STOCHASTIC SCHEDULING OF PARALLEL QUEUES WITH SET-UP COSTS

Izak Duenyas
Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor, MI 48109-2117

and

Mark P. Van Oyen
Dept of Industrial Eng and Operations Eng
Northwestern University
Evanston, IL 60208-3119

Technical Report 93-40

December 1993

Revised January 1995

Stochastic Scheduling of Parallel Queues with Set-Up Costs

Izak Duenyas
Department of Industrial and Operations Engineering
The University of Michigan, Ann Arbor, MI 48109-2117

Mark P. Van Oyen
Department of Industrial Engineering and Management Sciences
Northwestern University, Evanston, IL 60208-3119

Abstract

We consider the problem of allocating a single server to a system of queues with Poisson arrivals. Each queue represents a class of jobs and possesses a holding cost rate, general service distribution, and a set-up cost. The objective is to minimize the expected cost due to the waiting of jobs and the switching of the server. A set-up cost is required to effect an instantaneous switch from one queue to another. We partially characterize an optimal policy and provide a simple heuristic scheduling policy. The heuristic's performance is evaluated in the cases of two and three queues by comparison with a numerically obtained optimal policy. Simulation results are provided to demonstrate the effectiveness of our heuristic over a wide range of problem instances with four queues.

Keywords: stochastic scheduling, setup cost, control heuristic, polling system

1. Introduction

We consider a facility where customers of several different classes are competing for service from a single server. Holding costs are incurred for each unit of time that customers have to wait in

the system, and switchover costs are incurred when the server switches from serving one type of customer to another.

The optimal control of queues with multiple customer classes has been extensively addressed in the literature. The great majority of these models, however, have considered the case with zero switchover costs. It is well known, for example, that for an $M/G/1$ queue with multiple customer classes, if jobs of type i are charged holding costs at rate c_i and are processed at rate μ_i, the $c\mu$ rule minimizes the average holding cost per unit time (see Baras et al. [1], Buyukkoc et al. [3], Cox and Smith [4], Gittins [7], Nain [21], Nain et al. [22], and Walrand [30]). Other stochastic scheduling problems in the literature for which there are no costs for switching from one type of job to another may be found in Baras et al. [1], Dempster et al. [5], Gittins [7], Harrison [10],[11], Klimov [13], [14], Lai and Ying [16], Nain [21], Nain et al. [22], Varaiya et al. [29], and Walrand [30].

In most manufacturing environments, it is actually more appropriate to consider switchover or set-up *times* rather than switchover *costs*. There are certainly situations, however, where switching is costly and switchover times are small compared to the processing times. For example, in the production of asphalt shingles, changes in pebble color or shingle weight are accomplished while the process is running. Although the changeover can be accomplished quickly, scrap is produced and the cost of this scrap is the primary consideration. We also note that interest in just-in-time (JIT) production has led to significant reductions of set-up times. Hall [9] notes that significant reductions in set-up times can be obtained by differentiating between *external set-up* and *internal set-up*. External set-up refers to set-up work done while the machine is running. Internal set-up refers to set-up work done with the machine stopped. As Hall [9] emphasizes, one of the most significant ways to decrease set-up times is by converting as much set-up activity as possible from internal set-up to external set-up. Although a dramatic reduction of set-up times during which machines are not productive can be obtained by trading a maximum amount of internal set-up for external set-up, there is clearly a cost to performing the external set-up. At the very least, workers have to spend the time necessary to perform the external set-up. Although the objective is to simultaneously achieve small set-up costs and times, there will be situations in which the *internal set-up time* (the time the machine is stopped) is small and the *cost* of an external set-up becomes the dominant factor. (We note that in situations where the service of a queue is short, there may

not be enough time to perform the external set-up required for the next queue, and a set-up time may be impossible to avoid. This situation can best be modeled by considering set-up times and costs simultaneously, which is beyond the scope of this paper and remains a topic for future work.)

Because of the difficulties introduced by switching penalties, there are few known results for the optimal scheduling of such systems. Boxma et al. [2] developed effective static polling rules for the problem with set-up times. Gupta et al. [8] considered the problem in this paper with switching costs and only two types of jobs with the same processing time distributions. Hofri and Ross [12] considered a similar problem with switching times, switching costs, and two homogeneous classes of jobs. They conjectured that the optimal policy is of a threshold type. Concurrently with our work, Koole [15] and Reiman and Wein [24] treated the problem at hand in the case of two queues. For the case of exponential service distributions, Koole performed a numerical study and a dynamic programming analysis that partially characterizes an optimal policy (as nonidling and exhaustive at one queue, in addition to the asymptotic behavior of the switching curve with respect to the length of the other queue). Koole suggests a heuristic policy that requires the solution of a dynamic program. Reiman and Wein focused on the problem with two job types and switching costs or switching times and developed heuristics by approximating the dynamic scheduling problem by a diffusion control problem. We note that both Koole and Reiman and Wein propose methods to determine heuristic policies that possess the same structural features as the heuristic policy we derive (a top-priority queue, a constant switching threshold for switching from the lower priority queue, and thresholds for idling). In addition, we treat the case of N queues, propose a heuristic policy, and analyze its performance.

For problems with N queues, Duenyas and Van Oyen [6] addressed the problem with switching times. They partially characterized the optimal policy and developed some dynamic heuristics. Van Oyen et al. [27] treated versions without arrivals but with *heterogeneous* service processes and either set-up costs or set-up times. They proved that an index policy is optimal and computed the optimal indices for cases with lump-sum set-up costs and those with set-up times. Van Oyen and Teneketzis [28] further extended this approach for systems with switching penalties and without arrivals by treating connected queues such that the resulting network formed an in-forest. They identified conditions on the holding costs and service distributions for which an index rule is optimal in the case of tandem queues as well as conditions for which exhaustive service is optimal in any in-

forest. Recently, Rajan and Agrawal [23] and Liu et al. [19] have studied systems similar to the one considered here (as well as extensions to incomplete information) and have partially characterized an optimal policy for the case of homogeneous service processes.

Other work has concentrated on performance evaluation and stochastic comparisons of different policies (e.g., Levy and Sidi [17], Levy et al. [18], and Takagi [25]).

In this paper, we address the stochastic scheduling of a system with several different types of jobs and switching costs in the context of a multiclass $M/G/1$ queue. In Section 2, we formulate the problem. In Section 3, we partially characterize the optimal policy. In Section 4, we develop a heuristic policy. In Section 5, we test the performance of our heuristic by comparing it to other heuristics from the literature and to the optimal policy. The paper concludes in Section 6.

2. Problem Formulation

A single server is to be allocated to jobs in a system of parallel queues labeled $1, 2, \ldots, N$ and fed by Poisson arrivals. By parallel queues, we mean that a job served in any queue directly exits the system. Each queue (equivalently, node) n possesses a general, strictly positive service period σ_n with distribution B_n, mean μ_n^{-1} $(0 < \mu_n^{-1} < \infty)$, and a finite second moment. Successive services in node n are independent and identically distributed (i.i.d.) and independent of all else. Jobs arrive to queue n according to a Poisson process with rate λ_n (independent of all other processes). To ensure stability, we assume that $\rho = \sum_{i=1}^{N} \rho_i < 1$, where $\rho_i = \lambda_i/\mu_i$.

Holding cost is assessed at a rate of c_n $(0 < c_n < \infty)$ cost units per job per unit time spent in queue n (including time in service). A switching cost or set-up cost of K_n $(0 < K_n < \infty)$ cost units is incurred at each instant (including time 0) the server switches from a queue other than n to process a job in n. We assume that the set-up action is achieved instantaneously (zero switchover times). Random switching costs pose no significant problems, so we assume that the switching costs are deterministic only for simplicity.

A policy specifies, at each decision epoch, whether the server should remain working in the present queue, idle in the present queue, or set-up another queue for service. With $\mathbb{R}^+(\mathbb{Z}^+)$ denoting the nonnegative reals (integers), let $\{X_n^g(t) : t \in \mathbb{R}^+\}$ be the right-continuous queue length process of node n under policy g (including any customer of node n in service). Denote the vector of initial queue lengths by $X(0^-) \in (\mathbb{Z}^+)^N$, where $X(0^-)$ is fixed. Without loss of generality,

we assume that node one has been set up prior to time $t = 0$ and that the server is initially placed in node one. Let $n^g(t)$ be the right-continuous process describing the location of the server at t under policy g. Define $\Gamma_n^g = \{t \in \mathbb{R}^+ : n^g(t^-) \neq n,\ n^g(t) = n\}$ to be the set of random instances of switching into node n under g. The average cost per unit time of policy g, $\bar{J}(g)$, can now be expressed as

$$\bar{J}(g) = \limsup_{T\to\infty} \frac{1}{T} E\left\{ \int_0^T \left(\sum_{n=1}^N c_n X_n^g(t) \right) dt + \sum_{n=1}^N \sum_{t\in\Gamma_n^g} K_n \right\} . \tag{2.1}$$

The class of admissible strategies, G, is taken to be the set of non-preemptive and non-anticipative (possibly non-stationary and randomized) policies that are based on perfect observations of the queue length processes. Our restriction to non-preemptive policies requires that once the service of a job begins, that service cannot be discontinued, nor can it be interrupted by switching. Idling is allowed. The set of decision epochs is assumed to be the set of all arrival epochs, service completion epochs, and instances of idling. Although it seems clear that it suffices to consider G^{PM}, the class of pure Markov (that is, stationary and non-randomized) policies, the technical details of our proofs require us to consider the more general class of policies, G.

For policies which employ idling improperly, (2.1) may be infinite. For $\rho < 1$, it is well known that policies such as the exhaustive and gated cyclic polling strategies yield a stable system (see Levy and Sidi [17]). Thus, finite steady state average queue lengths exist under an optimal policy. Because an optimal policy requires at most one set-up per job served, it is also clear that the average switching cost per unit time is also finite. The objective is to minimize total of the weighted sum of the average queue lengths and the average switching cost per unit time.

Although our analysis is set within the general policy space G, in developing our heuristic, we focus on the subclass G^{PM} consisting of pure Markov policies. Under the restriction to pure Markov policies (and a memoryless arrival process), it suffices to regard the decision to idle as a commitment that the server idle for one (system) interarrival period. Thus, the state of the system is described by the vector $X(t) = (X_1(t), X_2(t), \ldots, X_N(t), n(t)) \in \mathcal{S}$, where $n(t)$ denotes that the server is set up to serve jobs at node $n(t)$ at time t and $\mathcal{S}$ denotes the state space $(\mathbf{Z}^+)^N \times \{1, 2, \ldots, N\}$. Let the action space $\mathcal{U} = \{1, 2, \ldots, N\} \times \{0, 1\}$ be defined in the following way. Suppose that at a decision epoch, t, the state is $X(t) = (x_1, x_2, \ldots, x_N, n(t)) \in \mathcal{S}$. Action $U(t) = (u, 1) \in \mathcal{U}$, where $u \neq n(t)$, causes the server to set up node u and to subsequently serve a single job (if any) in u.

Action $U(t) = (n(t), 1)$ results in the service of a job in $n(t)$, the currently set-up queue. Action $U(t) = (n(t), 0)$ selects the option to idle in the current queue until the next decision epoch, another system arrival. No other actions are possible.

3. On an Optimal Policy

In this section, we provide a partial characterization of an optimal policy within G. The special case with all switching costs equal to 0 has been well studied, with early results found in Cox and Smith [4]. The non-preemptive $c\mu$ rule is optimal: The index $c_i\mu_i$ is attached to each job in the ith queue. At any decision epoch, serve the available job possessing the largest index. Although these indices are most properly associated with individual jobs (see Varaiya et al. [29]), because the jobs in a given queue are indistinguishable, the indices can be associated with the queues. Note that the index of any queue is independent of both the queue length (provided it is strictly positive) and the arrival rate of that queue. Another special case has been treated in Liu et al. [19] and Rajan and Agrawal [23]. For problems that are completely homogeneous with respect to cost and to the service process, they *partially* characterized optimal policies as exhaustive and as serving the longest queue upon switching.

We begin our analysis with the following definitions:

Definition 1: A policy serves node i in a *greedy* manner if the server never idles in queue i while jobs are still available in i and queue i has been set up for service.

Definition 2: A policy serves node i in an *exhaustive* manner if it never switches out of node i while jobs are still available in i.

Definition 3: A *top-priority* queue refers to any queue (there may be more than one) that is served in a greedy and exhaustive manner.

Definition 4: A policy serves in a *patient* manner if it never switches into an empty queue.

Theorem 1 states that a top-priority queue always exists under an optimal policy and can be determined as the node maximizing $c_n\mu_n$ over all n. Theorem 1 is similar to the results presented in Gupta et al [8], Hofri and Ross [12], Liu et al. [19] and Rajan and Agrawal [23] for systems with homogeneous service processes. Although set-ups are not included in their model, Meilijson and Yechiali [20] prove the optimality (with respect to the total cost accrued up to a particular stopping time) of top-priority service to the job type maximizing $c_n\mu_n$ for a multi-class G/GI/1

(under non-preemptive service) with server breakdowns and idling. Our proof requires an intuitive lemma, which we state without proof.

Lemma 1: Consider a single stage optimization problem with a finite set of control actions, U. Action $u \in$ U results in an expected cost $\bar{c}_u \in \mathbb{R}$ and requires an expected length of time $\bar{\sigma}_u \in (0, \infty)$. Let $p_u \in [0,1]$ denote the probability that action u is taken where $\sum_u p_u = 1$. Then, the single-stage cost rate is at most $\max_{u \in \mathcal{U}} \bar{c}_u / \bar{\sigma}_u$; equivalently,

$$(\sum_{u \in \mathcal{U}} p_u \bar{c}_u)/(\sum_{u \in \mathcal{U}} p_u \bar{\sigma}_u) \leq \max_{u \in \mathcal{U}} \bar{c}_u / \bar{\sigma}_u. \tag{3.1}$$

Theorem 1: If $c_n \mu_n \geq c_j \mu_j$ for all $j = 1, 2, \ldots, N$, then there exists a policy for which queue n is a top-priority queue that is an optimal policy under the average cost per unit time criterion.

Proof: The argument is similar to that used in Duenyas and Van Oyen [6] for the case of set-up times. Relabel the queues so that $c_1 \mu_1 \geq c_j \mu_j$ for all $j = 1, 2, \ldots, N$. Suppose policy g is optimal but does not serve node one as a top priority node. We first prove the optimality of exhaustive service in node one, then justify greedy service in node one. Although g is assumed to be pure Markov for the sake of presentation, our argument applies to nonstationary and randomized feedback laws as well.

Suppose that policy g does not exhaust node one for some state $(x_1, \ldots, x_N, 1) \in \mathcal{S}$ with $x_1 \geq 1$. Suppose that g chooses to switch to node j at $t = 0$; thus $U^g(0) = (j, 1)$ for some $j \neq 1$. For $l \in \mathbb{N}$, let $t(l)$ denote the time at which the l^{th} control action is taken under policy g. Thus, $t(1) = 0$, and $t(2) = \sigma_j$. With respect to policy g, let the random variable $L \in \{\mathbb{N} \cup \infty\}$ denote the stage, or index of the decision epoch, at which g first chooses to serve a job of node one. Thus, $U^g(t(L-1)) \neq (1, \cdot)$, and $U^g(t(L)) = (1, 1)$. Because unstable policies cannot be optimal, L is finite with probability 1.

Along each sample path of the system, we construct a policy $\tilde{g}$, which interchanges the service of the job in queue one (stage L under g) with the first $L-1$ stages under g as follows. At time $t = 0$, $\tilde{g}$ serves the job in node one that is served under policy g at $t(L)$, which possesses the processing time σ_1. During $[\sigma_1, \sigma_1 + t(L))$, $\tilde{g}$ mimics the actions taken by g during $[0, t(L))$, the first $L-1$ stages. At time $t(L+1) = t(L) + \sigma_1$, both g and $\tilde{g}$ reach the same state along any realization, and $\tilde{g}$ mimics g from that point on. To compare g and $\tilde{g}$, we note that each job served during $(t(2), t(L)]$

under g is delayed by σ_1 time units under $\tilde{g}$, which represents an increased cost for $\tilde{g}$. On the other hand, the first job in queue 1 is completed at time σ_1 under $\tilde{g}$ and at $t(L)+\sigma_1$ under g, a cost savings of $c_1 t(L)$ for $\tilde{g}$.

To compare the difference between g and $\tilde{g}$, we define the costs associated with the stages $1,2,\ldots,L$ prior to the coupling of g and $\tilde{g}$. Let the random variable $g(l) \in \{1,2,\ldots,N+1\}$ denote the type of job, if any, served during stage l, where $g(l) = N+1$ with the probability that the server idled in any queue during stage l. Thus, $g(1) = j$ and $g(L) = 1$. Let the holding cost of the job, if any, served on stage l be denoted by the random variable $C(g(l))$, where $C(g(l)) = c_{g(l)}$ for $g(l) \leq N$ and $C(N+1) = 0$. Define $\sigma(g(l)) = t(l+1)-t(l)$. For outcomes $g(l) \leq N$, $\sigma(g(l)) = \sigma_{g(l)}$. As seen from (2.1), the average cost per unit time of policy g is defined by the limiting ratio of cumulative cost through T divided by T. It is convenient to first compute the *cumulative holding cost advantage* of policy $\tilde{g}$ over g, which equals

$$\begin{aligned}\Delta_h(g,\tilde{g}) &= E\{c_1 t(L) - \sum_{l=1}^{L-1} C(g(l))\sigma_1\} \\ &= E\{\sum_{l=1}^{L-1} E\{c_1\sigma(g(l)) - C(g(l))\sigma_1|L\}\} \ . \end{aligned} \tag{3.2}$$

To conclude that $\Delta_h(g,\tilde{g}) \geq 0$, it suffices to observe from Lemma 1 that for $l \in \{2,3,\ldots,L-1\}$,

$$E\{C(g(l))|L\}/E\{\sigma(g(l))|L\} \leq c_1/E\{\sigma_1\} = c_1\mu_1 \ . \tag{3.3}$$

If $x_1 = X_1(0) = 1$, then g serves only a single job on its first visit to queue 1 with the probability that there are no arrivals at queue 1 prior to the first visit of g to 1. If this event occurs, policy $\tilde{g}$ saves $K_1 > 0$ in cumulative expected switching cost with respect to g, because queue 1 need not be set up at $t = 0$. On the other hand, if $x_1(0) \geq 2$, then the above savings of a switch may be a zero probability event under g. Apply the argument thus far iteratively until the resulting modified policy saves a switch with strictly positive probability. Thus, the cumulative expected cost saving of $\tilde{g}$ with respect to g is strictly positive. We note that the preceding argument extends in a straightforward manner to the case of randomized policies and nonstationary policies. Since our construction may require $\tilde{g}$ to be nonstationary, our analysis is framed with the class G^{PM}.

To quantify the *average cost* advantage of top-priority service of queue 1, consider a policy g'' with a countable number of stages. Each stage applies the modification described above to a single instance at which queue one is not served exhaustively under g. If g only serves at most one type

one job nonexhaustively, g'' is the same as $\tilde{g}$ defined above and time $t(L+1)$ marks the end of the first (and only) stage. If g fails to exhaust queue one following the first stage, a second stage is added in the same way, and the construction continues in this manner. Each stage reverses a single instance of the non-exhaustion of queue 1 under g. At instances between these interchanges, g and g'' are identical. We observe the following: If g fails to serve only a finite number of jobs in queue one exhaustively, then $\bar{J}(g'') = \bar{J}(g)$. In general, g'' will perturb g at a countably infinite number of stages resulting in $\bar{J}(g'') \leq \bar{J}(g)$. If g is pure Markov, the strictly positive per stage savings of the modification yields $\bar{J}(g'') < \bar{J}(g)$. A strict average cost savings is not, however, essential to our argument. Thus we see that there exists a policy that always exhausts queue one and performs at least as well as any other policy in G.

A similar argument establishes the optimality of greedy service at node one. Suppose that at time $t = 0$, policy g idles the server in node one, and that after some random number of stages $L-1$, policy g first serves a job in node one at time $t(L)$. Because a zero rate of inventory cost reduction (reward rate) is earned during the first stage under g, and subsequent single-stage cost reduction rates cannot exceed $c_1\mu_1$, the modified policy $\tilde{g}$ as previously constructed performs strictly better than g in the sense of cumulative cost. As demonstrated above, this justifies the conclusion. □

Remarks: Although exhaustive service is not in general optimal, Theorem 1 provides a partial characterization of an optimal policy which is useful in the case of two queues and advances our intuition significantly in problems where multiple queues share the maximum $c\mu$ product. The proof of this property in the discounted cost case is similar.

One plausible extension to Theorem 1 is the restriction that upon *nonexhaustively* switching from a queue i, the server will only switch to a queue j with $c_j\mu_j > c_i\mu_i$ (an upstream queue). Unfortunately, much effort has not succeeded in establishing the validity of this property. For the majority of problem instances, we believe that by increasing in length, only an upstream queue can offer an incentive sufficient to justify a switch. We see this as a consequence of the fact that a reward rate of $c_i\mu_i$ is available by continuation in queue i, whereas the switching costs imply that a strictly lesser reward rate is available from queues downstream of i with respect to the $c\mu$ ordering.

Theorem 2, which follows, asserts the optimality of patience.

Theorem 2: There exists an optimal policy that is patient.

Proof:

Suppose an impatient policy, g, switches from queue i to j at $t = 0$, where $X_j(0) = 0$. Let τ denote the time at which g first serves a job and denote its queue by l. Construct $\tilde{g}$ to idle in queue i until τ, and to mimic g from that time forward (including the service of queue l at τ). Three cases are possible: $l = j$; $l = i$; $l \notin \{i, j\}$. If the first case occurs with probability one (and no additional switches are made under g), the performance is equal; otherwise, $\tilde{g}$ incurs a strictly lesser cumulative cost than g. The argument presented in Theorem 1 can be used to show that that there exists a policy $\tilde{g}'$ which performs at least as well as g in the sense of average cost per unit time. □

Remarks: The result is very intuitive. Since switches are instantaneous, it is optimal not to switch to an empty queue. Rather, switching cost is minimized by remaining to idle in i until the next job to be served arrives. Theorem 2 is very general and applies both to the expected discounted cost formulation of the problem as well as to problems with transition-dependent switching penalties with costs $K_{i,j}$ for each switch from i to j.

Although it seems clear that greedy policies are optimal for most problems of interest, this remains an open issue. Because the advantage of a greedy policy is not achieved along every sample path (indeed, idling performs better along some sample paths), it appears that a computationally stronger proof technique is required (if this property is in fact true).

In general, we expect that there exists a threshold type policy that is optimal. Take the case of two queues as an example. The server should switch from node 1 to 2 upon exhausting queue 1 if, and only if, the queue length $X_2(t)$ exceeds a threshold. For state $(x_1, x_2, n = 2)$, switch from queue 2 to 1 if, and only if, $x_1 \geq \Theta_2(x_2)$.

Although it is plausible to conjecture that $\Theta_2(x_2 + 1) \leq \Theta_2(x_2)$, our numerical determination of optimal policies suggests that this is not the case (at least when preemptive service is allowed). The threshold functions need not be monotonic. As an example, consider the two queue problem where $\mu_1 = \mu_2 = 0.6$, $\lambda_1 = \lambda_2 = 0.2$, $c_1 = 3, c_2 = 1$, $K_1 = 40$ and $K_2 = 20$. We numerically solved the dynamic program (for increasingly large state spaces) under the assumption of exponential service times and preemptive service, and the results are as follows. Suppose that the server is setup for queue 2. If there are 0 jobs at queue 2, the optimal threshold for switching to queue 1 is 1. That is, the server will switch to queue 1 as soon as one or more jobs are available at queue 1. When the

number of jobs in queue 2 is either 1 or 2, however, the optimal threshold for switching to queue 1 is 4. Finally, when there are at least three jobs in queue 2, it is optimal to switch to queue 1 when there are 3 or more jobs in queue 1. The intuition behind this behavior is that when there are no jobs at queue 2, or when there are a lot of jobs at queue 2, the server has less incentive to remain at queue 2. This is because when there are no jobs at queue 2, it is costly to wait for jobs to arrive at queue 2 when the server could be serving jobs at queue 1. If there are very few jobs (only 1 in our example) at queue 2, it might be more worthwhile to serve those jobs first before switching to queue 1, thereby possibly avoiding a switch back to queue 2 after exhaustive service of queue 1. When there are many jobs at queue 2, however, avoiding such a switch back becomes less likely. This example (and a similar example independently found by Koole [15] under the discounted cost criterion) clearly demonstrates for the case of preemptive scheduling that the threshold function is not, in general, monotonic. Numerically obtained examples such as this one indicate the complexity of problems of this type.

4. A Heuristic Policy

In this section, we develop a heuristic for the problem formulated in Section 2, where the queues are ordered such that $c_1\mu_1 \geq c_2\mu_2 \geq \ldots \geq c_N\mu_N$. We let x_i denote the queue length at queue i. We first develop a heuristic for the problem with two queues and then extend it to N queues.

4.1. Heuristic for Systems with Two Queues

By Theorem 2, we know that the server should not switch from queue 1 to queue 2 when queue 1 is not empty. To define a heuristic policy, we need to specify the conditions under which the server would switch from queue 2 to queue 1, when queue 2 is not empty. We also need to characterize a rule for idling (i.e., if the server is set up for queue i and queue i is empty, should it idle or switch to the other queue?). We first focus on the development of the rule for switching, then develop a rule for idling.

4.1.1. Rule for Switching

We assume that queues 1 and 2 are both nonempty ($x_1 > 0, x_2 > 0$), and focus on the question of when to switch from queue 2 to queue 1. We first state the following result which we will use in our heuristic.

Theorem 3: For state $(x_1, x_2, 2)$ and $z \in \{1, \ldots, x_2\}$, if $x_1 \geq Y/z$, then it is better to switch to queue 1 immediately than to serve exactly z consecutive jobs in queue 2 (along every sample path) prior to switching to queue 1, where

$$Y = \frac{(K_1 + K_2)(1 - \rho_1)\mu_1\mu_2}{c_1\mu_1 - c_2\mu_2} . \tag{4.4}$$

Proof: Suppose policy g is an optimal policy and chooses to process exactly z jobs of type 2, then switches to queue 1. We know by Theorem 1 that g must exhaust queue 1 upon switching to it. We construct the (non-stationary) policy g', which first switches to queue 1 and exhausts it, then switches back to queue 2 and serves z jobs, and then switches back to queue 1 to exhaust it once again. Under our construction, the processing time for the $i^{th}, i = 1, 2, ...,$ service from queue 1, and the $j^{th}, j = 1, ..., z$, service from queue 2, is the same for both policies. Therefore, the state of the queueing system is identical under g and g' upon completion of the sequence of actions described above, and thereafter g' takes exactly the same actions as g.

To compare g and g', observe that the x_1 jobs in queue 1 and the jobs arriving to queue 1 during the busy period generated by these x_1 jobs are processed earlier under g' than under g. In particular, this results in expected cost savings of $c_1 z \mu_2^{-1} x_1/(1 - \rho_1)$ for g', because the expected number of jobs served during a busy period generated by the x_1 jobs in queue 1 equals $x_1/(1 - \rho_1)$, and each of these jobs is expedited by z/μ_2 time units on average at a cost of c_1 per job per unit time. On the other hand, policy g' serves the z jobs in queue 2, on average, $x_1\mu_1^{-1}/(1 - \rho_1)$ time units later, thereby resulting in a loss of $c_2 z x_1 \mu_1^{-1}/(1 - \rho_1)$ compared to g. Policy g' also incurs an additional set-up cost of $K_1 + K_2$. Thus, (using the concluding argument of Theorem 2 to extend our inequality for cumulative cost differences to average cost per unit time differences) we see that g' outperforms g if, and only if,

$$\frac{c_1 z x_1}{\mu_2(1 - \rho_1)} - \frac{c_2 z x_1}{\mu_1(1 - \rho_1)} \geq K_1 + K_2. \tag{4.5}$$

Simplification yields a threshold as a function of z. Switch if, and only if,

$$x_1 \geq \frac{(K_1 + K_2)(1 - \rho_1)\mu_1\mu_2}{(c_1\mu_1 - c_2\mu_2)z} . \tag{4.6}$$

□

Remarks: A similar property was previously investigated in Gupta et al. [8] for the special case of homogeneous service ($\mu_1 = \mu_2$). Although the statement of the result is fairly weak, it is useful to us in the development of the heuristic. It is tempting to say that if $x_1 \geq Y$, then an optimal policy must switch immediately. This is incorrect, as the proof and its calculations restrict attention to (open loop) policies that serve a deterministic number of jobs in queue 2 prior to switching. In general, a closed loop policy will use the realization to selectively remain or switch, which is very difficult to capture computationally.

The limited generality of Theorem 3 points to the difficulty of precisely defining an optimal threshold within G^{PM}. Our objective is to provide an *analytical* switching rule which results in an effective policy. We further restrict our attention to defining a constant threshold rule. Under our heuristic, the server switches to queue 1 whenever $n(t) = 2$ and $X_1(t)$ exceeds a constant threshold, x_T (not depending on $X_2(t)$).

Our approach intentionally views the system simplistically by considering only the evolution of the mean queue length in node 1 and by applying Theorem 3 as if it applied to dynamic (closed-loop) policies in general. Assume an initial state $(x_1, x_2, 2)$ at time $t = 0$. We suppose that if the number in jobs in queue 1, x_1, is at least as large as $Y/(z+1)$, then this implies that a good policy will switch to queue 1 prior to serving $z+1$ jobs in queue 2. Based on this property (which relates x_1 to the number of jobs in queue 2) we desire to deduce a condition that justifies switching based only on the condition $x_1 \geq x_T$. We know that $Y/(z+1) \leq x_T \leq Y/z$ for some $z \geq 0$, so we solve for such a z.

Suppose that $x_1 = Y/(z+1)$ for some z. We thus anticipate that $z+1$ consecutive services in queue 2 is undesirable (by Theorem 3) and thus the mean queue length at node 1 will exceed x_T along this action sequence. That is, prior to the completion of z jobs in queue 2, we expect queue 1 to exceed the threshold x_T and thus avoid the undesirable situation of processing $z+1$ jobs consecutively in queue 2. Thus, we expect that a good choice of x_T will satisfy

$$\frac{Y}{z+1} + \lambda_1(\frac{z}{\mu_2}) \geq x_T \quad , \quad \frac{Y}{z+1} \leq x_T \leq \frac{Y}{z} \, . \tag{4.7}$$

We further simplify the problem by choosing $x_T = Y/z^*$, where z^* approximately solves

$$\frac{Y}{z+1} + z\lambda_1/\mu_2 = \frac{Y}{z} \, . \tag{4.8}$$

Since (4.8) reduces to $z^2(z+1) = Y\mu_2/\lambda_1$, by approximating $z^2(z+1)$ by z^3, we obtain a unique approximate solution as $z^* = (Y\mu_2/\lambda_1)^{1/3}$. Since $x_T = Y/z^*$, we get $x_T = (Y^2\lambda_1/\mu_2)^{1/3}$. Our policy permits a switch from queue 2 to queue 1 when $x_2 > 0$ and $x_1 \geq x_T$, where

$$x_T = ((\frac{(K_1+K_2)(1-\rho_1)\mu_1\mu_2}{c_1\mu_1 - c_2\mu_2})^2\lambda_1/\mu_2)^{1/3}. \tag{4.9}$$

Since the right-hand side of (4.9) is not an integer in general, we round it off to obtain our threshold. By Theorem 2 it is never optimal to switch from queue 2 to queue 1 when $x_1 = 0$, therefore even if the right-hand side rounds off to 0, we let $x_T = 1$.

4.1.2. Rule for Idling

In order to complete the characterization of our heuristic policy, we need to specify an idling rule. In particular, assume an initial state $(x_1, x_2, 2)$ with $x_2 = 0$, and the server is set-up for queue 2. If $x_1 = 0$ as well, the optimal policy is to idle, by Theorem 2. For the case $x_1 > 0$, we derive a crude idling rule, based on quantifying the tradeoffs between switching and idling. We focus only on the duration of time in which the server switches to queue 1, exhausts queue 1, and then switches back to queue 2. Idling at queue 2 permits the queue at 1 to grow, and therefore the set-up costs will be shared by a larger number of type 1 jobs if the server idles. In particular, if the server idles until the number of jobs in queue 1 reach I_1, and then switches to queue 1, the switching costs will be shared by the $I_1/(1-\rho_1) = I_1\mu_1/(\mu_1 - \lambda_1)$ jobs that the server will process on average before queue 1 is exhausted. (Clearly, jobs of type 2 could arrive before the number of jobs in queue 1 reaches I_1, but we ignore that.) However, if the server waits until the number in queue 1 reaches I_1, then the jobs in queue 1 will also incur additional costs since each job will be finished later than if the server had immediately switched. For $x_1 < I_1$ the server will idle, on average, $(I_1 - x_1)/\lambda_1$ time units before the number in queue 1 reaches I_1. Hence, the jobs in queue 1 will be finished $(I_1 - x_1)/\lambda_1$ time units later than they would have if the server had immediately switched. This will result in an additional per job cost of $c_1(I_1 - x_1)/\lambda_1$. We use this crude analysis in specifying the following optimization problem for choosing the value of I_1.

$$\min_{I_1} \frac{K_1+K_2}{\frac{I_1\mu_1}{\mu_1-\lambda_1}} + \frac{c_1(I_1-x_1)}{\lambda_1}. \tag{4.10}$$

We can solve for the best value of I_1 by differentiating the right hand side with respect to I_1, and we obtain

$$I_1 = \sqrt{\lambda_1(K_1 + K_2)(\mu_1 - \lambda_1)/c_1\mu_1} \ . \tag{4.11}$$

Once again, since the right-hand side of (4.11) is not an integer, we round it off (except when it rounds off to zero, in which case, we let $I_1 = 1$).

In the case where the state is $(0, x_2, 1)$, we use the same result even though the issue is even more complicated by the fact that it may be optimal to switch from queue 2 at a certain point if x_1 reaches x_T. Thus, the server switches to queue 2 if $x_2 > I_2$, where I_2 is the nearest integer (greater than or equal to 1) to $\sqrt{\lambda_2(K_1 + K_2)(\mu_2 - \lambda_2)/c_2\mu_2}$. Although these approximations appear crude, the results in Section 4 demonstrate that this approach works very well.

We are now ready to state our heuristic control rule in full:

Heuristic Policy for Two Parallel Queues

1. If the server is currently at node 1, and $x_1 > 0$, then serve one more job at node 1.

2. If the server is currently at node 2, and $x_2 = 0$, then set I_1 equal to the nearest integer (greater than or equal to 1) to $\sqrt{\lambda_1(K_1 + K_2)(\mu_1 - \lambda_1)/c_1\mu_1}$. If $x_1 \geq I_1$ then switch, otherwise idle.

3. If the server is currently at node 1, and $x_1 = 0$, then set I_2 equal to the nearest integer (greater than or equal to 1) to $\sqrt{\lambda_2(K_1 + K_2)(\mu_2 - \lambda_2)/c_2\mu_2}$. If $x_2 \geq I_2$ then switch, otherwise idle.

4. If the server is currently at node 2, and $x_2 > 0$; then if $c_1\mu_1 = c_2\mu_2$ set $x_T = +\infty$, otherwise set x_T equal to the nearest integer (greater than or equal to 1) to

$$((\frac{(K_1 + K_2)(1 - \rho_1)\mu_1\mu_2}{c_1\mu_1 - c_2\mu_2})^2\lambda_1/\mu_2)^{1/3}.$$

   If $x_1 \geq x_T$, then switch to queue 1, otherwise process another unit of type 2.

4.2. Heuristic for Systems with N Queues

Using the ideas developed previously for 2 queues, we extend our heuristic to the case where the system has any number of queues. We assume that the system consists of N queues numbered such that $c_1\mu_1 \geq c_2\mu_2 \geq \ldots \geq c_N\mu_N$. To begin, assume that the server is currently serving queue

i, and that $x_i > 0$. In this case, we consider whether the server should switch to queues $1, \ldots, i-1$. We disallow nonexhaustive switches from i to the "downstream" (with respect to the $c\mu$ ordering) queues $i+1, \ldots, N$. In deciding whether to switch to any of these queues, our heuristic first computes the number required in queues $1, 2, ..., i-1$, to make a switch from i worthwhile. To do this, we ignore the existence of all other queues except the present queue i, and candidate queue $j < i$. Then, we use (4.9) to compute the threshold $x_T(i,j)$ required to switch from queue i to queue $j = 1, \ldots, i-1$:

$$x_T(i,j) = ((\frac{(K_i + K_j)(1-\rho_j)\mu_i\mu_j}{c_j\mu_j - c_i\mu_i})^2 \lambda_j/\mu_i)^{1/3} \ . \tag{4.12}$$

Once these thresholds have been computed, our heuristic checks the number of jobs in each queue, x_j, to see whether queue j is a candidate for a switch (i.e. $x_j \geq x_T(i,j)$). When there is more than one candidate queue, our heuristic makes use of the notion of reward rates by choosing the queue with the highest reward rate as follows:

On average, the server processes μ_i jobs for every unit of time spent processing jobs from queue i. Since, for each time unit these jobs wait the system incurs a cost of c_i, the server is decreasing the costs for the system (or earning rewards) at the rate of $c_i\mu_i$ when serving queue i. Our heuristic computes the reward rate that is earned by switching from queue i to j, exhausting queue j, and then switching back to queue j. We denote this reward rate by

$$\varphi_{ij} = c_j\mu_j - (K_i + K_j)(\mu_j - \lambda_j)/x_j \ . \tag{4.13}$$

This computation follows from the assumption that when the server switches to queue j, and then eventually back to queue i, additional set-up costs of $K_i + K_j$ are incurred. By remaining in queue j until the end of its busy period, the server can earn rewards at a rate of $c_j\mu_j$ for an expected duration of $x_j/(\mu_j - \lambda_j)$. Therefore, (4.13) gives us the expected reward rate of a switch to queue j, and a switch back to i after the exhaustion of queue j. Although the server does not have to switch back to queue i immediately upon completion of queue j, we include K_i in our calculation because the unfinished jobs left behind in queue i eventually require an extra set-up to be served.

Clearly, once the server has switched to queue j, there is no guarantee that the server will exhaust it. It may in fact switch to another higher priority queue before exhausting queue j.

Equation (4.13) provides an estimate of the average reward rate that can be earned by switching to queue j. The dynamic implementation of the heuristic preserves the option to later switch to a more attractive queue as events unfold. Among all queues $j < i$, with $x_j \geq x_T(i,j)$, our heuristic switches to the queue with the largest reward rate φ_{ij}.

To complete the specification of our heuristic, we also need to define its action when the system is set up for queue i, and $x_i = 0$. In this case, our heuristic once again computes the idleness threshold for each queue. For each queue j, we derive the threshold,

$$I_j = \sqrt{\lambda_j K_j(\mu_j - \lambda_j)/c_j\mu_j} \; . \tag{4.14}$$

Among all those queues with $x_j \geq I_j$, our heuristic chooses the one with the highest reward rate. In this case, since the server is not leaving behind unfinished jobs, the set-up cost for queue i is not included in our reward rate calculations and the reward rate is given by

$$\phi_j = c_j\mu_j - K_j(\mu_j - \lambda_j)/x_j \; . \tag{4.15}$$

We are now ready to state our heuristic formally:

Heuristic Policy for N Parallel Queues

1. If the server is set up for queue i with $x_i > 0$: For $j = 1, ...i-1$, compute $x_T(i,j)$ using (4.12). Let $\sigma = \emptyset$. For $j = 1, \ldots, i-1$, if $x_j \geq x_T(i,j)$, then $\sigma = \sigma \cup j$. If σ is empty, then process one more unit from queue i. Otherwise, for all $j \in \sigma$ compute φ_{ij} using (4.13). Switch to the queue $j \in \sigma$ with the maximum φ_{ij} value.

2. If the server is set up for queue i with $x_i = 0$: For $j = 1, \ldots, N$, compute I_j using (4.14). Let $\sigma = \emptyset$. For $j = 1, \ldots, N$, if $x_j \geq I_j$ then $\sigma = \sigma \cup j$. If σ is empty, then idle until the next arrival to the system. Otherwise, for all $j \in \sigma$, compute ϕ_j using (4.15). Switch to the queue $j \in \sigma$ with the maximum ϕ_j value.

The heuristic which we have described above, is known to have optimal characteristics in the following limiting cases:

1. $K_i = 0$ *for all i:* If all set-up costs are zero, our heuristic reduces to the $c\mu$ rule, which is known to be optimal.

2. *Symmetrical Systems:* If all queues are identical with respect to holding costs, processing time distribution, arrival rates, and set-up costs, the heuristic serves each node exhaustively and upon switching chooses the longest queue. These policies have been shown to be optimal among the set of non-idling policies (Liu et al. [19], Rajan and Agrawal [23]). The optimal idling policy is not known.

Having developed our heuristic, and specified the cases where it has optimal characteristics, we next report the results of a numerical investigation in which we tested its performance.

5. A Numerical Study

The real test of any heuristic is its performance with respect to the optimal solution. In the problem considered here, however, the optimal solution is not known, except for a few special cases. Solving the problem using dynamic programming becomes very difficult, in general. Cases without exponential processing times are more complex than the exponential case. Even with exponential processing times, we need to take into account the fact that the state space is countably infinite. We solved dynamic programs with increasingly large state spaces and computed the average cost per unit time in each case. Once a desired level of accuracy was reached, we stopped. In a problem with only 4 queues, however, truncating the state space such that the content of each queue is allowed to vary between 0 and 99 still leads to a state space of size 4×10^8. In fact, solving the dynamic programming equations for systems with more than 2 queues is computationally very expensive.

Given the difficulty of computing the optimal solution even for the exponential case, we chose to compare our heuristic to the optimal pure Markov policy for systems with 2 and 3 queues and to compare it to other rules from the literature for systems with 4 queues. The cases that we tested included symmetric as well asymmetric queues, high and moderate utilization, and both equal and different holding costs for different job classes. For each case, we tested our heuristic by simulating 50000 job completions from the system. We repeated the simulation 10 times and computed the sample mean of the performance obtained in each run. We report these means and 99% confidence intervals in summary tables.

Ex.	c_1	c_2	μ_1	μ_2	λ_1	λ_2	K_1	K_2	*OPT.*	*HEURISTIC*	*EXH*	$c\mu$
1	1	1	0.6	0.6	0.2	0.2	5	5	2.69	2.69 ± 0.01	2.69 ± 0.01	2.69 ± 0.01
2	1	1	0.6	0.6	0.2	0.2	7	5	2.83	2.83 ± 0.01	2.84 ± 0.01	2.84 ± 0.01
3	1	1	0.6	0.6	0.2	0.2	50	50	6.09	6.13 ± 0.04	8.91 ± 0.11	8.91 ± 0.11
4	2	1	0.6	0.6	0.2	0.2	5	5	3.46	3.47 ± 0.05	3.69 ± 0.09	3.63 ± 0.13
5	2	1	0.6	0.6	0.2	0.2	7	5	3.62	3.64 ± 0.06	3,.83 ± 0.13	3.79 ± 0.18
6	2	1	0.58	0.58	0.21	0.21	10	10	4.97	5.06 ± 0.13	5.22 ± 0.19	5.42 ± 0.31
7	4	1	0.56	0.56	0.22	0.22	500	20	23.4	23.8 ± 0.3	39.9 ± 0.3	52.1 ± 0.5
8	5	1	0.56	0.56	0.22	0.22	10	100	14.2	14.3 ± 0.2	17.1 ± 0.4	16.9 ± 0.5
9	5	1	0.70	0.70	0.15	0.15	1	100	6.8	7.0 ± 0.1	8.9 ± 0.1	9.5 ± 0.2
10	3	1	0.54	0.54	0.23	0.23	10	50	12.0	12.3 ± 0.4	14.7 ± 0.5	13.1 ± 0.6
11	3	1	0.54	0.54	0.15	0.31	10	50	10.9	11.1 ± 0.4	13.7 ± 0.6	11.9 ± 0.7
12	3	1	0.54	0.54	0.31	0.15	10	50	12.7	13.0 ± 0.4	15.0 ± 0.5	13.5 ± 0.7
13	3	1	0.54	0.54	0.31	0.15	10	800	27.6	28.4 ± 1.0	47.2 ± 0.8	69.7 ± 1.2
14	4	1	0.53	0.53	0.21	0.26	100	200	24.0	24.4 ± 0.3	32.1 ± 1.2	37.7 ± 1.5
15	4	1	0.3	0.65	0.15	0.20	10	10	10.3	10.5 ± 0.4	11.3 ± 0.6	11.2 ± 0.5
16	4	1	0.3	0.65	0.15	0.20	10	500	21.9	22.3 ± 0.4	31.6 ± 0.5	39.3 ± 1.5
17	4	1	0.3	0.65	0.15	0.20	500	10	21.9	22.3 ± 0.4	31.6 ± 0.5	39.3 ± 1.5
18	3	1	0.35	0.70	0.25	0.05	50	50	11.7	11.9 ± 0.5	12.4 ± 0.5	12.8 ± 0.6
19	3	1	0.35	0.70	0.25	0.05	500	10	15.7	15.9 ± 0.4	22.1 ± 0.6	24.2 ± 0.7
20	3	1	0.35	0.70	0.25	0.05	100	200	14.0	14.4 ± 0.3	17.1 ± 0.6	18.4 ± 0.6
21	2	1	0.25	0.65	0.05	0.30	100	100	7.3	7.5 ± 0.2	9.1 ± 0.1	10.5 ± 0.5
22	3	1	0.45	0.60	0.15	0.25	100	100	12.6	12.7 ± 0.3	16.6 ± 0.2	19.7 ± 0.3

Table 1: Input Data and Results for Examples 1-22

We first tested our heuristic on a variety of problems with 2 queues. The data for these 22 problems as well as the results obtained from the heuristic and from the dynamic programming solution are displayed in Table 1. In all of these cases, the processing times were exponential for simplicity (though our heuristic does not require this assumption). We assumed that preemptive service was allowed (preemptive resume and preemptive repeat being equivalent for exponential service times). Hence, in these examples, we adapted our heuristic to the preemptive case. In particular, as soon as the number in queue 1 reached x_T, our heuristic switched to queue 1.

The input data and the results for the examples with two queues are displayed in Table 1. The examples include a variety of situations. For example, in the first 3 examples, queues 1 and 2 are identical with respect to their processing times, arrival times, and holding costs. In Examples 4-10, queue 1 and queue 2 have different holding and set-up costs, but the same arrival and service rates. In Examples 11-14, the queues have the same service rates, but different arrival rates and costs.

Ex.	μ	λ_1	λ_2	λ_3	K_1	K_2	K_3	*Opt.*	*Heuristic*	% Dif.	*EXH*	$c\mu$
23	0.6	0.2	0.1	0.1	5	5	5	5.42	5.64 ± 0.10	4.1	6.2 ± 0.2	5.8 ± 0.2
24	0.6	0.2	0.1	0.1	1	10	100	9.11	9.30 ± 0.19	2.1	11.7 ± 0.2	11.7 ± 0.3
25	0.6	0.2	0.1	0.1	200	200	200	23.10	24.16 ± 0.24	4.6	43.0 ± 0.2	48.5 ± 0.5
26	0.6	0.2	0.1	0.1	15	10	10	6.97	7.15 ± 0.14	2.6	7.5 ± 0.2	7.3 ± 0.2
27	0.6	0.05	0.30	0.05	5	5	5	4.46	4.55 ± 0.06	2.0	5.0 ± 0.2	4.7 ± 0.2
28	0.6	0.05	0.30	0.05	1	10	100	6.86	6.95 ± 0.05	1.3	8.5 ± 0.1	8.5 ± 0.2
29	0.6	0.05	0.30	0.05	100	10	1	8.31	8.42 ± 0.07	1.3	8.6 ± 0.2	9.1 ± 0.2
30	0.6	0.05	0.30	0.05	50	50	50	9.67	9.95 ± 0.07	2.9	10.7 ± 0.2	11.6 ± 0.5
31	0.65	0.07	0.08	0.20	5	5	5	2.78	2.89 ± 0.02	3.9	3.1 ± 0.1	2.9 ± 0.1
32	0.65	0.07	0.08	0.20	1	10	100	7.19	7.41 ± 0.02	3.1	9.9 ± 0.1	10.5 ± 0.2
33	0.65	0.07	0.08	0.20	200	200	200	18.98	19.49 ± 0.05	2.7	37.1 ± 0.3	40.5 ± 0.6
34	0.65	0.07	0.08	0.20	15	15	10	4.30	4.40 ± 0.03	2.3	4.5 ± 0.1	4.5 ± 0.1
35	0.65	0.07	0.08	0.20	50	20	10	6.26	6.44 ± 0.03	2.8	6.5 ± 0.1	6.6 ± 0.2
36	0.55	0.30	0.07	0.08	1	10	100	13.85	14.26 ± 0.44	3.0	16.8 ± 0.7	15.6 ± 0.6
37	0.55	0.30	0.07	0.08	100	10	1	15.80	16.72 ± 0.43	5.8	18.2 ± 0.6	18.3 ± 0.7
38	0.55	0.30	0.07	0.08	50	50	50	16.73	17.28 ± 0.29	3.3	19.0 ± 0.5	18.9 ± 0.7
39	0.55	0.30	0.07	0.08	200	200	200	26.86	28.40 ± 0.44	5.7	38.2 ± 0.6	44.9 ± 0.8
40	0.55	0.30	0.07	0.08	40	20	10	13.81	14.34 ± 0.28	3.8	15.8 ± 0.4	14.7 ± 0.4

Table 2: Input Data and Results for Examples 23-40

Example	c_1	c_2	c_3	c_4	μ_1	μ_2	μ_3	μ_4	λ_1	λ_2	λ_3	λ_4
41-45	2	1	0.5	0.2	2	2	1	1	0.5	0.4	0.2	0.2
46-52	2	1.2	0.5	0.2	1	1	2	2	0.2	0.2	0.5	0.5
53-60	1	1	1	1	5	2	1	0.5	1.125	0.45	0.225	0.1125
61-68	1	1	1	1	5	2	1	0.5	2.5	0.6	0.05	0.025
69-76	1	1	1	1	5	2	1	0.5	0.25	0.1	0.3	0.25
77- 84	1	1	1	1	5	2	1	0.5	0.625	0.25	0.125	0.0625
85-92	1	1	1	1	5	2	1	0.5	0.25	0.1	0.05	0.175

Table 3: Input Data for Examples 41-92

Finally, in Examples 15-22, the two queues have nothing in common. If in one example, the queue with the high utilization or holding cost had a high set-up cost, in the next example, we tested the case where the queue with the low utilization or holding cost had a high set-up cost, to see the effect of such variations on the heuristic. We also chose examples with a wide range of utilization, from a low of 0.42 (Example 9) to a high of 0.89 (Example 14).

We also compared our heuristic to other policies from the literature. We compared our heuristic to the exhaustive and gated polling rules as well as the $c\mu$ rule. The exhaustive policy (EXH) serves each of the queues in an exhaustive and cyclic manner. That is, the server finishes all jobs of type 1, then if there are any jobs of type 2, switches to queue 2 and exhausts all the jobs of type 2, and so forth. We found that not switching to an empty queue improved the performance of this rule, hence we prevented switching to an empty queue by forcing the server to switch to the next nonempty queue in the polling cycle. The gated policy (GATED) does not exhaust the jobs at each queue; rather the server gates all the jobs present at the time its set-up is completed, and serves only those jobs. Once again, we presented switches to empty queues (In all of the examples in Tables 1 and 2, GATED performed worse than EXH, therefore we only present the results for EXH). Finally, as a third alternative, we tested the $c\mu$ rule as a heuristic. In this case, at each decision epoch, the server processed a job from the queue with the highest $c\mu$ index.

The results in Table 1 indicate that our heuristic performed very well. These results are representative of our experience with the heuristic. The maximum deviation from the optimal solution in G^{PM} was 3%. The average suboptimality of the heuristic for Examples 1-22 was 1.5%, compared to 27% for EXH and 38% for $c\mu$. Our heuristic consistently performed well in examples with low or high utilization and low or high set-up costs. Given the fact that in most practical applications, even the input data can not be estimated to this level of accuracy, the results for the two queue case are very encouraging.

We note that in cases where the set-up costs were very low, there was much less difference between the heuristic and the best of the $c\mu$ and EXH heuristics. However, as the set-up costs became larger, $c\mu$ and EXH consistently became worse. This is demonstrated in Figure 1. In this case, we tested an example with two queues where $c_2 = 2, c_1 = 1$, $\mu_1 = \mu_2 = 0.6$, and $\lambda_1 = \lambda_2 = 0.2$. The set-up costs $K_1 = K_2$ were varied from 2 to 256. As Figure 1 clearly demonstrates, the heuristic remained close to the optimal policy following its concave trend as the set-up costs were increased,

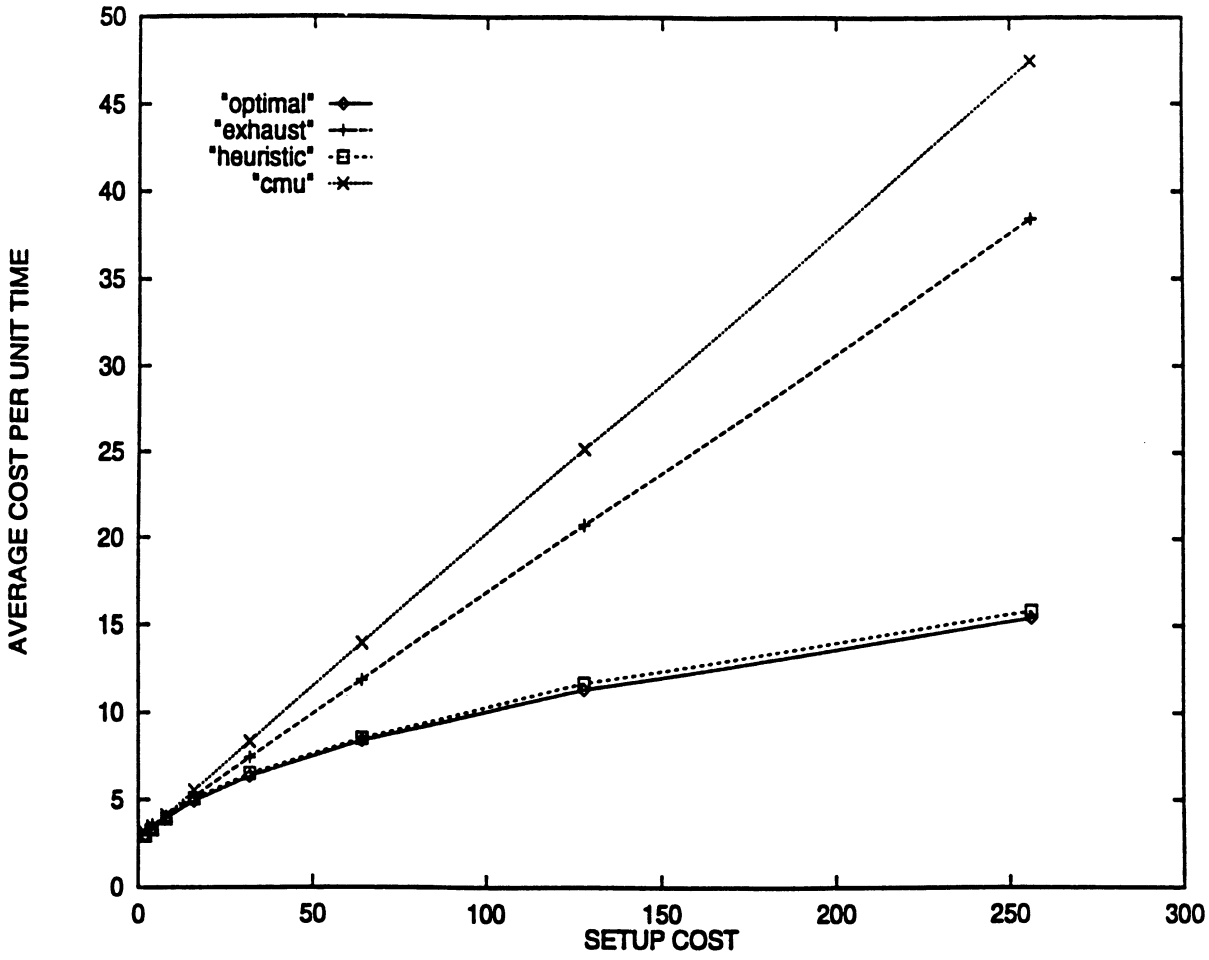

Figure 1: Average Cost per Unit Time as a Function of Set-up Costs

while the two other heuristics performed worse as the set-up costs increased, with the costs under these two policies exhibiting linear trends as the set-up costs were increased.

We next tested our heuristic on a variety of examples with 3 queues. The input data and the results for the three queue examples are displayed in Table 2. In all of these cases, the service rates were identical for each of the three queues and were equal to μ, and the holding costs were 4, 2 and 1 for queues 1, 2 and 3 respectively. Processing was assumed to be non-preemptive. Once again, our examples represent a variety of different situations. In Examples 23-26, the queue with the highest holding cost has the highest arrival rate. In Examples 27-30, the queue with the second highest holding cost has the highest arrival rate and in Examples 31-35, the queue with the lowest holding cost has the highest arrival rate. The utilization of the server was again varied to test the effect of utilization on the heuristic. The results in Table 2 show that despite the fact that the deviation from the optimal solution increased for examples with 3 queues, the heuristic still performed very well. The average suboptimality of the heuristic was around 3%, and was again much lower than the suboptimality of the EXH and $c\mu$ rules.

We next tested our heuristic on a set of examples with 4 queues. Non-preemptive service was

assumed in these examples. Given the computational difficulty of computing the optimal solution in this case, we chose to compare our heuristic only to the other policies from the literature.

The 52 examples on which we tested our heuristic include 7 sets of examples. In each set of examples, the holding cost, service rate and arrival rates were the same. The set-up costs were varied for each example to observe the effect of different set-up costs on the heuristic. The data on the holding costs, service rates and arrival rates for Examples 41-92 is displayed in Table 3. Examples 41-52 are examples where both the c and μ values for the different queues are different. In Examples 53- 92 all of the queues have the same c values, but different μ values. In Examples 53-76, the utilization for the system is high (0.9), while in Examples 77-92, the utilization for the system is low (0.5). In Examples 53-60, all of the four queues share the utilization equally (i.e., $\rho_i = 0.225$ for $i = 1, \ldots, 4$), while in Examples 61-68, the server spends the greatest proportion of time serving the queue with the highest $c\mu$ index. Similarly, in Examples 69-76, the server spends the greatest proportion of time serving the queue with the lowest $c\mu$ index. For the examples with low utilization, in Examples 77-84, the total utilization is allocated equally to the different queues, and in Examples 85-92, the server spends the greatest proportion of the time serving the queue with the lowest $c\mu$ index. Thus, Examples 41-92 cover a wide variety of situations, with low and high utilization, low and high arrival and service rates, and different combinations of set-up costs.

The results for Examples 41-76 are displayed in Table 4, and the results for Examples 77-92 are displayed in Table 5. These results strongly support the superior performance of our heuristic over several important policies taken from the literature. Our heuristic outperformed the other rules on every problem instance. For cases where the set-up costs were very low, the performance of our heuristic was again only slightly better than the performance of the other policies, and the $c\mu$ rule also performed well. In all other cases, our heuristic outperformed the other policies very significantly, and the results in Tables 4 and 5 include cases where the nearest competitor to the heuristic resulted in costs per unit time that were nearly 6 times worse than the heuristic. However, we note that the exhaustive, gated and $c\mu$ policies are rather naive policies, and it is possible to develop better static policies for the set-up cost problem similar to those developed for the set-up time problem by Boxma et al. [2]. Our focus is on the development of dynamic rules, and optimization within the class of static rules is beyond the scope of this paper.

Acknowledgment:

Ex.	K_1	K_2	K_3	K_4	EXH	GATED	$c\mu$	HEURISTIC
41	40	40	40	40	26.8 ± 0.4	32.1 ± 0.2	34.5 ± 0.3	18.6 ± 0.3
42	2	20	40	80	22.2 ± 0.4	26.0 ± 0.3	25.2 ± 0.2	13.8 ± 0.2
43	80	40	20	2	27.2 ± 0.4	32.8 ± 0.2	36.8 ± 0.3	18.8 ± 0.3
44	100	100	100	100	56.3 ± 0.6	69.9 ± 0.5	81.9 ± 0.6	29.4 ± 0.5
45	1	10	500	500	107.4 ± 1.6	134.3 ± 1.3	149.5 ± 1.5	30.7 ± 0.7
46	5	5	5	5	9.4 ± 0.5	9.9 ± 0.7	8.0 ± 0.3	7.8 ± 0.3
47	40	40	40	40	23.8 ± 0.2	28.8 ± 0.3	33.2 ± 0.3	17.9 ± 0.4
48	100	100	100	100	48.4 ± 1.6	61.4 ± 1.1	76.8 ± 0.5	27.0 ± 0.8
49	2	20	40	80	23.9 ± 0.3	29.1 ± 0.4	32.7 ± 0.4	14.8 ± 0.4
50	80	40	20	2	20.2 ± 0.3	23.6 ± 0.3	26.6 ± 0.4	18.3 ± 0.4
51	1	1	500	500	130.3 ± 4.3	171.6 ± 5.2	222.8 ± 2.2	32.1 ± 0.6
52	500	500	1	1	90.2 ± 3.4	114.5 ± 2.7	159.2 ± 1.3	40.1 ± 0.6
53	5	5	5	5	20.1 ± 1.3	19.9 ± 1.3	11.8 ± 0.3	11.3 ± 0.2
54	40	40	40	40	34.0 ± 2.1	38.1 ± 2.0	45.4 ± 0.2	25.0 ± 0.9
55	100	100	100	100	57.8 ± 1.5	69.5 ± 1.8	103.1 ± 0.4	36.8 ± 1.6
56	2	20	40	80	28.3 ± 1.4	30.6 ± 2.1	28.2 ± 0.3	19.2 ± 0.8
57	80	40	20	2	36.1 ± 1.0	41.0 ± 1.9	54.0 ± 0.3	24.1 ± 0.5
58	500	500	1	1	149.2 ± 4.2	189.7 ± 6.5	352.3 ± 2.1	55.7 ± 1.3
59	1	1	500	500	86.1 ± 1.3	105.9 ± 2.4	144.2 ± 0.6	37.8 ± 0.6
60	1	1000	1	1	134.8 ± 10.1	173.1 ± 5.3	300.3 ± 2.1	36.7 ± 0.9
61	5	5	5	5	13.9 ± 0.6	17.5 ± 1.3	11.2 ± 0.4	9.8 ± 0.2
62	40	40	40	40	28.5 ± 0.5	39.5 ± 1.0	42.9 ± 0.5	20.6 ± 0.3
63	100	100	100	100	54.1 ± 0.9	77.4 ± 0.7	97.5 ± 0.8	30.7 ± 0.1
64	2	20	40	80	18.7 ± 0.5	24.0 ± 1.6	19.2 ± 0.5	14.4 ± 0.3
65	80	40	20	2	34.6 ± 0.6	49.4 ± 0.7	58.6 ± 0.6	23.3 ± 0.5
66	500	500	1	1	194.7 ± 5.2	296.2 ± 7.8	427.5 ± 4.3	59.0 ± 0.7
67	1	1	500	500	41.4 ± 0.7	48.6 ± 1.8	41.7 ± 1.4	21.0 ± 0.7
68	1	1000	1	1	190.7 ± 3.2	288.3 ± 8.0	410.6 ± 4.8	46.6 ± 0.6
69	5	5	5	5	15.1 ± 1.2	14.0 ± 1.3	11.1 ± 0.5	10.3 ± 0.9
70	40	40	40	40	23.5 ± 0.9	25.5 ± 0.9	31.4 ± 1.4	20.7 ± 1.3
71	100	100	100	100	37.9 ± 0.8	45.2 ± 0.7	66.6 ± 1.6	30.4 ± 1.1
72	2	20	40	80	22.7 ± 0.6	24.6 ± 1.0	29.4 ± 1.4	18.7 ± 0.8
73	80	40	20	2	22.6 ± 0.6	23.4 ± 1.0	28.7 ± 1.3	17.6 ± 0.8
74	500	500	1	1	70.6 ± 1.7	84.7 ± 1.9	135.5 ± 2.1	34.1 ± 0.9
75	1	1	500	500	77.8 ± 2.6	104.2 ± 2.9	174.3 ± 2.8	37.8 ± 0.8
76	1	1000	1	1	57.7 ± 0.9	67.8 ± 1.4	92.1 ± 2.8	21.5 ± 0.9

Table 4: Data and Results for Examples 41-76

Example	K_1	K_2	K_3	K_4	EXH	GATED	$c\mu$	HEURISTIC
77	5	5	5	5	4.2 ± 0.1	4.4 ± 0.1	4.2 ± 0.1	3.9 ± 0.1
78	40	40	40	40	22.8 ± 0.2	24.8 ± 0.2	25.3 ± 0.3	13.0 ± 0.1
79	100	100	100	100	54.5 ± 0.4	59.8 ± 0.5	61.4 ± 0.7	20.7 ± 0.1
80	2	20	40	80	13.1 ± 0.1	14.3 ± 0.2	14.4 ± 0.2	9.2 ± 0.1
81	80	40	20	2	27.6 ± 0.2	30.1 ± 0.3	31.0 ± 0.4	13.6 ± 0.1
82	500	500	1	1	193.1 ± 0.8	211.3 ± 2.0	219.0 ± 3.0	38.1 ± 0.2
83	1	1	500	500	75.7 ± 1.3	84.0 ± 1.3	85.1 ± 1.5	20.5 ± 0.3
84	1	1000	1	1	164.6 ± 1.7	180.8 ± 1.8	83.5 ± 3.1	23.6 ± 0.2
85	5	5	5	5	3.2 ± 0.1	3.2 ± 0.1	3.2 ± 0.1	3.0 ± 0.1
86	40	40	40	40	14.6 ± 0.1	16.4 ± 0.1	16.8 ± 0.3	10.2 ± 0.1
87	100	100	100	100	34.3 ± 0.3	39.1 ± 0.2	40.2 ± 0.8	16.2 ± 0.2
88	2	20	40	80	12.4 ± 0.9	14.5 ± 0.1	14.7 ± 0.3	8.7 ± 0.1
89	80	40	20	2	15.0 ± 0.1	16.4 ± 0.1	16.9 ± 0.4	9.4 ± 0.1
90	500	500	1	1	97.9 ± 1.0	108.3 ± 0.9	112.1 ± 3.2	25.3 ± 0.2
91	1	1	500	500	69.4 ± 0.9	83.9 ± 0.7	85.8 ± 1.9	20.2 ± 0.2
92	1	1000	1	1	74.1 ± 1.1	81.0 ± 1.0	81.1 ± 3.2	15.3 ± 0.1

Table 5: Data and Results for Examples 77-92

We would like to thank Matthew Keblis for his help with the computations and Professor Demosthenis Teneketzis for many fruitful discussions. We are grateful to an anonymous referee for his detailed comments which improved the paper. This research is sponsored, in part, by NSF Grant No: DDM-9308290 to the University of Michigan.

References

[1] J.S. Baras, D.J. Ma, and A.M. Makowski, K competing queues with geometric service requirements and linear costs: the μc rule is always optimal, *Systems Control Letters* **6** (1985) 173-180.

[2] O.J. Boxma, H. Levy, and J.A. Weststrate, "Efficient Visit Frequencies for Polling Tables: Minimization of Waiting Cost," *Queueing Systems: Theory and Applications*, **9** (1991) 133-162.

[3] C. Buyukkoc, P. Varaiya, and J. Walrand, The $c\mu$-rule revisited, *Advances in Applied Probability* **17** (1985) 237-238.

[4] D.R. Cox and W.L. Smith, *Queues,* (Methuen, London, 1960).

[5] M.A.H. Dempster, J.K. Lenstra, and A.M.G. Rinnooy Kan, *Deterministic and Stochastic Scheduling,* (D. Reidel, Dordrecht, 1982).

[6] I. Duenyas and M.P. Van Oyen, Heuristic scheduling of parallel heterogeneous queues with set-ups, Technical Report 92-60, Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI (1993).

[7] J.C. Gittins, *Multi-armed Bandit Allocation Indices,* (Wiley, New York, 1989).

[8] D. Gupta, Y. Gerchak, and J.A. Buzacott, On optimal priority rules for queues with switchover costs, Working Paper, Department of Management Sciences, University of Waterloo (1987).

[9] R.W. Hall, *Zero Inventories,* (Irwin, Homewood, IL 1983).

[10] J.M. Harrison, A priority queue with discounted linear costs, *Operations Research,* **23** (1975) 260–269.

[11] J.M. Harrison, Dynamic scheduling of a multiclass queue: Discount optimality, *Operations Research,* **23** (1975) 270–282.

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

[13] G.P. Klimov, Time sharing service systems I, *Theory of Probability and Its Applications,* **19** (1974) 532-551.

[14] G.P. Klimov, Time sharing service systems II, *Theory of Probability and Its Applications* **23** (1978) 314-321.

[15] G. Koole, "Assigning a Single Server to Inhomogeneous Queues with Switching Costs," Technical Report, CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands (1994).

[16] T.L. Lai and Z. Ying, Open bandit processes and optimal scheduling of queueing networks, *Advances in Applied Probability,* **20** (1988) 447-472.

[17] H. Levy and M. Sidi, Polling systems: applications, modelling, and optimization, *IEEE Trans. Commun.* **38** (1990) 1750–1760.

[18] H. Levy, M. Sidi, and O.J. Boxma, Dominance relations in polling systems, Queueing Systems, **6** (1990) 155-172.

[19] Z. Liu, P. Nain, and D. Towsley, On optimal polling policies, *Queueing Systems (QUESTA)* **11** (1992) 59–84.

[20] I. Meilijson and U. Yechiali, On optimal right-of-way policies at a single-server station when insertion of idle times is permitted, *Stochast. Processes Appl.*, **6** (1977) 25–32.

[21] P. Nain, Interchange arguments for classical scheduling problems in queues, *Systems Control Letters,* **12** (1989) 177-184.

[22] P. Nain, P. Tsoucas, and J. Walrand, Interchange arguments in stochastic scheduling, *Journal of Applied Probability* **27** (1989) 815-826.

[23] R. Rajan, and R. Agrawal, Optimal server allocation in homogeneous queueing systems with switching costs, preprint, Electrical and Computer Engineering, Univ. of Wisconsin-Madison, Madison, WI 53706 (1991).

[24] M.I. Reiman, and L.M. Wein, "Dynamic Scheduling of a Two-Class Queue with Setups," Technical Report, Sloan School of Management, M.I.T., Cambridge, MA 02139 (1994).

[25] H. Takagi, Priority queues with set-up times, *Operations Research,* **38** (1990) 667-677.

[26] M.P. Van Oyen, *Optimal Stochastic Scheduling of Queueing Networks: Switching Costs and Partial Information*, Ph.D. Thesis, University of Michigan, Ann Arbor, MI (1992).

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

[28] M.P. Van Oyen, and D. Teneketzis, Optimal Stochastic Scheduling of Forest Networks with Switching Penalties, *Adv. Appl. Prob.* **26** (1994) 474–497.

[29] P. Varaiya, J. Walrand, and C. Buyukkoc, Extensions of the multi-armed bandit problem, *IEEE Transactions on Automatic Control,* **AC-30** (1985) 426-439.

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