https://doi.org/10.1016/j.ipl.2016.02.008

Faster Algorithms for Single Machine Scheduling with Release Dates and Rejection

Jinwen Ou
Department of Administrative Management
Jinan University
Guangzhou, 510632, People's Republic of China
Phone: +86-20-8522-3243
Email: *toujinwen@jnu.edu.cn*

Xueling Zhong
Department of Internet Finance and Information Engineering
Guangdong University of Finance
Guangzhou 510520, People's Republic of China
Phone: +86-20-3820-2796
Email: *zhongxuel@hotmail.com*

Chung-Lun Li*
Department of Logistics and Maritime Studies
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
Phone: +852-2766-7410
Email: *chung-lun.li@polyu.edu.hk*

9 March 2015
Revised 23 February 2016

*Corresponding author

Abstract

We consider the single machine scheduling problem with release dates and job rejection with an objective of minimizing the makespan of the job schedule plus the total rejection penalty of the rejected jobs. Zhang *et al.* [6] have presented a 2-approximation algorithm with an $O(n^2)$ complexity for this problem and an exact algorithm with an $O(n^3)$ complexity for the special case with identical job processing times. In this note, we show that the 2-approximation algorithm developed by Zhang *et al.* [6] can be implemented in $O(n \log n)$ time. We also develop a new exact algorithm with an improved complexity of $O(n^2 \log n)$ for the special case with identical job processing times. The second algorithm can be easily extended to solve the parallel-machine case with the same running time complexity, which answers an open question recently raised by Zhang and Lu [5].

Keywords: Scheduling; release dates; rejection penalty; algorithms; worst-case analysis

1 Introduction

In this note, we consider the following single-machine scheduling problem with release dates and job rejection decisions: There are a single machine and a set of n jobs $J = \{J_1, J_2, \ldots, J_n\}$. Each job $J_j \in J$ has a processing time $p_j \geq 0$, a release date $r_j \geq 0$, and a rejection penalty $w_j > 0$. Job J_j is either rejected, which incurs a rejection penalty w_j, or accepted and processed by the machine at or after its release date r_j. The machine can process at most one job at a time, and preemption is not allowed during job processing. The objective is to determine a subset of jobs to be accepted and processed on the machine, so as to minimize the makespan of the job schedule plus the total rejection penalty of all rejected jobs. This problem was introduced by Zhang *et al.* [6]. Following their notation, this problem is denoted as $1|r_j, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$, and the special case with identical job processing times is denoted as $1|r_j, p_j = p, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$, where R is the set of rejected jobs.

Zhang *et al.* [6] have shown the NP-hardness of problem $1|r_j, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$ and provided several exact and approximation algorithms for the problem. One of their main results is a 2-approximation algorithm with an $O(n^2)$ running time for this problem. Another main result is an exact algorithm with a running time of $O(n(r_{\max}+P))$, where $r_{\max} = \max_j\{r_j\}$ and $P = \sum_j p_j$, and this exact algorithm has an $O(n^3)$ running time when applied to the special case $1|r_j, p_j = p, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$. Although a lot of studies on related topics have been published recently (see [3] and [4] for recent reviews), to the best of our knowledge, no improved algorithms for these two specific problems have appeared in the literature, except for the recent work by He *et al.* [1] (see Remark 1 below). In the following sections, we first show that the 2-approximation algorithm developed by Zhang *et al.* [6] for problem $1|r_j, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$ can be implemented in $O(n \log n)$ time. We then develop a new exact algorithm with a reduced complexity of $O(n^2 \log n)$ for problem $1|r_j, p_j = p, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$. This algorithm can be extended to solve the parallel machine problem $P|r_j, p_j = p, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$, which answers an open question recently raised by Zhang and Lu [5].

2 Faster Approximation Algorithm for $1|r_j, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$

Zhang *et al.* [6] have developed an approximation algorithm with an $O(n^2)$ complexity for problem $1|r_j, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$. Denoting $p(S) = \sum_{J_j \in S} p_j$ for any job subset S, their algorithm is given as follows:

Algorithm A (Zhang *et al.* [6])

Step 1. For each $t \in \{r_j \mid j = 1, 2, \ldots, n\}$, we divide the jobs into three sets such that $S_1(t) = \{J_j \mid r_j \leq t \text{ and } p_j \leq w_j\}$, $S_2(t) = \{J_j \mid r_j \leq t \text{ and } p_j > w_j\}$, and $S_3(t) = \{J_j \mid r_j > t\}$.

Step 2. Accept all jobs in $S_1(t)$ and reject the jobs in $S_2(t) \cup S_3(t)$. Assign the accepted jobs to be processed in time interval $[t, t+p(S_1(t))]$ on the machine. The resulting schedule is denoted by $\pi(t)$.

Step 3. Let $Z(t)$ be the value of the objective function for each $\pi(t)$. Among all the schedules obtained above, select the one with the minimum $Z(t)$ value.

Zhang *et al.* [6, Thm. 4.1] have shown that Algorithm A has a performance ratio of 2. In their proof, they have implicitly assumed that at least one job must be accepted. If we allow the solution to reject all jobs, then Algorithm A does not have a constant performance ratio. To see this, consider an example with $n = 1$, $p_1 = 1$, $w_1 = 2$, and $r_1 = M$, where M is a large number. In this example, $S_1(M) = \{J_1\}$ and $S_2(M) = S_3(M) = \emptyset$. The only schedule generated by Steps 1–2 is $\pi(M)$, which processes job J_1 in the time interval $[M, M+1]$. The objective value of this solution is $M + 1$. On the other hand, the optimal solution is to reject J_1, and has an objective value of 2. Thus, the performance ratio of the solution generated by Algorithm A is $\frac{M+1}{2}$, which approaches infinity as $M \to \infty$. To ensure that the algorithm can also handle the case where all jobs are rejected, we consider a modified version of Algorithm A by adding the following step.

Step 4. Compute $W = \sum_{j=1}^{n} w_j$. If the minimum $Z(t)$ value obtained by Step 3 is greater than W, then reject all jobs.

We denote this modified algorithm as Algorithm A1. It is easy to check that after adding Step 4 to Algorithm A, the proof of Theorem 4.1 in [6] is valid even if the optimal solution is to reject all jobs. Hence, Algorithm A1 is a 2-approximation algorithm.

In the following, we show that Algorithm A1 can be implemented in $O(n \log n)$ time. We first re-index the jobs such that $r_1 \leq r_2 \leq \cdots \leq r_n$. This requires $O(n \log n)$ time. Denote $P_k = p(S_1(r_k))$ and $\bar{W}_k = \sum_{J_j \in J \setminus S_1(r_k)} w_j$, for $k = 1, 2, \ldots, n$. It is easy to check that $Z(r_k) = r_k + P_k + \bar{W}_k$. Next, we show that the values of $Z(r_1), Z(r_2), \ldots, Z(r_n)$ can be computed recursively in $O(n)$ time.

Consider any $k = 2, 3, \ldots, n$. If $p_k > w_k$, then $S_1(r_k) = S_1(r_{k-1})$, $P_k = P_{k-1}$, $\bar{W}_k = \bar{W}_{k-1}$, and therefore $Z(r_k) = r_k + P_k + \bar{W}_k = r_k + P_{k-1} + \bar{W}_{k-1} = r_k + Z(r_{k-1}) - r_{k-1}$. If $p_k \leq w_k$, then $S_1(r_k) = S_1(r_{k-1}) \cup \{J_k\}$, $P_k = P_{k-1} + p_k$, $\bar{W}_k = \bar{W}_{k-1} - w_k$, and therefore $Z(r_k) = r_k + P_k + \bar{W}_k = r_k + P_{k-1} + p_k + \bar{W}_{k-1} - w_k = r_k + Z(r_{k-1}) - r_{k-1} + p_k - w_k$. Thus, in both cases,

$$Z(r_k) = r_k + Z(r_{k-1}) - r_{k-1} + \min\{p_k - w_k, 0\}. \tag{1}$$

Clearly, $Z(r_1)$ can be determined in $O(n)$ time. By repeatedly applying equation (1), the values of $Z(r_2), Z(r_3), \ldots, Z(r_n)$ can be computed in $O(n)$ time. In other words, when the jobs are indexed in nondecreasing release dates, $Z(r_1), Z(r_2), \ldots, Z(r_n)$ can be obtained without determining the sets $S_1(t)$, $S_2(t)$, and $S_3(t)$ for $t = r_1, r_2, \ldots, r_n$. After computing $Z(r_1), Z(r_2), \ldots, Z(r_n)$, we determine $S_1(r_{k'})$, where $k' = \arg\min_{k=1,\ldots,n}\{Z(r_k)\}$. The schedule in Step 3 is obtained by accepting the jobs in $S_1(r_{k'})$ and assigning them to time interval $[r_{k'}, r_{k'} + P_{k'}]$. This step, as well as Step 4, requires $O(n)$ time. Hence, we have the following result.

Theorem 1 *Algorithm A1 can be implemented in $O(n \log n)$ time.*

To illustrate this computation process, consider an example with $n = 5$, $(r_1, r_2, r_3, r_4, r_5) = (0, 2, 6, 10, 16)$, $(p_1, p_2, p_3, p_4, p_5) = (7, 1, 10, 4, 3)$, and $(w_1, w_2, w_3, w_4, w_5) = (8, 4, 12, 3, 7)$. In this example, $P_1 = 7$, $\bar{W}_1 = 26$, and therefore $Z(r_1) = r_1 + P_1 + \bar{W}_1 = 33$. By equation (1),

$$\begin{aligned}
Z(r_2) &= r_2 + Z(r_1) - r_1 + \min\{p_2 - w_2, 0\} = 2 + 33 - 0 + \min\{-3, 0\} = 32;\\
Z(r_3) &= r_3 + Z(r_2) - r_2 + \min\{p_3 - w_3, 0\} = 6 + 32 - 2 + \min\{-2, 0\} = 34;\\
Z(r_4) &= r_4 + Z(r_3) - r_3 + \min\{p_4 - w_4, 0\} = 10 + 34 - 6 + \min\{1, 0\} = 38;\\
Z(r_5) &= r_5 + Z(r_4) - r_4 + \min\{p_5 - w_5, 0\} = 16 + 38 - 10 + \min\{-4, 0\} = 40.
\end{aligned}$$

Thus, $k' = 2$, and the solution generated by Step 3 is to accept the jobs in $S_1(r_2) = \{J_1, J_2\}$, assign

them to the time interval $[r_2, r_2 + P_2] = [2, 10]$, and reject the other jobs. The objective value of this solution is 32. Since $W = 34 > 32$, in Step 4 we keep this solution instead of rejecting all jobs.

Remark 1 We would like to point out that He *et al.* [1] have independently developed a 5/4-approximation for this problem recently, where the complexity is also $O(n \log n)$. However, their algorithm requires a heap to store the data in order to achieve an $O(n \log n)$ complexity.

3 Faster Exact Algorithm for $1|r_j, p_j = p, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$

In this section, we present an exact algorithm for problem $1|r_j, p_j = p, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$. This algorithm has a computational complexity of $O(n^2 \log n)$. From Lemma 3.1 of [6], there exists an optimal schedule such that the accepted jobs are processed according to the earliest release date rule (ERD-rule). The following lemma provides an additional optimality property of this problem.

Lemma 1 *Given any problem instance of* $1|r_j, p_j = p, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$. *If it is not optimal to reject all the jobs in* J, *then there exists an optimal solution with the accepted jobs processed according to the ERD-rule such that the completion time of an accepted job is equal to* $r_\mu + kp$ *for some* $k \in \{2 - \mu, 3 - \mu, \ldots, n + 1 - \mu\}$.

Proof: The proof is straightforward and is omitted. ∎

Let Π denote a restricted version of problem $1|r_j, p_j = p, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$ in which the property stated in Lemma 1 is required to be satisfied if the solution is not to reject all the jobs in J. Solving problem Π yields an optimal solution of $1|r_j, p_j = p, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$. Hence, our algorithm is developed for solving problem Π.

Lemma 1 implies that for each μ, there are n time slots that the accepted jobs can potentially be assigned to. We first re-index the jobs such that $r_1 \leq r_2 \leq \cdots \leq r_n$. For any $\mu = 1, 2, \ldots, n$, we define time intervals

$$\mathcal{I}_{\mu i} = [r_\mu + (i - \mu)p, r_\mu + (i + 1 - \mu)p], \quad \text{for } i = 1, 2, \ldots, n,$$

and we define job subsets

$$H_{\mu i} = \begin{cases} \{J_j \in J \mid r_j \le r_\mu + (1-\mu)p\}, & \text{for } i = 1; \\ \{J_j \in J \mid r_\mu + (i-1-\mu)p < r_j \le r_\mu + (i-\mu)p\}, & \text{for } i = 2, 3, \ldots, n. \end{cases}$$

Note that some of these job subsets may be empty, and that $H_{\mu 1} \cup H_{\mu 2} \cup \cdots \cup H_{\mu n}$ contains all the jobs that can be processed within the time period $[0, r_\mu + (n-\mu)p]$. Also note that $\mathcal{I}_{\mu 1}, \mathcal{I}_{\mu 2}, \ldots, \mathcal{I}_{\mu n}$ cover all the possible time slots that an accepted job can be assigned to. Some of $\mathcal{I}_{\mu 1}, \mathcal{I}_{\mu 2}, \ldots, \mathcal{I}_{\mu n}$ may cover certain negative time periods, but our algorithm will not assign any job to these time intervals.

In our algorithm, we use σ^* to keep track of the best solution obtained so far, and we use Z^* to denote the objective value of σ^*. The algorithm has the following steps:

Algorithm A2

Step 1: Re-index the jobs in ascending order of release dates.

Step 2: Let σ^* be the solution obtained by rejecting all the jobs in J, and set $Z^* \leftarrow \sum_{j=1}^{n} w_j$.

Step 3: For $\mu = 1, 2, \ldots, n$ do

Step 3.1: Construct job subsets $H_{\mu 1}, H_{\mu 2}, \ldots, H_{\mu n}$.

Step 3.2: Set $W_\mu \leftarrow \sum_{j=1}^{n} w_j$.

Step 3.3: For $i = 1, 2, \ldots, n$ do

Step 3.3.1: If $H_{\mu 1} \cup H_{\mu 2} \cup \cdots \cup H_{\mu i} \neq \emptyset$, then let J_x be the job in $H_{\mu 1} \cup H_{\mu 2} \cup \cdots \cup H_{\mu i}$ with the largest rejection penalty (with ties broken arbitrarily), remove J_x from $H_{\mu 1} \cup H_{\mu 2} \cup \cdots \cup H_{\mu i}$, assign J_x to time interval $\mathcal{I}_{\mu i}$, and set $W_\mu \leftarrow W_\mu - w_x$.

Step 3.3.2: Let $\sigma_{\mu i}$ denote the solution obtained by accepting those jobs assigned to time intervals $\mathcal{I}_{\mu 1}, \mathcal{I}_{\mu 2}, \ldots, \mathcal{I}_{\mu i}$ and rejecting the other jobs. If $r_\mu + (i+1-\mu)p + W_\mu < Z^*$, then set $\sigma^* \leftarrow \sigma_{\mu i}$ and $Z^* \leftarrow r_\mu + (i+1-\mu)p + W_\mu$.

Step 2 considers the possibility of rejecting all the jobs in J. Step 3 considers the possibility of not rejecting all the jobs in J and enumerates all possible values of μ. For each value of μ, Steps 3.1 and 3.2 initialize job subsets $H_{\mu 1}, H_{\mu 2}, \ldots, H_{\mu n}$ and variable W_μ, where variable W_μ keeps track

of the total rejection penalty of the non-accepted jobs, while Step 3.3 assigns jobs to the time intervals. In the ith iteration of Step 3.3, Step 3.3.1 selects a job from $H_{\mu 1} \cup H_{\mu 2} \cup \cdots \cup H_{\mu i}$ and assigns it to time interval $\mathcal{I}_{\mu i}$. The result of this step is a feasible solution $\sigma_{\mu i}$ with a makespan no greater than $r_\mu + (i + 1 - \mu)p$. Step 3.3.2 compares $\sigma_{\mu i}$ with the incumbent solution σ^* and selects the better of the two.

To illustrate the algorithm, consider a numerical example with $n = 5$, $p = 5$, $(r_1, r_2, r_3, r_4, r_5) = (0, 2, 6, 10, 16)$, and $(w_1, w_2, w_3, w_4, w_5) = (8, 4, 12, 3, 7)$. In Step 3, μ is set equal to $1, 2, 3, 4, 5$. Consider, for example, the case where $\mu = 3$. In this case, $\mathcal{I}_{3,1} = [-4, 1]$, $\mathcal{I}_{3,2} = [1, 6]$, $\mathcal{I}_{3,3} = [6, 11]$, $\mathcal{I}_{3,4} = [11, 16]$, $\mathcal{I}_{3,5} = [16, 21]$, $H_{3,1} = \emptyset$, $H_{3,2} = \{J_1\}$, $H_{3,3} = \{J_2, J_3\}$, $H_{3,4} = \{J_4\}$, and $H_{3,5} = \{J_5\}$. In Step 3.3, no job is assigned to time interval $\mathcal{I}_{3,1} = [-4, 1]$, and solution $\sigma_{3,1}$ is generated by rejecting all jobs, with a solution value of 34. Then, J_1 is assigned to time interval $\mathcal{I}_{3,2} = [1, 6]$, and solution $\sigma_{3,2}$ is generated by rejecting J_2, J_3, J_4, J_5, with a solution value of 32. Next, J_3 is assigned to time interval $\mathcal{I}_{3,3} = [6, 11]$, and solution $\sigma_{3,3}$ is generated by rejecting J_2, J_4, J_5, with a solution value of 25. Next, J_2 is assigned to time interval $\mathcal{I}_{3,4} = [11, 16]$, and solution $\sigma_{3,4}$ is generated by rejecting J_4, J_5, with a solution value of 26. Finally, J_5 is assigned to time interval $\mathcal{I}_{3,5} = [16, 21]$, and solution $\sigma_{3,5}$ is generated by rejecting J_4, with a solution value of 24. Hence, when $\mu = 3$, solution $\sigma_{3,5}$ is the best among the five solutions generated. In fact, after executing Step 3 for $\mu = 1, 2, 3, 4, 5$, we obtain $\sigma^* = \sigma_{3,5}$. Therefore, the solution value obtained by the algorithm is $Z^* = 24$.

Theorem 2 *Algorithm A2 solves problem* Π *optimally.*

Proof: Clearly, Algorithm A2 can identify the optimal solution if rejecting all the jobs in J is an optimal solution. We thus focus on the case where rejecting all the jobs in J is not optimal.

Consider any values of μ and i. Let $\Pi_{\mu i}$ denote a modified version of problem Π with an additional constraint that the makespan $C_{\max}$ must not exceed $r_\mu + (i + 1 - \mu)p$ and an objective of minimizing $r_\mu + (i + 1 - \mu)p + \sum_{J_j \in R} w_j$. The ith iteration of Step 3.3 aims to solve problem $\Pi_{\mu i}$. By Lemma 1, for each value of μ, the set $\{r_\mu + (i + 1 - \mu)p \mid i = 1, 2, \ldots, n\}$ covers all possible makespan values. So, Step 3.3 enumerates all possible makespan values. Hence, the solution generated by

Algorithm A2 is optimal to problem Π if the solution $\sigma_{\mu i}$ generated by the ith iteration of Step 3.3 is an optimal solution of problem $\Pi_{\mu i}$. Therefore, it suffices to show that assigning J_x to time interval $\mathcal{I}_{\mu i}$ in Step 3.3.1 will generate the best possible solution of $\Pi_{\mu i}$.

Suppose, to the contrary, that Step 3.3.1 does not generate the best possible solution of $\Pi_{\mu i}$. Let $\mathcal{I}_{\mu h}$, $1 \leq h \leq i$, denote the first time interval where Step 3.3.1 makes a suboptimal job assignment decision for problem $\Pi_{\mu i}$. Then, on one hand, Step 3.3.1 must assign some job to $\mathcal{I}_{\mu h}$ (otherwise both Step 3.3.1 and the optimal job assignment will not assign any job to $\mathcal{I}_{\mu h}$, and Step 3.3.1 will not make a suboptimal job assignment decision for time interval $\mathcal{I}_{\mu h}$). On the other hand, an optimal decision for problem $\Pi_{\mu i}$ selects either no job, or a job with different rejection penalty, for $\mathcal{I}_{\mu h}$. Let J_y be the job that Step 3.3.1 assigns to $\mathcal{I}_{\mu h}$. We consider two different cases:

Case 1: The optimal job assignment decision is not to assign any job to $\mathcal{I}_{\mu h}$. In this case, the optimal solution must assign J_y to one of the time intervals $\mathcal{I}_{\mu,h+1}, \mathcal{I}_{\mu,h+2}, \ldots, \mathcal{I}_{\mu i}$, otherwise assigning J_y to $\mathcal{I}_{\mu h}$ would yield a lower cost solution (since $w_y > 0$). However, we can move J_y from its assigned time interval to $\mathcal{I}_{\mu h}$ without increasing the total cost. Hence, assigning J_y to $\mathcal{I}_{\mu h}$ is not a suboptimal decision, which is a contradiction.

Case 2: The optimal job assignment decision is to assign some job, say J_z, to $\mathcal{I}_{\mu h}$. Note that Step 3.3.1 selects J_y because it has the largest rejection penalty among all job candidates. Thus, $w_z < w_y$. The optimal solution must assign J_y to one of the time intervals $\mathcal{I}_{\mu,h+1}, \mathcal{I}_{\mu,h+2}, \ldots, \mathcal{I}_{\mu i}$, otherwise replacing J_z by J_y in time interval $\mathcal{I}_{\mu h}$ would yield a lower cost solution (since $w_y - w_z > 0$). However, we can swap the assignments of J_y and J_z without increasing the total cost. Hence, assigning J_y to $\mathcal{I}_{\mu h}$ is not a suboptimal decision, which is a contradiction.

Combining Cases 1 and 2, we conclude that Step 3.3.1 always makes the optimal job assignment decision for problem $\Pi_{\mu i}$. This completes the proof of the theorem. ∎

Theorem 3 *Algorithm A2 can be implemented in $O(n^2 \log n)$ time.*

Proof: Step 3 is executed n times, each for a different value of μ. For each value of μ, we use a balanced binary tree data structure (also known as AVL tree) to store the rejection penalties of the elements of the set $H_{\mu 1} \cup H_{\mu 2} \cup \cdots \cup H_{\mu i}$. This enables us to insert an element to or remove

an element from the set in $O(\log n)$ time (see, e.g., [2], Sec. 6.2.3). Thus, in Step 3.3.1, searching and removing J_x from this set can be achieved in $O(\log n)$ time. At the beginning of each iteration of Step 3.3, we need to append a new set $H_{\mu i}$ to the set $H_{\mu 1} \cup H_{\mu 2} \cup \cdots \cup H_{\mu,i-1}$. To do so, we insert each element of $H_{\mu i}$ one by one to $H_{\mu 1} \cup H_{\mu 2} \cup \cdots \cup H_{\mu,i-1}$. Each insertion requires $O(\log n)$ time. Hence, appending $H_{\mu i}$ to $H_{\mu 1} \cup H_{\mu 2} \cup \cdots \cup H_{\mu,i-1}$ requires $O(|H_{\mu i}| \log n)$ time, and the total execution time of the n iterations of Step 3.3.1 is $O(\sum_{i=1}^{n} |H_{\mu i}| \log n) = O(n \log n)$.

In Step 3.3.2, instead of updating the incumbent solution σ^* whenever a better solution is found, we only keep track of the μ and i values of the incumbent solution. Then, each execution of Step 3.3.2 requires only $O(1)$ time. Hence, each iteration of Step 3 requires $O(n \log n)$ time, and the total running time of Steps 1–3 is $O(n^2 \log n)$.

Let μ^* and i^* be the μ and i values, respectively, of the optimal solution. When the algorithm completes Steps 1–3, we can construct the optimal schedule by executing the μ^*-th iteration of Step 3 once more (and stop after finishing the first i^* iterations of Step 3.3). This extra step does not affect the $O(n^2 \log n)$ complexity of the algorithm. ∎

Theorems 2 and 3 imply that problem $1|r_j, p_j = p, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$ can be solved in $O(n^2 \log n)$ time.

It is easy to extend Algorithm A2 to solve the parallel machine problem $P|r_j, p_j = p, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$ with the running time remaining at $O(n^2 \log n)$. To do so, we first observe that Lemma 1 remains valid for problem $P|r_j, p_j = p, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$ (the validity proof is straightforward and is omitted). Thus, it suffices to consider those solutions in which the completion time of any accepted job is equal to $r_\mu + kp$ for some $\mu \in \{1, 2, \ldots, n\}$ and $k \in \{2-\mu, 3-\mu, \ldots, n+1-\mu\}$. As in the single machine case, for each given μ, we generate n time intervals $\mathcal{I}_{\mu 1}, \mathcal{I}_{\mu 2}, \ldots, \mathcal{I}_{\mu n}$, partition J into n subsets $H_{\mu 1}, H_{\mu 2}, \ldots, H_{\mu n}$, and apply Algorithm A2 to assign jobs to those time intervals one by one. For the parallel-machine case, the only difference is that each time interval can be assigned up to m jobs, where m is the number of machines. In other words, for the parallel-machine case, we repeat Step 3.3.1 of Algorithm A2 m times if $H_{\mu 1} \cup H_{\mu 2} \cup \cdots \cup H_{\mu i}$ has not been emptied. It is easy to verify that the proof of Theorem 2 remains valid for this extended algorithm. The

argument in the proof of Theorem 3 also remains valid, except that in the extended algorithm, for each combination of μ and i, Step 3.3.1 may be repeated up to m times. However, for each μ, Step 3.3.1 will assign at most n jobs to time intervals $\mathcal{I}_{\mu 1}, \mathcal{I}_{\mu 2}, \ldots, \mathcal{I}_{\mu n}$, and each assignment requires $O(\log n)$ time. Hence, the total running time of Step 3.3.1 for different combinations of μ and i is $O(n^2 \log n)$. Therefore, the overall running time of the extended algorithm is $O(n^2 \log n)$. This answers an open question raised by Zhang and Lu [5] regarding the computational complexity status of problem $P|r_j, p_j = p, \text{reject}|C_{\max} + \sum_{J_j \in R} w_j$.

Acknowledgments

This research was supported in part by NSFC 71101064 and NSFC 71501051. The second author was supported in part by Humanities and Social Sciences Research Foundation of Ministry of Education of China 13YJC630239. The third author was supported in part by The Hong Kong Polytechnic University under grant 1-BBZL.

References

[1] C. He, J.Y.-T. Leung, K. Lee, M.L. Pinedo, Improved algorithms for single machine scheduling with release dates and rejections, 4OR (2016), to appear.

[2] D.E. Knuth, The Art of Computer Programming, Volume 3: Sorting and Searching, 2nd edition, Addison-Wesley, Reading, MA, 1998.

[3] D. Shabtay, N. Gaspar, M. Kaspi, A survey on offline scheduling with rejection, Journal of Scheduling 16 (2013) 3–28.

[4] S.A. Slotnick, Order acceptance and scheduling: A taxonomy and review, European Journal of Operational Research 212 (2011) 1–11.

[5] L.Q. Zhang, L.F. Lu, Parallel-machine scheduling with release dates and rejection, 4OR (2016), to appear.

[6] L.Q. Zhang, L.F. Lu, J.J. Yuan, Single machine scheduling with release dates and rejection, European Journal of Operational Research 198 (2009) 975–978.