Preemptive scheduling on two parallel machines with a single server

Yiwei Jiang a,*, Jianming Dong a, Min Ji b

a Department of Mathematics, Zhejiang Sci-Tech University, Hangzhou 310018, China
b School of Computer Science and Information Engineering, Contemporary Business and Trade Research Center, Zhejiang Gongshang University, Hangzhou 310018, China

ABSTRACT

This paper addresses a preemptive scheduling problem on two parallel machines with a single server. Each job has to be loaded (setup) by the server before being processed on the machines. The preemption is allowed in this paper. The goal is to minimize the makespan. We first show that it is no of use to pre-empt the job during its setup time. Namely, every optimal preemptive schedule can be converted to another optimal schedule where all the setup times are non-preemptively performed on one machine. We then present an algorithm with a tight bound of 4/3 for the general case. Furthermore, we show that the algorithm can produce optimal schedules for two special cases: equal processing times and equal setup times, which are NP-hard in the non-preemptive version.

© 2013 Elsevier Ltd. All rights reserved.

1. Introduction

In this paper, we consider parallel machine scheduling problem with a single server, which is used to execute some pre-operational work before processing the jobs. One application of the problem was found in Flexible Manufacturing Systems, where a robot (Koulamas, 1996) or an automated guided vehicle (AGV) (Ganesharajah, Hall, & Sriskandarajah, 1998) is shared among several pieces of equipment for tool change and part setup purposes. In automated material handling system of cellular manufacturing, common servers are used to load (or unload) the materials between processing stations (Dawande, Geismar, Sethi, & Sriskandarajah, 2005).

We study the preemptive version of the scheduling problem. This problem can be described as follows. We are given a sequence $J = \{J_1, J_2, \ldots, J_n\}$ of independent jobs, which must be scheduled on two identical machines $M_1$ and $M_2$. Job $J_j$ has a setup time $s_j$ and a processing time $p_j$. Prior to processing, a job must be loaded (setup) by a single server onto the machines. The setup cannot occur while the machine is processing another job. Preemption of setup and processing is allowed, which means that the setup (processing) time slot of each job may be preempted into a few pieces. These pieces are to be assigned to possibly on distinct machines. But, the preemption must be fulfilled the following requirements.

(R1) The pieces of the same job should be assigned in non-overlapping time slots;
(R2) For each job, all pieces of setup should be performed before processing any piece of the job. The objective is to minimize the makespan, i.e., the completion time of the latest job. Using the three parameter notation (Kravchenko & Werner, 1997; Hall, Potts, & Sriskandarajah, 2000), our problem could be denoted by $P2, S1|pmt|C_{\text{max}}$.

There have been many literatures studying the non-preemptive parallel machines scheduling problem with servers. For the unit setup times problem $P2, S1|s_j = 1|C_{\text{max}}$, Kravchenko and Werner (1997) showed that it is binary NP-hard and proposed a pseudo-polynomial algorithm. For unit processing times problem $P2, S1|p_j = 1|C_{\text{max}}$, it was polynomial solvable by Hall et al. (2000). They also showed the equal setup times problem $P2, S1|s_j = s|C_{\text{max}}$ is strongly NP-hard. For the equal processing times problem $P2, S1|p_j = p|C_{\text{max}}$, it was shown to be NP-hard by Brucker, Dhaene-Flipo, Knust, Kravchenko, and Werner (2002). Abdeldahae and Wirth (2002) and Abdeldahae, Wirth, and Gan (2004, 2006) studied the computational complexity of some special and general cases and provided some effective heuristics for them.

For the other objectives of this problem, Hall et al. (2000) studied the computational complexity of the problem for the scheduling objectives such as the maximum lateness of any job and the total completion time of all jobs, as well as some polynomial or pseudo-polynomial algorithms for them. Wang and Cheng (2001) proposed an approximation algorithm for minimizing the total weighted completion time. Ou, Qi, and Lee (2010) and Werner and Kravchenko (2010) studied the scheduling with multiple servers.

Recently, Zhang and Andrew (2009) considered $LS$ algorithm for three special cases of the online scheduling problem on two machines with a single server, where jobs arrive over list. Su...
(2012) studied the online LPT algorithm for the online version where jobs arrive over time.

Noting that in all above problems, preemption is not allowed. In this paper, we consider the preemptive scheduling on two identical machines with a single server, denoted by P2, S1|pmtn|Cmax. That is, when the setup is performing and the job is processing on the machines, we can preempt the operations and resume afterward on the different machines. We first analyze the structures of optimal schedules of our problem. It can be shown that each optimal preemptive schedule can be converted to another one where all the setup times are non-preemptively performed on one machine. Upon this analysis, we present an algorithm with a tight bound of 4/3. Furthermore, we show that the algorithm can produce optimal schedules for two special cases: equal processing times and equal setup times, denoted by P2, S1|pmtn, p1 = P|Cmax and P2, S1|pmtn, Sj = S|Cmax respectively. Note that both two problems are NP-hard in non-preemptive version.

The rest of the paper is organized as follows. In Section 2, we analyze the structures of optimal schedules. In Section 3, we present the algorithm for our problems and analyze its worst case ratio and optimality. Finally, Section 4 presents some concluding remarks.

2. The structures of the optimal preemptive schedules

In this section, we mainly analyze the structures of the optimal preemptive schedules to offer facility for designing algorithms afterward. Before then, we first introduce some notations required in the following of this paper. Let s(I) and p(I) denote the total setup times and total processing times of all the jobs in set I. Especially, let S = s(I) = \sum_{i=1}^{n} s_i and P = p(I) = \sum_{i=1}^{n} p_i. Denote by C^A and C^C the makespan produced by algorithm A and the optimal makespan, respectively.

The lower bounds of the optimal makespan can be obtained as follows.

**Theorem 2.1.** The makespan of the optimal preemptive scheduling on two machines with a single server satisfies that

(i) \( C^C \geq \frac{1}{2}(\min_{1 \leq i \leq n} s_i + S + P); \)

(ii) For any subset \( I \subseteq J \), \( C^C \geq s(I) + \min_{j \in I} p_j. \)

**Proof.**

(i) Suppose that \( J_k \) is the first job loaded onto machine \( M_1 \) in the optimal scheduling. According to the definition of our problem, it is not hard to obtain that there must exist an inevitable idle time with length of \( s_k \) on machine \( M_2 \) during loading job \( J_k \) onto \( M_1 \). Together with the total setup times \( S \) and the total processing times \( P \), we can conclude that the makespan of the optimal scheduling is at least \( \frac{1}{2}(s_k + S + P) \geq \frac{1}{2}(\min_{1 \leq i \leq n} s_i + S + P). \)

(ii) As the problem defined that each job can be processed on the machines only after the completion of its setup, we can conclude that for any subset \( I \), there is at least one job which will be processed after all setups in \( I \) have been performed. It follows that \( C^C \geq s(I) + \min_{j \in I} p_j. \)

We now give our main result in this section. That is, we can find an optimal preemptive schedule with a good structure. Specifically, for any optimal preemptive schedule as shown in Fig. 1(i), by making some changes for the jobs, we can obtain another optimal schedule in which all the setups are continuously performed on the \( M_1 \) throughout the interval \( [0, S] \) and no preemptions of setups are needed, as shown in Fig. 1(ii).

In the following, we will present our method to generate the optimal schedule (ii) from (i) as shown in Fig. 1. In order to make it easier to understand, we describe each step of the method and give the proof of feasibility at the same time.

**Step 1.** Exchange all setup time slots on the machine \( M_2 \) with the processing time slots on \( M_1 \) in the same time interval. Without loss of generality, we consider the time interval \([t_1, t_2]\) in the given schedule as shown in Fig. 2. By Step 1, we exchange the time slots in the interval \([s_3, s_4]\) on the two different machines and thus the state (i) is changed to the state (ii). Clearly, the feasibility and optimality of the schedule remain unchanged after Step 1.

**Step 2.** Move all the setup time slots to the interval \([0, S]\). Still taking Fig. 2 for an example, we first exchange the setup time slot in the interval \([t_1, t_2]\) with the processing time slot in \([t_3, t_4]\) (see the state (iii) in Fig. 3). Noting that this exchange may result in the infeasibility of the schedule. Namely, there is possibly a preempted job, denoted by \( J_k \), and the time slots assigned to different portions of the job \( J_k \) denoted by \( p_{k1} \) and \( p_{k2} \) are overlapping as shown in Fig. 3(iii). In fact, it is easy to deal with this situation. Because the given schedule is obviously feasible, there must be no any portion of the job \( J_k \) on the machine \( M_2 \) when the portion \( p_{k2} \) is processing on \( M_1 \), as shown in Fig. 3(i). Therefore, we only need to exchange \( p_{k1} \) and \( p_{k2} \) and thus obtain the feasible schedule as shown in Fig. 3(ii). What needs to be stressed is that all the changes we have made do not violate the rules R1 and R2 defined in the introduction, because all the moved setups are performed without delay and all the moved jobs are not processed earlier.

By the above two steps, we can obtain that all setups are performed in the interval \([0, S]\). However, some setups are possibly preempted. The following step is to shift all preempted parts of the setup of the same job together.

**Step 3.** Shift all parts of the setup of the same job together without increasing the completion of the setup of any job. Specifically, we only need to put the parts of the setup of the same job together and assign them in the same order of completion time. There is an instance in Fig. 4, where \( s_1'(s_1', s_2') \) and \( s_2'(s_2', s_3') \) denote two different parts of the setup of the job \( J_1(J_2) \) and \( C(C_1, C_2) \) denote the completion time of the setup of the job \( J_1(J_2) \). It is easy to see that the completion time of each setup is non-increasing and thus it will not influence the jobs processing.

Now can obtain the following theorem.

**Theorem 2.2.** For the problem P2, S1|pmtn|Cmax, there must exist an optimal schedule in which each setup of the job is non-preemptively scheduled one by one on one machine in the interval \([0, S]\).

3. Algorithm

In this section, we will present our algorithm, denoted by A, which has a worst case ratio of 4/3 for the problem P2,
that the makespan is as early as possible. Denote this schedule by \( \delta_2 \).

4. Choose the smaller one of the makespans of \( \delta_1 \) and \( \delta_2 \).

Before giving the worst case ratio of the algorithm \( A \), we first analyze the makespans of the schedules \( \delta_1 \) and \( \delta_2 \) as mentioned in steps 3.1 and 3.2. Let \( C(\delta) \) denote the makespan of the schedule \( \delta_i \), \( i = 1, 2 \). The following Lemmas will give the makespans by considering all the possibilities of \( \delta_i \).

For convenience, let \( a = \sum_{i=1}^{n} s_i \) and \( b = \sum_{i=1}^{n} s_i \) as shown in Figs. 5 and 6.

**Lemma 3.1.** \( C(\delta_1) = \max \{ C^*, a + \frac{1}{2} (b + p(I)) \} \).

**Proof.** We consider the schedule of the set \( I \). Clearly, the makespan is at least \( a + \frac{1}{2} (b + p(I)) \). Consider the subset of jobs \( \{J_1, J_2, \ldots, J_l\} \), noting that \( p_1 \geq p_2 \geq \cdots \geq p_n \), we can conclude that the optimal makespan for this subset is at least \( s_1 + s_2 + \cdots + s_l + \min_{1 \leq i \leq l} p_i = a + p_I \) by \( (ii) \) in Theorem 2.1, i.e., \( C^* \geq a + p_I \). Analogously we can obtain that \( C^* \geq s_1 + s_2 + \cdots + s_l + s_{l+1} + p_{l+1} = a + s_{l+1} + p_{l+1} \). Hence, we have \( C^* \geq \max\{a + p_l + s_{l+1} + p_{l+1}\} \). Clearly, if \( a + \frac{1}{2} (b + p(I)) \leq \max\{a + p_l + a + s_{l+1} + p_{l+1}\} \), we can obtain that \( \delta_1 \) is an optimal schedule as shown in Fig. 6.

Now we focus on the case \( a + \frac{1}{2} (b + p(I)) > \max\{a + p_l + a + s_{l+1} + p_{l+1}\} \). We will show \( C(\delta_1) = a + \frac{1}{2} (b + p(I)) \). That means we can find a feasible schedule such that the completion times of the two machines are the same without introduce any idle time after \( a \).

Noting that the setups are fixed. Three cases are distinguished according to the sizes of \( p, p_I \) and \( p_{l+1} \) as illustrated in Fig. 7. Noting that \( c = \frac{1}{2} (p(I) - b) \), we have \( a + b + c > \max\{a + p_l + a + s_{l+1} + p_{l+1}\} \) in the following cases.

**Case 1.** \( \tilde{p} + p_l + p_{l+1} \leq \frac{1}{2} (b + p(I)) = b + c \). An optimal schedule of \( I \) can be obtained as illustrated in Fig. 7(i). For the feasibility, we only need to show the two parts of the preempted job \( J_k \) are not overlapping. Actually, it suffices to show that \( p_k \leq c \). By the assumption and the definition of \( J_k \), we obtain that \( \tilde{p} + p_l + p_{l+1} \leq b + c \) and \( \tilde{p} + p_I > b \), which follows that \( p_{l+1} < c \). Hence, we have \( p_k \leq p_{l+1} < c \) by the algorithm rule.

**Case 2.** \( \tilde{p} + p_l + p_{l+1} > b + c \) and \( p_{l+1} < c \). An optimal schedule can be obtained as illustrated in Fig. 7(ii). Clearly, the schedule of the job \( J_k \) is feasible. We next show the schedule of \( p_{l+1} \) is valid, namely, the job is processed after the completion of its setup, i.e., \( a + s_{l+1} < a + b + c = p_{l+1} \). In fact, it is easy to obtained from the assumption of \( a + b + c > \max\{a + p_l + a + s_{l+1} + p_{l+1}\} \).

**Case 3.** \( \tilde{p} + p_l + p_{l+1} > b + c \) and \( p_{l+1} < c \). An optimal schedule can be obtained as illustrated in Fig. 7(iii). The feasibility of this schedule is obvious.
Hence, all above schedules are feasible and the desired result achieved. □

**Lemma 3.2.** $C(d_2) \leq \max\{a + 2b, a + b + p_1, (b + S + P)\}$.

**Proof.** By the rule of the algorithm in Step 3.2, we assign the jobs $J_{s+1}, \ldots, J_n$ to the idle times from time $b$ as early as possible. If all jobs can be processed before $a + b + \max\{p_i, \bar{p}\}$, then we obtain the schedules as shown in Fig. 8(i and ii) according to the sizes of $\bar{p}$ and $p_i$. Noting that $\bar{p} < a + b$ from Fig. 5, we have

$$C(d_2) \leq a + b + \max\{p_i, \bar{p}\} = \max\{a + 2b, a + b + p_1\}. \quad (1)$$

On the other hand, we can obtain the schedule as shown in Fig. 8(iii), where the completion time of both machines are same and no idle time introduce from time $b$. It follows that the makespan is $\frac{b + S + P}{2}$. The feasibility of the schedule can be achieved by the following reasons: (1) All jobs $J_{s+1}, \ldots, J_n$ are processed after their setups ($R1$) and (2) The makespan is greater than $a + b + p_1$ and $p_1$ is greater than $p_i$ for any $j > 1$ and thus all the jobs processed after $a + b$ can be scheduled without overlapping ($R2$). □

We now give the worst case ratio of the algorithm $A$ below.

**Theorem 3.1.** $\frac{C^A}{C} \leq \frac{4}{3}$

**Proof.** From Lemma 3.1 and Fig. 7, we have $C(d_1) = \max\{C', a + b + c\}$. Clearly, if $C(d_1) = C'$, we obtain the optimal schedule. Therefore, we assume that $C(d_1) = a + b + c$. It is easy to get that from Theorem 2.1

$$C' \geq \max\{\frac{a + 2b + 2c}{2}, a + b\}.$$ 

If $a \leq b + c$, we have

$$C^A \leq a + b + c = \frac{a + b + c}{2} \leq \frac{3}{3}.$$ 

If $a > b + c$ and $c \leq a$, we have

$$C^A \leq a + b + c = \frac{a + b + c}{2} \leq \frac{a + b}{3}.$$ 

If $a > b + c$ and $c > \frac{a}{2}$, then we have $a > 2b$ and consider the schedule $d_2$ as shown in Fig. 8. If the schedule $d_2$ is obtained from Fig. 8(i), then we have $C^d = a + 2b$ by (1). It follows that $\frac{C^A}{C} \leq \frac{a + 2b}{a + b} \leq \frac{3}{3}$, because $C' \geq a + b$ by (ii) of Theorem 2.1. For the schedule in Fig. 8(ii), we have $C^A = a + b + p_1$. Since $p_1$ is the smallest
processing time in the first \( l \) jobs and \( \sum_{i=1}^{l} s_i = a \), we can obtain 
\[ C^* \geq \max(a + p, a + b) \] from Theorem 2.1. It yields that
\[
\frac{a + b + p}{C^*} \leq \frac{a + 2b}{a + b + p} \quad \text{for} \quad p_1 < b, \\
\frac{a + b + p}{C^*} = \frac{a + 2b}{a + b + p} \quad \text{for} \quad p_1 \geq b.
\]

For the last case that the schedule obtained from Fig. 8(iii), we have 
\[ C^* = \frac{1}{2} (S + P) \] and 
\[ C^* = \frac{1}{2} (S + P). \] Noting that \( S = a + b \), we can obtain that
\[
\frac{a + b + p}{C^*} = \frac{a + 2b + P}{a + b + p} \quad \frac{a + 2b}{a + b + p} \quad \frac{a + b + P}{a + b + p} \quad \frac{a + b}{a + b + p} \quad \frac{a + b}{a + b + p} \quad \frac{a + b}{a + b + p}.
\]

Hence, we conclude that the worst case ratio of the algorithm \( A \) is \( 4/3 \).

Now we show the bound is tight. Let \( \epsilon \) be a sufficient small number and consider the following instance:

- The first job: \( s_1 = 1 - 2\epsilon \) and \( p_1 = 1/2 \).
- The second job: \( s_2 = 2\epsilon \) and \( p_2 = 1/2 - \epsilon \).
- A number of jobs with \( s_i = \epsilon \), \( p_i = 0 \) and the sum of all setup times is \( 1/2 \).

By the algorithm rule, in the schedule \( \delta_1 \), all setups are scheduled on \( M_1 \) from zero time in decreasing order, and \( p_1 \) and \( p_2 \) are scheduled on \( M_2 \) from time \( 1 - 2\epsilon \). It implies \( C(\delta_1) = 2 - 3\epsilon \). In the schedule \( \delta_2 \), all the setups and \( p_2 \) are assigned to \( M_1 \), thus we have \( C(\delta_2) = 2 - \epsilon \). Therefore,
\[
C^* = \min\{C(\delta_1), C(\delta_2)\} = 2 - 3\epsilon.
\]

In fact, based on \( \delta_1 \), exchange the first two setups and then \( p_2 \) can be processed on \( M_2 \) at time \( 2\epsilon \), which yields a schedule with makespan of \( 3/2 \). Hence
\[
C^* \geq \frac{2 - \epsilon}{3/2} \geq \frac{4}{3}
\]
with \( \epsilon \) tends to zero. \( \square \)

In the following, we will show that the schedule \( \delta_1 \) can always generate optimal schedules for both problems \( P_2, S_1|pmtn, p_j = p|C_{\text{max}} \) and \( P_2, S_1|pmtn, s_j = s|C_{\text{max}} \).

**Theorem 3.2.** For the problems \( P_2, S_1|pmtn, p_j = p|C_{\text{max}} \) and \( P_2, S_1|pmnt, s_j = s|C_{\text{max}} \), the schedule \( \delta_1 \) of the algorithm is optimal, i.e., \( C(\delta_1) = C^* \).

**Proof.** It is obvious that the schedule \( \delta_1 \) is optimal if it is obtained from Fig. 6. Then we only need to consider the schedule \( \delta_1 \) as illustrated in Fig. 7. Noting that in this case
\[
l < n,
\]
by the definition of \( p_1 \). By the Step 1 of the algorithm A, we can see that the setup times are non-increasing and the processing times are non-increasing for both problems \( P_2, S_1|pmtn, p_j = p|C_{\text{max}} \) and \( P_2, S_1|pmnt, s_j = s|C_{\text{max}} \). We claim that the makespan is equal to
\[
a + b + c = s_1 + \frac{1}{2} (S + P - s_1) = \frac{1}{2} (\min_{j \leq n} S + P)
\]
which follows the optimality of the schedule by (i) in Theorem 2.1. 

As \( s_1 \) is the smallest setup in both problems by the rule of the algorithm A, we only need to show that there is no idle time in the time interval \( [s_1, a] \) on the machine \( M_2 \). Suppose that the idle times exist. It implies that there must exist two jobs \( j_1 \) and \( j_1 \) such that \( p_1 \) is completed before the completion of \( s_{i+1} \) and \( s_{i+1} > p_i \).

Noting that the setup times are non-decreasing and the processing times are non-increasing, we can conclude that \( s_{i+1} \geq s_i > p_{i+1} \geq p_i \) for any \( i < j \leq n \). Therefore, all the jobs after \( j_1 \) can be processed right after the completion of its setup and there is an idle time between each pair of adjacent jobs. It yields that the makespan is \( \sum_{i=1}^{n} s_i + p_n = C^* \) by Theorem 2.1. Hence, we have \( l = n \) by the definition of \( j_1 \), a contradiction. \( \square \)

### 4. Conclusions

In this paper, we studied the preemptive scheduling on two identical machines with a single server, denoted by \( P_2, S_1|pmtn|C_{\text{max}} \). We presented an algorithm with tight bound of 4/3 for the general case and showed this algorithm can generate optimal schedules for two special cases: equal processing times and equal setup times, which are NP-hard in non-preemptive version.

The results in this paper suggest some problems deserving further study. An important and natural question is to give an algorithm to solve the problem \( P_2, S_1|pmtn|C_{\text{max}} \), or to show it is NP-hard. We think the former is more likely. It is also very interesting to extend the results to \( m \) machines case. In addition, it is worth studying the online version of the preemptive scheduling.

### Acknowledgments

Yiwei Jiang is supported by the National Natural Science Foundation of China (11001242 and 11071220), Jianming Dong is supported by Zhejiang Province Natural Science Foundation of China (LY13A010015) and Min Ji is supported in part by Zhejiang Provincial Natural Science Foundation of China (Y6100598), the National Basic Research Program of China (973 Program) (2012CB315804) and the key innovation team of Science Technology Department of Zhejiang Province (2010R50041).

### References


