Minimizing Sum of Completion Times for Batch Scheduling of Jobs with Deteriorating Processing Times

Joseph Y-T. Leung[a], C.T. Ng[b,*], T.C. Edwin Cheng[b]

[a]*Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, 07102, U.S.A.*

[b]*Department of Logistics, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong*

Abstract

We consider the problem of scheduling n jobs on $m \geq 1$ parallel and identical machines, where the jobs are processed in batches and the processing time of each job is a step function of its waiting time; i.e., the time between the start of the processing of the batch to which the job belongs and the start of the processing of the job. For each job i, if its waiting time is less than a given threshold D, then it requires a *basic* processing time $p_i = a_i$; otherwise, it requires an *extended* processing time $p_i = a_i + b_i$. The objective is to minimize the sum of completion times; i.e., $\sum_{i=1}^{n} C_i$, where C_i is the completion time of job i. We first show that the problem is NP-hard in the strong sense even if there is a single machine and $b_i = b$ for all $1 \leq i \leq n$. We then show that the problem is solvable in polynomial time if $a_i = a$ for all $1 \leq i \leq n$. Our algorithm runs in $O(n^2)$ time. Finally, we give an approximation algorithm for the special case where $b_i \leq D$ for all $1 \leq i \leq n$, and show that it has a performance guarantee of 2.

Keywords: Batch scheduling; Deteriorating processing times; Sum of completion times; Performance guarantee; Strong NP-hardness

*Corresponding author. Tel.: +852 2766 7364; fax: +852 2330 2704.
E-mail addresses: leung@oak.njit.edu (J.Y-T. Leung), lgtctng@polyu.edu.hk (C.T. Ng), lgtcheng@polyu.edu.hk (T.C.E. Cheng).

1 Introduction

Barketau et al. [2] have considered the following scheduling problem: There are n jobs to be scheduled on a single machine. The jobs are processed in batches. Each batch is preceded by a setup time S. The processing time of a job is a step function that depends on the time elapsed since the start of the processing of the batch to which the job belongs. Let s_i^l and s_i denote the starting time of the l-th batch to which job i belongs and the starting time of job i, respectively. If $s_i - s_i^l < D$, then the processing time of job i is equal to $p_i = a_i$; otherwise, it is equal to $p_i = a_i + b_i$. We call a_i and $a_i + b_i$ the *basic* and *extended* processing times of job i, respectively. The completion time of a job is the actual time when the job has finished processing ("job availability", see [5]), rather than the completion time of all jobs in its batch ("batch availability", see [5]). The problem Barketau et al. [2] considered is the minimization of the makespan (i.e., $C_{\max}$) of the schedule. The problem will be denoted by $1 \mid S, p_i = a_i + stp(s_i - s_i^l - D)b_i \mid C_{\max}$, where

$$stp(x) = \begin{cases} 1 & : \quad x \geq 0 \\ 0 & : \quad x < 0 \end{cases}$$

Barketau et al. [2] show that the above scheduling problem is NP-hard in the strong sense, even if $b_i = b$ for all $1 \leq i \leq n$. They then show that the problem is solvable in $O(n \log n)$ time if $a_i = a$ for all $1 \leq i \leq n$. Furthermore, they propose an $O(n^4)$ time approximation algorithm for the general problem, and show that it has a performance guarantee of $\frac{3}{2}$. Moreover, they show that there is no algorithm with a better performance guarantee than $\frac{3}{2}$ unless P = NP. Finally, for the strongly NP-hard special case of the problem in which $a_i < a_j$ implies that $b_i \geq b_j$, they propose an $O(n^3 \log(\sum_{i=1}^n a_i))$ time approximation algorithm with a performance guarantee of $\min\{\frac{3}{2}, \frac{11}{9} + \frac{1}{l^*}\}$, where l^* is the number of batches in an optimal schedule. Barketau et al. [2] give a couple of applications motivating the study of this problem.

In this paper we continue the work of Barketau et al. [2], focusing instead on the sum of completion times objective. We assume that there are $m \geq 1$ parallel and identical machines. Our objective is to minimize the sum of completion times; i.e., $\sum_{i=1}^n C_i$, where C_i is the completion time of job i. We denote this problem by $Pm \mid S, p_i = a_i + stp(s_i - s_i^l - D)b_i \mid \sum_{i=1}^n C_i$.

We show that this problem is strongly NP-hard even when there is a single machine and $b_i = b$ for all $1 \leq i \leq n$. We then give an $O(n^2)$ time optimal algorithm for the special case where $a_i = a$ for all $1 \leq i \leq n$. Finally, we give an $O(n^3)$ time approximation algorithm for the special case where $b_i \leq D$ for all $1 \leq i \leq n$, and show that it has a performance guarantee of 2. We feel that this special case is not too restrictive in practice, since D is the threshold and it is therefore reasonably large.

The described model falls into the category of scheduling problems with time-dependent processing times, which has recently been reviewed by Cheng et al. [3]; in the category of batch scheduling problems, which has been reviewed by Potts and Kovalyov [5]; and in the category of scheduling problems with setup times, which has been reviewed by Allahverdi et al. [1]. However, to the best of our knowledge, no work has been done on models with both aspects of batching and time-dependent processing times, other than the work of Barketau et al. [2]. We feel that this is a probable scenario in real-life scheduling environments, and hence deserves some attention.

This paper is organized as follows. In the next section we will show the strong NP-hardness result. In Section 3 we give the $O(n^2)$ time optimal algorithm for the special case where $a_i = a$ for all $1 \leq i \leq n$. In Section 4 we give the approximation algorithm for the special case where $b_i \leq D$ for all $1 \leq i \leq n$, and show that it has a performance guarantee of 2. The last section concludes.

2 Complexity results

In this section we show that the problem is strongly NP-hard even when there is a single machine and $b_i = b$ for all $1 \leq i \leq n$. We show this by reducing to it the strongly NP-complete 3-Partition problem, which is defined as follows [4].

3-Partition: Given a list Z of $3h$ integers, $Z = (z_1, z_2, \ldots, z_{3h})$, such that $\sum_{i=1}^{3h} z_i = hB$ and $\frac{B}{4} < z_i < \frac{B}{2}$ for each $1 \leq i \leq 3h$, can Z be partitioned into h disjoint sublists $Z_1, Z_2, \ldots, Z_h$ such that for each $1 \leq j \leq h$, $\sum_{z_i \in Z_j} z_i = B$?

Theorem 1 *The problem* $1 \mid S, p_i = a_i + stp(s_i - s_i^l - D)b_i \mid \sum_{i=1}^{n} C_i$ *is NP-hard in the strong sense even if* $b_i = b$ *for all* $1 \leq i \leq n$.

Proof. We shall reduce the 3-Partition problem to the $1 \mid S, p_i = a_i + stp(s_i - s_i^l - D)b_i \mid \sum_{i=1}^{n} C_i$ problem. Given an instance of the 3-Partition problem, we construct an instance of the scheduling problem as follows.

Let there be a single machine, $3h$ "partition" jobs and h "enforcer" jobs. For each partition job i, $1 \leq i \leq 3h$, $a_i = z_i$ and $b_i = b = (2h^2 + 2h + 1)S$. For each enforcer job i, $3h + 1 \leq i \leq 4h$, $a_i = B + 1$ and $b_i = b = (2h^2 + 2h + 1)S$. Let $D = B + 1$ and $S = h(2h - 1)(2B + 1) + 3hB$. Finally, let

$$C^* = \sum_{j=1}^{h} [4jS + (4j - 3)(2B + 1) + \frac{B}{2} + \frac{2B}{2} + \frac{3B}{2}] = (2h^2 + 2h + 1)S = b.$$

We shall show that the instance of the 3-Partition problem has a solution if and only if there is a schedule for the problem $1 \mid S, p_i = a_i + stp(s_i - s_i^l - D)b_i \mid \sum_{i=1}^{n} C_i$ with the sum of completion times not greater than C^*.

Assume that $Z_1, Z_2, \ldots, Z_h$ is a given partition of the instance of 3-Partition. Consider a schedule for the problem $1 \mid S, p_i = a_i + stp(s_i - s_i^l - D)b_i \mid \sum_{i=1}^{n} C_i$ with h batches. In the l-th batch, $1 \leq l \leq h$, the jobs that correspond to the elements of the sublist Z_l are processed first, in any order, followed by one of the enforcer jobs. Since $z_i < \frac{B}{2}$, it is easy to see that the sum of completion times of this schedule is not greater than C^*.

Now, suppose that there is a schedule for the instance of the problem $1 \mid S, p_i = a_i + stp(s_i - s_i^l - D)b_i \mid \sum_{i=1}^{n} C_i$ with the sum of completion times not greater than C^*. It is clear that no job (be it a partition job or an enforcer job) can be executed in its extended processing time in the schedule, since $b_i = b = C^*$. Thus, every job must be executed in its basic processing time only. Furthermore, there are at least h batches in the schedule. Otherwise, there will be at least one job executed in its extended processing time. Thus, if we can show that there are at most h batches in the schedule, then we have shown that there are exactly h batches in the schedule. Each batch will consist of three partition jobs and one enforcer job, and the basic processing times of the three partition jobs will add up to exactly B.

Our proof is thus reduced to showing that there are at most h batches in the schedule. Assume that there are $h+1$ batches in the schedule; the case with more than $h+1$ batches can be proved in a similar fashion. The best case scenario (in terms of the sum of completion

times of the jobs) is that there are four jobs scheduled in each of the first $h-1$ batches, three jobs scheduled in the h-th batch, and one job scheduled in the $(h+1)$-th batch. Observing that the basic processing time of each job is larger than $\frac{B}{4}$, we can compute a lower bound L for the sum of ompletion times of the jobs in this schedule as follows.

$$L = \sum_{j=1}^{h-1}[4jS+4(j-1)B+\frac{B}{4}+\frac{B}{2}+\frac{3B}{4}+B]+[3hS+(h-1)B+\frac{B}{4}+\frac{B}{2}+\frac{3B}{4}]+[(h+1)S+hB]$$

$$> (2h^2+2h+1)S = C^*.$$

This contradicts the fact that the schedule has the sum of completion times not greater than C^*. ■

Since the problem $1 \mid S, p_i = a_i + stp(s_i - s_i^l - D)b_i \mid \sum_{i=1}^n C_i$ is strongly NP-hard for the case $b_i = b$ for all $1 \leq i \leq n$, we immediately have the following results.

Corollary 1 *The problem* $1 \mid S, p_i = a_i + stp(s_i - s_i^l - D)b_i \mid \sum_{i=1}^n C_i$ *is strongly NP-hard for the special cases: (1)* $a_i < a_j$ *implies* $b_i \leq b_j$*, (2)* $a_i > a_j$ *implies* $b_i \leq b_j$*, (3)* $a_i < a_j$ *implies* $b_i \geq b_j$*, and (4)* $a_i > a_j$ *implies* $b_i \geq b_j$*.*

3 Optimal algorithm

In this section we shall give an optimal algorithm for the special case where $a_i = a$ for all $1 \leq i \leq n$. We consider the single machine case first, and then extend the algorithm to the parallel and identical machines case.

Suppose we have a single machine. The scheduling algorithm will schedule the jobs optimally in k batches for each $1 \leq k \leq \lceil \frac{n}{h} \rceil$, where h is the largest number of jobs that can be scheduled in a batch without causing any job to execute in its extended processing time. The algorithm then chooses the schedule with the minimum sum of completion times from among the $\lceil \frac{n}{h} \rceil$ schedules. Suppose we are scheduling the n jobs in k batches. Assume the jobs have been ordered in descending order of the b_i's (i.e., in the LPT order of the b_i's). The algorithm schedules as many jobs as possible in each batch, without causing any job to execute in its extended processing time. In other words, each job is scheduled with its basic processing time $p_i = a$ only. If all the jobs have been scheduled, then we are done. Otherwise, we schedule the remaining jobs in ascending order of the b_i's (i.e., in the SPT

order of the b_i's). For each such job, we either schedule it at the end of the last batch or we schedule it at the end of the first batch (provided that there are two or more batches), depending on which configuration will give a smaller sum of completion times. We keep assigning the jobs in this fashion until we have completely assigned all the jobs. Notice that in the last step, we initially assign some jobs to the end of the first batch, until at some point a job is assigned to the end of the last batch. From then on all the jobs will be assigned to the end of the last batch.

Algorithm OPT-1

Input: A set of n jobs with basic processing times $p_i = a$ and extended processing times $p_i = a + b_i$, $1 \leq i \leq n$. A single machine. Setup time S and threshold D.

Output: A schedule S_t with the minimum sum of completion times.

Method:

1. Sort the jobs in descending order of their extended processing times; i.e., $a + b_1 \geq a + b_2 \geq \cdots \geq a + b_n$.

2. Let h be the largest integer such that $(h-1)a < D$ but $ha \geq D$.

3. $k' \leftarrow \lceil \frac{n}{h} \rceil$.

4. Let S_t be the schedule where the first h jobs are scheduled in the first batch, the next h jobs are scheduled in the second batch, and so on, until the k'-th batch. Let SCT be the sum of completion times of the schedule S_t.

5. For $k = k' - 1$ to 1 do

6. { Let S'_t be the schedule where the first kh jobs are scheduled in the first k batches, with h jobs per batch.

7. For $j = n$ to $kh + 1$ do

8. { if ($hb_j < S$ and $k > 1$) then put job j at the end of the first batch, else put job j at the end of the k-th batch. }

9. Compute the sum of completion times of the schedule S'_t and assign it to SCT'.

10. if $(SCT' < SCT)$ then { $S_t \leftarrow S'_t$. $SCT \leftarrow SCT'$. }. }

11. Return schedule S_t.

Let us consider an example. Shown in Fig. 1(a) is a set of 10 jobs, with $S = 5$ and $D = 10$. The jobs are already in descending order of their extended processing times. In this example, $h = 3$ and $k' = 4$. We first construct the schedule with 4 batches, which is shown in Fig. 1(b). The sum of completion times of this schedule S_t is $SCT = 330$. We then successively construct the schedules with 3 batches, 2 batches and 1 batch, as shown in Figs. 1(c), 1(d) and 1(e), respectively. The schedule with 3 batches has the first 9 jobs scheduled in the first 3 batches; see Fig. 1(c). Job 10 is then scheduled at the end of the first batch, since $3b_{10} < 5$. Scheduling job 10 at the end of the first batch gives a smaller sum of completion times. The schedule with 2 batches has the first 6 jobs scheduled in the first 2 batches, as shown in Fig. 1(d). Job 10 is then scheduled at the end of the first batch, since $3b_{10} < 5$. However, job 9 is scheduled at the end of the last batch, since $3b_9 > 5$. From then on, job 8 and job 7 are also scheduled at the end of the last batch. The schedule with 1 batch has the first 3 jobs scheduled in the first batch, as shown in Fig. 1(e). From then on, job 10 until job 4 are also scheduled at the end of the first batch. The output of the algorithm is the schedule with 2 batches (Fig. 1(d)), since it has the smallest sum of completion times $SCT' = 320$.

i	1	2	3	4	5	6	7	8	9	10
a_i	4	4	4	4	4	4	4	4	4	4
b_i	7	7	6	5	5	4	3	2	2	1

$S = 5$, $D = 10$

Fig. 1(a) A set of 10 jobs

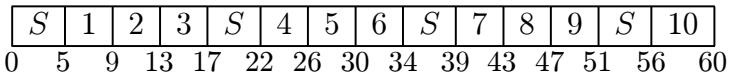

Fig. 1(b) Schedule S_t; $SCT = 330$

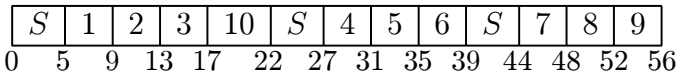

Fig. 1(c) Schedule S'_t; $SCT' = 322$

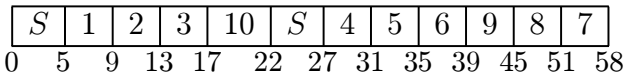

Fig. 1(d) Schedule S'_t; $SCT' = 320$

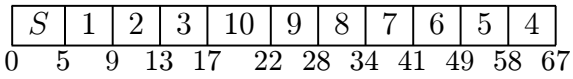

Fig. 1(e) Schedule S'_t; $SCT' = 336$

Integers within the cells in the depicted schedules are job indices.

Fig. 1 An example illustrating Algorithm OPT-1

Let us examine the running time of the algorithm. Sorting the jobs in descending order of the b_i's takes $O(n \log n)$ time. The algorithm takes $O(n)$ iterations, and each iteration takes linear time to schedule the jobs. Thus, the overall running time of the algorithm is $O(n^2)$.

Theorem 2 *Algorithm OPT-1 correctly outputs a schedule with the minimum sum of completion times on a single machine.*

Proof. Let h and k' be as defined in Algorithm OPT-1. It is clear that the optimal schedule will consist of k batches, for some $1 \leq k \leq k'$. Let us now focus on the optimal schedule with k batches. Assume that the jobs are sorted in descending order of their extended processing times, or equivalently, in descending order of the b_i's. The optimal schedule will execute the jobs with the largest extended processing times in their basic processing time mode. Therefore, the first kh jobs will all be executed in their basic processing time mode, while the remaining jobs will all be executed in their extended processing time mode.

Among the jobs $kh+1$, $kh+2$, ..., n, job n has the smallest extended processing time while job $kh+1$ has the largest extended processing time. Therefore, these jobs must be scheduled in the order of n, $n-1$, ..., $kh+1$. Moreover, they must be scheduled at the end of some batch. Suppose we are scheduling job n. Let us compare the effects of putting job n at the end of the $(k-1)$-th batch versus putting it at the end of the k-th batch. If job n is put at the end of the $(k-1)$-th batch, the sum of completion times of job n and all the jobs that follow it are given by

$$t_{k-1} = (X+a+b_n)+(X+a+b_n+S+a)+(X+a+b_n+S+2a)+\cdots+(X+a+b_n+S+ha),$$

where X is the starting time of job n. See Fig. 2 for an illustration.

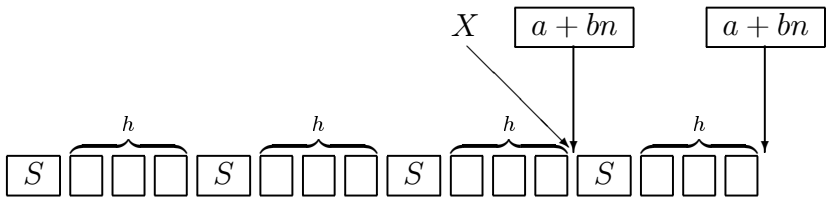

Fig. 2 Illustrating the proof of Theorem 2

On the other hand, if job n is put at the end of the k-th batch, the sum of completion times of the corresponding jobs are given by

$$t_k = (X+S+a)+(X+S+2a)+\cdots+(X+S+ha)+(X+S+ha+a+b_n).$$

Therefore, $t_{k-1} < t_k$ if and only if $hb_n < S$. In other words, it is more advantageous to put job n at the end of the $(k-1)$-th batch, rather than at the end of the k-th batch, if $hb_n < S$. By a similar reasoning, it is more advantageous to put job n at the end of the first batch, rather than at the end of any batch, if $hb_n < S$. On the other hand, if $hb_n > S$, then it is more advantageous to put job n at the end of the k-th batch, rather than at the end of any batch. In case $hb_n = S$, putting job n at the end of any batch gives the same sum of completion times.

Inductively, we can show that the scheduling of job j, $kh+1 \le j \le n$, depends on the relationship between hb_j and S. If $hb_j < S$, then job j should be put at the end of the

first batch; otherwise, it should be put at the end of the last batch. But this is exactly how Algorithm OPT-1 schedules these jobs. Therefore, Algorithm OPT-1 generates an optimal schedule. ■

We now extend the algorithm to the parallel and identical machines case. But first we need to introduce some notations. By "vertical scheduling of n jobs in k batches", we mean scheduling the n jobs vertically on the m machines so that the number of batches is exactly k, without causing any job to execute in its extended processing time mode; see Fig. 3 for an illustration.

		$\overbrace{\quad\quad\quad}^{h}$				$\overbrace{\quad\quad\quad}^{h}$					
M_1	S	1	5	9	S	13	17	21	S	25	27
M_2	S	2	6	10	S	14	18	22	S	26	
M_3	S	3	7	11	S	15	19	23			
M_4	S	4	8	12	S	16	20	24			

Integers within the cells in the depicted schedule are job indices.

Fig. 3 Vertical scheduling of 27 jobs in 10 batches

Shown in Fig. 3 is the vertical scheduling of 27 jobs in 10 batches; the number of machines is 10 and $h = 3$. As can be seen from Fig. 3, we schedule job 1 on machine 1, job 2 on machine 2, and so on, until job 12. At this time, the first 4 batches are full, with each batch having exactly h jobs. The next 4 batches are scheduled similarly. In the last 2 batches, we schedule job 25 on machine 1, job 26 on machine 2, and job 27 on machine 1 again. Notice that the last 2 batches are not full; batch 9 has only 2 jobs and batch 10 has only 1 job.

By "placing job j on machine i", we mean to put job j at the end of the first batch of machine i if $hb_j < S$; otherwise, we put job j at the end of the last batch of machine i. By "vertically placing jobs $n, \ldots, j$ on machines $i, \ldots, m$", we mean placing job n on machine i, placing job $n-1$ on machine $i+1$, ..., placing job $n-[m-(i-1)]-1$ on machine m, placing job $n-[m-(i-1)]-2$ on machine i again, placing job $n-[m-(i-1)]-3$ on machine $i+1$, and so on. In other words, the jobs are placed on the machines in a wrap-around manner.

Algorithm OPT-m

Input: A set of n jobs with basic processing times $p_i = a$ and extended processing times $p_i = a + b_i$, $1 \leq i \leq n$. m identical and parallel machines. Setup time S and threshold D.

Output: A schedule S_t with the minimum sum of completion times.

Method:

1. Sort the jobs in descending order of their extended processing times; i.e., $a + b_1 \geq a + b_2 \geq \cdots \geq a + b_n$.
2. Let h be the largest integer such that $(h-1)a < D$ but $ha \geq D$.
3. $k' \leftarrow m \times \lceil \frac{n}{mh} \rceil$.
4. Let S_t be the schedule where the n jobs are vertically scheduled in k' batches. Let SCT be the sum of completion times of the schedule S_t.
5. For $k = k' - 1$ to 1 do
6. { if $(n \leq kh)$ then { Let S'_t be the vertical schedule of the n jobs in k batches and let SCT' be the sum of completion times of the schedule S'_t. Goto Step (10). }.
7. Schedule the first kh jobs vertically in k batches.
8. if $(k \leq m)$ then { Vertically place the jobs $n, n-1, \ldots, kh+1$ on the machines $1, 2, \ldots, k$. }.

   else { Let $k = qm + r$ and $r' = m - r$. if $(n - kh \leq r'h)$ then { Vertically place the jobs $n, n-1, \ldots, kh+1$ on the machines $r+1, r+2, \ldots, m$. }.

   else { Vertically place the jobs $n, n-1, \ldots, n-r'h+1$ on the machines $r+1, r+2, \ldots, m$. Vertically place the jobs $n-r'h, n-r'h-1, \ldots, kh+1$ on the machines $1, 2, \ldots, m$. }. }.
9. Compute the sum of completion times of the schedule S'_t and assign it to SCT'.

10. if $(SCT' < SCT)$ then { $S_t \leftarrow S'_t$. $SCT \leftarrow SCT'$. }. }

11. Return schedule S_t.

Let us consider an example. Shown in Fig. 4(a) is a set of 20 jobs, with $S = 4$, $D = 3$ and $m = 3$. The jobs are already in descending order of their extended processing times. In this example, $h = 2$ and $k' = 12$. Let us consider step 6 of the algorithm for $k = 7$. Following the algorithm, the test in step 6 of the algorithm fails, so the algorithm proceeds to step 7. It schedules the first $kh = 14$ jobs vertically in 7 batches, as shown in Fig. 4(b). In step 8, the test fails, so it moves to the "else" part. The algorithm computes $r = 1$ and $r' = 2$, and tests if $n - kh \leq r'h$. The test fails, so it moves to the "else" part. Here it first vertically places jobs 20, 19, 18 and 17 on machines M_2 and M_3. It finally places jobs 16 and 15 vertically on machines M_1 to M_3. The final schedule S'_t is shown in Fig. 4(d), with the sum of completion times $SCT' = 309$.

i	1	2	3	4	5	6	7	8	9	10
a_i	2	2	2	2	2	2	2	2	2	2
b_i	9	9	9	8	8	7	7	7	6	6
i	11	12	13	14	15	16	17	18	19	20
a_i	2	2	2	2	2	2	2	2	2	2
b_i	5	5	5	4	4	3	1	1	1	1

$S = 4$, $D = 3$, $m = 3$

Fig. 4(a) A set of 20 jobs

M_1	S	1	4	S	7	10	S	13	14
M_2	S	2	5	S	8	11			
M_3	S	3	6	S	9	12			

Fig. 4(b) Vertically scheduling 14 jobs in 7 batches

M_1	S	1	4	S	7	10	S	13	14
M_2	S	2	5	20	18	S	8	11	
M_3	S	3	6	19	17	S	9	12	

Fig. 4(c) Vertically placing jobs 20, 19, 18, 17 on machines M_2 and M_3

M_1	S	1	4	S	7	10	S	13	14	16
M_2	S	2	5	20	18	S	8	11	15	
M_3	S	3	6	19	17	S	9	12		

Fig. 4(d) Vertically placing jobs 16 and 15 on machines M_1 and M_2

Integers within the cells in the depicted schedules are job indices.

Fig. 4 An example illustrating Algorithm OPT-m

Notice that Algorithm OPT-m covers some solutions which are obviously not optimal and can be skipped. Referring to Fig. 3, it is obvious that 10 batches are not optimal. After considering 8 batches, the next candidate for optimality is clearly 11 batches. The general rule is that after all the "full" batches have been scheduled, the remaining jobs should be allocated to x batches ($x \leq m$) "as equally as possible". However, this will not affect the output nor the running time of the algorithm.

Let us examine the running time of the algorithm. Sorting the jobs in descending order of the b_i's takes $O(n \log n)$ time. The algorithm takes $O(n)$ iterations, and each iteration takes linear time to schedule the jobs. Thus, the overall running time of the algorithm is $O(n^2)$, same as Algorithm OPT-1.

Theorem 3 *Algorithm OPT-m correctly outputs a schedule with the minimum sum of completion times on m parallel and identical machines.*

Proof. Let h and k' be as defined in Algorithm OPT-m. It is clear that the optimal schedule will consist of k batches, for some $1 \leq k \leq k'$. Let us now focus on the optimal schedule with k batches. Assume that the jobs are sorted in descending order of their extended processing times, or equivalently, in descending order of the b_i's. The optimal schedule will execute the jobs with the largest extended processing times in their basic processing time mode, while the remaining jobs will all be executed in their extended processing time mode.

If the number of jobs is less than or equal to kh, then the optimal schedule will vertically schedule these jobs in k batches, and we are done. Otherwise, it will vertically schedule the first kh jobs in k batches, and schedule the remaining jobs in their extended processing

times mode. Let us now focus on the remaining jobs. Let $k = qm + r$ and $r' = m - r$. Then the first r machines have $q+1$ batches while the last r' machines have q batches, $r + r' = m$. In other words, the first r machines have $(q+1)h$ jobs while the last r' machines have qh jobs. Thus, we need to assign $r'h$ jobs to the last r' machines before we assign any jobs to the first r machines. If the number of remaining jobs is less than or equal to $r'h$, then we vertically place these jobs in the last r' machines. Otherwise, we vertically place $r'h$ jobs on the last r' machines, so that each machine now has $(q+1)h$ jobs. The remaining jobs will then be vertically placed on the m machines. ■

4 Approximation algorithm

In this section we give an approximation algorithm for the general case of the problem $1 \mid S, p_i = a_i + stp(s_i - s_i^l - D)b_i \mid \sum_{i=1}^{n} C_i$. We first give the algorithm for a single machine, and then extend the algorithm to m parallel and identical machines.

The algorithm for a single machine first sorts the jobs in ascending order of their basic processing times; i.e., $a_1 \leq a_2 \leq \cdots \leq a_n$. It then schedules the n jobs into as many batches as needed, say k', without causing any job to execute in its extended processing time mode. Let this schedule be S_t and its sum of completion times be SCT. For each k between 1 and k', it tries to form a schedule with k batches. Finally, it outputs the schedule with the minimum sum of completion times from among the k' schedules.

Suppose we are scheduling the n jobs in k batches. The algorithm first schedules as many jobs as it can in k batches, say n', without causing any job to execute in its extended processing time mode. It then reorders the remaining $n - n'$ jobs in ascending order of their extended processing times. Scanning the list of jobs in this order, the algorithm tries to insert each job at the end of the j-th batch, $1 \leq j \leq k$, that will give the smallest sum of completion times. At the end, we get a schedule with exactly k batches.

Algorithm APP-1

Input: A set of n jobs with basic processing times $p_i = a_i$ and extended processing times $p_i = a_i + b_i$, $1 \leq i \leq n$. A single machine. Setup time S and threshold D.

Output: A schedule S_t on a single machine.

Method:

1. Sort the jobs in ascending order of their basic processing times; i.e., $a_1 \leq a_2 \leq \cdots \leq a_n$.

2. Let S_t be the schedule where the first x_1 jobs are scheduled in the first batch, the next x_2 jobs are scheduled in the second batch, and so on, until the k'-th batch. The jobs are all scheduled in their basic processing times mode. Let SCT be the sum of completion times of the schedule S_t.

3. For $k = k' - 1$ to 1 do

4. { Let S'_t be the schedule where the first x_1 jobs are scheduled in the first batch, the next x_2 jobs are scheduled in the second batch, and so on, until the k-th batch. All these jobs are scheduled in their basic processing times. Let S'_t contain n' jobs.

5. Sort the remaining $n - n'$ jobs in ascending order of their extended processing times; i.e., $a_{n'+1} + b_{n'+1} \leq a_{n'+2} + b_{n'+2} \leq \cdots \leq a_n + b_n$.

6. For $j = n' + 1$ to n do

7. { Schedule job j at the end of the l-th batch, $1 \leq l \leq k$, such that the l-th batch gives the smallest sum of completion times. }

8. Compute the sum of completion times of the schedule S'_t and assign it to SCT'.

9. if $(SCT' < SCT)$ then { $S_t \leftarrow S'_t$. $SCT \leftarrow SCT'$. }. }

10. Return schedule S_t.

Let us examine the running time of the algorithm. Sorting the jobs in ascending order of their basic processing times takes $O(n \log n)$ time. The algorithm has $O(n)$ iterations.

In each iteration, it needs to sort the jobs in ascending order of their extended processing times, which takes $O(n \log n)$ time. It then schedules the remaining jobs at the end of the batch that gives the smallest sum of completion times, which takes $O(n^2)$ time. Therefore, the overall runnig time of the algorithm is $O(n^3)$.

Theorem 4 *If $b_i \leq D$ for all $1 \leq i \leq n$, then Algorithm APP-1 has a performance guarantee of 2.*

Proof. Let k' be as defined in Algorithm APP-1. It is clear that the optimal schedule will consist of k batches, for some $1 \leq k \leq k'$. Let us consider the schedule produced by Algorithm APP-1 for k batches. Algorithm APP-1 schedules the n' jobs with the smallest basic processing times in the k batches, and the remaining jobs are scheduled with extended processing times. The optimal algorithm will, in the best case, schedule $n' + k$ jobs with basic processing times, and the remaining jobs must be scheduled with extended processing times. Since $b_i \leq D$ for all $1 \leq i \leq n$, the difference in the processing times of the two schedules is at most kD. Since the sum of completion times of the optimal schedule is at least kD, we have the performance ratio is at most 2. ∎

We now consider the case of m parallel and identical machines. First, we need to introduce some notations. By "putting job j on machine i", we mean to put job j at the end of the l-th batch of machine i, where the l-th batch gives the smallest sum of completion times. By "vertically putting jobs $n', \ldots, n$ on machines $i, \ldots, m$", we mean putting these jobs on the machines in a wrap-around manner. "Scheduling jobs vertically" has the same meaning as before. That is, jobs are scheduled in their basic processing times on the machines in a wrap-around manner.

Algorithm APP-m

Input: A set of n jobs with basic processing times $p_i = a_i$ and extended processing times $p_i = a_i + b_i$, $1 \leq i \leq n$. m parallel and identical machines. Setup time S and threshold D.

Output: A schedule S_t on m parallel and identical machines.

Method:

1. Sort the jobs in ascending order of their basic processing times; i.e., $a_1 \leq a_2 \leq \cdots \leq a_n$.

2. Let S_t be the schedule where the n jobs are scheduled vertically in as many batches as necessary, say k' batches. Let SCT be the sum of completion times of the schedule S_t.

3. For $k = k' - 1$ to 1 do

4. { Let S'_t be the schedule where the n jobs are scheduled vertically in k batches.

5. if all the jobs are scheduled, then { Compute the sum of completion times of the schedule S'_t and assign it to SCT'. Goto step (9). }

6. Suppose the first $n' < n$ jobs have been scheduled. Sort the remaining jobs in ascending order of their extended processing times; i.e., $a_{n'+1} + b_{n'+1} \leq a_{n'+2} + b_{n'+2} \leq \cdots \leq a_n + b_n$.

7. if $(k \leq m)$ then { Vertically put the jobs $n'+1, n'+2, \ldots, n$ on the machines $1, 2, \cdots, k$. }.

   else { Let $k = qm + r$ and $r' = m - r$. Vertically put the jobs $n'+1, n'+2, \ldots, n''$ on the machines $r+1, r+2, \ldots, m$, and then vertically put the jobs $n''+1, n''+2, \ldots, n$ on the machines $1, 2, \ldots, m$. }.

8. Compute the sum of completion times of the schedule S'_t and assign it to SCT'.

9. if $(SCT' < SCT)$ then { $S_t \leftarrow S'_t$. $SCT \leftarrow SCT'$. }. }

10. Return schedule S_t.

Let us examine the running time of the algorithm. Sorting the jobs in ascending order of their basic processing times takes $O(n \log n)$ time. The algorithm has $O(n)$ iterations. In each iteration, it needs to sort the jobs in ascending order of their extended processing times, which takes $O(n \log n)$ time. It then schedules the remaining jobs at the end of the

batch that gives the smallest sum of completion times, which takes $O(n^2)$ time. Therefore, the overall runnig time of the algorithm is $O(n^3)$, same as Algorithm APP-1.

The performance ratio of Algorithm APP-m is also 2 when $b_i \leq D$ for all $1 \leq i \leq n$. The proof is identical to the single machine case and will be omitted.

Theorem 5 *If $b_i \leq D$ for all $1 \leq i \leq n$, then Algorithm APP-m has a performance guarantee of 2.*

5 Conclusion

In this paper we have shown the strong NP-hardness of the problem $1 \mid S, p_i = a_i + stp(s_i - s_i^l - D)b_i \mid \sum_{i=1}^{n} C_i$. We gave an $O(n^2)$ time optimal algorithm for the special case where $a_i = a$ for all $1 \leq i \leq n$, as well as an $O(n^3)$ time approximation algorithm for the general case. We show that the approximation algorithm has a performance guarantee of 2 when $b_i \leq D$ for all $1 \leq i \leq n$.

Several problems remain open. What is the performance guarantee of the approximation algorithm for the general case? Is there any better approximation algorithm? Is there any optimal algorithm for the sum of weighted completion times objective when $a_i = a$ for all $1 \leq i \leq n$?

Acknowledgements

This research was supported in part by The Hong Kong Polytechnic University under grant number A628 from the *Area of Strategic Development in China Business Services*. The first author was also partially supported by the National Science Foundation under grant DMI-0300156.

References

[1] A. Allahverdi, C.T. Ng, T.C.E. Cheng and M.Y. Kovalyov, A survey of scheduling problems with setup times or costs, *European Journal of Operational Research* (to appear).

[2] M.S. Barketau, T.C.E. Cheng and C.T. Ng, Batch scheduling of step deteriorating items, Working Paper, Department of Logistics, The Hong Kong Polytechnic University, 2005.

[3] T.C.E. Cheng, Q. Ding and B.M.T. Lin, A concise survey of scheduling with time-dependent processing times, *European Journal of Operational Research*, 152:1-13, 2004.

[4] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, 1979.

[5] C.N. Potts and M.Y. Kovalyov, Scheduling with batching: a review, *European Journal of Operational Research*, 120:228-249, 2000.