Scheduling on identical machines : how good is LPT in an on-line setting?

Memorandum COSOR 96-11, 1996, Eindhoven University of Technology

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

Bo Chen * Arjen P.A. Vestjens

March 13, 1996

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 3/2. 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 $m = 2$.

1 Introduction

Until a few years ago, one of the basic assumptions in deterministic scheduling 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. Two on-line models have been proposed. In the first model the jobs arrive in a list. The on-line algorithm has to schedule the first job in the list before it sees the next job in the list and the scheduling is irrevocable (e.g., see [Chen, Van Vliet & Woeginger, 1994]). In the second model jobs arrive over time. Next to the presence of release dates, this model differs from the first one in that 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 address the second model, and deal with the problem of scheduling jobs on identical parallel machines with the objective of minimizing the makespan. This

*The main part of this work was carried out while the first author was visiting the Eindhoven University of Technology

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 [Hong & Leung, 1992]. Vestjens [1994] proved that the on-line preemptive version can even be solved for a special class of uniform machine problems. A well-known off-line algorithm for the problem without release dates is the LPT algorithm (e.g., see [Graham, 1969]). 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, where 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.

This note is organized as follows. In Section 2 we show that the on-line LPT algorithm has a performance guarantee of 3/2, independent of the number of machines. In Section 3 we provide 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 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 3/2. Our proof is by contradiction. Suppose the performance guarantee of the algorithm exceeds 3/2, then there exists an instance, which we call *counterexample*, for which the algorithm produces a schedule with makespan more than 3/2 times the makespan of an optimal schedule. Let $\mathcal{I}$ 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 $\mathcal{I}$, let π denote an optimal schedule, and let J_l be the job that finishes at $C_{\max}(\sigma)$. 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, that is, 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 $C_{\max}(\sigma) > (3/2)C_{\max}(\pi)$ and $r_l + p_l \leq C_{\max}(\pi)$, we have that $S_l(\sigma) - r_l = C_{\max}(\sigma) - p_l - r_l > (3/2)C_{\max}(\pi) - C_{\max}(\pi) = C_{\max}(\pi)/2$. □

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 the idle time interval that finishes last. If in σ all machines are busy before r_l, then let $t_s = t_f = 0$.

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

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

Consider the jobs that are started before time $S_l(\sigma)$ and completed at or after time $S_l(\sigma)$. There are m of such jobs, since no machine can be idle in the interval $[r_l, S_l(\sigma)]$ due to the selection mechanism of the LPT-rule. Let J_j be any of these m jobs. If $S_j(\sigma) \geq r_l$, then the LPT-rule preferred J_j to J_l, which implies that $p_j \geq p_l$. 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 time 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, $p_l \leq C_{\max}(\pi)/2$.

What is left to prove is that the 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, $S_j(\sigma) < t_f \leq r_l$. Hence, the intervals $[r_j, S_j(\sigma)]$ and $[r_l, S_l(\sigma)]$ are disjoint. Furthermore, $S_j(\sigma) + p_j + p_l \geq S_l(\sigma) + p_l = C_{\max}(\sigma) > (3/2)C_{\max}(\pi)$ and $C_{\max}(\pi) \geq r_j + p_j + p_l$, 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 does contain 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 $C_{\max}(\sigma) = S_l(\sigma) + p_l \leq (3/2)C_{\max}(\pi)$, 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 total amount of processing, and the second one is based on the amount of processing that has to be executed after period $[t_s, t_f]$.

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

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., $Q = \{J_j | t_f - p_j < S_j(\sigma) \leq t_s\}$. 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 $r_j \geq t_f$ 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 $|Q| \leq m-1$, we obtain the following lower bound based on the amount of processing that has to be executed after t_f in any schedule:

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

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,

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

Adding up the two lower bounds displayed above gives

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

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* $3/2$*, and this bound is tight.*

Proof. Using Lemmas 3 and 5 we derive

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

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 $3/2$.

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 $3/2$. □

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* $[1/3, 1/2]$*. 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 $S_1 \geq \alpha$, then we have that $C_{\max}(\sigma)/C_{\max}(\pi) \geq 1 + \alpha$. 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 $S_2 \leq S_1 + 1 - \alpha/(1-\alpha)$, 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 $C_{\max}(\sigma) \geq 1 + 2\alpha/(1-\alpha)$. Therefore, $C_{\max}(\sigma)/C_{\max}(\pi) \geq 1 + \alpha$.

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 $m \geq 3$. 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 $C_{\max}(\sigma) \geq 2 - \alpha/(1-\alpha)$. Hence, $C_{\max}(\sigma)/C_{\max}(\pi) \geq (2-3\alpha)/(1-\alpha) = 1+\alpha$.

If $m \geq 3$, 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 $C_{\max}(\sigma) \geq S_1 + 2 - \alpha/(1-\alpha)$. 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 $[1/3, 1/2]$. □

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. [1969], Bounds on multiprocessing timing anomalies, *SIAM Journal on Applied Mathematics* **17**, 416–429.

Hong, K.S., and J.Y-T. Leung [1992], On-line scheduling of real-time tasks, *IEEE Transactions on Computers* **41**, 1326–1331.

Vestjens, A.P.A. [1994], *Scheduling uniform machines on-line requires nondecreasing speed ratios*, Memorandum COSOR 94-35, Eindhoven University of Technology.