International Journal of Management and Marketing Research
Vol. 8, No. 2, 2015, pp. 19-35
ISSN: 1931-0269 (print)
ISSN: 2157-0698 (online)

www.theIBFR.com

DETERMINING OPTIMAL FLOW-TIME SCHEDULES FOR THE MULTIPLE-PRODUCT BATCH-FLOW PROBLEM

Paul Schikora, Indiana State University
Andrew Manikas, University of Louisville
Michael Godfrey, University of Wisconsin Oshkosh

ABSTRACT

We explore the problem of batch flow production scheduling on a single machine with deterministic demand and arrivals over a finite horizon. The objective of the production system is to minimize total flow-time over the horizon to reduce in-process inventory levels and to enable a company to compete on reduced lead-times. Prior research has established optimal single job batch quantities. However, with multiple jobs on the shop floor, a job may incur wait time, thus the optimal local batch size for a given job may not result in global minimization of the total flow-time over all jobs. Our algorithm provides optimal results for batching with different products in a capacitated production environment. Numerous recommendations for further research are presented.

JEL: M11

KEYWORDS: Scheduling, Single Machine, Batch, Flow-Time, Lead-Time

INTRODUCTION

Past research has demonstrated the impact of production-batching (or lot-sizing) decisions have on time-related performance measures when demand is deterministic, but batch sizes are flexible. One area of this research looks at batching jobs together for processing (e.g., Baptiste, 2000; Coffman, Yannakakis, Magazine, & Santos, 1990; Logendran, Carson, & Hanson, 2005; Chen, Du, & Huang, 2011; Logendran, deSzoeke, & Barnard, 2006, Ng, Cheng, & Kovalyov, 2003; Van Der Zee, 2007). A second, related, area of research looks at breaking a job into batches, i.e., batching products on a single machine along with the resultant effect on flow-times. Santos and Magazine (1985), Dobson, Karmarkar, and Rummel (1987), and Naddef and Santos (1988) studied the subject of batching to minimize flow-times on a single machine. Potts and Van Wassenhove (1992) used the term “lot-sizing” to refer to the decision on how to break a given job into smaller batches. A sub-topic of each of these papers involved determining and scheduling production batches when completed products are not available for movement until all items in a batch are complete. This situation was referred to as the batch-flow (BF) problem. All of these papers showed that the single-product batch-flow problem could be solved optimally.

Later, Shallcross (1992) presented a more efficient algorithm for solving the single-product batch-flow problem. Then, Mosheiov, Oron, and Ritov (2005) presented an integer solution to the single-product batch-flow problem. Yang (2009) extended the single-product case to include learning effects using a forward dynamic programming algorithm. Bukchin, Tzur, and Jaffe (2002) extended the single-product case for a two-machine environment. Dobson et al. (1987) explored sequencing batches to minimize flow-time on a single machine. They investigated three scenarios: 1) the item-flow problem where transfer batches may be of size one only, 2) a batch-flow problem for a single product where transfer batches are the same size as processing batches, and 3) a batch-flow problem for multiple products with processing and transfer batches the same size. They assumed that all setups performed before processing parts are independent of the prior part type processed (sequence-independent). They used a simple index rule for the item-flow problem. Because transfer b atches of size one are an inefficient use of resources, the authors looked at two variations of batch-flow problems where transfer batches were the same size as processing batches were. For the single product batch-flow problem, they assumed a setup time before each batch and found the optimal batch quantities to process. This single product formulation, although not very useful in realistic multi-product production environments, did allow insights into batch sizing and sequencing to be used for the multiple-product case. Finally, Dobson et al. (1987) solved the optimal number of batches and the quantities in those batches. They suggested using heuristics to make local batch improvements, but did not remove batches explicitly from the production sequence to reduce the total flow-time for all jobs.

The previous research has been limited in that the more difficult multiple-product case (with jobs consisting of different products with different unit processing and setup times) has not been solved optimally in any of the papers, though heuristics have been presented (e.g., Dobson et al., 1987). In addition, the previous research assumed that all jobs would be available for processing at time zero. Although providing essential insight into the batch-flow problem, the previous research left unexplored the more realistic issue of multiple jobs with non-zero release dates (or times within a day) over a finite time horizon. When analyzing production capacity over a finite horizon, applying the scheduling formulas from the earlier papers may result in wait times for arriving jobs. These wait times, in turn, would increase total flow-time over all of the jobs; therefore, different scheduling methods must be not applied.

We investigate how to achieve optimal flow-time schedules in this complex environment. We extend and expand upon the prior work of Dobson et al. (1987) by looking at the most difficult, and realistic, scenario – the multiple-product batch flow problem. We illustrate how scheduling issues may result when using the optimal batch sizes calculated in prior research. Those batches are optimal only if no subsequent jobs are ready to be processed until after the first job is finished completely. However, if jobs are ready and waiting, we want to provide guidance on how to determine whether reducing the number of batches (effectively by removing setups on earlier jobs) could reduce wait time of later jobs, thus resulting in global flow-time reduction. To provide fuller coverage of potential scenarios that a scheduler may encounter, we show how to process batches of identical size, discrete batch quantities, and jobs where batches are not allowed to be commingled (e.g., government materials). We examine scenarios where all jobs are released at time zero (as in Dobson et al. (1987) and with spaced inter-arrival times. We also extend the commingled batches in Dobson et al. (1987) to solve the situation where products must be kept segregated throughout processing. We develop an algorithm to sequence work optimally under all of these scenarios. We simulate diverse production environments to demonstrate that our proposed method does, indeed, outperform the prior methods presented in the literature.

The use of different scheduling methods is an important area to investigate further because inventory and throughput are related to scheduling decisions. Little's Law states that flow-time equals inventory divided by throughput. If we can reduce flow-time over the horizon, we are able either to lower average inventory on hand or to increase throughput (parts produced by time), or both. Having inventory in the form of raw materials or work in process is money tied up that needs to be borrowed. Alternatively, inventory incurs an opportunity cost. Throughput is the rate that the system makes money. If throughput were increased, we would be able to produce value-added products more quickly; thus, we could sell them sooner and be better off financially according to the time value of money. Being able to use the same machine to process all products required, but more quickly, could reduce money tied up in inventory, and increase customer service levels and revenue. The remainder of this paper is organized as follows. The next section provides a literature review of the batch-flow problem structure and the results of prior research for the single-machine case. Next, we present the data and the methodology used to study the specific batch-flow problem formulation examined in our paper. The subsequent section discusses the results from implementing our proposed algorithm. The paper closes with concluding comments.

LITERATURE REVIEW

The issue examined in our paper is a single-machine production loading and scheduling problem with deterministic system parameters. The single-machine problem is well researched and best presented by Dobson et al. (1987). In their model, they assumed that a known quantity of items is available for processing at a machine. This quantity is to be divided into batches for producing and transferring the items from the machine. After a batch is completed, it is removed from the machine and available for movement. The next batch then can be started. In this formulation, production batches and transfer batches are the same size. Because of the time required to move the completed products off the machine, each batch incurs a transfer time to remove the batch and to transfer it to the next work center, during which the machine is non-productive and not available to process the next batch. Using the terminology of Cheng, Mukherjee, and Sarin (2013), this transfer and removal time would have the same effect on flow-time that a sublot-attached setup would have. One example in which process and transfer batches are equal is found in shops where movement of batches between machines might be accomplished using containers such as pallets or carts (Webster & Baker, 1995). A second example provided by Coffman et al. (1990) considered pick and place machines loaded with chips of various sizes that are inserted into circuit boards. Upon completion, each circuit board is loaded onto a cart by the operator of the pick and place machine. Then, the operator of the pick and place machine periodically stops production and moves the cart to a soldering machine. Our research focuses on the single machine problem, with *n* jobs in an over-capacitated situation for both the single-product and the multiple-product cases. The formulation used in this paper for the single-product case follows:

i = the number of batches of a product in a job.
d = the number of items in a job.
p = the unit processing time for each unit in a job.
s = the removal and transfer time for each batch of a job. Alternatively, this term may be used for setup time before running an individual batch.
M = an upper bound on the number of batches for a job.

If q_i = quantity of items processed in batch *i*, the batch-flow problem for the single-product case [BF1] can be formulated as:

$$\min \sum_{i=1}^{M} q_i \sum_{k=1}^{i} (s + pq_k) \tag{1}$$

$$\sum_{i=1}^{M} q_i = d \tag{2}$$

$$\text{Where } q_i \geq 0, i = 1,..,M$$

Dobson et al. (1987) showed that the optimal number of batches (k^*) to solve [BF1] is found using the following:

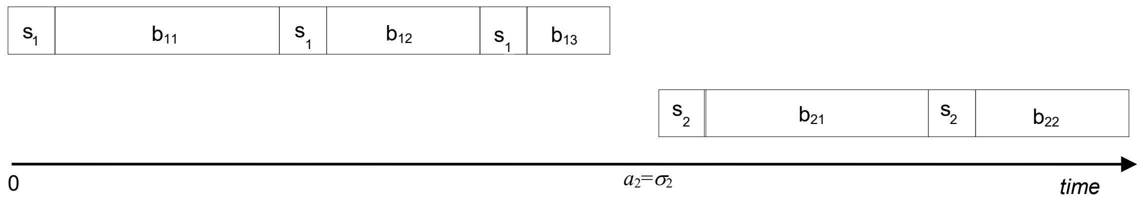
$$k^* = \left\lceil \sqrt{\frac{1}{4} + \frac{2dp}{s}} - \frac{1}{2} \right\rceil \tag{3}$$

The optimal size of each batch (q_i^*) is found using the modified formula in the second heuristic from Dobson et al. (1987). This modification is used such that the optimal number of batches from (3) could be used ($k' = k^*$), or any k' less than k^* could be used to produce fewer batches at the expense of increased job flow-time.

$$q_i^* = \frac{d_i}{k_i} + \frac{s}{p}\left(\frac{k_i - 1}{2}\right), i = 1,..,k' \tag{4}$$

In (4), d_i is the demand remaining after $i-1$ batches have been scheduled, $k_i = k' - (i-1)$ and the quantity for batch $i = \max(q_i^*, 0)$ to ensure positive quantity values. The batches per job are scheduled in non-increasing order. This ordering of batches may seem contrary to traditional scheduling methods based on the shortest processing time (SPT) concept, where the smallest processing time requirements are handled first. However, it does follow that when demand is heavy at a machine, larger batches should be produced first to decrease the size of the queue rapidly. As demand lessens, batch sizes could be decreased to concentrate more on individual item flow. These quantities produced from (4) also are supported by the findings of papers on the repetitive lots concept (Jacobs & Bragg, 1988) and lot streaming (Cheng et al., 2013; Kalir & Sarin, 2000; Ramasesh, Fu, Fong, & Hayya, 2000, Glass & Possani, 2011). Figure 1 illustrates the batching structure created by (3) and (4). The notation b_{ij} in Figure 1 represents the i^{th} batch in job j. For ease of understanding, we have used s_1 to represent the setup time before running each batch of product 1 and s_2 to represent the setup time before running each batch of product 2 However, the s terms may represent transfer times between operations in many businesses. The job arrival time is represented by a_j and the job processing start time is represented by σ_j.

Figure 1: Two-Job Schedule without Capacity Issues



This figure shows two jobs—Jobs 1 and 2. Job 1 is broken down into Batches 1, 2, and 3. Before each batch in Job 1 is run, a setup occurs. Job 2 is broken down into Batches 1 and 2. Before each batch in Job 2 is run, a setup occurs. As shown above, Batch 3 of Job 1 would be completed before Job 2 arrives. Hence, Job 2 would not incur wait time.

The batch flow-time is found by multiplying the quantity in each batch by the batch completion time. The total flow-time for a job is the sum of all batch flow-times using (1). Though Dobson et al. (1987) focused on choosing both the optimal number of batches and the batch sizes, (4) also could provide the optimal batch sizes given any value of k to which one might be restricted. If, for some reason, the scheduler were limited to breaking the processing requirement into a non-optimal number of batches (i.e., k'), the optimal batch sizes could be found by applying Equation (4) repetitively. For illustration, assume that job 1 arrives at time 0, with demand (d) of 21 units, batch setup time (s) of 25, and processing time per unit (p) of 5. From (3), the value of 2.44 would have a ceiling function applied, setting the optimal number of batches (k^*) to 3. We would set k' to our optimal k^* initially to compute the job flow-time. We then would use Equation (4) three times (i = 1, 2, 3) to calculate the appropriate batch quantities, as follows:

Step 1: Demand Remaining = 21 & Batch Quantity = 12.
Step 2: Demand Remaining = 9 & Batch Quantity = 7.

Step 3: Demand Remaining = 2 & Batch Quantity = 2.

Identical Batch Sizes

Dobson et al. (1987) also assumed that the scheduler is free to select arbitrary sizes for the different batches. Realistically, however, a scheduler might be limited to scheduling batches of as nearly identical size as possible. In that case, it can be shown that the optimal number of batches k^{**} is found by:

$$k^{**} = \left\lfloor \sqrt{\frac{2dp}{s}} \right\rfloor \tag{5}$$

The optimal batch sizes (q^{**}) all are set initially at:

$$q^{**} = \left\lfloor \frac{d}{k^{**}} \right\rfloor \tag{6}$$

The remaining *x* units would be assigned one each to the first *x* batches. For example, if *d* = 75, *p* = 1, and *s* = 2, then $k^{**} = 8, q^{**} = 9, x = 3$. The resultant batch sizes would be 10, 10, 10, 9, 9, 9, 9, and 9, with a total flow-time of 3,825.

Discrete Batch Sizes

Often, batch sizes calculated from (4) are non-integer. For discrete products, the quantities could be modified according to Mosheiov, Oron, and Ritov (2005) and Mosheiov and Oron (2008). For example, we could revisit our original example with job 1 with demand (*d*) of 21 units, setup time (*s*) of 25, and processing time per unit (*p*) of 4 (rather than 5). From (3), the optimal number of batches (k^*) would equal 3. We would set k' to our optimal k^* to compute the flow-time. Again, we would use (4) to calculate the appropriate continuous batch quantities. Next, the fractional units would be calculated as $q_i - \lfloor q_i \rfloor$. We then would use (7) to find the sum of fractional units to determine the number of batches (*B*) to round up to integer values.

$$B = \sum_{i=1}^{k^*} \left(q_i - \lfloor q_i \rfloor \right) \tag{7}$$

The *B* batches with the highest fractional portions would have a ceiling function applied. Next, the remaining ($k' - B$) batches would have a floor function applied. In the case where more than one batch had the same highest fractional portion, the ceiling function would be applied to the earliest of those batches. For our example, *B* =1 ([13.25 – 13] + [7 – 7] + [.75 – 0]). Batch 3 has the highest fractional portion of .75, so a ceiling function would be applied to make it a quantity of 1. The remaining two batches (1 & 2) would have a floor function applied to their quantities. The integer batches that produce the optimal flow-time for this job would be 13, 7, and 1 (as shown in Table 1). Note: For integer batches, it is theoretically possible to have a tie with other batch quantity values for minimum flow-time.

Table 1: Converting Fractional Batch Quantities to Integer Values

i	Demand Remaining	Batch Quantity q_i	Fractional Value	Integer q_i
1	21	13.25	0.25	13
2	7.75	7	0	7
3	0.75	0.75	0.75	1

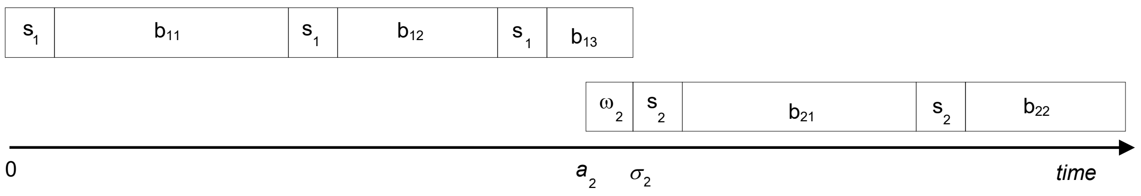
This table shows how to convert fractional batch quantities to integer values. We use the demand remaining in Equation (4) to calculate the continuous batch quantities. Next, we determine the fractional value for each batch. After that, we use Equation (7) to determine the number of batches to round up to integer values. Given that B = 1, we search for the one batch with the highest fractional value (Batch 1) and round its quantity up to an integer value. Then, we round down the quantities for the remaining batches with fractional values.

DATA AND METHODOLOGY

Problem Formulation

Dobson et al. (1987) provided many important results, including optimal formulas for the single-product batch-flow problem. However, they did not address how production capacity limits could affect multiple-product batching decisions and the resultant flow-time over a finite horizon. Also, they assumed that items arrive in groups for processing (hereafter, these groups are referred to as "jobs"), with the first job ready at time zero and the remaining jobs ready at known arrival points in the future. Finally, they did not provide guidance on how to sequence jobs when multiple jobs are ready to be processed at the same time. We fill in these gaps with our proposed model. To demonstrate the impact of capacity constraints, consider the two-job example in Figure 2, where each job is represented by a horizontal bar, with the length of the bar representing the processing time requirement of the job. Clearly, if all the jobs could be batched individually and processed completely before the next job arrival, then (3) would suffice to calculate the number of batches per job. However, consider when the processing time requirement of all the jobs approaches capacity over the horizon (or a portion of it). In that case, breaking up the jobs into batches, with the additional setup time required for each batch, may induce wait times for later jobs. For example, assume that job 2 arrives before the last batch of job 1 finishes.

Figure 2: Two-Job Schedule—Job 2 Arrives before Job 1 Finishes



This figure shows two jobs—Jobs 1 and 2. Job 1 is broken down into Batches 1, 2, and 3. Before each batch in Job 1 is run, a setup occurs. Job 2 is broken down into Batches 1 and 2. Before each batch in Job 2 is run, a setup occurs. As shown above, Batch 3 of Job 1 would not be completed before Job 2 arrives. Hence, Job 2 would incur wait time.

Because waiting time (denoted ω_j for job j) would increase the flow-time for a job, applying Equation (3) to determine the number of batches for each job in isolation may not result in minimum total flow-time over all of the jobs in the schedule. In that case, another method must be found to minimize total flow-time across all jobs. In this problem, multiple jobs are scheduled to arrive over a finite horizon at a single machine for processing. Each job contains multiple identical items, and different jobs may contain different products with unique processing and transfer times. Processing time requirements for the jobs are approaching, or exceeding, capacity over the horizon. The jobs are to be processed in a batch-flow method, i.e., no items in a batch are ready for movement until the last item in the batch is complete. The scheduler is free to choose the number and the size of batches for each job, and the objective is to minimize total flow-time over all jobs over the time horizon. Other assumptions of the model include the following: Job and machine characteristics are not alterable in the short term (i.e., there is no way to increase the machine capacities or to reduce product setup times, etc.). For notation, time zero is indexed at the arrival of the first job. Let there be *n* jobs to schedule for processing. Let $j = 1, \ldots, n$ index the jobs in the order of their arrival. Let k_j be the number of batches that job *j* is broken into, and $i = 1,..,k_j$ index the batches for job *j* in processing order. Also, let q_{ij} = the quantity processed in the i^{th} batch of job *j*, σ_{ij} = the start time of the i^{th} batch of job *j*, and τ_j = the completion time of job *j*. Assume that the following are known and fixed:

d_j = the number of units in job *j*.
p_j = the unit processing time for each unit of job *j*.
s_j = the transfer time (or setup time) of a batch of job *j*.
a_j = the arrival time of job *j*, measured from time zero.

The multiple-product problem now can be written as:

$$\min \sum_{j=1}^{n} \sum_{i=1}^{k_j} q_{ij} * \left[\sigma_{1j} - a_j + \sum_{m=1}^{i} \left[\left(q_{mj} * p_j \right) + s_j \right] \right] \tag{8}$$

s. t.

$$\sum_{i=1}^{k_j} q_{ij} = d_j \tag{9}$$

$$\tau_j = \sigma_{1j} + \left(d_j * p_j \right) + \left(k_j * s_j \right) \tag{10}$$

$$\sigma_{11} = 0 \tag{11}$$

$$\sigma_{1j} = \max[\tau_{j-1}, a_j], \quad j > 1 \tag{12}$$

$$\sigma_{ij} = \sigma_{1j} + \sum_{m=1}^{k_j - 1} \left[\left(q_{mj} * p_j \right) + s_j \right], \quad i > 1 \tag{13}$$

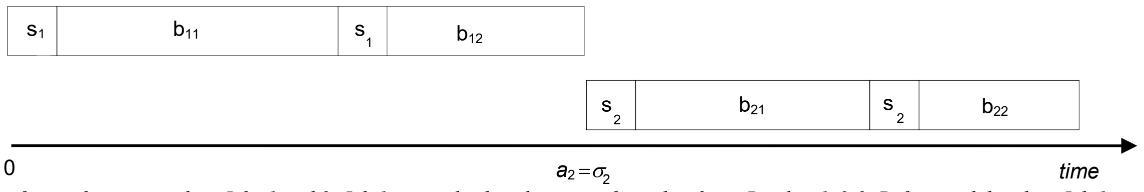
$$q_{ij} > 0, \quad \forall i, j \tag{14}$$

The first constraint ensures that all items are processed. The next four constraints ensure proper job start times based on processing precedence. The definition of τ_j is provided to simplify notation. The next section presents an algorithm developed to solve this problem. This algorithm is built upon the optimality properties of Equations (3) and (4) extended to the multiple-product problem. First, the logic behind the algorithm is explained, followed by a brief discussion of relaxation of the arrival/waiting constraint. Next, the algorithm is formally stated.

Multiple-Product Algorithm

Given that Equations (3) and (4) provide the optimal continuous batches for the single-product unconstrained problem, the results from Dobson et al. (1987) provide a good starting point for solving the multiple-product problem. First, consider the two-job problem illustrated in Figure 2. Given any start time for each job, the batches shown (assumed to be calculated from Equation (4)) would minimize each job's individual flow-time. Given that, it is obvious that the only way to reduce flow-time further would be to reduce the wait time for the second job ($\omega_2 = \sigma_2 - a_2$). Because job 2 cannot start until job 1 is completed, the only way to reduce the wait time for job 2 would be to reduce the makespan of job 1, thereby allowing job 2 to start earlier than currently scheduled. The current makespan of job 1 is $\left(d_1 * p_1\right) + 3s_1$. The processing time required by job 1 cannot be changed; however, the total transfer time associated with job 1 could be reduced by reducing the number of batches in job 1 (k_1). Each reduction in k_1 would reduce the total transfer time of job 1 and the makespan of job 1 by s_1. If a batch were removed from job 1, its flow-time normally would increase. Note: Because this is a discrete schedule, reducing the number of batches by one from the computed optimal value occasionally could result in no change in flow-time. For example, assume that job 1 in the two-job problem represented in Figure 2 is rescheduled into two batches, thus removing a batch from job 1. The revised schedule is depicted in Figure 3. Decreasing the number of batches in job 1 is not optimal for that job (i.e., removing a batch increases the flow-time for job 1). However, the resultant reduction in transfer time decreases the wait time for job 2, reducing the total flow-time of job 2, which in turn, may reduce the total flow-time over all jobs. In this example, job 2 can start immediately when it arrives (σ_2=a_2) after the re-batching of job 1.

Figure 3: Reduction of *k* for Job 1 Eliminates Wait Time for Job 2



This figure shows two jobs—Jobs 1 and 2. Job 1 now is broken down into fewer batches—Batches 1 & 2. Before each batch in Job 1 is run, a setup occurs. Job 2 still is broken down into Batches 1 and 2. Before each batch in Job 2 is run, a setup occurs. As shown above, Batch 2 of Job 1 now will be completed before Job 2 arrives. Hence, Job 2 will no longer incur wait time.

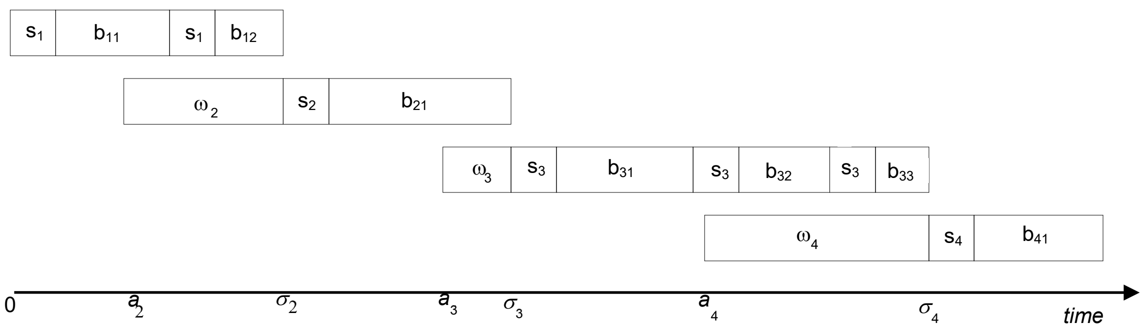
Generally, as a function of k_j, optimal job flow-time is convex, increasing left of k^* from Equation (3). However, ω_2 will decrease by $\delta_2 = \min\{\omega_2, s_1\}$, and the flow-time for job 2 will decrease by $\delta_2 * d_2$. If a reduction in k_1 results in a net increase in total flow-time for all jobs, there is no way to reduce total flow-time, and the current schedule is optimal. However, if a reduction in k_1 results in a net decrease in total flow-time over all jobs, this change should be made. In that case, job 1 would be rescheduled using (4) with $k_1 = k_1^* - 1$. If ω_2 were still positive, the rescheduling process would repeat as long as it results in a decrease in total flow-time, setting $k_1 = k_1^* - y$, where *y* is the number of batches removed from job 1. When $\omega_2 = 0, k_1 = 1$, or a reduction in k_1 results in an increase in total flow-time, the process stops, and the current schedule is optimal. The same logic could be extended to problems with more than two jobs. Figure 4 shows an example four-job problem where the wait time of jobs 2, 3, and 4 may be reduced by removing a batch from job 1.

For further illustration, assume a schedule with two jobs. Job 1 has $d = 200, p = 25, s = 100$ and arrival time by default set at $t = 0$. Job 2 has $d = 100$, $p = 1$, $s = 100$, arrival time $t = 2{,}000$. Using (3) and (4), we calculate the optimal number of batches and their respective quantities. In this example, $k_1^* = 10$, $k_2^* = 1$, $\sigma_2 = 5{,}000$, and $\omega_2 = 3{,}000$. From this, we can produce the data in Table 3, shown in Figure 5. The two lemmas below are demonstrated in Figure 5.

Lemma 1: Flow-time for downstream jobs may decrease linearly if wait time is reduced by eliminating batches from an upstream job. Eliminating y_j batches from the calculated k^* in Equation (3) for an upstream job *j* can decrease the flow-time of downstream jobs. Let this reduction of flow-time for downstream jobs from job *j* be represented by r_j^y, with an upper limit of:

$$\hat{r}_j^y = \min(w_{j+1}, y_j * s_j) * \sum_{m=j+1}^{n} d_m \tag{15}$$

Figure 4: Representative Four-Job Schedule



This figure shows four jobs broken into batches deemed optimal given prior research methods. However, as scheduled, Jobs 2, 3, and 4 incur wait time, which may increase the total flow-time. We propose that reducing the number of batches (setups) in upstream (earlier) jobs may decrease the total flow-time over all jobs.

The actual value of r_j^y depends on the structure of any specific problem and can be found from the resultant batch schedule for any value of y. As a function of y, r_j^y is stepwise linear, non-decreasing from y = 1 to k_j^* - 1. Its value would be calculated for increasing values of y_j (# of batches removed from job j). Removing a batch from an upstream job j may remove min$\{s_j, \omega_{j+1}\}$ from downstream jobs. Flow-time reduction for downstream jobs is the wait time removed per job multiplied by the demand (number of units) for that job. As long as there is remaining wait time for job j + 1, removing more batches from job j (increasing y_j) can reduce flow-time of the downstream jobs. The two-job example presented above is used to show this flow-time reduction of a downstream job in Table 2 and Figure 5, where the x-axis is the number of batches removed (y_1) from the $k_1{}^*$ calculated in (3) for job 1. The line labeled "Incremental r" on the graph in Figure 5 shows the incremental reduction in flow-time for job 2 from removing the y^{th} batch from job 1 ($r_1^y - r_1^{y-1}$). This equals zero for y = 0 and any value of y for which there is no reduction in ω2 . However, in this example, job 2 still would incur waiting time even if we removed all nine batches possible (recall that $k_1^* = 10$) from job 1.

Table 2: Batch Scheduling Results from Example Problem

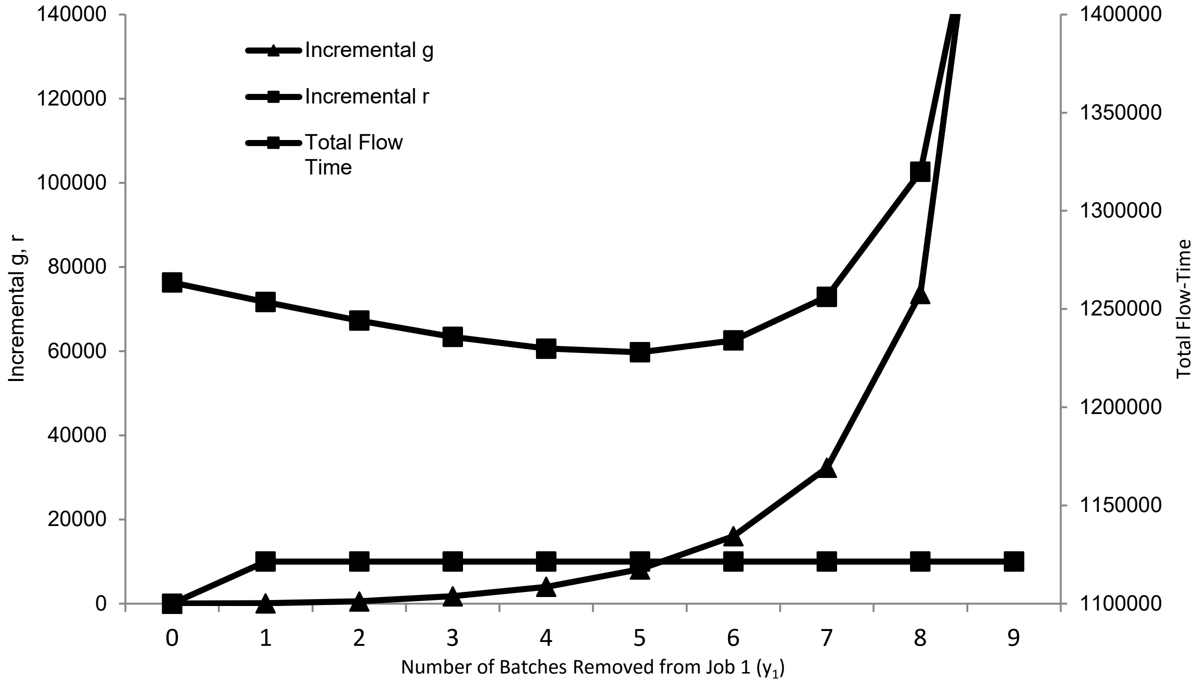
y	k_1	Job 1 Total Flow	Job 2 Total Flow	Job Set Total Flow	g_1^y	r_1^y	$g_1^y - g_1^{y-1}$	$r_1^y - r_1^{y-1}$
0	10	643,500	620,000	1,263,500	0	0	0	0
1	9	643,575	610,000	1,253,575	75	10,000	75	10,000
2	8	644,100	600,000	1,244,100	600	20,000	525	10,000
3	7	645,850	590,000	1,235,850	2,350	30,000	1,750	10,000
4	6	649,850	580,000	1,229,850	6,350	40,000	4,000	10,000
5	5	658,000	570,000	1,228,000	14,500	50,000	8,150	10,000
6	4	674,000	560,000	1,234,000	30,500	60,000	16,000	10,000
7	3	706,275	550,000	1,256,275	62,775	70,000	32,275	10,000
8	2	779,900	540,000	1,319,900	136,400	80,000	73,625	10,000
9	1	1,020,000	530,000	1,550,000	376,500	90,000	240,100	10,000

This table shows a numerical example with two jobs, each job's local (individual) flow-time, and total flow-time over all jobs after removing y setups from job 1. The cumulative increase in the local flow-time of job 1 due to removing setups (i.e., decreasing the number of batches from the optimal) is shown as g. The corresponding cumulative reduction in local flow-time for all downstream batches (job 2) is shown as r. The last two columns show the incremental effect on the g and r terms due to removing one more setup (batch) from job 1.

Lemma 2: Flow-time for a given job increases geometrically with the number of batches eliminated from the optimal k for that job as determined with (3). Removing batches for job j increases the flow-time for job j by delaying the completion of the original quantities of the $k_j{}^*$ - y batches. Let z_{ij}^y denote the incremental batch quantity for batch i on job j after reducing the number of batches by y (i.e., z_{ij}^y is the quantity increase to batch i of job j as the total demand needs to be satisfied using fewer batches (k_j^y) than the optimal(k_j^*)). Let $k_j^y = k_j^* - y$, and q_{ij} denote the original batch quantities for the k_j^* batches as determined by (3) and (4). For job j, the increase in flow-time (g_j^y) is given by Equation (16). Equation (16) follows from algebraic manipulation of the flow-times with k_j^y batches minus the flow-time with k_j^* batches.

$$g_j^y = \sum_{i=1}^{k_j^y}\left[q_{ij} * p_j * \sum_{x=1}^{i} z_{xj}^y\right] - \sum_{i=k_j^y+1}^{k_j^*}\left[q_{ij} * \sum_{x=1}^{i}\left[\left(q_{xj} * p_j\right) + s_j\right]\right] + \sum_{i=1}^{k_j^y}\left[z_{ij}^y * \sum_{x=1}^{i}\left[\left(\left(z_{xj}^y + q_{xj}\right) * p_j\right) + s_j\right]\right] \quad (16)$$

Figure 5: Net Change in Flow-Time as Batches Are Removed



This figure shows the individual effects of local flow-time for job 1 increasing (incremental g), flow-time of downstream jobs decreasing (incremental r), and the combined total flow-time net result. As long as the slope of the total flow-time is negative, removing a batch (and its setup) from an upstream job results in a net reduction in total flow-time over all jobs.

From Equation (4), batch quantities increase for remaining batches of a given job when its k is reduced. In addition, as the number of batches removed (y) increases, z_{ij}^y increases at an increasing rate for the remaining batches. The first term in Equation (16) is the incremental additional flow-time for the first k_j^y batch's original quantities caused by waiting for the additional processing time of their increased batch sizes (z increments). The second term is the removal of flow-time from the original last y batches because we are no longer using k_j^* batches, but rather are using $k_j^y = k_j^* - y$ batches. The final term adds back the flow-time for the additional z_{ij}^y units to be processed in each of the new k_j^y batches. The incremental effect of Equation (16) is shown in Figure 5. The line labeled "Incremental g" shows the incremental increase in the flow-time of job 1 by removing the y^{th} batch from job 1 ($g_1^y - g_1^{y-1}$). By comparing both incremental lines, we can see how large y should be to minimize total flow-time. As long as the incremental r is greater than the incremental g for a particular value of y, removing the y^{th} batch will reduce total flow-time for the entire schedule. Similar logic to that shown above for $n = 2$ can be applied to scheduling a multiple-product problem with $n > 2$. Consider the representative batch schedule shown in Figure 4 ($n = 4$), and assume the batches shown are optimal for each individual job. Because there is wait time for jobs 2 through 4, reductions in k_1, k_2 or k_3 will decrease the flow-time of the respective downstream job(s). Given this starting schedule, reducing k_1 will provide the greatest decrease in downstream flow-time. As k_1 is decreased, it eventually will stop being the best candidate for reduction as ω_2 is eliminated or the increase in flow-time for job 1 becomes so large that another k_j becomes the better choice for reduction. As long as at least one ω_j remains positive, there is a potential for benefit from reducing an appropriate k_{j-1}. For any $\omega_j = 0$, there will be no possible benefit from reducing k_{j-1}.

Proposition 1: After k_j^y is found for any job j such that the net total flow-time of all jobs does not decrease, further reductions in k_j^y can only further increase total flow-time.

Proof: Let $k_j^{y'}$ be the first number of batches evaluated (smallest y) where the total flow-time of all jobs does not decrease. This means the incremental increase in g_j^y is no longer fully offset by the incremental increase in r_j^y. From Lemma 1, r_j^y increases linearly, and from Lemma 2, g_j^y increases geometrically as k_j^y is decreased from k_j^*. Therefore, no number of batches less than $k_j^{y'}$ can further decrease total flow-time. By evaluating these net total flow-time trade-offs, an improved final solution could be found from the original one shown in Figure 4. Proposition 1 allows us to specify our algorithm. A logical method of evaluating and choosing reductions in k_j is presented in §4.3.

Discussion on Relaxation of Arrival Times

In the current model, jobs are processed in the order of their arrival. However, which job should be selected if two jobs are waiting at the completion of the current job, or arrive at the same time?

Proposition 2: If two jobs are ready at the same time, then if $d_2(k_1 s_1 + p_1 d_1) > d_1(k_2 s_2 + p_2 d_2)$, schedule job 2 first. Otherwise, schedule job 1 first.

Proof: The proof follows from algebraic manipulation of the flow-times of the two cases:

$A = d_1(k_1 s_1 + p_1 d_1) + d_2(k_1 s_1 + p_1 d_1 + k_2 s_2 + p_2 d_2)$ when job 1 is processed before job 2, and $B = d_2(k_2 s_2 + p_2 d_2) + d_1(k_1 s_1 + p_1 d_1 + k_2 s_2 + p_2 d_2)$ when job 2 is processed before job 1. Comparing A with B and simplifying leaves the inequality $d_2(k_1 s_1 + p_1 d_1) > d_1(k_2 s_2 + p_2 d_2)$ to indicate that job 2 should proceed before job 1 to minimize total flow-time.

Similarly, for m jobs waiting to be processed on the machine, the job j with the highest value of $d_j(k_i s_i + p_i d_i) - d_i(k_j s_j + p_j d_j),\ i, j = 1,..,m,\ i \neq j$, is processed first to minimize the total flow-time equations over all jobs competing for the same capacity. Similarly, for the remaining m-1 jobs, the next highest value of $d_j(k_i s_i + p_i d_i) - d_i(k_j s_j + p_j d_j),\ i, j = 1,..,m-1,\ i \neq j$ is then processed, with repeated application of that equation until there is only one job left to sequence.

Algorithm Statement

Because total flow-time is a function of k_j (the only variable in the problem), a simple type of search method (i.e., marginal analysis) was chosen to drive the multiple-product algorithm. The starting point for the algorithm is the solution where the number of batches for each job (k_j) is set at the optimal level (k_j^*) determined from Equation (3) from Dobson et al. (1987) for a single job. Next, each job is analyzed (beginning with job 1), subtracting one batch at a time until there is no more reduction in total flow-time, there is no more wait time in the system, or k_j has been set to one batch. This algorithm is applied to the first n – 1 jobs. From Proposition 1, we can set a general upper bound on the number of iterations of the algorithm until the optimal solution is achieved.

$$U = \sum_{j=1}^{n-1} k_j^* \tag{17}$$

Algorithm Steps

```
Calculate $k_j^*$ and $q_{ij}^*$ for each of the $j$ = 1,…, $n$ jobs using (3) and (4)

Set $k_j = k_j^*$ for all $j$
Set $j = 1$
Set $r_j^0 = g_j^0 = 0$
For each job $j$,    $j$ = 1, …, $n$-1
                For $y$ = 1 to $k_j^* - 1$

Calculate batch sizes from (4) and resultant schedule

Calculate $g_j^y$ and $r_j^y$
                    If $g_j^y - g_j^{y-1} > r_j^y - r_j^{y-1}$ , then exit loop, 'no more reductions possible'
                    Set $k_j = k_j^y$
                Next $y$
        Next job $j$
```

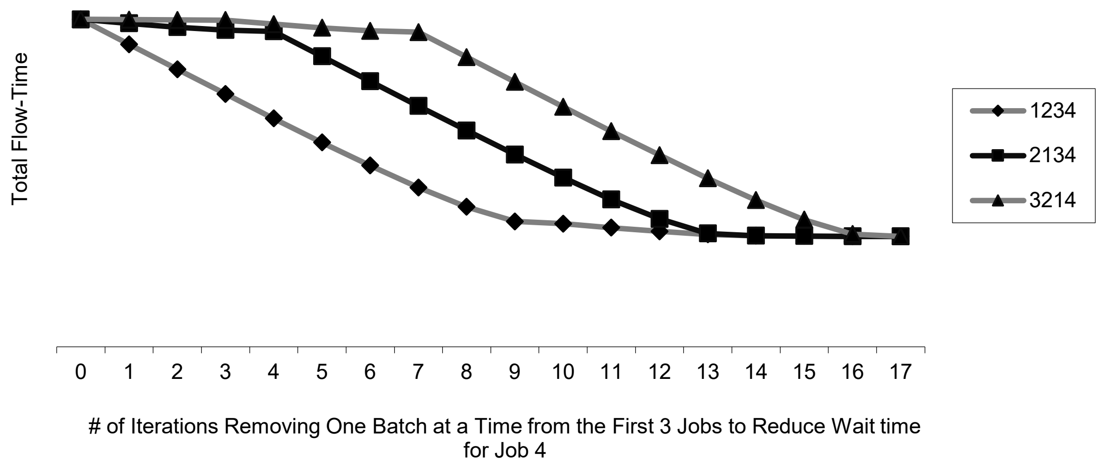
Although our proposed algorithm analyzes jobs in sequential order, this is not necessary to achieve the optimal flow-time in a fixed number of iterations (as bounded by Equation (17)). Figure 6 shows a four-job example, with the first three jobs ready to process at the same time and the fourth job released later (thus always scheduled last). The figure shows the effect on total flow-time by applying the algorithm to reduce setups on the first three jobs for three different sequences. The x-axis specifies the number of batches removed from the original $k_j^*, j = 1,...,n-1$, batches calculated using Equation (3). Although not drawn in the figure (for readability, only three sequences are shown), investigating all six possible sequences of the first three jobs for setup reduction has the same final flow-time effect. Therefore, the order in which jobs are investigated with our formulae does not affect attainment of the optimal flow-time. The next section describes the experimental design. This is followed by a section that re-examines the commingled batches all arriving at time 0, as done in Dobson et al. (1987). The final sections are our conclusions and recommendations for further research.

Experimental Design

To test the effectiveness of the multiple-product algorithm for the case where commingling of products is not allowed, the following experimental factors were considered:

Number of problems: 1,000
Number of jobs/products per problem: Randomly generated as U(2, 100)
Number of units in a job (demand): U(1, 1,000)
Unit processing time for each unit of a job: U(1, 50) minutes
Transfer time for a batch within a job: U(1, 100) minutes
Inter-arrival times between jobs: U(1, 1,000) minutes.

Figure 6: Flow-Time Reduction Per Iteration For Various Job Sequences (Four-Job Scenario)



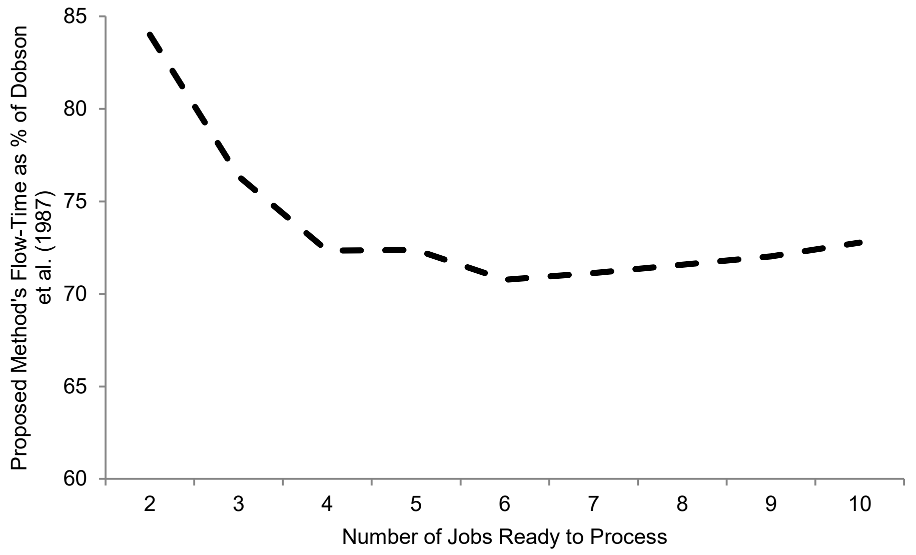
This figure shows an example with three jobs available for processing at the same time, with a fourth job arriving later. If when using Equations (15) and (16) we determine that removing a batch setup from an upstream job results in a net reduction in total flow-time, that setup should be removed. The three lines in the figure show the effect on total flow-time from removing all possible additional setups (beyond the required first setup) from each of the first three jobs in three sequences: Jobs 1-2-3-4, Jobs 2-1-3-4, and Jobs 3-2-1-4. For example, in the job sequence 1-2-3-4, all additional setups (beyond the first setup required for each job) would be removed from Jobs 1, 2, and 3. Each of these jobs would be processed with a single setup (batch) required for each job. Regardless of the sequence analyzed, minimum flow-time for the four jobs is achieved. The order in which jobs are examined for removing setups does not matter when using our algorithm.

We used a Visual Basic program to generate the 1,000 problems. For each problem, the algorithm was applied and the total flow-time was determined. Because, Dobson et al. (1987) assumed commingled batches and no inter-arrival times, we are demonstrating here just that our method achieves the optimal global flow-time quickly. In Section 6, we compare our proposed algorithm with Dobson et al. (1987) under scenarios mimicking what they used.

RESULTS AND DISCUSSION

As noted in our experimental setup, we allowed commingling (intermixing) of batches from different products, and sequenced these batches according to Conway et al. (1967). We used our new algorithm to calculate the *g* and *r* parameters to find batch setups that could be removed (increasing the local flow-time for a particular job, but resulting in a global reduction in flow-time). Figure 7 shows the results of 10,000 simulated scenarios. The *x*-axis shows the number of products to be produced and the y-axis shows what percent of Dobson et al.'s (1987) method flow-time our new method achieved. As can be seen, our method achieved a better than 26% reduction in flow-time (averaged over all 10,000 test cases) from the original algorithm proposed by Dobson et al. (1987). When there were only a few jobs to investigate for potential setup reductions, sometimes there were no improvements that could be made (i.e., with two products, a few scenarios did not have an opportunity for our method to improve upon Dobson et al. (1987)). However, with more jobs, there were more opportunities to apply our new algorithm to reduce total flow-time substantially.

Figure 7: Proposed Method's Flow-Time As % of Solution Using Dobson Et al. (1987) Algorithm



This figure shows the total flow-time our method achieved compared to that presented as optimal for the single-product case in Dobson et al. (1987). Given that Dobson et al. (1987) computed batch sizes for a single job only, their algorithm cannot be optimal when more than one job is available on the shop floor. We simulated 10,000 production schedules varying from 2 to 10 jobs available simultaneously. With only 2 or 3 jobs in a production schedule, there was less wait time to eliminate using our method. However, as more jobs (4 or more) were available, removing setups from upstream (earlier) jobs resulted in total flow-time reductions. As the figure shows, our total flow-time was less than 75% of the flow-time that would have resulted from applying the algorithm presented originally by Dobson et al. (1987).

As can be seen, our proposed method achieved a better than 26% reduction in flow-time (averaged over all 10,000 test cases) from the original single-product algorithm proposed by Dobson et al. (1987). When there were only a few jobs to investigate for potential setup reductions, sometimes there were no improvements that could be made (i.e., with two products, a few scenarios did not have an opportunity for our method to improve upon Dobson et al. (1987)). However, with more jobs, there were more opportunities to apply our new algorithm to reduce total flow-time substantially.

CONLUDING COMMENTS

Our proposed method addresses the multiple-product batch scheduling problem from a global flow-time minimization perspective. Dobson et al.'s (1987) method for determining optimal batch sizes is applicable only if jobs are never waiting to be processed on the machine. If jobs are waiting, it may be that removing a setup (moving away from the optimal number of batches calculated by Dobson et al. (1987)) could decrease the flow-time for processing all jobs over the production horizon. We have shown how our method can handle jobs where batches may be commingled or where they must be segregated (e.g., Department of Defense jobs). We have provided guidance on how to modify the number of batches per job if identical batches need to be processed given resource constraints. We also have shown how to convert fractional batch quantities to discrete ones. Finally, we have provided a rule for deciding which job to begin processing, if multiple jobs are waiting to minimize total flow-time. Our new method provides the minimum flow-time globally, by recognizing that local increases in flow-time may be more than offset by flow-time reductions for jobs processed later. Similar to the Dobson et al. (1987) article that inspired it, our research provides interesting insight into the multiple-product problem, and the algorithm developed is an important initial contribution to the problem. However, as this was an introductory study, the model defined herein was restricted, and leaves open many avenues for future research into the topic. This section presents suggestions for such research. The current model has restrictions, that when relaxed, make for a tougher and more interesting problem. Some of those restrictions are discussed below.

Include Sequence-Dependent Changeover Times in Addition to Transfer Times - The current model assumes sequence-independent setup (transfer) times. In many processing environments, though, changing from one item to another requires the machine to undergo some configuration changes -- usually referred to as a changeover. With the current model restrictions, the addition of changeover times between products would have little impact on the system (merely extending the system wait times), and no impact on the solution procedure. However, if the scheduler were allowed to choose from waiting jobs, changeover times would affect which job to choose. For example, if two jobs were waiting and one were the same item as the current job, no changeover time would be required for that job. This would give that job a flow-time advantage over the different type job, an advantage that would have to be weighed against other considerations (e.g., which job arrived first, which job is larger, etc.). The impact of this relaxation would vary based on what other changes are made to the model. Simchi-Levi and Berman (1991) investigated applying a traveling salesman algorithm for this problem.

Limit the Allowable Transfer Batch Size – Our model assumes that any transfer batch size is acceptable, allowing batch sizes as small as a single unit and as large as need be. In a practical application setting, that assumption probably would not hold. For example, if this were a pharmaceutical setting and the machine bottles pills, there probably would be a standard basic material handling platform (e.g., tray, bin, etc.) that could hold a specific number of bottles. Assuming the machine could automatically move that platform to a larger movement container (e.g., a pallet), a reasonable restriction on transfer batch size would be to allow batches only in multiples of the basic handling platform. In a large processing environment, even the ultimate movement container might limit the transfer batch size (e.g., batches no larger than a pallet).

This paper examined the issue of batch flow production scheduling on a single machine with deterministic demand and arrivals over a finite horizon. The objective of the model is to minimize total flow-time over the horizon. Because the problem appeared complex in mathematical form, a linear search algorithm was developed to solve this problem. It was demonstrated in Proposition 1 that due to the convex nature of the optimal total flow-time curve, the algorithm does, indeed, provide optimal results. A general upper bound on the number of steps required for the algorithm to find the optimal solution was presented. In Section 6, we demonstrated the robustness of our proposed method. Specifically, using a similar environment to that in Dobson et al. (1987), we showed a significant reduction in total flow-time over a wide range of jobs compared to prior results. Finally, numerous recommendations for further research were presented.

REFERENCES

Baptiste, P., 2000. Batching identical jobs. *Mathematical Methods of Operations Research*, 52 (3), 355-367.

Bukchin, J., Tzur, M., and Jaffe, M., 2002. Lot splitting to minimize average flow-time in a two-machine flow-shop. *IIE Transactions*, 34, 953-970.

Chen, H., Du, B., and Huang, G.Q., 2011. Scheduling a batch processing machine with non-identical job sizes: a clustering perspective. *International Journal of Production Research*, 49 (19), 5755-5778.

Cheng, M., Mukherjee, N.J., and Sarin, S.C., 2013. A review of lot streaming. *International Journal of Production Research*, 51 (23-24), 7023-7046.

Cheng, T.C.E, Kovalyov, M.Y., and Chakhlevich, K.N., 2004. Batching in a two-stage flowshop with dedicated machines in the second stage. *IIE Transactions*, 36 (1), 87-93.

Coffman, E.G., Yannakakis, M., Magazine, M.J., and Santos, C., 1990. Batch sizing and job sequencing on a single machine. *Annals of Operations Research* 26, 135-147.

Conway, R.W., Maxwell, W.L., and Miller, L.W., 1967. *Theory of Scheduling*. Reading, MA: Addison-Wesley.

Dobson, G., Karmarkar, U.S., and Rummel, J.L., 1987. Batching to minimize flow-times on one machine. *Management Science*, 33 (6), 784-799.

Glass, C.A. and Possani, E., 2011. Lot streaming multiple jobs in a flow shop. *International Journal of Production Research*, 49 (9), 2669-2681.

Jacobs, F.R. and Bragg, D.J., 1988. Repetitive lots: Flow-time reductions through sequencing and dynamic batch sizing. *Decision Sciences*, 19, 281-294.

Kalir, A.A. and Sarin, S.C., 2000. Evaluation of the potential benefits of lot streaming in flow-shop systems. *International Journal of Production Economics*, 66 (2), 131-142.

Logendran, R., Carson, S., and Hanson, E., 2005. Group scheduling in flexible flow shops. *International Journal of Production Economics*, 96 (2), 143-155.

Logendran, R., deSzoeke, P., and Barnard, F., 2006. Sequence-dependent group scheduling problems in flexible flow shops. *International Journal of Production* Economics, 102 (1), 66-86.

Mosheiov, G. and Oron, D., 2008. A single machine batch scheduling problem with bounded batch size. *European Journal of Operational Research*, 187, 1069-1079.

Mosheiov, G., Oron, D., and Ritov, R., 2005. Minimizing flow-time on a single machine with integer batch sizes. *Operations Research Letters*, 33, 497-501.

Naddef, D. and Santos, C., 1988. One-pass batching algorithms for the one-machine problem. *Discrete Applied Mathematics*, 21, 133-145.

Ng, C.T.D., Cheng, T.C.E., and Kovalyov, M.Y., 2003. Batch scheduling with controllable setup and processing times to minimize total completion time. T*he Journal of the Operational Research Society*, 54 (5), 499-506.

Potts, C.N. and Van Wassenhove, L.N., 1992. Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity. *The Journal of the Operational Research Society*, 43 (5), 395-406.

Ramasesh, R.V., Fu, H., Fong, D.K.H., and Hayya, J.C., 2000. Lot streaming in multistage production systems. *International Journal of Production Economics*, 66 (3), 199-211.

Santos, C. and Magazine, M.J., 1985. Batching in single operations manufacturing systems. *Operations Research Letters*, 4 (3), 99-103.

Shallcross, D.F., 1992. A polynomial algorithm for a one machine batching problem. *Operations Research Letters*, 11, 213-218.

Simchi-Levi, D. and Berman, O., 1991. Minimizing the total flow time of *n* jobs on a network. *IIE Transactions*, 23 (3), 236-244.

Van der Zee, D.J., 2007. Dynamic scheduling of batch-processing machines with non-identical product sizes. *International Journal of Production Research*, 45 (10), 2327-2349.

Webster, S. and Baker, K.R., 1995. Scheduling groups of jobs on a single machine. *Operations Research*, 43, 692-703.

Yang, W., 2009. A batching problem with learning effect considerations. *Asia-Pacific Journal of Operational Research*, 26 (2), 307-317.

BIOGRAPHY

Paul Schikora is Professor of Operations & Supply Chain Management in the Scott College of Business at Indiana State University. He earned his Ph.D. in Operations Management from Indiana University's Kelley School of Business. He earned his M.S. degree in Logistics Management from the Air Force Institute of Technology, and his B.S. degree from the Illinois Institute of Technology. His teaching interests are in operations management, with a focus on the application of technology to process analysis and improvement. His research interests include process simulation, quality management and improvement, manufacturing scheduling, and information systems management. Email: paul.schikora@indstate.edu

Dr. Manikas earned his B.S. in Computer Science and his MBA in Materials and Logistics Management from Michigan State University, and his Ph.D. from The Georgia Institute of Technology. Prior to that, he was an instructor for supply chain optimization courses for i2 Technologies and worked as a management consultant for KPMG Peat Marwick, CSC, and Deloitte Consulting. Dr. Manikas is an Assistant Professor in the Management Department at the University of Louisville. He is CIRM and CSCP through APICS, PMP through PMI, and a CPSM through ISM. Email: andrew.manikas@louisville.edu

Dr. Godfrey earned his B.S. in Operations Management and M.S. in Management Information Systems from Northern Illinois University, and his Ph.D. in Production & Operations Management from the University of Nebraska - Lincoln. He is A Full Professor of Supply Chain Management in the College of Business at the University of Wisconsin. He is a CFPIM through APICS and a CPSM through ISM. Email: godfrey@uwosh.edu