



## Correspondence:

# An optimal algorithm for preemptive scheduling on non-simultaneously available uniform machines\*

Hao ZHOU<sup>†1</sup>, Liping CAO<sup>2</sup>, Qi WEI<sup>3</sup>, Zhenyu SHU<sup>4</sup>, Yiwei JIANG<sup>†‡2</sup>

<sup>1</sup>College of Basic Science, Zhejiang Shuren University, Hangzhou 310015, China

<sup>2</sup>School of Management and E-Business, Zhejiang Gongshang University, Hangzhou 310018, China

<sup>3</sup>College of International Economics and Trade, Ningbo University of Finance and Economics, Ningbo 315175, China

<sup>4</sup>School of Computer and Data Engineering, NingboTech University, Ningbo 315100, China

<sup>†</sup>E-mail: zhouhao@zjsru.edu.cn; ywjiang@zjgsu.edu.cn

Received Nov. 9, 2023; Revision accepted Mar. 29, 2024; Crosschecked Mar. 3, 2025

<https://doi.org/10.1631/FITEE.2300767>

We study preemptive scheduling on  $m$  uniform machines with non-simultaneous available times to minimize the makespan. Each machine has a different speed and a different available time. We first provide a lower bound on the optimal makespan of the problem by converting the real machines to virtual machines that guarantee a machine with an earlier available time having a greater speed at any time. Then, we provide an algorithm with time complexity of  $O(nm+m^2)$  to find an optimal schedule with, at most,  $\frac{1}{2}(m^2+3m)-2$  preemptions, where  $n$  is the number of jobs.

## 1 Introduction

In this paper, we consider the problem of preemptive scheduling on parallel machines with non-simultaneous machine-available times. Our goal is to

minimize the maximum completion time, i.e., the makespan. Unlike classical parallel-machine scheduling where all the machines are available simultaneously at time zero, the machines in our problem may not be available at time zero. Lee (1991) first considered such a scheduling model, which has many applications in real industry settings (Lee et al., 1997); e.g., machines may require different setup times or warm-up times before they start processing jobs due to preventive maintenance and adjustment requirements.

We formally introduce the problem under study as follows: There are  $n$  independent jobs  $\{J_1, J_2, \dots, J_n\}$  to be processed on  $m$  parallel machines  $\{M_1, M_2, \dots, M_m\}$ . Each job  $J_j$  ( $j \in \{1, 2, \dots, n\}$ ) has a processing time  $p_j$  ( $p_j > 0$ ). Each machine  $M_i$  ( $i \in \{1, 2, \dots, m\}$ ) has a speed  $s_i$  ( $s_i > 0$ ) and is available at time  $r_i$ , which means that  $M_i$  can only start processing jobs at time  $r_i$ . Without loss of generality, we assume that  $r_1 \leq r_2 \leq \dots \leq r_m$ . Note that the processing time of job  $J_j$  is  $\frac{p_j}{s_i}$  if it is processed on machine  $M_i$ . Job preemption is allowed. Using the three-field notation for scheduling problems, we denote our problem as  $Qm, r_i | \text{pmtn} | C_{\max}$ .

For the problem of non-simultaneous parallel-machine scheduling, almost all the existing studies focus on the case where preemption is not permitted.

<sup>‡</sup> Corresponding author

\*Project supported by the Zhejiang Provincial Natural Science Foundation of China (Nos. LZ23G010001 and LZ25F020012), the National Natural Science Foundation of China (Nos. 62172356 and 61872321), and the Ningbo Major Special Projects of the "Science and Technology Innovation 2025" (No. 2024Z122)

ORCID: Yiwei JIANG, <https://orcid.org/0000-0002-9779-0613>

© Zhejiang University Press 2025

Lee (1991) showed that the longest processing time (LPT) algorithm has a tight worst-case ratio of  $\frac{2}{3} - \frac{1}{2m}$  for the case with  $m$  identical machines, i.e.,  $Pm, r_i \| C_{\max}$ . The author also provided a modified LPT (MLPT) algorithm with a worst-case ratio of  $4/3$ . He (1998) further showed that LPT is asymptotically optimal for the problem and provided a tight parametric worst-case bound for LPT. Lee et al. (2000) pointed out that there is a situation in which some machines do not process any job, so the lower bound on the makespan provided by Lee (1991) may not be valid. However, the bounds  $\frac{2}{3} - \frac{1}{2m'}$  for LPT and  $4/3$  for MLPT still hold, where  $m'$  is the number of used machines. Chang and Hwang (1999) developed the MULTIFIT algorithm for the problem and showed that its worst-case ratio is not greater than  $9/7$ ; Hwang and Lim (2014) improved this ratio to the tight bound of  $24/19$ . For scheduling on uniform non-simultaneously available machines, i.e.,  $Qm, r_i \| C_{\max}$ , He (2000) showed that the worst-case ratio of LPT is at most  $5/3$ , and provided an improved algorithm with a worst-case ratio of  $6/5$  for the two-machine case. Grigoriu and Friesen (2017) tackled the problem using MULTIFIT and showed that the worst-case ratio of MULTIFIT is at most  $1.382$  on general  $m$  machines and  $\sqrt{6}/2$  on two machines. In addition, some different models for scheduling on non-simultaneously available machines have been studied. Wang et al. (2014, 2015) investigated the scheduling of deteriorating jobs with non-simultaneous machine-available times. Shen et al. (2013) considered parallel-machine scheduling with non-simultaneous machine-available times to minimize a cost function. Huo (2019) considered a model with machine availability to minimize the makespan, subject to a total completion time constraint. Considering different types of jobs and performance measures, Kaabi and Harrath (2019) studied different versions of the problem on parallel uniform machines with an unavailability period. Mahjoub et al. (2021) applied a list scheduling (LS) algorithm to solve the problem of scheduling on identical machines with an arbitrary number of unavailability periods. Recently, Santoro and Junqueira (2023) introduced a new scheduling model for unrelated parallel machines with availability and eligibility constraints and presented mixed-integer linear

programming (MILP) models for this problem with the objective of minimizing the makespan. For the problem of non-resumable jobs, Kurt and Çetinkaya (2024) developed an MILP model and a heuristic algorithm to solve the problem.

Another closely related problem is classical preemptive scheduling on parallel machines. For the identical-machine case, McNaughton (1959) found an optimal schedule with no more than  $m-1$  preemptions in linear time. For the uniform-machine case, Liu and Yang (1974) presented a lower bound on the optimal preemptive schedule and provided an algorithm that obtains an optimal schedule for the special case where the first machine has a speed of 1 and the others have a speed greater than 1. Horvath et al. (1977) adapted the critical path algorithm proposed by Muntz and Coffman (1970) to address the problem with general  $m$  uniform machines, and obtained an optimal preemptive schedule with at most  $n^2(m-1)$  preemptions with time complexity of  $O(mn^2)$ . Gonzalez and Sahni (1978) provided an improved algorithm that generates an optimal schedule with at most  $2(m-1)$  preemptions. In addition, Schmidt (1984) considered the problem of preemptive scheduling on  $m$  identical machines with different intervals of availability. The author presented an algorithm with time complexity of  $O(n + m \log m)$  and showed that the number of preemptions is proportional to the total number of processing intervals of all the machines. Błażewicz et al. (2003) considered the problem of scheduling on parallel identical machines available only in restricted intervals of time, called time windows. For the objective of minimizing the makespan, they presented an optimal preemptive schedule, which can be seen as a general case of our problem for the identical-machine case.

In this paper, we consider preemptive scheduling on non-simultaneously available parallel uniform machines. We first convert the real machines to virtual machines that guarantee a machine with an earlier available time having a greater speed at any time. Then, we provide a lower bound on the optimal makespan and propose a polynomial time solution algorithm that produces an optimal schedule. The number of preemptions of the algorithm is no more than  $\frac{1}{2}(m^2 + 3m) - 2$ .

## 2 Lower bound

In this section, we derive a lower bound on the optimal makespan of the problem under consideration. Let  $C_Q^*$  be the makespan of the optimal schedule for problem  $Qm, r_i | \text{pmtn} | C_{\max}$ . For the sake of simplicity, let  $p_j^{\max}$  be the  $j^{\text{th}}$  largest processing time among all the jobs, i.e.,  $p_1^{\max} \geq p_2^{\max} \geq \dots \geq p_n^{\max}$ .

Since each machine has a different speed and is available at a different time, it is not easy to find a lower bound on the optimal makespan directly. So, we first convert all the  $m$  uniform machines to  $m$  virtual machines such that, at any time, the earlier the available time of a virtual machine, the greater its speed.

Let  $V = \{V_1, V_2, \dots, V_m\}$  be a set of virtual machines, where  $V_i$  is available at time  $r_i$ . We still assume that  $r_1 \leq r_2 \leq \dots \leq r_m$ . Let  $v_{ji}$  ( $i \leq j$ ,  $1 \leq j \leq m$ ) represent the  $i^{\text{th}}$  greatest speed in the set  $\{s_1, s_2, \dots, s_j\}$ , where  $s_k$  is the speed of real machine  $M_k$ ,  $1 \leq k \leq j$ . Let  $v_i(t)$  be the speed of  $V_i$  at time  $t$ , which we define as follows:

$$v_i(t) = \begin{cases} 0, & 0 < t \leq r_i, \\ v_{ji}, & r_j < t \leq r_{j+1}, j = i, i+1, \dots, m-1, \\ v_m, & t > r_m. \end{cases}$$

Fig. 1 illustrates the conversion process and shows that, at any time  $t$ , the virtual machine  $V_i$  always has the  $i^{\text{th}}$  greatest speed among all the available machines, i.e.,  $v_1(t) \geq v_2(t) \geq \dots \geq v_m(t)$  at any time. In addition, it is easy to find that the total processed size on  $V_i$  from  $t_1$  to  $t_2$  is  $\int_{t_1}^{t_2} v_i(t) dt$ .

Now we consider a lower bound on the optimal makespan. Let  $w_Q = \max\{T_1, T_2, \dots, T_m\}$ ; here,  $T_k$  ( $1 \leq k \leq m-1$ ) is the solution of  $\sum_{i=1}^k \int_{r_i}^{T_k} v_i(t) dt = \sum_{i=1}^k p_i^{\max}$  and  $T_m$  is the solution of  $\sum_{i=1}^m \int_{r_i}^{T_m} v_i(t) dt = \sum_{i=1}^n p_i^{\max}$ .

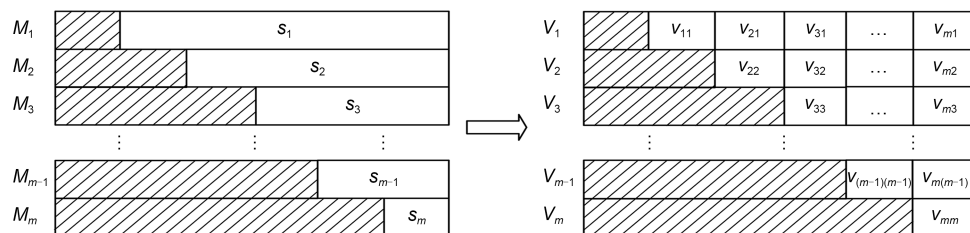


Fig. 1 Converting from real machines to virtual machines

We show that  $w_Q$  is a lower bound of the optimal makespan in the following lemma:

**Lemma 1** For problem  $Qm, r_i | \text{pmtn} | C_{\max}$ , we have  $C_Q^* \geq w_Q$ .

**Proof** Given  $v_1(t) \geq v_2(t) \geq \dots \geq v_m(t)$  and  $r_1 \leq r_2 \leq \dots \leq r_m$ , we derive  $C_Q^* \geq T_1$  because the completion time will be at least  $T_1$  by scheduling the job with the largest processing time  $p_1$  on the fastest machine  $V_1$ . Similarly, for the first  $k$  jobs ( $1 < k \leq m-1$ ), it is sufficient to use the first  $k$  machines and the completion time is at least  $T_k$  by the definition of  $T_k$ , so we have  $C_Q^* \geq T_k$ . Finally, after scheduling all the  $n$  jobs on the  $m$  virtual machines, we must have  $C_Q^* \geq T_m$  clearly.

## 3 An optimal solution algorithm

In this section, we provide a polynomial algorithm that can find an optimal schedule for  $Qm, r_i | \text{pmtn} | C_{\max}$  with at most  $\frac{1}{2}(m^2 + 3m) - 2$  preemptions. We denote by  $l_i$  the completion time of virtual machine  $V_i$  and refer to it as a “not full” virtual machine if  $l_i < w_Q$ . We denote by  $S_{\text{NFVM}} = \{V_{j1}, V_{j2}, \dots, V_{jq_{\text{NFVM}}}\}$  the set of all the “not full” virtual machines, where  $l_{j1} \leq l_{j2} \leq \dots \leq l_{jq_{\text{NFVM}}}$ . If  $l_i = w_Q$ , the virtual machine will be removed from  $S_{\text{NFVM}}$ . For simplicity, let  $q_{\text{NFVM}}$  denote the index of the last virtual machine in  $S_{\text{NFVM}}$ . Let  $W_i = \int_{l_i}^{w_Q} v_i(t) dt$  be the total size of the jobs processed on  $V_i$  from time  $l_i$  to  $w_Q$ .

We now present our algorithm (Algorithm 1) in the following. The main idea of the algorithm is to schedule the current job so as to make a virtual machine with as small a completion time as possible to be a full machine.

It is easy to derive that the time complexity of Algorithm 1 is  $O(mn)$ . Converting the uniform machines to virtual machines takes  $O(m^2)$  time. So, we

**Algorithm 1** Optimal schedule

---

// Step 1  
 Let  $l_i = r_i$  ( $i = 1, 2, \dots, m$ ),  $S_{\text{NFVM}} = \{V_1, V_2, \dots, V_m\}$ , and  $q_{\text{NFVM}} = m$ .

// Step 2  
 For any  $p_j, j = 1, 2, \dots, n$ .

// Step 3  
 If  $p_j \leq W_{q_{\text{NFVM}}}$   
   Schedule  $p_j$  on virtual machine  $V_{q_{\text{NFVM}}}$  as early as possible.  
   Find a time  $T$  such that  
     
$$\int_{l_{q_{\text{NFVM}}}}^T v_{q_{\text{NFVM}}}(t) dt = p_j.$$
  
   If  $T = w_Q$   
     Update  $S_{\text{NFVM}} = S_{\text{NFVM}} \setminus \{V_{q_{\text{NFVM}}}\}$  and  
      $q_{\text{NFVM}} = \max_{V_i \in S_{\text{NFVM}}} \{i\}; j = j + 1$ , go back to Step 2.  
   Else  
     Update  $l_{q_{\text{NFVM}}} = T$  and  $W_{q_{\text{NFVM}}} = W_{q_{\text{NFVM}}} - p_j$ ;  
      $j = j + 1$ , go back to Step 2.  
   End if

// Step 4  
 Else  
   Find two adjacent virtual machines  $V_k$  and  $V_h$  in  $S_{\text{NFVM}}$  such that  $W_k < p_j \leq W_h$ .  
   If  $\int_{l_h}^{l_k} v_h(t) dt + \int_{l_k}^{w_Q} v_k(t) dt \geq p_j$   
     Partition  $p_j$  into two parts  $p_j^{(1)}$  and  $p_j^{(2)}$  such that  
      $p_j^{(1)} = W_k$  and  $p_j^{(2)} = p_j - W_k$ .  
     Find a time  $T$  such that  $p_j^{(2)} = \int_{l_h}^T v_h(t) dt$ .  
     Schedule  $p_j^{(1)}$  on  $V_k$  in the time interval  $[l_k, w_Q)$  and  $p_j^{(2)}$  on  $V_h$  in  $[l_h, T)$ .  
     Update  $S_{\text{NFVM}} = S_{\text{NFVM}} \setminus \{V_k\}$ ,  $q_{\text{NFVM}} = \max_{V_i \in S_{\text{NFVM}}} \{i\}$ ,  
      $l_h = T$ , and  $W_h = W_h + W_k - p_j$ ;  $j = j + 1$ , go  
     back to Step 2.  
   // Step 5  
   Else  
     Find a time  $T$  such that  
     
$$\int_{l_h}^T v_h(t) dt + \int_T^{w_Q} v_k(t) dt = p_j.$$
  
     Partition  $p_j$  into two parts  $p_j^{(1)}$  and  $p_j^{(2)}$  such that  
      $p_j^{(1)} = \int_T^{w_Q} v_k(t) dt$  and  $p_j^{(2)} = p_j - p_j^{(1)} = \int_{l_h}^T v_h(t) dt$ .  
     Schedule  $p_j^{(1)}$  on  $V_k$  in the time interval  $[T, w_Q)$  and  $p_j^{(2)}$  on  $V_h$   
     in  $[l_h, T)$ . Change the time slot  $[l_k, T]$  of the two virtual  
     machines; we have a new virtual machine  $V_h$  with  $l_h = l_k$  and  
     a new “full” virtual machine  $V_k$ .  
     Update  $S_{\text{NFVM}} = S_{\text{NFVM}} \setminus \{V_k\}$ ,  $q_{\text{NFVM}} = \max_{V_i \in S_{\text{NFVM}}} \{i\}$ ,  
      $l_h = l_k$ , and  $W_h = W_h + W_k - p_j$ ;  $j = j + 1$ , go back  
     to Step 2.  
   End if  
 End if

---

can obtain an optimal schedule in  $O(mn + m^2)$  time. Fig. 2 shows the changes of machines before and after processing the jobs in Step 3, 4, or 5. We have the following observation about Algorithm 1:

**Observation 1** Supposing that  $S_{\text{NFVM}} = \{V_{j1}, V_{j2}, \dots, V_{jq_{\text{NFVM}}}\}$  after scheduling  $p_j$  ( $1 \leq j \leq n$ ), we have the following:

$$(i) l_{j1} \leq l_{j2} \leq \dots \leq l_{jq_{\text{NFVM}}}.$$

(ii) If  $p_j$  is scheduled in Step 4 or 5, the assignment is feasible and there is exactly one virtual machine to be removed in  $S_{\text{NFVM}}$ .

**Proof** (i) We prove it by induction. For  $j = 0$ , by the definition of  $S_{\text{NFVM}}$  and Step 1, the result holds obviously.

Suppose that the result holds for  $j - 1$ ,  $j \geq 1$ . We consider the assignment of  $p_j$ . It is clearly true if the job is processed on the last virtual machine in  $S_{\text{NFVM}}$  in Step 3. Therefore, we focus on the case where job  $J_j$  is processed in Step 4 or 5. In this case,  $p_j$  is partitioned into two parts with sizes  $p_j^{(1)}$  and  $p_j^{(2)}$ , which are scheduled on the virtual machine  $V_k$  and the adjacent virtual machine  $V_h$  in  $S_{\text{NFVM}}$ , respectively.

If job  $J_j$  is processed in Step 4, we have  $p_j^{(1)} = W_k = \int_{l_k}^{w_Q} v_k(t) dt$  and  $p_j^{(2)} = \int_{l_h}^T v_h(t) dt$ . By  $\int_{l_h}^{l_k} v_h(t) dt + \int_{l_k}^{w_Q} v_k(t) dt \geq p_j$ , we have

$$T \leq l_k. \quad (1)$$

It means that the completion time of  $V_h$  after processing  $J_j$  is not greater than that of  $V_k$  before processing  $J_j$ , as shown in Fig. 2b.

If job  $J_j$  is processed in Step 5, there is a time  $T$  such that  $\int_{l_h}^T v_h(t) dt + \int_T^{w_Q} v_k(t) dt = p_j$ . As shown in Fig. 2c, two parts of the job are respectively scheduled on two virtual machines  $V_k$  and  $V_h$  with processing times  $p_j^{(1)} = \int_T^{w_Q} v_k(t) dt$  and  $p_j^{(2)} = \int_{l_h}^T v_h(t) dt$ . By exchanging the time slot  $[l_k, T]$  of the two virtual machines, we have a new virtual machine  $V_h$  with  $l_h = l_k$ , and a new “full” virtual machine  $V_k$ .

By removing the new “full” machine  $V_k$ , and with the assumption that all the virtual machines in  $S_{\text{NFVM}}$  are in non-decreasing order at  $j - 1$ , we conclude that the result is still true at  $j$ .

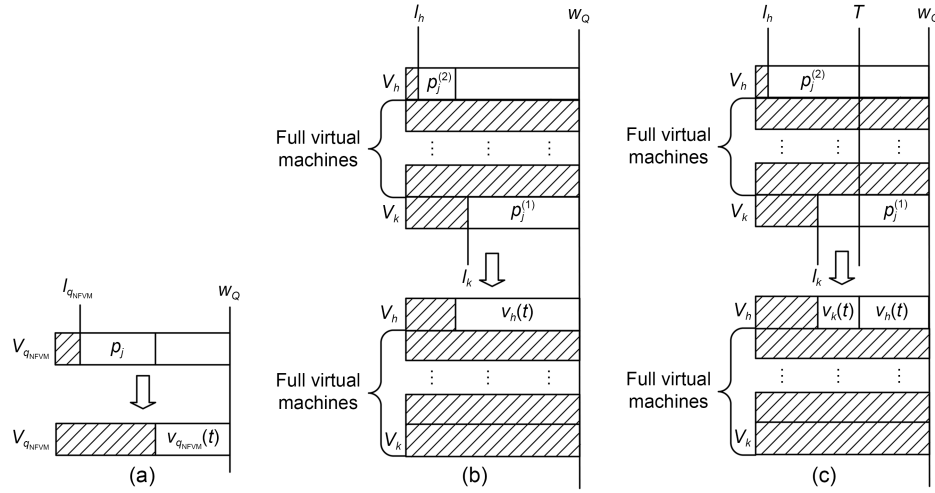


Fig. 2 Jobs scheduled in Step 3 (a), Step 4 (b), and Step 5 (c)

(ii) If  $p_j$  is scheduled in Step 4, we obtain that the time slots  $[l_k, w_Q)$  for  $p_j^{(1)}$  and  $[l_h, T)$  for  $p_j^{(2)}$  do not overlap by inequality (1), so the assignment of  $p_j$  is feasible. If  $p_j$  is scheduled in Step 5, it is easy to see that the time slots assigned to the two parts of  $p_j$  do not overlap. Clearly, there is a machine to be removed from  $S_{NFVM}$  in Step 4 or 5.

**Lemma 2** For any job  $J_j$ , we have  $p_j \leq W_e = \int_{l_e}^{w_Q} v_e(t) dt$ , where  $e = \min_{V_i \in S_{NFVM}} \{i\}$ .

**Proof** If  $e > 1$ , i.e., the machines  $V_1, V_2, \dots, V_{e-1}$  have been removed from  $S_{NFVM}$ , according to Observation 1(ii), once a job is processed in accordance with Step 4 or 5, it involves two machines, and the machine with the larger subscript becomes a full machine and is removed. Therefore, together with the definition of  $w_Q$ , the fact that the first  $e-1$  machines are removed indicates that the first  $e-1$  jobs with the largest processing time, i.e.,  $p_1^{\max}, p_2^{\max}, \dots, p_{e-1}^{\max}$ , are exactly processed on the first  $e-1$  virtual machines before scheduling  $p_j$ . Moreover, if  $l_e = r_e$ , i.e., there is no job on  $V_e$  before scheduling  $p_j$ , we must have  $p_j \leq W_e$ , since the first  $e$  largest jobs can be processed on the first  $e$  virtual machines by the definition of  $w_Q$ . So, we consider the case where  $l_e > r_e$ ; i.e., there are some jobs (or parts of the jobs) on  $V_e$  before scheduling  $p_j$ .

If there is only one virtual machine  $V_e$  in  $S_{NFVM}$ , i.e., all the other virtual machines have been removed with loads of  $w_Q$ , we must have  $p_j \leq W_e$ , as all the jobs can be processed on the  $m$  virtual machines before time  $w_Q$ .

Now, we focus on the case where there are at least two virtual machines in  $S_{NFVM}$ . It is easy to derive that all the jobs (or parts) on  $V_e$  must be scheduled in Step 4 or 5. Otherwise, if a job (or parts) on  $V_e$  is scheduled in Step 3, then  $V_e$  must be the only one in  $S_{NFVM}$ , which is a contradiction. By the rule of Step 4 or 5, we obtain that  $V_{e+1}$  must be removed. Let  $V_{k+2}$  be the current adjacent virtual machine of  $V_e$  in  $S_{NFVM}$  ( $e \leq k \leq m-2$ ); i.e., all the  $k-e+1$  virtual machines between  $V_e$  and  $V_{k+2}$  have been removed from  $S_{NFVM}$ . Because  $V_{k+2} \in S_{NFVM}$ , there is no job processed on both machines  $V_{k+2}$  and  $V_i$ , where  $1 \leq i \leq k+1$ . Moreover, by Steps 4 and 5 and Observation 1, we conclude that there are exactly  $k$  jobs with processing times  $p_{j1}, p_{j2}, \dots, p_{jk}$ , on the first  $k+1$  machines before scheduling  $p_j$ . By the definition of  $w_Q$ , we have  $\sum_{i=1}^{k+1} \int_{r_i}^{w_Q} v_i(t) dt \geq \sum_{i=1}^{k+1} \int_{r_i}^{T_{k+1}} v_i(t) dt = \sum_{i=1}^{k+1} p_i^{\max} \geq \sum_{i=1}^k p_{ji} + p_j$ , which yields  $p_j \leq \int_{l_e}^{w_Q} v_e(t) dt = W_e$ , given that  $\sum_{i=1}^k p_{ji} > \sum_{i=1}^{e-1} \int_{r_i}^{w_Q} v_i(t) dt + \sum_{i=e+1}^{k+1} \int_{r_i}^{w_Q} v_i(t) dt$ .

By Observation 1 and Lemmas 1 and 2, we obtain the following result:

**Theorem 1** Algorithm 1 generates an optimal schedule.

We now consider the number of preemptions. There are two types of preemptions in the algorithm. One is produced by partitioning a job into two parts and scheduling them on two different virtual machines as in Steps 4 and 5. The other is produced by the different speeds on the virtual machines. Clearly, there are at most  $m-1$  preemptions for the first type of preemptions.

For the second type, it suffices to calculate the number of speeds of each virtual machine. Initially, the number of speeds of virtual machine  $V_i$  is at most  $m+1-i$  after converting all the  $m$  uniform machines to  $m$  virtual machines. It follows that there are at most  $m-i$  preemptions on  $V_i$ , so the total number of preemptions is at most  $\sum_{i=1}^m (m-i) = \frac{m^2-m}{2}$ . Note that the speeds on the two virtual machines  $V_h$  and  $V_k$  are unchanged in Step 4, so we have no new preemptions. However, when a job is processed in Step 5, one more preemption is introduced because the time slot  $[l_k, T]$  of the new virtual machine  $V_h$  is from the virtual machine  $V_k$ . This produces a maximum of  $m-1$  preemptions. Hence, the total number of preemptions is at most  $\frac{m^2-m}{2} + 2(m-1) = \frac{m^2+3m}{2} - 2$ .

To illustrate Algorithm 1, we present an example as follows:

**Example 1** There are four jobs and three machines as shown in Table 1.

It is easy to find that  $w_Q=7$  for the above instance. Fig. 3 shows the virtual machines and the schedule

**Table 1** An example

| Symbol | Value | Symbol | Value | Symbol | Value |
|--------|-------|--------|-------|--------|-------|
| $p_1$  | 12    | $r_1$  | 1     | $s_1$  | 2     |
| $p_2$  | 11    | $r_2$  | 4     | $s_2$  | 3     |
| $p_3$  | 2     | $r_3$  | 5     | $s_3$  | 4     |
| $p_4$  | 4     |        |       |        |       |

$p$ : processing time of a job;  $r$ : starting available time of a machine;  $s$ : speed of a machine

of the jobs. The first job is assigned to  $V_1$  and  $V_2$  in Step 4, and  $V_2$  is removed from  $S_{\text{NFVM}}$ . The second job is assigned to  $V_1$  and  $V_3$  in Step 5, and the remaining time slots of  $V_1$  and  $V_3$  are spliced together into one time slot of the new virtual machine  $V_1$ . Finally, the third and fourth jobs are scheduled on  $V_1$  in Step 3.

## 4 Conclusions

In this paper, we studied preemptive scheduling on parallel uniform machines with non-simultaneous available times. Our objective was to minimize the makespan. We provided a lower bound on the optimal

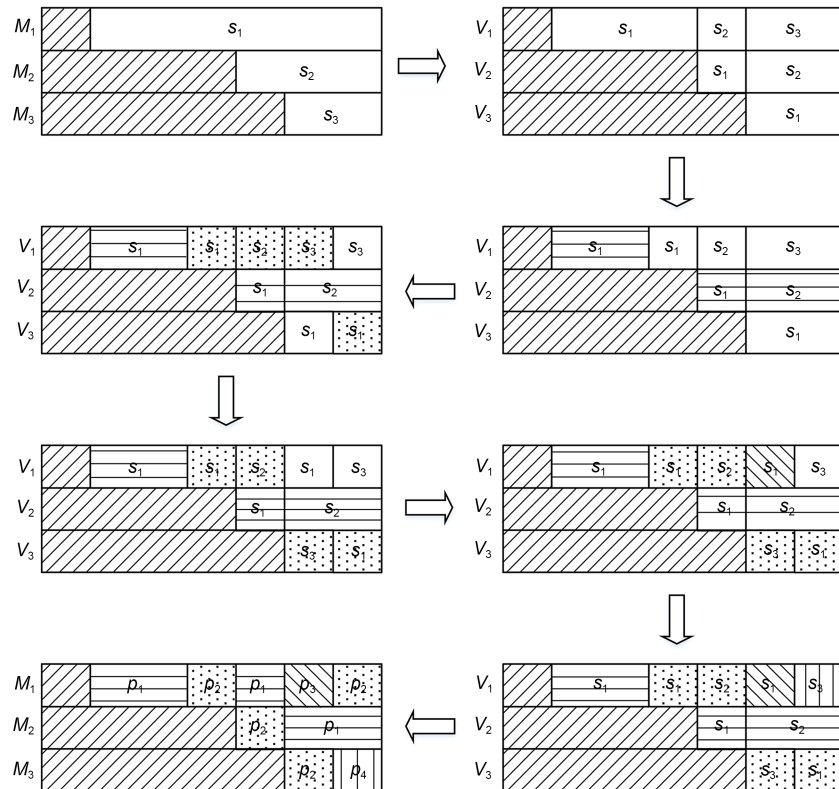


**Fig. 3** Scheduling process of Example 1

makespan by transforming the  $m$  uniform machines into  $m$  virtual machines, ensuring that a machine with an earlier available time has a greater speed at any time. We provided a solution algorithm for the problem. For future research, it is worth considering the online version of the problem for different parallel-machine settings, including identical machines and uniform machines.

## Contributors

Hao ZHOU designed the research. Liping CAO and Qi WEI designed and analyzed the algorithm. Hao ZHOU and Liping CAO drafted the paper. Zhenyu SHU helped organize the paper. Yiwei JIANG revised and finalized the paper.

## Conflict of interest

All the authors declare that they have no conflict of interest.

## References

- Błażewicz J, Dell’Olmo P, Drozdowski M, et al., 2003. Scheduling multiprocessor tasks on parallel processors with limited availability. *Eur J Oper Res*, 149(2):377-389. [https://doi.org/10.1016/S0377-2217\(02\)00760-9](https://doi.org/10.1016/S0377-2217(02)00760-9)
- Chang SY, Hwang HC, 1999. The worst-case analysis of the MULTIFIT algorithm for scheduling nonsimultaneous parallel machines. *Discr Appl Math*, 92(2-3):135-147. [https://doi.org/10.1016/S0166-218X\(99\)00049-9](https://doi.org/10.1016/S0166-218X(99)00049-9)
- Gonzalez T, Sahni S, 1978. Preemptive scheduling of uniform processor systems. *J ACM*, 25(1):92-101. <https://doi.org/10.1145/322047.322055>
- Grigoriu L, Friesen DK, 2017. Approximation for scheduling on uniform nonsimultaneous parallel machines. *J Sched*, 20(6):593-600. <https://doi.org/10.1007/s10951-016-0501-1>
- He Y, 1998. Parametric LPT-bound on parallel machine scheduling with non-simultaneous available time. *Asia-Pac J Oper Res*, 15(1):29-36.
- He Y, 2000. Uniform machine scheduling with machine available constraints. *Acta Math Appl Sin*, 16(2):122-129. <https://doi.org/10.1007/BF02677672>
- Horvath EC, Lam S, Sethi R, 1977. A level algorithm for preemptive scheduling. *J ACM*, 24(1):32-43. <https://doi.org/10.1145/321992.321995>
- Huo YM, 2019. Parallel machine makespan minimization subject to machine availability and total completion time constraints. *J Sched*, 22(4):433-447. <https://doi.org/10.1007/s10951-017-0551-z>
- Hwang HC, Lim K, 2014. Exact performance of MULTIFIT for nonsimultaneous machines. *Discr Appl Math*, 167:172-187. <https://doi.org/10.1016/j.dam.2013.10.022>
- Kaabi J, Harrath Y, 2019. Scheduling on uniform parallel machines with periodic unavailability constraints. *Int J Prod Res*, 57(1):216-227. <https://doi.org/10.1080/00207543.2018.1471242>
- Kurt A, Çetinkaya FC, 2024. Unrelated parallel machine scheduling under machine availability and eligibility constraints to minimize the makespan of non-resumable jobs. *Int J Ind Eng Manag*, 15(1):18-33. <https://doi.org/10.24867/IJIEEM-2024-1-345>
- Lee CY, 1991. Parallel machines scheduling with nonsimultaneous machine available time. *Discr Appl Math*, 30(1): 53-61. [https://doi.org/10.1016/0166-218X\(91\)90013-M](https://doi.org/10.1016/0166-218X(91)90013-M)
- Lee CY, Lei L, Pinedo M, 1997. Current trends in deterministic scheduling. *Ann Oper Res*, 70:1-41. <https://doi.org/10.1023/A:1018909801944>
- Lee CY, He Y, Tang GC, 2000. A note on “parallel machine scheduling with non-simultaneous machine available time.” *Discr Appl Math*, 100(1-2):133-135. [https://doi.org/10.1016/S0166-218X\(99\)00201-2](https://doi.org/10.1016/S0166-218X(99)00201-2)
- Liu JWS, Yang AT, 1974. Optimal scheduling of independent tasks on heterogeneous computing systems. *Proc ACM Annual Conf*, p.38-45. <https://doi.org/10.1145/800182.810377>
- Mahjoub A, Kaabi J, Harrath Y, 2021. Absolute bounds of list algorithms for parallel machines scheduling with unavailability periods. *Int Trans Oper Res*, 28(3):1594-1610. <https://doi.org/10.1111/itor.12589>
- McNaughton R, 1959. Scheduling with deadlines and loss functions. *Manag Sci*, 6(1):1-140. <https://doi.org/10.1287/mnsc.6.1.1>
- Muntz RR, Coffman EG, 1970. Preemptive scheduling of real-time tasks on multiprocessor systems. *J ACM*, 17(2):324-338. <https://doi.org/10.1145/321574.321586>
- Santoro MC, Junqueira L, 2023. Unrelated parallel machine scheduling models with machine availability and eligibility constraints. *Comput Ind Eng*, 179:109219. <https://doi.org/10.1016/j.cie.2023.109219>
- Schmidt G, 1984. Scheduling on semi-identical processors. *Z für Oper Res*, 28(5):153-162. <https://doi.org/10.1007/BF01920917>
- Shen LX, Wang D, Wang XY, 2013. Parallel-machine scheduling with non-simultaneous machine available time. *Appl Math Model*, 37(7):5227-5232. <https://doi.org/10.1016/j.apm.2012.09.053>
- Wang XY, Zhou ZL, Ji P, et al., 2014. Parallel machines scheduling with simple linear job deterioration and non-simultaneous machine available times. *Comput Ind Eng*, 74:88-91. <https://doi.org/10.1016/j.cie.2014.05.003>
- Wang XY, Hu XP, Liu WG, 2015. Scheduling with deteriorating jobs and non-simultaneous machine available times. *Asia-Pac J Oper Res*, 32(6):1550049. <https://doi.org/10.1142/S0217595915500499>