Batch Scheduling of Step Deteriorating Jobs

M.S. Barketau[a,b,c], T.C.E. Cheng[b,*], C.T. Ng[b],
Vladimir Kotov[a], Mikhail Y. Kovalyov[a,c]

[a]*Belarusian State University, Nezavisimosti 4, 220030 Minsk, Belarus*

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

[c]*United Institute of Informatics Problems, National Academy of Sciences of Belarus, Surganova 6, 220012 Minsk, Belarus*

Abstract

In this paper we consider the problem of scheduling n jobs on a single machine, where the jobs are processed in batches and the processing time of each job is a step function depending on its waiting time, which is 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 job i, if its waiting time is less than a given threshold value D, then it requires a basic processing time a_i; otherwise, it requires an extended processing time $a_i + b_i$. The objective is to minimize the completion time of the last job. We first show that the problem is NP-hard in the strong sense even if all b_i are equal, it is NP-hard even if $b_i = a_i$ for all i, and it is non-approximable in polynomial time with a constant performance guarantee $\Delta < 3/2$ unless $\mathcal{P} = \mathcal{NP}$. We then present $O(n \log n)$ and $O(n^{3F-1} \log n / F^F)$ algorithms for the case where all a_i are equal and for the case where there are F, $F \geq 2$, distinct values of a_i, respectively. We further propose an $O(n^2 \log n)$ approximation algorithm with a performance guarantee $\Delta \leq 1 + \lfloor \frac{m^*}{2} \rfloor / m^* \leq 3/2$ for the general problem, where m^* is the number of batches in an optimal schedule. All the above results apply or can be easily modified for the corresponding open-end bin packing problem.

Key Words: batching, scheduling, deterioration, open-end bin packing.

[*]Corresponding author

1 Introduction

Consider the production of custom industrial steel products such as vault doors or boiler covers, whereby raw iron is first converted into a batch of iron ingots in an electric furnace. The ingots are then sequentially processed on a machine into different products. An ingot has to reach a threshold temperature before it can be processed by the machine into a product. The longer an ingot waits for processing (i.e., the later it is processed by the machine), the cooler it becomes. After waiting for a period of time, an ingot needs to be reheated to the threshold temperature before the machine can work on it. Consequently, it requires extra time to produce each product from an ingot that has waited longer than a certain time interval.

In this paper we study a scheduling model that deals with the above situation. In the next section we formulate the problem, discuss its properties and give a literature review of the related topics.

2 Problem description

There are jobs of the set $N = \{1, \ldots, n\}$ to be scheduled for processing on a single machine. The jobs are processed in batches. Each batch is preceded by a setup time S. Job processing times are time dependent as follows. Assume that a job i is assigned to a batch B, the first job in the batch B starts its processing at time s_B, and job i starts its processing at time s_i. The processing time of job i is a step function depending on the difference $s_i - s_B$:

$$p_i = \begin{cases} a_i, & \text{if } s_i - s_B < D, \\ a_i + b_i, & \text{if } s_i - s_B \geq D. \end{cases}$$

Here D is a given *threshold value.* We call a_i and $a_i + b_i$ the *basic* and *extended processing time* of job i, respectively. We also call a job with a basic processing time and an extended processing time a *basic* and *extended* job, respectively. We call the last basic job in a batch a *straddling* job, which means that it can straddle the threshold value D (but this does not necessarily occur). The problem is to assign the jobs into batches and to sequence them so that the maximum job completion time, $C_{\max} = \max_i\{C_i\}$, is minimized, where C_i is the completion time of job i. We denote this problem by $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$, where $u_1(x) = \begin{cases} 1, x \geq 0, \\ 0, x < 0 \end{cases}$ is the *Heaviside step function.* All numerical data are assumed to be nonnegative integer numbers.

The described model falls into two categories: scheduling problems with start time dependent processing times, and batch scheduling problems. These two categories of scheduling problems have been extensively researched over the last two decades. However, to the best of our knowledge, no work has been done on models combining both aspects of batching and time-dependent processing times with regard to the start of the batch.

The first publication on scheduling jobs with start time dependent processing times was due to Melnikov and Shafransky [11] in 1980. Surveys of this area of research can be found in Gawiejnowicz [6], Aidaee and Wormer [1] and Cheng et al. [4]. In the classical model with start time dependent job processing times, no batching is allowed and the job or operation processing time is a nondecreasing function of its start time. A step function of the job processing time, $p_i = a_i + u_1(s_i - D)b_i$, was studied by Alidaee and Womer [1] and Cheng and Ding [3] for the single-machine environment. Cheng and Ding [3] presented enumeration algorithms for $C_{\max}$ and $\sum C_i$ minimization. Alidaee and Womer [1] suggested an $O(n \log n)$ algorithm to minimize $C_{\max}$ for the case where all $a_i = a$. Our $O(n \log n)$ algorithm in Section 4 is similar to this algorithm.

Batch scheduling problems have been reviewed by Potts and Kovalyov [12] and Allahverdi et al. [2]. The batch scheduling models most relevant to this study were studied by Cheng et al. [5] and Inderfurth et al. [7]. The objective functions in [5] include the costs for the number of batches and for the delivery waiting times $C_B - C_i$, where C_B is the completion time of the batch to which job i belongs. The model in [7] addresses a two-stage batching process of work and re-work and incorporates the costs for the waiting times between the work and rework operations on the same job.

An open-end bin packing problem studied by Leung et al. [10] is similar to the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$. In the former problem, n items with sizes $a_1, \ldots, a_n$ have to be packed into the minimum number of bins of the same capacity D. The "open-end" characteristic is that if the capacity D of a bin is not tightly reached, then any item can be placed in this bin. The open-end bin packing problem becomes a special case of the decision version of the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ if we choose sufficiently large values of b_i such that only schedules with no extended jobs are acceptable. In this case, the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ reduces to minimizing the number of batches, subject to no job is extended. Leung et al. [10] proved that the open-end bin packing problem is NP-hard in the strong sense and presented a fully polynomial asymptotic approximation scheme based on the algorithm of Karmarkar and Karp [8] for the classical bin packing problem.

The remaining part of the paper is organized as follows. In Section 3 we establish properties of an optimal schedule for the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$. We further show that the problem is NP-hard in the strong sense even if all b_i are equal, it is NP-hard even if $b_i = a_i$ for all i, and it is non-approximable in polynomial time with a constant performance guarantee $\Delta < 3/2$ unless $\mathcal{P} = \mathcal{NP}$. We present in Section 4 $O(n \log n)$ and $O(n^{3F-1} \log n/F^F)$ algorithms for the case where all a_i are equal and for the case where there are F distinct values of a_i, respectively. In Section 5 we describe an $O(n^2 \log n)$ approximation algorithm with a performance guarantee $\Delta \leq 1 + \lceil \frac{m^*-1}{2} \rceil / m^* \leq 3/2$ for the general problem, where m^* is the number of batches in an optimal schedule. All the above results apply or can be easily modified for the corresponding open-end bin packing problem. We summarize in Section 6 the results of this paper and suggest directions for future research.

3 Properties of an optimal schedule and complexity results

We begin with establishing some useful properties of an optimal schedule for the problem under study.

Given a schedule, denote by L (left), M (middle), and E (extended) the sets of the basic non-straddling, straddling and extended jobs, respectively. Observe that for any schedule,

$$C_{\max} = \sum_{j=1}^{n} a_j + mS + \sum_{j \in E} b_j,$$

where m is the number of batches. The variable part of the above formula is $mS + \sum_{j \in E} b_j$. Hence, the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ is equivalent to a modification of the open-end bin packing problem, which is to minimize a linear combination of the number of bins and the total weight of the non-packed items.

By the definition of the straddling job, $|M| = m$ for a schedule with m batches. Further properties are established in the following theorems.

Theorem 1 *There exists an optimal schedule for the problem* $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ *that simultaneously satisfies the following properties:*

(a) *all the extended jobs are scheduled last in the last batch;*

(b) *the straddling jobs have the largest basic processing times* a_i *among all the basic jobs;*

(c) *the straddling jobs have the largest* b_i *values among all the straddling and extended jobs.*

Proof. Since the positions of the extended jobs do not affect the variable part of the $C_{\max}$ value, there exists an optimal schedule satisfying property (a). Consider such a schedule and assume that property (b) or property (c) does not hold for it. Then the set

$$V_M = \{i \in M \mid (\exists j \in L : a_i < a_j) \text{ or } (\exists r \in E : b_i < b_r)\}$$

must be non-empty.

We now describe a job interchange technique that keeps the schedule optimal and satisfying property (a), reduces the cardinality of the set V_M and has a finite number of iterations. It is obviously sufficient for the proof.

Let j^* be a job in the set L with the largest a_j value and let r^* be a job in the set E with the largest b_j value. Consider $i \in V_M$. This inclusion implies 1) $a_i < a_{j^*}$ or 2) $b_i < b_{r^*}$.

Consider case 1) $a_i < a_{j^*}$. Interchange jobs i and j^*, i.e., reset $L := (L\backslash\{j^*\}) \cup \{i\}$ and $M := (M\backslash\{i\}) \cup \{j^*\}$. If $b_{j^*} \geq b_{r^*}$, then, by definition, we should reset $V_M := V_M\backslash\{i\}$ reducing the cardinality of this set by one. If $b_{j^*} < b_{r^*}$, then interchanging jobs j^* and r^* decreases the makespan, which contradicts the optimality of the schedule. Thus, $b_{j^*} \geq b_{r^*}$.

Case 2) $b_i < b_{r^*}$ similarly contradicts the optimality of the schedule. ■

Our next theorem shows that determining the straddling jobs is not a crucial issue in solving the problem. Let the jobs be numbered in the order that $a_1 \leq \cdots \leq a_n$ and $b_i \geq b_{i+1} \geq \cdots \geq b_j$ if $a_i = a_{i+1} = \cdots = a_j$ for $1 \leq i < j \leq n$, which by analogy with the well-known Shortest Processing Time (SPT) order, we call this the SPT order.

We skip considering a trivial situation where there exists an optimal schedule, in which there is no basic non-straddling job, i.e., $L = \phi$. In this case, due to Theorem 1, an optimal schedule with m batches is such that the straddling jobs have the largest b_i values among all the jobs, and the remaining jobs are scheduled as extended in the last batch. Furthermore, an optimal number of batches, denoted as $m^{(L=\phi)}$, is equal to the number of values b_i, $i = 1, \ldots, n$, such that $b_i > S$. Corresponding schedule with $m^{(L=\phi)}$ batches will always be considered as a candidate for an optimal one. Thus, we assume that $L \neq \phi$.

Theorem 2 *There exists an optimal schedule for the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ with m batches that satisfies property (a) in Theorem 1 and if $k = \max\{i|i \in L\}$, then $k \leq n - m$ and the set $M := M(k)$ comprises m jobs from the set $\{k+1, k+2, \ldots, n\}$ with the largest values of b_i.*

Proof. Consider an optimal schedule with m batches that satisfies properties (a),(b) and (c) in Theorem 1, and consider the corresponding job sets L, M and E. Let $k^0 = \max\{i|i \in L\}$.

Assume $a_{k^0-1} < a_{k^0}$, where $a_0 := -1$. In this case, due to property (b), $M \subseteq \{k^0+1, k^0+2, \ldots, n\}$, and due to property (c), M comprises m jobs of the latter set with the largest b_j values. Hence, $k^0 \leq n - m$ and we can set $k = k^0$.

It remains to consider the case $a_{i^0-1} < a_{i^0} = a_{i^0+1} = \cdots = a_{k^0-1} = a_{k^0}$. Denote $J := \{i^0, i^0+1, \ldots, k^0\}$. Let $J = J_{ME} \cup J_L$, where set J_{ME} comprises the straddling and extended jobs in J, and set J_L comprises the basic non-straddling jobs in J. Since the jobs in set J have equal a_i values, we can interchange them so that the jobs in set J_L have the largest b_i values in set J, the jobs in set J_{ME} have the smallest b_i values in set J, and the $C_{\max}$ value of the new schedule does not increase. Due to the original SPT job numbering, for this new optimal schedule we have $\{i^0, i^0+1, \ldots, i^0+|J_L|-1\} \subseteq L$ and $M \subseteq \{i^0+|J_L|, i^0+|J_L|+1, \ldots, n\}$. By property (c), set M comprises m jobs of the latter set with the largest b_i values. Hence, $i^0 + |J_L| - 1 \leq n - m$ and we can set $k = i^0 + |J_L| - 1$. ■

Theorem 2 shows that under the SPT numbering, the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ reduces to $n(n-1)/2$ subproblems, denoted as $\Pi(m,k)$, $m = 1, \ldots, n-1$, $k = 1, \ldots, n-m$, where the number of batches m and the largest non-straddling job index $k = \max\{j|j \in L\}$ are fixed and the m straddling jobs in set $M(k)$ are the jobs with the largest b_i values in the set $\{k+1, \ldots, n\}$. Problems $\Pi(m,k)$ will be used to develop an approximation algorithm for the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ in Section 5.

Observe that if $a_1 \geq D$, then no job can be basic non-straddling, and according to Theorem 2, there exists an optimal schedule in which m jobs with the largest b_j values are straddling and the remaining jobs are extended. In this case, the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ can be solved in $O(n \log n)$ time by comparing $C_{\max}$ values of the above mentioned schedules with $m = 1, \ldots, n$ batches. In the sequel, we assume that $a_1 \leq D - 1$.

The following theorem consisting of three parts establishes that the problem under study is computationally difficult with regard to finding exact and approximate solutions.

Theorem 3 *The problem* $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ *is*
1) NP-hard in the strong sense even if $b_i = b$, $i = 1, \ldots, n$;
2) NP-hard even if $b_i = a_i$, $i = 1, \ldots, n$;
3) non-approximable in polynomial time with a constant (independent of problem parameters) performance guarantee $\Delta < 3/2$ *unless* $\mathcal{P} = \mathcal{NP}$ *even if* $b_i = b$, $i = 1, \ldots, n$.

Proof. We use reductions from the strongly NP-complete problem 3-Partition and the NP-complete problem Partition.

3-Partition: Given $3q + 1$ positive integer numbers $h_1, \ldots, h_{3q}$ and H such that $H/4 < h_i < H/2$, $i = 1, \ldots, 3q$, and $\sum_{i=1}^{3q} h_i = qH$, is there a partition of the set $\{1, \ldots, 3q\}$ into q disjoint sets $X_1, \ldots, X_q$ such that $\sum_{i \in X_l} h_i = H$ for $l = 1, \ldots, q$?

Partition: Given $q+1$ positive integer numbers $h_1, \ldots, h_q$ and H such that $\sum_{i=1}^{q} h_i = 2H$, is there a subset $X \subset Q := \{1, \ldots, q\}$ such that $\sum_{i \in X} h_i = H$?

Part 1). Given an instance of 3-Partition, we construct the following instance of the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ with $b_i = b$, $i = 1, \ldots, n$. There are $4q$ jobs. Among them there are $3q$ *partition* jobs with parameters $a_i = h_i$ and $b_i = Y$, $i = 1, \ldots, 3q$, and q *enforcer* jobs with parameters $a_i = H+1$ and $b_i = Y$, $i = 3q+1, \ldots, 4q$, where $Y := q(3H+1)$ is an upper bound on the $C_{\max}$ value. The setup time is $S = H$ and the threshold value is $D = H + 1$.

It can be easily verified that 3-Partition has a solution $X_1, \ldots, X_q$ if and only if there exists a schedule for the constructed instance of the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ with value $C_{\max} \leq Y$. Such a schedule is given in Fig. 1. There are q batches and no job is extended in this schedule.

Part 2). Given an instance of Partition, construct the following instance of the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ with $b_i = a_i$, $i = 1, \ldots, n$. There are $q + 2$ jobs. Among them there are q *partition* jobs with parameters $a_i = b_i = h_i$, $i = 1, \ldots, q$, and two *enforcer*

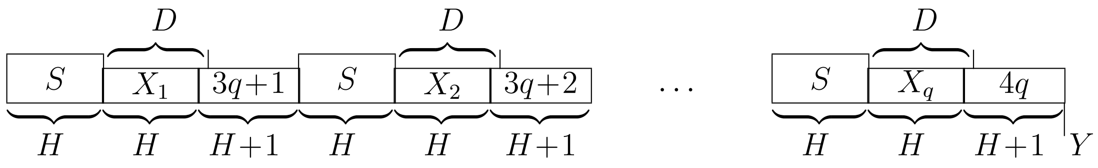

Figure 1: Structure of a schedule for part 1).

jobs $q+1$ and $q+2$ with parameters $a_{q+1} = b_{q+1} = a_{q+2} = b_{q+2} = H+1$. The setup time is $S = H$ and the threshold value is $D = H+1$.

It can be easily verified that Partition has a solution X if and only if there exists a schedule for the constructed instance of the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ with value $C_{\max} \le 6H+2$. Such a schedule is given in Fig. 2. There are two batches and no job is extended in this schedule.

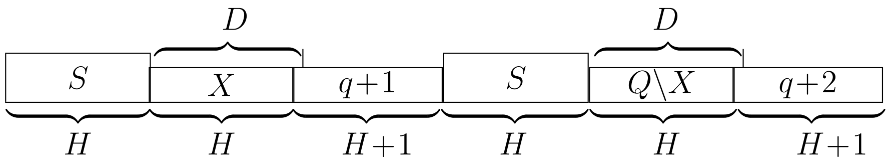

Figure 2: Structure of a schedule for part 2).

Part 3). Let C^* denote the optimal solution value for the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$. Assume that there exists a polynomial algorithm A for this problem that delivers a solution with a value C^A such that $C^A/C^* \le \Delta$ for a given constant $\Delta < 3/2$.

To facilitate discussion, we introduce a new constant $\delta > 0$ such that $\Delta = 3/2 - \delta$.

Observe that Partition remains NP-complete if we assume $H \ge 1/\delta$. Otherwise, all the numbers in this problem are bounded by a constant and it becomes polynomially solvable.

Given an instance of Partition, in which $H \ge 1/\delta$, we construct the following instance of the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ with $b_i = b$, $i = 1, \ldots, n$, denoted as I. There are $q+2$ jobs. Among them there are q *partition* jobs with parameters $a_i = h_i$, $b_i = H^2$, $i = 1, \ldots, q$, and two *enforcer* jobs $q+1$ and $q+2$ with parameters $a_{q+1} = a_{q+2} = H+1$ and $b_{q+1} = b_{q+2} = H^2$. The setup time is $S = H^2$ and the threshold value is $D = H+1$.

It can be easily verified that Partition has a solution X if and only if there exists a schedule for the instance I with value $C_{\max} \le Y := 2H^2 + 4H + 2$. Such a schedule is given in Fig. 3. There are two batches and no job is extended in this schedule.

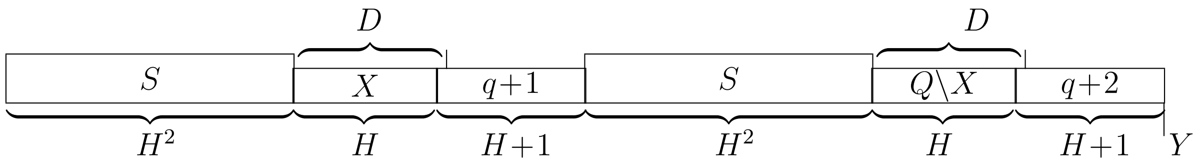

Figure 3: Structure of a schedule for part 3).

Furthermore, if PARTITION does not have a solution, then at least one of the following three cases takes place for any schedule in instance I: a) there is one batch and at least two extended jobs, b) there are two batches and at least one extended job, and c) there are three or more batches. In each of these cases, $C^* \geq 3H^2 + 4H + 2$.

Apply algorithm A for the instance I. If PARTITION has a solution, then taking into account $C^* \leq Y$ and $\delta \geq 1/H$ we obtain

$$C^A \leq \Delta C^* \leq \Delta Y = (3/2 - \delta)Y = 3H^2 + 6H + 3 - \delta(2H^2 + 4H + 2) < 3H^2 + 4H + 2.$$

If PARTITION does not have a solution, then $C^A \geq C^* \geq 3H^2 + 4H + 2$. Therefore, value C^A uniquely determines whether PARTITION has a solution or not. Since we assumed that this value can be found in polynomial time, PARTITION can be solved in polynomial time as well, which is unlikely unless $\mathcal{P} = \mathcal{NP}$. ■

Parts 1) and 3) in Theorem 3 imply the following corollary.

Corollary 1 *The open-end bin packing problem is NP-hard in the strong sense and it is non-approximable in polynomial time with a constant performance guarantee* $\Delta < 3/2$ *unless* $\mathcal{P} = \mathcal{NP}$.

Recall that the strong NP-hardness of the open-end bin packing problem has already been known due to Leung et al. [10].

4 Polynomially solvable cases

We first assume that all the jobs have equal basic processing times: $a_i = a$, $i = 1, \dots, n$. Similar to Theorem 1, a job interchange technique can be used to show that under this assumption, there exists an optimal schedule in which all the jobs are processed in the *Longest Processing Time (LPT)* order of their extended processing times b_i. Furthermore, each batch, with the possible exception of the last batch, contains the same number of basic jobs, which is the maximum number of basic jobs, denoted as k^*, permitted by the threshold value D : $k^* = \lceil D/a \rceil$. The problem reduces to finding the optimal number of batches, m^*. This discussion justifies algorithm A given below.

Algorithm A (for the case $a_i = a$, $i = 1, \ldots, n$)

Step 1 (Initialization) Re-number the jobs so that $b_1 \geq \cdots \geq b_n$. Calculate $k^* = \lceil \frac{D}{a} \rceil$ and an upper bound on the optimal number of batches, $U = \lceil \frac{n}{k^*} \rceil$. Set the initial value $m^* = U$. Set $m = 1$.

Step 2 (Determination of m^*) If $S \geq \sum_{i=mk^*+1}^{\min\{(m+1)k^*,n\}} b_i$, then re-set $m^* := m$ and stop. Otherwise, re-set $m := m + 1$. If $m = U$, then stop ($m^* = U$ in this case); otherwise repeat Step 2. ■

Theorem 4 *Algorithm A solves the problem* $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ *with* $a_i = a$, $i = 1, \ldots, n$, *and runs in* $O(n \log n)$ *time.*

Proof. A justification for algorithm A has already been given. The running time of this algorithm is determined by sorting the jobs in the LPT order of their extended processing times, which requires $O(n \log n)$ time. ■

Algorithm A is similar to the $O(n \log n)$ time algorithm of Alidaee and Womer [1] for scheduling deteriorating jobs with no batching.

We now consider another special case of the problem in which there are F, $F \geq 2$, distinct basic processing times a_i. Assume that the jobs are partitioned into F families such that family f comprises n_f jobs, denoted as (j, f), $j = 1, \ldots, n_f$, with the same basic processing time a'_f, $f = 1, \ldots, F$. We have $\sum_{f=1}^{F} n_f = n$. Assume without loss of generality that all a'_f are integer numbers. We present an enumeration algorithm, denoted as ENUM, for this special case. It uses as a subroutine in the algorithm of Leung [9] for the following problem, denoted as $P(m, F)$.

Problem $P(m, F)$: Given F families of jobs such that family f comprises jobs with the same processing time a'_f, $f = 1, \ldots, F$, can all these jobs be scheduled non-preemptively on m parallel identical machines so that $C_{\max} \leq D - 1$ for a given number $D > 0$? If they can, construct corresponding schedule.

Problem $P(m, F)$ can be solved in $O(n^{2(F-1)} \log m)$ time, where n is the total number of jobs, see [9].

Assume that the jobs in each family f are numbered in the LPT order of their extended processing times $b_i : b_{(1,f)} \geq b_{(2,f)} \geq \cdots \geq b_{(n_f,f)}$, $f = 1, \ldots, F$. Algorithm ENUM is justified by the following observations, which were proved in Theorem 1 or can be similarly proved by a job interchange technique. First, there exists an optimal schedule in which all the extended jobs are scheduled last in the last batch. Second, the extended jobs and the straddling jobs are the jobs with the largest indices in the same family. Third, the straddling jobs are the jobs with the largest b_i values among all the straddling and extended jobs, irrespective of their families. We consider only schedules that satisfy these properties.

Algorithm ENUM

Step 1 Generate a set X^0 of possible candidates for the set of the extended and straddling jobs in an optimal schedule:

$$X^0 := \{X_{(x_1,\ldots,x_F)} \mid x_f = 1, \ldots, n_f + 1,\ f = 1, \ldots, F\},$$

where

$$X_{(x_1,\ldots,x_F)} := \{(j, f) \mid x_f \leq n_f,\ j = x_f, x_f + 1, \ldots, n_f,\ f = 1, \ldots, F\}.$$

If $x_f = n_f + 1$, then no job of family f is present in the set $X_{(x_1,\ldots,x_F)}$.

The number of subsets $X_{(x_1,\ldots,x_F)}$ in the set X^0, denoted as $\mathcal{K}(X^0)$, can be evaluated as $\mathcal{K}(X^0) \leq \prod_{f=1}^{F}(n_f + 1)$. Since $\sum_{f=1}^{F} n_f = n$ and $\prod_{f=1}^{F} n_f$ is maximized when $n_f = n/F$, $f = 1, \ldots, F$, we further have $\mathcal{K}(X^0) \leq O(n^F/F^F)$.

Let Σ^* denote the set of the candidates for an optimal schedule. Initialize $\Sigma^* := \phi$.

For $m = 1, \ldots, n$, perform Step 2.

Step 2 For a set $X_{(x_1,\ldots,x_F)} \in X^0$, perform the following computations.

Determine whether there exists a schedule such that the jobs of the set $N \backslash X_{(x_1,\ldots,x_F)}$, where N is the set of all jobs, can be scheduled as basic in this schedule. For these purposes, solve the problem $P(m, F)$ with F families and the set of jobs $N \backslash X_{(x_1,\ldots,x_F)}$. If this problem has a solution, detect m jobs, denoted $j_1, \ldots, j_m$, with the largest values b_i in the set $X_{(x_1,\dots,x_F)}$ and create a schedule, denoted as $\sigma^m_{(x_1,\dots,x_F)}$. In this schedule, the jobs of the set $N\backslash X_{(x_1,\dots,x_F)}$ are scheduled according to the solution of the problem $P(m, F)$, i.e., the jobs assigned to machine l are basic jobs in the batch l, $l = 1, \dots, m$, jobs $j_1, \dots, j_m$ are straddling, and the jobs of the set $X_{(x_1,\dots,x_F)}\backslash\{j_1, \dots, j_m\}$ are extended. Calculate

$$C_{\max}(\sigma^m_{(x_1,\dots,x_F)}) = mS + \sum_{f=1}^{F} n_f a'_f + \sum_{i\in X_{(x_1,\dots,x_F)}\backslash\{j_1,\dots,j_m\}} b_i.$$

Add $\sigma^m_{(x_1,\dots,x_F)}$ to Σ^*. Remove $X_{(x_1,\dots,x_F)}$ from X^0 and repeat Step 2.

Step 3 Detect an optimal solution σ^* such that $C_{\max}(\sigma^*) = \min\{C_{\max}(\sigma) \mid \sigma \in \Sigma^*\}$. ■

The time complexity of algorithm ENUM is determined by its Step 2, which is repeated $n\mathcal{K}(X^0) \le O(n^{F+1}/F^F)$ times. One iteration of Step 2 requires $O(n^{2(F-1)} \log m)$ time. Therefore, the time complexity of algorithm ENUM is $O(n^{3F-1} \log n/F^F)$, which is polynomial if F is a constant.

Since the open-end bin packing problem is a special case of the problem $1|S, p_i = a_i+u_1(s_i-s_B - D)b_i|C_{\max}$ with sufficiently large b_i values, the following corollary holds.

Corollary 2 *The open-end bin packing problem with $a_i = a$, $i = 1, \dots, n$, is solvable in $O(n \log n)$ time. It is solvable in $O(n^{3F-1} \log n/F^F)$ time if there are F, $F \ge 2$, distinct item sizes a_i.*

5 Approximation algorithm for the general problem

In this section we present an $O(n^2 \log n)$ time approximation algorithm for the general problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ with a performance guarantee $\Delta \le 1 + \lfloor \frac{m^*}{2} \rfloor/m^* \le 3/2$, where m^* is the number of batches in an optimal schedule. We assume without loss of generality that the input data are all integer numbers. Recall that, given a schedule, L, M and E denote the sets of the basic non-straddling, straddling and extended jobs, respectively.

Assume that the jobs are numbered in the SPT order such that $a_1 \le \dots \le a_n$ and $b_i \ge b_{i+1} \ge \dots \ge b_j$ if $a_i = a_{i+1} = \dots = a_j$ for $1 \le i < j \le n$.

We first present an approximation algorithm, denoted as APPROX(m, k), for problem $\Pi^*(m, k)$, in which there exists an optimal schedule with m batches and the largest basic non-straddling job index $k = \max\{j|j \in L\}$, $k \le n - m$, is fixed. Problem $\Pi^*(m, k)$ differs from problem $\Pi(m,k)$ formulated in Section 3 in that the number of batches is required to be equal to m only in an optimal schedule and the set $M(k)$ of straddling jobs specified in Theorem 2 applies to an optimal schedule only. A feasible schedule for this problem may have the number of batches different from m and a different set M. Such a schedule will be constructed by algorithm APPROX(m,k).

We additionally present algorithm APPROX$(3,k,\rho)$, which is a subroutine in algorithm APPROX(m,k). Algorithm APPROX$(3,k,\rho)$ is an approximation algorithm for problem $\Pi^*(3,k,\rho)$, in which there exists an optimal schedule with three batches and the two largest basic non-straddling job indices k and ρ are fixed. We describe algorithms APPROX(m,k) and APPROX$(3,k,\rho)$ in a similar fashion.

An optimal solution for one of at most $n(n-1)/2$ problems $\Pi^*(m,k)$, $m=1,\dots,n-1$, $k=1,\dots,n-m$, such that $a_k \le D-1$, is an optimal solution for the problem $1|S,p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ (assuming $m^* \neq n$). In the case $a_k \ge D$, job k cannot be basic non-straddling by definition. By the same reason we assume $a_k \le D-1$ and $a_\rho \le D-1$ in problem $\Pi^*(3,k,\rho)$.

In the algorithm APPROX(m,k), sets Q_{small} and Q_{large} of jobs are calculated and used for the construction of an approximate schedule with at most $m + \lfloor \frac{m}{2} \rfloor$ batches, in which job k is basic non-straddling, jobs of the set $Q_{large} \cup Q_{small}$ are basic straddling and non-straddling, m jobs of the set $N \backslash (Q_{small} \cup Q_{large} \cup \{k\})$ with largest b_j values are straddling and the remaining jobs of the latter set are extended.

Similarly, in algorithm APPROX$(3,k,\rho)$, sets Q_{small} and Q_{large}, $|Q_{large}| \le 1$, are calculated and used for the construction of an approximate schedule with at most four batches. In this schedule jobs k and ρ are basic non-straddling, the only job of the set Q_{large} (if $|Q_{large}| = 1$) is basic straddling, jobs of the set Q_{small} are basic non-straddling and straddling (if $|Q_{large}| = 0$), three jobs of the set $N \backslash (Q_{small} \cup Q_{large} \cup \{k\} \cup \{\rho\})$ with the largest b_j values are straddling and the remaining jobs of the latter set are extended.

The batches are denoted as $B_1, B_2, \dots$

Algorithm APPROX$(3,k,\rho)$ (for problem $\Pi^*(3,k,\rho)$, $a_k \le D-1$, $a_\rho \le D-1$, $1 \le \rho < k \le n-3$)

Step 1 (Initialization) Set $Q_{small} = \phi$, $Q_{large} = \phi$. If $\rho = 1$, then go to Step 4.

Calculate $a'_j = \min\{a_j, \lceil \frac{D}{2} \rceil\}$, $j = 1, \dots, \rho - 1$. Re-index jobs $1, \dots, \rho - 1$ so that $b_1/a'_1 \geq \cdots \geq b_{\rho-1}/a'_{\rho-1}$. Set $r = 1$.

Step 2 (Computation of the sets Q_{small} and Q_{large}) If $a'_r < D/2$, then add job r to the set Q_{small}. If $a'_r = \lceil \frac{D}{2} \rceil$, then add job r to the set Q_{large}. If $|Q_{large}| = 1$, then computation of the set Q_{large} is completed. Continue computation of the set Q_{small}. If $\sum_{j \in Q_{small} \cup Q_{large}} a'_j \leq m(D-1) - (a_k + a_\rho)$ and $r \leq \rho - 1$, then re-set $r = r + 1$ and repeat Step 2. Otherwise, if $r = \rho - 1$ or $\sum_{j \in Q_{small} \cup Q_{large}} a'_j > m(D-1) - (a_k + a_\rho)$, then computation of the sets Q_{small} and Q_{large} is completed. Notice that $|Q_{large}| \leq 1$. If $Q_{large} = \phi$, then go to Step 4.

Step 3 (Assignment of the job of the set Q_{large}) Assign the only job of the set Q_{large} as straddling to batch B_4. Batch B_4 is considered as open for the assignment of basic non-straddling jobs.

Step 4 (Assignment of the jobs of the set $Q_{small} \cup \{k\} \cup \{\rho\}$) Assign job k to batch B_1. Assign job ρ to batch B_2. Assign jobs of the set Q_{small} to the (open) batches $B_1, \dots, B_4$ in increasing order of their indices by using the *First Fit (FF)* algorithm: a current job j is assigned to the batch with the minimal index where it can be scheduled as basic non-straddling or, if a job is assigned to batch B_4 that contains no job from Q_{large}, then as straddling. Below we will prove that all the jobs from the set Q_{small} can be scheduled as it is indicated here.

Step 5 (Assignment of the jobs of the set $N \backslash (Q_{small} \cup Q_{large} \cup \{k\} \cup \{\rho\})$) Select three jobs of the set $\{k+1, \dots, n\}$ with the largest b_j values and arbitrarily assign them as straddling to batches B_1, B_2, B_3.

Define all the remaining unassigned jobs as extended. They can be assigned to batch B_4. Compute the $C_{\max}$ value of the constructed solution. ■

Algorithm APPROX(m, k) (for problem $\Pi^*(m, k)$, $a_k \leq D - 1$, $1 \leq k \leq n - m$)

Step 1 (Initialization) Set $Q_{small} = \phi$, $Q_{large} = \phi$. If $k = 1$, then go to Step 4.

Calculate $a'_j = \min\{a_j, \lceil \frac{D}{2} \rceil\}$, $j = 1, \dots, k - 1$. Re-index jobs $1, \dots, k - 1$ so that $b_1/a'_1 \geq \cdots \geq b_{k-1}/a'_{k-1}$. Set $r = 1$.

Step 2 (Computation of the sets Q_{small} and Q_{large}) If $a'_r < D/2$, then add job r to the set Q_{small}. If $a'_r = \lceil \frac{D}{2} \rceil$ and $|Q_{large}| \leq m-2$, then add job r to the set Q_{large}. If $|Q_{large}| = m-1$, then computation of the set Q_{large} is completed. Continue computation of the set Q_{small}. If $\sum_{j \in Q_{small} \cup Q_{large}} a'_j \leq m(D-1) - a_k$ and $r < k-1$, then re-set $r = r+1$ and repeat Step 2. Otherwise, if $r = k-1$ or $\sum_{j \in Q_{small} \cup Q_{large}} a'_j > m(D-1) - a_k$, then computation of the sets Q_{small} and Q_{large} is completed. Notice that $|Q_{large}| \leq m-1$. If $Q_{large} = \phi$, then go to Step 4.

If $|Q_{large}| = 2$ and $m = 3$, then call APROX$(3, k, \rho)$ for every $\rho = 1, \ldots, k-1$, and select the schedule with the smallest C_{max} value as a solution. Stop.

Step 3 (Assignment of the jobs of the set Q_{large}) First assign arbitrarily jobs of the set Q_{large} as straddling to the batches B_i, $i = m+1, \ldots, m + \lfloor \frac{m}{2} \rfloor$. If not all of the jobs of the set Q_{large} are assigned, then assign the remaining such jobs as basic non-straddling to batches B_i, $i = m+1, \ldots, m + \lfloor \frac{m}{2} \rfloor$. After this, if two, one and no job is assigned to the same batch, then this batch is considered as closed, open for the assignment of basic non-straddling jobs and open for the assignment of basic non-straddling and straddling jobs, respectively. Notice that batches B_i, $i = m+1, \ldots, m + \lfloor \frac{m}{2} \rfloor$, can accommodate all the jobs from Q_{large} because $|Q_{large}| \leq m - 1 \leq 2\lfloor \frac{m}{2} \rfloor$.

Step 4 (Assignment of the jobs of the set $Q_{small} \cup \{k\}$) Assign job k to batch B_1. Assign jobs of the set Q_{small} to the (open) batches $B_1, \ldots, B_{m+\lfloor \frac{m}{2} \rfloor}$ in increasing order of their indices by using the FF algorithm: a current job j is assigned to the batch with the minimal index where it can be scheduled as basic non-straddling or, if a job is assigned to batch B_i with $i \geq m+1$ that contains no job from Q_{large}, then as straddling. Below we will prove that all the jobs from the set Q_{small} can be scheduled as indicated here.

Step 5 (Assignment of the jobs of the set $N \backslash (Q_{small} \cup Q_{large} \cup \{k\})$) Select m jobs of the set $\{k+1, \ldots, n\}$ with the largest b_j values and arbitrarily assign them as straddling to batches $B_1, \ldots, B_m$.

Define all the remaining unassigned jobs as extended. They can be assigned to batch $B_{m+\lfloor \frac{m}{2} \rfloor}$. Compute the $C_{\max}$ value of the constructed solution. ■

It is easy to see that, given a pair (m,k), the running time of either algorithm APPROX(m,k) or APPROX$(3,k,\rho)$ is $O(n\log n)$.

Denote the optimal $C_{\max}$ value in problem $\Pi^*(m,k)$ by $C^*_{\max}$. Let L^* and M^* denote the sets of basic non-straddling and basic straddling jobs, respectively, in an optimal schedule for this problem, which satisfy Theorem 2.

The following theorems are used in establishing the performance guarantee of algorithm APPROX(m,k).

Theorem 5 *For the sets Q_{small} and Q_{large} constructed in Step 2 of the algorithm APPROX(m,k), we have $\sum_{j\in Q_{small}\cup Q_{large}} b_j + b_k \geq \sum_{j\in L^*} b_j$.*

Proof. Consider a modification of problem $\Pi^*(m,k)$, denoted as $\Pi'(m,k)$, in which the basic processing times of the jobs $1,\dots,k-1$ are reset to $a'_j=\min\{a_j,\lceil\frac{D}{2}\rceil\}$ for $j=1,\dots,k-1$, and $a'_j=a_j$ for $j=k,k+1,\dots,n$. Denote the optimal $C_{\max}$ value of problem $\Pi'(m,k)$ by $C'_{\max}$.

Since the basic processing times in the modified problem are not greater than those in the original problem, we have $C'_{\max}\leq C^*_{\max}$.

Let L' and M' denote the sets of basic non-straddling and straddling jobs, respectively, in an optimal schedule for problem $\Pi'(m,k)$, which satisfy Theorem 2. Since the parameters of jobs $k+1,\dots,n$ are the same in problems $\Pi'(m,k)$ and $\Pi^*(m,k)$, we have $M'=M^*$.

Since the basic jobs in any schedule for problem $\Pi^*(m,k)$ remain basic in the same schedule for problem $\Pi'(m,k)$, we have $\sum_{j\in L^*} b_j \leq \sum_{j\in L'} b_j$.

It remains to show that $\sum_{j\in L'} b_j \leq \sum_{j\in Q_{small}\cup Q_{large}} b_j + b_k$. Recall that the computation of the sets Q_{small} and Q_{large} in Step 2 completes when 1) $\sum_{j\in Q_{small}\cup Q_{large}} a'_j \leq m(D-1)-a_k$ or 2) $\sum_{j\in Q_{small}\cup Q_{large}} a'_j > m(D-1)-a_k$.

In case 1), set Q_{small} consists of all the jobs from the set $\{1,\dots,k-1\}$ with $a'_j<\lceil\frac{D}{2}\rceil$ and Q_{large} consists of at most $m-1$ jobs from this set with $a'_j=\lceil\frac{D}{2}\rceil$ and the largest b_j values. Since $L'\subseteq\{1,\dots,k\}$ and L' can contain at most $m-1$ jobs with $a'_j=\lceil\frac{D}{2}\rceil$, we deduce that $\sum_{j\in L'} b_j \leq \sum_{j\in Q_{small}\cup Q_{large}} b_j + b_k$, as required.

In case 2), first observe that $\sum_{j\in L'} a'_j \leq m(D-1)$. Otherwise, at least one job in set L' is not basic non-straddling, which contradicts the definition of this set. Assume without loss of generality that jobs in set L' with $a'_j=\lceil\frac{D}{2}\rceil$ have the largest b_j values among all the jobs with $a'_j=\lceil\frac{D}{2}\rceil$. Consider sets $Q_1=(Q_{large}\cup Q_{small}\cup\{k\})\backslash L'$ and $L'_1=L'\backslash(Q_{large}\cup$

$Q_{small} \cup \{k\}$). From $\sum_{j\in L'} a'_j \le m(D-1)$ and $\sum_{j\in Q_{small}\cup Q_{large}} a'_j > m(D-1) - a_k$, it follows that $\sum_{j\in L'_1} a'_j < \sum_{j\in Q_1} a'_j$. Furthermore, $\max_{j\in L'_1}\{b_j/a'_j\} \le \min_{j\in Q_1}\{b_j/a'_j\}$ because of the job ordering in algorithm APPROX(m,k). The latter two relations imply

$$\sum_{j\in L'_1} b_j \le \min_{j\in Q_1}\{b_j/a'_j\} \sum_{j\in L'_1} a'_j < \min_{j\in Q_1}\{b_j/a'_j\} \sum_{j\in Q_1} a'_j \le \sum_{j\in Q_1} b_j.$$

We deduce that $\sum_{j\in L'} b_j < \sum_{j\in Q_{small}\cup Q_{large}} b_j + b_k$, which completes the proof. ■

Theorem 6 *For the sets Q_{small} and Q_{large} constructed in Step 2 of algorithm APPROX$(3,k,\rho)$, we have $\sum_{j\in Q_{small}\cup Q_{large}} b_j + b_k + b_\rho \ge \sum_{j\in L^*} b_j$.*

Proof. The proof is essentially the same as that for algorithm APPROX(m,k). ■

Theorem 7 *Algorithm APPROX(m,k) (algorithm APPROX$(3,k,\rho)$) assigns all the jobs of the sets Q_{large} and Q_{small} and job k (jobs k and ρ) to at most $m + \lfloor \frac{m}{2} \rfloor$ batches (four batches) with job characteristics (basic non-straddling or straddling) as it is indicated in its description.*

Proof. Note that algorithm APPROX$(3,k,\rho)$ performs the same steps as algorithm APPROX(m,k). The only difference is that in the former algorithm two largest basic non-straddling jobs are fixed instead of one.

We will provide a proof for algorithm APPROX(m,k). The proof for algorithm APPROX$(3,k,r)$ is essentially the same except that some cases are not possible.

Assume that the statement of the theorem is not satisfied. Since the jobs of the set Q_{large} and job k are obviously assigned to the required batches, assume that at least one job of the set Q_{small} was not assigned to any (open) batch. It implies that each open batch except batch B_1 contains at least two jobs from Q_{small}. This observation is used to prove the following facts.

Fact 1. *The sum of a'_i values of the non-straddling jobs in each open batch but one is at least $\frac{2(D-1)}{3}$.*

To prove Fact 1, assume that there are two open batches B_j and B_r, $j < r$, and the sum of a'_i values of the non-straddling jobs in each of which is less than $\frac{2(D-1)}{3}$. Batch B_r, $r \ne 1$, contains not less than two non-straddling jobs. At least one of these jobs should have a processing time less than $\frac{(D-1)}{3}$. This leads to a contradiction: this job should have been assigned to batch B_j.

Fact 2. *The sum of a'_i values of the non-straddling jobs in any three open batches is at least $2(D-1)$.*

To prove Fact 2, consider open batches B_j, B_r and B_v, $j < r < v$. Batch B_v, $v \neq 1$, contains at least two non-straddling jobs. None of these jobs was assigned to batches B_j and B_r because an assignment of any such job to batch B_j or B_r would make it closed for the assignment of the basic non-straddling jobs (the sum of a'_i values in the batch would be at least $D-1$). This means that the sum of a'_i values of the non-straddling jobs in $B_i \cup B_j \cup B_v$ is at least $2(D-1)$.

Consider the last job that was included in the set Q_{large} or Q_{small}. Let it be job l. By the definitions of the sets Q_{large} and Q_{small},

$$\sum_{i \in Q_{large} \cup Q_{small}} a'_i + a_k - a'_l \leq m(D-1). \tag{1}$$

There are the following two cases to consider:

1) $l \in Q_{small}$ and 2) $l \in Q_{large}$.

Consider case 1). By our contradiction assumption, if some job from Q_{small} was not assigned to a batch, then job l was not assigned to any batch either. Let P_0, P_1 and P_2 denote the sets of batches from $B_{m+1}, \cdots, B_{m+\lfloor \frac{m}{2} \rfloor}$ that contain no job from Q_{large}, one job from Q_{large} and two jobs from Q_{large}, respectively. We have $|P_0| + |P_1| + |P_2| = \lfloor \frac{m}{2} \rfloor$.

Taking into account Facts 1 and 2 and values $a'_i = \lceil \frac{D}{2} \rceil$ of the straddling jobs from Q_{large} in P_1, evaluate $\sum_{i \in B_1 \cup \cdots \cup B_m \cup P_1} a'_i \geq \frac{2(D-1)}{3}(m + |P_1|) + \lceil \frac{D}{2} \rceil |P_1|$. Furthermore, by the definitions of the sets P_0 and P_2 and because l was not assigned to any batch, $\sum_{i \in P_0 \cup P_2} a'_i \geq D(|P_0| + |P_2|)$.

Recall that $k \in B_1$. Calculate

$$\begin{aligned}
\sum_{i \in Q_{small}} a'_i + \sum_{i \in Q_{large}} a'_i + a_k - a'_l \quad &\geq \quad \sum_{i \in B_1 \cup \cdots \cup B_m \cup P_1} a'_i + \sum_{i \in P_0 \cup P_2} a'_i \\
&\geq \quad \frac{2(D-1)}{3}(m + |P_1|) + \lceil \frac{D}{2} \rceil |P_1| + D(|P_0| + |P_2|) \\
&> \quad \frac{2(D-1)}{3}(m + |P_1|) + \frac{D-1}{2}|P_1| + (D-1)(|P_0| + |P_2|) \\
&\geq \quad m(D-1),
\end{aligned}$$

which contradicts (1). Therefore, l should have been assigned to a batch.

Consider case 2). In this case we consider two subcases, namely 2a): $|P_2| = \lfloor \frac{m}{2} \rfloor$ and 2b): $|P_2| < \lfloor \frac{m}{2} \rfloor$.

In the case 2a), all the jobs from Q_{large} are assigned in pairs to batches $B_{m+1}, \ldots, B_{\lfloor \frac{m}{2} \rfloor}$. Notice that $m = 2|P_2| + 1$, $|P_0| = |P_1| = 0$. Let job $u \in Q_{small}$ be not assigned to any batch. Consequently, in any open batch including the batch with the minimal sum of a'_i values, denoted as B_{j^*}, the sum of a'_i values of the basic non-straddling jobs plus a'_u is greater than or equal to D. Taking into account Fact 1, evaluate the sum $\sum_{i\in(B_1\cup\cdots\cup B_m)\setminus B_{j^*}} a'_i + \sum_{i\in B_{j^*}} a'_i + a'_u \geq \frac{2(D-1)}{3}(m-1)+D$. By the definition of set P_2, $\sum_{i\in P_2\setminus B_{m+\lfloor\frac{m}{2}\rfloor}} a'_i + \sum_{i\in B_{m+\lfloor\frac{m}{2}\rfloor}\setminus\{l\}} a'_i \geq D(\lfloor\frac{m}{2}\rfloor-1)+\lceil\frac{D}{2}\rceil$. We have

$$\begin{aligned}
&\sum_{i\in Q_{small}} a'_i + \sum_{i\in Q_{large}} a'_i + a_k - a'_l \\
\geq\ & \sum_{i\in(B_1\cup\cdots\cup B_m)\setminus B_{j^*}} a'_i + \sum_{i\in B_{j^*}} a'_i + a'_u + \sum_{i\in P_2\setminus B_{m+\lfloor\frac{m}{2}\rfloor}} a'_i + \sum_{i\in B_{m+\lfloor\frac{m}{2}\rfloor}\setminus\{l\}} a'_i \\
\geq\ & \frac{2(D-1)}{3}(m-1)+D+D(\lfloor\frac{m}{2}\rfloor-1)+\lceil\frac{D}{2}\rceil \\
>\ & (D-1)(\frac{2}{3}(m-1)+\lfloor\frac{m}{2}\rfloor+\frac{1}{2}) \\
=\ & (D-1)(\frac{4}{3}|P_2|+|P_2|+\frac{1}{2}) = (D-1)(2|P_2|+\frac{1}{2}+\frac{|P_2|}{3}).
\end{aligned}$$

The latter expression is greater than or equal to $(D-1)m$ if $m>3$, which contradicts (1). Thus, job u must be assigned to some batch. In the case $m=3$, algorithm APPROX(m,k) calls for algorithm APPROX$(3,k,\rho)$ which is designed so that $|Q_{large}|\leq 1$ and thus case 2a) is not possible. In all other respects algorithm APPROX$(3,k,\rho)$ coincides with algorithm APPROX(m,k) and this proof can be used for its justification.

In case 2b), $|P_2|<\lfloor\frac{m}{2}\rfloor$ and consequently, $|P_1|\geq 1$.

Again, let job $u\in Q_{small}$ be not assigned to any batch. Then in the batch B_{j^*} the sum of a'_i values of basic non-straddling jobs plus a'_u is greater than or equal to D. Making use of Fact 1, evaluate $\sum_{i\in(B_1\cup\cdots\cup B_m\cup P_1)\setminus B_{j^*}} a'_i + \sum_{i\in B_{j^*}} a'_i + a'_u - a'_l \geq \frac{2(D-1)}{3}(m+|P_1|-1)+D+\lceil\frac{D}{2}\rceil(|P_1|-1)$. Taking into account that job u was not assigned to any batch from P_0 and the definition of set P_2, evaluate $\sum_{i\in P_0\cup P_2} a'_i \geq D(|P_0|+|P_2|)$. We have

$$\begin{aligned}
&\sum_{i\in Q_{small}} a'_i + \sum_{i\in Q_{large}} a'_i + a_k - a'_l \\
\geq\ & \sum_{i\in(B_1\cup\cdots\cup B_m\cup P_1)\setminus B_{j^*}} a'_i + \sum_{i\in B_{j^*}} a'_i + a'_u - a'_l + \sum_{i\in P_0\cup P_2} a'_i \\
\geq\ & \frac{2(D-1)}{3}(m+|P_1|-1)+D+\lceil\frac{D}{2}\rceil(|P_1|-1)+D(|P_0|+|P_2|) \\
>\ & (D-1)(\frac{2}{3}(m+|P_1|-1)+\frac{1}{2}(|P_1|-1)+1+|P_0|+|P_2|) \\
=\ & (D-1)(\frac{2}{3}m+|P_0|+|P_1|+|P_2|+\frac{|P_1|}{6}-\frac{1}{6}) \geq (D-1)m,
\end{aligned}$$

which contradicts (1). Thus, job u should have been assigned to some batch. ■

The performance guarantee of the algorithm APPROX(m, k) is given in the following theorem.

Theorem 8 *Algorithm APPROX(m, k) finds a solution with a $C_{\max}$ value of C' such that $C'/C^*_{\max} \le 1 + \lfloor \frac{m}{2} \rfloor / m$.*

Proof. Consider an arbitrary schedule with m batches and job sets L (basic non-straddling) and M (straddling). We use the following representation of the corresponding $C_{\max}$ value:

$$C_{\max} = \sum_{j \in N} (a_j + b_j) + mS - \sum_{j \in L \cup M} b_j.$$

Let L^* and M^* be the sets of basic non-straddling and straddling jobs, respectively, in an optimal schedule for problem $\Pi^*(m, k)$, which satisfies Theorem 2. Let L and M be the sets of basic non-straddling and straddling jobs, respectively, in the schedule found by algorithm APPROX(m, k). Assume first that there was no call to algorithm APPROX$(3, k, \rho)$.

We have

$$\begin{aligned}
\frac{C'}{C^*_{max}} &= \frac{\sum_{j \in N}(a_j + b_j) + (m + \lfloor \frac{m}{2} \rfloor)S - \sum_{j \in L \cup M} b_j}{\sum_{j \in N}(a_j + b_j) + mS - \sum_{j \in L^* \cup M^*} b_j} \\
&= \frac{\sum_{j \in N}(a_j + b_j) + (m + \lfloor \frac{m}{2} \rfloor)S - \sum_{j \in Q_{large} \cup Q_{small} \cup \{k\} \cup M^*} b_j}{\sum_{j \in N}(a_j + b_j) + mS - \sum_{j \in L^* \cup M^*} b_j} \\
&\quad [\text{ due to Theorem 5 }] \\
&\le \frac{\sum_{j \in N}(a_j + b_j) + (m + \lfloor \frac{m}{2} \rfloor)S - \sum_{j \in L^* \cup M^*} b_j}{\sum_{j \in N}(a_j + b_j) + mS - \sum_{j \in L^* \cup M^*} b_j} \\
&\le \frac{(m + \lfloor \frac{m}{2} \rfloor)S}{mS} = 1 + \lfloor \frac{m}{2} \rfloor / m.
\end{aligned}$$

In case there were calls to algorithm APPROX$(3, k, \rho)$, the only difference is that Theorem 6 should be used instead of Theorem 5. ■

Our approximation algorithm for the problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$, which we denote as APPROX, consists of application of algorithm APPROX(m, k), in which the schedule construction is skipped and only the corresponding $C_{\max}$ value is calculated, for $m = 1, \ldots, n-1$ and $k = 1, \ldots, n-m$, and then constructing the final approximate schedule that corresponds to the minimum $C_{\max}$ value, denoted as $C^0_{\max}$, among the calculated $C_{\max}$ values and the $C_{\max}$ value for an optimal schedule with no basic non-straddling job (see our discussion preceding Theorem 2).

Let us establish the computational complexity of algorithm APPROX. Denote by $R^{(m,k)}$, $Q_{large}^{(m,k)}$, $Q_{small}^{(m,k)}$ and $C_{\max}^{(m,k)}$ the set of the m jobs found by algorithm APPROX(m,k) in Step 5, the sets Q_{large} and Q_{small}, and the $C_{\max}$ value, respectively, found by this algorithm. We have

$$C_{\max}^{(m,k)} = \sum_{j\in N}(a_j+b_j) + (m+\lfloor\frac{m}{2}\rfloor)S - (\sum_{j\in Q_{large}^{(m,k)}\cup Q_{small}^{(m,k)}\cup\{k\}} b_j + \sum_{j\in R^{(m,k)}} b_j)$$

and

$$C_{\max}^{0} = \min\Big\{\min\{C_{\max}^{(m,k)} | m=1,\dots,n-1,\ k=1,\dots,n-m\}, C_{\max}^{(L=\phi)}\Big\},$$

where $C_{\max}^{(L=\phi)}$ is the makespan value of the schedule with no basic non-straddling job.

Thus, the value C_{max}^0 can be calculated in $O(n^2)$ time if we know the sums $BQ^{(m,k)} := \sum_{j\in Q_{large}^{(m,k)}\cup Q_{small}^{(m,k)}} b_j$ and $BR^{(m,k)} := \sum_{j\in R^{(m,k)}} b_j$ for $m=1,\dots,n-1$ and $k=1,\dots,n-m$. The corresponding approximate schedule can be constructed in $O(n\log n)$ time. In the following two theorems we establish the computational complexity for calculating the above mentioned sums.

Theorem 9 *The sets $Q_{large}^{(m,k)}\cup Q_{small}^{(m,k)}$ and the corresponding sums $BQ^{(m,k)}$ can be calculated in $O(n^2\log n)$ time for all the pairs (m,k).*

Proof. We provide a proof for the case $m\neq 3$. The case $m=3$ can be handled similarly. Calculate $a_j' = \min\{a_j, \lceil\frac{D}{2}\rceil\}$, $j=1,\dots,n$. We maintain sets $Q_{large}^{(m,k)}$ and $Q_{small}^{(m,k)}$ as heaps of the values b_i/a_i'. Our further proof is constructive: for a given m, we show how to construct the heaps and calculate the corresponding sum of b_j values for the pair $(m,k+1)$ based on the same information for the pair (m,k).

With the two heaps, we store the cardinality of the set $Q_{large}^{(m,k)}$, denoted as $q_{large}^{(m,k)}$, and the sums $BQ_{large}^{(m,k)} = \sum_{j\in Q_{large}^{(m,k)}} b_j$, $BQ_{small}^{(m,k)} = \sum_{j\in Q_{small}^{(m,k)}} b_j$ and $AQ_{small}^{(m,k)} := \sum_{j\in Q_{small}^{(m,k)}} a_j'$. By definition, for the pair $(m,1)$ heaps $Q_{large}^{(m,1)}$ and $Q_{small}^{(m,1)}$ are empty. Therefore, $q_{large}^{(m,1)}=0$ and $BQ_{large}^{(m,1)} = BQ_{small}^{(m,1)} = AQ_{small}^{(m,1)} = 0$. Assume that the above mentioned information is given for the pair (m,k), $k\geq 1$.

Notice that jobs are assigned to the sets $Q_{large}^{(m,h)}$ and $Q_{small}^{(m,h)}$ in the same order (in non-increasing b_j/a_j' values) for any h and only exceeding the upper bound $(D-1)m-a_h$ on the sum of the a_j' values stops the assignment. An assignment to the set $Q_{large}^{(m,h)}$ is also stopped if the upper bound $m-1$ on the cardinality of this set is reached. Due to the original SPT numbering, we have $a_k \leq a_{k+1}$. Therefore, $Q_{large}^{(m,k+1)} \cup Q_{small}^{(m,k+1)} \subseteq Q_{large}^{(m,k)} \cup Q_{small}^{(m,k)} \cup \{k\}$ and the set $Q_{large}^{(m,k+1)} \cup Q_{small}^{(m,k+1)}$ can be obtained from the set $Q_{large}^{(m,k)} \cup Q_{small}^{(m,k)} \cup \{k\}$ by removing jobs with the smallest b_j/a'_j values until the number of the remaining "large" jobs with $a'_j = \lceil \frac{D}{2} \rceil$ is equal to m and the upper bound $(D-1)m - a_{k+1}$ on the sum of the a'_j values is exceeded. For these purposes, introduce auxiliary job sets (heaps) Q'_{small} and Q'_{large}. Initialize these heaps as $Q'_{small} = Q_{small}^{(m,k)}$ and $Q'_{large} = Q_{large}^{(m,k)}$. If $a'_k = \lceil \frac{D}{2} \rceil$, then add job k to the heap Q'_{large} and calculate $Q'_{small} = Q_{small}^{(m,k)}$, $q' = q_{large}^{(m,k)} + 1$ and $Q'_{large} = Q_{large}^{(m,k)} \cup \{k\}$. If $a'_k < \lceil \frac{D}{2} \rceil$, then add job k to the heap Q'_{small} and calculate $Q'_{large} = Q_{large}^{(m,k)}$, $q' = q_{large}^{(m,k)}$ and $Q'_{small} = Q_{small}^{(m,k)} \cup \{k\}$. In either case, calculate $A' := \sum_{j \in Q'_{small} \cup Q'_{large}} a'_j$. Apply the following procedure.

Procedure HEAPS (for calculating heaps $Q_{large}^{(m,k+1)}$ and $Q_{small}^{(m,k+1)}$)

Step 1 If $q' = m$, i.e., k is a "large" job, then remove from the heap Q'_{large} a "large" job with the smallest value b_j/a'_j and re-set $q' := m-1$ and $A' := A' - \lceil \frac{D}{2} \rceil$.

Step 2 If $A' \leq (D-1)m - a_{k+1}$, then, by definition, $Q_{small}^{(m,k+1)} = Q'_{small}$ and $Q_{large}^{(m,k+1)} = Q'_{large}$. Stop. Otherwise, if $A' > (D-1)m - a_{k+1}$, then find a job with the smallest value b_j/a'_j in the heaps Q'_{small} and Q'_{large}. Let it be job j^0. If $A' - a'_{j^0} \leq (D-1)m - a_{k+1}$, then, by definition, $Q_{small}^{(m,k+1)} = Q'_{small}$ and $Q_{large}^{(m,k+1)} = Q'_{large}$. Stop. If $A' - a'_{j^0} > (D-1)m - a_{k+1}$, then remove job j^0 from the set (Q'_{large} or Q'_{small}) it belongs to, re-set $A' := A' - a'_{j^0}$ and repeat Step 2. ∎

The output of procedure HEAPS is the heaps $Q_{large}^{(m,k+1)}$ and $Q_{small}^{(m,k+1)}$ and the associated numerical values. Observe that each job can be added and removed to/from any of the considered heaps at most once. Such an addition or removal takes $O(\log n)$ time. Finding an element with the smallest value in a heap takes a constant time. Therefore, all the heaps $Q_{large}^{(m,h)}$ and $Q_{small}^{(m,h)}$ and the associated values $BQ^{(m,h)} = BQ_{large}^{(m,h)} + BQ_{small}^{(m,h)}$, $h = 1, \ldots, n-m$, can be found in $O(n \log n)$ time for a given m. The statement of the theorem immediately follows. ∎

Theorem 10 *The sets $R^{(m,k)}$ and the corresponding sums $BR^{(m,k)} = \sum_{j \in R^{(m,k)}} b_j$ can be calculated in $O(n^2 \log n)$ time for all the pairs (m,k).*

Proof. Again, we provide a proof for $m \neq 3$. Recall that set $R^{(m,k)}$ comprises m jobs with the largest b_j values in the set $\{k+1, \ldots, n\}$. Similar to the previous theorem, we will maintain the sets $R^{(m,k)}$ as heaps of the values b_i. For a given m, we will show how to construct the heap and calculate the corresponding sum of b_j values for the pair $(m, k-1)$ based on the same information about the pair (m, k).

Consider set $R^{(m,n-m)}$. By definition, $R^{(m,n-m)} = \{n-m+1, n-m+2, \ldots, n\}$ and $BR^{(m,n-m)} = \sum_{i=1}^{m} b_{n-m+i}$. Assume that set $R^{(m,k)}$ and the corresponding value $BR^{(m,k)}$ have been found, $k \leq n-m$. Introduce an auxiliary set (heap) R'. Initialize it as $R' := R^{(m,k)} \cup \{k\}$. Calculate $BR' := \sum_{j \in R'} b_j = BR^{(m,k)} + b_k$. In the heap R' find a job with the smallest b_j value. Let it be job j^0. Calculate $R^{(m,k-1)} = R' \backslash \{j^0\}$ and $BR^{(m,k-1)} = BR' - b_{j^0}$.

As in the previous theorem, each job can be added and removed to/from any of the considered heaps at most once. Such an addition or removal takes $O(\log n)$ time. Finding an element with the smallest value in a heap takes a constant time. Therefore, all the heaps $R^{(m,h)}$ and the associated values $BR^{(m,h)}$, $h = 1, \ldots, n-m$, can be found in $O(n \log n)$ time for a given m, and the statement of the theorem follows. ■

Let C^* be the optimal makespan. We have proved the following theorem.

Theorem 11 *Algorithm APPROX runs in $O(n^2 \log n)$ time and finds a schedule with the makespan value C^0_{max} such that $C^0_{max}/C^* \leq 1 + \lfloor \frac{m^*}{2} \rfloor / m^* \leq 3/2$, where m^* is the number of batches in an optimal schedule.*

Proof. The proof follows from Theorems 8-10. ■

We remark that a similar approximation algorithm with a performance guarantee 3/2 can be developed for the open-end bin packing problem. Since some of its technical details differ from algorithm APPROX, we do not give its description in this paper.

6 Conclusions

We studied a batch scheduling problem with job deterioration. In this problem jobs are processed in batches preceded by a setup time and the processing time of a job depends on the time elapsed since the start of the batch to which the job belongs. If this waiting time for job i is less than a given threshold value D, then the job requires a basic processing time a_i; otherwise, it requires an extended processing time $a_i + b_i$. The objective is to minimize the makespan. We showed that the special case of this problem where $b_i = b$ is NP-hard in the strong sense, and it is non-approximable with any constant performance guarantee $\Delta < 3/2$ in polynomial time unless $\mathcal{P} = \mathcal{NP}$. For the case with equal basic processing times, $a_i = a$, and the case with F distinct basic processing times, we presented $O(n \log n)$ and $O(n^{3F-1} \log n / F^F)$ time solution algorithms, respectively.

Let m^* denote the number of batches in an optimal schedule. We further presented an approximation $O(n^2 \log n)$ time algorithm with a performance guarantee $\Delta \leq 1 + \lfloor \frac{m^*}{2} \rfloor / m^* \leq 3/2$ for the general problem. All the results obtained in this paper can be applied or easily adapted for the open-end bin packing problem.

Further research can be undertaken in the direction of developing approximation algorithms with a good asymptotic behavior for the general problem $1|S, p_i = a_i + u_1(s_i - s_B - D)b_i|C_{\max}$ and its NP-hard special cases. A modification of the problem with a decreasing step deterioration function instead of an increasing one is worth studying, too.

Acknowledgements

Barketau was supported in part by The Hong Kong Polytechnic University under an International Postgraduate Scholarship for PhD studies. Cheng and Ng were supported in part by The Hong Kong Polytechnic University under a grant from the *Area of Strategic Development in China Business Services*. Barketau and Kovalyov were partially supported by INTAS under grant number 03-51-5501. Kotov was partially supported by INTAS under grant number 03-50-5975.

References

[1] Alidaee B., Womer N.K. (1999) Scheduling with time dependent processing times: review and extensions, *Journal of the Operational Research Society*, **50**, 711-720.

[2] Allahverdi A., Ng C.T., Cheng T.C.E., and Kovalyov M.Y. (2006) A review of scheduling problems with setup times or costs, *European Journal of Operational Research*, in press, available online 13 November 2006.

[3] Cheng T.C.E., Ding Q. (2001) Single machine scheduling with step-deteriorating processing times, *European Journal of Operational Research*, **134**, 623-630.

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

[5] Cheng T.C.E., Gordon V.S., Kovalyov M.Y. (1996) Single machine scheduling with batch deliveries, *European Journal of Operations Research*, **94**, 277-283.

[6] Gawiejnowicz S. (1996) Brief survey of continuous models of scheduling, *Foundations of Computer and Decision Science*, **21**, 81-100.

[7] Inderfurth K., Janiak A., Kovalyov M.Y., Werner F. (2006) Batching work and rework processes with limited deterioration of reworkables, *Computers and Operations Research*, **33**, 1595-1605.

[8] Karmarkar N., Karp R.M. (1982) An efficient approximation scheme for the one-dimensional bin packing problem, *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, IEEE Computer Society: Long Beach, CA, 312-320.

[9] Leung J.Y.-T. (1982) On scheduling independent tasks with restricted execution times, *Operations Research*, **30**, 163-171.

[10] Leung J.Y.-T., Dror M., Young G.H. (2001) A note on an open-end bin packing problem, *Journal of Scheduling*, **4**, 201-207.

[11] Melnikov O.I., Shafransky Y.M. (1980) Parametric problem of scheduling theory, *Cybernetics*, **15**, 352-357, in Russian.

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