of this paper are omitted; instead, see Kim and Van Oyen [8].

Theorem 1: *Provided $S_1 = S_2 = \ldots = S_N = 0$, the numerical solution of the problem with truncated queue lengths $M_1, \ldots, M_n$ identifies a stationary, non-randomized policy that provides a lower bound to the achievable performance for the case of infinite queue lengths over the class of randomized, non-stationary Markov policies ($U^{R,NS}$). The same holds true when the system being approximated has buffer sizes $M_1' \geq M_1, \ldots, M_N' \geq M_N$.*

Although insightful structural properties of an optimal policy are extremely difficult to prove in general, we are able to verify that the model constructed behaves with a monotonic performance that is intuitive. Because our model serves as a fundamental model of production, this has important implications for set-up time reduction. A longstanding and highly successful strategy for improving system efficiency aims to reduce set-up times and/or set-up costs. To the surprise of many, Sarkar and Zangwill [21] and Zangwill [28] rigorously demonstrate that this wisdom is simplistic and faulty. Seemingly paradoxical behavior has been demonstrated in a number of cases for which set-up time reductions result in increased inventory levels and degraded performance. It has not been a simple matter to set these interesting cases in a context that resolves their divergence from the conventional wisdom. In a different direction, Righter and Shanthikumar [17] indicates that if set-up times are decreased according to the convex ordering sense, average queue sizes for the original problem will be stochastically larger than those for the new system. This is done for particular policies (gated, cyclic exhaustive) that have been suggested and analyzed in the literature. Van Oyen [23], analyzes the infinite buffer version of the problem and showed that optimal policies are key to the realization of performance improvements by means of set-up time reduction.

In a similar spirit, we turn now to the current problem with capacitated queues and rejection penalties. We show how the paradoxical effects of set-up reduction can be eliminated by employing an optimal policy, provided set-up times are decreased in the usual sense of stochastic ordering (see Kim and Van Oyen [8] for the proof). Having restricted attention to exponential service distributions, this is equivalent to an increase in the service rate. In fact, a monotonicity property exists with respect to every system parameter except the queue lengths.

Theorem 2: *For the full stochastic multi-item production model with finite queue length capacities and having mean set-up time $\bar{D}_i$, mean service time $1/\mu_i$, holding cost c_i, rejection cost S_i, and arrival rate λ_i for queue $i \in \mathcal{N}$, a decrease in some or all of these parameters will result in an equal or lesser expected cost, provided an optimal policy is employed for both the original system and the new one.*

We note that the results presented in this subsection can be extended in a straightforward way to systems with general distributions on processing times and set-up times in the sense that stochastic reductions in these times lead to increased performance as a result of decreased mean WIP levels. It is also interesting to note that Theorem 2 extends to systems which include transition-dependent switching rates, $d_{n,m}$.

The only parameters of the system model not treated in Theorem 2 are the queue capacities $M_1, \ldots, M_N$, which were addressed in Theorem 1 under the restriction of zero rejection penalties. Unfortunately, a moment of reflection indicates that a decrease in M_i increases the steady state probability of buffer overflow and hence increases the rejection penalties. Thus, a blanket monotonicity property does not apply to the queue capacities. A thorough study of the optimal selection of M_i is beyond the scope of this paper, which focuses on how best to use available capacity. We offer some indicative insights into this question for $N = 2$ queues. It is natural to investigate whether or not the optimal cost is a convex function of the buffer sizes, which is a useful property in searching for a globally optimal pair of buffer sizes. Our numerical investigations suggest that such is the case. The *rejection* costs are proportional to the probability of overflow, which we believe is geometrically decreasing in the buffer size. Although excursions into regions of large WIP are rare, they can be expensive and can add significant *holding* cost as the buffers increase (see Theorem 1). We have made the assumption that job rejection penalties should be large relative to holding costs to be consistent with the notion that effective systems will keep rejections rare. Our numerical investigations show that although the objective function can be very sensitive to the buffer levels when buffers are relatively small and rejection events are frequent, the optimal buffer sizes occur in regions for which the objective function is relatively insensitive to the buffer sizes. For example, with $S_1 = S_2 = 100$, $c_1 = c_2 = \mu_1 = \mu_2 = d_1 = d_2 = 1$, and $\lambda_1 = \lambda_2 = 0.45$, the optimal buffer levels are $M_1 = M_2 = 14$. This yields $\bar{J}_{\pi^*} = 11.638$. To assess the proportionality of the buffer levels to the ratio of rejection cost to holding cost, we note that when the rejection costs are reduced by half, the optimal buffer levels are 8 each, while doubled rejection costs result in buffers of size 24 (with optimal costs 9.48 and 13.31, respectively). The same effect is obtained by halving the holding costs and keeping the rejection costs at 100. This is consistent with the following general principal. Given that all the other parameters remain unchanged, comparison of a system with cost parameters S_1, S_2, c_1, c_2 and one with costs scaled by $r \in (0, \infty)$ to get rS_1, rS_2, rc_1, rc_2 results in the following. The objective function is scaled by a factor of r, but the system

dynamics have not changed, and so the optimal policy remains exactly the same. Scaling of costs does not change the optimal buffer sizes.

Note that if rejection costs are zero, it is always better to have the buffers as small as possible (zero for our cost model). When holding costs are small, the holding cost penalty for large queue lengths is small, so the optimal buffer sizing is large so as to minimize rejection penalties. We observed that variation of the holding costs had a more pronounced affect on performance than did the rejection costs.

To indicate the sensitivity of optimal buffer sizes to holding cost and rejection cost, consider the original problem with only one change: $c_2 = 0.5$, half as large as before in queue 2 only. The optimal buffer levels now become $M_1 = 13,\ M_2 = 11$, which is still surprisingly symmetrical. This yields $\bar{J}_{\pi^*} = 9.04$. Next, we modify the original problem to have only one change: only half the rejection cost is applied to station 2. If one assesses the performance of the policy found for the previous case ($M_1 = 13,\ M_2 = 11$), the cost is 10.08. Pursuing optimality to within 10^{-6} precision, we find that any $M_1 \geq 28$ is optimal, provided $M_2 = 7$, and $\bar{J}_{\pi^*} = 9.667712$ (a savings of 4.1% compared with $M_1 = 13,\ M_2 = 11$). Although this problem displays an extraordinary sensitivity to M_1, it is gratifying that the optimal value of M_2 came out to be exactly one half the value of 14 in the original problem. To demonstrate the proportionality of the threshold to the rejection penalty in this particular problem, consider the case with the rejection penalty for station 2 at one fourth its original value: any $M_1 \geq 25$ is optimal, provided $M_2 = 3$, and $\bar{J}_{\pi^*} = 7.617255$. In general, however, this proportionality does not hold and the sensitivity to $M_1,\ M_2$ is difficult to predict.

4. A Heuristic Policy for the Loss Model with Rejection Penalties

In this section, we develop the CMIR heuristic for scheduling a *finite* multi-class M/M/1 queueing system with significant set-up times and $\rho < 1$ that incorporates the boundary effect caused by finite buffers. In Van Oyen et al. [24] and Van Oyen and Teneketzis [25], optimal scheduling policies with set-up penalties are investigated for problems without arrivals. These insights are employed in deriving both CMIR and the heuristic developed in Duenyas and Van Oyen [3], [4].

In the case of infinite buffers ($M_{(\cdot)} = \infty$), it is known (see Duenyas and Van Oyen [3]) that a top-priority queue exists, even with *general* service times with mean μ_n^{-1} where a *top-priority* queue refers to any queue (there may be more than one) that is served in a greedy and exhaustive manner. If $c_i\mu_i \geq c_j\mu_j$ for all $j = 1, \ldots, N$, then there exists a policy for which queue i is a top-priority queue that is optimal within $\mathcal{U}$, the class of admissible policies, under the discounted cost or average cost criterion. For an infinite multi-class M/M/1 queueing system with switching penalties, Duenyas and Van Oyen [3] presents a heuristic which we refer to as the Modified Index Rule (MIR), based on the reward rate indices corresponding to action sequences.

A moment of reflection will reveal that the feature of finite buffers destroys this structure. We

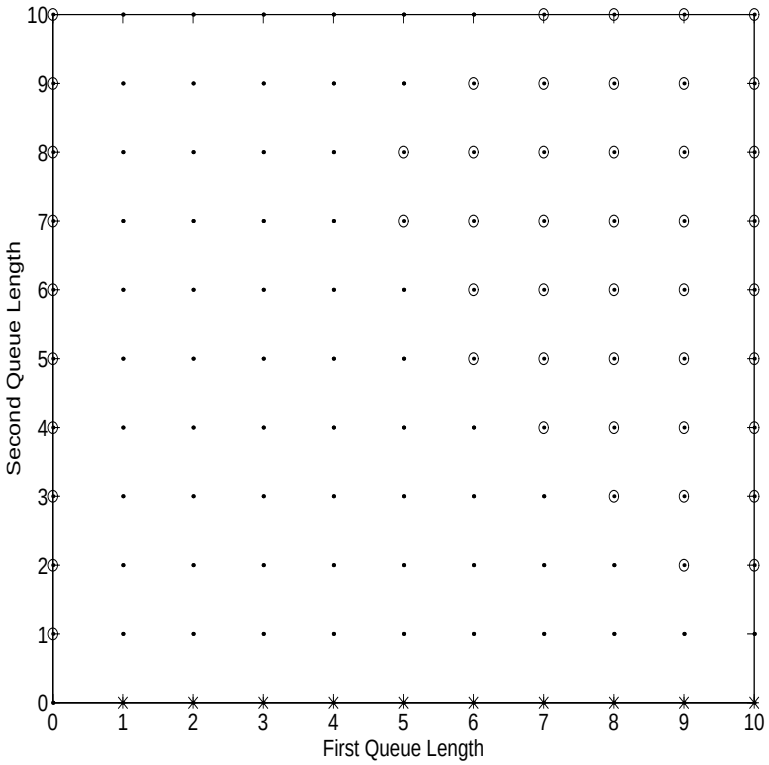

Figure 1: Optimal policy for Example 1.
$\cdot$: Serve the current queue (do not switch).
$\odot$: Serve queue 2 (switch to queue 2 if in 1).
$*$: Serve queue 1 (switch to queue 1 if in 2).

cannot in general expect the exhaustive property to hold for any queue, because as one continues to serve a queue n, the other queues grow and threaten to incur significant boundary effects. In the presence of rejection penalties, it is clear that the anticipation of rejection penalties will cause an optimal policy to attempt to serve such a queue prior to its overflow. Even in the case without rejection penalties, we find that there is an intrinsic boundary effect introduced by job loss (see Kim and Van Oyen [6] for systems without set-up times). This effect can be seen in the circles plotted in the northeast portion of an optimal policy depicted in Figure 1. In this first example (test case 1 of Table 1) we take $M_1 = M_2 = 10$, $S_1 = S_2 = 0$, $c_1 = c_2 = 1$, $\mu_1 = \mu_2 = 2$, $\lambda_1 = 1.0$, $\lambda_2 = 0.5$, and $d_1 = d_2 = 2$. Notice the asymmetric arrival rates, $\lambda_1 = 2\lambda_2$.

By the top-priority queue result of Duenyas and Van Oyen [3], both queues are top-priority queues in problems with $M_1 = M_2 = \infty$, but this is not observed; rather, only queue 2 is a top-priority queue. (Moreover, most problems with finite buffers do not give top-priority to either queue, unlike the infinite buffer case.) In this instance, the region of switching actions from queue 1 to queue 2 is enlarged and queue 2 is exhausted. This phenomenon is intuitively explained as follows. When $x_i = M_i/2$ for $i = 1, 2$, the inventory cost reduction rate (reward rate) for working

in queue i is in an instantaneous sense $c_i(\mu_i - \lambda_i) - c_j\lambda_j$, where j denotes the queue other than i. Because $c_1\mu_1 = c_2\mu_2$ in this example, we see that the instantaneous reward rates are equal in either queue and it is optimal to remain in the current queue. By explicitly including the terms $-c_i\lambda_i$, we call attention to the fact that inventory cost flows into the system with that intensity whenever $x_i < M_i$. The boundary effect occurs when $x_i = M_i$, because this term is absent. When queue i is empty, the instantaneous reward rate is $-c_j\lambda_j$, versus an instantaneous reward rate of $c_j(\mu_j - \lambda_j) - c_i\lambda_i$ if we were to serve in queue j. Thus, the boundary effect at zero causes one to prefer service to idling when the current queue is empty. (It is only the effect of set-up times that can make it optimal to employ idling). Thus, this instantaneous reward rate is helpful in determining the rough structure of an optimal policy. This line of thinking leads us to conclude that if $x_i = M_i$, then it is better to switch to queue $j \neq i$. This, however, is not the case, because the set-up times limit switching. Since $c_1\mu_1 = c_2\mu_2$, $c_1 = c_2$, and $\lambda_1 = 2\lambda_2$, when $x_i = M_i$ for $i = 1, 2$ the incentive to serve queue 2 (and allow queue 1 to overflow) is greater than serving 1: $c_2(\mu_2 - \lambda_2) = 1.5 > c_1(\mu_1 - \lambda_1) = 1.0$. Hence, switching to 2 is more favorable than serving queue 1. From this, we argue that when both queues have nearly full capacity, switching to queue 2 is more cost-effective than staying at queue 1 because the time required until queue 1 hits the boundary is less than the time required until queue 2 hits the boundary. The boundary effect occurs even deep in the interior of the state space, when the switch is from a short queue to a long one (such as $x_1 = 5, x_2 = 8$.) The same arguments apply when the server is in queue 2. The fact that the net cost-reduction rates derived from buffer overflow ($c_1\lambda_1$, $c_2\lambda_2$) differ by a factor of two causes the large degree of asymmetry.

If each queue has a finite buffer size, the properties used in developing MIR do not hold because of the boundary effect. Since MIR was based on intuitive principles derived from queueing theory, our approach here is to develop a redesigned extension of MIR that explicitly incorporates the boundary effect when computing reward rates corresponding to action sequences. Although we have developed a heuristic specifically designed for cases without rejection costs (see Kim and Van Oyen [8]), for brevity we present here only the case with rejection costs. To illustrate the significant impact of rejection penalties, Example 2 modifies the first example to include $S_1 = S_2 = 500.0$, which is shown in Figure 2. The difference is dramatic and points out the difficulty of designing a heuristic that is effective across a wide range of problem parameters.

Before detailing the development of CMIR, we introduce the following notation. Decisions are constrained by the nonpreemptive assumption for service and set-up times, so we need only consider decision epochs with $\delta(t) = 2$, which signals that a service or set-up has just been completed. At time t the vector of queue lengths is denoted by $x = (x_1, x_2, \ldots, x_N)$. Suppose that the server provides service to queue i. Let the expected length of a M/M/1 busy period when the server

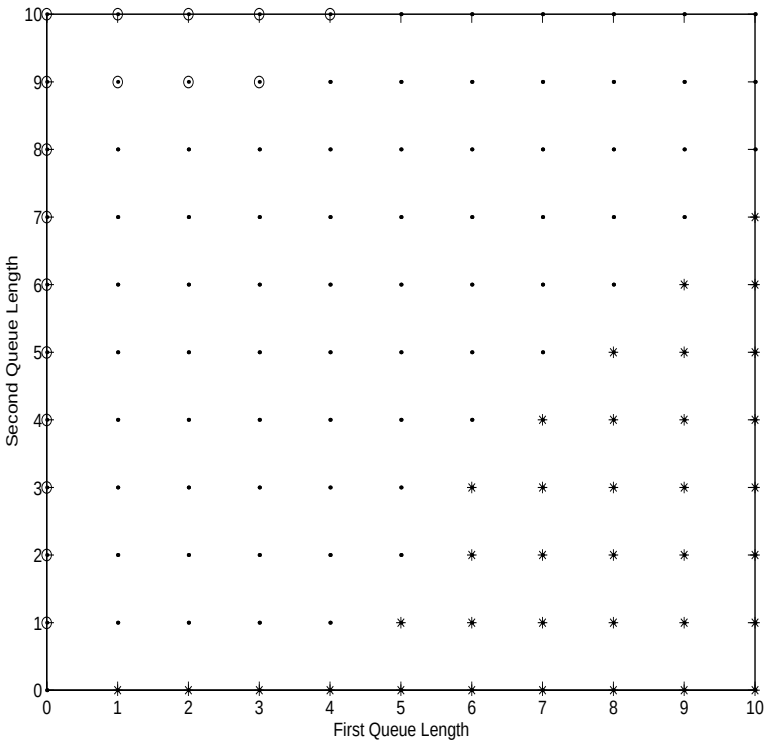

Figure 2: Optimal policy for Example 2.
· : Serve the current queue (do not switch).
⊙ : Serve queue 2 (switch to queue 2 if in 1).
* : Serve queue 1 (switch to queue 1 if in 2).

exhaustively serves queue i beginning with x_i jobs be denoted by

$$t_i(x_i) \triangleq x_i/(\mu_i - \lambda_i). \tag{4.1}$$

Let the expected time required for queue $j \neq i$ to reach M_j starting from x_j (assuming queue j is not served) be denoted by

$$s_j(x_j) \triangleq (M_j - x_j)/\lambda_j \ . \tag{4.2}$$

Now assume that the server determines to switch from queue i to j. Since queue j with initially x_j jobs must be set up before service can begin in queue j, the expected number of jobs present in queue j at the end of the set-up will be $x_j + \lambda_j \bar{D}_j$. Thus a rough approximation for the time that queue j will be in service under exhaustive service is given by

$$\tilde{t}_j(x_j) \triangleq \frac{\min\{M_j, x_j + \lambda_j \bar{D}_j\}}{\mu_j - \lambda_j}. \tag{4.3}$$

We begin with the development of switching condition and then consider idling condition. Our presentation will first explain how we capture the key factors when rejection penalties are neglected, and then we later show how these terms are incorporated.

Switching Condition

To derive a switching condition which dictates when to switch from a non-empty queue, say i, we compare the expected reward rates corresponding to the following two action sequences:

1. Serve one more job in current queue i.

2. Switch to queue $j \neq i$, serve it exhaustively, and return to queue i.

Suppose the server chooses action sequence 1. First, we compute the reward rate associated with remaining in queue i. If the server remains in queue i, it will earn rewards at a rate of $c_i\mu_i$ until the end of queue i's busy period. Furthermore, the boundary effect may contribute additional reward for queues which overflow. If queue j reaches M_j before the end of queue i's busy period, the server can earn the extra rewards at a rate of $c_j\lambda_j$ from the time when $x_j = M_j$ to the end of queue i's busy period. If the rejection penalties are neglected, our empirical test show that we can approximate the index to *remain* in queue i as $c_i\mu_i + (\sum_{j=1,j\neq i}^{N} c_j\lambda_j\ (\mu_i^{-1} - s_j(x_j))^+)/\mu_i^{-1}$, where $a^+ = \max\{a, 0\}$. We restrict the summation to $j \neq i$, because our simple approximations imply that queue i cannot hit the boundary while it is served. We now incorporate the rejection penalties. The main difference comes from the fact that in computing reward rates we subtract the rejection cost rate $\lambda_j S_j$ during intervals of time in which we expect queue j to be full. That is, during the period of buffer overflow, arrivals are lost from queue k at the rate λ_k. Each of them saves a holding cost c_k, but also incurs a rejection reward $-S_k$. If queue j already has customer overflow upon the service completion in queue i and we assume that the server switches to queue j to reduce the rejection costs, the actual length of the overflow will become $\mu_i^{-1} + \bar{D}_j - s_j(x_j)$. It is effective to approximate the expected reward rate for action sequence 1 by

$$\Phi_i(x, r) \triangleq \frac{c_i + \sum_{j=1,j\neq i}^{N}(c_j - S_j)\lambda_j(\mu_i^{-1} + \bar{D}_j - s_j(x_j))^+}{\mu_i^{-1}} \tag{4.4}$$

On the other hand, if the server decides to switch to queue j (action 2), it will first have to set-up queue j and earn no holding cost rewards for the expected duration of $\bar{D}_j$. For the purpose of computing the rewards rates, we assume that once the server switches to a queue, the server will remain in that queue until the end of its busy period. By switching to queue j, exhausting it, and returning to queue i, the server will have spent an expected total amount of time approximated by $T(x_j) \triangleq \bar{D}_j + \tilde{t}_j(x_j) + \bar{D}_i$. Since the server works only during $\tilde{t}_j(x_j)$, the reward rate earned by the server is $c_j\mu_j\tilde{t}_j(x_j)/T(x_j)$. In addition to this earned reward, the server can earn extra rewards which result from allowing the other queues to hit the boundary. As we see in (4.4), the rejection penalty is charged during periods of overflow in contrast to the holding cost saved during these same periods Hence, the expected reward rate when the server switches to queue j is given by

$$\Phi_{ij}(x, s) \triangleq \frac{c_j\mu_j\tilde{t}_j(x_j) + (c_j - S_j)\lambda_j(\bar{D}_j - s_j(x_j))^+ + \sum_{k=1,k\neq j}^{N}(c_k - S_k)\lambda_k(T(x_j) - s_k(x_k))^+}{T(x_j)}, \tag{4.5}$$

where the term $c_j\lambda_j(\bar{D}_j - s_j(x_j))^+$ is included for cases in which queue j hits the boundary prior to set-up completion.

To conclude the switching rule, we introduce two heuristic necessary conditions for switching, **N1** and **N2**.

Thus far, we have assumed that the server exhausts queue j prior to returning to queue i. In cases with rejection costs, however, the server might leave queue j with unfinished jobs when queue i is in danger of customer overflow. By the same argument, the growth of queue j during the service of queue i may require the server to return to queue j again before queue i is exhausted. Therefore, the heuristic should be designed to be well balanced in reducing both rejection and switching time penalties. One way of doing this is to allow the server to switch only when the size of the current queue, i, is small enough for the server to exhaust the other queue, j, and return to queue i prior to hitting M_i. In other words, if one expects the current queue i to face overflow while queue j is served, the policy should not allow the server to switch even though the reward rate obtained when switching is larger than that obtained when remaining. We formalize this idea as the following necessary condition for switching, **N1**:

$$\frac{M_i - x_i}{\lambda_i} > T(x_j). \tag{4.6}$$

The right side of the inequality approximates the expected time required for the server to switch to queue j with x_j jobs, exhaust it, and return to queue i, while the left side is the expected time to be taken until queue i is in danger of the customer overflow.

To conclude the rule for switching, we introduce the following condition **N2**:

$$\frac{\tilde{t}_j(x_j)}{T(x_j)} \geq \rho. \tag{4.7}$$

As pointed out in Duenyas and Van Oyen [3], overly frequent switches can lead to instability in scheduling the system. This condition requires the server to spend an average proportion of the time actually working that is greater than ρ (during the time interval consisting of the set-up of queue j, the service of queue j, and the subsequent set-up of queue i). Therefore, the three conditions $\Phi_{ij}(x,s) > \Phi_i(x,r)$, (4.6), and (4.7) must be satisfied to consider a switch from queue i to j. If there is more than one queue which meets these conditions, CMIR selects the queue with the highest reward rate, $\Phi_{ij}(x,s)$ among such queues.

Idling Condition

To complete the characterization of CMIR, we specify an idling policy for states in which there are no jobs in the current queue i. To decide whether the server should switch or idle, we consider the following action sequences:

1. Wait until the next arrival to the current queue i.

2. Immediately switch to queue j and exhaust it.

When there is no rejection penalty, sometimes it is cost-effective to wait until the next arrival to the current empty queue and serve the busy period generated by this new arrival. With a rejection penalty, however, this action cannot be justified as easily because some queues may be in danger of overflow. Therefore, the first action sequence should occur only when the system WIP is very low.

Queue j ($\neq i$) is considered to be in danger of overflow if it reaches full capacity before the time of its set-up completion, that is, $\bar{D}_j > s_j(x_j)$. Among such queues, a queue which has the largest expected rejection penalty will be given the highest priority for switching. Let $I_1(x,i) \stackrel{\Delta}{=} \{j : j \neq i, \bar{D}_j > s_j(x_j)\}$ denote the set of queues that are in danger of overflow. Suppose $I_1(x)$ is not empty, CMIR will switch to the highest priority queue, defined as

$$j^*(i) \stackrel{\Delta}{=} argmax_{j \in I_1(x,i)} \{S_j \lambda_j (\bar{D}_j - s_j(x_j))\}. \tag{4.8}$$

Next, suppose $I_1(x) = \emptyset$; that is, no other queues are in danger of overflow. For this case, for action sequence 2 we compute a reward rate index corresponding to action sequence 2.

Using the same argument as that in switching condition, the expected reward rate of switching to queue j is given by

$$\Psi_{ij}(x,s) \stackrel{\Delta}{=} \frac{c_j \mu_j \tilde{t}_j(x_j) + \sum_{k=1,k\neq j}^{N} (c_k - S_k)\lambda_k (T'(x_j) - s_k(x_k))^+}{T'(x_j)}, \tag{4.9}$$

where $T'(x_j) = \bar{D}_j + \tilde{t}_j(x_j)$. Since the server leaves an empty queue, we do not assume that the server returns to queue i, as we did for $T(x_j)$.

To make the idling condition in a non-full system work effectively, we place a necessary condition on switching. If the current queue i is empty and $I_1(x,i) = \emptyset$, queue j can be a candidate for switching by CMIR only if $x_j > \lambda_j \cdot \bar{D}_i$. This condition is suggested in Duenyas and Van Oyen [3] as a way to limit switches to j when the number of jobs there is relatively small. The judgment that queue j is relatively small is made when setting up queue i would on average result in more jobs arriving to j than are currently there. The effectiveness of this measure has been confirmed experimentally.

Let $I_2(x,i) \stackrel{\Delta}{=} \{j : j \neq i, x_j > \lambda_j \cdot \bar{D}_i\}$. If $I_1(x) = \emptyset$ and $I_2(x,i) \neq \emptyset$, the $\Psi_{ij}(x,s)$ index is used to select the queue $k^*(i)$ with the highest priority for switching according to

$$k^*(i) \stackrel{\Delta}{=} argmax_{j \in I_2(x,i)} \Psi_{ij}(x,s) \ . \tag{4.10}$$

Finally, if $I_1(x) = \emptyset$ and $I_2(x,i) = \emptyset$, CMIR follows action 1.

Summary of the CMIR Scheduling Heuristic

We now summarize CMIR, which is easily precomputed for every state in the state space with the vector of queue lengths denoted by $x = (x_1, x_2, \ldots, x_N)$:

1. If the current queue i is empty,

   (a) If $I_1(x,i) \neq \emptyset$, then switch to queue $j^*(i)$ defined by (4.8);

   (b) else if $I_1(x,i) = \emptyset$ and $I_2(x,i) \neq \emptyset$, then switch to queue $k^*(i)$ defined by (4.10);

   (c) else, idle at queue i until the next arrival to the system.

2. If the current queue i is not empty, let $I_S = \emptyset$. For all $j \neq i$ satisfying the necessary switching conditions **N1** (4.6), and **N2** (4.7), as well as the index ordering $\Phi_{ij}(x,s) > \Phi_i(x,r)$; set $I_S = I_S \cup \{j\}$.

   (a) If $I_S \neq \emptyset$, let j^* denote the queue such that $j^* = argmax_{j \in I_S}\{\Phi_{ij}(x,s)\}$ and then switch to queue j^*;

   (b) otherwise, process one more job in queue i.

Benchmarking the CMIR Scheduling Heuristic

In the following, we present numerical results for CMIR. Unlike infinite-buffer queueing systems, the optimal solution of a finite-buffer queueing system can be computed to within a prespecified tolerance, ϵ, using a dynamic program. The MIR heuristic for scheduling multi-class queueing systems is reported in Duenyas and Van Oyen [3] to outperform several other scheduling policies in the literature. Although MIR was intended to be applied to an infinite-buffer queueing system, we use its performance as well as the optimal performance as benchmarks to determine how much finite buffers affect the system performance. We compare the performance of our heuristic with respect to the optimal as well as MIR.

We report 36 examples exhibited in Tables $1-2$. Examples 1–26 and 27–36 are for two-queue and three-queue problems, respectively. In Table 1, Examples 4–12 are symmetric and the others are asymmetric. In Table 2, Examples 27–30 are symmetric. We display the numerical performance for these examples in Tables 3–4. Although we tested many other examples, the cases reported are representative of problems with similar scalings of the parameters. We computed the optimal average costs using VIPE algorithm developed in Kim et al. [9] and the average costs of CMIR and MIR using successive approximation with a termination criterion of $\epsilon = 10^{-5}$ for two-queue problems and 10^{-3} for three-queue problems.

For symmetric two queue examples (Examples 4–12), the results indicate that CMIR performs better as (1) ρ becomes large (Examples 4–6), (2) the rejection costs becomes small (Examples 7–9), and (3) the switching times become small (Examples 10–12). In Example 7 both heuristics work well; however, MIR works better and achieves nearly optimal performance.

For Examples 13 and 14, since $c_1\mu_1 > c_2\mu_2$, MIR exhausts queue 1. When ρ is low (Example 13), both heuristics are within 1% of optimal; however, when ρ is high (Example 14) the performance

Ex	M_1	M_2	S_1	S_2	c_1	c_2	μ_1	μ_2	λ_1	λ_2	d_1	d_2	ρ
1	10	10	0	0	1	1	2	2	1.0	0.5	2	2	0.75
2	10	10	500	500	1	1	2	2	1.0	0.5	2	2	0.75
3	10	10	50	50	1	1	2	2	1.0	0.5	2	2	0.75
4	7	7	50	50	1	1	2	2	0.5	0.5	2	2	0.5
5	7	7	50	50	1	1	2	2	0.7	0.7	2	2	0.7
6	7	7	50	50	1	1	2	2	0.9	0.9	2	2	0.9
7	7	7	20	20	1	1	2	2	0.6	0.6	2	2	0.6
8	7	7	100	100	1	1	2	2	0.6	0.6	2	2	0.6
9	7	7	500	500	1	1	2	2	0.6	0.6	2	2	0.6
10	7	7	50	50	1	1	2	2	0.8	0.8	2	2	0.8
11	7	7	50	50	1	1	2	2	0.8	0.8	4	4	0.8
12	7	7	50	50	1	1	2	2	0.8	0.8	10	10	0.8
13	7	7	50	50	2	1	2	1	0.6	0.3	2	2	0.6
14	7	7	50	50	2	1	2	1	0.8	0.4	2	2	0.8
15	7	7	100	100	2.1	1	1	2	0.2	1	2	2	0.7
16	7	7	100	100	2.1	1	1	2	0.2	1.2	2	2	0.8
17	7	7	100	100	2.1	1	1	2	0.2	1.4	2	2	0.9
18	7	7	20	100	3	1	1	2	0.2	1	1	1	0.7
19	10	15	50	100	3	1	2	1	0.4	0.4	1	.5	0.6
20	10	15	50	100	3	1	2	1	0.6	0.6	1	.5	0.9
21	15	10	50	100	2	1	2	2	0.6	0.6	.5	.5	0.6
22	15	10	200	50	2	1	2	1	0.4	0.5	1	1	0.7
23	15	10	60	180	4	1	2	1.5	0.8	0.4	1	1	0.67
24	5	5	40	40	2	1	2	1	0.6	0.3	2	1	0.6
25	5	5	100	20	2	2	1	1	0.4	0.3	1	1	0.7
26	5	5	20	100	5	1	1	1	0.3	0.3	1	1	0.6

Table 1: Input Data for Examples 1–24.

of MIR is more degraded by customer overflow, indicating that as one would expect, the effect of customer rejections is significant.

Examples 15–18 are designed to have $c_1\mu_1 > c_2\mu_2$ but $\lambda_1 S_1 << \lambda_2 S_2$. Thus, even though queue 1 is a top priority queue, queue 2 has greater overflow penalties. Because MIR exhausts queue 1, it performs three to five times worse than CMIR.

Examples 19–26 vary the buffer sizes, among other things. The results suggest that the performance of CMIR is effective for a variety of buffer sizes; however, test cases not reported here reveal that extremely small buffer sizes (3 or fewer jobs) render our approximations inaccurate and ineffective. If such cases are essential to an application, we suggest that a targeted analysis is appropriate for such cases.

Examples 27–30 of Table 4 depict representative examples with three queues indicating that CMIR works well. It is not hard to find problems with very small buffers and/or large utilization levels for which an optimal policy may forever abandon a queue, but such problems are not well motivated by applications. Examples 31–35 have $c_1\mu_1 \geq c_2\mu_2 \geq c_3\mu_3$ and $\lambda_1 S_1 \geq \lambda_2 S_2 \geq \lambda_3 S_3$. When buffer sizes are nearly symmetric (see Examples 31–33), both CMIR and MIR work well.

Ex	M_1	M_2	M_3	S_1	S_2	S_3	c_1	c_2	c_3	ρ
	μ_1	μ_2	μ_3	λ_1	λ_2	λ_3	d_1	d_2	d_3	
27	7	7	7	50	50	50	1	1	1	.6
	2	2	2	.4	.4	.4	2	2	2	
28	7	7	7	500	500	500	1	1	1	.6
	2	2	2	.4	.4	.4	2	2	2	
29	7	7	7	500	500	500	1	1	1	.75
	2	2	2	.5	.5	.5	2	2	2	
30	7	7	7	500	500	500	1	1	1	.9
	2	2	2	.6	.6	.6	2	2	2	
31	7	7	7	50	50	50	2	1.5	1	.9
	2	2	2	.2	.6	1	2	2	2	
32	7	7	7	100	100	100	2.1	1	1	.8
	1	2	2	.1	.7	.7	2	2	2	
33	7	7	8	50	50	70	1.2	1	1.2	.8
	2	2	1.5	.5	.5	.5	1	2	3	
34	10	10	2	100	100	100	1	1	1	.9
	2	2	2	.6	.6	.6	2	2	2	
35	7	7	5	95	90	100	1	1	.5	.75
	2	2	2	.6	.6	.3	2	3	1	
36	7	7	7	50	50	50	0	0	0	.75
	2	2	2	.5	.5	.5	1	1	1	

Table 2: Input Data for Examples 25–34.

Example 34 is similar to 30 and shows that CMIR can be effective in dealing with an extremely small buffer of size 2 for type 3. Both CMIR and MIR are sensitive to the buffer sizes in the examples we tested with three or more queues, which was not the case in our tests with two queues. As a special case, Example 36 has zero holding costs, which is handled well by CMIR. MIR; however, shows about 1,000% suboptimality, because it was never designed for such cases.

In addition to the examples presented here, we tested a variety of examples including problems with *four queues*. We noted that it is optimal to switch to a queue that is getting full less frequently than we expected. In fact, a large set-up time discourages the server from switching to a full queue even when it is in danger of customer overflow. MIR generally works well for our loss model despite the fact that it was not designed for models with rejection costs (and thus does not prescribe switching in response to full queues). This can be explained intuitively as follows. Set-up times reduce the effective server capacity for finite buffer systems, unlike infinite-buffer systems which can use arbitrarily large batch sizes (run lengths) to compensate for the overhead of each set-up. Thus, it is increasingly important as system utilization and the number of queues increases to avoid excessive switching, because significant customer overflow may be inevitable for systems with small buffers.

Example	Optimal	CMIR	Suboptimality (%)	MIR	Suboptimality (%)
1	4.2069	4.2813	1.8	4.2813	1.8
2	10.8325	12.3977	14.4	13.6411	25.9
3	5.1542	5.1757	0.4	5.2173	1.2
4	1.75168	1.75631	0.26	1.76758	0.9
5	4.8684	4.8769	0.2	4.9483	1.6
6	13.6522	13.6522	0.0	13.6522	0.0
7	2.5967	2.6019	0.2	2.5974	0.0
8	3.2121	3.2665	1.7	3.3825	5.3
9	6.0612	6.5891	8.7	7.3082	20.6
10	8.4025	8.4025	0.0	8.4467	0.5
11	6.3184	6.3911	1.1	6.4904	2.7
12	5.1707	5.2983	2.5	5.4864	6.1
13	3.6134	3.6427	0.8	3.6212	0.2
14	8.5932	8.9815	4.5	9.3891	9.3
15	6.4867	6.9287	6.8	7.7622	19.7
16	11.5917	12.2037	5.3	13.4441	16.0
17	19.8417	20.2216	1.9	21.8597	10.2
18	11.9042	12.2494	2.9	13.0799	9.9
19	7.9993	8.1102	1.4	8.1388	1.7
20	27.0431	27.1114	0.3	27.1272	0.3
21	11.1139	11.4664	3.2	12.5029	12.5
22	6.64800	6.9924	5.2	6.9183	4.1
23	12.6816	13.1406	3.6	13.0779	3.1
24	4.8715	4.8994	0.6	5.0962	4.6
25	7.9398	8.5098	7.2	8.4169	6.0
26	7.5926	8.0121	5.5	7.8502	3.4

Table 3: Performance Results for Examples 1–26.

Example	Optimal	CMIR	Suboptimality (%)	MIR	Suboptimality (%)
27	3.25	3.25	0.0	3.26	0.3
28	4.53	4.82	6.4	5.26	16.1
29	21.55	22.27	3.3	23.73	10.1
30	80.73	81.35	0.8	81.35	0.8
31	17.40	17.79	2.2	17.68	1.6
32	12.25	12.52	2.2	13.17	7.5
33	12.58	12.98	3.2	12.71	1.0
34	32.58	34.51	5.9	38.24	17.4
35	9.53	10.29	8.0	10.96	15.0
36	4.63	5.42	17.1	50.00	979.9

Table 4: Performance Results for Examples 27–36.

5. Conclusions

For systems with small buffers, we developed the CMIR heuristic policy, which captures the boundary effect introduced by customer loss with positive holding costs and rejection costs. Numerical results indicate that CMIR performs very well, in particular, for asymmetric and/or high rejection cost systems. This is significant to us for the following reasons. From an analytical perspective, finite buffers significantly complicate the problem. Because analytical investigation of the structure of an optimal policy and the boundary effects lies beyond the state of the art, we turned to the development of the CMIR heuristic policy to incorporate the conjectured relationships. Although the CMIR heuristic's primary value lies in providing a simple yet effective algorithm that incorporates the realistic feature of finite buffers, the effectiveness of CMIR also corroborates the validity of our insights into the the effects of small buffers.

Our approach used a natural truncation of the queue lengths, which we proved in Theorem 1 provides a lower bound (to within the tolerance specified) to the optimal value function of the infinite system. We demonstrated the monotonicity of optimal performance to all of the system parameters except for the queue capacities, which justifies managerial emphasis on set-up time reduction efforts.

Acknowledgments

The work of the authors was supported by the National Science Foundation under Grant No. DMI-9522795. This work is based on Eungab Kim's dissertation under the direction of Mark Van Oyen at Northwestern University. We wish to acknowledge Maria Rieders, who worked jointly with us on numerically characterizing the structure of optimal policies.

References

[1] Bertsekas, D.P. (1987) *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs.

[2] Browne, S. and Yechiali, U. (1989) Dynamic priority rules for cyclic-type queues, *Adv. Appl. Prob.,* **21**, 432–450.

[3] Duenyas, I. and Van Oyen, M. P. (1996) Heuristic scheduling of parallel heterogeneous queues with set-ups, *Management Science*, **42:6**, 814–829.

[4] Duenyas, I. and Van Oyen, M. P. (1996) Stochastic scheduling of parallel queues with set-up costs, *Queueing Systems (QUESTA)*, **19**, 421–444.

[5] Hofri, M. and Ross, K.W.(1987). On the optimal control of two queues with server setup times and its analysis. *SIAM Journal on Computing,* **16**, 399-420.

[6] Kim, E. and Van Oyen, M.P. (1998) Beyond the $c\mu$ rule: Dynamic scheduling of a two-class loss queue. *Mathematical Methods of Operations Research,* **48:1**.

[7] Kim, E. and Van Oyen, M.P. (1998) Dynamic scheduling to minimize lost sales subject to set-up costs. *Queueing Systems (QUESTA)*, **24**.

[8] Kim, E. and Van Oyen, M.P. (1998) Finite-capacity multi-class production scheduling with set-up times: Long Version, Technical Report, Department of Industrial Engineering and Management of Sciences, Northwestern University. (Available in postscript at http://www.iems.nwu.edu/~vanoyen/)

[9] Kim, E., Van Oyen, M.P., and Rieders, M. (1998) General Dynamic Programming Algorithms Applied to Polling Systems, *Communications in Statistics: Stochastic Models,* **14:5**.

[10] Koole, G. (1996) Assigning a single server to inhomogeneous queues with switching costs, *Theoretical Computer Science,* **182**, 203–216.

[11] Kumar, P.R. and Varaiya, P. (1986) *Stochastic Systems: Estimation, Identification, and Adaptive Control.* Prentice-Hall, Englewood Cliffs.

[12] Levy, H. and Sidi, M. (1990) Polling systems: applications, modeling, and optimization. *IEEE Transactions on Communication,* **38**, 1750-1760.

[13] Liu, Z. and Nain, P. and Towsley, D. (1992) On optimal polling policies. *Queueing Systems (QUESTA),* **11**, 59-83.

[14] Olsen, T.L. (1998) A practical scheduling method for multi-class production system with setups, Preprint.

[15] Rajan, R. and Agrawal, R. (1996) Server allocation and routing in homogeneous queues with switching penalties. *IEEE Transactions on Automatic Control*, **AC-41**: 1657–1661.

[16] Reiman, M.I. and Wein, L.M. (1994) Dynamic scheduling of a two-class queue with setups. *Operations Research*, **46:4**, 532–547.

[17] Righter, R. and Shanthikumar, J.G. (1994) Multi-class production systems with setup time. Preprint.

[18] Rosberg, Z. and Kermani, P. (1992) Customer scheduling under queueing constraints, *IEEE Transactions on Automatic Control* **AC-37:2**, 252-257.

[19] Ross, S.M. (1983) *Introduction to Stochastic Dynamic Programming.* Academic Press, New York.

[20] Ross, S.M. (1983) *Stochastic Processes.* Wiley, New York.

[21] Sarkar, D. and Zangwill, W.I. (1991) Variance Effects in Cyclic Production Systems. *Management Science,* **37**, 444–453.

[22] Takagi, H. (1994) Queueing Analysis of Polling Models: Progress in 1990–1993, in J.H. Dshalalow, Ed. *Frontiers in Queueing: Models, Methods and Problems.* CRC Press, Baton Rouge.

[23] Van Oyen, M.P., (1997) Monotonicity of Optimal Performance Measures for Polling Systems, *Probability in the Engineering and Informational Sciences,* **11:2**, 219-228.

[24] Van Oyen, M.P., Pandelis, D.G., and Teneketzis, D. (1992) Optimality of index policies for stochastic scheduling with switching penalties, *J. of Appl. Prob.*, **29**, 957–966.

[25] Van Oyen, M.P. and Teneketzis, D. (1994) Optimal Stochastic Scheduling of Forest Networks with Switching Penalties, *Advances in Applied Probability*, **26**, 474–497.

[26] Varaiya, P., Walrand, J., and Buyukkoc, C. (1985) Extensions of the multi-armed bandit problem, *IEEE Trans. Autom. Control,* **AC-30**, 426–439.

[27] Walrand, J. (1988) *An Introduction to Queueing Networks* Prentice-Hall, Englewood Cliffs.

[28] Zangwill, W.I. (1992) The Limits of Japanese Production Theory. *Interfaces,* **22:5**, 14–25.