Computing 70, 167–180 (2003)
Digital Object Identifier (DOI) 10.1007/s00607-002-1467-8

Computing
Printed in Austria

Scheduling a Single Server in a Two-machine Flow Shop

T. C. E. Cheng, Kowloon, Hong Kong, and **M. Y. Kovalyov,** Minsk, Belarus

Received November 20, 2001; revised October 18, 2002
Published online: January 16, 2003

Abstract

We study the problem of scheduling a single server that processes n jobs in a two-machine flow shop environment. A machine dependent setup time is needed whenever the server switches from one machine to the other. The problem with a given job sequence is shown to be reducible to a single machine batching problem. This result enables several cases of the server scheduling problem to be solved in $O(n \log n)$ by known algorithms, namely, finding a schedule feasible with respect to a given set of deadlines, minimizing the maximum lateness and, if the job processing times are agreeable, minimizing the total completion time. Minimizing the total weighted completion time is shown to be NP-hard in the strong sense. Two pseudopolynomial dynamic programming algorithms are presented for minimizing the weighted number of late jobs. Minimizing the number of late jobs is proved to be NP-hard even if setup times are equal and there are two distinct due dates. This problem is solved in $O(n^3)$ time when all job processing times on the first machine are equal, and it is solved in $O(n^4)$ time when all processing times on the second machine are equal.

AMS Subject Classifications: 90B35.

Keywords: Scheduling, batching, algorithms, dynamic programming.

1. Introduction

The problem of scheduling a single server in a two-machine flow shop can be formulated as follows. There are n independent non-preemptive jobs to be scheduled for processing by a server on two sequential machines. The server can be a human operator or a robot. Each job j requires two operations $(1j)$ and $(2j)$, which are performed on machines 1 and 2, respectively. The processing time of job j on machine l, i.e., the duration of the operation (lj), is p_{lj}, $l = 1, 2$. For each job, the second operation cannot be started before the first operation is completed. A setup time s_l is needed before the first job is processed on machine l or when the server switches from machine $3 - l$ to machine l. Each processing operation and each setup operation must be performed by the server, which can only perform one operation at a time. In this case, a schedule can be described by the job sequences on machines 1 and 2 and their partitions into *batches*, where a batch is a maximal set of jobs scheduled contiguously on the same machine. Each batch on machine l is preceded by the setup time s_l.

Given a schedule, the job *completion times* C_j, $j = 1, \ldots, n$, are easily determined. The completion time of a job is the time when its processing is finished on machine 2. Moreover, for each job j, one can determine its *lateness* $L_j = C_j - d_j$, where d_j is the *due date* or *deadline* of job j, and *tardiness* $T_j = \max\{0, L_j\}$.

The problem is to find a schedule that is *feasible with respect to the deadlines* such that $C_j \leq d_j$, $j = 1, \ldots, n$, or which minimizes a performance criterion depending on the job completion times. We consider the following performance criteria:

maximum lateness $L_{\max} = \max\{L_j\}$;

maximum tardiness $T_{\max} = \max\{T_j\}$;

maximum cost $f_{\max} = \max\{f_j(C_j)\}$, where f_j, $j = 1, \ldots, n$, are non-decreasing *cost functions*;

(weighted) number of late jobs $\sum(w_j)U_j$, where $w_j > 0$ is the *weight* of job j indicating its relative importance, and $U_j = 0$ if job j is *early*($C_j \leq d_j$) and $U_j = 1$ if job j is *late* ($C_j > d_j$);

total (weighted) tardiness $\sum(w_j)T_j$;

total (weighted) completion time $\sum(w_j)C_j$.

Here and below each maximum or summation is assumed to be taken over all jobs unless stated otherwise. All numerical parameters and function values are assumed to be non-negative integers. All the above performance criteria are *regular*, i.e., they are non-decreasing in job completion times.

The model studied in this paper combines three scheduling aspects: flow shop scheduling, server scheduling and batch scheduling.

Prior results for classical flow shop scheduling models are presented by Conway, Maxwell and Miller [13], Baker [3], Coffman [11]. More recent results are reported by Lawler et al. [21], Brucker [5], Blazewicz et al. [4], Pinedo and Chao [23].

There exist results for server scheduling models. However, these previously studied models assume that each operation consists of two phases: the setup phase and the processing phase. The server is responsible for the setup phase only. Such models are studied by Koulamas [19], Kravchenko and Werner [18], Glass, Shafransky and Strusevich [15], Hall, Potts and Sriskandarajah [16], Cheng, Wang and Sriskandarajah [10], Brucker et al. [6] and Brucker, Knust and Wang [8].

Reviews of batch scheduling research are provided by Potts and Van Wassenhove [25], Webster and Baker [28], Allahverdi, Gupta and Aldowaisan [2], and Potts and Kovalyov [24].

The model studied in this paper arises wherever the processing of items on sequential machines requires the continuous presence of a server. In spite of the fact that solving corresponding problems implies finding partitions of the sequences of jobs into batches, this model differs from other popular batch scheduling models such as the family scheduling model, batch availability model, batch delivery

model, multi-operation job model and batching machine model, see Potts and Kovalyov [24]. We, however, show that there exist some relationships between this model and the batch availability model. In the batch availability model, all jobs assigned to the same batch complete together when the latest job of the batch finishes its processing.

We call a job sequence $(i_1, i_2, \ldots, i_n)$ partitioned into batches $B_1, \ldots, B_r$ in this order a *batch sequence* and denote it by $(B_1, \ldots, B_r)$. Thus, a schedule is completely characterized by the batch sequences on machines 1 and 2.

The standard job shift technique can be used to prove the following.

Lemma 1. *For the server scheduling problem to minimize any regular performance criterion, there exists an optimal schedule in which the batch sequences are identical on both machines.*

In the following, we consider only schedules satisfying Lemma 1. We call such schedules *batching schedules*. An example of a batching schedule for seven jobs is given in Fig. 1.

In Sect. 2, we show that, when the job sequence is fixed, the server scheduling problem with various performance criteria reduce to the single machine batching problem with the same performance criteria. This result allows us to establish that the following cases of the problem are solvable in $O(n \log n)$ time: finding a schedule feasible with respect to the deadlines, minimizing $L_{\max}$ and minimizing $\sum C_j$, if the job processing times are *agreeable*, i.e., the jobs can be re-indexed in the shortest processing time (SPT) order such that $p_{l1} \leq p_{l2} \leq \cdots \leq p_{ln}$, $l = 1, 2$. The algorithm for the feasibility problem can also be used to minimize the maximum cost in polynomial time. On the other hand, the minimization of $\sum w_j C_j$ is shown to be NP-hard in the strong sense. However, this problem is solvable in $O(n)$ time when the job sequence is fixed. The problem of minimizing $\sum w_j U_j$ is studied in Sect. 3. Two pseudopolynomial dynamic programming algorithms are derived for this problem. When all weights are equal, one of the algorithms runs in $O(n^3)$ time if all processing times p_{1j} are equal and the other algorithm runs in $O(n^4)$ time if all processing times p_{2j} are equal. If p_{1j} and p_{2j} are arbitrary, then minimizing $\sum U_j$ is proved to be NP-hard even if $s_1 = s_2$ and there are two distinct due dates.

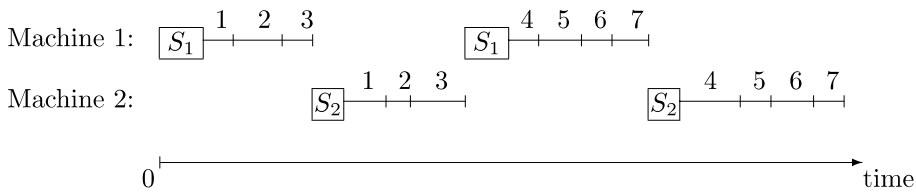

Fig. 1. Batching schedule $S = (B_1, B_2)$ with batches $B_1 = (1, 2, 3)$ and $B_2 = (4, 5, 6, 7)$

2. Given or Easy to Find Optimal Job Sequence

In this section, we assume that, for the specific case of the server scheduling problem under study, either an optimal job sequence can be efficiently found or a job sequence is given.

Let $(1,\ldots,n)$ be the given (optimal) job sequence. The problem reduces to partitioning the sequence into subsequences corresponding to batches.

Consider a batch sequence $S=(B_1,\ldots,B_r)$, where $B_k=(a_{k-1}+1,a_{k-1}+2,\ \ldots,\ a_k)$ is the batch sequenced k-th, $k=1,\ldots,r$, $a_0=0$. For this sequence, the completion time of job j assigned to batch B_k can be calculated as follows:

$$C_j(S)=k(s_1+s_2)+\sum_{i=1}^{a_k}p_{1i}+\sum_{i=1}^{j}p_{2i}. \tag{1}$$

We now formulate a single machine batching problem with a given job sequence. There are n independent non-preemptive jobs with processing times p_j and due dates (or deadlines) $\hat{d}_j$, $j=1,\ldots,n$. The job sequence $(1,\ldots,n)$ has to be partitioned into batches, each preceded by a setup time s, so as to minimize a performance criterion depending on the job completion times $\hat{C}_j$, $j=1,\ldots,n$. Here job completion times are defined according to the batch availability model such that $\hat{C}_j=\hat{C}(B)$ for all jobs j in batch B, where $\hat{C}(B)$ is the time when the machine finishes processing all jobs in batch B.

Define $\hat{L}_j(S)=\hat{C}_j(S)-\hat{d}_j$, $\hat{L}_{\max}=\max\{\hat{L}_j\}$, $\hat{T}_j(S)=\max\{0,\hat{L}_j(S)\}$, $\hat{T}_{\max}=\max\{\hat{T}_j\}$, $\hat{U}_j(S)=0$ if $\hat{C}_j\le\hat{d}_j$ and $\hat{U}_j(S)=1$ if $\hat{C}_j>\hat{d}_j$.

It is evident from Eq. (1) that, for the same batch sequence S, $C_j(S)=\hat{C}_j(S)+g_j$, $L_j(S)=\hat{L}_j(S)$, $T_j(S)=\hat{T}_j(S)$, and $U_j(S)=\hat{U}_j(S)$, where $g_j=\sum_{i=1}^{j}p_{2i}$, $\hat{d}_j=d_j-g_j$, $s=s_1+s_2$ and $p_j=p_{1j}$, $j=1,\ldots,n$. So, for the same batch sequence,

$$\sum w_jC_j=\sum w_j\hat{C}_j+\sum w_jg_j,\quad \sum w_jU_j=\sum w_j\hat{U}_j,$$
$$\sum w_jT_j=\sum w_j\hat{T}_j,\quad L_{\max}=\hat{L}_{\max},\quad T_{\max}=\hat{T}_{\max}.$$

It follows that, when either the job sequence can be optimally determined or is given, the server scheduling problem of finding a schedule feasible with respect to the deadlines or minimizing $\sum w_jC_j$, $\sum w_jU_j$, $\sum w_jT_j$ or $L_{\max}$ is equivalent to the corresponding single machine batching problem.

In particular, for minimizing $\sum w_jC_j$ when the job sequence is fixed, one can use the $O(n)$ algorithm suggested by Albers and Brucker [1], which is to formulate and solve a special shortest path problem.

For the maximum lateness criterion, the pairwise job interchange technique can be used to prove the following.

Lemma 2. *For the problem of minimizing $L_{\max}$, there exists an optimal batching schedule in which the jobs are sequenced in the earliest due date (EDD) order.*

Re-number the jobs so that $d_1 \leq \cdots \leq d_n$. The problems of finding a schedule feasible with respect to the deadlines, and minimizing $L_{\max}$ or $T_{\max}$ reduce to partitioning the sequence $(1, \ldots, n)$ into batches.

Hochbaum and Landy [17] give an $O(n)$ procedure to solve the single machine batching problem of partitioning a given sequence $(1, \ldots, n)$ into batches so that the resulting schedule is feasible with respect to the deadlines $(\hat{C}_j \leq \hat{d}_j$, $j = 1, \ldots, n)$, if any such schedule exists. In their approach, jobs $1, 2, \ldots$ are assigned to the first batch until its earliest deadline d_1 is exceeded. Then the second batch is started. This process is repeated until all jobs are assigned to batches or some job j cannot be assigned to the current batch or to the new batch without violating a deadline.

This procedure can be combined with a bisection search to solve the problem of minimizing the maximum cost $f_{\max}$ in $O(n \log(u - v) \log P)$ time, where u and v are lower and upper bounds, respectively, for the minimum $f_{\max}$ value and $P = n(s_1 + s_2) + \sum_{j=1}^{n}(p_{1j} + p_{2j})$. In each iteration of the bisection search procedure, the question if $f_{\max} \leq k$ has to be answered for a trial value $k \in [v, u]$. The latter inequality is equivalent to $C_j \leq t_j$, $j = 1, \ldots, n$, where t_j can be found in $O(\log P)$ by a bisection search in the range $[0, P]$ of C_j values for each j, $j = 1, \ldots, n$. More details of such an approach can be found, for example, in Brucker et al. [7].

Wagelmans and Gerodimos [29] describe an $O(n \log n)$ algorithm to solve the single machine batching problem of minimizing $\hat{L}_{\max}$. It can be used to solve our problem with the $L_{\max}$ criterion. Notice that the EDD sequence $(1, \ldots, n)$ such that $d_1 \leq \cdots \leq d_n$ must be taken as the input for the algorithm of Wagelmans and Gerodimos.

We now study the problem of minimizing the total completion time $\sum C_j$ under the assumption that the job processing times are agreeable. Consider an arbitrary job sequence. To simplify the notation, let the sequence be $(1, \ldots, n)$. As before, consider batch sequence $S = (B_1, \ldots, B_r)$, where $B_k = (a_{k-1} + 1, a_{k-1} + 2, \ldots, a_k)$ is the batch sequenced k-th, $k = 1, \ldots, r$, $a_0 = 0$. For this sequence, the total completion time can be calculated as follows. Denote by b_k the cardinality of the k-th batch: $b_k = |B_k|$.

$$\sum C_j(S) = \sum_{k=1}^{r} \sum_{j \in B_k} \left(k(s_1 + s_2) + \sum_{i=1}^{a_k} p_{1i} + \sum_{i=1}^{j} p_{2i} \right)$$

$$= (s_1 + s_2) \sum_{k=1}^{r} k b_k + \sum_{k=1}^{r} \sum_{j \in B_k} \left(p_{1j} \sum_{i=k}^{r} b_i + p_{2j}(n - j + 1) \right). \tag{2}$$

Since the job processing times are agreeable, applying the principle of minimizing a linear form over a set of permutations (see, for example, Tanaev, Gordon and Shafransky [27], p. 189) to the above formula shows that it is optimal to sequence the jobs in the SPT order such that $p_{11} \leq p_{12} \leq \cdots \leq p_{1n}$. This ordering implies $p_{21} \leq p_{22} \leq \cdots \leq p_{2n}$ because the job processing times are agreeable.

Let the jobs be numbered in the SPT order. The problem of minimizing $\sum C_j$ reduces to partitioning the sequence $(1, \ldots, n)$ into batches. Coffman et al. [12] present an $O(n)$ algorithm to solve the single machine batching problem of minimizing $\sum \hat{C}_j$ once the job sequence is fixed. Since in this case $\sum C_j = \sum \hat{C}_j + \sum g_j$, the algorithm of Coffman et al. can be used to minimize $\sum C_j$ for the server scheduling problem with agreeable job processing times. The latter assumption is quite reasonable. The job processing times are agreeable when the machines are uniform, i.e., the processing time of job j on machine l is equal to p_j/v_l, where v_l is the *speed* of machine l, $l = 1, 2$, or when $p_{lj} = p$ for all j and some l.

By modifying equation (2), it is easy to observe that the SPT sequence $(1, \ldots, n)$ is optimal for the problem of minimizing $\sum w_j C_j$ when the job processing times and weights are agreeable, i.e., when $w_1 \geq \cdots \geq w_n$ is additionally satisfied. In this case, an optimal partition into batches can be obtained by the $O(n)$ algorithm of Albers and Brucker [1].

Now, consider the general case of the problem of minimizing $\sum w_j C_j$. It is easy to verify that for an arbitrary job sequence $(i_1, \ldots, i_n)$, we have

$$\sum w_j C_j = \sum w_j \hat{C}_j + \sum_{j=1}^{n} w_{i_j} \sum_{h=1}^{j} p_{2i_h},$$

where $\hat{C}_j$ is the completion time of job j in the single machine batching problem with job processing times p_{1j} and machine setup time $s_1 + s_2$.

Albers and Brucker [1] prove that the single machine batching problem of verifying $\sum w_j \hat{C}_j \leq Y$ for a specified Y is NP-complete in the strong sense. If we choose the values of p_{2j} sufficiently small compared with the values of p_{1j}, we can easily show that the feasibility problem $\sum w_j C_j \leq y$ is equivalent to $\sum w_j \hat{C}_j \leq Y$ for an appropriate y. Thus, the server scheduling problem of minimizing $\sum w_j C_j$ is NP-hard in the strong sense.

3. Weighted Number of Late Jobs

It is easy to see that the server scheduling problem of minimizing $\sum w_j U_j$ is not easier than the classical single machine problem $1\|\sum w_j U_j$. Therefore, it is NP-hard. We show that it is only NP-hard in the ordinary sense by deriving pseudopolynomial algorithms for it. We further prove that even minimizing $\sum U_j$ is NP-hard, unlike its classical single machine counterpart $1\|\sum U_j$, which is solvable in $O(n \log n)$ time by Moore's [22] algorithm.

The pairwise job interchange technique can be used to prove the following lemma.

Lemma 3. *For the problem of minimizing $\sum w_j U_j$, there exists an optimal batching schedule in which the early jobs are sequenced in the EDD order and all late jobs are placed in a single batch sequenced after the last early job.*

This lemma does not fully specify an optimal job sequence for the problem of minimizing $\sum w_j U_j$. Therefore, the main result of the previous section cannot be used for solving this problem.

Lemma 3 forms the basis of dynamic programming algorithms A(x) and B(x), where x is an upper bound on the optimal objective value. In these algorithms, the rules of constructing partial schedules similar to those in the algorithms derived by Brucker and Kovalyov [9] and Hochbaum and Landy [17] for the single machine batching problem of minimizing $\sum w_j U_j$ are used. However, the corresponding dynamic programs are more involved because the latter problem is a special case of our problem in that the processing times on the second machine are extremely small and can be set to zero.

We call a batch of early jobs an *early batch*.

Assume that the jobs are numbered in the EDD order such that $d_1 \leq \cdots \leq d_n$ and $s_1 + s_2 + p_{1j} + p_{2j} \leq d_j$, $j = 1, \ldots, n$. If $s_1 + s_2 + p_{1j} + p_{2j} > d_j$ for some j, then j is late in any schedule and can be eliminated from further consideration.

In algorithm A(x) and B(x), the jobs are considered in the order $1, \ldots, n$. Three possible scheduling choices for each job are realized in the algorithms. They are the following:

(i) job j is scheduled as a late job,

(ii) job j is scheduled at the end of the last early batch and all jobs in this batch will be completed by their due dates,

(iii) job j is assigned to a new early batch and will be completed before the due date d_j.

We now give a justification for algorithm A(x). Consider a partial schedule for the jobs $1, \ldots, j-1$ constructed by using the above three rules. Let B be the last early batch in this schedule and let $C(B)$ be the completion time of this batch (on machine 2). We must have

$$C_i = C(B) - \sum_{r>i, r\in B} p_{2r} \leq d_i \quad \text{for all } i \in B,$$

which is equivalent to

$$C(B) \leq \min_{i\in B} \left\{ d_i + \sum_{r>i, r\in B} p_{2r} \right\}.$$

We call $d_i + \sum_{r>i, r\in B} p_{2r}$ the *modified due date* of job i in batch B and the minimum of these values the *earliest modified due date* of batch B.

Consider two partial schedules for the jobs $1, \ldots, j-1$ with the same weighted number of late jobs w and the same earliest modified due date e of the last early batches. We will show that the schedule with the minimum com-

pletion time of the last early batch *dominates* the other ones, i.e., it can be extended to a complete schedule without incurring a larger weighted number of late jobs.

Let job j be added to the last early batch of the non-dominant partial schedule. In this case, the "old" jobs in this batch and job j will be early. The earliest modified due date of this batch will either increase by p_{2j} or it will be equal to d_j. The completion time of this batch will increase by $p_{1j} + p_{2j}$.

Since the completion time of the last early batch in the dominant schedule is not larger than that in the non-dominant schedule, the same situation will arise if job j is added to the last early batch of the dominant schedule.

If job j starts a new early batch in the non-dominant schedule, then the earliest modified due date of this batch will be equal to d_j. The completion time of this batch will increase by $s_1 + s_2 + p_{1j} + p_{2j}$ compared with the previous early batch. The same situation will arise if job j starts a new early batch in the dominant schedule.

If j is set to be late in any of the two schedules, the weighted number of late jobs will increase by w_j and the earliest modified due date of the last early batch will not change, nor will its completion time.

Repeating the above reasoning for $j+1, \ldots, n$ proves that the schedule with the minimum completion time of the last early batch is indeed dominant among those for jobs $1, \ldots, j-1$ with the same weighted number of late jobs w and the same earliest modified due date e.

In algorithm A(x), the completion time of the last early job is a function value, while the weighted number of late jobs and the earliest modified due date in the last early batch are state variables. More precisely, we recursively compute the value $f_j(w, e)$, which represents the minimum completion time of the last early job, subject to jobs $1, \ldots, j$ are scheduled, the weighted number of late jobs is equal to w, and the earliest modified due date in the last early batch is e. If there are no early jobs in the schedule, then we set $e = 0$.

Set $d_0 = 0$, $E_0 = \{d_0\}$, $E_1 = \{d_0, d_1\}$, $p_{\min}^{(ij)} = \min_{i<r\le j}\{p_{2r}\}$, $p_{\Sigma}^{(ij)} = \sum_{i<r\le j} p_{2r}$ and

$$E_j = \Big\{ d_0, d_1, d_j, d_i + p^{(ij)} | i = 1, 2, \ldots, j-1, \\ p^{(ij)} = p_{\min}^{(ij)}, p_{\min}^{(ij)} + 1, \ldots, p_{\Sigma}^{(ij)}, d_i + p^{(ij)} \le d_j, \Big\}, \quad j = 2, \ldots, n.$$

The initialization is $f_0(0,0) = 0$, $f_j(w,e) = \infty$ for $(j,w,e) \neq (0,0,0)$, and the recursion for $j = 1, \ldots, n$, $w = 0, 1, \ldots, x$, and $e \in E_j$ is

$$f_j(w,e)=\min\begin{cases} f_{j-1}(w-w_j,e), & \\ f_{j-1}(w,e-p_{2j})+p_{1j}+p_{2j}, & \text{if } e\neq d_j \text{ and } f_{j-1}(w,e-p_{2j})+p_{1j}+p_{2j}\le e, \\ \min\limits_{h\in H_j(w)}\{f_{j-1}(w,h)\}+p_{1j}+p_{2j}, & \text{if } e=d_j, \text{ and } H_j(w)\neq\phi, \\ \min\limits_{h\in G_j(w)}\{f_{j-1}(w,h)\}+s_1+s_2+p_{1j}+p_{2j}, & \text{if } e=d_j \text{ and } G_j(w)\neq\phi, \end{cases}$$

where

$$H_j(w)=\{h\in E_{j-1}\mid h+p_{2j}>d_j,\ f_{j-1}(w,h)+p_{1j}+p_{2j}\le d_j\},$$
$$G_j(w)=\{h\in E_{j-1}\mid f_{j-1}(w,h)+s_1+s_2+p_{1j}+p_{2j}\le d_j\}.$$

The four terms in the bracket of the recursive relation correspond to the decisions that job j is late; job j is early, does not start a batch and the modified earliest due date of the last early batch is increased by p_{2j}; job j is early, does not start a batch and the modified earliest due date of the last early batch is changed to d_j; and job j is early and starts a batch, respectively.

The minimum weighted number of late jobs is then equal to the smallest value of w for which $\min\{f_n(w,e)\mid e\in E_n\}<\infty$.

Set $P_l=\sum_{i=1}^{n}p_{li},\ l=1,2$. The time complexity of the algorithm is $O(nx\min\{nP_2,d_n\})$ because there are at most n different values of j, x different values of w, and $|E_j|\le|E_n|\le\min\{nP_2,d_n\}$ for all j.

Clearly, algorithm A($\sum w_j$) solves the problem of minimizing $\sum w_jU_j$ in $O(n\min\{nP_2,d_n\}\sum w_j)$ time. Thus, this problem is not NP-hard in the strong sense.

If $p_{2j}=p$ for all j, then $|E_n|\le n^2$. In this case, $\sum w_jU_j$ can be minimized in $O(n^3\sum w_j)$ time and, hence, $\sum U_j$ can be minimized in $O(n^4)$ time.

In algorithm B(x), the function to be minimized is $g_j(w,\delta)$. The meaning of the function value is the same as that in A(x), i.e., it is the completion time of the last early job in a partial schedule. The state variables are the number of jobs j scheduled so far, the weighted number of late jobs w, and the difference δ between the final and current total processing time of jobs in the last early batch on machine 1. Notice that $\delta=0$ when the last early batch contains all its jobs.

The initialization is $g_0(0,0)=0$, $g_j(w,\delta)=\infty$ for $(j,w,\delta)\neq(0,0,0)$, and the recursion for $j=1,\ldots,n$, $w=0,1,\ldots,x$, and $\delta=0,1,\ldots,\min\{P_1,d_n\}$, is

$$g_j(w,\delta)=\min\begin{cases} g_{j-1}(w-w_j,\delta), & \\ g_{j-1}(w,\delta+p_{1j})+p_{2j}, & \text{if } g_{j-1}(w,\delta-p_{1j})+p_{2j}\le d_j, \\ g_{j-1}(w,0)+s_1+\delta+s_2+p_{2j}, & \text{if } g_{j-1}(w,0)+s_1+\delta+s_2+p_{2j}\le d_j. \end{cases}$$

The three terms in the bracket of the recursive relation correspond to the decisions that job j is late, job j is early and does not start a batch, and job j is early and starts a batch, respectively.

The minimum weighted number of late jobs is equal to the smallest value of w for which $\min\{g_n(w,\delta)|\delta = 0,1,\ldots,\min\{P_1,d_n\}\} < \infty$.

A justification for algorithm B(x) is that a partial schedule in the state (j,w,δ) with the minimum completion time of the last early job dominates any other schedule in the same state.

The time complexity of the algorithm is $O(nx\min\{P_1,d_n\})$ because there are at most n different values of j, x different values of w and $\min\{P_1,d_n\}$ different values of δ. If $p_{1j} = p$ for all j, then there are at most n different values of δ. In this case, $\sum w_jU_j$ can be minimized in $O(n^2\sum w_j)$ time and $\sum U_j$ can be minimized in $O(n^3)$ time.

Using standard rounding techniques (see, for example, Sahni [26] and Kovalyov [20]), we can transform algorithm A(x) for the case $p_{2j} = p$, $j = 1,\ldots,n$, and algorithm B(x) for the case $p_{1j} = p$, $j = 1,\ldots,n$, to a *fully polynomial time approximation scheme* (*FPTAS*) similar to the way it is done by Brucker and Kovalyov [9] for the single machine batching problem.

The time complexity to generate a schedule with a weighted number of late jobs that is no more than $1+\varepsilon$ times the optimal value is $O(n^4/\varepsilon + n^4\log\log n)$ and $O(n^3/\varepsilon + n^3\log\log n)$, respectively.

Now assume that the processing times p_{1j} and p_{2j} are arbitrary.

Theorem 1. *The problem of minimizing the number of late jobs is NP-hard, even if the setup times are equal and there are two distinct due dates.*

Proof. A polynomial reduction from the NP-complete problem Equal Cardinality Partition (see Garey and Johnson [14]) is used.

Equal Cardinality Partition: Given positive integers $e_1,e_2,\ldots,e_{2k}$ and E satisfying $\sum_{j=1}^{2k} e_j = 2E$, does there exist a subset X of the set $\{1,\ldots,2k\}$ such that $\sum_{j\in X} e_j = E$ and $|X| = k$? Assume, without loss of generality, that $e_j \le E$ for each j because otherwise the solution is trivial.

Let an instance of Equal Cardinality Partition be given. Compute $H = 2E+2$. Construct an instance of the number of late jobs problem, in which there are $2k+1$ jobs: *enforcer* job 0 and $2k$ *partition* jobs. The job processing times and due dates are given as follows: $(p_{1,0},p_{2,0},d_0) = (1,1,3E+2)$ and $(p_{1j},p_{2j},d_j) = (e_j,H-2e_j,d)$, $d = kH+E+2$, $j = 1,\ldots,2k$. Note that H is chosen such that $p_{2j} = H-2e_j = 2(E-e_j)+2 > 0$. Both setup times are equal: $s_1 = s_2 = E$.

We prove that Equal Cardinality Partition has a solution if and only if there exists a schedule for the constructed instance of the number of late jobs problem with an objective value $\sum U_j \le k$.

Suppose first that Equal Cardinality Partition has a solution and X is the required set. Construct a schedule S having a structure for the early jobs shown in Fig. 2.

In schedule S, there is one early batch. This batch consists of the enforcer job 0 and jobs of the set X. Job 0 completes at time d_0 and the latest of the jobs in X completes at time d. Therefore, there are exactly k late jobs in S.

Suppose now that a schedule S' with $\sum U_j \leq k$ exists. In this schedule, there must be exactly one early batch. Assume that there are two early batches B_1 and B_2 sequenced in this order. The earliest completion time for the batch B_2, $C(B_2)$, is achieved when job 0 is early. In this case, at least k partition jobs must be early. Let us estimate $C(B_2)$:

$$\begin{aligned} C(B_2) &\geq 2(s_1 + s_2) + p_{1,0} + p_{2,0} + \sum_{j \in B_1 \cup B_2 \setminus \{0\}} (p_{1j} + p_{2j}) \\ &= 4E + 2 + kH - \sum_{j \in B_1 \cup B_2 \setminus \{0\}} e_j \geq kH + 2E + 2 > d. \end{aligned}$$

Therefore, at least one job in $B_1 \cup B_2$ is late, which is a contradiction. The same situation arises when there are more than two early batches. Thus, there is exactly one early batch that we denote by B.

Assume that at least $k + 1$ partition jobs are early. In this case, for the completion time of the batch B, we have

$$\begin{aligned} C(B) &\geq s_1 + s_2 + \sum_{j \in B \setminus \{0\}} (p_{1j} + p_{2j}) \geq 2E + (k+1)H - \sum_{j \in B \setminus \{0\}} e_j \\ &\geq (k+1)H > kH + E + 2 = d, \end{aligned}$$

because $H = 2E + 2 > E + 2$. Hence, at least one job in B is late, a contradiction.

It follows that exactly k partition jobs and job 0 comprise the only early batch B in S'. According to Lemma 3, assume that job 0 precedes all the partition jobs in B.

Set $X = B \setminus \{0\}$. For the completion time of job 0, we have

$$C_0 = 2E + 2 + \sum_{j \in X} e_j \leq d_0 = 3E + 2.$$

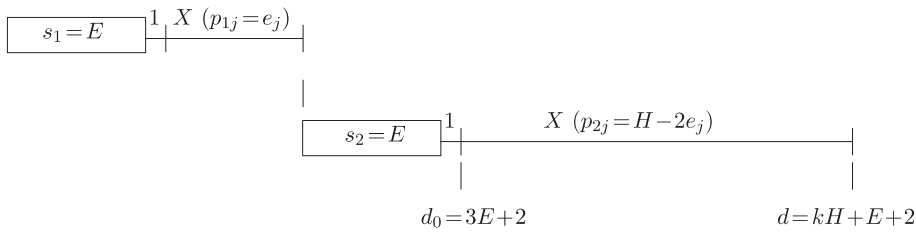

Fig. 2. Schedule for early jobs in S

Hence,

$$\sum_{j\in X} e_j \le E. \tag{3}$$

For the completion time of the batch B, we must have

$$d = kH + E + 2 \ge C(B) = s_1 + s_2 + 2 + \sum_{j\in X}(p_{1j} + p_{2j}) = 2E + 2 + kH - \sum_{j\in X} e_j,$$

from which we get $\sum_{j\in X} e_j \ge E$. This inequality and (3) imply $\sum_{j\in X} e_j = E$, as required. □

Since minimizing $\sum U_j$ is NP-hard for arbitrary p_{1j} and p_{2j}, no FPTAS exists for this problem unless $\mathscr{P} = \mathscr{NP}$. Indeed, if such a FPTAS would exist we could have set $\varepsilon = 1/(n+1)$ and found a schedule with a number of late jobs exceeding the optimal value by at most $n/(n+1) < 1$, i.e., an optimal schedule, in polynomial time.

Notice that a FPTAS might exist for the general problem of minimizing $\sum w_j U_j$.

4. Conclusions

The problem of scheduling a single server that processes n jobs in a two-machine flow shop has been studied. A machine dependent setup time is needed whenever the server switches from one machine to the other. Each processing operation and each setup operation is performed by the server. For many regular performance criteria, the problem is reduced to a single machine batching problem. Known algorithms for the latter problem can be used to solve the former ones. The derived computational complexity results are given in Table 1.

An open question is the complexity of the problem with the total completion time criterion when processing times are not agreeable. Further research can be undertaken to resolve this question. Problems with more than two machines and more than one server are also of interest for further research.

Table 1. Computational complexity of the server scheduling problem

Objective function, additional assumptions	Complexity
$L_{\max}, T_{\max}$	$O(n \log n)$
$f_{\max}$	$O(n \log (u-v) \log P)$
$\sum C_j$	Open
$\sum C_j$, agreeable p_{1j}	$O(n \log n)$
$\sum w_j C_j$	strongly NP-hard
$\sum U_j$	NP-hard
$\sum U_j$, $p_{1j} = p$	$O(n^3)$
$\sum U_j$, $p_{2j} = p$	$O(n^4)$
$\sum w_j U_j$	NP-hard, pseudopolynomially solvable

Acknowledgements

T.C.E. Cheng was partially supported by the Croucher Foundation under a Croucher Senior Research Fellowship. M.Y. Kovalyov was partially supported by INTAS (Project INTAS-00-217).

References

[1] Albers, S., Brucker, P.: The complexity of one-machine batching problems. Discrete Appl. Math. *47*, 87–107 (1993).
[2] Allahverdi, A., Gupta, J. N. D., Aldowaisan, T.: A review of scheduling research involving setup considerations. Omega *27*, 219–239 (1999).
[3] Baker, K. R.: Introduction to sequencing and scheduling. New York: Wiley 1974.
[4] Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., Weglarz, J.: Scheduling computer and manufacturing processes. Heidelberg: Springer 1996.
[5] Brucker, P.: Scheduling algorithms. Heidelberg: Springer 1995.
[6] Brucker, P., Dhaenens-Flipo, C., Knust, S., Kravchenko, S. A., Werner, F.: Complexity results for parallel machine problems with a single server. In: Extended Abstracts of Seventh International Workshop on Project Management and Scheduling, Osnabruck, Germany, April 17–19, 2000, pp 82–84.
[7] Brucker, P., Gladky, A., Hoogeveen, J. A., Kovalyov, M. Y., Potts, C. N., Tautenhahn, T., van de Velde, S. L.: Scheduling a batching machine. Journal of Scheduling *1*, 31–54 (1998).
[8] Brucker, P., Knust, S., Wang, G.: Complexity results for flow-shop problems with a single server. In: Proc. 5th Int. Conf. on Optimization: Techniques and Applications, Li, D. (ed.), 153–167 (2001).
[9] Brucker, P., Kovalyov, M. Y.: Single machine batch scheduling to minimize the weighted number of late jobs. Math. Methods Oper. Res. *43*, 1–8 (1996).
[10] Cheng, T. C. E., Wang, G., Sriskandarajah, C.: One-operator-two-machine flowshop scheduling problems with setup and dismounting times. Computers Oper. Res. *26*, 715–730 (1999).
[11] Coffman, Jr. E. G., (ed.): Computer and job-shop scheduling theory. New York: Wiley 1976.
[12] Coffman, Jr. E. G., Yannakakis, M., Magazine, M. J., Santos, C.: Batch sizing and job sequencing on a single machine. Ann. Oper. Res. *26*, 135–147 (1990).
[13] Conway, R. W., Maxwell, W. L., Miller, L. W.: Theory of scheduling. Addison Wesley, Reading 1967.
[14] Garey, M. R., Johnson, D. S.: Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman and Co., San Francisco 1979.
[15] Glass, C., Shafransky, Y. M., Strusevich, V. A.: Scheduling for parallel dedicated machines with a single server. Naval Res. Logistics *47*, 304–328 (1999).
[16] Hall, N. G., Potts, C. N., Sriskandarajah, C.: Parallel machine scheduling with a common server. Discr. Appl. Math. *102*, 223–243 (2000).
[17] Hochbaum, D. S., Landy, D.: Scheduling with batching: minimizing the weighted number of tardy jobs. Oper. Res. Lett. *16*, 79–86 (1994).
[18] Kravchenko, S. A., Werner, F.: Parallel machine scheduling problems with a single server. Math. Comp. Modelling *26*, 1–11 (1997).
[19] Koulamas, C. P.: Scheduling on two parallel machines for minimizing machine interference. Working paper. Department of Decision Sciences and Information Systems. Florida International University 1993.
[20] Kovalyov, M. Y.: A rounding technique to construct approximation algorithms for knapsack and partition type problems. Appl. Math. Comp. Sci. *6*, 101–113 (1996).
[21] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., Shmoys, D. B.: Sequencing and scheduling: algorithms and complexity. In: Logistics of Production and Inventory, Handbooks in Operations Research and Management Science, Graves, S. C., Zipkin, P. H., Rinnooy Kan, A. H. G. (eds.), vol. 4, North-Holland, Amsterdam, 445–522 (1993).
[22] Moore, J. M.: An *n* job, one machine sequencing algorithm for minimizing the number of late jobs. Management Sci. *15*, 102–109 (1968).
[23] Pinedo, M., Chao, X.: Operation scheduling with applications in manufacturing and services. New York: Irwin/McGraw-Hill, 1999.
[24] Potts, C. N., Kovalyov, M. Y.: Scheduling with batching: a review. Eur. J. Oper. Res. *120*, 228–249 (2000).
[25] Potts, C. N., Van Wassenhove, L. N.: Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. J. Oper. Res. Soc. *43*, 395–406 (1992).

[26] Sahni, S.: General techniques for combinatorial approximation. Oper. Res. *25*, 920–936 (1977).
[27] Tanaev, V. S., Gordon, V. S., Shafransky, Y. M.: Scheduling theory. Single-stage systems. Kluwer Academic Publishers, Dordrecht/Boston/London 1994.
[28] Webster, S. T., Baker, K. R.: Scheduling groups of jobs on a single machine. Oper. Res. *43*, 692–703 (1995).
[29] Wagelmans, A. P. M., Gerodimos, A. E.: Improved dynamic programs for some batching problems involving the maximum lateness criterion. Oper. Res. Lett. *27*, 109–118 (2000).

T. C. Edwin Cheng
Department of Management
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong

M. Y. Kovalyov
Faculty of Economics
Belarus State University
K. Marxa 31, room 70
220050 Minsk, Belarus
e-mail: koval@newman.bas-net.by

Verleger: Springer-Verlag KG, Sachsenplatz 4–6, A-1201 Wien. – Herausgeber: Prof. Dr. Wolfgang Hackbusch, Max-Planck-Institut für Mathematik in den Naturwissenschaften, Inselstraße 22–26, D-04103 Leipzig – Satz und Umbruch: Scientific Publishing Services (P) Ltd., Madras; Offsetdruck: Manz Crossmedia. 1051 Wien. – Verlagsort: Wien. – Herstellungsort: Wien. – Printed in Austria.