Theoretical Computer Science 411 (2010) 1140–1145

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Minimizing the makespan on a single parallel batching machine☆

Shenpeng Lu, Haodi Feng*, Xiuqian Li

School of Computer Science and Technology, Shandong University, China

ARTICLE INFO

Article history:
Received 19 May 2009
Received in revised form 18 September 2009
Accepted 13 December 2009
Communicated by X. Deng

Keywords:
Scheduling
Parallel batching
Rejection
Job size
Release date
PTAS

ABSTRACT

In this paper, we consider the problem of scheduling with release dates and rejection on a single parallel batching machine, where the jobs have non-identical sizes. Our objective is to minimize the makespan of the accepted jobs plus the total penalty of the rejected jobs. First, we give a polynomial time approximation scheme (PTAS) for the case where jobs can be split. Then, we propose a 2-approximation algorithm for the special case with identical release dates. Finally, we present an approximation algorithm for the general problem with worst-case ratio $2+\epsilon$, where $\epsilon > 0$ is an arbitrarily small constant.

1. Introduction

In this paper, we consider the problem of scheduling on a single parallel batching machine. We are given a set of n jobs and a single parallel batching machine with a capacity B. Each job j is characterized by a tuple of integer numbers $(r_j, p_j, size_j, w_j)$, where r_j is the release date before which job j cannot be scheduled, p_j is the processing time that specifies the minimum time needed to process job j without interruption, $size_j$ is the size of job j, and w_j is the penalty paid to job j if it is rejected. The machine can process a number of jobs simultaneously as a batch as long as the total size of the jobs in the batch does not exceed B. The processing time of a batch is the longest processing time of any job in the batch. Jobs processed in the same batch have the same start time and the same completion time. Our goal is to schedule the jobs so that the makespan of the accepted jobs plus the total penalty of the rejected jobs is minimized. Denote by W the total penalty of the rejected jobs. Using the general notation for scheduling problems introduced by Graham et al. [1], we denote our problem as $1|p\text{-}batch, size_j, r_j|C_{\max} + W$.

In the last decade, there has been significant interest in scheduling problems that involve batching. The motivation for batching jobs is a gain in efficiency: It may be cheaper or faster to process jobs in a batch than to process them individually. There are reviews of models which combine scheduling with batching by Webster and Baker [2] and Potts et al. [3,4]. Brucker et al. [5] gave deep studies on scheduling on a single parallel batching machine.

In real applications, due to the limited resources, the scheduler can have the option of rejecting some jobs. For the model of scheduling with rejection for minimizing makespan, Bartal et al. [6] first introduced the feature that jobs can be rejected at certain prices and studied the off-line as well as on-line versions of scheduling with rejection on identical parallel machines.

☆ This work was supported by the National Natural Science Foundation of China under grant No. 60603007 and the Natural Science Foundation of Shandong Province under grant No. Q2006G01.

* Corresponding address: School of Computer Science and Technology, Shandong University, Shun Hua Road, 250101 Jinan, Shandong Province, China. Tel.: +86 15253177789.

E-mail addresses: fenghaodi@hotmail.com, fenghaodi@sdu.edu.cn (H. Feng).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.12.008

Seiden [7] presented an improved on-line algorithm for the case when preemption is allowed for all the jobs. Hoogeveen et al. [8] considered an off-line version for the case when preemption is allowed. Cao et al. [9] considered this model as a bi-criteria optimization problem through treating W as a constraint. Cao and Zhang [10] studied the scheduling problem with release dates on one machine: They proved its NP-hardness and designed a PTAS for the off-line version. For the on-line version, they gave a best possible algorithm with competitive ratio $(\sqrt{5}+1)/2$. Wang et al. [11] gave a polynomial time algorithm for a single parallel batching machine with rejection. Zhang et al. [12] considered the single machine scheduling problem with release dates and rejection. Cheng and Sun [13] considered the single machine scheduling problem with deterioration and rejection, in which the processing time of a job is a linear function of its starting time. For the problem $1|p\text{-}batch, B < n, r_j|C_{\max}+W$, Lu et al. [14] proved that it is unary NP-hard, then gave a 2-approximation algorithm and a polynomial time approximation scheme. For the unbounded case, i.e. $1|p\text{-}batch, B > n, r_j|C_{\max}+W$, Lu et al. [15] proved its binary NP-hardness and gave a PTAS for a single parallel batching machine with release dates and rejection; at the same time, Feng and Liu [16] proposed a PTAS by using a different method.

For the model of scheduling on a single parallel batching machine with non-identical job sizes, Zhang et al. [17] proposed an approximation algorithm with worst-case ratio 7/4 for minimizing makespan and analyzed four heuristics of Uzsoy [18]. Then, Li et al. [19] presented an approximation algorithm with worst-case ratio $2+\epsilon$ for the case when the jobs have non-identical release dates, where ϵ can be made arbitrarily small.

From another viewpoint, if all jobs are released at the same time and have identical processing times, the problem is just the bin packing problem with rejection. The bin packing problem with rejection was introduced and studied by Dósa and He [20]. They suggested an interesting application for the on-line version of the problem which is related to caching. The bin packing problem with rejection is strongly NP-hard, and also, APX-hard, and there is no polynomial algorithm that can achieve a better worst-case ratio than 3/2 unless P = NP [21]. Hence, our general problem and the special case with identical release dates are both APX-hard.

Our work is motivated by Lu et al. [14] and Li et al. [19]. We first design a PTAS for the special case where jobs can be split; then give a 2-approximation algorithm for the special case with identical release dates; in the end we present a $(2+\epsilon)$-approximation algorithm for the general problem.

2. Preliminaries

In complexity theory, the class APX (an abbreviation of "approximable") is the set of optimization problems that allow polynomial time approximation algorithms with approximation ratio bounded by a constant (or constant-factor approximation algorithms for short). If there is a polynomial time algorithm for solving a problem within every fixed percentage (one algorithm for each percentage), then the problem is said to have a polynomial time approximation scheme (PTAS).

We use GP (General Problem) to denote the problem $1|p\text{-}batch, size_j, r_j|C_{\max}+W$, and use SP to denote the problem which is the same as GP except that all jobs can be split in size. In a schedule, a job is called a *split job* if it is split in size. Namely, each original job $j=(r_j, p_j, size_j, w_j)$ may be split into *partial jobs* $j(1),\ldots,j(m)$, with $j(k)=(r_j, p_j, size_{j(k)}, w_j)$ for each $1\le k\le m$. Here $size_j=\sum_{k=1}^{m} size_{j(k)}$ and $j(1),\ldots,j(m)$ are independent. It should be stressed that the term "split job" refers to an original job that is subjected to splitting.

Without loss of generality, we assume that $size_j \le B$ for each $j, j=1,\ldots,n$.

The outline of our work is as follows: In Section 3, we present a PTAS for SP. In Section 4, we propose a 2-approximation algorithm for GP with identical release dates and a $(2+\epsilon)$-approximation algorithm for GP.

3. Algorithms for SP

In this section, we first provide a polynomial time algorithm for the case with identical release dates. Then, we present a pseudo-polynomial time algorithm for the case with k distinct release dates, where k is a fixed positive integer. Finally, we give a 2-approximation algorithm and a polynomial time approximation scheme for the general SP.

3.1. The case with identical release dates

Assume that $r_j = r$ for $j=1,2,\ldots,n$. The problem of scheduling jobs with non-identical job sizes to minimize the makespan on a parallel batching machine, i.e. $1|p\text{-}batch, split, size_j|C_{\max}$, can be solved optimally by using the full batch longest processing time rule (FBLPT-rule) [19]:

Step 1. Sort jobs such that $p_1 \ge \cdots \ge p_n$.
Step 2. Open a batch, and fill it with the largest remaining jobs (the largest job may be a partial job). If the batch does not have enough room for some job, place part of the job into the batch such that the batch is completely full.
Step 3. Repeat Step 2 until the job list is empty.

By the optimality of the FBLPT-rule for $1|p\text{-}batch, split, size_j|C_{\max}$, we have the following lemma.

Lemma 1. *There exists an optimal schedule for* $1|p\text{-batch}, split, size_j|C_{\max}+W$ *in which the accepted jobs are assigned to the machine by the* FBLPT-rule.

Assume that the jobs have been indexed such that $p_1 \geq \cdots \geq p_n$. Let $P_j(s)$ be the optimal solution as well as its value to the restricted problem satisfying the following conditions: (1) The jobs under consideration are $J_1, \ldots, J_j$. (2) The total size of accepted jobs among $J_1, \ldots, J_j$ is exactly s. Consider an optimal schedule corresponding to $P_j(s)$, we have the following two cases:

Case 1. J_j is rejected.

Since J_j is rejected, the total size of accepted jobs among $J_1, \ldots, J_{j-1}$ is still s. Thus, we have $P_j(s) = P_{j-1}(s) + w_j$.

Case 2. J_j is accepted.

Since J_j is accepted, the total size of accepted jobs among $J_1, \ldots, J_{j-1}$ is $s - size_j$. Let $M = s \pmod B$, $N = (s - size_j) \pmod B$. Thus, if $s = size_j$, then $P_j(s) = P_{j-1}(0) + r + p_j$. If $s > size_j$ and $M > N$, J_j must be processed in the last batch together with some other jobs; then $P_j(s) = P_{j-1}(s - size_j)$. If $s > size_j$ and $M \leq N$, J_j cannot be contained in the last batch of $P_{j-1}(s - size_j)$, completely. By the optimality of the FBLPT-rule, J_j must be split and part of it is scheduled alone in a new batch so that the remaining part of J_j is processed completely; then $P_j(s) = P_{j-1}(s - size_j) + p_j$.

Combining the above two cases, we get the following dynamic programming algorithm *DP*1.

Dynamic Programming algorithm DP1. The boundary conditions:

$P_1(0) = w_1$, $P_1(size_1) = r + p_1$ and $P_1(s) = +\infty$ for any $s \neq \{0, size_1\}$.

The recursive function:

Let $M = s \pmod B$, $N = (s - size_j) \pmod B$.

$$P_j(s) = \begin{cases} P_{j-1}(s) + w_j & \text{if } s < size_j, \\ \min\{P_{j-1}(0) + r + p_j, P_{j-1}(s) + w_j\} & \text{if } s = size_j, \\ \min\{P_{j-1}(s - size_j), P_{j-1}(s) + w_j\} & \text{if } s > size_j \text{ and } M > N, \\ \min\{P_{j-1}(s - size_j) + p_j, & \\ P_{j-1}(s) + w_j\} & \text{if } s > size_j \text{ and } M \leq N. \end{cases}$$

The optimal solution is given by $\min\{P_n(s) : 0 \leq s \leq \sum_{j=1}^{n}(size_j)\}$.

Theorem 2. *Algorithm DP1 is an exact algorithm for* $1|p\text{-batch}, split, size_j, r_j = r|C_{\max}+W$, *with a running time of* $O(n^2B)$.

Proof. The correctness of the algorithm is guaranteed by the above discussion. Clearly, we have $1 \leq j \leq n$ and $0 \leq s \leq \sum_{j=1}^{n}(size_j)$. Because $size_j \leq B$ for each $j = 1, \ldots, n$, we have $s \leq nB$. Thus, the recursive function has at most $O(n^2B)$ states. Each iteration takes a constant time to execute. Hence, the time complexity is bounded by $O(n^2B)$. □

3.2. The case with k distinct release dates

For a schedule π, we say that the accepted jobs are processed using the *Modified FBLPT-rule*:

Step 1. Sort jobs such that $p_1 \geq \cdots \geq p_n$.

Step 2. Split J_j into $size_j$ partial jobs $J_{j(1)}, \ldots, J_{j(size_j)}$ with $size_{j(x)} = 1$, where $x = 1, \ldots, size_j, j = 1, \ldots, n$. We get a new job list $J_{11}, \ldots, J_{1(size_1)}, \ldots, J_{n(1)}, \ldots, J_{n(size_n)}$.

Step 3. Schedule the new job list according to the FBLPT-rule.

Step 4. For each batch, weld the consecutive partial jobs that are derived from the same original job together as a bigger partial job.

We say that in a schedule a batch splits a job j if this batch contains a partial job of job j while the preceding batches do not. We have the following lemma.

Lemma 3. *If a schedule is subjected to the Modified FBLPT-rule, then each batch in the schedule splits at most one job.*

Proof. It is easy to see that only the smallest job in a batch may be such a job. □

For the problem $1|p\text{-batch}, B < n, r_j|C_{\max}+W$ with a constant number of release dates, Lu et al. [14] presented a pseudo-polynomial time algorithm. By using the Modified FBLPT-rule and extending their algorithm, we present the following pseudo-polynomial time algorithm for the problem considered.

Assume that the jobs have been indexed such that $p_1 \geq \cdots \geq p_n$. Let $R_1, R_2, \ldots, R_k$ with $R_1 < R_2 < \ldots < R_k$ be the k distinct release dates. We divide $[R_1, +\infty)$ into k time intervals $[R_1, R_2), [R_2, R_3), \ldots, [R_k, R_{k+1})$, where $R_{k+1} = +\infty$. Let $f_j(p\_size; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w)$ be the optimal solution and its value to the problem considered satisfying the following conditions: (1) The jobs under consideration are $J_1, \ldots, J_j$. (2) A part of size $size_j - p\_size$ of job j is processed. (3) The first batch starting in $[R_i, R_{i+1})$ starts at time $b_i + R_i$. If no batch starts in $[R_i, R_{i+1})$, we set $b_i = +\infty$. (4) The total size of the jobs in the last batch starting in $[R_i, R_{i+1})$ is s_i. If no batch starts in $[R_i, R_{i+1})$, we set $s_i = B$. (5) The total length of the batches starting in $[R_i, R_{i+1})$ is l_i. If no batch starts in $[R_i, R_{i+1})$, we set $l_i = 0$. (6) The total penalty of rejected jobs is exactly w. It is obvious that $f_j(size_j; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w) = f_{j-1}(0; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w)$.

We get $f_j(p\_size; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w)$ in a recursive way. Now, we distinguish two cases in the following discussion.

Case 1. J_j is rejected.

Since J_j is rejected, we have $f_j(0; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w) = f_{j-1}(0; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w-w_j)+w_j$.

Case 2. J_j is accepted. Given a vector $(p\_size; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w)$, we assume that $b_l = \max\{b_i + R_i : b_i \neq +\infty\}$. We split J_j into $size_j$ jobs $J_{j(1)}, \ldots, J_{j(size_j)}$ with $size_{j(x)} = 1$, where $x = 1, \ldots, size_j$. Let $h_i(j, p\_size)$ be the optimal solution and its value under the constraint that the batch containing $J_{j(size_j-p\_size)}$ starts in $[R_i, R_{i+1})$, where $R_i \geq r_j$ and $i \leq l$. If $s_i > 1$, then $J_{j(size_j-p\_size)}$ must be processed in the last batch with some other partial jobs in $[R_i, R_{i+1})$. Thus, we have $h_i(j, p\_size) = f_j(p\_size + 1; b_1, \ldots, b_k; s_1, \ldots, s_{i-1}, s_i - 1, s_{i+1}, \ldots, s_k; l_1, \ldots, l_k; w)$. If $s_i = 1$ and $i < l$, then $J_{j(size_j-p\_size)}$ must be in a batch alone in $[R_i, R_{i+1})$ and this does not increase the makespan of the accepted jobs. Thus, we have $h_i(j, p\_size) = f_j(p\_size + 1; b_1, \ldots, b_k; s_1, \ldots, s_{i-1}, B, s_{i+1}, \ldots, s_k; l_1, \ldots, l_{i-1}, l_i - p_j, l_{i+1}, \ldots, l_k; w)$. If $s_i = 1$ and $i = l$, then $J_{j(size_j-p\_size)}$ must be in a batch alone in $[R_i, R_{i+1})$ and this will increase the makespan of the accepted jobs. And, we have $h_i(j, p\_size) = f_j(p\_size + 1; b_1, \ldots, b_k; s_1, \ldots, s_{i-1}, B, s_{i+1}, \ldots, s_k; l_1, \ldots, l_{i-1}, l_i - p_j, l_{i+1}, \ldots, l_k; w) + p_j$. Furthermore, we also have $f_j(0; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w) = \min\{f_{j-1}(0; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w - w_j) + w_j, \min\{h_i(j, 0) : R_i \geq r_j \text{ and } 1 \leq i \leq l\}\}$, $f_j(p\_size; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w) = \min\{h_i(j, p\_size) : R_i \geq r_j \text{ and } 1 \leq i \leq l\}$ for any other case $p\_size \neq \{0, size_j\}$, where

$$h_i(j, p\_size) = \begin{cases} f_j(p\_size + 1; b_1, \ldots, b_k; s_1, \ldots, s_{i-1}, \\ s_i - 1, s_{i+1}, \ldots, s_k; l_1, \ldots, l_k; w) & \text{if } s_i > 1, \\ f_j(p\_size + 1; b_1, \ldots, b_k; s_1, \ldots, s_{i-1}, \\ B, s_{i+1}, \ldots, s_k; l_1, \ldots, l_{i-1}, l_i - p_j, \\ l_{i+1}, \ldots, l_k; w) & \text{if } s_i = 1 \text{ and } i < l, \\ f_j(p\_size + 1; b_1, \ldots, b_k; s_1, \ldots, s_{i-1}, \\ B, s_{i+1}, \ldots, s_k; l_1, \ldots, l_{i-1}, l_i - p_j, \\ l_{i+1}, \ldots, l_k; w) + p_j & \text{if } s_i = 1 \text{ and } i = l. \end{cases}$$

Combining the above two cases, we design the following dynamic programming algorithm DP2.

Dynamic programming algorithm DP2. The boundary conditions: Given a state vector $(p\_size; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w)$, we assume that $b_l = \max\{b_i + R_i : b_i \neq +\infty\}$. We define $f_0(0; b_1, \ldots, b_k; B, \ldots, B; 0, \ldots, 0; 0) = b_l + R_l$, and for any other case, $f_0(0; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w) = +\infty$.

The recursive function:

$f_j(0; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w) = \min\{f_{j-1}(0; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w-w_j)+w_j, \min\{h_i(j, 0) : R_i \geq r_j \text{ and } 1 \leq i \leq l\}\}$,

$f_j(size_j; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w) = f_{j-1}(0; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w)$,

$f_j(p\_size; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w) = \min\{h_i(j, p\_size) : R_i \geq r_j \text{ and } 1 \leq i \leq l\}$ for any other cases $p\_size \neq \{0, size_j\}$, where

$$h_i(j, p\_size) = \begin{cases} f_j(p\_size + 1; b_1, \ldots, b_k; s_1, \ldots, s_{i-1}, \\ s_i - 1, s_{i+1}, \ldots, s_k; l_1, \ldots, l_k; w) & \text{if } s_i > 1, \\ f_j(p\_size + 1; b_1, \ldots, b_k; s_1, \ldots, s_{i-1}, \\ B, s_{i+1}, \ldots, s_k; l_1, \ldots, l_{i-1}, l_i - p_j, \\ l_{i+1}, \ldots, l_k; w) & \text{if } s_i = 1 \text{ and } i < l, \\ f_j(p\_size + 1; b_1, \ldots, b_k; s_1, \ldots, s_{i-1}, \\ B, s_{i+1}, \ldots, s_k; l_1, \ldots, l_{i-1}, l_i - p_j, \\ l_{i+1}, \ldots, l_k; w) + p_j & \text{if } s_i = 1 \text{ and } i = l. \end{cases}$$

The value of optimal solutions is given by $\min\{f_n(0; b_1, \ldots, b_k; s_1, \ldots, s_k; l_1, \ldots, l_k; w)\}$; and the corresponding optimal solutions can be derived using a backward method. Select one optimal solution that is subject to the Modified FBLPT-rule.

Theorem 4. *Algorithm DP2 solves* $1|p\text{-}batch, split, size_j, r_j \in \{R_i : 1 \leq i \leq k\}|C_{\max} + W$ *in* $O(n^2kB^{k+2}p_1^{k-1}(\sum w_j)(\sum p_j)^k)$ *time.*

Proof. The correctness of the algorithm is guaranteed by the above discussion. Clearly, we have $0 \leq p\_size \leq B$, $1 \leq s_i \leq B$, $0 \leq l_i \leq \sum p_j$ and $0 \leq w \leq \sum w_j$. Furthermore, if some batch starts in $[R_i, R_{i+1})$ for $i = 1, \ldots, k$, then we have $b_1 = 0$ and $0 \leq b_i < \min\{p_1, R_{i+1} - R_i\}$. Since all p_j, r_j and w_j are integers, we can assume that all l_i, b_i and w are integers. Thus, the recursive function has at most $O(nB^{k+1}p_1^{k-1}(\sum w_j)(\sum p_j)^k)$ states. Each iteration takes an $O(k)$ time to execute. To test whether the schedule is subjected to the Modified FBLPT-rule, we need to take $O(nB)$ time. Hence, the total running time is bounded by $O(n^2kB^{k+2}p_1^{k-1}(\sum w_j)(\sum p_j)^k)$. □

3.3. A 2-approximation algorithm for general SP

Assume that S is a set of jobs. We use $p(S) = \max_{J_j \in S}\{p_j\}$ and $w(S) = \sum_{J_j \in S} w_j$ to denote the processing time and the total penalty of S, respectively. We propose a 2-approximation algorithm for the general SP.

Approximation algorithm A.

Step 1. For each $t \in \{r_j : j = 1, \ldots, n\}$, we divide the jobs into two sets of jobs such that $S_1(t) = \{J_j : r_j \leq t\}$ and $S_2(t) = \{J_j : r_j > t\}$.
Step 2. Setting $r_j = 0$ for each $J_j \in S_1(t)$, we obtain a new instance $I(t)$ just containing the jobs in $S_1(t)$. Apply algorithm DP1 to $I(t)$, and let $B_1, \ldots, B_k$ be the batches of the accepted jobs obtained from DP1. Sequence $B_1, \ldots, B_k$ from time t in an arbitrary order on the machine and reject all the other jobs.
Step 3. Among all the schedules obtained above, select the one with the minimum objective value.

Let π be the schedule obtained by the above algorithm A. Let Z and Z^* be the objective values of schedule π and an optimal schedule π^*, respectively. We have the following conclusion.

Theorem 5. $Z \leq 2Z^*$.

Proof. The proof is similar to that of theorem 3.1 in [14]. □

3.4. A polynomial time approximation scheme for general SP

Let Z and Z^* be the objective values of schedule π obtained from the above algorithm A and an optimal schedule π^*, respectively. By Theorem 5, we have $Z^* \leq Z \leq 2Z^*$. For any job J_j with $w_j > Z$, J_j must be accepted in Z^*. Similarly, for any job J_j with $r_j > Z$ or $p_j > Z$, J_j must be rejected in Z^*. We modify r_j, p_j and w_j such that $r_j = \min\{r_j, Z\}$, $p_j = \min\{p_j, Z\}$, and $w_j = \min\{w_j, Z\}$. Clearly, this modification does not change the optimal objective value. Thus, we can assume that $\max\{r_j, p_j, w_j\} \leq Z$ for each $j = 1, \ldots, n$. Now, we propose a polynomial time approximation scheme for this problem.

Algorithm ScheduleSplit

Step 1. For any $\epsilon > 0$, set $\delta = \epsilon Z/2n$. Given an instance I, we define a new instance I' by rounding r_j, p_j and w_j in I such that $r'_j = \lfloor r_j/\delta \rfloor \delta$, $p'_j = \lfloor p_j/M \rfloor M$ and $w'_j = \lfloor w_j/M \rfloor M$ for $j = 1, \ldots, n$.
Step 2. Apply algorithm DP2 to the instance I' to obtain an optimal solution $\pi^*(I')$ for the instance I'.
Step 3. Increase the starting time of each job in $\pi^*(I')$ by δ and replace p'_j and w'_j by the original p_j and w_j in $\pi^*(I')$, respectively, for each $j = 1, \ldots, n$, to obtain a feasible solution π for the instance I.

Let Z_ϵ be the objective value of the schedule π obtained from A_ϵ. We have the following theorem.

Theorem 6. *Algorithm ScheduleSplit is a polynomial time approximation scheme for the problem* $1|p\text{-}batch, split, size_j, r_j|C_{\max} + W$.

Proof. Let $Z^*(I')$ be the objective value of the schedule $\pi^*(I')$. Clearly, we have $Z^*(I') \leq Z^*$. Increasing the starting time of each job in $\pi^*(I')$ by δ increases the objective value by at most $2\epsilon Z^*$. Thus, we have $Z_\epsilon \leq Z^*(I') + 2\epsilon Z^* + \sum_{j=1}^n (p_j - p'_j) + \sum_{j=1}^n (w_j - w'_j) \leq Z^* + 2\epsilon Z^* + nM + nM \leq (1 + 4\epsilon)Z^*$. Because $r_j \leq Z$, there are at most $(1/\epsilon) + 1$ distinct release dates in I'. Since $p_j \leq Z$ for $j = 1, \ldots, n$, we have $\sum_{j=1}^n \lfloor p_j/M \rfloor \leq 2n/\epsilon \sum_{j=1}^n (p_j/Z) \leq 2n^2/\epsilon$. Similarly, we have $\sum_{j=1}^n \lfloor w_j/M \rfloor \leq 2n/\epsilon \sum_{j=1}^n (w_j/Z) \leq 2n^2/\epsilon$. Thus, the time complexity of DP2 is $O(n^2 k B^{k+2} p_1^{k-1} (\sum w_j)(\sum p_j)^k) = O(n^2((1/\epsilon) + 1)B^{(1/\epsilon)+3}(2n^2/\epsilon)^{(2/\epsilon)+2})$, confirming that algorithm A_ϵ is a polynomial time approximation scheme. □

4. An approximation algorithm for GP

In this section, we give two constant-factor approximation algorithms for GP.

4.1. A 2-approximation algorithm for GP with identical release dates

By algorithm DP1, we get the following algorithm.

Algorithm SchedulingIdenticalR

Step 1. Get a schedule π_1 for SP with identical release dates using Algorithm DP1.
Step 2. Move out all split jobs from π_1 and open a new batch for each of them.
Step 3. Process the new batches one by one at the end of π_1^*, where π_1^* is the schedule that is obtained from π_1 after removing from it all split jobs.

Theorem 7. *Algorithm SchedulingIdenticalR is a* 2*-approximation algorithm for GP with identical release dates.*

Proof. Let π_1 and π^* be the optimal schedule of the same instance for SP and GP with identical release dates, respectively. Denote by π_2 the schedule given by *Algorithm SchedulingIdenticalR*. In addition, let $Cost_{\max}$ be the objective value of the schedule π_2. It is obvious that π_2 is a feasible schedule for GP and π^* is a feasible schedule for SP. Note that π_2 consists of two parts, one of which is the remaining part of π_1 after removing the split jobs, and the other is the new batches opened for the split jobs. Denote by $Cost^*$ and $Cost_1$ the objective values of schedule π^* and π_1, respectively. Clearly, $Cost_1 \leq Cost^*$. Consider the total processing time $Cost_2$ of the latter part of π_2. Since in π_1 each batch splits at most one job and the split job is the smallest one in this batch, we have $Cost_2 \leq Cost_1$. Thus we get $Cost_{\max} = Cost_1 + Cost_2 \leq 2Cost_1 \leq 2Cost^*$. □

4.2. A $(2+\epsilon)$-approximation algorithm for GP

By lemma 3 and algorithm ScheduleSplit, we get the following algorithm.

Algorithm ScheduleWhole

Step 1. Get a $(1+\epsilon/2)$-approximation schedule π_1 for SP using Algorithm ScheduleSplit.
Step 2. Move out all split jobs from π_1 and open a new batch for each of them.
Step 3. Process the new batches one by one at the end of π_1^*, where π_1^* is the schedule that is obtained from π_1 after removing from it all split jobs.

Theorem 8. *Algorithm ScheduleWhole is a $(2+\epsilon)$-approximation algorithm for GP, where $\epsilon > 0$ can be made arbitrarily small.*

Proof. The proof is similar to that of Theorem 7. □

5. Conclusion

In this paper, we study the parallel batch scheduling problem with non-identical job sizes. We first propose a PTAS for the case where jobs can be split. Then, we give a $(2+\epsilon)$-approximation algorithm for the general problem. We also consider several special cases and present corresponding algorithms for them. Our future work will focus on lowering the approximation ratios and designing an APTAS (Asymptotic Polynomial Time Approximation Scheme) for the problem considered.

References

[1] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, Annals of Discrete Mathematics 5 (1979) 1–15.
[2] S. Webster, K.R. Baker, Scheduling groups of jobs on a single machine, Operations Researching 43 (1995) 692–703.
[3] C.N. Potts, L.N. Van Wassenhove, Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity, Journal of the Operational Research Society 43 (1992) 345–406.
[4] C.N. Potts, M.Y. Kovalyov, Scheduling with batching: A review, European Journal of Operational Research 120 (2000) 228–249.
[5] P. Brucker, A. Gladky, H. Hoogeveen, M.Y. Kovalyov, C.N. Potts, T. Tautenhahn, S.L. van de Velde, Scheduling a batching machine, Journal of Scheduling 1 (1998) 31–54.
[6] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, L. Stougie, Multiprocessor scheduling with rejection, SIAM Journal on Discrete Mathematics 13 (2000) 64–78.
[7] S.S. Seiden, Preemptive multiprocessor scheduling with rejection, Theoretical Computer Science 262 (2001) 437–458.
[8] H. Hoogeveen, M. Skutella, G.J. Woeginger, Preemptive scheduling with rejection, Mathematical Programming 94 (2003) 361–374.
[9] Z. Cao, Z. Wang, Y. Zhang, S. Liu, On several scheduling problems with rejection or discretely compressible processing times, Lecture Notes in Computer Science 3959 (2006) 90–98.
[10] Z. Cao, Y. Zhang, Scheduling with rejection and non-identical job arrivals, Journal of Systems Science and Complexity 20 (2007) 529–535.
[11] Z. Wang, Z. Cao, Y. Zhang, Single machine batch scheduling with rejection to minimize makespan, Journal of Qufu Normal University Natural Science 33 (2007) 35–38 (In Chinese).
[12] L. Zhang, L. Lu, J. Yuan, Single machine scheduling with release dates and rejection, European Journal of Operational Research 198 (2009) 975–978.
[13] Y. Cheng, S. Sun, Scheduling linear deteriorating jobs with rejection on a single machine, European Journal of Operational Research 194 (2009) 18–27.
[14] L. Lu, T.C.E. Cheng, J. Yuan, L. Zhang, Bounded single-machine parallel-batch scheduling with release dates and rejection, Computers and Operations Research 36 (2009) 2748–2751.
[15] L. Lu, L. Zhang, J. Yuan, The unbounded parallel batch machine scheduling with release dates and rejection to minimize makespan, Theoretical Computer Science 396 (2008) 283–289.
[16] H. Feng, H. Liu, Minimizing makespan on an unbounded batch processor with rejection and release times, 2009 World Congress on Computer Science and Information Engineering, CSIE 2009, pp. 587–590.
[17] G. Zhang, X. Cai, C.-Y Lee, C.K. Wong, Minimizing makespan on a single batch processing machine with nonidentical job sizes, Naval Research Logistics 48 (2001) 226–240.
[18] R. Uzsoy, Scheduling a single batch processing machine with non-identical job sizes, International Journal of Production Research 32 (1994) 1615–1635.
[19] S. Li, G. Li, X. Wang, Q. Liu, Minimizing makespan on a single batching machine with release times and non-identical job sizes, Operations Research Letters 33 (2005) 157–164.
[20] G. Dósa, Y. He, Bin packing problems with rejection penalties and their dual problems, Information and Computation 204 (2006) 795–815.
[21] L. Epstein, Bin packing with rejection revisited, Algorithmica (2008) doi:10.1007/s00453-008-9188-9.