Minimizing Makespan on Parallel Machines with Machine Eligibility Restrictions

Chien-Hung Lin[1] and Ching-Jong Liao[*,2]

[1]*Jinwen University of Science and Technology, Taipei County, Taiwan*

[2]*Department of Industrial Management, National Taiwan University of Science and Technology, Taipei, Taiwan*

Abstract: In this paper, we consider a parallel machine problem where machines and jobs can be classified into two levels: high and low levels. A high-level machine can process all jobs while a low-level machine can process only low-level jobs. The objective of the problem is to minimize the makespan. This problem is a special case of the parallel machine problem with machine eligibility restrictions. The problem is NP-hard and a heuristic algorithm has recently been proposed. However, there are no algorithms in the literature that can solve the problem to optimality. In this paper, we develop such an exact algorithm by utilizing some useful properties inherent in the problem. Computational experiments show that the developed algorithm can find the optimal solution for various-sized problems in a short time.

Keywords: Parallel machines, Machine eligibility, Scheduling, Makespan.

INTRODUCTION

The parallel machine scheduling problem has been discussed extensively in the literature under the assumption that all the m machines are capable of processing each of the n jobs. However, in many real-world situations a job can only be processed on a subset of the m parallel machines. A common situation involves the acquisition of new machines that usually process a given job at the same speed as existing machines, but have the capability to process other jobs that existing machines cannot process [1]. Similarly, these new machines may not be able to process some jobs that existing machines, with older technologies, can process. As an example given by Centeno and Armacost [2], the Probe workcenter in wafer testing, one of the major phases in the manufacturing of integrated circuits, frequently involves a number of testing machines in parallel with different capabilities but with the same speed. This machine (server) eligibility restriction also exists in the service industry where various customers are entitled to many different grades of service levels. For example, credit card companies often have platinum or infinite members, who are more valued than the regular members and are usually entitled to premium services [3].

The purpose of this paper is to develop an exact algorithm to find optimal solutions for the parallel machine problem in the presence of machine eligibility restrictions. The objective is to minimize the makespan. Following the three-field notation, the problem can be denoted by $Pm/M_j/C_{\max}$, where job j is only allowed to be processed on subset M_j of the m machines [4]. We address a special case of $Pm/M_j/C_{\max}$ where both machines and jobs can be classified into two levels: high and low levels. Each job is allowed to be processed on a particular machine only when the level of the job is no lower than the level of the machine. The addressed problem has been shown to be NP-hard by Hwang *et al.* [3].

Due to the complexity of $Pm/M_j/C_{\max}$, exact algorithms can be developed only for problems with unit-length jobs, i.e., $Pm/p_j=1,M_j/C_{\max}$. Pinedo [4] showed that the least flexible job (LFJ) rule is optimal for $Pm/p_j=1,M_j/C_{\max}$ when the M_j sets are nested. The sets M_j are nested if the functionality of machines may overlap but not partially. Glass and Mills [5] improved Pinedo's algorithm with a new lower bound and further discussed other scheduling objectives on the same problem. For uniform parallel machines, Lin and Li [6] developed polynomial time algorithms to minimize the makespan with jobs requiring identical processing. Li [7] improved their algorithms and extended their models to cover various other scheduling objectives.

On the other hand, research on machine eligibility restrictions with general jobs (non-unit-length jobs) focuses on developing heuristics for various objectives. Centeno and Armacost [2] considered a maximum lateness problem with non-zero release times. They discussed the problem in a real industrial setting where due dates are equal to release dates plus a constant. Later, they studied an on-line makespan minimization problem with jobs having non-zero release times [1]. As an application in the service industry, Hwang *et al.* [3] considered a parallel machine scheduling problem with the makespan objective under different grades of service levels. They investigated the worst-case performance of LPT (longest processing time) and provided an LG-LPT (lowest grade-longest processing time) heuristic rule for the problem. According to the heuristic rule, jobs are first sequenced based on the priority of the grade of service levels, and then jobs within the same level are sequenced in the LPT order.

*Address correspondence to this author at the Department of Industrial Management, National Taiwan University of Science and Technology, 43 Keelung Road, Section 4, Taipei, Taiwan 106; E-mail: cjl@im.ntust.edu.tw

In this paper we consider the same problem as Hwang *et al*. [3]. To the best of our knowledge, the problem has not been solved to optimality by any algorithm. In the following, we will propose an algorithm for obtaining the optimal solution to the problem.

PROBLEM FORMULATION

Consider a two-level scheduling problem with $m\ (= m_h + m_l)$ machines and $n\ (= n_h + n_l)$ jobs, where the subscripts h and l stand for the high and low levels. Let p_j be the processing time of job j, $T_h\ (T_l)$ be the sum of processing times of all high-level (low-level) jobs, and $T\ (= T_h + T_l)$ be the sum of processing times of all jobs. The objective considered is minimizing the makespan. We consider the problem under the following assumptions:

- Both machines and jobs are classified into two levels: high and low levels. A high-level machine can process all jobs while a low-level machine can process only low-level jobs.
- All the machines process jobs at the same speed.
- The numbers of machines in both levels are fixed and known in advance, and so are the numbers of jobs in both levels.
- The processing times of all jobs are integer.
- The ready times of all jobs are zero.
- No machine may process more than one job at a time.
- Preemptions are not allowed.

SINGLE HIGH-LEVEL MACHINE PROBLEM

Since all jobs can be processed by the high-level machine, the number of high-level machines has a greater impact on the solution. Thus, we start to investigate the problem with a single high-level machine, i.e., $m_h = 1$.

Let M_h denote the single high-level machine and p_l denote the largest processing time of all low-level jobs. In the following theorem, we show that the problem with a single high-level machine can be formulated as a parallel machine problem without machine eligibility restrictions.

Theorem 1. The problem with a single high-level machine can be formulated as a $Pm // C_{\max}$ problem.

Proof. When $m_h = 1$, M_h must be busy for processing during the interval $[0, T_h]$. It remains to assign T_l, which is equivalent to a parallel machine problem with non-simultaneous machines where only one machine starts at time T_h and all the other m_l machines start at time zero. We can adopt the same idea of Liao *et al.* [8] by treating the interval $[0, T_h]$ as a composite job and scheduling it with all low-level jobs on the parallel machines, which is simply a $Pm // C_{\max}$ problem.

In the next theorem, we develop a lower bound on the makespan for the single high-level machine problem.

Theorem 2. For the problem with $m_h = 1$, a lower bound on the makespan is given by

$$\underline{C}_{\max} = \max\{p_l, T_h, \lceil T/m \rceil\}.$$

Proof. The processing requirement T_h has to be processed on M_h, and hence T_h is a lower bound. Now consider the following two cases:

(i) $T_h \geq T_l / m_l$: In this case, all the processing requirement T_l should be processed on the m_l machines, which is reduced to a parallel machine problem. A lower bound for the problem is $\max\{p_l, \lceil T_l / m_l \rceil\}$. Since $T_h \geq T_l / m_l$, the lower bound becomes $\max\{p_l, T_h\}$.

(ii) $T_h < T_l / m_l$: In this case, T_l cannot be completed at time T_h. To minimize the makespan, M_h may process part of T_l, so a lower bound is $\lceil T/m \rceil$.

Combining all the results, a lower bound can be established as $\underline{C}_{\max} = \max\{p_l, T_h, \lceil T/m \rceil\}$.

Example 1. As an illustration of Theorem 2, consider a three-machine ($m_h = 1$ and $m_l = 2$) seven-job problem with the data given in Table **1**. We can compute $T_h = p_1 + p_2 = 5 + 4 = 9, T_l = p_3 + ... + p_7 = 8 + 6 + 5 + 4 + 1 = 24, T = T_h + T_l = 33,$ and $\lceil T/m \rceil = \lceil 33/3 \rceil = 11$. So we can obtain a lower bound $\underline{C}_{\max} = \max\{8, 9, 11\} = 11$.

By applying Theorems 1 and 2, we can now develop an algorithm, named Algorithm 1, to solve the problem. Algorithm 1 is based on an algorithm proposed by Lin and Liao [9], but it is more efficient. The basic idea of Algorithm 1 is to check whether a specific completion time C, starting from $\underline{C}_{\max}$, can be achieved. The procedure focuses on only one machine and considers its assignable interval $(0, C)$. We assign jobs (workload) into $(0, C)$, and then allocate the rest of jobs to the remaining machines, which is again a parallel machine problem and can be solved by Algorithm 1. If the optimal solution of the embedded parallel machine problem has a makespan equal to C, the optimal solution is found; otherwise, we change the job assignment in $(0, C)$ and repeat the procedure. When all job assignments in $(0, C)$ have been tried by lexicographic search, we relax the assignable interval by setting $C = C + 1$ and redo the algorithm.

Table 1. Processing Times for Example 1

Job	1	2	3	4	5	6	7
Level	*h*	*h*	*l*	*l*	*l*	*l*	*l*
Processing time	5	4	8	6	5	4	1

h = high, *l* = low.

The Steps of Algorithm 1 are given as follows.

Algorithm 1

Step 1: Let job c be a composite job with processing time T_h. Arrange all the jobs in the LPT order. Set the upper bound $\overline{C}_{\max} = \infty$. Let $\underline{C}_{\max} = \max\{p_l, T_h, \lceil T/m \rceil\}$, $C = \underline{C}_{\max}$, $\underline{a} = T-(m-1)C$.

Step 2: Assign jobs, in order $(1,...,n)$, into the interval $(0,C)$ until the assigned workload equals to C (if possible). If job k cannot be put into the remaining interval, proceed with the next job (job $k+1$). Let π denote the resulting job sequence and a denote the associated workload.

Step 3: If $a < \underline{a}$, go to Step 5. Otherwise, assign the rest of jobs $(\notin \pi)$ to the remaining machines by applying this algorithm inside the loop. Let $C'_{\max}$ be the optimal makespan of the embedded subproblem.

Step 4: If $C'_{\max} < \overline{C}_{\max}$, update $\overline{C}_{\max}$. If $\overline{C}_{\max} = C$, the optimal makespan is C; stop.

Step 5: Apply lexicographic search to obtain a new job sequence π that is assigned into $(0,C)$. If job 1 is not in π, set $C = C+1$, update $\underline{a}$, and return to Step 2. Otherwise, return to Step 3.

We now elaborate the algorithm. In Step 1, we set the prescribed makespan $C = \underline{C}_{\max}$ and compute the lower bound on the assigned workload $\underline{a}$ in the interval $(0,C)$. The lower bound $\underline{a}$ will be established in Theorem 3. In Step 2, we assign jobs into $(0,C)$ and obtain a sequence π together with an assigned workload a. If $a < \underline{a}$ (Step 3), the maximum completion time on the remaining machines must be greater than C. This indicates that the prescribed makespan C cannot be achieved. So we proceed immediately to change the job sequence π (Step 5) without further scheduling the remaining machines; this eliminates much unnecessary computation. In Step 4, we update the incumbent solution if necessary and check whether the prescribed makespan has been achieved. In Step 5, we change the jobs in π by using lexicographic search. As an illustration, suppose job k is the last assigned job in π. Then we replace job k with job $k+1$ in π. If the last assigned job is job n, it is removed and we consider the second to the last position. The procedure is continued until C has been achieved. When job 1 is removed from π, it implies that C cannot be achieved. In such a situation, we relax the prescribed makespan by setting $C = C+1$ and redo the algorithm.

Theorem 3. In Algorithm 1, a lower bound on the assigned workload in the interval $(0,C)$ is given by

$$\underline{a} = T-(m-1)C$$

Proof. Assume that the prescribed makespan C can be achieved. Then the total processing requirement T can be assigned to the total capacity of machines $m \times C$, and the sum of the gaps on all the machines equals $(mC-T)$. Hence, the gap between C and a (the assigned workload) cannot be larger than $(mC-T)$, or mathematically

$$C-a \le mC-T$$

or $a \ge T-(m-1)C$

Therefore, a lower bound on a is $\underline{a} = T-(m-1)C$.

Example 2. As an illustration of Algorithm 1, consider the same numerical example as in Example 1. Application of Algorithm 1 results in the following steps:

Step 1. Let job c denote a composite job with $T_h = 9$. The LPT sequence is $(c,3,4,5,6,7)$. Let $\overline{C}_{\max} = \infty$, $C = \underline{C}_{\max} = 11$, and $\underline{a} = T-(m-1)C$ $= 33-(3-1)\times 11 = 11$.

Step 2. Assign $\pi = (c,7)$ with $a = 10$ into interval $(0,C)$.

Step 3. Since $a = 10 < \underline{a} = 11$, go to Step 5.

Step 5. By lexicographic search, we obtain a new job sequence $\pi = (c)$ with $a = 9$. Since job 1 is still in π, return to Step 3.

Step 3. Since $a = 9 < \underline{a} = 11$, go to Step 5.

Step 5. By lexicographic search, we obtain a new sequence $\pi = (3,7)$. Since job 1 is not in π, set $C = C+1 = 12$ and $\underline{a} = 33-(3-1)\times 12 = 9$. Return to Step 2.

Step 2. Assign $\pi = (c,7)$ with $a = 10$ into $(0,C)$.

Step 3. Since $a = 10 > \underline{a} = 9$, the rest of jobs $(3,4,5,6)$ are assigned to the remaining two machines by performing the algorithm with two machines. We obtain $C'_{\max} = 12$ along with the job sets $(3,6)$ and $(4,5)$ assigned to the two machines.

Step 4. Since $C'_{\max} = 12 < \overline{C}_{\max}$, we update $\overline{C}_{\max} = 12$. Since $\overline{C}_{\max} = C$, the optimal makespan is 12 and the algorithm is stopped.

The optimal schedule is to assign job set $(1,2,7)$ to the single high-level machine and job sets $(3,6)$ and $(4,5)$ to the remaining two low-level machines.

TWO HIGH-LEVEL MACHINE PROBLEM

In this section, we consider the problem with two high-level machines, i.e., $m_h = 2$. Let p_h be the largest processing time of all high-level jobs. In the next theorem, we establish a lower bound on the makespan for a problem with two or more high-level machines.

Theorem 4. For the problem with $m_h \ge 2$, a lower bound on the makespan is given by

$$\underline{C}_{\max} = \max\{p_h, p_l, \lceil T_h/m_h \rceil, \lceil T/m \rceil\}.$$

Proof. All the processing requirement T_h has to be processed by the m_h high-level machines, and hence an obvious lower bound is $\max\{\lceil T_h / m_h \rceil, p_h\}$. Now consider the following two cases:

(i) $T_h / m_h \geq T_l / m_l$: All T_l should be processed on the m_l low-level machines, so a lower bound is $\max\{\lceil T_l / m_l \rceil, p_l\}$. Since $T_h / m_h \geq T_l / m_l$, the lower bound becomes $\max\{\lceil T_h / m_h \rceil, p_l\}$.

(ii) $T_h / m_h < T_l / m_l$: In this case, the m_h high-level machines may process part of T_l, so a lower bound is $\lceil T / m \rceil$.

Combining all the results, a lower bound can be established as $\underline{C}_{\max} = \max\{p_h, p_l, \lceil T_h / m_h \rceil, \lceil T / m \rceil\}$.

Example 3. As an illustration of Theorem 4, consider a four-machine $(m_h = 2, m_l = 2)$, nine-job problem with the data given in Table **2**. We can compute $T_h = 22$, $T_l = 23$, $\lceil T_h / m_h \rceil = 11$, and $\lceil T / m \rceil = 12$. Hence $\underline{C}_{\max} = \max\{8, 9, 11, 12\} = 12$.

Before presenting the steps of the algorithm for $m_h = 2$, we briefly explain the basic idea of the algorithm. To begin with, we set a prescribed makespan $C = \underline{C}_{\max}$, where $\underline{C}_{\max}$ is computed according to Theorem 4. We focus on one of the two high-level machines and assign some high-level jobs (along with some low-level jobs if necessary) into $(0, C)$. The remaining problem is treated as a single high-level machine problem that can be solved by Algorithm 1. If C cannot be achieved, we change the job combinations in $(0, C)$ by lexicographic search. When all job combinations in $(0, C)$ have been tried, we relax the prescribed makespan by setting $C = C + 1$ and redo the algorithm.

The Steps of Algorithm 2 are given as follows.

Algorithm 2

Step 1: Arrange all the high-level (low-level) jobs in LPT such that $p_1 \geq \cdots \geq p_{n_h}$ $(p_{n_h+1} \geq \cdots \geq p_n)$. Let $\underline{C}_{\max} = \max\{p_h, p_l, \lceil T_h / 2 \rceil, \lceil T / m \rceil\}$, $C = \underline{C}_{\max}$, $\underline{a}_h = T_h - C$, $\underline{a} = T - (m-1)C$, $\overline{C}_{\max} = \infty$, and $A(k) = 0$ for $k = 1, ..., C$.

Step 2: Assign high-level jobs, in order $(1, ..., n_h)$, into the interval $(0, C)$ until the assigned workload equals to C (if possible). If job k cannot be put into the remaining interval, proceed with the next job (job $k+1$). Let π_h denote the resulting job sequence and a_h denote the associated workload.

Step 3: If $a_h < \underline{a}_h$ or $A(a_h) = 1$, go to Step 8. Otherwise, set $A(a_h) = 1$ and $A(T_h - a_h) = 1$. If $a_h = C$, set $\pi_l = \phi$, $a_l = 0$ and go to Step 5.

Step 4: Assign low-level jobs, in order $(n_h + 1, ..., n)$, into the remaining interval (a_h, C) until the assigned workload equals $C - a_h$ (if possible). Let π_l denote the resulting job sequence and a_l denote the associated workload.

Step 5: If $a_h + a_l \geq \underline{a}$, apply Algorithm 1 to assign the rest of jobs into the remaining machines. Let $C'_{\max}$ be the optimal makespan of the embedded subproblem.

Step 6: If $C'_{\max} < \overline{C}_{\max}$, update $\overline{C}_{\max}$. If $\overline{C}_{\max} = C$, the optimal makespan is C; stop. If $a_h = C$, go to Step 8.

Step 7: Apply lexicographic search to obtain a new π_l that is assigned into (a_h, C). If there exist jobs in π_l, return to Step 5.

Step 8: Apply lexicographic search to obtain a new π_h that is assigned into $(0, C)$. If job 1 is not in π_h, set $C = C + 1$, $A(k) = 0$ for $k = 1, ..., C$, update $\underline{a}_h$ and $\underline{a}$, and return to Step 2. Otherwise, return to Step 3.

Table 2. Processing Times for Example 3

Job	1	2	3	4	5	6	7	8	9
Level	*h*	*h*	*h*	*h*	*l*	*l*	*l*	*l*	*l*
Processing time	8	6	5	3	7	6	5	3	2

h =high, *l*=low.

We now explain the algorithm. In Step 1, a lower bound on the assigned workload of high-level jobs is established as $\underline{a}_h = T_h - C$, which is simply an application of Theorem 3 for $m = 2$. In Step 2, we assign high-level jobs into $(0, C)$ and obtain a sequence π_h together with an assigned workload a_h. The purpose of Step 3 is to avoid repeating the same procedure. Although the number of high-level job combinations in $(0, C)$ is quite large, the number of different workload values is relatively small. To achieve this purpose, we use $A(k)$ as an indicator to identify whether a workload k has been tried. The indicator $A(k) = 1$ if the workload k has been tried, and $A(k) = 0$ otherwise. In Step 4, we continue assigning low-level jobs into the remaining interval (a_h, C) and obtain a sequence π_l together with an assigned workload a_l. In the next step, if $a_h + a_l \geq \underline{a}$, it implies that the prescribed makespan may be achieved, so we apply Algorithm 1 to assign the rest of jobs to the remaining machines. Otherwise, we need to change the jobs in π_l by lexicographic search (Step 7). In Step 6, we update the incumbent solution if necessary and check whether the prescribed makespan has been achieved. If π_l is empty, we need to change the jobs in π_h. When job 1 is removed from π_h, it implies that the prescribed makespan C cannot be achieved, so we proceed with $C+1$ and redo the algorithm.

Example 4. As an illustration of Algorithm 2, consider the same numerical example as in Example 3. Application of Algorithm 2 results in the following steps:

Step 1. The LPT sequences are $(1,2,3,4)$ for high-level jobs and $(5,6,7,8,9)$ for low-level jobs. Let $\overline{C}_{\max}=\infty, C=\underline{C}_{\max}=12, \underline{a}_h=T_h-C=10,$ and $\underline{a}=T-(m-1)C=9.$

Step 2. Assign $\pi_h=(1,4)$ with $a_h=11$ into interval $(0,C)$.

Step 3. Since $a_h=11\ge \underline{a}_h=10$ and $A(11)=0,$ set $A(11)=1.$

Step 4. Since no low-level jobs can be assigned into the remaining interval (a_h,C), $\pi_l=\phi$ and $a_l=0$.

Step 5. Since $a_h+a_l=11\ge \underline{a}=9,$ apply Algorithm 1 to assign jobs 2, 3, 5, 6, 7, 8, 9 to the remaining three machines. We obtain $C'_{\max}=12$ along with the job sets (2, 3), (5, 8, 9) and (6,7).

Step 6. Since $C'_{\max}=12<\overline{C}_{\max}$, we update $\overline{C}_{\max}=12.$ Since $\overline{C}_{\max}=C,$ the optimal makespan is 12 and the algorithm is stopped.

The optimal schedule is to assign job sets (1, 4) and (2, 3) to the two high-level machines and job sets (5, 8, 9) and (6,7) to the two low-level machines.

MULTIPLE HIGH-LEVEL MACHINE PROBLEM

In the same manner, we can develop an algorithm for the problem with three or more high-level machines. Note that a lower bound on the makespan has been given in Theorem 4. Also, recall that in Algorithm 2 we focus on one of the two high-level machines and treat the remaining problem as a single high-level machine problem, which can be solved by Algorithm 1. Similarly, for the m_h high-level machine problem, we still focus on one high-level machine and treat the remaining problem as an (m_h-1) high-level machine problem, which can be solved by the associated algorithm. For notational convenience, the algorithm for solving the problem with m_h high-level machines is named Algorithm m_h. Algorithm m_h is similar to Algorithm 2 except the following two steps.

Algorithm m_h

Step 1: $\ldots \underline{C}_{\max}=\max\{p_h, p_l, \lceil T_h/m_h\rceil, \lceil T/m\rceil\}, \ldots$

Step 5: If $a_h+a_l\ge \underline{a}$, apply Algorithm (m_h-1) to assign the rest of jobs into the remaining machines. …

COMPUTATIONAL RESULTS

The computational experiments consist of two parts. In the first part, we justify the use of Theorem 3 in Algorithm 1. In the second part, we evaluate the developed Algorithms 1 and 2 and access the performance of the LG-LPT heuristic of Hwang *et al.* [3]. All the algorithms were coded in Visual Basic and run on a PC with Pentium 3.0 G CPU.

Consider the first part of the experiments, where Algorithm 1 was implemented with and without the use of Theorem 3. Various job-sized problems with five machines $(m_h=1, m_l=4)$ were solved. The processing times were randomly generated from a discrete uniform distribution [1, 500]. The results are summarized in Table **3**, which gives the average of 100 replications. It can be observed that without the use of Theorem 3 many problems cannot be solved within the 600-second limit, where the number of unsolved problems is given in parentheses. For small- and medium-sized problems the number of unsolvable problems is increased as the job number increases. However, the problem becomes easier for large-sized problems because the enormous job combinations may easily match the specified intervals to attain optimality. By comparing the results in the two columns, it shows clearly that the use of Theorem 3 in Algorithm 1 can improve the algorithm significantly.

In the second part of the experiments, we evaluate both the efficiency of the developed algorithm and the effectiveness of the LG-LPT heuristic proposed by Hwang *et al.* [3]. In the experiment, the processing times of jobs were randomly generated from a discrete uniform distribution $DU(1,b)$ with $b=25$, 50, 100, and 500. Problems were generated with number of machines $m=3,4,5$ and number of jobs $n=10$, 15, 20, 30, 50, 100, 500, 1000. The numbers of high-level machines and jobs were set as $m_h=1,2$ $(m_l\ge m_h\ge 1)$ and $n_h=\lfloor (m_h/m)\times n\rfloor$. For example, in a problem with $m=4$, $m_h=1$ and $n=30$, we have $n_h=8$. The combinations of the three factors give a total of 160 sets of problems. For each problem set, 100 replications are made. Hence, we report the results of the total 16,000 problems solved.

Table 3. Average Computation Times (in Seconds) by Algorithm 1 with or without Theorem 3

n	Algorithm 1 without Theorem 3	Algorithm 1 (with Theorem 3)
10	0.0125	0.0084
15	0.3406	0.0845
20	5.6136	0.0153
30	$0.6337^{(28)}$	0.0292
50	$0.0000^{(46)}$	0.0016
100	$0.0002^{(36)}$	0.0017
500	$0.0080^{(8)}$	0.0097
1000	$0.0249^{(4)}$	0.0267

The number in parentheses at the superscript represents the number of instances (out of 100) taking more than 600 seconds. The average computation time is computed excluding these instances.

Table 4. Average Computation Time (in 10^{-3} Seconds) for Algorithm 1 $(m_h = 1)$

n	***m* = 3**				***m* = 4**				***m* = 5**			
	(1,25)	**(1,50)**	**(1,100)**	**(1,500)**	**(1,25)**	**(1,50)**	**(1,100)**	**(1,500)**	**(1,25)**	**(1,50)**	**(1,100)**	**(1,500)**
10	0.3	0.2	0.5	2.3	0.8	1.7	5.6	18.1	1.3	2.0	2.5	8.4
15	0.2	0.3	0.3	1.1	0.5	3.8	1.7	32.7	2.0	13.9	63.8	84.5
20	0.5	0.5	0.5	2.2	0.6	0.2	0.5	1.3	0.2	0.6	1.4	15.3
30	0.6	0.5	0.2	0.5	0.2	0.3	0.5	1.3	0.5	0.5	0.6	29.2
50	0.6	0.5	0.6	0.3	0.5	0.6	0.8	0.9	0.8	0.8	0.9	1.6
100	0.6	0.3	0.8	0.8	1.3	1.3	0.6	2.0	0.8	0.8	1.1	1.7
500	7.3	7.5	8.1	8.1	8.1	7.3	8.1	8.0	7.7	8.1	8.1	9.7
1000	28	25.6	26.6	27.5	25.3	25.6	25.5	26.1	24.4	24.8	24.4	26.7

Tables **4** and **5** give the average computation time for each problem solved by Algorithms 1 and 2, respectively. Examining these tables, we observe that the algorithms appear to perform rather efficiently in deriving the optimal solutions, although they have exponential time complexities. In general, the computation time increases as the number of machines or the range of processing times increases. However, as stated earlier the computation time may not always increase as the number of jobs increases because it is easier to match the specified interval when there are more jobs.

Table **6** gives the mean percentage deviation (MPD) from optimum and the number of optimal solutions (No. Opt.) obtained by the LG-LPT heuristic in each set of 100 problem instances. The processing times were generated from $DU(1,500)$. It is observed that the MPD tends to decrease as the number of jobs and the number of machines increase because the problems become easier. For the same number of machines, problems with few high-level machines have smaller MPD. The results for the number of optimal solutions are similar to those of mean percentage deviations.

Table 5. Average Computation Time (in 10^{-3} seconds) for Algorithm 2 $(m_h = 2)$

n	***m* = 4**				***m* = 5**			
	(1,25)	**(1,50)**	**(1,100)**	**(1,500)**	**(1,25)**	**(1,50)**	**(1,100)**	**(1,500)**
10	0.8	1.3	4.8	24.8	2.3	2.0	5.0	27.2
15	1.4	5.3	8.4	101.1	5.3	6.4	69.2	125.8
20	0.6	1.9	8.1	119.2	0.9	7.3	30	206.9
30	0.2	0.3	0.9	59.1	0.5	0.2	0.6	29.5
50	0.6	0.3	0.3	5.2	0.5	1.1	1.3	3.1
100	0.5	0.8	0.9	2.8	0.6	1.3	1.1	2.2
500	8.3	8.4	8.1	8.6	8.4	8.3	8.1	8.9
1000	27.2	27	28.6	27.8	25.8	25.8	24.4	26.7

CONCLUSIONS

Although the parallel machine problem has attracted much attention, the studies on the practical problem with

Table 6. Mean Percentage Deviation and Number of Optimal Solutions Obtained by the LG-LPT Heuristic

n	***m***									
	$m_h = 1$						**$m_h = 2$**			
	***m* = 3**		***m* = 4**		***m* = 5**		***m* = 4**		***m* = 5**	
	MPD	**No. Opt.**	**MPD**	**No. Opt.**	**MPD**	**No. Opt.**	**MPD**	**No. Opt.**	**MPD**	**No. Opt.**
10	1.45	54	1.46	64	0.18	96	3.92	41	1.84	76
15	1.39	48	2.57	26	2.07	48	3.89	14	2.69	26
20	0.91	29	1.33	51	2.31	34	2.39	5	2.84	6
30	0.34	51	0.67	42	0.76	57	1.31	1	1.74	2
50	0.12	38	0.25	41	0.31	52	0.40	7	0.53	7
100	0.03	59	0.05	59	0.10	47	0.10	12	0.17	7
500	0.00	70	0.00	75	0.00	69	0.00	54	0.00	36
1000	0.00	88	0.00	86	0.00	80	0.00	79	0.00	73

machine eligibility restrictions are relatively few. Due to the complexity of the problem, most previous research has focused on the problem with unit-length jobs, and the limited research considering general jobs (non-unit-length jobs) has mainly aimed at developing heuristic algorithms. In this paper, we have proposed an algorithm that can be used to solve the problem with general jobs to optimality. The algorithm has employed some powerful properties, so that it can derive the optimal solutions for various-sized problems in a short time. The algorithm has also been used to evaluate the existing heuristic for the problem. Further research is needed to develop solution methods for other parallel machine problems with machine eligibility restrictions, such as non-nested machine eligibility or multi-level system.

REFERENCES

[1] Centeno G, Armacost RL. Minimizing makespan on parallel machines with release time and machine eligibility restrictions. Int J Prod Res 2004; 42: 1243-56.

[2] Centeno G, Armacost RL. Parallel machines scheduling with release time and machine eligibility restrictions. Comput Ind Eng 1997; 33: 273-6.

[3] Hwang HC, Chang SY, Lee K. Parallel machine scheduling under a grade of service provision. Comput Oper Res 2004; 31: 2055-61.

[4] Pinedo M. Scheduling: Theory, Algorithms, and Systems. 2nd ed. Prentice-Hall, Englewood Cliffs, NJ. 2002.

[5] Glass CA, Mills HR. Scheduling unit length jobs with parallel nested machine processing set restrictions. Comput Oper Res 2006; 33: 620-38.

[6] Lin Y, Li W. Parallel machine scheduling of machine-dependent jobs with unit-length. Eur J Oper Res 2004; 156: 261-6.

[7] Li CL. Scheduling unit-length jobs with machine eligibility restrictions. Eur J Oper Res 2006; 174: 1325-28.

[8] Liao CJ, Shyur DL, Lin CH. Makespan minimization for two parallel machines with an availability constraint. Eur J Oper Res 2005; 160: 445-56.

[9] Lin CH, Liao CJ. Makespan minimization subject to flowtime optimality on identical parallel machines. Comput Oper Res 2004; 31: 1655-66.

Received: December 31, 2007 Revised: February 20, 2008 Accepted: February 29, 2008