Computing 62, 179–187 (1999)

Computing

Semi On-Line Scheduling on Two Identical Machines

Y. He[*] and **G. Zhang**[**], Hangzhou

Received February 23, 1998; revised August 5, 1998

Abstract

This paper investigates two different semi on-line scheduling problems on a two-machine system. In the first case, we assume that all jobs have their processing times in between p and rp ($p > 0, r \geq 1$). In the second case, we assume that the largest processing time is known in advance. We show that one has a best possible algorithm with worst case ratio 4/3 while LS is still the best possible for the other problem with ratio $(r+1)/2$ which is still $3/2$ in the worst case $r = 2$.

AMS Subject Classifications: 90B35, 90C27.

Key Words: Analysis of algorithm, on-line scheduling, worst-case ratio.

1. Introduction

In the parallel identical machine scheduling problem, we are given a set $\mathcal{J} = \{p_1, p_2, \ldots, p_n\}$ of independent jobs, each with a positive processing time, that must be scheduled on m parallel and identical machines. We identify the jobs with their processing times. The jobs and machines are available at time zero, and no preemption is allowed. The objective is to minimize the overall completion time C_{max}, called *makespan*. This problem is one of the basic NP-complete problems [5] and usually denoted by $P||C_{max}$. A scheduling problem is called *on-line* if it requires to schedule jobs irrevocably on the machines as soon as they are given, without any knowledge about jobs that follow later on. If we have full information on the job data before constructing a schedule, this problem is called *off-line*. In practice, problems are often not really on-line or off-line but somehow in between. This means that, some partial information about the jobs is available and we cannot rearrange any job which has been assigned to machines. Such a case is defined as a *semi on-line* problem. Algorithms for a semi on-line problem are called semi on-line algorithms.

In a worst-case analysis, the performance of an on-line or a semi on-line algorithm

[*]Supported by NSFC grant (19701028) and SFB 003 "Optimierung und Kontrolle" Projektbereich Diskrete Optimierung.
[**]Supported by the National Natural Science Foundation of China (1980/032).

is measured through the worst-case ratio with respect to the optimal solution of the off-line problem. For a set $\mathcal{J}$ of jobs and an approximation algorithm A, let $w_A(\mathcal{J})$ denote the makespan produced by the algorithm A and let $w^*(\mathcal{J})$ denote the optimal makespan in an off-line version. Then the worst-case ratio of the algorithm A is defined as

$$R_A = \sup_{\mathcal{J}}\{\frac{w_A(\mathcal{J})}{w^*(\mathcal{J})}\}.$$

The simplest algorithm for the on-line parallel machine scheduling problem, is the *List Scheduling* (LS in short) algorithm, which was introduced by Graham [6] in 1966. This algorithm always assigns the current job to the machine with minimum workload on it at the moment. Graham showed $R_{LS} = 2 - 1/m$. Several algorithms have been proposed which have a slightly better worst-case ratio than the LS algorithm in [1, 2, 4, 7]. However, in 1989, Faigle, Kern and Turán [3] observed that LS is the best possible on-line algorithm for 2 and 3 machines. It means that there is no deterministic on-line algorithm for $P||C_{max}$ with a worst-case ratio better than $3/2$ and $5/3$ for $m = 2, 3$, respectively.

Recently, some new results on semi on-line algorithms for $P||C_{\max}$ were presented. In the semi on-line problem discussed in [10], the jobs have to be assigned to a machine one by one, the processing time of each job is unknown before it is assigned, but the jobs are known to arrive in non-increasing order of their processing times. For the cases of two and three parallel machines, the best possible algorithms are given. The corresponding worst-case ratios are 4/3 and 7/5, respectively. The general case in which the number of machine is arbitrary is also considered in the same paper. Kellerer et al. [9] investigate three semi on-line versions of the partition problems related to $P2||C_{\max}$. In the first problem, the jobs arrive one by one and a buffer of length k is available to maintain k jobs. Therefore, if the buffer is not full, an incoming job can either be immediately assigned to a machine or be temporarily assigned to the buffer. If there are already k jobs in the buffer, we can either assign the incoming job to a machine or assign one of the k jobs in the buffer to a machine and stock the incoming job in the buffer. In the second problem, again the jobs have to be assigned one by one, but two sets of two parallel machines are available for the computation of the solution. One copy is made for each incoming job and each of the two identical jobs has to be assigned to a machine by each of the two sets before arrival of the subsequent job. Finally the better of the two solutions independently obtained by the two sets is chosen. In the last problem, the jobs have to be assigned to a machine upon their arrival, but the total processing time of all jobs is known in advance. For each of three problems above, the best possible semi on-line algorithm with worst-case ratio 4/3 has been presented in [9]. Independently the first result has also been presented in [11]. Hence, LS is not the best semi on-line algorithm for each of the problems above.

In general, in a semi on-line version of a problem the conditions for a problem to be considered on-line are somehow relaxed. Different ways of relaxing the conditions give rise to different semi on-line versions. In many applications it is not easy to get the full information of all jobs before they come, but we may know some partial

information on the job data as a priori. For example, we know that the processing times of all jobs are normally distributed in a certain time-period (tightly-grouped processing times), or the processing time of some important jobs (for example, the largest job) is known in advance. In this paper, we consider these two semi on-line scheduling problems on two machines. For each of these problems, the best possible semi on-line algorithm is presented.

This paper is organized as follows. Section 2 deals with the first semi on-line problem, where all jobs have tightly-grouped processing times. Section 3 investigates a semi on-line problem in which the largest processing time is known in advance. Finally, Section 4 contains some final remarks.

2. Tightly-Grouped Processing Times

As above, we will consider the problem $P2||C_{max}$. We assume that the jobs arrive one by one and the processing times of all jobs are in the interval $[p, rp]$, where $p > 0, r \geq 1$. We call this problem $\mathcal{P}_1$. We first analyze the worst-case behaviour of LS algorithm, then we show that it is the best one for our problem $\mathcal{P}_1$.

Theorem 2.1. *Applying LS algorithm to $\mathcal{P}_1$, the worst-case ratios are*

(i) $R_{LS} = 3/2 \quad \text{for} \quad r \geq 2,$

(ii) $R_{LS} = (1 + r)/2 \quad \text{for} \quad 1 \leq r < 2.$

Proof: Since (i) immediately follows from Graham's work in [6], we only need to prove (ii). By normalizing all jobs we can assume that $p = 1$. First we will prove that for any list of jobs $R_{LS} \leq (1 + r)/2$, and then we give an instance which shows the bound is tight. Without loss of generality, we can suppose the last job p_n determines the LS makespan and s is the starting time of p_n. Then

$$w^*(\mathcal{J}) \geq \frac{\sum_{i=1}^{n} p_i}{2}, \quad s \leq \frac{\sum_{i=1}^{n-1} p_i}{2}. \tag{1}$$

Therefore,

$$\frac{w_{LS}(\mathcal{J})}{w^*(\mathcal{J})} = \frac{s + p_n}{w^*(\mathcal{J})} \leq 1 + \frac{p_n}{2w^*(\mathcal{J})}. \tag{2}$$

From the above inequality (2), we realize that, if $p_n \leq (r - 1)w^*(\mathcal{J})$, then we have $R_{LS} \leq (1 + r)/2$. So, in the following, we suppose $p_n > (r - 1)w^*(\mathcal{J})$. Let $k' = \lceil n/2 \rceil$. Then in the optimal solution, one of the two machines processes at least k' jobs. So $w^*(\mathcal{J}) \geq k'$. Since $r \geq p_n > (r - 1)w^*(\mathcal{J}) \geq (r - 1)k'$, i.e., $(k' - 1)r < k'$, we have

$$(i - 1)r < i, \quad \text{for } i = 1, \ldots, k'. \tag{3}$$

These inequalities (3) tell us the following two important facts.

Fact 1. In the LS schedule, p_{2i-1} and p_{2i} are processed by different machines, $i = 1, \ldots, \lfloor n/2 \rfloor$. That is to say, for $n = 2k$, each machine processes exactly k jobs; for $n = 2k + 1$, the machine determining makespan processes $k + 1$ jobs.

Fact 2. In the optimal schedule, for $n = 2k$, each machine must process k jobs (Otherwise $w^*(\mathcal{J}) \geq k + 1 > kr \geq w_{LS}(\mathcal{J})$); for $n = 2k + 1$, the machine determining the optimal makespan processes $k + 1$ jobs.

We consider two cases below with respect to the value of n. Firstly we define some notations. In the following of this section, let S_{ij} and S^*_{ij} denote the job sets processed on machine i ($i = 1, 2$) after processing p_j in the LS schedule and the optimal schedule, respectively. Let $l(S)$ denote the sum of the jobs in the set S.

Case 1. $n = 2k$. In this case, by Fact 1 , $|S_{1n-2}| = |S_{2n-2}| = k - 1$, $\mathcal{J} = S_{1n-2} \cup S_{2n-2} \cup \{p_{n-1}, p_n\}$. By Fact 2, $|S^*_{1n}| = |S^*_{2n}| = k$, $\mathcal{J} = S^*_{1n} \cup S^*_{2n}$.

Lemma 2.2. *For each $i = 1, 2$, there exists some $j \in \{1, 2\}$, such that*

$$|(S_{in-2} \cup \{p_n\}) \cap S^*_{jn}| \geq \lfloor \frac{k+1}{2} \rfloor.$$

Proof: We only prove the case $i = 1$. The case $i = 2$ can be shown analogously. If the above assertion is not true, then we have

$$|(S_{1n-2} \cup \{p_n\}) \cap S^*_{1n}| \leq \lfloor \frac{k+1}{2} \rfloor - 1, \tag{4}$$

$$|(S_{1n-2} \cup \{p_n\}) \cap S^*_{2n}| \leq \lfloor \frac{k+1}{2} \rfloor - 1. \tag{5}$$

Summing (4) and (5),

$$|(S_{1n-2} \cup \{p_n\}) \cap \mathcal{J}| \leq 2\lfloor \frac{k+1}{2} \rfloor - 2 \leq k - 1. \tag{6}$$

We thus get $|S_{1n-2}| \leq k - 2$, which is a contradiction. □

With Lemma 2.2, we are ready to get the required worst-case ratio. Since p_n, p_{n-1} are processed by different machines in the LS schedule, and p_n determines the makespan, we have

$$w_{LS}(\mathcal{J}) = \max\{l(S_{1n-2}), l(S_{2n-2})\} + p_n \text{ and } w^*(\mathcal{J}) \geq l(S^*_{jn}), \ j = 1, 2. \tag{7}$$

Suppose machine i determines the LS makespan. It means $w_{LS}(\mathcal{J}) = l(S_{in-2}) +$

p_n. Hence there exists some j such that

$$w_{LS}(\mathcal{J}) - w^*(\mathcal{J}) \le l(S_{in-2} \cup \{p_n\}) - l(S^*_{jn}) \le \frac{k(r-1)}{2}, \tag{8}$$

because Lemma 2.2 tells us that there are at most $k - \lfloor (k+1)/2 \rfloor \le k/2$ different jobs between $S_{in-2} \cup \{p_n\}$ and S^*_{jn}. Since $w^*(\mathcal{J}) \ge k$, we have

$$\frac{w_{LS}(\mathcal{J})}{w^*(\mathcal{J})} \le \frac{1+r}{2}, \tag{9}$$

which completes Case 1.

Case 2. $n = 2k+1$. Hence $|S_{1n-1}| = |S_{2n-1}| = k$, $\mathcal{J} = S_{1n-1} \cup S_{2n-1} \cup \{p_n\} = S^*_{1n} \cup S^*_{2n}$, $|S^*_{1n}| = k$ or $k+1$, $|S^*_{2n}| = k$ or $k+1$. In this case, $w_{LS}(\mathcal{J}) = \min\{l(S_{1n-1}), l(S_{2n-1})\} + p_n$. Similarly, we have

Lemma 2.3. *For every $i = 1, 2$, there exists some $j \in \{1, 2\}$, such that*

$$|(S_{in-1} \cup \{p_n\}) \cap S^*_{jn-1}| \ge \lfloor \frac{k}{2} \rfloor + 1.$$

From Lemma 2.3, we claim that there are at most $k + 1 - (\lfloor k/2 \rfloor + 1) \le k - (k-1)/2 = (k+1)/2$ different jobs between $S_{in-1} \cup \{p_n\}$ and S^*_{jn}. So with the same arguments as Case 1 we can meet the desired worst-case ratio. The instance $\mathcal{J} = \{1, 1, r\}$ implies that the bound $(1+r)/2$ is tight. Therefore, we finish the proof of Theorem 2.1. □

Theorem 2.4. *Any on-line algorithm A for $\mathcal{P}_1$ has a worst-case ratio as follows:*

$$R_A \ge \frac{1+r}{2} \quad \text{for} \quad r \le 2 \quad \text{and} \quad R_A \ge \frac{3}{2} \quad \text{for} \quad r > 2.$$

Proof: For $r \le 2$, consider the following instance. The first two jobs both have a processing time of 1. If an algorithm A assigns them to different machines, the next and last job with a processing time r comes, the makespan $w_A = 1 + r$ while the optimum makespan $w^* = 2$. It follows that $w_A/w^* = (1+r)/2$. If the first two jobs are assigned to the same machine by the algorithm A, then no further job comes any more. It follows $w_A/w^* = 2$. Therefore, we conclude that for any on-line algorithm A, $R_A \ge (1+r)/2$.

For $r > 2$, the instance is almost the same except that the processing time of the third job is always 2 if such a job is needed. □

The following theorem can be proved in the same way as that in Theorem 2.1.

Theorem 2.5. *Applying LS to the problem $Pm||C_{max}$, if all jobs have their processing time within interval $[p, rp]$, where $p > 0, r \leq 2$, then*

$$\frac{w_{LS}(\mathcal{J})}{w^*(\mathcal{J})} \leq 1 + \frac{(m-1)(r-1)}{m}.$$

3. The Largest Processing Time is Known

In this section, we still consider the problem $P2||C_{max}$. We assume that the jobs come one by one, and the largest processing time is known in advance. We call this problem $\mathcal{P}_2$.

Theorem 3.1. *Any algorithm A for $\mathcal{P}_2$ has a worst-case ratio $R_A \geq 4/3$.*

Proof: Consider the following instances. Suppose the largest processing time is 2. The first two jobs have the same processing time 1. If an algorithm A assigns them to different machines, the next (also the last) job with processing time 2 comes, the makespan $w_A = 3$ while the optimum makespan $w^* = 2$. It follows that $w_A/w^* = 3/2$. If the first two jobs are assigned to the same machine by algorithm A, then the incoming two jobs have the same processing time 2. Therefore, $w_A \geq 4$ and $w^* = 3$. It concludes that $w_A/w^* = 4/3$. Thus for any semi on-line algorithm A, $R_A \geq 4/3$. □

Denote by $p_{\max}$ the largest processing time. A job is called as a large job if its processing time is $p_{\max}$. We will give a best possible algorithm below.

Algorithm PLS (Premeditated List Scheduling).

Step 1. Always assign current jobs to Machine 1 unless one of the following cases happens.

1. The current job is a large job.

2. If the current job is assigned to Machine 1, then the workload of Machine 1 would be larger than $2p_{\max}$.

Step 2. Once 1 or 2 in Step 1 happens, then assign the current job into Machine 2. Thereafter apply LS algorithm to all subsequent jobs.

The difference between the workloads of the machines is no larger than the largest processing time. The main idea of the algorithm PLS is to leave some room for one largest job till both machines have workloads large enough. The worst-case performance result is provided in the following theorem.

Theorem 3.2. *The worst-case ratio of the algorithm PLS is not greater than* $4/3$.

Proof: We will prove that $w_{PLS}(\mathcal{J})/w^*(\mathcal{J}) \leq 4/3$ holds for any instance $\mathcal{J}$. Suppose that $\mathcal{J}$ is an instance and p_n is the last job. Immediately before p_n is assigned by PLS, the workloads of two machines are denoted by M_1 and M_2, respectively. We consider three cases below.

Case 1. $M_2 = 0$. In this case, $p_n = p_{\max}$. If $M_1 \leq p_{\max}$, then $w_{PLS}(\mathcal{J}) = p_{\max}$. The PLS schedule is thus optimal. If $M_1 > p_{\max}$, then $w_{PLS}(\mathcal{J}) = M_1$. From the algorithm, $M_1 \leq 2p_{\max}$. Moreover, $w^*(\mathcal{J}) \geq (M_1 + p_{\max})/2$. Therefore,

$$\frac{w_{PLS}(\mathcal{J})}{w^*(\mathcal{J})} \leq \frac{2M_1}{M_1 + p_{\max}} \leq \frac{4}{3}. \tag{10}$$

Case 2. $0 < M_2 \leq M_1$. If $M_2 < p_{\max}$, denote by p_s the first job assigned to Machine 2. By the algorithm PLS, $p_s + M_1 > 2p_{\max}$. Then $M_1 > p_{\max}$ and $M_1 + M_2 > 2p_{\max}$. Before the last job p_n comes, the workload of Machine 2 is always less than Machine 1, and no large job comes. Therefore, p_n is a large job and it is assigned to Machine 2. Machine 1 does not accept jobs any more after p_s comes. So,

$$M_1 \leq 2p_{\max} \quad \text{and} \quad M_2 + p_{\max} < 2p_{\max}. \tag{11}$$

From (11), we have

$$w_{PLS}(\mathcal{J}) = \max\{M_1, M_2 + p_{\max}\} \leq 2p_{\max}, \tag{12}$$

$$w^*(\mathcal{J}) \geq \frac{M_1 + M_2 + p_{\max}}{2} > \frac{3p_{\max}}{2}. \tag{13}$$

Inequalities (12) and (13) imply $w_{PLS}(\mathcal{J})/w^*(\mathcal{J}) < 4/3$.

If $M_2 \geq p_{\max}$, then

$$M_2 + p_n \leq 2M_1, \quad \text{i.e.,} \quad 3(M_2 + p_n) \leq 2(M_1 + M_2 + p_n), \tag{14}$$

which follows that $2(M_2 + p_n)/(M_1 + M_2 + p_n) \leq 4/3$. On the other hand, since $M_1 - M_2 \leq p_{\max}$, we have

$$3M_1 \leq 2M_1 + M_2 + p_{\max} \leq 2M_1 + 2M_2 + 2p_n,$$

$$\text{i.e.,} \quad \frac{2M_1}{(M_1 + M_2 + p_n)} \leq \frac{4}{3}. \tag{15}$$

Combining (14) and (15), we get

$$\frac{w_{PLS}(\mathcal{J})}{w^*(\mathcal{J})} \leq \frac{2\max\{M_1, M_2 + p_n\}}{M_1 + M_2 + p_n} \leq \frac{4}{3}. \tag{16}$$

Case 3. $M_2 > M_1$. In this case, p_n is assigned to Machine 1. $M_2 \geq p_{\max}$ must hold, otherwise $M_1 > p_{\max}$, by the same idea as in Case 2, which contradicts the assumption that $M_2 > M_1$. If $M_1 + p_n < p_{\max}$, by the algorithm PLS, Machine 2 accepts nothing but one large job. Then $w_{PLS}(\mathcal{J}) = M_2 = p_{\max}$ which means the schedule is optimal. Hence we can assume that

$$M_1 + p_n \geq p_{\max} \tag{17}$$

holds. Because of $M_2 > M_1$ and $M_2 > p_{\max}$, We have

$$2M_2 > M_1 + M_2 \geq M_1 + p_{\max} \geq M_1 + p_n, \quad \text{i.e.,} \quad \frac{2(M_1 + p_n)}{M_1 + M_2 + p_n} < \frac{4}{3}. \tag{18}$$

On the other hand, $M_2 - M_1 \leq p_{\max}$. Combining this with (17), we have $M_2 \leq M_1 + p_{\max} \leq 2(M_1 + p_n)$. It implies that

$$\frac{2M_2}{(M_1 + M_2 + p_n)} \leq \frac{4}{3}. \tag{19}$$

From (18) and (19), we have

$$\frac{w_{PLS}(\mathcal{J})}{w^*(\mathcal{J})} \leq \frac{2\max\{M_2, M_1 + p_n\}}{M_1 + M_2 + p_n} \leq \frac{4}{3}. \tag{20}$$

We thus get the result. □

By Theorem 3.1 and Theorem 3.2, the algorithm PLS is the best possible semi on-line algorithm which attains the tight bound $4/3$.

4. Final Remarks

In this paper we considered two related semi on-line scheduling problems. We analyzed the parametric behavior of the famous LS algorithm, where r is defined as a factor such that all processing times of jobs are in the interval $[p, rp]$ for some positive number p. We showed that LS is the best possible algorithm for $m = 2$. As a byproduct, we also estimated a worst-case ratio for $m > 2$. However we do not think the bounds of Theorem 2.5 are always tight. For example, for $m = 3$, the ratio of Theorem 2.5 is $(2r + 1)/3$, and the instance $\mathcal{J} = \{1, 1, 1, r, r, r, r\}$ shows that it is tight for $r \leq 3/2$. But for $r \geq 6$, we know [3] that the ratio is $5/3$. Furthermore, for $9/5 \leq r \leq 2$, we can prove the worst-case ratio is no greater than $(r + 1)/2$. The instance $\mathcal{J} = \{1, 1, 1, r\}$ shows the bound $(r + 1)/2$ is tight. For the remaining value of r, the worst case ratio of LS is still unknown. So we also do not know what is the best possible algorithm for $m = 3$, although we guess that LS should be. For the second problem, we propose a best possible algorithm when $m = 2$. It is also interesting to extend this result to the general case $m \geq 3$.

Acknowledgement

The authors wish to thank two referees for their valuable suggestion which improved the readability of this paper.

References

[1] Albers, S.: Better bounds for on-line scheduling. Proc. 29th Annual ACM Symp. on Theory of Computing, 130-139 (1997).
[2] Bartal, Y., Fiat, A., Karloff, A., Vohra, R.: New algorithm for an ancient scheduling problem. Proc. 24th Annual ACM Symp. on Theory of Computing, 51-58 (1992).
[3] Faigle, U., Kern, W. and Turán, G.: On the performance of on-line algorithm for particular problems. Acta Cybern. *9*, 107-119 (1989).
[4] Galambos, G., Woeginger, G.: An on-line scheduling heuristic with better worst-case ratio than Graham's list scheduling. SIAM J. Comput. *22*, 349-355 (1993).
[5] Garey, M. R., Johnson, D. S.: Computers and intractability: A guide to the theory of NP-completeness. San Francisco: Freeman, 1979.
[6] Graham, R. L.: Bounds for certain multiprocessing anomalies. Bell System Tech. *45*, 1563-1581 (1966).
[7] Karger, D. R., Phillips, S. J., Torng, E.: A better algorithm for an ancient scheduling algorithm. J. Alg. *20*, 400-430 (1996).
[8] Kellerer, H.: Bounds for nonpreemptive scheduling jobs with similar processing times on multi-processor systems using LPT-algorithm. Computing *46*, 183-191 (1991).
[9] Kellerer, H., Kotov., V., Speranza, M. R., Tuza, Z.: Semi on-line algorithms for the partition problem. Oper. Res. Lett. *21*, 235–242 (1997).
[10] Liu, W. P., Sidney, J. B., Vliet, A. van: Ordinal algorithms for parallel machine scheduling. Oper. Res. Lett. *18*, 223-232 (1996).
[11] Zhang, G.: A simple semi on-line algorithm for $P2||C_{max}$ with a buffer. Inf. Proc. Lett. *61*, 145-148 (1997).

Y. He
Department of Applied Mathematics
Zhejiang University
Hangzhou 310027
P. R. China
e-mail: heyong@math.zju.edu.cn

G. Zhang
Institute of Mathematics
Zhejiang University
Hangzhou 310027
P. R. China
e-mail: zgc@math.zju.edu.cn