For Final Version, see Duenyas, I. and M.P. Van Oyen, Heuristic scheduling of parallel heterogeneous queues with set-ups, Management Science, 42:6, (1996) 814-829.

Heuristic Scheduling of Parallel Heterogeneous Queues with Set-Ups

Izak Duenyas and Mark P. Van Oyen

Technical Report 92-60

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

October 1992
Revised June 1995

Heuristic Scheduling of Parallel Heterogeneous Queues with Set-Ups

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

Mark P. Van Oyen
Department of Industrial Engineering and Management Sciences
Northwestern University, Evanston, Illinois 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 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 a simple heuristic scheduling policy for this problem. Simulation results 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, 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 the average holding cost per unit time (see Baras et al. [3], Buyukkoc et al. [6], Cox and Smith [8], Gittins [13], Nain [25], Nain et al. [26], and Walrand [35]). Other stochastic scheduling problems in the literature for which there are no costs (or no time lost) for switching from one type of job to another may be found in Baras et al. [3], Dempster et al. [9], Gittins [13], Harrison[15],[16], Klimov [18], [19], Lai and Ying [21], Nain [25], Nain et al. [26], Varaiya et al. [34], and Walrand [35]. For systems with no switching times or costs, researchers such as Wein [36] and Wein and Chevalier [37] 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. [14] considered the problem with switching costs and only two types of jobs with the same processing time distributions. Hofri and Ross [17] 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 [27] and Liu et al. [24] have studied systems similar to the one considered here and have partially characterized an optimal policy (in the sense of the stochastic dominance of the queue length process) for the case of homogeneous service processes. Browne and Yechiali [5] considered cycle times in heterogeneous systems and completely characterized scheduling policies that optimize cycle times. Other work has concentrated on performance evaluation and stochastic comparisons of different policies (see Baker and Rubin [2], Levy and Sidi [22], Levy et al. [23], Takagi [30] and Srinivasan [29]). Recently, Federgruen and Katalan [10] have analyzed the performance of exhaustive and gated polling policies where the server is also allowed to idle.

Their policies are applicable to the make-to-stock version of the problem considered here as well. Federgruen and Katalan [11] have also analyzed the impact of changes in setup times on the performance of multi-class production systems.

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 while remaining 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 under a discounted cost criterion. We relate the discounted case to the average cost problem. 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. 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 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 policies 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 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 distribution with 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 strictly positive 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

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

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 (by job service, queue set-up, or idling) until its completion. Idling is allowed at any decision time. The set of decision epochs is assumed to be the set of all arrival epochs, service epochs, set-up completion epochs, and instances of idling. The objective of the optimization problem is to determine a policy $g^* \in G$ that minimizes $\bar{J}(g)$.

For many policies (2.1) may be infinite. To cite a well-studied example, the limited-l cyclic service policies become unstable for $\rho < 1$, as is demonstrated in Kuehn [20] and Georgiadis and Szpankowski [12]. For $\rho < 1$, it is well known that policies such as the exhaustive and gated cyclic polling strategies yield a stable system (see Altman et al. [1]). Thus, finite steady state average

queue lengths exist under an optimal policy, and the objective is to minimize the weighted sum of the average queue lengths.

Our analysis is framed within the class of policies, G, which contains (in general) nonstationary and randomized policies. Nevertheless, it is helpful to explicitly describe the subclass $G^{PM} \subset G$ 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) interarrival period. Thus, the state of the system is described by the vector $X(t) = (X_1(t), X_2(t), \ldots, X_N(t), n(t), \delta(t)) \in \mathcal{S}$, 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 $\mathcal{U} = \{0, 1, 2, \ldots, N\} \times \{0, 1, 2\}$. Suppose at a decision epoch, t, the state is $X(t) = (x_1, x_2, \ldots, x_N, n(t), \delta(t)) \in \mathcal{S}$. Thus, $\delta(t) = 1$, since we require non-preemptive set-ups. Action $U(t) = (n, 2) \in \mathcal{U}$, where $n \neq n(t)$, causes the server to set up node n. Action $U(t) = (n(t), 1)$ results in the service of a job in $n(t)$. 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 the class of policies G. The special case with all switching times equal to 0 has been well studied, with early results found in Cox and Smith [8]. 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. 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. [24] and Rajan and Agrawal [27]. 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.

Although our focus for our heuristic is on the average cost per unit time criterion, we have found it insightful to study the discounted cost criterion as well because it demonstrates the use of reward rate expressions which prove to be pertinent to the heuristic we develop. We define the total expected discounted cost criterion, which we note is finite for any policy: For discount parameter $\alpha > 0$, let

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

As in Harrison [15] we transform the cost criterion of (3.1) to a reward criterion using the device of Bell [4]. Letting $Y_n^g(t)$ denote the right-continuous cumulative departure process from node n under g through time t, we have $X_n^g(t) = X_n(0^-) + A_n(t) - Y_n^g(t)$. One can show that the minimization of $J_\alpha(g)$ is equivalent to the maximization of the following reward criterion:

$$R_\alpha(g) = E\{\sum_{n=1}^N c_n \alpha^{-1} \int_0^\infty e^{-\alpha t} dY_n^g(t)\} = E\{\sum_{n=1}^N \sum_{k=1}^\infty e^{-\alpha T_n^g(k)} c_n \alpha^{-1}\} \ , \tag{3.2}$$

where $T_n^g(k)$ is the kth service completion epoch under g corresponding to a service in node n . 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 $r_n(u)$. Using (3.2), $r_n(u) = c_n\alpha^{-1}E\{\int_0^\infty e^{-\alpha t} dY_n^{g'}(t)\}$. Let $f_{n,k}$ denote the sum of k service durations in queue n. Letting $S_n \stackrel{\Delta}{=} E\{e^{-\alpha f_{n,1}}\}$, we use the i.i.d. nature of successive services to get $E\{e^{-\alpha f_{n,k}}\} = S_n^k$. Thus

$$r_n(u) \ = \ c_n\alpha^{-1} S_n (1 - S_n)^{-1} (1 - S_n^u) E\{e^{-\alpha D_n}\} \ . \tag{3.3}$$

We define the reward rate associated with this sequence of actions to be the ratio of expected discounted reward, $r_n(u)$, to the expected discounted length of time required by the action sequence:

$$\frac{r_n(u)}{E\{\int_0^{D_n+f_{n,u}} e^{-\alpha t}dt\}} = \frac{c_n\alpha^{-1}S_n(1-S_n)^{-1}(1-S_n^u)E\{e^{-\alpha D_n}\}}{\alpha^{-1}(1-S_n^uE\{e^{-\alpha D_n}\})} < h_n, \tag{3.4}$$

where h_n is defined by

$$h_n \triangleq c_nS_n/(1-S_n)\ . \tag{3.5}$$

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 (3.4). 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 [14], Hofri and Ross [17], Liu et al. [24] and Rajan and Agrawal [27]. The novelty of our result lies primarily in our treatment of unequal or heterogeneous service distributions at each queue.

Theorem 1: If $h_i \geq h_j$ for all $j = 1, 2, \ldots, N$ then there exists a policy for which queue i is a top-priority queue that is optimal within G under the discounted cost criterion. Similarly, if $c_i\mu_i \geq c_j\mu_j$ for all $j = 1, 2, \ldots, N$ the same result holds under the average cost per unit time criterion.

Proof : The proof is found in the Appendix.

Since a top-priority policy is optimal for any discount factor $\alpha > 0$, we note that the discounted and average cost per unit time cases can be linked as follows:

$$\lim_{\alpha\to 0}\alpha h_n = \lim_{\alpha\to 0}\alpha c_nE\{e^{-\alpha f_{n,1}}\}/E\{\int_0^{f_{n,1}} e^{-\alpha t}dt\} = c_n\mu_n\ . \tag{3.6}$$

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. We use these concepts of reward rates in the next section to derive a heuristic policy for the problem.

4. A Heuristic Policy

We develop a greedy 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

Consistent with the result of Section 3 that top-priority service of queue 1 is optimal, we restrict attention to a policy that does 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 1 when queue 2 is not empty, as well as the characterization of a rule for idling (i.e., should the server idle at queue i or switch to the other queue?). Our heuristic is based in part on reward rate indices corresponding to action sequences. 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. We begin with the development of the rule for switching, then prescribe a rule for idling.

Rule for Switching

We assume that nodes 1 and 2 are both nonempty ($x_1 > 0, x_2 > 0$) and focus on the question of when to switch from node 2 to 1. We let $\varphi_i(x_i, r)$ denote the reward rate (or expected reward per expected unit of time) 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

$$\varphi_2(x_2, r) = c_2\mu_2 \text{ if } x_2 \geq 1. \tag{4.1}$$

On the other hand, if the server decides to switch to queue 1, 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. Although the server could actually remain at node 1 for a longer amount of time by idling at node 1 for a certain duration of time, we disallow idling in calculating an index for switching to node 1. We also 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 the busy period of node 1, 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 reward rate of this action sequence:

$$\varphi_1(x_1, s) = c_1\mu_1 \frac{\frac{x_1+\lambda_1 ED_1}{\mu_1-\lambda_1}}{ED_1 + \frac{x_1+\lambda_1 ED_1}{\mu_1-\lambda_1} + ED_2} \tag{4.2}$$

$$= c_1\mu_1 \frac{x_1 + \lambda_1 ED_1}{x_1 + \mu_1 ED_1 + (\mu_1 - \lambda_1)ED_2} \ . \tag{4.3}$$

Comparing the terms $\varphi_1(x_1, s)$ and $\varphi_2(x_2, r)$, it is easy to see that regardless of how large the expected set-up times ED_1 and ED_2 are, $\varphi(x_1, s)$ can be larger than $\varphi_2(x_2, r)$, even for $x_1 = 1$, if c_1 is sufficiently large. This means that for large values of c_1, the index for remaining at queue 2 would always be smaller than the index for switching to queue 1, even when queue 1 has only 1 job. It is clear, however, that when set-up times are non-zero, switching to queue 1 as soon as queue 1 has one job is not necessarily optimal, even if c_1 is large. To see this, note that one way to interpret ρ, the utilization of the server, is that (for a stable system) the server is busy processing jobs ρ proportion of the time. The proportion of the time available to the server for for set-ups and idling is $1 - \rho$. However, if the server switches to queue 1 when queue 1 has only one job and set-up times are high compared to processing times, the server may spend a much larger proportion of the time than $(1 - \rho)$ on switching. In such a case, the server spends less than the required proportion of time (λ_2/μ_2) serving queue 2, which would result in instability at queue 2. Note by (4.2), however, that the condition $\varphi(x_1, s) > \rho c_1\mu_1$ implies that the server , on average, spends a proportion greater than ρ of the time actually processing jobs, during the time interval consisting of the set-up of queue 1, the service of queue 1, and the subsequent set-up of queue 2. Hence, we impose this constraint as a requirement to be satisfied before the server is allowed to switch to queue 1. In particular, we use the following heuristic condition for switching from queue 2 to 1 when $x_2 \geq 1$:

$$\varphi_1(x_1, s) > \rho c_1\mu_1 + (1 - \rho)c_2\mu_2 = c_2\mu_2 + \rho(c_1\mu_1 - c_2\mu_2) \tag{4.4}$$

If (4.4) is satisfied, then the constraint $\varphi_1(x_1, s) > \rho c_1\mu_1$ is also satisfied. Also, if $c_1\mu_1 = c_2\mu_2$, then (4.4) will never be satisfied and by Theorem 1, both queues are top-priority queues and it is optimal never to switch from queue 2 to 1 (or from queue 1 to 2) when $x_2 > 0$ (when $x_1 > 0$). We note that the condition in (4.4) has some other desirable characteristics. As ED_1 or ED_2 get large,

x_1 must be increasingly large to merit a switch from 2 to 1 when $x_2 > 0$. Also, as ρ approaches 1, the number of jobs required at queue 1 before a switch is allowed increases, and the policy tends to serve the queues exhaustively.

Rule for Idling

To complete the characterization of our heuristic policy, we specify a policy for idling when there are no jobs in the node the server is currently set up to serve. Equation (4.4) 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 (4.3), a switch from node 1 to 2 that proceeds to exhaust node 2 and returns to set up node one 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

$$\varphi_2(x_2, s) = c_2\mu_2 \frac{x_2 + \lambda_2 ED_2}{x_2 + \mu_2 ED_2 + (\mu_2 - \lambda_2)ED_1} \ . \tag{4.5}$$

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 (only for the purpose of reward rate calculation) that no rewards are earned while idling, and compute the reward rate of the inadmissible policy that idles until the next arrival at node 2, then switches to node 2 to exhaust it, and returns to node one. This results in the following reward rate:

$$\varphi_2'(x_2, s) = c_2\mu_2 \frac{\frac{x_2+1+\lambda_2 ED_2}{\mu_2-\lambda_2}}{\frac{x_2+1+\lambda_2 ED_2}{\mu_2-\lambda_2} + \frac{1}{\lambda_2} + ED_1 + ED_2} \ . \tag{4.6}$$

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

$$\varphi_2'(x_2, s) < \varphi_2(x_2, s) \ , \tag{4.7}$$

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 (4.7) 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:

$$x_2 > \lambda_2 E D_1 \ . \tag{4.8}$$

Similarly, when the server is at node 2, and there are no more jobs to serve, it immediately switches to node 1 if $x_1 > \lambda_1 E D_2$. 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 now 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 $x_2 > \lambda_2 E D_1$. Else, idle until the next arrival to the system.

3. If the server is currently at node 2, $x_2 > 0$, and $\varphi_1(x_1, s) \leq c_1\mu_1\rho + (1-\rho)c_2\mu_2$ then serve one more job at node 2; otherwise

   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.

4. If the server is currently at node 2 and $x_2 = 0$, then switch to node 1 if $x_1 > \lambda_1 E D_2$. 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 (4.4) of our heuristic for two queues guarantees that eventually the length of queue 1 will be less than $\lambda_1 E D_2$ and the length of queue 2 will simultaneously be less than $\lambda_2 E D_1$, (i.e., that the queues will be stable). To see this, first suppose that (4.4) can be satisfied for a finite queue length x_1^*. Note that $x_1 > x_1^*$ is required for the server to switch from queue 2 to 1. Since we assume that $\mu_1 > \lambda_1$ and the server serves queue 1 exhaustively, only the stability of queue 2 is in question. Without loss of generality, assume that at time $t = 0$ the server is set-up to serve queue 2 and that

$x_2 > \lambda_2 ED_1$. The server will serve queue 2 either until it is exhausted or until $x_1 > x_1^*$ (in which case it switches to queue 1). The server will then alternate *without idling* between the exhaustive service of queue 1 and the (possibly non-exhaustive) service of queue 2. For the sake of argument, construe the set-ups of both queues as being associated with the service of queue 1. The epochs of switching to node one occur only at points under which (4.4) is satisfied. Hence, during 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 (i.e., the server is processing jobs and not being set-up nor idling) is greater than 100ρ percent (compare (4.3) and (4.4)). Under our construction, the server is 100% utilized during the remaining periods, which correspond to actual service in queue 2. Thus, the server is utilized greater than 100ρ percent of the time prior to the first instance of idling, and thereby efficiently works off both queues. The first instance of idling occurs when one queue, i, is exhausted and the other, say j, is such that $x_j < \lambda_j ED_i$. Thus, stability is ensured when x_1^* is finite. We conclude with the case where no finite x_1^* exists to satisfy (4.4). In that case, provided x_1 and x_2 are both large at $t = 0$, our heuristic serves both queues exhaustively and without idling until the point at which one queue, i, is exhausted and the other, say j, is such that $x_j < \lambda_j ED_i$. It is well known that exhaustive, nonidling service is stable for $\rho < 1$.

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. 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, it suffices to only consider switching from i to queue $j \in \{1, 2, \ldots, i-1\}$. Then to switch to any queue j, we require that

$$\varphi_j(x_j, s) \geq c_j\mu_j\rho + c_i\mu_i(1-\rho) \text{ and } j \in \{1, 2, \ldots, i-1\} \ , \tag{4.9}$$

where

$$\varphi_j(x_j, s) = c_j\mu_j \frac{x_j + \lambda_j ED_j}{x_j + \mu_j ED_j + (\mu_j - \lambda_j)ED_i}. \tag{4.10}$$

Unlike the case of two queues, there may be more than one queue j that satisfies the constraint (4.9). Thus, we require that the server switch to the one with the highest reward rate, $\varphi_j(x_j, s)$.

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 this case, the server must decide not only whether to idle but also to which queue to switch to. We place a constraint similar to (4.9) on switching from queue i when $x_i = 0$. To develop such a rule, we first note that the reward rate of (4.10), $\varphi_j(x_j, s)$, includes both ED_i and 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

$$\phi_j(x_j, s) = c_j\mu_j \frac{x_j + \lambda_j ED_j}{x_j + \mu_j ED_j}. \tag{4.11}$$

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 $\phi_j(x_j, s) > c_j\mu_j\rho$, 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 $x_k > \lambda_k ED_i$, 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 a queue in this set, it would actually be processing jobs at least ρ fraction of the time until the end of that queue's busy period. From this set, it selects *as a candidate* the queue that has the highest reward rate. On the other hand, if the set σ is empty, another queue may yet be attractive enough to

justify a switch, and the heuristic selects *as a candidate* the queue that has the highest reward rate among all queues $1, ..., N$. The procedure then uses the simple rule developed for the case of two queues to decide whether to idle or to switch to the candidate queue. Having explained the logic of our heuristic, we can now state it formally.

Heuristic Policy for N Queues

Assume that $c_1\mu_1 \geq c_2\mu_2 \geq \ldots \geq c_N\mu_N$, 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 $x_i > 0$ and no jobs have been served in queue i since the last set-up, serve a job in i; otherwise, employ the following switching rule: For all $j < i$, compute $\varphi_j(x_j, s)$ using (4.10). Let $\sigma = \emptyset$. For $j = 1, \ldots, i-1$, if queue j satisfies constraint (4.9), then $\sigma = \sigma \cup j$. If σ is nonempty, then switch to the queue $j \in \sigma$ that has the highest index $\varphi_j(x_j, s)$; otherwise serve one more job of type i.

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

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. [24], Rajan and Agrawal [27]). The optimal idling policy is not known.

3. $\lambda_i = 0$ *for all* $i = 1, \ldots, N$*:* In the case of no arrivals, our heuristic serves all the queues in an exhaustive manner. Once a queue is exhausted, the server switches to the queue that has the highest index $c_j \mu_j \frac{x_j \mu_j^{-1}}{x_j \mu_j^{-1} + D_j}$. Van Oyen et al. [32] 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 the problem considered here, however, an optimal solution is 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. 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, and have obtained results very similar 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. Thus, these 8 cases test the idling rule of our heuristic. They include cases with high as well as moderate utilization and mean set-up times. If one test case pairs the queue with the high arrival rate with a high set-up time as well, the next case pairs that queue with a low set-up time.

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, and so forth. (We found that not switching to any empty queue improved performance, hence our exhaustive and gated policies do not switch into queues that are empty). 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 exhaustive, threshold policies by simulation to find the best policy of that class. Specifically, we denote by (EX-TR) the class of exhaustive, threshold policies defined by the pair (y_1, y_2) 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 the server 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. For this reason, we searched the class of nonexhaustive-threshold policies, which we denote by (NONEX-TR). A policy in this class is described by three variables (y_1, y_2, y_{21}). For $i \neq j$, if the server is currently set-up to serve jobs of type i, and $x_i = 0$, then the server switches to queue j if, and only if, $x_j > y_j$. Finally, if the server is set-up to process jobs of type 2 and $x_2 > 0$, the server switches to queue 1 if, and only 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 the class NONEX-TR. Hence, our heuristic can not do better than the best policy found by a very computationally expensive search over this class of policies. Thus, the difference in performance between our heuristic and the best policy in NONEX-TR is one measure of the success of the heuristic.

In Table 2, we tabulate the average holding costs per unit time (and 95% confidence intervals for the simulation results) 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. For problems 1-8, the queues are symmetrical with respect to service rate and

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.

Example	Heuristic	Exhaustive	Gated	$c\mu$	EX-TR	NONEX-TR
1	1.26 ± 0.01	1.28 ± 0.01	1.41 ± 0.03	1.47 ± 0.04	1.21 ± 0.01	1.21 ± 0.01
2	5.42 ± 0.07	5.65 ± 0.08	8.63 ± 0.12	∞	5.20 ± 0.06	5.20 ± 0.06
3	1.27 ± 0.01	1.29 ± 0.01	1.41 ± 0.02	1.44 ± 0.02	1.27 ± 0.01	1.27 ± 0.01
4	5.58 ± 0.08	5.65 ± 0.09	8.50 ± 0.15	∞	5.36 ± 0.06	5.36 ± 0.06
5	5.20 ± 0.08	5.21 ± 0.07	6.81 ± 0.10	18.46 ± 0.75	5.13 ± 0.08	5.13 ± 0.08
6	17.73 ± 0.42	18.36 ± 0.55	35.41 ± 1.42	∞	17.46 ± 0.45	17.46 ± 0.45
7	5.12 ± 0.07	5.28 ± 0.07	6.51 ± 0.12	15.88 ± 0.83	5.10 ± 0.06	5.10 ± 0.06
8	17.73 ± 0.27	18.01 ± 0.25	34.62 ± 1.55	∞	17.66 ± 0.35	17.66 ± 0.35
9	2.29 ± 0.04	2.37 ± 0.04	2.51 ± 0.06	2.49 ± 0.07	2.36 ± 0.03	2.19 ± 0.03
10	8.24 ± 0.32	8.55 ± 0.35	13.29 ± 0.37	∞	7.96 ± 0.21	7.96 ± 0.21
11	1.96 ± 0.02	2.06 ± 0.03	2.34 ± 0.05	2.12 ± 0.02	2.04 ± 0.02	1.96 ± 0.02
12	8.29 ± 0.41	8.41 ± 0.31	13.05 ± 0.60	∞	7.84 ± 0.40	7.84 ± 0.40
13	3.25 ± 0.08	3.62 ± 0.09	3.88 ± 0.10	3.68 ± 0.08	3.58 ± 0.08	3.15 ± 0.09
14	52.5 ± 3.6	199.4 ± 13.7	58.9 ± 3.7	155.4 ± 15.1	142.3 ± 12.5	37.3 ± 2.1

Table 2: Results for Examples 1-14

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										0.1	1.0	2.0
26										1.0	0.1	2.0
27										1.0	2.0	0.1
28										0.1	0.1	2.0
29										0.1	2.0	0.1
30	5	1	0.5	1	1	1	0.6	0.15	0.15	0.1	0.1	0.1
31										1.0	1.0	1.0
32										0.1	1.0	2.0
33										1.0	0.1	2.0
34										1.0	2.0	0.1
35										0.1	0.1	1.0
36										0.1	1.0	0.1
37										1.0	0.1	0.1
38	1	5	2	4	0.2	0.25	0.8	0.04	0.1	0.1	0.1	0.1
39										1.0	1.0	1.0
40										3.0	0.1	0.1
41										0.1	3.0	0.1
42										0.1	0.1	3.0
43										3.0	3.0	0.1
44										0.1	3.0	3.0

Table 3: Input Data for Examples 15-44.

holding cost. Hofri and Ross [17] conjecture that an exhaustive, single-threshold policy is optimal. Indeed the best exhaustive, threshold policy (EX-TR) performed as well as any policy tested. In examples 9-14, our heuristic again performed very well and the difference between our heuristic and the best policy found in the class of non-exhaustive threshold policies was in general not large. Considering the fact that the search for the best threshold policy is a nontrivial computational problem, our heuristic, requiring no search, performed very well.

We tested our heuristic on a large sample of problems with three different types of jobs. In the first set of test problems (Examples 15-37), the mean processing times of the three different jobs were the same but their holding costs were different. On the other hand, in Examples 38-44, all

jobs have different holding costs and different mean processing times. 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 23-29 represent systems where the arrival rates of jobs of differing importance are equal. Examples 30-37 represent systems 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. For Examples 15-44, we again compared our heuristic to the exhaustive, gated and $c\mu$ rules.

The results in Table 4 indicate that 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 15-37 have jobs with the same processing times but different holding costs, Examples 45-68 have jobs with different mean processing times, but the same holding cost rates (see Table 5). Examples 45-52 represent cases where each of the three queues have the same utilization. Examples 53-60 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 61-68 represent situations where the firm spends most of its time processing jobs that require much processing, but gets fewer quick jobs.

We test six policies in Examples 45-68. These include the heuristic developed in this paper; the exhaustive, gated, and $c\mu$ rules; as well as two scheduling policies due to Browne and Yechiali [5]. 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 or maximizing 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 considered jobs having different processing time distributions and not different holding costs, we did not test their policies in Examples 15-44.) In particular, in a symmetric system, Browne and Yechiali's rules for maximizing the cycle time reduce to serving the longest queue (over one cycle). Since this is

Example	Heuristic	Exhaustive	Gated	$c\mu$
15	9.4 ± 0.6	19.2 ± 0.6	13.5 ± 0.8	8.9 ± 0.9
16	29.1 ± 1.5	32.6 ± 1.4	32.9 ± 1.3	∞
17	27.4 ± 1.0	33.5 ± 1.3	35.5 ± 1.8	∞
18	31.9 ± 1.6	35.1 ± 1.4	34.9 ± 1.7	∞
19	23.1 ± 1.4	34.5 ± 1.7	32.5 ± 2.0	∞
20	26.2 ± 0.4	30.5 ± 1.1	29.7 ± 1.4	∞
21	28.8 ± 0.8	30.0 ± 1.0	27.1 ± 1.0	∞
22	12.9 ± 0.5	29.9 ± 1.2	25.9 ± 1.1	∞
23	5.0 ± 0.2	7.7 ± 0.3	7.8 ± 0.2	5.1 ± 0.1
24	12.6 ± 0.2	14.9 ± 0.3	17.4 ± 0.3	∞
25	11.5 ± 0.2	15.6 ± 0.3	18.1 ± 0.4	∞
26	12.8 ± 0.3	15.0 ± 0.3	18.0 ± 0.4	∞
27	12.9 ± 0.2	15.8 ± 0.3	17.7 ± 0.5	∞
28	9.1 ± 0.3	12.9 ± 0.4	14.8 ± 0.3	∞
29	9.5 ± 0.4	13.2 ± 0.2	15.3 ± 0.6	∞
30	16.0 ± 1.0	27.1 ± 1.4	37.8 ± 2.0	16.2 ± 1.0
31	36.1 ± 2.5	45.3 ± 2.1	$110.\ 7 \pm 4.3$	∞
32	29.9 ± 1.9	47.6 ± 3.8	102.3 ± 8.2	∞
33	32.0 ± 2.1	48.2 ± 2.5	97.4 ± 8.4	∞
34	34.3 ± 2.5	46.1 ± 2.8	104.9 ± 9.1	∞
35	20.3 ± 1.0	33.1 ± 1.8	57.8 ± 5.2	∞
36	21.7 ± 1.7	32.0 ± 2.4	58.8 ± 5.4	∞
37	24.8 ± 2.0	38.4 ± 2.9	63.5 ± 4.0	∞
38	11.4 ± 0.7	21.0 ± 1.2	17.7 ± 0.6	11.4 ± 0.9
39	19.6 ± 0.7	26.0 ± 1.4	26.1 ± 1.3	∞
40	23.0 ± 1.0	27.0 ± 1.5	28.5 ± 1.4	∞
41	16.5 ± 1.4	26.6 ± 2.0	25.8 ± 1.6	∞
42	20.0 ± 0.7	27.5 ± 1.4	29.2 ± 1.7	∞
43	29.7 ± 1.6	33.9 ± 1.2	37.6 ± 2.0	∞
44	25.9 ± 1.4	35.9 ± 1.8	38.4 ± 1.7	∞

Table 4: Results for Examples 15-44

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
45	1	1	1	8	2	0.5	2	0.5	0.125	0.1	0.1	0.1
46										0.5	0.5	0.5
47										1.0	1.0	0.1
48										0.1	1.0	1.0
49										1.0	0.1	1.0
50										0.1	0.1	2.0
51										2.0	0.1	0.1
52										0.1	2.0	0.1
53	1	1	1	6	2	0.5	3.6	0.2	0.05	0.1	0.1	0.1
54										1.0	1.0	1.0
55										2.0	2.0	0.1
56										2.0	0.1	2.0
57										0.1	2.0	2.0
58										0.1	0.1	2.0
59										0.1	2.0	0.1
60										2.0	0.1	0.1
61	1	1	1	6	2	0.5	1.2	0.2	0.3	0.1	0.1	0.1
62										1.0	1.0	1.0
63										2.0	2.0	0.1
64										2.0	0.1	2.0
65										0.1	2.0	2.0
66										0.1	0.1	2.0
67										0.1	2.0	0.1
68										2.0	0.1	0.1

Table 5: Input Data for Examples 45-68.

similar to the optimal policy for symmetric systems, which serves queues exhaustively and switches to the longest queue once a queue has been exhausted, we tested the performance of the Browne and Yechiali rules for maximizing the cycle times. In the exhaustive rule developed by Browne and Yechiali, (EXH-BY), at 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 switches to the queue with the highest index. Once this queue is exhausted, the indices for the queues that have not been served in that cycle are recalculated, and the server switches to the one with the highest index among these remaining queues in the cycle, 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 decreasing order of $\frac{x_i \mu_i^{-1} + (1+\rho_i) ED_i}{\rho_i}$. Browne and Yechiali showed that these rules maximize the cycle time, and it is easy to see that for the case where the queues are homogeneous, these rules reduce to serving the longest queue among all queues yet unserved in the cycle. As before, we found that not switching to an empty queue improved the performance of the rules, and thus the rule we implemented prevented switching into empty queues.

The results for Examples 45-68 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. (It is interesting to note that whereas Levy et al. [23] have shown that the total workload in the system is less under the exhaustive rule than under the gated policy, this result does not extend to the average weighted waiting time criterion, as indicated by our simulation results in which the gated policy outperformed the exhaustive rule for some examples and was outperformed by the exhaustive rule for others.) Our heuristic, however, consistently gave the best results.

Finally, examples 69-75 demonstrate that the performance of our heuristic does not deteriorate as the number of queues increase. In these examples, the system has six queues, and the holding

Example	Heuristic	Exhaustive	Gated	$c\mu$	EXH-BY	GATE-BY
45	5.1 ± 0.3	8.8 ± 0.5	8.9 ± 0.4	5.4 ± 0.2	8.9 ± 1.0	9.3 ± 0.4
46	10.2 ± 0.6	13.5 ± 0.3	15.6 ± 0.8	∞	13.6 ± 1.1	15.9 ± 1.0
47	13.5 ± 0.5	16.7 ± 0.8	20.5 ± 0.9	∞	17.0 ± 0.5	21.5 ± 0.6
48	11.1 ± 0.4	16.0 ± 0.4	19.1 ± 0.8	∞	16.3 ± 0.7	20.5 ± 0.5
49	14.1 ± 0.3	15.8 ± 0.6	18.7 ± 0.9	∞	16.5 ± 0.7	20.6 ± 0.7
50	11.4 ± 0.6	16.2 ± 1.4	18.4 ± 1.0	∞	16.9 ± 0.9	19.6 ± 0.5
51	16.4 ± 0.8	17.9 ± 1.0	22.8 ± 1.5	∞	17.6 ± 0.6	23.7 ± 0.7
52	11.6 ± 0.4	17.5 ± 1.0	21.6 ± 1.2	∞	17.1 ± 0.6	23.5 ± 0.7
53	5.5 ± 0.2	7.3 ± 0.5	9.7 ± 0.1	5.5 ± 0.1	7.3 ± 0.2	9.7 ± 0.7
54	15.2 ± 0.7	18.9 ± 1.1	42.1 ± 2.0	∞	18.8 ± 0.8	44.5 ± 3.7
55	22.5 ± 0.8	26.0 ± 1.1	70.5 ± 4.1	∞	25.5 ± 1.0	70.7 ± 3.4
56	20.7 ± 0.7	24.8 ± 0.9	59.1 ± 3.5	∞	24.0 ± 0.3	63.1 ± 3.1
57	17.2 ± 0.5	23.1 ± 1.8	55.9 ± 3.0	∞	23.8 ± 0.5	60.3 ± 1.9
58	10.0 ± 0.4	13.8 ± 0.7	20.4 ± 1.0	∞	13.5 ± 0.7	22.3 ± 1.3
59	12.7 ± 0.5	17.1 ± 0.9	36.0 ± 1.7	∞	16.5 ± 0.8	38.7 ± 1.0
60	15.8 ± 0.6	16.9 ± 0.5	39.3 ± 1.5	∞	17.0 ± 0.7	40.9 ± 2.0
61	13.7 ± 0.6	31.4 ± 2.9	23.0 ± 1.9	25.4 ± 2.5	28.5 ± 2.1	21.4 ± 1.3
62	46.5 ± 2.7	50.2 ± 2.5	49.4 ± 2.8	∞	50.0 ± 3.0	48.1 ± 2.0
63	51.1 ± 3.2	60.1 ± 4.1	62.3 ± 4.2	∞	57.4 ± 3.1	63.2 ± 5.2
64	56.3 ± 3.5	60.1 ± 4.1	60.8 ± 3.7	∞	59.5 ± 2.5	63.8 ± 5.1
65	43.2 ± 2.7	60.0 ± 3.5	61.4 ± 1.9	∞	59.4 ± 2.2	64.6 ± 5.8
66	34.5 ± 2.3	49.1± 4.5	42.4 ± 3.5	∞	45.5 ± 2.1	40.4 ± 2.4
67	20.0 ± 1.5	45.9 ± 3.5	43.4 ± 2.5	∞	45.7 ± 1.6	41.2 ± 2.4
68	38.7 ± 1.9	45.9 ± 2.5	42.7 ± 1.7	∞	45.5 ± 1.4	43.6 ± 3.0

Table 6: Results for Examples 45-68

Example	ED_1	ED_2	ED_3	ED_4	ED_5	ED_6	Heuristic	Exhaustive	Gated
69	0.1	0.1	0.1	0.1	0.1	0.1	5.4 ± 0.2	8.3 ± 0.4	9.7 ±0.6
70	1.0	1.0	1.0	1.0	1.0	1.0	16.5 ± 0.8	25.0 ± 0.8	35.4 ± 2.1
71	3.0	3.0	3.0	0.1	0.1	0.1	31.9 ± 1.4	38.6 ± 1.5	54.8 ± 2.3
72	0.1	0.1	0.1	3.0	3.0	3.0	12.9 ± 0.7	36.9 ± 1.8	52.8 ± 1.9
73	0.1	0.1	0.1	0.1	5.0	5.0	14.8 ± 0.7	41.6 ± 1.5	58.8 ± 2.9
74	0.1	0.1	5.0	5.0	0.1	0.1	15.5 ± 0.7	42.2 ± 2.5	57.7 ± 1.9
75	0.1	0.1	5.0	5.0	5.0	5.0	25.7 ± 0.7	75.8 ± 4.8	100.8 ± 4.1

Table 7: Results for Examples 69-75

costs for the queues are respectively 5, 2, 0.5, 0.4, 0.3, 0.2. The processing rate is 1 for all the queues, while the arrival rates equal 0.2 for queues 1 and 2 and 0.1 for all other queues. The mean set-up times for each queue as well as the average holding cost per unit time obtained under each policy is displayed in Table 7. The results in Table 7 are representative of the performance of the heuristic as the number of queues increases. We found that as the number of queues increases, the difference in the performance of our heuristic and the exhaustive and gated policies increased due to the server's having more opportunities to switch to queues with higher reward rates.

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 policy. 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, our study suggests that the heuristic substantially outperforms other widely used policies that have been analyzed in the literature. Moreover, the simplicity of the algorithm enhances its attractiveness.

Further research is necessary to develop a more complete characterization of the optimal policy. This would aid in developing new and possibly more effective heuristics. This is doubtless 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. Such a system without set-up costs has recently been addressed by Wein and Chevalier [37].

Acknowledgments:

The work of the first author was partially supported by NSF Grant No. DDM-9308290, and that of the second author by Northwestern University Grant 510-24XJ. The authors would like to thank Professors Demosthenis Teneketzis, Rajeev Agrawal, and Awi Federgruen, as well as two anonymous reviewers for many helpful comments that have improved the content and clarity of

this paper.

Appendix: Proof of Theorem 1:

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 $\bar{r}_u \in \mathbb{R}$ and requires an expected discounted 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 reward rate is at most $\max_{u \in \mathcal{U}} \bar{r}_u / \bar{\sigma}_u$; equivalently,

$$(\sum_{u \in \mathcal{U}} p_u \bar{r}_u)/(\sum_{u \in \mathcal{U}} p_u \bar{\sigma}_u) \leq \max_{u \in \mathcal{U}} \bar{r}_u / \bar{\sigma}_u. \tag{A.1}$$

Proof of Theorem 1 for the Discounted Cost Case: Without loss of generality, suppose h_1 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 because jobs of type one offer the greatest single stage reward rate, an optimal policy must serve node one exhaustively. We then justify greedy service in node one. For the sake of presentation, we initially assume g to be non-randomized and stationary, and we remove this restriction later.

Suppose that policy g does not exhaust node one. 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 g chooses to switch to node j at $t = 0$; thus $U^g(0) = (j, 2)$ 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 $t(2) = D_j$. 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, $U^g(t(L-1)) = (1, 2)$, 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'. Let the random variable $g(l)$ taking values in $\{1, 2, \ldots, N+1\}$ denote the job, if any, served during stage l, where $g(l) = N+1$ with the probability that the server

idled in or set up any queue during stage l. Thus, $g(1) = N+1$ and $g(2) = j$. Let the random variable $r(g(l))$ denote the single stage reward associated with stage l and control selection $g(l)$, where by (3.2), $r(g(l)) = c_{g(l)}\alpha^{-1}e^{-\alpha f_{g(l),1}}$ for $g(l) \leq N$ and $r(g(l)) = 0$ for the aggregated state $g(l) = N+1$. Define $\sigma(g(l)) = t(l+1) - t(l)$. For $g(l) \leq N$, $\sigma(g(l)) = f_{g(l),1}$.

In accordance with (3.2), we define $R_\alpha(g^{L-1})$ to be the total expected discounted reward earned under policy g from stages $1, 2, \ldots, L-1$ during $[0, t(L))$. 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 $f_{1,1}$. During $[f_{1,1}, f_{1,1} + t(L))$, $\tilde{g}$ mimics the actions taken by g during $[0, t(L))$, the first $L-1$ stages. At time $t(L+1) = t(L) + f_{1,1}$, both g and $\tilde{g}$ reach the same state along any realization, and $\tilde{g}$ mimics g from that point on. Note that the construction of $\tilde{g}$ is feasible, and that the average single-stage reward earned by serving a single job of node one is given by

$$E\{e^{-\alpha f_{1,1}}\}c_1\alpha^{-1} = h_1\alpha^{-1}(1 - S_1) = h_1 E\{\int_0^{f_{1,1}} e^{-\alpha t}dt\} \ . \tag{A.2}$$

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 (3.2) and (A.2) as

$$\begin{aligned}
&R_\alpha(\tilde{g}) - R_\alpha(g) \\
&= E\{e^{-\alpha f_{1,1}}[c_1\alpha^{-1} + R_\alpha(g^{L-1})]\} - [R_\alpha(g^{L-1}) + E\{e^{-\alpha t(L)}c_1\alpha^{-1}e^{-\alpha f_{1,1}}\}] \\
&= [h_1\alpha^{-1}(1 - S_1) + S_1 R_\alpha(g^{L-1})] - [R_\alpha(g^{L-1}) + E\{e^{-\alpha t(L)}\}h_1\alpha^{-1}(1 - S_1)] \\
&= \alpha^{-1}(1 - S_1)(1 - E\{e^{-\alpha t(L)}\})[h_1 - R_\alpha(g^{L-1})/(\alpha^{-1}(1 - E\{e^{-\alpha t(L)}\}))] \ .
\end{aligned} \tag{A.3}$$

Let $H(l)$ be defined as the information history vector that records current and past states and decision epochs: $\{X(t(i)), t(i) : i = 0, 1, \ldots, l\}$. Since $r(g(1)) = 0$, we see that

$$R_\alpha(g^{L-1}) = E\{\sum_{l=2}^{L-1} e^{-\alpha t(l)} r(g(l))\} = \sum_{l=2}^{\infty} E\{\mathbb{1}\{L > l\}e^{-\alpha t(l)}E\{r(g(l)) \mid H(l), L > l\} \ \} \ . \tag{A.4}$$

Using Lemma 1 and the definition of h_1, it follows that

$$E\{r(g(l)) \mid H(l), L > l\}/E\{\int_0^{t(l+1)-t(l)} e^{-\alpha t} dt \mid H(l), L > l\} \leq h_1 \ . \tag{A.5}$$

Thus, (A.4) and (A.5) yield

$$R_\alpha(g^{L-1}) \leq \sum_{l=2}^{\infty} E\{\mathbb{1}\{L > l\} h_1 E\{\int_{t(l)}^{t(l+1)} e^{-\alpha t} dt | H(l), L > l\}\} = h_1 E\{\int_{t(2)}^{t(L)} e^{-\alpha t} dt\} \ . \tag{A.6}$$

Since $\alpha^{-1}(1 - E\{e^{-\alpha t(L)}\}) = E\{\int_0^{t(L)} e^{-\alpha t} dt\}$ and $Et(2) > 0$, it follows from (A.3) and (A.6) that $R_\alpha(\tilde{g}) > R_\alpha(g)$. Repeated application of the preceding argument at every point of non-exhaustive service at node one establishes the optimality of exhaustive service at node one.

The preceding construction applies to a randomized and/or nonstationary policy g as well. For example, g is randomized and chooses with probability p to leave queue one nonexhaustively at a given instance, the policy $\tilde{g}$ is simply specified to incorporate the interchange with probability p.

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 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. □

Proof of Theorem 1 for the Average Cost Case:

The argument is similar to the proof for the discounted cost case, so we present the differences. Let queue one maximize $c_i\mu_i$ and define as before the initial condition at $t = 0$, L (a random variable), policies g and $\tilde{g}$, $t(\cdot)$, $g(\cdot)$, and $\sigma(g(\cdot))$. Because g is assumed optimal, we recall that $\bar{J}(g) < \infty$ and the lim sup in (2.1) reduces to a *lim* for g and any other policy ($\tilde{g}$) of no greater cost. We find that if $L = \infty$ with strictly positive probability, then $t(L) = \infty$ with strictly positive probability and it can be shown that $\bar{J}(g) = \infty$. Thus policy g cannot be optimal because stable policies exist, and L is finite with probability 1. Instead of comparing g and $\tilde{g}$ using rewards and reward rates, we use the cost formulation directly. For our construction, policies g and $\tilde{g}$ are coupled at time $t(L+1) = t(L) + f_{1,1}$ and incur identical costs thereafter. Thus, we compare the expected cumulative costs incurred by g and $\tilde{g}$ prior to $t(L+1)$. We note that each job served

during $(t(2), t(L)]$ under g is delayed by $f_{1,1}$ 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 $f_{1,1}$ under $\tilde{g}$ and at $t(L)+f_{1,1}$ 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 holding cost of the stage l action be denoted by $C(g(l))$, where $C(g(l)) = c_{g(l)}$ for $g(l) \leq N$ and $C(N+1) = 0$. Because $C(N+1) = 0$, for our purposes, it suffices to note that for the aggregated state $N+1$, $\sigma(N+1)$ has a finite mean. We note that $g(1) = N+1$. From time $t(L+1)$ onwards, $\tilde{g}$ has an expected *cumulative* (not average cost per unit time) cost advantage over policy g, which we denote as $Z(g, \tilde{g})$. Thus,

$$
\begin{aligned}
Z(g,\tilde{g}) &= E\{\int_0^\infty \sum_{n=1}^{N} c_n \left(X_n^g(t) - X_n^{\tilde{g}}(t)\right) dt\} && \text{(A.7)}\\
&= E\{c_1 t(L) - \sum_{l=1}^{L-1} C(g(l)) f_{1,1}\} && \text{(A.8)}\\
&= \sum_{l=2}^{\infty} E\{\mathbb{1}\{L > l\}(c_1 \sigma(g(l)) - C(g(l)) f_{1,1})\} + c_1 E\{D_j\} && \text{(A.9)}\\
&> \sum_{l=2}^{\infty} E\{\mathbb{1}\{L > l\}(c_1 E\{\sigma(g(l))|H(l), L > l\} - E\{C(g(l))|H(l), L > l\} E\{f_{1,1}\})\}\ , && \text{(A.10)}
\end{aligned}
$$

where we have used the fact that $f_{1,1}$ is independent of all else. To conclude that $Z(g, \tilde{g}) \in (0, \infty]$, it suffices to show that for $l \in \{2, 3, \ldots, L-1\}$,

$$
E\{C(g(l))|H(l), L > l\} / E\{\sigma(g(l))|H(l), L > l\} \leq c_1 / E\{f_{1,1}\} = c_1 \mu_1\ . \qquad \text{(A.11)}
$$

This follows from Lemma 1. There exists a perturbation of g that serves a single additional job of queue one at the first instance of non-exhaustion and results in an expected cumulative cost savings in $(0, \infty]$. If, following the job of queue one inserted at time $t(1) = 0$, additional jobs remain in queue 1, apply the argument thus far iteratively until the resulting perturbation of g, say g', exhaustively serves queue 1 during the visit at $t(1)$. Thus, $Z(g, g') \in (0, \infty]$.

To conclude, we build on this result to show that a top-priority policy exists which performs at least as well as g with respect to *average cost per unit time*. Consider a policy g'' with a countable number of stages. The nth stage removes the nth instance of non-exhaustion of queue 1. The sequencing of jobs not in queue 1 is unaffected. Note that our construction implies $Z(g, g'') \in (0, \infty]$

and since $\bar{J}(g) \in (0, \infty)$, it follows that $\bar{J}(g'') \in (0, \infty)$. There exists a policy that always exhausts queue one and performs at least as well as any other policy in G.

The proof of the greedy property follows using the argument made in the discounted case, now extended as above to the average cost case. □

Bibliography

[1] Altman, E., Konstantopoulos, P., and Liu, Z. (1992) Stability, monotonicity and invariant quantities in general polling systems, *Queueing Systems* **11**, 35–57.

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

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

[4] 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.

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

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

[7] Conway, R.W., Maxwell, W.L., and Miller, L.W. (1967) *Theory of Scheduling,* Addison-Wesley, Reading, MA.

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

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

[10] Federgruen, A. and Z. Katalan (1993a) "The stochastic economic lot scheduling problem: Cyclical base-stock policies with idle times," Working paper, Graduate School of Business, Columbia University, New York, NY.

[11] Federgruen, A. and Z. Katalan (1993b) "The impact of setup times on the performance of multi-class service and production systems," Working paper, Graduate School of Business, Columbia University, New York, NY.

[12] Georgiadis, L. and Szpankowski, W. (1992) Stability of Token Passing Rings, *Queueing Systems*, **11**, 7–34.

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

[14] 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.

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

[16] Harrison, J.M. (1975b) Dynamic Scheduling of a Multiclass Queue: Discount Optimality, *Operations Research* **23**, 270–282.

[17] 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.

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

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

[20] Kuehn,P.J. (1979) Multiqueue systems with nonexhaustive cyclic service, *Bell Syst. Tech. J.* **58**, 671-698.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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