Received: 7 May 2018 | Revised: 2 March 2019 | Accepted: 6 April 2019 | Published on: 25 April 2019

DOI: 10.1002/nav.21846

RESEARCH ARTICLE

WILEY

Server scheduling on parallel dedicated machines with fixed job sequences

T.C.E. Cheng[1] | Svetlana A. Kravchenko[2] | Bertrand M. T. Lin[3]

[1]Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University, Hung Hom, Hong Kong
[2]United Institute of Informatics Problems, National Academy of Sciences of Belarus, Minsk, Belarus
[3]Institute of Information Management, National Chiao Tung University, Hsinchu, Taiwan

Correspondence
Bertrand M. T. Lin, Institute of Information Management, National Chiao Tung University, Hsinchu 300, Taiwan.
Email: bmtlin@mail.nctu.edu.tw

Funding information
Ministry of Science and Technology of Taiwan, MOST 106-2410-H-009-007-MY2. Fung Yiu King-Wing Hang Bank Endowed Professorship in Business Administration, The Hong Kong Polytechnic University.

Abstract
We consider server scheduling on parallel dedicated machines to minimize the makespan. Each job has a loading operation and a processing operation. The loading operation requires a server that serves all the jobs. Each machine has a given set of jobs to process, and the processing sequence is known and fixed. We design a polynomial-time algorithm to solve the two-machine case of the problem. When the number of machines is arbitrary, the problem becomes strongly *NP*-hard even if all the jobs have the same processing length or all the loading operations require a unit time. We design two heuristic algorithms to treat the case where all the loading times are unit and analyze their performance.

KEYWORDS
fixed job sequence, makespan, parallel dedicated machines, single server

1 | INTRODUCTION

Limited availability of resources is one of the major constraints in managerial decisions. In this paper, we consider scheduling of jobs with fixed processing sequences on parallel dedicated machines with a common server, where both the server and machines are limited resources. We first study the two-machine case, followed by the case with an arbitrary number of machines. Specifically, m disjoint job sets $\mathcal{J}_k = \{J_{k,1}, \dots, J_{k,n_k}\}, 1 \le k \le m$, are given. The jobs of set $\mathcal{J}_k$ must be processed by their dedicated machine M_k in the order as specified by their indices, that is, machine M_k processes its jobs in the order $J_{k,1}, \dots, J_{k,n_k}$. Each job $J_{k,j}$ comprises two operations, that is, loading and processing, requiring $s_{k,j}$ and $p_{k,j}$ units of time, respectively. We denote the completion time of job $J_{k,j}$ in a particular schedule as $C_{k,j}$. A common server, for example, a technician or hoister, is available for performing the loading operations. So, given the limited availability of the server, at most one loading operation can be performed at any time. The processing of a job can start only when its loading operation is completed. The objective of the problem is to find a feasible schedule that minimizes the makespan, the maximum job completion time $C_{\max} = \max\{C_{1,n_1}, \dots, C_{m,n_m}, \}$. We denote the problem as $PD, S1 \mid \text{fixed-seq} \mid C_{\max}$, where PD is for dedicated machines, $S1$ is for a single server, and "fixed-seq" in the second field indicates the assumption that the job processing sequences are fixed. We suppose that all the data in this paper are nonnegative integers.

Kravchenko and Werner (1997) considered the server scheduling problem $P2, S1 \| C_{\max}$. The *NP*-hardness of the problem directly follows from the classical two parallel-machine scheduling problem to minimize the makespan. They developed a pseudo-polynomial-time algorithm to solve the case $P2, S1 \mid s_j = 1 \mid C_{\max}$, where $s_j = 1$ indicates unit loadings for all jobs. The problem becomes strongly *NP*-hard when the loading times are the same but not necessarily equal to one. The case $P, S1 \mid p_j = 1 \mid C_{\max}$ is polynomially solvable (Hall, Potts, & Sriskandarajah, 2000). Brucker, Dhaenens-Flipo, Knust, Kravchenko, and Werner (2002) proved that the case $P2, S1 \mid p_j = p \mid C_{\max}$ is binary *NP*-hard. Glass, Shafransky, and Strusevich (2000) proved that $PD, S1 \| C_{\max}$ remains strongly *NP*-hard even if there are only two machines and all the loading times are equal or all the processing times are equal. Hasani, Kravchenko, and Werner (2014) formulated $P2, S1 \| C_{\max}$ as a mixed integer program using block models. Through computational experiments, they studied the gap performance of the proposed models and several heuristics in the literature. Jiang, Dong, and Ji (2013) considered two-machine server scheduling with preemption of job loading and processing. They designed a heuristic that attains a tight performance ratio of 4/3, and proposed polynomial-time solution algorithms for the two special cases where $p_j = p$ and $s_j = s$. The complexity status of the two-machine preemptive version had remained open until Cheng, Kravchenkov, and Lin (2017) proposed a pseudo-polynomial-time algorithm to solve it. They also showed that the case with an arbitrary number of machines is strongly *NP*-hard. They designed the longest processing time (LPT) heuristic for this case and showed that it has a tight performance ratio of 2. Several special cases are polynomially solvable. We refer the reader to Brucker et al. (2002), Abdekhodaee, Wirth, and Gan (2006), and Werner and Kravchenko (2010) for comprehensive information on variants of the server scheduling problem.

While the $P2, S1 \| C_{\max}$ problem is intractable, permutation-based approximation algorithms such as local search and meta-heuristics have been developed. For such solution approaches, the objective values of the solutions that are represented in the corresponding job sequences need to be computed in an efficient manner. This requirement also arises from the deployment of exact solution methods such as the branch-and-bound approach. Since machine allocation has been determined under such approaches, we assume that the setup and processing of a job are consecutively executed on the same machine. So we focus on the specific variant of the problem under the assumption that the jobs have been assigned to machines following fixed processing sequences.

We organize the rest of the paper as follows: in Section 2 we develop a dynamic programming algorithm to solve $PD2, S1 \mid \text{fixed - seq} \mid C_{\max}$. In Section 3 we show that the case where the number of machines is part of the input is strongly *NP*-hard. In Section 4 we propose two heuristics for the case where all the loading operations require a unit time and study their performance through theoretical analysis and computational tests. We conclude the paper and suggest topics for future research in Section 5.

2 | TWO MACHINES

In this section we consider the case of the problem where there are only two dedicated machines, that is, $PD2, S1 \mid \text{fixed - seq} \mid C_{\max}$.

For most scheduling problems, given a job sequence, computing the objective value is easy. However, it is not trivial for $PD2, S1 \mid \text{fixed - seq} \mid C_{\max}$. To solve this problem, the first decision is to determine which of the first setup operations on the two machines should proceed first, which will affect the later decisions. Consider the three-job instance shown in Figure 1A, where jobs $J_{1,1}$ and $J_{1,2}$ are assigned to machine M_1, and job $J_{2,1}$ to machine M_2. There are two schedules that start with M_1 and M_2, respectively. The schedule starting with M_1 completes earlier. Figure 1B shows two schedules with

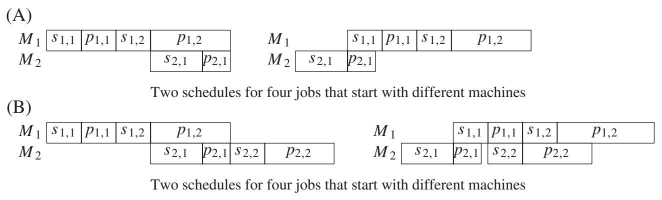
(A)
M_1 $s_{1,1}$ $p_{1,1}$ $s_{1,2}$ $p_{1,2}$
M_2 $s_{2,1}$ $p_{2,1}$
M_1 $s_{1,1}$ $p_{1,1}$ $s_{1,2}$ $p_{1,2}$
M_2 $s_{2,1}$ $p_{2,1}$
Two schedules for four jobs that start with different machines
(B)
M_1 $s_{1,1}$ $p_{1,1}$ $s_{1,2}$ $p_{1,2}$
M_2 $s_{2,1}$ $p_{2,1}$ $s_{2,2}$ $p_{2,2}$
M_1 $s_{1,1}$ $p_{1,1}$ $s_{1,2}$ $p_{1,2}$
M_2 $s_{2,1}$ $p_{2,1}$ $s_{2,2}$ $p_{2,2}$
Two schedules for four jobs that start with different machines

FIGURE 1 Schedules starting with different machines

one more job $J_{2,2}$ to process on M_2. In this case, starting with M_2 results in a shorter makespan. The above examples indicate the necessity for examining different combinations of a series of decisions. A similar challenge exists in the scheduling problems studied in Hwang, Kovalyov, and Lin (2014a, 2014b), Lin and Hwang (2011), Shafransky and Strusevich (1998), and Sourd (2005). Resolution of such decisions has its theoretical interest, as well as meeting the need to efficiently compute the objective values in permutation-based algorithms.

We apply dynamic programming to solve the problem. To determine the optimal value of a specific state, we resort to job-by-job recursions. Nevertheless, removing a job from an optimal schedule may induce different scenarios that make the analysis hard to follow. In addition to job-based recursions, we introduce blocks to simplify the analysis.

A block is a set of jobs that can be processed consecutively without creating forced idle time except the first loading. The Gantt chart of a schedule starts with a setup operation on one machine and an idle time on the other machine. Subsequently, the Gantt chart can be interpreted as a juxtaposition of components that are separated by idle slots on either of the machines. The components could be individual jobs or sub-sequences of jobs that are processed without any idle time. For $1 \le i_1 \le j_1 \le n_1$ and $1 \le i_2 \le j_2 \le n_2$, we define *block* $B(i_1, j_1, i_2, j_2)$ of the processing of the two sub-sequences $(J_{1,i_1}, J_{1,i_1+1}, \dots, J_{1,j_1})$ and $(J_{2,i_2}, J_{2,i_2+1}, \dots, J_{2,j_2})$. In addition to meeting the availability constraint of the common server, block $B(i_1, j_1, i_2, j_2)$ must satisfy the following three conditions

- C1: No idle time is incurred between any two jobs on the same machine in the block, and no idle time is created during loadings except for the first one in the block.
- C2: Setup s_{2,i_2} starts at the completion of setup s_{1,i_1}, or vice versa.
- C3: If $C_{1,j_1} \neq C_{2,j_2}$, then J_{1,j_1} is the only job that satisfies $C_{1,j_1} > C_{2,j_2}$ or J_{2,j_2} is the only job that satisfies $C_{2,j_2} > C_{1,j_1}$.

For example the left schedule in Figure 1B can be represented by three blocks as follows: the first block is the only job $J_{1,1}$, the second block comprises the two jobs $\{J_{1,2}, J_{2,1}\}$, and the third block contains only the job $J_{2,2}$. Note that $\{J_{1,2}, J_{2,1}, J_{2,2}\}$ does not form a block because of the created idle time during the third setup. The right schedule in Figure 1B is represented by two blocks. The first block

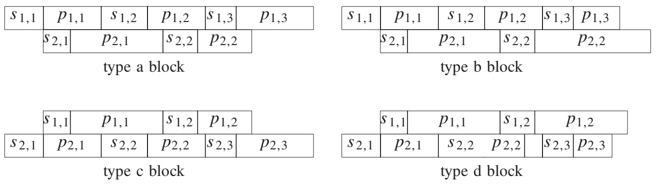

FIGURE 2 Four types of blocks

contains only the job $J_{2,1}$ and the second block contains the three jobs $\{J_{1,1}, J_{2,2}, J_{1,2}\}$. Note that jobs $J_{1,1}$ and $J_{2,1}$ do not form a block because of the forced idle time during the second setup.

Condition (C3) prevents a block from having a long tail of individual jobs on one machine. Given the job indices i_1, j_1, i_2, and j_2, there are four types of blocks, depending on which machine starts first and which machine finishes last as follows:

Type a: Block $B(i_1,j_1,i_2,j_2)$ starts and finishes both on machine M_1.
Type b: Block $B(i_1,j_1,i_2,j_2)$ starts on machine M_1 and finishes on machine M_2.
Type c: Block $B(i_1,j_1,i_2,j_2)$ starts and finishes both on machine M_2.
Type d: Block $B(i_1,j_1,i_2,j_2)$ starts on machine M_2 and finishes on machine M_1.

Figure 2 shows example blocks of the four types. For each $B(i_1,j_1,i_2,j_2)$, we should check if it is an admissible block satisfying the above three conditions. Therefore, to check if $(J_{1,i_1}, J_{1,i_1+1}, \dots, J_{1,j_1})$ and $(J_{2,i_2}, J_{2,i_2+1}, \dots, J_{2,j_2})$ constitute block $B(i_1,j_1,i_2,j_2)$, we have to schedule the given jobs in the way as shown in Figure 2 and check the three conditions for the obtained schedule. To this end, for all the job indices, we deploy a procedure that examines all $B(i_1,j_1',i_2,j_2')$, $i_1 \le j_1' \le j_1, i_2 \le j_2' \le j_2$. First, $B(i_1,i_1,i_2,i_2)$ is examined. If it is admissible, then we examine $B(i_1,i_1+1,i_2,i_2)$ and $B(i_1,i_1,i_2,i_2+1)$. The process continues until we reach $B(i_1,j_1,i_2,j_2)$. If some $B(i_1,j_1',i_2,j_2')$ is found inadmissible, then $B(i_1,j_1,i_2,j_2)$ is inadmissible.

To facilitate discussion, we introduce the notion of the rear part of a block. If a block $B(i_1,j_1,i_2,j_2)$ starts with machine M_1 (type a or type b), then it has a tail of length

$$|C_{1,j_1} - C_{2,j_2}| = \left| p_{1,i_1} + \sum_{\ell=i_1+1}^{j_1} (s_{1,\ell} + p_{1,\ell}) - \sum_{\ell=i_2}^{j_2} (s_{2,\ell} + p_{2,\ell}) \right|.$$

On the other hand, if it starts with machine M_2 (type c or type d), then its tail length is given by

$$|C_{1,j_1} - C_{2,j_2}| = \left| p_{2,i_2} + \sum_{\ell=i_2+1}^{j_2} (s_{2,\ell} + p_{2,\ell}) - \sum_{\ell=i_1}^{j_1} (s_{1,\ell} + p_{1,\ell}) \right|.$$

Based upon the block structure, we can then develop a backward dynamic programming algorithm to examine all the possible combinations of the jobs and/or blocks. Define the state (k, i_1, i_2), $k \in \{1, 2\}$, for the scenario where

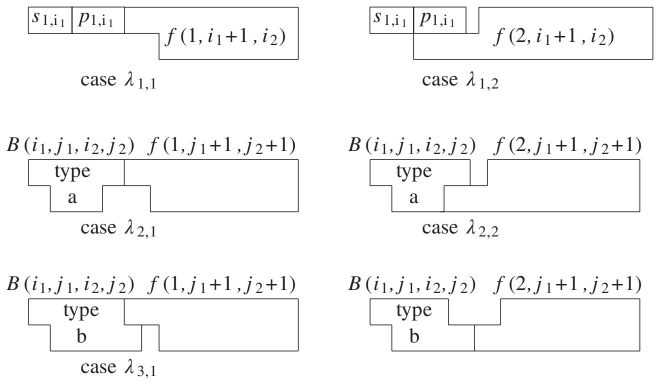

FIGURE 3 Types of recursion in Algorithm DP

(1) the sub-sequences $(J_{1,i_1}, J_{1,i_1+1}, \dots, J_{1,n_1})$ and $(J_{2,i_2}, J_{2,i_2+1}, \dots, J_{2,n_2})$ are considered, and
(2) the schedule starts with job J_{k,i_k} on machine M_k.

Let $f(k, i_1, i_2)$ be the optimal makespan of the schedules for the state (k, i_1, i_2). We introduce the auxiliary jobs J_{1,n_1+1} and J_{2,n_2+1} with $p_{1,n_1+1} = s_{1,n_1+1} = p_{2,n_2+1} = s_{2,n_2+1} = 0$ to define the boundary conditions. To calculate the value of $f(1, i_1, i_2)$, we first consider three disjoint cases for the first block of the schedule, which can be one of the following cases:

(1) a single job,
(2) a type a block, and
(3) a type b block.

Each case can be further categorized into two sub-cases concerning the machine on which the sub-schedule would start. Please refer to Figure 3 for the six cases. The value of $f(2, i_1, i_2)$ is similarly computed.

We now present the dynamic programming algorithm. Note that the algorithm generates only semiactive schedules, that is, no job (operation) can be processed earlier without changing the processing order or violating the constraints (Brucker, 2007, p. 7).

Algorithm DP: Initialization:

$$f(1, n_1+1, n_2+1) = f(2, n_1+1, n_2+1) = 0;$$

$$f(1, i_1, n_2+1) = \sum_{\ell=i_1}^{n_1} (s_{1,\ell} + p_{1,\ell}), \quad i_1 = 1, 2 \dots, n_1;$$

$$f(2, n_1+1, i_2) = \sum_{\ell=i_2}^{n_2} (s_{2,\ell} + p_{2,\ell}), \quad i_2 = 1, 2 \dots, n_2;$$

$f(k, i_1, i_2) = \infty$ for other combinations of k, i_1, and i_2.

Recursion:

For $i_1 = n_1, n_1 - 1, \dots, 1$ and $i_2 = n_2, n_2 - 1, \dots, 1$, do the following steps:

Case 1 The first block is a single job J_{1,i_1}:

/* In the recursion, the sub-schedule starts with machine M_1, see Figure 3 case $\lambda_{1,1}$.

$\lambda_{1,1} = s_{1,i_1} + p_{1,i_1} + f(1, i_1+1, i_2);$

/* In the recursion, the sub-schedule starts with machine M_2, see Figure 3 case $\lambda_{1,2}$.

$$\lambda_{1,2} = \begin{cases} s_{1,i_1} + f(2, i_1+1, i_2), & \text{if } s_{2,i_2} \geq p_{1,i_1}; \\ \infty, & \text{otherwise.} \end{cases}$$

$\lambda_1 = \min\{\lambda_{1,1}, \lambda_{1,2}\}$.

Case 2 The first block is of type a:

Let $\mathcal{B}_a(i_1, i_2)$ be the set of ordered pairs (j_1, j_2), $1 \leq i_1 < j_1 \leq n_1$ and $1 \leq i_2 < j_2 \leq n_2$, for which the sub-sequences $(J_{1,i_1}, J_{1,i_1+1}, \dots, J_{1,j_1})$ and $(J_{2,i_2}, J_{2,i_2+1}, \dots, J_{2,j_2})$ form type a blocks.

/* In the recursion, the sub-schedule starts with machine M_1, see Figure 3 case $\lambda_{2,1}$.

$$\lambda_{2,1} = \min_{(j_1,j_2)\in\mathcal{B}_a(i_1,i_2)} \left\{ \sum_{\ell=i_1}^{j_1} (s_{1,\ell} + p_{1,\ell}) + f(1, j_1+1, j_2+1) \right\}.$$

/* In the recursion, the sub-schedule starts with machine M_2, see Figure 3 case $\lambda_{2,2}$.

$$\lambda_{2,2} = \min_{(j_1,j_2)\in\mathcal{B}_a(i_1,i_2)} \begin{cases} s_{1,i_1} + \sum_{\ell=i_2}^{j_2} (s_{2,\ell} + p_{2,\ell}) + f(2, j_1+1, j_2+1), \\ \qquad \text{if } s_{2,j_2+1} \geq | C_{1,j_1} - C_{2,j_2} |; \\ \infty, \qquad \text{otherwise.} \end{cases}$$

$\lambda_2 = \min\{\lambda_{2,1}, \lambda_{2,2}\}$.

Case 3 The first block is of type b:

Let $\mathcal{B}_b(i_1, i_2)$ be the set of ordered pairs (j_1, j_2), $1 \leq i_1 < j_1 \leq n_1$ and $1 \leq i_2 < j_2 \leq n_2$, for which the sub-sequences $(J_{1,i_1}, J_{1,i_1+1}, \dots, J_{1,j_1})$ and $(J_{2,i_2}, J_{2,i_2+1}, \dots, J_{2,j_2})$ form type b blocks.

/* In the recursion, the sub-schedule starts with machine M_1, see Figure 3 case $\lambda_{3,1}$.

$$\lambda_{3,1} = \min_{(j_1,j_2)\in\mathcal{B}_b(i_1,i_2)} \begin{cases} \sum_{\ell=i_1}^{j_1} (s_{1,\ell} + p_{1,\ell}) + f(1, j_1+1, j_2+1), \\ \qquad \text{if } s_{1,j_1+1} \geq | C_{1,j_1} - C_{2,j_2} |; \\ \infty, \qquad \text{otherwise.} \end{cases}$$

/* In the recursion, the sub-schedule starts with machine M_2, see Figure 3 case $\lambda_{3,2}$.

$$\lambda_{3,2} = \min_{(j_1,j_2)\in\mathcal{B}_b(i_1,i_2)} \left\{ s_{1,i_1} + \sum_{\ell=i_2}^{j_2} (s_{2,\ell} + p_{2,\ell}) + f(2, j_1+1, j_2+1) \right\}.$$

$\lambda_3 = \min\{\lambda_{3,1}, \lambda_{3,2}\}$.

Analysis of the above three cases leads to $f(1, i_1, i_2) = \min\{\lambda_1, \lambda_2, \lambda_3\}$ and the function values $f(2, i_1, i_2)$ are derived by a similar process.

Goal: Find $\min\{f(1, 1, 1), f(2, 1, 1)\}$.

In **Algorithm DP**, the *if*-conditions in $\lambda_{1,2}$, $\lambda_{2,2}$, and $\lambda_{3,1}$ are used to ensure the existence of idle time required to separate the first component from the sub-schedule for recursions. If the first component and the sub-schedule of the recursion do not introduce any idle time on either of the two machines, then a larger component should be considered.

Theorem 1 ***Algorithm DP*** *finds the optimal values $f(k, i_1, i_2)$ for all $k \in \{1, 2\}$, $i_1 \in \{1, \dots, n_1+1\}$, and $i_2 \in \{1, \dots, n_2+1\}$.*

Proof Recall that we consider only semiactive schedules, that is, no job (operation) can be processed earlier without changing the processing order or violating the constraints. It is easy to see that only the cases in Figure 3 can arise in an optimal schedule. For the instance shown in Figure 4A, the first setup s_{2,i_2} on machine M_2 in the partial schedule corresponding to $f(2, i_1+1, i_2)$ can be shifted to the left without violating feasibility. The same issue arises in the schedule shown in Figure 4B. Moreover, whenever **Algorithm DP** yields ∞ (in $\lambda_{1,2}$, $\lambda_{1,2}$ and $\lambda_{3,1}$), the corresponding schedule is not active.

The correctness of the proof is based on the following three observations: (1) Any schedule can be decomposed into a sequence of blocks and single jobs. (2) Any schedule can be represented as a sequence of states, where any two neighboring states represent a block or a single job. (3) In a sequence of states representing an optimal schedule for any state (k, i_1, i_2), the value $f(k, i_1, i_2)$ is the minimum.

We consider only the case $k = 1$. The case $k = 2$ can be similarly analyzed. We show that **Algorithm DP** finds the optimal makespan $f(1, i_1, i_2)$ for $i_1 \in \{1, \dots, n_1+1\}$ and $i_2 \in \{1, \dots, n_2+1\}$. Assume that the optimal values $f(k', i'_1, i'_2)$ are known for $k' \in \{1,2\}$, $i'_1 \in \{i_1, \dots, n_1+1\}$, and $i'_2 \in \{i_2, \dots, n_2+1\}$, excluding $i'_1 = i_1$ and $i'_2 = i_2$.

Consider some optimal schedule σ associated with the state $(1, i_1, i_2)$. If the job sequence on M_2 is empty, that is, $i_2 = n_2 + 1$, then the optimal value $f(1, i_1, n_2+1)$ is determined in the Initialization step. Assume that the job sequence on M_2 is not empty and that setup s_{2,i_2} starts at the completion of s_1, i'_1, $\quad i_1 \leq i'_1$. We consider two disjoint cases as follows:

(1) $i_1 < i'_1$:

In this case, the optimal value $f(1, i_1, i_2)$ is obtained by a series of single-job recursions, together with the value $f(1, i'_1, i_2)$ as in Case 1-$\lambda_{1,1}$ in the algorithm.

(2) $i_1 = i'_1$:

The analysis proceeds with a focus on the first idle time inserted between two jobs on the same machine. If there is no such idle time in σ, then σ comprises a single block and a tail on either of the machines. The optimal value

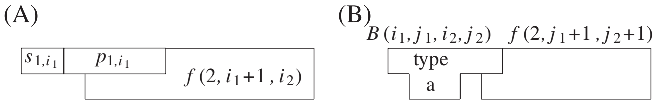

FIGURE 4 Nonsemiactive schedules

$f(1, i_1, i_2)$ is thus computed by a block-based recursion with a function value found in Initialization. Now, we assume that the earliest idle time in schedule σ occurs between jobs J_{k,j_k} and J_{k,j_k+1}, $i_k \le j_k \le n_k$. If $k = 2$, then a block $B(1, i_1, j_1, i_2, j_2) \in \mathcal{B}_a(i_1, i_2)$ is examined, together with $f(1, j_1+1, j_2+1)$ as in Case 2-$\lambda_{2,1}$, or a block $B(1, i_1, j_1, i_2, j_2) \in \mathcal{B}_b(i_1, i_2)$ is examined, together with $f(1, j_1+1, j_2+1)$ as in Case 3-$\lambda_{3,1}$. On the other hand, if $k = 1$, then three settings will arise: (i) job J_{1,i_1} and $f(2, i_1+1, i_2)$ are examined as in Case 1-$\lambda_{1,2}$; (ii) block $B(1, i_1, j_1, i_2, j_2) \in \mathcal{B}_a(i_1, i_2)$ is examined, together with $f(1, j_1+1, j_2+1)$ as in Case 2-$\lambda_{2,2}$; and (iii) block $B(1, i_1, j_1, i_2, j_2) \in \mathcal{B}_b(i_1, i_2)$ is examined, together with $f(2, j_1+1, j_2+1)$ as in Case 3-$\lambda_{3,2}$.

The above cases are disjoint and complete. The optimal value $f(1, i_1, i_2)$ is thus derived from recursions with different *first* components being considered. ∎

Running Time: There are $O(n_1 n_2)$ states, each of which requires to examine $O(n_1 n_2)$ first blocks of $\mathcal{B}_r$, $r \in \{a, b, c, d\}$. The total of the loading times and processing times can be computed by a pre-processing procedure. Therefore, the overall running time is $O(n_1^2 n_2^2)$. The eligibility of the 4-tuple vectors (i_1, j_1, i_2, j_2) for being a block can be checked by direct construction in a linear time. If (i_1, j_1, i_2, j_2) constitutes a block, then examining (i_1, j_1+1, i_2, j_2) or (i_1, j_1, i_2, j_2+1), and calculating its tail length can be performed in a constant time. On the other hand, if (i_1, j_1, i_2, j_2) is not a block, then (i_1, j_1+1, i_2, j_2) and (i_1, j_1, i_2, j_2+1) are not examined further. Therefore, the time required by a pre-processing procedure is $O(n_1^2 n_2^2)$, the same as that of **Algorithm DP**.

Theorem 2 ***Algorithm DP*** *solves* $PD2$, $S1 \mid \text{fixed-seq} \mid C_{\max}$ *in* $O(n_1^2 n_2^2)$ *time.*

3 | ARBITRARY NUMBER OF MACHINES: COMPLEXITY

A natural generalization of the two-machine case is PD, $S1 \mid \text{fixed-seq} \mid C_{\max}$, where the number of machines is part of the input rather than a constant. We show that this case is strongly NP-hard even if (1) all the jobs require the same processing time or (2) all the loading operations take a unit time. The first proof is based on a reduction from the strongly NP-hard 3-PARTITION problem (Garey & Johnson, 1979).

3-PARTITION: Given a positive integer E and a set of $3q$ elements $I = \{1, 2, \ldots, 3q\}$, in which each element i has a size $e_i \in Z^+$ such that $E/4 < e_i < E/2$ and $\sum_{i \in I} e_i = qE$, can I be partitioned into q subsets $I_1, I_2, \ldots, I_q$ such that each subset I_j satisfies $\sum_{i \in I_j} e_i = E$?

Theorem 3 PD, $S1 \mid \text{fixed-seq}, p_j = p \mid C_{\max}$ *is strongly NP-hard.*

Proof Given a 3-PARTITION instance, we create an instance of PD, $S1 \mid \text{fixed-seq}, p_j = p \mid C_{\max}$ that consists of $3q+1$ machines $\{M_1, M_2, \ldots, M_{3q+1}\}$ and $4q+1$ jobs $\{J_{1,1}, J_{2,1}, \ldots, J_{3q,1}, J_{3q+1,1}, J_{3q+1,2}, \ldots, J_{3q+1,q+1}\}$. Each machine M_k, $1 \le k \le 3q$, processes a single job $J_{k,1}$, while machine M_{3q+1} processes $q+1$ jobs $J_{3q+1,1}, \ldots, J_{3q+1,q+1}$. We define the jobs as follows:

$$s_{k,1} = e_k, p_{k,1} = E, \quad 1 \le k \le 3q;$$

$$s_{3q+1,j} = 1, p_{3q+1,j} = E, \quad 1 \le j \le q+1.$$

It is not hard to verify the claim that the answer to 3-PARTITION is affirmative if and only if there is a feasible schedule whose makespan is not greater than $(q+1)(E+1)$. ∎

The next proof is for the case where all the loading operations take one unit of time. We polynomially transform the 3-DIMENSIONAL MATCHING (3DM) problem to the decision version of PD, $S1 \mid \text{fixed-seq}, s_j = 1 \mid C_{\max}$. The 3DM problem, shown to be NP-hard by Karp (1972), is as follows:

3-DIMENSIONAL MATCHING: Given a set $U \subseteq T \times T \times T$, where $T = \{t_1, \ldots, t_q\}$ is a finite set, does $U = \{u_1, \ldots, u_z\}$ contain a perfect matching, that is, a subset $W \subseteq U$ such that its cardinality $|W| = q$ and no two elements of W agree in any coordinate?

Theorem 4 PD, $S1 \mid \text{fixed-seq}, s_j = 1 \mid C_{\max}$ *is strongly NP-hard.*

Proof Given an instance of 3DM, we create an instance of PD, $S1 \mid \text{fixed-seq}, s_j = 1 \mid C_{\max}$ with n jobs, where $n = z^5(3z^2 + z + 1)(z - q) + z^5 + z^4 + 3z^3 + z^2 + 2z + 1$. The number of machines is $m = z + 1$.

Machine M_r, $1 \le r \le z$, corresponds to the triple $u_r \in T \times T \times T$ and processes the $n_r = 3z^2 + z + 1$ jobs $J_{r,1}, \ldots, J_{r,n_r}$. The processing times of these n_r jobs are defined by the coordinate values t_1^r, t_2^r, and t_3^r of the corresponding triple $u_r = (t_1^r, t_2^r, t_3^r)$. For the case $t_1^r = t_i$, $t_2^r = t_j$, and $t_3^r = t_k$, we define the processing times as follows:

One job with $p_{r,1} = z^4 + (i-1)z^2 + q - i$ that starts the processing on the machine.

$3z^2$ jobs:

$$p_{r,2} = \cdots = p_{r,z^2} = 0,\ p_{r,z^2+1} = (q-i)z^2 + (j-1)z^2,$$
$$p_{r,z^2+2} = \cdots = p_{r,2z^2} = 0,\ p_{r,2z^2+1} = (q-j)z^2 + (k-1)z^2,$$
$$p_{r,2z^2+2} = \cdots = p_{r,3z^2} = 0,\ p_{r,3z^2+1} = (q-k)z^2 + iz.$$

Call them matching jobs.

z jobs:

$$p_{r,3z^2+2} = \cdots = p_{r,3z^2+z} = 0,\ p_{r,3z^2+z+1} = (q-1)z.$$

Call them separating jobs.

Machine M_{z+1} handles $n_{z+1} = z^5(3z^2+z+1)(z-q) + z^5 + z^4 + z + 1$ jobs, where the jobs are defined as follows:

One job with $p_{z+1,1} = q$ that starts the processing on the machine.

z^4 jobs:

$$p_{z+1,2} = \cdots = p_{z+1,z^4} = 0,\ p_{z+1,z^4+1} = 3qz^2.$$

Call them matching jobs.

z jobs:

$$p_{z+1,z^4+2} = \cdots = p_{z+1,z^4+z} = 0,\ p_{z+1,z^4+z+1} = qz.$$

Call them separating jobs.

$z^5(3z^2+z+1)(z-q)$ jobs:

$$p_{z+1,(\psi-1)z^5+z^4+z+2} = \cdots = p_{z+1,(\psi-1)z^5+z^5+z^4+z} = 0,$$
$$p_{z+1,(\psi-1)z^5+z^5+z^4+z+1} = 1,$$

for $\psi = 1, 2, \ldots, (z-q)(3z^2+z+1)$.

Call them finishing jobs.

z^5 jobs:

$$p_{z+1,z^5(3z^2+z+1)(z-q)+z^4+z+2} = \cdots = p_{z+1,z^5(3z^2+z+1)(z-q)+z^5+z^4+z+1} = 0.$$

Call them final jobs.

The roles of the jobs are based on the following consideration: given a specific time point K of a schedule, see Figure 6, which will be defined later, the matching jobs processed before K constitutes a solution for 3DM. The separating jobs act as a buffer between the matching jobs scheduled before and after K. The finishing jobs on machine M_{z+1} provide a time interval to accommodate the matching jobs of the other machines that are scheduled after K. The final jobs anchor the specified makespan, that is, $C_{\max}$.

One can see that the reduction is polynomial in the input length of 3DM. We prove the claim that the answer to 3DM is positive if and only if there exists a feasible schedule with

$$C_{\max} \le n = z^5(3z^2+z+1)(z-q) + z^5 + z^4 + 3z^3 + z^2 + 2z + 1. \tag{3.1}$$

Since the total of the loadings for the given scheduling instance is equal to the total number of jobs n, Equation (3.1) holds only if the server works without idle time. Machine M_{z+1} handles $n_{z+1} = z^5(3z^2+z+1)(z-q) + z^5 + z^4 + z + 1$ jobs. For M_{z+1}, the total processing times, excluding the loading time, of the starting job, matching jobs, separating jobs, and finishing jobs are q, $3qz^2$, qz, and $(z-q)(3z^2+z+1)$, respectively. So the total pure processing time on machine M_{z+1} is their sum, that is, $q + 3qz^2 + qz + (z-q)(3z^2+z+1)$. Therefore, machine M_{z+1} needs to be active for

$$\begin{aligned} n_{z+1} &+ (q + 3qz^2 + qz + (z-q)(3z^2+z+1)) \\ &= (z^5(3z^2+z+1)(z-q) + z^5 + z^4 + z + 1) \\ &\quad + (q + 3qz^2 + qz + (z-q)(3z^2+z+1)) \\ &= z^5(3z^2+z+1)(z-q) + z^5 + z^4 + z + 1 \\ &\quad + z(3z^2+z+1) = n \end{aligned}$$

units of time. Comparing the engagement length of machine M_{z+1} with the right hand side of Equation (3.1), we construct the proof by determining a feasible schedule in which machine M_{z+1} is working without any downtime in the period $[0, n)$.

Thus, the question reduces to the following: Does there exist a feasible schedule without server idleness or machine M_{z+1} downtime in the period $[0, n)$?

(Only-If part) First, we show that if 3DM has a solution, then a schedule σ with $C_{\max} \le n$ exists. For simplicity, let the triples $u_1, u_2, \ldots, u_q$ constitute the known solution and $t_1^r = t_r$ for $r = 1, \ldots, q$. From this solution, we construct a schedule σ as follows: machine M_{z+1} starts loading at time zero, and then continues processing and loading without any downtime. Therefore, machine M_{z+1} is working in the whole interval $[0, n)$.

Machines M_r, $r = 1, \ldots, q$, work without any downtime in σ. Since M_r has to work for $z^4 + 3qz^2 + i(z-1) + q(z+1) + 1$ time units and $i = r$ by the condition $t_1^r = t_r$, we obtain that M_r has to work for $z^4 + 3qz^2 + r(z-1) + q(z+1) + 1$ time units. Thus, in schedule σ, machines M_r, $r = 1, \ldots, q$, process all the prescribed jobs in the interval $[r, z^4 + 3qz^2 + rz + q(z+1) + 1)$, respectively, see Figure 5. Note that since $u_1, u_2, \ldots, u_q$ represent a solution for 3DM, the server works without idle time in $[T_1, T_2) = [z^4 + q + 1, z^4 + 3qz^2 + q + 1)$. The condition

$t_1^r = t_r$ for $r = 1, \dots, q$ entails the continuous working of the server within the time intervals $[1, q+1)$ and $[T_2 + z, T_3) = [T_2 + z, T_2 + z + qz) = [z^4 + 3qz^2 + q + z + 1, z^4 + 3qz^2 + qz + q + z + 1)$.

Machines $M_{q+1}, \dots, M_z$ process the prescribed jobs after the time point $K = T_3 + z^5$, see Figure 6. This is permissible because after time K, machine M_{z+1} leaves the server free for $(3z^2 + z + 1)(z - q)$ unit-length intervals. As a consequence, we obtain a feasible schedule σ without any server idle time.

(If part) Next, we show that if a feasible schedule σ with $C_{\max} \le n$ exists for the scheduling instance, then 3DM has a solution. Recall that the ideas behind the reasoning are based on the job functions: the matching jobs processed before the time point K define a solution for 3DM, the finishing jobs provide the possibility to schedule the remaining matching jobs after K, and the separating jobs act as buffers between the matching jobs scheduled before K and after K.

It is easy to show that in schedule σ, machine M_{z+1} has to work without downtime, starting from 0, and that the server has to work without idle time within $[0, C_{\max}(\sigma)) = [0, n)$. One can note that machines $M_1, \dots, M_z$ can process all the starting jobs within time interval $[1, K)$, and if the starting job is loaded within $[1, q+1)$, then the corresponding machine can process all the assigned jobs within $[1, K)$, but if the starting job is loaded after $q+1$, then the corresponding machine cannot process any other job within $[1, K)$.

We show that in σ, all the machines performing the loading operations within the interval $[1, q+1)$ constitute a solution for the instance of 3DM.

Order the machines in such a way that for each $r = 1, \dots, q$, machine M_r performs loading in $[r, r+1)$. Form a schedule σ' (may be infeasible), where for each $r = 1, \dots, q$, machine M_r processes the prescribed jobs without downtime starting from the time point r. Now for each machine M_r, $r = 1, \dots, q$, we select a triple of time points $(t_1^r, t_2^r, t_3^r,)$ in the following way. Set $e_0 = z^4 + q + 1 + 0.5z^2$. By the job definition, machine M_r has to be loaded in one of the next q unit-length intervals

$$[e_0, e_0 + 1), [e_0 + z^2, e_0 + z^2 + 1), \dots, \\ [e_0 + (q-1)z^2, e_0 + (q-1)z^2 + 1).$$

Let this interval be the ath among the above intervals. Set $t_1^r = t_a$.

Similarly, there is only one unit-length interval among the next q intervals

$$[e_0 + qz^2, e_0 + qz^2 + 1), \dots, \\ [e_0 + qz^2 + (q-1)z^2, e_0 + qz^2 + (q-1)z^2 + 1),$$

where machine M_r, $r = 1, \dots, q$, can be loaded. Let the sequence number of this interval be b. Set $t_2^r = t_b$.

Finally, there is only one unit interval among the next q intervals

$$[e_0 + 2qz^2, e_0 + 2qz^2 + 1), \dots, \\ [e_0 + 2qz^2 + (q-1)z^2, e_0 + 2qz^2 + (q-1)z^2 + 1),$$

where machine M_r can be loaded. Let the sequence number of this interval be c. Set $t_3^r = t_c$.

Using schedule σ', we obtain the triple $(t_1^r, t_2^r, t_3^r) = (t_a, t_b, t_c)$ for each machine M_r, $r = 1, \dots, q$. We claim that the obtained set of triples constitutes a perfect matching. If this is not the case, then there are two triples, say, (t_1^1, t_2^1, t_3^1) and (t_1^2, t_2^2, t_3^2), agreeing in some coordinate, say, the first coordinate, that is, $t_1^1 = t_1^2$. In schedule σ', there would be a time interval $[e_0 - \delta, e_0 + \delta]$ where M_1 and M_2 are both loaded for the time 2δ, where $\delta \ge 0.5z^2 - q$ holds. Let X denote the set of jobs loaded on M_1 in the interval $[e_0 - \delta, e_0 + \delta]$ and Y the set of jobs loaded on M_2 in $[e_0 - \delta, e_0 + \delta]$.

Now consider σ and both sets X and Y. In schedule σ, the maximum distance between any job belonging to X and any job belonging to Y must be more than $4\delta = 2z^2 - 4q$. All the jobs from $X \cup Y$ can be ordered by their finishing times. Let $v \in X \cup Y$ denote the job with the largest finishing time. Let M_1 be the machine that processes v. In schedule σ, the separating jobs assigned to M_1 have to be loaded after time point T_3, implying that the server is idle within the interval $[1, T_3]$. It follows from the fact that the set of jobs loaded on $M_1, \dots, M_z$ within $[0, T_3)$ is limited by the following types of jobs: it can be the starting job or it can be any job assigned to the machine loaded within $[1, q+1]$. The total loading time for any machine $M_1, \dots, M_z$ loaded within $[1, q+1]$ is $3z^2 + z + 1$. Thus, the maximum loading time on $M_1, \dots, M_z$ within $[1, T_3]$ is $MAX = q(3z^2 + z + 1) + z - q$. Since the separating jobs assigned to M_1 are not loaded within $[1, T_3]$ the maximum loading time within $[1, T_3]$ is $L = MAX - z = q(3z^2 + z + 1) - q = 3qz^2 + qz$. On the other hand, machine M_{z+1} is free from the loadings within $[1, T_3]$ for $F = 3qz^2 + qz + q$, that is, $L < F$, which implies that the server is idle within $[1, T_3]$. A contradiction thus arises because the server has to work without idle time.

Therefore, the set of machines loaded with $[1, q+1)$ constitutes a perfect matching. ■

4 | HEURISTIC ALGORITHMS

After the complexity analysis, we design approximation algorithms and analyze their performance for the unit-loading case of the problem $PD, S1 \mid \text{fixed-seq}, s_j = 1 \mid C_{\max}$.

The first heuristic, called MINIMUM-LOADING-TIME, starts with a schedule of all the jobs. The starting schedule could be infeasible. The heuristic sequentially shifts all the jobs with overlapped loadings to create a feasible schedule.

HEURISTIC MINIMUM-LOADING-TIME:

Step 1: For each machine M_k, $k = 1, \ldots, m$, schedule all the jobs $J_{k,1}, \ldots, J_{k,n_k}$ in the time interval $\left[0, \sum_{j=1}^{n_k}(1 + p_{k,j})\right)$.

Step 2: Find the smallest index i such that the interval $[i, i+1)$ contains two or more loadings.

Step 3: Keep the loading operation on machine M_k with the minimum value n_k. For all the other machines with loadings in $[i, i+1)$, shift their processing one time unit later from $[t, t+1)$ to $[t+1, t+2)$ for each $t = i, i+1, \ldots$.

Step 4: If the obtained schedule is feasible, then output the schedule; otherwise, go to Step 2.

To attain a polynomial running time, we detail the design as follows: enumerate all machines in nondecreasing order of their n_k-values. It takes $O(m\log m)$ time. For each machine M_k, an ordered list $t_{k,1}, \ldots, t_{k,n_k}$ is created, where $t_{k,j}$ is the starting time of the loading for the corresponding job $J_{k,j}$. It takes $O(n)$ time. Besides, for each machine, we introduce the variable y_k, defined as a shift of the unscheduled jobs in the corresponding list. Initially, $y_k = 0$ for all $k = 1, \ldots, m$. In the first iteration, we consider all the values $t_{1,1} + y_1, \ldots, t_{m,1} + y_m$ in this order, select the first minimum value, say, $t_{q,1} + y_q$, and mark all M_k, $k \neq q$, with $t_{k,1} + y_k = t_{q,1} + y_q$. It takes $O(m)$ time. Schedule the job corresponding to $t_{q,1}$ in the time interval $[t_{q,1} + y_q, t_{q,1} + y_q + 1)$, indicate the next value $t_{q,2}$ as the first in the list $t_{q,1}, \ldots, t_{q,n_q}$, and set $y_k = y_k + 1$ for all the marked machines M_k. It takes $O(m)$ time. Thus, all the n iterations take $O(nm)$ time. Therefore, the overall running time of MINIMUM-LOADING-TIME is $O(nm)$.

We next analyze the heuristic's performance in terms of the deviation of its solution from the optimal solution. Denote the makespan of the schedule obtained by MINIMUM-LOADING-TIME by $C^L_{\max}$ and that of an optimal schedule by $C^*_{\max}$. We first present a property useful for the analysis.

Lemma 1 *Let* $n = \sum_{k=1}^{m} n_k$ *and* $n_{\max} = \max_{1 \le k \le m}\{n_k\}$. *By the pigeonhole principle, we know that* $n_{\max} \ge n/m$.

We have the following result on the performance of the proposed heuristic.

Theorem 5 *For* $PD, S1 \mid \text{fixed-sequences}, s_j = 1 \mid C_{\max}$, *Heuristic* MINIMUM-LOADING-TIME *constructs a schedule satisfying* $C^L_{\max} - C^*_{\max} \le n\left(1 - \frac{1}{m}\right)$ *and the bound is tight.*

Proof Let the value $C^L_{\max}$ be the completion time of job J_{x,n_x} on some machine M_x. One can see that $n_x + \sum_{j=1}^{n_x} p_{x,j}$ is the total actual working time of M_x. Let

$$[t_1, t_1 + 1), \ldots, [t_z, t_z + 1)$$

be the z unit-length intervals within which machine M_x is idle for some nonnegative z, that is, the number of idle unit-length slots. Let $\{M_{i_1}, \ldots, M_{i_y}\}$ be the set of machines performing loading operations in the intervals $[t_1, t_1 + 1), \ldots, [t_z, t_z + 1)$. Consider the machines of $\{M_{i_1}, \ldots, M_{i_y}\} \cup \{M_x\}$. By the machine selection rule of Step 3, we have $n_x \ge n_{i_1}, \cdots, n_x \ge n_{i_y}$. Therefore,

$$\begin{aligned} C^L_{\max} &= n_x + \sum_{j=1}^{n_x} p_{x,j} + z \\ &\le \left(n_x + \sum_{j=1}^{n_x} p_{x,j}\right) + n_{i_1} + \cdots + n_{i_y} \\ &\le C^*_{\max} + n_{i_1} + \cdots + n_{i_y} \\ &\le C^*_{\max} + n - n_{\max} \\ &\le C^*_{\max} + n - \frac{n}{m}. \end{aligned} \tag{4.1}$$

The last inequality follows from Lemma 1. We then have $C^L_{\max} \le C^*_{\max} + n\left(1 - \frac{1}{m}\right)$.

To show the tightness of the above ratio, we consider the following instance with m machines and m^2 jobs, where each machine M_i, $1 \le i \le m-1$, has exactly m jobs $J_{i,1}, \ldots, J_{i,m}$ with $p_{i,1} = \cdots = p_{i,m} = 0$, while machine M_m also has exactly m jobs $J_{m,1}, \ldots, J_{m,m}$, with $p_{m,1} = \cdots = p_{m,m-1} = 0$ and $p_{m,m} = (m-1)m$.

Step 1 of the heuristic has all the loading operations performed in the interval $[0, m)$. Since $n_1 = n_2 = \cdots = n_m = m$, the heuristic may perform the loading operations in any order of the machine indices. So it can happen that all the loading operations are performed on machines $M_1, \ldots, M_{m-1}$ in the interval $[0, m(m-1))$ and then machine M_m performs its loading

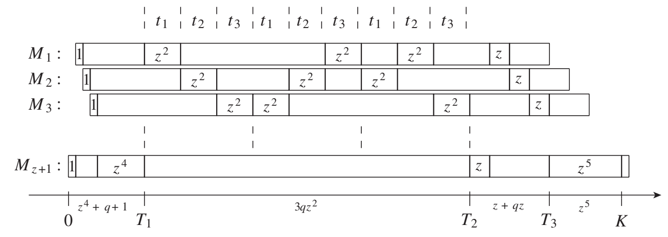

FIGURE 5 Constructed schedule within the time interval $[0, K)$ for $q = 3$ and the known solution (t_1, t_3, t_2), (t_2, t_2, t_1), and (t_3, t_1, t_3). The number inside each box means the time of the corresponding loading operation. To estimate the real-time scale, the length of the time interval between two points is specified below the time axis. $z > q$ holds

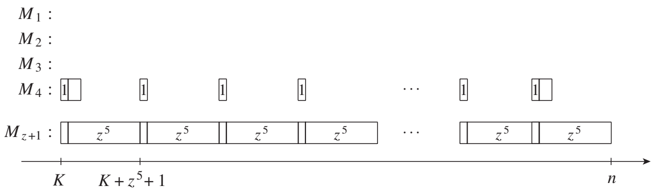

FIGURE 6 Constructed schedule for $z = 4$ within the time interval $[K, n)$. The number inside each box means the time of the corresponding loading operation

operations in the interval $[m(m-1), m^2)$. Then, $C^L_{\max} = 2m(m-1) + m$, while the optimal makespan $C^*_{\max} = m^2$. So $C^L_{\max} - C^*_{\max} = m^2 - m = n(1 - 1/m)$ holds. ■

In Step 3, another strategy for selecting the machine is to keep its loading operation when multiple loading operations occur in the same time slot. This leads to a new heuristic, called MAXIMUM-REMAINING-WORK, which selects the machine with the maximum remaining work (we acknowledge that this heuristic is suggested by one of the reviewers).

HEURISTIC MAXIMUM-REMAINING-WORK:

Step 1: For each machine M_k, $k = 1, \ldots, m$, schedule all the jobs $J_{k,1}, \ldots, J_{k,n_k}$ in the time interval $\left[0, \sum_{j=1}^{n_k}(1 + p_{k,j})\right)$.

Step 2: Find the smallest index i such that the interval $[i, i+1)$ contains two or more loadings.

Step 3: Keep the loading operation on machine M_k with the maximum remaining work for processing. For all the other machines with loadings in $[i, i+1)$, shift their processing one time unit later from $[t, t+1)$ to $[t+1, t+2)$ for each $t = i, i+1, \ldots$.

Step 4: If the obtained schedule is feasible, then output the schedule; otherwise, go to Step 2.

Denote the makespan of the schedule obtained by HEURISTIC MAXIMUM-REMAINING-WORK by $C^W_{\max}$. We do not provide an upper bound on the deviation $C^W_{\max} - C^*_{\max}$. Instead, we derive the following lower bound on the heuristic's performance.

Theorem 6 *For PD,* $S1 \mid \text{fixed-sequences}, s_j = 1 \mid C_{\max}$, HEURISTIC MAXIMUM-REMAINING-WORK *constructs a schedule satisfying* $C^W_{\max} - C^*_{\max} \geq n\left(1 - \frac{1}{m}\right)$.

Proof To establish the above deviation, we consider the following instance with m machines and $n = klm$ jobs, where $k \geq 1$, $l \geq 1$, and each machine M_i, $1 \leq i \leq m$ has to process exactly kl jobs $J_{i,1}, \ldots, J_{i,kl}$. For each q, $0 \leq q < k$, the corresponding subset of l jobs $J_{i,ql+1}, \ldots, J_{i,ql+l}$ is defined by $p_{i,ql+1} = \cdots = p_{i,ql+(l-1)} = 0$, and $p_{i,ql+l} = l(m-1)$. To construct an optimal schedule we process all jobs assigned to M_q, for $1 \leq q \leq m$, in the time interval $[(q-1)l, (q-1)l + klm]$, see Figure 7. Thus, the optimal $C^*_{\max}$ value is $klm + l(m-1)$.

The schedule constructed by HEURISTIC MAXIMUM-REMAINING-WORK can be described in the following way. All jobs assigned to M_q, for $1 \leq q \leq m$, are scheduled in the time interval $[(q-1), (q-1) + kl(2m-1) - km + k)]$ in such a way that after processing each job with zero processing time corresponding machine is idle for $m-1$ time units, and after processing each job with nonzero processing time corresponding machine starts loading immediately, see Figure 8. The obtained $C^W_{\max}$ value is $kl(2m-1) - km + k + m - 1$.

Thus, for the considered example

$$C^W_{\max} - C^*_{\max}$$
$$= klm\left(1 - \frac{1}{m} - \frac{1}{l} - \frac{1}{k} + \frac{1}{lm} + \frac{1}{km} + \frac{1}{kl} - \frac{1}{klm}\right)$$
$$= n\left(1 - \frac{1}{m} - \frac{1}{l} - \frac{1}{k} + \frac{1}{lm} + \frac{1}{km} + \frac{1}{kl} - \frac{1}{klm}\right).$$

The last value tends to $n\left(1 - \frac{1}{m}\right)$ when $k \to \infty$ and $l \to \infty$, and therefore the inequality $C^W_{\max} - C^*_{\max} \geq n(1 - 1/m)$ holds. ■

4.1 | Computational study

In addition to the above theoretical performance analysis of the two heuristics, we conducted a computational study to

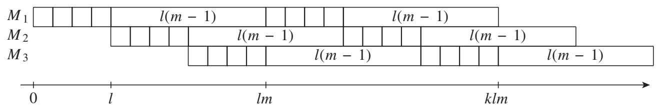

FIGURE 7 Optimal schedule for the case $m = 3$, $l = 4$, and $k = 2$. The number inside a box means the processing time of the corresponding job

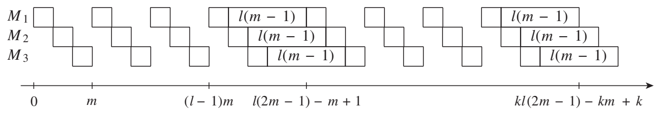

FIGURE 8 The schedule constructed by HEURISTIC MAXIMUM-REMAINING-WORK for the case $m = 3$, $l = 4$, and $k = 2$. The number inside a box means the corresponding processing time

assess the general performance of the heuristics. We developed a mixed integer programming (MIP) model to find the optimal solutions $C^*_{\max}$ of the tested instances. We implemented the heuristics in Python and solved the MIP model using the Gurobi Optimization Solver version 8.0.0 under a complimentary education licence. The platform of experiments was a personal computer equipped with an Inter i7-6800k CPU and 64GB RAM. We generated test instances with $m = 3$ or 5 machines as follows:

The number of jobs were $n = 30, 50, 100$;
The processing times $p_{k,j}$ were drawn from the uniform distribution $U[0, 20]$.

For each n, we consider two scenarios: even distribution of the jobs across the machines and one machine processes more jobs than the other machines. Altogether, we tested 12 settings, each of which is denoted by $(n.\ m.\ k)$, where n is the number of jobs, m is number of machines, and k is the machine distribution scenario as follows:

$$\begin{aligned}
&30.3.1 : n_1 = n_2 = n_3 = 10\\
&30.3.2 : n_1 = 14, n_2 = n_3 = 8\\
&50.3.1 : n_1 = 16, n_2 = n_3 = 17\\
&50.3.2 : n_1 = 20, n_2 = n_3 = 15\\
&100.3.1 : n_1 = 34, n_2 = n_3 = 33\\
&100.3.2 : n_1 = 40, n_2 = n_2 = 20\\
&30.5.1 : n_1 = \cdots = n_5 = 6\\
&30.5.2 : n_1 = 10, n_2 = \cdots = n_5 = 5\\
&50.5.1 : n_1 = \cdots = n_5 = 10\\
&50.5.2 : n_1 = 18, n_2 = \cdots = n_5 = 8\\
&100.5.1 : n_1 = \cdots = n_5 = 20\\
&100.5.2 : n_1 = 32, n_2 = \cdots = n_5 = 17.
\end{aligned}$$

For each of the above 12 testing settings, we generated 10 independent random instances. We measure the heuristic solution quality by its gap from the optimal solution in percentage, which is defined by

$$\frac{C^H_{\max} - C^*_{\max}}{C^*_{\max}} \times 100\%,$$

where $C^H_{\max}$, $H = L$ or W, is the solution value produced by the corresponding heuristic, respectively. Table 1 reports the results of the computational study. The columns "$C^*_{\max}$" and "$C^H_{\max}$" contain the average solution values over the 10 instances for each test setting produced by the MIP model and the heuristics. The column "Opt" contains the number of instances that are optimally solved by the corresponding heuristic. The columns "Gap", "M-Gap", and "m-Gap" show the average gap, the largest gap, and the smallest gap, all in percentages, between the heuristic solutions and the optimal ones over the 10 instances for each test testing, respectively.

Although the theoretical performance of HEURISTIC MINIMUM-LOADING-TIME is not worse than that of HEURISTIC MAXIMUM-REMAINING-WORK, the computational results indicate that in the case of uniform distribution, HEURISTIC MAXIMUM-REMAINING-WORK outperforms HEURISTIC MINIMUM-LOADING-TIME in view of the number of optimally solved instances and the average gap. HEURISTIC MAXIMUM-REMAINING-WORK optimally solves 62 of the 120 tested instances, while HEURISTIC MINIMUM-LOADING-TIME solves only eight instances to optimality. The average gap of HEURISTIC MINIMUM-LOADING-TIME ranges from 0.89% to 4.49%. On the other hand, most of the average gaps of HEURISTIC MAXIMUM-REMAINING-WORK are below 1%. For instances with balanced machine workloads, HEURISTIC MINIMUM-LOADING-TIME achieves smaller average gaps. HEURISTIC MINIMUM-LOADING-TIME, on the contrary, has better performance when a machine has more jobs to process. The reason could be that when the machine having a greater workload accords a higher priority to perform its loading operations, the increase in the maximum job completion time, that is, the makespan, will be confined. Another observation is that when the number of jobs on a machine increases, the average gaps of both heuristics become smaller, that is, both heuristics achieve better performance for larger instances.

5 | CONCLUSIONS

We consider parallel dedicated machines scheduling with a common server to minimize the makespan under the assumption that the job processing sequence on each machine is given and fixed. Our study evinces the fact that in server scheduling, to produce an optimal schedule from a given job sequence is not as easy as it seems. We design an $O(n_1^2 n_2^2)$ dynamic programming algorithm to solve the two-machine case of the problem. When the number of machines is arbitrary, we show that the problem becomes strongly NP-hard even if (1) all the jobs have the same processing time, that is, $p_j = p$, or (2) all the loading operations take a unit time, that is, $s_j = 1$. We provide two heuristics to solve the case where all the loading operations take a unit of time. We analyze their performance bounds and assess their solution

TABLE 1 Computational results of the two heuristics

		Minimum-loading-time					Maximum-remaining-work				
Instance	$C^*_{\max}$	$C^L_{\max}$	Opt	Gap	M-Gap	m-Gap	$C^W_{\max}$	Opt	Gap	m-Gap	m-Gap
30.3.1	132.3	133.8	3	1.19	4.00	0.00	133.0	5	0.55	2.48	0.00
30.3.2	158.7	161.4	0	1.72	3.45	1.07	158.7	10	0.00	0.00	0.00
50.3.1	214.2	217.2	2	1.46	3.11	0.00	215.1	4	0.42	1.90	0.00
50.3.2	232.1	235.9	0	1.65	2.43	0.83	232.5	7	0.17	0.84	0.00
100.3.1	424.0	427.8	2	0.89	2.10	0.00	425.0	3	0.24	0.75	0.00
100.3.2	489.5	495.3	0	1.20	2.13	0.21	490.2	3	0.14	0.21	0.00
30.5.1	83.1	86.8	1	4.49	8.57	0.00	84.4	2	1.55	3.57	0.00
30.5.2	122.0	127.3	0	4.45	7.21	2.76	123.0	4	0.96	4.55	0.00
50.5.1	138.2	142.2	0	2.92	5.63	0.63	138.9	5	0.52	1.54	0.00
50.5.2	213.5	220.0	0	3.08	4.57	1.84	214.3	7	0.40	2.53	0.00
100.5.1	257.2	261.5	0	1.67	3.42	0.41	259.3	0	0.82	1.66	0.36
100.5.2	363.6	372.0	0	2.32	3.04	1.86	365.1	2	0.41	0.80	0.00

quality through a computational study of randomly generated problem instances.

We note that our dynamic programming solution algorithm can be easily adapted to deal with the cases where the objective is to minimize the total completion time $\sum C_j$ or the maximum lateness $L_{\max}$. Take the objective of minimizing $\sum C_j$ as an example, we can compute

$$\lambda_{1,1} = (s_{1,i_1} + p_{1,i_1})(j_1 - i_1 + 1 + j_2 - i_2 + 1) + f(1, i_1 + 1, i_2),$$

$$\lambda_{2,1} = \min_{(j_1,j_2)\in\mathcal{B}_a(i_1,i_2)} \left\{ \sum_{\ell=i_1}^{j_1} (s_{1,\ell} + p_{1,\ell})(j_1 - i_1 + 1 + j_2 - i_2 + 1) + f(1, j_1 + 1, j_2 + 1) \right\}$$

to reflect the increase in the total completion time caused by the first job. The definitions of the other λ variables can accordingly be made. For $L_{\max}$, we have

$$\lambda_{1,1} = \max\{s_{1,i_1} + p_{1,i_1} - d_{1,i_1}, s_{1,i_1} + p_{1,i_1} + f(1, i_1 + 1, i_2)\},$$

$$\lambda_{2,1} = \min_{(j_1,j_2)\in\mathcal{B}_a(i_1,i_2)} \left\{ \max\left\{ L_{\max}(i_1, j_1, i_2, j_2), \sum_{\ell=i_1}^{j_1} (s_{1,\ell} + p_{1,\ell}) + f(1, j_1 + 1, j_2 + 1) \right\} \right\},$$

where $L_{\max}(i_1, j_1, i_2, j_2)$ is the maximum lateness in the block $B(i_1, j_1, i_2, j_2)$.

For future work, it is interesting to design polynomial-time algorithms for the case with a constant number of machines. The key is to define the states for the development of dynamic programming algorithms. The case where preemptions are allowed is another topic worthy of future research endeavors. It is easy to see that an optimal schedule for $PD, S1\,|\,\text{fixed-seq, pmtn}\,|\,C_{\max}$ can be found among the class of schedules where there is no preemption for any processing operation. It is also easy to note that there are heuristics such that $\widetilde{C} - C^* \le S$ and $\widetilde{C} - C^* \le \left(1 - \frac{1}{m}\right) S$, where $\widetilde{C}$ is the heuristic solution, C^* is the optimal solution, and S is the sum of the loading times. Therefore, developing approximation algorithms with better performance is a promising research direction.

ACKNOWLEDGMENTS

We thank the anonymous referees and editors for their helpful comments, which have helped us to significantly improve our paper. Cheng was supported in part by The Hong Kong Polytechnic University under the Fung Yiu King—Wing Hang Bank Endowed Professorship in Business Administration. Kravchenko is grateful to The Hong Kong Polytechnic University for financing a visit to the Department of Logistics and Maritime Studies where this research was partially undertaken. Lin was supported in part by the Ministry of Science and Technology of Taiwan under grant number MOST 106-2410-H-009-007-MY2.

ORCID

Bertrand M. T. Lin https://orcid.org/0000-0003-0456-296X

REFERENCES

Abdekhodaee, A. H., Wirth, A., & Gan, H. S. (2006). Scheduling two parallel machines with a single server: The general case. *Computers & Operations Research*, *33*, 994–1009.

Brucker, P. (2007). *Scheduling algorithms* (5th ed.). Heidelberg: Springer-Verlag.

Brucker, P., Dhaenens-Flipo, C., Knust, S., Kravchenko, S. A., & Werner, F. (2002). Complexity results for parallel machine problems with a single server. *Journal of Scheduling*, *5*, 429–457.

Cheng, T. C. E., Kravchenkov, S. A., & Lin, B. M. T. (2017). Preemptive parallel-machine scheduling with a common server to minimize makespan. *Naval Research Logistics*, *64*, 388–398.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco, CA: Freedman.

Glass, C. A., Shafransky, Y. M., & Strusevich, V. A. (2000). Scheduling for parallel dedicated machines with a single server. *Naval Research Logistics*, *47*, 304–328.

Hall, N., Potts, C., & Sriskandarajah, C. (2000). Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, *102*, 223–243.

Hasani, K., Kravchenko, S. A., & Werner, F. (2014). Block models for scheduling jobs on two parallel machines with a single server. *Computers & Operations Research*, *41*, 94–97.

Hwang, F. J., Kovalyov, M. Y., & Lin, B. M. T. (2014a). Total completion time minimization in two-machine flow shop scheduling problems with a fixed job sequence. *Discrete Optimization*, *9*, 29–39.

Hwang, F. J., Kovalyov, M. Y., & Lin, B. M. T. (2014b). Scheduling for fabrication and assembly in a two-machine flowshop with a fixed job sequence. *Annals of Operations Research*, *217*, 263–279.

Jiang, Y., Dong, J., & Ji, M. (2013). Preemptive scheduling on two parallel machines with a single server. *Computers & Industrial Engineering*, *66*, 514–518.

Karp, R. M. (1972). *Reducibility among combinatorial problems*. In R. E. Miller & J. W. Thatcher (Eds.), Complexity of computer computations (pp. 85–103). New York: Plenum Press.

Kravchenko, S. A., & Werner, F. (1997). Parallel machine scheduling problems with a single server. *Mathematical and Computer Modelling*, *26*, 1–11.

Lin, B. M. T., & Hwang, F. J. (2011). Total completion time minimization in a 2-stage differentiation flowshop with fixed sequences per job type. *Information Processing Letters*, *111*, 208–212.

Shafransky, Y. M., & Strusevich, V. A. (1998). The open shop scheduling problem with a given sequence of jobs on one machine. *Naval Research Logistics*, *45*, 705–731.

Sourd, F. (2005). Optimal timing of a sequence of tasks with general completion costs. *European Journal of Operational Research*, *165*, 82–96.

Werner, F., & Kravchenko, S. A. (2010). Scheduling with multiple servers. *Automation and Remote Control*, *71*, 2109–2121.

How to cite this article: Cheng TCE, Kravchenko SA, Lin BMT. Server scheduling on parallel dedicated machines with fixed job sequences. *Naval Research Logistics* 2019;66:321–332. https://doi.org/10.1002/nav.21846