ELSEVIER

Operations Research Letters 21 (1997) 165–169

operations research letters

Scheduling on identical machines: How good is LPT in an on-line setting?[1]

Bo Chen[a], Arjen P.A. Vestjens[b,*]

[a] *Warwick Business School, University of Warwick, Coventry CV4 7AL, UK*
[b] *Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, Netherlands*

Received 1 April 1996; revised 1 April 1997

Abstract

We consider a parallel machine scheduling problem where jobs arrive over time. A set of independent jobs has to be scheduled on m identical machines, 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 processing time becomes known at its arrival. We deal with the problem of minimizing the makespan, which is the time by which all jobs have been finished. We propose and analyze the following on-line LPT algorithm: At any time a machine becomes available for processing, schedule an available job with the largest processing time. We prove that this algorithm has a performance guarantee of $\frac{3}{2}$, and that this bound is tight. Furthermore, we show that any on-line algorithm will have a performance bound of at least 1.3473. This bound is improved to $(5-\sqrt{5})/2 \approx 1.3820$ for $m=2$. © 1997 Elsevier Science B.V.

Keywords: On-line scheduling; Worst-case analyses

1. Introduction

Until a few years ago, one of the basic assumptions in most deterministic scheduling literature was that all of the information needed to define the problem instance was known in advance. This assumption is often not valid in practice, however. Abandoning it has led to the rapidly emerging field of on-line scheduling. Since then, several on-line models have been proposed. The main difference between the models are the assumptions on when the information becomes available to the scheduler. For a description of these on-line models we refer to the survey of Sgall [4]. By *on-line*, we here mean that jobs arrive over time, and all job characteristics become known at their arrival time. Jobs do not have to be scheduled immediately upon arrival. At each time a machine is idle and a job is available, the algorithm decides which one of the available jobs is scheduled, if any.

In this note, we deal with the problem of scheduling jobs on identical parallel machines with the objective of minimizing the makespan. This problem is NP-hard when the off-line version is considered, although it can be solved in polynomial time by an on-line algorithm if preemption is allowed [3]. Vestjens [6] proved that the on-line preemptive version can even be solved for a special class of uniform machine problems. Hall and Shmoys [2] prove that on-line list

* Corresponding author.

[1] The main part of this work was carried out while the first author was working at the Eindhoven University of Technology.

scheduling has a performance guarantee of 2, even in the presence of precedence constraints. The *performance guarantee* of an algorithm is an upper bound of its *worst-case performance ratio*, which is defined to be the smallest constant ρ such that, for any instance of the problem, the algorithm finds a schedule with makespan no more than ρ times that of an optimal schedule. Shmoys et al. [5] describe a general method to use off-line algorithms to obtain algorithms that can handle unknown release dates. Although this method can be used for many different environments, it gives a performance guarantee of at least 2, even if the corresponding off-line problem can be solved to optimality. In the same paper Shmoys et al. provide a lower bound of $\frac{10}{9}$ on the worst-case performance ratio of any on-line algorithm. A well-known off-line algorithm for the problem without release dates is the LPT algorithm (e.g., see [1]). We consider the on-line version of this algorithm: At any time a machine becomes available for processing, schedule an available job with the largest processing time. Until now, nobody has proved any performance guarantee for this on-line algorithm.

This note is organized as follows. In Section 2 we show that the on-line LPT algorithm has a performance guarantee of $\frac{3}{2}$, independent of the number of machines. In Section 3 we provide new lower bounds on the worst-case performance of any on-line algorithm. These lower bounds indicate that the LPT algorithm performs quite well from a worst-case point of view.

2. The on-line LPT algorithm

The on-line LPT algorithm can be formally described as follows: At any time a machine becomes available for processing, schedule an available job the with largest processing time. If no jobs are available, then wait until the next arrival. Number the jobs according to the order of appearance. Let r_j and p_j be the release date and the processing time of job J_j, respectively. For a schedule σ, let $C_{\max}(\sigma)$ denote the makespan and let $S_j(\sigma)$ and $C_j(\sigma)$ denote the starting time and the completion time of job J_j in the schedule, respectively.

In this section we show that the on-line LPT algorithm has a performance guarantee of $\frac{3}{2}$. Our proof is by contradiction. Suppose the performance guarantee of the algorithm exceeds $\frac{3}{2}$, then there exists an instance, which we call *counterexample*, for which the algorithm produces a schedule with makespan more than $\frac{3}{2}$ times the makespan of an optimal schedule. Let $\mathscr{I}$ be a *smallest* counterexample, i.e., a counterexample consisting of a minimum number of jobs. Let σ denote the schedule produced by the on-line LPT algorithm for the instance $\mathscr{I}$, let π denote an optimal schedule, and let J_l be the job that finishes at $C_{\max}(\sigma)$.

In the rest of the section, we will investigate this smallest counterexample by establishing various relations between the optimal makespan and job-related information of J_l in σ. These relations are based on the argument that some amount of processing in σ has to be executed during some particular period of time in any optimal schedule. We start by characterizing some properties of σ.

Observation 1. *Without loss of generality, we may assume that, at any time before $C_{\max}(\sigma)$ in schedule σ, at least one machine is not idle.*

Proof. First, we show that, if there is a common idle period in σ before time $C_{\max}(\sigma)$, during which all machines are idle, then the schedule must start with the common idle period. Suppose this is not the case, i.e. at least one job has finished before the common idle period. Note that, according to the LPT algorithm, jobs that are scheduled after the common idle period must be released after this period. If we remove all the jobs that finish before this idle period, then the makespan of the schedule created by the LPT algorithm does not change, whereas the corresponding optimal makespan does not increase. Hence, the new instance is a smaller counterexample, which contradicts the fact that we are considering a smallest counterexample.

Therefore, schedule σ starts with a common idle period, if any. Note that the first job is released at time r_1, which is the end of this idle period. If we alter our problem instance by decreasing the release dates of all jobs by r_1, then the makespans of both the LPT schedule and the optimal schedule will decrease by this same amount, which implies that the ratio of the makespans will increase, and that the altered instance is also a minimal counterexample. □

Lemma 2. $S_l(\sigma) - r_l > C_{\max}(\pi)/2$.

Proof. Since $C_{\max}(\sigma) > \frac{3}{2}C_{\max}(\pi)$ and $r_l + p_l \leqslant C_{\max}(\pi)$, we have that $S_l(\sigma) - r_l = C_{\max}(\sigma) - p_l - r_l > \frac{3}{2}C_{\max}(\pi) - C_{\max}(\pi) = C_{\max}(\pi)/2$. □

In schedule σ there may exist a time interval before time r_l during which a machine is idle. Let $[t_s, t_f]$ be such an idle time interval with maximum t_f. If in σ all machines are busy before r_l, then let $t_s = t_f = 0$.

Lemma 3. $p_l \leqslant (C_{\max}(\pi) - t_f)/2$.

Proof. We will indicate $m + 1$ jobs with processing times greater than or equal to p_l and then show that the two jobs that are processed by the same machine in π, are released at or after time t_f. Hence, $C_{\max}(\pi) \geqslant t_f + 2p_l$, which yields the desired result.

Consider the jobs that are started before time $S_l(\sigma)$ and completed at or after time $S_l(\sigma)$. Due to the selection mechanism of the LPT-rule, no machine can be idle in the interval $[r_l, S_l(\sigma)]$, which has a positive length according to Lemma 2, and hence there are m of such jobs. Let J_j be any of these m jobs. If $S_j(\sigma) \geqslant r_l$, then since J_j started before J_l, the LPT-rule preferred J_j to J_l, which implies that $p_j \geqslant p_l$. If $S_j(\sigma) < r_l$, then $p_j = C_j(\sigma) - S_j(\sigma) > S_l(\sigma) - r_l > C_{\max}(\pi)/2$ by Lemma 2. So all of these m jobs together with J_l have processing times greater than $C_{\max}(\pi)/2$ if $p_l > C_{\max}(\pi)/2$, which is impossible since at least two of these jobs are processed by the same machine in π. Therefore, $p_l \leqslant C_{\max}(\pi)/2$.

What is left to prove is that the two jobs that are executed by the same machine in π, say J_j and J_k, are not available before time t_f. Suppose to the contrary that $r_j < t_f$. Since J_j is available before time t_f, $S_j(\sigma) < t_f \leqslant r_l$. Hence, the intervals $[r_j, S_j(\sigma)]$ and $[r_l, S_l(\sigma)]$ are disjoint. Furthermore, $S_j(\sigma) + p_j + p_l \geqslant S_l(\sigma) + p_l = C_{\max}(\sigma) > \frac{3}{2}C_{\max}(\pi)$ and $C_{\max}(\pi) \geqslant r_j + p_j + p_k \geqslant r_j + p_j + p_l$, which imply that $S_j(\sigma) - r_j > C_{\max}(\pi)/2$. Since all machines are busy during the two disjoint intervals $[r_j, S_j(\sigma)]$ and $[r_l, S_l(\sigma)]$, each of which has a length more than $C_{\max}(\pi)/2$, the total processing time of all jobs is more than $mC_{\max}(\pi)$, which is obviously impossible. □

Corollary 4. *The schedule σ contains idle time.*

Proof. If there is no idle time in σ, i.e., $t_f = 0$, then $S_l(\sigma)$ is an obvious lower bound for $C_{\max}(\pi)$ based on the total processing time, which, together with Lemma 3, implies that $C_{\max}(\sigma) = S_l(\sigma) + p_l \leqslant \frac{3}{2}C_{\max}(\pi)$, contradicting the fact that we are considering a counterexample. □

We have found an upper bound on the processing requirement of job J_l. We will also establish an upper bound on the start time $S_l(\sigma)$ of this job. We do this by deriving two complementary lower bounds for $C_{\max}(\pi)$ in terms of $S_l(\sigma)$. The first lower bound is based on the amount of processing that has to be executed after time t_f, and the second one is based on the total amount of processing.

Lemma 5. $S_l(\sigma) \leqslant C_{\max}(\pi) + (m-1)t_f/(2m) - p_l/m$.

Proof. Let $\bar{Q}$ be the set of all jobs that finish after time t_f in σ, and let Q be the subset of $\bar{Q}$ containing those jobs that start at or before time t_s, i.e., $Q = \{J_j \mid t_f - p_j < S_j(\sigma) \leqslant t_s\}$. Since during the interval $[t_s, t_f]$, which has a positive length according to Corollary 4, at least one machine is idle, it is easy to see that, for any job of $\bar{Q} \setminus Q$, the processing that is executed after t_f in σ cannot be scheduled earlier than t_f in any optimal schedule. In fact, for any job $J_j \in \bar{Q} \setminus Q$, either $r_j \geqslant t_f$ or $r_j = S_j(\sigma)$. On the other hand, for each job in Q, the maximum amount of processing currently executed after t_f in σ that could be executed before t_f in another schedule is δ, where $\delta = \max_{J_j \in Q}(S_j(\sigma) - r_j)$ and $\delta = 0$ if $Q = \emptyset$. Therefore, taking into account that all machines are busy during $(t_f, S_l(\sigma))$ and that $|Q| \leqslant m - 1$, we obtain the following lower bound based on the amount of processing that has to be executed after t_f in any schedule:

$$C_{\max}(\pi) \geqslant t_f + (S_l(\sigma) - t_f) + p_l/m - |Q|\delta/m$$
$$\geqslant S_l(\sigma) + (p_l - (m-1)\delta)/m,$$

where the third term on the right-hand side of the first inequality is the addition of p_l to the average load and the fourth term is the maximum we can subtract from the average load.

Now let us consider all the jobs. Note that, if $\delta > 0$, then in σ there is a period of time before t_s of length of at least δ, during which all machines are occupied. We already know that during the interval $(t_f, S_l(\sigma))$ all machines are occupied and that at any other time at least one machine is busy according to Observation 1.

Therefore, based on the total amount of processing, we obtain another lower bound:

$$\begin{aligned} C_{\max}(\pi) &\geqslant \delta + (S_l(\sigma) - t_f) + (p_l + t_f - \delta)/m \\ &= S_l(\sigma) + (p_l - (m-1)(t_f - \delta))/m. \end{aligned}$$

Adding up the two lower bounds displayed above gives

$$2C_{\max}(\pi) \geqslant 2S_l(\sigma) - (m-1)t_f/m + 2p_l/m,$$

from which the desired result follows immediately. □

Now we are ready to prove the main result of this section.

Theorem 6. *The on-line* LPT *algorithm has a performance guarantee of $\frac{3}{2}$, and this bound is tight.*

Proof. Using Lemmas 3 and 5 we derive

$$\begin{aligned} C_{\max}(\sigma) &= S_l(\sigma) + p_l \\ &\leqslant C_{\max}(\pi) + (m-1)t_f/(2m) - p_l/m \\ &\quad + (C_{\max}(\pi) - t_f)/2 \\ &= \tfrac{3}{2}C_{\max}(\pi) - t_f/(2m) - p_l/m \\ &\leqslant \tfrac{3}{2}C_{\max}(\pi). \end{aligned}$$

Since this contradicts the fact that we are considering a counterexample, we conclude that there is no counterexample and, therefore, the performance guarantee of the on-line LPT algorithm is at most $\frac{3}{2}$.

It is easy to find an instance for which the bound is asymptotically reached. For example, take the instance of $m+1$ jobs with $r_1 = \cdots = r_m = 0$, $p_1 = \cdots = p_m = 1$, and $r_{m+1} = \varepsilon$, $p_{m+1} = 2 - \varepsilon$, where ε is some small positive number. The LPT algorithm creates a schedule with makespan $3 - \varepsilon$, whereas an optimal schedule has a makespan of 2. If we let ε tend to zero, then the ratio tends to $\frac{3}{2}$. □

3. Lower bounds

In this section we show that, because of the lack of information concerning the future, no on-line algorithm can perform very well from a worst-case point of view.

Theorem 7. *Any on-line algorithm has a worst-case performance ratio of at least $1 + \alpha \approx 1.3473$, where α is the solution of $\alpha^3 - 3\alpha + 1 = 0$ in the interval $[\frac{1}{3}, \frac{1}{2}]$. For $m = 2$ this bound can be improved to $(5 - \sqrt{5})/2 \approx 1.3820$ by letting $\alpha = (3 - \sqrt{5})/2$.*

Proof. We show this result by describing a set of instances for which no on-line algorithm can guarantee an outcome strictly less than $1 + \alpha$ times the optimum. We again use σ to denote the schedule created by any given on-line algorithm and π to denote an optimal schedule.

Consider the following situation. The first job arrives at time 0 and has a processing time 1. The on-line algorithm decides to schedule the job at time S_1. If $S_1 \geqslant \alpha$, then we have that $C_{\max}(\sigma)/C_{\max}(\pi) \geqslant 1 + \alpha$. Note that α is chosen differently according to the number of machines. If $S_1 < \alpha$, then we let a second job arrive at time S_1 with processing time $\alpha/(1-\alpha)$.

If the algorithm decides to start this job at time $S_2 \leqslant S_1 + 1 - \alpha/(1-\alpha)$, then we let $m-1$ jobs arrive at time S_2, all having a processing time $1 + \alpha/(1-\alpha) - S_2$. An optimal schedule π for this instance has J_1 and J_2 scheduled on the same machine, and all other jobs get a machine of their own, which implies that $C_{\max}(\pi) = 1 + \alpha/(1-\alpha)$. In σ, at least one of the jobs that arrive at time S_2 cannot start before J_2 has finished, which implies that $C_{\max}(\sigma) \geqslant 1 + 2\alpha/(1-\alpha)$. Therefore, $C_{\max}(\sigma)/C_{\max}(\pi) \geqslant 1 + \alpha$.

On the other hand, if the on-line algorithm does not start the second job by time $S_1 + 1 - \alpha/(1-\alpha)$, then at this time some other jobs are released. We distinguish between the situations $m = 2$ and $m \geqslant 3$. If $m = 2$, then one job is released with processing time $1 - \alpha/(1-\alpha) - S_1$; recall that $\alpha = (3 - \sqrt{5})/2$ for $m = 2$. In π, J_2 and J_3 will be combined and J_1 gets its own machine, which implies that $C_{\max}(\pi) = 1$. In σ, the best way to complete the schedule is by immediately scheduling J_2 and by putting J_3 after J_1, which implies that $C_{\max}(\sigma) \geqslant 2 - \alpha/(1-\alpha)$. Hence, $C_{\max}(\sigma)/C_{\max}(\pi) \geqslant (2 - 3\alpha)/(1-\alpha) = 1 + \alpha$.

If $m \geqslant 3$, then $m-2$ jobs are released with processing time $\alpha/(1-\alpha)$ and one job with processing time $1 - \alpha/(1-\alpha)$. In π every job gets a machine of its own, except for the second and the last job, which implies that $C_{\max}(\pi) = S_1 + 1$. Since, in σ, J_2 is not started until time $S_1 + 1 - \alpha/(1-\alpha)$, we have that $C_{\max}(\sigma) \geqslant S_1 + 2 - \alpha/(1-\alpha)$. Since $S_1(\sigma) < \alpha$, the ratio $C_{\max}(\sigma)/C_{\max}(\pi)$

is minimized by letting $S_1(\sigma) = \alpha$, and this choice yields the ratio $(2 - 2\alpha - \alpha^2)/(1 - \alpha^2) = 1 + \alpha$ if we choose α to be the solution of $\alpha^3 - 3\alpha + 1 = 0$ in the interval $[\frac{1}{3}, \frac{1}{2}]$. □

References

[1] R.L. Graham, Bounds on multiprocessing timing anomalies, SIAM J. Appl. Math. 17 (1969) 416–429.

[2] L.A. Hall, D.B. Shmoys, Approximation schemes for constrained scheduling problems, Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 134–139.

[3] K.S. Hong, J.Y.-T. Leung, On-line scheduling of real-time tasks, IEEE Trans. Comput. 41 (1992) 1326–1331.

[4] J. Sgall, On-line scheduling – a survey, in: A. Fiat and G.J. Woeginger (eds.), On-line Algorithms, Lecture Notes in Computer Science, to appear.

[5] D.B. Shmoys, J. Wein, D.P. Williamson, Scheduling parallel machines on-line, SIAM J. Comput. 24 (1995) 1313–1331.

[6] A.P.A. Vestjens, Scheduling uniform machines on-line requires nondecreasing speed ratios, Math. Programming (B), to appear.