ISSN 2683-345X

journal homepage: http://ijiemjournal.uns.ac.rs/

International Journal of Industrial Engineering and Management

Volume 15 / No 1 / March 2024 / 18 - 33

Original research article

Unrelated parallel machine scheduling under machine availability and eligibility constraints to minimize the makespan of non-resumable jobs

A. Kurt[a], F. C. Çetinkaya[b,*]

[a] *Alanya Alladdin Keykubat University, Industrial Engineering Department, Antalya, Turkey;*

[b] *Çankaya University, Industrial Engineering Department, Ankara, Turkey*

ABSTRACT

This study considers the scheduling problem of multiple independent and non-resumable jobs on unrelated parallel machines subject to machine availability and eligibility constraints. For each machine, there is a maximum continuous working time due to an unavailable period required for maintenance or tool changeover so that multiple unavailable periods on each machine may occur. The start time of an unavailable period on each machine is flexible and depends on the sum of the processing times of all jobs completed before this unavailability period. The objective is to minimize the makespan, which is the time to complete the processing of all non-resumable jobs. We develop a mixed integer linear programming (MILP) model to solve the problem optimally and a heuristic algorithm to solve the problem instances for which the MILP model cannot achieve an optimal solution in a reasonable allowed solution time. Computational experiments are done to evaluate our solution approaches' performance in terms of quality and time. The results show that using a mixed integer linear programming model is not a practical alternative, especially for large-sized problem instances. However, the proposed heuristic algorithm finds near-optimal solutions in a very short time.

ARTICLE INFO

Article history:

Received February 20, 2023
Revised November 8, 2023
Accepted January 10, 2024
Published online January 19, 2024

Keywords:
Scheduling;
Unrelated parallel machines;
Machine availability and eligibility constraints;
Non-resumable jobs;
Makespan;
Mixed integer linear programming

*Corresponding author:
Ferda Can Çetinkaya
cetinkaya@cankaya.edu.tr

1. Introduction

Scheduling problems in manufacturing environments are concerned with the optimal allocation of resources over time to a set of tasks. Resources and tasks are usually called *machines* and *jobs* in these environments [1]. The parallel machine scheduling problem is one of the well-known scheduling problems where a set of jobs is to be processed exactly by one of the multiple machines working in parallel. Scheduling activity in this environment includes assigning jobs to the parallel machines and sequencing jobs on each machine [2].

Parallel machine scheduling problem is divided into three main categories. The first category is *identical parallel machine scheduling*, as each job has identical processing time requirements on each machine. If the processing time of a job on a machine depends on the machine's speed, then the problem is defined as *uniform parallel machine scheduling*. The general case of parallel machine scheduling is *unrelated parallel machine scheduling*, in which the processing time of jobs on each machine is different and independent [2].

Most studies in the machine scheduling literature assume a stable machining environment in which all

Published by the University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia.

http://doi.org/10.24867/IJIEM-2024-1-345

machines are continuously available for processing jobs throughout the scheduling period. However, this assumption may only be realistic in some industrial environments since machines may not be available during specific periods. Machine unavailability may occur due to preventive maintenance, tool change-overs, material shortages, and machine breakdowns [3]–[6].

There are many industrial applications in which the machine availability constraint is observed [7]. An example is the scheduling of computer numerical control (CNC) machines. CNC machines cannot be continuously available due to tool wear. They have a maximum continuous working time, limited by the known and constant tool wear time. Thus, their tools need to be changed within a pre-determined time. Machines cannot process jobs during the known and constant tool change times [6], [8]. Another example is the case when the machines are unavailable due to preventive maintenance periods [9]. Preventive maintenance activities (such as lubrication, cleaning, and adjusting) reduce the machine breakdowns and improve the machine's life. Therefore, this leads to the reduction of rework and scrap rate. Consequently, preventive maintenance is a practical activity to prevent the high cost of corrective maintenance, which should be done after machine breakdowns [4], [10]-[12].

Two job characteristics, *resumable* and *non-resumable*, are extensively studied in the literature on scheduling problems with machine availability [3], [5]. In the case of the *resumable job*, when the processing of a job cannot be finished before the machine's unavailability period, its processing can be continued after the same machine becomes available again. That is, job processing is allowed to be interrupted, and the interrupted job must continue to be processed on the machine it is interrupted [13]. In the *non-resumable jobs* case, job processing must start and finish in the machine's available period (maximum working time). In the literature, the terms *preempt-resume* and *preempt-repeat* have also been used for resumable and non-resumable, respectively [2].

A well-known assumption in the parallel machine scheduling literature is that machines can process each job. However, this assumption must be revised for some manufacturing environments like the semiconductor industry [13]. Any machine may not process jobs due to the capabilities of machines. Sophistication and technological capabilities affect the processing of jobs so that jobs may have specific machine sets to be processed. This situation is known as the *machine eligibility* restriction in the scheduling literature [14]. The parallel machine scheduling problem with eligibility restriction is a particular case of the traditional parallel machine scheduling problem in which the processing time of a job is set to a sufficiently large positive number if this job is not eligible to be processed by a machine.

The traditional identical parallel machine scheduling problem to minimize the makespan, which is the time to complete the processing of all jobs, involves only one decision. This decision is the assignment of jobs to the machines [1], [2]. If the machine availability constraint is considered, then the problem must involve two decisions: the assignment of jobs to the machines and the scheduling of jobs during the availability periods on their assigned machines [3], [5]. Furthermore, if we consider the machine eligibility constraint, the number of machines that can be used to process each job will reduce. Thus, the solution space will decrease. On the other hand, when the processing time of jobs on each machine is different and independent (i.e., unrelated parallel machine scheduling environment), the selection of the machine to process each job will be critical due to different processing times on machines [15].

In this study, we address the problem of scheduling non-resumable jobs to minimize the makespan on unrelated parallel machines with machine availability and eligibility constraints simultaneously. It is desired to find a schedule allocating the jobs to the machines, assigning the jobs to availability batches (periods) on each machine, and determining the start times of unavailable periods on the machines. The contribution of our study is threefold. First, to our knowledge, in the literature, the study by Santoro and Junqueira [16] is the only one that considers the unrelated parallel machine scheduling problem with machine availability and eligibility constraints simultaneously. Thus, our study is the second attempt for unrelated parallel machine scheduling with non-resumable jobs and machine availability and eligibility constraints simultaneously. In contrast, unavailability periods in our study have fixed duration, and the resulting schedule determines the start times of the unavailability periods. Secondly, a mixed integer programming model has been developed to solve the problem under study optimally, especially for small and medium-sized problem instances. Thirdly, the proposed heuristic algorithm in our study is easy to implement for solving large-sized problems with a sound quality solution rather than using exact solution approaches with more computational effort.

The rest of this paper is organized as follows. In Section 2, we give a related review of the literature

on the unrelated parallel machine scheduling problems considering machine availability and eligibility constraints and minimizing the makespan. Section 3 briefly defines the problem under study and provides some properties of the optimal solution to the problem. In Section 4, we propose a mixed integer linear programming (MILP) model to solve the problem optimally. Section 5 describes our proposed heuristic algorithm, which can solve large-sized problems when the MILP cannot provide solutions. The computational tests to evaluate the performance of the proposed MILP model and the heuristic algorithm are presented in Section 6. Conclusions and several directions for future research are discussed in Section 7.

2. Literature review

The problem studied in this paper falls at the intersection of parallel machine scheduling problems with machine availability and eligibility constraints. Thus, in this section, we provide a brief overview of these studies to position our study in the related literature properly. Here, we limit our literature review to the studies considering the makespan as the scheduling performance (criterion).

For the last two decades, many researchers have studied the parallel machine scheduling problem by independently considering machine availability and eligibility constraints. In the literature, many studies only investigate the parallel machine scheduling problem with a machine availability constraint. The reader may refer to the paper by Ma et al. [7] for a comprehensive review of the literature on scheduling with a machine availability constraint and other scheduling performance measures and machining environments such as single-machine, flow shops, and job shops.

Most scheduling studies with an availability constraint in the literature consider makespan as the scheduling criterion since it focuses on improving resource utilization and productivity and reducing the energy consumption [17]. Table 1 provides a brief overview of the closely related studies dealing with makespan minimization on parallel machines with an availability constraint. All these studies assume that the start time of each unavailable period is known in advance. Table 1 shows that the studies concerning unrelated parallel machines are limited [18]-[21]. Our study differs from theirs since we assume that jobs have specific machine sets to be processed (i.e., machine eligibility constraint is considered), and the start time of each unavailability period is flexible since the time between two consecutive periods of unavailability should be less than or equal to a predetermined time (maximum continuous working time). Moreover, our problem environment under study is an extension of the single-machine environment, in which the two assumptions above exist, described in [6], [8], to the unrelated parallel machine environment.

On the other hand, a few studies only consider the unrelated parallel machine scheduling problem with machine eligibility constraint. Table 2 provides a brief overview of the related studies dealing with makespan minimization on parallel machines with eligibility constraint. As seen in Table 2, the study by Afzalirad and Rezaeinan [47] is the only one that investigates the unrelated parallel machine scheduling problem with machine eligibility constraint. However, the most recent study in the literature is by Maeckar et al. [48]. They investigated the unrelated parallel machine scheduling problem, considering machine eligibility constraints and delivery times to minimize the total weighted tardiness of jobs. They proposed a mathematical model and various heuristics to address the problem. For a comprehensive review of the literature on scheduling problems with machine eligibility constraints and other scheduling performance measures and machining environments such as single-machine, flow shops, and job shops, the reader may refer to [49], [50].

Although there are numerous studies considering machine availability and eligibility constraints independently, only a few studies consider these constraints simultaneously for parallel machine scheduling problems. For instance, Sheen et al. [51] delved into the problem of minimizing the maximum lateness. They developed a Branch and Bound algorithm, which can solve up to 50 jobs and seven machines in a reasonable timeframe. Moreover, Cunha et al. [52] considered a related study to minimize a weighted function that includes tardiness and idleness. Their proposed solution is based on an Iterated Local Search (ILS) algorithm, which they applied to a ship scheduling problem within the oil industry domain.

A study closely related to ours has been conducted by Santoro and Junqueira [16]. In their work, they address the challenge of considering machine availability and eligibility constraints simultaneously in the context of unrelated parallel machines to minimize the makespan. Their contribution includes the development of a mathematical model applicable to non-resumable and resumable cases. It is worth noting that in their study, the unavailability periods are predetermined, fixed, and changing in duration.

Table 1. Parallel machine scheduling studies with availability constraint to minimize the makespan

Reference	Type of parallel machines	Number of machines having unavailable periods	Number of unavailable periods on the machines	Length of the available periods	Solution approach
Chang and Hwang [22]	P_m	$< m$	Single	Unequal	Worst Case Analysis of MULTIFIT rule
Lee [3]	P_m	$< m$	Single	Unequal	Heuristic based on LPT rule
Hwang and Chang [4]	P_m	m	Single	Unequal	Worst Case Analysis of LPT rule
Gharbi and Haouari [23]	P_m	m	Single	Unequal	Branch and Bound Algorithm
Hwang et al. [24]	P_m	m	Single	Unequal	Worst Case Analysis of LPT and MLPT rules
Lee and Wu [25]	P_m	m	Single	Unequal	Heuristic based on LDR and MULTIFIT rules
Grigoriu and Friesen [26]	P_m	m	Single	Unequal	Heuristic based on LPT rule
Masmoudi and Benbrahim [27]	P_m	m	Single	Unequal	Heuristic Algorithms (Dividing to 5 Cases)
Pries and Sikora [28]	P_m	m	Single	Unequal	Benders Decomposition
He et al. [29]	P_m	m	Multiple	Unequal	Heuristic based on LPT rule
Beaton et al. [30]	P_m	m	Multiple	Unequal	Mathematical model and several heuristic algorithms
Xu and Yang [31]	P_m	m-1	Multiple	Equal	Average Case Analysis
Xu et al. [32]	P_m	m	Multiple	Equal	Worst Case Analysis of BFD-LPT rule
Xu et al. [33]	P_m	m	Multiple	Equal	Heuristic based on BFD-LPT rule
Li et al. [34]	P_m	m	Multiple	Equal	Two mathematical models and heuristic algorithms
Chen et al. [35]	P_m	m	Multiple	Equal	Mathematical models and three heuristic algorithms
Yong [36]	Q_m	m	Single	Unequal	Worst Case Analysis of LPT rule
Kaabi [37]	Q_m	m	Multiple	Unequal	A mathematical model and heuristic algorithm
Kaid et al. [18]	R_m	m	Single	Unequal	Tabu Search and Simulated Annealing
Suresh and Chaudhuri [19]	R_m	m	Multiple	Unequal	Heuristic based on Neighborhood Search
Lei and He [20]	R_m	m	Multiple	Unequal	Adaptive Artificial Bee Colony
Rosales et al. [21]	R_m	m	Multiple	Equal	Metaheuristic based on multi-start strategy

P_m: m dentical parallel machines; Q_m: m uniform parallel machines; R_m: m unrelated parallel machines;
LPT: Longest processing time; MLPT: Modified longest processing time; LPTX: Longest processing time based;
LS: List scheduling; MULTIFIT: Multiple-Fit; BFD: Best fit decreasing; LDR: Largest deterioration rate;
HLF: Highest Level First

In contrast, unavailability periods in our study have fixed duration, and the resulting schedule determines the start times of the unavailability periods.

3. Problem definition

In this section, we first briefly define the problem under study and its assumptions, next give its complexity and finally provide a property of the optimal schedule for the problem. The followings are the assumptions for the scheduling problem under study:

(1) There are n jobs and m unrelated parallel machines ready for processing at time zero.
(2) Each job has one operation that can only be processed on a set of eligible machines among m unrelated parallel machines. That is, machine eligibility constraint exists.
(3) Each machine cannot process more than one job at a time. That is, machines are not batch-processing machines.

Table 2. Parallel machine scheduling studies with eligibility constraint to minimize the makespan

Reference	Type of parallel machines	Additional Problem Characteristics	Solution approach
Vairaktarakisi and Cai [38]	P_m		Lower bounds, heuristic algorithm, and branch-and-bound procedure
Lin and Li [39]	P_m	Unit length	Binary search algorithm
Glass and Kellerer [40]	P_m		Matching based polynomial time ε-approximation
Liao and Sheen [41]	P_m	Machine availability	Binary search algorithm
Ou et al. [42]	P_m		Heuristic algorithm
Hu et al. [43]	P_m	Precedence constraints	Heuristic algorithm combined of the largest total amount of processing first rule and the enhanced smallest machine load first rule
Huo and Leung [44]	P_m		Heuristic algorithm
Li and Wang [45]	P_m	Job release time	Heuristic algorithm and polynomial time approximation scheme
Edis and Ozkarahan [46]	P_m	Resource-constrained	Combined Integer Programming/Constraint Programming
Afzalirad and Rezaeian [47]	R_m	Resource constraints, sequence-dependent setup times, different release dates, and precedence constraints	Genetic algorithm and artificial immune system

(4) The processing time of each job on each eligible machine is known.
(5) Job processing cannot be interrupted. That is, jobs are non-resumable.
(6) Machines are not continuously available due to the unavailable periods. That is, machine availability constraint exists.
(7) Durations of the unavailable (down) time T_U and the maximum continuous working time T_W between two consecutive unavailable periods are known.
(8) The maximum continuous working time is greater than or equal to the processing time of every job this machine can process. Otherwise, no feasible solution is achieved.
(9) Unavailable periods of the machines may overlap.

Based on the assumptions given above, the problem we deal with in this study can be stated as follows. There is a set of independent jobs with a single operation to be processed by one of the unrelated parallel machines subject to machine availability and eligibility constraints. It is desired to find a schedule allocating the jobs to the machines, assigning the jobs to availability batches (periods) on each machine, and determining the start times of unavailable periods on the machines to minimize the time to complete all jobs (makespan).

The following propositions give the complexity of the problem under study and its optimal solution property, respectively.

Proposition 1. *The problem under study is NP-hard in the strong sense.*
Proof: When machine availability and eligibility restrictions are omitted, we observe that the problem reduces to the scheduling of n jobs on m unrelated parallel machines to minimize the makespan, which is shown to be strongly *NP*-hard by Lenstra et al. [15]. Thus, our problem is also *NP*-hard in the strong sense.

Proposition 2. *For the problem under study, an optimal schedule exists such that the machine determining the optimal makespan does not have any idle time.*
Proof: If an idle time exists on the machine determining the makespan of a schedule, subsequent jobs or unavailable periods may be moved earlier without increasing the makespan of the current schedule.

Suppose jobs processed between two consecutive unavailable periods on a machine are considered a *batch* (group). In that case, a schedule on a machine can be viewed as a series of batches of jobs separated by unavailable periods on that machine. Figure 1 illustrates a feasible schedule of eleven non-resumable jobs on two consecutive machines. As it is illustrated

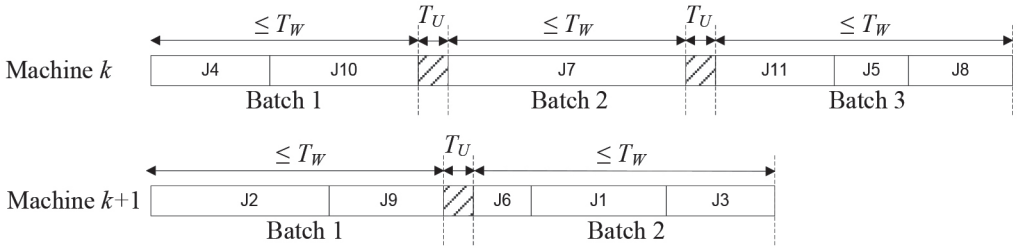

Figure 1. A feasible schedule of non-resumable jobs on two consecutive machines

in Figure 1, the batch lengths should be less than the maximum continuous working time T_W. They may vary since the sum of the processing times of jobs in different batches may be different since the sum of the job processing times in batches may not be equal. Thus, the batch length of a set of jobs on a machine shows the portion of the maximum continuous working time used. The number of batches gives the number of unavailable periods due to maintenance activities. Note that the last unavailable periods on the machines are not illustrated in Figure 1 since they do not affect the time to complete all jobs.

4. The proposed MILP model

Based on the assumptions, propositions, and definitions in Section 3, we now develop a mixed integer linear programming (MILP) model to solve the problem optimally. The decision here is to allocate the jobs to the batches on the machines so that an optimal schedule minimizing the makespan of all jobs is obtained. The following indices, sets, parameters, and decision variables are used in this model.

Indices and sets:

j Index for jobs, $j = 1,2, ..., n$
k Index for machines, $k = 1,2, ..., m$
M_j Set of eligible machines for job j
J_k Set of jobs that can be processed by machine k
t Index for batches, $t = 1,2, ..., |J_k|$ where $|J_k|$ is the number of jobs that can be processed by machine k

Parameters:

T_U Duration of an unavailable (down) period
T_W Maximum continuous working time between two consecutive unavailable periods
P_{jk} Processing time of job j on machine k where P_{jk} is set to a sufficiently large positive number if job j is not eligible to be processed by machine k

Decision variables:

X_{jkt} 1, if job j is assigned to batch t on machine k; 0, otherwise
Y_{kt} 1, if batch t is used on machine k; 0, otherwise
C_{max} Makespan

MILP model for the problem under study can be formulated as follows:

$$\text{Minimize} \quad C_{max} \tag{1}$$

$$\text{Subject to} \quad \sum_{k\in M_j}\sum_{t=1}^{|J_k|} X_{jkt}=1 \text{ for } j = 1,2,...,n \tag{2}$$

$$\sum_{j\in J_k} P_{jk}\times X_{jkt} \le T_W \times Y_{kt} \quad \text{for } k = 1,2,...m;\ t = 1,2,...,|J_k| \tag{3}$$

$$\sum_{j\in J_k}\sum_{t=1}^{|J_k|} P_{jk}\times X_{jkt} + T_U \times \left(\sum_{t=1}^{|J_k|} Y_{kt} - 1\right) \le C_{max} \quad \text{for } k = 1,2,...m \tag{4}$$

$$X_{jkt},\ Y_{kt} \in \{0,1\} \ \text{ for } j = 1,2,...,n;\ k = 1,2,...m;\ t = 1,2,...,|J_k| \tag{5}$$

In the above MILP model, the objective in (1) is to minimize the makespan. Constraint set (2) ensures that each job is assigned to one batch of a machine in the job's eligible set. Constraint set (3) guarantees that the sum of the processing times of jobs in each batch of every machine cannot exceed the maximum continuous working time. The maximum completion time (makespan) of the jobs is given by the constraint set (4). Constraint sets (5) impose binary restrictions on the decision variables.

In order to strengthen the model, we introduce the lower and upper bounds on the makespan. A simple lower bound for the C_{max} value is found by assuming that each job is assigned to its minimum processing time machine and that the total work is equally allocated among these machines. Then we have

$$LB = \max\left\{\frac{1}{m}\sum_{j=1}^{n}\min_{k\in M_j}\{P_{jk}\},\ \max_{j=1,...,n}\left\{\min_{k\in M_j}\{P_{jk}\}\right\}\right\} + \left\lfloor \max\left\{\frac{1}{m}\sum_{j=1}^{n}\min_{k\in M_j}\{P_{jk}\},\ \max_{j=1,...,n}\left\{\min_{k\in M_j}\{P_{jk}\}\right\}\right\} / \mathrm{T_W} \right\rfloor \times \mathrm{T_U} \tag{6}$$

where $\lfloor x \rfloor$ is the largest integer smaller than or equal to x.

Instead of assuming an initial upper bound on the makespan as infinity, the makespan value obtained from the initial phase of the proposed heuristic algorithm discussed in Section 6 can be used as an upper bound UB . i.e.,

$$UB = min\{C_{max}^{S1}, C_{max}^{S2}\} \quad (7)$$

where C_{max}^{S1} and C_{max}^{S2} are the makespans of the Initial Schedules 1 and 2, respectively, obtained in the first phase of the proposed algorithm. Thus, we add the following constraints to the mathematical model:

$$C_{max} \geq LB \quad (8)$$

$$C_{max} \leq UB \quad (9)$$

5. The proposed heuristic algorithm

As discussed in Section 6, the size of the MILP model increases drastically with an increase in the number of jobs. Therefore, the optimal solution to large-sized problem instances may not be obtained within reasonable computational times. Moreover, the existence of a polynomial-time algorithm to solve the problem optimally is unlikely. These facts motivated us to develop a fast algorithm that provides optimal or near-optimal solutions quickly.

The algorithm proposed in this section is a *local search algorithm with four phases*. In the algorithm's first phase (namely, *finding an initial schedule*), initial feasible schedules are generated. In the second phase (namely, *improvement by shifting jobs to other machines*), the initial feasible schedules are improved by shifting jobs from their current machines to each other. In the third phase (*improvement by swapping (pairwise interchanging) jobs between machines*), the jobs on different machines in the resulting schedule obtained by the second phase are pairwise interchanged to further improve the current solution. Finally, in the fourth phase (namely, *building a new schedule by perturbing the current one*), the schedule found in the third phase is improved by making a machine in the set of eligible machines for a job ineligible to process this job. The algorithm is run for two initial schedules, and the best of the resulting schedules is selected.

In the heuristic algorithm, we solve the bin-packing problems for assigning jobs to the batches of a machine, where each batch has a total processing capacity limited by the maximum continuous working time T_W. For solving the bin-packing problems, we use the heuristic algorithm SAWMBS proposed by Fleszar and Charalambous [53] since it is one of the best heuristic algorithms for solving the bin-packing problem. Because of page limitation, we do not provide this bin-packing algorithm. We refer the interested reader to [53].

5.1. Phase 1: Finding an initial schedule

Using the property of the optimal schedule given in Proposition 2, we generate two initial schedules for the non-resumable jobs case. If one of these schedules has a makespan value equal to the lower bound LB in equation (6), then this initial schedule is optimal. Otherwise, we continue with the remaining phases described in Sections 5.2 through 5.4 for each initial schedule and select the better of two schedules as the proposed schedule by the algorithm.

Initial Schedule 1

The stepwise description of the procedure that determines the Initial Schedule 1 is as follows:

Step 1. (i) Repeat this step for each job j in set J. If job j has only one eligible machine, assign job j to the eligible machine and calculate the workload of the eligible machine when job j is assigned to that machine; otherwise, assign job j to unassigned jobs set U.

(ii) Repeat this step for each job j in set U. Find the machine(s) having the minimum processing time for job j. If only one eligible machine has the minimum processing time for job j, assign job j to this machine, and calculate the workload of the eligible machine when job j is assigned to that machine. Otherwise (i.e., there are more than one eligible machine having the same minimum processing time for job j),

a. select the machine having the minimum work load among the set of eligible machines with the same minimum processing time for job j,
b. assign job j to the selected machine, and
c. calculate the workload of the machine for which job j is assigned.

Step 2. For each machine, apply the bin-packing heuristic algorithm SAWMBS to find an initial schedule of all jobs assigned and calculate

the associated completion time on this machine as $C_k = \sum_{j \in J_k} P_{jk} + (B_k - 1) \times T_U$ where B_k is the number of batches on machine k determined by the bin-packing heuristic algorithm SAWMBS. According to Proposition 1, we did not consider the idle times on machines.

Step 3. Calculate the makespan of the initial schedule of all jobs by selecting the maximum completion time among all machines as $C_{max} = \max_{k=1,\ldots,m} \{C_k\}$.

Initial Schedule 2

We only present the first step of the procedure that determines the Initial Schedule 2 since the last two steps of the procedures that determine the Initial Schedules 1 and 2 are identical.

Step 1. Repeat this step for each job j in set J. Find the machine(s) having the minimum processing time for job j. If only one eligible machine has the minimum processing time for job j, assign job j to this machine. Otherwise (i.e., there is more than one eligible machine with the same minimum processing time for job j), select the machine with the minimum index, and assign job j to the selected machine.

5.2. Phase 2: Improvement by shifting jobs to other machines

In this phase, the initial feasible schedule generated in Phase 1 is improved by shifting jobs from their current machines to other machines in their eligible sets. Below is the stepwise description of Phase 2:

Step 1. Rank the machines in nonincreasing order of their completion times C_k.

Step 2. Select a machine k according to this order.

(i) Arrange the jobs assigned to machine k in nonincreasing order of their processing times P_{jk}. Call this sequence LPT.

(ii) Select a job j from the LPT sequence. If there is only one machine in the set of eligible machines for job j, then consider a different job in the LPT sequence; otherwise, consider each machine k' ($k' \in M_j$ and $k' \neq k$) that is eligible to process job j. Temporarily assign job j to each machine k' if the condition $C_{k'} + P_{jk'} < C_k$ is satisfied, and go to Step 2(iii). If the condition $C_{k'} + P_{jk'} < C_k$ is not satisfied for every machine k', then go to Step 2(ii) to select a different job from the sequence.

(iii) If the jobs are non-resumable, apply the bin-packing heuristic algorithm SAWMBS for every machine k' on which job j is temporarily assigned; otherwise, go to Step (iv).

(iv) Calculate the associated temporary completion time on machine k' as $C_{k'} = \sum_{j \in J_{k'}} P_{jk'} + (B_k - 1) \times T_U$.

(v) Among the machines on which job j is temporarily assigned, determine machine s having the shortest completion time.

(vi) If the completion time of machine s is less than the completion time of machine k, keep the schedule in which job j is temporarily shifted to machine s and go to Step 1; otherwise, do not shift job j and go to Step 2(ii) to select another job.

Step 3. If all jobs in the LPT sequence of machine k are considered, then go to Step 2 to select another machine. Repeat until all machines are considered.

5.3. Phase 3: Improvement by swapping (pairwise interchanging) jobs between machines

This phase takes the schedule obtained in Phase 2 and aims to improve it by swapping jobs between machines. The steps of Phase 3 are as follows:

Step 1. Rank the machines in nonincreasing order of their completion times C_k.

Step 2. Select two machines, f and l, where f and l are the first and the last machines of the list found in Step 1, respectively.

(i) Arrange the jobs assigned to machine f in nonincreasing order of their processing times on this machine. Call this sequence LPT(1).

(ii) Arrange the jobs assigned to machine l in nonincreasing order of their processing times on this machine. Call this sequence LPT(2).

Step 3. Starting from the beginning of the LPT(1) sequence, select one job, i. Similarly, starting from the beginning of the LPT(2) sequence, select another job, j.

(i) If both machines f and l are eligible to process jobs i and j and condition

$$\max\{C_f, C_l\} > \max\{C_f + P_{jf} - P_{if}, C_l + P_{il} - P_{jl}\}$$

is satisfied, then temporarily exchange the jobs and go to Step 3(ii); otherwise,

go to Step 3 to select another pair of jobs.

(ii) Apply the bin-packing heuristic algorithm SAWMBS for machines f and l and calculate the temporary completion times on machines f and l.

(iii) Determine the temporary makespan of the schedule of all jobs by calculating the makespan as $C_{max}=\max_{k=1,\ldots,m}\{C_k\}$.

(iv) If the temporary makespan is greater than the completion time of machine f, do not exchange the jobs i and j, and go to Step 3 until all job pairs are considered for swapping. Otherwise, do not exchange the jobs, and go to Step 2 until all remaining machine pairs are considered.

5.4. Phase 4: Building a new solution by perturbing the current solution machines

The last phase of the heuristic algorithm aims to improve the schedule found in Phase 3 by making a machine in the set of eligible machines for a job ineligible to process this job. This approach is known as *perturbing a solution*. The steps of Phase 4 are as follows:

Step 1. If an improved solution is obtained, consider this improved schedule and go to Step 2; otherwise, convert the set of eligible machines for the selected job determined in Step 2 to its original set, and repeat this step for all remaining jobs.

Step 2. Select a job with the longest processing time from the most loaded machine.

Step 3. Make the machine on which the selected job is already assigned, ineligible to process the selected job (i.e., the machine is not in the set of eligible machines for the selected job), and go to Phase 1.

6. Computational experiments

In this section, we describe our computational tests to evaluate the effectiveness and efficiency of the MILP model and the proposed heuristic algorithm in finding the optimal schedules. We also compare the proposed algorithm with the mathematical model. The mathematical model is solved by the optimization software package GAMS (General Algebraic Modeling System), and the proposed heuristic algorithm is coded in Microsoft Visual C++.

6.1. Parameter setting and problem instances generation

The values of the parameters used in our experiments are generated as follows:

(1) *Machine eligibility sets*: To set the machines eligible to process jobs, we adopt a similar method followed by Alagoz and Azizoglu [54]. For each machine and job combination, we generate a random number between 0 and 1. If this number is greater than 0.3, then the machine is assumed to be eligible to process the job and added to the eligibility set of this job. If no machine is eligible for this job, we generate another random number between 0 and 1 until an eligible machine is found.

(2) *Processing times*: They are generated from discrete uniform distributions $U[1, a]$, where a=10, 50. If a machine is not eligible for a job, then the processing time of this job is set to a sufficiently large positive number.

(3) *Number of jobs and machines*: The numbers of machines are taken as 2, 3, 5, 10, and 20, whereas the numbers of jobs are taken as 10, 20, 50, 100, and 200 by preserving the requirement that the number of jobs should be greater than the number of machines in a problem instance.

(4) *Availability and unavailability times*: Maximum continuous working time T_W is set to a, $2a$, and $3a$, whereas the duration of the unavailable period T_U is set to $0.2T_W$, $0.5T_W$, and $1.0T_W$. We run our problem instances according to the combination of these maximum continuous working and unavailability times.

For each possible combination of the above parameters, ten problem instances are generated. Hence, a total of 3,420 problems are tested in our computational study.

6.2. Performance measures

The software package GAMS gives three types of solutions for the MILP models. One of the solutions is the *optimal solution*, which is the desired one; the other is the *best integer solution*, in which the solution is integer but not optimal. The third type of solution is the one in which no integer solution is achieved, but we have the *LP-relaxation solution*, which is the best lower bound on the optimal makespan value.

If the LP-relaxation solution equals the best integer solution, we conclude that the MILP model achieves the optimal solution. For the problem instances with no optimal integer solution, we compare the makespan value obtained by our heuristic algorithm with the makespan value of the LP-relaxation solution. In our experiments, we limit the runtime to 10,800 seconds for solving each problem instance by GAMS using its solver CPLEX.

The percent deviation (gap) between the best integer solution and the LP-relaxation solution obtained by the MILP model is calculated to measure the effectiveness of the MILP model. The percent gap is $PD^M = 100 \times (C_{max}^{BI} - C_{max}^{LR})/C_{max}^{LR}$, where C_{max}^{BI} and C_{max}^{LR} are the makespan values of the best integer and the LP-relaxation solutions, respectively. Similarly, the percent deviation of the solution obtained by the heuristic algorithm from the best possible solution (LP relaxation solution) obtained by the MILP model is calculated as $PD^H = 100 \times (C_{max}^{H} - C_{max}^{LR})/C_{max}^{LR}$, where C_{max}^{H} is the makespan value of the solution obtained by the heuristic algorithm. Note that $C_{max}^{LR} \leq C_{max}^{BI}$, and $C_{max}^{LR} = C_{max}^{BI}$ at the optimal solution.

We also report the number of optimum solutions obtained for problem instances in which $C_{max}^{LR} = C_{max}^{BI}$. For some problem instances, an integer solution cannot be found by solving the MILP model. In order to find the number of unsolved problem instances by the MILP model, we use the number of the LP relaxation solutions obtained only.

The efficiency measure of the proposed MILP model and the heuristic algorithm is the computational time required to solve the problem.

6.3. Discussion of the results

This section discusses the performance of solution approaches, the performance of the initial schedules generated in the first phase of the heuristic algorithm, and the effects of the phases in the heuristic algorithm.

Performance of the solution approaches

We now discuss the performance of our solution approaches by using both solution quality and CPU time. We first investigate the effects of the number of machines m and the number of jobs n on the performance of solution approaches when the maximum working time T_W and the duration of unavailable period T_U are fixed. As it is discussed in Section 6.1, there are eighteen T_W and T_U combinations. The results of our computational experiments reveal that the change in the number of machines and the number of jobs show nearly the same characteristics on the solution approaches' performance for each of the eighteen T_W and T_U combinations. Due to the page limitation, we only discuss the effects of change in the number of machines and jobs for problem instances with $pj \sim U[1, 50]$ when the T_W is 50 and T_U is 10.

In Table 3, the number of optimum solutions obtained by each solution approach is reported. The number of jobs significantly affects the number of optimum solutions obtained. The number of problem instances solved optimally decreases as the number of jobs increases. In the MILP, it is because the total number of constraints and variables increases as the number of jobs increases. Table 2 also illustrates the number of problem instances with the best integer solutions obtained by the MILP. Although we have yet to get any information about the optimality of these solutions, these integer solutions might be either optimum or not. MILP could not find any integer solution in some problem instances, such as when m is two and n is 200. The number of optimum solutions obtained for this set of 10 problem instances is zero, the number of best integer solutions is 4, and the number of non-integer solutions is 6.

Percent deviations of the solution approaches are also illustrated in Table 3. The percent deviation for each solution approach is zero when the number of jobs is small (i.e., when n is 10 or 20). For each solution approach, the percent deviations increase as the number of jobs increases. The performance of each solution approach also decreases as the number of machines increases. From these observations, we conclude that the performance of both MILP and the heuristic algorithm decreases as the problem size increases.

As seen from Table 3, the number of jobs significantly affects the maximum and average CPU times for the solution approaches. The CPU time of each solution approach increases rapidly as the number of jobs increases. The same table shows that the CPU time of the best integer solution obtained by the MILP approach is approximately equal to the average CPU time of small-sized problem instances. It is significantly smaller than the CPU time of the large-sized problem instances. In the MILP approach, some problem instances cannot be solved optimally within the limit of 10,800 seconds. The CPU time of the MILP does not show any systematic behavior when the number of machines increases. However, the number of machines has a little increasing effect on the CPU time of the heuristic algorithm.

Table 3. Performance of the solution approaches when T_W = 50 and T_U = 10

		MILP Model								Heuristic Algorithm				
m	*n*	NOS	NBIS	NNS	Ave. PD	Max. PD	Ave. CPU time	Max. CPU time	Average CPU time for BIS	NOS	Ave. PD	Max. PD	Ave. CPU time	Max. CPU time
2	10	10	0	0	0.00	0.00	1.59	1.75	1.59	10	0.00	0.00	0.00	0.00
	20	10	0	0	0.00	0.00	1.75	2.61	1.75	10	0.00	0.00	0.00	0.02
	50	2	8	0	0.95	2.26	8641.94	10800.00	18.38	2	0.95	2.26	0.07	0.11
	100	0	8	2	1.32	2.37	10800.00	10800.00	255.83	0	0.72	1.20	0.81	1.19
	200	0	4	6	1.28	1.82	10800.00	10800.00	2017.60	0	0.38	0.67	6.63	13.47
3	10	10	0	0	0.00	0.00	1.09	1.36	1.09	10	0.00	0.00	0.00	0.00
	20	10	0	0	0.00	0.00	1.75	4.72	1.75	9	0.19	1.89	0.01	0.02
	50	4	6	0	1.45	3.82	7411.40	10800.00	66.84	3	1.54	3.44	0.12	0.23
	100	0	8	2	1.85	5.19	10800.00	10800.00	1676.73	0	0.97	1.82	0.69	1.00
	200	0	9	1	1.47	2.44	10800.00	10800.00	870.16	0	0.50	1.02	4.38	7.42
5	20	10	0	0	0.00	0.00	3.22	8.45	3.22	9	0.31	3.08	0.02	0.03
	50	10	0	0	0.00	0.00	618.21	3657.55	429.59	5	0.47	1.95	0.13	0.20
	100	0	10	0	3.39	7.91	10800.00	10800.00	1467.95	0	2.75	5.59	1.15	1.97
	200	0	10	0	1.91	3.94	10800.00	10800.00	1353.87	0	1.43	2.24	6.72	8.27
10	50	10	0	0	0.00	0.00	574.62	5398.69	63.94	6	1.29	5.00	0.21	0.27
	100	0	10	0	6.02	11.04	10800.00	10800.00	947.85	0	6.15	11.04	1.60	2.13
	200	0	10	0	6.35	9.82	10800.00	10800.00	666.50	0	5.11	8.77	15.22	19.53
20	100	10	0	0	0.00	0.00	561.55	3438.72	337.49	4	3.41	8.70	4.48	5.78
	200	0	10	0	3.80	6.67	10800.00	10800.00	1365.05	0	5.23	7.14	29.15	37.23
Total (out of NPI =190)		86	93	11						68				
Average					1.57	3.02	6054.35	6912.33	607.74		1.65	3.46	3.76	5.20

NPI: Number of problem instances, NOS: Number of optimally solved instances, NBIS: Number of problem instances with best integer solutions, NNS: Number of instances with LP-Relaxation (non-integer) solution, NNS = NPI - (NOS + NBIS)

We also investigate the effects of changing the maximum working time T_W and the unavailability time T_U of the machines on the performance of solution approaches. Table 4 is formed by considering the last two rows of Table 3 and all other seventeen tables for the remaining T_W and T_U combinations. As seen from Table 4, the number of problem instances solved optimally by the solution approaches increases as the value of T_W increases since the number of working batches is small in long maximum working times. However, T_U values do not affect the number of problem instances solved optimally by the solution approaches. The number of optimum solutions obtained in the problem instances with $pj \sim U[1, 10]$ is higher than in the problem instances with $pj \sim U[1, 50]$. This situation is because the range of solution space to be searched for an integer solution is relatively large for problem instances with $pj \sim U[1, 50]$. T_W value also significantly affects the total number of integer solutions. It is seen that when T_W is 50 and T_U is 10, the number of integer solutions obtained is 179 (i.e., 86+93=179) out of 190 problem instances. This means the MILP could not find any integer solution in 11 problem instances out of 190. On the contrary, the number of problem instances without an integer solution becomes zero when T_W is 150 and T_U is 30. If T_W is high and the range of processing times is less, an integer solution can be found easily by the MILP.

In Table 4, we also report the effects of maximum working time T_W and unavailability time T_U values on the performance of the solution approaches. The CPU time of the MILP solution decreases as T_W increases. This situation is because the number of working batches decreases when T_W increases. However, increasing the T_U value increases the CPU time of the MILP solution. The same observations can be seen in the average CPU time for the best integer solution. Another observation is that the CPU time of the MILP solution in the

problem instances with $pj\sim U[1, 50]$ is higher than in the problem instances with $pj\sim U[1, 10]$. As seen from Table 3, the change in T_W time does not consistently affect the CPU time of the heuristic algorithm. On the contrary, increasing T_U values has a slightly negative effect on the CPU of the algorithm in the problem instances with $pj\sim U[1, 10]$, and it has no significant effect for the problem instances with $pj\sim U[1, 50]$. Thus, we conclude that the range of the processing times significantly affects the CPU time of the algorithm. However, the CPU time of the problem sets with $pj\sim U[1, 10]$ is smaller than that of the problem instances with $pj\sim U[1, 50]$.

Table 4 shows that T_W and T_U values significantly affect the percent deviation of both MILP and the heuristic algorithm. Performance of the solution approaches improves for both problem instances with $pj\sim U[1, 10]$ and $pj\sim U[1, 50]$ since the number of working batches increases as T_W increases. On the contrary, increasing T_U values reduce the performance of the solution approaches by increasing the percent deviation since the solution space to be searched increases with the increase on T_U.

The solution approaches provide better solutions to the problem instances with a small range of processing times since the difference between best-integer and LP-relaxation solutions is small and thus the solution space to be searched for an integer solution is reduced. Moreover, the number of alternative solutions obtained increases as the range of the processing times decreases since the probability of having equal processing times for a job on different machines increases as the range of processing times decreases.

The average percent deviation of the MILP and the heuristic algorithm is 0.80 and 1.26, respectively. For the optimally solved problem instances, the percent deviation of the heuristic algorithm is between 0.50 and 1.26. Therefore, we can conclude that the heuristic algorithm is notably effective since it performs very well in short average CPU times with 2.77 seconds.

Table 4. Performance of the solution approaches when T_W and T_U changes

			MILP Model								Heuristic Algorithm				
p_j	T_W	T_U	NOS	NBIS	NNS	Ave. PD	Max. PD	Ave. CPU time	Max. CPU time	Ave. CPU time for BIS	NOS	Ave. PD	Max. PD	Ave. CPU time	Max. CPU time
$U[1, 10]$	10	2	139	50	1	0.37	1.47	3053.71	6369.43	231.16	122	0.95	4.45	1.42	2.89
		5	143	46	1	0.47	2.15	2929.65	6420.99	188.49	121	1.24	6.89	1.77	4.27
		10	147	42	1	0.50	2.76	2633.63	5826.84	269.14	130	1.52	10.49	2.41	7.61
	20	4	176	14	0	0.06	0.40	830.41	3475.87	13.26	142	0.84	3.41	1.46	2.87
		10	176	14	0	0.14	1.10	825.09	2922.91	14.36	143	0.88	4.26	1.86	4.35
		20	174	16	0	0.29	1.39	944.98	2914.05	19.18	141	0.98	5.31	1.88	4.45
	30	6	183	7	0	0.02	0.14	404.54	2298.24	5.40	147	0.82	3.97	1.37	2.63
		15	181	9	0	0.05	0.32	517.10	2844.93	2.91	147	0.87	4.48	1.47	3.04
			180	10	0	0.19	1.79	579.89	2322.83	8.72	145	0.86	5.04	1.47	3.04
	50	10	86	93	11	1.57	3.02	6054.35	6912.33	607.74	68	1.65	3.46	3.76	5.20
		25	90	91	9	1.95	4.69	5784.14	7379.53	473.30	70	1.79	4.70	4.36	6.31
		50	83	99	8	3.57	12.78	6235.78	7646.24	809.64	67	2.17	6.71	5.45	11.15
$U[1, 50]$	100	20	128	61	1	0.46	1.20	3612.49	5438.93	290.02	85	1.04	2.80	3.34	4.87
		50	128	61	1	0.52	1.75	3588.54	4836.67	318.69	86	0.98	2.71	3.59	6.29
		100	127	62	1	0.47	1.73	3673.17	4789.80	295.42	84	0.95	2.81	3.59	6.26
	150	30	144	46	0	0.53	1.66	2696.81	4055.94	214.97	90	1.18	3.31	3.45	5.27
		75	140	50	0	0.99	3.33	2973.29	4167.07	301.97	89	1.61	4.80	3.63	6.02
		150	142	48	0	2.28	6.43	2917.58	4884.92	310.06	87	2.32	7.34	3.62	6.02
Total (out of NPI=3420)			2567	819	34					1964					
Average						0.80	2.67	2791.95	4750.42	243.02		1.26	4.83	2.77	5.14

NPI: Number of problem instances, NOS: Number of optimally solved instances, NBIS: Number of best integer solutions obtained only, NNS: Number of instances with LP-Relaxation (non-integer) solution, NNS = NPI - (NOS + NBIS)

Performance of the initial schedules for the heuristic algorithm

As it is mentioned in Section 5, two initial schedules are generated in the first phase of the heuristic algorithm. In our computational study, we also test the performance of the initial schedules. Our computational results show that the heuristic algorithm with Initial Schedule 1 finds 3,214 best solutions for 3,420 solved problem instances. However, the heuristic algorithm with Initial Schedule 2 finds 2,556 best solutions. This result implies that the performance of Initial Schedule 1 is better than that of Initial Schedule 2.

Effects of the phases in the heuristic algorithm

As discussed in Section 5, the heuristic algorithm has four phases, and applying each phase decreases the percent error of the heuristics' makespan from the optimal one but increases the computational effort. Thus, a user may not want to apply some of the phases in order to get a solution in a very short time.

To evaluate the effects of phases in the heuristic algorithm, we consider the percent improvement on the makespan from one phase to another. Percent improvement obtained by a phase is calculated as $PI^{P_i} = 100 \times (C_{max}^{P_i} - C_{max}^{P_{i+1}})/C_{max}^{P_i}$ where $C_{max}^{P_i}$ is the makespan value obtained by Phase i *(i=2,3,4)*. We have investigated the performance of phases in the algorithm concerning the changes in T_W and T_U values. The percent improvement on the makespan by Phase 2 strongly increases as the maximum working time and the range of processing times increase. The length of the unavailable period is significantly effective and improves the performance of Phase 2. Phase 3 slightly improves when the maximum working time increases for the problem sets with *pj~U*[1, 10], and it fluctuates randomly in problem sets with *pj~U*[1, 50]. Furthermore, Phase 3 does not illustrate any observable behavior when the length of the unavailable period increases. In the problem sets with *pj~U*[1, 50], the percent improvement of Phase 3 is better than the problem sets with *pj~U*[1, 10]. We have also observed that the percent improvement obtained by Phase 4 increases as the length of the unavailable period increases. The percent improvement of Phase 4 decreases as the maximum working time increases in the problem sets with *pj~U*[1, 10]. However, it does not show systematic behavior for percent improvement of Phase 4 in the problem sets with *pj~U*[1, 50]. In our study, we observe that the most significant average improvement is achieved by Phase 2, and its average improvement is 17.46 percent since the percent improvements by Phases 3 and 4 are 0.70 and 1.45, respectively. Therefore, Phase 2 of the algorithm is the most effective. Its average improvement decreases as the number of jobs increases and increases as the number of machines increases.

In our study for evaluating the performance of phases in the algorithm, we have observed that the average CPU times for Phases 1, 2, and 3 are close to zero, less than 0.02 seconds for each phase, to comment on the effects of T_W and T_U values. Phase 4 requires more CPU time, with an average of about 2.77 seconds. It does not illustrate any relation with different T_W and T_U values. In other words, a random pattern is observed as T_W and T_U values change in Phase 4. However, the average CPU time of Phase 4 in the problem sets with *pj~U*[1, 50] is higher than in the sets with *pj~U*[1, 10]. Thus, we propose that the user either employ the first three phases with short computational times or use all phases to find better solutions by consuming more time.

6.4. Theoretical implications of the study

The proposed MILP model in Section 4 represents a novel approach to address unrelated parallel machine scheduling problems under machine availability constraints. In comparison to similar models provided in the existing literature, this model shows an advantage in terms of requiring a relatively small number of decision variables and constraints. Consequently, it can be inferred that this model offers ease of implementation and applicability to solve small-sized problem instances. Furthermore, it serves as a foundational framework that can be adapted for addressing related problems with different objective functions and constraints. For instance, total tardiness and completion time can be minimized with minor adjustments in the current MILP model.

Similarly, the proposed heuristic algorithm exhibits simplicity and practicality in its application. The heuristic algorithm can also be used as a local search procedure in developing metaheuristic algorithms such as Tabu search and simulated annealing. Furthermore, with slight modifications, our heuristic algorithm can be adapted to handle alternative performance measures such as total tardiness and completion time. Moreover, the algorithm can be used with minor modifications for studying different problem characteristics such as sequence-dependent setup times, ready times, and delivery times.

7. Conclusions and future research directions

This paper considers a makespan minimization problem of scheduling multiple independent and non-resumable jobs on unrelated parallel machines subject to machine availability and eligibility constraints simultaneously. We develop a mixed integer linear programming model and propose a heuristic algorithm.

Our computational experiments show that solving the problem using a standard MILP solver is not a practical alternative, especially for large problem instances. The results also reveal that the proposed heuristic algorithm finds promising results as it optimally solves small-and medium-sized problem instances and finds near-optimal solutions for large instances in a short computational time. We also observe that increasing the maximum working time has a significant positive effect on the performance of the MILP and the proposed heuristic algorithm. Furthermore, we observe that increasing the range of processing times makes the problem difficult to be solved. To guide users, we also discuss the effects of the phases in our proposed heuristic algorithm. We observe from our computational experiments that the computational times of the first three phases are minimal, and the solution quality is mostly improved in Phase 2. Although the effect of Phase 4 is entirely satisfactory, it requires more computational time. Therefore, one can use the first three phases to get good results in a very short time or can use all phases to get better results at the expense of more computational effort.

Our study has some limiting assumptions for the problem environment under study. We assume that the setup time before processing a job is negligibly small so that it is added to the processing time of the job. However, setup times may not be insignificant and depend on the previously processed job on the same machine. That is, sequence-dependent setup times should be considered. In our study, we also assume that jobs are non-resumable, although some or all jobs may be semi-resumable where a setup may be needed when a job is resumed.

Furthermore, our problem assumes that the number of available tool changers (maintenance teams) is equal to or greater than the number of machines. This assumption leads to the possibility of concurrent tool changes during the job schedules. However, it may need more tool changers to change all the tools concurrently. In such cases, a more complex approach is required, involving the simultaneous scheduling of tool changers and jobs.

In addition to the possible future research directions by relaxing the limiting assumptions mentioned above, our study may have several extensions, which are open for future investigation. An extension would be studying the same problem considered in this paper for other performance measures, such as mean flow time, maximum lateness, number of tardy jobs, and total weighted tardiness as in the study by Maecker et al. [48]. As the second future research issue, the more realistic case with both resumable and non-resumable jobs may also be worth studying. A third future research issue may be the study of the same problem with a constraint on overlapping unavailable periods on different machines when there is only one worker or a group of workers to do the maintenance tasks on all machines. On the other hand, considering future research issues with more realistic assumptions makes the problem more complicated. Hence, developing new mathematical models and heuristic algorithms seems to be a future research topic.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

[1] K.R. Baker, Introduction to sequencing and scheduling, New York: Wiley, 1974.

[2] M. Pinedo, Scheduling: Theory, algorithms, and systems, New York: Prentice Hall, 2008, doi: 10.1007/978-0-387-78935-4.

[3] C.Y. Lee, "Parallel machine scheduling with nonsimultaneous machine available time," Discrete Appl. Math., vol. 30, no. 1 pp. 53-61, Jan. 1991, doi: 10.1016/0166-218X(91)90013-M.

[4] H.C. Hwang and S.Y. Chang, "Parallel machines scheduling with machine shutdowns," Comput. Math. Appl., vol. 36, No. 3, pp. 21-31, Aug. 1998, doi: 10.1016/S0898-1221(98)00126-6.

[5] C.Y. Lee, "Machine scheduling with an availability constraint," J. Glob. Optim., vol. 9, pp. 395-416, Dec. 1996, doi: 10.1007/BF00121681.

[6] M.S. Akturk, J.B. Ghosh, and E.D. Gunes, "Scheduling with tool changes to minimize total completion time: Basic results and SPT performance," Eur. J. Oper. Res., vol. 157, no. 3, pp. 784-790, Sep. 2004, doi: 10.1002/nav.10045.

[7] Y. Ma, C. Chu, and C. Zuo, "A survey of scheduling with deterministic machine availability constraints," Comput. Ind. Eng., vol. 58, no. 2, pp. 199-211, Mar. 2010, doi: 10.1016/j.cie.2009.04.014.

[8] J.S. Chen, "Optimization models for the tool change scheduling problem," Omega-Int. J. Manage. S., vol. 36, no. 5, pp. 888-894, Oct. 2008, doi: 10.1016/j.omega.2006.04.006

[9] M. Geurtsen, J.B.H.C Didden, J. Adan, Z. Atan, and I. Atan, "Production, maintenance and resource scheduling: a survey," Eur. J. Oper. Res., vol. 305, no. 2, pp. 501-529, Mar. 2023, doi: 10.1016/j.ejor.2022.03.045

[10] L.M. Pintelon and L.F. Gelders, "Maintenance management decision making," Eur. J. Oper. Res., vol. 58, no. 3, pp. 301,317, May. 1992, doi: 10.1016/0377-2217(92)90062-E

[11] S. Batun and M. Azizoglu, "Single machine scheduling with preventive maintenances," Int. J. Prod. Res, vol. 47, no. 7, pp.1753-1771, Apr. 2009, doi: 10.1080/00207540701636348

[12] G. Pinto, F.J.G. Silva, N.O. Fernandes, R. Casais, A. Paptista, and C. Carvalho, "Implementing a maintenance strategic plan using TPM methodology," Int. J. Ind. Eng. Manag., vol. 11, no. 3, pp. 192-204, Sep. 2020, doi: 10.24867/IJIEM-2020-3-26.

[13] J. Blazewicz, M. Drozdowski, P. Formanowicz, W. Kubiak, and G. Schmidt, "Scheduling preemptable tasks on parallel processors with limited availability," Parallel Com., vol. 26, no. 9, pp. 1195-1211, Jul. 2000, doi: 10.1016/S0167-8191(00)00035-1.

[14] G. Centeno and R.L. Armacost, "Parallel machine scheduling with release time and machine eligibility restrictions," Comput. Ind. Eng., vol. 33, no. 1-2, pp. 273-276, Oct. 1997, doi: 10.1016/S0360-8352(97)00091-0.

[15] J.K. Lenstra, D.B. Shmoys, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," Math. Program., vol. 46, pp. 259-271, Jan. 1990, doi: 10.1007/BF01585745

[16] M.C. Santoro and L. Junqueira "Unrelated parallel machine scheduling models with machine availability and eligibility constraints," Comput. Ind. Eng., vol. 179, 109219, May 2023, doi: 10.1016/j.cie.2023.109219.

[17] F. D'Amico, D.A. Rossit, and M. Frutos, "Lot streaming permutation flow shop with energy awareness," Int. J. Ind. Eng. Manag., vol. 12, no. 1, pp. 25-36, Mar. 2021, doi: 10.24867/IJIEM-2021-1-274.

[18] H. Kaid, A. Al-Ahmari, A. Al-Shayea, E. Abouel Nasr, A.K. Kamrani, and H.A. Mahmoud, "Metaheuristics for optimizing unrelated parallel machines scheduling with unreliable resources to minimize makespan," Adv. Mech. Eng., vol. 14, no. 5, pp. 1-12, 2022, doi: 10.1177/16878132221097023

[19] V. Suresh and D. Chaudhuri, "Scheduling of unrelated parallel machines when machine availability is specified," Prod. Plan. Control, vol. 7, no. 4, pp. 393-400, Apr. 1996, doi: 10.1080/09537289608930367

[20] D. Lei and S. He, "An adaptive artificial bee colony for unrelated parallel machine scheduling with additional resource and maintenance," Expert Syst. Appl., vol. 205, Nov. 2022, doi: 10.1016/j.eswa.2022.117577.

[21] O. Avalos-Rosales, F. Angel-Bello, A. Álvarez, and Y. Cardona-Valdés, "Including preventive maintenance activities in an unrelated parallel machine environment with dependent setup times," Comput. Ind. Eng., vol. 123, pp.364-377, Sep. 2018, doi: 10.1016/j.cie.2018.07.006.

[22] S.Y. Chang and H.C. Hwang, "The-worst case analysis of the MULTIFIT algorithm for scheduling non-simultaneous parallel machines," Discrete Appl. Math., vol. 92, no. 2, pp. 135-147, June 1999, doi: 10.1016/S0166-218X(99)00049-9.

[23] A. Gharbi and M. Haouari, "Optimal parallel machines scheduling with availability constraints," Discrete Appl. Math. Vol. 148, no. 1, pp. 63-87, Apr. 2005, doi: 10.1016/j.dam.2004.12.003.

[24] H.C. Hwang, K. Lee, and S.Y. Chang, "The effects of machine availability on the worst-case performance of LPT," Discrete Appl. Math., vol. 148, no. 1, pp. 49-61, Apr. 2005, doi: 10.1016/j.dam.2004.12.002.

[25] W.C. Lee and C.C. Wu, "Multi-machine scheduling with deteriorating jobs and scheduled maintenance," Appl. Math. Model., vol. 32, no. 3, pp. 362-373, Mar. 2008, doi: 10.1016/j.apm.2006.12.008.

[26] L. Grigoriu and D.K. Friesen, "Scheduling on same-speed processors with at most one downtime on each machine," Discrete Optim., vol. 7, no. 4, pp. 212-221, Nov. 2010, doi: 10.1016/j.disopt.2010.04.003.

[27] A.A. Masmoudi and M. Benbrahim, "New heuristics to minimize makespan for two identical parallel machines with one constraint of unavailability on each machine," in International Conference on Industrial Engineering and Systems Management (IESM), Seville, Spain, 2015, doi: 10.1016/j.disopt.2010.04.003.

[28] S. Pries and C.G.S Sikora, "Decomposition approach for integrated production scheduling and maintenance planning on parallel machines," in Selected topics on integrated production-scheduling and maintenance-planning problems, S.H. Pries, Ed. Hamburg, Germany, Univesitat Hamburg, 2022, pp. 50-68.

[29] J. He, Q. Li, and D. Xu, "Scheduling two parallel machines with machine-dependent availabilities," Comput. Oper. Res., vol. 72, pp. 31-42, Aug. 2016, doi: 10.1016/j.cor.2016.01.021.

[30] C. Beaton, C. Diallo, and E. Gunn, "Makespan minimization for parallel machine scheduling of semi-resumable and non-resumable jobs with multiple availability constraints," INFOR, vol. 54, no.4, pp. 305-316, Mar. 2016, doi: 10.1080/03155986.2016.1166795.

[31] D. Xu and D.L. Yang, "Makespan minimization for two parallel machines scheduling with a periodic availability constraint: Mathematical programming model, average-case analysis, and anomalies," Appl. Math. Model., vol. 37, No. 14-15, PP. 7561-7567, Aug. 2013, doi: 10.1016/j.apm.2013.03.001.

[32] D. Xu, K. Sun, and H. Li, "Parallel machine scheduling with almost periodic maintenance and non-preemptive jobs to minimize makespan," Comput. Oper. Res., vol. 35, no. 4, pp. 1344-1349, Apr. 2008, doi: 10.1016/j.cor.2006.08.015.

[33] D. Xu, Y. Yin, and H. Li, "Scheduling jobs under increasing linear machine maintenance time," J. Sched., vol.13, pp. 443-449, June 2010, doi: 10.1007/s10951-010-0182-0.

[34] G. Li, M. Liu, S.P. Sethi, and D. Xu, "Parallel-machine scheduling with machine-dependent maintenance periodic recycles," Int. J. Prod. Econ., vol. 186, pp. 1-7, Apr. 2017, doi: 10.1016/j.ijpe.2017.01.014.

[35] Y.Y. Chen, P.Y. Huang, C.J. Huang, S.Q. Huang, and F.D. Chou, "Makespan minimization for scheduling on two identical parallel machines with flexible maintenance and nonresumable jobs," J. Ind. Prod. Eng., vol. 38, no. 4, pp. 271-284, Mar. 2021, doi: 10.1080/21681015.2021.1883131.

[36] H. Yong H, "Uniform machine scheduling with machine available constraints," Acta Math. Appl. Sin., vol. 16, pp.122-129, Apr. 2000, doi: 10.1007/BF02677672.

[37] J. Kaabi, "Modeling and solving scheduling problem with m uniform parallel machines subject to unavailability constraints," Algo., vol. 12, no. 12 pp. 247, Nov. 2019, doi: 10.3390/a12120247.

[38] G.L. Vairaktarakis and X. Cai, "The value of processing flexibility in multipurpose machines," IIE Trans., vol. 35(8) 35, no. 8, pp. 763-774, 2003, doi: 10.1080/07408170304349.

[39] Y. Lin and W. Li, "Parallel machine scheduling of machine-dependent jobs with unit-length," Eur. J. Oper. Res., vol. 156, no. 1, pp. 261-266, July. 2004, doi: 10.1016/S0377-2217(02)00914-1.

[40] C.A. Glass and H. Kellerer, "Parallel machine scheduling with job assignment restrictions," Nav. Res. Logist., vol. 54, no. 3, pp. 250-257, Dec. 2007, doi: 10.1002/nav.20202.

[41] L.W. Liao and G.J. Sheen, "Parallel machine scheduling with machine availability and eligibility constraints," Eur. J. Oper. Res., vol. 184, no. 2, pp. 458-467, Jan. 2008, doi: 10.1016/j.ejor.2006.11.027.

[42] J. Ou, J.Y.T. Leung, and C.L Li, "Scheduling parallel machines with inclusive processing set restrictions," Nav. Res. Logist., vol. 55, no. 4, pp. 328-338., Mar. 2008, doi: 10.1002/nav.20286.

[43] X. Hu, J.S. Bao, and Y. Jin, "Minimising makespan on parallel machines with precedence constraints and machine eligibility restriction," Int. J. Prod. Res., vol. 48, no. 6, pp. 1639-1651, Apr. 2009, doi: 10.1080/00207540802620779.

[44] Y. Huo and J.Y.T. Leung, "Parallel machine scheduling with nested processing set restrictions," Eur. J. Oper. Res., vol. 204, no. 2, pp. 229-236, Jul. 2010, doi: 10.1016/j.ejor.2009.10.025.

[45] C.L. Li and X. Wang, X. "Scheduling parallel machines with inclusive processing set restrictions and job release times," Eur. J. Oper. Res., vol. 200, no. 3, pp. 702-710, Feb. 2010, doi: 10.1016/j.ejor.2009.02.011.

[46] E.B. Edis and I. Ozkarahan, "A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions," Eng. Optim, vol. 43, no. 2, pp. 135-157, Feb. 2011, doi: 10.1080/03052151003759117.

[47] M. Afzalirad and J. Rezaeian, "Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions," Comput. Ind. Eng., vol. 98, pp. 40-52, Aug. 2016, doi: 10.1016/j.cie.2016.05.020.

[48] S. Maecker, L. Shen, and L. Mönch, "Unrelated parallel machine scheduling with eligibility constraints and delivery times to minimize total weighted tardiness," Comput. Oper. Res., vol 149, 105999, Jan. 2023, doi: 10.1016/j.cor.2022.105999.

[49] J.Y.T. Leung and C.L. Li, "Scheduling with processing set restrictions: a survey," Int. J. Prod. Econ., vol. 116, no.2, pp. 251-262, Dec. 2008, doi: 10.1016/j.ijpe.2008.09.003.

[50] J.Y.T. Leung and C.L. Li, "Scheduling with processing set restrictions: a literature update," Int. J. Prod. Econ., vol. 175, pp. 1-1, May. 2016, doi: 10.1016/j.ijpe.2014.09.038.

[51] G.J. Sheen, L.W. Liao, and C.F. Lin, "Optimal parallel machines scheduling with machine availability and eligibility constraints," Int. J. Adv. Manuf. Technolf, vol. 36, pp. 132-139, Feb. 2008, doi: 10.1007/s00170-006-0810-1.

[52] V. Cunha, I. Santos, L. Pessoa, and S. Hamacher, "An ILS heuristic for the ship scheduling problem: application in the oil industry," Intl. Trans. in Op. Res., vol. 27, no. 1, pp. 197-218, Jan. 2020, doi: 10.1111/itor.12610.

[53] K. Fleszar and C. Charalambous, "Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem," Eur. J. Oper. Res., vol. 210, no. 2, pp. 176-184, Apr. 2011, doi: 10.1016/j.ejor.2010.11.004.

[54] O. Alagoz and M. Azizoglu, "Rescheduling of identical parallel machines under machine eligibility constraints," Eur. J. Oper. Res., vol. 149, no. 3, pp. 523-532, Sep. 2003, doi: 10.1016/S0377-2217(02)00499-X.