Probability in the Engineering and Informational Sciences, **15**, 2001, 465–479. Printed in the U.S.A.

A STOCHASTIC BATCHING AND SCHEDULING PROBLEM

GER KOOLE
Division of Mathematics and Computer Science
Vrije Universiteit
Amsterdam, The Netherlands
E-mail: Koole@cs.vu.nl

RHONDA RIGHTER
Department of Operations and Management Information Systems
Santa Clara University
Santa Clara, CA 95053
E-mail: RRighter@scu.edu

We consider a batch scheduling problem in which the processing time of a batch of jobs equals the maximum of the processing times of all jobs in the batch. This is the case, for example, for burn-in operations in semiconductor manufacturing and other testing operations. Processing times are assumed to be random, and we consider minimizing the makespan and the flow time. The problem is much more difficult than the corresponding deterministic problem, and the optimal policy may have many counterintuitive properties. We prove various structural properties of the optimal policy and use these to develop a polynomial-time algorithm to compute the optimal policy.

1. INTRODUCTION

We consider a batch scheduling problem in which there is a single machine that can process several jobs at once, and the processing time of a batch of jobs equals the maximum of the processing times of all jobs in the batch. This is the case, for example, for burn-in operations in semiconductor manufacturing and other testing operations. For semiconductor burn-in, each chip has a minimum burn-in time (its processing time) and chips may be batched together in the burn-in oven. The burn-in

 0269-9648/01 $12.50

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

time for the batch is the maximum of the minimum burn-in times for all chips in the oven. Another application is for batch transportation to parallel machines. In this case, at most m jobs are transported together on a pallet to a station with m parallel machines, and the pallet cannot be transported to the next station until all jobs are complete.

There is a fixed set of jobs to be processed, processing times are random, and we consider minimizing the makespan (time to process all jobs) and the flow time (the sum of job completion times).

The deterministic problem, with various objective functions, has been studied by many authors, including Ahmadi et al. [1], Albers and Brucker [2], Brucker [3], Chandru et al. [4,5], Cheng and Kovalyov [6], Coffman et al. [7], Dobson et al. [9], Glassey and Weng [10], Hochbaum and Landy [11,12], Ikura and Gimple [13], Nadder and Santos [14], Shallcross [16], Sung and Choung [17], and Wagelmans and Gerodimos [18]. See Potts and Kovalyov [15] for a review. Prior work on the stochastic problem, when processing times are random, has assumed that the processing time of a batch has the same distribution regardless of the number of jobs in the batch and the job identities (all jobs are identical). See, for example, Deb and Serfozo [8].

The stochastic problem is much more difficult than the corresponding deterministic problem, and the optimal policy may have many counterintuitive properties. For example, for the deterministic problem and for both makespan and flow-time objectives, it is always optimal to batch jobs with the same processing time together. However, when processing times are random, it may be optimal to process jobs with the same processing time distributions separately or with jobs with different processing time distributions, rather than processing these jobs together. We give examples later. We prove various structural properties of the optimal policy for the makespan problem in Section 2 and for the flow-time problem in Section 3. We are able to more completely characterize the optimal flow-time policy when the maximal batch size is two and processing times are exponential, and in this case, we develop an algorithm to find the optimal policy.

2. MAKESPAN

We first consider the problem of minimizing the makespan, or time until all jobs are completed. In this case, we need only determine the batches, since the makespan is the same for any sequence. Also, since there is a single machine, the optimal policy within the class of all dynamic policies is static; that is, it is sufficient to determine the batches off-line. Suppose the maximal batch size is b. For the deterministic problem, an optimal solution to the batching problem can be obtained with the following algorithm [17].

Algorithm A

1. Batch together the b largest jobs.
2. Repeat Step 1 until there are fewer than b jobs left.
3. Batch any remaining jobs together.

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

Intuitively, this is completely clear: The largest job should be part of some batch and, obviously, we should add to this batch the next $b-1$ largest jobs. Continuing in this manner gives Algorithm A.

For a nondeterministic processing time, it is tempting to apply the same algorithm with the means of the processing times. That Algorithm A is no longer optimal for this case is illustrated by the following example. Suppose $b = 2$, and we have three jobs, two with deterministic processing times of 1 and one job with processing time 0 or 3, each with probability 1/2. The latter job has the largest mean processing time, so Algorithm A would give an expected makespan of 3, whereas scheduling the two deterministic jobs together gives expected makespan of $2\frac{1}{2}$.

There is also an optimal solution to the deterministic problem such that all jobs with the same processing times are batched together, perhaps in multiple batches if there are more than b jobs with the same processing times. So, for the above example, another optimal solution for the deterministic version is to batch together the two jobs with processing times of 1 and to process the job with the processing time of $1\frac{1}{2}$ separately. Again, it is tempting to suppose that an optimal solution to the problem with random processing times would also batch together jobs with the same mean processing times. However, we show by a counterexample at the end of the section that this is not the case.

The counterexamples illustrate that we cannot hope for a complete solution in the case of arbitrary random processing times. However, some general results can be obtained, and under certain assumptions, a complete solution can be derived. We start with a general result. It is "intuitively optimal" that we want to batch as many jobs together as possible. This is, in fact, true for the makespan objective even when processing times are random.

LEMMA 2.1: *For arbitrarily distributed processing times, if there are two batches of size n_1 and n_2 such that $n_1 + n_2 \leq b$, then the expected makespan will be smaller if the two batches are batched together.*

When we have processing times that can be stochastically ordered, as is the case with exponential processing times, Algorithm A is optimal, as we show below. We need a few preliminary results. The proof of the following theorem is similar to that of Theorem 3.2, so we omit it.

THEOREM 2.2: *Suppose that for a set of $k_1 + k_2$ jobs, $k_1 \leq k_2$, all processing times can be stochastically ordered, k_1 of the jobs are to be processed together in a batch, and the remaining k_2 jobs are to be processed together in a batch. Then, to minimize expected makespan, the first batch should contain the k_1 jobs with the smallest means, and the other should contain the k_2 jobs with the largest means.*

The following simple fact will be useful throughout.

Fact A: For any random variables X_1 and X_2, $E\max(X_1, X_2) = EX_1 + EX_2 - E\min(X_1, X_2)$.

THEOREM 2.3: *Suppose that for a set of n jobs, where $b < n < 2b$, all processing times can be stochastically ordered and these n jobs are to be processed in two*

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

batches. Then, to minimize the expected makespan, one batch should contain the b jobs with the largest means, and the other should contain the rest.

PROOF: If one batch contains b jobs, we know from Theorem 2.2 that it should contain the jobs with the largest means. Therefore, suppose, by way of contradiction, that schedule π processes in batches of size k and $n-k$, where $k \le n-k < b$. By Theorem 2.2, if we order the jobs so that $X_1 \le_{\text{st}} X_2 \le_{\text{st}} \cdots \le_{\text{st}} X_n$, then one batch should contain jobs $1,\ldots,k$, and the other should contain jobs $k+1,\ldots,n$. Consider schedule π' that schedules jobs $1,\ldots,k-1$ in one batch and jobs $k,\ldots,n$ in the other. Let U be the contribution to the makespan of the other jobs (in addition to the n we are considering). From Fact A, the expected makespan under π is

$$\begin{aligned} &U + E\max(X_1,\ldots,X_k) + E\max(X_{k+1},\ldots,X_n) \\ &\quad = U + E\max(X_1,\ldots,X_{k-1}) + EX_k - E\min(X_k,\max(X_1,\ldots,X_{k-1})) \\ &\qquad + E\max(X_{k+1},\ldots,X_n), \end{aligned}$$

and under π', it is

$$\begin{aligned} &U + E\max(X_1,\ldots,X_{k-1}) + E\max(X_k,\ldots,X_n) \\ &\quad = U + E\max(X_1,\ldots,X_{k-1}) + EX_k + E\max(X_{k+1},\ldots,X_n) \\ &\qquad - E\min(X_k,\max(X_{k+1},\ldots,X_n)). \end{aligned}$$

Since $\max(X_{k+1},\ldots,X_n) \ge_{\text{st}} \max(X_1,\ldots,X_{k-1})$ for $k \le n-k$, and therefore $E\max(X_{k+1},\ldots,X_n) \ge E\max(X_1,\ldots,X_{k-1})$, we have that the expected makespan is smaller under π' than under π, and we are done. ■

COROLLARY 2.4: *If all processing times can be stochastically ordered, Algorithm A minimizes the expected makespan.*

We have shown that a solution that is optimal for the deterministic problem (with all processing times identically equal to their means) is also optimal for the nondeterministic problem, as long as processing times are stochastically ordered. As we mentioned earlier, there is also an optimal solution to the deterministic problem such that all jobs with the same processing times are batched together. Such a solution is no longer necessarily optimal when processing times are random, even when they are stochastically ordered.

For example, if we have four jobs with deterministic processing times 1/3, 1/2, 1/2, and 1, and $b = 2$, it is optimal to have two batches, one with the two jobs with processing times of 1/2, and one with the other two jobs. The makespan would be 3/2.

Now consider the same example with exponentially distributed processing times and with mean processing times of 1/3, 1/2, 1/2, and 1. The expected makespan when the jobs with the same processing time distributions are batched together is, from Fact A,

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

$$\frac{1}{2}+\frac{1}{2}-\frac{1}{4}+\frac{1}{3}+1-\frac{1}{4}=\frac{11}{6}.$$

However, if the job with a mean processing time of 1/3 were batched with one of the ones with a mean processing time of 1/2, and the other two were batched together (Algorithm A), the expected makespan would be $9/5 < 11/6$.

3. FLOW TIME

Fix any schedule and let n_i be the number of jobs in the ith batch, K be the total number of batches, and X_j^i be the processing time of the jth job in the ith batch. We denote the schedule by

$$(1,\ldots,n_1)(n_1+1,\ldots,n_1+n_2)\cdots\left(\sum_{i=1}^{K-1}n_i+1,\ldots,\sum_{i=1}^{K}n_i\right).$$

The flow time (FT) for this schedule is

$$\mathrm{FT}=\sum_{i=1}^{K}n_i\sum_{l=1}^{i}\max_{j=1,\ldots,n_i}X_j^i=\sum_{i=1}^{K}\left(\sum_{l=i}^{K}n_l\right)\max_{j=1,\ldots,n_i}X_j^i.$$

We say that $\sum_{l=i}^{K}n_l\max_{j=1,\ldots,n_i}X_j^i$ is the contribution to the flow time of the ith batch.

Since there is a single machine, the optimal policy within the class of all dynamic policies is static; that is, it is sufficient to determine the batches and the sequencing of batches off-line. When the batches have been determined, it is easy to show that they should be processed according to the WSEPT (weighted shortest expected processing time first) sequence, where the weights are the reciprocal batch size.

LEMMA 3.1: *Batches should be processed in increasing order of* $(E\max_{j=1,\ldots,n_i}X_j^i)/n_l$.

Note that WSEPT is equivalent to the "$c\mu$ rule," where c is the batch size and μ is the rate of processing or the reciprocal of the mean processing time for the batch. Under the $c\mu$ rule, batches are processed in decreasing order of $c\mu$.

For the makespan, we found in Lemma 2.1 that joining batches is better if possible. This is not the case for the flow time, as can be shown with a simple deterministic counterexample. We can, however, prove the counterpart of Theorem 2.2 for processing times that are stochastically ordered.

THEOREM 3.2: *Suppose that for a set of* k_1+k_2 *jobs,* $k_1\le k_2$, *all processing times can be stochastically ordered;* k_1 *of the jobs are to be processed together in a batch, with the remaining* k_2 *jobs to be processed together in a batch later in the schedule. Then, to minimize the mean flow time, the first batch should contain the* k_1 *jobs with the smallest means, and the other should contain the* k_2 *jobs with the largest means.*

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

PROOF: Let the processing times of the $k_1 + k_2$ jobs be $X_1 \leq_{\text{st}} X_2 \leq_{\text{st}} \cdots \leq_{\text{st}} X_{k_1+k_2}$, and let $S_1 = \{i_1, i_2, \ldots, i_{k_1}\}$ and $S_2 = \{j_1, j_2, \ldots, j_{k_2}\}$ be a partition of $\{1,2,\ldots,k_1+k_2\}$, so $S_1 \cup S_2 = \{1,2,\ldots,k_1+k_2\}$. Let π be a schedule that processes S_1 before S_2 and is otherwise arbitrary. Suppose $S_1 \neq \{1,2,\ldots,k_1\}$. Let $T_1 = S_1 \cap \{1,2,\ldots,k_1\}$, $T_2 = S_2 \cap \{1,2,\ldots,k_1\}$, $T_3 = S_1 \cap \{k_1+1, k_1+2, \ldots, k_1+k_2\}$, and $T_4 = S_2 \cap \{k_1+1, k_1+2, \ldots, k_1+k_2\}$. Then, $|T_1| =: l$, $|T_4| =: k_2 - k_1 + l \geq l$, and $|T_2| = |T_3| = k_1 - l$, and $X_i^1 \leq_{\text{st}} X_i^4$, $i = 1,\ldots,l$, and $X_i^2 \leq_{\text{st}} X_i^3$, $i = 1,\ldots,k_1 - l$, where X_i^j is the ith largest processing time for the jobs in T_j, $j = 1,2,3,4$. Therefore, $W_1 \leq_{\text{st}} W_4$ and $W_2 \leq_{\text{st}} W_3$, where $W_i = \max_{j \in T_i} X_j$ for $i = 1,2,3,4$ and $\max_{j \in \emptyset} X_j := 0$. Let π' agree with π except that it schedules $T_1 \cup T_2 = \{1,2,\ldots,k_1\}$ in the position that π schedules $T_1 \cup T_3 = S_1$ and it schedules $T_3 \cup T_4 = \{k_1+1, k_1+2, \ldots, k_1+k_2\}$ in the position that π schedules $T_2 \cup T_4 = S_2$. We must show that the expected flow time under π' is less than under π. Let n_i be the number of uncompleted jobs at the time π schedules S_i, where, by definition, $n_1 \geq n_2$. The flow time under π is

$$U + n_1 E[\max(W_1, W_3)] + n_2 E[\max(W_2, W_4)],$$

whereas that under π' is

$$U + n_1 E[\max(W_1, W_2)] + n_2 E[\max(W_3, W_4)],$$

where U is the contribution to the total mean flow time for the rest of the schedule excluding S_1 and S_2. Since $W_1 \leq_{\text{st}} W_4$ and $W_2 \leq_{\text{st}} W_3$, we can generate and couple $\widetilde{W}_i$, $i = 1,2,3,4$ so that the marginal distributions are correct, $\widetilde{W}_1 \leq \widetilde{W}_4$ and $\widetilde{W}_2 \leq \widetilde{W}_3$ with probability 1, $\widetilde{W}_1$ is independent of $\widetilde{W}_2$ and $\widetilde{W}_3$, and $\widetilde{W}_4$ is independent of $\widetilde{W}_2$ and $\widetilde{W}_3$. The result follows. ■

Theorem 3.2 greatly reduces the complexity of computing an optimal policy. Unfortunately, as we will see in examples below, the theorem does not hold for $k_1 > k_2$, so it is not the case that all jobs will be processed in increasing order of their mean processing times. Indeed, we have not been able to show that the jobs in the same batch should be consecutive in terms of their processing times. This is easy to show when processing times are deterministic.

3.1. Maximal Batch Size of Two

To obtain more complete results, we now restrict our attention to a maximal batch size of $b = 2$. We will refer to batches of size 2 as double batches, and batches of size 1 as single batches.

Let m_i be the mean processing time of job i, $m_i = EX_i$, and let $m_{ij} = E\min\{X_i, X_j\}$. Recall Fact A that $E\max(X_i, X_j) = m_i + m_j - m_{ij}$.

We first consider the choice of processing two jobs as a double batch versus processing them singly. The following is easily shown.

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

LEMMA 3.3: *If there are n remaining jobs and two jobs are to be processed next (call them jobs 1 and 2 with $m_1 \leq m_2$), we prefer batching the two together rather than processing them singly one after the other if and only if*

$$n \geq \frac{m_2}{m_{12}}.$$

A consequence of Lemma 3.3 is that when there are only two jobs with the same distribution, the flow time may be smaller when they are processed one at a time than when they are batched together. For example, suppose that the processing times of each of two jobs are Bernoulli with parameter $p \leq 1/2$, say $p = 1/4$. Then, $n = 2 < (1/4)/(1/16) = m_2/m_{12} = 1/p$ and the flow time when they are processed singly is $3/4 = 3p$, whereas it is $14/16 > 3/4 = 2p - p^2$ when they are batched together. Of course, if the processing times were deterministic, it would always be better to batch two jobs with the same processing times together.

From the previous example, we see that we will be more likely to batch jobs with the same distribution together if the distribution has less variability in some sense. We now make this notion precise. Suppose jobs 1 and 2 have the same processing time distribution, F. From Lemma 3.3, we will be more likely to batch them together if the mean of the minimum processing time is larger relative to the mean processing time for each job. Thus, if $\hat{F}$ has smaller variability than F in the convex ordering sense, we will be more likely to batch the two jobs together if their common distribution is $\hat{F}$ rather than F. (Recall that $\hat{F}$ is smaller than F in the convex ordering sense, or it has smaller variability in that sense, if

$$\int g(x)\, d\hat{F}(x) \leq \int g(x)\, dF(x)$$

for all convex functions g.) This follows because $\min(x, a)$ is convex in x for all a.

Now we consider the batching and scheduling choices associated with a set of three jobs.

Choose three arbitrary jobs, called jobs 1, 2, and 3, and consider a schedule in which one job is processed as a single batch at some point before the other two, which are batched together. Let $n \geq 2$ be the number of remaining jobs when the double batch is processed, and let $n + k$, $k \geq 1$, be the number of remaining jobs when the single job is processed. Then, we prefer job 1 to be the single job rather than job 2 (i.e., $S_1(1)S_2(2,3)S_3 > S_1(2)S_2(1,3)S_3$, where S_1, S_2, and S_3 are arbitrary subschedules and $>$ means the schedule has smaller mean flow time) if

$$(n + k)m_1 + n(m_2 + m_3 - m_{23}) \leq (n + k)m_2 + n(m_1 + m_3 - m_{13});$$

that is, if

$$0 \leq k(m_2 - m_1) + (m_{23} - m_{13}).$$

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

The right-hand side will be positive if $m_2 \geq m_1$ and $m_{23} \geq m_{13}$. A sufficient condition for both of these inequalities is $X_2 \geq_{\text{icv}} X_1$, where we say that $X \geq_{\text{icv}} Y$, or X is larger than Y in the increasing concave sense, if $Ef(X) \geq Ef(Y)$ for all increasing and concave f. Note that $m_2 \geq m_1$ is not sufficient for $m_{23} \geq m_{13}$. For example, suppose $X_1 \equiv 1$, X_2 is equally likely to be 0 or 3, and $X_3 \equiv 1.5$. Then, $m_1 = 1$, $m_2 = 1.5$, $m_{13} = 1$, and $m_{23} = 0.75$.

The above argument gives us the following lemma.

LEMMA 3.4: *Consider three jobs with processing times such that $X_1 \leq_{\text{icv}} X_2$ and $X_1 \leq_{\text{icv}} X_3$. If one job is to be processed by itself when there are $n + k$ remaining jobs and the other two are to be batched together and processed when there are n remaining jobs, then the single batch should be job 1 to minimize mean flow time; that is, $S_1(1)S_2(2,3)S_3 > S_1(2)S_2(1,3)S_3$ and $S_1(1)S_2(2,3)S_3 > S_1(3)S_2(1,2)S_3$.*

3.2. Exponential Processing Times and $b = 2$

Now suppose all processing times are exponential and $b = 2$.

3.2.1. General properties. Since exponential random variables can be ordered in the stochastic sense, we know from Theorem 3.2 that for all double batches, for all single batches, and for single batches preceding double batches, jobs should be sequenced in increasing order of their mean processing times. However, as we will see later, it may be optimal for the job in a single batch to have a smaller mean processing time than the jobs in a preceding double batch. From Lemma 3.3 and Lemma 3.4 or Theorem 3.2, we have the following, where we use the fact that exponential random variables can be ordered in the increasing concave and stochastic senses.

LEMMA 3.5: *If there are n remaining jobs and two jobs with processing rates $\lambda_1 \geq \lambda_2$ are to be processed next, we prefer batching the two together to processing them singly one after the other if and only if*

$$n \geq \frac{\lambda_1 + \lambda_2}{\lambda_2}.$$

LEMMA 3.6: *For three jobs with processing rates $\lambda_1 \geq \lambda_2 \geq \lambda_3$, if one job is to be processed by itself when there are $n + k$ remaining jobs and the other two are to be batched together and processed when there are n remaining jobs, then the single batch should be the job with the smallest mean processing time in order to minimize mean flow time.*

Now we consider the sequence of jobs when a single batch follows a double batch. Choose three arbitrary jobs (called jobs 1, 2, and 3) and consider a schedule in which two of the jobs are batched together and scheduled at some point before the third job, which is processed by itself. Let $n \geq 1$ be the number of remaining jobs when the third job is processed and let $n + k$, $k \geq 2$, be the number of remaining jobs

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

when the batch of two jobs are processed. Suppose that $\lambda_1 \geq \lambda_2 \geq \lambda_3$. Then, we prefer job 3 to be the job that is processed by itself rather than job 2, $S_1(1,2)S_2(3)S_3 \succ S_1(2,3)S_2(1)S_3$, if

$$(n+k)\left(\frac{1}{\lambda_1}+\frac{1}{\lambda_2}-\frac{1}{\lambda_1+\lambda_2}\right)+n\frac{1}{\lambda_3}-(n+k)\left(\frac{1}{\lambda_1}+\frac{1}{\lambda_3}-\frac{1}{\lambda_1+\lambda_3}\right)$$
$$-n\frac{1}{\lambda_2}\leq 0,$$

which is equivalent to

$$n \leq k\lambda_1 \frac{\lambda_1+\lambda_2+\lambda_3}{\lambda_2\lambda_3}.$$

Similarly, we prefer job 3 to be the single job rather than job 1 if

$$n \leq k\lambda_2 \frac{\lambda_1+\lambda_2+\lambda_3}{\lambda_1\lambda_3} \leq k\lambda_1 \frac{\lambda_1+\lambda_2+\lambda_3}{\lambda_2\lambda_3},$$

and we prefer job 2 to be the single job rather than job 1 if

$$n \leq k\lambda_3 \frac{\lambda_1+\lambda_2+\lambda_3}{\lambda_1\lambda_2} \leq k\lambda_2 \frac{\lambda_1+\lambda_2+\lambda_3}{\lambda_1\lambda_3}.$$

Thus, we have the following.

LEMMA 3.7: *Suppose that for three jobs with processing rates $\lambda_1 \geq \lambda_2 \geq \lambda_3$, two of the three are to be batched together and processed when there are $n+k$, $k \geq 2$, remaining jobs, and the third is to be processed by itself when there are $n \geq 1$ remaining jobs. Then, job 3 should be the job processed by itself (i.e., $S_1(1,2)S_2(3)S_3 \succ S_1(2,3)S_2(1)S_3$) if*

$$n \leq k\lambda_2 \frac{\lambda_1+\lambda_2+\lambda_3}{\lambda_1\lambda_3},$$

otherwise job 1 should be the job processed by itself ($S_1(1,2)S_2(3)S_3 \prec S_1(2,3)S_2(1)S_3$).

Note that when $\lambda_1 = \lambda_2$, job 3 should be the job processed by itself if

$$n \leq k\frac{2\lambda_1+\lambda_3}{\lambda_3},$$

and when $\lambda_2 = \lambda_3$, job 3 should be the job processed by itself if

$$n \leq k\frac{\lambda_1+2\lambda_2}{\lambda_1}.$$

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

Thus, when processing three jobs where two have the same mean processing time, we sometimes do not want the two with the same mean processing times to be batched together. This is in contrast with the deterministic problem in which it is always optimal to batch jobs with the same processing times together. For example, with only three deterministic jobs, jobs 1, 2, and 3, with rates 3, 2, and 2 (means 1/3, 1/2, and 1/2), respectively, the optimal schedule is (2,3)(1). For exponential processing times, this policy gives a mean flow time of 2.5. The optimal policy for exponential processing times is (1,2)(3). This policy gives a mean flow time of 2.4. As another example, consider 11 jobs with rates 10, 10, 9, 1, 1, 1, 1, 1, 1, 1, 1. The optimal policy in the deterministic case is (1,2), 3, (4,5), (6,7), (8,9), (10,11), but in the exponential case, it is (2,3), 1, (4,5), (6,7), (8,9), 10, 11 (or jobs 10 and 11 may be batched together).

We now can describe some of the structure of an optimal policy. Let us order the jobs in increasing order of their mean processing times, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, where n is the number of remaining jobs. From Lemma 3.1 (or Theorem 3.2) we have the following corollary.

COROLLARY 3.8: *The jobs in single batches should be processed in order (in increasing order of their indices), and the jobs in double batches should be processed in order.*

From Lemma 3.6 and Corollary 3.8, we have the following corollaries.

COROLLARY 3.9: *If there are any single batches before a double batch, the jobs in those single batches should have smaller processing times (smaller indices) than any of the later jobs. Hence, if there are any single jobs at the beginning of the schedule, they should be the shortest jobs.*

COROLLARY 3.10: *If there is a set of consecutive double batches with no single batches scheduled after them (except for job n—see Corollary 3.12—and, possibly, job $n-1$ if $\lambda_{n-1} = \lambda_n$), then they should be the longest jobs except for job n and, possibly, job $n-1$.*

From Lemmas 3.6 and 3.7 and Theorem 3.2, we have the following corollaries.

COROLLARY 3.11: *Each double batch should consist of consecutive jobs.*

COROLLARY 3.12: *The last batch should be a single batch containing the job with the longest expected processing time, job n.*

PROOF: The job that is processed last will either be processed by itself or batched with another job. By Lemma 3.5, it is better to process the two jobs separately than together because when they are the only jobs left, n will be 2 and $(\lambda_1 + \lambda_2)/\lambda_2 \geq 2$. (If they have the same rate, we are indifferent.) Thus, the last job should be processed by itself. From Theorem 3.2, this job should have the longest mean processing times of all jobs that are processed by themselves. Consider the last batch of two jobs in the schedule. (Call these jobs 1 and 2.) From Theorem 3.2, the processing times of jobs 1 and 2 should be the largest of all processing times for jobs that are

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

scheduled in batches of two. Therefore, we need only show that the last job (call it job 3) has a longer mean processing time than job 1 or 2. However, this follows from Lemma 3.7 because when the last job is processed, $n = 1$ and

$$k\lambda_2 \frac{\lambda_1 + \lambda_2 + \lambda_3}{\lambda_1 \lambda_3} \geq k \geq 2. \qquad \blacksquare$$

An argument similar to that of the above proof, along with Corollaries 3.8 and 3.9, gives us the following corollary.

COROLLARY 3.13: *The penultimate batch should contain job $n-1$, in either a single or double batch. If it is a double batch, it will also contain job $n-2$.*

3.2.2. Properties for groups of identical jobs. We now consider the case when we have multiple jobs with the same processing time. Suppose that among all the jobs, s of them have the same processing rate, λ_s (call them s-jobs, s for "stochastically the same"). From Lemma 3.5, we will prefer to have a batch of two s-jobs than to consecutively process them singly. Thus, the only way we could have two single s-jobs in an optimal schedule is to have double batches between them. The jobs in these double batches would have to have larger mean processing times than the s-jobs, by Corollary 3.9. Consider schedule A: At some point in its schedule, process a single s-job, followed by l double batches of jobs with larger mean processing times, followed by another single s-job [i.e., abusing notation a bit, $A = S_1(s)S_2(s)S_3$]. Consider schedule B, which agrees with schedule A except that it processes the two s-jobs together as a double batch before the l double batches, $B = S_1(s,s)S_2S_3$. Let Δ be the flow time for A minus the flow time for B, so we want to show that $\Delta \geq 0$. For the jobs that come after the second single s-job in schedule A, say r of them, the flow time is larger under A than B by $2/\lambda_s - (2/\lambda_s - 1/(2\lambda_s)) = 1/(2\lambda_s)$. For the $2l$ jobs in the double batches and the first single s-job, the flow time for B is larger by $(2/\lambda_s - 1/(2\lambda_s)) - 1/\lambda_s = 1/(2\lambda_s)$. For the second single s-job, the flow time under B is smaller by $2/\lambda_s - (2/\lambda_s - 1/(2\lambda_s)) + \sum_{i=1}^{l} M_l$, where M_l is the mean processing time of batch l and $M_l \geq (2/\lambda_s - 1/(2\lambda_s)) = 3/(2\lambda_s)$. Hence,

$$\Delta \geq \frac{r}{2\lambda_s} - \frac{2l+1}{2\lambda_s} + \frac{1}{2\lambda_s} + \frac{3l}{2\lambda_s} = \frac{r+l}{2\lambda_s} \geq 0.$$

We therefore have the following proposition.

PROPOSITION 3.14: *If there are multiple jobs with the same processing rates, at most one of them should be scheduled as a single batch.*

From Corollary 3.11, at most two of the double batches containing s-jobs will contain jobs other than s-jobs, and the two other jobs, call them o_1 and o_2, will be such that $\lambda_{o_1} \geq \lambda_s \geq \lambda_{o_2}$. If there is a single s-job, it should be scheduled after all of the other s-jobs. This is a consequence of Lemmas 3.6 and 3.15.

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

LEMMA 3.15: *For three jobs such that $\lambda_1 = \lambda_2 \geq \lambda_3$, and such that job 1 is to be processed as a single batch and jobs 2 and 3 are to be processed as a double batch, the flow time will be smaller if the double batch is processed first; that is, $S_1(2,3)S_2(1)S_3 > S_1(1)S_2(2,3)S_3$.*

PROOF: From Lemma 3.1, we need only show $(1/\lambda_2 + 1/\lambda_3 - 1/(\lambda_2 + \lambda_3))/2 \leq 1/\lambda_1 = 1/\lambda_2$, which follows after some algebra. ■

Putting our results together, we have the following corollaries.

COROLLARY 3.16: *For a set of jobs (s-jobs) with the same processing rate, all but at most three should be batched together into double batches. At most, one s-job should be scheduled as a single batch, and it should be scheduled after all the other s-jobs. At most, one s-job should be batched with a consecutive job with smaller processing time, and at most, one s-job should be batched with a consecutive job with larger processing time.*

COROLLARY 3.17: *If $\lambda_1 = \lambda_2 = \cdots = \lambda_k$, then the first $k-2$ (if k is even, otherwise $k-1$) jobs should be batched into double batches and they should be scheduled first.*

Note that after applying Corollary 3.17, the batched jobs at the beginning of the schedule can be ignored thereafter, since later decisions depend on the number of remaining jobs and not on the number of jobs that have already been processed.

3.2.3. Scheduling the shortest job. Now we consider job 1. From Lemma 3.5, if $\lambda_1 \geq (n-1)\lambda_2$, we prefer that jobs 1 and 2 be processed as single batches at the beginning of the schedule rather than being processed together. If they are processed together, then it will be at the beginning of the schedule, from Corollaries 3.8 and 3.9. Also, if it is optimal to process job 1 in a double batch, then it must be batched with job 2 from Corollary 3.11. Thus, if $\lambda_1 \geq (n-1)\lambda_2$, we will want to process job 1 as a single batch, and the only remaining question for job 1 is where it should be placed in the schedule. (From Lemma 3.7 we may want to schedule short single batches after longer jobs in double batches.) If $n = 2$, we are done, so suppose $n > 2$.

If job 1 is to be a single batch and jobs 2 and 3 are to be a double batch, then from Lemma 3.1, we prefer job 1 to be scheduled first if $1/\lambda_1 \leq (1/\lambda_2 + 1/\lambda_3 - 1/(\lambda_2 + \lambda_3))/2$; that is, if

$$\lambda_1 \geq \frac{2\lambda_2\lambda_3(\lambda_2 + \lambda_3)}{\lambda_2^2 + \lambda_3^2 + \lambda_2\lambda_3}.$$

Note that the right-hand side is at most $(4/3)\lambda_2$, so if $\lambda_1 \geq (n-1)\lambda_2 \geq 2\lambda_2$, then $\lambda_1 \geq (4/3)\lambda_2 \geq (4/3)\lambda_k$ for all $k \geq 2$. Therefore, if $(1)(2)S > (1,2)S$ for any subschedule S, then job 1 should be scheduled as a single batch before all other jobs. We have the following:

PROPOSITION 3.18: *If $\lambda_1 \geq (n-1)\lambda_2$, then job 1 should be scheduled as a single batch at the beginning of the schedule.*

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

Now, suppose $\lambda_1 \leq (n-1)\lambda_2$ or, equivalently, $n \geq (\lambda_1 + \lambda_2)/\lambda_2$. Then, we prefer that job 1 be in a double batch rather than that jobs 1 and 2 be processed as single batches at the beginning of the schedule. It is still possible that the optimal policy processes job 1 as a single batch, but it would have to either batch job 2 with job 3, or schedule job 2 as a single batch later, after some double batches. We now show that the latter will not be optimal. Suppose an optimal schedule, call it schedule A, processes job 1 as a single batch, followed by l double batches of jobs with larger mean processing times than either jobs 1 or 2, followed by job 2 as a single batch. Consider schedule B, which agrees with schedule A except that it processes jobs 1 and 2 together as a double batch before the l double batches. Let Δ be the flow time for schedule A minus the flow time for schedule B, so we want to show that $\Delta \geq 0$. For the jobs that come after job 2 in schedule A, $n-2-2l$ of them, the flow time is larger under schedule A than schedule B by $1/\lambda_1 + 1/\lambda_2 - (1/\lambda_1 + 1/\lambda_2 - 1/(\lambda_1 + \lambda_2)) = 1/(\lambda_1 + \lambda_2)$. For the $2l$ jobs in the double batches and job 1, the flow time for B is larger by $(1/\lambda_1 + 1/\lambda_2 - 1/(\lambda_1 + \lambda_2)) - 1/\lambda_1 = 1/\lambda_2 - 1/(\lambda_1 + \lambda_2)$. For job 2, the flow time under schedule B is smaller by $1/\lambda_1 + 1/\lambda_2 - (1/\lambda_1 + 1/\lambda_2 - 1/(\lambda_1 + \lambda_2)) + \sum_{i=1}^{l} M_l = 1/(\lambda_1 + \lambda_2) + \sum_{i=1}^{l} M_i$, where M_i is the mean processing time of batch i. For schedule A to be optimal, it must be the case that $M_i \geq 2/\lambda_2$ for all i. Otherwise, by Lemma 3.1, it would be better to schedule job 2 before some of the l double batches. Hence,

$$\Delta \geq \frac{n-2-2l}{\lambda_1+\lambda_2} - (2l+1)\left(\frac{1}{\lambda_2} - \frac{1}{\lambda_1+\lambda_2}\right) + \frac{1}{\lambda_1+\lambda_2} + \frac{2l}{\lambda_2}$$

$$= \frac{n}{\lambda_1+\lambda_2} - \frac{1}{\lambda_2} \geq 0,$$

where the second inequality follows from the assumption that $n \geq (\lambda_1 + \lambda_2)/\lambda_2$. We have reached a contradiction, so schedule A cannot be optimal. Therefore, when $\lambda_1 \leq (n-1)\lambda_2$, it is not optimal for jobs 1 and 2 to both be processed singly. Since job 2 cannot be processed as a single job unless job 1 is, this means that job 2 should be processed in a double batch. By Corollary 3.11, job 2 must be batched with either job 1 or job 3. We therefore have the following proposition.

PROPOSITION 3.19: *If $\lambda_1 \leq (n-1)\lambda_2$, then the optimal schedule batches job 2 with either job 1 or job 3.*

3.2.4. Algorithm to compute the optimal policy. Now, we can specify an algorithm to compute the optimal policy that uses our structural results. We assume jobs are ordered so that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. It will be convenient to define two algorithms, one assuming an arbitrary set of jobs and no preconditions (Algorithm B) and the other assuming that we have determined that job 1 is to be processed singly but we have not yet determined where in the schedule that will occur (Algorithm C).

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

Algorithm B: Compute the optimal policy for an arbitrary set of jobs

Do while there are more than three remaining jobs, $n > 3$.

1. If $\lambda_1 = \lambda_2 = \cdots = \lambda_k$, let $\hat{k} = k - 2$ if k is even and $\hat{k} = k - 1$ otherwise. The first $\hat{k}$ jobs should be batched into double batches and they should be scheduled first. Compute the optimal policy with the remaining $n - \hat{k}$ jobs (relabeled $1,2,\ldots,n - \hat{k}$, and n becomes $n - \hat{k}$), using Algorithm B. If $k \leq 2$, go to Step 2.
2. If $\lambda_1 \geq (n-1)\lambda_2$, then job 1 should be scheduled as a single batch at the beginning of the schedule. Compute the optimal policy with the remaining $n - 1$ jobs using Algorithm B. Otherwise, go to Step 3.

3a. Compute the optimal schedule for the remaining $n - 2$ jobs assuming jobs 1 and 2 are batched together and scheduled first, using Algorithm B.

3b. If

$$\frac{1}{\lambda_2} + \frac{1}{\lambda_3} - \frac{1}{\lambda_2 + \lambda_3} \geq \frac{2}{\lambda_1},$$

compute the optimal schedule, using Algorithm B, for the remaining $n - 3$ (relabeled) jobs assuming the schedule begins with (1)(2,3). Otherwise, compute the optimal policy for the remaining (relabeled) $n - 2$ jobs (including job 1) assuming the schedule begins with (2,3) and that job 1 will be processed as a single batch (Algorithm C below).

3c. Compare the flow times for the policies computed in Steps 3a and 3b. The one with the smallest flow time is the optimal policy.

If $n = 3$ and $\lambda_1 \geq 2\lambda_2$, process all three jobs, in order, singly. If $n = 3$ and $\lambda_1 < 2\lambda_2$, the remaining schedule should be (1,2)(3). If $n = 2$, process the jobs singly.

Algorithm C: Compute the optimal policy when job 1 is to be processed singly

Do while there are more than three remaining jobs, $n > 3$.

1. If $\lambda_1 \geq (n-1)\lambda_2$, the schedule begins with (1). Compute the optimal policy for the remaining $n - 1$ jobs using Algorithm B. Otherwise, go to Step 2.
2. If

$$\frac{1}{\lambda_2} + \frac{1}{\lambda_3} - \frac{1}{\lambda_2 + \lambda_3} \geq \frac{2}{\lambda_1},$$

the schedule begins with (1)(2,3). Compute the optimal schedule for the remaining $n - 3$ jobs assuming using Algorithm B. Otherwise, the schedule begins with (2,3). Compute the optimal policy for the remaining $n - 2$ jobs (including job 1) assuming job 1 will be processed as a single batch, using Algorithm C.

If $n \leq 3$, process any remaining jobs singly in decreasing order of λ_i.

https://doi.org/10.1017/S0269964801154033 Published online by Cambridge University Press

Acknowledgment

The authors appreciate the referee's careful reading and minor corrections.

References

1. Ahmadi, J.H., Ahmadi, R.R., Dasu, S., & Tang, C.S. (1992). Batching and scheduling jobs on batch and discrete processors. *Operations Research* 40: 750–763.
2. Albers, S. & Brucker, P. (1993). The complexity of one-machine batching problems. *Discrete Applied Mathematics* 47: 87–107.
3. Brucker, P. (1991). Scheduling problems in connection with flexible production systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1778–1783.
4. Chandru, V., Lee, C.Y., & Uzsoy, R. (1993). Minimizing total completion time on batch processing machines. *International Journal of Production Research* 31: 2097–2121.
5. Chandru, V., Lee, C.Y., & Uzsoy, R. (1993). Minimizing total completion time on a batch processing machine with families. *Operations Research Letters* 13: 61–65.
6. Cheng, T.C.E. & Kovalyov, M.Y. (2000). Parallel machine batching and scheduling with deadlines. *Journal of Scheduling* 3: 109–123.
7. Coffman, E., Jr., Yannakakis, M., Magazine, M., & Santos, C. (1990). Batch sizing and sequencing on a single machine. *Annals of Operations Research* 26: 135–147.
8. Deb, R.K. & Serfozo, R.F. (1973). Optimal control of batch service queues. *Advances in Applied Probability* 5: 340–361.
9. Dobson, G., Karmarkar, U.S., & Rummel, J.L. (1987). Batching to minimize flow times on one machine. *Management Science* 33: 784–799.
10. Glassey, C.R. & Weng, W.W. (1991). Dynamic batching heuristics for simultaneous processing. *IEEE Transactions on Semiconductor Manufacturing* 4: 77–82.
11. Hochbaum, D. & Landy, D. (1994). Scheduling with batching: Minimizing the weighted number of tardy jobs. *Operations Research Letters* 16: 79–86.
12. Hochbaum, D. & Landy, D. (1997). Scheduling semiconductor burn-in operations to minimize total flowtime. *Operations Research* 45: 874–885.
13. Ikura, Y. & Gimple, M. (1986). Scheduling algorithms for a single batch processing machine. *Operations Research Letters* 5: 61–65.
14. Nadder, D. & Santos, C. (1988). One-pass batching algorithms for the one machine scheduling problem. *Discrete Applied Mathematics* 21: 133–145.
15. Potts, C.N. & Kovalyov, M.Y. (2000). Scheduling with batching. A review. *European Journal of Operational Research* 120: 228–249.
16. Shallcross, D. (1992). A polynomial algorithm for a one machine batching problem. *Operations Research Letters* 11: 213–218.
17. Sung, C.S. & Choung, Y.I. (2000). Minimizing makespan on a single burn-in oven in semiconductor manufacturing. *European Journal of Operational Research* 120: 559–574.
18. Wagelmans, A.P.M. & Gerodimos, A.E. (2000). Improved dynamic programs for some batching problems involving the maximum lateness criterion. *Operations Research Letters* 27: 109–118.