Optimal on-line algorithms for single-machine scheduling

Memorandum COSOR 95-39, 1995, Eindhoven University of Technology

Optimal on-line algorithms for single-machine scheduling

J.A. Hoogeveen A.P.A. Vestjens

Department of Mathematics and Computing Science,
Eindhoven University of Technology,
P.O.Box 513, 5600 MB, Eindhoven, The Netherlands

Abstract

We consider single-machine on-line scheduling problems where jobs arrive over time. A set of independent jobs has to be scheduled on the machine, where preemption is not allowed and the number of jobs is unknown in advance. Each job becomes available at its release date, which is not known in advance, and its characteristics, e.g., processing requirement, become known at its arrival. We deal with two problems: minimizing total completion time and minimizing the maximum time by which all jobs have been delivered. For both problems we propose and analyze an on-line algorithm based on the following idea: As soon as the machine becomes available for processing, choose an available job with highest priority, and schedule it if its processing requirement is not too large. Otherwise, postpone the start of this job for a while. We prove that our algorithms have performance bound 2 and $(\sqrt{5}+1)/2$, respectively, and we show that for both problems there cannot exist an on-line algorithm with a better performance guarantee.

Keywords: on-line algorithms, single-machine scheduling, worst-case analysis.

1 Introduction

Until a few years ago, one of the basic assumptions made in deterministic scheduling was that all of the information needed to define the problem instance was known in advance. This assumption is usually not valid in practice, however. Abandoning it has led to the rapidly emerging field of on-line scheduling. Two on-line models have been proposed. The first one assumes that there are no release dates and that the jobs arrive in a list. The on-line algorithm has to schedule the first job in this list before it sees the next job in the list (e.g., see [Graham, 1966] and [Chen, Van Vliet & Woeginger, 1994]). The second model assumes that jobs arrive over time. Next to the presence of release dates, the main difference between the models is that in the second model jobs do not have to be scheduled immediately upon arrival. At each time that the machine is idle, the algorithm decides which one of the available jobs is scheduled, if any. In this paper we consider two single-machine on-line scheduling problems with release dates.

We deal with the single-machine scheduling problems of minimizing total completion time and maximum time by which all jobs have been delivered, respectively. In the latter problem, after their processing on the machine, the jobs need to be delivered, which takes a certain *delivery time*. The corresponding off-line problems are both strongly NP-hard, but the preemptive versions can be solved in polynomial time through an on-line algorithm (e.g., see [Lawler, Lenstra, Rinnooy Kan & Shmoys, 1993]). Well known on-line algorithms for the problems are the SPT-rule and LDT-rule: choose from among the available jobs the one with the shortest processing time and largest delivery time, respectively. If all release dates are equal, then the problems are solved by these algorithms. For the case that the release dates are not equal, Mao, Kincaid & Rifkin [1995] prove that SPT has a performance guarantee of n, where n is the number of jobs, and Kise, Ibaraki & Mine [1979] prove that LDT has a performance guarantee of 2. The question is of course: can we do better from a worst-case point of view?

Throughout the paper we use J_j to denote job j, and r_j, p_j, and q_j to denote the release date, processing requirement, and delivery time of J_j, respectively. We denote by $S_j(\sigma)$, $C_j(\sigma)$, and $L_j(\sigma)$, the starting time, completion time, and the time by which J_j is delivered in schedule σ. We use σ to denote the schedule produced by the heuristic and π to denote an optimal schedule.

This paper is organized as follows. In Section 2 we consider the problem of minimizing total completion time on a single-machine. We prove that any on-line algorithm for this problem has a worst-case ratio of at least 2, and we present an algorithm that achieves this bound. Independent of this work both Phillips, Stein & Wein [1995] and Stougie [1995] developed algorithms with equal performance guarantee; the lower bound of 2 was achieved by Stougie as well. We present both algorithms and compare them to our algorithm. In Section 3 we consider the problem of minimizing the time by which all jobs have been delivered. We show that any on-line algorithm has a worst-case ratio of at least $(\sqrt{5}+1)/2$. Moreover, we present an algorithm that achieves this bound.

2 Total completion time

In this section, we present an on-line 2-approximation algorithm for the single-machine scheduling problem of minimizing total completion time and show that no on-line algorithm can do better from a worst-case point of view. At the end of this section, we compare the algorithms of Phillips et al. and Stougie to our algorithm.

We first show that 2 is a lower bound on the worst-case ratio of any on-line algorithm; this result follows from an example. For a given schedule σ, we use $C(\sigma)$ as a short notation for the total completion time of σ, i.e., $C(\sigma) = \sum_j C_j(\sigma)$.

Theorem 2.1. *Any on-line algorithm has a worst-case ratio of at least 2.*

Proof. We show this result by describing a set of instances for which no on-line algorithm can guarantee an outcome strictly less than twice the optimum. Consider the following situation. The first job arrives at time 0 and has processing requirement p. The on-line algorithm decides to schedule the job at time S. Depending on S, either no jobs arrive anymore or $n-1$ jobs with processing requirement 0 arrive at time $S+1$. In the first case we get a ratio of $C(\sigma)/C(\pi) = (S+p)/p$, whereas in the second case we get a ratio of $C(\sigma)/C(\pi) \geq n(S+p)/(n(S+1)+p)$.

Hence,

$$\frac{C(\sigma)}{C(\pi)} \geq \max\left\{\frac{S+p}{p}, \frac{n(S+p)}{n(S+1)+p}\right\}.$$

The algorithm may choose S so as to minimize this expression. Some simple algebra shows that the best choice for S is

$$S = \frac{n-1}{n}p - 1.$$

This implies a worst-case ratio of

$$\frac{C(\sigma)}{C(\pi)} \geq 2 - \frac{1}{n} - \frac{1}{p}.$$

If we let both n and p tend to infinity, then we get the desired ratio of 2. □

We can use the example of Theorem 2.1 to show that any on-line algorithm that schedules a job as soon as the machine is available will have an unbounded worst-case ratio. If an algorithm wants to guarantee a better performance bound, then it needs a waiting strategy. For example, if an available job has a large processing requirement compared to the optimal solution of the currently available instance, the algorithm should wait for extra information. To incorporate this, we slightly modify the SPT-rule and call the new rule the delayed SPT-rule (D-SPT).

ALGORITHM D-SPT

If the machine is idle and a job is available at time t, determine an unscheduled job with smallest processing requirement, say J_i. If there is a choice, take the job with the smallest release date. If $p_i \leq t$, then schedule J_i; otherwise, wait until time p_i or until a new job arrives, whichever happens first.

As we have already seen, the worst-case bound of any algorithm is at least equal to 2. If the performance guarantee exceeds 2, then there exists an instance, which we call *counterexample*, for which the algorithm produces a schedule with value more than twice the optimal value. We show that our algorithm has a performance bound exactly equal to 2, by showing that there does not exist such a counterexample. Thereto, we first derive some characteristics of a *smallest counterexample*, i.e., a counterexample consisting of a minimum number of jobs. Let $\mathcal{I}$ be such a smallest counterexample, and let σ be the schedule created by D-SPT for this instance.

Observation 2.2. The schedule σ consists of a single block: it possibly starts with idle time after which all jobs are executed contiguously.

Proof. Suppose that σ contains idle time between the execution of the jobs. The jobs scheduled before this idle period do not influence the scheduling decision concerning the jobs scheduled after this idle period, and vice versa. Therefore, the instance can be split into two independent smaller instances. For at least one of these partial instances D-SPT creates a schedule with value more than twice the optimal value, which contradicts the assumption that we considered an instance with a minimum number of jobs. □

From now on, we assume that the jobs are numbered according to their position in the schedule σ. We partition σ into *subblocks*, such that within every subblock the jobs are ordered according to the SPT-rule, and that the last job of a subblock is larger than the first job of the succeeding subblock if it exists. We denote these subblocks by $B_1, \ldots, B_k$; subblock B_{i+1} consists of the jobs $J_{b(i)+1}, \ldots, J_{b(i+1)}$, where the indices $b(i)$ are determined recursively as $b(i) = \min\{j > b(i-1) \mid p_j > p_{j+1}\}$. The number of subblocks, k, in which the schedule is partitioned, follows from the recursion scheme.

For ease of exposition, we define a dummy job J_0 with $p_0 = S_1(\sigma)$, which will not be scheduled. Although J_0 will not be scheduled, we define $S_0(\sigma) = p_0$. Let $m(i)$ be the index of the job that has the largest processing requirement in the first i blocks, i.e., $p_{m(i)} = \max_{0 \le j \le b(i)} p_j$. We define a *pseudo-schedule* ψ for the schedule σ as follows. The order of the jobs in ψ is the same as in σ, but the first job in B_{i+1} starts at time $S_{b(i)+1}(\sigma) - p_{m(i)}$. Furthermore, all jobs in a block are scheduled contiguously. It is easy to verify that ψ is not a real schedule, since some jobs start before their release date and some jobs overlap. Note that ψ contains no idle time. Let ϕ be an optimal preemptive schedule for $\mathcal{I}$.

Lemma 2.3. *For all $J_j \in \mathcal{I}$, we have that $C_j(\sigma) - C_j(\psi) \le C_j(\phi)$.*

Proof. Consider an arbitrary job, say J_j, and suppose that $J_j \in B_{i+1}$. For this job $C_j(\sigma) - C_j(\psi) = p_{m(i)}$. If $p_j < p_{m(i)}$, then $r_j > S_{m(i)}(\sigma) \ge p_{m(i)}$, because D-SPT always schedules the smallest available job first and never starts a job before a time smaller than its own processing time. Therefore, either $p_j \ge p_{m(i)}$ or $r_j > p_{m(i)}$, which implies that $C_j(\phi) \ge r_j + p_j \ge p_{m(i)}$. Hence, $C_j(\sigma) - C_j(\psi) \le C_j(\phi)$. □

Lemma 2.4. $C(\psi) \le C(\phi)$.

Proof. Let I denote the job set corresponding to the smallest counterexample. Using this instance and the pseudo-schedule ψ for this instance we create a new instance $\mathcal{I}'$. The instance $\mathcal{I}'$ consists of all jobs in I. The processing requirements of the jobs remain the same, but the release dates r'_j are set equal to $\min\{r_j, S_j(\psi)\}$.

Let ϕ' be the optimal preemptive schedule for the instance $\mathcal{I}'$. Determine the first job in ϕ' that starts earlier in ϕ' than in ψ; suppose this job, say J_j, belongs to B_{i+1} in σ. If $p_j \ge p_{m(i)}$, then all jobs scheduled before J_j in ψ have a higher priority, i.e., either they have a smaller processing requirement or they have equal processing requirement and a smaller release date. This implies that in the preemptive schedule these jobs also have a higher priority and hence will be scheduled before J_j, which contradicts the fact that $S_j(\phi') < S_j(\psi)$. If $p_j < p_{m(i)}$, then all jobs that are executed in the interval $[r_j + p_{m(i)}, S_j(\sigma)]$ in σ have a higher priority than J_j. Hence, all jobs executed in the interval $[r_j, S_j(\psi)]$ in ψ have a higher priority than J_j; let V denote the set containing all these jobs. Since J_j is the first job in ϕ' with $S_j(\phi') < S_j(\psi)$, there is no room to start one of the jobs in V before time r_j. Hence, one of the jobs in V must be postponed in ϕ' to enable J_j to start before time $S_j(\psi)$, which is inconsistent with the way ϕ' has been constructed. Therefore, no job starts earlier in ϕ' than in ψ, which implies $C_j(\phi') \ge C_j(\psi)$ for all $j = 1, \ldots, n$.

As the release dates in $\mathcal{I}'$ are smaller than or equal to the release dates in $\mathcal{I}$, we have that $C(\phi) \geq C(\phi')$. Together this implies that $C(\psi) \leq C(\phi') \leq C(\phi)$. □

Theorem 2.5. $C(\sigma) \leq 2C(\phi)$.

Proof. Combining Lemmas 2.3 and 2.4 we obtain that $C(\sigma) \leq C(\phi) + C(\psi) \leq 2C(\phi)$. □

Corollary 2.6. *The on-line algorithm* D-SPT *has performance bound 2.* □

The algorithm ONE-MACHINE (1-M) developed by Phillips et al. uses the preemptive schedule. The algorithm maintains a list of jobs that have been completed in the preemptive schedule. As soon as a job has been finished in the preemptive schedule it will be appended to the end of this list. As soon as the machine becomes idle, the first job in this list will be assigned to the machine.

The algorithm developed by Stougie modifies the release dates of the jobs, before they are presented to the on-line algorithm. The release date of each job is increased by its own processing requirement. For this new instance the algorithm uses the SPT-rule. Since the algorithm first shifts the release dates and then uses SPT, we call it the shifted SPT-rule (S-SPT).

All three algorithms 1-M, S-SPT, and D-SPT create schedules with cost no more than twice the value of the optimal preemptive schedule. It is not to difficult to see that D-SPT does not create more idle time than S-SPT, which again does not create more idle time than 1-M. Hence, we might expect that on average D-SPT performs slightly better than the other two algorithms. There exist instances, however, for which one algorithm performs twice as well as the other ones. Table 1 shows these instances. The values of $C(\sigma)$ are the limiting values for $\varepsilon \downarrow 0$, and σ denotes only the order in which the jobs are scheduled.

Table 1: Instances to compare the worst-case behavior of D-SPT, S-SPT, and 1-M.

Instance						Algorithm	σ	$C(\sigma)$
j	1	2	$\cdots$	n		D-SPT	(2,…,n,1)	n+1
r_j	0	1	$\cdots$	1		S-SPT	(1,…,n)	2n
p_j	1	ε	$\cdots$	ε		1-M	(1,…,n)	2n
j	1	2	$\cdots$	n		D-SPT	(1,…,n)	2n
r_j	ε	$1+\varepsilon/2$	$\cdots$	$1+\varepsilon/2$		S-SPT	(2,…,n,1)	n+1
p_j	1	$\varepsilon/2$	$\cdots$	$\varepsilon/2$		1-M	(1,…,n)	2n
j	1	2	3	$\cdots$	n+1	D-SPT	(1,…,n+1)	2n
r_j	0	0	$1+\varepsilon/2$	$\cdots$	$1+\varepsilon/2$	S-SPT	(1,…,n+1)	2n
p_j	ε	1	0	$\cdots$	0	1-M	(1,3,…,n+1,2)	n+1

3 Maximum delivery time

In this section, we present an on-line α-approximation algorithm for the single-machine scheduling problem of minimizing the time by which all jobs have been delivered, where $\alpha = (\sqrt{5}+1)/2$, and show that no on-line algorithm can do better from a worst-case point of view. We start with the latter. Again, we prove the lower bound on the worst-case ratio by means of an example for which any on-line algorithm will have at least the required ratio. Let $L_{\max}(\pi)$ denote the minimum time by which all jobs can be delivered, and let $L_{\max}(\sigma)$ denote the time by which all jobs are delivered in schedule σ, where σ is the schedule obtained through some on-line algorithm.

Theorem 3.1. *Any on-line algorithm has a worst-case ratio of at least α.*

Proof. Consider the following situation. The first job arrives at time 0 and has processing requirement $p_1 = p$ and delivery time $q_1 = 0$. The on-line algorithm decides to schedule the job at time S. Depending on S, either no jobs arrive any more or one job with processing requirement $p_2 = 1$ and delivery time $q_2 = p$ arrives at time $r_2 = S + 1$. In the first case we get a ratio of $L_{\max}(\sigma)/L_{\max}(\pi) = (S + p)/p$; in the second case we get a ratio of $L_{\max}(\sigma)/L_{\max}(\pi) \geq (S+2p+1)/(S+p+2)$. Hence,

$$\frac{L_{\max}(\sigma)}{L_{\max}(\pi)} \geq \max\left\{\frac{S+p}{p}, \frac{S+2p+1}{S+p+2}\right\}.$$

The algorithm may choose S so as to minimize this expression. Some simple algebra shows that the best choice for S is

$$S = \frac{p}{2}(\sqrt{5-\frac{4}{p^2}}-1)-1.$$

This implies a worst-case ratio of

$$\frac{L_{\max}(\sigma)}{L_{\max}(\pi)} \geq \frac{1}{2}(\sqrt{5-\frac{4}{p^2}}+1)-\frac{1}{p}.$$

If we let p tend to infinity, then we get the desired ratio of α. □

We can use the example of Theorem 3.1 to show that any on-line algorithm that schedules a job as soon as the machine is available will have a worst-case ratio of at least 2. Note that a simple algorithm like LDT already achieves this bound. Again, if an algorithm wants to guarantee a better performance bound, then it needs a waiting strategy. Therefore, we modify the LDT-rule and call the new rule the delayed LDT-rule (D-LDT). The basic idea behind the algorithm is that, if no jobs with a large processing requirement are available, then we should schedule the job with the largest delivery time; otherwise, we should decide whether to schedule the large job, the job with the largest delivery time, or no job at all.

Throughout this section, we use the following notation:

- $p(S)$ denotes the total processing time of all jobs in S;
- $J(t)$ is the set containing all jobs that arrived at or before time t;
- $U(t)$ is the set containing all jobs in $J(t)$ that have not been started at time t;

- t_1 denotes the start time of the last idle time period before time t; if there is no idle time, then define $t_1 = 0$.
- We call a job J_j *big* if $p_j > (\alpha - 1)\, p(J(t) \setminus U(t_1))$. Note that $J(t) \setminus U(t_1)$ contains all jobs that arrived at or before time t and that were not completed at time t_1;
- $J_i(t)$ denotes the job with the largest processing time in $U(t)$.
- $J_m(t)$ denotes the job with the largest delivery time in $U(t)$.

ALGORITHM D-LDT

Wait until the machine is idle and a job is available. Suppose this happens at time t. If there is no big job available, then schedule $J_m(t)$. Otherwise, do the following.

- If $J_i(t)$ is the only available job, then wait until a new job arrives or until time $r_i + (\alpha - 1)\, p_i$, whichever happens first.
- Otherwise,

  if $t + p(U(t)) > r_i + \alpha p_i$, schedule $J_m(t)$ if $q_m > (\alpha - 1)\, p_i$ and $J_i(t)$, otherwise;

  else, if $J_m(t) \neq J_i(t)$, schedule $J_m(t)$, else schedule the job with the second largest delivery time.

Again we work with a smallest counterexample, where smallest refers to the number of jobs. Let $\mathcal{I}$ be such a smallest counterexample, and let σ be the schedule created by D-LDT for $\mathcal{I}$. We suppose that J_l denotes the first completed job in σ that assumes the value $L_{\max}(\sigma)$.

Observation 3.2. The schedule σ consists of a single block: it possibly starts with idle time after which all jobs are executed contiguously.

Proof. Suppose to the contrary that σ does not have this form. We will show that then either we can find a counterexample that consists of a smaller number of jobs, or that this alleged counterexample is not a counterexample at all.

Suppose that σ consists of more than one block. Suppose that block B is a block that contains a job J_l with $L_l(\sigma) = L_{\max}(\sigma)$; consider any block that precedes B. Since the algorithm bases its choices on the set $J(t) \setminus U(t_1)$, the existence of the jobs that are completed before the start of block B does not influence the start time of B and the order in which the jobs are executed. Therefore, we can remove all jobs that are completed before the start of block B without changing the value $L_{\max}(\sigma)$ and without increasing $L_{\max}(\pi)$. Similarly, we can remove all jobs from $\mathcal{I}$ that are released after the start of J_l in σ. Therefore, we may assume that our counterexample consists of the jobs from block B and the jobs that are available at the start of J_l in σ but that are scheduled in another block. Since the algorithm always starts a job if more than one job is available and the machine is empty, we know that there is at most one job that is available at time $S_l(\sigma)$ and does not belong to B; moreover, we know that this job, which we denote by J_i, must be marked big. Let $S(B)$ and $C(B)$ denote the start time of the first job and the completion time of the last job in B. Since J_i is big, $(\alpha - 1)\, p_i > p(B)$. Let J_1 be the first available job, which may be equal to J_i. Due to the operation of the algorithm, $S(B) = \min\{r_1 + (\alpha - 1)\, p_1, r_2\}$, where r_2

denotes the release date of the second available job. Since J_l is a job in B, we know that $L_{\max}(\sigma) = L_l(\sigma) = C_l(\sigma) + q_l \leq C(B) + q_l$. If J_1 is the first job in π, then $L_{\max}(\pi) \geq L_l(\pi) \geq r_1 + p_1 + p_l + q_l > S(B) + q_l$, from which we derive that $L_{\max}(\sigma) - L_{\max}(\pi) < C(B) - S(B) = p(B) < (\alpha - 1)\, p_i \leq (\alpha - 1) L_{\max}(\pi)$, which disproves the validity of our counterexample. If J_1 is not the first job in π, then the first job in π cannot start before time $r_2 \geq S(B)$, which implies that $L_{\max}(\pi) \geq L_l(\pi) \geq S(B) + p_l + q_l$, and we again have that $L_{\max}(\sigma) - L_{\max}(\pi) \leq C(B) - S(B)$, from which we deduce that $\mathcal{I}$ does not correspond to a counterexample. □

From now on, we let J_0 denote the job that arrives first in $\mathcal{I}$. Note that without loss of generality we may assume that $r_0 = 0$.

Observation 3.3. For all $J_j \in I \setminus \{J_0\}$, we have that $p_j \leq (\alpha - 1)\, p(\mathcal{I})$.

Proof. Suppose to the contrary that there does exist a job J_1 with $r_1 \geq r_0$ that has $p_1 > (\alpha - 1)\, p(\mathcal{I})$, i.e., $\alpha p_1 > p(\mathcal{I})$. Then at time r_1 there are at least two jobs available, which implies that the algorithm starts a job if it had not done so already. On basis of Observation 3.2, we may conclude that there is no idle time in the remainder of the schedule. But since J_1 is marked as big by the algorithm, this can only be the case if the other jobs are able to keep the machine busy from time r_1 to time $r_1 + (\alpha - 1)\, p_1$. In that case, however, $(\alpha - 1)\, p_1 \leq p(\mathcal{I}) - p_1 < \alpha p_1 - p_1 = (\alpha - 1)\, p_1$, which is a contradiction. □

We let J_k denote the last job in σ before J_l with a delivery time smaller than q_l, and we let $G(l)$ denote the set containing J_l and all jobs between J_k and J_l in σ. Note that all jobs in $G(l)$ have delivery time greater than or equal to q_l.

Observation 3.4. $p_k > (\alpha - 1)\, p(\mathcal{I})$.

Proof. If J_k does not exist, then

$$L_{\max}(\pi) \geq \sum_{j \in G(l)} p_j + q_l.$$

Since the first job in the block starts at time $(\alpha - 1)\, p_0$ at the latest,

$$L_{\max}(\sigma) = C_l(\sigma) + q_l \leq (\alpha - 1)\, p_0 + \sum_{j \in G(l)} p_j + q_l \leq (\alpha - 1)\, p_0 + L_{\max}(\pi) \leq \alpha L_{\max}(\pi),$$

which contradicts the fact that we consider a counterexample. Therefore, we assume from now on that such a job J_k exists. There are two possibilities for the algorithm to select J_k and not one of the jobs from $G(l)$:

(1) All jobs in $G(l)$ have a release date larger than $S_k(\sigma)$.

(2) There is one job from $G(l)$ available, which we denote by J_1, that is marked as big and cannot be started yet. Note that, since J_1 cannot be started yet, we must have that $S_k(\sigma) + p_k \leq r_1 + (\alpha - 1)\, p_1$.

For case (1), we have that

$$L_{\max}(\pi) \geq \min_{j \in G(l)} r_j + \sum_{j \in G(l)} p_j + q_l > S_k(\sigma) + \sum_{j \in G(l)} p_j + q_l,$$

and since $L_{\max}(\sigma) = C_l(\sigma) + q_l = S_k(\sigma) + p_k + \sum_{j \in G(l)} p_j + q_l$, we deduce that

$$L_{\max}(\sigma) - L_{\max}(\pi) < p_k \leq (\alpha - 1)\, p(\mathcal{I}) \leq (\alpha - 1) L_{\max}(\pi).$$

Concerning case (2), we have that

$$L_{\max}(\pi) \geq \min_{j \in G(l)} r_j + \sum_{j \in G(l)} p_j + q_l = r_1 + \sum_{j \in G(l)} p_j + q_l,$$

from which we deduce that

$$L_{\max}(\sigma) - L_{\max}(\pi) < S_k(\sigma) + p_k - r_1 \leq r_1 + (\alpha - 1)\, p_1 - r_1 \leq (\alpha - 1) L_{\max}(\pi).$$

Since neither of both cases corresponds to a counterexample, we conclude that J_k must be big. □

Corollary 3.5. $J_k = J_0$. □

For our analysis in Theorem 3.7, we need the following lemma.

Lemma 3.6. *Either J_0 is the first job in π, or $C_{\max}(\pi) \geq C_{\max}(\sigma) \geq \alpha p_0$.*

Proof. Let J_1 be the first job other than J_0 that becomes available. As there are two jobs available at time r_1, the algorithm starts one of the jobs if the machine is still idle. Therefore, the first job in σ starts no later than the first job in π, and since there is no idle time in σ, we have $C_{\max}(\sigma) \leq C_{\max}(\pi)$. It is easily checked that $C_{\max}(\sigma) \geq \alpha p_0$. □

Theorem 3.7. *The on-line algorithm* D-LDT *has performance bound α.*

Proof. Suppose to the contrary that there exists an instance for which the algorithm finds a schedule σ with $L_{\max}(\sigma) > \alpha L_{\max}(\pi)$. Obviously, then there exists a counterexample $\mathcal{I}$ with a minimum number of jobs. On basis of Observations 3.2 through 3.4, we may assume that the first job available in $\mathcal{I}$, which is defined to be J_0, has $p_0 > (\alpha - 1)\, p(\mathcal{I})$. Note that, due to Corollary 3.5, J_0 is the last job before J_l in σ with a delivery time smaller than q_l. J_0 starts no later than at time $(\alpha - 1)\, p_0$ unless some job with delivery time greater than $(\alpha - 1)\, p_0$ is available. Let $G(h)$ denote the set of jobs that were selected instead of J_0 when J_0 was eligible for being scheduled; $G(h)$ may be empty. Let $S_h(\sigma)$ denote the start time of the first job in this set if available; $S_h(\sigma) \leq (\alpha - 1)\, p_0$. Note that, if $S_0(\sigma) > (\alpha - 1)\, p_0$, then $G(h) \neq \emptyset$.

The proof proceeds by a case-by-case analysis. There are two reasons possible for starting J_0 at time $S_0(\sigma)$ instead of a job from $G(l)$. The first one is that simply none of the jobs in $G(l)$ were available, i.e., $r_j > S_0(\sigma)$ for all $J_l \in G(l)$. The second one is that the available jobs in $G(l)$ all have a delivery time at most equal to $(\alpha - 1)\, p_0$. We cover both cases by distinguishing between

(1) $r_j > S_0(\sigma)$ for all $J_j \in G(l)$, and

(2) $q_j \leq (\alpha - 1)\, p_0$ for some $J_j \in G(l)$.

Case 1. Since none of the jobs in $G(l)$ is available at time $S_0(\sigma)$,

$$L_{\max}(\pi) > S_0(\sigma) + \textstyle\sum_{j \in G(l)} p_j + q_l, \quad \text{and}$$
$$L_{\max}(\sigma) = S_0(\sigma) + p_0 + \textstyle\sum_{j \in G(l)} p_j + q_l.$$

Hence, $L_{\max}(\sigma) - L_{\max}(\pi) < p_0$. If J_0 is not the first job in π, then according to Lemma 3.6 $L_{\max}(\pi) \geq C_{\max}(\pi) \geq \alpha p_0$, which implies that $L_{\max}(\sigma) - L_{\max}(\pi) < (\alpha - 1) L_{\max}(\pi)$. Therefore, we assume that J_0 is the first job in π. Then

$$L_{\max}(\pi) \geq p_0 + \sum_{j \in G(l)} p_j + q_l,$$

and hence, $L_{\max}(\sigma) - L_{\max}(\pi) \leq S_0(\sigma)$. Now, either $S_0(\sigma) \leq (\alpha - 1) p_0$, which disqualifies the counterexample, or $G(h) \neq \emptyset$. Note that all jobs in $G(h)$ have a delivery time greater than $(\alpha - 1) p_0$. Since J_0 is the first job in π, $L_{\max}(\pi) > \alpha p_0$, and we do not have a counterexample.

Case 2. Since all jobs in $G(l)$ have a delivery time that is at least as large as q_l, we have that $q_l \leq (\alpha - 1) p_0$. If J_0 is not the first job in π, then according to Lemma 3.6, $C_{\max}(\sigma) \leq C_{\max}(\pi)$, and we get

$$L_{\max}(\sigma) = C_l(\sigma) + q_l \leq C_{\max}(\sigma) + q_l \leq C_{\max}(\pi) + q_l \leq L_{\max}(\pi) + q_l \leq$$
$$L_{\max}(\pi) + (\alpha - 1) p_0 \leq \alpha L_{\max}(\pi).$$

Therefore, we assume that J_0 is the first job in π. Since all jobs in $G(h)$ have a delivery time greater than $(\alpha - 1) p_0$, J_l is the job with the smallest delivery time in $G(h) \cup G(l)$. Combining all this yields

$$L_{\max}(\pi) \geq p_0 + \textstyle\sum_{j \in G(h) \cup G(l)} p_j + q_l, \quad \text{and}$$
$$L_{\max}(\sigma) = S_h(\sigma) + p_0 + \textstyle\sum_{j \in G(h) \cup G(l)} p_j + q_l,$$

which implies that $L_{\max}(\sigma) - L_{\max}(\pi) \leq S_h(\sigma)$, and we are done since $S_h(\sigma) \leq (\alpha - 1) p_0$.

Since we have checked all possibilities, we conclude that there is no counterexample to Theorem 3.7. $\square$

References

CHEN, B., A. VAN VLIET, AND G.J. WOEGINGER [1994], New lower and upper bounds for on-line scheduling, *Operations Research Letters* **16**, 221–230.

GRAHAM, R.L. [1966], Bounds for certain multiprocessing anomalies, *Bell System Technical Journal* **45**, 1563–1581.

KISE, H., T. IBARAKI, AND H. MINE [1979], Performance analysis of six approximation algorithms for the one-machine maximum lateness scheduling problem with ready times, *Journal of the Operations Research Society of Japan* **22**, 205–224.

LAWLER, E. L., J.K. LENSTRA, A.H.G. RINNOOY KAN, AND D.B. SHMOYS [1993], Sequencing and scheduling: Algorithms and complexity, in: S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin (eds.), *Logistics of Production and Inventory*, Handbooks in OR & MS 4, Elsevier Science Publishers B.V., Amsterdam, Chapter 9, 445–522, ISBN 0-444-87472-0.

MAO, W., R.K. KINCAID, AND A. RIFKIN [1995], On-line algorithms for a single machine scheduling problem, in: S.G. Nash and A. Sofer (eds.), *The impact of emerging technologies on computer science and operations research*, Kluwer Academic Press, Chapter 8, 157–173.

PHILLIPS, C., C. STEIN, AND J. WEIN [1995], Scheduling jobs that arrive over time, *Proceedings of the Fourth Workshop on Algorithms and Data Structures*, Lecture Notes in Computer Science 955, Springer.

STOUGIE, L. [1995], personal communication.