Scheduling Linear Deteriorating Jobs with an Availability Constraint on a Single Machine[1]

Min Ji [a, b, 2] Yong He [b, 3] T.C.E. Cheng [c, 4]

[a] College of Computer Science & Information Engineering, Zhejiang Gongshang University, Hangzhou 310035, P.R. China
[b] Department of Mathematics, State Key Lab of CAD & CG, Zhejiang University, Hangzhou 310027, P.R. China
[c] Department of Logistics, The Hong Kong Polytechnic University, Kowloon, Hong Kong

[1]This research was supported by the Teaching and Research Award Program for Outstanding Young Teachers in Higher Education Institutions of the MOE, China, and the National Natural Science Foundation of China (10271110, 60021201). The third author was supported in part by The Hong Kong Polytechnic University under a grant from the *Area of Strategic Development in China Business Services*.

[2]Email: jimkeen@163.com
[3]Email: mathhey@zju.edu.cn
[4]Corresponding author. Email: LGTcheng@polyu.edu.hk

Abstract

We consider a single machine scheduling problem in which the processing time of a job is a simple linear increasing function of its starting time and the machine is subject to an availability constraint. We consider the non-resumable case. The objectives are to minimize the makespan and the total completion time. We show that both problems are NP-hard and present pseudo-polynomial time optimal algorithms to solve them. Furthermore, for the makespan problem, we present an optimal approximation algorithm for the on-line case, and a fully polynomial time approximation scheme for the off-line case. For the total completion time problem, we provide a heuristic and evaluate its efficiency by computational experiments.

Keywords. Scheduling; Computational complexity; Approximation algorithms; Deteriorating job; Availability constraint

1 Introduction

For most scheduling problems it is assumed that the job processing times are fixed parameters [17], and the machines are available at any time. However, such restrictive assumptions represent an oversimplified view of reality. Job processing times are not necessarily deterministic because jobs may deteriorate while waiting to be processed. Examples can be found in financial management, steel production, resource allocation and national defense, etc., where any delay in processing a job may result in deterioration in accomplishing the job. For a list of applications, the reader is referred to Kunnathur and Gupta [12], and Mosheiov [16]. Such problems are generally known as the deteriorating job scheduling problem. The assumption of the continuing availability of machines may not be valid in a real production situation, either. Scheduling problems with machine availability constraints often arise in industry due to preventive maintenance (a deterministic event) or breakdown of machines (a stochastic phenomenon) over the scheduling horizon. In this paper we will study the deteriorating job scheduling problem with a machine availability constraint due to a deterministic event.

Work on the deteriorating job scheduling problem was initiated by Brown and Yechiali [3] and Gupta and Gupta [10]. They focused on the single machine makespan problem under linear deteriorating conditions. Since then, scheduling problems with time-dependent processing times have received increasing attention. An extensive survey of different models and problems was provided by Alidaee and Womer [1]. Cheng, Ding and Lin [5] recently presented an updated survey of the results on scheduling problems with time-dependent processing times.

Graves and Lee [9] point out that machine scheduling with an availability constraint is very important but still relatively unexplored. They studied a scheduling problem with a machine availability constraint in which maintenance needs to be performed within a fixed period. Lee [13] presented an extensive study of the single and parallel machine scheduling problems with an availability constraint with respect to various performance measures. Two cases are usually considered for such problems. If a job cannot be finished before the next down period of a machine and the job can continue after the machine becomes available again, it is called *resumable*. On the other hand, it is called *non-resumable* if the job has to restart rather than continue. For more details, the reader may refer to Lee, Lei and Pinedo [14].

The problem under consideration can be formally described as follows: There are n independent jobs $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$ to be processed non-preemptively on a single machine which is available at time $t_0 > 0$. Let p_j, α_j and C_j denote the actual processing time, the growth (or deteriorating) rate and the completion time of J_j, respectively. The actual processing time of job J_j is $p_j = \alpha_j s_j$, where s_j is the starting time of J_j in a schedule. We assume that the machine is not available during the period between time b_1 and b_2 (which is called the *non-available period*), where $b_2 > b_1 > t_0$. The processing of any job is non-resumable. Let $C_{\max} = \max_{1,\cdots,n}\{C_j\}$ and $Z = \sum_{j=1}^{n} C_j$ denote the makespan and the total completion time of a given schedule, respectively. The objectives are

to minimize the makespan and the total completion time. Using the three-field notation of [8], we denote these two problems as $1/nr-a, p_j = \alpha_j s_j / C_{\max}$ and $1/nr-a, p_j = \alpha_j s_j / \sum C_j$, respectively. Using the same denotation as Lee [13], here $nr-a$ in the second field denotes *a non-resumable availability constraint*.

The above defined problems may date back to Mosheiov [15], who first considered a special case of the problems where the machine is available at any time from t_0. The most commonly used performance measures were considered, such as makespan, total completion time, total weighted completion time, total weighted waiting time, total tardiness, number of tardy jobs, maximum lateness and maximum tardiness. He shows that all these models are polynomially solvable. Chen [4] extended the study to parallel machines, and considered $P/p_j = \alpha_j s_j / \sum C_j$. He shows that the problem is NP-hard even with a fixed number of machines. When the number of the machine is arbitrary, he proves that there is no polynomial approximation algorithm with a constant worst-case ratio. He also gives an approximation algorithm with a parameter dependent worst-case ratio for the two machine case.

For the problem with a machine availability constraint, Wu and Lee [18] studied the resumable case of the makespan problem, denoted by $1/r-a, p_j = \alpha_j s_j / C_{\max}$. They show that the problem can be transformed into a 0-1 integer program and a linear equation problem. However, the computational complexity of this problem is still unknown. To the best of our knowledge, $1/nr-a, p_j = \alpha_j s_j / C_{\max}$ and $1/nr-a, p_j = \alpha_j s_j / \sum C_j$ are still unexplored.

In scheduling theory, a problem is called *on-line* (over list) if jobs come one by one in a list, and they are scheduled irrevocably on the machines as soon as they arrive without any information about the jobs that will come later. On the other hand, if we have full information about the jobs before constructing a schedule, the problem is called *off-line*. Algorithms for on-line and off-line problems are called *on-line* and *off-line algorithms*, respectively. The quality of an approximation algorithm is usually measured by its worst-case ratio (for off-line problems) or competitive ratio (for on-line problems), respectively. Specifically, let $C_A(I)$ (or briefly C_A) denote the objective value yielded by an approximation algorithm A, and $C_{OPT}(I)$ (or briefly C_{OPT}) denote the objective value produced by an optimal off-line algorithm. Then the *worst-case ratio* (or *competitive ratio*) of algorithm A is defined as the smallest number c such that for any instance I, $C_A(I) \leq cC_{OPT}(I)$. An on-line algorithm A is called *optimal* if there does not exist any other on-line algorithm with a competitive ratio smaller than that of A.

In this paper we show that both of the problems $1/nr-a, p_j = \alpha_j s_j / C_{\max}$ and $1/nr-a, p_j = \alpha_j s_j / \sum C_j$ are NP-hard and present their respective pseudo-polynomial time optimal algorithms. Furthermore, for the problem $1/nr-a, p_j = \alpha_j s_j / C_{\max}$, we present an optimal approximation algorithm for the on-line case, and a fully polynomial time approximation scheme for the off-line case. For the problem $1/nr-a, p_j = \alpha_j s_j / \sum C_j$, we provide a heuristic and evaluate its efficiency by computational experiments.

In the following, we use the symbol [ ] to denote the order of jobs in a sequence. Thus, the actual processing time of the job scheduled in the first position is $p_{[1]} = \alpha_{[1]}t_0$, and its completion time is $C_{[1]} = t_0 + p_{[1]} = t_0(1 + \alpha_{[1]})$. Similarly, by induction, the completion time of the job in the jth position is $C_{[j]} = C_{[j-1]} + p_{[j]} = t_0 \prod_{i=1}^{j}(1 + \alpha_{[i]})$, if it is processed before the non-available period.

2 Minimizing the makespan

2.1 NP-hardness

Theorem 1 *The problem* $1/nr - a, p_j = \alpha_j s_j / C_{\max}$ *is NP-hard.*

Proof. We show the result by reducing the Subset Product problem, which is NP-hard [6, 11], to our problem in polynomial time. An instance I of the Subset Product problem is formulated as follows:

Given a finite set $S = \{1, 2, \cdots, k\}$, a size $x_j \in Z^+$ for each $j \in S$, and a positive integer A, does there exist a subset $T \subseteq S$ such that the product of the sizes of the elements in T satisfies $\prod_{j \in T} x_j = A$?

In the above instance, we can omit the element $j \in S$ with $x_j = 1$ because it will not affect the product of any subset. Therefore, without loss of generality, we can assume that $x_j \geq 2$ for every $j \in S$. Furthermore, we can assume that $B = \prod_{j \in S} x_j / A$ is an integer since otherwise it can be immediately answered that there is no solution to the instance. We set $D = \prod_{j \in S} x_j = AB$. Then $D \geq 2^k$, since every $x_j \geq 2$.

For any given instance I of the Subset Product problem, we construct the corresponding instance II of our problem as follows:

– Number of jobs: $n = k$.

– Jobs' available time: $t_0 > 0$, arbitrary.

– The start time of the non-available period: $b_1 = t_0 A$.

– The end time of the non-available period: $b_2 > b_1$, arbitrary.

– Jobs' growth rates: $\alpha_j = x_j - 1$, for $j = 1, 2, \cdots, n$.

– Threshold: $G = b_2 B$.

It is clear that the reduction can be done in polynomial time. We prove that the instance I has a solution if and only if the instance II has a schedule with makespan no greater than G.

If I has a solution, then we can process the jobs in $\{J_j | j \in T\}$ before b_1 since $t_0 \prod_{j \in T}(1 + \alpha_j) = t_0 \prod_{j \in T} x_j = t_0 A = b_1$, and process the jobs in $\{J_j | j \in S \setminus T\}$ at or after b_2 without introducing any idle time between consecutive jobs. Thus, we get a feasible schedule with makespan

$$C_{\max} = b_2 \prod_{j \in S \setminus T} (1 + \alpha_j) = b_2 \prod_{j \in S \setminus T} x_j = G.$$

Hence, we obtain a solution for II.

If II has a solution, then there exists a schedule in the form of (R_1, R_2) with $C_{\max} \leq G$, where R_1 and R_2 are subsets of S. The jobs in $\{J_j | j \in R_1\}$ start before b_1, while the jobs in $\{J_j | j \in R_2\}$ start at or after b_2. We have $t_0 \prod_{j \in R_1}(1+\alpha_j) \leq b_1$. It implies that $\prod_{j \in R_1} x_j \leq A$.

If $\prod_{j \in R_1} x_j < A$, then $\prod_{j \in R_2} x_j = D/(\prod_{j \in R_1} x_j) > B$. It follows that

$$C_{\max} = b_2 \prod_{j \in R_2}(1+\alpha_j) = b_2 \prod_{j \in R_2} x_j > b_2 B = G,$$

a contradiction. So $\prod_{j \in R_1} x_j = A$, and we get a solution for I. □

To show the problem is not strongly NP-hard, we give a pseudo-polynomial time algorithm based on dynamical programming for our problem. In this paper, we assume that all parameters of the problems are integers when we present pseudo-polynomial time algorithms. In the remainder of this section, we assume that $t_0 \prod_{j=1}^{n}(1+\alpha_j) > b_1$. Otherwise, all jobs can be finished by the non-available period and the problem becomes trivial. It is clear that the jobs processed before the non-available period are sequence independent, and so are the jobs processed after the non-available period in our problem.

Let $f_j(u)$ be the minimum total processing time of the jobs that are processed at or after b_2, if (i) we have assigned jobs $J_1, J_2, \cdots, J_j$, and (ii) the total processing time of the jobs assigned before b_1 is u. Given $f_{j-1}(u)$ for $0 \leq u \leq b_1 - t_0$, we can process J_j at time either $s_j < b_1$ or $s_j \geq b_2$. In the former case, the total processing time of the jobs processed at or after b_2 does not change, but u increases by $\alpha_j(u+t_0)$. In the latter case, the makespan increases by $\alpha_j(f_{j-1}(u)+b_2)$, while u does not change. We have the following initial condition:

$$f_j(u) = \begin{cases} 0, & \text{if } j = 0, u = 0, \\ \infty, & \text{otherwise.} \end{cases}$$

And the recursion for $j = 1, \cdots, n$, and $u = 0, \cdots, b_1 - t_0$, is

$$\begin{aligned} f_j(u) &= \begin{cases} \min\left\{ f_{j-1}\left(\frac{u-\alpha_j t_0}{1+\alpha_j}\right), f_{j-1}(u) + \alpha_j(b_2 + f_{j-1}(u)) \right\}, & \text{if } \frac{u-\alpha_j t_0}{1+\alpha_j} \text{ is an integer,} \\ f_{j-1}(u) + \alpha_j\left(b_2 + f_{j-1}(u)\right), & \text{otherwise,} \end{cases} \\ &= \begin{cases} \min\left\{ f_{j-1}\left(\frac{u-\alpha_j t_0}{1+\alpha_j}\right), (1+\alpha_j) f_{j-1}(u) + a_j b_2 \right\}, & \text{if } \frac{u-\alpha_j t_0}{1+\alpha_j} \text{ is an integer,} \\ (1+\alpha_j) f_{j-1}(u) + a_j b_2, & \text{otherwise.} \end{cases} \end{aligned}$$

The optimal makespan is then determined as

$$C_{OPT} = b_2 + \min_{0 \leq u \leq b_1 - t_0} f_n(u).$$

It is clear that this algorithm requires at most $O(n(b_1 - t_0))$ time. Hence, we have the following conclusion.

Corollary 2 *The problem $1/nr-a, p_j = \alpha_j s_j / C_{\max}$ is NP-hard in the ordinary sense.*

2.2 On-line algorithm

In this section we give an optimal approximation algorithm for the on-line case of the problem $1/nr-a, p_j = \alpha_j s_j / C_{\max}$.

Algorithm *LS*: Always schedule an incoming job such that it can be completed as early as possible.

It is clear that the time complexity of algorithm LS is $O(n)$. The following theorem shows that this algorithm is optimal. In the remainder of this section, denote by $\mathcal{T}_{OPT}$ the set consisting of all of the jobs processed after the non-available period, and by $\mathcal{E}_{OPT}$ the remaining jobs that are processed before the non-available period, in an optimal schedule.

Theorem 3 *Algorithm LS is an optimal online algorithm for the problem* $1/nr-a, p_j = \alpha_j s_j / C_{\max}$ *with a competitive ratio* $\frac{b_1}{t_0}$.

Proof. It is clear that $\mathcal{T}_{OPT} \neq \emptyset$. Otherwise, we would obtain $C_{LS} = C_{OPT} = t_0 \prod_{j=1}^{n}(1+\alpha_j)$. So we have

$$t_0 \prod_{J_j \in \mathcal{E}_{OPT}} (1+\alpha_j) \leq b_1, \tag{1}$$

and

$$C_{OPT} = b_2 \prod_{J_j \in \mathcal{T}_{OPT}} (1+\alpha_j). \tag{2}$$

Eq. (2) implies that $b_2 \prod_{j=1}^{n}(1+\alpha_j) = C_{OPT} \prod_{J_j \in \mathcal{E}_{OPT}}(1+\alpha_j)$. Combining this with (1), we get $b_2 \prod_{j=1}^{n}(1+\alpha_j) \leq \frac{b_1}{t_0} C_{OPT}$. On the other hand, we have $C_{LS} \leq b_2 \prod_{j=1}^{n}(1+\alpha_j)$, since $b_2 \prod_{j=1}^{n}(1+\alpha_j)$ is the objective value if we process all the jobs after the non-available period. So we have

$$C_{LS} \leq \frac{b_1}{t_0} C_{OPT}.$$

To show the optimality of algorithm LS, we consider the following instances. The first job J_1 with $\alpha_1 = \varepsilon$ comes. Suppose an algorithm A processes J_1 at time x. If $x \geq b_2$, then no more job comes. We have $C_A \geq (1+\alpha_1)b_2$, $C_{OPT} = (1+\alpha_1)t_0$, and $\frac{C_A}{C_{OPT}} \geq \frac{b_2}{t_0} > \frac{b_1}{t_0}$. If $x < b_1$, then the last job J_2 with $\alpha_2 = \frac{b_1}{t_0} - 1$ comes. We have $C_A \geq (1+\alpha_2)b_2 = \frac{b_1}{t_0} b_2$, $C_{OPT} = (1+\alpha_1)b_2 = (1+\varepsilon)b_2$, and thus $\frac{C_A}{C_{OPT}} \geq \frac{b_1/t_0}{1+\varepsilon} \to \frac{b_1}{t_0}$ when $\varepsilon \to 0$. Hence, we conclude that any on-line algorithm A has a competitive ratio no less than $\frac{b_1}{t_0}$ and algorithm LS is optimal. □

2.3 Off-line algorithm

In this subsection we consider the off-line case. It is natural to modify algorithm LS by adding a preparatory step that re-orders the jobs in non-increasing order of their growth rates. Denote the modified algorithm by algorithm LGR (largest-growth-rate first). However, we show in the following that this greedy-like algorithm cannot have a constant worst-case ratio.

Algorithm *LGR***:** First re-order the jobs such that $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_n$, then schedule them in this order by algorithm LS.

It is clear that the time complexity of algorithm LGR is $O(n \log n)$.

Theorem 4 *Algorithm LGR has a tight worst-case ratio of*

$$c = \begin{cases} 1 + \alpha_{\min}, & \text{if } 1 + \alpha_{\min} \leq \frac{b_1}{t_0}, \\ 1, & \text{else,} \end{cases}$$

where $\alpha_{\min} = \min_{j=1,2,\cdots,n} \alpha_j$.

Proof. If $1 + \alpha_{\min} > \frac{b_1}{t_0}$, i.e., $t_0(1 + \alpha_{\min}) > b_1$, then all the jobs must be processed after the non-available period, and hence LGR produces an optimal schedule. So we are left to consider the case $1 + \alpha_{\min} \leq \frac{b_1}{t_0}$. We prove the result by contradiction. Suppose that there exists a counterexample that violates our ratio $1+\alpha_{\min}$, hence a *minimal counterexample* with the fewest possible jobs should exist. From now on, we assume that we are dealing with the minimal counterexample, denoted by $I = (\mathcal{J}, t_0, b_1, b_2)$.

Lemma 5 *In the minimal counterexample, if a job J_j is processed before the non-available period in the LGR schedule, then it must be processed after the non-available period in the optimal schedule; and if a job J_j is processed after the non-available period in the LGR schedule, then it must be processed before the non-available period in the optimal schedule.*

Proof. We only prove the first conclusion, and the second one can be proved similarly. If a job J_j is completed before the non-available period not only in the LGR schedule but also in the optimal schedule, we can construct a new instance I' from I with $I' = (\mathcal{J} \setminus \{J_j\}, t_0, b_1 = b_1/(1 + \alpha_j), b_2 = b_2/(1 + \alpha_j))$. It is obvious that $C_{LGR}(I') = C_{LGR}(I)/(1 + a_j)$ and $C_{OPT}(I') \leq C_{OPT}(I)/(1 + \alpha_j)$. Then, $\frac{C_{LGR}(I')}{C_{OPT}(I')} \geq \frac{C_{LGR}(I)/(1+\alpha_j)}{C_{OPT}(I)/(1+\alpha_j)} \geq \frac{C_{LGR}(I)}{C_{OPT}(I)}$, and hence I' is a smaller counterexample, a contradiction. □

Lemma 6 *In the minimal counterexample, the job J_i with $\alpha_i = \alpha_{\min}$ must be processed after the non-available period in the LGR schedule.*

Proof. If the job J_i is processed before the non-available period in the LGR schedule, then we can construct a new instance I' from I by deleting the job J_i from $\mathcal{J}$. It is clear that $C_{LGR}(I') = C_{LGR}(I)$ and $C_{OPT}(I') \leq C_{OPT}(I)$. Therefore, $\frac{C_{LGR}(I')}{C_{OPT}(I')} \geq \frac{C_{LGR}(I)}{C_{OPT}(I)}$, which states that I' is a smaller counterexample, a contradiction. □

Now we turn to the proof of Theorem 4. Denote $\mathcal{T}_{LGR}$={the jobs processed after the non-available period in the LGR schedule}. Then, from Lemma 5, we know that in the minimal counterexample:

(i) all of the jobs in $\mathcal{T}_{OPT}$ are processed before the non-available period in the LGR schedule,

(ii) all of the jobs in $\mathcal{T}_{LGR}$ are processed before the non-available period in the optimal schedule.

From (i), we know that $t_0 \prod_{J_j \in \mathcal{T}_{OPT}}(1+\alpha_j) \cdot (1+\alpha_i) > b_1$ for any $J_i \in \mathcal{T}_{LGR}$. Then by Lemma 6, we get $t_0 \prod_{J_j \in \mathcal{T}_{OPT}}(1+\alpha_j) \cdot (1+\alpha_{\min}) > b_1$, or equivalently $\prod_{J_j \in \mathcal{T}_{OPT}}(1+\alpha_j) > \frac{b_1}{t_0} \cdot \frac{1}{1+\alpha_{\min}}$. On the other hand, from (ii) it follows that $t_0 \prod_{J_j \in \mathcal{T}_{LGR}}(1+\alpha_j) \leq b_1$, or equivalently $\prod_{J_j \in \mathcal{T}_{LGR}}(1+\alpha_j) \leq \frac{b_1}{t_0}$. Since $C_{LGR} = b_2 \prod_{J_j \in \mathcal{T}_{LGR}}(1+\alpha_j)$ and $C_{OPT} = b_2 \prod_{J_j \in \mathcal{T}_{OPT}}(1+\alpha_j)$, we have

$$\frac{C_{LGR}}{C_{OPT}} \leq \frac{\frac{b_1}{t_0}}{\frac{b_1}{t_0} \cdot \frac{1}{1+\alpha_{\min}}} = 1+\alpha_{\min},$$

this is the desired contradiction.

To show that the worst-case ratio cannot be smaller than $1+\alpha_{\min}$, we consider the following instance: $I = (\{J_1, J_2, J_3\}, t_0, b_1, b_2)$ with $\alpha_1 = \sqrt{\frac{b_1}{t_0}} + \varepsilon - 1$, $\alpha_2 = \alpha_3 = \sqrt{\frac{b_1}{t_0}} - 1$. It is not difficult to obtain that $C_{LGR}(I) = b_2 \frac{b_1}{t_0}$ and $C_{OPT}(I) = b_2(\sqrt{\frac{b_1}{t_0}} + \varepsilon)$. Thus, the worst-case ratio tends to $\sqrt{\frac{b_1}{t_0}} = 1+\alpha_{\min}$ when ε tends to 0. By now we have completed the proof of Theorem 4. □

In the remainder of this subsection, we give an FPTAS for the problem $1/nr-a, p_j = \alpha_j s_j / C_{\max}$. We apply the FPTAS for the classical 0-1 Minimum Knapsack problem as a sub-procedure. Recall that for any instance of the 0-1 Minimum Knapsack problem, we are given n items, each with a profit c_j and a weight w_j, and a knapsack with capacity C. We wish to put items into the knapsack such that the total weight of the selected items is not greater than C and the total profit of unselected items is minimized. For this problem, Babat [2] presented an FPTAS with time complexity $O(n^4/\varepsilon)$, and Gens and Levner [7] proposed an FPTAS with time complexity $O(n^2/\varepsilon)$.

To construct an FPTAS for our problem, it is crucial to determine which jobs are processed after the non-available period. We do it as follows: For any instance I of our problem and any positive number $\varepsilon > 0$, we set $D = \prod_{j=1}^{n}(1+\alpha_j)$ and $\delta = \log_D(1+\varepsilon)$. We construct the instance II of the Minimum Knapsack problem in the following way: For each job $J_j, j = 1, \cdots, n$, define an item with profit $c_j = \ln(1+\alpha_j)$ and weight $w_j = c_j$, and set the capacity of the knapsack as $C = \ln \frac{b_1}{t_0}$. Let B_{KNAP} denote the optimal value of the constructed instance II. Apply any FPTAS to instance II such that its objective value is not larger than $(1+\delta)B_{KNAP}$. Thus we obtain a partial solution for instance I. Namely, for every item put into the knapsack by the FPTAS, we schedule the corresponding job before the non-available period, and schedule all remaining jobs after the non-available period. Since if we denote by $\mathcal{E}_{KNAP}$ all the selected items in the instance II, then we have $\sum_{J_j \in \mathcal{E}_{KNAP}} \ln(1+\alpha_j) \leq C = \ln \frac{b_1}{t_0}$, which implies that $t_0 \prod_{J_j \in \mathcal{E}_{KNAP}}(1+\alpha_j) \leq b_1$.

Algorithm KP**:**

Step 1. If $t_0 \prod_{j=1}^{n}(1+\alpha_j) \leq b_1$, then output $C_{KP} = t_0 \prod_{j=1}^{n}(1+\alpha_j)$. Else, goto Step 2.

Step 2. Determine the jobs processed after the non-available period by applying the FPTAS for the Minimum Knapsack problem as above. Denote by $\mathcal{T}_{KP}$ the set consisting of all jobs processed after the non-available period. Then, the resulting makespan is $C_{KP} = b_2 \prod_{J_j \in \mathcal{T}_{KP}}(1+\alpha_j)$.

Theorem 7 *Algorithm KP is an FPTAS for the problem* $1/nr-a, p_j=\alpha_j s_j/C_{\max}$, *which runs in* $O(n^2/\varepsilon)$, *i.e., for any positive number* $\varepsilon>0$, *we have* $\frac{C_{KP}}{C_{OPT}}\leq 1+\varepsilon$.

Proof. It is clear that $\mathcal{T}_{OPT}\neq\emptyset$. Otherwise, we obtain $C_{KP}=C_{OPT}=t_0\prod_{j=1}^{n}(1+\alpha_j)$. Hence, $C_{OPT}=b_2\prod_{J_j\in\mathcal{T}_{OPT}}(1+\alpha_j)$. Denote $B_{OPT}=\sum_{J_j\in\mathcal{T}_{OPT}}\ln(1+\alpha_j)$. Then $B_{KNAP}\leq B_{OPT}$ holds obviously. From the rule of algorithm KP, we have

$$\sum_{J_j\in\mathcal{T}_{KP}}\ln(1+\alpha_j)\leq(1+\delta)B_{KNAP}\leq(1+\delta)B_{OPT}. \tag{3}$$

Eq. (3) implies that

$$\begin{aligned}\prod_{J_j\in\mathcal{T}_{KP}}(1+\alpha_j) &\leq \left(\prod_{J_j\in\mathcal{T}_{OPT}}(1+\alpha_j)\right)^{\delta}\cdot\left(\prod_{J_j\in\mathcal{T}_{OPT}}(1+\alpha_j)\right)\\ &\leq D^{\delta}\cdot\prod_{J_j\in\mathcal{T}_{OPT}}(1+\alpha_j)\\ &= (1+\varepsilon)\prod_{J_j\in\mathcal{T}_{OPT}}(1+\alpha_j).\end{aligned} \tag{4}$$

It follows that

$$\frac{C_{KP}}{C_{OPT}}=\frac{b_2\prod_{J_j\in\mathcal{T}_{KP}}(1+\alpha_j)}{b_2\prod_{J_j\in\mathcal{T}_{OPT}}(1+\alpha_j)}\leq 1+\varepsilon. \tag{5}$$

It is clear that algorithm KP has the same time complexity $O(n^2/\varepsilon)$ as that of the FPTAS for the Minimum Knapsack problem. □

3 Minimizing the total completion time

3.1 NP-hardness

Theorem 8 *The problem* $1/nr-a, p_j=\alpha_j s_j/\sum C_j$ *is NP-hard.*

Proof. We again show the result by a reduction from the Subset Product problem. Let I be an instance of the Subset Product problem described in Section 2.1, and we construct the corresponding instance II of the problem as follows:

– Number of jobs: $n=k+4$.

– Jobs' available time: $t_0>0$, arbitrary.

– The start time of the non-available period: $b_1=t_0D^5$.

– The end time of the non-available period: $b_2>b_1$, arbitrary.

– Jobs' growth rates: $\alpha_j=x_j-1$, for $j=1,2,\cdots,k$; $\alpha_{k+1}=DA-1$, $\alpha_{k+2}=DB-1$, $\alpha_{k+3}=\alpha_{k+4}=D^3-1$.

– Threshold: $G=(k+2)b_2D^2+(t_0+b_2)D^5$.

We prove that the instance I has a solution if and only if the instance II has a schedule with the total completion time no greater than G. We do it by verifying the following lemmas.

Lemma 9 *For any subset $T \subseteq S$, we have $B\prod_{j\in T} x_j + A\prod_{j\in S\setminus T} x_j \geq 2D$, and the equality holds if and only if $\prod_{j\in T} x_j = A$ and $\prod_{j\in S\setminus T} x_j = B$.*

Proof. The result follows immediately from the well-known inequality $a + b \geq 2\sqrt{ab}$ $(a, b \geq 0)$ and the equality holds if and only if $a = b$. □

Lemma 10 *If there exists a solution for the instance I, then there exists a schedule π for the instance II with the total completion time $Z(\pi) \leq G$.*

Proof. If there exists a subset $T \subseteq S$ such that $\prod_{j\in T} x_j = A$ (and hence $\prod_{j\in S\setminus T} x_j = B$), then we can construct a schedule π as follows: First process all the jobs of $\{J_j | j \in S \setminus T\}$, and jobs J_{k+1}, J_{k+4} from time t_0 to b_1. Then from time b_2, process all the jobs of $\{J_j | j \in T\}$, and jobs J_{k+2}, J_{k+3}. We have

$$C_{k+1} = t_0(\prod_{j\in S\setminus T}(1+\alpha_j))(1+\alpha_{k+1}) = t_0(\prod_{j\in S\setminus T} x_j)DA = t_0BDA = t_0D^2,$$

$$C_{k+2} = b_2(\prod_{j\in T}(1+\alpha_j))(1+\alpha_{k+2}) = b_2(\prod_{j\in T} x_j)DB = b_2ADB = b_2D^2,$$

$$C_{k+3} = C_{k+2}(1+\alpha_{k+3}) = b_2D^5, \qquad C_{k+4} = C_{k+1}(1+\alpha_{k+4}) = t_0D^5 = b_1.$$

Since $C_j < C_{k+2} = b_2D^2$ for every $j \in S$ and $C_{k+1} < C_{k+2}$, we have

$$\begin{aligned} Z(\pi) &= \sum_{j\in S} C_j + \sum_{j=1}^{4} C_{k+j} < (k+2)C_{k+2} + C_{k+3} + C_{k+4} \\ &= (k+2)b_2D^2 + (t_0+b_2)D^5 = G. \end{aligned}$$

□

Lemma 11 *If there exists a schedule π for our problem with the total completion time $Z(\pi) \leq G$, then the following results must hold:*
(1) one of the jobs J_{k+3} and J_{k+4} is processed before the non-available period, and the other after the non-available period;
(2) one of the jobs J_{k+1} and J_{k+2} is processed before the non-available period, and the other after the non-available period.

Proof. (1) If both of the jobs J_{k+3} and J_{k+4} are processed before the non-available period, then at least one of their completion times will not be less than $t_0(1+\alpha_{k+3})(1+\alpha_{k+4}) = t_0D^6 > b_1$, a contradiction. If they are both processed after the non-available period, then at least one of their completion times will be greater than or equal to $b_2(1+\alpha_{k+3})(1+\alpha_{k+4}) = b_2D^6$. Noting that $G = (k+2)b_2D^2 + (t_0+b_2)D^5 < b_2D^4 + 2b_2D^5 \leq b_2D^6$, we obtain a contradiction to $Z(\pi) \leq G$, and thus obtain the conclusion.

(2) If both of the jobs J_{k+1} and J_{k+2} are processed before the non-available period, then by (1), either job J_{k+3} or J_{k+4} is also processed before the non-available period. Since $\alpha_{k+3} = \alpha_{k+4}$,

the maximum completion time of these three jobs will be greater than or equal to $t_0(1+\alpha_{k+1})(1+\alpha_{k+2})(1+\alpha_{k+3}) = t_0D^6 > b_1$, a contradiction. If both of the jobs J_{k+1} and J_{k+2} are processed after the non-available period, we can obtain a contradiction similarly. □

Lemma 12 *If there exists a schedule π of the instance II with the total completion time $Z(\pi) \leq G$, then there is a solution for the instance I.*

Proof. From Lemma 11 and the fact that there is no difference between jobs J_{k+3} and J_{k+4}, without loss of generality, we only need to consider the following two cases: (i) jobs J_{k+1} and J_{k+3} are processed before the non-available period, and jobs J_{k+2} and J_{k+4} are processed after the non-available period, (ii) jobs J_{k+2} and J_{k+3} are processed before the non-available period, and jobs J_{k+1} and J_{k+4} are processed after the non-available period.

We first consider case (i). Let $\{J_j | j \in T,\ T \subseteq S\}$ denote the jobs processed after the non-available period. So the maximum completion time of the jobs processed before the non-available period is

$$C_{LB} = t_0(\prod_{j\in S\setminus T}(1+\alpha_j))(1+\alpha_{k+1})(1+\alpha_{k+3}) = t_0(\prod_{j\in S\setminus T} x_j)DAD^3 = t_0D^4(A\prod_{j\in S\setminus T} x_j),$$

and the maximum completion time of the jobs processed after the non-available period is

$$C_{LA} = b_2(\prod_{j\in T}(1+\alpha_j))(1+\alpha_{k+2})(1+\alpha_{k+4}) = b_2(\prod_{j\in T} x_j)DBD^3 = b_2D^4(B\prod_{j\in T} x_j).$$

Then we have

$$\begin{aligned} Z(\pi) &> C_{LB} + C_{LA} = t_0D^4(A\prod_{j\in S\setminus T} x_j) + b_2D^4(B\prod_{j\in T} x_j) \\ &= b_2D^4(A\prod_{j\in S\setminus T} x_j + B\prod_{j\in T} x_j) - (b_2-t_0)D^4(A\prod_{j\in S\setminus T} x_j). \end{aligned} \tag{6}$$

It is clear that $C_{LB} \leq b_1$, i.e., $t_0D^4(A\prod_{j\in S\setminus T} x_j) \leq t_0D^5$, which implies that

$$A\prod_{j\in S\setminus T} x_j \leq D. \tag{7}$$

From (7), it follows that $\prod_{j\in T} x_j \geq A$. If there is no solution for the instance I, then we have $\prod_{j\in T} x_j > A$. From Lemma 9, we have $A\prod_{j\in S\setminus T} x_j + B\prod_{j\in T} x_j > 2D$, and hence

$$A\prod_{j\in S\setminus T} x_j + B\prod_{j\in T} x_j \geq 2D+1 \tag{8}$$

since A, B, and x_j are all positive integers. Combining (6) and (8), we have

$$Z(\pi) > b_2D^4(2D+1) - (b_2-t_0)D^4(A\prod_{j\in S\setminus T} x_j). \tag{9}$$

Substituting (7) into (9), we get

$$Z(\pi) > b_2 D^4(2D+1) - (b_2 - t_0)D^5 = b_2 D^4 + (t_0 + b_2)D^5 > G,$$

a contradiction. Hence, we conclude that there exists a solution for the instance I.

Using the same method for case (ii), we can obtain the same conclusion. So we have completed the proof. □

Lemmas 10 and 12 complete the proof of Theorem 8. □

To show that the problem is not strongly NP-hard, we provide a pseudo-polynomial time algorithm based on dynamical programming for our problem. Using the interchanging argument, the following property for the optimal schedule can be obtained easily.

Property 1 *In the optimal schedule, the jobs processed before the non-available period are processed by the SGR (smallest-growth-rate first) rule, and so are the jobs processed after the non-available period.*

So in the remainder of this subsection, we assume that the jobs are re-indexed in the SGR order. Let $Y = b_2 \prod_{j=1}^{n}(1+\alpha_j)$. We define $f_j(u,v)$ as the minimum total completion time of the jobs that have been processed, if (i) we have assigned jobs $J_1, J_2, \cdots, J_j$, (ii) the total processing time of the jobs assigned before b_1 is u, and the total processing time of the jobs assigned at or after b_2 is v. Given $f_{j-1}(u,v)$ for $0 \le u \le b_1 - t_0$ and $0 \le v \le Y$, we can process J_j at time either $s_j < b_1$ or $s_j \ge b_2$. In the former case, v does not change, but u is increased by $\alpha_j(u+t_0)$ and the total completion time is increased by $(1+\alpha_j)(u+t_0)$. In the latter case, v is increased by $\alpha_j(v+b_2)$ and the total completion time is increased by $(1+\alpha_j)(v+b_2)$, while u remains unchanged. We have the following initial condition:

$$f_j(u,v) = \begin{cases} 0, & \text{if } j = 0, u = 0, v = 0 \\ \infty, & \text{otherwise.} \end{cases}$$

And the recursion for $j = 1, \cdots, n$, $u = 0, \cdots, b_1 - t_0$, and $v = 0, \cdots, Y$, is

$$f_j(u,v) = \begin{cases} \min\left\{ f_{j-1}\left(\frac{u-\alpha_j t_0}{1+\alpha_j}, v\right) + (u+t_0), f_{j-1}\left(u, \frac{v-\alpha_j b_2}{1+\alpha_j}\right) + (v+b_2) \right\}, \\ \qquad \text{if } \frac{u-\alpha_j t_0}{1+\alpha_j} \text{ and } \frac{v-\alpha_j b_2}{1+\alpha_j} \text{ are integers,} \\ f_{j-1}\left(\frac{u-\alpha_j t_0}{1+\alpha_j}, v\right) + (u+t_0), \\ \qquad \text{if } \frac{u-\alpha_j t_0}{1+\alpha_j} \text{ is an integer, and } \frac{v-\alpha_j b_2}{1+\alpha_j} \text{ is not an integer,} \\ f_{j-1}\left(u, \frac{v-\alpha_j b_2}{1+\alpha_j}\right) + (v+b_2), \\ \qquad \text{if } \frac{u-\alpha_j t_0}{1+\alpha_j} \text{ is not an integer, and } \frac{v-\alpha_j b_2}{1+\alpha_j} \text{ is an integer,} \\ \infty, \qquad \text{otherwise.} \end{cases}$$

The optimal objective value is then determined as

$$Z_{OPT} = \min_{0 \le u \le b_1 - t_0, 0 \le v \le Y} f_n(u,v).$$

It is clear that this algorithm requires at most $O(n(b_1 - t_0)Y)$ time. Hence, the problem can be solved in pseudo-polynomial time. Now we can conclude that

Corollary 13 *The problem* $1/nr-a, p_j = \alpha_j s_j / \sum C_j$ *is NP-hard in the ordinary sense.*

3.2 A heuristic algorithm

Next, we construct and experimentally test a heuristic for the problem $1/nr-a, p_j = \alpha_j s_j / \sum C_j$. First we introduce a procedure $LSGR$.

Procedure $LSGR(S)$**:**

Step 1. For a given order S of the job set $\mathcal{J}$, construct a partition of $\mathcal{J}$ in the following way: Let $\mathcal{E}$ be the jobs processed before the non-available period, and $\mathcal{T}$ be the jobs processed after the non-available period, if we schedule all the jobs by algorithm LS according to S.

Step 2. Process the jobs in $\mathcal{E}$ before the non-available period by the SGR rule, and process the jobs in $\mathcal{T}$ after the non-available period by the SGR rule as well.

Now we give a formal description of the algorithm, which is made up of three parallel procedures.

Algorithm $RSGR$**:**

Step 1. Re-order the jobs such that $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_n$.

Step 2. For each order S_j of the job set $\mathcal{J}$ given below, run the procedure $LSGR(S_j)$, $j = 1, 2, 3$. Then choose the best solution as output.

(1) $S_1 : J_1, J_2, \cdots, J_n$.

(2) $S_2 : J_2, J_3, \cdots, J_n, J_1$.

(3) $S_3 : J_1, J_3, \cdots, J_{n-3}, J_{n-1}, J_2, J_4, \cdots, J_{n-2}, J_n$ if n is even, and $J_1, J_3, \cdots, J_{n-2}, J_n, J_2, J_4, \cdots, J_{n-3}, J_{n-1}$ if n is odd.

It can easily be seen that algorithm $RSGR$ can be implemented in $O(n \log n)$. We can conclude by intuition that it is crucial to decrease the number of jobs processed after the non-available period, since the actual processing time of a job is proportional to its starting time. At the same time, we expect that the growth rates of the jobs processed after the non-available period are not so large. The procedure $LSGR(S_1)$ implements the above idea in a greedy way. The procedure $LSGR(S_2)$ tries to avoid the following bad case: the maximum growth rate is so large that the number of the jobs processed before the non-available period is very small. And the procedure $LSGR(S_3)$ is a kind of trade-off between the former two procedures. The three parallel procedures seek to balance the possible bad cases for different instances.

We evaluated algorithm $RSGR$ by experimental tests. The optimal objective value was found by complete enumeration. Because of the exorbitant time needed to find the optimal solutions for large-sized problems, complete enumeration can be applied only for $n \leq 15$. For $n = 10$, 9 different tests with 100 randomly generated instances for each test were performed. Problem parameters were randomly generated according to the uniform distribution except t_0. In all the tests, we set

Table 1: Experimental results for $n = 10$.

Interval for b_1	Interval for $b_2 - b_1$	$\frac{RSGR}{OPT}$	$\max_{i=1}^{100} \frac{RSGR_i}{OPT_i}$	$\frac{RSGR^1}{OPT}$	$\frac{RSGR^2}{OPT}$	$\frac{RSGR^3}{OPT}$
$[A/4, A/2)$	$(0, 10]$	1.146215	1.395317	1.411498	1.193111	1.204305
$[A/4, A/2)$	$(10, 100]$	1.193226	1.532669	1.540146	1.251648	1.276607
$[A/4, A/2)$	$(100, 1000]$	1.223028	1.586359	1.626683	1.290432	1.327243
$[A/2, 3A/4)$	$(0, 10]$	1.020548	1.100006	1.478944	1.028092	1.266856
$[A/2, 3A/4)$	$(10, 100]$	1.048857	1.316561	1.704705	1.068175	1.400590
$[A/2, 3A/4)$	$[100, 1000]$	1.090441	1.474237	1.975430	1.131206	1.576460
$[3A/4, A)$	$(0, 10]$	1.027801	1.167420	1.146719	1.063409	1.055392
$[3A/4, A)$	$(10, 100]$	1.032412	1.441846	1.195518	1.179593	1.065919
$[3A/4, A)$	$(100, 1000]$	1.056544	1.631607	1.283370	1.434572	1.107645

$t_0 = 1$ without loss of generality. The growth rates were generated from the interval $[0, 1]$. We set $A = t_0 \prod_{i=1}^{n}(1+\alpha_i)$. The values of b_1 were generated from the intervals $[A/4, A/2)$, $[A/2, 3A/4)$ and $[3A/4, A)$. The values of $b_2 - b_1$ were generated from the intervals $(0, 10]$, $(10, 100]$ and $(100, 1000]$. Therefore, the combination of all the intervals yields 9 different cases. For a given case, 100 instances were generated. For each instance i, it was solved by the $RSGR$ heuristic and the optimal value was calculated. We denote the corresponding values as $RSGR_i$ and OPT_i. Furthermore, to verify the performance of the three parallel procedures, we denote $RSGR_i^j$ as the value yielded by the procedure $LSGR(S_j)$, $j = 1, 2, 3$. The average ratio $\sum_{i=1}^{100} \frac{RSGR_i}{OPT_i}/100$, the worst ratio $\max_{i=1}^{100} \frac{RSGR_i}{OPT_i}$, and the average ratios $\sum_{i=1}^{100} \frac{RSGR_i^j}{RSGR_i}/100$ are reported in Table 1, where $j = 1, 2, 3$. For simplicity, we denote $\frac{RSGR}{OPT}$, and $\frac{RSGR^j}{RSGR}$ for $j = 1, 2, 3$, as the average ratios mentioned in the above.

Table 1 indicates that algorithm $RSGR$ yields a solution much better than that yielded individually by each procedure. And a solution produced by algorithm $RSGR$ is on average no more than 9.1% worse than an optimal solution except for $b_1 \in [A/4, A/2)$. Even for $b_1 \in [A/4, A/2)$, this value increases to 22.4%.

In addition, we evaluated the worst-case behavior of the algorithm. The solution delivered by the algorithm is no more than 63.2% worse than an optimal solution. It indicates that the performance of algorithm $RSGR$ is bounded well, and the proposed algorithm is acceptable for the considered NP-hard problem.

Table 1 also indicates that the second procedure performs better than the other two procedures except for $b_1 \in [3A/4, A)$, while the third procedure performs the best for $b_1 \in [3A/4, A)$.

Furthermore, the relative performance of the procedures was evaluated. Nine different tests with 100 randomly generated instances were performed. For each instance i of each test, we denote $RSGR_i^{jk}$ as the minimum value of the values yielded by the procedures $LSGR(S_j)$ and $LSGR(S_k)$,

Table 2: Experimental results for $n = 10$ and $n = 20$.

Interval for		$n = 10$			$n = 20$		
b_1	$b_2 - b_1$	$\frac{RSGR^{12}}{RSGR}$	$\frac{RSGR^{23}}{RSGR}$	$\frac{RSGR^{31}}{RSGR}$	$\frac{RSGR^{12}}{RSGR}$	$\frac{RSGR^{23}}{RSGR}$	$\frac{RSGR^{31}}{RSGR}$
$[A/4, A/2)$	$(0, 10]$	1.033579	1.001001	1.049005	1.016530	1.000000	1.185884
$[A/4, A/2)$	$(10, 100]$	1.039559	1.001407	1.067329	1.016701	1.000000	1.190629
$[A/4, A/2)$	$(100, 1000]$	1.043281	1.002068	1.081569	1.017546	1.000000	1.216755
$[A/2, 3A/4)$	$(0, 10]$	1.006282	1.000009	1.236990	1.000000	1.000000	1.503232
$[A/2, 3A/4)$	$(10, 100]$	1.015927	1.000173	1.324239	1.000059	1.000000	1.513900
$[A/2, 3A/4)$	$(100, 1000]$	1.030397	1.001068	1.422773	1.001103	1.000000	1.579965
$[3A/4, A)$	$(0, 10]$	1.012609	<u>1.000000</u>	1.025693	1.003110	1.000000	1.126958
$[3A/4, A)$	$(10, 100]$	1.037631	1.003526	1.024513	1.004436	1.000000	1.125144
$[3A/4, A)$	$(100, 1000]$	1.078416	1.018389	1.023082	1.014696	<u>1.000060</u>	1.118945

$j, k = 1, 2, 3,\ j \neq k$. The average ratios $\sum_{i=1}^{100} \frac{RSGR_i^{jk}}{RSGR_i}/100$ were calculated. In the same way as above, we denote $\frac{RSGR^{jk}}{RSGR}$ as the average ratios for simplicity, where $j = 1, 2, 3;\ k = 1, 2, 3;\ j \neq k$. For $n = 10$ and $n = 20$, the results are reported in Table 2; for $n = 50$ and $n = 100$, the results are listed in Table 3.

From the Tables 2 and 3, we can see that the average ratios $\frac{RSGR^{23}}{RSGR}$ are the best among $\{\frac{RSGR^{12}}{RSGR}, \frac{RSGR^{23}}{RSGR}, \frac{RSGR^{31}}{RSGR}\}$, and are almost equal to 1 if $n \geq 20$ except the underlined result. But this is not the case if n is small. For $n = 10$, the average ratios $\frac{RSGR^{23}}{RSGR}$ are larger than 1 except the underlined result. So we can conclude that $RSGR$ can be revised and simplified through deleting the procedure $LSGR(S_1)$ if n is large enough. Table 3 shows that the parameter $b_2 - b_1$, i.e., the time duration of the non-available period, has no influence on the ratio only except the bold result if n is large enough. The average ratios $\frac{RSGR^{31}}{RSGR}$ are much larger than $\frac{RSGR^{12}}{RSGR}$ and $\frac{RSGR^{23}}{RSGR}$, which also indicates that the second procedure performs better than others on average.

4 Conclusions

In this paper we considered the problem of scheduling deteriorating jobs on a single machine with an availability constraint. We studied the non-resumable case with the objective of minimizing the makespan and total completion time. We showed that both problems are NP-hard in the ordinary sense. For the makespan problem, we presented an optimal approximation algorithm for the on-line case, and a fully polynomial time approximation scheme for the off-line case. For the total completion time problem, we provided a heuristic and evaluated its effectiveness by computational experiments. The computational results show that the heuristics is efficient in obtaining near-optimal solutions.

It will be interesting to find out if an approximation algorithm with a constant worst-case ratio

Table 3: Experimental results for $n = 50$ and $n = 100$.

Interval for		$n = 50$			$n = 100$		
b_1	$b_2 - b_1$	$\frac{RSGR^{12}}{RSGR}$	$\frac{RSGR^{23}}{RSGR}$	$\frac{RSGR^{31}}{RSGR}$	$\frac{RSGR^{12}}{RSGR}$	$\frac{RSGR^{23}}{RSGR}$	$\frac{RSGR^{31}}{RSGR}$
$[A/4, A/2)$	$(0, 10]$	1.003285	1.000000	1.319414	1.000751	1.000000	1.572155
$[A/4, A/2)$	$(10, 100]$	1.003285	1.000000	1.319414	1.000751	1.000000	1.572155
$[A/4, A/2)$	$(100, 1000]$	1.003285	1.000000	1.319414	1.000751	1.000000	1.572155
$[A/2, 3A/4)$	$(0, 10]$	1.000000	1.000000	2.069904	1.000000	1.000000	2.634543
$[A/2, 3A/4)$	$(10, 100]$	1.000000	1.000000	2.069904	1.000000	1.000000	2.634543
$[A/2, 3A/4)$	$(100, 1000]$	1.000000	1.000000	2.069906	1.000000	1.000000	2.634543
$[3A/4, A)$	$(0, 10]$	1.000786	1.000000	1.353337	1.000474	1.000000	1.584777
$[3A/4, A)$	$(10, 100]$	1.000786	1.000000	1.353337	1.000474	1.000000	1.584777
$[3A/4, A)$	$(100, 1000]$	**1.000787**	1.000000	1.353337	1.000474	1.000000	1.584777

exists for the total completion time problem. Extending our problems to parallel machines or flowshops is also an interesting issue. In addition, it is worth studying the problem with the objective of minimizing other scheduling performance criteria.

References

[1] B. Alidaee, N. K. Womer, Scheduling with time dependent processing times: Review and extensions, J. Oper. Res. Soc., 50 (1999) 711-720.

[2] L. G. Babat, Linear functions on the n-dimensional unit cube, Doklady Akademiia Nauk SSSR, 221 (1975) 761-762. (Russian)

[3] S. Brown, U. Yechiali, Scheduling deteriorating jobs on a single process, Oper. Res., 38 (1990) 495-498.

[4] Z. L. Chen, Parallel machine scheduling with time dependent processing times, Discrete Applied Mathematics, 70 (1996) 81-93.

[5] T. C. E. Cheng, Q. Ding, B. M. T. Lin, A concise survey of scheduling with time-dependent processing times, European J. Oper. Res., 152 (2004) 1-13.

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

[7] G. V. Gens, E. V. Levner, Approximate algorithms for certain universal problems in scheduling theory, Engineering Cybernetics, 6 (1978) 38-43. (Russian)

[8] 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) 287-326.

[9] G. H. Graves, C. Y. Lee, Scheduling maintenance and semi-resumable jobs on a single machine, Naval Res. Logist., 46 (1999) 845-863.

[10] J. N. D. Gupta, S. K. Gupta, Single facility scheduling with nonlinear processing times, Comput. Industr. Engrg., 14 (1988) 387-393.

[11] D. S. Johnson, The NP-complete columns: an ongoing guide, J. Algorithms, 2 (1981) 402.

[12] A. S. Kunnathur, S. K. Gupta, Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem, European J. Oper. Res., 47 (1990) 56-64.

[13] C. Y. Lee, Machine scheduling with an availability constraint, J. Global Optimization, 8 (1996) 395-417.

[14] C. Y. Lee, L. Lei, M. Pinedo, Current trend in deterministic scheduling, Annals Oper. Res., 70 (1997) 1-42.

[15] G. Mosheiov, Scheduling jobs under simple linear deterioration, Comput. Oper. Res., 21 (1994) 653-659.

[16] G. Mosheiov, Scheduling jobs with step-deterioration: minimizing makespan on single and multi-machine, Comput. Industr. Engrg., 28 (1995) 869-879.

[17] M. Pinedo, Scheduling: theory, algorithms, and systems, Prentice-Hall, Upper Saddle River, NJ, 2002.

[18] C. C. Wu, W. C. Lee, Scheduling linear deteriorating jobs to minimize makespan with an availability constraint on a single machine, Inform. Process. Lett., 87 (2003) 89-93.