STOCHASTIC SCHEDULING OF A BATCH PROCESSING MACHINE WITH INCOMPATIBLE JOB FAMILIES

Izak Duenyas
and
John J. Neale
Department of Industrial & Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117

Technical Report 95-1

January 1995

STOCHASTIC SCHEDULING OF A BATCH PROCESSING MACHINE WITH INCOMPATIBLE JOB FAMILIES

IZAK DUENYAS and JOHN J. NEALE
Department of Industrial and Operations Engineering
University of Michigan, Ann Arbor, Michigan, 48109

Abstract

We consider the control of a single batch processing machine with random processing times and incompatible job families (jobs of different families cannot be processed together in the same batch). Holding costs are incurred for each unit of time that a job waits in the system before being served, and the objective is to minimize the long-run average cost per unit time. We first determine optimal policies for the static problem where all jobs are available simultaneously. We next characterize the optimal policies for certain problems with dynamic arrivals of jobs under the restriction that the machine is not allowed to idle. Finally, we develop a simple heuristic scheduling policy to control the machine. Simulation results are provided to demonstrate the effectiveness of our heuristic over a wide range of problem instances and to compare its performance with existing heuristics.

1 Introduction

There are many manufacturing operations in which a single machine processes several parts simultaneously. Examples include plating baths, heat-treating ovens, kilns for drying lumber, and diffusion and oxidation ovens in semiconductor wafer fabrication [5]. Despite the numerous practical examples of such batch processing machines, how to optimally control these machines remains largely unknown. We will define a batch processing machine, also referred to as a bulk service queue in the queueing literature, to be a single machine which can process up to a certain number of jobs (the capacity of the machine) at the same time. The processing time is independent of the number

of jobs in the batch. Systems that have such characteristics also arise in bus, subway and elevator operations [1], as well as in material handling.

When a batch processing machine is available and the number of jobs ready to be processed is less than the capacity of the machine, a nontrivial decision must be made to either process a partial load or wait for additional jobs to arrive. When the machine serves multiple job families and jobs from different families cannot be processed together, the decisions also include selecting which job family to serve and thus become even more complicated.

There is extensive literature in queueing theory on bulk processing. Most of this literature, however, focuses on performance evaluation rather than control. Neuts [11] and Deb and Serfozo [3] have addressed the problem of controlling a bulk service queue with Poisson arrivals of a single type of job. Deb and Serfozo proved that the optimal policy for this system is a control limit policy. Powell and Humblet [13] developed efficient computational procedures to describe the characteristics of a queue controlled by this optimal policy. Makis [10] considered the case where the waiting time of any job can not exceed a constant T. Gurnani et al. [6] considered a two-stage serial-batch system with the serial stage feeding a single batch processing machine. They assumed a control-limit policy for loading the batch machine and performed an average cost analysis to compute the optimal control limit.

In the deterministic scheduling literature, the problem of scheduling batch processing machines has also been addressed. Ikura and Gimple [8] studied the problem of scheduling a single batch processing machine in the presence of release dates and due dates. Lee et al. [9] developed an efficient algorithm to minimize the number of tardy jobs under the same assumptions. Uzsoy [15] provided a review of the deterministic literature on scheduling of batch processing machines and also developed an efficient optimal algorithm for minimizing the makespan and heuristic algorithms for minimizing maximum lateness. Chandru et al. [2] and Lee et al. [9] have also considered deterministic scheduling of batch processing machines where jobs from different families can be processed together in the same batch.

The highly stochastic environment of wafer fabrication has led to recent work in the semiconductor manufacturing literature to address batch processing with stochastic arrivals (Uzsoy et al. [16], [17] provided a comprehensive review of production planning and scheduling models in the semiconductor industry). Glassey and Weng [5] developed a dynamic batching heuristic (DBH) which uses information about future arrivals to minimize waiting time in queue for the case of a single job

type. Fowler et al. [4] developed two heuristics for the single and multiple product cases which also take into account information about future arrivals. Their Next Arrival Control Heuristics (NACH) consider only the arrival time of the next job from each family in their control decisions. Weng and Leachman [18] also developed an effective heuristic for the single and multiple products cases. However, their heuristic assumes that arrival times for all future jobs arriving during a planning horizon are known with certainty. All of the above papers which focus on the problem with multiple, incompatible job types assume that the arrival times of some future jobs are known. The problem of how to control a batch processing machine when no information on future arrivals is available has received less attention.

In this paper, we focus on the optimal control of a batch processing machine with incompatible job families. We derive new structural results for the optimal policy and develop a new heuristic for the cases with and without future arrival time information. The rest of the paper is organized as follows. In the next section, we formulate the problem and introduce notation that will be used throughout the paper. Section 3 contains results for the static version of the problem where all jobs are available simultaneously. Section 4 characterizes the optimal policies for some dynamic versions of the problem under the added restriction that the machine is not allowed to idle. In section 5 we present a simple heuristic scheduling policy to control the machine, and provide simulation results in section 6 which demonstrate the heuristic's effectiveness. Finally, section 7 contains some concluding remarks and identifies future research directions.

2 Problem Formulation and Notation

We consider the problem of scheduling jobs from incompatible families on a single batch processing machine. The jobs belong to m different job families, and only jobs of the same family may be processed together in a batch. All jobs require one unit of machine capacity, but the machine may have different capacities for the different job families. Let K_j denote the capacity of the machine when serving family j. Jobs from family j require a random processing time (strictly positive) with mean $1/\mu_j$ and cumulative distribution function F_j. The processing times are assumed independent. We assume that any preparation time is included in the service time, and that preemptions are not allowed.

Jobs of family j waiting in queue to be served incur a holding cost of c_j per job per unit time, where $c_j \geq 0$ for all j. The buffer in front of the machine is assumed to be unlimited. The objective

is to minimize the long run average cost per unit time. Note that by setting $c_j = 1$ for all j, this objective will minimize the average waiting time of jobs in the queue and consequently the average number of jobs in the queue by Little's law.

We consider both static and dynamic versions of this problem. In the static version, all jobs are assumed ready to be processed at the present time, with no future arrivals. In the dynamic version, jobs of family j arrive to the machine with rate λ_j independent of all other processes. As a necessary condition for stability, we assume that $\sum_{j=1}^{m} \frac{\lambda_j}{K_j \mu_j} < 1$.

3 Static Problem

In this section, we show how a deterministic result obtained by Uzsoy [15] can be extended to the stochastic model of this paper. We consider the static problem of n jobs awaiting scheduling at a batch service machine. Let n_j denote the number of jobs of family j and B_i denote the ith batch in the final schedule. The following Lemma describes the structure of the batches that are formed in the optimal solution to this static problem.

Lemma 1 *There exists an optimal schedule in which all batches are full except possibly the last batch of each family occurring in the schedule.*

Proof: Consider an optimal schedule which contains a partially full batch B_k of jobs from family j which is not the last batch containing jobs of that family. Let B_l be the last batch in the schedule containing jobs of family j. Denote the time at which batch B_k begins processing under this schedule by the random variable t_k and the time at which batch B_l begins processing by the random variable t_l. Since B_l comes after B_k in the schedule, we know that $E[t_l] - E[t_k] \geq 0$. Take any job from batch B_l and include it in batch B_k, keeping the rest of the schedule exactly the same. We know that this can be done since B_k is not full. In the new schedule that results from this change, all jobs will begin service at exactly the same time as before except for the job that was changed. This job will begin service $t_l - t_k$ time units earlier than before, so the new schedule will have an expected total cost which is $c_j E[t_l - t_k]$ less than the original schedule. Since $c_j E[t_l - t_k] \geq 0$, the new schedule is no worse than the original. Repeating this procedure results in an optimal schedule with the desired property. □

This result, together with the fact that jobs of the same family have the same holding cost, allows us to determine an optimal formation of batches. For each job family j, arbitrarily select

jobs to form $\lfloor n_j/K_j \rfloor$ full batches and (possibly) one partially full batch of $n_j - \lfloor \frac{n_j}{K_j} \rfloor K_j$ jobs, where $\lfloor x \rfloor$ denotes the largest integer smaller than x. Since each job of family j has the same holding cost, it does not matter how these jobs are assigned to the batches. As in Uzsoy [15], we can now view each batch B_k containing jobs of family j as an individual job with mean processing time $1/\mu_j$ and weight W_k, where W_k is the sum of the weights (holding costs) of all jobs in batch k. The optimal sequencing of these $\sum_{j=1}^{m} \lceil \frac{n_j}{K_j} \rceil$ batches, where $\lceil x \rceil$ denotes the smallest integer larger than x, then follows from the optimality of the WSEPT rule for a single unit capacity machine.

Theorem 1 *The following procedure creates an optimal schedule:*

1. *For each family j, arbitrarily select jobs to form $\lfloor n_j/K_j \rfloor$ full batches and (possibly) one partially full batch of $n_j - \lfloor \frac{n_j}{K_j} \rfloor K_j$ jobs.*

2. *Sequence these $\sum_{j=1}^{m} \lceil \frac{n_j}{K_j} \rceil$ batches in decreasing order of $W_k \mu_j$, where W_k is the sum of the holding costs of all jobs in the batch and $1/\mu_j$ is the mean processing time for the family j which makes up the batch. Note that $W_k = c_j K_j$ for all but possibly one batch of type j. The remaining batch has $W_k = (n_j - \lfloor \frac{n_j}{K_j} \rfloor K_j) c_j$.*

Proof: The optimality of the first step follows from the fact that it satisfies Lemma 1. The optimality of the second step follows from the well-known result that sequencing jobs in decreasing order of $w_j/E[X_j]$, where X_j is the random processing time of job j and w_j is its weight, minimizes $E[\sum_j w_j C_j]$, where C_j is the completion time of job j (see, for example, Pinedo [12]). Note that $C_j = D_j + X_j$, where D_j is the time spent by job j waiting in queue to be processed, so the above sequence minimizes $E[\sum_j w_j D_j] + E[\sum_j w_j X_j]$. However, $E[\sum_j w_j X_j]$ is a constant which does not depend on the sequencing rule, so the above sequence must minimize $E[\sum_j w_j D_j]$. □

4 Optimal Results for Dynamic Problems

In the remaining sections, we consider the problem of scheduling a batch processing machine in the presence of dynamic job arrivals. This section contains problems for which the structure of the optimal policy can be determined.

We will initially consider problems with Poisson arrivals and $m = 2$, and add the restriction that the machine is not allowed to idle. Under these assumptions, the problem of minimizing long-run

average costs per unit time can be formulated as a semi-Markov decision problem. The following notation will be used to describe the system:

n_j = the number of jobs of type j waiting to be processed (the state of the system)

S_j = a random variable with mean $1/\mu_j$ and cumulative distribution function F_j (S_j represents an arbitrary service time for jobs of type j)

$X_j(t)$ = the (random) number of arrivals of type j over time t, which has a Poisson distribution with rate λ_j

$(x)^+$ = $\max\{x, 0\}$

g = the optimal long-run average cost per unit time

$V(n_1, n_2)$ = the relative cost of being in state (n_1, n_2) and following the optimal policy

The state of the system will be reviewed immediately after each service completion. These review points are the only times at which a decision needs to be made. The memoryless property of the exponential interarrival time distribution guarantees that the system satisfies the Markov property. That is, the future evolution of the system depends only on the current state and the action taken, and not on the elapsed time since the last arrival or any other information about the past behavior of the system. Since the time between consecutive decision epochs is random, the semi-Markov decision model applies. The underlying recursive equation is:

$$V(n_1,n_2)+g=\min\begin{cases} E[\int_0^{S_1} c_1[(n_1-K_1)^+ + X_1(t)] + c_2[n_2 + X_2(t)]\,dt \\ \quad + V((n_1-K_1)^+ + X_1(S_1), n_2 + X_2(S_1))] \\ E[\int_0^{S_2} c_1[n_1 + X_1(t)] + c_2[(n_2-K_2)^+ + X_2(t)]\,dt \\ \quad + V(n_1 + X_1(S_2), (n_2-K_2)^+ + X_2(S_2))] \end{cases} \tag{4.1}$$

The first term in the minimization represents the decision to serve jobs of type 1. If this decision is made, we will serve all waiting jobs of type 1 if $n_1 \leq K_1$, or serve a full batch of K_1 jobs if $n_1 > K_1$. The remaining jobs of family 1 and all jobs of family 2, as well as any jobs that arrive during the service time, incur holding costs until the next decision epoch. At this time the state of the system is $((n_1 - K_1)^+ + X_1(S_1),\ n_2 + X_2(S_1))$. Similarly, the second term in the minimization represents the decision to serve jobs of family 2.

The structural form of the optimal policy for this semi-Markov decision problem is described by

the following theorem.

Theorem 2 *The optimal policy is completely characterized by the control-limit function $l(n_1)$, such that in state (n_1, n_2), the decision is to serve type 2 if $n_2 \geq l(n_1)$ and to serve type 1 otherwise. Furthermore, $l(n_1)$ is increasing in n_1.*

Proof: The proof is given in Appendix A.

The optimal policy is further characterized by the next theorem.

Theorem 3 *Label the job families so that $c_1\mu_1 K_1 \geq c_2\mu_2 K_2$. If the machine completes service and $n_1 \geq K_1$, then an optimal schedule exists which serves a full batch of family 1.*

Proof: Consider an optimal schedule S which serves jobs of family 2 at a time t_0 when the machine is ready to be loaded and $n_1 \geq K_1$. Let t_2 be the first time after time t_0 that schedule S serves jobs of family 1, and let t_3 be the time that this batch of type 1 is completed. We will refer to this batch of type 1 as B_1. The batch that completed service immediately before B_1 must have been a batch of type 2. We will refer to this batch of type 2 as B_2. Let t_1 be the time that B_2 is started under schedule S. Construct a new schedule S' by interchanging B_1 and B_2 while keeping all else the same. This interchange decreases costs by $K_1c_1S_2$ and increases costs by at most $\min\{n_2, K_2\}c_2S_1$, where n_2 is the number of jobs of family 2 waiting at time t_1. The increase in costs may in fact be less than $\min\{n_2, K_2\}c_2S_1$ if $n_2 < K_2$. This is because under S' some jobs of type 2 that arrive during the processing of B_1 can be served in B_2 if $n_2 < K_2$. Let $C(S)$ $(C(S'))$ represent the expected total cost under schedule S (S') up until time t_3. Then $C(S) - C(S') \geq E[K_1c_1S_2 - \min\{n_2, K_2\}c_2S_1]$. The fact that $K_1c_1\mu_1 \geq K_2c_2\mu_2$ then implies that $C(S) - C(S') \geq 0$. Additionally, the state of the system at time t_3 will be at least as small under schedule S' as it will be under S (the number of type 1 jobs will be the same under both schedules while the number of type 2 jobs under S' will be less than or equal to the number of type 2 jobs under S). As a result, the expected cost under schedule S' is less than or equal to the expected cost under S both before and after time t_3, so schedule S' must also be optimal. Repeating this adjacent pairwise interchange results in a schedule with the desired property. □

We note that Theorem 3 remains valid even when idling is allowed and there are more than two families by a similar interchange argument.

We can now completely characterize the optimal policy for this two family problem without idling. If $n_1 \geq K_1$ then it is optimal to process a full batch of family 1. Otherwise, for each fixed $n_1 < K_1$, a value $l(n_1)$ can be found such that the optimal policy is to serve family 1 if $n_2 < l(n_1)$ and to serve family 2 if $n_2 \geq l(n_1)$. The control limits $l(n_1)$ are increasing in n_1. Therefore, when idling is not allowed, the optimal policy is completely specified by K_1 numbers which are increasing.

We initially conjectured that when idling is allowed, the optimal policy would be specified by two control-limit functions. In particular, we conjectured that for any fixed n_1, there would be two control-limit functions $l_1(n_1)$ and $l_2(n_1)$ such that i) for $n_2 < l_1(n_1)$, the optimal policy would be to process type 1, ii) for $n_2 > l_2(n_1)$, the optimal policy would be to process type 2, and iii) for $l_1(n_1) \leq n_2 \leq l_2(n_1)$, the optimal policy would be to idle. However, we have found that the monotonicity and submodularity conditions sufficient to satisfy this structure do not necessarily hold. Furthermore, we were able to find some counterexamples to this structure (although nearly all of the other examples we numerically solved yielded this structure). Consider a two-type problem with exponential interarrival and service times and $c_1 = 1.00, c_2 = 1.50$, $K_1 = 10, K_2 = 8$, $\lambda_1 = 2.00, \lambda_2 = 1.00$, $\mu_1 = 0.40$, and $\mu_2 = 0.50$. A numerical solution of this problem (with the arrivals truncated when the number of jobs for any class exceeds 160; we have made sure that the truncation limit is sufficiently large) results in a policy such that when $n_1 = 8$, it is optimal to idle if $n_2 \leq 2$, to serve type 1 for $n_2 = 3$ and $n_2 = 4$, to idle for $n_2 = 5$, and to serve type 2 for $n_2 > 5$. This example demonstrates that the optimal policy when idling is allowed has a complex structure even for the two-family case.

We have found that problems with more than two families are also very complex, even when idling is not allowed. We can only characterize the optimal solution for completely symmetric problems (i.e., all families have the same arrival rates, service rates, holding costs, and capacities) without idling. However, we do not need to assume Poisson arrivals.

Theorem 4 *If $c_j = c$, $\mu_j = \mu$, $\lambda_j = \lambda$, and $K_j = K$ for all $j = 1, \ldots, m$, then there exists an optimal schedule for the problem without idling which always serves the family with the greatest number of jobs waiting at each decision epoch.*

Proof: The result can be obtained by a simple interchange argument and we omit the proof in the interest of space. □

We note that even when only the arrival rates differ across families, Theorem 4 no longer holds and the structure of the optimal policy is complex. The complexity of the optimal solution and the

difficulty of its computation led us to develop a simple, implementable heuristic which we describe in the next section.

5 A Heuristic Policy

For most instances of the dynamic problem formulated in Section 2, the optimal policy can not be generated by a simple scheduling rule. Additionally, dynamic programming is not a practical solution technique for realistically sized problems since the size of the state space grows non-polynomially in the number of job families. For these reasons, we focus on developing a simple, implementable heuristic policy.

We first observe that the only times when decisions are made are either when a new job arrives and the machine is available, or when the machine completes service and there are jobs waiting. At each of these decision epochs, a heuristic policy must select one of the m job families to serve or instead decide to idle (note that if the decision is to serve family j, it will always be best to process $\min\{n_j, K_j\}$ jobs). The logic of our heuristic divides this decision into three steps. The first step uses stability considerations and an M/G/1 nonpreemptive priority queue analysis to determine which job families are eligible to be selected for service. In the second step, the heuristic selects one of the eligible job families as the best candidate for service. The third step then decides whether to serve the family selected in the previous step or to idle until the next decision epoch, at which time the decision process will begin all over again.

We next describe the development of these steps for systems with two types of jobs, and then extend the results to systems with any number of job families. Finally, we describe how the heuristic would utilize information about future arrival times if such information were available. In what follows we assume that the job families are labelled so that $c_j\mu_j K_j > c_{j+1}\mu_{j+1}K_{j+1}$ for $j = 1, \ldots, m-1$, and that arrivals of family j occur according to a Poisson process with rate λ_j. As before, we let n_j represent the number of jobs of family j currently waiting at the machine.

5.1 Heuristic for Systems with Two Job Families

For systems with two job families, the three steps of the heuristic operate as follows.

Determination of Eligible Job Families

When $n_1 \geq K_1$, it is optimal to serve family 1 regardless of the value of n_2. This follows from the extension of Theorem 3 to allow idling. When $n_1 < K_1$ and $n_2 \geq K_2$, it may still be preferable to

serve a partial load of family 1 even though a full load of family 2 is available. This is due to the fact that the larger holding cost (or faster service rate, or both) of family 1 might make it worthwhile to serve a partial load. However, we need to ensure that serving partial loads of family 1 when full loads of family 2 are available does not result in an unstable system. This is achieved by making family 1 eligible for service when $n_2 \geq K_2$ only if $n_1 \geq M$ where M is the smallest integer which satisfies

$$\frac{\lambda_1}{M\mu_1} + \frac{\lambda_2}{K_2\mu_2} < 1. \tag{5.2}$$

This condition guarantees that serving partial batches of family 1 when there is a full load of family 2 available does not result in the number of type 2 jobs growing without bound. However, M computed from (5.2) is not necessarily the "optimal" minimum batch size of family 1 to make family 1 eligible for service. Rather, it is a lower bound on the "optimal" minimum batch size which ensures stability.

To compute an approximately "optimal" minimum batch size B^* such that whenever $n_1 > B^*$, family 1 is eligible for service when full loads of family 2 are available, we use an M/G/1 nonpreemptive priority queue analysis. That is, we assume that jobs of family 2 are always served in batches of size K_2, and that jobs of family 1 are always served in batches of $B^* \geq M$. Furthermore, we assume that a batch of family 1 always has nonpreemptive priority over a batch of family 2.

The expected time for a full batch of type 2 to form is K_2/λ_2, which gives an arrival rate for type 2 batches of λ_2/K_2. Similarly, the arrival rate of type 1 batches of size B is λ_1/B. For simplicity, we approximate the distribution of the arrival process of full batches by a Poisson process. Therefore, if we let d_j denote the delay that a batch of family j experiences after a full batch (of size B for family 1 and size K_2 for family 2) has formed, using standard results for priority queues we can write

$$d_j = \frac{\lambda E[S^2]}{2(1 - \sum_{i<j} \rho_i)(1 - \sum_{i \leq j} \rho_i)}, \tag{5.3}$$

where λ is the overall arrival rate of batches, S denotes a generic "overall" service time, and ρ_i is the utilization factor for batches of family i [19]. For our two family system, the overall arrival rate is $\lambda = \lambda_2/K_2 + \lambda_1/B$ and the utilization factors are $\rho_1 = \lambda_1/B\mu_1$ and $\rho_2 = \lambda_2/K_2\mu_2$. The value $E[S^2]$ can be calculated from

$$E[S^2] = p_1 E[S_1^2] + p_2 E[S_2^2], \tag{5.4}$$

where p_j is the probability that an arrival is a batch with priority j and S_j is an arbitrary service time for batches of family j. For this system, $p_1 = \lambda_1/B\lambda$ and $p_2 = \lambda_2/K_2\lambda$.

In addition to the delay d_j incurred by batches waiting to be served, individual jobs must also wait within a batch for the complete batch to form. For batches of family 1, the first job to arrive must wait for $B-1$ other arrivals before the batch is complete. The second job must wait for $B-2$ other arrivals, and so on. This results in a total expected waiting time of

$$\sum_{i=1}^{B-1} i\left(\frac{1}{\lambda_1}\right) = \frac{B(B-1)}{2\lambda_1}$$

for each batch of size B, or an average of $(B-1)/(2\lambda_1)$ per job. Similarly, the expected waiting time for a job of family 2 within each batch of size K_2 is $(K_2-1)/(2\lambda_2)$.

We can now write the total expected cost per job as

$$\left(\frac{\lambda_1}{\lambda_1+\lambda_2}\right)\left[c_1\left(d_1+\frac{B-1}{2\lambda_1}\right)\right]+\left(\frac{\lambda_2}{\lambda_1+\lambda_2}\right)\left[c_2\left(d_2+\frac{K_2-1}{2\lambda_2}\right)\right]. \tag{5.5}$$

The best minimum batch size B^* for family 1 is then the integer value of B which minimizes (5.5) over the range $M \leq B \leq K_1$.

Therefore, when $n_1 \geq K_1$, our heuristic immediately decides to serve a batch of family 1. When $B^* \leq n_1 < K_1$ and $n_2 \geq K_2$, both family 1 and family 2 are eligible for service. When $n_1 < B^*$ and $n_2 \geq K_2$, only family 2 is eligible for service. Finally, when $n_1 < K_1$ and $n_2 < K_2$, both family 1 and family 2 are eligible for service.

Selection of Job Family for Service

Our heuristic selects one of the eligible job families as the best candidate for service based on a simple rule. It is well known that for an M/G/1 queue with multiple job types, the non-preemptive $c\mu$ rule minimizes the average holding cost per unit time [19]. This rule attaches the index $c_j\mu_j$ to each job in the jth queue, and at each decision epoch serves the available job possessing the largest index. We develop a similar rule for a batch processing machine by viewing each batch of family j as a single job with expected processing time $1/\mu_j$ and holding cost $c_j \min\{n_j, K_j\}$. Our rule then attaches the index $c_j \min\{n_j, K_j\}\mu_j$ to each eligible family j, and at each decision epoch serves the available family possessing the largest index. We will refer to this rule as the $c\min\{n, K\}\mu$ rule.

Rule for Idling

Once a family has been selected as the best candidate for service, the net benefit of idling until the next decision epoch is evaluated. If the net benefit of idling is found to be positive, the decision is to idle until the next decision epoch at which time the decision process will begin all over again. If the net benefit is not positive, the decision is to serve the selected family immediately.

If the heuristic has selected family 1, the net benefit of idling is approximated as

$$-(c_1 n_1 + c_2 n_2)\left(\frac{1}{\lambda_1+\lambda_2}\right) + \left(\frac{\lambda_1}{\lambda_1+\lambda_2}\right)\left(\frac{c_1}{\mu_1}\right) + \left(\frac{\lambda_2}{\lambda_1+\lambda_2}\right)\max\{c_2(n_2+1)(1/\mu_1) - c_1 n_1(1/\mu_2), 0\}. \quad (5.6)$$

The first term in (5.6) represents the cost of idling. By idling until the next decision epoch, we have delayed the starting time of all jobs currently in the system by an expected time of $1/(\lambda_1+\lambda_2)$. The remaining terms in (5.6) represent the benefits of delaying service until the next arrival. If the next arrival is a job of family 1 (which happens with probability $\lambda_1/(\lambda_1+\lambda_2)$), the heuristic will again select family 1 as the best candidate for service. Had the heuristic not chosen to idle, this arrival of type 1 would have incurred holding costs for a period of time *at least* as long as the remaining processing time of the current batch of family 1. We approximate this period of time by the expected service time of family 1, resulting in a savings of c_1/μ_1 due to idling. If the next arrival is a job of family 2 (with probability $\lambda_2/(\lambda_1+\lambda_2)$), then there may or may not be a benefit from idling. If the heuristic still selects family 1 for service even after this additional arrival of type 2, then idling will not produce a benefit. However, if the heuristic now selects to serve family 2 after this additional arrival, idling will produce a benefit. We approximate the benefit of the decision to idle and then serve family 2 by $c_2(n_2+1)(1/\mu_1) - c_1 n_1(1/\mu_2)$, where the first term represents the holding costs saved for the n_2+1 jobs of family 2 and the last term represents the holding costs now incurred for the n_1 jobs of family 1. Note that this benefit is positive only if an additional arrival of type 2 results in the heuristic selecting type 2 (i.e. $c_2(n_2+1)\mu_2 > c_1 n_1 \mu_1$).

If the heuristic has selected family 2 as the best candidate for service, similar reasoning gives the following expression for the net benefit of idling:

$$-(c_1 n_1 + c_2 n_2)\left(\frac{1}{\lambda_1+\lambda_2}\right) + \left(\frac{\lambda_2}{\lambda_1+\lambda_2}\right)\left(\frac{c_2}{\mu_2}\right) + \left(\frac{\lambda_1}{\lambda_1+\lambda_2}\right)\max\{c_1(n_1+1)(1/\mu_2) - c_2 n_2(1/\mu_1), 0\}. \quad (5.7)$$

We now describe in full our heuristic policy for systems with two families.

Heuristic Policy for Systems with Two Families

At each decision epoch:

1. If $n_1 \geq K_1$ then serve a full batch of family 1.

2. If $n_1 < K_1$ and $n_2 \geq K_2$ then:

(a) If $n_1 < B^*$ then serve a full batch of family 2.

(b) If $n_1 \geq B^*$ then use the $c\min\{n, K\}\mu$ rule to decide which family to serve (i.e. if $c_1\mu_1 n_1 > c_2\mu_2 K_2$ then serve family 1, otherwise serve family 2).

3. If $n_1 < K_1$ and $n_2 < K_2$ then:

(a) Use the $c\min\{n, K\}\mu$ rule to select the best candidate family for service.

(b) Use the appropriate idling expression ((5.6) or (5.7)) to determine whether to serve the selected family or to idle until the next decision epoch.

5.2 Heuristic for Systems with m Job Families

Using the ideas developed previously for two job families, we extend our heuristic to systems with any number of job families.

Determination of Eligible Job Families

If $n_1 \geq K_1$, then as before it is optimal to immediately serve a full batch of family 1 regardless of the number of jobs of any other family waiting at the machine. However, if there exists a family l ($l \neq 1$) such that $n_l \geq K_l$ and $n_j < K_j$ for all $j < l$, then the optimal policy is not clear. In this situation it will never be optimal to serve a family j with $j > l$, so our heuristic only considers serving a full batch of family l or a partial batch of families with $j < l$. To ensure that the stability of the system is maintained, a partial batch of family j ($j < l$) must be at least as large as M_{lj}, where the M_{lj} satisfy

$$\sum_{j=1}^{l-1} \frac{\lambda_j}{M_{lj}\mu_j} + \sum_{j=l}^{m} \frac{\lambda_j}{K_j\mu_j} < 1 \tag{5.8}$$

To be able to solve this inequality for the $l-1$ M_{lj}'s, we add the $l-2$ equations $M_{lj} = (c_1 M_{l1}\mu_1)/(c_j\mu_j)$ for $j = 2, \ldots, l-1$. These additional equations make the minimum batches for each family equally attractive. Note that if this results in $M_{lj} > K_j$ for some family j, then we set $M_{lj} = K_j$ for this family and resolve (5.8).

As in the heuristic for the 2-family case, (5.8) only guarantees that serving partial batches of a family when full loads of another family are available will not result in instability. To find approximately "optimal" minimum batch sizes, we once again use an M/G/1 nonpreemptive priority queue analysis. However, in this case we need to carry out the analysis $m-1$ times to cover all possible full load scenarios.

Suppose that family l is the family with the lowest index among all families that have a full load. To carry out the M/G/1 analysis, we assume that family j is served in batches of size B_{lj} for $j < l$ and in batches of size K_j for $j \geq l$. Each batch can then be viewed as a single job and (5.3) can again be used to find the expected delay d_j for batches of family j. The delay incurred by individual jobs waiting within a batch for the complete batch to form is calculated exactly as before, resulting in the following expression for the total expected cost per job in the system:

$$\sum_{j=1}^{l-1}\left(\left(\frac{\lambda_j}{\sum_{i=1}^{m}\lambda_i}\right)c_j\left(d_j+\frac{B_{lj}-1}{2\lambda_j}\right)\right)+\sum_{j=l}^{m}\left(\left(\frac{\lambda_j}{\sum_{i=1}^{m}\lambda_i}\right)c_j\left(d_j+\frac{K_j-1}{2\lambda_j}\right)\right) \tag{5.9}$$

Again using the $l-2$ equations $B_{lj} = (c_1 B_{l1}\mu_1)/(c_j\mu_j)$ for $j = 2,\ldots,l-1$, (5.9) can be written as a function of the single variable B_{l1} and minimized over the range $M_{l1} \leq B_{l1} \leq K_1$. The value B^*_{l1} that minimizes this cost function then determines the remaining B^*_{lj} (if this results in $B^*_{lj} > K_j$ for some family j, then B^*_{lj} is set equal to K_j for this family and we resolve (5.9)). The family l and all families j which satisfy $j < l$ and $n_j \geq B^*_{lj}$ are then eligible for service. We note that the B^*_{lj} values need to be computed only once and their computation is very rapid.

Selection of Job Family for Service

The selection rule developed for systems with two families extends without change to systems with any number of families. The index $c_j \min\{n_j, K_j\}\mu_j$ is attached to each eligible family j, and at each decision epoch the family possessing the largest index is selected.

Rule for Idling

The rule for idling developed for systems with two families extends with only slight modifications to systems with any number of families. Using the reasoning described previously for two families, the net benefit of idling until the next decision epoch given that the heuristic has selected family i as the best candidate for service is approximated by:

$$\left(-\sum_{j=1}^{m}c_j n_j\right)\left(\frac{1}{\sum_{j=1}^{m}\lambda_j}\right)+\left(\frac{\lambda_i}{\sum_{j=1}^{m}\lambda_j}\right)\left(\frac{c_i}{\mu_i}\right)+\sum_{j=1,j\neq i}^{m}\left(\frac{\lambda_j}{\sum_{k=1}^{m}\lambda_k}\max\{c_j(n_j+1)(1/\mu_i)-c_i n_i(1/\mu_j),0\}\right) \tag{5.10}$$

If (5.10) is positive, the decision is to idle until the next decision epoch, at which time the decision process will begin all over again. Otherwise, the decision is to serve the selected family i immediately.

We now describe in full our heuristic policy for systems with any number of job families.

Heuristic Policy for Systems with m Families

At each decision epoch:

1. If at least one full load is available:

   (a) Let l denote the smallest value of j such that $n_j \geq K_j$. The family l and all families j with $j < l$ and $n_j \geq B^*_{lj}$ are eligible for service.

   (b) Serve the eligible family with the largest value of $c_j \min\{n_j, K_j\}\mu_j$.

2. If no full loads are available:

   (a) Calculate the index $c_j \min\{n_j, K_j\}\mu_j$ for each of the m families and select the family possessing the largest index as the best candidate for service.

   (b) If the net benefit of idling as given by (5.10) is positive, idle until the next decision epoch. Otherwise, serve the selected family immediately.

5.3 Heuristic with Next Arrival Information

Manufacturers may have shop-floor control systems which provide a means to predict with reasonable accuracy the timing of future arrivals to a batch processing machine. Such prediction capability can significantly enhance the performance of a batch processing machine. Our heuristic can be easily modified to make use of such information if it exists. We will consider only information about the next arrival from each job family because the return from more information has rapidly decreasing marginal value [5], and getting more information may be difficult in practice. We note that nearly all of the literature to date on batch service queues with incompatible job families has assumed that some information on future arrivals is available.

The flow of our heuristic remains the same with next arrival information. The determination of eligible job families and the selection of the best eligible family for service are performed exactly as they were for the case with no knowledge of future arrivals. Only the idling rule changes when the timing of future arrivals can be predicted. If family i is selected as the best candidate for service, the benefit of idling until the next arrival of family j is calculated for each family $j = 1, \ldots, m$. If any one of these m benefits is positive, the decision is to idle until the next decision epoch.

The net benefit of idling until the next arrival of each family given that family i is the best candidate for service is found as follows. Let T_j represent the time until the next arrival of family

j. First, if $c_i(n_i+1)\mu_i \geq c_j(n_j+1)\mu_j$ for all j such that $T_j \leq T_i$, the net benefit of idling until the next arrival of family i is approximated by

$$-\left(\sum_{l=1}^{m} c_l n_l\right) T_i - \sum_{l=1}^{m} c_l \max\{T_i - T_l, 0\} + (c_i/\mu_i). \tag{5.11}$$

We note that if $c_i(n_i+1)\mu_i < c_j(n_j+1)\mu_j$ for some j with $T_j \leq T_i$, there is no benefit to idling until the next arrival of family i since the heuristic will no longer select to serve family i at that decision epoch. Next, for each family j with $T_j < T_i$, the net benefit of idling until the next arrival is approximated by

$$-\left(\sum_{l=1}^{m} c_l n_l\right) T_j - \sum_{l=1}^{m} c_l \max\{T_j - T_l, 0\} + c_j(n_j+1)(1/\mu_i) - c_i n_i (1/\mu_j). \tag{5.12}$$

Finally, for each family j with $T_j \geq T_i$, the net benefit of idling until the next arrival is approximated by

$$-\left(\sum_{l=1}^{m} c_l n_l\right) T_j - \sum_{l=1}^{m} c_l \max\{T_j - T_l, 0\} + c_j(n_j+1)(1/\mu_i) - c_i(n_i+1)(1/\mu_j). \tag{5.13}$$

The reasoning behind these expressions is identical to that discussed for the idling rule without knowledge of future arrivals, with the addition of a term to account for the possible holding costs incurred for the next arrival of each family.

In summary, when there is knowledge of future arrivals, the heuristic operates exactly as described in Section 5.2 with only a slight change in the rule for idling. When no full loads are available, the idling decision is made by checking to see if any of equations (5.11), (5.12), or (5.13) are positive. If any of them are positive, then the decision is to idle until the next decision epoch (i.e., the next arrival from any family). Otherwise, we choose to serve the best candidate found in step 2a of the algorithm.

Finally, we note that our heuristic with next arrival information can easily be applied even when interarrival times do not follow the exponential distribution. The only place where our heuristic requires the exponential interarrival time distribution is in the calculation of the B^*_{lj} values using M/G/1 nonpreemptive priority queue results. If interarrival times are non-exponential, the B^*_{lj} values can be computed using simulation of G/G/1 priority queues. However, we have found that computing these values using M/G/1 queue results works very well even when the interarrival times are not exponential. In the next section, we give examples with uniform interarrival times where we computed the B^*_{lj} values assuming exponential interarrival time distributions.

6 Computational Results

The real test of any heuristic is its performance with respect to the optimal solution. For the problem considered here, however, the optimal solution can only be computed without great difficulty for examples with exponential interarrival times and only two or three families. Increasing the number of families to four or five leads to very large state space sizes. For example, even truncating the state space such that the content of each queue is allowed to vary between 0 and 100 still leads to a state space size of over 10^8 in problems with four families. There also does not seem to be an easy way to compute the optimal solution for problems where future arrival information is available.

Given the difficulty of computing the optimal solution for problems with more than three families, we chose to compare our heuristic to the optimal solution for problems with two and three families and exponential interarrival and processing times under the assumption that future arrival information is not available. For these same problems, we also compared our heuristic to a generalization of the heuristic by Fowler et al. [4] under the assumption that the arrival time for the next job from each family is exactly known. The original multiple product heuristic of Fowler et al. does not allow for different holding costs or random processing times, so we modified their heuristic, NACHM, to meet the assumptions of this paper (in the discussion below, we refer to this modified heuristic as MNACHM). We summarize their heuristic and our modification of it in Appendix B.

We first tested our heuristic on a variety of problems with two job families. The data for the 31 different examples with two families is displayed in Table 1. As Table 1 indicates, the examples represent a wide variety of different situations. The examples include moderate through high traffic intensities, and cover most combinations of equal and different holding costs, capacities, arrival rates, and service rates. As required by the heuristic, all families are labelled so that $c_j K_j \mu_j \geq c_{j+1} K_{j+1} \mu_{j+1}$. In our examples, the ratio of the holding costs for the two families, C_1/C_2, ranges from 1/3 to 10. The ratio of the arrival rates, λ_1/λ_2, ranges from 3/16 to 2.5, and the ratio of the service rates, μ_1/μ_2, ranges from 0.75 to 3.00. We also consider different ratios of K_1/K_2, hence our examples represent a wide variety of situations that might arise in practice.

The two family examples of Table 1 were used to test the performance of the heuristic assuming 1) exponential interarrival and processing times and 2) uniform interarrival times and deterministic processing times. With uniform interarrival times, we used the same mean $1/\lambda_i$ as we did for exponential interarrival times. The range of the uniform interarrival times was from $\frac{1}{2\lambda_i}$ to $\frac{3}{2\lambda_i}$. When processing times were deterministic, they were assumed to be exactly equal to $1/\mu_i$.

The optimal long-run average cost per unit time under the assumption of exponential interarrival and processing times was found by formulating the problem with idling as a Markov Decision Process. To avoid the difficulties of an infinite state space, an upper limit was placed on the number of jobs of each family allowed in queue. Therefore, the dynamic programming solution gave a lower bound on the actual optimal cost. We then compared this result to the performance of our heuristic under the assumption of no knowledge of future arrivals. For the same data with exponential assumptions, we also compared the performance of our heuristic with next arrival information and MNACHM. Finally, for the case with uniform interarrival and deterministic processing times, we again compared the performance of our heuristic with next arrival information with MNACHM. As described in the previous section, we used the M/G/1 analysis to obtain the B^* values for our heuristic.

Table 2 summarizes the results we obtained. It contains the lower bound on the optimal average cost per unit time obtained by solving the MDP for the problem without next arrival information (LOPT), the results for our heuristic without next arrival information (H) and with next arrival information (HA), and the results for MNACHM. To obtain the simulation results, we used a GPSS/H simulation program. The simulations were run for 264,000 time units, with a warmup period of 8000 time units and the average cost calculated every 4000 time units. We report the average cost as well as 95 % confidence intervals for each example.

The results in Table 2 clearly show that our heuristic works very well with or without next arrival information. When interarrival and processing times were exponential, our heuristic H gave results that were very close to the lower bound on the optimal cost computed by solving the MDP. On average, the cost obtained by our heuristic H was less than 2 % above the lower bound on the optimal solution. When future arrival information was available, our heuristic HA outperformed MNACHM for every example. In fact, on average MNACHM resulted in costs that were 7.5 % higher than those obtained by HA. Interestingly, in many examples our heuristic H, which uses no information on future arrivals, was able to outperform MNACHM, which assumes such information. However, on average MNACHM was better than H by nearly 1.7 %.

Our heuristic HA also outperformed MNACHM in all but one example when arrivals were uniform and services were deterministic. However, under these assumptions the average difference between the two heuristics was much less (3.3 %). This is intuitive as there is much less variability in this case, and therefore sequencing is less likely to have as significant an effect on performance.

We also tested our heuristic on examples with three job families. In this case, all our examples

Example	c_1	c_2	K_1	K_2	λ_1	λ_2	μ_1	μ_2
1	1.0	1.0	10	10	1.0	1.0	0.5	0.5
2	1.0	1.0	10	10	1.5	1.5	0.5	0.5
3	1.0	1.0	10	10	1.0	1.0	0.6	0.2
4	1.0	1.0	10	10	2.5	1.0	0.5	0.5
5	1.0	1.0	12	3	1.0	1.0	0.5	0.5
6	2.0	1.0	10	10	1.0	1.0	0.5	0.5
7	2.0	1.0	10	10	2.0	2.0	0.5	0.5
8	2.0	1.0	10	10	2.0	1.0	0.5	0.5
9	2.0	1.0	10	10	1.0	2.0	0.5	0.5
10	2.0	1.0	5	10	1.5	1.5	0.6	0.6
11	2.0	1.0	10	4	0.5	1.5	0.9	0.6
12	1.1	1.0	8	7	2.0	1.0	0.7	0.4
13	1.1	1.0	10	9	0.7	0.8	0.4	0.4
14	1.2	1.0	10	6	1.0	2.0	0.6	0.5
15	1.3	1.0	10	8	0.8	0.7	0.3	0.4
16	3.0	1.0	10	3	1.5	1.4	1.3	0.7
17	3.0	1.0	8	7	1.0	2.0	0.6	0.5
18	5.0	1.0	10	4	1.0	2.0	0.8	0.8
19	10.0	1.0	8	8	0.5	2.0	0.6	0.5
20	1.5	1.0	8	8	1.2	1.3	0.4	0.5
21	1.5	1.0	6	10	1.0	2.0	0.8	0.6
22	1.7	1.0	6	8	1.4	1.2	0.5	0.6
23	2.0	1.0	5	10	2.0	1.0	0.8	0.6
24	1.0	1.5	6	9	1.0	1.5	1.0	0.3
25	1.0	2.0	10	7	0.9	0.9	0.7	0.3
26	1.0	2.0	10	8	1.3	1.6	0.7	0.4
27	1.0	3.0	10	5	1.2	1.1	0.8	0.4
28	2.0	2.0	10	5	0.8	1.3	0.8	0.6
29	2.0	1.0	7	9	0.5	0.7	0.4	0.5
30	5.0	1.0	10	4	0.3	1.6	0.5	0.6
31	10.0	1.0	10	3	0.5	2.0	1.2	1.0

Table 1: Input Data for Two Family Examples

	Poisson arrivals, exponential services				uniform arrivals, det. services	
Ex.	LOPT	H	HA	MNACHM	HA	MNACHM
1	5.72	5.73±0.08	5.05±0.08	5.21±0.08	3.54±0.01	3.66±0.01
2	10.87	11.01±0.25	10.18±0.24	10.25±0.20	5.54±0.01	5.73±0.01
3	13.67	14.09±0.52	13.38±0.59	13.73±0.61	6.24±0.01	6.43±0.01
4	15.29	15.70±0.40	14.76±0.41	15.11±0.39	6.73±0.01	6.84±0.01
5	14.00	14.00±0.53	13.63±0.60	15.06±0.95	5.02±0.01	5.01±0.01
6	8.31	8.32±0.11	7.33±0.09	7.69±0.12	5.26±0.01	5.48±0.01
7	31.94	32.74±1.10	31.93±1.21	34.65±1.47	11.94±0.02	12.47±0.02
8	16.28	16.73±0.32	15.31±0.29	15.67±0.27	9.33±0.01	9.36±0.01
9	14.30	14.41±0.20	13.34±0.20	13.81±0.27	7.32±0.01	7.71±0.01
10	19.40	19.59±0.50	18.65±0.60	19.06±0.58	6.97±0.01	7.29±0.01
11	8.36	8.55±0.22	7.84±0.23	8.76±0.23	3.32±0.01	3.47±0.01
12	12.70	12.92±0.27	12.17±0.32	12.58±0.33	5.90±0.01	6.01±0.01
13	5.57	5.57±0.07	4.90±0.08	5.02±0.07	3.44±0.01	3.55±0.01
14	20.48	21.45±1.10	20.56±0.96	23.27±1.52	6.86±0.01	6.89±0.01
15	7.85	7.97±0.16	7.07±0.12	7.24±0.12	4.54±0.01	4.69±0.01
16	16.18	17.29±0.43	16.55±0.40	18.42±0.43	6.95±0.02	7.05±0.02
17	23.21	23.74±0.62	22.62±0.69	24.01±0.63	10.06±0.02	10.10±0.02
18	21.24	22.50±0.43	21.06±0.47	23.43±0.41	10.48±0.02	10.69±0.02
19	21.69	22.30±0.35	20.23±0.36	23.93±0.42	11.41±0.03	12.61±0.04
20	15.29	15.29±0.28	14.26±0.33	14.96±0.39	6.40±0.01	6.71±0.01
21	8.40	8.40±0.14	7.60±0.12	7.82±0.11	4.56±0.01	4.69±0.01
22	15.42	15.92±0.42	14.30±0.36	14.80±0.46	5.98±0.01	6.26±0.01
23	13.57	13.95±0.29	12.59±0.29	13.21±0.29	6.68±0.01	6.71±0.01
24	18.35	18.60±0.55	17.74±0.70	18.21±0.64	6.57±0.01	6.74±0.01
25	11.53	11.62±0.25	10.37±0.20	10.75±0.25	5.77±0.01	5.93±0.01
26	19.92	19.98±0.53	19.04±0.56	19.15±0.56	8.17±0.01	8.55±0.02
27	20.53	20.67±0.77	18.57±0.55	19.96±0.65	7.50±0.01	7.95±0.01
28	10.36	10.54±0.17	9.17±0.16	9.63±0.18	5.22±0.01	5.40±0.01
29	5.03	5.03±0.06	4.19±0.06	4.40±0.06	3.09±0.01	3.22±0.01
30	15.42	15.83±0.46	14.77±0.48	16.85±0.44	6.92±0.02	6.97±0.02
31	14.77	15.10±0.28	14.63±0.32	22.93±0.53	6.58±0.02	7.03±0.03

Table 2: Results for Two Family Examples

Example	c_1	c_2	c_3	K_1	K_2	K_3	λ_1	λ_2	λ_3	μ_1	μ_2	μ_3
1	1.0	1.0	1.0	5	5	5	0.7	0.7	0.7	0.7	0.7	0.7
2	1.0	1.0	1.0	5	5	5	0.7	0.7	0.7	1.0	0.7	0.4
3	1.0	1.0	1.0	5	5	5	1.1	0.7	0.3	1.0	0.7	0.4
4	1.0	1.0	1.0	5	5	5	0.3	0.7	1.1	1.0	0.7	0.4
5	2.0	1.5	1.0	5	5	5	0.7	0.7	0.7	1.0	0.7	0.4
6	2.0	1.5	1.0	5	5	5	1.1	0.7	0.3	1.0	0.7	0.4
7	3.0	2.0	1.0	5	5	5	0.7	0.7	0.7	0.7	0.7	0.7
8	3.0	2.0	1.0	5	5	5	0.3	0.7	1.3	0.7	0.7	0.7
9	3.0	2.0	1.0	5	5	5	0.3	0.7	1.3	0.4	0.6	1.0
10	3.0	2.0	1.0	7	5	3	1.0	0.7	0.4	0.7	0.7	0.7
11	3.0	2.0	1.0	5	5	3	0.3	0.7	1.1	0.7	0.7	0.7
12	3.0	2.0	1.0	3	5	7	0.3	0.7	1.3	0.7	0.5	0.7
13	3.0	2.0	1.0	3	5	7	1.0	0.8	0.6	1.0	0.7	0.7
14	1.2	1.1	1.0	5	5	5	0.7	0.7	0.7	0.7	0.7	0.7
15	1.2	1.1	1.0	5	5	5	1.1	0.7	0.3	1.0	0.7	0.4
16	1.0	1.5	2.0	5	5	5	0.7	0.7	0.7	1.2	0.7	0.4
17	1.0	1.5	2.0	5	5	5	0.3	0.7	1.3	1.5	1.0	0.5
18	1.0	2.0	3.0	7	5	3	0.4	0.7	0.4	1.0	0.7	0.4
19	1.0	1.5	2.0	4	5	6	0.7	0.7	0.7	2.0	1.0	0.5
20	1.0	1.2	1.4	7	5	3	1.5	1.0	0.5	1.0	1.0	1.0
21	1.0	1.5	1.0	7	5	3	0.7	0.7	0.7	1.0	0.7	1.0
22	1.0	2.0	1.0	7	5	3	0.5	0.6	0.5	1.2	0.8	0.6
23	1.2	1.0	1.2	5	7	5	0.5	0.8	0.5	1.3	0.8	0.6
24	1.0	2.0	2.0	6	4	4	0.3	0.5	0.7	1.0	0.5	0.5
25	2.0	1.5	1.0	5	5	5	0.6	0.5	0.4	0.8	0.7	0.6
26	2.0	1.5	1.0	5	5	5	0.4	0.5	0.6	1.0	1.0	1.0
27	5.0	2.0	1.0	5	6	2	0.7	0.7	0.7	0.8	0.8	0.8
28	10.0	5.0	1.0	7	5	3	0.5	0.7	0.9	1.2	1.0	0.8
29	10.0	1.0	1.0	7	3	3	0.6	1.0	1.0	2.0	1.0	1.0
30	1.0	1.0	1.0	4	5	6	0.5	0.8	0.9	1.2	0.9	0.6
31	1.2	1.1	1.0	5	5	5	0.6	0.5	0.4	1.0	0.9	0.8
32	3.0	2.0	1.0	5	5	4	0.5	0.7	0.6	1.1	1.2	1.1
33	1.0	1.2	1.0	5	5	5	0.6	0.4	0.6	1.0	0.7	0.7
34	1.5	1.0	1.2	5	6	4	1.0	0.6	0.8	0.7	0.8	0.7
35	1.0	1.4	1.2	6	4	5	0.7	0.7	0.7	1.4	1.2	1.0
36	2.0	1.5	1.0	7	2	3	0.5	0.6	0.5	0.8	1.0	0.8
37	1.0	1.0	1.0	6	5	4	0.2	0.6	1.0	1.5	1.0	0.5

Table 3: Input Data for Three Family Examples

Example	LOPT	H	HA	MNACHM
1	6.39	6.41±0.09	5.75±0.10	5.85±0.10
2	8.55	8.55±0.19	8.04±0.19	8.46±0.20
3	6.07	6.07±0.08	5.50±0.08	5.82±0.09
4	13.56	13.76±0.56	13.26±0.62	14.52±0.67
5	11.66	11.80±0.22	11.00±0.25	11.71±0.23
6	9.23	9.40±0.12	8.53±0.13	9.05±0.13
7	11.82	11.82±0.16	10.66±0.16	11.14±0.17
8	11.73	11.81±0.19	10.88±0.17	11.39±0.21
9	13.67	13.77±0.30	12.74±0.29	13.80±0.32
10	12.43	12.65±0.16	11.52±0.16	12.02±0.17
11	16.26	16.52±0.41	15.85±0.42	17.45±0.68
12	14.22	14.26±0.26	13.36±0.32	13.58±0.29
13	14.80	14.80±0.22	13.39±0.26	14.02±0.28
14	6.95	6.97±0.11	6.23±0.11	6.37±0.09
15	6.72	6.72±0.11	6.09±0.10	6.48±0.09
16	12.71	12.81±0.26	11.67±0.29	12.90±0.27
17	14.40	14.52±0.33	13.43±0.29	14.56±0.38
18	10.70	10.80±0.20	9.49±0.20	10.34±0.21
19	7.00	7.01±0.09	6.09±0.09	8.22±0.08
20	6.74	6.74±0.08	6.07±0.09	6.25±0.09
21	5.44	5.57±0.07	4.85±0.07	5.02±0.06
22	4.76	4.79±0.06	4.19±0.05	4.69±0.07
23	4.02	4.07±0.05	3.48±0.04	3.83±0.05
24	11.11	11.28±0.25	10.26±0.25	11.45±0.25
25	5.35	5.35±0.06	4.59±0.06	4.82±0.06
26	2.95	2.97±0.03	2.43±0.03	2.68±0.03
27	17.10	17.37±0.28	16.43±0.25	17.57±0.23
28	16.78	17.73±0.21	16.21±0.20	20.04±0.26
29	13.18	13.41±0.25	13.27±0.32	20.95±0.29
30	5.19	5.19±0.06	4.60±0.06	4.88±0.07
31	2.70	2.70±0.03	2.24±0.02	2.41±0.02
32	4.32	4.32±0.04	3.63±0.04	3.93±0.04
33	3.52	3.52±0.04	2.96±0.04	3.15±0.04
34	10.33	10.33±0.19	9.50±0.18	9.92±0.18
35	3.32	3.33±0.03	2.76±0.03	2.95±0.03
36	5.88	5.94±0.09	5.32±0.08	5.61±0.09
37	5.77	5.77±0.11	5.32±0.11	5.87±0.11

Table 4: Results for Three Family Examples

Traffic Intensity	H	NACHM	MCR	HA
0.1	0.1856 ± 0.0043	0.1545	0.1505	0.1605 ± 0.0033
0.2	0.7917 ± 0.0115	0.6503	0.6487	0.6567 ± 0.0090
0.3	1.7507 ± 0.0148	1.4584	1.4388	1.4075 ± 0.0130
0.4	2.6801 ± 0.0200	2.4449	2.3396	2.2961 ± 0.0186
0.5	3.6678 ± 0.0223	3.5516	3.3198	3.2868 ± 0.0205
0.6	4.7686 ± 0.0301	4.8138	4.4175	4.3953 ± 0.0323
0.7	6.0070 ± 0.0336	6.3066	5.7459	5.6669 ± 0.0350
0.8	7.6919 ± 0.0623	8.1426	7.5121	7.3936 ± 0.0621
0.9	10.8808 ± 0.1442	11.6875	11.0549	10.6759 ± 0.1522

Table 5: Average Total Queue Length for Four Family Examples

had exponential interarrival and processing time distributions. Once again, we considered examples that represent many different situations. The input data for our examples is given in Table 3 and our results are reported in Table 4. In the case with no future arrival information, our heuristic H resulted in costs that were very close to the lower bound on the optimal cost. On average, the difference between H and LOPT was less than 1 %. In the case with known next arrival times, our heuristic HA once again outperformed MNACHM. On average, MNACHM resulted in costs which were more than 9 % higher than those of HA.

Finally, we tested our heuristic on 4-family examples presented in Weng and Leachman [18]. In these examples, families 1, 2, 3, and 4 have deterministic processing times equal to 60, 120, 180, and 240, respectively. The capacity of the machine is equal to five for all job families. The arrival distribution is assumed to be Poisson and the arrival rates for all four families are the same. With traffic intensity ρ defined as $\rho = \sum_{j=1}^{4} \frac{\lambda_j}{K_j \mu_j}$, Weng and Leachman report the average total queue lengths that NACHM, as well as their heuristic MCR, obtained for ρ values between 0.1 and 0.9. We refer the reader to [18] for a detailed description of the MCR heuristic. However, we note that at any decision epoch, MCR requires knowledge of the arrival times of the next $K_j - n_j$ jobs from each family j. For instance, in the example we tested, when there is one job from each family in the system, MCR requires knowledge of 16 arrival times before making a decision.

We report the results obtained by our heuristics H and HA in Table 5, along with the results obtained by Weng and Leachman for MCR and NACHM. Except in extremely low traffic intensities ($\rho = 0.1, 0.2$), HA outperformed both MCR and NACHM. The difference between HA and NACHM was very significant, with HA outperforming NACHM by nearly 10 % for traffic intensities above 0.5. The difference between HA and MCR was very small. However, we note the fact that HA achieves

the same or better performance than MCR with potentially much less information required about future arrivals. Interestingly, even H, which uses no information on future arrivals, outperforms NACHM when the traffic intensity is 0.6 or above and outperforms MCR when the traffic intensity equals 0.9. However, for lower utilizations H does not perform well. This is intuitive as the value of the information about the exact arrival time of a job is greater when the traffic intensity is low (i.e., arrivals are rare).

7 Conclusions and Further Research

In this paper, we obtained structural results and an effective heuristic for the control of a batch processing machine with multiple incompatible job families. Our heuristic performed very well both with and without future arrival information. Its simplicity and good performance make it a good candidate for implementation in environments where effective control of batch processing machines is important, such as wafer fabrication.

Further research should focus on the control of a stochastic batch processing machine where jobs from different families can be mixed together in the same batch. In these systems, the processing time of a batch is determined by the job in the batch with the longest processing time. Such systems have been modelled under deterministic assumptions (see [2] and [9]). However, to our knowledge, the stochastic control of these systems has not been addressed. Further research is also needed for the problem of batch machines in parallel, or in a network. Problems where a setup is required before switching from serving one family to another also need to be considered.

Appendix A: Proof of Theorem 2

Using the data transformation suggested in Tijms [14], the semi-Markov decision model (4.1) can be converted into a discrete-time Markov decision model such that for each stationary policy the average costs per unit time are the same in both models. This enables us to use a value iteration algorithm to solve the original semi-Markov decision model. Labelling the job families so that $E[S_1] \leq E[S_2]$, we get the following recursive equation for the value iteration algorithm:

$$V_i(n_1,n_2) = \min \begin{cases} \mu_1 E[\int_0^{S_1} c_1[(n_1-K_1)^+ + X_1(t)] + c_2[n_2 + X_2(t)]\,dt] \\ \quad + E[V_{i-1}((n_1-K_1)^+ + X_1(S_1), n_2 + X_2(S_1))] \\ \mu_2 E[\int_0^{S_2} c_1[n_1 + X_1(t)] + c_2[(n_2-K_2)^+ + X_2(t)]\,dt] \\ \quad + \frac{\mu_2}{\mu_1} E[V_{i-1}(n_1 + X_1(S_2), (n_2-K_2)^+ + X_2(S_2))] \\ \quad + (1-\frac{\mu_2}{\mu_1}) V_{i-1}(n_1,n_2) \end{cases} \tag{A.1}$$

With $U_i(n_1,n_2,1)$ and $U_i(n_1,n_2,2)$ defined by

$$\begin{aligned} U_i(n_1,n_2,1) &= \mu_1 E[\int_0^{S_1} c_1[(n_1-K_1)^+ + X_1(t)] + c_2[n_2 + X_2(t)]\,dt] \\ &\quad + E[V_{i-1}((n_1-K_1)^+ + X_1(S_1), n_2 + X_2(S_1))] \qquad \text{(A.2)} \\ U_i(n_1,n_2,2) &= \mu_2 E[\int_0^{S_2} c_1[n_1 + X_1(t)] + c_2[(n_2-K_2)^+ + X_2(t)]\,dt] \\ &\quad + \frac{\mu_2}{\mu_1} E[V_{i-1}(n_1 + X_1(S_2), (n_2-K_2)^+ + X_2(S_2))] \qquad \text{(A.3)} \\ &\quad + (1-\frac{\mu_2}{\mu_1}) V_{i-1}(n_1,n_2) \end{aligned}$$

the recursion (A.1) can be written concisely as

$$V_i(n_1,n_2) = \min \begin{cases} U_i(n_1,n_2,1) \\ U_i(n_1,n_2,2) \end{cases} \tag{A.4}$$

It follows from (A.4) that it will be optimal to serve family 2 at stage i if $U_i(n_1,n_2,1) - U_i(n_1,n_2,2) \geq 0$. To show the existence of a control-limit function, $l(n_1)$, it suffices to show that for each stage i, $U_i(n_1,n_2,1) - U_i(n_1,n_2,2)$ is increasing in n_2 for fixed n_1. Similarly, to show that $l(n_1)$ is increasing in n_1, it is sufficient to show that $U_i(n_1,n_2,2) - U_i(n_1,n_2,1)$ is increasing in n_1 for fixed n_2 for each i. We will prove these by induction. To start the induction, let $V_0(n_1,n_2) = 0$ for all n_1 and n_2. We need the following lemma for the proof of Theorem 2.

Lemma 2 *The following conditions on U_i*

1a) $U_i(n_1,n_2,2) - U_i(n_1,n_2,1)$ *is increasing in* n_1 *for fixed* n_2,

1b) $E[\frac{\mu_2}{\mu_1} U_i(n_1 + X_1(S_2), (n_2-K_2)^+ + X_2(S_2), 1)] + (1-\mu_2/\mu_1) U_i(n_1,n_2,1)$
$- E[U_i((n_1-K_1)^+ + X_1(S_1), n_2 + X_2(S_1), 2)]$ *is increasing in* n_1 *for fixed* n_2,

1c) $U_i(n_1,n_2,1)$ *and* $U_i(n_1,n_2,2)$ *are increasing in* n_1 *for fixed* n_2,

1d) $U_i(n_1,n_2,1)-U(n_1,n_2-1,1)$ *and* $U_i(n_1,n_2,2)-U_i(n_1,n_2-1,2)$ *are increasing in* n_1 *for fixed* n_2,

1e) $U_i(n_1,n_2,1)-U_i(n_1,n_2,2)$ *is increasing in* n_2 *for fixed* n_1,

1f) $E[U_i((n_1-K_1)^+ + X_1(S_1), n_2+X_2(S_1),2)] - \frac{\mu_2}{\mu_1}E[U_i(n_1+X_1(S_2),(n_2-K_2)^+ + X_2(S_2),1)]$
$-(1-\mu_2/\mu_1)U_i(n_1,n_2,1) + c_2(n_2-(n_2-K_2)^+)$ *is increasing in* n_2 *for fixed* n_1,

1g) $U_i(n_1,n_2,1)$ *and* $U_i(n_1,n_2,2)$ *are increasing in* n_2 *for fixed* n_1,

1h) $U_i(n_1,n_2,1)-U_i(n_1-1,n_2,1)$ *and* $U_i(n_1,n_2,2)-U_i(n_1-1,n_2,2)$ *are increasing in* n_2 *for fixed* n_1,

hold if the following conditions hold on V_{i-1}

2a) $E[\frac{\mu_2}{\mu_1}V_{i-1}(n_1+X_1(S_2),(n_2-K_2)^+ + X_2(S_2))] + (1-\mu_2/\mu_1)V_{i-1}(n_1,n_2)$
$-E[V_{i-1}((n_1-K_1)^+ + X_1(S_1), n_2+X_2(S_1)]$ *is increasing in* n_1 *for fixed* n_2.

2b) $V_{i-1}(n_1,n_2)$ *is increasing in* n_1 *for fixed* n_2,

2c) $V_{i-1}(n_1,n_2)-V_{i-1}(n_1,n_2-1)$ *is increasing in* n_1 *for fixed* n_2.

2d) $E[V_{i-1}((n_1-K_1)^+ + X_1(S_1), n_2+X_2(S_1)] + c_2(n_2-(n_2-K_2)^+)$
$-\frac{\mu_2}{\mu_1}E[V_{i-1}(n_1+X_1(S_2),(n_2-K_2)^+ + X_2(S_2))] - (1-\mu_2/\mu_1)V_{i-1}(n_1,n_2)$ *is increasing in* n_2 *for fixed* n_1.

2e) $V_{i-1}(n_1,n_2)$ *is increasing in* n_2 *for fixed* n_1,

2f) $V_{i-1}(n_1,n_2)-V_{i-1}(n_1-1,n_2)$ *is increasing in* n_2 *for fixed* n_1.

Furthermore, conditions (2a-2f) hold for V_i *if conditions (1a-1h) hold for* U_i.

Proof: The actual proof is *very* lengthy, so in the interest of space we give the proofs for a few of the cases. We note that whereas conditions of (1a-1d, 2a-2c) state the monotonicity of the relative value functions with respect to the number of jobs of type 1, conditions (1e-1h, 2d-2f) state the completely symmetric monotonicity conditions with respect to the number of jobs of type 2. Hence, their proofs are almost identical.

We first show that condition (1a) holds for U_i if conditions (2a-2c) hold for V_{i-1}. To see this, note that using (A.2) and (A.3), $U_i(n_1, n_2, 2) - U_i(n_1, n_2, 1)$ can be written as:

$$c_1[n_1 - (n_1 - K_1)^+] + \mu_2 E[\int_0^{S_2}(c_1 X_1(t) + c_2[(n_2 - K_2)^+ + X_2(t)])dt]$$
$$-\mu_1 E[\int_0^{S_1}(c_1 X_1(t) + c_2[n_2 + X_2(t)])dt] + E[(\mu_2/\mu_1)V_{i-1}(n_1 + X_1(S_2), (n_2 - K_2)^+ + X_2(S_2))]$$
$$+(1 - \mu_2/\mu_1)V_{i-1}(n_1, n_2) - E[V_{i-1}((n_1 - K_1)^+ + X_1(S_1), n_2 + X_2(S_1))]] \tag{A.5}$$

The first term in (A.5) is increasing in n_1. The second and third terms do not depend on n_1, and hence are also increasing with respect to n_1. The sum of the fourth, fifth and sixth terms is increasing because of condition (2a).

Next, we show that condition (2a) holds for V_i if conditions (1a-1d) hold for U_i. To see this note that using (A.4), after some simplification we get:

$$E[\tfrac{\mu_2}{\mu_1}V_i(n_1 + X_1(S_2), (n_2 - K_2)^+ + X_2(S_2))] + (1 - \tfrac{\mu_2}{\mu_1})V_i(n_1, n_2)$$
$$- E[V_i((n_1 - K_1)^+ + X_1(S_1), n_2 + X_2(S_1)] =$$
$$E[\tfrac{\mu_2}{\mu_1}U_i(n_1 + X_1(S_2), (n_2 - K_2)^+ + X_2(S_2), 1)]$$
$$+ E[\tfrac{\mu_2}{\mu_1}\min\{0, U_i(n_1 + X_1(S_2), (n_2 - K_2)^+ + X_2(S_2), 2) - U_i(n_1 + X_1(S_2), (n_2 - K_2)^+ + X_2(S_2), 1)\}]$$
$$+ (1 - \tfrac{\mu_2}{\mu_1})U_i(n_1, n_2, 1) + (1 - \tfrac{\mu_2}{\mu_1})\min\{0, U_i(n_1, n_2, 2) - U(n_1, n_2, 1)\}$$
$$- E[U_i((n_1 - K_1)^+ + X_1(S_1), n_2 + X_2(S_1), 2)]$$
$$+ E[\min\{0, U_i((n_1 - K_1)^+ + X_1(S_1), n_2 + X_2(S_1), 2) - U_i((n_1 - K_1)^+ + X_1(S_1), n_2 + X_2(S_1), 1)\}] \tag{A.6}$$

We note that the second, fourth and sixth terms on the right hand side of (A.6) are increasing in n_1 by condition (1a), and the sum of the first, third and fifth terms is increasing in n_1 by condition (1b).

We now show that condition (1b) holds for U_i if conditions (2a-2c) hold for V_{i-1}. Using (A.2) and (A.3), after a lot of algebra and simplification we get:

$$
\begin{aligned}
&E[\tfrac{\mu_2}{\mu_1}U_i(n_1+X_1(S_2),(n_2-K_2)^+ + X_2(S_2),1)] + (1-\tfrac{\mu_2}{\mu_1})U_i(n_1,n_2,1) \\
&- E[U_i((n_1-K_1)^+ + X_1(S_1), n_2 + X_2(S_1), 2)] = \\
&\quad E[\tfrac{\mu_2}{\mu_1}c_1[(n_1+X_1(S_2)-K_1)^+ - (n_1-K_1)^+]] + E[(\mu_1-\mu_2)\textstyle\int_0^{S_1}(c_1X_1(t)+c_2(n_2+X_2(t))dt] \\
&\quad + E[E[\mu_1\textstyle\int_0^{S_1}(c_1X_1(t)+c_2[(n_2-K_2)^+ + X_2(S_2)+X_2(t)])dt]] \\
&\quad - E[E[\mu_2\textstyle\int_0^{S_2}(c_1[X_1(S_1)+X_1(t)]+c_2[(n_2+X_2(S_1)-K_2)^+ + X_2(t)])dt]] \\
&\quad + \tfrac{\mu_2}{\mu_1}(E[E[V_{i-1}((n_1+X_1(S_2)-K_1)^+ + X_1(S_1),(n_2-K_2)^+ + X_2(S_2)+X_2(S_1)) \\
&\quad - V_{i-1}((n_1-K_1)^+ + X_1(S_1)+X_1(S_2),(n_2+X_2(S_1)-K_2)^+ + X_2(S_2))]])
\end{aligned}
\tag{A.7}
$$

The first term on the right hand side of (A.7) is increasing in n_1, while the second, third and fourth terms do not depend on n_1. Hence, it is sufficient to show that the fifth term is also increasing in n_1. We will show this by establishing that the term

$$
\begin{aligned}
&V_{i-1}((n_1+X_1(S_2)-K_1)^+ + X_1(S_1),(n_2-K_2)^+ + X_2(S_2)+X_2(S_1)) \\
&-V_{i-1}((n_1-K_1)^+ + X_1(S_1)+X_1(S_2),(n_2+X_2(S_1)-K_2)^+ + X_2(S_2))
\end{aligned}
\tag{A.8}
$$

is increasing in n_1 for each realization of $X_1(S_1)$, $X_1(S_2)$, $X_2(S_1)$ and $X_2(S_2)$. There are 9 possible cases that need to be checked:

1.) $n_1 \geq K_1, n_2 \geq K_2$: In this case, (A.8) equals 0 and is therefore increasing in n_1.

2.) $K_1 - X_1(S_2) \leq n_1 < K_1, n_2 \geq K_2$: In this case, (A.8) is increasing in n_1 since $V_{i-1}(n_1,n_2)$ is increasing in n_1 by assumption (2b).

3.) $n_1 < K_1 - X_1(S_2), n_2 \geq K_2$: In this case, (A.8) does not depend on n_1 and is therefore increasing in n_1.

4.) $n_1 \geq K_1$, $K_2 - X_2(S_1) \leq n_2 < K_2$: In this case, (A.8) is increasing in n_1 since $V_{i-1}(n_1,n_2) - V_{i-1}(n_1,n_2-1)$ is increasing in n_1 by (2c).

5.) $K_1 - X_1(S_2) \leq n_1 < K_1$, $K_2 - X_2(S_1) \leq n_2 < K_2$: In this case (A.8) is increasing in n_1 by condition (2b).

6.) $n_1 < K_1 - X_1(S_2)$, $K_2 - X_2(S_1) \leq n_2 < K_2$: In this case, (A.8) does not depend on n_1.

7.) $n_1 \geq K_1$, $n_2 < K_2 - X_2(S_1)$: In this case, (A.8) is increasing by condition (2c).

8.) $K_1 - X_1(S_2) \leq n_1 < K_1, n_2 < K_2 - X_2(S_1)$: In this case, (A.8) is increasing in n_1 by condition (2b).

9.) $n_1 < K_1 - X_1(S_2), n_2 < K_2 - X_2(S_1)$: In this case, (A.8) does not depend on n_1.

Therefore, we have shown that the last term in (A.7) is also increasing in n_1, and that condition (1b) holds for U_i, when conditions (2a-2c) hold for V_{i-1}.

The rest of the cases are similar, and we omit the details. □.

We can now complete the proof of Theorem 2. To begin the successive approximation algorithm, set $V_0(n_1, n_2) = 0$ for all n_1, n_2. Note that V_0 satisfies conditions (2a-2f), therefore all U_i and V_i successively computed using (A.2), (A.3), and (A.4) will satisfy (1a-1h, 2a-2f). By Theorem 2.2 of Hernandez-Lerma [7], $U_i(n_1, n_2, 1)$ and $U_i(n_1, n_2, 2)$ converge to $U(n_1, n_2, 1)$ and $U(n_1, n_2, 2)$ for the average cost problem, and since they will satisfy conditions (1a) and (1e), the proof of Theorem 2 is complete.

Appendix B: The Heuristic of Fowler, Phillips, and Hogg

Fowler, Phillips, and Hogg [4] develop the NACHM heuristic for multiple job families under slightly different assumptions than those of this paper. They consider the dynamic problem formulated in Section 2, but with constant processing times, holding costs of one for all families, and knowledge of future arrivals. Specifically, they assume that the time of the next arrival of each job family can be predicted.

The decision logic of NACHM is separated into "push" decision logic and "pull" decision logic. The "push" decision logic is called when a job arrives and the machine is free. In this case, NACHM only considers the arriving job family in determining whether or not to start a batch. If the arriving job is of family j, the decision is to idle until the next arrival if

$$n_j \times T_j < 1/\mu_j - T_j \tag{B.1}$$

where n_j is the number of jobs of type j currently in queue, T_j is the time until the next arrival of type j, and $1/\mu_j$ is the constant processing time for family j. (B.1) holds when the delay caused by idling for those lots already waiting in queue is less than the waiting time saved for the next arriving job. If (B.1) does not hold, a batch of family j is started at the present time.

The "pull" decision logic is called when the machine completes a service. If there is a full load (a batch which equals the machine capacity) of at least one job family available at this time, a Weighted Shortest Processing Time (WSPT) scheme is used to select among those products with a full load. The value

$$W_j = \left((\sum_{i=1}^{m} n_i) - n_j \right) \times \frac{1}{\mu_j} \tag{B.2}$$

is calculated for each family j with a full load, and the family with the minimum value of W_j is selected for service.

When no full loads are available, (B.1) is first used for each job family to determine whether it is better to start a batch of that family now or idle until the next arrival. If (B.1) holds for all job families, then the decision is to idle. If (B.1) does not hold for each family, the WSPT scheme of (B.2) is used to select which family to start. Finally, if (B.1) holds for some families but not for others, the total delay D_j caused by following the course of action suggested by (B.1) is calculated for each job family over the horizon $1/\mu_j$. For those families for which (B.1) does not hold,

$$D_j = \sum_{i=1, i \neq j}^{m} (n_i/\mu_j) + \sum_{i=1}^{m} \max\{0, 1/\mu_j - T_i\}. \tag{B.3}$$

For those families for which (B.1) holds,

$$D_j = \sum_{i=1}^{m} (n_i \times T_j) + \sum_{i=1, i \neq j}^{m} ((n_i/\mu_j) + \max\{0, 1/\mu_j + T_j - T_i\}). \tag{B.4}$$

The job family with the minimum value of D_j is selected, and the action suggested by (B.1) (i.e. idle if it holds, begin service if it does not hold) is taken.

We extended NACHM to be able to use it under the more general assumptions of this paper. The modified version MNACHM is obtained from NACHM by simply substituting the expected service time $E[S_j]$ for the constant processing time $1/\mu_j$, and by multiplying all waiting time expressions by the appropriate holding cost. Equations (B.1) through (B.4) become

$$c_j \times n_j \times T_j < c_j(E[S_j] - T_j) \tag{B.5}$$

$$W_j = \left((\sum_{i=1}^{m} c_i n_i) - c_j n_j \right) \times E[S_j] \tag{B.6}$$

$$D_j = \sum_{i=1, i \neq j}^{m} (c_i n_i E[S_j]) + \sum_{i=1}^{m} c_i \max\{0, E[S_j] - T_i\} \tag{B.7}$$

$$D_j = \sum_{i=1}^{m} (c_i n_i T_j) + \sum_{i=1, i \neq j}^{m} ((c_i n_i E[S_j]) + c_i \max\{0, E[S_j] + T_j - T_i\}). \tag{B.8}$$

Acknowledgement:

We would like to thank Frank Fontana for his help with the computations. This research is partially supported by Grant No: DDM-9308290 from the National Science Foundation and a grant from the Center for Display Technology and Manufacturing at the University of Michigan.

Bibliography

[1] Barnett, A., and D.J. Kleitman, "Some Optimization Problems with Bulk-Service Queues," *Studies in Applied Mathematics,* 58 (1978) 277-290.

[2] Chandru, V., C.Y. Lee, and R. Uzsoy, "Minimizing Total Completion Time on Batch Processing Machines," *International Journal of Production Research,* 31 (1993): 2092-2121.

[3] Deb, R.K. and R.F. Serfozo, "Optimal Control of Batch Service Queues," *Advances in Applied Probability,* 5 (1973): 340–361.

[4] Fowler, J.W., D.T. Phillips, and G.L. Hogg, "Real-Time Control of Multiproduct Bulk-Service Semiconductor Manufacturing Processes," *IEEE Transactions on Semiconductor Manufacturing,* 5 (1992): 158–163.

[5] Glassey, C.R. and W.W. Weng, "Dynamic Batching Heuristic for Simultaneous Processing," *IEEE Transactions on Semiconductor Manufacturing,* 4 (1991): 77–82.

[6] Gurnani, H., R. Anupindi, and R. Akella, "Control of Batch Processing Systems in Semiconductor Wafer Fabrication Facilities," *IEEE Transactions on Semiconductor Manufacturing* 5 (1992): 319–328.

[7] Hernandez-Lerma, O., *Adaptive Markov Control Processes,* New York: Springer-Verlag, 1989.

[8] Ikura, Y. and M. Gimple, "Efficient Scheduling Algorithms for a Single Batch Processing Machine," *Operations Research Letters,* 5 (1986): 61–65.

[9] Lee, C.Y., R. Uzsoy, and L.A. Martin-Vega, "Efficient Algorithms for Scheduling Batch Processing Machines," *Operations Research,* 40 (1992): 764-775.

[10] Makis, V., "Optimal Control of a Batch Service Queueing System with Bounded Waiting Time," *Kybernetika,* 21 (1985): 262-271.

[11] Neuts, M.F., "A general class of bulk queues with Poisson input," *Annals of Mathematical Statistics,* 38 (1967): 759-770.

[12] Pinedo, M, *Scheduling: Theory, Algorithms, and Systems,* Englewood Cliffs, N.J.: Prentice Hall, 1995.

[13] Powell, W. and P. Humblet, "The Bulk Service Queue with a General Control Strategy: Theoretical Analysis and a New Computational Procedure," *Operations Research* 34 (1986): 267–275.

[14] Tijms, H.C, *Stochastic Modelling and Analysis: A Computational Approach,* New York: John Wiley & Sons, 1986.

[15] Uzsoy, R, "Scheduling Batch Processing Machines with Incompatible Job Families," Research Memorandum 94-3, School of Industrial Engineering, Purdue University, West Lafayette, IN, 47907–1287 (1994).

[16] Uzsoy, R., C.Y. Lee, and L.A. Martin-Vega, "A Review of Production Planning and Scheduling Models in the Semiconductor Industry Part I: System Characteristics, Performance Evaluation and Production Planning," *IIE Transactions on Scheduling and Logistics,* 24 (1992) 47-61.

[17] Uzsoy, R., C.Y. Lee, and L.A. Martin-Vega, "A Review of Production Planning and Scheduling Models in the Semiconductor Industry Part II: Shop Floor Control," *IIE Transactions on Scheduling and Logistics,* 26 (1994) 44-55.

[18] Weng, W.W. and R.C. Leachman, "An Improved Methodology for Real-Time Production Decisions at Batch-Process Work Stations," *IEEE Transactions on Semiconductor Manufacturing,* 6 (1993): 219–225.

[19] Wolff, R.W., *Stochastic Modeling and the Theory of Queues,* Englewood Cliffs, N.J.: Prentice Hall, 1989.