ELSEVIER

Operations Research Letters 27 (2000) 193–197

operations research letters

www.elsevier.com/locate/dsw

On-line scheduling on a single machine: maximizing the number of early jobs

Han Hoogeveen[a], Chris N. Potts[b], Gerhard J. Woeginger[c, *, 1]

[a] *Institute of Information and Computing Sciences, Utrecht University, 3584 CH Utrecht, The Netherlands*
[b] *Faculty of Mathematical Studies, University of Southampton, Southampton SO17 1BJ, UK*
[c] *TU Graz, Institut für Mathematik B, Steyrergasse 30, A-8010 Graz, Austria*

Received 11 November 1999

Abstract

This note deals with the scheduling problem of maximizing the number of early jobs on a single machine. We investigate the on-line version of this problem in the Preemption-Restart model. This means that jobs may be preempted, but preempting results in all the work done on this job so far being lost. Thus, if the job is restarted, then it has to be done from scratch.

We prove that the shortest remaining processing time (SRPT) rule yields an on-line algorithm with competitive ratio $\frac{1}{2}$. Moreover, we show that there does not exist an on-line algorithm with a better performance guarantee.

Keywords: Combinatorial problems; On-line algorithms; Competitive analysis; Worst-case bounds; Single-machine scheduling

1. Introduction

An instance of the scheduling problem of maximizing the number of early jobs on a single machine consists of a sequence $\mathcal{J}$ of jobs J_j for $j = 1, \ldots, n$. Every job J_j is specified by its processing time p_j, its release date r_j, and its due date d_j. A job can only be processed after it has been released, and the machine can process at most one job at a time. If in some schedule a job is completed before or at its due date, then it is called *early* (or *on time*); otherwise, the job is *late*. Our goal is to find a schedule that maximizes the number of early jobs.

In the *off-line* version of this problem, all the job data are known a priori. This version is well-understood. If preemption is allowed, then the problem can be solved in polynomial time [4,1]. Alternatively, if preemption is not allowed, then the problem is NP-hard to solve to optimality [5,6]. In this note, however, we are mainly interested in the *on-line* version of this problem for which the job data are not known a priori: the jobs arrive over time, and the decision maker does not have any knowledge on the future of the system. Thus, a job J_j only becomes

* Corresponding author.
E-mail addresses: slam@cs.uu.nl (H. Hoogeveen), c.n.potts@maths.soton.ac.uk (C.N. Potts), gwoegi@opt.math.tu-graz.ac.at (G.J. Woeginger).

[1] Supported by the START program Y43-MAT of the Austrian Ministry of Science.

0167-6377/00/$ - see front matter
PII: S0167-6377(00)00061-4

known at its release date r_j. At this moment, the decision maker also discovers the processing time p_j and the due date d_j of this job. At any time that the machine is idle, the decision maker specifies which of the available jobs should be processed or whether the machine should be better kept idle. There are three standard models for on-line scheduling where jobs arrive over time.

1. *The non-preemptive model.* Once a job is started on the machine, it must be run to completion. In this model, the finally constructed schedule is non-preemptive.
2. *The preemption-resume model.* The currently processed job may be preempted at any moment in time, and it may be resumed at any later moment in time. A job J_j is completed, as soon as it has been processed for a total of p_j time units on the machine. In this model, the finally constructed schedule is in general preemptive.
3. *The preemption-restart model.* The currently processed job may be preempted at any moment in time. However, by preempting a job, all the progress that has been made on this job so far is lost. Thus, if the job is restarted at some later moment in time, then it has to be done from scratch. In this model, the finally constructed schedule is non-preemptive.

The quality of an on-line algorithm A is usually measured by its *competitive ratio* (or *worst-case ratio*) R_A. In maximization problems, this competitive ratio is defined by

$$R_A = \inf\{A(\mathcal{J})/\mathrm{Opt}(\mathcal{J}) \mid \mathcal{J} \text{ is a sequence of jobs, } \mathrm{Opt}(\mathcal{J}) > 0\}. \quad (1)$$

Here, $A(\mathcal{J})$ denotes the number of early jobs in the schedule constructed by the on-line algorithm A for the job sequence $\mathcal{J}$, whereas $\mathrm{Opt}(\mathcal{J})$ denotes the number of early jobs in an optimal off-line schedule for $\mathcal{J}$. In the non-preemptive model and in the preemption-restart model we take an optimal non-preemptive schedule when computing $\mathrm{Opt}(\mathcal{J})$, and in the preemption-resume model we take the optimal preemptive schedule when computing $\mathrm{Opt}(\mathcal{J})$. Clearly, $0 \leqslant R_A \leqslant 1$ holds. The closer the ratio R_A comes to one, the better is the quality of the on-line algorithm A.

Note that it is more conventional in scheduling to minimize the number of tardy jobs, rather than maximize the number of early jobs. However, the number of tardy jobs in an off-line schedule may be zero, which destroys the possibility of finding a finite competitive ratio. Consequently, it is usual in competitive analysis to adopt the goal of maximizing the number of early jobs.

Let us briefly discuss our scheduling problem in the context of these three on-line models. First, consider the non-preemptive model. Suppose that, at time 0, a job J_1 with $p_1 = 1$ and $d_1 = 1$ is released. How should a reasonable on-line algorithm A behave? If it does not start to process J_1 at time 0, then no further job may arrive. In this case, $A(\mathcal{J}) = 0$, $\mathrm{Opt}(\mathcal{J}) = 1$, and $R_A = 0$. If, on the other hand, algorithm A decides to process the job starting at time 0, then an instant later a huge number of jobs may arrive, all with tiny processing times and all with due dates equal to 1. Then, $\mathrm{Opt}(\mathcal{J})$ can become arbitrarily large, whereas $A(\mathcal{J}) = 1$. Again, $R_A = 0$. Hence, in the non-preemptive model, any on-line algorithm A yields $R_A = 0$ and thus must behave very poorly in the worst case; a hopeless situation! Next, consider the Preemption-Resume model. Baruah et al. [2] prove that, in this model also, any on-line algorithm A yields $R_A = 0$. The only known positive result is due to Kalyanasundaram and Pruhs [3], who design a randomized on-line algorithm A whose (very very small) competitive ratio R_{A} is strictly bounded away from 0. To the best of our knowledge, no results are known so far for the preemption-restart model.

In this note, we study the competitive ratio of on-line algorithms for maximizing the number of early jobs on a single machine in the preemption-restart model. Sgall [7] gives a comprehensive survey on on-line scheduling. Most of the results derived so far in this area seem to be in the non-preemptive and in the preemption-resume model. However, Shmoys et al. [7] introduce the preemption-restart model and present several results for scheduling m parallel machines to minimize the makespan. Specifically, they present a $(2 - 1/m)$-competitive for the case identical parallel machines, and show that this bound is the best possible. They also show that for uniform parallel machines the best competitive ratio is $\Theta(\log m)$, and for unrelated parallel machines the best competitive ratio lies between the lower bound of $\Omega(\log m)$ and the upper bound of $\mathrm{O}(\log n)$. In his Ph.D.

thesis, Vestjens [9] derives several negative results for the preemption-restart model. For minimizing the maximum job delivery time on a single machine in the preemption-restart model, Vestjens shows that the competitive ratio of an on-line algorithm cannot be better than $\frac{3}{2}$. Van den Akker et al. [8] provide a matching positive result for this problem. We are not aware of any other literature dealing with on-line scheduling for preemption-restart model.

Our contribution is to design an on-line algorithm with competitive ratio $\frac{1}{2}$ for maximizing the number of early jobs on a single machine in the preemption-restart model. The algorithm and its analysis are contained in Section 2, while Section 3 demonstrates using an adversary argument that there cannot exist an on-line algorithm whose competitive ratio is strictly better than $\frac{1}{2}$.

2. Proof of the positive result

In this section, we describe and analyze algorithm SRPT, which is based on the *shortest remaining processing time* (SRPT) rule. Algorithm SRPT constructs a schedule of early jobs only, since any tardy jobs can be appended to this schedule in an arbitrary order. At any time t, let $\bar{p}_j$ denote the remaining processing time of a job J_j with $r_j \leqslant t$. From this definition, $\bar{p}_j = p_j$ if the machine does not process J_j immediately before time t. On the other hand, if the machine processes job J_j throughout some interval $[s,t]$, but J_j is not processed immediately before time s, then $\bar{p}_j = p_j - (t-s)$. Algorithm SRPT makes a decision about which job to process next whenever a new job is released, and whenever the machine completes the processing of a job. At any decision point t, a job J_j for which

$$\bar{p}_j = \min\{\bar{p}_k \mid J_k \in \mathscr{J},\ r_k \leqslant t,\ \bar{p}_k > 0,\ t + \bar{p}_k \leqslant d_k\} \tag{2}$$

is selected to be processed next. No further early jobs are scheduled by Algorithm SRPT if, at some decision point t, no jobs are subsequently released and $t + \bar{p}_k > d_k$ for all jobs J_k with $r_k \leqslant t$ and $\bar{p}_k > 0$. Note that Algorithm SRPT also implements an earliest completion time (ECT) policy: when SRPT makes a decision to process a job J_j that has the shortest remaining processing time among available jobs, then if it is not preempted, J_j completes no later than if any other available job were to be processed next. Thus, a job is preempted only if a newly released job can complete earlier.

In a schedule of early jobs generated by Algorithm SRPT, each job starts processing either at its release date or when the machine completes processing a previous early job. Thus, a schedule obtained from Algorithm SRPT can be represented by a sequence that defines the order in which the early jobs are completed. By scheduling each job of the sequence to start processing as early as possible, and not allowing preemption, we obtain a corresponding schedule of early jobs. Similarly, an optimal schedule of early jobs can also be represented by the sequence in which these jobs are completed. In the following, a sequence defining a schedule of early jobs is called a *feasible* sequence.

We now present the main result in this section.

Theorem 1. *Algorithm SRPT has a competitive ratio* $R_{\mathrm{SRPT}} \geqslant \frac{1}{2}$.

Proof. Let e^* be the number of early jobs in an optimal schedule, and let sequence $\sigma^* = (J_{\sigma^*(1)}, \ldots, J_{\sigma^*(e^*)})$ define some optimal schedule. Similarly, let e be the number of early jobs in the schedule generated by Algorithm SRPT, and let sequence $\sigma = (J_{\sigma(1)}, \ldots, J_{\sigma(e)})$ define this schedule. To establish the inequality $R_{\mathrm{SRPT}} \geqslant \frac{1}{2}$, we prove that $e \geqslant e^*/2$. Suppose for the sake of contradiction that the inequality $e < e^*/2$ holds. Our aim is to use the sequences σ and σ^* to construct feasible sequences of early jobs, which we denote by $\pi^0, \ldots, \pi^e$. Each feasible sequence π^j for $j = 0, \ldots, e$ satisfies the following two conditions:

(i) π^j contains at least $e^* - j$ jobs;
(ii) the first j jobs in π^j are given by $\pi^j(i) = \sigma(i)$ for $i = 1, \ldots, j$.

From these conditions, sequence π^e contains at least $e^* - e$ early jobs, where $e^* - e > e$, and has σ as a subsequence. Thus, job $J_{\pi^e(e+1)}$ is early if it is scheduled after the last job of σ, which contradicts the termination rule of Algorithm SRPT. Therefore, to establish the contradiction and to prove the theorem, we only need to specify how to construct sequences $\pi^1, \ldots, \pi^e$ with the desired properties. This is done using an inductive procedure that we describe next.

To start our inductive construction, we set $\pi^0 = \sigma^*$. To construct π^j from π^{j-1} for $j = 1, \ldots, e$, we proceed as follows. First note that by condition (i) sequence π^{j-1} contains at least $e^* - j + 1 > 2e - j + 1 \geqslant j$ jobs. If $\pi^{j-1}(j) = \sigma(j)$, then we set $\pi^j = \pi^{j-1}$. The feasibility of π^j follows from the feasibility of π^{j-1}, and π^j obviously satisfies conditions (i) and (ii). Alternatively, if $\pi^{j-1}(j) \neq \sigma(j)$, then we obtain π^j from π^{j-1} by first removing job $J_{\sigma(j)}$ if it is present in π^{j-1}, and then removing job $J_{\pi^{j-1}(j)}$ and replacing it with job $J_{\sigma(j)}$. Removing job $J_{\sigma(j)}$ does not affect feasibility. Moreover, from the ECT policy of the SRPT rule, job $J_{\sigma(j)}$, which replaces job $J_{\pi^{j-1}(j)}$, completes no later in π^j than job $J_{\pi^{j-1}(j)}$ completes in π^{j-1}. Thus, the feasibility of sequence π^j follows from that of π^{j-1}. From the construction, sequence π^j contains either the same number of jobs as sequence π^{j-1}, or one less job. Hence, condition (i) for π^{j-1} implies condition (i) for π^j. Condition (ii) for π^j also follows from the construction. □

In the next section, we show that $R_{\mathrm{SRPT}} = \frac{1}{2}$ by providing an instance which shows that no on-line algorithm can have a competitive ratio exceeding $\frac{1}{2}$.

3. Proof of the negative result

In this section, we describe adversary strategies against which any on-line algorithm must perform poorly. Our main tool is the following lemma.

Lemma 2. *For every integer $k \geqslant 0$ and for all real numbers r and d with $r < d$, there exists an adversary strategy $\mathscr{S} = \mathscr{S}(k, r, d)$ with the following properties.*

(i) *$\mathscr{S}$ creates $2k + 1$ jobs. The earliest release date of these jobs is r, and the latest due date of these jobs is d.*

(ii) *There exists a feasible schedule in which all $2k + 1$ jobs are early. In such a schedule, the machine is continuously busy throughout the interval $[r, d]$.*

(iii) *The adversary strategy $\mathscr{S}$ can prevent any on-line algorithm from scheduling more than $k + 1$ jobs to be early.*

Proof. The proof is by induction on k. For $k = 0$, the adversary simply releases a job with processing time $d - r$ at time r. For $k \geqslant 1$, the adversary $\mathscr{S}$ proceeds as follows. Let $L = (d - r)/8$, and note that $d = r + 8L$. The adversary releases two jobs J_1 and J_2 with processing times $p_1 = 3L$ and $p_2 = 4L$ at time r. Both jobs have due date d. Then the adversary waits until time $r + 2L$. □

Case 1. If at time $r + 2L$ the on-line algorithm is processing job J_1, then $\mathscr{S}$ calls the sub-adversary $\mathscr{S}(k - 1, r + 4L, r + 5L)$.

Case 2. Otherwise, $\mathscr{S}$ calls the sub-adversary $\mathscr{S}(k - 1, r + 3L, r + 4L)$.

Property (i) holds, since $\mathscr{S}$ creates the two new jobs J_1 and J_2 together with the $2k - 1$ jobs of the sub-adversary. To prove property (ii), consider the following schedules with all jobs early. In case 1, process job J_1 during the interval $[r + 5L, d]$ and job J_2 during the interval $[r, r + 4L]$, and process all jobs of the sub-adversary by induction. Similarly, in case 2, process J_1 during $[r, r + 3L]$, J_2 during $[r + 4L, d]$, and all jobs of the sub-adversary by induction.

It remains to establish property (iii). First, assume that the on-line algorithm schedules at most one of the jobs J_1 and J_2 to be early. By induction, the on-line algorithm can schedule at most k of the jobs that are created by the sub-adversary to be early. This shows that at most $k + 1$ jobs are early. Next, assume that the on-line algorithm schedules both jobs J_1 and J_2 to be early. Then, in case 1, the on-line algorithm must process job J_2 entirely during the time interval $[r + 2L, d]$. But then the processing of J_2 covers all of $[r + 4L, r + 5L]$, and consequently none of the sub-adversary's jobs can be early. This yields at most 2 early jobs. Finally, in case 2, the on-line algorithm must process all of J_1 *after* time $r + 2L$. Hence, at least L units of J_2 must be processed *before* time $r + 2L$. But if J_2 is started between time r and time $r + 2L$, then its processing covers all of $[r + 3L, r + 4L]$. Again, none of the sub-adversary's jobs can be early. This again yields at most 2 early jobs. □

Theorem 3. *Every on-line algorithm A for maximizing the number of early jobs in the Preemption-Restart model has competitive ratio $R_A \leqslant \frac{1}{2}$.*

Acknowledgements

The authors are grateful to an anonymous referee for suggesting a neater way to present the proof of

Theorem 1. We would also like to thank the organizers of the Dagstuhl Seminar 99431 on 'Scheduling in Computer and Manufacturing Systems' during which the results presented in this paper were established.

References

[1] P. Baptiste, An O(n^4) algorithm for preemptive scheduling of a single machine to minimize the number of late jobs, Oper. Res. Lett. 24 (1999) 175–180.
[2] S. Baruah, J. Haritsa, N. Sharma, On-line scheduling to maximize task completions, Proceedings of the 15th IEEE Real-time Systems Symposium (RSS'96), 1994, pp. 228–237.
[3] B. Kalyanasundaram, K. Pruhs, Maximizing job completions on-line, Proceedings of the Sixth European Symposium on Algorithms (ESA'98), Springer Lecture Notes in Computer Science, vol. 1461, 1998, pp. 235–246.
[4] E.L. Lawler, A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs, Ann. Oper. Res. 26 (1990) 125–133.
[5] J.K. Lenstra, A.H.G. Rinnooy Kan, P. Brucker, Complexity of machine scheduling problems, Ann. Oper. Res. 1 (1977) 343–362.
[6] J. Sgall, On-line scheduling, in: A. Fiat, G.J. Woeginger (Eds.), On-line Algorithms: The State of the Art, Lecture Notes in Computer Science, vol. 1442, Springer, Berlin, 1998,, pp. 196–231.
[7] D.B. Shmoys, J. Wein, D.P. Williamson, Scheduling parallel machines on-line, SIAM J. Comput. 24 (1995) 1313–1331.
[8] J.M. van den Akker, J.A. Hoogeveen, N. Vakhania, Restarts can help in the on-line minimization of the maximum delivery time on a single machine, Proceedings of the Eighth European Symposium on Algorithms (ESA'2000), Lecture Notes in Computer Science, vol. 1879, Springer, Berlin, 2000.
[9] A.P.A. Vestjens, On-line machine scheduling, Ph.D. Thesis, Eindhoven University of Technology, 1997.