Online Scheduling of Simple Linear Deteriorating Jobs to Minimize Total General Completion Time

Sheng Yu* † Prudence W. H. Wong‡

†School of Business Administration, Zhongnan University of Economics and Law, Wuhan, China.
yusheng@znufe.edu.cn
‡Department of Computer Science, University of Liverpool, Liverpool, UK.
pwong@liverpool.ac.uk

November 18, 2012

Abstract

Traditional scheduling assumes that the processing time of a job is fixed. Yet there are numerous situations that the processing time increases (deteriorates) as the start time increases. In particular, lots of work has been devoted to jobs with simple linear deterioration. The processing time p_j of job J_j is a simple linear function of its start time s_j, precisely, $p_j = b_j s_j$, where b_j is the deteriorating rate. In this paper, we study the problem of online non-preemptive scheduling of jobs with arbitrary release times and simple linear deteriorating rates on a single machine to minimize the total general completion time. We present an algorithm DSDR (Delayed Smallest Deteriorating Rate) and prove that it achieves the best-possible competitive ratio $(1 + b_{\max})^{\alpha}$ for all deterministic online algorithms, where α is the general index of completion time and $\alpha > 0$.

Keywords: Online scheduling, competitive analysis, simple linear deterioration, total general completion time, single machine, release time

*The work is partly done while Sheng Yu was visiting University of Liverpool.

1 Introduction

Scheduling of deteriorating jobs. Scheduling of jobs (with fixed processing time) is a classical problem [20]. Yet, there are numerous situations that the processing time increases (deteriorates) as the start time increases. For example, to schedule maintenance or cleaning, a delay often requires additional effort to accomplish the task. Other examples are found in fire fighting, steel production and financial management [12, 16]. Scheduling of deteriorating jobs was first introduced by Browne and Yechiali [4], and Gupta and Gupta [8] independently. Both considered scheduling a set of deteriorating jobs on a single machine to minimize makespan. In [4], the processing time of a job is a monotone linear function of its starting time while non-linear functions are considered in [8]. Since then, the problem has attracted a lot of attention, and has been studied in other time dependent models with various objective functions. Comprehensive surveys can be found in [1, 6, 7]. Following most of the existing work, we focus on non-preemptive scheduling.[1]

Simple linear deterioration. We focus on jobs with simple linear deterioration, which has been studied in more detail due to its simplicity while capturing the essence of real life situations. A job satisfies *linear deterioration* if its processing time is an increasing linear function of its start time, i.e., $p_j = a_j + b_j s_j$, where $a_j \geq 0$ is the "normal" processing time, $b_j > 0$ is the deteriorating rate, and s_j is the start time. In other words, the processing time differs with different schedules. Linear deterioration is further said to be *simple* if $p_j = b_j s_j$. In this case, in order to avoid trivial solution, it is natural to assume that the start time of the first job is $t_0 > 0$ since a start time of zero means that the processing time of all jobs is zero. Mosheiov [15, 16] justified simple linear deterioration as follows: as the number of jobs increases, the start time of jobs gets larger, and the actual processing time of infinitely many jobs is no longer affected by the normal processing time but only by the deteriorating rate.

Release times and online algorithms. The study of scheduling deteriorating jobs has been focused on the setting where all jobs are available for processing at the very beginning. In practice, jobs may be released at arbitrary times. We may also have to make decisions based on the jobs currently presented without information of future jobs. The performance of online algorithms is typically measured by competitive analysis [3]. An online algorithm is c-competitive if for any input instance, its cost is no more than c times that of the optimal offline algorithm. Online algorithms for jobs with release times have been studied extensively for fixed processing time [21]. Yet, not much is known for deteriorating jobs with release times, let alone online algorithms. Recently, there is some work on online algorithms for linear deteriorating jobs to minimize makespan [5, 24].

Total general completion time. One of the typical objective functions for scheduling problems is measuring the completion time. Let c_j denote the completion time of a job J_j in a certain schedule. The total completion time of scheduling n jobs is defined as $\sum_{1 \leq j \leq n} c_j$. Furthermore, the general completion time [10, 13, 23][2] attempts to characterize the scenario that dissatisfaction increases with delay in processing in a manner of a power function. Motivations of this objective function have been discussed in [10]. The *general completion time* of J_j is defined as c_j^α, where $\alpha > 0$ is a constant. The objective of the problem is to minimize the total

[1] Offline preemptive scheduling of deteriorating jobs has been considered on a single machine [18].

[2] For simplicity, we follow the convention in [13] and call this objective general completion time.

general completion time, i.e., $\sum_{1\le j\le n} c_j^\alpha$. When the processing time is independent of the start time (fixed processing time) and jobs have arbitrary release times, it has been shown in [9] that for the problem of minimizing total completion time, the algorithm DSPT (Delayed Shortest Processing Time) is an optimal online algorithm with competitive ratio 2. Other 2-competitive algorithms have also been proposed [14, 19]. Liu et al. [13] extended the work by Hoogeveen and Vestjens [9] to total general completion time and showed a lower bound of 2^α. They also claimed that DSPT is 2^α-competitive, which is proved in [25].[3]

For simple linear deterioration, Mosheiov [16] has considered the case when all jobs are available at the beginning and proved that to minimize total completion time, SDR (Smallest Deteriorating Rate) is an optimal algorithm. Note that SDR is an online-list algorithm as well.[4] It remains open to obtain a competitive online algorithm when jobs have arbitrary release times and the objective is total completion time or total general completion time.

Our contributions. In this paper, we consider non-preemptive scheduling of jobs with simple linear deterioration and arbitrary release times on a single machine to minimize the total general completion time. This extends current work in two directions: from all jobs being available at the very beginning to jobs having arbitrary release times; and from minimizing total completion time to total general completion time. We first prove that no deterministic online algorithm can be better than $(1 + b_{\max})^\alpha$-competitive, where $b_{\max}$ is the maximum deteriorating rate of all jobs. We then present an algorithm DSDR (Delayed Smallest Deteriorating Rate) and prove that it is $(1 + b_{\max})^\alpha$-competitive, matching the lower bound.

Technically speaking, we adopt the approach in [9] for fixed processing time to compare our online schedule with an optimal offline preemptive schedule. Preemption of jobs with simple linear deterioration has been formalized in [18]. The major challenge to adopt the fixed processing time approach is that the preemption of simple linear deteriorating jobs makes the comparison more difficult and requires more careful accounting. The rough idea is that for any job J, we find a set of jobs that have been processed by DSDR before J and show that any algorithm including the optimal offline preemptive algorithm has to process these jobs before J. Thus the start time and hence the completion time of J by DSDR is bounded by a factor times that by the optimal algorithm.

Remark. Scheduling of linear deteriorating jobs has also been studied in the multiple machines setting [5, 11, 17, 22, 24]. The objective of all these work is to minimize makespan.

Organization of the paper. In Section 2, we formally define the problem and give some notations necessary for discussion. We give the lower and upper bounds in Sections 3 and 4, respectively. We then conclude in Section 5.

2 Preliminaries

We consider *non-preemptive* job scheduling on a single machine. Once a job has been processed, it cannot be interrupted by any other job until it is finished. The input is a set $\mathcal{J}$ of n jobs. A job J_j is released at time r_j with processing time p_j which is a function on the start

[3] A proof is given in [13] for DSPT being 2^α-competitive but a flaw in the arguments has been pointed out in [25], which also gives a proof on the competitive ratio.

[4] In the online-list model, jobs are available to be processed at the beginning but are presented one by one. Each job is to be scheduled before the next job is presented.

time s_j. The job has to be processed contiguously for p_j time units. In particular, we consider *simple linear deterioration* in which jobs are characterized by a deteriorating rate $b_j > 0$ such that $p_j = b_j s_j$. Denote by $b_{\max}$ the maximum of b_j. We assume that the start time of the first job is $t_0 > 0$ since a start time of zero means that the processing time of all jobs is zero.

An *online algorithm* has to determine at any time which job to run without future information about jobs that have not been released yet. For any algorithm $\mathcal{A}$, we use $\mathcal{A}$ to denote its schedule. In a schedule $\mathcal{A}$, we denote the start time and completion time of J_j by $s_j(\mathcal{A})$ and $c_j(\mathcal{A})$, respectively. When the context is clear, we simply use s_j and c_j. The total completion time is defined as $\sum_j c_j$. Furthermore, the *general completion time* of J_j is defined as c_j^α, where $\alpha \geq 0$ is a constant. The objective of the problem is to minimize the total general completion time, i.e., $\sum_j c_j^\alpha$. The performance of an online algorithm is typically measured by competitive analysis [3]. An online algorithm is said to be c-competitive if for all input job sets, its total general completion time is at most c times that of the optimal offline algorithm.

It has been showed that if all jobs are available at the very beginning, the optimal algorithm is to schedule according to the rule SDR (Smallest Deteriorating Rate) [16]. Yet we observe that when jobs have arbitrary release times, one can first release a job J and then n jobs with very small deteriorating rate arrive just moment after J, this would make SDR no longer competitive and thus we need a different algorithm.

Optimal offline preemptive algorithm. Following a similar framework as in [9], the analysis of our online algorithm compares our schedule with that of an optimal offline preemptive algorithm. We describe here the notion of preemption with resume in the presence of simple linear deteriorating jobs. We adopt the same idea as in [18]. Suppose a job J_j starts processing at s and is preempted at t. Note that the processing time of J_j when it starts at s equals to $s \cdot b_j$ and its completion time is expected to be at $s(1 + b_j)$ if it is not preempted. When J_j is preempted at t, we define the *remaining deteriorating rate* b'_j as the value such that the job would be completed at $s(1 + b_j)$ if it is resumed at t, i.e., $t(1 + b'_j) = s(1 + b_j)$. If J_j is to be resumed at s_2 and preempted again later at t_2, the remaining deteriorating rate is then defined in a similar way based on s_2, t_2 and b'_j. Note that the definition here is equivalent to that in [18] although the discussion there uses different formula. The definition leads to the following lemma.

Lemma 1 ([18])**.** *Suppose that $[t_{s_i}, t_{f_i}]$ for $i = 1, \cdots, q$ are q disjoint time intervals, where $0 < t_{s_i} < t_{f_i} < t_{s_{i+1}}$. Then there is a schedule in which J_j is scheduled in these time intervals if and only if $r_j \leq t_{s_1}$ and $\prod_{i=1}^{q} \frac{t_{f_i}}{t_{s_i}} = 1 + b_j$.*

An optimal preemptive algorithm has been presented in [18], which we call *Smallest Remaining Deteriorating Rate* (SRDR). For the sake of completeness, we describe SRDR here and state its optimality in Lemma 2. Note that SRDR is indeed an online algorithm.

SRDR (Smallest Remaining Deteriorating Rate): At any time, schedule the job with the smallest remaining deteriorating rate, in other words, the current job is preempted when a job with a smaller deteriorating rate is released.

Lemma 2 ([18])**.** *Consider minimizing total generalized completion time of jobs with simple linear deteriorating rate and arbitrary release times. The algorithm* SRDR *returns an optimal preemptive schedule.*

3 General lower bound

In this section, we show a lower bound of $(1+b_{\max})^\alpha$ on the competitive ratio of any deterministic online algorithm.

Theorem 3. *Consider minimizing total generalized completion time of jobs with simple linear deteriorating rate and arbitrary release times. No deterministic online algorithm is better than $(1+b_{\max})^\alpha$-competitive.*

Proof. Let $\mathcal{A}$ be any online algorithm and $\mathcal{O}$ be an optimal offline algorithm. The adversary first releases at time t_0 a job J_1 with deteriorating rate b_1. Suppose $\mathcal{A}$ schedules J_1 at time t. We consider two cases depending on the value of t.

Case 1: $t \geq t_0(1+b_1)$. In this case, the adversary does not release more jobs. The completion time $c_1(\mathcal{A}) = t(1+b_1)$ and $c_1(\mathcal{O}) = t_0(1+b_1)$. Therefore,

$$\frac{\sum c_j^\alpha(\mathcal{A})}{\sum c_j^\alpha(\mathcal{O})} = \frac{t^\alpha}{t_0^\alpha} \geq (1+b_1)^\alpha = (1+b_{\max})^\alpha \ .$$

Case 2: $t < t_0(1+b_1)$. In this case, at $t+\beta$ for some small constant $\beta > 0$, the adversary releases $n-1$ jobs $J_2, \cdots, J_n$ with deteriorating rate ϵ. In this case, the completion time of J_j equals $t(1+b_1)(1+\epsilon)^{j-1}$. Therefore,

$$\sum c_j^\alpha(\mathcal{A}) = t^\alpha(1+b_1)^\alpha \sum_{1\leq j\leq n} (1+\epsilon)^{(j-1)\alpha} = t^\alpha(1+b_1)^\alpha \cdot \frac{(1+\epsilon)^{\alpha n}-1}{(1+\epsilon)^\alpha - 1} \ .$$

On the other hand, the optimal offline algorithm $\mathcal{O}$ schedules J_1 last and the completion time $c_j(\mathcal{O}) = (t+\beta)(1+\epsilon)^{j-1}$ for $2 \leq j \leq n$ and $c_1(\mathcal{O}) = (t+\beta)(1+\epsilon)^{n-1}(1+b_1)$. Therefore,

$$\begin{aligned}\sum c_j^\alpha(\mathcal{O}) &= (t+\beta)^\alpha \left(\sum_{2\leq j\leq n} (1+\epsilon)^{(j-1)\alpha} + (1+\epsilon)^{(n-1)\alpha}(1+b_1)^\alpha \right) \\ &\leq (t+\beta)^\alpha \left(\frac{(1+\epsilon)^{\alpha n}}{(1+\epsilon)^\alpha - 1} + (1+\epsilon)^{(n-1)\alpha}(1+b_1)^\alpha \right)\end{aligned}$$

In other words, we have

$$\frac{\sum c_j^\alpha(\mathcal{A})}{\sum c_j^\alpha(\mathcal{O})} \geq (1+b_1)^\alpha \left(\frac{t}{t+\beta}\right)^\alpha \frac{(1+\epsilon)^{\alpha n}-1}{(1+\epsilon)^{\alpha n} + ((1+\epsilon)^\alpha - 1)(1+\epsilon)^{(n-1)\alpha}(1+b_1)^\alpha}$$

If we choose an arbitrarily large n and a corresponding small ϵ, the last fraction approaches 1. We can further set β to be arbitrarily small and have $(\frac{t}{t+\beta})^\alpha$ arbitrarily close to 1. Therefore, $\frac{\sum c_j^\alpha(\mathcal{A})}{\sum c_j^\alpha(\mathcal{O})}$ approaches $(1+b_1)^\alpha = (1+b_{\max})^\alpha$. The theorem then follows. □

4 Optimal online algorithm DSDR

DSDR. As mentioned before, when all jobs are available at the beginning, the optimal algorithm is SDR, yet SDR is not competitive when jobs have arbitrary release time. We adapt the algorithm DSPT (Delayed Shortest Processing Time) in [9] and derive our algorithm DSDR (Delayed Smallest Deteriorating Rate). Consider an input job set $\mathcal{J}$. DSDR runs as follows.

Step 1: Consider time $t \geq t_0$ when the machine is idle. If an unscheduled job is available at that time, let J_j be the one with the smallest deteriorating rate. Ties are broken by taking the one with the earliest release time[5].

Step 2: If $t \geq t_0(1 + b_j)$, then schedule J_j at t; otherwise, wait until time $t_0(1 + b_j)$ or until a new job arrives, whichever happens first.

Step 3: If all jobs are scheduled, stop; otherwise, go to Step 1.

Without loss of generality, we assume that DSDR and SRDR break ties in the same way. To illustrate DSDR, we present the following example.

Example 1. *Suppose* $t_0 = 1$. *Consider two jobs* J_1 *and* J_2 *where* $r_1 = 1, b_1 = 2, r_2 = 2, b_2 = 1$. *At time* 1, DSDR *does not process* J_1 *immediately since* $t_0(1 + b_1) = 3 > t_0$. *It would process* J_2 *at time* 2 *since* $b_2 < b_1$ *and* $t_0(1 + b_2) = 2$. *The processing time of* J_2 *is* 2 *and it completes at time* 4, *when* DSDR *starts processing* J_1 *for* 8 *time units until time* 12.

Framework of analysis. Our analysis follows a similar framework as that in [9]. Consider a job set $\mathcal{J}$. Let $\mathcal{A}$ and $\mathcal{O}$ be the schedules by DSDR and the optimal algorithm, respectively. First of all, we observe that it suffices to consider instances such that DSDR schedules jobs contiguously without idle time. A similar observation was made for jobs with fixed processing time when the objective is the sum of total completion time [9] and the sum of total general completion time [13]. We observe that this observation applies to deteriorating jobs as well (Observation 4). We include the proof here for the sake of completeness.

We then define a pseudo-schedule $\mathcal{A}'$ from $\mathcal{A}$ and construct a new job set $\mathcal{J}'$ from $\mathcal{A}'$. Let $\mathcal{O}_{\mathrm{p}}$ and $\mathcal{O}'_{\mathrm{p}}$ be the optimal preemptive algorithm for $\mathcal{J}$ and $\mathcal{J}'$, respectively. Note that the subscript indicates we refer to the preemptive algorithm. It is relatively easy to relate the total general completion time of $\mathcal{O}$, $\mathcal{O}_{\mathrm{p}}$ and $\mathcal{O}'_{\mathrm{p}}$ (Lemma 6). The more tricky analysis is to relate the general completion time of $\mathcal{A}'$ and $\mathcal{O}'_{\mathrm{p}}$ and it is done via comparing the start time and completion time of jobs in $\mathcal{A}'$ and $\mathcal{O}'_{\mathrm{p}}$ (Corollary 10). Finally, from the definition of $\mathcal{A}'$ from $\mathcal{A}$, we can bound the total general completion time of $\mathcal{A}$ by that of $\mathcal{A}'$ and hence by that of $\mathcal{O}'_{\mathrm{p}}$, $\mathcal{O}_{\mathrm{p}}$ and $\mathcal{O}$ (Theorem 11).

We first show the following observation (extended from [9, 13]) that allows us to focus on DSDR schedules that consist of no idle time.

Observation 4. *The schedule* $\mathcal{A}$ *consists of a single block: it possibly starts with idle time after which all jobs are executed contiguously.*

Proof. Consider a job instance $\mathcal{J}$ and let $\mathcal{O}$ be the optimal schedule. Suppose that $\mathcal{A}$ contains some idle time. The jobs scheduled before this idle interval do not influence the scheduling decisions for the jobs scheduled after this idle interval, and vice versa. Let us split the instance into two independent smaller instances $\mathcal{J}_1$ and $\mathcal{J}_2$. Denote the cost of the two instances by $\mathcal{A}$ ($\mathcal{O}$ resp.) as $\mathcal{A}_1$ and $\mathcal{A}_2$ ($\mathcal{O}_1$ and $\mathcal{O}_2$ resp.). Note that $\mathcal{A} = \mathcal{A}_1 + \mathcal{A}_2$. If we restrict $\mathcal{O}$ to the jobs in $\mathcal{J}_1$ ($\mathcal{J}_2$ resp.), we get a feasible schedule for $\mathcal{J}_1$ ($\mathcal{J}_2$ resp.). Therefore, $\mathcal{O}_1 + \mathcal{O}_2 \leq \mathcal{O}$. Then we have $\frac{\mathcal{A}}{\mathcal{O}} \leq \frac{\mathcal{A}_1+\mathcal{A}_2}{\mathcal{O}_1+\mathcal{O}_2} \leq \max\{\frac{\mathcal{A}_1}{\mathcal{O}_1}, \frac{\mathcal{A}_2}{\mathcal{O}_2}\}$. Therefore, the competitive ratio of $\mathcal{A}$ on the smaller instances implies the competitive ratio on the original instance. □

[5]Further ties are broken by taking the one with smaller job ID.

4.1 Modified schedule $\mathcal{A}'$ and job set $\mathcal{J}'$

Consider any job set $\mathcal{J}$. From now on, we number the jobs in $\mathcal{J}$ as $J_1, J_2, \cdots, J_n$ according to the order that DSDR schedules them, i.e., $s_j(\mathcal{A}) \leq s_{j+1}(\mathcal{A})$ for $1 \leq j < n$. By Observation 4, there is no idle time in $\mathcal{A}$ and so we have the following property.

Property 5. *For* $1 \leq j < n$, $s_{j+1}(\mathcal{A}) = c_j(\mathcal{A}) = s_j(\mathcal{A})(1 + b_j)$.

We first define a modified schedule $\mathcal{A}'$ as follows. To simplify discussion, we define a dummy job J_0 with deteriorating rate b_0 such that $t_0(1 + b_0) = s_1(\mathcal{A})$. This job is not to be scheduled although we define $s_0(\mathcal{A}) = t_0(1 + b_0) = s_1(\mathcal{A})$.

We partition the schedule $\mathcal{A}$ into subblocks $V_1, V_2, \cdots, V_k$, such that each subblock is a maximal contiguous sequence of jobs ordered from smallest to largest deteriorating rate, i.e., the last job of a subblock (except the last subblock) has larger deteriorating rate than the first job in the next subblock. We also define a dummy subblock V_0 which contains J_0. This means that in every subblock, jobs are ordered according to the SDR rule. We denote by $v(i)$ the index of the last job in subblock V_i and we set $v(0) = 0$. Formally, $v(i)$ is defined as $v(i) = \min\{j > v(i-1) \mid b_j > b_{j+1}\}$. For the last subblock, $v(k)$ is defined as n. We further define $m(i)$ to be the largest index of the job having the largest deteriorating rate in the first i subblocks, i.e., $b_{m(i)} = \max_{0 \leq j \leq v(i)} b_j$ or $b_{m(i)} = \max_{0 \leq j \leq i} b_{v(j)}$.

We now define a *pseudo-schedule* $\mathcal{A}'$ based on schedule $\mathcal{A}$. The order of the start time of the jobs in $\mathcal{A}'$ is the same as in $\mathcal{A}$, yet for J_j in subblock V_i, the start time is moved forward to $\frac{s_j(\mathcal{A})}{1+b_{m(i-1)}}$. That is,

$$s_j(\mathcal{A}') = \frac{s_j(\mathcal{A})}{1 + b_{m(i-1)}} \qquad \text{and} \qquad c_j(\mathcal{A}') = \frac{c_j(\mathcal{A})}{1 + b_{m(i-1)}} \ . \tag{1}$$

Note that $\mathcal{A}'$ is not a genuine schedule for $\mathcal{J}$ since a job may start before its release time and the execution of jobs may overlap with each other. In $\mathcal{A}'$, Property 5 holds for jobs in the same subblock in $\mathcal{A}'$ because the denominator in Equation (1) is the same for jobs in the same subblock. Furthermore, there is no idle time in the schedule $\mathcal{A}'$.

Property 5a. *Consider subblock* V_i *with* $1 \leq i \leq k$. *For any* $v(i-1) + 1 \leq j < v(i)$, *we have* $s_{j+1}(\mathcal{A}') = c_j(\mathcal{A}') = s_j(\mathcal{A}')(1 + b_j)$. *Furthermore,* $s_{v(i-1)+1}(\mathcal{A}') \leq c_{v(i-1)}(\mathcal{A}')$ *and* $s_{v(i-1)+1}(\mathcal{A}') \leq s_{v(i-1)}(\mathcal{A})$.

We then construct $\mathcal{J}'$ based on the instance $\mathcal{J}$ and the pseudo-schedule $\mathcal{A}'$ for $\mathcal{J}$. For each job J_j, we define a corresponding job J'_j with the same deteriorating rate b'_j but the release time r'_j may be moved forward. Precisely,

$$b'_j = b_j \ , \qquad r'_j = \min\{r_j, s_j(\mathcal{A}')\} \qquad \text{and} \qquad \mathcal{J}' = \{J'_1, J'_2, \cdots, J'_n\} \ .$$

We denote by $\mathcal{O}_{\mathrm{p}}$ and $\mathcal{O}'_{\mathrm{p}}$ the optimal preemptive algorithm for $\mathcal{J}$ and $\mathcal{J}'$, respectively. It is easy to observe the following lemma.

Lemma 6. *Consider any* $1 \leq j \leq n$. (i) $\sum c_j^{\alpha}(\mathcal{O}'_{\mathrm{p}}) \leq \sum c_j^{\alpha}(\mathcal{O}_{\mathrm{p}})$. (ii) $\sum c_j^{\alpha}(\mathcal{O}_{\mathrm{p}}) \leq \sum c_j^{\alpha}(\mathcal{O})$.

Proof. (i) Since the release time of a job in $\mathcal{J}'$ is not later than that of the corresponding job in $\mathcal{J}$, any valid (preemptive) schedule for $\mathcal{J}$ is also a valid (preemptive) schedule for $\mathcal{J}'$. Therefore, the total general completion time of any preemptive schedule for $\mathcal{J}$ (including $\mathcal{O}_{\text{p}}$) is at least that of $\mathcal{O}'_{\text{p}}$ for $\mathcal{J}'$, i.e., $\sum c_j^{\alpha}(\mathcal{O}'_{\text{p}}) \leq \sum c_j^{\alpha}(\mathcal{O}_{\text{p}})$.

(ii) This is due to the fact that an optimal non-preemptive schedule is also a valid preemptive schedule and thus $\sum c_j^{\alpha}(\mathcal{O}_{\text{p}}) \leq \sum c_j^{\alpha}(\mathcal{O})$. □

4.2 Analysis

In this section, we mainly analyze the total general completion time of $\mathcal{A}'$ and $\mathcal{O}'_{\text{p}}$. Based on this, we can then bound the total general completion time of $\mathcal{A}$ in terms of $\mathcal{O}$. The main result is that the start time and completion time of a job J_j in $\mathcal{A}'$ is at most those of J'_j in $\mathcal{O}'_{\text{p}}$. We first observe that to bound the completion time in $\mathcal{O}'_{\text{p}}$, it suffices to bound the start time in $\mathcal{O}'_{\text{p}}$. For any $1 \leq j \leq n$, $c_j(\mathcal{A}') = s_j(\mathcal{A}')(1 + b_j)$ and $c_j(\mathcal{O}'_{\text{p}}) \geq s_j(\mathcal{O}'_{\text{p}})(1 + b_j)$. Therefore, $s_j(\mathcal{A}') \leq s_j(\mathcal{O}'_{\text{p}})$ implies that $c_j(\mathcal{A}') \leq c_j(\mathcal{O}'_{\text{p}})$.

Observation 7. *For any $1 \leq j \leq n$, if $s_j(\mathcal{A}') \leq s_j(\mathcal{O}'_{\text{p}})$, then $c_j(\mathcal{A}') \leq c_j(\mathcal{O}'_{\text{p}})$.*

It remains to show that for any job J_j, $s_j(\mathcal{A}') \leq s_j(\mathcal{O}'_{\text{p}})$. Roughly speaking, for any job J_j, we are going to find a set of sufficiently many jobs such that these jobs have higher priority than J_j according to $\mathcal{O}'_{\text{p}}$, are released after a certain time t and cannot be processed earlier than t, hence, implying a lower bound on the start time of J_j. More precisely, we show that it is possible to find a sequence of consecutive subblocks $V_{h^*+1}, \cdots, V_h$ such that all the jobs in these blocks have to be processed in $\mathcal{O}'_{\text{p}}$ at or after $s_{v(h^*)+1}(\mathcal{A}')$ and that all these jobs have higher priority than J_j according to $\mathcal{O}'_{\text{p}}$. Then J_j can only start in $\mathcal{O}'_{\text{p}}$ after all these jobs are completed, which is at least $s_{v(h^*)+1}(\mathcal{A}') \prod_{i=v(h^*)+1}^{j-1}(1 + b_i)$. We can then show that the start time property is satisfied by J_j. We formalize this by defining the notion "bounding subblock" of a job. Consider a job J_j in subblock V_h, for $1 \leq h \leq k$. We say that subblock V_{h^*} is a *bounding subblock* of J_j, for some $0 \leq h^* < h$, if the following properties hold for all jobs J_i from subblock V_{h^*+1} until J_j, i.e., for all $v(h^*) + 1 \leq i \leq j$:

1. $r'_i \geq s_{v(h^*)+1}(\mathcal{A}')$; and
2. EITHER $b_i < b_j$ OR $b_i = b_j$ and $r'_i \leq r'_j$.

Note that the two properties involve the release time of J'_i and J'_j in $\mathcal{J}'$, not the original release time in $\mathcal{J}$. The second property implies that if J'_j and J'_i are considered by $\mathcal{O}'_{\text{p}}$ at some time, J'_j would not preempt J'_i.[6]

We first show in Lemma 8 that if there exists a bounding subblock for J_j, then $s_j(\mathcal{A}') \leq s_j(\mathcal{O}'_{\text{p}})$. Then we show in Lemma 9 that we can find a bounding subblock for each job J_j.

Lemma 8. *If there exists a bounding subblock for the job J_j, then $s_j(\mathcal{A}') \leq s_j(\mathcal{O}'_{\text{p}})$.*

Proof. Suppose J_j is in subblock V_h and V_{h^*} is a bounding subblock of J_j, where $1 \leq h \leq k$ and $0 \leq h^* < h$. By Property 5, for any J_i with $v(h^*) + 1 \leq i \leq j$, we have $s_i(\mathcal{A}) = s_{v(h^*)+1}(\mathcal{A}) \prod_{\ell=v(h^*)+1}^{i-1}(1 + b_\ell)$. Since $m(\cdot)$ is increasing, we have $s_i(\mathcal{A}')$, and hence r'_i, is at

[6] $\mathcal{O}'_{\text{p}}$ may still start processing J'_j before J'_i if $b_i < b_j$ but $r'_i > r'_j$.

most $s_{v(h^*)+1}(\mathcal{A}')\prod_{\ell=v(h^*)+1}^{i-1}(1+b_\ell)$. In other words, J'_i is available for process at or before $s_{v(h^*)+1}(\mathcal{A}')\prod_{\ell=v(h^*)+1}^{i-1}(1+b_\ell)$.

By the first property of bounding subblock, all jobs $J'_{v(h^*)+1},\cdots,J'_j$ must be processed in $\mathcal{O}'_{\text{p}}$ at or after $s_{v(h^*)+1}(\mathcal{A}')$. We claim that $\mathcal{O}'_{\text{p}}$ starts processing J'_i before J'_j for all $v(h^*)+1 \le i < j$. By the second property of bounding subblock, J'_j has a lower priority than J'_i according to $\mathcal{O}'_{\text{p}}$ and would not preempt J'_i. The claim then implies $s_j(\mathcal{O}'_{\text{p}}) \ge s_{v(h^*)+1}(\mathcal{A}')\prod_{i=v(h^*)+1}^{j-1}(1+b_i) \ge s_j(\mathcal{A}')$; the latter inequality is due to Property 5a. Then the lemma follows.

We now proceed to proving the claim. To simplify the discussion, we let $t_1 = s_{v(h^*)+1}(\mathcal{A}')$, $t_2 = s_{v(h^*)+1}(\mathcal{A}')\prod_{i=v(h^*)+1}^{j-1}(1+b_i)$, and $\mathcal{J}'_j = \{J'_{v(h^*)+1},\cdots,J'_{j-1}\}$. Note that $\mathcal{J}'_j$ does not contain J'_j. Consider any time t where $t_1 \le t \le t_2$. We define the notion *density* at t to be

$$\text{den}(t) = \frac{t_1}{t}\left(\prod_{x:r'_x\le t,\ J'_x\in\mathcal{J}'_j}(1+b_x)\right) .$$

Intuitively, $\text{den}(t)$ indicates whether there is any job in $\mathcal{J}'_j$ that are available for process at t in any preemptive schedule. By Lemma 1, in any preemptive schedule, $t_1\prod_{x:r'_x\le t,\ J'_x\in\mathcal{J}'_j}(1+b_x)$ is the completion time if we run all these jobs contiguously from t_1. If this value is larger than t, it means that by the time t, we still have jobs available to process. In other words, at time t, if $\text{den}(t) > 1$, no matter how one schedule these jobs, there are at least some jobs still available to be processed and hence $\mathcal{O}'_{\text{p}}$ would not start J'_j because of its lower priority.

We now prove that at any time $t \in [t_1, t_2)$, $\text{den}(t) > 1$. As we have shown earlier, for any $J'_i \in \mathcal{J}'_j$, $r'_i \le t_1\prod_{\ell=v(h^*)+1}^{i-1}(1+b_\ell)$. For $i = v(h^*)+1$, this means that for any $t \in [t_1, t_1(1+b_{v(h^*)+1}))$, $t_1\prod_{x:r'_x\le t,\ J'_x\in\mathcal{J}'_j}(1+b_x) \ge t_1(1+b_{v(h^*)+1}) > t$, and hence $\text{den}(t) > 1$. In general, for any $t \in [t_1\prod_{\ell=v(h^*)+1}^{i-1}(1+b_\ell), t_1\prod_{\ell=v(h^*)+1}^{i}(1+b_\ell))$, we have $\prod_{x:r'_x\le t,\ J'_x\in\mathcal{J}'_j}(1+b_x) \ge t_1\prod_{\ell=v(h^*)+1}^{i}(1+b_\ell) > t$, implying $\text{den}(t) > 1$. Therefore, for any time $t \in [t_1, t_2)$, $\text{den}(t) > 1$ and the claim follows. □

Lemma 9. *For any $1 \le j \le n$, we can find a bounding subblock for J_j.*

Proof. Suppose J_j is in subblock V_h. We consider two cases.

Case 1: $b_j \ge b_\ell$ for all $1 \le \ell \le j$. In this case, we set $h^* = 0$ and hence, $s_{v(h^*)+1}(\mathcal{A}') = t_0$ and $r'_i \ge t_0$ for all i. The first property of bounding subblock thus holds. For the second property, we only need to consider when $b_i = b_j$ and $i < j$. Since DSDR schedules J_i before J_j, we have $r_i \le r_j$. Furthermore, by Property 5a, $s_j(\mathcal{A}) \ge s_i(\mathcal{A})(1+b_i)$ and hence $s_j(\mathcal{A}') \ge s_i(\mathcal{A})(1+b_i)/(1+b_{m(h-1)}) \ge s_i(\mathcal{A}) \ge s_i(\mathcal{A}')$. The second inequality is due to the fact that b_j, and hence b_i, is the largest deteriorating rate so far. Therefore, $r'_j \ge r'_i$.

Case 2: $b_j < b_\ell$ for some $1 \le \ell \le j$. In this case, there is at least a subblock V_{h^*} with $1 \le h^* \le h-1$ such that $b_{v(h^*)} > b_j$. We assume h^* is the largest such number, and let i be an integer such that $v(h^*) < i \le j$. In other words, $b_{v(h^*)} > b_i$. Furthermore, $b_j \ge b_i$, otherwise it violates the definition of h^* being the largest number with $b_{v(h^*)} > b_j$.

Consider the first property of bounding subblock. Since $\mathcal{A}$ schedules $J_{v(h^*)}$ before J_i, we have $r_i > s_{v(h^*)}(\mathcal{A})$. Note that $s_{v(h^*)+1}(\mathcal{A}') = s_{v(h^*)}(\mathcal{A})(1+b_{v(h^*)})/(1+b_{m(h^*)}) \le s_{v(h^*)}(\mathcal{A})$ because $b_{v(h^*)} \le b_{m(h^*)}$. Therefore, we have $r_i > s_{v(h^*)+1}(\mathcal{A}')$. Since $b_{v(h^*)} > b_i$ for all

$v(h^*) < i \leq j$, $m(x) = m(h^*)$ for all $h^* \leq x < h$. Then we have $s_i(\mathcal{A}') = \frac{s_i(\mathcal{A})}{1+b_{m(h^*)}} \geq \frac{s_{v(h^*)+1}(\mathcal{A})}{1+b_{m(h^*)}} = s_{v(h^*)+1}(\mathcal{A}')$. As a result, $r'_i = \min\{r_i, s_i(\mathcal{A}')\} \geq s_{v(h^*)+1}(\mathcal{A}') \geq r'_{v(h^*)+1}$. The first property is satisfied.

For the second property, we have observed that $b_i \leq b_j$ and thus we only need to consider the case when $b_i = b_j$ and $i < j$. Since DSDR schedules J_i before J_j, we have $r_i \leq r_j$. As observed in the proof of the first property, the start time of both J'_i and J'_j in $\mathcal{A}'$ is the corresponding value in $\mathcal{A}$ divided by $1 + b_{m(h^*)}$. Therefore, $s_i(\mathcal{A}') \leq s_j(\mathcal{A}')$. Together with $r_i \leq r_j$, we have $r'_i \leq r'_j$ and the property holds. □

With Lemmas 8 and 9 and Observation 7, we have the following corollary.

Corollary 10. *For any* $1 \leq j \leq n$, $s_j(\mathcal{A}') \leq s_j(\mathcal{O}'_{\mathrm{p}})$ *and* $c_j(\mathcal{A}') \leq c_j(\mathcal{O}'_{\mathrm{p}})$.

We can then prove the following theorem on the competitive ratio of DSDR.

Theorem 11. *The online algorithm* DSDR *is* $(1 + b_{\max})^{\alpha}$*-competitive.*

Proof. By Observation 4, we only need to consider instances where the DSDR schedule $\mathcal{A}$ has a single block without idle time. We first claim that $b_0 \leq b_{\max}$. By the definition of DSDR, the start time of the first job J_1 in $\mathcal{A}$ is $t_0(1 + b_1)$. The definition of the dummy job satisfies $t_0(1 + b_0) = s_1(\mathcal{A}) = t_0(1 + b_1)$, therefore, $b_0 = b_1 \leq b_{\max}$.

We then relate $\sum c_j^{\alpha}(\mathcal{A})$ and $\sum c_j^{\alpha}(\mathcal{O})$. Consider any job J_j in subblock V_h. By definition, $c_j(\mathcal{A}) = c_j(\mathcal{A}')(1 + b_{m(h-1)}) \leq c_j(\mathcal{A}')(1 + b_{\max}) \leq c_j(\mathcal{O}'_{\mathrm{p}})(1 + b_{\max})$. The last inequality is due to Corollary 10. Then by Lemma 6, we have

$$\sum_{1 \leq j \leq n} c_j^{\alpha}(\mathcal{A}) \leq (1 + b_{\max})^{\alpha} \sum_{1 \leq j \leq n} c_j^{\alpha}(\mathcal{O}'_{\mathrm{p}}) \leq (1 + b_{\max})^{\alpha} \sum_{1 \leq j \leq n} c_j^{\alpha}(\mathcal{O}) \ .$$

□

By Theorems 3 and 11, we conclude with the following corollary.

Corollary 12. *Algorithm* DSDR *achieves the best-possible competitive ratio of* $(1 + b_{\max})^{\alpha}$.

5 Summary and future work

In this paper, we study online single machine scheduling of jobs with simple linear deteriorating rate and arbitrary release times. The objective is to minimize the total general completion time. We show that the algorithm DSDR is an optimal online algorithm with competitive ratio $(1 + b_{\max})^{\alpha}$.

A future direction is to consider more general deterioration like $p_j = a_j + b_j s_j$, non-linear deterioration, or other time dependent functions [7], e.g., decrease in processing time as start time increase captures the learning effect. A related objective function is weighted completion time for which jobs with fixed processing time have been studied in [2], showing that the online algorithm DSWPT (Delayed Shortest Weighted Processing Time) is 2-competitive (the best possible). For jobs with simple linear deteriorating rates, the offline setting has been studied in [16]. Extension from [2, 16] to the online setting for deteriorating jobs would be of interest.

We study scheduling on a single machine. It is desirable to extend the study to multiple machines. Furthermore, online makespan scheduling of deteriorating jobs has been considered [5, 24]. It is interesting to consider other objective functions about waiting time, tardiness, lateness, deadline feasibility, throughput, etc.

6 Acknowledgments

This work is partially supported by NSF of China under Grants 71071123, 60736027 and 60921003.

References

[1] B. Alidaee and N. K. Womer. Scheduling with time dependent processing times: Review and extensions. *Journal of the Operational Research Society*, 50(7):711–720, 1999.

[2] E. J. Anderson and C. N. Potts. Online scheduling of a single machine to minimize total weighted completion time. *Mathematics of Operation Research*, 29(3):686–697, Aug. 2004.

[3] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, Cambridge, 1998.

[4] S. Browne and U. Yechiali. Scheduling deteriorating jobs on a single processor. *Operations Research*, 38(3):495–498, 1990.

[5] M. B. Cheng and S. J. Sun. A heuristic MBLS algorithm for the two semi-online parallel machine scheduling problems with deterioration jobs. *Journal of Shanghai University*, 11(5):451–456, 2007.

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

[7] S. Gawiejnowicz. *Time-Dependent Scheduling*. Springer-Verlag, Berlin, 2008.

[8] J. N. D. Gupta and S. K. Gupta. Single facility scheduling with nonlinear processing times. *Computers and Industrial Engineering*, 14(4):387–393, 1988.

[9] J. Hoogeveen and A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In W. Cunningham, S. McCormick, and M. Queyranne, editors, *Proceedings of the Fifth Conference on Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*, pages 404–414. Springer, 1996.

[10] A. Janiak, T. Krysiak, C. P. Pappis, and T. G. Voutsinas. A scheduling problem with job values given as a power function of their completion times. *European Journal of Operational Research*, 193(3):836–848, 2009.

[11] A. Kononov. Scheduling problems with linear increasing processing times. In e. a. Zimmermann U, editor, *Operations Research Proceedings 1996. Selected Papers of the Symposium on Operations Research (SOR 96)*, pages 208–212, Berlin, 1997. Springer.

[12] A. S. Kunnathur and S. K. Gupta. Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem. *European Journal of Operational Research*, 47(1):56–64, 1990.

[13] M. Liu, C. Chu, Y. Xu, and J. Huo. An optimal online algorithm for single machine scheduling to minimize total general completion time. *Journal of Combinatorial Optimization*, to appear. http://dx.doi.org/10.1007/s10878-010-9348-0.

[14] X. Lu, R. A. Sitters, and L. Stougie. A class of online scheduling algorithms to minimize total completion time. *Operations Research Letters*, 31:232–236, 2003.

[15] G. Mosheiov. V-shaped policies for scheduling deteriorating jobs. *Operations Research*, 39:979–991, November 1991.

[16] G. Mosheiov. Scheduling jobs under simple linear deterioration. *Computers and Operations Research*, 21(6):653–659, 1994.

[17] G. Mosheiov. Multi-machine scheduling with linear deterioration. *INFOR: Information Systems and Operational Research*, 36(4):205–214, 1998.

[18] C. T. Ng, S. S. Li, T. C. E. Cheng, and J. J. Yuan. Preemptive scheduling with simple linear deterioration on a single machine. *Theoretical Computer Science*, 411(40-42):3578–3586, 2010.

[19] C. Phillips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199–223, 1998.

[20] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Upper Saddle River, 2002.

[21] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In J. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, pages 15.1–15.42. Chapman and Hall, Boca Raton, 2004.

[22] C. R. Ren and L. Y. Kang. An approximation algorithm for parallel machine scheduling with simple linear deterioration. *Journal of Shanghai University*, 11(4):351–354, 2007.

[23] W. Townsend. The single machine problem with quadratic penalty function of completion times: A branch-and-bound solution. *Management Science*, 24(5):530–534, Jan. 1978.

[24] S. Yu, J.-T. Ojiaku, P. W. H. Wong, and Y. Xi. Online makespan scheduling of linear deteriorating jobs on parallel machines. Manuscript, 2011.

[25] S. Yu and P. W. H. Wong. A note on "An optimal online algorithm for single machine scheduling to minimize total general completion time". *Information Processing Letters*, 112(1-2):55–58, 2012.