The bounded single-machine parallel-batching scheduling problem with family jobs and release dates to minimize makespan

Q.Q. Nong[1], **C.T. Ng**[2,*], **T.C.E. Cheng**[2]

[1]Department of Mathematics, Zhengzhou University, Zhengzhou, Henan 450052, People's Republic of China

[2]Department of Logistics, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong, People's Republic of China

Abstract. We consider the problem of scheduling family jobs with release dates on a bounded batching machine. Our objective is to minimize the maximum completion time of the jobs (i.e., the makespan). We deal with two variants of the problem. One is the identical job size model, in which the machine can process up to b jobs simultaneously as a batch. The other is the non-identical job size model, in which each job has a size no more than 1 and the machine can handle a number of jobs simultaneously as a batch as long as the total size of the jobs does not exceed 1. In this paper we first present a 2-approximation algorithm for the former model and a polynomial time approximation scheme (PTAS) for the case when the number of distinct families is constant; and then derive an approximation algorithm with a worst-case ratio of 5/2 for the latter model.

Keywords: single-machine scheduling; worst-case analysis; approximation algorithm; family; batching

1 Introduction

We are given a set $\mathcal{J}$ of n jobs that belong to m different families, $\mathcal{F}_1, \ldots, \mathcal{F}_m$. Each job J_j has a processing time p_j and a release date r_j before which it cannot be scheduled. The given machine is a parallel-batching machine. The jobs are processed in batches, where a batch is a subset of the given jobs, and we require that jobs from different families cannot be placed in the same batch. The processing time of a batch B, $p(B)$, is equal to the maximum processing time among the jobs in it, i.e., $p(B) = \max\{p_j : J_j \in B\}$. A batch can start processing only after each job in it is released. Our objective is to partition $\mathcal{J}$ into batches and decide a sequence of the starting times of the resulting batches such that the maximum completion time of the jobs (i.e., the makespan), $\max_{1\leq j\leq n} C_j$, is minimized, where C_j is the completion time of job J_j. Two variants of the problem are considered. One is the identical job size model, in which the machine can process up to b ($b < n$) jobs simultaneously as a batch. Following [3], this model is denoted by

$$1|\textit{family-jobs}, r_j, b < n|C_{\max}.$$

The other is the non-identical job size model, in which each job J_j has a size $s_j \leq 1$ and the machine can handle a number of jobs simultaneously as a batch as long as the total size of the

*Corresponding author. E-mail address: lgtctng@polyu.edu.hk (C.T. Ng) jobs does not exceed 1. According to [3], this model is expressed as

$$1|\textit{family-jobs}, r_j, \textit{size}, b = 1|C_{\max}.$$

The parallel-batching scheduling problem is motivated by the burn-in operations in semiconductor manufacturing, in which a batch of integrated circuits are placed in an oven and exposed to a high temperature to test their thermal standing ability. The circuits are heated inside the oven until all the circuits are burned. The burn-in times (processing times) of the circuits may be different. When a circuit is burned, it has to wait inside the oven until all the circuits are burned. Therefore, the processing time of a batch of circuits is equal to the longest processing time of the circuits in the batch. Usually, batching jobs yields efficiency gain, i.e., it is cheaper or faster to process jobs in batches than to process them individually.

Many parallel-batching scheduling problems are NP-hard, i.e., for many of them there does not exist any polynomial time algorithm unless $P = NP$. Researchers therefore turn to studying approximation algorithms for these kinds of problems. The quality of an approximation algorithm is often measured by its *worst-case ratio*: the smaller the ratio is, the better the algorithm will be. We say that an algorithm H has a worst-case ratio ρ (or is a ρ-approximation algorithm) if for any input instance, it always returns in polynomial time of the input size a feasible solution with an objective value not greater than ρ times of the optimal value. Furthermore, a family of approximation algorithms is called a *polynomial time approximation scheme* (PTAS) if, for any fixed $\epsilon > 0$, at least one of the algorithms has a worst-case ratio no more than $1 + \epsilon$.

Many problems similar or related to $1|\textit{family-jobs}, r_j, b < n|C_{\max}$ have been addressed by different researchers. Yuan et al. [13] proved that the problem $1|\textit{family-jobs}, r_j, b = \infty|C_{\max}$, in which a machine can process an infinite number of jobs simultaneously, is strongly NP-hard, and they gave two dynamic programming algorithms and a 2-approximation algorithm for the problem. Lee and Uzsoy [8] studied the problem $1|r_j, b < n|C_{\max}$, which is strongly NP-hard [4], and they presented a number of heuristics. Deng et al. [5] presented a PTAS with time complexity $O(4^{2/\epsilon}n^{8/\epsilon+1}b^{2/\epsilon}/\epsilon^{4/\epsilon-2})$ for the problem $1|r_j, b < n|C_{\max}$. Lee et al. [7] considered the problem $P|b < n|C_{\max}$ and developed a $(\frac{4}{3} - \frac{1}{3M})$-approximation algorithm, where M is the number of batching machines. Li et al. [11] solved the problem $P|r_j, b < n|C_{\max}$ by establishing a PTAS for it.

If all the jobs have identical processing times and are released at the same time and $m = 1$, then the problem $1|\textit{family-jobs}, r_j, \textit{size}, b = 1|C_{\max}$ is the one-dimensional bin packing problem. The bin packing problem is strongly NP-hard [6] and there is no approximation algorithm with a worst-case ratio less than 3/2 unless $P = NP$ [9]. Uzsoy [12] proposed a number of heuristics for the problem $1|\textit{size}, b = 1|C_{\max}$, but he did not analyze their worst-case ratios. Zhang et al. [14] analyzed these heuristics and showed that two of them have a worst-case ratio no greater than 2. In the same paper, Zhang et al. also proposed an approximation algorithm with a worst-case ratio of 7/4 for the problem $1|\textit{size}, b = 1|C_{\max}$. Li et al. [10] developed a $2 + \epsilon$-approximation algorithm with time complexity $O(n \log n + (1/\epsilon^3 - 1/\epsilon + 2)^{1/\epsilon+1})$ for the problem $1|r_j, \textit{size}, b = 1|C_{\max}$, where $\epsilon > 0$ can be sufficiently small.

In this paper we first present a 2-approximation algorithm for the problem $1|\textit{family-jobs}, r_j, b < n|C_{\max}$. We then restrict our discussion to the case where the number of families is constant, and propose a PTAS for this case. Finally, we provide a 5/2-approximation algorithm with time complexity $O(n \log n)$ for the problem $1|\textit{family-jobs}, r_j, \textit{size}, b = 1|C_{\max}$.

2 Problem $1|\textit{family-jobs}, r_j, b < n|C_{\max}$

In this section we consider the problem $1|\textit{family-jobs}, r_j, b < n|C_{\max}$. We will present a 2-approximation algorithm for the problem and a PTAS for the special case in which the number of distinct families is constant. Before explaining our algorithms, we first describe the FBLPT (Full Batch Large Processing Time) algorithm.

Algorithm FBLPT

For $i = 1$ to m, do as follows:

(1) Sort the jobs in $\mathcal{F}_i$ in non-increasing order of their processing times and obtain a job list;

(2) If there are more than b jobs in the job list, then place the first b jobs in a batch and iterate. Otherwise, place the remaining jobs in a batch.

If all the jobs are released at the same time and there is only one family, then Algorithm FBLPT is a polynomial time algorithm for the problem $1|\textit{family-jobs}, r_j, b < n|C_{\max}$, i.e.,

Lemma 2.1[2] Algorithm FBLPT optimally solves the problem $1|b < n|C_{\max}$ in $O(n \log n)$ time. □

The following theorem is an easy generalization of Lemma 2.1.

Theorem 2.2 Algorithm FBLPT optimally solves the problem $1|\textit{family-jobs}, b < n|C_{\max}$ in $O(n \log n)$ time. □

We apply Algorithm FBLPT to the job set $\mathcal{J}$ and obtain a set of batches. Denote by P the total processing time of these batches. Given an instance of $1|\textit{family-jobs}, r_j, b < n|C_{\max}$, we denote by *opt* its optimal value, and by $C(\pi)$ the objective value of a feasible solution π. Define $r_{\max} = \max_{1 \le j \le n}\{r_j\}$. The following bounds for *opt* are easy to establish:

Lemma 2.3 $\max\{r_{\max}, P\} \le \textit{opt} \le r_{\max} + P$. □

2.1 A heuristic

We say that a batch B is *available* at time t if $t \ge \max\{r_j : J_j \in B\}$ and the batch has not yet been assigned to the machine. Consider the following algorithm:

Algorithm H_1

Step 1: Use Algorithm FBLPT to partition the jobs in $\mathcal{J}$ into batches.

Step 2: Whenever the machine is idle and there is a batch available, assign the batch to the machine.

Theorem 2.4 Algorithm H_1 is an approximation algorithm with a worst-case ratio of 2 for the problem $1|\textit{family-jobs}, r_j, b < n|C_{\max}$.

Proof. Let π be the schedule produced by Algorithm H_1. Then,

$$C(\pi) \le r_{\max} + P \le 2\max\{r_{\max}, P\} \le 2\textit{opt},$$

Where the last inequality holds from Lemma 2.3. □

The result of Theorem 2.4 is the best possible, i.e., the worst-case ratio of Algorithm H_1 cannot be less than 2. The tightness example given in Theorem 4.3.2 of [13] shows this result. We omit this example here and refer the reader to [13].

2.2 A PTAS

In this subsection we assume that m is fixed, and we propose a PTAS for this case. To accomplish this, we make a series of transformations of the input instance such that its structure is simplified. Each transformation may increase the objective value by $1+O(\epsilon)$, where ϵ is a positive number. When we describe such a transformation, similar to [1], we say that it produces $1+O(\epsilon)$ loss.

Define $\delta = \epsilon \max\{r_{\max}, P\}$. Then, by Lemma 2.3, $\delta \le \epsilon opt$. The lemma below guarantees that we need to concern about only a small number of distinct release dates.

Lemma 2.5 With $1+\epsilon$ loss, we can assume that all the release dates in an instance are multiples of δ, and the number of distinct release dates is at most $\lfloor 1/\epsilon \rfloor + 1$.

Proof. Given an instance of the problem $1|\textit{family-jobs}, r_j, b < n|C_{\max}$, we round every release date down to the nearest multiple of δ. Then the optimal value of the resulting instance is not greater than that of the original one. Let π^* be an optimal schedule of the rounded instance. Increase the starting time of each batch in π^* by δ. It is easy to verify that this gives a feasible schedule of the original instance with makespan increasing by $\delta \le \epsilon opt$. On the other hand, since $\delta = \epsilon \max\{r_{\max}, P\}$ and hence $r_{\max} \le \frac{1}{\epsilon}\delta$, there are at most $\lfloor 1/\epsilon \rfloor + 1$ distinct release dates in the rounded instance. Thus the result follows. □

One can see that all the jobs in $\mathcal{J}$ can be scheduled in the time interval $[0, r_{\max}+P]$. We partition this time interval into $k = \lceil \frac{r_{\max}+P}{\delta} \rceil$ disjoint intervals in the form $[R_i, R_{i+1}]$, where $R_i = (i-1)\delta$ for each $1 \le i \le k-1$ and $R_k = r_{\max}+P$. Since $\delta = \epsilon \max\{r_{\max}, P\}$, we have $k \le 2/\epsilon + 1$. Note that each of the first $k-1$ intervals has a length δ, and the last one has a length at most δ. By Lemma 2.5, we can assume that every job in $\mathcal{J}$ is released at some R_i $(1 \le i \le \lfloor 1/\epsilon \rfloor + 1)$. We say that a job is *short* if its processing time is smaller than $\epsilon\delta$; and *long*, otherwise.

2.2.1 Dealing with the short jobs

In this part we concentrate on the case in which all the jobs are short and m is constant. Based on the idea of [10], we present a very simple and easy to analyze approximation scheme for this case.

Denote by $\mathcal{F}_{ij}$ the subset of jobs in $\mathcal{F}_i$ that are released at R_j $(1 \le j \le \lfloor 1/\epsilon \rfloor + 1)$ for the rounded instance. Our PTAS runs as follows:

Algorithm Short

Step 1: For $j = 1$ to $\lfloor 1/\epsilon \rfloor + 1$:

Apply Algorithm FBLPT to $\bigcup_{i=1}^{m} \mathcal{F}_{ij}$ and obtain a set of batches.

Step 2: Consider the batches obtained in the above step. Start the processing of a batch whenever the machine is idle and there is a batch available.

Theorem 2.6 If all the jobs are short and m is constant, then Algorithm Short is a PTAS for the problem $1|\textit{family-jobs}, r_j, b < n|C_{\max}$.

Proof. Let π be the schedule produced by Algorithm Short. Suppose that $B_{ij}^1, B_{ij}^2, \ldots, B_{ij}^{k_{ij}}$ are the batches in π whose jobs are from $\mathcal{F}_{ij}$ $(1 \le i \le m, 1 \le j \le \lfloor 1/\epsilon \rfloor + 1)$. For each $1 \le x \le k_{ij}$, denote by $q(B_{ij}^x)$ the processing time of the shortest jobs in B_{ij}^x if B_{ij}^x is full; and set $q(B_{ij}^x) = 0$, otherwise. Without loss of generality, assume that $p(B_{ij}^1) \ge p(B_{ij}^2) \ge \cdots \ge p(B_{ij}^{k_{ij}})$. Then, by the features of Algorithm FBLPT, we conclude that $B_{ij}^1, B_{ij}^2, \ldots, B_{ij}^{k_{ij}-1}$ are full and

$$q(B_{ij}^x) \ge p(B_{ij}^{x+1}) \tag{1}$$

for each $1 \le x \le k_{ij} - 1$. Consider a schedule, denoted by π', which is obtained by modifying π as follows:

- For each batch B_{ij}^x in π, reduce the processing time of each job in it to $q(B_{ij}^x)$;
- The machine processes the resulting batches greedily, i.e., start the processing of a batch whenever the machine is idle and there is a batch available.

Then we have

$$C(\pi) \le C(\pi') + \sum_{i=1}^{m} \sum_{j=1}^{\lfloor 1/\epsilon \rfloor + 1} \sum_{x=1}^{k_{ij}} (p(B_{ij}^x) - q(B_{ij}^x)). \tag{2}$$

Inequality (1) implies that the second term on the right-hand-side of (2) is bounded above by

$$\sum_{i=1}^{m} \sum_{j=1}^{\lfloor 1/\epsilon \rfloor + 1} p(B_{ij}^1) < m \cdot (\lfloor 1/\epsilon \rfloor + 1) \cdot \epsilon\delta \le m(1+\epsilon)\delta \le m(\epsilon + \epsilon^2)\textit{opt}, \tag{3}$$

where the above inequalities are from the facts that $p(B_{ij}^1) < \epsilon\delta$ and $\delta \le \epsilon \textit{opt}$. On the other hand, we claim that $C(\pi') \le \textit{opt}$. To see this, let t be the latest idle time point prior to $C(\pi')$ in π'. Then t is the starting time of some batch and each job processed after t must be released at or after t. Moreover, each batch scheduled in $[t, C(\pi')]$ is full and the processing time of each job in the same batch is identical and is no more than its original value. Thus the claim holds. The claim, together with (2) and (3), implies that $C(\pi) \le \textit{opt} + m(\epsilon + \epsilon^2)\textit{opt}$, completing the proof of the theorem. $\square$

2.2.2 General case

By the job interchange argument, we obtain the following lemma, which plays an important role in our PTAS.

Lemma 2.7 Consider an instance of $1|\textit{family-jobs}, r_j, b < n|C_{\max}$. There exists an optimal schedule satisfying the following properties:

- Each batch B contains as many as possible the longest available jobs that have processing times no greater than $p(B)$.
- The batches starting in the same interval are obtained by applying Algorithm FBLPT to the jobs in the interval.
- The batches processed in the same interval are scheduled in non-increasing order of their processing times. □

The following lemma is useful:

Lemma 2.8 With $1+2m\epsilon+m\epsilon^2$ loss, we can assume that no batch contains both the short jobs and long jobs.

Proof. Let π^* be an optimal schedule. By Lemma 2.7, the batches starting in the same interval are obtained by applying Algorithm FBLPT to the jobs in the interval, for each family there is at most one batch containing both the short jobs and long jobs in each time interval. Recall that the processing times of the short jobs are less than $\epsilon\delta$ and there are at most $2/\epsilon+1$ time intervals occupied by π^*. Replacing each of these batches by two batches, one containing the short jobs only and the other containing the long jobs only, would increase the makespan of π^* by at most $\epsilon\delta \cdot (2/\epsilon+1)\cdot m \leq (2m\epsilon+m\epsilon^2)opt$. □

Suppose that there are l distinct processing times, $L_1, L_2, \dots, L_l$, of the long jobs. Suppose further that $L_1 > L_2 > \cdots > L_l$.

Subsection 2.2.1 has offered an effective way to schedule the short jobs. We now turn to dealing with the long jobs. Our basic idea is to perform enumeration. To do this, we first define a *family configuration* with respect to a family $\mathcal{F}_i$ and a schedule π, by a vector $(c_1, c_2, \dots, c_k)$, where c_h $(1 \leq h \leq k)$ is a l-tuple $(y_{h1}, y_{h2}, \cdots, y_{hl})$, which means that there are y_{hj} batches of $\mathcal{F}_i$ with processing time L_j starting in the time interval $[R_h, R_{h+1}]$. We then define a *machine configuration* as a tuple $(v_1, v_2, \cdots, v_\sharp)$, where $\sharp$ is the number of family configurations to consider, and v_j is the number of families with configuration j. By Lemma 2.7, when a machine configuration is given, the jobs and their processing order in each interval will be definitely defined. We say a machine configuration is *feasible* if it is possible and each batch in it can start in the interval as defined by the configuration.

We find in the sequel that if we round the processing times of the long jobs as stated in the following lemma, then, with a little change in the objective value, the number of all the possibilities of machine configurations is bounded by a polynomial in n and m when ϵ is fixed.

Lemma 2.9 With $1+2\epsilon+\epsilon^2$ loss, we can assume that all the processing times of the long jobs are multiples of $\epsilon^2\delta$.

Proof. Round the processing time of each long job up to the nearest multiple of $\epsilon^2\delta$. Consider an optimal schedule of the original instance. Recall that the processing time of a long job is at least $\epsilon\delta$ and the length of each interval is at most δ. Thus, each interval can process at most $1/\epsilon$ long jobs. Therefore, replacing the processing time of each long job by the rounded one will increase the objective value by at most

$$\epsilon^2\delta \cdot \frac{1}{\epsilon} \cdot k \leq \epsilon^2\delta \cdot \frac{1}{\epsilon} \cdot (2/\epsilon+1) \leq \epsilon(2+\epsilon)opt,$$

where the inequalities are from the facts that $k \le 2/\epsilon + 1$ and $\delta \le \epsilon opt$. $\square$

Corollary 2.10 With $1 + 2\epsilon + \epsilon^2$ loss, the number of distinct processing times of the long jobs, l, is bounded by $1/\epsilon^3 - 1/\epsilon + 1$.

Proof. Each long job J_j satisfies $\epsilon\delta \le p_j \le P$. Recall that $\delta = \epsilon \max\{r_{\max}, P\}$. Thus, $\epsilon\delta \le p_j \le P \le \frac{1}{\epsilon}\delta$. Noting further that in the rounded instance, all the processing times of the long jobs are multiples of $\epsilon^2\delta$, we conclude that there are at most

$$(\frac{1}{\epsilon}\delta - \epsilon\delta)/\epsilon^2\delta + 1 = 1/\epsilon^3 - 1/\epsilon + 1$$

distinct processing times of the long jobs. $\square$

Corollary 2.11 With $1 + 2\epsilon + \epsilon^2$ loss, the number of distinct machine configurations is at most $(m+1)^\sharp$, where $\sharp = (1 + l + \cdots + l^{1/\epsilon})^{2/\epsilon+1}$.

Proof. Consider a family configuration $(c_1, c_2, \ldots, c_k)$ of a rounded instance. Recall that $c_h = (y_{h1}, y_{h2}, \cdots, y_{hl})$ $(1 \le h \le k)$. Define $||c_h|| = \sum\limits_{j=1}^{l} y_{hj}$. Then, $||c_h|| \le 1/\epsilon$, since there are at most $1/\epsilon$ long batches processed in each interval. One can find that the total number of the possibilities of c_h is bounded by $1 + l + \cdots + l^{1/\epsilon}$. Thus, there are no more than $(1+l+\cdots+l^{1/\epsilon})^k \le (1+l+\cdots+l^{1/\epsilon})^{2/\epsilon+1}$ family configurations. Since a machine configuration is described by a tuple $(v_1, v_2, \cdots, v_\sharp)$ with $0 \le v_j \le m$, there are at most $(m+1)^\sharp$, a polynomial of m, machine configurations. $\square$

We here present our algorithm.

Algorithm H_2

Step 1: *Round the release dates and processing times.* (1) Round each release date down to the nearest multiple of δ; (2) Round the processing time of each long job up to the nearest multiple of $\epsilon^2\delta$.

Step 2: *Schedule the long jobs.* Obtain all the feasible machine configurations.

Step 3: *Schedule the short jobs.* Schedule the short jobs by running Algorithm Short in the spaces left by the long jobs. If a short batch crosses an interval, then stretch the interval with length $\epsilon\delta$ such that the batch can finish in the interval.

Step 4: *Select the best solution.* Among all the schedules obtained above, select the one with the minimum makespan.

Theorem 2.12 Algorithm H_2 is a PTAS for the problem $1|\textit{family-jobs}, r_j, b < n|C_{\max}$ when m is constant.

Proof. Since Algorithm H_2 tries all the feasible possibilities to schedule the long jobs, there is at least one with a machine configuration consistent with that of the optimal schedule. On the other hand, Step 1 and Step 3, rounding and running Algorithm Short to schedule the short jobs and stretch the intervals with the short jobs crossing, respectively, yield at most $1 + O(m\epsilon)$ loss. Thus, Algorithm H_2 is a $1 + O(m\epsilon)$-approximation algorithm. Note that the time complexity of the algorithm is $O(n \log n + n \cdot (m+1)^\sharp)$. The theorem follows. $\square$

3 An Approximation Algorithm for $1|\mathit{family\text{-}jobs}, r_j, \mathit{size}, b = 1|C_{\max}$

In this section we provide an approximation algorithm with a worst-case ratio of $\frac{5}{2}$ for the problem

$$1|\mathit{family\text{-}jobs}, r_j, \mathit{size}, b = 1|C_{\max}.$$

Before describing the algorithm, we first present a variant of FBLPT algorithm, which will be used later.

Algorithm FBLPT-size

For $i = 1$ to m, do the following:

(1) Index the jobs in $\mathcal{F}_i$ in non-increasing order of their processing times;

(2) Place the job with the longest processing time in a batch. If the batch has enough room for the next job in the job list, then put the job in the batch; otherwise, place part of the job in the batch such that the batch is completely full and put the remaining part of the job at the head of the remaining job list and continue. □

A job is called a *split* job if it is split in size. Note that in the problem $1|\mathit{family\text{-}jobs}, r_j, \mathit{size}, b = 1|C_{\max}$, splitting is not allowed. We say that a job is *big* if its size is larger than $\frac{1}{2}$; and *small*, otherwise.

Now we are ready to describe an approximation algorithm for $1|\mathit{family\text{-}jobs}, r_j, \mathit{size}, b = 1|C_{\max}$. The algorithm consists of two steps. The first step is assigning each big job to a batch and processing the resulting batches greedily. The second step is first assigning the small jobs to batches by Algorithm FBLPT-size, and then moving the split jobs out and rearranging them in new batches and processing the bathes greedily.

Algorithm H_3

Step 1: *Process the big jobs.* Let $J_1, J_2, \ldots, J_x$ be the set of the big jobs. Without loss of generality, assume that $r_1 \leq r_2 \leq \cdots \leq r_x$. Put each big job in a batch. Process J_1 at time r_1 and J_j $(2 \leq j \leq x)$ at time $\max\{r_j, C_{j-1}\}$.

Step 2: *Process the small jobs.*

(1) Assign the small jobs to batches by Algorithm FBLPT-size.

(2) For $i = 1$ to m, do the following: Assume that it results in u batches $B_i^1, B_i^2, \ldots, B_i^u$ for the jobs in $\mathcal{F}_i$ in (1). Let $J_{i_1}, J_{i_2}, \ldots, J_{i_q}$ $(q \leq u-1)$ be the set of split jobs in these batches, where $p_{i_1} \geq p_{i_2} \geq \cdots \geq p_{i_q}$. Remove all the split jobs from $B_i^1, B_i^2, \ldots, B_i^u$. For $v = 1, 2, \ldots, \lceil q/2 \rceil$, put $J_{i_{2v-1}}$ and $J_{i_{2v}}$ together in a new batch B_i^{u+v}.

(3) From time C_x onwards, process the resulting batches greedily.

Without loss of generality, suppose that $p(B_i^1) \geq p(B_i^2) \geq \cdots \geq p(B_i^u)$. As before, we use *opt* and $C(\pi)$ to denote the optimal value and the makespan of schedule π, respectively. Evidently, we have a lower bound on *opt*.

Lemma 3.1 $\mathit{opt} \geq \max\{r_{\max}, C_x\}$.

Proof. Obviously, $\mathit{opt} \geq r_{\max}$. Note that one batch can only contain at most one big job and the first step of the above algorithm processes all the big jobs greedily. Thus, $opt \geq C_x$, and the result follows. □

Zhang et al. [14] proved that if all the jobs can be split in size, then Algorithm FBLPT-size is a polynomial time algorithm for the problem $1|size, b = 1|C_{\max}$, a special case in which all the jobs are released at the same time and $m = 1$. In fact, the result still holds when m is arbitrary. Note further that the optimal value of an instance of $1|family\text{-}jobs, r_j, size, b = 1|C_{\max}$ will not increase if all the jobs in it are allowed to be split in size. It implies the following lemma:

Lemma 3.2 $\sum\limits_{i=1}^{m}\sum\limits_{j=1}^{u} p(B_i^j) \leq opt.$

Lemma 3.3 $p_{i_{2v-1}} \leq p(B_i^{2v}).$

Proof. Suppose that the first part of $J_{i_{2v-1}}$ is placed in B_i^j. Then, $j \geq 2v-1$, and thus B_i^{2v} contains a part of $J_{i_{2v-1}}$. Hence, $p_{i_{2v-1}} \leq p(B_i^{2v})$. □

By Lemma 3.2 and Lemma 3.3, we have the following corollary:

Corollary 3.4 $\sum\limits_{i=1}^{m}\sum\limits_{v=1}^{\lceil q/2\rceil} p(B_i^{u+v}) \leq \frac{1}{2}opt.$

Proof. By our algorithm, we have $p(B_i^{u+v}) = p_{i_{2v-1}}$. Recall that $p(B_i^{2v}) \leq p(B_i^{2v-1})$ and $q \leq u-1$. By Lemma 3.3,

$$\sum_{v=1}^{\lceil q/2\rceil} p_{i_{2v-1}} \leq \sum_{v=1}^{\lceil q/2\rceil} p(B_i^{2v}) \leq \frac{1}{2}\sum_{j=1}^{u} p(B_i^j).$$

Thus,

$$\sum_{i=1}^{m}\sum_{v=1}^{\lceil q/2\rceil} p(B_i^{u+v}) = \sum_{i=1}^{m}\sum_{v=1}^{\lceil q/2\rceil} p_{i_{2v-1}} \leq \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{u} p(B_i^j) \leq \frac{1}{2}opt,$$

where the last inequality holds from Lemma 3.2. □

Theorem 3.5 The worst-case ratio of Algorithm H_3 is $\frac{5}{2}$.

Proof. Let π be the schedule produced by Algorithm H_3. Then,

$$C(\pi) \leq \max\{C_x, r_{\max}\} + \sum_{i=1}^{m}\sum_{j=1}^{u} p(B_i^j) + \sum_{i=1}^{m}\sum_{v=1}^{\lceil q/2\rceil} p(B_i^{u+v}) \leq opt + opt + \frac{1}{2}opt = \frac{5}{2}opt,$$

completing the proof of the theorem. □

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.*

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein and M. Sviridenko, Approximation schemes for minimizing average weighted completion time with release dates, *Proceedings of the 40th Annual IEE Symposium on Foundations of Computer Science*, New York, October, 1999, 32-43.

[2] J.J. Bartholdi, unpublished manuscript, 1988.

[3] P. Brucker, *Scheduling Algorithms*, Springer-Verlag, Berlin, 2001.

[4] P. Brucker, A. Gladky, H. Hoogeveen, M.Y. Kovalyvov, C.N. Potts, T. Tautehahn and S.L. van de Velde, Scheduling a batching machine. *Journal of Scheduling*, **1**(1998), 31-54.

[5] X. Deng, C.K. Poon and Y. Zhang, Approximation algorithms in batch processing, *Lecture Notes in Computer Science*, **1741**(2000), 153-162.

[6] M.R. Garey and D.S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco, 1979.

[7] C.Y. Lee and R. Uzsoy, Minimizing makespan on a single batch processing machine with dynamic job arrivals, *International Journal of Production Research*, **37**(1999), 219-236.

[8] C.Y. Lee, R. Uzsoy and L.A. Martin Vega, Efficient algorithms for scheduling semiconductor burn-in operations, *Operations Research*, **40**(1992), 764-775.

[9] J.K. Lenstra and D.B. Shmoys, Computing near-optimal schedules, *Scheduling Theory and its Application*, Wiley, New York, 1995.

[10] S. Li, G. Li, X. Wang and Q. Liu, Minimizing makespan on a single batching machine with release times and non-identical job sizes, *Operations Research Letters*, **33**(2005), 157-164.

[11] S. Li, G. Li and S. Zhang, Minimizing makespan with release times on identical parallel batching machines, *Discrete Applied Mathematics*, **148**(2005), 127-134.

[12] R. Uzsoy, A single batch processing machine with non-identical job sizes, *International Journal of Production Research*, **32**(1994), 1615-1635.

[13] J.J. Yuan, Z.H. Liu, C.T. Ng and T.C.E. Cheng, The unbounded single machine parallel batch scheduling problem with family jobs and release dates to minnimize makespan, *Theoretical Computer Science*, **320**(2004), 199-212.

[14] G. Zhang, X. Cai, C.Y. Lee and C.K. Wong, Minimizing makespan on a single batch processing machine with nonidentical job sizes, *Naval Research Logistics*, **48**(2001), 226-240.