CONTROL OF A QUEUEING SYSTEM WITH SEQUENCE-DEPENDENT SET-UPS

Mark P. Van Oyen
GE Corporate Research and Development
P.O. Box 8
Schenectady, NY 12301

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

Technical Report 92-51

September 1992

Control of a Queueing System with Sequence-Dependent Set-Ups

Mark P. Van Oyen
GE Corporate Research and Development
P.O. Box 8
Schenectady, NY 12301

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

September 25, 1992

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 general set-up time distribution. The objective is to minimize the expected holding cost due to the waiting of jobs. A set-up time is required to switch from one queue to another. We provide a limited characterization of the optimal policy and provide a simple heuristic scheduling policy. Simulation results are provided to demonstrate the effectiveness of our heuristic over a wide range of problem instances.

1. Introduction

In many manufacturing environments, a facility processes several different kinds of jobs. In many cases, a set-up time is required before the facility can switch from producing one type of job to another. If several different types of jobs are waiting when a unit has been completed, the decision maker is faced with the problem of deciding whether to produce one more unit of the same type of job that the machine is currently set up to produce, or to set up the machine to process a different kind of job.

The control decisions of when to set up the system and which type of job to produce have important effects on the performance of the system. First of all, for each unit of Work-In-Process Inventory that is waiting at a machine to be processed, the firm incurs significant holding costs. Second, companies quote due dates to customers based on the work load they have in the system. To quote feasible due dates, the facility manager must understand the job scheduling policy employed. To quote competitive due dates, the manager must have an efficient policy for scheduling service. It is essential to provide an efficient rule for the sequencing of jobs in the facility to quote customers feasible and competitive due dates.

The optimal control of the Work- In-Process in manufacturing systems without set-up times as well as the problem of quoting customer due dates have been extensively addressed in the literature. It is well known, for example, that for an $M/G/1$ queue with multiple job types, if jobs of type i

are charged holding costs at rate c_i and are processed at rate μ_i, the $c\mu$ rule (Average Weighted Processing Time rule) minimizes holding costs per unit time (see Baras et al. [2], Buyukkoc et al. [5], Gittins [8], Nain [17], Nain et al. [18], and Walrand [25]). Other stochastic scheduling problems in the literature where there are no costs (or no time lost) for switching from one type of job to another include Baras et al. [2], Dempster et al. [6], Gittins [8], Klimov [12], [13], Lai and Ying [14], Nain [17], Nain et al. [18], Varaiya et al. [24], and Walrand [25]. For systems with no switching times or costs, researchers such as Wein [26] and Wein and Chevalier [27] have developed due date setting rules.

There are few known results for the optimal scheduling of systems with switching costs or switching times. The reason for this is the difficulty of the problem. Most intuitive results developed for systems without switching penalties no longer hold in this case. Gupta et al. [9] considered the problem with switching costs and only two types of jobs with the same processing time distributions. Hofri and Ross [11] 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. Recently, Rajan and Agrawal [19] and Liu et al. [16] have studied systems similar to the one considered here and have partially characterized an optimal policy for the case of homogeneous service processes. Other work has concentrated on performance evaluation of different policies (see Baker and Rubin [1], Browne and Yechiali [4], Levy and Sidi [15], Takagi [22] and Srinivasan [21]). To our knowledge, there is a lack of literature comparing the performance of different heuristic policies for this problem.

In this paper, we address the stochastic scheduling of a system with several different types of jobs and switching times (equivalently set-up times) in a multiclass $M/G/1$ queue. Our purpose is to develop a heuristic that is simple enough to be implemented in a manufacturing environment and yet is also highly effective. We contribute a perspective on this problem based on reward rates. The applicability of reward rate notions is demonstrated by their use in partially characterizing an optimal scheduling policy. We then use this perspective to develop a heuristic for the stochastic scheduling problem with set-up times. The heuristic that we develop is extremely simple, since it is based only on statistical averages and uses a couple of simple inequalities. Moreover, our simulation results indicate that our heuristic policy consistently outperforms other policies suggested in the literature.

The rest of the paper is organized as follows. In Section 2, we formulate the problem. In Section 3, we partially characterize an optimal policy. In Section 4, we develop a heuristic policy, and indicate special cases under which the heuristic is optimal. In Section 5, we extensively test this heuristic by comparing this heuristic to other heuristics in the literature. Our results indicate that for a large variety of problems, our heuristic consistently outperforms other heuristics in the literature. The paper concludes in Section 6.

2. Problem Formulation

A single server is to be allocated to jobs in a system of parallel queues labelled $1, 2, \ldots, N$ 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 distribution with mean μ_n^{-1} such that $0 < \mu_n^{-1} < \infty$ and the second moment is also finite. 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). As a necessary condition for 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 ($c_n \geq 0$) cost units per job per unit time spent in queue n (including time in service). 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. The switching time, D_n, represents a period of time which is required to prepare the server for processing jobs in a queue different than the current one. We assume that successive set-ups for node n require strictly positive periods which are i.i.d., possess a finite mean and second moment, and are independent of all else.

A policy specifies, at each decision epoch, that the server either 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. The average cost per unit time of policy g, $\bar{J}(g)$, can now be expressed as

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

Assuming finite steady state average queue lengths exist under an optimal policy, the criterion to be minimized is that of the weighted sum of the average queue lengths. The class of admissible strategies, G, is taken to be the set of non-preemptive and non-anticipative policies that are based on perfect observations of the queue length processes. By non-preemptive, we mean that neither the service of a job nor the execution of a set-up can be interrupted until its completion. Idling is allowed at any decision time, but idle periods may not preempt job service or set-up. The objective of the optimization problem is to determine a policy $g^* \in G$ that minimizes $\bar{J}(g)$. The set of decision epochs is assumed to be the set of all arrival epochs, service epochs, instances of idling, and instances of set-up time completion.

In developing our heuristic, we focus on the subclass of G consisting of pure Markov (that is, stationary and non-randomized) 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) inter-arrival period. Thus, the state of the system is described by the vector $X(t) = (X_1(t), X_2(t), \ldots, X_N(t), n(t), \delta(t)) \in \mathcal{S}$, where $n(t)$ denotes that the server is located at node $n(t)$ at time t, $\delta(t)$ is zero if the set-up of node $n(t)$ is not complete at time t and is one otherwise, and $\mathcal{S}$ denotes the state space $(\mathbb{Z}^+)^N \times \{1, 2, \ldots, N\} \times \{0, 1\}$. Let the action space be $\mathcal{U} = \{0, 1, 2, \ldots, N\} \times \{0, 1, 2\}$. The action $U(t) = (n, 0) \in \mathcal{U}$ causes the server to idle in node $n = n(t^-)$ provided $\delta(t) = 1$ (and is not feasible otherwise). Action $U(t) = (n, 1)$ causes the server to serve one job of node $n = n(t^-)$ provided $\delta(t) = 1$ (and is not feasible otherwise). Action $U(t) = (n, 2)$ causes the server to set up node n provided $\delta(t) = 0$ (and is not feasible otherwise). Thus, a pure Markov policy g is equivalent to a mapping from $\mathcal{S}$ to $\mathcal{U}$.

3. On an Optimal Policy

3.1. Context

We begin by considering the problem without switching times ($D_i \equiv 0 \ \ \forall i$). In this case, the problem has been well studied. Proposition 8.2.6 of Walrand [25] states that the non-preemptive $c\mu$ rule is optimal: The index $c_i\mu_i$ is attached to each job in the ith queue. It is optimal to serve at any decision epoch the available job possessing the largest index. (The decision epochs here are the set of all arrival epochs and service epochs.) Although these indices are most properly associated with individual jobs (see Varaiya et al. [24], because the jobs in a given queue are indistinguishable, the indices can be associated with the queues. If this is done, the index of any queue is independent of

both the queue length (provided it is strictly positive) and the arrival rate of that queue. Moreover, the fact that an "index rule" is optimal represents a great reduction in complexity over general pure Markov policies (which specify a mapping of the state space to the action space). In particular, an index rule is extremely simple in that the merit of choosing a given queue can be expressed via the index without regard to the states of other queues.

For the problem with switching times formulated above, the $c\mu$ rule provides a heuristic which one might naturally consider. Clearly, the $c\mu$ rule will perform well if the switching times are sufficiently small as to yield a negligible impact on performance. However, it is also clear that as switching times become longer, and hence a more significant component of performance, the $c\mu$ rule will tend to suffer from switching too frequently.

To prove structural properties of optimal policies, researchers such as Liu, Nain and Towsley [16] and Rajan and Agrawal [19] have restricted attention primarily to problems that are completely homogeneous with respect to cost and to the service processes. In fact, for systems which are completely symmetrical (including the arrival and switching time processes), they prove the optimality of certain index policies that can be described as exhaustive "serve the largest queue" policies.

3.2. Analysis

In this section, we provide a partial characterization of an optimal policy. We begin with several 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.

Although our focus is on the average cost per unit time criterion, we have found it convenient to study this criterion as a limiting case of the discounted cost criterion. We demonstrate the existence of a top-priority queue for an optimal policy under discounting, then extend the result to the average cost per unit time case.

We define the total expected discounted cost criterion: For discount parameter $\alpha > 0$, let

$$J_\alpha(g) = E\{\int_0^\infty \left(\sum_{n=1}^N c_n X_n^g(t)\right) e^{-\alpha t} dt\} \ . \tag{3.1}$$

The total discounted cost of (3.1) is finite. To see this, define $A_n(t)$ to be the cumulative number of arrival through time t. Note that in the worst case, $X_n^g(t) = X_n(0^-) + A_n(t)$. Thus, $J_\alpha(g) \leq \sum_{n=1}^N c_n(X_n(0^-)\alpha^{-1} + \lambda_n \alpha^{-2})$. .

As in Harrison [10] we transform the cost criterion of (3.1) to a reward criterion using the device of Bell [3]. Although Harrison's model did not include switching times , the results we quote carry over. The key observation is that one must focus on the (controlled) departure process from the system, since the arrival process is uncontrolled. Letting $Y_n^g(t)$ denote the right-continuous cumulative departure process from node n under g through time t, we have $X_n^g(t) = X_n(0^-) + A_n(t) - Y_n^g(t)$. Because

$$J_\alpha(g) = \sum_{n=1}^N c_n \left(X_n(0^-)\alpha^{-1} + \lambda_n \alpha^{-2} - E\{\int_0^\infty Y_n^g(t) e^{-\alpha t} dt\}\right) \ , \tag{3.2}$$

the minimization of $J_\alpha(g)$ is equivalent to the maximization of the following reward criterion:

$$R_\alpha(g) = \sum_{n=1}^{N} c_n E\{\int_0^\infty Y_n^g(t) e^{-\alpha t} dt\} \ . \tag{3.3}$$

In (3.3), c_n is interpreted as a holding cost charge per unit time. As Harrison shows, we may choose instead to focus on a lump sum reward of amount $c_n\alpha^{-1}$ earned upon service completion. Thus, (3.3) can be expressed as

$$\begin{aligned} R_\alpha(g) &= E\{\sum_{n=1}^{N} c_n\alpha^{-1} \int_0^\infty e^{-\alpha t} dY_n^g(t)\} & (3.4)\\ &= E\{\sum_{n=1}^{N}\sum_{k=1}^{\infty} e^{-\alpha T_n^g(k)} c_n\alpha^{-1}\} \ , & (3.5) \end{aligned}$$

where $T_n^g(k)$ is the kth service completion epoch under g corresponding to a service in node n . In contrast to (3.3), along any realization only one queue can contribute to the reward stream of (3.4) at any given t (and does so only if a service completion occurs at t). The term $c_n\alpha^{-1}$ is interpreted as the reward received upon job completion, and it equals the discounted cost of holding that job forever.

It is useful to consider the policy, call it g', that at time $t = 0$ sets-up node n, serves a deterministic number u jobs, and then idles thereafter. We denote the expected discounted reward earned from this action sequence by $C_n(u)$. Using (3.4), $C_n(u) = c_n\alpha^{-1}E\{\int_0^\infty e^{-\alpha t} dY_n^{g'}(t)\}$. Let $f_{n,k}$ denote the sum of k service durations. Letting $S_n \triangleq E\{e^{-\alpha f_{n,1}}\}$, we use the i.i.d. nature of successive services to get $E\{e^{-\alpha f_{n,k}}\} = S_n^k$. Thus

$$\begin{aligned} C_n(u) &= c_n\alpha^{-1}E\{\sum_{k=1}^{u} e^{-\alpha(D_n + f_{n,k})}\} & (3.6)\\ &= c_n\alpha^{-1}S_n(1-S_n)^{-1}(1-S_n^u)E\{e^{-\alpha D_n}\} \ . & (3.7) \end{aligned}$$

We define the reward rate associated with this sequence of actions to be the ratio of expected discounted reward, $C_n(u)$, to the expected discounted length of time required; that is,

$$\frac{C_n(u)}{E\{\int_0^{D_n+f_{n,u}} e^{-\alpha t} dt\}} = \frac{c_n\alpha^{-1}S_n(1-S_n)^{-1}(1-S_n^u)E\{e^{-\alpha D_n}\}}{\alpha^{-1}(1-S_n^u E\{e^{-\alpha D_n}\})} < h_n, \tag{3.8}$$

where h_n is defined by

$$h_n \triangleq c_n S_n/(1-S_n) \ . \tag{3.9}$$

Given a discount parameter $\alpha > 0$, the reward rate earned by serving a single job in node i (without a set-up) is h_n. To see this key fact, simply set $D_n = 0$ in (3.8). Theorem 1, which follows, states that a top-priority queue always exists under an optimal policy and can be determined as the node maximizing h_n over all n. Theorem 1 is similar to the results presented in Gupta et al [9], Hofri and Ross [11], Liu et al. [16] and Rajan and Agrawal [19]. The novelty of our result and proof lies primarily in our treatment of unequal or heterogenous service distributions at each queue. We first state a purely technical lemma which we will use in the proof of the theorem. The proof of Lemma 1 is straightforward and we omit it.

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

$$(\sum_{u \in \mathcal{U}} p_u Er_u)/(\sum_{u \in \mathcal{U}} p_u E\sigma_u) \leq \max_{u \in \mathcal{U}} Er_u / E\sigma_u. \tag{3.10}$$

Theorem 1: Consider the discounted cost criterion of (3.1) for some discount parameter $\alpha > 0$. If $h_i \geq h_j$ for all $j = 1, 2, \ldots, N$, then only a policy for which queue i is a top-priority queue can be an optimal policy.

Proof: Without loss of generality, suppose node one maximizes h_n over n. Suppose policy g is optimal but does not serve node one as a top priority node. We first prove that an optimal policy must serve node one exhaustively, then we indicate how a similar argument justifies 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 serve node one exhaustively. Thus, for some state $(x_1, \ldots, x_n, 1, 1) \in \mathcal{S}$ with $x_1 \geq 1$, policy g chooses to switch to node j. We assume, without loss of generality, that $X(0^-) = (x_1, \ldots, x_n, 1, 1)$ and that at time $t = 0$, $U^g(0) = (j, 2)$ for some $j \neq 1$ (that is, g chooses to switch to node j at $t = 0$). We proceed to show that the reward rate earned by serving jobs in queues other than one cannot exceed the reward rate earned by serving a job of node one at $t = 0$. Thus, a policy that serves node one at $t = 0$ does strictly better than g.

For $l \in \mathbb{N}$, let $t^\pi(l)$ denote the time at which the l^{th} control action is taken under policy π. We write $t^g(l)$ more briefly as $t(l)$. Thus, $t(1) = 0$, and $t(2) = D_j$. With respect to policy g, let $L \in \{\mathbb{N} \cup \infty\}$ denote the stage, or index of the decision epoch, at which g first chooses to serve a job of node one. Thus, $U^g(t(L-1)) = (1, 2)$, and $U^g(t(L)) = (1, 1)$. If g never serves a job in node one with probability p', then L takes on the value ∞ with probability p'. We find it convenient to use the process $\{t(l) : l = 1, 2, \ldots, L-1\}$ in defining the reward earned by g prior to its first service in node one. Using (3.4), we write this quantity as $R_\alpha(g^{L-1})$. Thus,

$$R_\alpha(g^{L-1}) = E\{\sum_{l=1}^{L-1} e^{-\alpha t(l)} r(l)\} \tag{3.11}$$

where $r(l)$ is the discounted reward (discounted to time $t(l)$) earned by policy g during the l^{th} stage. Note that the actions of idling and set-up result in a zero reward for that stage; thus $r(1) = 0$. Because, $T_n^g(k)$ of (3.7) denotes the time of service completion, we write (3.11) as

$$\begin{aligned} R_\alpha(g^{L-1}) &= E\{\sum_{l=2}^{L-1} e^{-\alpha t(l)} r(l)\} && (3.12) \\ &= \sum_{l=2}^{\infty} E\{e^{-\alpha t(l)} r(l) \mathbb{1}[L > l]\} && (3.13) \\ &= \sum_{l=2}^{\infty} E\{e^{-\alpha t(l)} E\{r(l) \mid L > l, t(l)\}\} && (3.14) \end{aligned}$$

Equation (3.14) involves a conditional expectation of the expected discounted reward earned under g during stage l. Service in node n yields an expected discounted reward rate of h_n where

$h_n \leq h_1$. Using Lemma 1, we see that

$$E\{r(l)|L > l, t(l)\} = \sum_{n=2}^{N} S_n c_n \alpha^{-1} Pr(U^g(t(l)) = (n,1)|L > l, t(l)) \quad (3.15)$$

$$\leq h_1 \sum_{n=2}^{N} E\{\int_0^{f_{n,1}} e^{-\alpha t} dt\} Pr(U^g(t(l)) = (n,1)|L > l, t(l))\} \quad (3.16)$$

$$\leq h_1 E\{\int_0^{t(l+1)-t(l)} e^{-\alpha t} dt | L > l, t(l)\} \ . \quad (3.17)$$

Equation (3.17) follows from the fact that the set-up and idling actions, which contribute zero reward, may also be exercised at time $t(l)$ and are not included in (3.16). Using the upper bound of (3.17), we proceed to place an upper bound on $R_\alpha(g^{L-1})$. From (3.14) we get

$$R_\alpha(g^{L-1}) \leq \sum_{l=2}^{\infty} E\{e^{-\alpha t(l)} h_1 E\{\int_0^{t(l+1)-t(l)} e^{-\alpha t} dt | L > l, t(l)\}\} \quad (3.18)$$

$$= h_1 \sum_{l=2}^{\infty} E\{e^{-\alpha t(l)} \int_0^{t(l+1)-t(l)} e^{-\alpha t} dt \mathbb{1}[L > l]\} \quad (3.19)$$

$$= h_1 E\{\int_{t(2)}^{t(L)} e^{-\alpha t} dt\} \ . \quad (3.20)$$

Thus, the reward rate, which we denote by ψ, earned under g prior to serving node one is strictly less than h_1. To see this, note that $t(2) = D_j$, and so

$$\psi = \frac{R_\alpha(g^{L-1})}{E\{\int_0^{t(L)} e^{-\alpha t} dt\}} \quad (3.21)$$

$$\leq h_1 \left(1 - \frac{E\{\int_0^{D_j} e^{-\alpha t} dt\}}{E\{\int_0^{t(L)} e^{-\alpha t} dt\}}\right) \quad (3.22)$$

$$\leq h_1 \left(1 - \frac{\alpha^{-1}(1 - E\{e^{-\alpha D_j}\})}{\alpha^{-1}}\right) \quad (3.23)$$

$$= h_1 E\{e^{-\alpha D_j}\} \quad (3.24)$$

$$< h_1 \ . \quad (3.25)$$

In (3.23), we used the fact that $t(L) \leq \infty$, and obtained the bound by setting $t(L) = \infty$. A reward rate of h_1 is earned during the service of a job in node one, while g earns a reward rate $\psi < h_1$ prior to stage L. Thus, we show that g is strictly suboptimal as follows. Along each sample path of the arrival, service, and set-up processes, we construct a policy $\tilde{g}$, which is a modification of g. At time $t = 0$, $\tilde{g}$ serves the job in node one that is served under policy g at $t(L)$. Let $f_{1,1}$ denote the duration of the processing time for this job. During $[f_{1,1}, f_{1,1} + t(L)]$, $\tilde{g}$ mimics the actions taken by g during $[0, t(L)]$, the first L stages. At time $t(L+1) = t(L) + f_{1,1}$, both g, and $\tilde{g}$ reach the same state along any realization. Because of this coupling, $\tilde{g}$ is able to mimic g from that point on. We note that the construction of $\tilde{g}$ is feasible because , following the service of a job in node one, $\tilde{g}$ takes all the actions taken by g during the first $L-1$ stages of g. With respect to any stage l under g, the corresponding stage under $\tilde{g}$ occurs at a later point in time with at least as much information available to $\tilde{g}$.

We proceed to show that g incurs a strictly greater expected discounted cost than $\tilde{g}$. Because of the coupling at time $t(L+1)$, policies g and $\tilde{g}$ incur identical costs from that time onward. Also, note that the single-stage reward earned by serving a single job of node one is given by

$$E\{e^{-\alpha f_{1,1}}\}c_1\alpha^{-1} = h_1\alpha^{-1}(1-S_1) = h_1 E\{\int_0^{f_{1,1}} e^{-\alpha t}dt\} \ . \tag{3.26}$$

Thus, the difference in expected discounted reward of policy $\tilde{g}$ with respect to g results from the first L stages and can be computed from (3.4), (3.11), (3.21), (3.26) as

$$\begin{aligned} R_\alpha(\tilde{g}) - R_\alpha(g) &= E\{e^{-\alpha f_{1,1}}\}[c_1\alpha^{-1} + R_\alpha(g^{L-1})] \\ &\quad -[R_\alpha(g^{L-1}) + E\{e^{-\alpha t(L)}\}c_1\alpha^{-1}E\{e^{-\alpha f_{1,1}}\}] \qquad (3.27) \\ &= [h_1\alpha^{-1}(1-S_1) + S_1\psi E\{\int_0^{t(L)} e^{-\alpha t}dt\}] \\ &\quad -[\psi E\{\int_0^{t(L)} e^{-\alpha t}dt\} + E\{e^{-\alpha t(L)}\}h_1\alpha^{-1}(1-S_1)] \qquad (3.28) \\ &= \alpha^{-1}(1-S_1)(1-E\{e^{-\alpha t(L)}\})[h_1-\psi] \qquad (3.29) \\ &> 0 \ , \qquad (3.30) \end{aligned}$$

where the inequality follows from (3.25). Thus $\tilde{g}$ incurs a strictly lower expected discounted cost than g. Repeated application of the preceeding argument at every point of non-exhaustive service at node one establishes the optimality of exhaustive service at node one.

The argument made for the optimality of exhaustive service can also be used to establish the optimality of greedy service at node one. We proceed to make this connection precise. 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 returns at time $t(L)$ to serve a job in node one. Because a zero reward rate is earned during the first stage under g, and subsequent single-stage reward rates cannot exceed h_1, the modified policy $\tilde{g}$ as previously constructed performs strictly better than g. □

Corollary 1: Consider the discounted cost criterion of (3.1). If, for $i = 1,2,\ldots,N$, the service period distribution of queue i is exponential with rate μ_i and $c_i\mu_i = c\mu$, then every queue is a top-priority queue under an optimal policy.

Proof: For exponential service times, we have $S_i = \mu_i/(\alpha+\mu_i)$, and thus $h_i = c_i\mu_i\alpha^{-1}$. Theorem 1 gives the result. □

We note that, for problems satisfying the condition of Corollary 1, the above result indicates that exhaustive and greedy service of every queue is optimal. The specification of an optimal policy requires the resolution of two additional elements: (1) the proper manner of idling in an empty queue when the system is not empty and (2) the proper selection of the next queue upon deciding not to idle.

The next corollary extends Theorem 1 to the average cost per unit time case.

Corollary 2: For the average cost per unit time criterion, if $c_i\mu_i > c_j\mu_j$ for all $j = 1,2,\ldots,N$, then there exists an optimal policy that serves node i as a top-priority node.

Proof: The result follows from Theorem 1 by a standard Abelian argument (cf. Feigin [7]). □

The purpose of this section on analysis has been two-fold. First, we have demonstrated some structural properties of optimal policies. The heuristic that we develop in the next section is consistent with these structural properties. Our second purpose was to present notions of *reward rate* as they apply to the problem at hand. The characterization of single-stage reward rates is crucial to the analysis of Theorem 1. The characterization of reward rates for sequences of actions is crucial to the heuristic that we develop. For the average cost problem, the quantity $c_n\mu_n$ can be regarded as the (reward) rate at which holding costs are reduced by serving a job in node n. On the other hand, a reward rate of zero is earned during idle periods and set-up periods. With M_i denoting a (possibly random) sequence of service, set-up, and idle periods in node i, let $R(M_i)$ denote the total time required to complete M_i. We define the reward rate of the action sequence M_i to be $\bar{v}(M_i)$, where

$$\bar{v}(M_i) = ER(M_i)/ET(M_i) \tag{3.31}$$

and $R(M_i)$ equals $c_i\mu_i$ times the total time spent serving jobs during play M_i (switching times and idle periods are excluded). It follows from this that $0 \leq \bar{v}(M_i) \leq c_i\mu_i$. Note that these reward rates can be derived as the limit as $\alpha \searrow 0$ of α times the discounted reward rate. We use these concepts of reward rates in the next section to derive a heuristic for the problem.

4. A Heuristic

We develop a greedy heuristic for the problem formulated in Section 2. We first define (without loss of generality) queue 1 to be the top-class queue (i.e., $c_1\mu_1 \geq c_i\mu_i$ for all i). We let x_i be the queue length at queue i. We first develop a heuristic for the problem with two queues and then extend it to the case with any number of queues.

4.1. Heuristic for Systems with Two Queues

By Theorem 1, we know that the server should not switch from queue 1 to queue 2 when queue 1 is not empty. Defining a heuristic policy for two queues requires deciding when to switch from queue 2 to queue 1 when queue 2 is not empty, as well as the characterization of a policy for idling, which we refer to as an idling policy (i.e., should the server idle at queue i or switch to the other queue?). Our heuristic defines indices for each of the queues and defines a policy using these indices.

To begin, we assume that nodes 1 and 2 are both nonempty and focus on the question of when to switch from node 2 to 1. Later we will prescribe a policy for idling. In computing indices for each queue, we assume that once the server switches to a queue, the server will remain at that queue until the end of its busy period. Clearly, by Theorem 1, if the server switches from queue 2 to queue 1, it will remain at queue 1 until the end of its busy period. However, the server may switch from queue 2 to queue 1 before queue 2 is exhausted. We let $\varphi_i(x_i, r)$ denote the expected reward associated with remaining in queue i. If the server remains at queue i and $x_i > 0$, it will continue earning rewards at a rate of $c_i\mu_i$ until the end of queue i's busy period. Hence, we define the index to remain in queue 2 if there is a job at queue 2 to be

$$\varphi_2(x_2, r) = c_2\mu_2 \text{ if } x_2 \geq 1. \tag{4.32}$$

On the other hand, if the server decides to switch to queue 1 when it could have served queue 2, it will first have to set-up queue 1 and earn no rewards for the (random) duration of time D_1. Then, by Theorem 1, it will serve queue 1 until the end of its busy period. It could actually remain at node 1 for a longer amount of time by idling at node 1 for a certain duration of time. However, in calculating an index for switching to node 1, we disregard idling. We assume that at the end of the busy period of node 1, the server switches back to node 2, and for a (random) duration of

time D_2 again earns no rewards. Hence, by switching to node 1 to serve the jobs at node 1 and returning to 2 at the end of node 1's busy period, the server will have spent an expected total amount of time $ED_1 + ED_2 + \frac{x_1+\lambda_1 ED_1}{\mu_1-\lambda_1}$. On average, however, the server will have earned a reward only for the expected duration of time equal to $\frac{x_1+\lambda_1 ED_1}{\mu_1-\lambda_1}$. Hence, the index for switching to node 1 is given by the expected rate at which rewards are earned for the time consisting of setting up node 1, exhausting the queue at node 1, and setting up node 2 again:

$$\varphi_1(x_1, s) = c_1\mu_1 \frac{\frac{x_1+\lambda_1 ED_1}{\mu_1-\lambda_1}}{ED_1 + \frac{x_1+\lambda_1 ED_1}{\mu_1-\lambda_1} + ED_2} . \tag{4.33}$$

Simplifying (4.33) yields

$$\varphi_1(x_1, s) = c_1\mu_1 \frac{x_1 + \lambda_1 ED_1}{x_1 + \mu_1 ED_1 + (\mu_1 - \lambda_1)ED_2} . \tag{4.34}$$

Equations (4.32) and (4.34) immediately suggest a heuristic policy for switching from queue 2 to queue 1. If the expected reward per unit time earned by switching to queue 1 $(\varphi_1(x_1, s))$ is greater than the expected reward per unit time earned by remaining at queue 2 $(\varphi_2(x_2, r))$, then the server would switch to queue 1. We note, however, that this heuristic switching policy does not take into account the utilization of the server ($\rho = \lambda_1/\mu_1 + \lambda_2/\mu_2$). It is intuitive that whether the server chooses to switch to queue 1 with $x_2 \geq 1$ will depend on the utilization of the server. For example, c_1 can be made artificially high so that regardless of the values of ED_1 and ED_2, $\varphi_1(x_1, s) > \varphi_2(x_2, r)$. Clearly, how often the server switches from 2 to 1 with $x_2 \geq 1$ should depend on ρ. Another way to look at this problem is that one way to ensure that the system will be stable is by requiring the server to be busy at least ρ fraction of the time when it switches from 2 to 1. We impose this as a constraint to be satisfied before the server can switch from 2 to 1. Hence, we use the following heuristic condition for switching from 2 to 1 when $x_2 \geq 1$:

$$\varphi_1(x_1, s) \geq \rho c_1\mu_1 + (1 - \rho)c_2\mu_2 = c_2\mu_2 + \rho(c_1\mu_1 - c_2\mu_2) . \tag{4.35}$$

If (4.35) is satisfied, then $\varphi_1(x_1, s) > \rho c_1\mu_1$ and by (4.33) the service of queue 1 and the subsequent set-up of queue 2 will on average result in a utilization of the server exceeding ρ for this action sequence. Note that the condition in (4.35) has some desirable characteristics. For example, as ρ approaches 1, x_1 must increase to signal a switch from 2 to 1, and hence the policy tends to serve the queues exhaustively. Similarly, as ED_1 or ED_2 get large, x_1 must be increasingly large to merit a switch from 2 to 1 when $x_2 > 0$.

Theorem 1, along with equation (4.35) completely characterize our heuristic policy for switching from one node to another when the node that is currently being served is not empty. However, to complete the characterization of our heuristic policy, we also need to specify a policy for idling when there are no jobs in the node the server is currently set up to serve. Equation (4.35) does not apply in this case, since the server receives no rewards by idling at the current node. In order to decide whether to switch to the other node or to idle, we compare the reward rate at which the server will earn rewards by immediately switching to the other node with that of idling until one more arrival occurs at the other node. As in the derivation of (4.34), a switch from node 1 to 2 that proceeds to exhaust node 2 and returns to set up node 1 will earn a reward rate of $\varphi_2(x_2, s)$ for the next $ED_1 + ED_2 + \frac{x_2+\lambda_2 ED_2}{\mu_2-\lambda_2}$ units of time (on average), where

$$\varphi_2(x_2, s) = c_2\mu_2 \frac{x_2 + \lambda_2 ED_2}{x_2 + \mu_2 ED_2 + (\mu_2 - \lambda_2)ED_1} . \tag{4.36}$$

Now, consider the policy that idles at node 1 until the next arrival at node 2 and then switches to node 2. Of course, before the next arrival at node 2, arrivals could occur at node 1, and the server would earn some reward by serving them. However, we assume that no rewards are earned while idling, and compute the reward rate of the inadmissable policy that idles until the next arrival at node 2, then switches to node 2 to exhaust it, and returns to node 1. This results in the following reward rate:

$$\varphi_2'(x_2, s) = c_2\mu_2 \frac{\frac{x_2+1+\lambda_2 ED_2}{\mu_2-\lambda_2}}{\frac{x_2+1+\lambda_2 ED_2}{\mu_2-\lambda_2} + \frac{1}{\lambda_2} + ED_1 + ED_2} . \tag{4.37}$$

The condition for switching from 1 to 2 when there are no jobs at node 1 is then given by

$$\varphi_2'(x_2, s) < \varphi_2(x_2, s) , \tag{4.38}$$

which implies that the server earns rewards at a higher rate by switching now than by waiting for one more arrival at node 2. Simplifying (4.38) leads to a very simple formula for the number required at node 2 so that the server will switch to node 2 from node 1 without idling:

$$x_2 > \lambda_2 ED_1 . \tag{4.39}$$

Similarly, when the server is at node 2, and there are no more jobs to serve, it immediately switches to node 1 if $x_1 > \lambda_1 ED_2$. We also note that our simulation experience indicates that requiring the server to serve at least one job upon switching to a queue before it can switch to another queue improves the performance of the heuristic. Hence, we also place this constraint on the server. We are now in a position to describe our heuristic control rule in full:

Heuristic Policy for Systems with Two 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 1 and $x_1 = 0$, then switch to node 2 if $x_2 > \lambda_2 ED_1$. Else, idle until the next arrival to the system.
3. If the server is currently at node 2 and $x_2 > 0$, and $\varphi_1(x_1, s) < c_1\mu_1\rho + (1-\rho)c_2\mu_2$ then serve one more job at node 2.
4. If the server is currently at node 2, and $x_2 > 0$ and $\varphi_1(x_1, s) \geq c_1\mu_1\rho + (1-\rho)c_2\mu_2$, then
   a) If no jobs have been processed since the last set-up, process one more job at node 2.
   b) If at least one job has been processed since the last set-up, switch to node 1.
5. If the server is currently at node 2 and $x_2 = 0$, then switch to node 1 if $x_1 > \lambda_1 ED_2$. Else, idle until the next arrival to the system.

We note that regardless of the initial number of jobs in either queue 1 or queue 2, the condition in (4.35) guarantees that under our heuristic for two queues, the number of jobs in queue 1 will eventually be 0 and the number of jobs in queue 2 will eventually be less than $\lambda_2 ED_1$, (i.e., that the queues will be stable). To see this, note that whenever the number in queue 1 exceeds x_1 where x_1 is large enough that $\varphi_1(x_1, s)$ satisfies the condition in (4.35), the server switches to queue 1. Thereafter, the server will serve queue 1 until no jobs remain at queue 1, and since we assume that $\mu_1 > \lambda_1$, the number of jobs in queue 1 will (with probability one) eventually reach 0. Now, assume that the number in queue 2 exceeds $\lambda_2 ED_1$. Without loss of generality, assume that the server is

initially set-up to serve queue 2. Now, in this case, the server will serve queue 2 until the number of jobs in queue 1 is large enough that the condition in (4.35) is satisfied, and the server switches to queue 1. The server will then serve queue 1 exhaustively. Then, if the number of jobs in queue 2 is still higher than $\lambda_2 ED_1$, the server will immediately switch to queue 2 without idling. Now, note that in the interval where the server serves queue 2, the percentage of time that the server does useful work (we define useful work as processing jobs, while set-ups and idling do not constitute useful work) is 100%. Similarrly, due to the condition in (4.35), in the time interval consisting of setting up queue 1, processing jobs in queue 1, and setting-up queue 2, the percentage of time that the server does useful work is greater than 100ρ percent (compare (4.34) and (4.35)). Once the server is in queue 2, the cycle repeats. Hence, we can conclude that so long as the number of jobs in queue 2 is above $\lambda_2 ED_1$, the percentage of time in which the server is actually processing jobs is greater than 100ρ percent. This means that the number of jobs in queue 2 will eventually be lower than $\lambda_2 ED_1$.

4.2. Heuristic For Systems with N Queues

Using the ideas developed previously for 2 queues, we can now extend our heuristic to the case where the system has any number of queues. Again, we first renumber the queues so that $c_1\mu_1 \geq c_2\mu_2 \ldots \geq c_N\mu_N$. As before, defining a policy requires specifying an idling policy and characterizing server switches from one queue to another one when the current queue is not empty. To begin, assume that the server is currently serving queue i and that $x_i > 0$. Because a reward rate of $c_i\mu_i$ can be achieved by serving a job in node i and a reward rate of at most $c_j\mu_j$ can be achieved by serving jobs in node j, we need only consider switching from i to queue $j \in \{1, 2, \ldots, i-1\}$. Then to switch to any queue j, we require that

$$\varphi_j(x_j, s) \geq c_j\mu_j\rho + c_i\mu_i(1-\rho) \text{ and } j \in \{1, 2, \ldots, i-1\}\ , \tag{4.40}$$

where

$$\varphi_j(x_j, s) = c_j\mu_j \frac{x_j + \lambda_j ED_j}{x_j + \mu_j ED_j + (\mu_j - \lambda_j)ED_i}. \tag{4.41}$$

However, in this case, there may be more than one queue j that satisfies the constraint (4.40). In that case, the server switches to the one that has the highest reward rate. That is, the switching policy is to switch to the queue that has the highest index $\varphi_j(x_j, s)$ which satisfies the constraint in (4.40). As before, we require that once a set up has been completed, the server processes at least one job before switching to another queue.

Similar to the case of two queues, we define an idling policy to treat the case where the server is in queue i with $x_i = 0$. In the case of n queues, this is slightly more complicated than for two, since the server must decide not only whether to idle, but also which queue the server should switch to. One obvious policy would be to choose the queue that has the highest reward rate index, $\varphi_j(x_j, s)$. However, in the case where there are more than 2 queues, this may cause a queue with a low value of $c\mu$ not to be served. For example, consider a case where there are 3 kinds of jobs and $c_1\mu_1 >> c_3\mu_3$, and $c_2\mu_2 >> c_3\mu_3$, and the set-up times and arrival rates are the same for all of the queues. In this case, since the reward rates at queues 1 and 2 are much greater than the reward rate at queue 3, the only time queue 3 would be visited might be when queues 1 and 2 are both empty, causing queue 3 to be unstable. That is, if the reward rates of the different queues are far enough apart, a queue with a high reward rate may be preferred to a queue with a low reward rate even if the high reward rate queue has only one job and the low reward rate queue has a very high number of jobs. We place a constraint similar to (4.40) on switching from queue i when $x_i = 0$. To develop such a rule, we first note that the reward rate of (4.41), $\varphi_j(x_j, s)$, includes both ED_i and

also ED_j. This is because by switching from queue i to queue j, the server is leaving behind some unfinished jobs at queue i and must return to finish them at a certain point. If $x_i = 0$, however, there will be no jobs left behind and in this case, we define the reward rate earned by switching to queue i as

$$\phi_j(x_j, s) = c_j \mu_j \frac{x_j + \lambda_j ED_j}{x_j + \mu_j ED_j}. \tag{4.42}$$

We use the following idling procedure for choosing the queue to switch to when the server is in queue i and $x_i = 0$.

1. Let $\sigma = \emptyset$.
2. For all $j \neq i$, if $\phi_j(x_j, s) \geq c_j \mu_j \rho$, then let $\sigma = \sigma \cup j$.
3. If $\sigma \neq \emptyset$ then among all $j \in \sigma$, let k denote the queue such that $k = \arg\max_{j \in \sigma} \phi_j(x_j, s)$. If $x_k \geq \lambda_k ED_i$, then switch to queue k, else idle until the next arrival to the system.
4. If $\sigma = \emptyset$, then let k denote the queue such that $k = \arg\max_{j \neq i} \phi(x_j, s)$. If $x_k > \lambda_k ED_i$ then switch to queue k, else idle until the next arrival to the system.

The above procedure determines the set of queues such that if the server switched to such a queue, it would actually be processing jobs ρ fraction of the time until the end of that queue's busy period. Among all the queues satisfying this constraint, it selects as a candidate the one that has the largest reward rate until the end of its busy period. On the other hand, if it can find no queue that satisfies this constraint, then it chooses the one that has the highest reward rate. Then, the procedure uses the simple rule developed for the case of two queues to decide whether to idle or to switch to the candidate queue. We note that the constraint in the second step of our idling policy is very similar to the constraint for the case where $x_i > 0$. The basic idea is that in order to switch to another queue, it is not enough that that queue has a high reward rate, the server must also be spending a significant portion of its time actually processing jobs. This prevents frequent switches to queues that have high reward rates regardless of the queue length. In fact, we note that as either set-up times increase or the utilization increases, the server will tend to serve the queues in an exhaustive and cyclic manner.

Having explained the logic of our heuristic, we can now state it formally.

Heuristic Policy for N Queues

Assume that the server is set-up to serve queue i and queue i contains x_i jobs.

1. If $x_i = 0$, use the idling policy developed above.
2. If $x_i > 0$, then for all $j < i$, compute $\varphi_j(x_j, s)$ using (4.41). Let $\sigma = \emptyset$. For $j = 1, \ldots, i-1$, if queue j satisfies constraint (4.40), then $\sigma = \sigma \cup j$. Among all $j \in \sigma$, switch to the queue that has the highest index $\varphi_j(x_j, s)$. If σ is empty then serve one more job of type i.

Having developed our heuristic, we next describe the limiting cases where it is known to have optimal characteristics.

1. $D_i = 0$ *for all* i*:* In the case where all the set-up times are zero, our heuristic reduces to the $c\mu$ rule which is known to be optimal. That is, at each instant serve the job that maximizes $c_i \mu_i$.

2. *Symmetrical systems:* Suppose that all the queues are identical with respect to holding costs, service distribution, arrival rate, and set-up distribution. In this case, the heuristic would serve each node exhaustively, and upon switching would always choose the queue that has the largest number of jobs. These policies have been shown to be optimal among the set of non-idling policies (Liu et al. [16], Rajan and Agrawal [19]). The optimal idling policy is not known.

3. $\lambda_i = 0$ *for all* $i = 1, \ldots, N$*:* In the case of no arrivals, Van Oyen et al. [23] have proven that all the queues are served in an exhaustive manner. In that case, using the policy we described above, once a queue is exhausted the next queue to switch to is the one that has the highest index $c_j\mu_j \frac{x_j\mu_j^{-1}}{x_j\mu_j^{-1}+D_j}$. In fact Van Oyen et al. [23] proved this index policy to be optimal for the system with an initial number of jobs in each queue and no arrivals.

Having developed our heuristic, and specified the cases where it has optimal characteristics, we undertake a simulation study in the next section to test its performance.

5. A Simulation Study

The real test of any heuristic is its performance with respect to the optimal solution. In our problem, however, the characteristics of the optimal solution are not known, except for a few special cases. Hence, we chose to compare our heuristic to other widely used policies in the literature. To test our heuristic, we generated a large variety of problems with 2 and 3 queues. The cases that we tested included symmetric as well as asymmetric queues, high and moderate utilization, and both equal and different holding costs for different job classes. For each of the cases, we tested our heuristic by simulating 50000 job completions from the system. We repeated the simulation 10 times and averaged the holding cost per unit time that we obtained in each run.

We first tested our heuristic on a variety of problems with 2 queues. The data for the 14 different examples with 2 queues are displayed in Table 1. In all of the test problems that we report here, the service times and the set-up times are exponential. However, we have also tested our heuristic with other distributions including the uniform, normal and deterministic cases. In each case, we obtained very similar results to those reported here. Examples 1-8 have $c_1\mu_1 = c_2\mu_2$. For these cases, the best policy that we know of is of an exhaustive, threshold type such that the server remains at each queue until it is exhausted and idles until the number of jobs at the other queue is beyond a certain threshold. These 8 cases are thus testing the idling policy of our heuristic. They include cases with high as well as moderate utilization and mean set-up times. We also changed the data so that if in one problem, the queue that had the high arrival rate had the high set-up time as well, in the next one the queue that had the high arrival rate had a low set-up time, to see the effect of each combination of set-up time and arrival rate.

We compared our heuristic to five widely used and analyzed policies from the literature. The first of these (Exhaustive) serves each of the queues in an exhaustive and cyclic manner. That is, the server finishes all of the 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 etc. The second alternative, the gated heuristic, 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. As a third alternative, we tested the (exhaustive, strict-priority) $c\mu$ rule as a heuristic. We also tested heuristic policies requiring a search. We searched a class of exhausive, threshold policies by simulation to find the best policy of that class. Specifically, we denote by (EX-TR) the class of exhaustive, threshold policies which serve both nodes 1 and 2 exhaustively and idle in queue $j \neq i$ unless queue i exceeds a threshold y_i, upon which event

Example	c_1	c_2	μ_1	μ_2	λ_1	λ_2	ED_1	ED_2
1	1.0	1.0	2.0	2.0	0.3	0.7	0.1	0.4
2	1.0	1.0	2.0	2.0	0.3	0.7	1.0	4.0
3	1.0	1.0	2.0	2.0	0.7	0.3	0.1	0.4
4	1.0	1.0	2.0	2.0	0.7	0.3	1.0	4.0
5	1.0	1.0	2.0	2.0	0.6	1.0	0.1	0.4
6	1.0	1.0	2.0	2.0	0.6	1.0	1.0	4.0
7	1.0	1.0	2.0	2.0	1.0	0.6	0.1	0.4
8	1.0	1.0	2.0	2.0	1.0	0.6	1.0	4.0
9	1.5	1.0	2.0	1.5	0.3	0.7	0.1	0.4
10	1.5	1.0	2.0	1.5	0.3	0.7	1.0	4.0
11	1.5	1.0	2.0	1.5	0.7	0.3	0.1	0.4
12	1.5	1.0	2.0	1.5	0.7	0.3	1.0	4.0
13	2.0	1.0	2.0	1.0	0.4	0.4	0.5	0.5
14	5.0	1.0	2.0	0.5	0.3	0.4	0.2	0.2

Table 1: Input Data for Examples 1-14.

the sever immediately switches to i. For problems 1-8, the queues are symmetrical with respect to service rate and holding cost, while in Problems 9-14, the $c\mu$ values are not equal. For cases 9-14, it may make sense to switch from queue 2 to queue 1 without exhausting it. In this case, we searched over all possible threshold policies as well as a larger class of nonexhaustive-threshold policies that we denote by (NONEX-TR). A policy in this class is described by three variables (y_1, y_2, y_{21}). If the server is currently set-up to serve jobs of type 2, and $x_2 = 0$, then the server switches to queue 1 if $x_1 > y_1$. Similarly, if the server is currently set-up to serve jobs of type 1, and $x_1 = 0$, then the server switches to queue 2, if $x_2 > y_2$. Finally, if the server is set-up to process jobs of type 2, and $x_2 > 0$, the server switches to queue 1, if $x_1 > y_{21}$. We note that this is a fairly general class of policies for the case of two job classes. In particular, our heuristic represents a special case within this class of policies. Hence, our heuristic can not do better than the best policy found by a computationally very expensive search over this class of policies. Thus, the difference in performance between our heuristic and the best policy in this class is one measure of the success of the heuristic.

In Table 2, we tabulate the average holding costs per unit time under our heuristic policy as well as the other policies. (In the case of $c_1\mu_1 = c_2\mu_2$, we assumed queue 1 had priority over queue 2 for the $c\mu$ rule.) The results in Table 2 show that our heuristic performed well. The heuristic outperformed the exhaustive, gated and $c\mu$ rule heuristics in every example considered. For problems 1-8, the queues are symmetrical with respect to service rate and holding cost. Hofri and Ross [11] conjecture that an exhaustive, single-threshold policy is optimal. Indeed the best exhaustive, threshold policy (EX-TR) performed as well as any policy tested. The results suggest that our heuristic performed almost optimally. Similarly, in problems 9-14, our heuristic outperformed the best exhaustive threshold policy (EX-TR) in five of six cases, and it performed almost as well as the best policy found in the class of nonexhaustive threshold policies. Considering the fact that the search for the best threshold policy is a nontrivial computational problem, our heuristic, which requires no search, performed very well.

We next tested our heuristic on a large sample of problems with three different types of jobs. In the first set of test problems (Examples 15-40), the mean processing times of the three different

Example	Heuristic	Exhaustive	Gated	$c\mu$	EX-TR	NONEX-TR
1	1.26	1.28	1.41	1.47	1.21	1.21
2	5.42	5.65	8.63	∞	5.20	5.20
3	1.27	1.29	1.41	1.44	1.27	1.27
4	5.58	5.65	8.50	∞	5.36	5.36
5	5.20	5.21	6.81	18.46	5.13	5.13
6	17.73	18.36	35.41	∞	17.46	17.46
7	5.12	5.28	6.51	15.88	5.10	5.10
8	17.73	18.01	34.62	∞	17.66	17.66
9	2.29	2.37	2.51	2.49	2.36	2.19
10	8.24	8.55	13.29	∞	7.96	7.96
11	1.96	2.06	2.34	2.12	2.04	1.96
12	8.29	8.41	13.05	∞	7.84	7.84
13	3.25	3.62	3.88	3.68	3.58	3.15
14	52.5	199.4	58.9	155.4	142.3	37.3

Table 2: Results for Examples 1-14

jobs were the same but their holding costs were different. Examples 15-22 represent systems where the firm gets a large number of jobs that are not very important (low holding costs), and a smaller number of urgent jobs. Examples 23-31 represent systems where the arrival rates of jobs of differing importance are equal. Finally, Examples 32-40 represent a case where the jobs with higher holding costs also have the higher arrival rates. For each of these sets of examples, we varied the set-up times. In Examples 15-40, we again compared our heuristic to the exhaustive, gated and $c\mu$ rules.

The results in Table 4 show that in every problem instance tested for the case of three queues, our heuristic easily outperforms all of these widely used rules. In general, if the set-up times were high enough, the exhaustive regime performed well, and if the set-up times were close to 0, the $c\mu$ rule performed well. However, our heuristic was the only rule that performed well for all of the problems.

Whereas Examples 15-40 have jobs with the same processing times but different holding costs, Examples 41-64 have jobs with different mean processing times, but the same holding cost rates. The data for Examples 41-64 are given in Table 5. Examples 41-49 represent cases where each of the three queues have the same utilization. Examples 50-57 represent cases where the firm has a large quantity of jobs that can be processed very quickly, and a small number of jobs that require a large amount of processing. Finally, Examples 58-64 represent situations where the firm spends most of its time processing jobs that require much processing, but gets a lesser number of quick jobs.

We test six policies in Examples 41-64. These include the heuristic developed in this paper; the exhastive, gated, and $c\mu$ rules; as well as two scheduling policies due to Browne and Yechiali [4]. Browne and Yechiali point out that since the problem of minimizing the sum of (weighted) waiting times appears to be "computationally hard", another objective that can be considered is the (greedy) objective of minimizing the cycle time where the cycle time is the amount of time it takes the server to visit each queue once. (Since Browne and Yechiali only conidered jobs having different processing time distributions and not different holding costs, we did not test their policies in Examples 15-40.) In the exhaustive rule developed by Browne and Yechiali, (EXH- BY), at

Example	c_1	c_2	c_3	μ_1	μ_2	μ_3	λ_1	λ_2	λ_3	ED_1	ED_2	ED_3
15	5	1	0.5	1	1	1	0.2	0.1	0.6	0.1	0.1	0.1
16										1.0	1.0	1.0
17										0.1	1.0	2.0
18										1.0	0.1	2.0
19										1.0	2.0	0.1
20										0.1	0.1	2.0
21										2.0	0.1	0.1
22										0.1	2.0	0.1
23	5	1.5	0.5	1	1	1	0.25	0.25	0.25	0.1	0.1	0.1
24										1.0	1.0	1.0
25										5.0	5.0	5.0
26										0.1	1.0	2.0
27										1.0	0.1	2.0
28										1.0	2.0	0.1
29										0.1	0.1	2.0
30										2.0	0.1	0.1
31										0.1	2.0	0.1
32	5	1	0.5	1	1	1	0.6	0.15	0.15	0.1	0.1	0.1
33										1.0	1.0	1.0
34										5.0	5.0	5.0
35										0.1	1.0	2.0
36										1.0	0.1	2.0
37										1.0	2.0	0.1
38										0.1	0.1	1.0
39										0.1	1.0	0.1
40										1.0	0.1	0.1

Table 3: Input Data for Examples 15-40.

Example	Heuristic	Exhaustive	Gated	$c\mu$
15	9.4	19.2	13.5	8.9
16	29.1	32.6	32.9	∞
17	27.4	33.5	35.5	∞
18	31.9	35.1	34.9	∞
19	23.1	34.5	32.5	∞
20	26.2	30.5	29.7	∞
21	28.8	30.0	27.1	∞
22	12.9	29.9	25.9	∞
23	5.0	7.7	7.8	5.1
24	12.6	14.9	17.4	∞
25	47.1	50.4	71.2	∞
26	11.5	15.6	18.1	∞
27	12.8	15.0	18.0	∞
28	12.9	15.8	17.7	∞
29	9.1	12.9	14.8	∞
30	12.2	13.2	15.2	31.6
31	9.5	12.2	15.3	∞
32	16.0	27.1	37.8	16.2
33	36.1	45.3	110.7	∞
34	110.9	127.1	381.4	∞
35	29.9	47.6	102.3	∞
36	32.0	48.2	97.4	∞
37	34.3	46.1	104.9	∞
38	20.3	33.1	57.8	∞
39	21.7	32.0	58.8	∞
40	24.8	38.4	63.5	∞

Table 4: Results for Examples 15-40

Example	c_1	c_2	c_3	μ_1	μ_2	μ_3	λ_1	λ_2	λ_3	ED_1	ED_2	ED_3
41	1	1	1	8	2	0.5	2	0.5	0.125	0.1	0.1	0.1
42										0.5	0.5	0.5
43										1.0	1.0	0.1
44										0.1	1.0	1.0
45										1.0	0.1	1.0
46										0.1	0.1	2.0
47										2.0	0.1	0.1
48										0.1	2.0	0.1
49	1	1	1	6	2	0.5	3.6	0.2	0.05	0.1	0.1	0.1
50										1.0	1.0	1.0
51										2.0	2.0	0.1
52										2.0	0.1	2.0
53										0.1	2.0	2.0
54										0.1	0.1	2.0
55										0.1	2.0	0.1
56										2.0	0.1	0.1
57	1	1	1	6	2	0.5	1.2	0.2	0.3	0.1	0.1	0.1
58										1.0	1.0	1.0
59										2.0	2.0	0.1
60										2.0	0.1	2.0
61										0.1	2.0	2.0
62										0.1	0.1	2.0
63										0.1	2.0	0.1
64										2.0	0.1	0.1

Table 5: Input Data for Examples 41-64.

Example	Heuristic	Exhaustive	Gated	$c\mu$	EXH-BY	GATE-BY
41	5.1	8.8	8.9	5.4	9.0	9.5
42	10.2	13.5	15.6	∞	14.6	17.8
43	13.5	16.7	20.5	∞	17.4	24.2
44	11.1	16.0	19.1	∞	17.2	22.2
45	14.1	15.8	18.7	∞	18.6	24.3
46	11.4	16.2	18.4	∞	19.2	21.2
47	16.4	17.9	22.8	∞	19.2	28.1
48	11.6	17.5	21.6	∞	50.0	78.5
49	5.5	7.3	9.7	5.5	7.7	12.9
50	15.2	18.9	42.1	∞	20.5	56.5
51	22.5	26.0	70.5	∞	26.5	78.1
52	20.7	24.8	59.1	∞	26.2	74.8
53	17.2	23.1	55.9	∞	27.1	71.5
54	10.0	13.8	20.4	∞	15.2	28.2
55	12.7	17.1	36.0	∞	18.1	40.1
56	15.8	16.9	39.3	∞	19.3	51.5
57	13.7	31.4	23.0	25.4	28.4	23.1
58	46.5	50.2	49.4	∞	52.3	47.9
59	51.1	60.1	62.3	∞	57.4	64.7
60	56.3	60.1	60.8	∞	66.2	66.3
61	43.2	60.0	61.4	∞	58.6	67.0
62	34.5	49.1	42.4	∞	52.1	44.1
63	20.0	45.9	43.4	∞	50.1	44.9
64	38.7	45.9	42.7	∞	48.2	47.8

Table 6: Results for Examples 41-64

the beginning of each cycle, the server calculates the index $\frac{x_i\mu_i^{-1}+ED_i}{\rho_i}$ for each queue i, where $\rho_i = \lambda_i/\mu_i$. The server first exhausts the jobs in the queue that has the lowest index, then switches to the queue with the next lowest index, and so on. Once all of the queues have been visited, the indices are calculated again. The gated regime developed by Browne and Yechiali (GATE-BY) is similar, except that the server ranks the different queues in increasing order of $\frac{x_i\mu_i^{-1}+(1+\rho_i)ED_i}{\rho_i}$. Browne and Yechiali showed that these rules minimize the cycle time.

The results for Examples 41-64 are displayed in Table 6. Our heuristic consistently gave the best results, sometimes resulting in an average holding cost of 50% of that of its nearest competitor. In general, we found that when a queue with the lower $c\mu$ value had a high utilization, ρ_i, then the gated rules did better than the exhaustive rule. On the other hand, if a high $c\mu$ node also had a high utilization, then the exhaustive rules were better than the gated rules since the server remained at the high reward node until it exhausted it. In general, as the set-up times grew large, the cyclic exhaustive rule performed well. However, as the examples in this section clearly indicate, our heuristic outperformed all of these rules from the literature.

6. Conclusions and Further Research

Using notions of reward rate, we have partially characterized an optimal policy for the scheduling of (parallel) queues with set-up times. We used this insight to develop a heuristic. Our simulation

study indicates that, in the case of two queues, the heuristic performs nearly as well as computationally expensive search-based rules. In the case of problems with more than two queues, we have found that our heuristic substantially outperforms other policies analyzed in the literature. Moreover, the simplicity of our algorithm enhances its attractiveness.

Further research should focus on developing a more complete characterization of the optimal policy. This would aid in developing new and possibly more effective heuristics. This is bound to be a very challenging problem, however, since even in the case of controlling two queues with set-up costs, the optimal policy has not yet been completely characterized. Further research should also address systems in which a job has to be processed by more than one server and follows a general route through the system. Recently, Wein and Chevalier [27] addressed a general job-shop scheduling problem where customers arrive randomly to the system and decisions must be made concerning what due date to quote to these customers, when to release their orders into the system, and how to sequence the jobs at each machine. For this problem, they developed effective heuristics by approximating the problem as a Brownian control problem. However, they assumed that no set-up is required between different job classes. Further research is necessary to develop heuristics for this problem when set-up is required between different job classes at each machine.

Acknowledgement:

The first author wishes to thank Professor Demosthenis Teneketzis for many fruitful discussions. The work of the first author was performed while a student at the University of Michigan and was supported under a Graduate Fellowship from the Electrical Engineering and Computer Science Department.

Bibliography

[1] Baker, J.E. and Rubin, I. (1987) Polling with a general-service order table, *IEEE Trans. Comm.* **COM-35**, 283–288.

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

[3] Bell, C. (1971) Characterization and computation of optimal policies for operating an $M/G/1$ queueing system with removable server, *Operations Research* **19**, 208–218.

[4] Browne, S. and Yechiali U. (1989) Dynamic priority rules for cyclic-type queues, *Advances in Applied Probability*, **21**, 432-450.

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

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

[7] Feigin, G. (1991) Non-preemptive scheduling of multiclass single server queues, preprint.

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

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

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

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

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

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

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

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

[16] Liu, Z., Nain, P., and Towsley, D. (1992) On optimal polling policies, to appear in *Queueing Systems.*

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

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

[19] Rajan, R. and Agrawal, R. (1991) Stochastic dominance in homogeneous queueing systems with switchover costs, Preprint, Department of Electrical and Computer Engineering, University of Wisconsin-Madison.

[20] Santos, C. and Magazine, M. (1985) Batching in single operation manufacturing systems, *Operations Res. Letters* **4**, 99–103.

[21] Srinivasan, M.M. (1991) Nondeterministic Polling Systems, *Management Science* **37** 667.

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

[23] Van Oyen, M.P., Pandelis, D.G., and Teneketzis, D. (1992) Optimality of index policies for stochastic scheduling with switching penalties, to appear in *Journal of Applied Probability*

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

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

[26] Wein, L.M., (1991) Due date setting and priority sequencing in a multiclass M/G/1 queue, *Management Science* **37**, 834-850.

[27] Wein, L.M., and Chevalier P., (1992) A broader view of the job shop scheduling problem, *Management Science* **38**, 1018-1033.